

# COMMUNICATIONS

CACM.ACM.ORG

OF THE

# ACM

07/08 VOL.51 NO.7

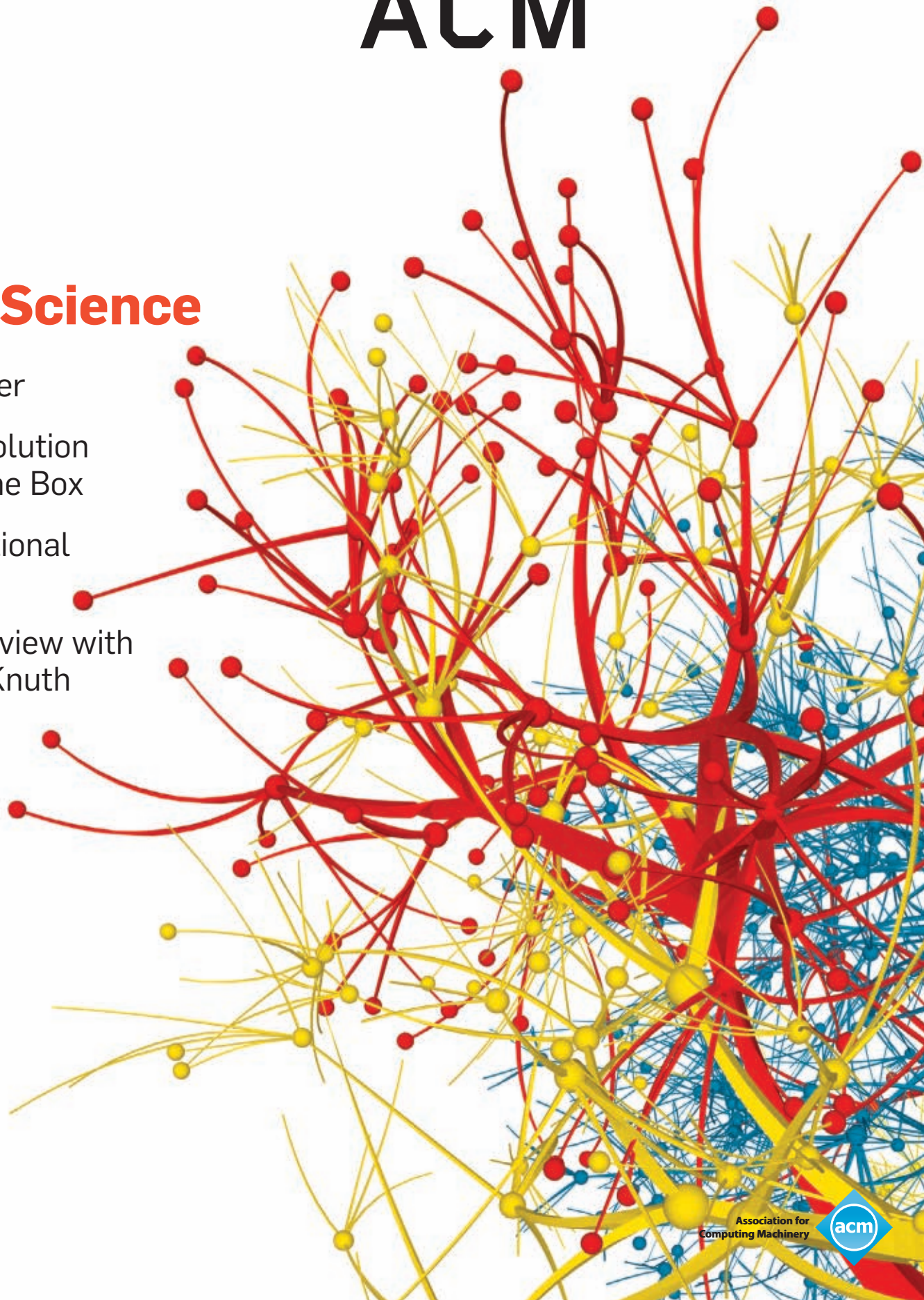
## Web Science

XML Fever

The Revolution  
Inside the Box

Transactional  
Memory

An Interview with  
Donald Knuth



# ACM, Uniting the World's Computing Professionals, Researchers, Educators, and Students

Dear Colleague,

At a time when computing is at the center of the growing demand for technology jobs worldwide, ACM is continuing its work on initiatives to help computing professionals stay competitive in the global community. ACM delivers resources that advance computing as a science and profession.

As a member of ACM, you join nearly 90,000 other computing professionals and students worldwide to define the largest educational, scientific, and professional computing society. Whether you are pursuing scholarly research, building systems and applications, or managing computing projects, ACM offers opportunities to advance your interests.

## MEMBER BENEFITS INCLUDE:

- A subscription to the completely redefined **Communications of the ACM**, ACM's flagship monthly magazine
- The option to subscribe to the full **ACM Digital Library**, with improved search functionalities and **Author Profile Pages** for almost every author in computing
- The **Guide to Computing Literature**, with over one million bibliographic citations
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full and unlimited access to over 3,000 online courses** from SkillSoft
- **Full and unlimited access to 1,100 online books**, featuring 500 from Books24x7®, and 600 from Safari® Books Online, including leading publishers such as O'Reilly (Professional Members only)
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- **TechNews**, ACM's tri-weekly email digest delivering stories on the latest IT news
- **CareerNews**, ACM's bi-monthly email digest providing career-related topics
- **MemberNet**, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and high-quality **Postini spam filtering**
- And much, much more!

ACM's worldwide network ranges from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network, and enjoy the advantages that come from sharing in their collective expertise, all of which serves to keep our members at the forefront of the technology world.

I invite you to share the value of ACM membership with your colleagues and peers who are not yet members, and I hope you will encourage them to join and become a part of our global community.

Thank you for your membership in ACM.

Sincerely,



John R. White  
Executive Director and Chief Executive Officer  
Association for Computing Machinery



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*



Association for  
Computing Machinery

Advancing Computing as a Science & Profession

# membership application & digital library order form

Priority Code: ACACM29

## You can join ACM in several easy ways:

**Online** <http://www.acm.org/join>      **Phone** +1-800-342-6626 (US & Canada)  
+1-212-626-0500 (Global)      **Fax** +1-212-944-1318

Or, complete this application and return with payment via postal mail

### Special rates for residents of developing countries:

<http://www.acm.org/membership/L2-3/>

### Special rates for members of sister societies:

<http://www.acm.org/membership/dues.html>

Please print clearly

Name \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State/Province \_\_\_\_\_ Postal code/Zip \_\_\_\_\_

Country \_\_\_\_\_ E-mail address \_\_\_\_\_

Area code & Daytime phone \_\_\_\_\_ Fax \_\_\_\_\_ Member number, if applicable \_\_\_\_\_

### Purposes of ACM

ACM is dedicated to:

- 1) advancing the art, science, engineering, and application of information technology
- 2) fostering the open interchange of information to serve both professionals and the public
- 3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature \_\_\_\_\_

ACM Code of Ethics:

<http://www.acm.org/serving/ethics.html>

## choose one membership option:

### PROFESSIONAL MEMBERSHIP:

- ACM Professional Membership: \$99 USD
- ACM Professional Membership plus the ACM Digital Library: \$198 USD (\$99 dues + \$99 DL)
- ACM Digital Library: \$99 USD (must be an ACM member)

### STUDENT MEMBERSHIP:

- ACM Student Membership: \$19 USD
- ACM Student Membership plus the ACM Digital Library: \$42 USD
- ACM Student Membership PLUS Print CACM Magazine: \$42 USD
- ACM Student Membership w/Digital Library PLUS Print CACM Magazine: \$62 USD

All new ACM members will receive an  
ACM membership card.  
For more information, please visit us at [www.acm.org](http://www.acm.org)

Professional membership dues include \$40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

### RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.  
General Post Office  
P.O. Box 30777  
New York, NY 10087-0777

Questions? E-mail us at [acmhelp@acm.org](mailto:acmhelp@acm.org)  
Or call +1-800-342-6626 to speak to a live representative

**Satisfaction Guaranteed!**

### payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

Visa/MasterCard       American Express       Check/money order

Professional Member Dues (\$99 or \$198) \$ \_\_\_\_\_

ACM Digital Library (\$99) \$ \_\_\_\_\_

Student Member Dues (\$19, \$42, or \$62) \$ \_\_\_\_\_

**Total Amount Due** \$ \_\_\_\_\_

Card # \_\_\_\_\_ Expiration date \_\_\_\_\_

Signature \_\_\_\_\_



## Departments

- 5 **Editor's Letter**  
"Where Do You Come From?  
And Where Are You Going?"  
*By Moshe Y. Vardi*
- 
- 7 **Publisher's Corner**  
**The Art and Business of Revitalizing  
A 50-Year-Old Science and  
Technology Magazine**  
*By Scott E. Delman*
- 
- 8 **CACM Online**  
**Your Attention, Please**  
*By David Roman*
- 
- 107 **Careers**
- 
- 109 **Calendar**

## Last Byte

- 112 **Q&A**  
**Talking Model-Checking Technology**  
A conversation with the 2007  
ACM A.M. Turing Award winners.  
*By Leah Hoffman*

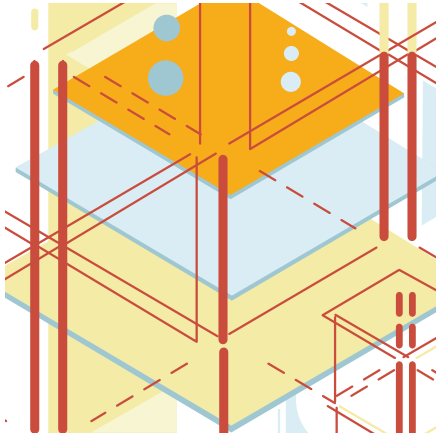
## News

- 9 **Cloud Computing**  
As software migrates from local PCs  
to distant Internet servers, users and  
developers alike go along for the ride.  
*By Brian Hayes*
- 
- 12 **Quantum Computing**  
Researchers are optimistic, but  
a practical device is years away.  
*By Michael Ross*
- 
- 14 **In Search of Dependable Design**  
How can developers increase the  
reliability of their designs?  
*By Leah Hoffman*

## Viewpoints

- 17 **Emerging Markets**  
**India's Role in the Globalization of IT**  
Tracing the exponential growth  
of the Indian IT industry.  
*By Alok Aggarwal*
- 
- 20 **Legally Speaking**  
**Revisiting Patentable Subject Matter**  
Is everything under the sun made by  
humans patentable subject matter?  
*By Pamela Samuelson*
- 
- 23 **Kode Vicious**  
**Beautiful Code Exists,  
If You Know Where to Look**  
Coding is his game,  
pleasantries distained.  
*By George V. Neville-Neil*
- 
- 27 **Point/Counterpoint**  
**Technology Curriculum for  
the Early 21st Century**  
In case you missed IT,  
the world has changed.  
*By Stephen J. Andriole/Eric Roberts*
- 
- 33 **Image Crisis**  
**Inspiring a New Generation  
of Computer Scientists**  
Consider what you can do to  
encourage young people to pursue  
technology-related career paths.  
*By Rick Rashid*
- 
- 35 **Interview**  
**The 'Art' of Being Donald Knuth**  
In this first of a two-part talk, the  
renowned scholar and computer  
scientist reflects on the influences  
that set the course for his  
extraordinary career.  
*By Len Shustek*

## Practice

- 
- 40 **XML Fever**  
Don't let delusions about  
XML develop into a virulent  
strain of XML fever.  
*By Erik Wilde and Robert J. Glushko*
- 
- 47 **Flash Storage Memory**  
Can flash memory become  
the foundation for a new tier  
in the storage hierarchy?  
*By Adam Leventhal*
- 
- 52 **Beyond Relational Databases**  
There is more to data access than SQL.  
*By Margo Seltzer*

## Contributed Articles

60 **Web Science: An Interdisciplinary Approach to Understanding the Web**

The Web must be studied as an entity in its own right to ensure it keeps flourishing and prevent unanticipated social effects.

*By James Hendler, Nigel Shadbolt, Wendy Hall, Tim Berners-Lee, and Daniel Weitzner*

70 **The Revolution Inside the Box**

How changes in computer architecture are about to impact everyone in the IT business.

*By Mark Oskin*

## Review Articles

80 **Transactional Memory**

Is TM the answer for improving parallel programming?

*By James Larus and Christos Kozyrakis*

## Research Highlights

90 **Technical Perspective**  
**Computer Science Takes On Molecular Dynamics**  
*By Bob Colwell*

91 **Anton, a Special-Purpose Machine for Molecular Dynamics Simulation**  
*By David E. Shaw, Martin M. Deneroff, Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J.P. Grossman, C. Richard Ho, Douglas J. Ierardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang*

98 **Technical Perspective**  
**The Physical Side of Computing**  
*By Feng Zhao*

99 **The Emergence of a Networking Primitive in Wireless Sensor Networks**  
*By Philip Levis, Eric Brewer, David Culler, David Gay, Sam Madden, Neil Patel, Joe Polastre, Scott Shenker, Robert Szewczyk, and Alec Woo*

**About the Cover:** Marius Watz is a distinguished digital artist whose bold abstract compositions are created directly through computer code. His tool of choice is Processing, a language built on Java and intended for use by artists and designers. As a student of computer science in the early 1990s, Watz would peruse *Communications of the ACM* at his university's library, looking for articles on computer graphics. With this cover illustration, one might say a circle has been completed.



Association for Computing Machinery  
Advancing Computing as a Science & Profession



# COMMUNICATIONS OF THE ACM

A monthly publication of ACM Media

*Communications of the ACM* is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

**Executive Director and CEO**  
John White  
**Deputy Executive Director and COO**  
Patricia Ryan  
**Director, Office of Information Systems**  
Wayne Graves  
**Director, Office of Financial Services**  
Russell Harris  
**Director, Office of Membership**  
Lillian Israel  
**Director, Office of Publications**  
Mark Mandelbaum  
**Director, Office of SIG Services**  
Donna Cappo

**ACM COUNCIL**  
**President**  
Stuart I. Feldman  
**Vice-President**  
Wendy Hall  
**Secretary/Treasurer**  
Alain Chesnais  
**Past President**  
David A. Patterson  
**Chair, SGB Board**  
Joseph A. Konstan  
**Co-Chairs, Publications Board**  
Ronald Boisvert, Holly Rushmeier  
**Members-at-Large**  
Michel Beaudouin-Lafon (2000-2008);  
Chuang Lin (2007-2008);  
Bruce Maggs (2006-2010);  
Jennifer Rexford (2007-2008);  
Barbara Ryder (2000-2008);  
David S. Wise (2004-2008)  
**SGB Council Representatives**  
Norman Jouppi (2006-2009);  
Robert A. Walker (2006-2008);  
Alexander Wolf (2005-2009)

**PUBLICATIONS BOARD**  
**Co-Chairs**  
Ronald F. Boisvert and Holly Rushmeier  
**Board Members**  
Gul Agha; Michel Beaudouin-Lafon;  
Jack Davidson; Carol Hutchins;  
Ee-ping Lim; M. Tamer Ozsu; Vincent Shen;  
Mary Lou Soffa; Ricardo Baeza-Yates

**ACM U.S. Public Policy Office**  
Cameron Wilson, Director  
1100 Seventeenth St., NW, Suite 507  
Washington, DC 20036 USA  
T (202) 659-9711; F (202) 667-1066

**Computer Science Teachers Association**  
Chris Stephenson  
Executive Director  
2 Penn Plaza, Suite 701  
New York, NY 10121-0701 USA  
T (800) 401-1799; F (541) 687-1840

**Association for Computing Machinery (ACM)**  
2 Penn Plaza, Suite 701  
New York, NY 10121-0701 USA  
T (212) 869-7440; F (212) 869-0481

## STAFF

**GROUP PUBLISHER**  
Scott E. Delman

**Executive Editor**  
Diane Crawford  
**Managing Editor**  
Thomas E. Lambert  
**Senior Editor**  
Andrew Rosenbloom  
**Senior Editor/News**  
Jack Rosenberger  
**Web Editor**  
David Roman  
**Editorial Assistant**  
Zarina Strakhan  
**Rights and Permissions**  
Deborah Cotton

**Art Director**  
Andrij Borys  
**Associate Art Director**  
Alicia Kubista  
**Assistant Art Director**  
Mia Angelica Balaquiot  
**Production Manager**  
Lynn D'Addesio  
**Director of Media Sales**  
Jonathan Just  
**Advertising Coordinator**  
Graciela Jacome  
**Marketing & Communications Manager**  
Brian Hebert  
**Public Relations Coordinator**  
Virginia Gold  
**Publications Assistant**  
Emily Eng

**Columnists**  
Alok Aggarwal; Phillip G. Armour  
Martin-Campbell-Kelly,  
Michael Cusumano; Peter J. Denning;  
Shane Greenstein; Mark Guzdial;  
Peter Harsha; Leah Hoffman  
Deborah Johnson; Susan Landau;  
Mari Sako; Pamela Samuelson;  
Gene Spafford; Cameron Wilson

**CONTACT POINTS**  
**Copyright permission**  
permissions@acm.org  
**Calendar items**  
calendar@acm.org  
**Change of address**  
acmcoa@acm.org

**WEB PRESENCE**  
http://cacm.acm.org

## ADVERTISING

**ACM ADVERTISING DEPARTMENT**  
2 Penn Plaza, Suite 701, New York, NY  
10121-0701  
T (212) 869-7440  
F (212) 869-0481

**Director of Media Sales**  
Jonathan M. Just  
jonathan.just@acm.org

**For the latest media kit—including rates—contact Graciela Jacome at jacome@acm.org**

## EDITORIAL BOARD

**EDITOR-IN-CHIEF**  
Moshe Y. Vardi

**NEWS Co-chairs**  
Mark Najor and Prabhakar Raghavan  
**Board Members**  
Brian Bershad; Hsiao-Wuen Hon;  
Mei Kobayashi; Rajeev Rastogi;  
Jeannette Wing

**VIEWPOINTS Co-chairs**  
William Aspray and  
Susanne E. Hambrusch  
**Board Members**  
Stefan Bechtold; Judith Bishop;  
Peter van den Besselaar; Soumitra Dutta;  
Peter Freeman; Seymour Goodman;  
Shane Greenstein; Mark Guzdial;  
Richard Heeks; Susan Landau;  
Carlos Jose Pereira de Lucena;  
Helen Nissenbaum; Beng Chin Ooi

**PRACTICE Chair**  
Stephen Bourne  
**Board Members**  
Eric Allman; Charles Beeler;  
David J. Brown; Bryan Cantrill;  
Terry Coatta; Mark Compton;  
Ben Fried; Pat Hanrahan  
Marshall Kirk McKusick;  
George Neville-Neil

**CONTRIBUTED ARTICLES Co-chairs**  
Al Aho and George Gottlob  
**Board Members**  
Yannis Bakos; Gilles Brassard; Peter  
Buneman; Andrew Chien; Anja Feldman;  
Blake Ives; Takeo Kanade; James Larus;  
Igor Markov; Gail C. Murphy; Shree Nayar;  
Lionel M. Ni; Sriram Rajamani; Avi Rubin;  
Ron Shamir; Larry Snyder;  
Wolfgang Wahlster; Andy Chi-Chih Yao;  
Willy Zwaenepoel

**RESEARCH HIGHLIGHTS Co-chairs**  
David A. Patterson and  
Stuart J. Russell  
**Board Members**  
Martin Abadi; P. Anandan; Stuart K. Card;  
Deborah Estrin; Stuart I. Feldman;  
Shafi Goldwasser; Maurice Herlihy;  
Norm Jouppi; Andrew B. Kahng; Linda  
Petzold; Michael Reiter;  
Mendel Rosenblum; Ronitt Rubinfeld;  
David Salesin; Lawrence K. Saul;  
Guy Steele, Jr.; Gerhard Weikum

**WEB Co-chairs**  
Marti Hearst and James Landay  
**Board Members**  
Jason I. Hong; Jeff Johnson;  
Wendy MacKay; Jian Wang

The Practice section of the CACM Editorial Board also serves as the Editorial Board of *ACM Queue*.

**AUTHOR GUIDELINES**  
http://cacm.acm.org/guidelines

## ACM Copyright Notice

Copyright © 2008 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page or screen display, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center; www.copyright.com.

**Subscriptions**  
Annual subscription cost is included in the society member dues of \$99.00 (for students, cost is included in \$42.00 dues); the nonmember annual subscription rate is \$100.00.

**ACM Media Advertising Policy**  
*Communications of the ACM* and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting [http://www.acm.org/publications/about\\_advertising?searchterm=advertising](http://www.acm.org/publications/about_advertising?searchterm=advertising) or by contacting ACM Media Sales at (212) 626-0654.

**Single Copies**  
Single copies of *Communications of the ACM* are available for purchase. Please contact [acmhelp@acm.org](mailto:acmhelp@acm.org).

**COMMUNICATIONS OF THE ACM** (ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

**POSTMASTER**  
Please send address changes to *Communications of the ACM*  
2 Penn Plaza, Suite 701  
New York, NY 10121-0701 USA



Association for Computing Machinery

Printed in the U.S.A.





Moshe Y. Vardi

DOI: 10.1145/1364782.1364783

# 'Where Do You Come From? And Where Are You Going?'

The noted management consulting firm Booz Allen Hamilton recently issued a report identifying the world's 10 most enduring institutions of the 20<sup>th</sup> and 21<sup>st</sup> centuries.

More interesting than their findings is their list of chosen determinants: innovative capabilities; governance and leadership; information flow; culture and values; adaptive response; risk structure; and legitimacy.

It is useful to keep these determinants in mind when we consider that *Communications*, having celebrated its 50th anniversary last January, is now older than most of its readers. Keeping a magazine in a leadership position for over 50 years is a daunting challenge indeed. As the Red Queen in Lewis Carroll's *Through the Looking Glass*, proclaimed: "Now, here, you see, it takes all the running you can do to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!" In a fast-changing discipline such as ours, we need to run incredibly fast if we want *Communications* to remain the foremost monthly magazine for the leadership of the computing field.

In the anniversary issue last January, I wrote an essay ("CACM: Past, Present, and Future," pg. 44) describing the process initiated in early 2005 by then ACM President David Patterson to revitalize *Communications*. Upon ACM Council's approval of a new editorial model in June 2007, a major initiative was launched to achieve this vision. This issue is the culmination of that effort. To appreciate the magnitude of the task, consider the analogy "replacing the engines of a jet plane in mid-flight." A monthly publication

cannot take a break. *Communications* continued to appear every month while behind the curtains major changes were taking place. Over the next few months, I will discuss these changes in greater detail and, more importantly, why these changes are necessary for *Communications* to maintain its leadership position.

An important component in this revitalization is the fortification of the professional staff producing it. A key addition to the fold is Scott Delman as Group Publisher. For *Communications* to continue to maintain its leadership position, it must excel not only editorially, but also as a business. Flagship publications of professional societies consume nontrivial fractions of their societies' budgets, and continual business innovation is critical to their success. Scott brings extensive experience in the scholarly publication marketplace; we are fortunate to have him join the team.

Another change is the establishment of a new editorial board (see masthead, pg. 4 or <http://www.acm.org/publications/cacm/?pageIndex=5>). This outstanding board brings together many of the leaders of the computing field, representing its diversity along many dimensions. The board is organized by teams, roughly corresponding to the different sections within the magazine. Unlike some distinguished editorial boards, this board is a working board. Producing a monthly publication requires an ongoing effort to "procure"

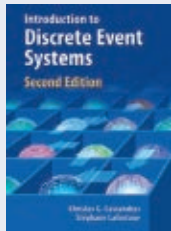
high-quality material, which is the task of this editorial board. The quality of a publication such as *Communications* is critically tied to the quality of its editorial board. ACM is lucky to have so many dedicated volunteers.

Let me close by mentioning one new feature of the new content model with ties to the old days when *Communications* was a venue for top research papers. The new Research Highlights section provides readers with a collection of outstanding research articles, selected from the broad spectrum of computing research conferences. This section provides a broad overview of the most significant developments in computing research. Articles appearing in this section are first nominated by Editorial Board members or Approved Nominating Organizations, and are then subject to final selection by the Editorial Board. Prior to publication, authors are requested to rewrite and expand the scope of their articles, as appropriate for *Communications'* broad-based readership. Each selected Research Highlights article is preceded by a one-page Technical Perspective providing readers with an overview of the underlying motivation of the research, the important ideas to emerge from the work, and its scientific and practical significance. These Technical Perspectives are written by noted experts in the field addressed in the research article.

**Moshe Y. Vardi**, EDITOR-IN-CHIEF



# New and Noteworthy



## Introduction to Discrete Event Systems

Cassandras, Christos G.; Lafortune, Stephane

A comprehensive introduction to the field of discrete event

systems, offering a breadth of coverage that makes the material accessible to readers of varied backgrounds. The book emphasizes a unified modeling framework that transcends specific application areas, linking the following topics in a coherent manner: language and automata theory, supervisory control, Petri net theory, Markov chains and queueing theory, discrete-event simulation, and concurrent estimation techniques. Distinctive features of the second edition include ► more detailed treatment of equivalence of automata, event diagnosis, and decentralized event diagnosis ► expanded treatment of centralized and decentralized control of partially-observed systems ► new sections on timed automata with guards (in the Alur-Dill formalism) and hybrid automata ► an introduction to hybrid systems ► updated coverage of discrete event simulation, including new software tools available ► recent developments in sensitivity analysis for discrete event systems as well as hybrid systems

2nd ed., 2008, XXIV, 776 p., Hardcover  
ISBN 978-0-387-33332-8 ► **\$89.95**



## A Brief History of Computing

O'Regan, Gerard

This useful and lively text provides a comprehensive introduction to the key topics in the history of

computing, in an easy-to-follow and concise manner. It covers the significant areas and events in the field - from the ancient Egyptians through to the present day - and both gives the reader a flavour of the history and stimulates further study in the subject.

2008, XX, 252 p. 72 illus., Hardcover  
ISBN 978-1-84800-083-4 ► **\$29.95**

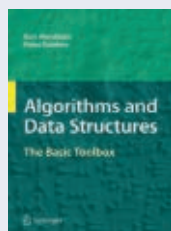


## Rightshore! Successfully Industrialize SAP® Projects Offshore

Hendel, Anja; Messner, Wolfgang; Thun, Frank (Eds.)

This book describes successful global delivery models utilizing industrialized methods to deliver SAP® projects from India. While the first part is devoted to management concepts, service offerings and the peculiarities of working together with India, the second part features eight case studies from different industries and from around the world describing how India delivery centers have been successfully deployed in SAP® development projects.

2008, XVIII, 292 p. 74 illus., Hardcover  
ISBN 978-3-540-77287-3 ► **\$64.95**



## Algorithms and Data Structures

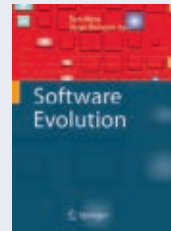
### The Basic Toolbox

Mehlhorn, Kurt; Sanders, Peter

This book is a concise introduction addressed to students and

professionals familiar with programming and basic mathematical language. Individual chapters cover arrays and linked lists, hash tables and associative arrays, sorting and selection, priority queues, sorted sequences, graph representation, graph traversal, shortest paths, minimum spanning trees, and optimization. The algorithms are presented in a modern way, with explicitly formulated invariants, and comment on recent trends such as algorithm engineering, memory hierarchies, algorithm libraries and certifying algorithms. The authors use pictures, words and high-level pseudocode to explain the algorithms, and then they present more detail on efficient implementations using real programming languages like C++ and Java.

2008, Approx. 310 p., Hardcover  
ISBN 978-3-540-77977-3 ► **\$44.95**



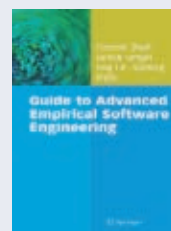
## Software Evolution

Mens, Tom; Demeyer, Serge (Eds.)

Mens and Demeyer, both international authorities in the field of software evolution, together with

the invited contributors, focus on novel trends in software evolution research and its relations with other emerging disciplines such as model-driven software engineering, service-oriented software development, and aspect-oriented software development. They do not restrict themselves to the evolution of source code but also address the evolution of other, equally important software artifacts such as databases and database schemas, design models, software architectures, and process management. The contributing authors provide broad overviews of related work, and they also contribute to a comprehensive glossary, a list of acronyms, and a list of books, journals, websites, standards and conferences that together represent the community's body of knowledge.

2008, XVIII, 347 p. 100 illus., Hardcover  
ISBN 978-3-540-76439-7 ► **\$89.95**



## Guide to Advanced Empirical Software Engineering

Shull, Forrest; Singer, Janice; Sjøberg, Dag I.K. (Eds.)

Empirical studies have become an integral element of software engineering research and practice. This unique text/reference includes chapters from some of the top international empirical software engineering researchers and focuses on the practical knowledge necessary for conducting, reporting and using empirical methods in software engineering.

2008, XII, 388 p. 37 illus., Hardcover  
ISBN 978-1-84800-043-8 ► **\$99.00**





Scott E. Delman

DOI:10.1145/1364782.1364784

# The Art and Business of Revitalizing a 50-Year-Old Science and Technology Magazine

By the time you flip to this page you will have noticed there is something dramatically different with *Communications of the ACM*.

The name on the cover remains the same, but even at first glance it is clear that the differences far outweigh the similarities with what we can now fondly and lovingly refer to as the old CACM.

Every decade or so, it is common for magazines to reinvent themselves. This is the case for many reasons, some related to the publishers and the changing dynamics of the publishing industry and some related to changing market and readership demographics. Very few magazines are able to survive over time without reacting to these changes and *Communications of the ACM* is no exception. In fact, magazines in the technology sector are even less immune to such changes, because of the rapid growth of the industry as a whole and trend toward specialization. As new areas of research and technology emerge, new publications are launched to satisfy the information needs of those new communities. The number of new technology magazines launched over the past 10 years for this reason is startling, each carving out a highly targeted and dedicated niche.

Equally startling is the number of technology magazines that have folded or merged into other publications over the past two years. As many magazines are heavily dependent on advertising revenue to fund operations, recent economic conditions and the rapid migration of advertising revenue from print to online have had a sobering effect on the technology magazine publishing industry.

Publications that have the ability to transform themselves editorially and appeal to

the changing needs of their readership have the greatest ability to succeed. Publications that can thrive even under these adverse market conditions are truly exceptional.

For 50 years, *Communications* has stood the test of time and when necessary reinvented itself to keep pace with ACM's diverse and growing membership and with the leadership of the computing field. In recent years, the computing community has started to indicate to ACM's leadership that it was time for a change—perhaps even a dramatic one—for the flagship publication.

The ACM membership has consistently grown more diverse over the past decade and its information needs have grown more demanding. No longer is it possible to categorize ACM's membership into a few distinct buckets, such as Educator or Researcher or Practitioner. Such distinctions make sense on paper and are favorable for commercial reasons, but they are also extremely limiting and do not reflect the way things often work in the real world, where these lines are often less defined. Practitioners are in fact interested in what next-generation research is coming down the pipeline, researchers are of course interested in what major technology challenges exist, and both groups have a vested interest in issues related to computing education. But as publishers it is far too easy to draw the distinctions instead of the similarities and to produce publications that target specific categories of readers instead of large and diverse communities that share common goals and interests.

When the field of computing was in its infancy, *Communications of the ACM* was created to serve as a single source of high-quality authoritative information to help bring together a growing community of scientists, technologists, and educators by highlighting the best the field had to offer. Some 50 years later, even though the field has grown tremendously, it continues to experience growing pains, and now more than ever requires a revitalized publication to bring this community together. The new *Communications of the ACM* is in many ways a homecoming for the field of computing itself and an opportunity to help guide the field through what many believe is a critical stage in the field's maturation into adulthood.

Over the coming months, *Communications'* dynamic new Editor-in-Chief, Moshe Y. Vardi, an all-star lineup of contributors, and I will introduce you to more of the innovations that form the basis for the new *Communications*. But it is most important to note that all of the changes to this new vision reflect the best the international computing community has to offer.

The true innovation, however, lies in the expansion of the magazine's editorial scope, which will appeal to the community's diverse mix of researchers, practitioners, and educators in all areas of computing and information technology.

It is with great pride and appreciation for your continued support of ACM's flagship publication that I welcome you to the new *Communications of the ACM*.

**Scott E. Delman**, GROUP PUBLISHER

# Your Attention, Please

*The redesigned CACM Web site will deliver exactly what you want.*

**H**OW DO YOU grab and hold someone's attention? Strategies vary. While polite interjections and attentive gazes work in some circles, they are ineffective on the Internet, where Web sites employ a host of in-your-face, lapel-grabbing techniques in the war for eyeballs.

We're in the middle of redesigning *Communications'* Web site. The plan is to make [cacm.acm.org](http://cacm.acm.org) as engaging and as awesome as a summer fireworks display, but without the noise. The new site won't be ready for months, but it's safe to say it will include some long-overdue content, notably news, but will mostly muffle the raised-voice techniques commonly shouted by popular sites, and will avoid these tactics spotted in the last 24 hours (where noted). There will be no blood ([NYtimes.com](http://NYtimes.com)), no fires ([CNN.com](http://CNN.com)), no violence ([washingtonpost.com](http://washingtonpost.com)), and no Paris Hilton ([youtube.com](http://youtube.com)). What's left?

Plenty. "Journalism is so much more than blood and sex," says Erica Stone, the fictional professor played by Doris Day in *Teacher's Pet*, a 1958 film that is both old fashioned and surprisingly fresh. "Your friend's kind of reporting went out with Prohibition," Day tells one of her night-school students, a seasoned newspaper editor played by Clark Gable. "TV and radio announce spot news minutes after it happens. Newspapers can't compete in reporting what happened anymore. But they can and should tell the public *why* it happened. Ask anybody. You'll find that today the average man wants to know *why*."

The scene is funny not because Day is perky, smart, and slightly oblivious (which she is), or because Gable is playing his archetype—the smirking, worldly, leering male (which he is). It's because adding "the Internet" to Professor Stone's roster of fast media has her delivering word-for-word the same lecture that newspaper publishers humbled by the Internet are repeating



50 years later. That's less astonishing foresight than an assertion of the ongoing need for change amidst increasing competition for readers' attention, whatever the media du jour.

This redesigned edition attests to that need for change; the upcoming Web site will do the same. Both will hold your attention by providing engaging, interesting, and meaningful content. Web inventor Tim Berners-Lee recently told BBC News that the Web is "still in its infancy." Created "by so many people collaborating across the globe," the world has "only started to explore the possibilities of [the Web]." That spirit of openness, collaboration, and creativity informs the redesign of *Communications'* site. It will provide more than what you see in the magazine, and will have you coming back for more.

No change is being made for change's sake. Every addition, every feature, every design decision is being bounced off ACM members to make sure it meets your needs and interests. The site will serve a rich menu of content that's been taste-tested by members through focus groups, one-on-one in-

terviews, and surveys. If this doesn't sound like what the doctor ordered, it's not. It's what you ordered.

"If it has good content, I don't care" if it's in news, blog, or video format, said one surveyed member. "Content trumps all," said another.

The site will present traditional *Communications* fare plus other computing stories. Development is still in flux, but the site is likely to publish opinion and commentary from readers and invited experts to explain the importance or significance of a work. "Some editorial guidance would be most useful to help readers understand the relevance of and keep updated on the newest research," one member said. "Reader discussion and commentary [are important]," another said, "because in the computer world, there is so much to be talked about. People must bounce ideas off of one another in order to get some kind of understanding and general consensus on anything which has any importance." Roger that.

Advertising is also on the menu. The vast majority of members, 93%, are open to or unruffled by ads on the site. That's an endorsement of ACM's strategy to find revenue to expand the range of services available to members and non-members alike. "Google makes a lot of money out of (relevant) ads," one member said. "They are ok for me. You should consider generating revenue from ads."

Nothing will get onto the site before it gets a thumbs up from members. To that end, ACM is still asking for input and will continue to do so until the site launches, probably in early 2009. It's all being done with an eye on holding your attention by delivering what you want. If you have opinions about the redesign that you'd like to share, go to [www.acm.org/publications/cacm/acm-member-feedback](http://www.acm.org/publications/cacm/acm-member-feedback).

Thanks for your attention.

**David Roman** ([roman@hq.acm.org](mailto:roman@hq.acm.org)) is *Communications'* Web Editor.

## Cloud Computing

*As software migrates from local PCs to distant Internet servers, users and developers alike go along for the ride.*

**T**HE GREEK MYTHS tell of creatures plucked from the surface of the Earth and enshrined as constellations in the night sky. Something similar is happening today in the world of computing. Data and programs are being swept up from desktop PCs and corporate server rooms and installed in “the compute cloud.”

Whether it's called *cloud computing* or *on-demand computing*, *software as a service*, or *the Internet as platform*, the common element is a shift in the geography of computation. When you create a spreadsheet with the Google Docs service, major components of the software reside on unseen computers, whereabouts unknown, possibly scattered across continents.

The shift from locally installed programs to cloud computing is just getting under way in earnest. Shrink-wrap software still dominates the market and is not about to disappear, but the focus of innovation indeed seems to be ascending into the clouds. Some substantial fraction of computing activity is migrating away from the desktop and the corporate server room. The change will affect all levels of the computational ecosystem, from casual user to software developer, IT manager, even hardware manufacturer.

In a sense, what we're seeing now is the second coming of cloud computing. Almost 50 years ago a similar transformation came with the creation of service bureaus and time-sharing systems that provided access to computing machinery for users who lacked a mainframe in a glass-walled room down the hall. A typical time-sharing service had a hub-and-spoke configuration. Individual users at terminals communicated over telephone lines with a central site where all the computing was done.

When personal computers arrived in the 1980s, part of their appeal was the promise of “liberating” programs and data from the central computing center. (Ted Nelson, the prophet of hypertext, published a book titled *Computer Lib/Dream Machines* in 1974.) Individuals were free to control their own computing environment, choosing software to suit their needs and customizing systems to their tastes.

But PCs in isolation had an obvious weakness: In many cases the sneaker-net was the primary means of collaboration and sharing. The client-server model introduced in the 1980s offered a central repository for shared data while personal computers and workstations replaced terminals, allowing individuals to run programs locally.

In the current trend, the locus of





computation is shifting again, with functions migrating outward to distant data centers reached through the Internet. The new regime is not quite a return to the hub-and-spoke topology of time-sharing systems, if only because there is no hub. A client computer on the Internet can communicate with many servers at the same time, some of which may also be exchanging information among themselves. However, even if we are not returning to the architecture of time-sharing systems, the sudden stylishness of the cloud paradigm marks the reversal of a long-standing trend. Where end users and corporate IT managers once squabbled over possession of computing resources, both sides are now willing to surrender a large measure of control to third-party service providers. What brought about this change in attitude?

For the individual, total control comes at a price. Software must be installed and configured, then updated with each new release. The computational infrastructure of operating systems and low-level utilities must be maintained. Every update to the operating system sets off a cascade of subsequent revisions to other programs. Outsourcing computation to an Internet service eliminates nearly all these concerns. Cloud computing also offers end users advantages in terms of mobility and collaboration.

For software vendors who have shifted their operations into the cloud, the incentives are similar to those motivating end users. Software sold or licensed as a product to be installed on the user's hardware must be able to cope with a baffling variety of operating environments. In contrast, software offered as an Internet-based service can be developed, tested, and run on a comput-

## For most applications, the entire user interface resides inside a single window in a Web browser.

ing platform of the vendor's choosing. Updates and bug fixes are deployed in minutes. (But the challenges of diversity don't entirely disappear; the server-side software must be able to interact with a variety of clients.)

Although the new model of Internet computing has neither hub nor spokes, it still has a core and a fringe. The aim is to concentrate computation and storage in the core, where high-performance machines are linked by high-bandwidth connections, and all of these resources are carefully managed. At the fringe are the end users making the requests that initiate computations and who receive the results.

Although the future of cloud computing is less than clear, a few examples of present practice suggest likely directions:

*Wordstar for the Web.* The kinds of productivity applications that first attracted people to personal computers 30 years ago are now appearing as software services. The Google Docs programs are an example, including a word processor, a spreadsheet, and a tool for creating PowerPoint-like presentations. Another undertaking of this kind is Buzzword, a Web-based word processor acquired by Adobe Systems in 2007.

Another recent Adobe product is Photoshop Express, which has turned the well-known image-manipulation program into an online service.

*Enterprise computing in the cloud.* Software for major business applications (such as customer support, sales, and marketing) has generally been run on corporate servers, but several companies now provide it as an on-demand service. The first was Salesforce.com, founded in 1999, offering a suite of online programs for customer relationship management and other business-oriented tasks; the company's slogan is "No software!"

*Cloudy infrastructure.* It's all very well to outsource the chore of building and maintaining a data center, but someone must still supply that infrastructure. Amazon.com has moved into this niche of the Internet ecosystem. Amazon Web Services offers data storage priced by the gigabyte-month and computing capacity by the CPU-hour. Both kinds of resources expand and contract according to need. IBM has announced plans for the "Blue Cloud" infrastructure. And Google is testing the App Engine, which provides hosting on Google server farms and a software environment centered on the Python programming language and the Bigtable distributed storage system.

*The cloud OS.* For most cloud-computing applications, the entire user interface resides inside a single window in a Web browser. Several initiatives aim to provide a richer user experience for Internet applications. One approach is to exploit the cloud-computing paradigm to provide all the facilities of an operating system inside a browser. The eyeOS system, for example, reproduces the familiar desktop metaphor—with icons for files, folders,

### Information Technology

## Gartner's Seven IT Grand Challenges

What are the most important IT challenges for the next 25 years? At the recent Gartner Emerging Trends Symposium/ITxpo, Gartner analysts identified seven IT grand challenges that, if met, will have profound economic, scientific and societal impacts. They are:

- ▶ Eliminate the need to manually recharge wireless devices
  - ▶ Parallel programming applications that fully exploit multicore processors
  - ▶ Non-tactile, natural computing interfaces
  - ▶ Automated computer-to-human speech translation
  - ▶ Reliable, long-term digital storage
  - ▶ Increase programmer productivity by 100 percent
  - ▶ Identify the financial consequences of IT investments
- "IT leaders should always be looking ahead for the emerging technologies that will have

a dramatic impact on their business, and information on many of these future innovations are already in some public domain," says Gartner VP Ken McGee. To find such information, Gartner suggests examining relevant research papers, patents, and production prototypes.



and applications—all living in a browser window. Another solution would bypass the Web browser, substituting a more-capable software system that runs as a separate application on the client computer and communicates directly with servers in the cloud. This is the idea behind AIR (formerly Apollo) being tested by Adobe Systems. OpenLaszlo, an open-source project, works in much the same way.

For those deploying software out in the cloud, scalability is a major issue—the need to marshal resources in such a way that a program continues running smoothly even as the number of users grows. It's not just that servers must respond to hundreds or thousands of requests per second; the system must also coordinate information coming from multiple sources, not all of which are under the control of the same organization. The pattern of communication is many-to-many, with each server talking to multiple clients and each client invoking programs on multiple servers.

The other end of the cloud-computing transaction—the browser-based user interface—presents challenges of another kind. The familiar window-and-menu layer of modern operating systems has been fine-tuned over decades to meet user needs and expectations. Duplicating this functionality inside a Web browser is a considerable feat. Moreover, it has to be done in a comparatively impoverished development environment. A programmer creating a desktop application for Windows or one of the Unix variants can choose from a broad array of programming languages, code libraries, and application frameworks; major parts of the user interface can be assembled from pre-built components. The equivalent scaffolding for the Web computing platform is much more primitive.


A major challenge of moving applications to the cloud is the need to master multiple languages and operating environments. In many cloud applications a back-end process relies on a relational database, so part of the code is written in SQL or other query language. On the client side, program logic is likely to be implemented in JavaScript embedded within HTML documents. Standing between the database and the client is a server application that might be written in a scripting language (such

as PHP, Java, and Python). Information exchanged between the various layers is likely to be encoded in some variation of XML.

Even though the new model of remote computing seems to reverse the 1980s “liberation” movement that gave individual users custody over programs and data, the shift does not necessarily restore control to managers in the corporate IT department.

To the extent that cloud computing succeeds, it represents an obvious competitive challenge to vendors of shrink-wrap software. Ironically, the open-source movement could also have a tough time adapting to the new computing model. It's one thing to create and distribute an open-source word processor competing with Microsoft Word; not so obvious is how a consortium of volunteers would create a Web service to compete with Google Docs.

Finally, cloud computing raises questions about privacy, security, and reliability—a major subject of discussion at a workshop held last January at the Center for Information Technology Policy at Princeton University. Allowing a third-party service to take custody of personal documents raises awkward questions about control and ownership: If you move to a competing service provider, can you take your data with you? Could you lose access to your documents if you fail to pay your bill? Do you have the power to expunge documents that are no longer wanted?

The issues of privacy and confidentiality are equally perplexing. In one frequently cited scenario, a government agency presents a subpoena or search warrant to the third party that has possession of your data. If you had retained physical custody, you might still have been compelled to surrender the information, but at least you would have been able to decide for yourself whether or not to contest the order. The third-party service is presumably less likely to go to court on your behalf. In some circumstances you might not even be informed that your documents have been released. It seems likely that much of the world's digital information will be living in the clouds long before such questions are resolved. 

**Brian Hayes** writes about science and technology from Durham, NC.

## Virtual Reality

# A Fly's Life

A team of Swiss and U.S. researchers have developed an interactive virtual-reality display system that enables them to better understand fruit flies' behavior and movement in response to their visual environment, *NewScientist* reports.

Led by Steven Fry of the Institute of Neuroinformatics in Zurich, the Swiss-U.S. team built a wind tunnel in which changing scenes or images are projected onto its walls. A camera tracks a fruit fly in 3D, making the scenes or images move in response to the animal's activity inside the wind tunnel. Previous research had involved tethered flies which, Fry said, “is very unnatural and it becomes very difficult to interpret the data because of the strong interference by the experimenter.”

The team's research, which has implications for animal behavior and biomimetic design control, can be readily reproduced, according to Fry. “Being based on standard hardware and software techniques, our methods provide an affordable, easy to replicate, and general solution for a broad range of behavioral applications in freely moving animals,” he says.

## Information Technology

# Wireless Conductor Paths?

The conductor paths in sensor systems have traditionally consisted of thin wires—until now. Researchers at the Fraunhofer Institute for Manufacturing Engineering and Applied Materials Research in Bremen, Germany, have developed a new technique that prints conductor paths, using a contactless aerosol ink with nano-sized silver particles. In tests conducted with the Institute for Microsensors, Actuators and Systems at the University of Bremen, the printed conductor paths have proven to be nearly 500 times thinner than wire bonds, and the sensors provide significantly more accurate measurements.

# Quantum Computing

*Researchers are optimistic, but a practical device is years away.*

**S**INCE QUANTUM ALGORITHMS and architectures will ultimately need hardware on which to run, we've explored how the principal experimental efforts are striving to produce it. Even 15 years ago, a quantum computer was generally viewed by computer scientists and physicists alike as an intriguing but probably unattainable theoretical curiosity. But interest exploded in 1994 after Peter Shor, then at Bell Laboratories (now at MIT), published his famous quantum factoring algorithm capable of undermining widely used cryptosystems that relied on the difficulty of factoring large numbers. Today, several thousand physics, computer science, and engineering researchers in more than 100 groups in universities, institutes, and companies around the world are exploring the frontiers of quantum information, encompassing quantum computing, as well as recently commercialized quantum cryptography and quantum teleportation communication techniques. Accelerating progress on virtually all fronts in this worldwide research community is yielding confidence that a practical quantum computer is indeed achievable.

Quantum computing's potential has always been tantalizing: Exponentially scalable computing power that could solve problems beyond the capabilities of conventional computers. The key is exploiting the superposition of quantum-entangled information units, or qubits. But the research challenges are daunting: How to create and reliably compute with the qubits, which require the seemingly mutually exclusive conditions of exquisite classical control while being isolated from any external influences that could destroy the entanglement.

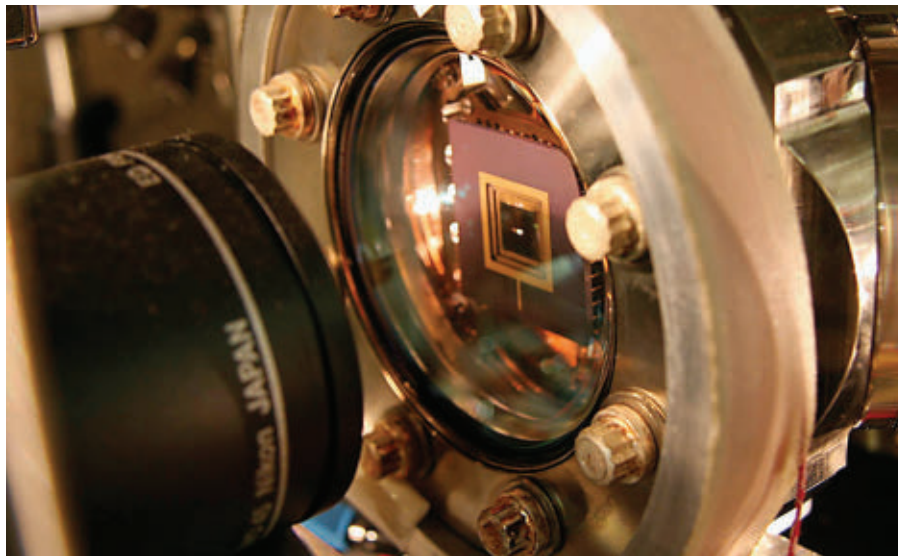
The computing power of a quantum computer grows exponentially with the number of qubits it uses. Dozens to hundreds of qubits will be needed for a

quantum computer to solve interesting problems using quantum algorithms (along with appropriate quantum error-correction techniques needed to be sure the answer is correct). The qubits must also be connected by quantum communication channels into logic gates that can be manipulated to implement the algorithms.

However, merely having and connecting qubits is not sufficient for a quantum computer. They must remain entangled long enough to complete the number of gate operations required by

ing. Typically, the qubit is a two-level motion mode for a trapped ion. The modes are modulated by laser pulses. The ion motion acts like a data bus, and gates are implemented by modulating neighboring ions.

"Our decoherence times can be up to 10 minutes—very long compared with other quantum computing techniques," says Dave Wineland of the National Institute of Standards and Technology, Boulder, CO. "But our gates are rather slow, about five microseconds for our two-qubit gates." Since



Miniature ion trap manufactured by Sandia National Laboratories.

the algorithm and mandatory error correction. Faster gate operation, higher fidelity (percentage of gate operations completed correctly), and greater error-correcting efficiency can speed the calculation or reduce the number of qubits needed to solve the problem.

More than a dozen different ways of creating qubits—each with its own strengths and challenges—have been developed to date. The following is a rundown of the leading candidates:

*Ion traps* use electrical and/or magnetic fields and laser-cooling to create a "pseudo-molecule" quantum register with micron-scale inter-ion spac-

ing. factoring a 100-to-200-digit number would require a million operations, even error-free implementation would take far longer than the qubit could be maintained. Researchers, led by Rainer Blatt of the Institut für Experimentalphysik Universität Innsbruck in Austria, recently set the record for qubit fidelity: 99.3%.

Integrating CMOS chips with ion traps is a recent innovation that permits quantum communication but uses classical control and measurement. One design created at Lucent by Richard Slusher (now at the Georgia Tech Quantum Institute) and Jungsang Kim

(now at Duke) is an ion-trap analogue to the electron-based charge-coupled device chips used in digital cameras. “This design gives addressability to an enormous number of ions,” says MIT’s Isaac Chuang.

*Quantum dots* are a popular solid-state host for qubits. Lieven Vandersypen’s research group in Delft, The Netherlands, reported last November that it had used an alternating electrical field to control single electrons contained in gallium arsenide quantum dots. Electrical control is more selective than the previously used magnetic fields. Their fidelity of flipping a single-electron spin (a simple gate) is a little lower than the 73% attained with magnetic-field control but is expected to increase as they gain experience. Future research will venture into no-spin hosts, such as silicon and carbon (carbon-12, nanotubes, graphene), that are expected to have much longer decoherence times.

*Linear optic* qubits are created by simultaneously producing forward and backward photons and encoding their logical states into vertical and horizontal polarizations. This approach has the advantage of long decoherence times and compatibility with fiber optics but needs higher photon-creation and -detection efficiencies. Last December, Andrew White’s group from the University of Queensland reported that it had used a linear optic circuit involving four qubits to find the prime factors of 15 (5 and 3) thus demonstrating that system’s ability to perform the core processes required for implementing Shor’s algorithm.

In April, Prem Kumar of Northwestern University announced a quantum gate created within an optical fiber. A few years ago, Kumar showed that photons can remain entangled within a fiber for a distance of 100 kilometers. The recent result will be useful in creating quantum repeaters for a distributed quantum information network.

*Superconducting* qubits can be made in three types: charge, flux, and phase. Each uses excitation states of Josephson junctions: two superconductor pieces separated by an insulator thin enough for Cooper pairs of electrons to tunnel across. This approach is scalable, since superconductivity enables fast control and readout and large,

## More than a dozen different ways of creating qubits have been developed to date.

controllable couplings between widely separated qubits. However, it also requires extremely low temperatures—milliKelvins—and tends to have short decoherence times (to date only a few microseconds). John Martinis’s group at the University of California, Santa Barbara, recently measured single-qubit fidelity of 98% in a phase-qubit system. And Robert Schoelkopf’s group at Yale developed a “transmon” qubit 300 microns long and very stable against noise.

*Hybrid approaches* combine the best features of their parents. Chuang’s research group is integrating ion traps into superconducting qubits. “Ion traps are very hard to connect to anything,” Chuang says. “It would be really nice to have an ion trap with wires coming in and out.” Vandersypen says that as ion traps get smaller they might eventually look similar to quantum dots.

*D-Wave Systems*, a venture-backed company based in Vancouver, BC, made news twice last year when it announced the operation of a 16-qubit (in February) and 28-qubit (in November) “adiabatic” quantum computer. However, many scientists are skeptical of both the claims and their importance. The adiabatic method, which is a quantum version of simulated annealing, involves slowly evolving a system toward the solution. Skeptics say this approach will prove to be neither fault-tolerant nor scalable. Although the company revealed scant scientific detail about its approach, its president and CEO, Geordie Rose, says it has no proof yet of entanglement—the hallmark of quantum computation.

*Diamond-based systems* are an intriguing recent entrant. The qubit is the spin state of a nitrogen impurity adjacent to a vacancy in the carbon crystal.

Stemming from Tom Kennedy’s 2003 research at the Naval Research Laboratory, “nitrogen-vacancy color centers” have two compelling advantages: their spin state can be both initialized and read out optically at room temperature and weak spin-orbit coupling in diamond makes this qubit well-decoupled from its environment. Several groups have jumped in to study this system, and its decoherence time has increased from about 50 microseconds to nearly a millisecond.

Challenges include devising ways to control individual qubits and couple them together. But David Awschalom of the University of California, Santa Barbara, is optimistic. “If you had told me,” he says, “a few years ago that you were going to try to control a single electron at gigahertz frequencies in a solid-state material at room temperature, I’d have said, ‘Good luck!’ Now we’re doing just that.”

This system is also useful as an accessible test bed for studying spin interaction in solid-state materials that may contribute to the success of other systems and quantum physics knowledge in general. Awschalom’s group recently watched quantum information from a single “nitrogen-vacancy center” spin disappear into the “bath” of spins associated with the much more common nitrogen impurities not associated with vacancies...then reappear. “In quantum physics, this is a big deal,” Awschalom says. “It’s an age-old problem. There have been 1,000 theory papers on this, but no experiments.”

The recent across-the-board progress, however, has stimulated optimism in the eventual success and impact of quantum computing inconceivable 15 years ago. For example, IBM Research’s David DiVincenzo, who proved in 1995 that quantum algorithms could be executed using only two-qubit operations and later devised seven widely accepted criteria for a practical quantum computer, says, “I’m confident that the quantum computer will eventually change the world and will deeply influence how information processing will be done in the future.”

**Michael Ross** writes about science and technology from San Jose, CA.

**Mark Oskin**, University of Washington, Seattle, contributed to this article.



# In Search of Dependable Design

*How can software and hardware developers increase the reliability of their designs?*

**I**N 1994, an obscure circuitry error was discovered in Intel's Pentium I microprocessor. Thomas R. Nicely, a mathematician then affiliated with Lynchburg College in Virginia, noticed that the chip gave incorrect answers to certain floating-point division calculations. Other researchers soon confirmed the problem and identified additional examples. And though Intel initially tried to downplay the mistake, the company eventually responded to mounting public pressure by offering to replace each one of the flawed processors.

"It was the first error to make the evening news," recalls Edmund Clarke of Carnegie Mellon University. The cost to the company: around \$500 million.

Nearly 15 years later, the Pentium bug continues to serve as a sobering reminder of how expensive design flaws can be. The story is no different for software: a \$170 million virtual case management system was scrapped by the FBI in 2005 due to numerous failings, and a flawed IRS tax-processing system consumed billions of dollars in the late 1990s before it was finally fixed. And in an era in which people rely on computers in practically every aspect of their lives—in cars, cell phones, airplanes, ATMs, and more—the cost of unreliable design is only getting higher. Data is notoriously difficult to come by, but a 2002 study conducted by the National Institute of Standards and Technology (NIST) estimated that faulty software alone costs the U.S. economy as much as \$59.5 billion a year in lost information, squandered productivity, and increased repair and maintenance.

But it's not just a matter of money—increasingly, people's lives are at stake. Faulty software has plunged cockpit displays into darkness, sunk oil rigs, and caused missiles to malfunction.



"There have been only a few real disasters due to software. But we're walking closer and closer to the edge," says MIT's Daniel Jackson.

Experts agree that flaws typically arise not from minor bugs in code, but during the higher-level design process. (Security flaws, which tend to be caused by implementation-level vulnerabilities, are often an exception to this rule.) One class of problems arises at the requirements phase: program design requirements are often poorly articulated, or poorly understood. Another class arises from insufficient human factors design, where engineers make unwarranted assumptions about the environment in which software or hardware will operate. If a program isn't capable of handling those unforeseen conditions, it may fail.

But mistakes can happen at any time. "Since humans aren't perfect, humans make mistakes, and mistakes can be made in any step of the development process," cautions Gerard Holz-

mann of the NASA/JPL Laboratory for Reliable Software.

Holzmann is among a small group of researchers who are committed to developing tools, techniques, and procedures for increasing design reliability. Currently, most programs are debugged and then refined by random testing. Testing can be useful to pinpoint smaller errors, say researchers, but inadequate when it comes to identifying structural ones. And tests designed for specific scenarios may not be able to explore combinations of behavior that fall outside of anticipated patterns. The search is therefore on for additional strategies.

One promising technique is known as model checking. The idea is to verify the logic behind a particular software or hardware design by constructing a mathematical model and using an algorithm to make sure it satisfies certain requirements. Though the task can be time consuming, it forces developers to articulate their requirements in a systematic, mathematical way, thereby minimizing ambiguity. More importantly, however, model checkers automatically give diagnostic counterexamples when mistakes are found, helping developers pinpoint what went wrong and catch flaws before they are coded.

"When people use the term 'reliability,' they might have some probabilistic notion that 'only rarely' do errors crop up, whereas people in the formal verification community mean that all behaviors are correct against all specified criteria," explains Allen Emerson of the University of Texas at Austin. (In recognition of the importance of formal verification techniques, the 2007 ACM A.M. Turing Award was given to Edmund Clarke, Allen Emerson, and Joseph Sifakis for their pioneering work in model checking. A Q&A with



the three Turing recipients can be found on page 112.)

Model checking has proven extremely successful at verifying hardware designs. In fact, Xudong Zhao, a graduate student of Clarke's, showed that model checking could have found Intel's floating-point division error—and that the company's fix did indeed correct the problem. Since then, Intel has been a leading user of the technique.

But because even small programs can have millions of different states (a dilemma known to the discipline as the “state explosion problem”), there are limits to the size and complexity of designs that model checking can verify, and it's been less immediately successful for software. The verification of reactive systems—the combination of hardware and software interacting with an external environment—also remains problematic, due mainly to the difficulty of constructing faithful models.

“We've come a long way in the last 28 years, and there's a huge, huge difference in the scale of problems we can address now as opposed to 1980,” says Holzmann. “But of course we are more ambitious and our applications have gotten more complex, so there is a lot more to be done.”

Other techniques include specialized programming languages and environments that facilitate the creation of reliable, reusable software modules. Eiffel, developed by the Swiss Federal

**“How can you ever hope to build a dependable system if you don't know what ‘dependable’ means?” asks MIT's Daniel Jackson.**

Institute of Technology's Bertrand Meyer and recipient of ACM's 2006 Software System Award, is one well-known example; Alloy, a tool developed by Daniel Jackson and the MIT Software Design Group, has also shown great promise.

To supplement the new languages and techniques, other researchers have focused on outlining more effective procedures and methodologies for developers to follow as they work.

“I'm not a great believer in formal analysis,” says Grady Booch of IBM Research. “Problems tend to appear at this curious intersection of the technological and the social.” After monitoring 50 developers for 24 hours, for example, Booch found that only 30% of their time was spent coding—the rest was spent talking to

other members of their team. Avoiding miscommunication, he believes, is therefore critical. Booch is perhaps best known for developing (with Ivar Jacobson and James Rumbaugh) the Unified Markup Language, or UML, a language that uses graphical notations to create an abstract model of a software or hardware system and helps teams communicate, explore, and validate potential designs. More recently, he has continued to focus on the big picture of development with the online Handbook of Software Architecture, which brings together a large collection of software-intensive systems and presents them in a manner that “exposes their essential patterns and that permits comparisons across domains and architectural styles.” The ultimate goal, of course, is to help developers apply that time-tested knowledge to their own programming projects.

“Reuse is easier at a higher level of abstraction,” explains Booch. “So we can reuse patterns, if not necessarily code.”

MIT's Daniel Jackson is another strong believer in the “big picture” approach. “The first thing we need to do is be honest about the level of reliability that we need,” he asserts. “The second thing is to think about what really cannot go wrong—about what's mission critical and what's not.”

Rather than starting with a typical requirements document that outlines

## Awards

# 2007 ACM Award Winners

**Outstanding Contribution to ACM Award**  
Robert A. Walker,  
Kent State University

**Distinguished Service Award**  
David A. Patterson, University  
of California at Berkeley

**Eugene L. Lawler Award for Humanitarian Contributions within Computer Science and Informatics**  
Randy Wang,  
Microsoft Research India

**Paris Kanellakis Theory and Practice Award**  
Bruno Buchberger,  
Johannes Kepler University

**Karl V. Karlstrom Outstanding Educator Award**  
Randy Pausch,  
Carnegie Mellon University

**Grace Murray Hopper Award**  
Vern Paxson, International  
Computer Science Institute  
and University of California at  
Berkeley/Lawrence Berkeley  
National Laboratory

**A.M. Turing Award**  
Edmund M. Clarke,  
Carnegie Mellon University  
E. Allen Emerson,  
University of Texas at Austin  
Joseph Sifakis, Centre National  
de la Recherche Scientifique  
and Verimag Laboratory

**Software System Award**  
David Harel, The Weizmann  
Institute of Science  
Hagi Lachover  
Amnon Naamad, EMC Corporation  
Amir Pnueli, NYU Courant  
Institute of Mathematical Sciences  
Michal Politi,  
Tadiran Electronic Systems  
Rivi Sherman, Negevtech  
Mark Trakhtenbrot,  
Holon Academic Institute of  
Technology and The Open  
University of Israel  
Aron Trauring, Zotecabv

**ACM – Infosys Foundation Award**  
Daphne Koller, Stanford University

**Doctoral Dissertation Award**  
Sergey Yekhanin, Princeton University

**Honorable Mentions:**  
Benny Applebaum,  
Princeton University  
Vincent Conitzer, Duke University  
Yan Liu, IBM

**ACM-W Athena Lecturer Award**  
Shafi Goldwasser, MIT and The  
Weizmann Institute of Science

**ACM – AAI Allen Newell Award**  
Leonidas J. Guibas, Stanford University

Several award winners had yet to be announced at press time. We'll have more news about the 2007 ACM award winners in next month's issue.

tasks in a procedural way, says Jackson, developers must first make sure they understand what the system is really about. What are its essential properties? Who are its stakeholders? What level of dependability does it need?

“How can you ever hope to build a dependable system if you don’t know what ‘dependable’ means?” he asks. The task itself is abstract, but Jackson believes that articulating all requirements and assumptions is crucial to tackling it—ideally in a formal, methodological way. The most important thing, according to Jackson, is the act of articulation itself. “When you write things down, you often find that you didn’t understand them nearly as well as you thought you did.” And there’s always a temptation to jump to the solution before you’ve fully understood the problem. “That’s not to say that automated tools and techniques like model checking aren’t useful, of course. Tools are an important support, but they’re secondary,” says Jackson.

And the more safety-critical the application, the more rigorous developers must be. “If your computer crashes,

## Simpler programs are easier to verify with tools like model checkers.

it’s inconvenient, but it’s not a threat to anyone’s life,” says Holzmann. Among the approaches he and his lab—who work to guarantee the safety of the computer systems that run spacecraft—are currently looking into is the development of simple, yet effective, coding standards. His recommendations may seem somewhat draconian (in safety-critical applications, they forbid the use of goto statements, setjmp or longjmp constructs, and direct or indirect recursion, for example), but they are intended to increase simplicity, prevent common coding mistakes, and force developers to create more logical architectures. Simpler programs are also easier to verify with tools like model checkers. After overcoming their initial reluctance, Holzmann says, developers often find that the restrictions are a worthwhile trade-off for increased safety.

A rigorous focus on simplicity can be costly, of course, especially for complex legacy systems that would be prohibitively expensive to replace but that need, nonetheless, to be updated or further developed. So can taking the time out to formally articulate all requirements and assumptions, or to verify software designs. Yet the cost of fixing an error in the initial stages of development is far less than fixing it at the end—a lesson that Intel, for one, now knows well.

“Computer science is a very young discipline,” explains Joseph Sifakis, research director at CNRS. “We don’t have a theory that can guarantee system reliability, that can tell us how to build systems that are correct by construction. We only have some recipes about how to write good programs and how to design good hardware. We’re learning by a trial-and-error process.” ■

Leah Hoffman is a Brooklyn-based freelance writer.

## Computer Science

# Winning Strategy

St. Petersburg University of Information Technology, Mechanics and Optics recently won the 32<sup>nd</sup> annual ACM International Collegiate Programming Contest (ICPC) World Finals, held in Branff, Canada. It was the university’s second ACM-ICPC world championship in four years.

The annual programming contest started with 6,700 teams from 1,821 universities in 83 countries, competing at 213 sites around the world. Through a series of regional competitions, the field narrowed to 100 teams. At the World Finals, each three-person team had one computer and five hours to solve 11 programming problems.

“The main goal at the World Finals is to solve problems,” says Andrey Stankevich, coach of the St. Petersburg University of Information Technology, Mechanics and Optics team, who was interviewed via email. “If you use your time to solve problems (and not to look for bugs in the problems already solved, but not accepted by the judges) you have time to solve more. So, the way to win the World Finals is to solve problems in such way that you don’t make bugs, and if the problem is accepted, you can immediately start solving another one. This requires cooperation in both thinking about problems and writing code.”

The winning team solved eight problems, followed by second-place Massachusetts Institute of Technology, third-place Izhevsk State Technical University, fourth-place Lviv National University and fifth-place Moscow State University, each of which solved seven problems.

The competition at each ACM-ICPC World Finals appears to be stronger than the previous one, and longtime contest sponsor IBM believes the global contest is good for the IT industry. “The value proposition for IBM is not only about the students who go on to work for IBM, but who go on to work for our clients and our business partners, or who become faculty members,” says IBM director of talent Margaret Ashida. “It’s a win for everyone.”

### Coming Next Month in COMMUNICATIONS

*Scaling Massive Multiplayer Online Game Infrastructure*

*Techniques for Designing Games with a Purpose*

*Computer Science and Game Theory*

*The Rise and Fall of CORBA*

*Evaluating Methodology for the 21st Century*

*Composable Memory Transactions*

*Envisioning the Future of Computing Research*

*CTO Roundtable*

*Part II of an interview with Donald Knuth*

**And the latest news about game theory, assistive technologies, and computing and the developing world.**



DOI: 10.1145/1364782.1364789

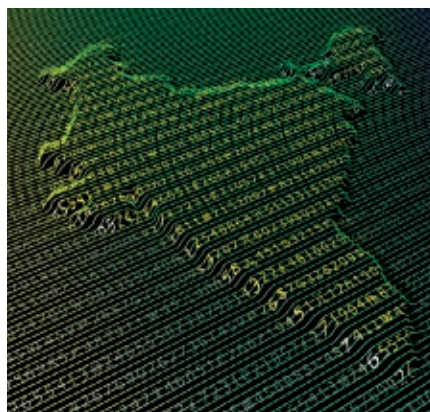
Alok Aggarwal

## Emerging Markets India's Role in the Globalization of IT

*Tracing the exponential growth of the Indian IT industry.*

**I**NDIA'S INFORMATION TECHNOLOGY (IT) industry is a product of serendipity—it happened mainly by accident and partly by design. In the 1960s and 1970s, there was no separate IT industry in India. Multinational companies such as U.S.-based IBM and U.K.-based ICL were the largest providers of hardware, which was bundled with operating systems and few software packages that were generally written in FORTRAN and COBOL. Furthermore, Indian import duties on hardware were extremely high (almost 300%), and even IBM used to sell old, refurbished, and antiquated machines (because that is all most Indian companies could afford). Hence, large enterprises (including the Indian defense department and other public organizations) that needed customized applications usually employed in-house teams that did everything from installing systems to writing software.

The first software company in India was Tata Consulting Services (TCS), which began operations in 1968. Fortunately, after executing a few local or-



ders, TCS obtained its first big export assignment in 1973–1974, when it was asked to build an inventory control software solution for an electricity generation unit in Iran. During this period, TCS also developed a hospital information system in the U.K. in cooperation with Burroughs Corporation (at that time the second-largest hardware company in the world). Through its software exports and collaborations, TCS became a role model for other Indian IT companies later. Also, during the late 1970s, the Indian government lowered import duties on all IT equip-

ment. But there was a catch. Importers had to recover in exports twice the value of the foreign exchange they spent on importing computers. Partly as a result, by the early 1980s, India was the only developing nation to have any significant software exports with 30 companies that were beginning to export IT services. If we now look back at the 1970–1980 era, it is clear that the following four unrelated incidents contributed heavily in shaping the Indian IT industry:

► In late 1970s, the Indian government passed a controversial law (only repealed in 1992) that forced all multinational companies to reduce their equity share in their Indian subsidiaries to less than 50%. Since IBM did not want to comply, it decided to leave India. This opened the market for local IT competitors and made Indian companies generally less reliant on mainframe computers.

► The advent of personal computers in the 1980s reduced the cost of importing hardware substantially, thereby, spawning an industry that has more than 3,100 companies today.

## Annual revenue and number of IT professionals employed by Indian IT services industry.

|   | 2001-02         | 2003-04          | 2005-06          | 2007-08*           | 2009-10*           | 2011-12*           | 2013-14*            | 2015-16*            |
|---|-----------------|------------------|------------------|--------------------|--------------------|--------------------|---------------------|---------------------|
| <b>IT Services</b>  |                 |                  |                  |                    |                    |                    |                     |                     |
| Exports   | 4.5; 150        | 7.3; 220         | 13.2; 345        | 23.1; 510          | 36.7; 720          | 54.5; 975          | 76.9; 1,225         | 101.7; 1,490        |
| Domestic  | 2.1; 160        | 2.7; 190         | 3.9; 250         | 7.9; 400           | 14.3; 615          | 24.0; 870          | 37.5; 1,155         | 54.0; 1,485         |
| <b>Engineering Services, R&amp;D, Software Products</b>       |                 |                  |                  |                    |                    |                    |                     |                     |
| Exports   | 1.6; 55         | 2.5; 75          | 3.9; 105         | 6.3; 145           | 10.0; 205          | 15.1; 280          | 21.0; 365           | 27.8; 430           |
| Domestic  | 0.5; 40         | 0.8; 60          | 1.3; 85          | 2.2; 115           | 3.2; 135           | 5.1; 165           | 6.7; 200            | 9.6; 240            |
| <b>Total IT Industry (excluding Hardware and BPO sectors)</b> | <b>8.7; 405</b> | <b>13.3; 545</b> | <b>22.3; 785</b> | <b>39.5; 1,170</b> | <b>64.2; 1,675</b> | <b>98.7; 2,290</b> | <b>142.1; 2,945</b> | <b>193.1; 3,645</b> |

USD (Billion); Number of IT Professionals ('000) \*Expected

► Realizing the Indian college system was unable to provide much IT training, three Indian entrepreneurs took it upon themselves in 1982 to provide IT tutorials and training classes. Their early days were often marked with one of them driving a motorcycle and the other riding behind with a PC in his lap so that they could impart this training in some rented school space (in evenings and on weekends). Today, their institute (NIIT) is a multinational company that has helped build a substantial base of IT skills in India.

► In 1985, Texas Instruments set up an office in Bangalore with a direct satellite link to the U.S. By 1989, the government had also commissioned a direct 64Kbps satellite link to the U.S.,

**Economic hardships forced U.S. companies to reduce costs. As a result, they transferred even more IT work to India, thereby fueling the growth of an industry that was already growing.**

which offered software exporters a new way to transfer data and services and set the foundation for offshore business models that could compete with the onsite “body shopping.”

In 1993, the U.S. Immigration and Naturalization Service made changes that made it difficult to get B-1 visas. Furthermore, the U. S. Department of Labor required that companies applying for new H-1 visas needed to certify that prevailing market wages were being paid to immigrant workers. Also, Indian software professionals who were brought under the umbrella of the Immigration Act, had to pay Social Security and related taxes to the U.S. government, creating an additional burden on the employees as well as their employers. These factors led a few IT companies in India to adopt a mixed model, which satellite links had already enabled, and in which some software programmers would work at the client’s premises (in the U.S.) whereas others would continue to work in their offices in India. Nevertheless, the move to this new business model was gradual because cost savings for the onsite model were still quite large, and there were clearly advantages of being in close proximity to the client. Even today, some IT companies continue to follow the old model and send 15%–25% of their programmers to the U.S. and other developed countries.

By 1998, the IT industry in the U.S. and other developed countries was consumed by the Y2K problem, and two industries—telecommunications and

the Internet (with its associated dot-com start-ups)—were booming. This resulted in U.S. companies hiring increased numbers of computer programmers, and since the Y2K problem was mainly related to legacy software written in old languages like COBOL, India was one of the few countries that could still provide a sufficient number of such programmers. Consequently, the U.S. government was forced to increase its H-1 quota from 65,000 in 1998 to 130,000 in 1999 and then to 195,000 soon thereafter, and many Indian IT professionals moved temporarily or permanently to the U.S. However, in spite of the large influx of IT professionals, the U.S. industry still could not fulfill its programming needs and started outsourcing large amounts of programming and maintenance work to India.

By early 2000, the Y2K problem had been solved, and both the telecommunications and dot-com booms had suffered downturns. In 2001, the U.S. went into a recession and the U.S. government reduced its H-1 quota back to 65,000. Far from harming India, though, these events showed that offshore outsourcing grows in both good times and bad times. Economic hardships forced U.S. companies to reduce costs. As a result, they transferred even more IT work to India, thereby fueling the growth of an industry that was already growing exponentially.

The table here lists the revenue earned and the number of professionals employed by the Indian IT services industry during alternate years spanning 2001–2002 and projects numbers



for future years up to 2015–2016. These figures do not include professionals who are getting trained by their employers and also do not include revenues or the number of professionals related to other business process services or hardware products. The revenue numbers given in the first four columns have been sourced from the National Association of Software and Services Companies (NASSCOM; [www.nasscom.org](http://www.nasscom.org)); all other figures have been taken from Evalueserve ([www.evalueserve.com](http://www.evalueserve.com)), a global research and analytics firm.

Since India's GDP is expected to be \$1,100 billion in 2007–2008 and since this GDP is growing annually at an average of 8.5% in real terms and 14% in nominal terms, this GDP is likely to be \$2,400 billion in 2015–2016. Consequently, if the forecasts provided by Evalueserve regarding the Indian IT industry turn out to be true, then by 2015–2016, the number of professionals working in the IT industry would have grown tenfold (from 2001–2002 to 2015–2016), and, in nominal terms, the total revenue would have grown by 22 times, which would end up being approximately 8% of India's GDP.

To assess what may be ahead, consider the current and future status of three key elements of the Indian IT industry:

*Export of IT Services:* IT export services provided from India include custom application development, application management, information systems outsourcing, software and hardware development and support, training, education and helpdesks, IT consulting, systems integration, software testing, network consulting, and network integration. During the last 10 years, exports of these services have been growing at an annual rate of 32%. However, Evalueserve expects this growth rate to slow to approximately 19% in the next five to six years because of a lack of availability of enough talent, rising wages, and increased attrition. The U.S. and U.K. remain the largest export markets, accounting for approximately 61% and 18% of exports respectively in 2007–2008. However, IT exports have also been steadily increasing to other countries. In particular, IT exports to continental Europe have witnessed notable gains, growing at an annual rate of more than 55% during the period 2004–2007.

*Domestic Use of IT Services:* India

## The U.S. and U.K. remain the largest export markets, accounting for approximately 61% and 18% of exports respectively in 2007–2008. However, IT exports have also been steadily increasing to other countries.

was a closed economy until 1991 and the Indian government owned many banks and companies that had little or no use for IT. However, in 1991, its government started opening up and most Indian companies had to compete with both domestic and multinational companies that wanted to sell in India. Consequently, many such companies—including domestic banks, airlines, railways, telecommunications companies, and other government-owned companies—have become or are in the process of becoming avid users of IT. Hence, the domestic IT services industry has been growing at an annual rate of 41% during the last two years and is expected to continue growing at 5%–6% per year more than export services for the next seven to eight years. This implies that by 2015–2016, the number of IT professionals employed in the domestic IT industry would be comparable to that employed in the IT exports industry. The domestic IT industry, which contributed only 0.8% to India's GDP in 2006–2007, is likely to contribute 2.7% by 2015–2016. At present, internal company departments provide more than 90% of all domestic IT services. That provides a large opportunity market for third-party vendors, particularly as liberalization and globalization mean the types of domestic IT services provided within India are

similar to those found in the industrialized world.

*Import of IT Products and Services into India:* In India, the number of mobile phones has been increasing at approximately nine million per month, and the total number is likely to exceed 340 million by the end of 2008, thereby making India the second-largest mobile phone market after China. Interestingly, IBM is already servicing approximately 50% of the mobile phone subscriber base in India after signing three 10-year contracts with Bharti Airtel in 2004, Idea Cellular in 2006, and Vodafone Essar in 2007. Most of these agreements require IBM to consolidate, transform, and manage comprehensive infrastructure and applications, as well as to jointly develop marketing IT and telecommunications solutions and services. Clearly, such a move frees up these clients to do aggressive marketing, sales, and business development. Although IBM may not have been the least-expensive provider, it probably won these contracts because it was able to bring its intellectual property, products, and services from other parts of the world where it has already helped other very large telecommunications companies. Since Indian financial services (such as banks and insurance companies) and transportation (especially airlines) are also expanding and globalizing at a phenomenal pace, these sectors are likely to follow suit.

Across all parts of Indian IT, then, we see the synergistic impact of globalization. Globalization helps Indian IT companies to grow, while Indian IT is becoming a digital foundation for many globalizing firms. As the Indian economy becomes more integrated into the global economy, there is another two-way effect—more opportunity for global IT firms to sell to Indian clients. And, of course, more opportunity for Indian IT firms to sell globally. ■

This is the first of the "Emerging Markets" columns. Subsequent columns will address the roles of other emerging countries in the globalization of IT, including China, several Eastern European countries, the Middle East, and Latin America.

**Alok Aggarwal** ([alok.aggarwal@evalueserve.com](mailto:alok.aggarwal@evalueserve.com)) is the co-founder and chairman of Evalueserve, Inc. in Saratoga, CA.

© 2008 ACM 0001-0782/08/0700 \$5.00



## Legally Speaking

# Revisiting Patentable Subject Matter

*Is everything under the sun made by humans patentable subject matter?*

**A**RE BUSINESS METHODS and software algorithms patentable? Many of us think they shouldn't be. However, under a 1998 U.S. Federal Circuit Court of Appeals decision in *State Street Bank Bank v. Signature Financial Group*, they seem to be. That case opined that business methods could be patented and regarded any process conforming to a dictionary definition as patentable subject matter, as long as it produces a "useful, concrete, and tangible result." This would include program algorithms.

Because of *State Street Bank's* very broad interpretation of patentable subject matter, the U.S. Patent and Trademark Office (PTO) has been flooded with applications for patents on methods of all kinds, including business methods, methods of meditation, dating methods, sports moves, tax strategies, and even plots for novels. This capacious view of patentable subject matter may, however, be about to change.

This past February, the Federal Circuit decided to hear en banc (with the full court, not just the usual panel of three judges) an appeal by Bernard Bilski of a decision by the PTO Board of Patent Appeals and Interferences (BPAI) denying Bilski's application for a patent on a method for managing energy consumption risks owing to vagaries of the weather for failure to claim patentable subject matter.

The order announcing the en banc review invited interested parties to file amicus curiae (friend of the court) briefs to address not only whether Bilski's patent application should be granted, but also what test or standard should be used for judging what processes are eligible for patent protection. The order even asks whether the *State Street Bank* decision should be overturned.

### Bilski's Claim

Claim 1 of Bilski's application sets forth three steps of his method for energy risk management: initiating a series of transactions between a commodity provider and consumers of the commodity whereby consumers would purchase the commodity at a fixed rate based on historical averages (setting the risk position of the consumers); identifying market participants for the commodity who have a counter-risk position to that of consumers; and initiating a series of transactions between the commodity provider and market participants having a counter-risk position at a second fixed rate such that transactions of the market participants balance out the risks to consumers.

Bilski relies upon the *State Street Bank* decision in support of his claim. He asserts his claim recites a process and this method produces a useful, concrete, and tangible result. To understand why BPAI rejected Bilski's claim, a brief historical review is in order.

### The Supreme Court on Process Patents

The Supreme Court first considered whether computer program processes could be patented in its unanimous 1972 decision in *Gottschalk v. Benson*. Benson had applied for a patent on a method for transforming binary coded decimals to pure binary form. One claim called for implementing this algorithm in a programmed computer; a second was for the algorithm as such.

Section 101 of U.S. patent law states that "[w]hoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefore." Benson's algorithm was a process in the dictionary sense of the word, but that didn't necessarily mean it was a process within the meaning of section 101.

Under the Court's past decisions, patentable processes had been those that transformed matter from one physical state to another. Benson's process didn't do this. Past decisions had also excluded laws of nature, mathematical and scientific principles, mental processes, and abstract ideas from patent protection. Because Benson's method could be carried out in a person's head or with aid of paper and pencil, it seemed like a mental process or abstract idea, and perhaps a mathematical principle. The Court was also disturbed that Benson's

claims would cover (and therefore preempt) all uses of the algorithm, not just those applied to particular industrial ends.

In 1978, the Court revisited *Benson* in *Parker v. Flook*, which considered the patentability of a process for calculating and updating alarm limits for catalytic conversion plants. By a 6–3 majority, the Court rejected Flook’s claim because the only new thing about it was an algorithm onto which had been tacked conventional post-solution activity (updating the alarm limits).

Three years later, in *Diamond v. Diehr*, the Court once again considered the patentability of a process involving a computer program. In *Diehr*, the Court ruled by a 5–4 majority that a process for curing rubber, one step of which involved a computer program that continuously calculated temperatures inside the mold to determine when the rubber was properly cured, was a patentable process. Because *Diehr* did not abrogate *Benson* or *Flook* and involved a conventional process that transformed matter from one physical state to another, *Diehr* was initially viewed as a narrow decision for patenting software innovations.

The Federal Circuit, however, construed *Diehr* more broadly by focusing on its dicta that everything under the sun made by man should be considered patentable subject matter. For the past 27 years, this court has fashioned its own conception of patentable subject matter, culminating in the *State Street Bank* decision. Its recent willingness to reconsider *State Street Bank* and its approach to determining patentable subject matter may well be due to its sense that the Supreme Court is not satisfied with its rulings.

### Renewed Supreme Court Interest in Patents

In the past few years, the Supreme Court has reviewed several Federal Circuit decisions and reversed that court’s rulings every time. In *eBay v. MercExchange*, for example, the Court rejected the Federal Circuit’s rigid rule that courts must issue injunctions in virtually all patent infringement cases. In *KSR v. Teleflex*, the Court reinvigorated the nonobviousness standard for invention by rejecting the Federal Circuit’s approach to judging obvious-

ness. In *Microsoft v. AT&T*, the Court overturned a Federal Circuit ruling that Microsoft was liable for offshore patent infringement based on its shipment to another country of a disk of software containing a component that infringed a U.S. patent.

The Supreme Court has also signaled renewed interest in the Federal Circuit’s approach to patentable subject matter. In 2005, it accepted Lab Corp’s appeal from a Federal Circuit ruling that it had induced patent infringement by providing test results about levels of homocysteine in patients’ blood to doctors who then correlated elevated levels of this amino acid with vitamin deficiencies and enhanced risks of heart disease, thereby infringing Metabolite’s patent. Justice Breyer dissented from the decision to drop the *Lab Corp.* appeal because in his view, Metabolite’s patent claimed a process that “is no more than an instruction to read some numbers in light of medical knowledge.” He believed this ran afoul of the longstanding rule that laws of nature and natural phenomena cannot be patented.

A year later, during oral argument in the *Microsoft v. AT&T* case, five members of the Court asked questions about the patentability of computer programs, even though that issue was not before the Court. These questions revealed that patentable subject matter was on many Justices’ minds.

Given the Court’s renewed interest in patentable subject matter, the BPAI seems to have decided that it was time to push back on the Federal Cir-

cuit’s overbroad conception of patentable subject matter. So it rejected Bilski’s claims.

### BPAI on Patentable Subject Matter

*Bilski* sets forth a series of propositions about patentable subject matter. The first is that the constitutional subject matter for patent protection is inventions in the “useful arts,” by which the founders meant what today we would call “technological arts,” or more simply, “technology.” Section 101 names four kinds of technologies that are eligible for patent protection: processes, machines, manufactures, and compositions of matter.

BPAI thinks that Congress intended to make every human-made machine, manufacture, and composition of matter eligible for patent protection, but doubts it intended everything under the sun made by humans to be eligible for patent protection. Computer programs, documents, music, art, and literature are innovations made by humans; yet, none is eligible for patent protection.

Technological processes that transform matter from one physical state to another (for example, chemical A mixed with chemical B to produce chemical C) are clearly patentable, as are processes that are tied to specific mechanical implementations. But if processes do not transform matter and are not tied to technical implementations, BPAI thinks they are not technological enough to be patentable. Such methods should be considered “abstract ideas” that are excluded from patent protection, as was the algorithm in *Benson*.

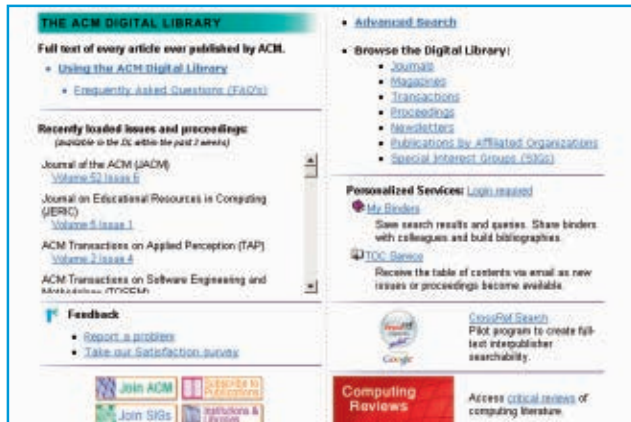
*Bilski* questions the Federal Circuit’s “useful, concrete, and tangible result” test for patentable subject matter as lacking an authoritative basis and a sound rationale. *Bilski* observes that Justice Breyer’s dissent in *Lab Corp.* questioned the “useful, concrete, and tangible result” test for patentable subject matter as not having been endorsed by the Court. (Indeed, this test is inconsistent with *Benson* and *Flook*.)

Because *Bilski*’s method wasn’t tied to an implementation in a specific device and didn’t transform matter from one physical state to another, the BPAI concluded it was an unpatentable

**The Federal Circuit has the opportunity in *Bilski* to clarify the standard by which to judge what processes are eligible for patent protection.**

# ACM Digital Library

www.acm.org/dl



## The Ultimate Online INFORMATION TECHNOLOGY Resource!

- Over 40 ACM publications, plus conference proceedings
- 50+ years of archives
- Advanced searching capabilities
- Over 2 million pages of downloadable text

Plus over one million bibliographic citations are available in the ACM Guide to Computing Literature

To join ACM and/or subscribe to the Digital Library, contact ACM:

Phone: 1.800.342.6626 (U.S. and Canada)  
+1.212.626.0500 (Global)  
Fax: +1.212.944.1318  
Hours: 8:30 a.m.-4:30 p.m., Eastern Time  
Email: acmhelp@acm.org  
Join URL: www.acm.org/joinacm  
Mail: ACM Member Services  
General Post Office  
PO Box 30777  
New York, NY 10087-0777 USA

process and an abstract idea. BPAI indicated its conclusion would be no different even if Bilski altered his claims to mention use of some technology, such as a telephone or computer, to carry out the method because the method was essentially still an abstract one akin to the claim in *Benson* for implementing the algorithm in a programmed computer.

### What Will Happen?

The Federal Circuit has the opportunity in *Bilski* to clarify the standard by which to judge what processes are eligible for patent protection and why this is the right standard. It also has the opportunity to give substance to the abstract idea exclusion from patent protection.

If the Federal Circuit affirms the BPAI rejection of Bilski's application and rules that Bilski's method is unpatentable as an abstract idea and/or as a non-technological process, the Supreme Court will probably be satisfied that the Federal Circuit has gotten the message that it should pay closer attention to the Court's prior rulings and narrow the scope of patentable subject matter.

If, however, the Federal Circuit reverses the BPAI's ruling in *Bilski* or is deeply split and issues multiple opinions expressing divergent theories about patentable subject matter, the Supreme Court will probably review the *Bilski* case to clarify what standards the PTO and Federal Circuit should apply in judging which processes are eligible for patent protection.

A decision upholding the unpatentability of Bilski's process will not do away with all software patents because some do claim technological processes, but many patents issued under the *State Street Bank* test, whether for software innovations, business methods, dating methods, and the like, would then be rendered ineffectual. As things go, this would be progress. This is another patent reform that can and should be carried out through the courts. ■

**Pamela Samuelson** (pam@law.berkeley.edu) is the Richard M. Sherman Distinguished Professor of Law and Information at the University of California, Berkeley.





## Kode Vicious

# Beautiful Code Exists, If You Know Where to Look

*Coding is his game, pleasantries distained.*

**Dear KV,**

I've been reading your rants in *Queue* for a while now and I can't help asking, is there any code you *do* like? You always seem so negative; I really wonder if you actually believe the world of programming is such an ugly place or if there is, somewhere, some happy place that you go to but never tell your readers about.

**A Happy Programmer**

**Dear Mr. Happy,**

While I will try not to take exception to your calling my writings "rants," I have to say that I am surprised by your question. KV is a happy, cheerful, outgoing kind of guy who not only has a "happy place," but also carries it with him wherever he goes, sharing joy and laughter with everyone around him and giving sweets to small children. (cough)

Now that I've bleached my brain, I can answer a bit more honestly. Yes, in fact, there are good systems and I have seen good code, sometimes even great code, in my time. I would like to describe one such chunk of good code right now. Unfortunately, it will require a bit of background to explain what the code is, but please stick with me. Perhaps you can relax by going to your "happy place" first.

One of my recent projects has been to extend support for something called hardware performance monitoring counters (hwpmc) on FreeBSD, the op-

erating system I work on. As the name indicates, hwpmc is implemented in hardware, and in this case hardware means on the CPU. I don't know if you've ever read CPU or chip documentation, but there is little in our industry that is more difficult to write or less pleasant to read. It's bad enough that the subject is as dry as the surface of the moon, but it's much worse because the people who write such documentation either don't understand the technology or are terrible writers, and often there is a fatal combination of the two. Starting from that base, the typical software engineer produces code that somewhat mirrors the specification, and as things that grow in poison soil themselves become poison, the code is often as confusing as the specification.

What is hwpmc? It is a set of counters that reside on the CPU that can record various types of events of interest to engineers. If you want to know if your code is thrashing the L2 cache or if the compiler is generating sub-optimal code that's messing up the pipeline, this is a system you want to use. Though these things may seem esoteric, if you're working on high-performance computing, they're vitally important. As you might imagine, such counters are CPU specific, but not just by company, with Intel being different from AMD: even the model of CPU bears on the counters that are present, as well as how they are accessed.

The sections covering hwpmc in Intel's current manual, *Intel® 64 and*

*IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2*, encompass 249 pages: 81 to describe the various systems on various chips and 168 to cover all the counters you can use on the chips. That's a decent-size novel, but of course without the interesting story line. Kudos to Intel's tech writers, as this is not the worst chip manual I have ever read, but I would still rather have been reading something else. Once I had read through all of this background material, I was a bit worried about what I would see when I opened the file.

But I wasn't too worried, because I knew the programmer who wrote the code personally. He's a very diligent engineer who not only is a good coder but also can explain what he has done and why. When I told him that I would be trying to add more chip models to the system he wrote, he sent me a 1,300-word email message detailing just how to add support for new chips and counters to the system.

What's so great about this software? Well, let's look at a few snippets of the code. It's important to always read the header files before the code, because header files are where the structures are defined. If the structures aren't defined in the header file, you're doomed from the start. Looking at the top of the very first header file I opened we see the code snippet shown in Figure 1. Why do these lines indicate quality code to me? Is it the capitalization? Spacing? Use of tabs? No, of course not! It's the

**Figure 1: Header file showing version numbers.**

```
#define PMC_VERSION_MAJOR 0x03
#define PMC_VERSION_MINOR 0x00
#define PMC_VERSION_PATCH 0x0000
```

fact that there are version numbers. The engineer clearly knew his software would be modified not only by himself but also by others, and he has specifically allowed for that by having major, minor, and patch version numbers. Simple? Yes. Found often? No.

The next set of lines—and remember this is only the first file I opened—were also instructive, as shown in Figure 2. Frequent readers of KV might think it was the comment that made me happy, but they would be wrong. It was the translation of constants into intelligible textual names. Nothing is more frustrating when working on a piece of software than having to

remember yet another stupid, usually hex, constant. I am not impressed by programmers who can remember they numbered things from 0x100 and that 0x105 happens to be significant. Who cares? I don't. What I want is code that uses descriptive names. Also note the constants in the code aren't very long, but are just long enough to make it easy to know in the code which chip we're talking about.

Figure 3 shows another fine example from the header file. I've used this snippet so I can avoid including the whole file. Here, machine-dependent structures are separated from machine-independent structures. It would seem obvious that you want to separate the bits of data that are specific to a certain type of CPU or device from data that is independent, but what seems obvious is rarely done in practice. The fact that the engineer thought about which bits should go where indicates a

high level of quality in the code. Read the descriptive comments for each element and the indication of where the proper types can be found, as in the case of `pmd_cputype` being from `enum pmd_cputype`.

One final comment on this file. Note that the programmer is writing objects in C. Operating system kernels and other low-level bits of code are still written in C, and though there are plenty of examples now of people trying to think differently in this respect (such as Apple's Mac OS X drivers being written in C++) low-level code will continue to be written in C. That does not mean programmers should stop using the lessons they learned about data encapsulation, but rather that it is important to do the right thing when possible. The structure listed here is an object. It has data and methods to act upon it. The BSD kernels have used this methodology for 20-plus years at this point, and it's a lesson that should be learned and remembered by others.

These are just a few examples from this code, but in file after file I have found the same level of quality, the same beautiful code. If you're truly interested in seeing what good code looks like, then I recommend you read the code yourself. If any place is a "happy place," it is in code such as this.

**KV**

P.S. Complete code cross-references for many operating system kernels, including FreeBSD, can be found at <http://fxr.watson.org/> and the code you're looking for can be found at: <http://fxr.watson.org/fxr/source/dev/hwpmc/>.

A similar set of cross-references can be found on the codespelunking site: <http://www.codespelunking.org/frebsd-current/htags/>.

**Dear KV,**

In his book *The Mythical Man-Month*, Frederick P. Brooks admonishes us with grandfatherly patience to plan to build a prototype—and to throw it away. You will anyway.

At one point this resulted in a fad-of-the-year called prototyping (the programming methodology formerly known as trial and error), demonstrating that too little and too much are equally as bad.

**Figure 2: Translation of constants into descriptive names.**

```
/*
 * Kinds of CPUs known
 */

#define __PMC_CPUS() \
__PMC_CPU(AMD_K7, "AMD K7") \
__PMC_CPU(AMD_K8, "AMD K8") \
__PMC_CPU(INTEL_P5, "Intel Pentium") \
__PMC_CPU(INTEL_P6, "Intel Pentium Pro") \
__PMC_CPU(INTEL_CL, "Intel Celeron") \
__PMC_CPU(INTEL_PII, "Intel Pentium II") \
__PMC_CPU(INTEL_PIII, "Intel Pentium III") \
__PMC_CPU(INTEL_PM, "Intel Pentium M") \
__PMC_CPU(INTEL_PIV, "Intel Pentium IV")
```

**Figure 3: Machine-dependent structures are separated from machine-independent structures.**

```
/*
 * struct pmd_mdep
 *
 * Machine dependent bits needed per CPU type.
 */

struct pmd_mdep {
    uint32_t pmd_cputype; /* from enum pmd_cputype */
    uint32_t pmd_npmc; /* max PMCs per CPU */
    uint32_t pmd_nclass; /* # PMC classes supported */
    struct pmd_classinfo pmd_classes[PMC_CLASS_MAX];
    int pmd_nclasspmcs[PMC_CLASS_MAX];

    /*
     * Methods
     */

    int (*pmd_init)(int_cpu); /* machine dependent initialization */
    int (*pmd_cleanup)(int_cpu); /* machine dependent cleanup */
}
```

What is your view of creating prototypes, and particularly on the question of how faithful a prototype needs to be to resolve the really tricky details, as opposed to just enabling the marketing department to get screenshots so they can strut the stuff?

Signed,

**An (A)typical Engineer**

**Dear Atypical,**

What do you mean by “formerly known as trial and error”?! Are you telling me that this fad has died? As far as I can tell, it’s alive and well, though perhaps many of its practitioners don’t actually know their intellectual parentage. Actually, I suspect most of its practitioners can’t spell *intellectual parentage*.

Alas, it is often the case that a piece of good advice is taken too far and becomes, for a time, a mantra. Anything repeated often enough seems to become truth. Mr. Brooks’ advice, as I’m sure you know, was meant to overcome the “it must be perfect” mantra that is all too prevalent in computer science. The idea that you can know everything in the design stage is a fallacy that I think started with the mathematicians, who were the world’s first programmers. If you spend your days looking at symbols on paper, and then only occasionally have to build those symbols into working systems, you rarely come to appreciate what happens when the beauty of your system meets the ugly reality that is real hardware.

From that starting point, it’s easy to see how programmers of the 1950s and 1960s would want to write everything down first. The problem is that a piece of paper is a very poor substitute for a computer. Paper doesn’t have odd delays introduced by the speed of electrons in copper, the length of wires, or the speed of the drum (now disk, soon to be flash). Thus, it made perfect sense at the time to admonish people just to build the damned thing, no matter what it was, and then to take the lessons learned from the prototype and integrate them into the real system.

The increasing speeds of computers since that advice was first given have allowed people to build bigger, faster, and certainly more prototypes in the same amount of time that a single system could have been built in the past.

**The idea that you can know everything in the design stage is a fallacy that I think started with the mathematicians, who were the world’s first programmers.**

The sufferers of prototypitis are really just chicken. Not putting a line in the sand is a sign of cowardice on the part of the engineer or team. “This is just a prototype” is too often used as an excuse to avoid looking at the hard problems in a system’s design. In a way, such prototyping has become the exact opposite of what Mr. Brooks was trying to do. The point of a prototype is to find out where the hard problems are, and once they are identified, to make it possible to finish the whole system. It is not to give the marketing department something pretty to show potential customers—that’s what paper napkins and lots of whiskey are for.

Where do I stand on prototypes? The same place that I stand on layering or the breaking down of systems into smaller and smaller objects. You should build only as many prototypes as are necessary to find and solve the hard problems that result from whatever you’re trying to build. Anything else is just navel-gazing. Now, don’t get me wrong, I like navel-gazing as much as the next guy, perhaps more, but what I do when I delve into my psychedelia collection has nothing, I assure you, to do with writing software.

**KV**

**George V. Neville-Neil** ([kv@acm.org](mailto:kv@acm.org)) is the proprietor of Neville-Neil Consulting. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.



# ICL2008



## Conference chair

**M. E. Auer**  
(CTI Villach, Austria)

## Program committee

### Senior Members

**A. Y. Al-Zoubi**,  
PSUT Amman, Jordan,  
**Peter Baumgartner**,  
Danube University Krems, Austria  
**Christos. Bouras**  
University and CTI Patras, Greece  
**Rhena Delpport**, University of  
Pretoria, South Africa  
**Christian Dorninger**,  
BMBWK Vienna, Austria  
**Arthur Edwards**, Universidad de  
Colima, Mexico  
**David Guralnick**, Kaleidoscope  
Learning New York, USA  
**Martin Hitz**,  
University of Klagenfurt, Austria  
**Josef Hvorecky**,  
VSM Bratislava, Slovakia  
**Trayan Iliev**, University of Sofia,  
Bulgaria  
**George Ioannidis**,  
University of Patras, Greece  
**Göran Karlsson**,  
KTH Stockholm, Sweden  
**Barbara Kerr**, Concordia University  
Montreal, Canada  
**Andreas Pester**, Carinthia Tech  
Institute Villach, Austria  
**Serge Ravet**, European Institute for  
E-Learning, France  
**Rob Reilly**, MIT Media Lab, USA  
**Cornel Samoila**, University  
Transylvania Brasov, Romania  
**Wolfgang Scharl**,  
Technikum Vienna, Austria  
**Jeanne Schreurs**, University of  
Hasselt, Belgium  
**Thomas Schmidt**,  
HAW Hamburg, Germany  
**Richard Straub**,  
IBM, France and ELIG  
**Andras Szucs**,  
EDEN, TU Budapest, Hungary  
**Linmi Tao**, Tsinghua University  
Peking, China  
**Doru Ursutiu**, University  
Transylvania Brasov, Romania

## More information

<http://www.icl-conference.org>  
[info@icl-conference.org](mailto:info@icl-conference.org)  
phone: +43-4242-90500-2115

## Registration

<http://www.conftool.com/icl-conference/>

## FIRST CALL FOR PAPERS

International Conference  
Villach/Austria  
■ Interactive Computer Aided Learning **ICL**

**11<sup>th</sup> International Conference on Interactive Computer aided Learning**  
[www.icl-conference.org](http://www.icl-conference.org)

**September 24 – 26, 2008, Villach, Austria**

This interdisciplinary conference aims to focus on the exchange of relevant trends and research results as well as the presentation of practical experiences in interactive computer aided learning. Therefore pilot projects, applications and products will also be welcome.

This conference will be organized by the **Carinthia University of Applied Sciences, Villach/Austria** in cooperation with:

- Federal Ministries for Science, Education and Culture of Austria,
- IEEE Education Society,
- European Distance and E-learning Network (EDEN)
- International E-Learning Association (IELA)
- European Learning Industry Group (ELIG)
- International Society of Engineering Education (IGIP)
- International Association of Online Engineering (IAOE)
- European Inst. of E-Learning (EifEL)
- Austrian Computer Society (OCG)
- IT Campus Carinthia

## Topics of Interest

**ICL2008 will have a focus on Educational MashUps, Collaborative Learning Environments as well as E-Portfolios.** General topics are for example:

- Web based learning (WBL)
- Life long learning
- Intelligent content
- Adaptive and intuitive environments
- Responsive environments
- Mobile learning environments and applications
- Computer aided language learning incorporating AI techniques (ICALL)
- Platforms and authoring tools
- Educational MashUps
- Networks/Grids for learning
- Knowledge management and learning
- Educational Virtual Environments
- Collaborative learning
- Applications of the Semantic Web
- E-Portfolios
- Standards and style-guides
- Remote and virtual laboratories
- Multimedia applications and virtual reality
- Pedagogical and psychological issues
- Evaluation and outcomes assessment
- New learning models and applications
- Cost-effectiveness
- Real world experiences
- Pilot projects / Products / Applications
- ... and others

## Types of Contributions

- **Full Papers** (20 minutes presentation followed by a panel discussion)
- **Short Papers** (15 minutes presentation)
- **Interactive Demonstrations** (15 minutes, also on-line demonstrations)
- **Posters**

All submissions are subject to a double blind reviewing process.

Other opportunities to participate:

- **Run a workshop or tutorial.**
- **Organize a thematic session.** Proposals should include a minimum of three papers.
- **Exhibit** at ICL (products and developments of e-learning technology)

## Important Dates

**May 16, 2008**  
**June 20, 2008**  
**Sept. 10, 2008**  
**Sept. 24-26, 2008**

**Submission of extended abstracts** (two pages)  
**Notification of acceptance**  
**Camera-ready due**  
**Conference ICL2008**

## Proceedings

All accepted submissions will be published in the conference proceedings (ISBN 978-3-89958-353-3).



DOI: 10.1145/1364782.1364792

Stephen J. Andriole Eric Roberts

## Point/Counterpoint Technology Curriculum for the Early 21st Century

*In case you missed IT, the world has changed.*

### Point: Stephen J. Andriole

**T**HE FIELD OF information technology is changing and those responsible for educating the next generation of technology (information systems, computer science, and computer engineering) professionals have responded with curriculum that fails to address the depth, speed, or direction of these changes. If we want our students to enjoy productive and meaningful careers, we need to radically change the content of the curriculum of our technology majors.<sup>a</sup>

### Change

The structure of the hardware/software/services technology industry is changing—morphing quickly from a set of fragmented hardware and software activities and vendors to a hardware/software/service provider model dominated by a shrinking number of vendors. IBM, Microsoft, HP, Dell, Intel, Oracle, Cisco, Accenture, EDS, CSC, and a few other companies are included in this \$10B+ in revenue per year group.<sup>b</sup>

a. A more extensive discussion of these issues can be found in Stephen J. Andriole, “Business Technology Education in the Early 21st Century: The Ongoing Quest for Relevance,” *Journal of Information Technology Education*, September 2006. I am referring here primarily to our undergraduate educational efforts.  
b. Matthew Aslett, “CBR 50 Largest IT Vendors,” *Computer Business Review*, July 19, 2006.

Is it improper to profile what these companies do and reverse engineer curricula? Most of these companies have robust R&D programs, manufacture hardware and software, and solve industry problems with technology. Their activities might well provide a useful—and obviously relevant—curriculum roadmap.

Another major trend is the standardization of software packages as the primary platform on which large enterprises compute and communicate. The software necessary to connect disparate software is no longer exclusively defined as proprietary middleware; instead, it’s embedded in applications by the major vendors through interoperability standards based on Web Services and its extensions, service-oriented architecture (SOA), and event-driven architecture (EDA). Software is also installed less as more more companies rent applications from hosting vendors like Salesforce.com, Microsoft, and now even SAP. Many CIOs really want to get out of the enterprise software acquisition, deployment, or support business: the demand curve for software-as-a-service (SaaS) is steep.<sup>c</sup>

c. SaaS is growing in popularity as more companies appreciate the benefits of renting software. This avoids the in-house implementation phase and large enterprise software licensing fees. Industry analysts from the Gartner Group and Forrester Research, among others, report that by 2012 25% of all software will be rented. The decline of proprietary software will also

Relatively few vendors will produce most of the world’s mainstream software in the coming years. The standardization of software will result in a concentration of software suppliers complying with a set of expanding integration and interoperability standards incarnated in evolving (service-oriented and event-driven) architectures. Just look at the mergers and acquisitions that have occurred in the software industry over the past few years. How many business intelligence (BI) vendors are independent? How many enterprise resource planning (ERP) vendors are left? Finally, greater amounts of software will exist only on servers accessed by increasingly thin clients. “Thin clients”—which have no local processing—will replace many “fat clients”—machines with lots of software, processing power, and storage.

When we layer outsourcing trends onto software trends, we see industry turning to offshore providers to satisfy their operational support requirements rather than U.S.-educated professionals who are not receiving enough of the knowledge or skills that industry values (or is willing to pay for, compared to offshore labor rates). Today those requirements are relatively low-level operational requirements but over time offshore providers will climb the food chain to more strategic technology capabilities. It’s these latter areas that should

be accelerated by the rising adoption of open source software.

catch the attention of U.S. educators preparing their students for technology careers since the sourcing battle for technology infrastructure and support is all but over—those who seek support for computing and communications infrastructures are driven more by labor rates than tradition, more by the advantages of commoditization than by customization. In fact, methodologies like ITIL (Information Technology Infrastructure Library) and COBIT (Control Objectives for Information and Related Technology) increasingly provide the means to cost-effectively manage and optimize infrastructures, making the infrastructure support business even less generous to technology professionals.

There are other trends changing the industry. R&D outsourcing is expanding. Data mining has become customer profiling, customization, and personalization. Supply chains are becoming transparent and have gone global. Real-time dynamic pricing (via intelligent rules engines) is spreading. Adding to this is the convergence of all things digital.

Where does technology curriculum address all of these trends? Where are the academic programs and certificates in SOA, EDA, hosting, SaaS, integration and interoperability, Web 2.0, Web 3.0, thin-client architecture, Web Services, open source software, sourcing and technology performance management? Where do students learn about

interoperable architectures, roaming connectivity, real-time processing, rich converged media, user-generated content, global supply chain optimization, full-view business intelligence, predictive analytics, master data management, and crowdsourcing-based problem-solving?

### Response

Several curriculum changes and guidelines have been proposed that attempt to address the changes in technology and design optimal pedagogical approaches in response to these changes. The Joint Task Force for Computing Curricula on Computing Curricula for the early 21st century identified five areas of computing degree concentrations: computer engineering, computer science, information systems, information technology, and software engineering.

These areas represent the academic programs that the Joint Task Force believes represent the state of the field and the educational outcomes our students should pursue. They've identified a suite of "computing" and "non-computing" areas that students in each of the five areas should understand. The list of knowledge and skills areas identified by the Joint Task Force that defines the components of the five areas was derived from academic programs and curricula that have evolved over a long period of time. I collected

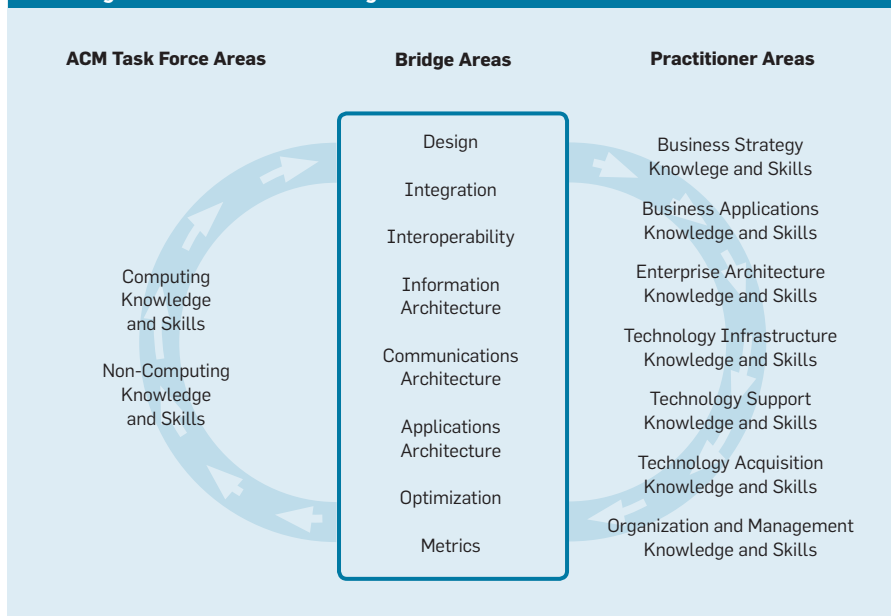
some data that also identified knowledge and skills areas—but from a practitioner's perspective.<sup>d</sup>

The table here presents the two sets of knowledge/skills areas side-by-side. The contrast is dramatic. The Joint Task Force's list barely correlates with the list developed from the practitioner surveys. Academic programs should acknowledge the widening gap between theory and practice, especially since it has enormous impact on their students' employment prospects. Regardless of what we call the academic majors and degrees, it's the content of each degree's curriculum that will determine our students' ability to find gainful employment.

One of the most important corporate knowledge areas today—in fact, the essence of business technology convergence—is enterprise architecture. Enterprise business-technology architecture is the linchpin among business strategy, strategic applications, technology infrastructure, and technology support. As business is enabled by technology and technology defines new business models and processes, the importance of enterprise business-technology architecture is increasing. This emerging core competency for the practice of the technology profession is unrepresented in the Joint Task Force's list of knowledge and skills areas—though it is a huge area in our practitioner survey. Similarly, business technology optimization is an opportunity area for educators. Increasing numbers of companies are struggling to optimize the performance of their software applications, networks, database management platforms, and infrastructure.

d. During the period from 2002–2005, an online survey sponsored by the Cutter Consortium (a technology industry research organization; [www.cutter.com](http://www.cutter.com)) collected data from Chief Information Officers (CIOs), Chief Technology Officers (CTOs), technology managers, Chief Executive Officers (CEOs), Chief Financial Officers (CFOs), technology consultants and vendors about the content of the field, the skill sets necessary to succeed, and the technologies most likely to be applied, neglected, or decommissioned. Over 1,000 professionals responded to the survey. The survey data was subsequently presented to—and validated by—the Villanova University CIO Advisory Council, which consists of 25 CIOs from the Philadelphia, PA region.

### Knowledge and skills areas and bridges.





## Recommendations

While distinctions between computer engineering (CE) and the other disciplines are relatively easy to appreciate—especially because of the role that hardware plays in CE programs—the differences between information systems, information technology, software engineering, and computer science are much more difficult to understand and define, especially when we reference the changes occurring in the field. I believe there should be three flavors: computer engineering, computer science, and information systems.<sup>e</sup>

CS programs should focus less on alternative programming languages and more on architectures, integration, and interoperability; less on algorithms and discrete structures and more on software engineering best practices. SOA and EDA should be owned by CS curricula as should Web 2.0 and 3.0, SaaS, thin client architecture, digital security, open source software, interoperable architectures, roaming connectivity, near-real-time processing, and rich converged media, among other related areas. Programming? Who programs? And where does—and more importantly, will—programming occur? Programming will ultimately evolve to component assembly and components will be generated by relatively few professionals located in the U.S., Bangalore, Moscow, and Shanghai working for IBM, Oracle, SAP, Microsoft, Tata, Infosys, and Google. Put another way, is “programming” the core competency of computer science?

Of course there will be programming jobs for our students. But the number of those jobs will decline, become more specialized, and distributed across the globe. A simple metric: how many Fortune 1000 companies still hire programmers? In the 1980s and 1990s, companies like CIGNA—where I was CTO—had hundreds of programmers on staff. Today, Fortune 1000 companies have far fewer programmers than they did because of the rise of packaged applications and the labor-rate-driven sourcing options they

e. The focus here is on the relationship between computer science and information systems; CE will likely remain primarily hardware focused and in engineering colleges within the nation's universities.

## ACM Joint Task Force knowledge and skills areas and practitioner areas.

### ACM Task Force Areas

#### Computing Knowledge and Skills

- ▶ Programming Fundamentals
- ▶ Integrative Programming
- ▶ Algorithms and Complexity
- ▶ Computer Architecture and Organization
- ▶ Operating Systems Principles and Design
- ▶ Net Centric Principles and Design
- ▶ Platform Technologies
- ▶ Theory of Programming Languages
- ▶ Human-Computer Interactions
- ▶ Graphics and Visualization
- ▶ Intelligent Systems (AI)
- ▶ Information Management (Database) Theory
- ▶ Information Management (Database) Practice
- ▶ Scientific Computing (Numerical Methods)
- ▶ Legal/Professional/Ethics/Society
- ▶ Information Systems Development
- ▶ Analysis of Technical Requirements
- ▶ Engineering Foundations for Software
- ▶ Engineering Economics for Software
- ▶ Software Modeling and Analysis
- ▶ Software Design
- ▶ Software Verification and Validation
- ▶ Software Evolution (Maintenance)
- ▶ Software Process
- ▶ Software Quality
- ▶ Computer Systems Engineering
- ▶ Digital Logic
- ▶ Distributed Systems
- ▶ Security: Issues and Principles
- ▶ Security: Implementation and Management
- ▶ Systems Administration
- ▶ Systems Integration
- ▶ Digital Media Development
- ▶ Technical Support

#### Non-Computing Knowledge and Skills

- ▶ Organizational Theory
- ▶ Management of Information Systems Organization
- ▶ Decision Theory
- ▶ Organizational Behavior
- ▶ Organizational Change Management
- ▶ E-business
- ▶ General Systems Theory
- ▶ Risk Management (Project, Safety Risk)
- ▶ Project Management
- ▶ Analysis of Business Requirements
- ▶ Embedded Systems
- ▶ Circuits and Systems

### Practitioner Areas

#### Business Strategy Knowledge and Skills

- ▶ Collaboration
- ▶ Customization and Personalization
- ▶ Supply Chain Management
- ▶ Business and Technology Convergence Strategy
- ▶ Competitor Intelligence
- ▶ Business Process Management

#### Business Applications Knowledge and Skills

- ▶ Business Application Optimization
- ▶ Core Business Applications Management
- ▶ Business Analytics

#### Enterprise Architecture Knowledge and Skills

- ▶ Applications Architectures
- ▶ Data Architectures
- ▶ Security Architectures
- ▶ Business Scenario Development
- ▶ Enterprise Technology Architecture Modeling
- ▶ Enterprise Architecture

#### Technology Infrastructure Knowledge and Skills

- ▶ Messaging/Workflow/Calendar
- ▶ Automation
- ▶ Database/Content/ Knowledge Management

#### Technology Support Knowledge and Skills

- ▶ Desktop/Laptop/PDA/Thin Client Support
- ▶ Data Center Operations
- ▶ Server Farm Design and Maintenance
- ▶ Network Design and Support
- ▶ Security and Privacy
- ▶ Procurement and Asset Management
- ▶ Asset Disposal

#### Technology Acquisition Knowledge and Skills

- ▶ Business Technology Acquisition Strategy
- ▶ RFP and SLA Development

#### Organization and Management Knowledge and Skills

- ▶ Reporting Relationships
- ▶ Centralization and Decentralization
- ▶ Governance

now have. This trend will accelerate resulting in fewer programming jobs for our students. Should we continue to produce more programmers?

In addition to the basics like data communications, database management, and enterprise applications, 21st-century IS programs should focus on business analytics, supply-chain optimization, technology performance management, business process modeling, full-view business intelligence, sourcing, and large amounts of technology management skills—in short, many of the items on the list of practitioner knowledge and skills.

CS programs can enable IS programs. The knowledge and skills areas proposed by the Joint Task Force should be extended to link to the knowledge and skills on the IS side. Clearly, the programs need to be coordinated—if we want to produce marketable human products.<sup>f</sup> The figure here suggests how this might work. The Joint Task Force

f. Most CS and IS programs exist on islands in most universities. They seldom coordinate curricula and generally have relatively little contact.

knowledge and skills areas appear on the left and the practitioner knowledge and skills appear on the right side of the figure. In the middle are some “bridges” that might shrink the gap between the two areas. These bridges might become required for both CS and IS curricula and help CS programs become more relevant and IS programs more grounded in the enabling technology that supports business processes and transactions.

The essence of these suggestions is that CS and IS curriculum must dramatically change if we are to help our students compete. What was technologically significant 10 years ago is not nearly as significant today: hardly anyone needs to know how to program in multiple languages or craft complex, elegant algorithms that demonstrate alternative paths to the same computational objective. We know more about what software needs to do today than we did a decade ago—and you know what? There’s less to do and support. This is the effect standards and commoditization have on an industry.

Our job as educators is to prepare students for the technology world-to-be, not the-one-that-was. A simple way to design

new CS and IS curriculum is to observe what practitioners do today, project what they’ll do tomorrow, and then identify the requisite enabling technologies (which will lead to new CS curriculum) and applied technologies and best practices (which will lead to new IS curriculum). I have attempted to energize this process by contrasting the Joint Task Force and practitioner knowledge and skills areas. I believe strongly in relevance-driven education and training, but also realize that not everyone believes education and training are closely related or that universities are responsible for preparing students for successful careers. Many believe the creation and communication of selected knowledge—regardless of its relevance to practice or professional careers—is the primary role of the modern university.

Differences of opinion are usually healthy, so let the debate begin. ■

**Stephen J. Andriole** ([stephen.andriole@villanova.edu](mailto:stephen.andriole@villanova.edu)) is the Thomas G. Labrecque Professor of Business at Villanova University where he conducts applied research in business-technology convergence.

© 2008 ACM 0001-0782/08/0700 \$5.00

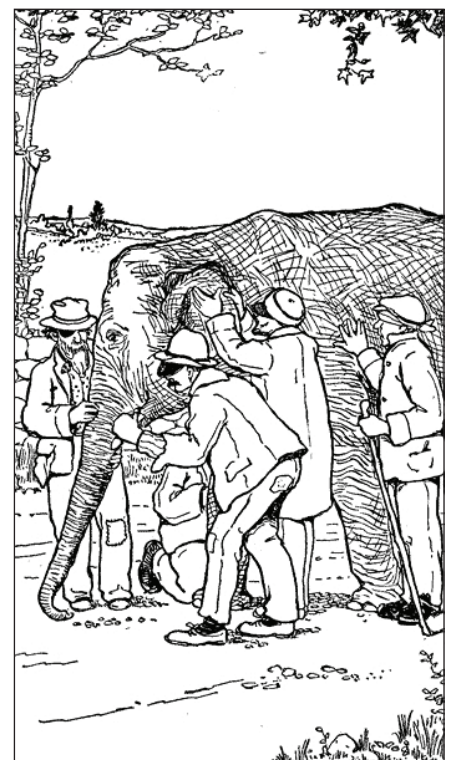
### Counterpoint: Eric Roberts

**A**S I READ Stephen Andriole’s critique of computing education, I was reminded of the classic South Asian folk tale of the blind men and the elephant. You know the story: six blind men each try to describe an elephant after touching only a part of it. The trunk is like a snake, the tail is like a rope, the ear is like a fan, and so on. Each description contains a kernel of truth, but none comes close to capturing the reality of the elephant as a whole.

Andriole’s characterization of computing in the early 21<sup>st</sup> century suffers from much the same failing in that it attempts to generalize observations derived from one part of the field to the entire discipline. He begins by observing, correctly, that the last few years have seen increasing “standardization of software packages as the primary platform on which large enterprises

compute and communicate.” But enterprise software is only part of the computing elephant. Computing is integral to many sectors of the modern economy: entertainment, education, science, engineering, medicine, economics, and many more. In most of those sectors, software is far from being a commodity product. Innovation in these areas continues to depend on developing new algorithms and writing the software necessary to make those algorithms real.

As an example, software development remains vital in the video game industry, which accounts for more than \$10 billion a year in revenue. This sector is looking for people with an entirely different set of skills than those Andriole enumerates in his survey of “professionals” in the field—a category that he restricts largely to senior management concerned with enterprise-level information technology. That the hiring criteria of a CIO for a Fortune 500 company would differ from those of a video



The Blind Men and the Elephant

game developer is hardly surprising. The two are looking at different parts of the elephant.

And what does the video game industry look for in its technology hires? As much as anything, video game companies are in the market for people with strong programming skills. At the 2007 conference on Innovation and Technology in Computer Science Education (ITiCSE) in Dundee, Scotland, keynote speaker Chris van der Kuyl, Scotland's leading entrepreneur in the video game industry, assured his audience that the greatest single factor limiting growth in his sector is a shortage of programming talent.

That any segment of the industry might be starved for programming talent will likely come as a surprise to someone who sees programming as a soon-to-be-obsolete skill. "Programming? Who programs?" Andriole asks, with rhetorical flourish. The answer, of course, is that millions of people around the world are productively engaged in precisely that activity.

Contrary to the impression Andriole creates in his column, there is no evidence that the demand for highly skilled software developers is declining. The agencies charged with predicting employment trends expect a substantial increase in employment for people with software development skills. The Bureau of Labor Statistics, in its December 2007 report *Employment Projections: 2006–16*, identifies "network systems and data communications analyst" as the single most rapidly growing occupational category over the next decade, with "computer software engineers, applications" in fourth place on that same survey. This data is hardly suggestive of a job category in decline.

Employment projections are by no means the only evidence of continued demand for people with software development skills. Business leaders from the top software companies routinely cite the shortage of technical expertise as the biggest stumbling block they face. Consider, for example, the following remarks by Microsoft chairman Bill Gates in a February 19, 2008 op-ed article for the *San Jose Mercury News*: "Today, there simply aren't enough people with the right skills to fill the growing demand for computer scientists and computer engineers. This is a critical problem because technology holds the key to progress,

and to addressing many of the world's most pressing problems, including health care, education, global inequality and climate change." Other industry leaders—including Rick Rashid at Microsoft (see his column in this issue) and Google founders Larry Page and Sergey Brin—have raised similar concerns.

It is clear from such responses that not everyone in the computing industry shares Andriole's conviction that traditional software-development skills are no longer relevant. Even so, industry leaders across all sectors nonetheless have something in common: they cannot find enough people with the skills they seek. Faced with a shortfall in the hiring pipeline, it is perhaps natural to argue that educational institutions should stop wasting time on other aspects of the discipline and focus on the skills that are just right for one particular environment. That argument would have merit if there were an imbalance between supply and demand, with too many degree recipients trained for some occupations while other jobs went begging. That situation, however, does not exist in the computing industry today. There is a shortfall across the board, with not enough graduates to supply any of the major subdisciplines.

The most powerful illustration I have seen documenting the magnitude of this shortfall comes from a talk presented by John Sargent, Senior Policy Analyst for the Department of Commerce, at a February 2004 research conference sponsored by the Computing Research Association (CRA). The figure here combines the data from several of Sargent's slides into a single graphic that plots statistics on degree production against the anticipated annual demand for people with those degrees. As you can see from the leftmost set of bars, the projected annual number of job openings for engineers is approximately two-thirds the number of bachelor's degrees produced each year. The situation in the physical sciences is similar at a somewhat smaller scale. In biology, by contrast, the annual number of job openings is only about 10% of the number of bachelor's degrees. This situation suggests an oversupply that allows for increased selectivity on the part of employers, who are unlikely to hire biologists without advanced degrees.

The bar graph for computer science at the right of the figure, however, reveals an entirely different situation. According to projections from the Bureau of Labor Statistics, the number of job openings for computer science exceeds the number of people receiving bachelor's degrees by almost a factor of four. Even if the industry were to hire every computer science graduate it would still have to look elsewhere for most of its new hires. That, indeed, is precisely what is happening. According to data presented by Caroline Wardle of the National Science Foundation at the CRA Snowbird conference in 2002, less than 40% of employees in computing-related jobs have computing degrees—a figure that stands in dramatic contrast to most other disciplines in which a degree in the field is part of the entry requirements. It is not that employers *prefer* candidates without formal training, but simply that there are nowhere near enough qualified graduates to satisfy the demand.

The problem that we face in computing education, therefore, is to increase the number of students. We cannot do that by arguing that only certain computing fields are worthy. The shortfall exists across the entire field. We need more students in each of the disciplines identified by the Joint ACM/IEEE-CS Task Force on Computing Curricula: computer science, computer engineering, software engineering, information systems, and information technology. Andriole would have us abandon software engineering, despite the fact that *Money* magazine recently put "software engineer" in first place in a list of the best jobs in the U.S. and despite the fact that the Bureau of Labor Statistics identifies "software engineer, applications" as one of the fastest-growing job categories.

Unfortunately, one of the biggest challenges that the ACM faces in its efforts to increase student interest in computing careers is precisely to counter the mythology about the dangers of offshoring that Andriole perpetuates in his column. His assertion that "programming will ultimately...be generated by relatively few professionals" largely located in places like Bangalore, Moscow, and Shanghai validates the fears so many high-school students express that computing careers will vanish as software development moves overseas.



The 2006 ACM report on *Globalization and Offshoring of Software*—a report to which Andriole contributed—finds no evidence to support this view. If anything, the opening of the offshore labor market in computing seems to have increased the number of computing jobs in the U.S., as illustrated by the following paragraph from the Executive Summary: “The economic theory of comparative advantage argues that if countries specialize in areas where they have a comparative advantage and they freely trade goods and services over the long run, all nations involved will gain greater wealth....This theory is supported to some extent by data from the U.S. Bureau of Labor Statistics (BLS). According to BLS reports, despite a significant increase in offshoring over the past five years, more IT jobs are available today in the U.S. than at the height of the dot-com boom. Moreover, IT jobs are predicted to be among the fastest-growing occupations over the next decade.”

The reality is that the shortage of people with the expertise industry needs is so severe that companies will go anywhere in the world that can provide workers with the necessary skills. If those people exist in Bangalore, Moscow, or Shanghai, then companies will hire them there. And if those people exist in the U.S., those same companies will hire them here.

Unfortunately, all too many people seem to believe that companies always seek to minimize labor costs, typically by employing workers at the lower salaries that prevail in developing countries. That view, however, represents a fundamental misunderstanding of labor economics. Companies are not primarily concerned with minimizing costs; after all, they could accomplish that goal by shutting down. Companies are in the business of maximizing return.

A simple thought experiment will make this difference clear. Suppose you are Microsoft and are looking to hire people with stellar software development skills. One of the candidates you

are considering is a recent graduate from a top-notch Silicon Valley university; given current salaries in the U.S., the cost of hiring this candidate might be, considering benefits and structural costs, approximately \$200,000 a year. You have another candidate in Bangalore who will cost you only \$75,000. Both candidates seem extraordinarily well qualified and show every sign of being extremely productive software engineers, capable of generating perhaps \$1,000,000 in annual revenue. What do you do?

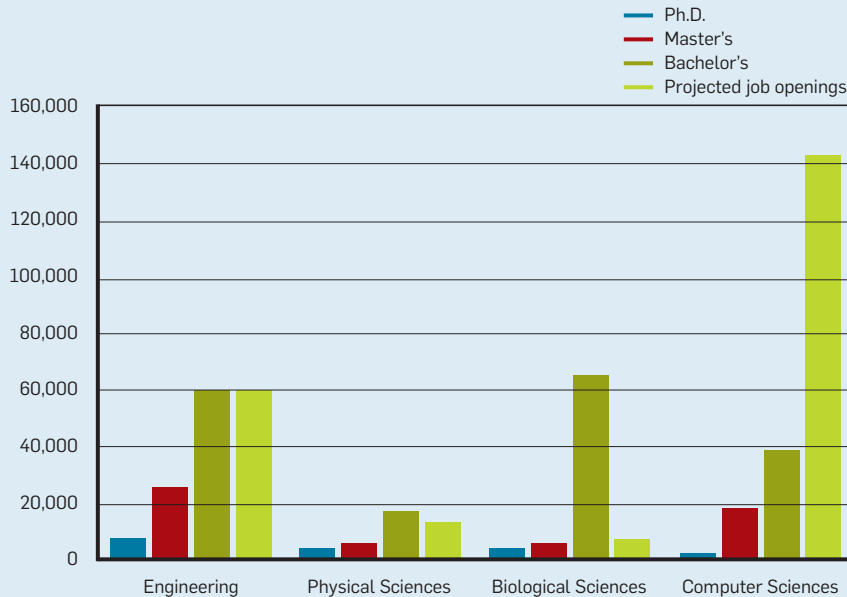
The answer, of course, is that Microsoft hires them both. Although the software engineer in Bangalore might be more cost-effective, what possible reason could there be for throwing away \$800,000 a year? As long as qualified candidates are scarce and capital is plentiful, companies will hire anyone for whom the marginal value exceeds the marginal cost. The value that a company can recognize from the services of talented software developers vastly exceeds their costs, irrespective of in which country they reside or in what currency they are paid.

The only way software development jobs will move entirely overseas is if the U.S. abandons the playing field by failing to produce students with the necessary skills. As the *New York Times* editorial page observed on March 1, 2006, shortly after the publication of the ACM globalization report: “Perhaps that explains what the report says is declining interest in computer science among American college students. Students may think, Why bother if all the jobs are in India? But the computer sector is booming, while the number of students interested in going into the field is falling. The industry isn’t gone, but it will be if we don’t start generating the necessary dynamic work force.” Andriole’s failure to understand that the computing industry extends far beyond enterprise software and his perpetuation of the myths that drive students away can only make it more difficult to generate the dynamic work force the U.S. needs to remain competitive in the global marketplace. □

**Eric Roberts** (eroberts@cs.stanford.edu) is a professor of computer science at Stanford University, co-chair and principal author of the computer science volume produced by the joint ACM/IEEE-CS Task Force on Computing Curricula 2001, and past chair of the ACM Education Board.

© 2008 ACM 0001-0782/08/0700 \$5.00

#### U.S. degree production and annual employment projections.



Source: Adapted from a presentation by John Sargent, Senior Policy Analyst, Department of Commerce, at the CRA Computing Research Summit, February 23, 2004. Original sources listed as National Science Foundation/Division of Science Resources Statistics; degree data from Department of Education/National Center for Education Statistics: Integrated Postsecondary Education Data System Completions Survey; and NSF/SRS; Survey of Earned Doctorates; and Projected Annual Average Job Openings derived from Department of Commerce (Office of Technology Policy) analysis of Bureau of Labor Statistics 2002–2012 projections. See [www.cra.org/govaffairs/content.php?cid=22](http://www.cra.org/govaffairs/content.php?cid=22).

Even though the statistics in the figure are derived from surveys taken several years ago, there is no reason to believe the situation has changed in any qualitative way. Comparing the 2002 and 2006 reports from the Bureau of Labor Statistics suggests that employment demand may have shifted by as much as 10% percent in certain categories. The fundamental message of the figure would not change even if the numbers were off by a factor of two.

## Image Crisis

# Inspiring a New Generation of Computer Scientists

*Consider what you can do to encourage young people to pursue technology-related career paths.*

**I**S COMPUTER SCIENCE a dying profession? That may seem like an odd question. After all, computers are omnipresent in our day-to-day lives. Their importance to the way we run our businesses, communicate, and use information has never been greater. Computing is an essential tool for discovery and advancement in virtually every field of science. And as we move forward, computing holds the key to progress in almost every human endeavor.

And yet the fact remains that, in the U.S. at least, computer science as a profession is beginning to wither away.

There is ample evidence to support this conclusion. A recent UCLA survey found that in 2006, barely 1% of incoming freshman planned to major in computer science, compared with nearly 5% 25 years ago. According to the most recent version of the Computer Research Association's annual Taulbee report, just 12,498 computer science and computer engineering degrees were awarded last year, a one-year drop of almost 20%. Even more alarming, total undergraduate enrollment in computer science and computer engineering has fallen 50% during the past five years, to just 46,000 students.

All this comes at a time when demand for computer scientists is stronger than it has been for many years. Today, IT employment is 17% higher than it was at the height of the dot-com bubble. According to the U.S.

Bureau of Labor Statistics, we will add an annual average of 100,000 new computer-related jobs through 2014, with careers in computer science the fastest-growing of all "professional and related occupations."

These numbers actually understate the severity of the problem. Enrollment in computer science and computer engineering programs in the U.S. consists of a disproportionate number of foreign-born students, particularly at the graduate level: last year, more than half of master's and doctoral degrees granted by U.S. universities were awarded to non-U.S. citizens. Thanks to a combination of security restrictions here and increasing job opportunities in their home countries, fewer numbers of these students are choosing to remain in the U.S. to work.

Left unchecked, these trends will inevitably undermine our ability to compete in the global economy. For decades, the ability of U.S. companies to transform innovations into successful businesses has been the foundation for our economic growth. Technologies such as the microprocessor, the Internet, and fiber optics that were developed by scientists and engineers trained in U.S. universities laid the foundation for new industries that generated millions of high-paying jobs.

But if the number of young people in the U.S. who study computer science continues to decline, the center of gravity for innovation will shift to countries where students flock to uni-

versities to pursue degrees in the technical fields that will enable tomorrow's breakthroughs.

As head of Microsoft Research, I am acutely aware of the impact that the shortage of computer professionals can have. Although the majority of our researchers are based in the U.S. and these facilities continue to grow, we are expanding our research facilities in other parts of the world, in part because we recognize that this may be the only way we can continue to find and hire the world's top computer scientists. I also see the increasing difficulty that Microsoft has in filling positions that require a high level of training and skill in computer science and engineering.

And, as co-chair of the Image of Computing Task Force with Jim Foley of Georgia Tech, I am committed to working with colleagues from industry, academia, and government to understand why interest in computer science is declining in the U.S. and learn what we can do to encourage young people to pursue technology-related careers. Founded by Foley, and based at the University of Colorado in Boulder, the Image of Computing Task Force is spearheading a national effort to help young people recognize the vital role that computing plays in almost every field and see the opportunities that come with a solid background in computer science.

Through my work with Jill Ross, director of the Image of Computing Task Force, I've spoken with high school

and college students from across the country. What I've learned from these conversations and from the growing body of research into why students are losing interest in computing gives me hope that we can inspire a new generation of bright young people to pursue careers in computer science and related fields. At the heart of falling interest in computer science are fundamental misconceptions about the work we do, our ability to make a difference in the world, and the job opportunities our field offers. If we can change these perceptions, we can ensure that instead of withering, our profession will thrive.

One barrier to interest in computer science is the unfortunate and deeply held stereotype of the solitary male programmer who slaves over a keyboard and subsists on snack food. A majority of young people subscribe to this stereotype and believe the job of the computer scientist consists of endless days spent alone in front of a computer screen. A survey of high school students enrolled in calculus and pre-calculus courses—students likely to have an aptitude for computer science—found that half have already decided not to pursue computer science as a major because they don't want to “sit in front of a computer all day.”

The problem is even more acute among women. A study of college undergraduate women who had achieved high SAT scores found that 70%–80% of them chose not to major in computer science and computer engineering because they felt they “would not enjoy the work.” Young people also underestimate the role that computing can play in changing the world. To most high school and college students, the job of the computer scientist is simply to write code. What they don't understand is that most of us chose to write code because we understood the power of computing as a tool for tackling important problems.

A study that compared computer science graduates at Georgia Institute of Technology with students who switched from computer science to another major is instructive. In that study, a typical graduate student who stayed in the major defined computing as “creating the applications...that allow computers to solve real-world problems.” Students who left the ma-

ior saw it as an exercise in “learning how to manipulate code,” and they assumed their work experience would be “boring...debugging code in front of a computer screen all day.”

Finally, the post-dot-com downturn notion that there aren't many openings in the field persists, compounded by the belief that computer-related jobs are quickly being outsourced. The message that many students hear from parents and teachers is that computer science is not a good career choice, despite U.S. Bureau of Labor Statistics reports indicating it is one of this country's fastest-growing professions.

Last February, I met with a number of high school and college students who are deeply interested in computer science. My goal was to learn what inspired them and find out what they think we can do to help inspire their peers. Part of what I learned was the important role an adult—a good teacher or an interested mentor—can play in encouraging an interest in computing.

One such student is Evie Powell, a Ph.D. candidate in game design at the University of North Carolina at Charlotte. From a family that discouraged her love of math and computers, she struggled as an undergraduate until she took an introductory game development course. “This class and its professor turned out to be the inspiration I needed,” she said.

Today, Evie is also active in the STARS Alliance, a program that aims to increase participation by women and minorities in computing. “I hope to reach out to those who feel like they don't have a place in such a technical field of study,” she said. “And hopefully show them early on that they too...have much to offer to the discipline.”

I also met UNC Charlotte student Lane Harrison. He started college with a vague interest in computing but felt he lacked the background and confidence to succeed. Now a third-year computer science and mathematics major, he says exceptional teaching was the catalyst for his decision to pursue computing. He too is active in the STARS Alliance and has spoken to more than 1,000 high school students about his enthusiasm for computing.

I was thrilled by the passion for computing that Evie, Lane, and the other students I met with share. I also came away

from those meetings feeling that those of us already in the field should see the work they are doing to encourage other young people as a personal challenge.

As ACM members, shouldn't we be even more committed to spending time out in the community and sharing our enthusiasm for computing than Evie and Lane are? Isn't it really up to us to show the next generation of potential computer scientists how exciting it is to work in a field where we have the opportunity to advance science, cure diseases, and tackle global warming? Shouldn't we be the ones out there demonstrating that our work consists less of debugging code than it does of collaborating with colleagues to develop new ideas and create solutions to difficult challenges?

How can we do this? By visiting schools and community groups to share our passion for our field and to make clear that a career in computing is filled with great jobs and incredible opportunities. By bringing young people to the places where we work so they can see what we really do. We can do it by offering internships and taking the time to mentor a young person and encourage their interest in math, science, and computing.

We also need to reach out to the people who have the greatest influence on young people: parents, teachers, counselors, and the media. Talk with teachers and professors and encourage them to show their students not only how to write code, but why computing is such a powerful way to solve problems.

Speak with parents or guidance counselors and make clear to them that computing is a career path that offers high-paying job opportunities unmatched by almost any other profession. Talk to journalists and emphasize the importance of computing as a driver for innovation and progress, and encourage them to provide a realistic picture of the work we do that goes beyond the traditional stereotype of the geek programmer.

If you are like me, you entered this field to make a difference. This is your opportunity. The future of our profession depends on it. ■

---

**Rick Rashid** (rashid@microsoft.com) is a senior vice president for research at Microsoft Corporation.

© 2008 ACM 0001-0782/08/0700 \$5.00



## Interview

# The ‘Art’ of Being Donald Knuth

*In this first of a two-part talk, the renowned scholar and computer scientist reflects on the influences that set the course for his extraordinary career.*

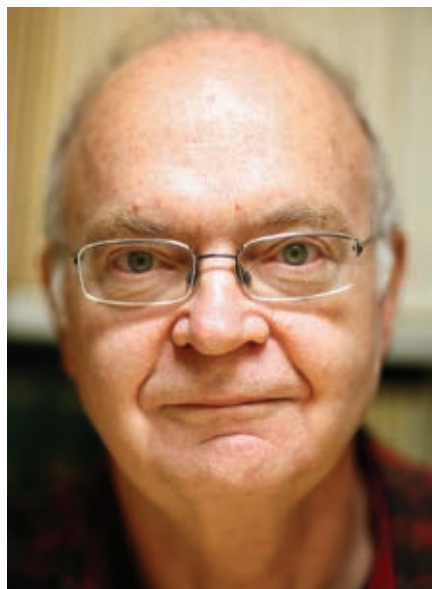
**T**HE COMPUTER HISTORY Museum has an active program to gather videotaped histories from people who have done pioneering work in this first century of the information age. These tapes are a rich aggregation of stories that are preserved in the collection, transcribed, and made available on the Web to researchers, students, and anyone curious about how invention happens.

The oral histories are conversations about people’s lives. We want to know about their upbringing, their families, their education, and their jobs. But above all, we want to know how they came to the passion and creativity that leads to innovation.

Presented here in two installments (concluding next month) are excerpts<sup>a</sup> from an interview conducted by Edward Feigenbaum in March 2007 of Donald E. Knuth, Professor Emeritus of The Art of Computer Programming at Stanford University. — L. S.

### Don talks about his family background.

My father was the first person among all his ancestors who had gone to college. My mother was the first person in all of her ancestors who had gone to a



year of school to learn how to be a typist. My great-grandfather was a blacksmith. There was no tradition in our family of higher education at all. These people were pretty smart, but they didn’t have an academic background.

### Some people know from an early age what they want to do. Don didn’t, but he knew he wanted to work hard.

My main interest in those days was music. But at the college where I had been admitted, people emphasized how easy it was going to be there as a music major. When I got the chance to go to Case Institute of Technology in Ohio instead, I was intrigued by the idea that Case was going to make me work hard. I was

scared that I was going to flunk out, but still I was ready to work.

### He initially aspired to be a physicist, but something happened along the way.

In my sophomore year in physics I had to take a required class of welding. Welding was so scary and I was a miserable failure at it, so I decided maybe I can’t be a physicist. On the other hand—mathematics! In the sophomore year for mathematicians, they give you courses on what we now call discrete mathematics, where you study logic and things that are integers instead of continuous quantities. I was drawn to that. That was something, somehow, that had great appeal to me.

I think that there is something strange inside my head. It’s clear that I have much better intuition about discrete things than continuous things. In physics, for example, I could pass the exams and I could do the problems in quantum mechanics, but I couldn’t intuit what I was doing. But on the other hand, in my discrete math class, these were things that really seemed a part of me. There’s definitely something in how I had developed by the time I was a teenager that made me understand discrete objects, like zeros and ones of course, or things that are made out of alphabetical letters, much better than things like Fourier transforms or waves.

I’m visualizing the symbols. To me, the symbols are reality, in a way. I take

<sup>a</sup> Oral histories are not scripted, and a transcript of casual speech is very different from what one would write. I have taken the liberty of editing and reordering freely for presentation. For the original transcript, see <http://archive.computerhistory.org/search/oh/>

**My first program taught me a lot about the errors that I was going to be making in the future, and also about how to find errors. That's sort of the story of my life, making errors and trying to recover from them. I try to get things correct. I probably obsess about not making too many mistakes.**

a mathematical problem, I translate it into formulas, and then the formulas are the reality.

#### **He discovers computers, and how hard programming is.**

I wrote my first program for the IBM 650 [a vacuum tube magnetic drum computer from the 1950s], probably in the spring of my freshman year, and debugged it at night. The first time I wrote the program, to find the prime factors of a number, it was about 60 instructions long in machine language. They were almost all wrong. When I finished, it was about 120 or 130 instructions. I made more errors in this program than there were lines of code!

My first program taught me a lot about the errors that I was going to be making in the future, and also about how to find errors. That's sort of the story of my life, making errors and trying to recover from them. I try to get things correct. I probably obsess about not making too many mistakes.

#### **At Case he learns about early compilers**

For the IT ("Internal Translator") program for the 650 you would punch an

algebraic formula on cards and feed the cards into the machine. The lights spin around for a few seconds and then out come machine language instructions that set  $X_1$  equal to  $X_2 + X_4$ . Automatic programming coming out of an algebraic formula! Well, this blew my mind. I couldn't understand how it was possible to do this miracle. I could understand how to write a program to factor numbers, but I couldn't understand how to write a program that would convert algebra into machine instructions.

#### **It hadn't yet occurred to him that the computer was a general symbol-manipulating device?**

No. That occurred to Lady [Ada] Lovelace, but it didn't occur to me. I'm slow to pick up on these things, but then I persevere.

I got hold of the source code for IT. I went through every line of that program. During the summer we typically had a family get-together on a beach on Lake Erie where we spent time playing cards and playing tennis. But that summer, I spent most of the time going through this listing, trying to find out the miracle of how IT worked. Okay, it wasn't impossible after all. In fact, I thought of better ways to do it than were in that program.

The code, once I saw how it happened, was inspiring to me. Also, the discipline of reading other people's programs was something good to learn early. Throughout my life I've had a love of reading source materials—reading something that pioneers had written and trying to understand their thought processes, especially when they're solving a problem I don't know how to solve. This is the best way for me to get my own brain past the stumbling blocks. At Case I remember looking at papers that [Pierre de] Fermat had written in Latin in the 17th century, in order to understand how that great number theorist approached problems.

#### **But it's been hard to communicate the love of reading historical programs.**

I would say that's my major disappointment with my teaching career. I was not able to get across to any of my students this love for that kind of



scholarship—reading source material. I was a complete failure at passing this on to the people that I worked with the most closely.

#### **He graduates from Case and becomes a professional compiler writer while traveling to the California Institute of Technology for graduate school.**

I had learned about the Burroughs 205 machine language, and it was kind of appealing to me. So I made my own





proposal to Burroughs. I said, "I'll write you an ALGOL compiler for \$5,000. But I can't implement all of ALGOL for this; I am just one guy. Let's leave out procedures." Well, this is a big hole in the language! Burroughs said, "No, you've got to put in procedures." I said, "Okay, I will put in procedures, but you've got to pay me \$5,500." That's what happened. They paid me \$5,500, which was a fairly good salary in those days. So between graduating from Case and going to Caltech, I worked on this compiler.

Heading out to California, I drove 100 miles each day and then sat in a motel and wrote code.

**But he rejects "compiler writer" as a career, and decides what is important in life.**

Then a startup company came to me and said, "Don, write compilers for us and we will take care of finding computers to debug them. Name your price." I said, "Oh, okay, \$100,000," assuming that this was

[outrageous]. The guy didn't blink. He agreed. I didn't blink either. I said, "I'm not going to do it. I just thought that was an impossible number." At that point I made the decision in my life that I wasn't going to optimize my income.

I spent a day that summer looking at the mathematics of how fast linear probing works. I got lucky, and I solved the problem. I figured out some math, and I kept two or three sheets of paper with me and I typed



it up.<sup>b</sup> This became the genesis of my main research work, which developed not to be working on compilers, but to be working on the analysis of algorithms. It dawned on me that this was just one of many algorithms that would be important, and each one would lead to a fascinating mathematical problem. This was easily a good lifetime source of rich problems to work on.

If you ask me what makes me most happy, number one would be somebody saying “I learned something from you.” Number two would be somebody saying “I used your software.”

### **At Caltech he finds a mentor, but can't talk to him.**

I went to Caltech because they had [strength] in combinatorics, although their computing system was incredibly arcane and terrible. Marshall Hall was my thesis advisor. He was a world-class mathematician, and for a long time had done pioneering work in combinatorics. He was my mentor. But it was a funny thing, because I was in such awe of him that when I was in the same room with him I could not think straight. I wouldn't remember my name. I would write down what he was saying, and then I would go back to my office so that I could figure it out. We couldn't do joint research together in the same room. We could do it back and forth.

He also was an extremely good advisor, in better ways than I later was with my students. He would keep track of me to make sure I was not slipping. When I was working with my own graduate students, I was pretty much in a mode where they would bug me instead of me bugging them. But he would actually write me notes and say, “Don, why don't you do such and such?”

### **The research for his Ph.D. thesis takes an hour.**

I got a listing from a guy at Princeton who had just computed 32 solutions to a problem that I had been looking at for a homework problem in my combinatorics class. I was riding up on the

<sup>b</sup> “Notes on Open Addressing.” Unpublished memorandum, July 22, 1963; but see <http://algo.inria.fr/AofA/Research/11-97.html>

**If you ask me what makes me most happy, number one would be somebody saying “I learned something from you.” Number two would be somebody saying “I used your software.”**

elevator with Olga Todd, one of our professors, and I said, “Mrs. Todd, I think I'm going to have a theorem in an hour. I am going to psyche out the rule that explains why there happen to be 32 of each kind.” Sure enough, an hour later I had seen how to get from each solution on the first page to the solution on the second page. I showed this to Marshall Hall. He said, “Don, that's your thesis. Don't worry about this block design with  $\square=2$  business. Write this up instead and get out of here.” So that became my thesis. And it is a good thing, because since then only one more design with  $\square=2$  has been discovered in the history of the world. I might still be working on my thesis if I had stuck to that problem. But I felt a little guilty that I had solved my Ph.D. problem in one hour, so I dressed it up with a few other chapters of stuff.

### **He's never had trouble finding problems to work on.**

The way I work it's a blessing and a curse that I don't have difficulty thinking of questions. I have to actively suppress stimulation so that I'm not working on too many things at once. The hard thing for me is not to find a problem, but to find a good problem. One that will not just be isolated to something that happens to be true, but also will be something that will have spin-offs, so that once you've solved the problem, the techniques are going to apply to many other things.

### **He starts *The Art of Computer Programming*.**

A man from Addison-Wesley came to visit me and said “Don, we would like you to write a book about how to write compilers.” I thought about it and decided “Yes, I've got this book inside of me.” That day I sketched out—I still have that sheet of tablet paper—12 chapters that I thought should be in such a book. I told my new wife, Jill, “I think I'm going to write a book.” Well, we had just four months of bliss, because the rest of our marriage has all been devoted to this book. We still have had happiness, but really, I wake up every morning and I still haven't finished the book. So I try to organize the rest of my life around this, as one main unifying theme.

George Forsythe [founder of the Computer Science Department at Stanford] came down to southern California for a talk, and he said, “Come up to Stanford. How about joining our faculty?” I said “Oh no, I can't do that. I just got married, and I've got to finish this book first. I think I'll finish the book next year, and then I can come up [and] start thinking about the rest of my life. But I want to get my book done before my son is born.” Well, John is now 40-some years old and I'm not done with the book.

This is really the story of my life, because I hope to live long enough to finish it. But I may not because it's turned out to be such a huge project.

### **1967 was a big year.**

It was certainly a pivotal year in my life. You can see in retrospect why I think things were building up to a crisis, because I was just working at high pitch all the time. I was on the editorial board of *Communications of the ACM* and *Journal of the ACM*—working on their programming languages sections—and I took the editorial duties very seriously. I was a consultant to Burroughs on innovative machines. I was consumed with getting *The Art of Computer Programming* done. And I was a father and husband. I would start out every day saying “Well, what am I going to accomplish today?” Then I would stay up until I finished it.

It was time for me to make a career decision. The question was where should I spend the rest of my life?

Should I be a mathematician? Should I be a computer scientist? By this time I had learned that it was actually possible to do mathematical work as a computer scientist. I had analysis of algorithms to do. What would be a permanent home? My model of my life was going to be that I was going to make one move in my lifetime to a place where I had tenure, and I would stay there forever.

### The crisis comes.

At Caltech, I was preparing my class lectures, or typing my book. I didn't have time to do research. If I had a new idea, if I said "Here's a problem that ought to be solved," when was I going to solve it? Maybe on the airplane. We were doing a lot of experiments but I didn't have time to sit down at home and work out the theory for it. I had attribute grammars coming up in February, and these reductions systems coming up in March, and I was supposed to be grinding out Volume Two of *The Art of Computer Pro-*

**I told my new wife, Jill, "I think I'm going to write a book." Well, we had just four months of bliss, because the rest of our marriage has all been devoted to this book. We still have had happiness, but really, I wake up every morning and I still haven't finished the book. So I try to organize the rest of my life around this, as one main unifying theme.**

*gramming*. I was scheduled in June to lecture at a summer school in Copenhagen about how to parse, what's called top-down parsing.

What happened then, in May, is I had a massive bleeding ulcer, and I was hospitalized. My body gave out. I was just doing all this stuff, and it couldn't take it.

I learned about myself. The doctor showed me his textbook that described the typical ulcer patient: what people call the "Type A" personality. It described me to a T. All of the signs were there. I was an automaton, I think, basically. I saw a goal and I put myself to it, and I worked on it and pushed it through. I didn't say no to people when they asked, "Don, can you do this for me?" At this point I saw I had this problem. I shouldn't try to do the impossible.

### He changes his lifestyle, and moves to Stanford.

I wrote a letter to my publisher, framed in black, saying, "I'm not going to be able to get the manuscript of Volume Two to you this year. I'm sorry." I resigned from 10 editorial boards. No more JACM, no more CACM. I gave up all of the editorships in order to cut down my workload. I started working on Volume Two where I left off at the time of the ulcer, but I would be careful to go to sleep and keep a regular schedule. I went to a conference in Santa Barbara on combinatorial mathematics and had three days to sit on the beach and develop the theory of attribute grammars, this idea of top-down and bottom-up parsing.

In February of 1968 I finally got the offer from Stanford. The committees were saying, "This guy is just 30 years old." But when they looked at the book, they said, "Oh, there's some credibility here." That helped me.

### Why he writes his books with a pencil.

I love keyboards, but my manuscripts are always handwritten. The reason is that I type faster than I think. There's a synchronization problem. I can think of ideas at about the rate I can write them down with a pencil. But with typing I'm going faster, so I have to sync, and my thoughts have to start up and stop again in a way that involves more of my brain.

### Three volumes of "The Art" are done, but it's time for a pause.

Volume Four is about combinatorial algorithms. Combinatorial algorithms were such a small topic in 1962, when I made that Chapter Seven of my outline, that Johan Dahl asked me, "How did you ever think of putting in a chapter about combinatorial algorithms in 1962?" I said, "Well, the only reason was that it was the part I thought was most fun." But there was almost nothing known about it at the time.

The way I look at it, this is where you've got to use some art. You've got to be really skillful, because one good idea can save you six orders of magnitude and make your program run a million times faster. People are coming up with these ideas all the time. For me, the combinatorial explosion was the explosion of research in combinatorics. Not the problems exploding, but the ideas were exploding. There's that much more to cover now.

It's true that in the back of my mind I was scared stiff that I can't write Volume Four anymore. So maybe I was waiting for it to simmer down. Somebody did say to me once, after I solved the problem of typesetting, maybe I would start to look at binding or something, because I had to have some other reason [to delay]. I've certainly seen enough graduate student procrastinators in my life. Maybe I was in denial. ■

*He solves the problem of typesetting? Stay tuned for Part II of this interview in the August issue and learn how Knuth interrupted his life's work on The Art of Computer Programming to create a system that makes digitally produced books beautiful.*

Edited by **Len Shustek**, Chair, Computer History Museum

© 2008 ACM 0001-0782/08/0700 \$5.00

**Don't let delusions about XML develop into a virulent strain of XML fever.**

BY ERIK WILDE AND ROBERT J. GLUSHKO

# XML Fever

THE EXTENSIBLE MARKUP LANGUAGE (XML), which just celebrated its 10<sup>th</sup> birthday,<sup>4</sup> is one of the big success stories of the Web. Apart from basic Web technologies (URIs, HTTP, and HTML) and the advanced scripting driving the Web 2.0 wave, XML is by far the most successful and ubiquitous Web technology. With great power, however, comes great responsibility, so while XML's success is well earned as the first truly universal standard for structured data, it must now deal with numerous problems that have grown up around it. These are not entirely the fault of XML itself, but instead can be attributed to exaggerated claims and ideas of what XML is and what it can do.

This article is about the lessons gleaned from learning XML, from teaching XML, from dealing with overly optimistic assumptions about XML's powers, and from helping XML users in the real world recover from these misconceptions. Shamelessly copying Alex Bell's "Death by UML Fever,"<sup>1</sup> we frame our observations and the root of the problems along with possible cures in terms of different categories and strains of "XML fever." We didn't invent this term, but it embodies many interesting metaphors for understanding the use and abuse of XML, including disease symptoms, infection methods, immunization and preventive measures, and various remedies for treating those suffering from different strains.

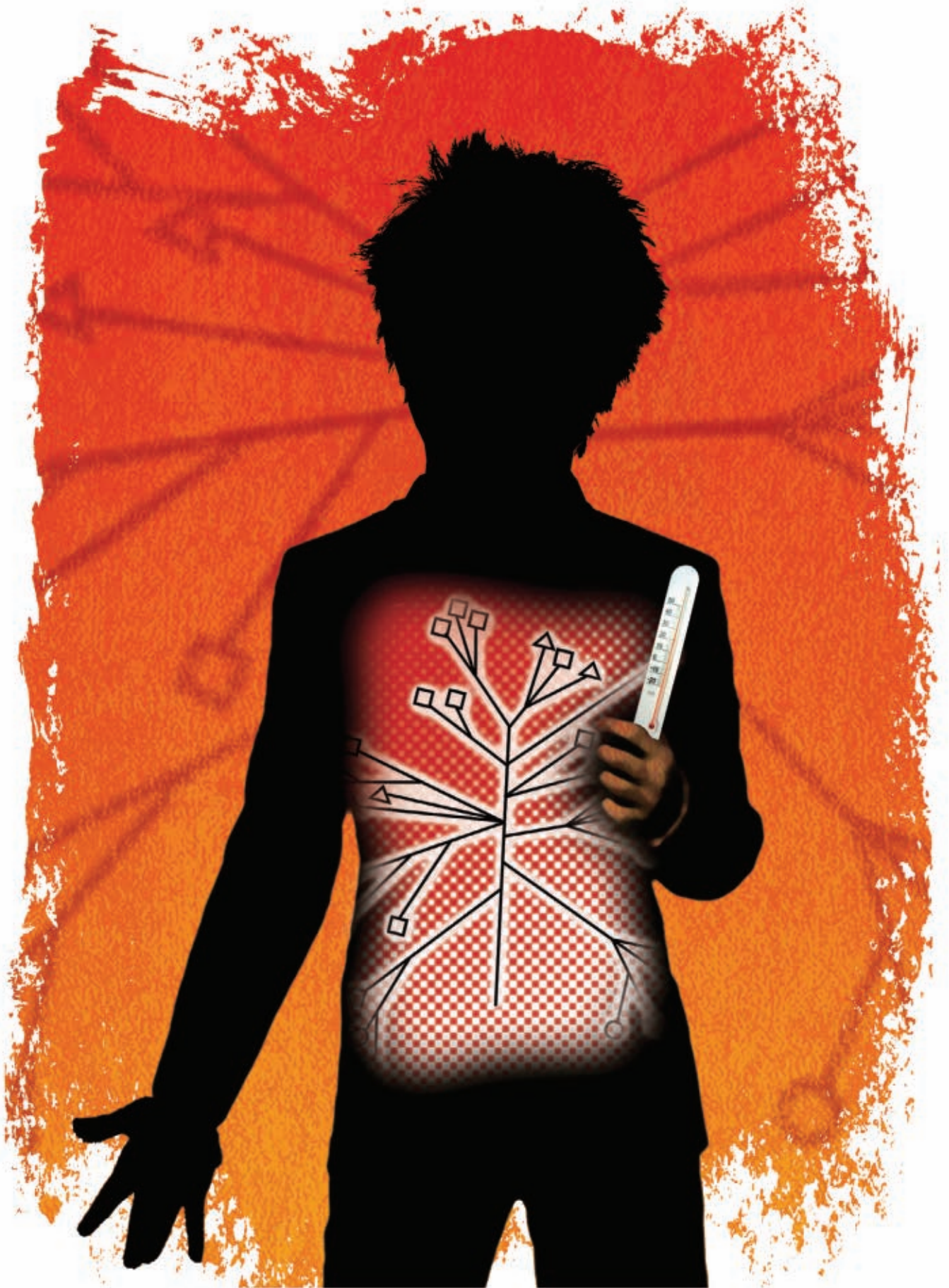
XML fever can be acquired in many different ways, but the most prevalent way is to be infected by the idea that XML enables almost magical universal interoperability of information producers and consumers. XML fevers can be classified as basic, intermediate, and advanced:

*Basic strains* infect XML neophytes, but most of them recover quickly. It can be disappointing to discover that the landscape of XML technologies is not as simple as expected, and that working with the associated tools requires some getting used to, but most people develop some immunity to the XML hype and quickly begin to do useful work with it.

*Intermediate strains* of XML fever are contracted when XML users move beyond simple applications involving structured information and encounter models of data, documents, or processes. A recurring symptom in these varieties of XML fever is mild paralysis brought on by having to select a schema language to encode a model, trying to choose among the bewildering number of features in some languages, or trying to "round-trip" a model between different environments.

*Advanced strains* of XML fever often take hold after exposure to the proliferation of more complex and esoteric XML-based technologies layered on top of it. These advanced diseases are





harder to catch, but they are also harder to remedy because people who have caught these advanced strains tend to congregate with others with the same diseases and they are continually reinfecting each other.

### Basic Strains

One of our favorite teaching moments is to start an introductory XML lecture with the statement, “XML is a syntax for trees,” and that this is all there is to it, so no further explanation is required. Of course, there is more to it, and we manage to fill a complete course with it, but the essence of XML really is simple and small. This is elegant to us but a disappointment to many XML beginners who expect something bigger and more complicated to match up with all the hype they have heard. In fact, XML’s character-based format lures many XML beginners to assume they can simply use their trusted text-processing tools, which is the inevitable path to the first XML fever:

*Parsing pain.* At first sight, XML’s syntax looks as if it would be easy to use simple text-processing tools for accessing XML data, so that a “desperate Perl hacker” could implement XML in a weekend. Unfortunately, not all XML documents use the same character encoding; character references must be interpreted; entities must be resolved; and so on... As soon as the output from a wider array of XML producers is considered, it becomes apparent that for robustly parsing XML with text-processing tools, the tools must implement a complete XML parser. This becomes most painfully evident when XML processing needs to take XML Namespaces into account (often leading to an infection with the intermediate namespace nausea fever).

After overcoming parsing pain and starting to use an XML parser, beginners usually understand what we mean when we say that XML is a syntax for trees, but they do not as quickly grasp that XML uses multiple tree models, and depending on which XML technology one is using, the “XML tree” looks slightly different. Thus, the second basic strain of XML fever is:

*Tree trauma.* This is caused by exposure to XML’s various tree models, such as XML itself, DOM, the Infoset, XPath 1.0, PSVI, and XDM. All of these

tree models share XML’s basic idea of trees of elements, attributes, and text, but have different ways of exposing that model and handling some of the details. In fact, while XML itself explicitly states that XML processors must implement all of XML (apart from validation, the standard has no optional parts, which is a smart thing for a standard to do), some of the more recent tree models exhibit the “extended subset” nature of technologies, which can often lead to incompatibilities among implementations. For example, PSVI—the data model of an XML document validated by an XML Schema (for the rest of the article, we refer to W3C’s language as XSDL)—is based on the Infoset, which is a subset of the full information of an XML document, and extends that subset with information made available by the schema and the validation process.

While XML is available in a number of various “tree flavors,” the W3C has settled (after a very long process) on the Infoset model as the core of many XML technologies. This means it would be technically more accurate to say that most XML technologies available today are actually Infoset technologies. XML has become one way (and so far the only standardized one, but with the upcoming binary Infoset format EXI as a more compact alternative) of representing Infosets. Of course, the W3C does not want to give up the brand name of XML and still calls everything “XML-based.” As a result, XML users can easily get affected by a peculiar ailment:

*Infoset ignorance.* Instead of XPath, XSLT, and XQuery, these technologies’ proper names would be IPath, ISLT, and IQuery, because they are Infoset-based. Victims of Infoset ignorance take the W3C’s branding of everything as XML at face value and sometimes invest a lot of energy trying to build XML processing pipelines that preserve character references and other markup details. Infoset ignorance prevents its victims from seeing that this approach cannot succeed as long as they are using standards-based tools.

The remedy for Infoset ignorance is to select a set of XML technologies with compatible tree models. This usually also cures tree trauma, because now XML users can focus on a specific variety of XML tree. Depending on the specific

technologies chosen, though, tree trauma can metastasize into a more severe disease caused by failure to appreciate the somewhat obscure ways in which some XML technologies process trees:

*Default derangement.* Tree trauma can develop into default derangement if XML users are exposed to and experimentation with schema languages such as DTDs and XSDL that allow default values. These languages cause XML trees to change based on validation, which means that XML processing is critically based on validation. Because it is often not feasible to quarantine XML users to keep them from these schema languages, a better prescription is to put them on a strict diet of design guidelines to avoid these potentially dangerous features.

Among the core components of virtually all XML scenarios today are XML Namespaces. They are essential for turning XML’s local names into globally unique identifiers, but the specifics of how namespaces can be declared in documents, and the fact that namespace names are URIs that do not need to be dereferenceable, have not yet failed once to confound everybody trying to start using them. A very popular XML fever thus is:

*Namespace nausea.* No matter how often we try to explain that XML Namespaces have no functionality beyond the simple association of local names and namespace names, many myths and assumptions surround them. For example, many students assume that namespaces must refer to existing resources and ask us how to “call the namespace in a program.” And even though they should be simple, XML is often serialized by tools that do not allow much control over how namespaces are treated, creating XML documents that exhibit various kinds of correct but very confusing ways of using namespaces. A particularly nasty secondary infection caused by namespace nausea can be contracted when using a specific kind of XML vocabulary:

*Context cataracts.* If QName (the colon-separated names combining namespace prefixes and local names) are allowed to appear as content of XML documents (such as in attribute values or element content), they make the content context dependent. This means that such XML content can be




correctly interpreted only within its context in the XML document (where all in-scope namespace declarations can be accessed), or it must be decontextualized by parsing it and replacing each QName with a context-independent representation. Unfortunately, no standard exists for this latter approach, which makes this contextualized content brittle and hard to work with.


The strains described so far manifest themselves in basic XML processing tasks. As soon as XML users begin work with business information and processes, they must confront the challenge of understanding what XML structures actually mean. This task exposes them to a dangerous virus encoded in the catchy slogan that XML is “self-describing.”

We could be charitable and assume that when people say XML is self-describing, what they really mean is “compared with something else that clearly isn’t.” The least self-describing information consists of just a stream of alphanumeric characters of some text format, as they might be on a punch card. This delimiter-less encoding does not even make explicit the tokenization of the characters into meaningful values, so there is not any “self” to which any description could be assigned. The possibility of self-description emerges only when we separate the values with commas or some other delimiter character, which tells us what information components must be described. XML goes one step further with the syntactic mechanisms of paired text labels to distinguish the information components in a stream of text and quotes to associate one bit of information as an attribute of another. It is certainly fair to say that XML is on average more self-describing than other text-based encoding syntaxes, but that is like saying the average dwarf is taller than the average baby; neither is tall enough to excel at basketball.

From a more technical perspective, it is also true that XML is self-describing in the limited sense that the data structure (one of the XML trees, see tree trauma) can be reconstructed from an XML document (and maybe its schema, if processing takes place in an environment susceptible to default derangement).



**While XML’s success is well earned as the first truly universal standard for structured data, it must now deal with numerous problems that have grown up around it. These are not entirely the fault of XML itself, but instead can be attributed to exaggerated claims and ideas of what XML is and what it can do.**



When most people say that XML is self-describing, however, they are being captured by a delusion that this refers to actual semantics, overlooking the fact the XML has almost no predefined semantics (the only exception being one predefined attribute for identifying languages). The disease is most likely caused by the many XML examples that show element and attribute names that seem to be self-describing because they are labeling the syntactic components. It could be prevented with examples that merely show how the XML markup characters distinguish the information being described from the markup that is part of its structural description:

```
<xxx yyy="4567">850</xxx>
<zzz>20060812</zzz>
```

Using syntactic mechanisms to provide clues to the element and attribute semantics is convenient, but this is the cause of a very common strain of XML fever:

*Self-description delusion.* XML’s ability to define names for elements and attributes, and the widespread assumption that these names have some intrinsic semantics, often cause victims to assume that the semantics of an XML document are self-evident, openly available just by looking at it and understanding the names. Frequently, this strain of XML fever causes great discomfort when the victims learn that XML does not deal with semantics, and that common understanding has to be established through other mechanisms. Victims weakened by self-description delusion are often infected by one or more of the intermediate or advanced strains of XML fever, which promise to easily and permanently cure the pain caused by self-description delusion.

Recovery from self-description delusion can take a great deal of personal commitment and effort. Victims must learn how to define or adapt an XML vocabulary, or to adopt technologies that are explicitly focused on semantics, not just syntax. In either case, these steps risk exposure to strains of XML fever beyond the basic types.

### Intermediate Strains

If self-description delusion is appropriately diagnosed and treated, XML users



often recover with improved insight. They now realize that XML's basic technologies and toolset can be employed for basic processing tasks involving structured data, but that most applications involve models of the application data or processes. XML is based on tree structures as the basic model, and this does not always provide the best fit for application-level models, which can cause trouble when mapping these nontree structures to XML:

*Tree tremors.* Whereas tree trauma (discussed earlier) is a basic strain of XML fever caused by the various flavors of trees in XML technologies, tree tremors are a more serious condition afflicting victims trying to manage data in XML that is not inherently tree-structured. The most common causes are data models requiring nontree graph structures and document models needing overlapping structures. In both cases, mapping these models to XML's tree model results in XML structures that cannot conveniently represent the application-level model.

We often tell students that "the best thing about XML is the ease with which you can create a new vocabulary." But because XML allows well-formed documents (as opposed to valid documents that must conform to some schema), it is actually possible to use vocabularies that have never been explicitly created: documents can simply use elements and attributes that were never declared (let alone defined) anywhere. Well-formedness can be appropriate during prototyping but is reckless during deployment and almost certainly subverts interoperability. Unfortunately, many XML users suffer from a condition that prevents them from seeing these dangers:

*Model myopia.* Starting from a prototype based on well-formed documents, some developers never bother to develop a schema, let alone a well-defined mapping between such a schema and the application-level data model. In scenarios leading to this condition, validation often is only by eye (key phrases for this technique are "looks good to me" or "our documents usually sort of look like these two examples here"), which makes it impossible to test documents strictly for correctness. Round-trip XML-to-model and reverse transformations cannot be reliably implemented,

and assumptions and hacks get built into systems, which inevitably cause interoperability problems later on.

If model myopia is diagnosed (often by discovering that two implementations do not interoperate correctly because of different sets of assumptions built into these implementations), the key step in curing it is to define a schema so that the XML structures to be used in documents are well defined and can be validated using existing tools. As soon as this happens, the obvious question is which schema language to use. This can be the beginning of another troublesome development:

*Schema schizophrenia.* DTDs are XML's built-in schema language, but they are limited in their expressiveness and do not support essential XML features (most notably, they do not work well with XML Namespaces). After considering various alternative languages, the W3C eventually settled on XSDL, a rather complex schema language with built-in modeling capabilities. XSDL's expressiveness can directly cause an associated infection, caused by the inability to decide between modeling alternatives:

*Schema option paralysis.* XSDL's complexity allows a given logical model to be encoded in a plethora of ways (this fever will mutate into an even more serious threat with the upcoming XSDL 1.1, which adds new features that overlap with existing features). A cure for schema option paralysis is to use alternative schema languages with a better separation of concerns (such as limiting itself to grammars and leaving data types and path-based constraints to other languages), most notably RELAX NG.

Using more focused schema languages and targeting a separation of concerns leaves schema developers with a choice of schema languages. In addition, at times it would be ideal to combine schema languages to capture more constraints than any one could enforce on its own. The choice of schema languages, however, is more often determined by available tool support and acquired habits than by a thorough analysis of what would be the most appropriate language.

Since schema schizophrenia (with occasional bouts of schema option paralysis) can be a painful and long-

lasting condition, one tempting way out is not to use schema languages as the normative encoding form for models and instead generate schemas from some more application-oriented modeling environment or tool. Very often, however, these tools have a different built-in bias, and they rarely support document modeling. This causes a very specific problem for generated schemas:

*Mixed content crisis.* XML's origin as a document representation language gives it capabilities to represent complex document structures, most notably mixed content, essential in publications and other narrative document types. Most non-XML modeling environments and tools, however, are data oriented and lack support for mixed content. These tools produce XML structures that look like table dumps from a relational database, lacking the nuanced document structures that are crucial in a document-processing environment.

Because the approach of generating schemas has the advantage that developers of XML schemas never have to actually write them (or even look at them), it also can be the cause of one of the most troubling XML problems that is often experienced when encountering schemas generated from UML models or spreadsheets:

*Generated schema indigestion.* More abstract models have to be mapped to XML vocabularies for XML-based information exchange. Most modeling tools and development environments export models to XSDL and use that schema for serializing and parsing instances. Because of the perniciousness of schema schizophrenia, however, this model-to-schema encoding is complex and tool dependent. Generated schema indigestion often afflicts those who try to use the schema or instances outside the context of the tools that generated them. This first contact with generated schemas can be very frustrating and distasteful, because unless the same XML encoding rules are followed in both contexts, XML might not be easy to work with and certainly is neither interoperable nor extensible.

These intermediate strains of XML fever mostly revolve around the problem of how to create and use well-defined descriptions of XML vocabular-


ies. Before we continue to describe the more advanced strains of XML fever that may result from these intermediate fevers and attempts to cure them, it is important to point out that a good way of avoiding them is to reuse existing XML languages, thus avoiding the efforts and risks of inventing something new.

In an online follow-up to “On Language Creation,”<sup>3</sup> Tim Bray (one of the creators of XML) says, “If you’re going to be designing a new XML language, first of all, consider not doing it.” This is a very important point, because the ubiquity of XML makes it likely that for any given problem, somebody else might have already encountered it and solved it. Or for a given problem, it might be possible to divide it into smaller parts or to map it to a more general problem, and to find existing solutions for these.


Of course, there is a chance that no prior work exists or that the available solutions are unsatisfactory, but it really is worth the effort to evaluate existing solutions because a vocabulary can represent hundreds or even thousands of hours of analysis and encoding. For example, the Universal Business Language (UBL), a set of information building blocks common to business transactions and several dozen standard documents that reuse them, is the result of years of work by numerous XML and business experts—and the UBL effort itself began in 2001 by building on the XML Common Business Library (xCBL), on which work began in 1997.

We always tell students the worst thing about XML is the same as the best thing: the ease with which you can create a new vocabulary. Language design is fundamentally hard, but XML has made it deceptively simple by lowering the syntactic threshold. The conceptual tasks of creating shared vocabularies that are globally understood, well defined in every necessary respect, and reasonably easy to use have not been made easier by XML. XML has just given us a good toolset to describe and work with these languages once we have them, but defining them still is hard work.

This, of course, is not a secret to computer scientists, and the fact that XML has no semantics when they are essential to meaningful information



**We always tell students the worst thing about XML is the same as the best thing: the ease with which you can create a new vocabulary.**



exchange led to the idea of the Semantic Web.<sup>2</sup> The value proposition of the Semantic Web is compelling: a common way of representing semantics makes it easier to express, understand, exchange, share, merge, and agree on them. The Semantic Web, however, is also the leading cause of the more advanced strains of XML fever.

#### **Advanced Strains**

If semantics are important, and since an XML schema defines only structures (that is, syntax), then semantics must be specified in some other way. This can happen informally by prose describing the meaning of the individual components and parts of a schema, or more formally, by using some model for specifying semantics. The Semantic Web is the most popular candidate for such an environment; it is based on a model for making statements about resources, the Resource Description Framework (RDF), with various technologies layered on top of that, such as those for describing schemas for RDF.

One important observation about the Semantic Web that is often missed is that it introduces not only models for semantics (various schema languages for RDF), but also a new data model, which means that XML’s tree structures are no longer the core data structures for representing data. RDF can be expressed in XML, but there are many different ways of doing it, which can cause a very specific illness:

*RDF rage.* RDF’s most widely used syntax is XML based, but there are many different ways in which the same set of RDF triples can be expressed as XML, so working with RDF data is almost impossible using basic XML tools, even for simple tasks such as comparing RDF data. This inability to use a seemingly related toolset for a seemingly related task often is the first symptom through which XML users learn that they are now suffering from more advanced strains of XML fever.

In a more classical view of information organization, the meaning of terms can be specified in a variety of ways. Ordered by complexity, popular approaches are controlled vocabularies, taxonomies, thesauri, and ontologies. RDF can be used to implement any of these concepts, but RDF schemas are most often referred to as on-

tologies. This is in part a result of free standards-based tools for creating ontologies such as Protégé and SWOOP; just as we mentioned with schema option paralysis, the availability of tools shapes the languages people use and the choices they make. The relative unfamiliarity and the vague “hipness” of the “ontology” world, however, can give XML users anxiety about their ability to adjust to the RDF/OWL world with more rigorous semantics. As a result, they often overcompensate:

*Ontology overkill.* Operating in an environment that focuses on semantics, victims of ontology overkill tend to overmodel semantics, creating abstractions and associations that are of little value to the application but make the model much harder to understand and use. Ontology overkill forces its sufferers not only to overmodel, but also often to fail at doing so, because it is much harder to define an ontology (in its fullest sense), and to identify, understand, and validate all its implications, than it is to define a controlled vocabulary.

If XML fever sufferers come in contact with communities where Semantic Web ideas are widespread and well established, they quickly discover that most of their knowledge acquired in the basic and intermediate phases of the XML learning curve does not apply anymore. The reason for this is that the Semantic Web creates a completely self-contained world on top of other Web technologies, with the only intersection being the fact that resources are identified by URI. As a result, Semantic Web users become blissfully unaware that the Web may have solutions for them or that there could be a simpler way of solving problems. Seeing the Semantic Web as the logical next step of the Web’s evolution, we can observe the following condition:

*Web blindness.* This is a condition in which the victim settles into the Semantic Web to a degree where the non-Semantic Web does not even exist anymore. In the pure Semantic Web, lower-level technologies no longer need to evolve, because every problem can be solved on the semantic layer. Web blindness victims often are only dimly aware that many problems in the real world are and most likely will be solved with technologies other than

Semantic Web technologies.

If victims of Web blindness have adjusted to their new environment of abundant RDF and start embracing the new world, they may come in contact with applications that have aggregated large sets of RDF data. While RDF triples are a seemingly simple concept, the true power of RDF lies in the fact that these triples are combined to form interconnected graphs of statements about things, and statements about statements, which quickly makes it impossible to use this dataset without specialized tools. These tools require specialized data storage and specialized languages for accessing these stores. Handling these large sets of data is the leading cause of an RDF-specific ailment:

*Triple shock.* While RDF itself is simple, large datasets easily contain millions of triples (for truly large datasets this can go up to billions), and managing and querying such a big dataset can become a considerable challenge. If the schema of these large datasets is simple, but ontology overkill has set in and it has been reformulated as an ontology, handling this dataset may become considerably harder, without any immediate benefit.

Semantic Web technologies may be the correct choice for projects requiring fully developed ontologies, but Semantic Web technologies have little to do with the plain Web and XML. This means that neither should be regarded as a cure for basic or intermediate XML fevers, and that each has its own set of issues, which are only partially listed here.

### The Prescription

We probably cannot prevent these varieties of XML fever, especially the basic strains, because it is undoubtedly a result of the hype and overbroad claims for XML that many people try it in the first place. We can do a better job of inoculating XML novices and users against the intermediate and advanced strains, however, by teaching them that the appropriate use of XML technologies depends on the nature and scope of the problems to which they are applied. Heavyweight XML specifications such as those developed by OASIS, OMG, and other standards organizations are necessary to build robust enterprise-class XML applications, and

Semantic Web concepts and tools are prerequisites for knowledge-intensive computation, but more lightweight approaches for structuring and classifying information such as microformats will do in other contexts.

When someone first learns about it, XML may seem like the hammer in the cliché about everything looking like a nail. Those of us who teach XML, write about it, or help others become effective users of it, however, can encourage a more nuanced view of XML tools and technologies that portrays them as a set of hammers of different sizes, with a variety of grips, heads, and claws. We need to point out that not everyone needs a complete set of hammers, but information architects should know how to select the appropriate hammer for the kind of hammering they need to do. And we should always remember that pounding nails is only one of the tasks involved in design and construction.

XML has succeeded beyond the wildest expectations as a convenient format for encoding information in an open and easily computable fashion. But it is just a format, and the difficult work of analysis and modeling information has not and will never go away. ■

### References

1. Bell, A.E. Death by UML fever. *ACM Queue* 2, 1 (Mar. 2004), 72-80.
2. Berners-Lee, T., Hendler, J.A., Lassila, O. The Semantic Web. *Scientific American* 284, 5 (May 2001), 34-43.
3. Bray, T. On language creation. In *Proceedings of XML 2005* (Atlanta, GA, Nov. 2005).
4. Bray, T., Paoli, J., Michael Sperberg-McQueen, C. Extensible markup language (XML) 1.0. World Wide Web Consortium, Recommendation REC-xml-19980210 (Feb. 1998).

**Erik Wilde** (dret@berkeley.edu) is a visiting assistant professor in the School of Information at the University of California at Berkeley, where he is also technical director of the Information and Service Design program.

**Robert J. Glushko** (glushko@ischool.berkeley.edu) is an adjunct professor at the University of California at Berkeley in the School of Information, the director of the Center for Document Engineering, and one of the founding faculty members of the Information and Service Design program.

© 2008 ACM 0001-0782/08/0700 \$5.00



---

**Can flash memory become the foundation for a new tier in the storage hierarchy?**

---

**BY ADAM LEVENTHAL**

---

# Flash Storage Memory

THE PAST FEW YEARS have been an exciting time for flash memory. The cost has fallen dramatically as fabrication has become more efficient and the market has grown; the density has improved with the advent of better processes and additional bits per cell; and flash has been adopted in a wide array of applications.

The flash ecosystem has expanded and continues to expand—especially for thumb drives, cameras, ruggedized laptops, and phones in the consumer space. One area where flash has seen only limited success, however, is in the primary storage market. As the price trend for flash became clear in recent years, the industry anticipated its ubiquity for primary storage, with some so bold as to predict the impending demise of rotating media (undeterred, apparently, by the obduracy of magnetic tape). But flash has not lived up to these high expectations. The brunt of the effort to bring flash to primary storage has taken the form of solid-state disks (SSDs), flash memory pack-

aged in hard-drive form factors and designed to supplant conventional drives. This technique is alluring because it requires no changes to software or other hardware components, but the cost of flash per gigabyte, while falling quickly, is still far more than hard drives. Only a small number of applications have performance needs that justify the expense.

Although flash's prospects are tantalizing, the challenge is to find uses for it that strike the right balance between cost and performance. Flash should be viewed not as a replacement for existing storage, but rather as a means to enhance it. Conventional storage systems mix dynamic memory (DRAM)

and hard drives; flash is interesting because it falls in a sweet spot between those two components for both cost and performance in that flash is significantly cheaper and denser than DRAM and significantly faster than disk. Flash can accordingly augment the system to form a new tier in the storage hierarchy—perhaps the most significant *new tier* since the introduction of the disk drive with RAMAC in 1956.

### Properties of Flash

Flash has two distinct categories: NAND and NOR—designations that refer to the way the flash cells are arranged. NOR flash allows for random access and is best suited for random access memory, while NAND must be treated as blocks and is ideal for persistent storage. The rest of this article examines only NAND flash, the cheaper and more common variety, of which again there are two types: single-level cell (SLC) and multilevel cell (MLC). SLC stores a single binary value in each memory cell. The binary value is distinguished by two threshold voltages. MLC supports four or, recently, eight distinct values per memory cell corresponding to two or three bits of storage. Because of its improved longevity and performance, the conventional wisdom is that SLC is best suited for enterprise (that is, not consumer-grade) solutions, so our focus here is on SLC flash, its cost, power dissipation, performance, and longevity as compared with DRAM and disk drives (see Figure 1).

The cost per unit storage is what has brought flash to the forefront in recent years (see Figure 2). Earlier this decade, flash costs were on par with those of DRAM; now, flash devices are much less expensive: \$10–\$35 per GB for an SLC flash device compared with around \$100 per GB for DRAM. The cost trend appears to be continuing to widen the gap between flash and DRAM. Disk drives are still much cheaper than flash, weighing in at less than \$1 per GB for 7,200RPM drives and in the neighborhood of \$3 per GB for 15,000RPM drives.

The other exciting attribute of flash is its low power consumption. As the cost of power and the impetus toward green computing rise, so does the attractiveness of lower-power solutions.

While completely accurate comparisons between flash, DRAM, and hard drives are difficult because of differences in capacity and interfaces, it's fair to say that flash consumes significantly less power than those other system components, especially on a per-gigabyte basis. The accompanying table records the power consumption for some typical components to provide a broad sense for each type of device.

The performance of flash is a bit unusual in that it's highly asymmetric, posing a challenge for using it in a storage system. A block of flash must be erased before it can be written, which takes on the order of 1–2 ms for a block, and writing to erased flash requires around 200–300  $\mu$ s. For this reason flash devices try to maintain a pool of previously erased blocks so that the latency of a write is just that of the program operation. Read operations are much faster: approximately 25  $\mu$ s for 4k. By comparison, raw DRAM is even faster, able to perform reads and writes in much less than a microsecond. Disk-drive latency depends on the rotational speed of the drive: on average 4.2 ms for 7,200RPM, 3 ms for 10,000RPM, and 2 ms for 15,000RPM. Adding in the seek time bumps these latencies up an additional 3–10 ms depending on the quality of the mechanical components.

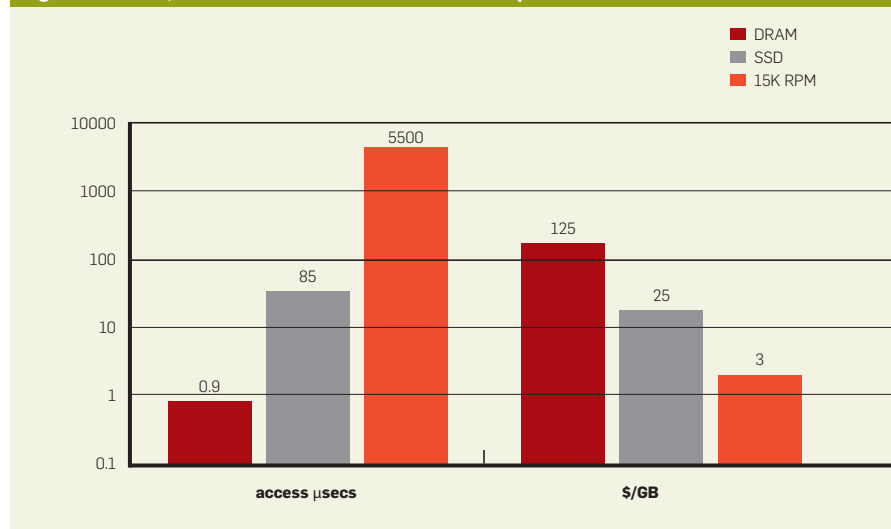
SLC flash is typically rated to sustain one million program/erase cycles per block. As flash cells are stressed, they lose their ability to record and retain values. Because of the limited lifetime, flash devices must take care to ensure

that cells are stressed uniformly so that “hot” cells don't cause premature device failure, a technique known as wear-leveling. Just as disk drives keep a pool of spare blocks for bad-block remapping, flash devices typically present themselves to the operating system as significantly smaller than the amount of raw flash to maintain a reserve of spare blocks (and pre-erased blocks to improve write performance). Most flash devices are also capable of estimating their own remaining lifetimes so systems can anticipate failure and take prophylactic action.

### Today's Storage Hierarchy

Whether over a network or for local access, primary storage can be succinctly summarized as a head unit containing CPUs and DRAM attached to drives either in storage arrays or JBODs (just a bunch of disks). The disks comprise the primary repository for data—typical modern data sets range from a few hundred gigabytes up to a petabyte or more—while DRAM acts as a very fast cache. Clients communicate via read and write operations. Read operations are always synchronous in that the client is blocked until the operation is serviced, whereas write operations may be either synchronous or asynchronous depending on the application. For example, video streams may write data blocks asynchronously and verify only at the end of the stream that all data has been quiesced; databases, however, use synchronous usually writes to ensure that every transaction has been

Figure 1: DRAM, 15K RPM drives and SSD: Price and performance.



committed to stable storage.

On a typical system, the speed of a synchronous write is bounded by the latency of nonvolatile storage, as writes must be committed before they can be acknowledged. Read operations first check in the DRAM cache providing very low-latency service times, but cache misses must also wait for the slow procession of data around the spindle. Since it's quite common to have working sets larger than the meager DRAM available, even the best prefetching algorithms will leave many read operations blocked on the disk.

A brute-force solution for improving latency is simply to spin the platters faster to reduce rotational latency, using 15,000RPM drives rather than 10,000 or 7,200RPM drives. This will improve both read and write latency, but only by a factor of two or so. For example, using drives from a major vendor, at current prices, a 10TB data set on a 7,200RPM drive would cost about \$3,000 and dissipate 112 watts; the same data set on a 15,000RPM drive would cost \$22,000 and dissipate 473 watts—all for a latency improvement of a bit more than a factor of two. The additional cost and power overhead make this an unsatisfying solution, though it is widely employed absent a clear alternative.

A focused solution for improving the performance of synchronous writes is to add nonvolatile RAM (NVRAM) in the form of battery-backed DRAM, usually on a PCI card. Writes are committed to the NVRAM ring buffer and immedi-

ately acknowledged to the client while the data is asynchronously written out to the drives. Once the data has been committed to disk, the corresponding record can be freed in the NVRAM. This technique allows for a tremendous improvement for synchronous writes, but suffers some downsides. NVRAM is quite expensive; batteries fail (or leak, or, worse, explode); and the maximum size of NVRAM tends to be small (2GB–4GB)—small enough that workloads can fill the entire ring buffer before it can be flushed to disk.

### Flash as a Log Device

One use of flash is as a stand-in for NVRAM that can improve write performance as a log device. To that end you need a device that mimics the important properties of NVRAM (fast, persistent writes), while avoiding the downsides (cost, size, battery power). Recall, however, that while achieving good write bandwidth is fairly easy, the physics of flash dictate that individual writes exhibit relatively high latency. However, it's possible to build a flash-based device that can service write operations very quickly by inserting a DRAM write cache and then treating that write cache as nonvolatile by adding a supercapacitor to provide the necessary power to flush outstanding data in the DRAM to flash in the case of power loss.

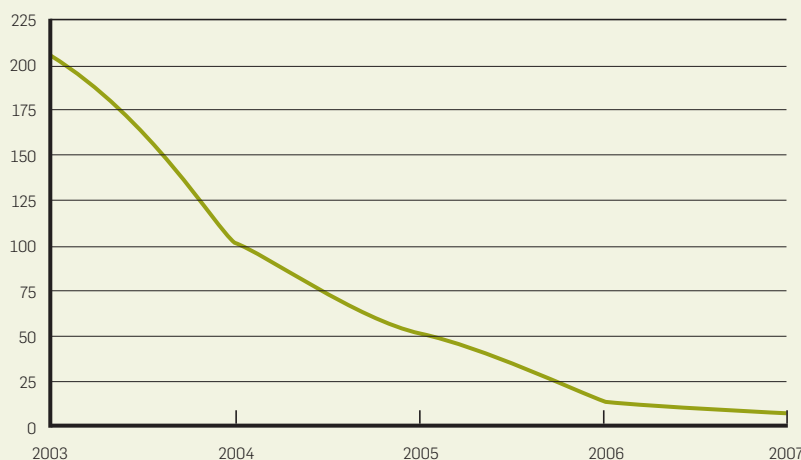
Many applications such as databases can use a dedicated log device as a way of improving the performance of write operations; for these applica-

tions, such a device can be dropped in easily. To bring the benefits of a flash log device to primary storage, and therefore to a wide array of applications, we need similar functionality in a general-purpose file system. Sun's ZFS provides a useful context for the use of flash. ZFS, an enterprise-class file system designed for the scale and requirements of modern systems, was implemented from scratch starting in 2001. It discards the model of a file system sitting on a volume manager in favor of pooled storage both for simplicity of management and greater flexibility for optimizing performance. ZFS maintains its on-disk data structures in way that is always consistent, eliminating the need for consistency checking after an unexpected power failure. Furthermore, it is flexible enough to accommodate new technological advances, such as new uses of flash. (For a complete description of ZFS, see <http://opensolaris.org/os/community/zfs/>.)

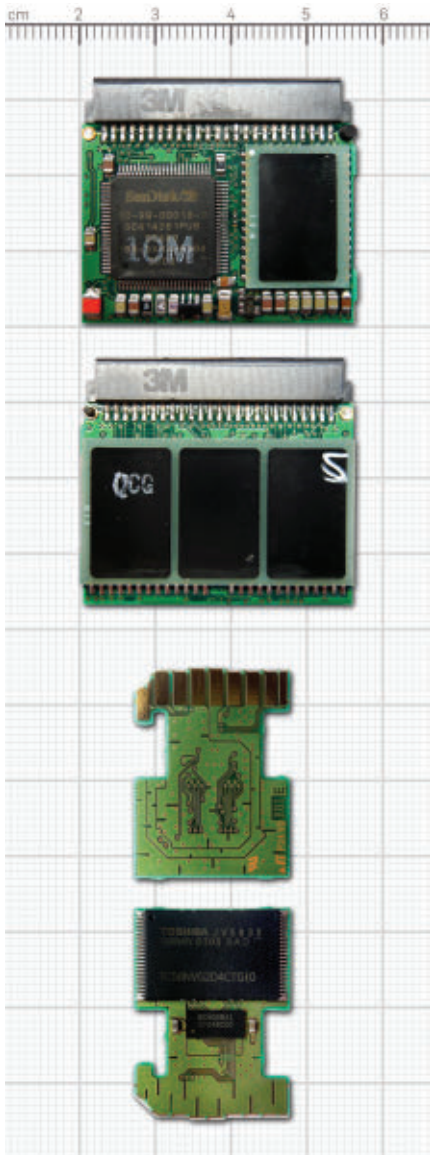
ZFS provides for the use of a separate intent-log device (a slog in ZFS jargon) to which synchronous writes can be quickly written and acknowledged to the client before the data is written to the storage pool. The slog is used only for small transactions, while large transactions use the main storage pool—it's tough to beat the raw throughput of large numbers of disks. The flash-based log device would be ideally suited for a ZFS slog. The write buffer on the flash device has to be only large enough to saturate the bandwidth to flash. Its DRAM size requirements—and therefore the power requirements—are quite small. Note also the write buffer is much smaller than the required DRAM in a battery-backed NVRAM device. There are effectively no constraints on the amount of flash that could be placed on such a device, but experimentation has shown that 10GB of delivered capacity is more than enough for the vast majority of use cases.

Using such a device with ZFS in a test system, we measured latencies in the range of 80–100  $\mu$ s. This approaches the performance of NVRAM and has many other benefits. A common concern for flash is its longevity. SLC flash is often rated for one million write/erase cycles, but beyond several hundred thousand, the data-retention

Figure 2: Flash cost per GB.



period can drop to just a few weeks. ZFS will write to this device as a slog in 8KB chunks with each operation taking 80  $\mu$ s. On a device with 10GB of raw flash, this equates to about 3½ years of constant use. A flash device with a formatted capacity of 10GB will, however,



**Top: A 10MB compact flash card from 1996.**  
**Bottom: A 2GB SD flash card from 2008.**

typically have 20%–50% more flash held in reserve, easily taking the longevity of such a device under constant use to five years, and the device itself can easily report its expected remaining lifetime as it counts down its dwindling reserve of spare blocks. Further, data needs to be retained only long enough for the system to recover from a fatal error; a reasonable standard is 72 hours, so a few weeks of data retention, even for very old flash cells, is more than adequate and a vast improvement on NVRAM.

### Flash as a Cache

The other half of this performance picture is read latency. Storage systems typically keep a DRAM cache of data the system determines a consumer is likely to access so that it can service read requests from that cache rather than waiting for the disk. In ZFS, this subsystem is called the adaptive replacement cache (ARC). The policies that determine which data is present in the ARC attempt to anticipate future needs, but read requests can still miss the cache as a result of bad predictions or because the working set is simply larger than the cache can hold—or even larger than the maximum configurable amount of DRAM on a system. Flash is well suited for acting as a new second-level cache in between memory and disk in terms of capacity and performance. In ZFS, this is called the L2ARC.

ZFS fills the L2ARC using large, asynchronous writes and uses the cache to seamlessly satisfy read requests from clients. The requirements here are a perfect fit for flash, which inherently has sufficient write bandwidth and fantastic read latency. Since these devices can be external—rather than being attached to the main board, as is the case with DRAM—the size of the L2ARC is limited only by the amount of DRAM

required for bookkeeping (at a ratio of 50:1 in the current ZFS implementation). For example, the maximum memory configuration on a four-socket machine is usually around 128GB; such a system can easily accommodate 768GB or more using flash SSDs in its internal drive bays. ZFS's built-in checksums catch cache inconsistencies and mean that defective flash blocks simply lead to fewer cache hits rather than data loss.

In the context of the memory hierarchy, caches are often populated as entries are evicted from the previous layer—in an exclusive cache architecture, on-chip caches are evicted to off-chip caches, and so on. With a flash-based cache, however, the write latency is so poor the system could easily be bogged down waiting for evictions. Accordingly, the L2ARC uses an evict-ahead policy: it aggregates ARC entries and predictively pushes them out to flash, thus amortizing the cost over large operations and ensuring that there is no additional latency when the time comes to evict an entry from the ARC. The L2ARC iterates over its space as a ring, starting back at the beginning once it reaches the end, thereby avoiding any potential for fragmentation. Although this technique does mean that entries in the L2ARC that may soon be accessed could be overwritten prematurely, bear in mind that the hottest data will still reside in the DRAM-based ARC. ZFS will write to the L2ARC slowly, meaning that it can take some time to warm up; but once warm, it should remain so, as long as the writes to the cache can keep up with data churn on the system.

It's worth noting that to this point the L2ARC hasn't even taken advantage of what is usually considered to be a key feature of flash: nonvolatility. Under normal operation, the L2ARC treats flash as cheap and vast storage. As it writes blocks of data to populate the cache devices, however, the L2ARC includes a directory so that after a power loss, the contents of the cache can be identified, thus pre-warming the cache. Although resets are rare, system failures, power failures, and downtime due to maintenance are all inevitable; the instantly warmed cache reduces the slow performance ramp typical of a system after a reset. Since

### Power consumption for typical components.

| Device                             | Approximate Power Consumption |
|------------------------------------|-------------------------------|
| DRAM DIMM module (1GB)             | 5W                            |
| 15,000 RPM drive (300GB)           | 17.2W                         |
| 7,200 RPM drive (750GB)            | 12.6W                         |
| High-performance flash SSD (128GB) | 2W                            |




the L2ARC writes slowly to its flash devices and data on the system may be modified quickly (especially with the use of flash as a log device), the contents of the L2ARC may not reflect the same data stored on disk. During normal operation, dirtied and stale entries are marked as such so they are ignored. After a system reset, though stale data may be read off the cache device, metadata kept on the device and ZFS's built-in checksums are used to identify this condition and seamlessly recover by reading the correct data from disk.


For working sets that are larger than the DRAM capacity, flash offers an avenue to access that working set much faster than could otherwise be done by disks of any speed. Even for working sets that could comfortably fit in DRAM, if the absolute performance of DRAM isn't necessary, it may be more economical to skimp on DRAM for the main ARC and instead cache the data on flash. As this use of flash meshes perfectly with its natural strengths, suitable devices can be produced quite cheaply and still have a significant performance advantage over fast disks. Although flash is still more expensive than fast disks per unit storage, caching even a very large working set in flash is often cheaper than storing all data on fast disks.

### The Impact of Flash

By combining the use of flash as an intent-log to reduce write latency with flash as a cache to reduce read latency, we can create a system that performs far better and consumes less power than other systems of similar cost. It is now possible to construct systems with a precise mix of write-optimized flash, flash for caching, DRAM, and cheap disks designed specifically to achieve the right balance of cost and performance for any given workload, with data automatically handled by the appropriate level of the hierarchy. It is also possible to address specific performance problems with directed rather than general solutions. Through the use of smarter software, we can build systems that integrate different technologies to extract the best qualities of each. Further, the use of smarter software will allow flash vendors to build solutions for specific problems rather than gussying up flash to fit the anachronistic



**Although flash's prospects are tantalizing, the challenge is to find uses for it that strike the right balance between cost and performance. Flash should be viewed not as a replacement for existing storage, but rather as a means to enhance it.**



constraints of a hard drive. ZFS is just one example among many of how one could apply flash as a log and a cache to deliver total system performance. Most generally, this new flash tier can be thought of as a radical form of hierarchical storage management (HSM) without the need for explicit management. Although these solutions offer concrete methods of integrating flash into a storage system, they also raise a number of questions and force us to reconsider many aspects of the system. For example, how should we connect flash to the system? SSDs are clearly an easy approach, but there may be faster interfaces such as the memory bus. More broadly, how will this impact the balance of a system? As more requests are serviced from flash, it may be possible to provision systems with far more network connectivity to clients than bus connectivity to disks.

In that vein, flash opens the possibility of using disks that are even slower, cheaper, and more power efficient. We can now scoff at a 15,000RPM drive as an untargeted half-measure for a variety of problems, but couldn't the same argument be applied to a 7,200RPM drive? Just because it's at the low end of the performance curve doesn't mean it's at the bottom. The 5,400RPM drive is quite common today and consumes less power still. Can the return of the 3,600RPM drive be far behind? The cost of power has continued to rise, but even if that trend were to plateau, a large portion of the total cost of ownership of a storage system is directly tied to its power use—and that's to say nothing of the increased market emphasis on green design. Flash provides solutions that require us to rethink how we build systems and challenge us to develop smarter software to drive those systems; the result will be faster systems that are cheaper and greener. ■

---

**Adam Leventhal** (ahl@sun.com) is a staff engineer on Sun's Microsystems' Fishworks advanced product development team, San Francisco, CA.

Props to Neil Perrin for developing slogs, to Brendan Gregg for developing the L2ARC, and to Jeff Bonwick and Matt Ahrens for reinventing storage with ZFS.

© 2008 ACM 0001-0782/08/0700 \$5.00

**There is more to data access than SQL.**

BY MARGO SELTZER

# Beyond Relational Databases

THE NUMBER AND VARIETY of computing devices in the environment are increasing rapidly. Real computers are no longer tethered to desktops or locked in server rooms. PDAs, highly mobile tablet and laptop devices, palmtop computers, and mobile telephony handsets now offer powerful platforms for the delivery of new applications and services. These devices are, however, only the tip of the iceberg. Hidden from sight are the many computing and network elements required to support the infrastructure that makes ubiquitous computing possible.

With so much computing power traveling around in briefcases and pockets, developers are building applications that would have been impossible just a few years ago. Among the interesting services available today are text and multimedia messaging, location-based search and information services (for example,

on-demand reviews of nearby restaurants), and ad hoc multiplayer games. Over the next several years, new classes of mobile and personalized services, impossible to predict today, will certainly be developed.

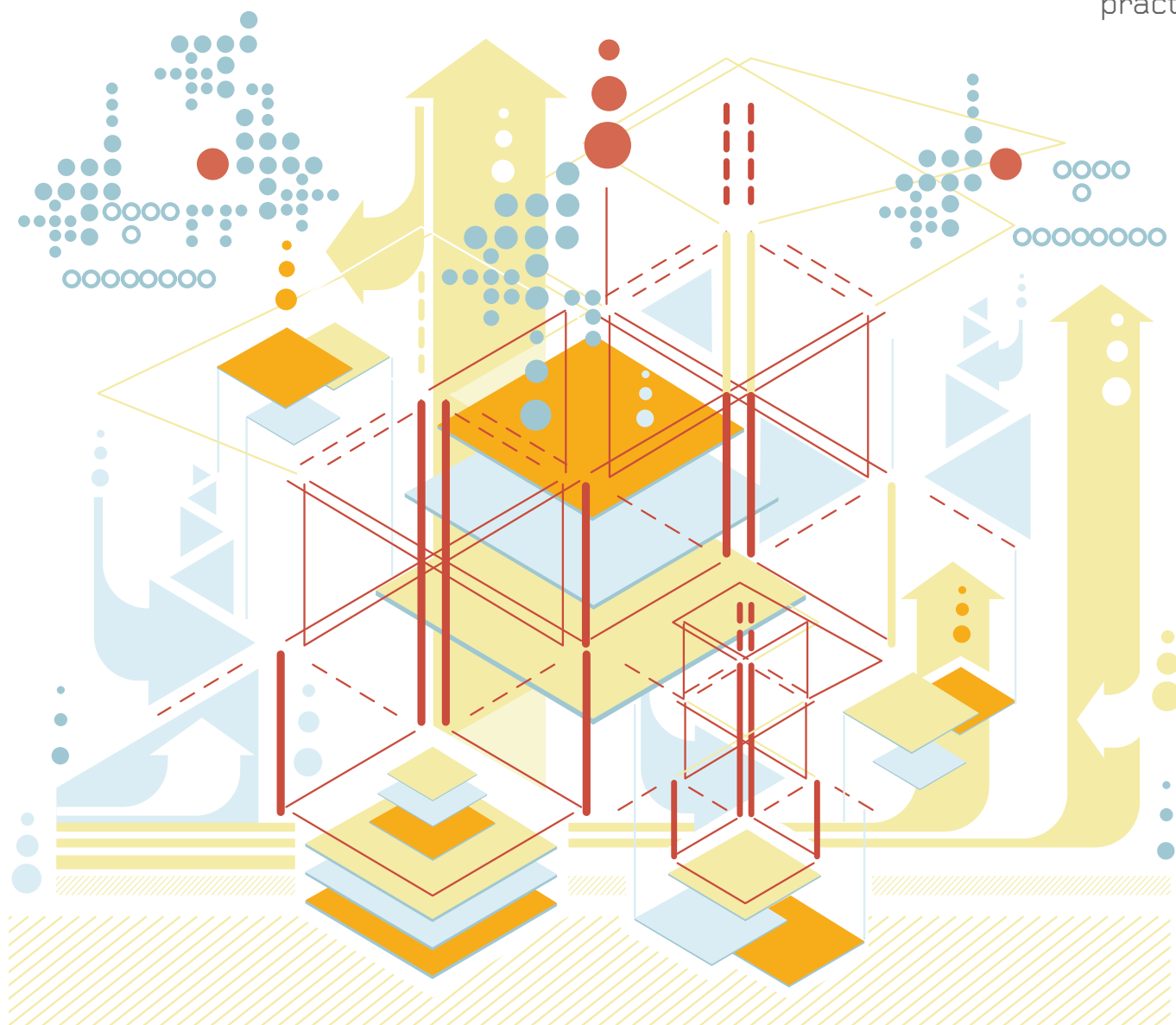
While these services differ from one another in major ways, they also share some important attributes. One—the focus of this article—is the need for data storage and retrieval functions built into the application. Messaging applications need to move messages around the network reliably and without loss. Location-based services need to map physical location to logical location (for example, GPS or cell-tower coordinates to postal code) and then look up location-based information. Gaming applications must record and share the current state of the game on distributed devices and must manage content retrieval and delivery to each of the devices in real time. In all these cases, fast, reliable data storage and retrieval are critical.

As soon as the discussion turns to data storage and retrieval, relational databases come to mind. Relational databases have been tremendously successful over the past three decades and SQL has become the lingua franca for data access. While *data management* has become almost synonymous with RDBMS, however, there are an increasing number of applications for which lighter-weight alternatives are more appropriate.

This article begins with a brief review of how relational systems came to dominate the data management landscape, and discusses how the relational technologies have evolved. It presents a data-centric overview of today's emergent applications, and delves into data management needs for today's and tomorrow's applications.

## Relational Prehistory

Relational databases came out of research at IBM<sup>1,5</sup> and the University of California at Berkeley<sup>7</sup> in the 1970s. Relational databases were fundamentally a reaction to escalating costs in deploy-



ing and maintaining complex systems.

The key observation was that programmers, who were very expensive, had to rewrite large amounts of application software manually whenever the content or physical organization of a database changed. Because the application generally knew in detail how its data was stored, including its on-disk layout, reorganizing databases or adding new information to existing databases forced wholesale changes to the code accessing those databases.

Relational databases solved this problem in two ways. First, they hid the physical organization of the database from the application and provided only a logical view of the data. Second, they used a declarative language to describe the data of interest in a particular query, rather than forcing the programmer to write a collection of function calls

to fetch the data. These two changes allowed programmers to describe the information they wanted and to leave the details of optimization and access to the database management system. This transformation relieved programmers of the burden of rewriting application code whenever the database layout or organization changed.

Relational databases enjoyed tremendous success in the IT shops and data centers of the world. Businesses with large quantities of data to manage and sophisticated applications using that data adopted the new technology quickly. Demand for relational products created a market worth billions of dollars in licensing revenue per year. Several RDBMS vendors arose in the 1980s to compete for this lucrative business.

In the 20 years that followed, two

related trends emerged. First, the RDBMS vendors increased functionality to provide market differentiators and to address each new market niche as it arose. Second, few applications need all the features available in today's RDBMSs, so as the feature set size increased, each application used a decreasing fraction of that feature set.

This drive toward increasing DBMS functionality has been accompanied by increasing complexity, and most deployments now require a specialist, trained in database administration, to keep the systems and applications running. Since these systems are developed and sold as monolithic entities, even though applications may require only a small subset of the system's functionality, each installation pays the price of the total overall complexity. Surely, there must be a better way.



## The New Frontier

We are not the first to notice these tides of change. In 1998, the leading database researchers concluded that database management systems were becoming too complex and that automated configuration and management were becoming essential.<sup>2</sup> Two years later, Surajit Chaudhuri and Gerhard Weikum proposed radically rethinking database management system architecture.<sup>4</sup> They suggested that database management systems be made more modular and that we broaden our thoughts about data management to include rather simple, component-based building blocks. Most recently, Michael Stonebraker joined the chorus, arguing that “one size no longer fits all,” and citing particular application examples where the conventional RDBMS architecture is inappropriate.<sup>8</sup>

As argued by Stonebraker, the relational vendors have been providing the illusion that an RDBMS is the answer to any data management need. For example, as data warehousing and decision support emerged as important application domains, the vendors adapted products to address the specialized needs that arise in these new domains. They do this by hiding fairly different data management implementations behind the familiar SQL front end. This model breaks down, however, as one begins to examine emerging data needs in more depth.

**Data warehousing.** Retail organizations now have the ability to record every customer transaction, producing an enormous data source that can be mined for information about custom-

ers’ purchasing patterns, trends in product popularity, geographical preferences, and countless other phenomena that can be exploited to increase sales or decrease the cost of doing business. This database is *read-mostly*: it is updated in bulk by periodically adding new transactions to the collection, but it is read frequently as analysts cull the data extracting useful tidbits. This application domain is characterized by enormous tables (tens or hundreds of terabytes), queries that access only a few of the many columns in a table, and a need to scan tables sorted in a number of different ways.

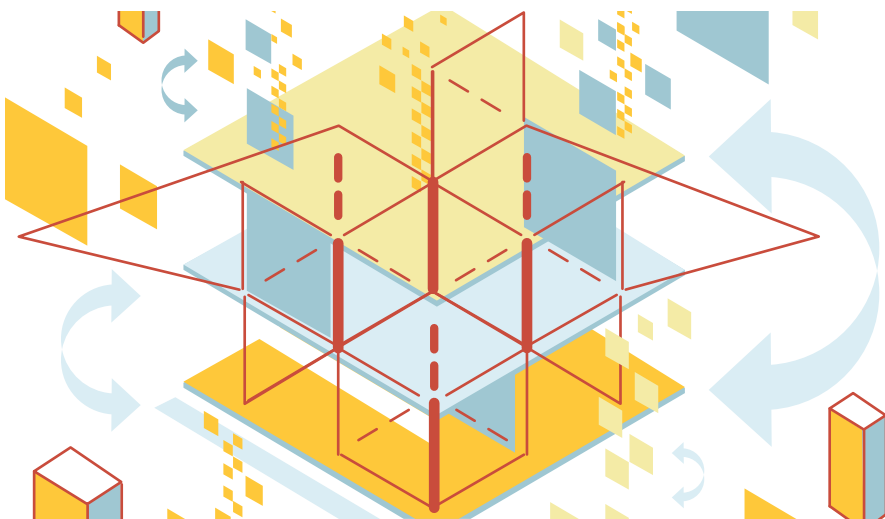
**Directory services.** As organizations become increasingly dependent upon distributed resources and personnel, the demand for directory services has exploded.<sup>3</sup> Directory servers provide fast lookup of entities arranged in a hierarchical structure that frequently matches the hierarchical structure of an organization. The LDAP standard emerged in the 1990s in response to the heavyweight ISO X.400/X.500 directory services. LDAP is now at the core of authentication and identity management systems from a number of vendors (for example, IBM Tivoli’s Directory Server, Microsoft’s Active Directory Server, the Sun ONE Directory Server). Like data warehousing, LDAP is characterized by read-mostly access. Queries are either single-row retrieval (find the record that corresponds to this user) or lookups based on attribute values (find all users in the engineering department). The prevalence of multivalued attributes makes a relational representation quite inefficient.

**Web search.** Internet search engines lie at the intersection of database management and information retrieval. The objects upon which they operate are typically semistructured (that is, HTML instead of raw text), but the queries posed are most often keyword lookups where the desired response is a sorted list of possible answers. Practically all the successful search engines today have developed their own data management solution to this problem, constructing efficient inverted indices and highly parallelized implementations of index and lookup. This application is read-mostly with bulk updates and nontraditional indexing.

**Mobile device caching.** The prevalence of small, mobile devices introduces yet another category of application: caching relevant portions of a larger dataset on a smaller, low-functionality device. While today’s users think of their cell phone’s directory as their own data collection, another view might be to think of it as a cache of a global phone and address directory. This model has attractive properties—in particular, the ability to augment the local dataset with entries as they are used or needed. Mobile telephony infrastructure requires similar caching capabilities to maintain communication channels to the devices. The access pattern observed in these caches is also read-mostly, and the data itself is completely transitory; it can be lost and regenerated if necessary.

**XML management.** Online transactions are increasingly being conducted by exchanging XML-encoded documents. The standard solution today involves converting these documents into a canonical relational organization, storing them in an RDBMS, and then converting again when one wishes to use them. As more documents are created, transmitted, and operated upon in XML, these translations become unnecessary, inefficient, and tedious. Surely there must be a better way. Native XML data stores with Xquery and Xpath access patterns represent the next wave of storage evolution. While new items are constantly added to and removed from an XML repository, the documents themselves are largely read-only.

**Stream processing.** Stream processing is a bit of an outcast in this laundry list of data-intensive applications.




Strictly speaking, stream processing is not a data management task; it is a data-filtering task. That is, data is produced at some source and sent streaming to recipients that filter the stream for “interesting” events. For example, financial institutions watch stock tickers looking for hotly traded items and/or stocks that aren’t being traded as heavily as expected.

The reason that these stream-processing applications are included here is a linguistic one: the filters that are typically desired in these environments *look* like SQL; however, while SQL was designed to operate on persistently stored tables, these queries act upon a real time stream of data values. Stonebraker explains in some depth how poorly equipped databases are for this task. Perhaps the bigger surprise is not that database systems are poorly equipped to address this task, but that because SQL appears to be the “right” query language, developers use relational database systems for applications that have no persistent storage!


Stream processing represents a class of applications that could benefit from a SQL-like query language atop a data management system with properties that are radically different from an RDBMS. Since streaming queries frequently operate on data observed during a time window, some transient local storage is necessary, but this storage needn’t be persistent, transactional, or support complex query processing. Instead, it must be blindingly fast. Although relational databases are well-equipped to handle dynamic queries over relatively static or slowly changing data, this application class is characterized by a fairly static query set over highly dynamic data.

### Flexible Solutions

Relational systems have been designed to satisfy online transaction processing (OLTP) workloads characterized by ad hoc queries, significant write traffic, and the need for strong transactional and integrity guarantees. In contrast, the applications described here are almost all read-dominated, and streaming applications don’t even take advantage of persistent data, just an SQL-like query language. Few of these applications require transactional guarantees, and there is little inherently relational



**There are fundamentally two properties that a solution must possess to address the wide range of application needs emerging today: modularity and configurability.**



about the data being accessed. Thus, the data management question becomes how best to satisfy the needs of these different types of applications. We claim (like Stonebraker) that there really is no single right answer. Instead, we must focus on flexible solutions that can be tailored to the needs of a particular application.

There are several ways to deliver flexibility in today’s changing data environment. The back-to-basics approach is to require that every single application build its own data storage service. This option, while seemingly simple, is impractical in all but the simplest of applications. Some data-intensive applications running today, however, are built upon simple, homegrown solutions.

The second way to address the need for flexibility is to provide a smorgasbord of data management options, each of which addresses a particular application class. We see this approach emerging in the traditional relational market, where the SQL veneer is used to hide the different capabilities required for OLTP and data warehousing.

The third approach to flexibility is to produce a storage engine that is more configurable so that it can be tuned to the requirements of individual applications. This solution has the advantage of allowing concentrated investment in a single storage system, improving quality. Configurability, however, makes new demands of developers who use the database, since they must understand the configuration options and then integrate the data management component properly into their product designs.

In fact, the solution emerging in the marketplace is to have a handful of reasonably configurable storage systems, each of which is useful across a broad application class.


There are fundamentally two properties that a solution must possess to address the wide range of application needs emerging today: modularity and configurability. Few applications require all the functionality possible in a data management system. If an application doesn’t need functionality, it should not have to “pay” for that functionality in size (footprint, memory consumption, disk utilization, and so on), complexity, or cost. Therefore, a flexible engine must allow

the developer to use or exclude major subsystems depending on whether the application needs them. Once a system is sufficiently modular to permit a truly small footprint, we will find that system deployed on an array of hardware platforms with staggeringly large differences in capabilities. In these cases, the system must be configurable to its operating environment: the specific hardware, operating system, and application using it.


### Modularity

Some argue that database architecture is in need of a revolution akin to the RISC revolution in computer hardware. The conventional monolithic DBMS architecture is not facile enough to adapt to today's data demands, so we must build data management capabilities out of a collection of small, simple, reusable components. For example, instead of viewing SQL as a simple binary decision, Chaudhuri and Weikum argue that query capabilities should be provided at different levels of sophistication: a single-table selection processor that has a B+ tree index that supports simple indexing, updating, and selection. To this, you might add transactions. Continuing up the complexity hierarchy, consider a select-project-join processor. Next, add aggregates. In this manner, you transform SQL from a monolithic language into a family of successively richer languages, each of which is provided as a component and satisfies a significant number of application domains. Any particular application selects the components it needs. This idea of a component-based architecture can be extended to include several other aspects of database design: concurrency control, transactions, logging, and high availability.

Concurrency control lends itself to a hierarchy similar to that presented in the language example. Some applications are completely single-threaded and require no locking; others have low levels of concurrency and would be well served by table-level locks or API-level locks (allowing only one writer or multiple readers into the database system simultaneously); finally, highly concurrent applications need fine-grain locking and multiple degrees of isolation (potentially allowing applications to see values that have been written by



**Old-style database systems solve old-style problems; we need new-style databases to solve new-style problems.**



incomplete transactions).<sup>6</sup> In a conventional database management system, locking is assumed; in the brave new world discussed here, locking is optional and different components can be used to provide different levels of concurrency.

Transactions provide the illusion that a collection of operations are applied to a database in an atomic unit and that once applied, the operations will persist, even in the face of application or system failure. Transaction management is at the heart of most database management systems, yet many applications do not require transactions. In a component-based world, transactions, too, are optional. When they are present, a system might still have a number of different components providing basic transactional mechanisms, savepoints (the ability to identify a point in time to which the database may be rolled back), two-phase commit to support transactions that span multiple databases, nested transactions to decompose a large operation into a number of smaller ones, and compensating transactions to undo high-level, logical operations.

Many transaction systems use some form of logging to provide rollback and recovery capabilities. In that context, it hardly seems necessary to treat logging as a separable component, but it should be. A transactional component might be designed to work with multiple implementations, some of which do not use logging (for example, no-overwrite schemes such as shadow-pages). Perhaps even more interesting, a logging system might be useful outside the context of transactions; it might be used for auditing or provide some sort of backup mechanism. In either case, it should be an application designer's decision whether logging is necessary rather than having it imposed by the database vendor.

Finally, data is sometimes so critical that downtime is unacceptable. Many database systems provide replicated or highly available systems to address this need. Although this functionality is often available as an add-on in today's systems, they have not gone far enough. A developer may wish to use a database's HA (high-availability) configuration, but may use it in conjunction with some other company's HA substrate. If



the application already has a substrate that performs heartbeat protocols (or any other mechanism that notifies the application or system when a component fails), fail-over, and redundant communication channels, then you will want to exclude those components from the database management system and hook into the existing functionality. Monolithic systems do not allow this, whereas a component-based, modular architecture does.

In addition to providing smaller, simpler applications, components with well-defined, clean, exposed interfaces provide for a degree of extensibility that is simply not possible in a monolithic system. For example, consider the basic set of components needed to construct a transactional system: a transaction manager, a lock manager, and a log manager. If these modules are open and extensible, then the developer can build systems that incorporate items that are not managed by the database system into transactions. Consider, for example, a network switch: the state of the configuration database depends on the state of hardware inside the device, and vice versa. If the electrical control over chips and boards can be incorporated into transactions, by allowing the programmer to extend the locking and logging system to communicate with them, then operations such as “power up the backup network interface card” can be made transactional.

Modularity is a powerful tool for managing size and complexity of applications and systems while also enabling the application and data management capabilities to seamlessly interact. Thus, we have proposed an architecture that enables developers to exclude functionality they do not need and include functionality they do need but is not provided by the database vendor.

### Configurability

The second property of a flexible data management system is configurability. Whereas modularity is an architectural mechanism, configuration is mostly a runtime mechanism. With a component-based architecture, the build-time configuration is involved in selecting appropriate components. A single collection of components may still run on a range of systems with wildly different capabilities. For example, just because

two applications both want transactions and B-trees, this does not mean that both can support a multi-gigabyte in-memory cache. The ability to adapt to radically different circumstances is critical. Configurability refers to how well a system can be matched to its environment and application needs. In this article we discuss configurability with respect to the hardware, the environment in which the application runs (for example, the operating system), the application’s software architecture, and the “natural” data format of the application.

Hardware environments introduce variability in CPU speed, memory size, and persistent storage capabilities. Variability in CPU speed and persistent storage introduces the possibility of trading computation for disk bandwidth. On a fast processor, it may be beneficial to compress data, consuming CPU cycles, in order to save I/O; on a PDA, where CPU cycles are sparse and persistent I/O is fast, compression might not be the right trade-off.

In a world where resource-constrained devices require potentially sophisticated data management, developers must have control over the memory and disk consumption policies of the database. In different environments, applications may need control over the maximum size of in-memory data structures, the maximum size of persistent data, and the space consumed by transactional logs. Policies for consumption of these resources must be set by the application developer, not the end user, since the developer is more likely to have the technical savvy necessary to

make the right decisions.

Variability in persistent storage technologies places new demands on the database engine as well. Not only must it work well in the presence of spinning, magnetic storage, but it should also run well on other media (for example, flash) with constraints on behaviors (such as the number of writes to a particular memory location), and it may need to run in the absence of any persistent storage. For example, some applications want to manage data entirely in main memory, with no persistence; some want to manage data with full synchronous transactional guarantees on updates; and some need something in the middle. Each of these policies should be implemented by the same transactional component, but the database should allow the programmer to control whether or not data persists across power-down events and the strictness of any transactional assurances that the system makes to the end user.

Although many embedded systems are now able to use commodity off-the-shelf hardware platforms, many proprietary devices still exist. The ubiquitous data management solution will be portable to these special-purpose hardware devices. It will also be portable to a variety of operating systems as well; the services available from the operating system on a mobile telephone handset are different from those available on a 64-way multiprocessor with gigabytes of RAM, even if both are running Linux. If the data management system is to run everywhere, then it must rely only on the services common to most oper-



ating systems, and it must provide explicit mechanisms to allow portability, through simple interposition libraries or source-code availability.

Even on a single platform, the developer makes architectural choices that affect the database system. For example, a system may be built using: a single thread of control; a collection of cooperating processes, each of which is single-threaded; multiple threads of control in a single process; multiple multithreaded processes; or a strictly event-based architecture. These choices are driven by a combination of the application's requirements, the developer's preferences, the operating system, and the hardware. The database system must accommodate them.

The database must also avoid making decisions about network protocols. Since the database will run in environments where communication takes place over backplanes, as well as environments where it takes place over WANs, the developer should select the appropriate communication infrastructure. A special-purpose telephone switch chassis may include a custom backplane and protocol for fast communication among redundant boards; the database must not prevent the developer from using it.

Up to this point, configurability has revolved around adapting to the hardware and software environment of the application. The last area of configuration that we address revolves around the application's data. Data layout, indexing, and access are critical performance considerations. There are three main design points with respect to data: the physical clustering, the indexing mechanism, and the internal structure of items in the database. Some of these, like the indexing mechanism, really are runtime configuration decisions, whereas others are more about giving the application the ability to make design decisions, rather than having designers forced into decisions because of the database management system.

Database management systems designed for spinning magnetic media expend considerable effort clustering related data together on disk so that seek and rotation times can be amortized by transferring a large amount of data per repositioning event. In general, this clustering is good, as long as


the data is clustered according to the correct criteria. In the case of a configurable database system, this means that the developer needs to retain control over primary key selection (as is done in most relational database management systems) and must be able to ignore clustering issues if the persistent medium either does not exist or does not show performance benefits to accessing locations that are "close" to the last access.

On a related note, the developer must be left the flexibility to select an indexing structure for the primary keys that is appropriate for the workload. Workloads with locality of reference are probably well served by B+ trees; those with huge datasets and truly random access might be better off with hash tables. Perhaps the data is highly dimensional and require a completely different indexing structure; the extensibility discussed in the previous section should allow a developer to provide an application-specific indexing mechanism and use it with all of the system's other features (for example, locking, transactions). At a minimum, the configurable database should provide a range of alternative indexing structures that support iteration, fast equality searches, and range searches, including searches on partial keys.

Unlike relational engines, the configurable engine should permit the programmer to determine the internal structure of its data items. If the application has a dynamic or evolving schema or must support ad hoc queries, then the internal structure should be one that enables high-level query access such as SQL, Xpath, Xquery, LDAP, etc. If, however, the schema is static and the query set is known, selecting an internal structure that maps more directly to the application's internal data structures provides significant performance improvements. For example, if an application's data is inherently nonrelational (for example, containing multivalued attributes or large chunks of unstructured data), then forcing it into a relational organization simply to facilitate SQL access will cost performance in the translation and is unlikely to reap the benefits of the relational store. Similarly, if the application's data was relational, forcing it into a different format (for example,

XML, object-oriented, among others) would add overhead for no benefit. The configurable engine must support storing data in the format that is most natural for the application. It is then the programmer's responsibility to select the format that meets the "most natural" criteria.

### New-Style Databases for New-Style Problems

Old-style database systems solve old-style problems; we need new-style databases to solve new-style problems. While the need for conventional database management systems isn't going away, many of today's problems require a configurable database system. Even without a crystal ball, it seems clear that tomorrow's systems will also require a significant degree of configurability. As programmers and engineers, we learn to select the right tool to do a job; selecting a database is no exception. We need to operate in a mode where we recognize that there *are* options in data management, and we should select the right tool to get the job done as efficiently, robustly, and simply as possible. 

#### References

1. Astrahan, M.M. System R: Relational approach to database management. *ACM Trans. Database Systems* 1, 2 (1976), 97-137.
2. Bernstein, P. The Asilomar Report on database research. *ACM SIGMOD Record* 27, 4 (1998); [www.sigmod.org/record/issues/9812/asilomar.html](http://www.sigmod.org/record/issues/9812/asilomar.html).
3. Broussard, F. Worldwide IT asset management software forecast and analysis, 2002-2007. (2004). IDC Doc. #30277; [www.idc.com/getdoc.jsp?containerId=30277&pid=35178981](http://www.idc.com/getdoc.jsp?containerId=30277&pid=35178981).
4. Chaudhuri, S., and Weikum, G. Rethinking database system architecture: Towards a self-tuning RISC-style database system. *The VLDB Journal*. (2000), 1-10; [www.vldb.org/conf/2000/P001.pdf](http://www.vldb.org/conf/2000/P001.pdf).
5. Codd, E.F. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970): 377-387.
6. Gray, J., and Reuter, A. *Transaction Processing: Concepts and Technologies*. Morgan Kaufman, San Mateo, CA, 1993, 397-402.
7. Stonebraker, M. The design and implementation of Ingres. *ACM Trans. Database Systems* 1, 3 (1976), 189-222.
8. Stonebraker, M., and Cetintemel, U. One size fits all: An idea whose time has come and gone. In *Proceedings of the 2005 International Conference on Data Engineering* (April 2005); [http://www.cs.brown.edu/~ugur/fits\\_all.pdf](http://www.cs.brown.edu/~ugur/fits_all.pdf).

**Margo I. Seltzer** ([margo@eecs.harvard.edu](mailto:margo@eecs.harvard.edu)) is the Herchel Smith Professor of Computer Science and a Harvard College Professor in the Division of Engineering and Applied Sciences at Harvard University, Cambridge, MA. She is also a founder and CTO of Sleepycat Software, the makers of Berkeley DB.

*A previous version of this article appeared in the April 2005 issue of ACM Queue, Vol. 3, No. 3.*

introducing...

# ACM's

## Newly Expanded Online Books & Courses Programs!

Helping Members Meet Today's Career Challenges

### 3,000 Online Courses from SkillSoft

The ACM Online Course Collection features **unlimited access to 3,000 online courses** from SkillSoft, a leading provider of e-learning solutions. This new collection of courses offers a host of valuable resources that will help to maximize your learning experience. Available on a wide range of information technology and business subjects, these courses are open to ACM Professional and Student Members.



SkillSoft courses offer a number of valuable features, including:

- **Job Aids**, tools and forms that complement and support course content
- **Skillbriefs**, condensed summaries of the instructional content of a course topic
- **Mentoring** via email, online chats, threaded discussions - 24/7
- **Exercises**, offering a thorough interactive practice session appropriate to the learning points covered previously in the course
- **Downloadable content** for easy and convenient access
- **Downloadable Certificate of Completion**

*"The course Certificate of Completion is great to attach to job applications!"*

ACM Professional Member

### 600 Online Books from Safari

The ACM Online Books Collection includes **unlimited access to 600 online books** from Safari® Books Online, featuring leading publishers including O'Reilly. Safari puts a complete IT and business e-reference library right on your desktop. Available to ACM Professional Members, Safari will help you zero in on exactly the information you need, right when you need it.



### 500 Online Books from Books24x7

All Professional and Student Members also have **unlimited access to 500 online books** from Books24x7®, in ACM's rotating collection of complete unabridged books on the hottest computing topics. This virtual library puts information at your fingertips. Search, bookmark, or read cover-to-cover. Your bookshelf allows for quick retrieval and bookmarks let you easily return to specific places in a book.



Association for  
Computing Machinery

*Advancing Computing as a Science & Profession*

[pd.acm.org](http://pd.acm.org)  
[www.acm.org/join](http://www.acm.org/join)



DOI: 10.1145/1364782.1364798

**The Web must be studied as an entity in its own right to ensure it keeps flourishing and prevent unanticipated social effects.**

BY JAMES HENDLER, NIGEL SHADBOLT, WENDY HALL,  
TIM BERNERS-LEE, AND DANIEL WEITZNER

# Web Science: An Interdisciplinary Approach to Understanding the Web

DESPITE THE WEB'S great success as a technology and the significant amount of computing infrastructure on which it is built, it remains, as an entity, surprisingly unstudied. Here, we look at some of the technical and social challenges that must be overcome to model the Web as a whole, keep it growing, and understand its continuing social impact. A systems approach, in the sense of "systems biology," is needed if we are to be able to understand and engineer the future Web.

ILLUSTRATION BY MARTIUS WATZ

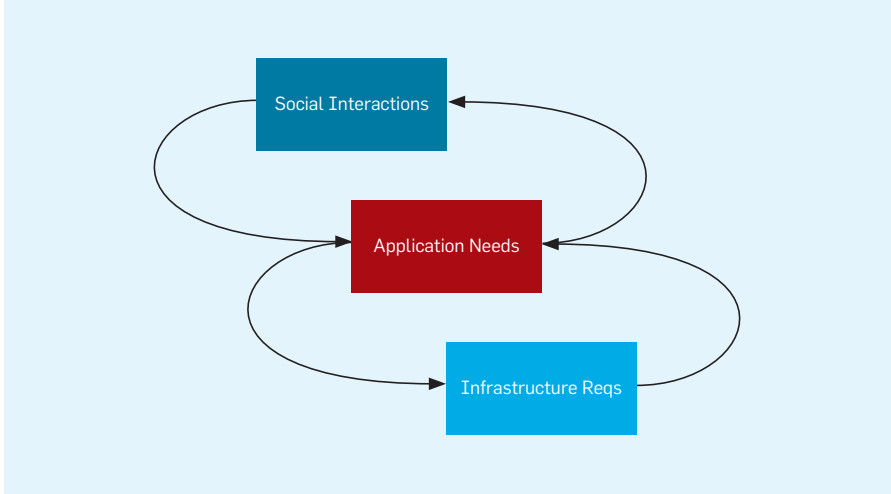








**Figure 1: The social interactions enabled by the Web put demands on the Web applications behind them, in turn putting further demands on the Web's infrastructure.**



Despite the huge effect the Web has had on computing, as well as on the overall field of computer science, the best keyword indicator one can find in the ACM taxonomy, the one by which the field organizes many of its research papers and conferences, is “miscellaneous.” Similarly, if you look at CS curricula in most universities worldwide you will find “Web design” is taught as a service course, along with, perhaps, a course on Web scripting languages. You are unlikely to find a course that teaches Web architecture or protocols. It is as if the Web, at least below the browser, simply does not exist. Many “information schools” and “informatics departments” offer courses that focus on applications on the Web or on such topics as “Web 2.0,” but the protocols, architectures, and underlying principles of the Web per se are rarely covered.

Simplifying a bit, part of the reason for this is that networking has long been part of the systems curricula in many departments, and thus the Internet, defined via the TCP/IP networking protocols, has long been considered an important part of CS work. The Web, despite having its own protocols, algorithms, and architectural principles, is often viewed by people in the CS field as an application running on top of the Net, more than as an entity unto itself.

This is odd, as the Web is the most used and one of the most transformative applications in the history of computing, even of human communications. It has changed how those in

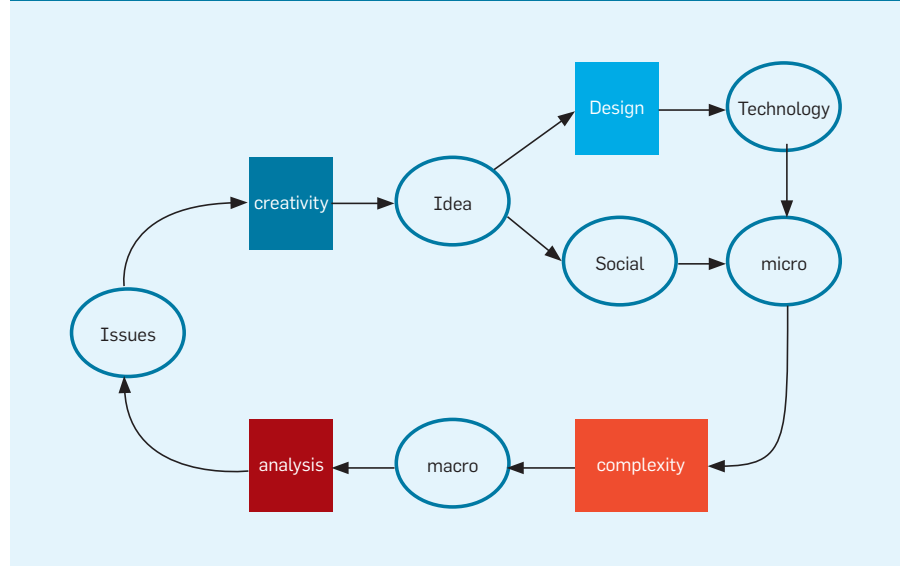
academia teach, communicate, publish, and do research. In industry, it has not only created an entire sector (or, arguably, multiple sectors) but affected the communications and delivery of services across the entire industrial spectrum. In government, it has changed not only the nature of how governments communicate with their citizens but also how these populations communicate and even, in some cases, how they end up choosing their governments in the first place; recall the U.S. presidential debates in which candidates took questions online and through YouTube videos. It is estimated that the size of the human population is on the order of  $10^{10}$  people,

whereas the number of separate Web documents is more than  $10^{11}$ .

Computing has made significant contributions to the Web. Our everyday use of the Web depends on fundamental developments in CS that took place long before the Web was invented. Today’s search engines are based on, for example, developments in information retrieval with a legacy going back to the 1960s. The innovations of the 1990s<sup>9,23</sup> provide the crucial algorithms underlying modern search and are fundamental to Web use. New resources (such as Hadoop, [lucene.apache.org/hadoop/](http://lucene.apache.org/hadoop/), an open-source software framework that supports data-intensive distributed applications on large clusters of commodity computers) make it possible for students to explore these algorithms and experiment with large-scale Web-programming practices like MapReduce parallelism<sup>11</sup> in a way not previously accessible beyond a few top universities.

Other aspects of human interaction on the Web have been studied elsewhere. Of special note, many interesting aspects of the use of the Web (such as social networking, tagging, data integration, information retrieval, and Web ontologies) have become part of a new “social computing” area at some of the top information schools. They offer classes in the general properties of networks and interconnected systems in both the policy and political aspects of computing and in the economics

**Figure 2: The Web presents new challenges to software engineering and application development.**






of computer use. However, in many of these courses, the Web itself is treated as a specific instantiation of more general principals. In other cases, the Web is treated primarily as a dynamic content mechanism that supports the social interactions among multiple browser users. Whether in CS studies or in information-school courses, the Web is often studied exclusively as the delivery vehicle for content, technical or social, rather than as an object of study in its own right.

Here, we present the emerging interdisciplinary field of Web science<sup>5</sup>,<sup>6</sup> taking the Web as its primary object of study. We show there is significant interplay among the social interactions enabled by the Web's design, the scalable and open applications development mandated to support them, and the architectural and data requirements of these large-scale applications (see Figure 1). However, the study of the relationships among these levels is often hampered by the disciplinary boundaries that tend to separate the study of the underlying networking from the study of the social applications. We identify some of these relationships and briefly review the status of Web-related research within computing. We primarily focus on identifying emerging and extremely challenging problems researchers (in their role as Web scientists) need to explore.


### What Is It?

Where physical science is commonly regarded as an analytic discipline that aims to find laws that generate or explain observed phenomena, CS is predominantly (though not exclusively) synthetic, in that formalisms and algorithms are created in order to support specific desired behaviors. Web science deliberately seeks to merge these two paradigms. The Web needs to be studied and understood as a phenomenon but also as something to be engineered for future growth and capabilities.

At the micro scale, the Web is an infrastructure of artificial languages and protocols; it is a piece of engineering. However, it is the interaction of human beings creating, linking, and consuming information that generates the Web's behavior as emergent properties at the macro scale. These properties often generate surprising proper-



**A large-scale system may have emergent properties not predictable by analyzing micro technical and/or social effects.**



ties that require new analytic methods to be understood. Some are desirable and therefore to be engineered in; others are undesirable and if possible engineered out. We also need to keep in mind that the Web is part of a wider system of human interaction; it has profoundly affected society, with each emerging wave creating new challenges and opportunities in making information more available to wider sectors of the population than ever before.

It may seem that the best way to understand the Web is as a set of protocols that can be studied for their properties, with individual applications analyzed for their algorithmic properties. However, the Web wasn't (and still isn't) built using the specify, design, build, test development cycle CS has traditionally viewed as software engineering best practice.

Figure 2 outlines a new way of looking at Web development. A software application is designed based on an appropriate technology (such as algorithm and design) and with an envisioned "social" construct; it is indeed a contradiction in terms to talk about a Web application built for a single user on a single machine. The system is generally tested in a small group or deployed on a limited basis; the system's "micro" properties are thus tested. In some cases, when more and more people accept the micro system, accelerating "viral" scaling occurs. For example, when Mosaic, the first popular Web browser, was released publicly in 1992, the number of users quickly grew by several orders of magnitude, with more than a million downloads in the first year; for more recent examples, consider photo-sharing on Flickr, video-uploading on YouTube, and social-networking sites like mySpace and Facebook.

The macro system, that is, the use of the micro system by many users interacting with one another in often-unpredicted ways, is far more interesting in and of itself and generally must be analyzed in ways that are different from the micro system. Also, these macro systems engender new challenges that do not occur at the micro scale; for example, the wide deployment of Mosaic led to a need for a way to find relevant material on the growing Web, and thus search became an important applica-

tion, and later an industry, in its own right. In other cases, the large-scale system may have emergent properties that were not predictable by analyzing the micro technical and/or social effects. Dealing with these issues can lead to subsequent generations of technology. For example, the enormous success of search engines has inevitably yielded techniques to game the algorithms (an unexpected result) to improve search rank, leading, in turn, to the development of better search technologies to defeat the gaming.

The essence of our understanding of what succeeds on the Web and how to develop better Web applications is that we must create new ways to understand how to design systems to produce the effect we want. The best we can do today is design and build in the micro, hoping for the best, but how do we know if we've built in the right functionality to ensure the desired macroscale effects? How do we predict other side effects and the emergent properties of the macro? Further, as the success or failure of a particular Web technology may involve aspects of social interaction among users, a topic we return to later, understanding the Web requires more than a simple analysis of technological issues but also of the social dynamic of perhaps millions of users.

Given the breadth of the Web and its inherently multi-user (social) nature, its science is necessarily interdisciplinary, involving at least mathematics, CS, artificial intelligence, sociology, psychology, biology, and economics. We invite computer scientists to expand the discipline by addressing the challenges following from the widespread adoption of the Web and its profound influence on social structures, political systems, commercial organizations, and educational institutions.

### Beneath the Web Graph

One way to understand the Web, familiar to many in CS, is as a graph whose nodes are Web pages (defined as static HTML documents) and whose edges are the hypertext links among these nodes. This was named the “Web graph” in <sup>22</sup>, which also included the first related analysis. The in-degree of the Web graph was shown in Kleinberg et al.<sup>3</sup> and Kumar et al.<sup>24</sup> to follow a power-law distribution; a similar effect

was shown in Broder et al.<sup>10</sup> for the out-branching of vertices in the graph. An important result in Dill et al.<sup>12</sup> showed that large samples of the Web, generated through a variety of methods, all had similar properties—important as the Web graph grows, reported in 2005 to be on the order of seven million new pages a day.<sup>17</sup> Various models have been proposed as to how the Web graph grows and which models best capture its evolution; see Donato et al.<sup>14</sup> for an analysis of a number of these models and their properties.

Along with analyses of this graph and its growth, a number of algorithms have been devised to exploit various properties of the graph. For example, the HITS algorithm<sup>23</sup> and PageRank<sup>9</sup> assume that the insertion of a hyperlink from one page to another can be taken as a sort of endorsement of the “authority” of the page being linked to, an assumption that led to the development of powerful search engines for finding pages on the Web. While modern search engines use a number of heuristics beyond these page-authority calculations, due in part to competitive pressure from those trying to spoof the algorithms and get a higher rank, these Web-graph-based models still form the heart of the critical crawlers and rank-assessment algorithms behind Web search.

The links in this Web graph represent single instantiations of the results of calling the HTTP protocol with a GET request that returns a particular representation (in this case an HTML page) of a document based on a universal resource identifier (URI) that serves as an identifier common across the entire Web. So, for example, the URI <http://www.acm.org/publications/cacm> typed into a standard Web browser invokes the hypertext transfer protocol (HTTP) and returns an HTML page that contains content describing the publication known as *Communications of the ACM*. Note, however, that the content itself contains other URIs that are themselves pointers to objects that are also displayed (such as icons and images) and that the formatting of the page itself may require retrieving other resources (such as cascaded style sheets) or XML DTD documents. So what we might naively view as a single link from, say, a research group's Web

page to an article on a *Communications* page will actually involve a number of requests among a number of servers; at the time of this writing, typing the URI for *Communications* into a browser will cause more than 20 different HTTP-GET requests to occur for seven different types of Web formats. Crawlers can capture these links and create the Web graph as, essentially, a static snapshot of the linking of the Web.

However, the Web graph is just one abstraction of the Web based on one part of the processing and protocols underlying its function. While it is an important result that the Web graph is scale-free, it is the design of the protocols and services that we now call the Web that makes it possible for it to be this way. The Web was built around a set of core design components defined in *The Architecture of The World Wide Web, Volume 1*<sup>21</sup> as “the identification of resources, the representation of resource state, and the protocols that support the interaction between agents and resources in the space.”


A feature of the Web is that, depending on the details of a request, different representations may be served up to different requesters. For example, the HTML produced may vary based on conditions hidden from the client (such as which particular machines in a back-end server farm process the request) and by the server's customization of the response. Cookies, representing previous state, may also be used, causing different users to see different content (and thus have different links in the Web graph) based on earlier behavior and visits to the same or to other sites. This sort of user-dependent state is not directly accounted for in current Web-graph models.

There are also other ways the Web, as an application of the Internet, cannot simply be analyzed using the model of a quasi-static graph of linked hypertext pages. For example, many Web sites use Web forms to access a wealth of information behind the servers, where that information, sometimes called “the deep Web,” is not visible in the Web model. For many sites, in which the applications's data forms a linked Web, the links are not explicit, and HTTP-POST requests are used instead of the HTTP-GETs in the Web graph. In other cases, these sites generate com-


plex URIs that use GET requests to pass on state<sup>a</sup>, thus obscuring the identity of the actual resources.

URIs that carry state are used heavily in Web applications but are, to date, largely unanalyzed. For example, in a June 2007 talk, Udi Manber, Google's VP of engineering, addressed the issue of why Web search is so difficult,<sup>25</sup> explaining that on an average day, 20%–25% of the searches seen by Google have never been submitted before and that each of these searches generates a unique identifier (using server-specific encoding information). So a Web-graph model would represent only the requesting document (whether a user request or a request generated by, for example, a dynamic advertisement content request) linked to the `www.google.com` node. However if, as is widely reported, Google receives more than 100 million queries per day, and if 20% of them are unique, then more than 20 million links, represented as new URIs that encode the search term(s), should show up in the Web graph every day, or around 200 per second. Do these links follow the same power laws? Do the same growth models explain these behaviors? We simply don't know.

Analyzing the Web solely as a graph also ignores many of its dynamics (especially at short timescales). Many phenomena known to Web users (such as denial-of-service attacks caused by flooding a server and the need to click the same link multiple times before getting a response) cannot be explained by the Web-graph model and often can't be expressed in terms amenable to such graph-based analysis. Representing them at the networking level, ignoring protocols and how they work, also misses key aspects of the Web, as well as a number of behaviors that emerge from the interactions of millions of requests hitting many thousands of servers every second. Web dynamics were analyzed more than a decade ago,<sup>20</sup> but the combination of (i) the exponential growth in the amount of Web content, (ii) the change in the number, power, and diversity of Web servers and appli-



**Today's interactive applications are very early social machines, limited by the fact that they are largely isolated one from another.**



cations, and (iii) the increasing number of diverse users from everywhere in the world makes a similar analysis impossible today without creating and validating new models of the Web's dynamics. Such models must also pay special attention to the details of the Web's architecture, as well as to the complexity of the interactions actually taking place there.

Additionally, modern, sophisticated Web sites provide powerful user-interface functionality by running large script systems within the browser. These applications access the underlying remote data model through Web APIs. This application architecture allows users and entrepreneurs to quickly build many new forms of global systems using the processing power of users' machines and the storage capacity of a mass of conventional Web servers. Like the basic Web, each such system is interesting mainly for its emergent macro-scale properties, of which we have little understanding. Are such systems stable? Are they fair? Do they effectively create a new form of currency? And if they do should it be regulated?

Similarly, many user-generated content sites now store personal information yet have rather simplistic systems to restrict access to a person's "friends." This information is not available to wide-scale analysis. Some other sites must be allowed to access the sites by posing as the user or as a friend; a number of three-party authentication protocols are being deployed to allow this. A complex system is thus being built piece by piece, with no invariants (such as "my employer will never see this picture") assured for the user.

The purpose of this discussion is not to go into the detail of Web protocols or the relative merits of Web-modeling approaches but to stress that they are critical to the current and continued working of the Web. Understanding the protocols and issues is important to understanding the Web as a technical construct and to analyzing and modeling its dynamic nature. Our ability to engineer Web systems with desirable properties at scale requires that we understand these dynamics. This analysis and modeling are thus an important challenge to computer scientists if they are to be able to understand

a. These characters, including `?.# =, and &`, followed by keywords, may follow the last "slash" in the URI, thus making for the long URIs often generated by dynamic content servers.



the growth and behaviors of the future Web, as well as to engineer systems with desired properties in a way that is significantly less hit or miss.

### From Power Laws to People

Mathematically based analysis of the Web involves another potential failing. Whereas the structure and use of various Web sites (taken mathematically) may have interesting properties, these properties may not be very useful in explaining the behavior of the sites over time. Consider the following example: Wikipedia ([www.wikipedia.org](http://www.wikipedia.org)), the

the linguistic content of its pages. The figure shows the same kind of Zipf-like distribution found in the original Web graph analyses. There is also some evidence<sup>16</sup> and a lot of speculation<sup>29</sup> that similar effects can be seen in the use of tags in Web-based tagging systems. Current research is also exploring whether these results depart from such models as preferential attachment<sup>3</sup> used to explain the scale-free features of Web graphs.

Unfortunately, whatever explains these effects, another aspect of Wikipedia's use is not explained by these

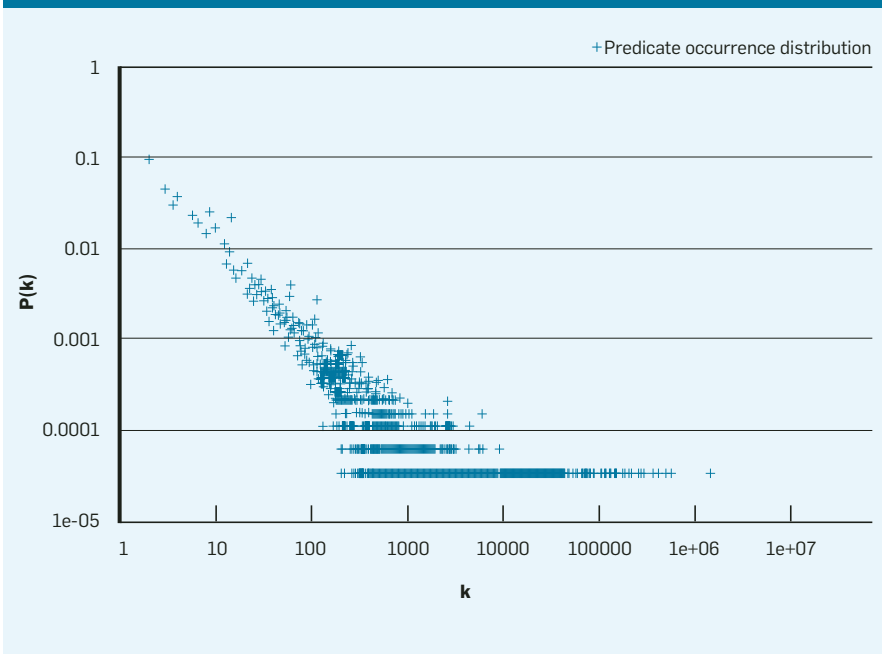
in ways allowed by that technology is more difficult to explain. The dynamics of any "social machine" are highly complex, and dozens of academic papers, from multiple disciplines, have been written about it; [en.wikipedia.org/wiki/Wikipedia:Wikipedia\\_in\\_academic\\_studies](http://en.wikipedia.org/wiki/Wikipedia:Wikipedia_in_academic_studies) uses Wikipedia itself to maintain an up-to-date reference list.

The idea of a social machine was introduced in *Weaving the Web*,<sup>8</sup> which hypothesized that the architectural design of the Web would allow developers, and thus end users, to use computer technology to help provide the management function for social systems as they were realized online. The social machine includes the underlying technology (mediaWiki in the case of Wikipedia) but also the rules, policies, and organizational structures used to manage the technology. Examples abound on the Web today. Consider the coupling of the application design of blogging-support systems (such as LiveJournal and WordPress) with the social mechanisms provided by blogrolls, permalinks, and trackbacks that have led to the so-called blogosphere. Similarly, the protocols used by social networking sites like MySpace and Facebook have much in common, but the success or failure of the sites hinges on the rules, policies, and user communities they support. Given that the success or failure of Web technologies often seems to rely on these social features, the ability to engineer successful applications requires a better understanding of the features and functions of the social aspects of the systems.<sup>b</sup>

Today's interactive applications are very early social machines, limited by the fact that they are largely isolated from one another. We hypothesize that (i) there are forms of social machine that will someday be significantly more effective than those we have today; (ii) that different social processes interlink in society and therefore must be interlinked on the Web; and (iii) that they are unlikely to be developed through a single deliberate effort in a single proj-

b. When we say "success" or "failure," we are referring not to the business factors that determine whether, for example, Facebook or MySpace will attract more users but to the success or failure of the sites to provide the particular types of social interaction for which they are designed.

**Figure 3: Results of an analysis of the link structure of Wikipedia with respect to the use of link labels, not the linguistic content of pages.**



online wiki-based encyclopedia, includes more than two million articles in English and more than six million in all languages combined. They are hyperlinked, and it is logical to ask whether the hyperlinks have structure similar to those on the Web in general or whether, since this is a managed corpus, they have yet other properties.

Answering can be done in a number of ways; Figure 3 shows the result of one of them. In this case, DBpedia ([dbpedia.org](http://dbpedia.org)), which is a dump of the link structure of Wikipedia using the labeled links of the resource description framework, or RDF, has been analyzed with respect to the use of the link labels; that is, we are looking at the structure of Wikipedia as opposed to

models and does not necessarily follow from these properties. Wikipedia is built on top of the MediaWiki software package ([www.mediawiki.org/wiki/MediaWiki](http://www.mediawiki.org/wiki/MediaWiki)), which is freely available and used in many other Web applications besides Wikipedia. While some of them have also been successful, many have failed to generate significant use. A purely "technological" explanation cannot account for this; rather, something about the organizational structures of Wikipedia and the needs of its users accounts for its success over other systems built from the same code base. The model by which articles are created, edited, and tracked is provided by the underlying technology. The social model enabled by humans interacting

ect or site; rather, technology is needed to allow user communities to construct, share, and adapt social machines so successful models evolve through trial, use, and refinement.

A number of research challenges and questions must be resolved before a new generation of interacting social machines can be created and evolved this way:


- ▶ What are the fundamental theoretical properties of social machines, and what kinds of algorithms are needed to create them?;

- ▶ What underlying architectural principles are needed to guide the design and efficient engineering of new Web infrastructure components for this social software?;


- ▶ How can we extend the current Web infrastructure to provide mechanisms that make the social properties of information-sharing explicit and guarantee that the use of this information conforms to relevant social-policy expectations?; and

- ▶ How do cultural differences affect the development and use of social mechanisms on the Web? As the Web is indeed worldwide, the properties desired by one culture may be seen as counterproductive by others. Can Web infrastructure help bridge cultural divides and/or increase cross-cultural understanding?

In addition, a crucial aspect of human interaction with information is our ability to represent and reason over such attributes as trustworthiness, reliability, and tacit expectations about the use of information, as well as about privacy, copyright, and other legal rules. While some of this information is available on the Web today, we lack structures for formally representing and computing over them. Traditional cryptographic security research and well-known access-control-policy frameworks have failed to meet these challenges in today's online environment and are thus insufficient as a foundation for the social machines of the future. Recent work on formal models for privacy<sup>b</sup> has demonstrated that traditional cryptographic approaches to privacy protection can fail in open Web environments. Similar problems with copyright enforcement have also hampered the flow of commercial and scholarly information on the



**The Web is changing at a rate that may be greater than even the most knowledgeable researcher's ability to observe it.**



Web.<sup>27</sup> To this end, an exemplar Web science research area we are pursuing involves interdisciplinary research toward augmenting Web architecture with technical and social conventions that increase individual accountability to social and legal rules governing information use.<sup>31</sup> Continued failure to develop scalable models for handling policy will impede the ability of the Web to be the best possible medium for exchanging cultural, scientific, and political information.

Further, we can see from the dramatic growth of new collaborative styles of creating and publishing information on the Web that many of the social institutions we rely on to judge trustworthiness and veracity are missing from our online information life. Being able to engineer the Web of the future requires not only understanding it as a computational structure but also how it interacts with and supports interaction among its users.

An important aspect of research exploring the influence of the Web on society involves online societies using Web infrastructure to support dynamic human interaction. This work—seen in [trout.cpsr.org](http://trout.cpsr.org) and other such efforts—explores how the Web can encourage more human engagement in the political sphere. Combining it with the emerging study of the Web and the coevolution of technology and social needs is an important focus of designing the future Web.<sup>30</sup>

### **The Web of Data**

This emerging area of study involves the heavy use of tagging provided by many of what are known as Web 2.0 technologies. Articles, blogs, photos, videos, and all manner of other Web resources may be annotated with user-generated keywords, or tags, that can later be used for searching or browsing these resources. Much has been made of how “folksonomies,” or taxonomies that emerge through the use of tags, can be used as metadata to help explain the content of the objects being described.

One aspect of tagging generating interest today is the need for “social context” in tagging.<sup>26</sup> Many tags involve terms that are extremely ambiguous in a general context. For example, first names are popular tags on Flickr,

though they are not good general search terms. On the other hand, in a specific social context (such as a particular person's photos), the same tag can be useful since it can designate a particular individual. The use of a tag as metadata often depends on such a context, and the "network effect" in these cites is thus socially organized.<sup>19</sup>

A more ambitious use of metadata involves recent applications of semantic Web technologies<sup>7</sup> and represents an important paradigm shift that is a significant element of emerging Web technologies. The semantic Web represents a new level of abstraction from the underlying network infrastructure, as the Internet and Web did earlier. The Internet allowed programmers to create programs that could communicate without concern for the network of cables through which the communication had to flow. The Web allows programmers and users to work with a set of interconnected documents without concern for the details of the computers storing and exchanging them.

The semantic Web will allow programmers and users alike to refer to real-world objects—people, chemicals, agreements, stars, whatever—without concern for the underlying documents in which these things, abstract and concrete, are described. While basic semantic Web technologies have been defined and are being deployed more widely, little work has sought to explain the effect of these new capabilities on the connections within the Web of people who use them.<sup>28</sup>

The semantic Web arena reflects two principle nexuses of activity. One tends to involve data (and the Web), and the other on the domain (and semantics). The first, based largely on innovation in data-integration applications, focuses on developing Web applications that employ only limited semantics but provide a powerful mechanism for linking data entities using the URIs that are the basis of the Web. Powered by the RDF, these applications focus largely on querying graph-oriented triple-store databases using the emerging SPARQL language, which helps create Web applications and portals that use REST-based models, integrating data from multiple sources without preexisting schema. The second, based largely on the Web Ontology Language, or OWL,

looks to provide models that can be used to represent expressive semantic descriptions of application domains and provide inferencing power for both Web and non-Web applications that need a knowledge base.

Current research is exploring how the databases of the semantic Web relate to traditional database approaches and to scaling semantic Web stores to very large scales.<sup>1</sup> In terms of modeling, one goal is to develop tools to speed inference in large knowledge bases (without sacrificing performance), including how to exploit trade-offs between expressivity and reasoning to provide the capabilities needed for Web scale.<sup>15</sup> A market is beginning to emerge for "bottom-up" tools driven by data and "top-down" technologies driven by Web ontologies. Creating back-ends for the semantic Web is being transitioned (bottom-up) from an arcane art into an emerging Web application programming approach, as new open-source technologies integrate well with traditional Web servers. At the same time, new tools support ontology development and deployment (top-down), and tens of thousands of OWL ontologies are available for jumpstarting new domain-modeling efforts. In addition, approaches using rule-based reasoning modified for the Web have also gained attention.<sup>4</sup> Engineering the future Web includes the design and use of these emerging technologies, along with how they differ from traditional approaches to databases, in one case creating back-ends for the semantic Web, in the other new tools for ontology-based applications.

The semantic Web is a key emerging technology on the Web, but, also, as we've discussed, there are different opinions as to what it is best for and, more important, what the macro effects might be. Our lack of a better understanding of how Web systems develop makes it difficult for us to know the kinds of effects the technology will produce at scale. What social consequences might there be from greater public exposure and the sharing of information hidden away in databases? A better understanding of how Web systems move from the micro to the macro scale would provide a better understanding of how they could be

developed and what their potential societal effects might be.

## Conclusion

The Web is different from most previously studied systems in that it is changing at a rate that may be of the same order as, or perhaps greater than, even the most knowledgeable researcher's ability to observe it. An unavoidable fact is that the future of human society is now inextricably linked to the future of the Web. We therefore have a duty to ensure that future Web development makes the world a better place. Corporations have a responsibility to ensure that the products and services they develop on the Web don't produce side effects that harm society, and governments and regulators have a responsibility to understand and anticipate the consequences of the laws and policies they enact and enforce.

We cannot achieve these aims until we better understand the complex, cross-disciplinary dynamics driving development on the Web—the main aim of Web science. Just as climate-change scientists have had to develop ways to gather and analyze evidence to prove or disprove theories about the effect of human behavior on the Earth's climate, Web scientists need new methodologies for gathering evidence and finding ways to anticipate how human behavior will affect development of a system that is evolving at such an amazing rate. We also must consider what would happen to society if access to the Web was denied to some or all and to raise awareness among major corporations and governments that the consequences of what appear to be relatively small decisions can profoundly affect society in the future by affecting Web development today.

Computing plays a crucial role in the Web science vision, and much of what we know about the Web today is based on our understanding of it in a computational way. However, as we've explored here, significant research must still be done to be able to engineer future successful Web applications. We must understand the Web as a dynamic and changing entity, exploring the emergent behaviors that arise from the "macro"



interactions of people enabled by the Web's technology base. We must therefore understand the "social machines" that may be the critical difference between the success or failure of Web applications and learn to build them in a way that allows inter-linking and sharing.

**Acknowledgments**

Figure 2 is taken from talks Tim Berners-Lee gave in 2007 ([www.w3.org/2007/Talks/1018-websci-mit-tbl/Overview.html](http://www.w3.org/2007/Talks/1018-websci-mit-tbl/Overview.html)). We also thank the other members of the WSRI Scientific Council ([webscience.org/about/people/](http://webscience.org/about/people/)) for input relating to the goals of Web science and the interaction of the Web and computer and information sciences. We are indebted to Konstantin Mertsalov of Rensselaer Polytechnic Institute for the DBpedia analysis discussed in the section on power laws. ■

**References**

1. Abadi, D., Marcus, A., Madden, S., and Hollenbach, K. Scalable semantic Web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria, Sept. 23–27). VLDB Endowment, Heidelberg, 2007.
2. Backstrom, L., Dwork, C., and Kleinberg, J. Wherefore art thou R3579X? Anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th International World Wide Web Conference* (Banff, Alberta, Canada, May 8–12). ACM Press, New York, 2007.
3. Barabasi, A. and Albert, A. Emergence of scaling in random networks. *Science* 286 (1999).
4. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., and Hendler, J. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming* (2008).
5. Berners-Lee, T., Hall, W., Hendler, J., Shadbolt, N., and Wietzner, D. Creating a science of the Web. *Science* 311 (2006).
6. Berners-Lee, T., Hall, W., Hendler, J., O'Hara, K., Shadbolt, N., and Weitzner, D. A framework for Web science. *Foundations and Trends in Web Science* 1, 1 (Sept. 2006).
7. Berners-Lee, T., Hendler, J., and Lassila, O. The semantic Web. *Scientific American* (May 2001).
8. Berners-Lee, T. and Fischetti, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper Collins, New York, 1999.
9. Brin, S. and Page, L. The anatomy of large-scale hypertextual Web search engine. Presented at the Sixth International World Wide Web Conference (Santa Clara, CA, Apr. 7–11, 1997).
10. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. Graph structure in the Web. In *Proceedings of the Ninth International World Wide Web Conference* (Amsterdam, The Netherlands, May 15–19). Elsevier, Amsterdam, The Netherlands, 2000.
11. Dean, J. and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation* (San Francisco, Dec. 6–8). USENIX Association, Berkeley, CA, 2004.
12. Dill, S., Kumar, R., McCurley, K., Rajagopalan, S., Sivakumar, D. and Tomkins, A. Self-similarity in the Web. In *Proceedings of the 27th International Conference on Very Large Data Bases* (Rome, Italy, Sept. 11–14). Morgan Kaufmann Publishers, Inc., San Francisco, 2001.
13. Domingos, P., Golbeck, J., Mika, P., and Nowak, A. Social networks and intelligent systems. *IEEE Intelligent Systems, Trends & Controversies* 20, 1 (Jan./Feb. 2005).

14. Donato, D., Laura, L., Leonardi, S., and Millozzi, S. The Web as a graph: How far we are. *ACM Transactions on Internet Technology* 7, 1 (Feb. 2007).
15. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., and Srinivas, K. The Summary Abox: Cutting ontologies down to size. In *Proceedings of the International Semantic Web Conference* (Athens, GA, Nov. 5–9). Springer Berlin, Heidelberg, 2006.
16. Golder, S. and Huberman, B. *The Structure of Collaborative Tagging Systems* (2005); [arxiv.org/abs/cs/0508082](http://arxiv.org/abs/cs/0508082).
17. Gulli, A. and Signorini, A. The indexable Web is more than 11.5 billion pages. In the special-interest tracks and posters of the 14th International World Wide Web Conference (Chiba, Japan, May 10–14). ACM Press, New York, 2005.
18. Hendler, J. Web 3.0: Semantic Web chicken farms. *IEEE Computer* 41, 1 (Jan. 2008).
19. Hendler, J. and Golbeck, J. Metcalfe's Law, Web 2.0, and the semantic Web. *Journal of Web Semantics* 6, 1 (Feb. 2008).
20. Huberman, B. and Lukose, R. Social dilemmas and Internet congestion. *Science* 277, 5325 (July 1997).
21. Jacobs, I. and Walsh, N. *Architecture of the World Wide Web, Vol. One*. W3C Recommendation, Dec. 15, 2004; [www.w3.org/TR/webarch/](http://www.w3.org/TR/webarch/).
22. Kleinberg, J., Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. The Web as a graph: Measurements, models, and methods. In *Proceedings of the Fifth Annual International Conference on Computing and Combinatorics* (Tokyo, July 26–28). Springer, New York, 1999.
23. Kleinberg, J. Authoritative sources in a hyperlinked environment. *Journal of the ACM* 46, 5 (Sept. 1997).
24. Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. Trawling the Web for emerging cyber communities. In *Proceedings of the Eighth International World Wide Web Conference* (Toronto, May 11–14). Elsevier North-Holland, Inc., New York, 1999.
25. Manber, U. *Why Search Is a Hard Problem*. Presentation at Supernova 2007 (San Francisco, June 16–18, 2008); [www.readwriteweb.com/archives/udi\\_manber\\_search\\_is\\_a\\_hard\\_problem.php](http://www.readwriteweb.com/archives/udi_manber_search_is_a_hard_problem.php)
26. Marcus, A. and Perez, A. m-YouTube mobile UI: Video selection based on social influence. In *Proceedings of the 12th International HCI Conference* (Beijing, July 22–27). Springer, 2007.
27. Samuelson, P. Copyright's fair use doctrine and digital data. *Commun. ACM* 37, 1 (Jan.1994), 21–27.
28. Shadbolt, N., Hall, W., and Berners-Lee, T. The semantic Web revisited. *IEEE Intelligent Systems* 21, 3 (May/June 2006).
29. Shirky, C. *Power Laws, Weblogs, and Inequality*. In Clay Shirky's blog (2003); [www.shirky.com/writings/powerlaw\\_weblog.html](http://www.shirky.com/writings/powerlaw_weblog.html).
30. Shneiderman, B. Web science: A provocative invitation to computer science. *Commun. ACM* 50, 6 (June 2007), 25–27.

31. Weitzner, D., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., and Sussman, G. Information accountability. *Commun. ACM* 51, 6 (June 2008).
32. Weitzner, D., Hendler, J., Berners-Lee, T., and Connolly, D. Creating a policy-aware Web: Discretionary, rule-based access for the World Wide Web. In *Web and Information Security*, E. Ferrari and B. Thuraisingham, Eds. IRM Press, Hershey, PA, 2006.

Funding for this work comes from the U.S. National Science Foundation (Policy Aware Web and Transparency Aware Data Mining Projects), iARPA (End-to-End Semantic Accountability), the U.K. Engineering and Physical Sciences Research Council (Advanced Knowledge Technologies Project), and the U.S. Army Research Laboratory and U.K. Ministry of Defence (U.S./U.K. Information Technology Alliance). We also thank industrial and individual donors to the authors' research at RPI, Southampton, and MIT and to the Web Science Research Initiative ([www.webscience.org](http://www.webscience.org)).

**James Hendler** ([hendler@cs.rpi.edu](mailto:hendler@cs.rpi.edu)) is the Tetherless World Chair of Computer and Cognitive Science at Rensselaer Polytechnic Institute, Troy, NY.

**Nigel Shadbolt** ([nrs@ecs.soton.ac.uk](mailto:nrs@ecs.soton.ac.uk)) is professor of artificial intelligence and deputy head of the School of Electronics and Computer Science at Southampton University, Southampton, U.K.

**Wendy Hall** ([wh@ecs.soton.ac.uk](mailto:wh@ecs.soton.ac.uk)) is a professor of computer science at the University of Southampton, Southampton, U.K.

**Tim Berners-Lee** ([timbl@csail.mit.edu](mailto:timbl@csail.mit.edu)) is the Director of the World Wide Web Consortium and holds the 3Com Founders chair and is a senior research scientist in the Laboratory for Computer Science and Artificial Intelligence at the Massachusetts Institute of Technology, Cambridge, MA.

**Daniel Weitzner** ([djweitzner@csail.mit.edu](mailto:djweitzner@csail.mit.edu)) is director of the Massachusetts Institute of Technology Decentralized Information Group and principle research scientist in the MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.



ILLUSTRATION BY MARIUS WATZ

DOI: 10.1145/1364782.1364799

**How changes in computer architecture are about to impact everyone in the IT business.**

BY MARK OSKIN

# The Revolution Inside the Box

COMPUTER ARCHITECTURE research is undergoing a renewed vitality. No longer is the road ahead clear for microprocessors. Indeed, a decade ago the road seemed straightforward: deeper pipelines, more complex microprocessors, and little change to the core instruction set architecture. No longer. For a variety of technological reasons, manufacturers have embraced multicore CPUs for the mainstream of desktop computing. Such a change represents the biggest single risk these vendors have taken in decades, as they are now expecting software developers to embrace a programming model they have been reluctant to target in the past.

In this article I look back on computer architecture research over the past 10 years, including what accounted for this change and what will happen because of it. In addition, I survey the field of computer architecture research, looking at what types of problems we once thought were important to explore and how those problems are exacerbated or mitigated in the future.

Seven years ago, when I started as a young assistant professor, my computer science colleagues felt computer architecture was a solved problem. Words like “incremental” and “narrow” were often used to describe research under way in the field. In some ways, who could blame them? To a software developer, the hardware/software interface—the very core of the computer architecture research field—had remained unchanged for most of their professional lifetimes. Even the key microarchitectural innovations (pipelining, branch prediction, caching, and others) appeared to be created long ago. From the perspective of the rest of computer science, architecture was a solved problem. This perception of computer architecture research had some very real consequences. NSF folded the computer architecture (CSA) program together with a grab bag of areas from VLSI to graphics into an omnibus “computing processes and artifacts” cluster. Large-scale DARPA programs to fund innovative architecture research in academia have recently wound down.

Around 2000, I would also characterize the collective mood of researchers in computer architecture as overly self-critical and bored of examining certain core topics in the field. The outside perspective of computer architecture had become the inside one. We would bemoan our field, nicknaming our premier technical conference as the “International Symposium on Cache Architecture,” instead of its true title “Computer Architecture.” We amusingly called our own innovations “yet another”<sup>12</sup> take on an old problem.

ILLUSTRATION BY GARY NEILL







**Part I: The End of Innocence**

In 2000, the roadmap ahead for desktop processing seemed clear to many. Processors with ever deeper pipelines and faster clock frequencies would scale performance into the future.<sup>1,7</sup> Researchers, myself among them, focused on the consequences of this, such as the wire delay problem. It was hypothesized that clock frequencies would grow so fast and wires so slow, that it would take tens of cycles to send information across a large chip. The microarchitectures we build and ship today really are not equipped to work under such delay constraints.

However, faster clocks and deeper pipelines ran into more fundamental problems. More deeply pipelined pro-

cessors are extremely complex to design and validate. As designers struggled to manage this complexity they also were obtaining diminishing performance returns from the approach. More pipeline stages increased the length of critical loops<sup>3</sup> in the processor, lengthening the number of cycles on the critical path of execution. Finally, while those pipeline stages enabled processors to be clocked faster, a linear increase in clock frequency creates a cubic increase in power consumption.<sup>6</sup> The power that a commodity desktop processor can consume and still be economically viable is capped by packaging and cooling costs, which assert a downward pressure on clock frequency.

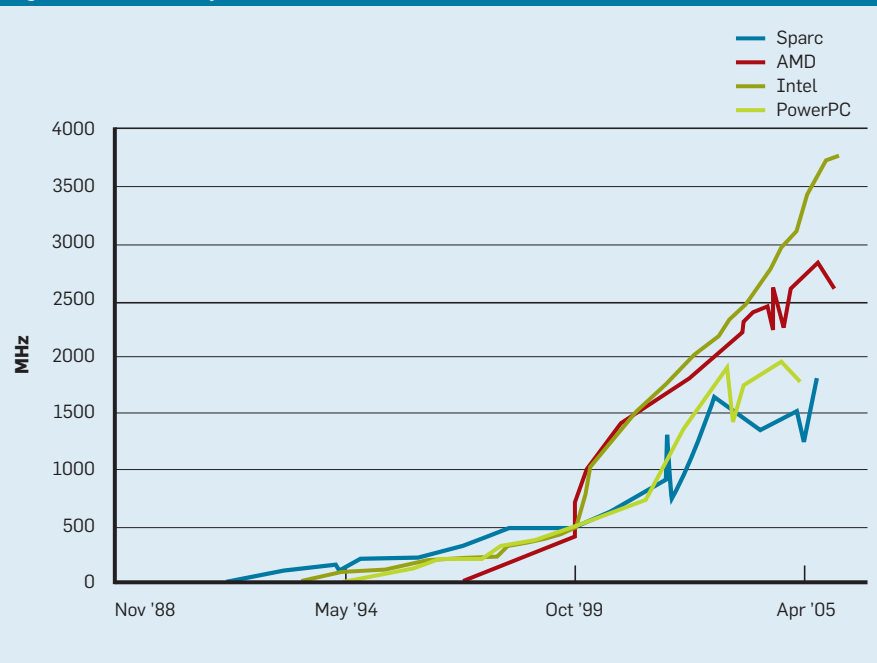
Collectively, these effects manifested themselves as a distinct change in the growth of processor frequency in 2004 (as indicated in Figure 1). Intel, in fact, stepped back from aggressive clock scaling in the Pentium 4 and with later products such the Core 2. AMD never attempted to build processors of the same frequency as Intel, but consequently suffered in the marketing game, whereby consumers erroneously assume frequency is the only indicator of CPU performance.

Clock frequency is clearly not the same thing as performance. CPU performance must be measured by observing the execution time of real applications. Reasonable people can argue about the validity of the SPEC benchmark suite. Most would admit it underrepresents memory and I/O effects. Nevertheless, when we consider much larger trends in performance over several years it is a reliable indicator of the progress computer architects and silicon technology have made.

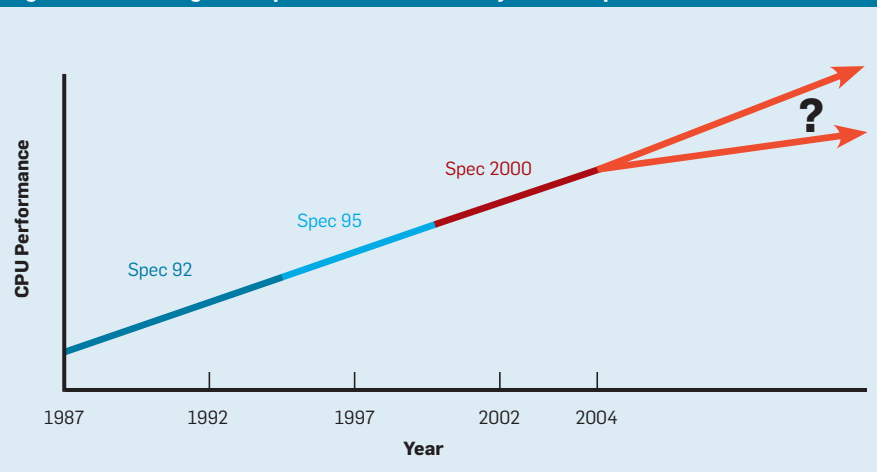
Figure 2 depicts CPU performance from 1982 to 2007, as measured by several different generations of SPEC integer benchmarks. The world changed by June 2004. Examining this 25-year time span, and now with four years of hindsight, it's clear we have a problem. We no longer are able to exponentially improve the rate of performance of single threaded applications.

The fact that we have been able to improve performance rates in the past has been a tremendous boon for the IT industry. Imagine if other industries, such as the auto or airline business, had at their core a driving source of exponential improvement. How would the auto or airline industries change if miles per gallon or transport speed doubled every two years? Exponential performance improvement drives down cost and increases the user experience by enabling ever richer applications. In fact, manufacturing, materials, architects, and compiler writers have been so effective at translating Moore's Law exponential increase in chip resources<sup>23</sup> into exponential performance improvements, that many people erroneously use the terms interchangeably. The question before us as a research field and an industry is, now that we no longer know how to translate Moore's Law growth in avail-

**Figure 1: CPU clock speed.**



**Figure 2: SPEC integer CPU performance over a 25-year time span.**




able silicon area per unit dollar into exponential performance increases at relatively fixed cost, what are we going to do instead?


**A Savior?** Processor manufacturers have bet their future on a relatively straightforward (for them) solution. That is, if we can't make one core execute a thread any faster, let's just place two cores on the die and modify the software to utilize the extra core. In the next generation, place four cores. The generation after that, eight, and so on. From a manufacturing standpoint, multicore or "manycore," as this approach is called, has several attractive qualities. First, we know how to build systems with higher peak performance. If the software can utilize them, then more cores per die will equate to improved performance. Unlike single-threaded performance, where we really have no clear ideas left to scale performance, multicore appears to offer us a path to salvation.

Second, again, if the software is there, a host of technological problems are mitigated by multicore. For example, as long as thread communication is kept to a minimum, it is more energy efficient to complete a fixed task using multiple threads, compared to executing one thread faster. Multiple smaller, simpler cores are easier to design than larger complex ones, thus mitigating the design and verification costs. Reliability, a growing problem in processor design, also becomes easier: simply place redundant cores on the die and post-fabrication route requests for defective units to one of the redundant cores, much as we do today with DRAMs. Or, even simpler, map them out entirely and sell a lower-cost part to a different market segment, as Sun Microsystems now does. Finally, wire delay—that grand challenge that motivated a flurry of research almost a decade ago—is also mitigated: simpler cores are smaller and clock frequency can be reduced as performance can be had through thread-level parallelism.

All of this sounds fantastic, except for one thing: it is predicated on the software being multithreaded. Just as important for future scalability, thread parallelism must be found in software at a rate commensurate with Moore's Law, which means if today we must find four independent threads of computa-



**The question before us as a research field and an industry is, now that we no longer know how to translate Moore's Law growth in available silicon area per unit dollar into exponential performance increases at relatively fixed chip cost, what are we going to do instead?**



tion, in two years there must be eight, and two years after that 16.

Processor manufacturers are not asking a small favor from software developers. From a programmer's perspective, multicore CPUs currently look no different than Symmetric Multiprocessors (SMPs) that have been around for decades. Such systems are not widely deployed on the home and business desktop, for good reason. They cost more and there isn't a significant performance advantage for the applications these users employ. So a reasonable question then is to ask: What makes us think this time it's going to work?

An optimist will make the following arguments: First, the cost difference is now in the reverse direction. Assuming we could build a faster single-threaded core, it will cost more. Design, validation, cooling, and manufacturing will assure that fact. Second, we do know more about parallel programming now than ever before. Tools have actually improved, with methods to look for race conditions and automatically parallelize loops,<sup>10</sup> and the resurgent interest in transactional programming will bear fruit. We've had many years of successful experience using parallelism in the graphics, server, and scientific computing domains. Third, and perhaps most importantly, it just has to work. For this reason software companies that need their products to achieve scalable performance must invest heavily into parallel programming. The hope is the commercial emphasis on parallel computing will create solutions.

A pessimist will counter thusly: Parallelism on the desktop has never worked because the technical requirements it takes to write threaded code just don't align with the economic forces driving desktop software developers. Writing parallel code is more difficult than writing sequential code. It's more error prone and difficult to debug, due to the non-determinism of thread memory interleavings. Furthermore, I have yet to meet anyone that thinks the industry will successfully parallelize its large legacy code bases. Once a large application has been designed for a single-threaded execution model, it is extremely difficult to tease it apart and parallelize it. What this means is programmers must feel the economic

forces to create threaded code from day one, not as a revision of the code base. Parallelizing code must be as much of a priority as writing correct code, or achieving a certain time to market.


A more realistic view of the future is somewhere between these two extremes. Parallelizing legacy code is widely viewed as a dead-end, but building compelling add-ons to existing applications and then “bolting on” these features to legacy codes is possible. One does not need to change the entire code base of a word processor, for example, in order to bolt on a speech recognition engine that exploits multicore. Furthermore, some applications that drive sales of new machines, such as interactive video games, have ample data parallelism that is relatively easy to extract with stream-based programming.

Finally, programmers will end up writing parallel software without realizing that is what they are doing. For example, programmers who utilize SQL databases will see their application’s performance improve just by virtue of some other developer’s effort spent on parallelizing the database engine itself. Extending this idea further, building parallel frameworks that fit various application classes (business, Web services, games, and so on) will enable programmers to more easily exploit multicore processors without having to bite off the whole complexity of parallel programming.


## Part II: The Architecture Research Community

Given this technology environment, what do computer architects currently research? To answer this question, it is best to look back over the last decade and understand what we thought were important research problems, and what happened to them.

**The memory wall.** A workshop, held in conjunction with the 1997 International Symposium of Computer Architecture (ISCA), focused on the memory wall and the research occurring on proposed solutions to it. The memory wall is the problem that accesses to main memory are significantly slower than computation. There are two aspects to it, a high latency to memory (hundreds of times the latency of a basic ALU operation inside a CPU) and a constrained



**Parallelizing legacy code is widely viewed as a dead-end, but building compelling add-ons to existing applications that take advantage of multicore, and then “bolting on” these features to legacy codes is possible.**



bandwidth. Excitement at the time was over solutions that proposed placing computational logic in the DRAM.<sup>11, 19, 20, 29, 32</sup> Such solutions never achieved broad acceptance in the marketplace because they required programmers to alter their software and they required DRAM manufacturers to restructure their business models. DRAM is a commodity, and businesses compete on cost. Adding logic to DRAM makes the devices expensive and system specific.

While technically feasible, it is a different business that DRAM manufacturers chose not to enter. However, less radical solutions, such as prefetching, stream buffers,<sup>18</sup> and ever larger on-chip caches,<sup>22</sup> did take hold commercially. Moreover, programmers became more amenable to tuning their applications to the memory hierarchy architects provide them. Cache-conscious data structures and algorithms are an effective, yet burdensome, way to achieve performance.

The memory wall is still with us. Accessing DRAM continues to require hundreds of more cycles than performing a basic ALU operation. While the drop in the growth of processor clock speed means that memory latency is less of a growing concern, the switch to multicore actually presents new challenges with bandwidth and consistency. Having all these CPU cores on a single die means they will need a Moore’s Law growth in bandwidth to memory in order to operate efficiently. At the moment, we are not pin limited in providing this bandwidth, but we quickly will be; so we can expect a host of future research that looks at the memory wall again, but this time from a bandwidth, instead of a latency perspective.

Along with memory performance is the evolution of the memory model. In the past, it was thought providing a sequentially consistent system with reasonable performance was not possible. Hence, we devised a range of relaxed consistency approaches.<sup>30</sup> It is natural for programmers to assume multicore systems are sequentially consistent, however, and recent work<sup>8</sup> suggests that architectures can use speculation<sup>16</sup> to provide it. Looking forward, as much as can be done, must be done, to make programming parallel systems as easy as possible. This author believes this will push



hardware vendors toward providing sequentially consistent systems.

**Power.** ISCA 98 brought a whole new vocabulary to architects. Terms such as power, energy, energy-delay product, bips-per-watt, and so on, would henceforth be part of the research parlance, with real debate about what quantity was most important to optimize. The slide that defined the power problem<sup>34</sup> depicted process generation on the x-axis, power per unit area on a log-scale on the y-axis, and various points depicting Intel processors, a hot plate, a nuclear reactor, a rocket nozzle, and the surface of the sun. The message from the slide was clear: change something, or processors would quickly have to be cooled by some technology capable of cooling the surface of the sun, clearly a ridiculous design point. The research on power was vast, starting with techniques for measuring power<sup>5, 42</sup> in microarchitectures. Since then, authors either included a power analysis of their work in their papers, or, reviewers would ask for it!

In reality, at the microarchitectural level, dynamic voltage/frequency scaling<sup>27</sup> (DVFS)—a circuit technique that reduces operating frequency and supply voltage when processors are idle, or need only operate at reduced performance—and clock-gating are highly effective. Several ideas for reducing power beyond DVFS and clock-gating have been proposed, and do work, but the most bang for the buck comes from doing these two techniques well. Looking forward, power will continue to cap performance and drive design considerations. The macroscopic environment in which we consider power issues has changed slightly from

1998, however. The massive power consumption occurring in data-centers makes companies that operate them power-hungry, shopping for the best physical and regulatory environment in which to obtain cheap energy. These companies will benefit from multicore devices, as their software is task-parallel, and using multiple simple cores is a more energy efficient means to compute than with single complex CPUs. The usefulness of portable devices is also effectively constrained by power, as improvements in battery technology continue to lag in the single digits. Thus, architects will continue to consider power in their ideas, as it continues to be an important design consideration.

**Design Complexity.** In the mid-1990s, a community of architects began to focus on the complexity of modern CPU designs.<sup>31</sup> Processors today contain approximately 1,000 times more core (non-cache) transistors than 30 years ago. It is just not possible to have a bug-free design for such a complex device with the engineering methodologies that we currently employ. Such complex designs are difficult to innovate, as design changes cannot be reasoned about locally. Moreover, large monolithic designs often have long wires, which consume power and constrain the clock cycle. Several projects were spurred by these motivations to propose fairly radical changes to the processing model.<sup>26, 36, 37</sup> A wealth of less radical, more localized solutions were developed, among them ways to reduce instruction scheduling logic,<sup>14</sup> reduce the complexity of out of order structures,<sup>9, 38</sup> and provide

a perceived increase in processor issue width by coarser management of fetched instructions.<sup>4</sup>

Design complexity is still an issue today, but the switch to multicore has effectively halted the growth in core complexity. With processor vendors banking their future on multicore, they are expecting performance to come from additional cores, not more complex ones. Moreover, there are strong arguments to be made, that if the thread parallelism is available in the applications, then vendors will switch to more energy-efficient, simpler cores. In effect, the trend in core complexity could actually reverse. How far this trend will go no one knows, but if software does indeed catch up and become thread-parallel, then we could see heterogeneous multicore devices, with one or a handful of complex cores and a sea of simple, reduced-ISA ones<sup>43</sup> that will provide the most performance per dollar and per watt.

**Reliability.** In 2001, concerns about both hard and soft faults began to appear at ISCA. Yet another new vocabulary term appeared in our literature: the high-energy particle. As silicon feature sizes shrink, the quantity of charge held on any particular wire in a microprocessor also is reduced. Normally this has a positive benefit (lower power, faster), but it also means that the charge on that line can be on the order of that induced by an alpha particle striking the silicon lattice. This is not a new problem, as “hardened” microprocessors have been built for decades for the space industry as electronics in space must operate without the natural high-energy particle absorption effect of the atmosphere. Now our earth-

## Computer Architecture in Education

My survey of the top 10 CS and engineering departments suggests that a strong majority (80%) require some form of computer architecture class in order to receive a bachelor's degree in CS. At the moment, advanced architecture, such as parallel systems, and advanced programming techniques, such as parallel programming, appear to be relegated to

electives. If computer architects fail to get microprocessors back on an exponential performance curve, as appears likely at the moment, then clearly the curriculum in our universities will need revision.

Students graduating with a competence in engineering software for parallel systems will have a distinct advantage in the work force over those

who do not. But creating this competence is elusive. Most universities that I surveyed begin their programs by teaching basic programming—or programming for single-core devices. What this means is undergraduates are taught from day one in CS departments to think about engineering sequential solutions. Clearly this will not do in a multicore era, still

the complexities of teaching threads, locks, and barriers to freshmen computer scientists seem daunting. Nevertheless, departments that are able to integrate parallel computing throughout their curriculum will position their undergraduates to be the future leaders in the IT industry. Thus, they should be well motivated to think about curriculum change.

bound devices must also deal with this problem. Architects proposed a cornucopia of techniques to deal with faults, from the radical, which proposed alternative processor designs<sup>2</sup> and ways to use simultaneous multithreaded devices,<sup>24, 39</sup> to the more easily adoptable by industry, such as cache designs with better fault resilience. Important work also better characterized what parts of the microarchitecture are actually susceptible to a dynamic fault.

Reliability continues to play an important part of architecture research, but the future presents some differing technology trends. It is this author's opinion that Moore's Law will not stop anytime soon, but it won't be because we shrink feature sizes down to a handful of atoms in width.<sup>44</sup> Rather, die-stacking will continue to provide ever more chip real estate. These dies will have a fixed (or even larger) feature size, and thus the growth in dynamic faults due to reduced feature sizes should actually stop. Moreover, if multicore does actually prove to be a market success, then reliability can be achieved without enormous complexity: processors with manufactured faults can be mapped out, and for applications that require high reliability, multiple cores can be used to redundantly perform the computation. Nevertheless, despite this positive long-term outlook, work to improve reliability will always have purpose, as improved reliability

leads directly to improved yields (and in the future, improved performance if redundant cores are not required), and thus reduced costs.

**Evaluation techniques.** How architects do research has changed dramatically over the decades. When ISCA first started in the 1970s, papers typically provided paper designs and qualitative or simple analytical arguments to the idea's effectiveness. Research techniques changed significantly in the early 1980s with the ability to simulate new architecture proposals, and thus provide quantitative evidence to back up intuition. Simulation and quantitative approaches have their place, but misused, they can provide an easy way to produce a lot of meaningless, but convincing looking data. Sadly, it is now commonly accepted in our community that the absolute value of any data presented in a paper isn't meaningful. We take solace in the fact the trends—the relative difference between two data points—likely have a corresponding difference in the real world.

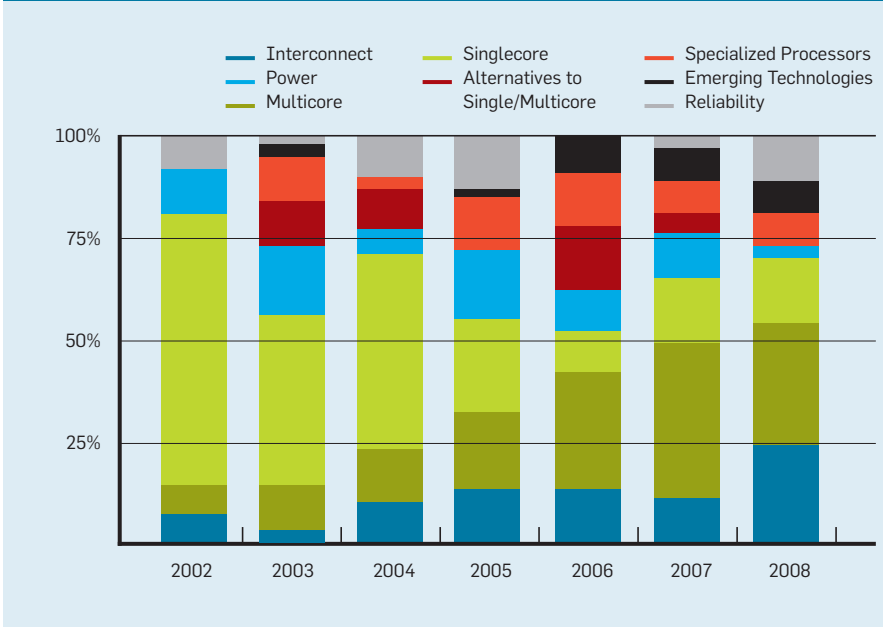
As an engineer, this approach to our field is sketchy, but workable; as a scientist, this seems like a terrible place to be. It is nearly impossible to do something as simple as reproduce the results in a paper. Doing so from the paper alone requires starting from the same simulation infrastructure as the authors, implementing the idea as the authors did, and then execut-

ing the same benchmarks, compiled with the same compiler with the same settings, as the authors. Starting from scratch on this isn't tractable, and the only real way to reproduce a paper's results is to ask the authors to share their infrastructure. Another, more insidious problem with simulation is its too easy to make mistakes when implementing a component model. Because it is common, and even desirable, to separate functional ISA modeling from performance modeling, these performance model errors can go unnoticed, thus leading to entirely incorrect data and conclusions. Despite these drawbacks, quantitative data is seductive to reviewers, and simulation is the most labor-efficient way to produce it.

Looking forward, the picture is muddled. Simulation will continue to be the most important tool in the computer architect's toolbox. The need to model ever more parallel architectures, however, will create the need to continue to explore different modeling techniques because, for the moment, the tools used in computer architecture research are built on single-threaded code bases. Thus, simulating an exponentially increasing number of CPU cores means an exponential increase in simulation time. Fortunately, several paths forward exist. Work on high level performance models<sup>13, 28</sup> provides accurate relative performance data quickly, suitable for coarsely mapping a design space. Techniques to sample<sup>35, 41</sup> simulation data enable architects to explore longer-running simulations with reasonable confidence. Finally, renewed interest in prototyping and using FPGAs for simulation<sup>40</sup> will allow architects to explore ideas that require cooperation with language and application researchers as the speed of FPGA-based simulation is just fast enough to be usable by software developers.

There are several advantages to pre-built and shared tools for architecture research. They are enablers, allowing research groups to not start from scratch. Having shared tools has another benefit: the bugs and inaccuracies in those tools can be revealed and fixed over time. Shared tools also enable re-creating other people's work easier. There has been, and there will continue to be, a downside to the avail-


Figure 3: Papers published in ISCA 2001–2006.




ability of pre-built tools for architecture research, however. Just as SimpleScalar<sup>7</sup> created a flood of research on super-scalar microarchitecture, the availability of pre-canned tools and benchmarks for CMPs will create a flood of research that is one delta away from existing CMP designs. But is this the type of research academics should be conducting? As academics, shouldn't we be looking much farther downfield, to the places where industry is not yet willing to go? This is an age-old quandary in our community and will likely continue to be so. Such a debate will certainly continue to exist in our research community for the foreseeable future.

In the computer architecture field there is a cynical saying that goes something like “we design tomorrow's systems with yesterday's benchmarks.” This author finds this statement extreme, but there is some underlying merit to it. For example, there are far more managed-code and scripting language developers out there than C/C++ ones. Yet the majority of benchmarks used in our field are written in C. Fortunately, this is changing, with newer benchmarks such as SPECjvm and SPECjbb. Moreover, a few researchers are starting to focus on performance issues of managed and scripted code. Looking forward, there is a very real need for realistic multithreaded benchmarks. Recent work, suggests a kernel-driven approach is sufficient.<sup>33</sup> As with the whole of architecture evaluation techniques, the jury is still out on what is the proper methodology.

**Instruction-Level Parallelism.** Finally, a large number of architects, myself among them, are still putting enormous effort into finding additional instruction-level parallelism (ILP). Some of these architects don't have complete faith that multicore will be a success. Others recognize that improvements in single-threaded performance benefit multicore as well, as parts of applications will be sequential or require a few threads to execute quickly. Over the years, these researchers have sought to find ILP in every nook and cranny of the research space. Everything from new instruction set architectures, new execution models, to better branch predictors, caches, register management, instruc-



**People far older and wiser than me contend this is the most exciting time for architecture since the invention of the computer. What makes it exciting is that architecture is in the unique position of being at the center of the future of computer science and the IT industry.**



tion scheduling, and so on. The list of areas explored is endless.

Alongside the development of x86 microprocessors in the 1990s and 2000s, Intel and HP sunk enormous effort and dollars into developing another line of processors, Itanium,<sup>22</sup> that gain performance from ILP. Itanium is a Very Long Instruction Word (VLIW) processor, in the mold of far earlier work on the subject.<sup>15</sup> Such processors promise performance from ILP at reduced complexity compared to superscalar designs, by relying on sophisticated compilation technology. VLIW is a fine idea; it communicates more semantic knowledge about fine-grained parallelism from the software to the hardware. If technically such an approach is useful, why don't you have an Itanium processor on your desktop? In a nutshell, such processors never achieved a price point that fit well in the commodity PC market. Moreover, in order to maintain binary compatibility with x86, sophisticated binary translation mechanisms had to be employed. After such translation, existing code saw little to no performance benefit from executing on Itanium. Consumers were loath to spend more on a system that was no faster, if not slower, than the cheaper alternative, for the promise that someday faster native-code applications would arrive. There is a lesson here for multicore systems as well: without tangible benefits, consumers will not spend money on new hardware just for its technical superiority.

Does ILP still matter? This author would argue it does. As mentioned earlier, even parallel programs have sequential parts. Legacy code still matters to the IT industry. Is there more ILP to be had? This is a more difficult question to answer. The seminal work in this area<sup>21</sup> suggests there is. Extracting it from applications, however, is no trivial matter. The low-hanging fruit was gone before I even entered the field! Aggressive speculation to address the memory wall, the inherent difficulty in predicting certain branches, and the false control and memory dependencies introduced by the imperative language programming model is required. This must be carried out by architectures that are simple to design and validate, lack monolithic control



structures, and that are backward compatible, if not with the binaries, with the programming model.

**What are we doing now?** A look at the ISCA 2007 conference program provides a good overview of the type of research being done in our community. A survey of papers published in that year reveals the following: 18 papers focused on multicore (eight core and memory design, six transactional programming, four on-chip interconnect); six papers were focused on single-core devices and/or applications, six papers were focused on special-purpose or streaming/media devices, four papers were focused on power reduction and three were in the general area of “beyond CMOS” Figure 3 extends this data out for the last seven years of ISCA. This data extends the work of Hill,<sup>45</sup> who tracked papers published in ISCA by category from 1973–2001. That data showed a precipitous rise and fall of interest in multiprocessor research, while data from the last seven years depicts a renewed and vigorous multiprocessor research environment.

### The Most Exciting Time

In my lifetime, this is the most exciting time for computer architecture research; indeed, people far older and wiser than me<sup>46</sup> contend this is the most exciting time for architecture since the invention of the computer. What makes it exciting is that architecture is in the unique position of being at the center of the future of computer science and the IT industry. Innovations in architecture will impact everything from education to determining who are the new winners and losers in the IT business. Central to this excitement for me as an academic, is there is no real clear way to proceed. Multicore devices are being sold, and parts of the software ecosystem will utilize them, but the research and product space is far more fluid and open to new ideas now than ever before. Thus, while we are central to the future directions of computer science, we really lack a clear vision for how to proceed. What could be better than that? □

#### References

1. Agarwal, V., Hrishikesh, M.S., Keckler, S.W., and Burger, D. Clock rate versus IPC: The end of the road for

- conventional microarchitectures. *SIGARCH Comput. Archit. News* 28, 2, (2000), 248–259.
2. Austin, T.M. Diva: A reliable substrate for deep submicron microarchitecture design. *Micro*. 00 196, 18–29.
3. Borch, E. Tune, E., Manne, S., and Emer, J. Loose loops sink chips. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*. Feb. 2–6, 2002, 299–310.
4. Bracy, A., Prahlad, P., and Roth, A. Dataflow mini-graphs: Amplifying superscalar capacity and bandwidth. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Washington, D.C., 2004, 18–29.
5. Brooks, D., Tiwari, V., and Martonosi, M. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News* 28, 2, (2000), 83–94.
6. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.-D., Zyuban, V., Gupta, M., and Cook, P.W. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro* 20, 6 (2000), 26–44.
7. Burger, D., and Austin, T.M. The simplescalar tool set, version 2.0. *SIGARCH Comput. Archit. News* 25, 3 (1997), 13–25.
8. Ceze, L., Tuck, J., Montesinos, P., and Torrellas, J. Bulks: Bulk enforcement of sequential consistency. *SIGARCH Comput. Archit. News* 35, 2 (2007), 278–289.
9. Cristal, A., Ortega, D., Llosa, J., and Valero, M. Out-of-order commit processors. *hpc*, 00:48, 2004.
10. Dagum, R., Menon, L. Openmp: An industry standard api for shared-memory programming. *Computational Science and Engineering* 5, 11, (Jan-Mar 1998) 46–55.
11. Draper, J., Chame, J., Hall, M., Steele, C., Barrett, T., LaCoss, J., Granacki, J., Shin, J., Chen, C., Kang, C.W., Kim, I., and Daglikoga, G. The architecture of the diva processing-in-memory chip. In *Proceedings of the 16th International Conference on Supercomputing*, ACM, NY, 2002, 14–25.
12. Eden, T., Mudge, A.N., The YAGS branch prediction scheme. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture* (Nov. 30–Dec. 2, 1998), 69–77.
13. Eeckhout, L., Stougie, B., Bosschere, K.D., and John, L.K. Control flow modeling in statistical simulation for accurate and efficient processor design studies. *SIGARCH Comput. Archit. News* 32, 2 (2004), 350.
14. Ernst, D., Hamel, A., and Austin, T. Cyclone: A broadcast-free dynamic instruction scheduler with selective replay. In *Proceedings of the 30th Annual International Symposium on Computer Architecture* (June 9–11, 2003), 253–262.
15. Fisher, J.A. Very long instruction word architectures and the eli-512. In *Proceedings of the 10th Annual International Symposium on Computer Architecture* (Los Alamitos, CA, 1983). IEEE Computer Society Press, 140–150.
16. Hill, M.D. Multiprocessors should support simple memory-consistency models. *IEEE Computer* 31, 8 (1998), 28–34.
17. Hinton, G., Upton, M., Sager, D., Boggs, D., Carmean, D., Roussel, P., Chappell, T., Fletcher, T., Milshtein, M., Sprague, M., Samaan, S., and Murray, R. A 0.18-μm CMOS ia-32 processor with a 4-ghz integer execution unit. *IEEE Journal of Solid-State Circuits*, 36, 11 (Nov. 2001), 1617–1627.
18. Jouppi, N.P. Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers. *SIGARCH Comput. Archit. News*, 18, 3a (1990), 364–373.
19. Kang, Y., Huang, W., Yoo, S.-M., Keen, D., Ge, Z., Lam, V., Torrellas, J., and Pattanaik, P. Flexram: Toward an advanced intelligent memory system. *ICCD* 00:192, 1999.
20. Kogge, P., Sunaga, T., Miyatake, H., Kitamura, K., and Retter, E. Combined DRAM and logic chip for massively parallel systems. *arXiv* 04:4, 1995.
21. Lam, M.S., and Wilson, R.P. Limits of control flow on parallelism. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*. ACM, NY, 1992, 46–57.
22. McNairy, D., Soltis, C. Itanium 2 processor microarchitecture. *IEEE Micro* 23, 2 (Mar.–Apr. 2003), 44–55.
23. Moore, G. Cramming more components onto integrated circuits. *Electronics* (Apr. 1965), 114–117.
24. Mukherjee, S., Kontz, M., and Reinhardt, S. Detailed design and evaluation of redundant multithreading

- alternatives. In *Proceedings of the 29th Annual International Symposium on Computer Architecture* (2002), 99–110.
25. Mukherjee, S., Weaver, C., Emer, J., Reinhardt, S., and Austin, T. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture* (Dec. 3–5, 2003), 29–40.
26. Nagarajan, R., Sankaralingam, K., Burger, D., and Keckler, S.W. A design space evaluation of grid processor architectures. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society, Washington, D.C. 2001, 40–51.
27. Nielsen, L.S., and Niessen, C. Low-power operation using self-timed circuits and adaptive scaling of the supply voltage. *IEEE Trans. Very Large Scale Integr. Syst.*, 2, 4 (1994), 391–397.
28. Oskin, M., Chong, F.T., and Farnens, M. HIS: Combining statistical and symbolic simulation to guide microprocessor designs. *SIGARCH Comput. Archit. News* 28, 2 (2002), 71–82.
29. Oskin, M., Chong, F.T., and Sherwood, T. Active pages: A computation model for intelligent memory. *SIGARCH Comput. Archit. News* 26, 3 (1998), 192–203.
30. Pai, V.S., Ranganathan, P., Adve, S.V., and Harton, T. An evaluation of memory consistency models for shared-memory systems with ilp processors. *SIGPLAN Notices* 31, 9 (1996), 12–23.
31. Palacharla, S. Complexity-effective superscalar processors. Ph.D. thesis, 1998.
32. Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R., and Yelick, K. A case for intelligent RAM. *IEEE Micro* 17, 2 (Mar.–Apr. 1997), 34–44.
33. Patterson, D., Keutzer, K., Asanovic, K., Yelick, K., and Bodik, R. The landscape of parallel computing research: A view from Berkeley. 2007.
34. Pollack, F., Keynote: New microarchitecture challenges in the coming generations of CMOS process technologies, 1999.
35. Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, NY, 2002, 45–57.
36. Swanson, S., Michelson, K., Schwerin, A., and Oskin, M. Wavescalar. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Washington, D.C., 2001.
37. Taylor, M.B., Kim, J., Miller, J., Wentzlaff, D., Ghodrati, F., Greenwald, B., Hoffman, F., Johnson, P., Lee, J.-W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpfen, V., Frank, M., Amarasinghe, S., and Agarwal, A. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro* 22, 2 (2002), 25–35.
38. Valero, M., Gonzalez, A., Topham, N.P., and Cruz, C. Multiple-banked register file architectures. *isca* 00:316, 2000.
39. Vijaykumar, P., Pomeranz, I., and Cheng, K. Transient-fault recovery using simultaneous multithreading. In *Proceedings of the 29th Annual International Symposium on Computer Architecture* (2002), 87–98.
40. Wawrzynek, J., Patterson, D., Oskin, M., Lu, S.-L., Kozyrakis, C., Hoe, J. C., Chiou, D., and Asanovi, K. RAMP: Research accelerator for multiple processors. *IEEE Micro* 27, 2 (2007), 46–57.
41. Wunderlich, R., Wenisch, T., Falsafi, B., and Hoe, J. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture* (June 9–11, 2003), 84–95.
42. Ye, W., Vijaykrishnan, N., Kandemir, M., and Irwin, M.J. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the 37th Conference on Design Automation*. ACM, NY (2000), 340–345.
43. <http://www.news.com/2100-10063-6119618.html>.
44. <http://www.itrs.net/>.
45. <http://pages.cs.wisc.edu/markhill/mp2001.html>.
46. Personal communication with Burton Smith.

**Mark Oskin** (oskin@cs.washington.edu) is an associate professor in the Department of Computer Science and Engineering at University of Washington, Seattle.

©2008 ACM 0001-0782/08/0700 \$5.00

The 1st ACM SIGGRAPH Conference and Exhibition in Asia  
[www.siggraph.org/asia2008](http://www.siggraph.org/asia2008)

## New Horizons

### ACM SIGGRAPH launches the premiere SIGGRAPH Asia in Singapore

Programs include:

- Papers
- Sketches and Posters
- Courses
- Art Gallery
- Computer Animation Festival
- Educators Program
- Emerging Technologies
- Exhibition

***Calling all creative researchers, artists,  
digital innovators and animators!***

This is your opportunity to present your stellar work or attend the 1st ACM SIGGRAPH conference and Exhibition in Asia.

#### ***Queries?***

Contact Conference and Exhibitions  
Management at [asia2008@siggraph.org](mailto:asia2008@siggraph.org) or  
Tel: +65 6500 6700



SIGGRAPHASIA2008

Conference and Exhibition on Computer Graphics and Interactive Techniques

**Singapore, 10-13 December 2008**

Held in  
**UNIQUELY  
Singapore**  
[visitsingapore.com](http://visitsingapore.com)



## Is TM the answer for improving parallel programming?

BY JAMES LARUS AND CHRISTOS KOZYRAKIS

# Transactional Memory

AS COMPUTERS EVOLVE, programming changes as well. The past few years mark the start of a historic transition from sequential to parallel computation in the processors used in most personal, server, and mobile computers. This shift marks the end of a remarkable 30-year period in which advances in semiconductor technology and computer architecture improved the performance of sequential processors at an annual rate of 40%–50%. This steady performance increase benefited all software, and this progress was a key factor driving the spread of software throughout modern life.

This remarkable era stopped when practical limits on the power dissipation of a chip ended the continual increases in clock speed and limited instruction-level parallelism diminished the benefit of increasingly

complex processor architectures. The era did not stop because Moore's Law<sup>a</sup> ended. Semiconductor technology is still capable of doubling the transistors on a chip every two years. However, this flood of transistors now increases the number of independent processors on a chip, rather than making an individual processor run faster. The resulting computer architecture, named Multicore, consists of several independent processors (cores) on a chip that communicate through shared memory. Today, two-core chips are common and four-core chips are coming to market, and there is every reason to believe that the number of cores will continue to double for a number of generations. On one hand, the good news is that the peak performance of a Multicore computer doubles each time the number of cores doubles. On the other hand, achieving this performance requires a program execute in parallel and scale as the number of processors increase.

Few programs today are written to exploit parallelism effectively. In part, most programmers did not have access to parallel computers, which were limited to domains with large, naturally parallel workloads, such as servers, or huge computations, such as high-performance computing. Because mainstream programming was sequential programming, most existing programming languages, libraries, design patterns, and training do not address the challenges of parallelism *programming*. Obviously, this situation must change before programmers in general will start writing parallel programs for Multicore processors.

A primary challenge is to find better abstractions for expressing parallel computation and for writing parallel programs. Parallel programming encompasses all of the difficulties of sequential programming, but also introduces the hard problem of coordinating interactions among concurrently executing tasks. Today, most parallel

a. The doubling every 18–24 months of the number of transistors fabricable on a chip.





programs employ low-level programming constructs that are just a thin veneer over the underlying hardware. These constructs consist of threads, which are an abstract processor, and explicit synchronization (for example, locks, semaphores, and monitors) to coordinate thread execution. Considerable experience has shown that parallel programs written with these constructs are difficult to design, program, debug, maintain, and—to add insult to injury—often do not perform well.

Transactional memory (TM)—proposed by Lomet<sup>19</sup> and first practically implemented by Herlihy and Moss<sup>13</sup>—is a new programming construct that offers a higher-level abstraction for writing parallel programs. In the past few years, it has engendered consider-

able interest, as transactions have long been used in databases to isolate concurrent activities. TM offers a mechanism that allows portions of a program to execute in isolation, without regard to other, concurrently executing tasks. A programmer can reason about the correctness of code within a transaction and need not worry about complex interactions with other, concurrently executing parts of the program. TM offers a promising, but as yet unproven mechanism to improve parallel programming.

### What is Transaction Memory?

A transaction is a form of program execution borrowed from the database community.<sup>8</sup> Concurrent queries conflict when they read and write an item in a database, and a conflict can produce

an erroneous result that could not arise from a sequential execution of the queries. Transactions ensure that all queries produce the same result as if they executed serially (a property known as “serializability”). Decomposing the semantics of a transaction yields four requirements, usually called the “ACID” properties—atomicity, consistency, isolation, and durability.

TM provides lightweight transactions for threads running in a shared address space. TM ensures the atomicity and isolation of concurrently executing tasks. (In general, TM does not provide consistency or durability.) Atomicity ensures program state changes effected by code executing in a transaction are indivisible from the perspective of other, concurrently executing

transactions. In other words, although the code in a transaction may modify individual variables through a series of assignments, another computation can only observe the program state immediately before or immediately after the transaction executes. *Isolation* ensures that concurrently executing tasks cannot affect the result of a transaction, so a transaction produces the same answer as when no other task was executing. Transactions provide a basis to construct parallel abstractions, which are building blocks that can be combined without knowledge of their internal details, much as procedures and objects provide composable abstractions for sequential code.

**TM Programming Model.** A programming model provides a rationale for the design of programming language constructs and guidance on how to construct programs. Like many aspects of TM, its programming model is still the subject of active investigation.

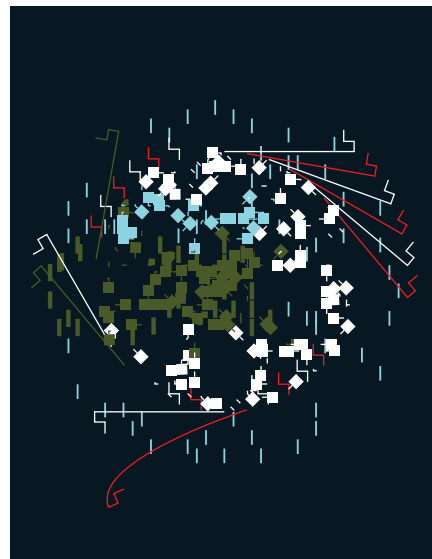
Most TM systems provide simple atomic statements that execute a block of code (and the routines it invokes) as a transaction. An atomic block isolates the code from concurrently executed threads, but a block is not a replacement for general synchronization such as semaphores or condition variables.<sup>2</sup> In particular, atomic blocks by themselves do not provide a means to coordinate code running on parallel threads.

Automatic mutual exclusion (AME), by contrast, turns the transactional model “inside-out” by executing most of a program in transactions.<sup>15</sup> AME supports asynchronous programming, in which a function starts one or more asynchronous computations and later rendezvous to retrieve their results. This programming model is a common way to deal with unpredictable latency in user-directed and distributed systems. The atomicity provided by transactions ensures that an asynchronous computation, which executes at an unpredictable rate, does not interfere with other, simultaneously active computations.

**Advantages of Transactional Memory.** Parallel programming poses many difficulties, but one of the most serious challenges in writing correct code is coordinating access to data shared by several threads. Data races, deadlocks,

and poor scalability are consequences of trying to ensure mutual exclusion with too little or too much synchronization. TM offers a simpler alternative to mutual exclusion by shifting the burden of correct synchronization from a programmer to the TM system.<sup>9</sup> In theory, a program’s author only needs to identify a sequence of operations on shared data that should appear to execute atomically to other, concurrent threads. Through the many mechanisms discussed here, the TM system then ensures this outcome.

Harris and Peyton-Jones<sup>11</sup> argued that, beyond providing a better programming abstraction, transactions



also make synchronization composable, which enables the construction of concurrency programming abstractions. A programming abstraction is composable if it can be correctly combined with other abstractions without needing to understand how the abstractions operate.

Simple locking is not composable. Consider, as an example, a class that implements a collection of bank accounts. The class provides thread-safe `Deposit` and `Withdraw` operations to add and remove money from a bank account. Suppose that we want to compose these operations into a thread-safe `Transfer` operation, which moves money from one account to another. The intermediate state, in which money was debited but not credited, should not be visible to other threads (that is, the transfer should be atomic). Since `Deposit` and `Withdraw` lock an

account only during their operation, properly implementing `Transfer` requires understanding and modifying the class’s locking discipline by adding a method to either lock all accounts or lock a single account. The latter approach allows non-overlapping transfers to execute concurrently, but introduces the possibility of deadlock if a transfer from account A to B overlaps with a transfer from B to A.

TM allows the operations to be composed directly. The `Deposit` and `Withdraw` operations each execute in a transaction, to protect their manipulations of the underlying data. The `Transfer` operation also executes in a transaction, which subsumes the underlying operations into a single atomic action.

**Limitations of Transactional Memory.** Transactions by themselves cannot replace all synchronization in a parallel program.<sup>2</sup> Beyond mutual exclusion, synchronization is often used to coordinate independent tasks, for example, by ensuring that one task waits for another to finish or by limiting the number of threads performing a task.

Transactions by themselves provide little assistance in coordinating independent tasks. For example, consider a producer-consumer programming relationship, in which one task writes a value that another task reads. Transactions can ensure the tasks’ shared accesses do not interfere. However, this pattern is expensive to implement with transactions, whose goal is to shield a task from interactions with other tasks. If the consumer transaction finds the value is not available, it can only abort and check for the value later. Busy waiting by aborting is inefficient since an aborted transaction rolls back its entire computation. A better solution is for the producer to signal the consumer when the value is ready. However, since a signal is not visible in a transaction, many TM systems provide a guard that prevents a transaction from starting execution until a predicate becomes true.

Haskell TM introduced the `retry` and `orElse` constructs as a way for a transaction to wait until an event occurs and to sequence the execution of two transactions.<sup>11</sup> Executing a `retry` statement causes the surrounding transaction to abort. It does not reexecute until a location it previously

read changes value, which avoids the crudest form of busy waiting in which a transaction repeatedly reads an unchanging value and aborts. The `orElse` construct composes two transactions by starting the second one only if the first transaction fails to commit. This pattern—which arises in situations as simple as checking for a value in a cache and recomputing it if necessary—is difficult to express otherwise, since a transaction’s failure and reexecution is transparent to other computation.

We still do not understand the trade-offs and programming pragmatics of the TM programming model. For example, the semantics of nested transactions is an area of active debate. Suppose that code in a transaction *O* invokes a library routine, which starts its own transaction *I*. Should the two transactions interact in any way, and if so, what are the implications for the TM implementation and for programmers building modular software and libraries? Consider when transaction *I* commits. Should its results be visible only to code in transaction *O* (closed nesting) or also to other threads (open nesting)? If the latter, what happens if transaction *O* aborts? Similarly, if transaction *I* aborts, should it terminate transaction *O* as well, or should the inner transaction be rolled back and restarted independently?

Finally, the performance of TM is not yet good enough for widespread use. Software TM systems (STM) impose considerable overhead costs on code running in a transaction, which saps the performance advantages of parallel computers. Hardware TM systems (HTM) can lower the overhead, but they are only starting to become commercially available, and most HTM systems fall back on software for large transactions. Better implementation techniques are likely to improve both types of systems and are an area of active research.

### Transactional Memory Implementation

TM can be implemented entirely in software (STM) or with specialized hardware support (HTM). Many different implementation techniques have been proposed, and this paper, rather than surveying the literature, focuses

on a few key techniques. A more complete overview is available elsewhere.<sup>18</sup>

Most TM systems of both types implement optimistic concurrency control in which a transaction executes under the assumption that it will not conflict with another transaction. If two transactions conflict, because one modifies a location read or modified by the other, the TM system aborts one of the transactions by reversing (rolling back) its side effects. The alternative pessimistic concurrency control requires a transaction to establish exclusive access to a location (for example, by acquiring a lock) before modifying it. This approach also



may abort and roll back a transaction, in case of deadlock.

#### Software Transactional Memory.

The initial paper on STM by Shavit and Touitou<sup>29</sup> showed it was possible to implement lock-free, atomic, multi-location operations entirely in software, but it required a program to declare in advance the memory locations to be accessed by a transaction.

Herlihy et al.’s Dynamic STM (DSTM)<sup>14</sup> was the first STM system that did not require a program to declare the memory locations accessed by a transaction. DSTM is an object-granularity, deferred-update STM system, which means that a transaction modifies a private copy of an object and only makes its changes visible to other transactions when it commits. The transaction exclusively accesses the copy without synchronization. However, another transaction can access the

original, underlying object while the first transaction is still running, which causes a logical conflict that the STM system detects and resolves by aborting one of the two transactions.

An STM system can detect a conflict when a transaction first accesses an object (early detection) or when the transaction attempts to commit (late detection). Both approaches yield the same results, but may perform differently and, unfortunately, neither is consistently superior. Early detection prevents a transaction from performing unnecessary computation that a subsequent abort will discard. Late detection can avoid unnecessary aborts, as when the conflicting transaction itself aborts because of a conflict with a third transaction.

Another complication is a conflict between a transaction that only reads an object and another that modifies the object. Since reads are more common than writes, STM systems only clone objects that are modified. To reduce overhead, a transaction tracks the objects it reads and, before it commits, ensures that no other transaction modified them.

DSTM is a library. An object manipulated in a transaction is first registered with the DSTM system, which returns a `TMOBJECT` wrapper for the object (as illustrated in the accompanying figure). Subsequently, the code executing the transaction can open the `TMOBJECT` for read-only or read-write access, which returns a pointer to the original or cloned object, respectively. Either way, the transaction manipulates the object directly, without further synchronization.

A transaction ends when the program attempts to commit the transaction’s changes. If the transaction successfully commits, the DSTM system atomically replaces, for all modified objects, the old object in a `Locator` structure with its modified version.

A transaction *T* can commit successfully if it meets two conditions. The first is that no concurrently executing transaction modified an object read by *T*. DSTM tracks the objects a transaction opened for reading and validates the entries in this read set when the transaction attempts to commit. An object in the read set is valid if its version is unchanged since transaction



T first opened it. DSTM also validates the read set every time it opens an object, to avoid allowing a transaction to continue executing in an erroneous program state in which some objects changed after the transaction started execution.

The second condition is that transaction T is not modifying an object that another transaction is also modifying. DSTM prevents this type of conflict by only allowing one transaction to open an object for modification. When a write-write conflict occurs, DSTM aborts one of the two conflicting transactions and allows the other to proceed. DSTM rolls the aborted transaction back to its initial state and then allow it to reexecute. The policy used to select which transaction to abort can affect system performance, including liveness, but it should have no effect on the semantics of the STM system.<sup>28</sup>

The performance of DSTM, like other STM systems, depends on the details of the workload. In general, the large overheads of STM systems are more expensive than locking on a small number of processors. However, as the number of processors increases, so does the contention for a lock and the cost of locking. When this occurs and conflicts are rare, STMs have been shown to outperform locks on small benchmarks.

**Deferred-Update Systems.** Other deferred-update STM systems investigated alternative implementation

techniques. Harris and Fraser's WSTM system detects conflicts at word, not object, granularity. This approach can avoid unnecessary conflicts if two transactions access different fields in an object, but it complicates the implementation sufficiently that few STM systems adopted the idea (although, HTM systems generally detect conflicts at word or cache line granularity). WSTM also was the first STM system integrated into a programming language. Harris and Fraser extended Java with an atomic statement that executed its block in a transaction, for example:

```
atomic {
    int x = lst.head;
    lst = lst.tail;
    ...
}
```

The construct also provided an optional guard that prevents a transaction from executing until the condition becomes true.

Considerable research has investigated the policies that select which transaction to abort at a conflict.<sup>10, 28</sup> No one policy performs best in all situations, though a policy called "Polka" performed well overall. Under this policy, each transaction tracks the number of objects it has open and uses this count as its priority. A transaction attempting to acquire access to an object immediately aborts a conflicting, lower-priority transaction. If the ac-

quiring transaction's priority is lower, it backs off N times, where N is the difference in priority, with an exponentially increasing interval between the retries. The transaction aborts and re-executes if it cannot acquire an object within N attempts.

**Direct Update Systems.** In a direct-update STM system, transactions directly modify an object, rather than a copy.<sup>1, 12, 27</sup> Eliminating the copy potentially is more efficient, since it does not require a clone of each modified object. However, direct-update systems must record the original value of each modified memory location, so the system can restore the location if the transaction aborts. In addition, a direct update STM must prevent a transaction from reading the locations modified by other, uncommitted transactions, thereby reducing the potential for concurrent execution.

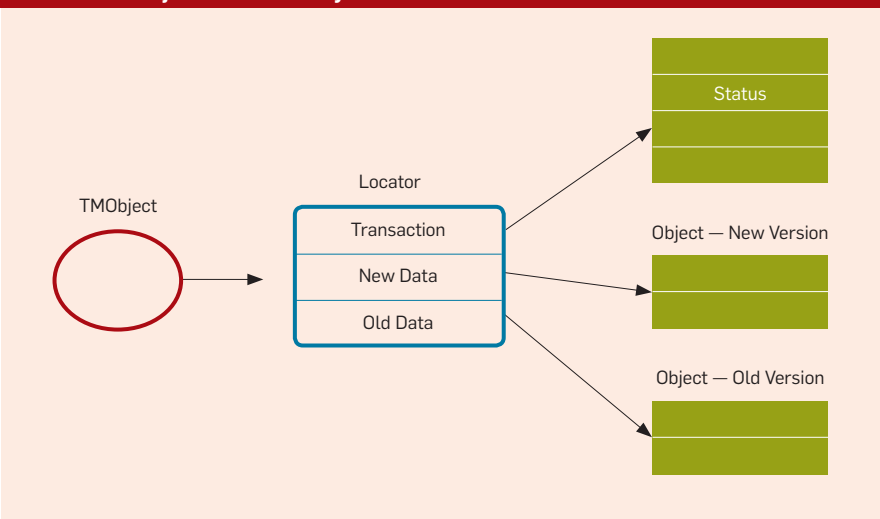
Direct update STM systems also require a lock to prevent multiple transactions from updating an object concurrently. Because of the high cost of fair multiple reader-single writer locks, the systems do not lock a read-only object and instead rely on read-set validation to detect concurrent modification of read-only objects. These lock sets incur the same high overhead cost as in deferred-update systems.

The locks used to prevent multiple writes to a location, however, raise the possibility of stalling many transactions when a transaction is suspended or descheduled while holding locks. Deferred-update STM systems typically use non-blocking data structures, which prevented a failed thread from obstructing other threads. Direct-update STM systems provide similar forward progress guarantees to an application by detecting and aborting failed or blocked threads.

**Hardware Support for Transactional Memory**

The programming effort necessary to exploit parallelism is justified if the new code performs better or is more responsive than sequential code. Even though the performance of recent STM systems scales with the number of processors in a Multicore chip, the overhead of the software systems is significant. Even with compiler optimizations, a STM thread may run two

A transacted object in the DSTM system.



TMOBJECT is a handle for the object. It points to a Locator, which in turn points to the Transaction that opened the object, the original ("old") version of the object, and the transaction's private ("new") clone of the object.

to seven times slower than sequential code.<sup>22, 26</sup>


HTM can improve the performance of STM. While still an active area of research, proposed systems fall into two broad categories: those that accelerate key STM operations and those that implement transactional bookkeeping directly in hardware.

#### Hardware Acceleration for STM.


The primary source of overhead for an STM is the maintenance and validation of read sets. To track a read set, an STM system typically invokes an instrumentation routine at every shared-memory read. The routine registers the object's address and optionally performs early conflict detection by checking the object's version or lock. To validate a transaction, the STM must traverse the read set and ensure each object has no conflicts. This instrumentation can impose a large overhead if the transaction does not perform a large amount of computation per memory access.

The hardware-accelerated STM (HASTM) by Saha et al. was the first system to propose hardware support to reduce the overhead of STM instrumentation.<sup>26</sup> The supplementary hardware allows software to build fast filters that could accelerate the common case of read set maintenance. HASTM provides the STM with two capabilities through per-thread mark bits at the granularity of cache blocks. First, software can check if a mark bit was previously set for a given block of memory and that no other thread wrote to the block since it was marked (conflict detection). Second, software can query if potentially there were writes by other threads to any of the memory blocks that the thread marked (validation).

HASTM proposed implementing mark bits using additional metadata for each block in the per-processor cache of a Multicore chip. The hardware tracks if any marked cache block was invalidated because it was evicted from the cache or written to by another thread. An STM uses mark bits in the following way. The read instrumentation call checks and sets the mark bit for the memory block that contains an object's header. If the mark bit was set, indicating that the transaction previously accessed the object, it is not added to the read set again. To validate the transaction, the STM queries the



**The programming effort necessary to exploit parallelism is justified if the new code performs better or is more responsive than sequential code.**



hardware to determine if any marked cache blocks were invalidated. If not, all objects accessed through instrumentation were private to the thread for the duration of the transaction and no further validation is required. If some marked blocks were invalidated, the STM must rely on software-based validation to check the version numbers or locks for all objects in the read set. This expensive validation step determines if a marked block was evicted because of limited cache capacity or because of true conflicts between concurrent transactions.

HASTM allows transactions to span system events such as interrupts, context switches, and page faults, as the mark bits function only as a filter. If servicing a system event causes the eviction of some marked blocks, a pending transaction can continue its subsequent execution without aborting. The transaction will simply fall back on software validation before it commits. Similarly, HASTM allows a transaction to be suspended and its speculative state inspected by a component such as a garbage collector or a debugger running in another thread.

It is also possible to accelerate STM conflict detection without modifying hardware caches. First-level caches are typically in the critical path of a processor and interact with complex subsystems such as the coherence protocol. Even minor changes to caches can affect the processor's clock frequency and increase design and verification complexity. The signature-accelerated STM (SigTM) proposed by Cao Minh et al. uses hardware signatures to encode pessimistically the read set and write set for software transactions.<sup>22</sup> A hardware Bloom filter outside of the caches computes the signatures.<sup>b</sup> Software instrumentation provides the filters with the addresses of the objects read or written within a transaction. To detect conflicts, hardware in the computer monitors coherence traffic for requests for exclusive accesses to a cache block, which indicates a memory update. The hardware tests if the address in a re-

b. A Bloom filter efficiently represents a superset of the elements in a set and allows fast set membership queries. The use of Bloom filters for dependency detection in thread-level speculation and transactions was originally proposed by Ceze et al.<sup>6</sup>

quest is potentially in a transaction's read or write set by examining the transaction's signatures. If so, the memory reference is a potential conflict and the STM can either abort a transaction or turn to software validation.

Both HASTM and SigTM accelerate read set tracking and validation for STM systems. Nevertheless, the architectures differ. SigTM encodes read set and write sets whose size exceeds the size of private caches. Capacity and conflict misses do not cause software validation, as with HASTM. On the other hand, SigTM uses probabilistic signatures, which never miss a true conflict, but may produce false conflicts due to address aliasing in a Bloom filter. Moreover, the hardware signatures are relatively compact and easy to manipulate, so they can be saved and restored across context switches and other interruptions. In HASTM, the mark bits may be lost if a processor is used to run other tasks. On the other hand, SigTM signatures track physical addresses and their content must be discarded after the virtual page mapping is modified.

Hardware acceleration for read set management has been shown to improve the performance of lock-based, direct-update, and deferred-update STM systems by a factor of two.<sup>22, 26</sup> Additional improvements are possible with hardware mechanisms that target version management for the objects written by the STM.<sup>31</sup> Nevertheless, since most programs read significantly more objects than they write, these performance improvements are small.

**Hardware Transactional Memory.** The interest in full hardware implementation of TM (HTM) dates to the initial two papers on TM by Knight<sup>16</sup> and Herlihy and Moss<sup>13</sup> respectively. HTM systems require no software instrumentation of memory references within transaction code. The hardware manages data versions and tracks conflicts transparently as software performs ordinary read and write accesses. Eliminating instrumentation reduces program overhead and eliminates the need to specialize function bodies so they can be called within and outside of a transaction.

HTM systems rely on a computer's cache hierarchy and the cache coher-

ence protocol to implement versioning and conflict detection. Caches observe all reads and writes issued by a processor, can buffer a significant amount of data, and can be searched efficiently because of their associative organization. All HTMs modify the first-level caches, but the approach extends to higher-level caches, both private and shared. To illustrate the organization and operation of HTM systems, we will describe the TCC architecture in some detail and briefly mention the key attributes of alternative designs.

The Transactional Coherence and Consistency (TCC) system is a deferred-



update HTM that performs conflict detection when a transaction attempts to commit.<sup>21</sup> To track the read set and write set, each cache block is annotated with R and W tracking bits, which are set on the first read or write access to the block from within the transaction. Cache blocks in the write set act as a write buffer and do not propagate the memory updates until the transaction commits.

TCC commits transactions using a two-phase protocol. First, the hardware acquires exclusive access to all cache blocks in the write set using coherence messages. If this step is successful, the transaction is considered validated. Next, the hardware instantaneously resets all W bits in the cache, which atomically commits the updates by this transaction. The new versions of the data are now globally accessible by all processors through the normal

coherence protocol of a Multicore chip. If validation fails, because another processor is also trying to commit a conflicting transaction, the hardware reverts to a software handler, which may abort the transaction or attempt to commit it under a contention management policy. When a transaction commits or aborts, all tracking bits are simultaneously cleared using a gang reset operation. Absent conflicts, multiple transactions may be committing in parallel.

Conflict detection occurs as other processors receive the coherence messages from the committing transaction. Hardware looks up the received block address in the local caches. If the block is in a cache and has its R or W bit set, there is a read-write or a write-write conflict between the committing and the local transaction. The hardware signals a software handler, which aborts the local transaction and potentially retries it after a backoff period.

Similar hardware techniques can support HTM systems with direct memory updates or early detection of conflicts.<sup>23</sup> For direct updates, the hardware transparently logs the original value in a memory block before its first modification by a transaction. If the transaction aborts, the log is used to undo any memory updates. For early conflict detection, the hardware acquires exclusive access to the cache block on the first write and maintains it until the transaction commits. Under light contention, most HTM designs perform similarly. Under heavier contention, deferred updates and late conflict detection lead to fewer pathological scenarios that can be easily handled with a backoff policy.<sup>3</sup>

The performance of an HTM thread is typically within 2%–10% of the performance of non-transactional code. An HTM system can outperform a lock-based STM by a factor of four and the corresponding hardware-accelerated STM by a factor of two.<sup>22</sup> Nevertheless, HTM systems face several system challenges that are not an issue for STM implementations. The caches used to track the read set, write set, and data versions have finite capacity and may overflow on a long transaction. The transactional state in caches is large and is difficult to save and restore at interrupts, context switching, and paging



events. Long transactions may be rare, but they still must execute in a manner that preserves atomicity and isolation. Placing implementation-dependent limits on transaction sizes is unacceptable from a programmer's perspective.

A simple mechanism to handle cache overflows or system events is to ensure the offending transaction executes to completion.<sup>21</sup> When one of these events occurs, the HTM system can update memory directly without tracking the read set, write set, or old data versions. At this point, however, no other transactions can execute, as conflict detection is no longer possible. Moreover, direct memory updates without undo logging preclude the use of explicit abort or retry statements in a transaction.

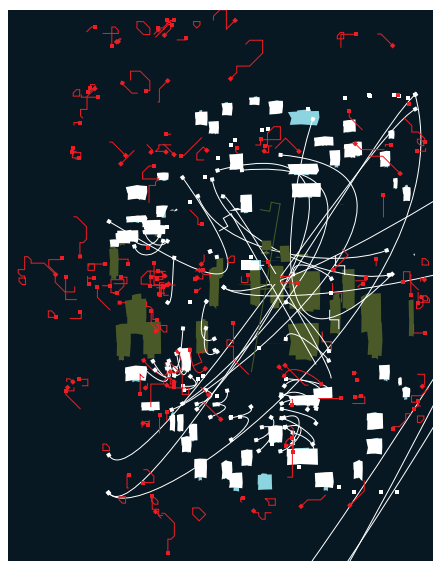
Rajwar et al. proposed Virtualized TM (VTM), an alternative approach that maintains atomicity and isolation for even if a transaction is interrupted by a cache overflow or a system event.<sup>25</sup> VTM maps the key bookkeeping data structures for transactional execution (read set, write set, write buffer or undo-log) to virtual memory, which is effectively unbounded and is unaffected by system interruptions. The hardware caches hold the working set of these data structures. VTM also suggested the use of hardware signatures to avoid redundant searches through structures in virtual memory.

A final technique to address the limitation of hardware resources is to use a hybrid HTM–STM system.<sup>7, 17</sup> A transaction starts in the HTM mode using hardware mechanisms for conflict detection and data versioning. If HTM resources are exceeded, the transaction is rolled back and restarted in the STM mode with additional instrumentation. This approach requires two versions of each function, but it provides good performance for short transactions. A challenge for hybrid systems is to detect conflict between concurrently HTM and STM transactions.

**Hardware/Software Interface for Transactional Memory.** Hardware designs are optimized to make the common case fast and reduce the cost of correctly handling rare events. Processor vendors will follow this principle in introducing hardware support for transactional execution. Initial systems are likely to devote modest hardware resources to TM. As more applications

use transactions, more aggressive hardware designs, including full-featured HTM systems, may become available.

Regardless of the amount of hardware used for TM, it is important that HTM systems provide functionality that is useful in developing practical programming models and execution environments. A significant amount of HTM research has focused on hardware/software interfaces that can support rich software features. McDonald et al. suggested four interface mechanisms for HTM systems.<sup>20</sup> The first mechanism is a two-phase commit protocol that architecturally separates transaction validation from commit-



ting its updates to memory. The second mechanism is transactional handlers that allow software to interface on significant events such as conflict detection, commit, or abort. Sriraman et al. suggest alert-on-update, a similar mechanism that invokes a software handler when HTM hardware detects conflicts or experiences overflows.<sup>31</sup> The third mechanism is support for closed and open-nested transactions. Open nesting allows software to interrupt the currently executing transaction and run some service code (for example, a system call) with independent atomicity and isolation guarantees. Other researchers suggest it is sufficient to suspend HTM transactions to service system calls and I/O operations.<sup>24, 32</sup> Nevertheless, if the service code uses transactions to access shared data in memory, the requirements of transaction pausing are

not significantly different from those of open-nested transactions. Finally, both McDonald et al. and Sriraman et al. propose multiple types of load and store instructions that allow compilers to distinguish accesses to thread-private, immutable, or idempotent data from accesses to truly shared data. By providing such mechanisms, HTM systems can support software features ranging from conditional synchronization and limited I/O within transactions<sup>5,32</sup> to high-level concurrency models that avoid transaction aborts on memory conflicts if the application-level semantics are not violated.<sup>4</sup>

### Open Issues

Beyond the implementation issues discussed here, TM faces a number of challenges that are the subject of active research. One serious difficulty with optimistic TM is that a transaction that executed an I/O operation may roll back at a conflict. I/O in this case consists of any interaction with the world outside of the control of the TM system. If a transaction aborts, its I/O operations should roll back as well, which may be difficult or impossible to accomplish in general. Buffering the data read or written by a transaction permits some rollbacks, but buffering fails in simple situations, such as a transaction that writes a prompt and then waits for user input. A more general approach is to designate a single privileged transaction that runs to completion, by ensuring it triumphs over all conflicting transactions. Only the privileged transaction can perform I/O (but the privilege can be passed between transactions), which unfortunately limits the amount of I/O a program can perform.

Another major issue is strong and weak atomicity. STM systems generally implement weak atomicity, in which non-transactional code is not isolated from code in transactions. HTM systems, on the other hand, implement strong atomicity, which provides a more deterministic programming model in which non-transactional code does not affect the atomicity of a transaction. This difference presents several problems. Beyond the basic question of which model is a better basis for writing software, the semantic differences makes it difficult to develop software that runs on both types of systems.

The least common denominator is the weak model, but erroneous programs will produce divergent results on different systems. An alternative viewpoint is that unsynchronized data accesses between two threads is generally an error, and if only one thread is executing a transaction, then there is insufficient synchronization between the threads. Therefore, the programming language, tools, runtime system, or hardware should prevent or detect unsynchronized sharing between transactional and non-transactional code, and a programmer should fix the defect.

Weakly atomic systems also face difficulties when an object is shared between transactional and non-transactional code.<sup>30</sup> Publication occurs when a thread makes an object visible to other threads (for example, by adding it to a global queue) and privatization occurs when a thread removes an object from the global shared space. Private data should be manipulatable outside of a transaction without synchronization, but an object's transition between public and private must be coordinated with the TM system, lest it attempt to roll back an object's state while another thread assumes it has sole, private access to the data.

Finally, TM must coexist and interoperate with existing programs and libraries. It is not practical to require programmers to start afresh and acquire a brand new set of transactional libraries to enjoy the benefits of TM. Existing sequential code should be able to execute correctly in a transaction, perhaps with a small amount of annotation and recompilation. Existing parallel code that uses locks and other forms of synchronization, must continue to operate properly, even if some threads are executing transactions.

## Conclusion

Transactional memory by itself is unlikely to make Multicore computers readily programmable. Many other improvements to programming languages, tools, runtime systems, and computer architecture are also necessary. TM, however, does provide a time-tested model for isolating concurrent computations from each other. This model raises the level of abstraction for reasoning about concurrent tasks and helps avoid many insidious parallel

programming errors. However, many aspects of the semantics and implementation of TM are still the subject of active research. If these difficulties can be resolved in a timely fashion, TM will likely become a central pillar of parallel programming. □

## References

- Adl-Tabatabai, A.R., Lewis, B.T., Menon, V., Murphy, B.R., Saha, B., and Shpeisman, T. Compiler and runtime support for efficient software transactional memory. In *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Ottawa, Ontario, Canada, 2006). ACM, NY, 26–37.
- Blundell, C., Lewis, E.C., and Martin, M.M.K. Subtleties of transactional memory atomicity semantics. *IEEE Computer Architecture Letters* 5 (Nov. 2006).
- Bobba, J., Moore, K.E., Volos, H., Yen, L., Hill, M.D., Swift, M.M., and Wood, D.A. Performance pathologies in hardware transactional memory. In *Proceedings of the 34th International Symposium on Computer Architecture* (San Diego, CA, 2007). ACM, NY, 81–91.
- Carlstrom, B. D., McDonald, A., Carbin, M., Kozyrakis, C., and Olukotun, K. Transactional collection classes. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (San Jose, CA, 2007). ACM, NY, 56–67.
- Carlstrom, B.D., McDonald, A., Chafi, H., Chung, J., Minh, C.C., Kozyrakis, C., and Olukotun, K. The Atomos transactional programming language. In *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Ottawa, Ontario, Canada, 2006). ACM, NY, 1–13.
- Ceze, L., Tuck, J., Torrellas, J., and Cascaval, C. Bulk disambiguation of speculative threads in multiprocessors. In *Proceedings of the 33rd International Symposium on Computer Architecture* (Boston, MA, 2006). ACM, NY, 227–238.
- Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., and Nussbaum, D. Hybrid transactional memory. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, 2006). ACM, NY, 336–346.
- Gray, J. and Reuter, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, 1992.
- Grossman, D. The transactional memory / garbage collection analogy. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications* (Montreal, Canada, 2007). ACM, NY, 695–706.
- Guerraoui, R., Herlihy, M., and Pochon, B. Polymorphic contention management. In *Proceedings of the 19th International Symposium on Distributed Computing* (Krakow, Poland, 2005). Springer Verlag, 303–323.
- Harris, T., Marlow, S., Peyton-Jones, S., and Herlihy, M. Composable memory transactions. In *Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Chicago, IL, 2005). ACM, NY, 48–60.
- Harris, T., Plesko, M., Shinnar, A., and Tarditi, D. Optimizing memory transactions. In *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Ottawa, Ontario, Canada, 2006). ACM, NY, 14–25.
- Herlihy, M. and Moss, J.E.B. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th International Symposium on Computer Architecture*. ACM, 1993, 289–300.
- Herlihy, M., Luchangco, V., Moir, M., and Scherer III, W.N. Software transactional memory for dynamically sized data structures. In *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing* (Boston, MA, 2003), 92–101.
- Isard, M., and Birrell, A. Automatic mutual exclusion. In *Proceeding of the Usenix 11th Workshop on Hot Topics in Operating Systems* (San Diego, CA, 2007).
- Knight, T.F. An architecture for mostly functional languages. In *Proceedings of the 1986 ACM Lisp and Functional Programming Conference*. ACM, NY.
- Kumar, S., Chu, M., Hughes, C.J., Kundu, P., and Nguyen, A. Hybrid transactional memory. In *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, NY, 2006, 209–220.
- Larus, J.R. and Rajwar, R. *Transactional Memory*. Morgan & Claypool, 2006.
- Lomet, D.B. Process structuring, synchronization, and recovery using atomic actions. In *Proceedings of the ACM Conference on Language Design for Reliable Software* (Raleigh, NC, 1977). ACM, NY, 128–137.
- McDonald, A., Chung, J., Brian, D.C., Minh, C.C., Chafi, H., Kozyrakis, C., and Olukotun, K. Architectural semantics for practical transactional memory. In *Proceedings of the 33rd International Symposium on Computer Architecture*. ACM, 2006, 53–65.
- McDonald, A., Chung, J., Chafi, H., Cao Minh, C., Carlstrom, B.D., Hammond, L., Kozyrakis, C., and Olukotun, K. Characterization of TCC on chip-multiprocessors. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*. (St Louis, MO, 2005). IEEE, 63–74.
- Minh, C. C., Trautmann, M., Chung, J., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C., and Olukotun, K. An effective hybrid transactional memory system with strong isolation guarantees. In *Proceedings of the 34th International Symposium on Computer Architecture* (San Diego, CA, 2007) ACM, NY, 69–80.
- Moore, K.E., Bobba, J., Moravan, M.J., Hill, M.D., and Wood, D.A. LogTM: Log-based transactional memory. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture* (Austin, TX, 2006). IEEE, 254–265.
- Moravan, M.J., Bobba, J., Moore, K.E., Yen, L., Hill, M.D., Liblit, B., Swift, M.M., and Wood, D.A. Supporting Nested Transactional Memory in LogTM. *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, 2006). ACM, NY, 359–370.
- Rajwar, R., Herlihy, M., and Lai, K. Virtualizing transactional memory. In *Proceedings of the 32nd International Symposium on Computer Architecture* (Madison, WI, 2005). ACM, NY, 494–505.
- Saha, B., Adl-Tabatabai, A. R., and Jacobson, Q. Architectural support for software transactional memory. In *Proceedings of the 39th International Symposium on Microarchitecture* (Orlando, FL, 2006). IEEE, 185–196.
- Saha, B., Adl-Tabatabai, A.R., Hudson, R.L., Minh, C.C., and Hertzberg, B. McRT-STM: A high performance software transactional memory system for a multi-core runtime. In *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2006). ACM, NY, 187–197.
- Scherer III, W.N., and Scott, M.L. Advanced contention management for dynamic software transactional memory. In *Proceedings of the Twenty-fourth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (Las Vegas, NV, 2005). ACM Press, 240–248.
- Shavit, N. and Touitou, D. Software transactional memory. In *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing* (Ottawa, Canada, 1995). ACM, NY, 204–213.
- Shpeisman, T., Menon, V., Adl-Tabatabai, A.-R., Balensiefer, S., Grossman, D., Hudson, R.L., Moore, K.F., and Saha, B. Enforcing isolation and ordering in STM. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Diego, CA, 2007). ACM, NY, 78–88.
- Shriraman, A., Spear, M.F., Hossain, H., Marathe, V.J., Dwarkadas, S., and Scott, M.L. An integrated hardware-software approach to flexible transactional memory. In *Proceedings of the 34th International Symposium on Computer Architecture* (San Diego, CA, 2007). ACM, NY, 104–115.
- Zilles, C. and Baugh, L. Extending hardware transactional memory to support nonbusy waiting and nontransactional actions. In *Proceedings of the First ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing* (Ottawa, Canada, 2006). ACM, NY.

**James Larus** (larus@microsoft.com) is a research area manager at Microsoft Research, Redmond, WA.

**Christos Kozyrakis** (christosee.stanford.edu) is an assistant professor of electrical engineering and computer science at Stanford University, Stanford, CA.

©2008 ACM0001-0782/08/0700 \$5.00

---

P. 90

**Technical  
Perspective  
Computer Science  
Takes On  
Molecular Dynamics**

By Bob Colwell

P. 91

## Anton, a Special-Purpose Machine for Molecular Dynamics Simulation

By David E. Shaw, Martin M. Deneroff, Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J.P. Grossman, C. Richard Ho, Douglas J. Ierardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang

---

P. 98

**Technical  
Perspective  
The Physical Side  
of Computing**

By Feng Zhao

P. 99

## The Emergence of a Networking Primitive in Wireless Sensor Networks

By Philip Levis, Eric Brewer, David Culler, David Gay, Sam Madden, Neil Patel, Joe Polastre, Scott Shenker, Robert Szewczyk, and Alec Woo



# Technical Perspective

## Computer Science Takes on Molecular Dynamics

By Bob Colwell

PUT THIS ON your to-do list: read the following paper by researcher David Shaw and colleagues that describes their Anton molecular dynamics (MD) engine. Shaw's Anton engine applies leading-edge computer science concepts to the biologically crucial problem of modeling molecular interactions. In an era when much of our most advanced computer technology is spent creating ever more horrible creatures that we can shoot ever bigger virtual holes in, the idea of productively using this technology to explore nature at its most up-close-and-personal is both exciting and reassuring.

The nature of the computational problem Anton aims to solve, and the unique aspects of the resulting design, are fascinating peeks into a corner of the computer design space we seldom get to visit—even though each of us is a biological machine that relies on the correct functioning of molecular mechanisms. When diseases cause these mechanisms to go awry, medical researchers try to infer the causes and possible remedies from very indirect and error-prone evidence, as they lack direct means of measuring or simulating the molecular underpinnings. David Shaw calls his new instrument a “computational microscope,” and if successful it stands to make the same kind of game-changing impact that Anton van Leeuwenhoek's original optical microscope once did. (Shaw's machine was named in van Leeuwenhoek's honor.)

To appreciate what Shaw's machine is attempting, consider a system containing a realistic protein molecule together with a few layers of water molecules, which might together encompass tens of thousands of atoms. If calculation of the force between any two atoms takes 10 computer operations, then the total ops required per time step would be  $(10^4 \text{ atoms}) \times (10^4 \text{ atoms}) \times 10 \text{ ops/atom} = 10^9 \text{ ops}$ . Time slices are on the order of femtoseconds

( $10^{-15}$  seconds), and simulations must run for milliseconds ( $10^{-3}$  seconds) to capture the biology being modeled. So we'll need to run those  $10^9$  ops for  $10^{12}$  slices to reach a simulated millisecond—that's 31,000 years. We need six orders of magnitude speedup, roughly three orders of magnitude beyond today's fastest supercomputers.


But even if you weren't a biological unit with a vested interest in this effort, you could still appreciate the Anton design from a computer system perspective. General-purpose computer systems aspire to run everything well, but no one thing spectacularly well. Anton is designed to run a specific molecular dynamics workload spectacularly well. While a well-designed general system can bottleneck 100 different ways on 100 different benchmarks, Anton must try, in essence, to bottleneck everywhere, all at once, on its one workload.

This balancing act must be attempted in the face of imperfect knowledge of that one workload. For example, electrostatic interactions between two atoms that aren't sharing any electrons are considered to be well understood, and are the most numerous, so Anton applies very specific, very parallel, and very inflexible hardware to handling them. Less is known about the infrequent bonded interactions, so those calculations are allocated to a much more flexible subsystem that will allow experimentation with various “force field” models and algorithms.

What might go wrong with the Anton effort? Subtle errors arising from the class of force fields that Anton is designed to handle efficiently may accumulate over the extremely long MD runtimes; in a custom machine with no operational experience, soft errors could strike much more often and substantially slow its performance; quantum effects may turn out to be necessary, beyond the classical force field being modeled here; some clever graduate student may come up with a

software-based approach that reduces Anton's two-orders-of-magnitude performance advantage to only one (which might no longer be enough to justify its hardware expenditure). Or Anton might become a victim of its own success if early learnings point to much better (and much different) MD algorithms that no longer fit well into Anton's overall structure.

But what if things go *right*? Benoit Roux, an MD researcher now at the University of Chicago, said that as soon as Anton has delivered its first verified scientific result he will want an engine of his own, and so will everyone in the entire MD field. Roux points out that molecular biologists must normally have “elaborate strategies to prevent fooling themselves” in their macro-scale experiments. With Anton, “we'll be able to do insane things with unknown problems and two weeks later we'll discover how the molecules actually move. ... Anton will revolutionize molecular biology.”

It is not often that a science reaches a clear tipping point—when it advances very quickly, virtually exploding into a new shape and venue. Our own field of computing has done that several times. Many physicists expect this of the Large Hadron Collider currently being completed in Europe. Shaw and his coworkers are attempting nothing less in the field of molecular dynamics. As a computing professional, I am proud of their efforts, I salute their attempt to drive an extremely important basic science forward, and I heartily recommend their paper. 

---

**Bob Colwell** (bob.colwell@comcast.net), former chief architect of Intel's IA-32 microprocessors, is now an independent consultant.

# Anton, a Special-Purpose Machine for Molecular Dynamics Simulation

By David E. Shaw, Martin M. Deneroff, Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J.P. Grossman, C. Richard Ho, Douglas J. Jerardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang

## Abstract

**The ability to perform long, accurate molecular dynamics (MD) simulations involving proteins and other biological macromolecules could in principle provide answers to some of the most important currently outstanding questions in the fields of biology, chemistry, and medicine. A wide range of biologically interesting phenomena, however, occur over timescales on the order of a millisecond—several orders of magnitude beyond the duration of the longest current MD simulations.**

We describe a massively parallel machine called Anton, which should be capable of executing millisecond-scale classical MD simulations of such biomolecular systems. The machine, which is scheduled for completion by the end of 2008, is based on 512 identical MD-specific ASICs that interact in a tightly coupled manner using a specialized high-speed communication network. Anton has been designed to use both novel parallel algorithms and special-purpose logic to dramatically accelerate those calculations that dominate the time required for a typical MD simulation. The remainder of the simulation algorithm is executed by a programmable portion of each chip that achieves a substantial degree of parallelism while preserving the flexibility necessary to accommodate anticipated advances in physical models and simulation methods.

## 1. INTRODUCTION

Molecular dynamics (MD) simulations can be used to model the motions of molecular systems, including proteins, cell membranes, and DNA, at an atomic level of detail. A sufficiently long and accurate MD simulation could allow scientists and drug designers to visualize for the first time many critically important biochemical phenomena that cannot currently be observed in laboratory experiments, including the “folding” of proteins into their native three-dimensional structures, the structural changes that underlie protein function, and the interactions between two proteins or between a protein and a candidate drug molecule. Such simulations could answer some of the most important open questions in the fields of biology and chemistry, and have the potential to make substantial contributions to the process of drug development.

Many of the most important biological processes occur over timescales on the order of a millisecond. MD simulations on this timescale, however, lie several orders of magnitude beyond the reach of current technology; only a few MD

runs have thus far reached even a microsecond of simulated time, and the vast majority have been limited to the nanosecond timescale. Millisecond-scale simulations of a biomolecular system containing tens of thousands of atoms will in practice require that the forces exerted by all atoms on all other atoms be calculated in just a few microseconds—a process that must be repeated on the order of  $10^{12}$  times. These requirements far exceed the current capabilities of even the most powerful commodity clusters or general-purpose scientific supercomputers.

This paper describes a specialized, massively parallel machine, named Anton, that is designed to accelerate MD simulations by several orders of magnitude, bringing millisecond-scale simulations within reach for molecular systems involving tens of thousands of atoms. The machine, which is scheduled for completion by the end of 2008, will comprise 512 processing nodes in its initial configuration, each containing a specialized MD computation engine implemented as a single ASIC. The molecular system to be simulated is decomposed spatially among these processing nodes, which are connected through a specialized high-performance network to form a three-dimensional torus. Anton’s expected performance advantage stems from a combination of MD-specific hardware that achieves a very high level of arithmetic density and novel parallel algorithms that enhance scalability by reducing both intra- and inter-chip communication. Figure 1 is a photograph of one of the first Anton ASICs.

In designing Anton and its associated software, we have attempted to attack a somewhat different problem than the ones addressed by several other projects that have deployed significant computational resources for MD simulations. The Folding@Home project,<sup>16</sup> for example, has obtained a number of significant and interesting scientific results by using as many as 250,000 PCs (made available over the Internet by volunteers) to simulate a very large number of separate molecular trajectories, each of which is limited to the timescale accessible on a single PC. While a great deal can be learned from a large number of independent MD trajectories, many other important problems require the examination of a single, very long trajectory—the principal task for which Anton is designed. Other projects, such as FASTRUN,<sup>6</sup> MDGRAPE,<sup>22</sup> and MD Engine,<sup>23</sup> have produced special-purpose hardware to accelerate the most computationally expensive elements of an MD simulation. Such hardware reduces the cost of MD simulations, particularly for large molecular systems, but Amdahl’s law and communication bottlenecks prevent the efficient use of enough such chips in parallel

**Figure 1: Anton ASIC.** One of the first Anton ASICs, which arrived in January 2008.



to extend individual simulations beyond microsecond timescales.

Anton is named after Anton van Leeuwenhoek, whose contributions to science and medicine we hope to emulate in our own work. In the 17th century, van Leeuwenhoek, often referred to as the “father of microscopy,” built high-precision optical instruments that allowed him to visualize for the first time an entirely new biological world that had previously been unknown to the scientists of his day. We view Anton (the machine) as a sort of “computational microscope.” To the extent that we and other researchers are able to increase the length of MD simulations, we would hope to provide contemporary biological and biomedical researchers with a tool for understanding organisms and their diseases on a still smaller length scale.

## 2. MD COMPUTATION ON ANTON

An MD computation simulates the motion of a collection of atoms (the *chemical system*) over a period of time according to the laws of classical physics.<sup>1</sup> Time is broken into a series of discrete *time steps*, each representing a few femtoseconds of simulated time. A time step has two major phases. *Force calculation* computes the force on each particle due to other particles in the system. *Integration* uses the net force on each particle to update that particle’s position and velocity.

### 2.1. Force calculation

Interatomic forces are calculated based on a *molecular mechanics force field* (or simply *force field*), which models the forces on each atom as a function of the spatial coordinates of all atoms. In commonly used biomolecular force fields,<sup>9, 11, 15</sup> the forces consist of three components: *bond forces*, involving groups of atoms separated by no more than three covalent bonds; *van der Waals forces*, computed between pairs of atoms separated by less than some cutoff radius (usually chosen between 5 and 15 Å); and *electrostatic forces*, which are the most computation-

ally intensive as they must be computed between all pairs of atoms.

Anton uses the *k*-space Gaussian split Ewald method (*k*-GSE)<sup>18</sup> to reduce the computational workload associated with the electrostatic interactions. This method divides the electrostatic force calculation into two components. The first decays rapidly with particle separation and is computed directly for all particle pairs separated by less than a cutoff radius. We refer to this contribution, together with the van der Waals interactions, as *range-limited interactions*. The second component, *long-range interactions*, decays more slowly, but can be computed efficiently by mapping charge from particles to a regular mesh (*charge spreading*), taking the fast Fourier transform (FFT) of the mesh charges, multiplying by an appropriate function in Fourier space, performing an inverse FFT, and then computing forces on the particles from the resulting mesh values (*force interpolation*).

To parallelize range-limited interactions, our machine uses an algorithm we developed called the *NT method*.<sup>19</sup> The NT method achieves both asymptotic and practical reductions in required interprocessor communication bandwidth relative to traditional parallelization methods. It is one of a number of *neutral territory methods* that employ a spatial assignment of particles to nodes, but that often compute the interaction between two particles using a node on which neither particle resides.<sup>4, 7, 10, 14, 17, 21</sup>

### 2.2. Integration

The integration phase uses the results of force calculation to update atomic positions and velocities, numerically integrating a set of ordinary differential equations describing the motion of the atoms. The numerical integrators used in MD are nontrivial for several reasons. First, the integration algorithm and the manner in which numerical issues are handled can have a significant effect on accuracy. Second, some simulations require the integrator to calculate and adjust global properties such as temperature and pressure. Finally, one can significantly accelerate most simulations by incorporating constraints that eliminate the fastest vibrational motions. For example, constraints are typically used to fix the lengths of bonds to all hydrogen atoms and to hold water molecules rigid.

## 3. WHY SPECIALIZED HARDWARE?

A natural question is whether a specialized machine for molecular simulation can gain a significant performance advantage over general-purpose hardware. After all, history is littered with the corpses of specialized machines, spanning a huge gamut from Lisp machines to database accelerators. Performance and transistor count gains predicted by Moore’s law, together with the economies of scale behind the development of commodity processors, have driven a history of general-purpose microprocessors outpacing special-purpose solutions. Any plan to build specialized hardware must account for the expected exponential growth in the capabilities of general-purpose hardware.

We concluded that special-purpose hardware is warranted in this case because it leads to a much greater improve-



ment in absolute performance than the expected speedup predicted by Moore's law over our development time period, and because we are currently at the cusp of simulating timescales of great biological significance. We expect Anton to run simulations over 1000 times faster than was possible when we began this project. Assuming that transistor densities continue to double every 18 months and that these increases translate into proportionally faster processors and communication links, one would expect approximately a tenfold improvement in commodity solutions over the five-year development time of our machine (from conceptualization to bring-up). We therefore expect that a specialized solution will be able to access biologically critical millisecond timescales significantly sooner than commodity hardware.

To simulate a millisecond within a couple of months, we must complete a time step every few microseconds, or every few thousand clock ticks. The sequential dependence of successive time steps in an MD simulation makes speculation across time steps extremely difficult. Fortunately, specialization offers unique opportunities to accelerate an individual time step using a combination of architectural features that reduce both computational latency and communication latency.

For example, we reduced computational latency by designing:

- Dedicated, specialized hardware datapaths and control logic to evaluate the range-limited interactions and to perform charge spreading and force interpolation. In addition to packing much more computational logic on a chip than is typical of general-purpose architectures, these pipelines use customized precision for each operation.
- Specialized, yet programmable, processors to compute bond forces and the FFT and to perform integration. The instruction set architecture (ISA) of these processors is tailored to the calculations they perform. Their programmability provides flexibility to accommodate various force fields and integration algorithms.
- Dedicated support in the memory subsystem to accumulate forces for each particle.

We reduced communication latency by designing:

- A low-latency, high-bandwidth network, both within an ASIC and between ASICs, that includes specialized routing support for common MD communication patterns such as multicast and compressed transfers of sparse data structures.
- Support for choreographed “push”-based communication. Producers send results to consumers without the consumers having to request the data beforehand.
- A set of autonomous direct memory access (DMA) engines that offload communication tasks from the computational units, allowing greater overlap of communication and computation.
- Admission control features that prioritize packets carrying certain algorithm-specific data types.

We balance our design very differently from a general-purpose supercomputer architecture. Relative to other high-performance computing applications, MD uses much communication and computation but surprisingly little memory. The entire architectural state of an MD simulation of 25,000 particles, for example, is just 1.6 MB, or 3.2 KB per node in a 512-node system. We exploit this property by using only SRAMs and small L1 caches on our ASIC, with all code and data fitting on-chip in normal operation. Rather than spending silicon area on large caches and aggressive memory hierarchies, we instead dedicate it to communication and computation.

It is serendipitous that the most computationally intensive parts of MD—in particular, the electrostatic interactions—are also the most well established and unlikely to change as force field models evolve, making them particularly amenable to hardware acceleration. Dramatically accelerating MD simulation, however, requires that we accelerate more than just an “inner loop.”

Calculation of electrostatic and van der Waals forces accounts for roughly 90% of the computational time for a representative MD simulation on a single general-purpose processor. Amdahl's law states that no matter how much we accelerate this calculation, the remaining computations, left unaccelerated, would limit our maximum speedup to a factor of 10. Hence, we dedicated a significant fraction of silicon area to accelerating other tasks, such as bond force computation, constraint computation, and velocity and position updates, incorporating programmability as appropriate to accommodate a variety of force fields and integration methods.

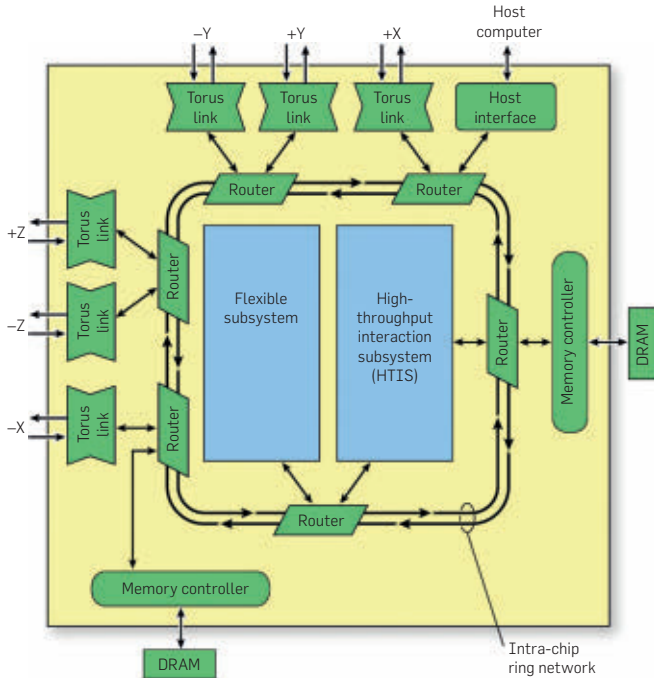
## 4. SYSTEM ARCHITECTURE

The building block of the system is a node, depicted in Figure 2. Each node comprises an MD-specific ASIC, attached DRAM, and six ports to the system-wide interconnection network. Each ASIC has four major subsystems, which are described briefly in this section. The nodes, which are logically identical, are connected in a three-dimensional torus topology (which maps naturally to the periodic boundary conditions frequently used in MD simulations). The initial version of Anton will be a 512-node torus with eight nodes in each dimension, but our architecture also supports larger and smaller toroidal configurations. The ASICs are clocked at a modest 400 MHz, with the exception of one double-clocked component in the *high-throughput interaction subsystem* (HTIS), discussed in the following section.

### 4.1. High-throughput interaction subsystem

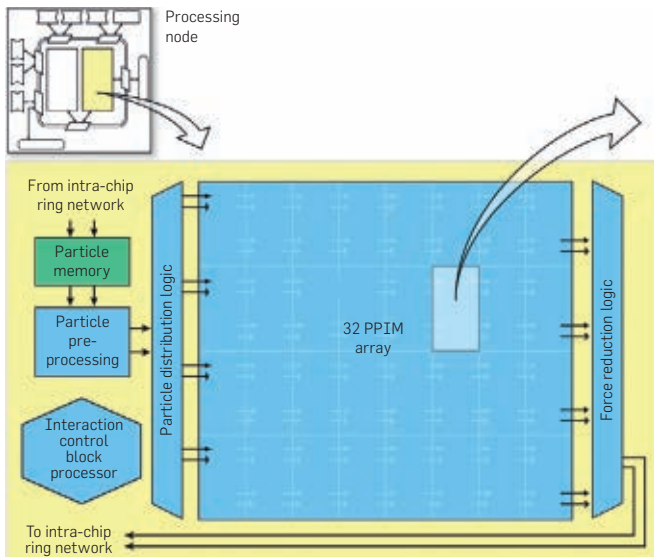
The HTIS calculates range-limited interactions and performs charge spreading and force interpolation. The HTIS, whose internal structure is shown in Figure 3, applies massive parallelism to these operations, which constitute the bulk of the calculation in MD. It provides tremendous arithmetic throughput using an array of 32 *pairwise point interaction modules* (PPIMs) (Figure 3), each of which includes a force calculation pipeline that runs at 800 MHz and is capable of computing the combined electrostatic and van der

**Figure 2: Anton processing node.** The HTIS performs the most demanding calculations in an MD simulation. The flexible subsystem performs the remaining MD calculations, coordinates MD time step activity, and manages housekeeping tasks.



Waals interactions between a pair of atoms at every cycle. This 26-stage pipeline (Figure 4) includes adders, multipliers, function evaluation units, and other specialized datapath elements. Inside this pipeline, we use customized numerical precisions: functional unit width varies across the different pipeline stages but still produces a sufficiently accurate 32-bit result.

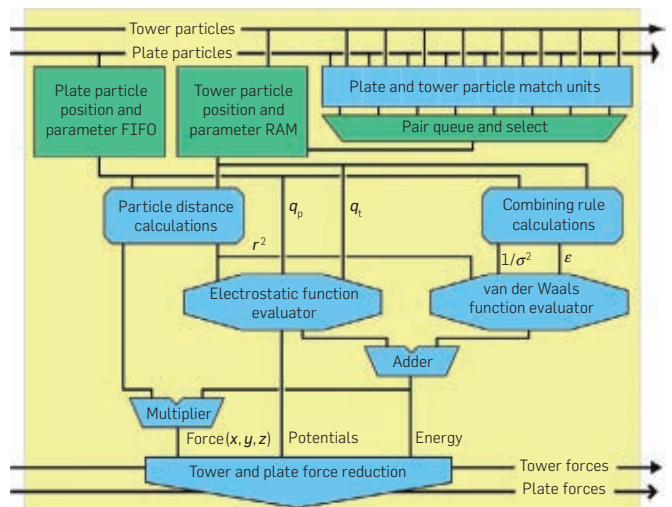
**Figure 3: High-throughput interaction subsystem.** The HTIS comprises an array of 32 PPIMs and an embedded control processor to coordinate the distribution of particles to the PPIM array.



In order to keep the pipelines busy with useful computation, the remainder of the HTIS must determine pairs of atoms that need to interact, feed them to the pipelines, and aggregate the pipelines' outputs. This proves a formidable challenge given communication bandwidth limitations between ASICs, between the HTIS and other subsystems on the same ASIC, and between pipelines within the HTIS. We address this problem using an architecture tailored for *direct product selection reduction operations* (DPSRs), which take two sets of points and perform computation proportional to the product of the set sizes but only require input and output volume proportional to the sum of their sizes. The HTIS considers interactions between all atoms in a region called the *tower* and all atoms in a region called the *plate*. Each atom in the tower is assigned to one PPIM, while each atom in the plate streams by all the PPIMs. Eight *match units* in each PPIM perform several tests, including a low-precision distance check, to determine which pairs of plate and tower particles are fed to the force calculation pipeline. Because the HTIS is a streaming architecture, with no feedback in its computational path, it is simple to scale the PPIM array to any number of PPIMs. The HTIS also includes an *interaction control block* processor, which controls the flow of data through the HTIS. More detail about the HTIS and about DPSR operations can be found in the proceedings of this year's HPCA conference.<sup>13</sup>

The PPIMs are the most hard-wired component of our architecture, reflecting the fact that they handle the most computationally intensive parts of the MD calculation. That said, even the PPIMs include programmability where we anticipate potential future changes to force fields. For instance, the functional forms for van der Waals and electrostatic interactions are specified using SRAM look-up tables, whose contents are determined at runtime.

**Figure 4: PPIM detail.** This figure gives a sense of the numerical calculation units in a PPIM. The top portion of the figure shows the match units and particle memories. The lower portion shows the general structure of the force calculation pipelines.



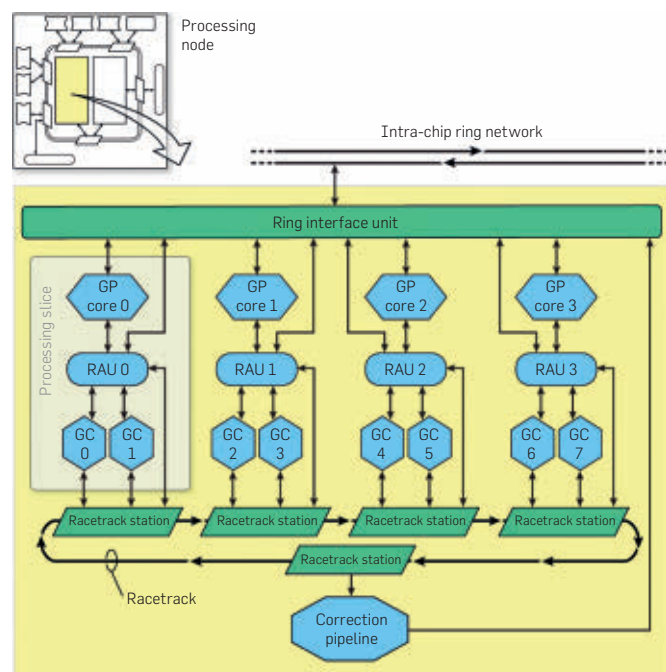
## 4.2. Flexible subsystem

The *flexible subsystem* controls the ASIC and handles all other computations, including the bond force calculations, the FFT, and integration. Figure 5 shows the components of the flexible subsystem. Four identical *processing slices* form the core of the flexible subsystem. Each slice comprises a general-purpose core with its caches, a *remote access unit* (RAU) that performs autonomous data transfers, and two *geometry cores* (GCs), which are programmable cores that perform most of the flexible subsystem's computation. The RAU is a programmable data transfer engine that enables the flexible subsystem to participate in "push" communication, both of-flooding messages sent from the processor cores and tracking incoming messages to determine when work is ready to be done. Each GC is a dual-issue, statically scheduled, 4-way SIMD processor with pipelined multiply accumulate support and instruction set extensions to support common MD calculations. Other components of the flexible subsystem include a correction pipeline, which computes force correction terms; a racetrack, which serves as a local, internal interconnect for the flexible subsystem components; and a ring interface unit, which allows the flexible subsystem components to transfer packets to and from the communication subsystem. More detail about the flexible subsystem is given in a second paper at this year's HPCA conference.<sup>12</sup>

## 4.3. Communication subsystem

The *communication subsystem* provides high-speed, low-latency communication both between ASICs and among

**Figure 5: Flexible subsystem.** It is a collection of four identical processing slices (one of which is indicated by a box at the left) and a correction pipeline unit. The processing slices communicate with each other and with the correction pipeline via the racetrack. The various components communicate with the intra-chip communication ring via the ring interface unit shown at the top of the figure.



the subsystems within an ASIC. Between chips, each torus link provides 5.3 GB/s full-duplex communication with a hop latency around 50 ns. Within a chip, two 256-bit, 400 MHz communication rings link all subsystems and the six inter-chip torus ports. The communication subsystem supports efficient multicast, provides flow control, and provides class-based admission control with rate metering. The communication subsystem also allows access to an external host computer system for input and output of simulation data.

## 4.4. Memory subsystem

The *memory subsystem* provides access to the ASIC's attached DRAM. In addition to basic memory read/write access, the memory subsystem supports accumulation and synchronization. Special memory write operations numerically add incoming write data to the contents of the memory location specified in the operation. These operations implement force, energy, potential, and spread charge accumulations, reducing the computation and communication load on the flexible subsystem. By taking advantage of the attached DRAM, Anton will be able to simulate chemical systems with billions of atoms.

## 5. PERFORMANCE AND ACCURACY MEASUREMENTS

In this section, we show that the performance of Anton significantly exceeds that of other MD platforms, and that Anton is capable of performing simulations of high numerical accuracy. Because we do not yet have a working 512-node segment, performance estimates for our machine come from our performance simulator. The cycle fidelity of this simulator varies across components, but we expect overall fidelity better than  $\pm 20\%$ .

### 5.1. Performance comparison

We compare the performance of various MD platforms in terms of simulation rate (nanoseconds of simulated time per day of execution) on a particular chemical system. In this section and in Section 5.2, we use a system with 23,558 atoms in a cubic box measuring 62.2 Å on a side. This system represents dihydrofolate reductase (DHFR), a protein targeted by various cancer drugs, surrounded by water.

The highest-performing MD codes achieve a simulation rate of a few nanoseconds per day for DHFR on a single state-of-the-art commodity processor core.<sup>8</sup> Existing multiprocessor machines with high-performance interconnects achieve simulation rates up to a few hundred nanoseconds per day using many hundreds or thousands of processor cores.<sup>2,3,5</sup>

We expect a 512-node Anton system to achieve a simulation rate of approximately 14,500 nanoseconds per day for DHFR, enabling a millisecond simulation in just over two months. While the performance of general-purpose machines will undoubtedly continue to improve, Anton's performance advantage over other MD platforms significantly exceeds the speedup predicted by Moore's law over the next few years. A more detailed performance comparison of Anton and other MD platforms is given in the proceedings of last year's ISCA conference.<sup>20</sup>



## 5.2. Accuracy

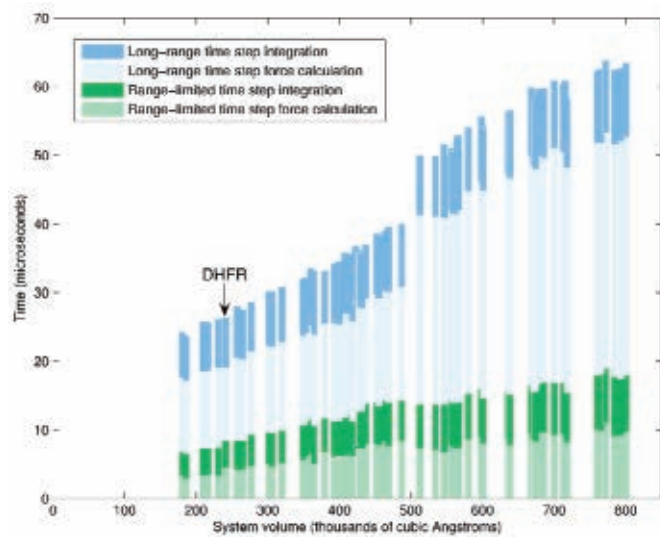
To quantify the accuracy of force computation on Anton, we measured the *relative rms force error*, defined as the rms error in the force on all particles divided by the rms force.<sup>18</sup> For the DHFR system with typical simulation parameters, Anton achieves a relative rms force error of  $1.5 \times 10^{-4}$ . A relative rms force error below  $10^{-3}$  is generally considered sufficiently accurate for biomolecular MD simulations.<sup>25</sup>

We also measured *energy drift* to quantify the overall accuracy of our simulations. An exact MD simulation would conserve energy exactly. Errors in the simulation generally lead to an increase in the overall energy of the simulated system with time, a phenomenon known as energy drift. We measured energy drift over 5 ns of simulated time (2 million time steps) for DHFR using a bit-accurate numerical emulator that exactly duplicates Anton's arithmetic. While the simulation exhibited short-term energy fluctuations of a few kcal/mol (about 0.001% of the total system energy), there was no detectable long-term trend in total energy. MD studies are generally considered more than adequate even with a significantly higher energy drift.<sup>24</sup>

## 5.3. Scaling with chemical system size

Figure 6 shows the scaling of performance with chemical system size. Within the range where chemical systems fit in on-chip memory, we expect performance to scale roughly linearly with the number of atoms, albeit with occasional jumps as different operating parameters change to optimize performance while maintaining accuracy. The largest discontinuity in simulation rate occurs at a system volume

**Figure 6: Scaling of performance for a 512-node version of Anton with increasing chemical system size. The graph shows a stacked bar chart for each chemical system, with the height of each stack proportional to the simulation time, assuming that long-range forces are evaluated every other time step. Each stack represents the time required to execute two consecutive time steps; one is a "long-range time step" that includes calculation of long-range electrostatics by *k*-GSE, and the other is a "range-limited time step" that does not. The chemical systems represent proteins and nucleic acids of various sizes, surrounded by water.**



of approximately  $500,000 \text{ \AA}^3$  when we change from a  $32 \times 32 \times 32$  FFT grid to a  $64 \times 64 \times 64$  FFT grid, reflecting the fact that our code supports only power-of-two-length FFTs. This lengthens the long-range calculation because the number of grid points increases by a factor of 8. Overall, the results are consistent with supercomputer scaleup studies—as we increase chemical system size, Anton's efficiency improves because of better overlap of communication and computation, and because calculation pipelines operate closer to peak efficiency.

## 6. CONCLUSION

We are currently in the process of building a specialized, massively parallel machine, called Anton, for the high-speed execution of MD simulations. We expect Anton to be capable of simulating the dynamic, atomic-level behavior of proteins and other biological macromolecules in an explicitly represented solvent environment for periods on the order of a millisecond—about three orders of magnitude beyond the reach of current MD simulations. The machine uses specialized ASICs, each of which performs a very large number of application-specific calculations during each clock cycle. Novel architectural and algorithmic techniques are used to minimize intra- and inter-chip communication, providing an unusually high degree of scalability.

While it contains programmable elements that could in principle support the parallel execution of algorithms for a wide range of other applications, Anton was not designed to function as a general-purpose scientific supercomputer, and would not in practice be well suited for such a role. Rather, we envision Anton serving as a computational microscope, allowing researchers to observe for the first time a wide range of biologically important structures and processes that have thus far proven inaccessible to both computational modeling and laboratory experiments. □

## References

- Adcock, S.A. and McCammon, J.A. Molecular dynamics: Survey of methods for simulating the activity of proteins. *Chemical Review*, 106:1589–1615, 2006.
- Bhatele, A., Kumar, S., Mei, C., Phillips, J.C., Zheng, G., and Kale, L.V. Overcoming scaling challenges in biomolecular simulations across multiple platforms, to appear in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008)*, Miami, FL, 2008.
- Bowers, K.J., Chow, E., Xu, H., Dror, R.O., Eastwood, M.P., Gregersen, B.A., Klepeis, J.L., Kolossvary, I., Moraes, M.A., Sacerdoti, F.D., Salmon, J.K., Shan, Y., and Shaw, D.E. Scalable algorithms for molecular dynamics simulations on commodity clusters. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC06)*, Tampa, FL, 2006.
- Bowers, K.J., Dror, R.O., and Shaw, D.E. Zonal methods for the parallel execution of range-limited N-body problems. *Journal of Computational Physics*, 221(1):303–329, 2007.
- Fitch, B.G., Rayshubskiy, A., Eleftheriou, M., Ward, T.J.C., Giampapa, M.E., Pitman, M.C., Pitera, J.W., Swope, W.C., and Germain, R.S. Blue matter: scaling of N-body simulations to one atom per node. *IBM Journal of Research and Development*, 52(1/2), 2008.
- Fine, R.D., Dimmler, G., and Levinthal, C. FASTRUN: A special purpose, hardware computer for molecular simulation. *Proteins: Structure, Function, and Genetics*, 11(4):242–253, 1991 (erratum: 14(3):421–422, 1992).
- Germain, R.S., Fitch, B., Rayshubskiy, A., Eleftheriou, M., Pitman, M.C., Suits, F., Giampapa, M., and Ward, T.J.C. Blue matter on blue gene/L: Massively parallel computation for biomolecular simulation. *Proceedings of the Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS '05)*, New York, NY, 2005.
- Hess, B., Kutzner, C., van der Spoel, D., and Lindahl, E. GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of Chemical Theory and Computation*, 4(2):435–447, 2008.
- Jorgensen, W.L., Maxwell, D.S., and Tirado-Rives, J. Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids.

- Journal of the American Chemical Society*, 118(45):11225–11236, 1996.
10. Kalé, L., Skeel, R., Bhandarkar, M., Brunner, R., Gursoy, A., Krawetz, N., Phillips, J., Shinozaki, A., Varadarajan, K., and Schulten, K., NAMD2: Greater scalability for parallel molecular dynamics. *Journal of Computational Physics*, 151(1):283–312, 1999.
  11. Kollman, P.A., Dixon, R.W., Cornell, W.D., Fox, T., Chipot, C., and Pohorille, A. The development/application of a "Minimalist" organic/biomolecular mechanic forcefield using a combination of ab initio calculations and experimental data, in *Computer Simulation of Biomolecular Systems: Theoretical and Experimental Applications*, van Gunsteren, W.F. and Weiner, P.K. eds., Dordrecht, Netherlands:ESCOM, pp. 83–96, 1997.
  12. Kuskin, J.S., Young, C., Grossman, J.P., Batson, B., Deneroff, M.M., Dror, R.O., and Shaw, D.E. Incorporating flexibility in Anton, a specialized machine for molecular dynamics simulation. *Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA-14)*, Salt Lake City, UT, 2008.
  13. Larson, R.H., Salmon, J.K., Dror, R.O., Deneroff, M.M., Young, C., Grossman, J.P., Shan, Y., Klepeis, J.L., and Shaw, D.E. High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation. *Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA-14)*, Salt Lake City, UT, 2008.
  14. Liem, S.Y., Brown, D., and Clarke, J.H.R. Molecular dynamics simulations on distributed memory machines. *Computer Physics Communications*, 67(2):261–267, 1991.
  15. MacKerell, A.D. Jr., Bashford, D., Bellott, M., Dunbrack, R.L., Evanseck, J.D., Field, M.J., Fischer, S., Gao, J., Guo, H., Ha, S., Joseph-McCarthy, D., Kuchnir, L., Kuczera, K., Lau, F.T.K., Mattos, C., Michnick, S., Ngo, T., Nguyen, D.T., Prodhom, B., Reiher, III, W.E., Roux, B., Schlenker, M., Smith, J.C., Stote, R., Straub, J., Watanabe, M., Wiórkiewicz-Kuczera, J., Yin, D., and Karplus, M.J. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *Journal of Physical Chemistry B*, 102(18):3586–3616, 1998.
  16. Pande, V.S., Baker, I., Chapman, J., Elmer, S.P., Khalil, S., Larson, S.M., Rhee, Y.M., Shirts, M.R., Snow, C.D., Sorin, E.J., and Zagrovic, B. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2003.
  17. Plipton, S.J., Attaway, S., Hendrickson, B., Swegle, J., Vaughan, C., and Gardner, D. Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics. *Proceedings of the ACM/IEEE Conference on Supercomputing (Supercomputing '96)*, Pittsburgh, PA, 1996.
  18. Shan, Y., Klepeis, J.L., Eastwood, M.P., Dror, R.O., and Shaw, D.E. Gaussian split Ewald: A fast Ewald mesh method for molecular simulation. *Journal of Chemical Physics*, 122:054101, 2005.
  19. Shaw, D.E. A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions. *Journal of Computational Chemistry*, 26(13):1318–1328, 2005.
  20. Shaw, D.E., Deneroff, M.M., Dror, R.O., Kuskin, J.S., Larson, R.H., Salmon, J.K., Young, C., Batson, B., Bowers, K.J., Chao, J.C., Eastwood, M.P., Gagliardo, J., Grossman, J.P., Ho, C.R., Ierardi, D.J., Kolossváry, I., Klepeis, J.L., Layman, T., McLeavey, C., Moraes, M.A., Mueller, R., Priest, E.C., Shan, Y., Spengler, J., Theobald, M., Towles, B., and Wang, S.C., Anton, a special purpose machine for molecular dynamics simulation. *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, San Diego, CA, 2007.
  21. Snir, M. A Note on N-body computations with cutoffs. *Theory of Computing Systems*, 37:295–318, 2004.
  22. Tajiri, M., Narumi, T., Ohno, Y., Futatsugi, N., Suenaga, A., Takada, N., and Konagaya, A., Protein explorer: A petaflops special-purpose computer system for molecular dynamics simulations. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC03)*, Phoenix, AZ, 2003.
  23. Toyoda, S., Miyagawa, H., Kitamura, K., Amisaki, T., Hashimoto, E., Ikeda, H., Kusumi, A., and Miyakawa, N. Development of MD engine: High-speed accelerator with parallel processor design for molecular dynamics simulations. *Journal Computational Chemistry*, 20(2): 185–199, 1999.
  24. Wang, W. and Skeel, R.D. Fast evaluation of polarizable forces. *Journal of Chemical Physics*, 123(16):164107, 2005.
  25. Zhou, R., Harder, E., Xu, H., and Berne, B.J. Efficient multiple time step method for use with Ewald and particle mesh Ewald for large biomolecular systems. *Journal of Chemical Physics*, 115(5): 2348–2358, 2001.

**David E. Shaw** (David.Shaw@DEShawResearch.com) Center for Computational Biology and Bioinformatics, Columbia University, New York, NY 10032

© 2008 ACM 0001-0782/08/0700 \$5.00

## ACM TRANSACTIONS ON INTERNET TECHNOLOGY



**Transactions on Internet Technology (TOIT).** This quarterly publication encompasses many disciplines in computing—including computer software engineering, middleware, database management, security, knowledge discovery and data mining, networking and distributed systems, communications, and performance and scalability—all under one roof. *TOIT* brings a sharper focus on the results and roles of the individual disciplines and the relationship among them. Extensive multi-disciplinary coverage is placed on the new application technologies, social issues, and public policies shaping Internet development. **Subscribe Today!**

### PRODUCT INFORMATION

ISSN: 1533-5399

Price: \$41 Professional Member

Order Code: (140)

\$36 Student Member

\$170 Non-Member

\$16 Air Service (for residents outside North America only)

### TO PLACE AN ORDER

#### Contact ACM Member Services

Phone: 1.800.342.6626  
(U.S. and Canada)

Email: acmhelp@acm.org

+1.212.626.0500  
(Global)

Mail: ACM Member Services

General Post Office

P.O. Box 30777

New York, NY

10087-0777 USA

Fax: +1.212.944.1318

(Hours: 8:30am–4:30pm, Eastern Time)

[www.acm.org/pubs/periodicals/toit](http://www.acm.org/pubs/periodicals/toit)



Association for  
Computing Machinery

Advancing Computing as a Science & Profession

AD28

# Technical Perspective

## The Physical Side of Computing

By Feng Zhao

WIRELESS SENSOR NETWORKS (OR sensornets) represent a new computing platform that blends computation, sensing, and communication with a physical environment such as a bird habitat, bridges, or power grid. This new class of networked embedded computers requires new programming models, abstractions, and management tools. They change the way we think about computation and challenge the design of the next-generation Internet that not only connects people together but also connects people with the physical environment.

This computing platform is characterized by the embedding in the physical world and (often) unattended operation for years, severe constraints in resources especially energy, unreliable hardware and communication links, and the need to respond to time-critical events. The implications are twofold. A sensornet must gather and act on sensor data in a timely manner. The value of information and window of opportunity for action may dwindle as time elapses. Consequently, sensornets should support reliable and timely data collection and dissemination despite significant link and data variability and hardware flakiness. Second, because of limited battery capacity, a sensor node limits the amount of onboard memory and uses low-power microprocessor and low data rate radio with power-saving dials. The data collection and dissemination must be handled in an energy-efficient manner.

A fundamental computer science question arising from sensornets is the role of energy and how we think about it in relation to performance and quality metrics such as latency and data yield. Much of CS has been built on the analysis of the time and space complexity of algorithms that has informed the design of processor, memory, and I/O in computing systems. Only recently have we confronted the energy problem head on, in designing high-perfor-

mance servers as well as low-power sensornets (supercomputing addressed the cooling problem before). Multi-/many-core is one answer in the upper tier of the computing ecosystem. The tiny computers in sensornets expose another rich area where energy trades with performances in a decentralized, fine-grained way. For example, communication in data dissemination may be delayed, to reduce collision and hence energy due to excessive retransmission, at the expense of a larger latency. Sensor data may be locally compressed at the node, to reduce the data volume sent over the wireless network, trading the communication energy with that of processing. This points to the need for establishing a theory of “energy complexity” in computing that provides models for energy and its trade-offs with other system metrics.

The decade of sensornet research has produced a rich collection of algorithms, protocols, system architectures, tools, and several generations of hardware platforms. The energy constraints, for example, led many to design extremely efficient systems that break the traditional networking and systems layers in order to squeeze the last Joule out of the operation. Naturally, one asks, what are the reusable building blocks and common abstractions that emerge from these works? Some of the techniques address the deeper problems of energy complexity, system scalability, and robustness. Others may just be artifacts of the current hardware limitations.

Levis et al. answers the question with Trickle, a building block for algorithms that move data around quickly in a sensornet while conserving its limited energy. Realizing that the one-to-many and many-to-one data dissemination and collection in a network rely on a common primitive to detect when the state of a node becomes inconsistent in a network of shared variables and to propagate the information when

inconsistency arises, they propose an epidemic-style algorithm that does so on an as-needed basis. It was originally designed for distributing code in a sensornet, as in re-tasking or code patching. To detect whether a node has the latest version, each node declares to others which version it currently has. An inconsistency triggers the propagation on demand, suppressing the transmissions of others, thus more energy efficient than flooding.

The key idea behind Trickle is to maintain a constant number of message transmissions per area, and use a feedback mechanism to regulate that as node density changes. A node only decides to transmit if it has not heard from a sufficient number of its neighbors. This way, the more nodes in an area, the less likely each node will decide to transmit as the likelihood of others already having advertised increases. Trickle provides the dials to trade energy expenditure of the network with the speed of the propagation. This self-regulation mechanism is similar to how nature regulates the population of a species, where growth is self-limiting because of the finite sustainable food supply.

A useful primitive finds itself in many applications. Trickle is promising; since the publication of the original paper, the idea of Trickle has found itself in data dissemination as well as data collection algorithms, including TinyOS 2.0 CTP, a data collection protocol.

Gordon Bell posits every decade or so a new computing platform emerges due to advantages in form factor, interface, and functionality/price. The wireless sensor network is such a new computing platform. I expect emerging primitives and abstractions like Trickle, being developed by the research community, to help us conceptualize and modularize the design of this new platform and to become part of a standard TTL-like catalog for building scalable, reliable, and energy-efficient sensornet systems. □

Feng Zhao (zhao@microsoft.com) is a principal researcher at Microsoft Research, Redmond, WA.



# The Emergence of a Networking Primitive in Wireless Sensor Networks

By Philip Levis, Eric Brewer, David Culler, David Gay, Sam Madden, Neil Patel, Joe Polastre, Scott Shenker, Robert Szewczyk, and Alec Woo

## Abstract

The wireless sensor network community approached networking abstractions as an open question, allowing answers to emerge with time and experience. The Trickle algorithm has become a basic mechanism used in numerous protocols and systems. Trickle brings nodes to eventual consistency quickly and efficiently while remaining remarkably robust to variations in network density, topology, and dynamics. Instead of flooding a network with packets, Trickle uses a “polite gossip” policy to control send rates so each node hears just enough packets to stay consistent. This simple mechanism enables Trickle to scale to 1000-fold changes in network density, reach consistency in seconds, and require only a few bytes of state yet impose a maintenance cost of a few sends an hour. Originally designed for disseminating new code, experience has shown Trickle to have much broader applicability, including route maintenance and neighbor discovery. This paper provides an overview of the research challenges wireless sensor networks face, describes the Trickle algorithm, and outlines several ways it is used today.

## 1. WIRELESS SENSOR NETWORKS

Although embedded sensing applications are extremely diverse, ranging from habitat and structural monitoring to vehicle tracking and shooter localization, the software and hardware architectures used by these systems are surprisingly similar. The typical architecture is embodied by the mote platforms, such as those shown in Figure 1. A microcontroller provides processing, program ROM, and data RAM, as well as analog-to-digital converters for sensor inputs, digital interfaces for connecting to other devices, and control outputs. Additional flash storage holds program images and data logs. A low-power CMOS radio provides a simple link layer. Support circuitry allows the system to enter a low-power sleep state, wake quickly, and respond to important events.

Four fundamental constraints shape wireless embedded system and network design: power supply, limited memory, the need for unattended operation, and the lossy and transient behavior of wireless communication. A typical power envelope for operating on batteries or harvesting requires a 600  $\mu$ W average power draw, with 1% of the time spent in a 60 mW active state and the remainder spent in a very low power 6  $\mu$ W passive state.

Maintaining a small memory footprint is a major requirement of algorithm design. Memory in low-cost, ultra-low-power devices does not track Moore’s Law. One indication of this is that microcontroller RAM costs three orders of magnitude more than PC SRAM and five orders more than PC DRAM. More importantly, SRAM leakage current, which grows with capacity, dictates overall standby power consumption and, hence, lifetime. Designs that provide large RAMs in conjunction with 32-bit processors go to great lengths to manage power. One concrete example of such nodes is the Sun SPOT,<sup>20</sup> which enters a low-power sleep state by writing RAM contents to flash. Restoring memory from flash on wakeup uses substantial power and takes considerable time. The alternative, taken in most sensor node designs, is to have just a few kilobytes of RAM. This, in turn, imposes limits on the storage complexity of network (and other) protocols, requiring routing tables, buffering, and caches be kept small. The historical trends of monetary and energy costs suggest these constraints are likely to last.

Wireless sensors are typically embedded in the physical environment associated with their application. Com-

**Figure 1: EPIC, KMote, and Telos motes. Each has an 8MHz microcontroller, 10kB of RAM, 48kB of program flash, and a 250kbps radio.**



munication connectivity varies due to environmental and electromagnetic factors, with the additional constraint that no human being will shepherd the device to a better setting, as with a cell phone or a laptop. The degree of the network at a node, i.e., the number of nodes in its communication neighborhood, is determined not by the desired network organization but by the physical device placement, which is often dictated by application requirements and physical constraints. There may be thousands of nodes in close proximity, or just a few. A single transmission may be received by many devices, so any retransmission, response, or even a simple acknowledgment, may cause huge contention, interference, and loss. Redundancy is essential for reliability, but it also can be a primary cause of loss.

This last point is one of the key observations that have emerged from a decade of development of networking abstractions for wireless sensor networks: the variety of network topologies and densities across which sensor network protocols must operate calls for a polite, density-aware, local retransmission scheme. This paper describes the Trickle algorithm, which uses such a communication pattern to provide an eventual consistency mechanism to protocols and services. In the past ten years, a key insight that has emerged from the wireless sensor network community is that many protocol problems can be reduced to maintaining eventual consistency. Correspondingly, Trickle has emerged as the core networking primitive at the heart of practical, efficient, and robust implementations of many sensor network protocols and systems. Before diving into the details of the Trickle, however, we review how core sensor networking protocols work and differ from conventional networking protocols, with the goal of exploring how a Trickle-like primitive satisfies some of their needs.

## 2. NETWORKING PROTOCOLS

Networking issues are at the core of embedded sensor network design because radio communication—listening, receiving, and transmitting—dominates the active energy budget and defines system lifetime. The standard energy cost metric for multihop protocols, in either link layer meshing or network layer routing, is communication cost, defined as the number of individual radio transmissions and receptions. One protocol is more efficient than another if it can provide equivalent performance (e.g., throughput, latency, delivery ratio) at a lower communication cost. Protocols focus on minimizing transmissions and making sure transmitted packets arrive successfully.

Almost all sensor network systems rely on two multihop protocols for their basic operation: a *collection* protocol for pulling data out of a network and a *dissemination* protocol for pushing data into a network through one or more distinguished nodes or egress routers. Many higher level protocols build on dissemination and collection. For example, reprogramming services such as Deluge<sup>9</sup> use dissemination to deliver commands to change program images. Management layers<sup>22</sup> and remote source-level debuggers<sup>25</sup> also use dissemination. Reliable transport protocols, such as RCRT,<sup>18</sup> and rate control protocols such as

IFRC,<sup>19</sup> operate on collection trees. Point-to-point routing schemes, such as S4,<sup>16</sup> establish overlays over multiple parallel collection topologies.

While collection and dissemination have the opposite communication patterns (all-to-one vs. one-to-all) and differ in reliability (unreliable vs. reliable), both maintain eventually consistent shared state between nodes. The rest of this section provides a high-level overview of these two protocol classes. It provides details on the challenging problems they introduce, and how some of them can be solved through eventual consistency.

### 2.1. Pushing data in: dissemination

One problem sensor network administrators face is dynamically changing how a network collects data by changing the sampled sensors, the sampling rate, or even the code running on the nodes by disseminating the change to every node in a network. We begin with a discussion of dissemination protocols because they were the original impetus for Trickle and are its simplest application.

Early systems used packet floods to disseminate changes. Flooding protocols rebroadcast packets they receive. Flooding is very simple—often just a line or two of code—but has many problems. First, floods are unreliable. Inevitably, some nodes do not receive the packet, so users typically repeatedly flood until every node receives it. Second, in high density networks, many nodes end up rebroadcasting packets at the same time. These messages collide and cause a form of network collapse called a “broadcast storm.”<sup>17</sup>

Second-generation dissemination and network programming systems like Xnp<sup>3</sup> and TinyDB<sup>15</sup> use an adaptive flood combined with a protocol to request missing messages. Adaptive flooding uses an estimate of the node density to limit the flooding rate. The missing message protocol allows nodes to request the (hopefully few) missing messages from their neighbors. Unfortunately, getting such protocols to work well can be tricky, especially across a range of network densities and object sizes.

Another way to look at dissemination protocols is that they ensure that every node has an eventually consistent version of some shared state, such as the value of a configuration parameter or command. Data consistency is when all nodes have the same version of that state, and nodes resolve inconsistencies by updating neighbors to the newer version. Inductively, these definitions cause the network to converge on the most recent version. To disseminate a command, a system installs it on one node as a newer version and initiates the consistency protocol.

Casting dissemination as a data consistency problem means it does not provide full reliability. Eventual consistency only promises to deliver the most recent version to connected nodes. Disconnected nodes can and often do miss updates. In practice, however, this limitation is rarely problematic. An administrator who changes the data reporting rate three times then adds some new nodes and expects them to receive the most recent reporting rate, not all three. Similarly, when sending commands, users do not expect a new node to receive the entire history of all commands injected into a net-

work. A node that is disconnected for several minutes will still receive the most recent command when it reconnects, however.

Dissemination protocols succeed where flooding and its derivatives fail because they cast the problem of delivering data into maintaining data consistency among neighbors. This allows them to provide a very useful form of reliability in arbitrary topologies with no a priori topology knowledge or configuration. An effective dissemination protocol, however, needs to bring nodes up to date quickly while sending few packets when every node has the most recent version: this is correspondingly a requirement for the underlying consistency mechanism.

## 2.2. Pulling data out: collection

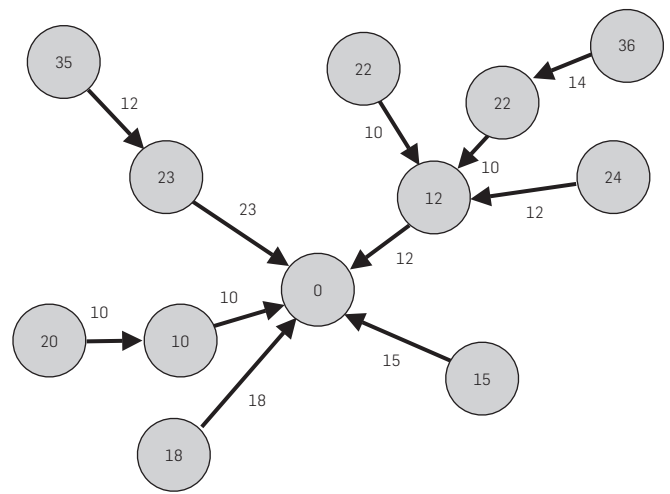
As the typical sensor network goal is to report observations on a remote environment, it is not surprising that data collection is the earliest and most studied class of protocol. There are many collection protocol variations, similar to how there are many versions of TCP. These differences aside, all commonly used collection protocols provide unreliable datagram delivery to a collection point using a minimum-cost routing tree. Following the general goal of layer 3 protocols, cost is typically measured in terms of expected transmissions, or ETX:<sup>2</sup> nodes send packets on the route that requires the fewest transmissions to reach a collection point.

The earliest collection protocol, directed diffusion, proposed dynamically setting up collection trees based on data-specific node requests.<sup>10</sup> Early experiences with low-power wireless, however, led many deployments to move towards a much simpler and less general approach, where each node decides on a single next hop for all forwarded data traffic, thereby creating routing trees for fixed collection points. The network builds this tree by establishing a routing cost gradient. A collection point has a cost of 0. A node calculates the cost of each of its candidate next hops as the cost of that node plus the cost of the link to it. Inductively, a node's cost is the sum of the costs of the links in its route. Figure 2 illustrates an example topology.

Collection variations boil down to how they quantify and calculate link costs, the number of links they maintain, how they propagate changes in link state amongst nodes, and how frequently they re-evaluate link costs and switch parents. Early protocols used hop-counts<sup>8</sup> as a link cost metric, similar to MANET protocols such as AODV and DSDV; second-generation protocols such as MintRoute<sup>24</sup> and Srcr<sup>2</sup> estimated the transmissions per delivery on a link using periodic broadcasts; third-generation protocols, such as MultiHopLQI, added physical layer signal quality to the metric; current generation collection protocols, such as Collection Tree Protocol (CTP), unify these approaches, drawing on information from multiple layers.<sup>6</sup>

Most collection layers operate as anycast protocols. A network can have multiple data collection points, and collection automatically routes to the closest one. As there is only one destination—any collection point—the required routing state can be independent of network density and size. Most protocols use a small, fixed-size table of candidate next

**Figure 2: Sample collection tree, showing per-link and node costs. The cost of a node is its next hop's cost plus the cost of the link.**



hops. They also attempt to strike a balance between route stability and churn to discover new, possibly better parents by switching parents infrequently and using damping mechanisms to limit the rate of change.

As collection protocols have improved and become better at choosing routes, reducing control traffic has become an increasingly important component of efficiency. While nodes can piggyback some control information on data packets, they need to send link-layer broadcasts to their local neighbors to advertise their presence and routing cost. Choosing how often to send these advertisements introduces a difficult design tension. A slow rate imposes a low overhead, but limits how quickly the tree can adapt to failures or link changes, making its data traffic less efficient. A fast rate imposes a higher overhead, but leads to an agile tree that can more accurately find the best route to use.

This tension is especially challenging when a network only collects data in response to events, and so can go through periods of high and low data rates. Having a high control rate during periods of low traffic is highly inefficient, while having a low control rate during periods of high traffic makes the tree unable to react quickly enough to changes. When starting a burst of transmissions, a node may find that link costs have changed substantially necessitating a change in its route and, as a result, advertised routing cost. Changes in costs need to propagate quickly, or the topology can easily form routing loops. For example, if a link's cost increases significantly, then a node may choose one of its children as its next hop. Since the protocol state must be independent of the topology, a node cannot avoid this by simply enumerating its children (constraining tree in-degree to a constant leads to inefficient, circuitous topologies in dense networks).

Current protocols, such as CTP<sup>21</sup> and ArchRock's routing layer,<sup>1</sup> resolve this tension by reducing the routing gradient as a data consistency problem. The gradient is consistent as long as children have a higher cost than their parent. An inconsistency can arise when costs change enough to violate



this constraint. As long as routing costs are stable, nodes can assume the gradient is consistent and avoid exchanging unnecessary packets.

### 2.3. A general mechanism

The examples above described how two very different protocols can both address a design tension by reducing a problem to maintaining data consistency. Both examples place the same requirements on a data consistency mechanism: it needs to resolve inconsistencies quickly, send few packets when data is consistent, and require very little state. The Trickle algorithm, discussed in the next section, meets these three requirements.

### 3. TRICKLE

The Trickle algorithm establishes a density-aware local broadcast with an underlying consistency model that guides when a node communicates. When a node's data does not agree with its neighbors, it communicates quickly to resolve the inconsistency. When nodes agree, they slow their communication rate exponentially, such that in a stable state nodes send at most a few packets per hour. Instead of flooding a network with packets, the algorithm controls the send rate so each node hears a small trickle of packets, just enough to stay consistent. Furthermore, by relying only on local broadcasts, Trickle handles network repopulation, is robust to network transience, loss, and disconnection, and requires very little state (implementations use 4–11 bytes).

While Trickle was originally designed for reprogramming protocols (where the data is the code of the program being updated), experience has shown it to be a powerful mechanism that can be applied to wide range of protocol design problems. For example, routing protocols can use Trickle to ensure that nodes in a given neighborhood have consistent, loop-free routes. When the topology is consistent, nodes occasionally gossip to check that they still agree, and when the topology changes they gossip more frequently, until they reach consistency again.

For the purpose of clearly explaining the reasons behind Trickle's design, all of the experimental results in this section are from simulation, in some cases very high-level abstract simulators. In practice, Trickle's simplicity means it works remarkably well in the far more challenging and difficult real world. The original Trickle paper,<sup>13</sup> as well as Deluge<sup>9</sup> and DIP<sup>14</sup> report experimental results from real networks.

### 3.1. Algorithm

Trickle's basic mechanism is a randomized, suppressive broadcast. A Trickle has a time interval of length  $\tau$  and a redundancy constant  $k$ . At the beginning of an interval, a node sets a timer  $t$  in the range of  $\frac{\tau}{2}, \tau$ . When this timer fires, the node decides whether to broadcast a packet containing metadata for detecting inconsistencies. This decision is based on what packets the node heard in the interval before  $t$ . A Trickle maintains a counter  $c$ , which it initializes to 0 at the beginning of each interval. Every time a node hears a Trickle broadcast that is consistent with its own state, it increments  $c$ . When it reaches time  $t$ , the Trickle broadcasts

Figure 3: Trickle parameters and variables.

|          |   |
|----------|---|
| $\tau$   | Communication interval length               |
| $T$      | Timer value in range $\frac{\tau}{2}, \tau$ |
| $C$      | Communication counter                       |
| $K$      | Redundancy constant                         |
| $\tau_l$ | Smallest $\tau$                             |
| $\tau_h$ | Largest $\tau$                              |

if  $c < k$ . Randomizing  $t$  spreads transmission load over a single-hop neighborhood, as nodes take turns being the first node to decide whether to transmit. Figure 3 summarizes Trickle's parameters.

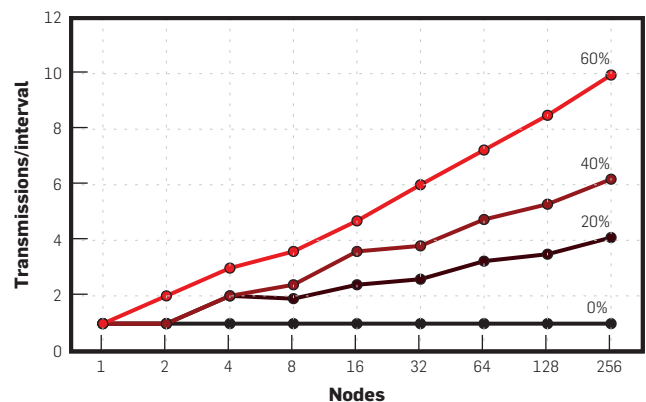
### 3.2. Scalability

Transmitting only if  $c < k$  makes a Trickle density aware, as it limits the transmission rate over a region of the network to a factor of  $k$ . In practice, the transmission load a node observes over an interval is  $O(k \cdot \log(d))$ , where  $d$  is the network density. The base of the logarithm depends on the packet loss rate  $PLR$ : it is  $\frac{1}{1-PLR}$ .

This logarithmic behavior represents the probability that a single node misses a number of transmissions. For example, with a 10% loss rate, there is a 10% chance that a node will miss a single packet. If a node misses a packet, it will transmit, resulting in two transmissions. There is correspondingly a 1% chance a node will miss two packets from other nodes, leading to three transmissions. In the extreme case of a 100% loss rate, each node is by itself: transmissions scale linearly.

Figure 4 shows this scaling. The number of transmissions scales logarithmically with density and the slope line (base of the logarithm) depends on the loss rate. These results come from a Trickle-specific algorithmic simulator we implemented to explore the algorithm's behavior under controlled conditions. Consisting of little more than an event queue, this simulator allows configuration of all of Trickle's parameters, run duration, and the boot time of nodes. It models a uniform packet loss rate (same for all links) across a single hop network. Its output is a packet send count.

Figure 4: Trickle's transmissions per interval scales logarithmically with density. The base of the logarithm is a function of the packet loss rate (the percentages)



### 3.3. Synchronization

The scaling shown in Figure 4 assumes that all nodes are synchronized, such that the intervals during which they are awake and listening to their radios line up perfectly. Inevitably, this kind of time synchronization imposes a communication, and therefore energy, overhead. While some networks can provide time synchronization to Trickle, others cannot. Therefore, Trickle is designed to work in both the presence and absence of synchronization.

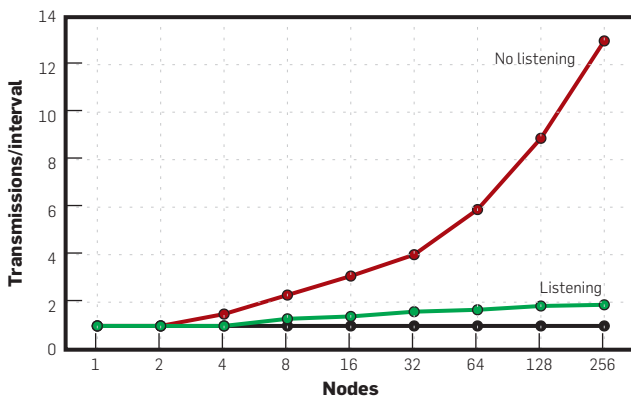
Trickle chooses  $t$  in the range of  $(\frac{\tau}{2}, \tau]$  rather than  $(0, \tau]$  because the latter causes the transmission load in unsynchronized networks to scale with  $O(\sqrt{d})$ . This undesirable scaling occurs due to the *short listen problem*, where some subset of motes gossip soon after the beginning of their interval. They listen for only a short time, before anyone else has a chance to speak up. This is not a problem if all of the intervals are synchronized, since the first gossip will quiet everyone else. However, if nodes are not synchronized, a node may start its interval just after another node's broadcast, resulting in missed messages and increased transmission load.

Unlike loss, where the extra  $O(\log(d))$  transmissions keep the worst case node that missed several packets up to date, the additional transmissions due to the short listen problem are completely wasteful. Choosing  $t$  in the range of  $(\frac{\tau}{2}, \tau]$  removes this problem: it defines a "listen-only" period of the first half of an interval. A listening period improves scalability by enforcing a simple constraint. If sending a message guarantees a silent period of some time  $T$  that is independent of density, then the send rate is bounded above (independent of the density). When a mote transmits, it suppresses all other nodes for at least the length of the listening period. Figure 5 shows how a listen period of  $\frac{\tau}{2}$  bounds the total sends in a lossless single-hop network to be  $2k$ . With loss, transmissions scale as  $O(2k \cdot \log(d))$  per interval, returning scalability to the  $O(\log(d))$  goal.

### 3.4. Controlling $\tau$

A large  $\tau$  (gossiping interval) leads to a low communication overhead, but propagates information slowly. Conversely,

**Figure 5: Without a listen-only period, Trickle's transmissions scale with a square root of the density when intervals are not synchronized. With a listen-only period of duration  $\frac{\tau}{2}$ , the transmissions per interval asymptotically approach  $2k$ . The black line shows how Trickle scales when intervals are synchronized. These results are from lossless networks.**



**Figure 6: Trickle pseudocode.**

| Event                     | Action  |
|---------------------------|---|
| $\tau$ Expires            | Double $\tau$ , up to $\tau_h$ . Reset $c$ , pick a new $t^*$ |
| $t$ Expires               | If $c < k$ , transmit   |
| Receive consistent data   | Increment $c$   |
| Receive inconsistent data | Set $t$ to $\tau_r$ . Reset $c$ , pick a new $t$              |

\* $t$  is picked from the range  $[\frac{\tau}{2}, \tau]$

a small  $\tau$  imposes a higher communication overhead, but propagates data more quickly. These two goals, rapid propagation and low overhead, are fundamentally at odds: the former requires communication to be frequent, while the latter requires it to be infrequent.

By dynamically scaling  $\tau$ , Trickle can quickly make data consistent with a very small cost.  $\tau$  has a lower bound,  $\tau_r$ , and an upper bound  $\tau_h$ . When  $\tau$  expires without a node receiving a new update,  $\tau$  doubles, up to a maximum of  $\tau_h$ . When a node detects a data inconsistency (e.g., a newer version number in dissemination, a gradient constraint violation in collection), it resets  $\tau$  to be  $\tau_r$ .

Essentially, when there is nothing new to say, motes gossip infrequently:  $\tau$  is set to  $\tau_h$ . However, as soon as a mote hears something new, it gossips more frequently, so those who have not heard the new data receive it quickly. The chatter then dies down, as  $\tau$  grows from  $\tau_r$  to  $\tau_h$ .

By adjusting  $\tau$  in this way, Trickle can get the best of both worlds: rapid consistency and low overhead when the network is consistent. The cost per inconsistency (shrinking  $\tau$ ) is approximately  $\log(\frac{\tau_h}{\tau_r})$  additional sends. For a  $\tau_r$  of 1 s and a  $\tau_h$  of 1 h, this is a cost of 11 packets to obtain a 3000-fold decrease in the time it takes to detect an inconsistency (or, from the other perspective, a 3000-fold decrease in maintenance overhead). The simple Trickle policy, "every once in a while, transmit unless you have heard a few other transmissions," can be used both to inexpensively maintain that the network is consistent as well as quickly inform nodes when there is an inconsistency.

Figure 6 shows pseudocode for the complete Trickle algorithm.

### 3.5. Case study: Maté

Maté is a lightweight bytecode interpreter for wireless sensor networks.<sup>11</sup> Programs are tiny sequences of optimized bytecodes. The Maté runtime uses Trickle to install new programs in a network, by making all nodes consistent to the most recent version of a script.

Maté uses Trickle to periodically broadcast version summaries. In all experiments, code routines fit in a single packet (30 bytes). The runtime registers routines with a Trickle propagation service, which then maintains all of the necessary timers and broadcasts, notifying the runtime when it installs new code. Maté uses a very simple consistency resolution mechanism. It broadcasts the missing routines three times: 1, 3, and 7 s after hearing there is an inconsistency.

Figure 7 shows simulation results of Maté's behavior during a reprogramming event. These results come from the TOSSIM simulator,<sup>12</sup> which simulates entire sensor network applications and

models wireless connectivity at the bit level. In these experiments,  $\tau_i$  is 1 s and  $\tau_h$  is 1 min.

Each simulation had 400 nodes regularly placed in a square grid with node spacings of 5, 10, 15, and 20 ft. By varying network density, we were able to examine how Trickle's propagation rate scales over different loss rates and physical densities. Density ranged from a 5 ft spacing between nodes up to 20 ft (the networks were  $95' \times 95'$  to  $380' \times 380'$ ). Crossing the network in these topologies takes from six to forty hops.<sup>a</sup> Time to complete propagation varied from 16 s in the densest network to about 70 s for the sparsest, representing a latency of 2.7 and 1.8 s per hop, respectively. The minimum per-hop Trickle latency is  $\frac{\tau_i}{2}$  and the consistency mechanism broadcasts a routine 1 s after discovering an inconsistency, so the best case latency is 1.5 s per hop. Despite an almost complete lack of coordination between nodes, Trickle still is able to cause them to cooperate efficiently.

Figure 8 shows how adjusting  $\tau_h$  changes the propagation time for the 5 and 20 ft spacings. Increasing  $\tau_h$  from 1 to 5 min does not significantly affect the propagation time; indeed, in the sparse case, it propagates faster until roughly the 95th percentile. This result indicates that there may be little trade-off between the maintenance overhead of Trickle and its effectiveness in the face of a propagation event.

A very large  $\tau_h$  can increase the time to discover inconsistencies to be approximately  $\frac{\tau_h}{2}$ . However, this is only true when two stable subnets ( $\tau = \tau_h$ ) with different code reconnect. If new code is introduced, it immediately triggers nodes to reset  $\tau$  to  $\tau_p$ , bringing them quickly to a consistent state.

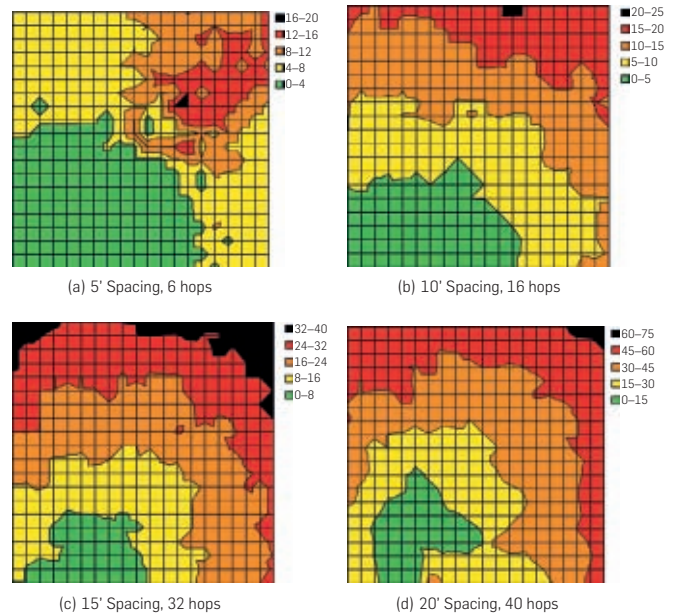
The Maté implementation of Trickle requires few system resources. It requires approximately 70 bytes of RAM; half of this is a message buffer for transmissions, a quarter is pointers to code routines. Trickle itself requires only 11 bytes for its counters; the remaining RAM is for internal coordination (e.g., pending and initialization flags). The executable code is 1.8 K (90 lines of code). Other implementations have similar costs. The algorithm requires few CPU cycles, and can operate at a very low duty cycle.

### 3.6. Uses and improvements

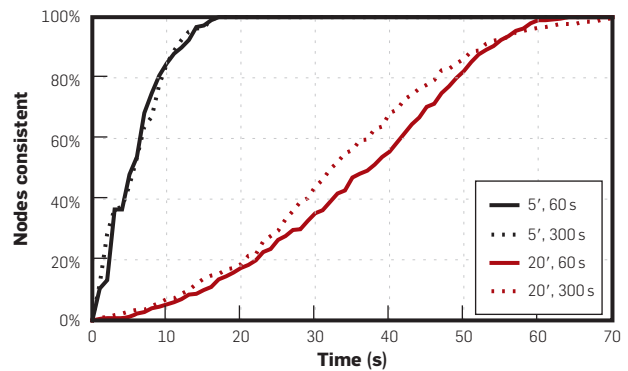
Trickle is not just used by Maté; it and its derivatives are used in almost every dissemination protocol today. The Deluge binary dissemination protocol for installing new sensor node firmware uses Trickle to detect when nodes have different firmware versions<sup>9</sup> ( $\tau_i = 500$  ms,  $\tau_h = 1.1$  h). The MNP binary dissemination protocol ( $\tau_i = 16$  s,  $\tau_h = 512$  s) adjusts Trickle so that nodes with more neighbors are more likely to send updates by preventing low degree nodes from suppressing high degree ones.<sup>23</sup> The Drip command layer of the Sensornet Management System uses Trickle ( $\tau_i = 100$  ms,  $\tau_h = 32$  s) to install commands.<sup>22</sup> The Tenet programming ar-

<sup>a</sup> These hop count values come from computing the minimum cost path across the network loss topology, where each link has a weight of  $\frac{1}{1-\text{loss}}$ , i.e. the expected number of transmissions to successfully traverse that link.

**Figure 7: Time to consistency in 20 × 20 TOSSIM grids (seconds). The hop count values in each legend are the expected number of transmissions necessary to get from corner to corner, considering loss.**



**Figure 8: Rate nodes reach consistency for different  $\tau_h$ s in TOSSIM. A larger  $\tau_h$  does not slow reaching consistency.**



chitecture uses Trickle ( $\tau_i = 100$  ms,  $\tau_h = 32$  s) to install small dynamic code tasks.<sup>7</sup>

In the past few years, as collection protocols have improved in efficiency, they have also begun to use Trickle. The CTP, the standard collection layer in the TinyOS operating system distribution,<sup>21</sup> uses Trickle timers ( $\tau_i = 64$  ms,  $\tau_h = 1$  h) for its routing traffic. The 6LoWPAN IPv6 routing layer in Arch Rock's software uses Trickle to keep IPv6 routing tables and ICMP neighbor lists consistent.<sup>1</sup> As protocols continue to improve, Trickle's efficacy and simplicity will cause it to be used in more protocols and systems.

One limitation with Trickle as described in this paper is that its maintenance cost grows  $O(n)$  with the number of data items, as nodes must exchange version numbers. This growth may be a hindering factor as Trickle's use increases. Recent work on the DIP protocol addresses this



concern by using a combination of hash trees and randomized searches, enabling the maintenance cost to remain  $O(1)$  while imposing a  $O(\log(n))$  discovery cost.<sup>14</sup>

#### 4. DISCUSSION

Wireless sensor networks, like other ad hoc networks, do not know the interconnection topology a priori and are typically not static. Nodes must discover it by attempting to communicate and then observing where communication succeeds. In addition, the communication medium is expected to be lossy. Redundancy in such networks is both friend and foe, but Trickle reinforces the positive aspects and suppresses the negative ones.

Trickle draws on two major areas of prior research. The first area is controlled, density-aware flooding algorithms for wireless and multicast networks.<sup>5,17</sup> The second is epidemic and gossiping algorithms for maintaining data consistency in distributed systems.<sup>4</sup> Although both techniques—broadcasts and epidemics—have assumptions that make them inappropriate in their pure form to eventual consistency in sensor networks, they are powerful techniques that Trickle draws from. Trickle's suppression mechanism is inspired by the request/repair algorithm used in Scalable and Reliable Multicast (SRM).<sup>5</sup> Trickle adapts to local network density as controlled floods do, but continually maintains consistency in a manner similar to epidemic algorithms. Trickle also takes advantage of the broadcast nature of the wireless channel, employing SRM-like duplicate suppression to conserve precious transmission energy and scale to dense networks.

Exponential timers are a common protocol mechanism. Ethernet, for example, uses an exponential backoff to prevent collisions. While Trickle also has an exponential timer, its use is reversed. Where Ethernet defaults to the smallest time window and increases it only in the case of collisions, Trickle defaults to the largest time window and decreases it only in the case of an inconsistency. This reversal is indicative of the different priorities in ultra-low-power networks: minimizing energy consumption, rather than increasing performance, is typically the more important goal.

In the case of dissemination, Trickle timers spread out packet responses across nodes while allowing nodes to estimate their degree and set their communication interval. Trickle leads to energy efficient, density-aware dissemination not only by avoiding collisions through making collisions rare, but also by suppressing unnecessary retransmissions.

Instead of trying to enforce suppression on an abstraction of a logical group, which can become difficult in multihop networks with dynamically changing connectivity, Trickle suppresses in terms of an implicit group: nearby nodes that hear a broadcast. Correspondingly, Trickle does not impose the overhead of discovering and maintaining logical groups, and effortlessly deals with transient and lossy wireless links. By relying on this implicit naming, the Trickle algorithm remains very simple: implementations can fit in under 2K of code, and require a mere 11 bytes of state.


Routing protocols discover other routers, exchange routing information, issue probes, and establish as well as tear

down links. All of these operations can be rate-controlled by Trickle. For example, in our experiences exploring how wireless sensor networks can adopt more of the IPv6 stack in 6LoWPAN, Trickle provides a way to support established ICMP-v6 mechanisms for neighbor discovery, duplicate address detection, router discovery, and DHCP in wireless networks. Each of these involves advertisement and response. Trickle mechanisms are a natural fit: they avoid loss where density is large, allow prompt notifications of change and adapt to low energy consumption when the configuration stabilizes. By adopting a model of eventual consistency, nodes can locally settle on a consistent state without requiring any actions from an administrator.

Trickle was initially developed for distributing new programs into a wireless sensornet: the title of the original paper is "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks."<sup>13</sup> Experience has shown it to have much broader uses. Trickle-based communication, rather than flooding, has emerged as the central paradigm for the basic multihop network operations of discovering connectivity, data dissemination, and route maintenance.

Looking forward, we expect the use of these kinds of techniques to be increasingly common throughout the upper layers of the wireless network stack. Such progress will not only make existing protocols more efficient, it will enable sensor networks to support layers originally thought infeasible. Viewing protocols as a continuous process of establishing and adjusting a consistent view of distributed data is an attractive way to build robust distributed systems.

#### Acknowledgments

This work was supported, in part, by the Defense Department Advanced Research Projects Agency (grants F33615-01-C-1895 and N6601-99-2-8913), the National Science Foundation (grants No. 0122599, IIS-033017, 0615308, and 0627126), by the California MICRO program, Intel Corporation, DoCoMo Capital, Foundation Capital, and a by Stanford Terman Fellowship. Research infrastructure was provided by the National Science Foundation (grants No. 9802069). We would also like to thank Sylvia Ratnasamy for her valuable insights into the early stages of this research, as well as Jonathan Hui, for his ideas on applying Trickle to new problems. 

#### References

1. Arch Rock Corporation. An IPv6 Network Stack for Wireless Sensor Networks. <http://www.archrock.com>.
2. Couto, D.D., Aguayo, D., Bicket, J., and Morris, R. A high-throughput path mMetric for multi-hop wireless routing. *Proceedings of the Ninth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2003.
3. Crossbow, Inc. Mote in Network Programming User Reference. <http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/Xnp.pdf>.
4. Demers, A., Greene, D., Hauser, C., Irish, W., and Larson, J. Epidemic Algorithms for Replicated Database Maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1987.
5. Floyd, S., Jacobson, V., McCanne, S., Liu, C.-G., and Zhang, L. A reliable multicast framework for light-weight sessions and application level framing. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 1995.
6. Fonseca, R., Gnawali, O., Jamieson, K., and Levis, P. Four bit wireless link estimation. *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*, 2007.
7. Gnawali, O., Greenstein, B., Jang, K.-Y., Joki, A., Paek, J., Vieira, M., Estrin, D., Govindan, R., and Kohler,



Association for  
Computing Machinery

Advancing Computing as a Science & Profession

ACM  
Transactions on  
Asian Language  
Information Processing

# ACM Transactions On Asian Language Information Processing

The Asian Language Information Processing Transaction (TALIP) publishes high quality original archival papers and technical notes in the areas of computation and processing of information in Asian languages and related disciplines. Some of the subjects to be covered by this quarterly publication are: Computational Linguistics; Linguistic Resources; Hardware and Software Algorithms and Tools for Asian Language Processing; Machine Translation; and Multimedia Asian Information Processing. Emphasis will be placed on the originality and the practical significance of the reported research.

To learn more about TALIP, please visit  
[www.acm.org/pubs/talip/](http://www.acm.org/pubs/talip/)

## SUBSCRIBE TODAY!

### PRODUCT INFORMATION

ISSN: 1530-0226

Order Code: 138

Price: \$38 Professional Member

\$33 Student Member

\$160 Non-Member

\$16 Air Service (for residents  
outside North America only)

### TO PLACE AN ORDER

Contact ACM Member Services

Phone: 1.800.342.6626 (U.S. and Canada)  
+1.212.626.0500 (Global)

Fax: +1.212.944.1318  
(Hours: 8:30am—4:30pm, Eastern Time)

Email: [acmhelp@acm.org](mailto:acmhelp@acm.org)

Mail: ACM Member Services  
General Post Office  
PO Box 30777  
New York, NY 10087-0777 USA

[www.acm.org/pubs/talip/](http://www.acm.org/pubs/talip/)

AD28

## References

- E. The TENET architecture for tiered sensor networks. *Proceedings of the Fourth International Conference on Embedded Networked Sensor Systems (Sensys)*, 2006.
- Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., and Pister, K.S.J. System architecture directions for networked sensors. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLoS)*, 2000.
- Hui, J.W. and Culler, D. The dynamic behavior of a data dissemination protocol for network programming at scale. *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.
- Intanagonwiwat, C., Govindan, R., and Estrin, D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2000.
- Levis, P., Gay, D., and Culler, D. Active sensor networks. *Proceedings of the Second USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2005.
- Levis, P., Lee, N., Welsh, M., and Culler, D. TOSSIM: accurate and scalable simulation of entire TinyOS applications. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.
- Levis, P., Patel, N., Culler, D., and Shenker, S. Trickle: a self-regulating algorithm for code maintenance and propagation in wireless sensor networks. *Proceedings of the First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.
- Lin, K. and Levis, P. Data discovery and dissemination with DIP. *Proceedings of the Seventh International Symposium on Information Processing in Sensor Networks (IPSN)*, 2008.
- Madden, S., Franklin, M.J., Hellerstein, J.M., and Hong, W. TinyDB: an acquisitional query processing system for sensor networks. *Transactions on Database Systems (TODS)*, 2005.
- Mao, Y., Wang, F., Qiu, L., Lam, S., and Smith, J. S4: small state and small stretch routing protocol for large wireless sensor networks. *Proceedings of the Fourth USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.
- Ni, S.-Y., Tseng, Y.-C., Chen, Y.-S., and Sheu, J.-P. The broadcast storm problem in a mobile ad hoc network. *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1999.
- Paek, J. and Govindan, R. RCRT: rate-controlled reliable transport for wireless sensor networks. *Proceedings of the Fifth International Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.
- Rangwala, S., Gummadi, R., Govindan, R., and Psounis, K. Interference-aware fair rate control in wireless sensor networks. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2006.
- Sun Microsystems Laboratories. Project Sun SPOT: Small Programmable Object Technology. <http://www.sunspotworld.com/>.
- TinyOS Network Protocol Working Group. TEP 123: The Collection Tree Protocol. <http://www.tinyos.net//tinyos-2.x/doc/txt/tep123.txt>, 2007.
- Tolle, G. and Culler, D. Design of an application-cooperative management system for wireless sensor networks. *Proceedings of the Second European Workshop of Wireless Sensor Networks (EWSN)*, 2005.
- Wang, L. MNP: Multihop network reprogramming service for sensor networks. *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.
- Woo, A., Tong, T., and Culler, D. Taming the underlying challenges of multihop routing in sensor networks. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.
- Yang, J., Soffa, M.L., Selavo, L., and Whitehouse, K. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. *Proceedings of the Fifth International Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

**Philip Levis** ([pal@cs.stanford.edu](mailto:pal@cs.stanford.edu))  
Assistant Professor, Stanford University,  
Stanford, CA, USA

**Eric Brewer** ([brewer@cs.berkeley.edu](mailto:brewer@cs.berkeley.edu))  
Professor, U.C. Berkeley, Berkeley,  
CA, USA

**David Culler** ([culler@cs.berkeley.edu](mailto:culler@cs.berkeley.edu))  
Professor, U.C. Berkeley, Berkeley,  
CA, USA

**David Gay** ([david.e.gay@intel.com](mailto:david.e.gay@intel.com)) Senior  
Researcher, Intel Research Berkeley,  
Berkeley, CA, USA

**Samuel Madden** ([madden@csail.mit.edu](mailto:madden@csail.mit.edu)) Associate Professor, MIT CSAIL,  
Cambridge, MA, USA

**Neil Patel** ([neilp@cs.stanford.edu](mailto:neilp@cs.stanford.edu)) Ph.D.  
Student, Stanford University, Stanford,  
CA, USA

**Joe Polastre** ([joe@sentilla.com](mailto:joe@sentilla.com)) CTO,  
Sentilla Corporation, Redwood City,  
CA, USA

**Scott Shenker** ([shenker@icsi.berkeley.edu](mailto:shenker@icsi.berkeley.edu)) Professor, U.C. Berkeley, Berkeley,  
CA, USA

**Robert Szewczyk** ([rob@sentilla.com](mailto:rob@sentilla.com))  
Principal Engineer, Sentilla Corporation,  
Redwood City, CA, USA

**Alec Woo** ([awoo@archrock.com](mailto:awoo@archrock.com))  
Technical Staff, Arch Rock Corporation,  
San Francisco, CA, USA

© 2008 ACM 001-0782/08/0700 \$5.00

## Howard Hughes Medical Institute Lead Software Engineer (Computational Biologist)

Howard Hughes Medical Institute Janelia Farm Research Campus Position Announcement Lead Software Engineer (Computational Biologist) A full time Lead Software Engineer (Computational Biologist) position is available at the Janelia Farm Research Campus. The Howard Hughes Medical Institute's Janelia Farm Research Campus is a unique, world-class research community in the Washington, D.C. area. Over the next four years, Janelia Farm Research Campus (JFRC) will grow to over 400 employees, to include top scientists, physicists, engineers and operations staff all working in a uniquely supportive campus environment. The candidate will work with computational biologists to develop algorithms and software that serves the needs of neuroscientists at Janelia Farm Research Campus. The candidate will have a solid understanding of numerical computation, algorithm development, and technical programming in C/C++. Knowledge of MATLAB and Numerical Python preferred while experience with high performance computing applications and GUI development is a plus. Responsible for developing algorithms and software that serves the needs of neuroscientists and will work closely with computational biologists to develop common software tools and methodologies that can be used across various research groups. The ideal candidate should have a BS/MS/PhD in Computer Science, Computational Biology or related field AND a minimum of 10 years of programming experience in C/C++ in the biological sciences; experience working closely with scientists in a pure research environment a plus. Solid understanding of numerical computation, algorithm development, and technical programming in C/C++; knowledge of MATLAB and Numerical Python preferred while experience with high performance computing applications and GUI development is a plus. HHMI offers a competitive salary and excellent benefits package. For consideration, please forward your resume in confidence to [jfrc-jobs@janelia.hhmi.org](mailto:jfrc-jobs@janelia.hhmi.org). Please include a cover letter detailing previous research experience and three references. Please also include job title in the subject line. To learn more about HHMI and Janelia Farm visit [www.hhmi.org/janelia](http://www.hhmi.org/janelia). HHMI is an Equal Opportunity Employer

## Howard Hughes Medical Institute Senior Software Engineer (Scientific Visualization)

Howard Hughes Medical Institute Janelia Farm Research Campus Position Announcement Senior Software Engineer (Scientific Visualization) A full time Senior Software Engineer (Scientific Visualization) position is available at the Janelia Farm Research Campus. The Howard Hughes Medical Institute's Janelia Farm Re-

search Campus is a unique, world-class research community in the Washington, D.C. area. Over the next four years, Janelia Farm Research Campus (JFRC) will grow to over 400 employees, to include top scientists, physicists, engineers and operations staff all working in a uniquely supportive campus environment. The candidate will work with neuroscientists and other members of the Scientific Computing team to design visualization tools and user interfaces to further research at Janelia Farm Research Campus. The candidate will have a solid understanding of graphics/user interface toolkits like VTK and OpenGL. The candidate will be expected to program in Python, Java, and MATLAB. Responsible for designing, architecting, and maintaining software/hardware architectures and frameworks and will work closely with neuroscientists and develop visualization tools and user interfaces. Will initiate discussions with researchers to understand the thrust of their domain science and their software and programming needs and will program software in Python, Java, and MATLAB. The ideal candidate should have a BS/MS in the Physical Sciences or preferably the Natural Sciences AND a minimum of 5 years of programming experience in visualization/graphics/user interfaces in the biological sciences; experience working closely with scientists in a pure research environment a plus. Proven skill designing displays for scientific data, especially in the natural sciences and proven knowledge of visualization toolkits like OpenGL and VTK desirable. HHMI offers a competitive salary and excellent benefits package. For consideration, please forward your resume in confidence to [jfrcjobs@janelia.hhmi.org](mailto:jfrcjobs@janelia.hhmi.org). Please include a cover letter detailing previous research experience and three references. Please also include job title in the subject line. To learn more about HHMI and Janelia Farm visit [www.hhmi.org/janelia](http://www.hhmi.org/janelia). HHMI is an Equal Opportunity Employer

## Qualcomm, Inc. Software Engineer (to develop technical training)

Qualcomm is seeking talented individuals to become part of our Learning Center team.

### Skills/Experience:

- ▶ Work with a team of technical specialists, engineers, and consultants.
- ▶ Develop and lead key engineering and technical initiatives, providing strategic guidance to staff and team members.
- ▶ Design, develop, implement, and evaluate engineer-related solutions
- ▶ Design and execute initiatives to accelerate organizational and team performance, specifically in the areas of software, hardware, systems, and test
- ▶ Technical understanding of software design processes and coding

- ▶ Technical understanding in programming engineering tools, telecommunications, RF theory, networking, computer systems, and wireless technologies
- ▶ Experience in conducting needs assessments and organizational surveys
- ▶ Ability to analyze data and translate into resources that meet business unit objectives
- ▶ Programming skills with C, C++, C#.
- ▶ Experience developing on Windows, Unix or Linux and/or any real-time OS experience (Vx-Works, etc..)

### Education:

Bachelor's degree in electrical engineering or computer science required.

## Rockwell Automation Sr Software Engineer

Design and develop new software features, participating in the full development lifecycle. Contributes to system design and independently develops subsystem designs that meet the requirements. Provides manpower/time estimation for the design. Qualifications: Requires a Bachelor's degree and a minimum of 4 years experience in a software product development environment.

## Rockwell Automation Sr Project Engineer Software

In this position you will design and develop new software features with a focus on sequential control for batch and continuous manufacturing processes. Responsibilities include providing technical and team leadership of a development team from requirements definition through product release,

### ACM Policy on Nondiscriminatory Advertising

ACM accepts recruitment advertising under the basic premise the advertising employer does not discriminate on the basis of age, color, race, religion, gender, sexual preference, or national origin. ACM recognizes, however, that laws on such matters vary from country to country and contain exceptions, inconsistencies, or contradictions. This is true of laws in the United States of America as it is of other countries. Thus ACM policy requires each advertising employer to state explicitly in the advertisement any employer restrictions that may apply with respect to age, color, race, religion, gender, sexual preference, or national origin. (Observance of the legal retirement age in the employer's country is not considered discriminatory under this policy.) ACM also reserves the right to unilaterally reject any advertising. ACM provides notices of positions available as a service to the entire membership ACM recognizes that from time to time there may be some recruitment advertising that may be applicable to a small subset of the membership, and that this advertising may be inherently discriminatory. ACM does not necessarily endorse this advertising, but recognizes the membership has a right to be informed of such career opportunities.





### Windows Kernel Source and Curriculum Materials for Academic Teaching and Research.

The Windows® Academic Program from Microsoft® provides the materials you need to integrate Windows kernel technology into the teaching and research of operating systems.

The program includes:

- **Windows Research Kernel (WRK):** Sources to build and experiment with a fully-functional version of the Windows kernel for x86 and x64 platforms, as well as the original design documents for Windows NT.
- **Curriculum Resource Kit (CRK):** PowerPoint® slides presenting the details of the design and implementation of the Windows kernel, following the ACM/IEEE-CS OS Body of Knowledge, and including labs, exercises, quiz questions, and links to the relevant sources.
- **ProjectOZ:** An OS project environment based on the SPACE kernel-less OS project at UC Santa Barbara, allowing students to develop OS kernel projects in user-mode.

*These materials are available at no cost, but only for non-commercial use by universities.*

For more information, visit [www.microsoft.com/WindowsAcademic](http://www.microsoft.com/WindowsAcademic) or e-mail [compsci@microsoft.com](mailto:compsci@microsoft.com).

spanning multiple product releases. Qualifications: Bachelor's degree and minimum of 12 years experience in a software product development environment. For more information and to apply, please visit <http://www.rockwellautomation.com/>.

### Rockwell Automation Sr. Interaction Designer Software

Lead the information design of an industry leading commercial software application for a core, high growth Rockwell Automation product line. Lead interaction design for our next generation software products supporting industrial automation system design. Qualifications: Masters degree in Interaction Design, Human Factors, Cognitive or Behavioral Science, or related field required. Experience in interaction design, user center design and user interface. For more information and to apply, please visit <http://www.rockwellautomation.com/>.

### Wisconsin Alumni Research Foundation

Postdoctoral Positions in Computation and Informatics in Biology and Medicine University of Wisconsin-Madison Morgridge Institute for Research

The University of Wisconsin-Madison, with support from the Morgridge Institute for Research, has several postdoctoral research positions in computation and informatics for researchers wishing to solve biomedical problems

## The MIT Press

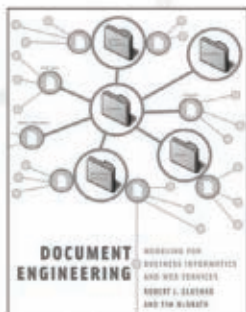


### A Semantic Web Primer

Second Edition

Grigoris Antoniou and Frank van Harmelen

"This book is essential reading for anyone who wishes to learn about the Semantic Web. By gathering the fundamental topics into a single volume, it spares the novice from having to read a dozen dense technical specifications. I have used the first edition in my Semantic Web course with much success." — Jeff Heflin, Associate Professor, Department of Computer Science and Engineering, Lehigh University  
Cooperative Information Systems series • 288 pp., 38 illus., \$42 cloth



NOW IN PAPER

### Document Engineering

Analyzing and Designing Documents for Business Informatics and Web Services

Robert J. Glushko and Tim McGrath

"This manifesto for the document engineering revolution gives you the what, the why, and the how of automating your business processes with XML, leading to greater cost savings, higher quality, and more flexibility." — Hal Varian, Haas School of Business and Department of Economics, University of California, Berkeley  
278 pp., \$22 paper

The MIT Press To order call 800-405-1619 • <http://mitpress.mit.edu>

requiring strengths in both computational and biological sciences.

These positions are being offered in cooperation with the Computation and Informatics in Biology and Medicine Training Program (CIBM; [www.cibm.wisc.edu](http://www.cibm.wisc.edu)). The 45 CIBM faculty span 15 different departments and five colleges at UW-Madison and includes several faculty at the Marshfield Clinic Research Foundation (located about 100 miles north of Madison). These positions are open to both US and non-US Citizens with a Ph.D., or equivalent, in computer science. The positions are funded for up to two years, renewable for a second year pending satisfactory progress, with an annual stipend up to \$65,000 per year.

**The research focus is in the development of:**

- ▶ Novel bioinformatics algorithms to analyze molecular data, including genome sequences, proteins (levels, interactions, structures), and regulatory pathways,
- ▶ New tools for imaging and genetic analysis,
- ▶ Development of health delivery systems,
- ▶ Translational bench-to-bedside medicine

For more information about the position and application materials, please contact Louise Pape at [lpape@wisc.edu](mailto:lpape@wisc.edu) or call 608-265-7935.

For more information about the Morgridge Institute for Research please visit [www.morgridgeinstitute.org](http://www.morgridgeinstitute.org) or contact Laura M. Heisler, Ph.D., Program Developer, Morgridge Institute for Research, 614 Walnut St., Madison, WI 53726 608.261.1022, [lheisler@morgridgeinstitute.org](mailto:lheisler@morgridgeinstitute.org)



## ADVERTISING IN CAREER OPPORTUNITIES

**How to Submit a Classified Line Ad: Send an e-mail to [jonathan.just@acm.org](mailto:jonathan.just@acm.org). Please include text, and indicate the issue/ or issues where the ad will appear, and a contact name and number.**

**Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. NO PROOFS can be sent. Classified line ads are NOT commissionable.**

**Rates: \$295.00 for six lines of text, 40 characters per line. \$80.00 for each additional three lines. The MINIMUM is six lines.**

**Deadlines: Five weeks prior to the publication date of the issue (which is the first of every month). Latest deadlines: <http://www.acm.org/publications>**

**Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at: <http://campus.acm.org/careercenter> Ads are listed for a period of six weeks.**

**For More Information Contact:**

**JONATHAN JUST**  
Director of Media Sales  
at 212-626-0687 or  
[jonathan.just@acm.org](mailto:jonathan.just@acm.org)

### Kuwait University Faculty of Science Kuwait

The Department of **Mathematics and Computer Science** in the Faculty of Science at Kuwait University invites applications for appointment of faculty members starting from September 2008, for the academic year 2008/2009, in one of the following areas:

**Networks, Operating Systems, Mobile Computing, Multimedia Systems, Computer Architecture, Theoretical Computer Science and Parallel & Distributed Computing**

**Required Qualifications:**

- ▶ Ph.D. degree in the area of specialization from a reputable University.
- ▶ The applicants GPA in first university degree should be 3 points out of 4 (or equivalent).
- ▶ Research experience and significant publications in refereed international journals.
- ▶ Full command of teaching in English.
- ▶ Minimum of 5 years in University teaching experience in the specified field.
- ▶ The successful candidates are expected to have a strong commitment and dedication to quality teaching and research.

Benefits include attractive tax-free salary according to rank and teaching experience (Professor's monthly salary varies from 2950 to 3192 KD., Associate Prof.'s salary varies from KD. 2265 to 2507, Assistant Professor's monthly salary varies from KD. 1830 to 2070 - [KD.1 = \$3.40]), annual air tickets for the faculty member and his/her family (spouse and up to three children under the age of 20), a one time settling-in allowance, housing allowance, free national health medical care, paid mid-term holidays and summer vacations, and end-of-contract gratuity. The University also offers an excellent academic environment and financial support for research projects.

To apply, send by express mail/courier service or email, within **two weeks** of the date of announcement, a completed application form, updated curriculum vitae (including mailing address, phone and fax numbers, e-mail address, academic qualifications, teaching and research experience, and a list of publications in professional journals up to 10 reprints), three copies of Ph.D., Masters, and Bachelor certificates and transcripts (An English translation of all documents in other languages should be enclosed), a copy of the passport, three recommendation letters, and names and addresses of three persons well-acquainted with the academic and professional work of the applicant. Please use PDF format for all electronic application materials. Applications and inquiries should be addressed to:

Dr. Salem Al-Yakoob  
Chairman  
Department of Mathematics  
and Computer Science  
Faculty of Science, Kuwait University  
P.O. Box 5969, Safat, 13060, Kuwait  
Tel: (965) 4813129  
Fax: +965 4817201  
E-Mail: [math@sci.kuniv.edu.kw](mailto:math@sci.kuniv.edu.kw)  
<http://www.sci.kuniv.edu.kw>

# Calendar of Events

## July

### July 20-23

**International Symposium on Symbolic and Algebraic Computation Linz/Hagenberg, Australia, Contact: Juan R. Sendra, Phone: 341-885-4902, Email: [rafael.sendra@uah.es](mailto:rafael.sendra@uah.es)**

### July 20-24

**International Symposium on Software Testing and Analysis Seattle, WA, Contact: Barbara G. Ryder, Phone: 732-445-6430 x3699, Email: [ryder@cs.rutgers.edu](mailto:ryder@cs.rutgers.edu)**

### July 21-25

**Mobiquitos08: 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services Dublin, Ireland, Contact: Liviu Iftode, Phone: 732-445-2001, Email: [iftode@cs.rutgers.edu](mailto:iftode@cs.rutgers.edu)**

### July 22-30

**Oregon Programming Languages Summer School Eugene, OR, Contact: Yannis Smaragdakis, Phone: 541-346-3491, Email: [yannis@cs.uoregan.edu](mailto:yannis@cs.uoregan.edu)**

### July 28-31

**5th International ICST Conference on Heterogeneous Networking for Quality, Reliability and Security Hong Kong, Contact: Qian Zhang, Phone: 852-23588766, Email: [qianzh@cs.eust.hk](mailto:qianzh@cs.eust.hk)**

## August

### August 4-6

**International Symposium on Low Power Electronics and Design Bangalore, India, Contact: Vijaykrishnan Narayanan, Email: [vijay@cse.psu.edu](mailto:vijay@cse.psu.edu)**

### August 9-10

**APGV '08: ACM Symposium on Applied Perception in Graphics and Visualization Los Angeles, CA, Contact: Bobby Bodenheimer, Phone: 615-322-3555, Email: [bobbyb@vuse.vanderbilt.edu](mailto:bobbyb@vuse.vanderbilt.edu)**

### August 11-13

**International Symposium on Low Power Electronics and Design Bangalore, India, Contact: Vijaykrishnan Narayanan, Email: [vijay@cse.psu.edu](mailto:vijay@cse.psu.edu)**

[CONTINUED FROM P.112]

on the data being exchanged in the critical sections. Many communication protocols had the same property. We decided to see if we could analyze finite-state programs by algorithmic means.

#### How exactly does that work?

**EAE** You have a program described by its text and its specification described by its text in some logic. It's either true or false that the program satisfies the specification, and one wants to determine that.

**JOSEPH SIFAKIS** Right. You build a mathematical model [of the program], and on this model, you check some properties, which are also mathematically specified. To check the property, you need a model-checking algorithm that takes as input the mathematical model you've constructed and then gives an answer: "yes," "no," or "I don't know." If the property is not verified, you get diagnostics.

#### And to formalize those specifications, those properties...

**EAE** What people really want is the program they desire, an inherently pre-formal notion. They have some vague idea about what sort of program they want, or perhaps they have some sort of committee that came up with an English prose description of what they want the program to do, but it's not a mathematical problem.

#### So one benefit of model checking is that it forces you to precisely specify your design requirements.

**EMC** Yes. But for many people, the most important benefit is that if the specification isn't satisfied, the model checker provides a counterexample execution trace. In other words, it provides a trace that shows you exactly how you get to an error that invalidates

**"The idea behind model checking was to avoid having humans construct proofs."**

your specification, and often you can use that to find really subtle errors in design.

#### How have model-checking algorithms evolved over the years?

**EMC** Model-checking algorithms have evolved significantly over the past 27 years. The first algorithm for model checking, developed by Allen and myself, and independently by Queille and Sifakis, was a fixpoint algorithm, and running time increased with the square of the number of states. I doubt if it could have handled a system with a thousand states. The first implementation, the EMC Model Checker (EMC stands for "Extended Model Checker"), was based on efficient graph algorithms, developed together with Allen and Prasad Sistla, another student of mine, and achieved linear time complexity in the size of the state space. We were able to verify designs with about 40,000 states. Because of the state-explosion problem, this was not sufficient in many cases; we were still not able to handle industrial designs. My student Ken McMillan then proposed a much more powerful technique called symbolic model checking. We were able to check some examples with 10 to the one-hundredth power states (1 with a hundred zeros after it). This was a dramatic breakthrough but was still unable to handle the state-explosion problem in many cases. In the late 1990s, my group developed a technique called bounded model checking, which enabled us to find errors in many designs with 10 to the 10,000 power states.

**EAE** These advances document the basic contribution of model checking. For the first time, industrial designs are being verified on a routine basis. Organizations, such as IBM, Intel, Microsoft, and NASA, have key applications where model checking is useful. Moreover, there is now a large model-checking community, including model-checking users and researchers contributing to the advance of model-checking technology.

#### What are the limitations of model checking?

**JS** You have two basic problems: how to build a mathematical model of the system and then how to check a property, a requirement, on that mathematical model.

First of all, it can be very challenging to construct faithful mathematical models of complex systems. For hardware, it's relatively easy to extract mathematical models, and we've made a lot of progress. For software, the problem is quite a bit more difficult. It depends on how the software is written, but we can verify a lot of complex software. But for systems consisting of software running on hardware, we don't know how to construct faithful mathematical models for their verification.

The other limitation is in the complexity of the checking algorithm, and here we have a problem called the state-explosion problem (that Clarke referred to earlier), which means that the number of the states may go exponentially high with the number of components of the system.

**EMC** Software verification is a Grand Challenge. By combining model checking with static analysis techniques, it is possible to find errors but not give a correctness proof. As for the state-explosion problem, depending on the logic and model of computation, you can prove theoretically that it is inevitable. But we've developed a number of techniques to deal with it.

#### Such as?

**EMC** The most important technique is abstraction. The basic idea is that part of the program or the protocol you're verifying doesn't really have any effect on the particular properties that you're checking. So what you can do is simply eliminate those particular parts from the design.

You can also combine model checking with compositional reasoning, where you take a complex design and break it up into smaller components. Then you check those smaller components to deduce the correctness of the entire system.

#### How large are the programs we can currently verify with model checking?

**EMC** Well, first of all, there's not always a natural correspondence between a program's size and its complexity. But I would say we can often check circuits with around 10 to the 100th power states (1 with a hundred zeros after it).

**JS** Right. We know how to verify systems of medium complexity today—it's difficult to say but perhaps a program of around 10,000 lines. But we



don't know how to verify very complex systems.

**EMC** We're always playing a catch-up game; we're always behind. We've developed more powerful techniques, but it's still difficult to keep up with the advance of technology and the complexity of new systems.

**Can we use model checking to check concurrent programs?**

**EAE** Arguably, model checking is a very natural fit for parallel programming. Typically, we treat parallelism as a nondeterministic—or, informally, random—choice, so, in a way a parallel program is a more complex sequential program, with many nondeterministic behaviors. Model checking is very well suited to describing and reasoning about the associated coordination and synchronization properties of parallel programs.

**EMC** Concurrent programs are much more difficult to debug because it's difficult for humans to keep track of a lot of things that are happening all at once. Model checking is ideal for that.

**JS** But if you have programs that interact with the physical environment, time becomes very important. For these systems, verification is much more complicated.

**Do we have any algorithms that can operate directly on implementable code?**

**EMC** To verify the process of translating a design to code, or to verify the code itself, is much more difficult. Some successful model checkers use this approach, however. The Java Pathfinder model checker developed at NASA Ames generates byte code for a Java program and simulates the byte code to find errors.

**JS** The best available technology is proprietary technology that was developed by U.S. companies. But most of the code-level model checkers are used to verify sequential software. If you want to verify concurrent software, then you need to be very careful.

**EMC** The SLAM model checker developed at Microsoft Research for finding errors in Windows device drivers is probably the most successful software model checker. It is now distributed to people who want to write device drivers for Windows. However, it is hardly a general-purpose software model checker.

**“If you have programs that interact with the physical environment, time becomes very important. For these systems, verification is more complicated.”**

**EAE** In hardware verification, Verilog and VHDL are widely used design description languages. Many industrial model checkers typically accept designs described in these languages.

**Is model checking something currently taught to undergraduates?**

**JS** Formal verification is definitely taught in Europe. Europe has traditionally had a stronger community in formal methods, and I'd like to say it has also traditionally had a stronger community in semantics and languages.

**EMC** Yes, there's always been more interest in verification in Europe than in the U.S. Most of the major universities here—CMU, Stanford, UC Berkeley, U. Texas, and so on—do offer courses in model checking at both undergraduate and graduate levels, but it hasn't filtered down to schools where no one is doing research in the topic. Part of that has to do with the availability of appropriate textbooks; good books are just beginning to come out.

**EAE** Formal methods are being taught with some frequency [in the U.S.], but they are not broadly incorporated into the core undergraduate curriculum as required courses to the extent that operating systems and data structures are. It is probably more prevalent at the graduate level. But the distinction between undergraduate and graduate is not clear-cut. At many schools advanced undergrad and beginning grad overlap.

**What's in store for model checking and formal verification?**

**EMC** I intend to continue looking at ways of making model checking more powerful. The state explosion phenomenon is still a difficult problem. I have worked on it for 27 years and probably will continue to do so. Another thing I want to do is focus on embedded software systems in automotive and avionics applications. These programs are often safety-critical. For example, in a few years, cars will be “drive-by-wire”; there will be no mechanical linkage between the steering wheel and the tires. The software will definitely need to be verified. Fortunately, embedded software is usually somewhat simpler in structure, without complex pointers; I think it may be more amenable to model checking techniques than general software.

**JS** Personally, I believe we should look into techniques that allow some sort of compositional reasoning, where we infer global properties from local properties of the system, because of the inherent limitations of techniques based on the analysis of a global model. I'm working on this, as well as on theories of how to build systems out of components, component-based systems.

**EAE** Model checking has caused a sea change in the way we think about establishing program correctness, from proof-theoretic (deductive proof) to model-theoretic (graph search). I think we will continue to make more or less steady progress, but the pace of development of hardware and software is going to accelerate. Whether we ever catch up I don't know. Systems that are being designed are getting bigger and messier. The seat-of-the-pants approach will no longer work. We'll have to get better at doing things modularly, and we'll have to have better abstractions. □

**Leah Hoffman** writes about science and technology from Brooklyn, NY.

© 2008 ACM 001-0782/08/0700 \$5.00

# Q&A

## Talking Model-Checking Technology

*A conversation with the 2007 ACM A.M. Turing Award winners.*

**E**DMUND M. CLARKE, E. Allen Emerson, and Joseph Sifakis were honored for their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries.

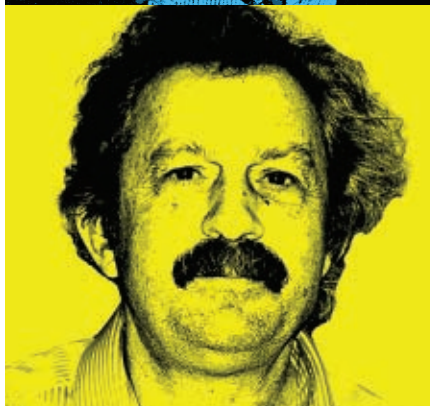
### Let's talk about the history of formal software verification.

**E. ALLEN EMERSON** By the late 1960s, we recognized that a program should be viewed as a mathematical object. It has a syntax and semantics and formally defined behavior engendered by that syntax and semantics. The idea was to give a mathematical proof that a program met a certain correctness specification. So one would have some axioms characterizing the way the program worked for such-and-such an instruction and some inference rules, and one would construct a formal proof of the system, like philosophers do sometimes.

But it never really seemed to scale up to large programs. You ended up with something like 15-page papers proving that a half-page program was correct. It was a great idea but didn't seem to pan out in practice.

### What about the history of model checking?

**EDMUND M. CLARKE** The birth of model checking was quite painful at times. Like most research on the boundary between theory and practice, theoreticians thought the idea was trivial, and system builders thought it was too theoretical. Researchers in formal methods were



even less receptive. Research in the formal-methods community in the 1980s usually consisted of designing and verifying tricky programs with fewer than 50 lines using only pen and paper. If anyone asked how such a program worked in practice on a real computer, it would have been interpreted as an insult or perhaps simply as irrelevant.

**EAE** The idea behind model checking was to avoid having humans construct proofs. It turns out that many important programs, such as operating systems, have ongoing behavior and ideally run forever; they don't just start and stop. In 1977, Amir Pnueli suggested that temporal logic could be a good way to describe and reason about these programs. Now, if a program can be specified in temporal logic, then it can be realized as a finite state program—a program with just a finite number of different configurations. This suggested the idea of model checking—to check whether a finite state graph is a model of a temporal logic specification. Then one can develop efficient algorithms to check whether the temporal-logic specification is true of the state graph by searching through the state graph for certain patterns.

**EMC** Yes, Allen and I noticed that many concurrent programs had what we called “finite state synchronization skeletons.” (Joseph Sifakis and J.P. Queille made the same observation, independently.) For example, the part of a mutual-exclusion program that handles synchronization does not depend

[CONTINUED ON P.110]

essays, {craft, art, science} of software, python, eclipse, agile development, onward!, {generative, functional} programming, .net, open source, concurrency, smalltalk, aspects, second life, ruby, service-orientation, objects, embedded, ultra large scale {model, test}-driven passion, fun!, agents, domain-specific use cases, movies, lightning talks, systems, objective-c, development, c#, design patterns, languages, wiki, product-lines, java, refactoring, plop



## DEFINE THE FUTURE OF SOFTWARE

[www.oopsla.org/submit](http://www.oopsla.org/submit)

### CONFERENCE CHAIR

**GAIL E. HARRIS**

Instantiated Software Inc.  
chair@oopsla.org

### PROGRAM CHAIR

**GREGOR KICZALES**

University of British Columbia  
papers@oopsla.org

### ONWARD! CHAIR

**DIRK RIEHLE**

SAP Research  
onward@oopsla.org

### CALL FOR PAPERS

March 19, 2008

Due date for Research Program, Onward!, Development Program, Educators' Symposium, Essays and proposals for Tutorials, Panels, Workshops and DesignFest

July 2, 2008

Due date for Development Program Briefs, Doctoral Symposium and Student Volunteers



Association for  
Computing Machinery

NASHVILLE CONVENTION CENTER, NASHVILLE, TN

October 19 - 23, 2008



# Moore's Law. Fermat's Theorem. Euler's Function. \_\_\_\_\_ 's Law.



**Kristen Grauman**  
The University of Texas  
at Austin



**Susan Hohenberger**  
Johns Hopkins University



**Robert Kleinberg**  
Cornell University



**Philip Levis**  
Stanford University



**Russell Tedrake**  
Massachusetts Institute  
of Technology

Who's next? The Microsoft® Research Faculty Fellowship Award recognizes exceptionally talented computing academics with the freedom to explore at the edge of computing. Please join us in congratulating this year's Fellows. To find out more, please visit <http://research.microsoft.com/nff>

Microsoft  
**Research**