

COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

07/2010 VOL 53 NO.07



Charles P. Thacker

ACM's A.M. Turing Award Winner

Association for
Computing Machinery



IPDPS 2011

A N C H O R A G E

25TH IEEE International Parallel and Distributed Processing Symposium

May 16-20, 2011 • Anchorage (Alaska) • USA

Embraced by six mountain ranges, with views of Mount McKinley in Denali National Park, and warmed by a maritime climate, Anchorage offers year-round adventure and recreation. It is a fitting destination for IPDPS to mark a quarter century of tracking developments in computer science. To celebrate the 25th year of IPDPS, plan to come early and stay late to enjoy a modern city surrounded by spectacular wilderness.

GENERAL CHAIR

Alan Sussman (University of Maryland, USA)

PROGRAM CHAIR

Frank Mueller (North Carolina State University, USA)

PROGRAM VICE-CHAIRS

ALGORITHMS: Olivier Beaumont (INRIA, France)

APPLICATIONS: Leonid Oliker (Lawrence Berkeley National Laboratory, USA)

ARCHITECTURES: Mahmut Taylan Kandemir (The Pennsylvania State University, USA)

SOFTWARE: Dimitrios S. Nikolopoulos (FORTH-ICS and University of Crete, Greece)

WORKSHOPS CHAIR

Ümit V. Çatalyürek (Ohio State University, USA)

PHD FORUM CHAIR

Luc Bougé (ENS Cachan, France)

STEERING CO-CHAIRS

Viktor K. Prasanna (University of Southern California, USA)

George Westrom (Discovery Science Center & FSEA, USA)

CALL FOR PARTICIPATION

In addition to the regular conference program of contributed papers, invited speakers and panels, PhD forum, and tutorials, IPDPS features workshops on the first and last days of the conference. Watch the IPDPS Web for updates and information on how to participate in all events.

CALL FOR PAPERS


Scope: Authors are invited to submit manuscripts that present original unpublished research in all areas of parallel and distributed processing, including the development of experimental or commercial systems. Work focusing on emerging technologies is especially welcome. Topics of interest include, but are not limited to:

- Parallel and distributed algorithms, focusing on issues such as: stability, scalability, and fault-tolerance of algorithms and data structures for parallel and distributed systems, communication and synchronization protocols, network algorithms, scheduling, and load balancing.
- Applications of parallel and distributed computing, including web applications, peer-to-peer computing, cloud and grid computing, scientific applications, and mobile computing. Papers focusing on applications using novel commercial or research architectures, or discussing scalability toward the exascale level are encouraged.
- Parallel and distributed architectures, including architectures for instruction-level and thread-level parallelism; petascale and exascale systems designs; special-purpose architectures, including graphics processors, signal processors, network processors, media accelerators, and other special purpose processors and accelerators; impact of technology on architecture; network and interconnect architectures; parallel I/O and storage systems; architecture of the memory hierarchy; power-efficient architectures; dependable architectures; and performance modeling and evaluation.
- Parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, operating systems, resource management, middleware for grids and clouds, libraries, performance modeling and evaluation, parallel programming paradigms, and programming environments and tools.

Best Papers Awards: Awards will be given for one best paper in each of the four conference technical tracks: algorithms, applications, architectures, and software. Selected papers will be considered for possible publication in a special issue of the Journal of Parallel and Distributed Computing.

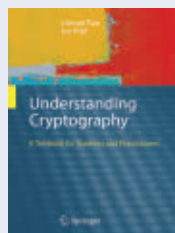
What/Where to Submit: More details on submissions and instructions for submitting files are available at www.ipdps.org or may be obtained by sending email to cfp@ipdps.org for an automatic reply. IPDPS will again require submission of abstracts one week before the paper submission deadline without any late exceptions (see important dates). All submitted manuscripts will be reviewed. Submitted papers may NOT have appeared in, or be under consideration for, another conference or workshop, or for a journal.



 Sponsored by IEEE Computer Society Technical Committee on Parallel Processing. In cooperation with ACM SIGARCH, IEEE Computer Society Technical Committee on Computer Architecture, and IEEE Computer Society Technical Committee on Distributed Processing.

IMPORTANT DATES	
24 September	2010 Abstracts due(required)
1 October 2010	Submissions due(hard deadline, no extensions)
10-12 November	Rebuttal period
17 December	Author notification
1 February 2011	Camera-ready papers

Noteworthy Titles



Understanding Cryptography

A Textbook for Students and

Practitioners

C. Paar, J. Pelzl, Ruhr-Universität Bochum, Germany

Foreword by: B. Preneel

Uniquely designed for students of engineering and applied computer science, and engineering practitioners. The authors have considerable experience teaching applied cryptography to engineering and computer science students and to professionals, and they make extensive use of examples, problems, and chapter reviews, while the book's website offers slides, projects and links to further resources.

1st Edition. 2nd Printing 2010. XVIII, 372 p. 320 illus., 160 in color. Hardcover ISBN 978-3-642-04100-6 ► **\$49.95**



Handbook of Ambient Intelligence and Smart Environments

H. Nakashima, Future University, Hakodate, Hokkaido, Japan;

H. Aghajan, Stanford University, Stanford, CA, USA; J. C. Augusto, University of Ulster at Jordanstown, Newtownabbey, UK (Eds.)

Provides readers with comprehensive, up-to-date coverage in this emerging field. Organizes all major concepts, theories, methodologies, trends and challenges into a coherent, unified repository. Covers a wide range of applications relevant to both ambient intelligence and smart environments. First available reference into this emerging area of research.

2010. XVIII, 1294 p. 100 illus. Hardcover ISBN 978-0-387-93807-3 ► **\$229.00**



Introduction to Databases

From Biological to Spatio-Temporal

P. Revesz, University of Nebraska-Lincoln, Lincoln, NE, USA

Provides a comprehensive coverage of the field of databases. This textbook reveals the workings of numerous database systems, enabling readers to develop complex database applications. Students will gain hands-on experience by following the exercises. The theory is complemented by detailed examination of specific applications from a broad range of areas.

2010. XVIII, 754 p. 346 illus., 173 in color. (Texts in Computer Science) Hardcover ISBN 978-1-84996-094-6 ► **\$119.00**



Patent Law for Computer Scientists

Steps to Protect Computer-Implemented Inventions

D. Closa, A. Gardiner,

F. Giemsa, J. Machek, European Patent Office, Munich, Germany

Explains patent laws in Europe, the US, and Japan. Gives the reader a guide to a patent examiner's way of thinking. Shows the step-by-step development of a patent application. Presents more than 10 detailed case studies from different computer science applications. Condenses over 70 years experience of the authors in the patent business.

2010. XV, 194 p. Hardcover ISBN 978-3-642-05077-0 ► **\$49.95**



Requirements Engineering

K. Pohl, Universität Duisburg-Essen,

Germany

Most comprehensive description of requirements engineering (RE) foundations and principles as well as up-to-date techniques, such as goal-oriented RE and scenario-based RE. Presentation of a didactically sound and industrially validated framework to structure the RE process and procedures. Many checklists and guidelines to support readers in their application of the concepts presented.

2010. 1050 p. Hardcover ISBN 978-3-642-12577-5 ► **\$79.95**



Algorithmic Adventures

From Knowledge to Magic

J. Hromkovic, ETH Zentrum, Zürich, Switzerland

Good explanation of

even the most basic techniques. Demonstrates the power and magic of the underlying principles. Fascinating read for students of all levels and for those curious to learn about the science and magic of algorithmics

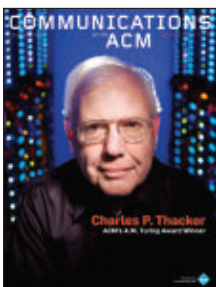
2009. XIII, 363 p. Hardcover ISBN 978-3-540-85985-7 ► **\$59.95**

Departments

- 5 **Editor's Letter**
Hypercriticality
By Moshe Y. Vardi
-
- 6 **Letters To The Editor**
**Don't Ignore Security Offshore,
or in the Cloud**
-
- 9 **In the Virtual Extension**
-
- 10 **BLOG@CACM**
**Software Development
and Crunch Time; and More**
Ruben Ortega discusses developers
and crunch time; Mark Guzdial
on the impact of open source
practices on computing education;
and Daniel Reed on the shift
from computational paucity to
computational plethora.
-
- 12 **CACM Online**
In Case You Missed It
By David Roman
-
- 39 **Calendar**
-
- 107 **Careers**

Last Byte

- 112 **Q&A**
From Single Core to Multicore
Charles P. Thacker discusses the
legendary Alto personal computer,
the invention of the Ethernet,
and his current research on
multicore architectures.
By Leah Hoffmann



About the Cover:
Charles P. Thacker, ACM's
2009 A.M. Turing Award
winner, as photographed
by Richard Morgenstein
in the data center
of Microsoft's offices
in Mountain View, CA.
For more on Morgenstein's
work, see <http://www.morgenstein.com>.

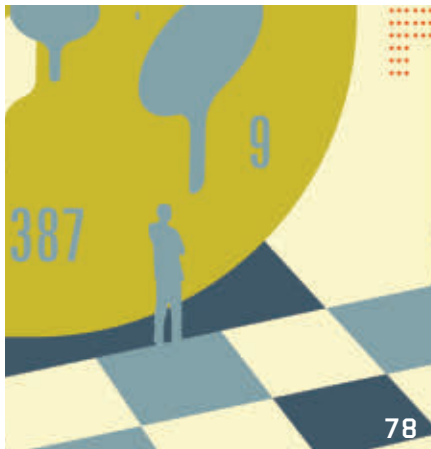
News

- 13 **Sharing Computational Perspectives**
Computer scientists are now
making intellectual contributions
to a wide range of other disciplines,
including evolutionary theory,
physics, and economics.
By David Lindley
-
- 16 **Censored!**
Countries use Internet censorship
to dominate the political
dialogue, but also to create
favorable conditions for
government-controlled businesses.
By Samuel Greengard
-
- 19 **Mainstreaming Augmented Reality**
Advancements in computer
vision, object recognition, and
related technologies are leading
to new levels of sophistication in
augmented-reality applications,
and presenting new ways for humans
to relate to the natural world.
By Kirk L. Kroeker
-
- 22 **Committed to Success**
Charles P. Thacker discusses the
importance of simplicity, reusable
tools, thinking broadly, and his
practice of Tom Sawyering.
By Gary Anthes
-
- 24 **Eric Brewer: Change Agent**
Eric Brewer's latest project
involves designing and deploying
low-cost wireless infrastructure
in developing regions.
By Gregory Goth
-
- 25 **Visions of the Future**
ACM joined forces with the British
Computer Society to deliver its
first academic research conference
in Europe.
By Sarah Underwood

Viewpoints

- 27 **Technology Strategy and Management**
Outsourcing Versus Shared Services
Choosing between outsourcing
and shared services has significant
implications for long-term
corporate strategy.
By Mari Sako
-
- 30 **Computing Ethics**
Work Life in the Robotic Age
Technological change results in
changes in expectations, in this case
affecting the workplace.
By Jason Borenstein
-
- 32 **Legally Speaking**
**Should the Google Book
Settlement Be Approved?**
Considering the precedent
that could be established by
approval of the controversial
Google book settlement.
By Pamela Samuelson
-
- 35 **Broadening Participation**
**Cultivating Cultural Diversity
in Information Technology**
Introducing CMD-IT, a new center
focused on synergistic activities
related to ethnic minorities and
people with disabilities.
By Valerie E. Taylor
-
- 37 **Viewpoint**
Is Computer Science Truly Scientific?
Reflections on the (experimental)
scientific method in computer
science.
By Gonzalo Génova
-
- 40 **Distinguished Members**
Advice to Members
Seeking ACM Distinction
By Marc Snir and Telle Whitney

Practice



- 42 **The Ideal HPC Programming Language**
Maybe it's Fortran. Or maybe it just doesn't matter.
By Eugene Loh

- 48 **Visualizing System Latency**
Heat maps are a unique and powerful way to visualize latency data. Explaining the results, however, is an ongoing challenge.
By Brendan Gregg

- 55 **You're Doing It Wrong**
Think you've mastered the art of server performance? Think again.
By Poul-Henning Kamp



Articles' development led by **acmqueue**
queue.acm.org

Review Articles

- 78 **Algorithmic Game Theory**
A new era of theoretical computer science addresses fundamental problems about auctions, networks, and human behavior.
By Tim Roughgarden

Contributed Articles

- 60 **Commonsense Understanding of Concurrency: Computing Students and Concert Tickets**
Innate understanding of concurrency helps beginners solve CS problems with multiple processes executing at the same time.
By Gary Lewandowski, Dennis J. Bouvier, Tzu-Yi Chen, Robert McCartney, Kate Sanders, Beth Simon, and Tammy VanDeGrift

- 71 **Computer Graphics for All**
Interactive computer graphics would rival word-processing and presentation programs for everyday communications.
By Takeo Igarashi

Research Highlights

- 88 **Technical Perspective**
A Solid Foundation for x86 Shared Memory
By Hans-J. Boehm

- 89 **x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors**
By Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen

- 98 **Technical Perspective**
Technology Scaling Redirects Main Memories
By Mary Jane Irwin

- 99 **Phase Change Memory Architecture and the Quest for Scalability**
By Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger

Virtual Extension

As with all magazines, page limitations often prevent the publication of articles that might otherwise be included in the print edition.

To ensure timely publication, ACM created *Communications'* Virtual Extension (VE).

VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on their merit. These articles are now available to ACM members in the Digital Library.

- IT Programs in High Schools: Lessons from the Cisco Networking Academy Program**
Alan R. Dennis, Thomas M. Duffy, and Hasan Cakir

- Creating the Experience Economy in E-Commerce**
Wei-Lun Chang, Soe-Tsyr Yuan, and Carol W. Hsu

- How Distributed Data Mining Tasks Can Thrive as Knowledge Services**
Domenico Talia and Paolo Trunfio

- ERP: Drilling for Profit in the Oil and Gas Industry**
Jorge A. Romero, Nirup Menon, Rajiv D. Banker, and Mark Anderson

- Why Do People Tag? Motivations for Photo Tagging**
Oded Nov and Chen Ye

- Using ESI Discovery Teams to Manage Electronic Data Discovery**
John C. Ruhnka and John W. Bagby

- Application Service Providers: Market and Adoption Decisions**
Yurong Yao, Edward Watson, and Beverly K. Kahn



Association for Computing Machinery
Advancing Computing as a Science & Profession



Moshe Y. Vardi

DOI: 10.1145/1785414.1785415

Hypercriticality

In the two years since we launched the revitalized *Communications of the ACM*, I have received hundreds of email messages from readers. The feedback has been

mostly, but not universally, positive. Many people do note places where we can do better. Some readers point out errors in published articles. Nothing in life is perfect. *Communications* is an ongoing project; continuous improvement is the name of the game.

At the same time, I have also received a fair number of notes with nothing short of withering criticism. For example, six issues into the revitalized *Communications*, I received this comment from a leading computer scientist: "Although I have looked at every issue and at least glanced at every article, I have not yet found one good one."

Do you find this statement harsh? It surely pales in comparison to this: "The level is unbelievably poor. It reads sometimes like a PR article for big companies. Donation to the ACM seems to be the main reviewing criterion. I would call the policy of ACM scientific prostitution, and I don't want to pay for a prostitute."

I believe most of us have received at some point very harsh reviews—though, hopefully, not that harsh—on papers or proposals we have written. If you are an experienced researcher, you have undoubtedly dealt with papers and proposals being declined. Still, the harsh tone of negative reviews can be quite unsettling even to experienced authors. When I talk to colleagues about this, they just shrug, but I think this phenomenon, which I call "hypercriticality," deserves our collective attention. Other people recently commented on this issue. In

the context of proposal reviewing, Ed Lazowska coined the phrase "circling the wagons and shooting inwards," and John L. King, in a recent CCC blog, referred to such verbal assaults as "Fratricide." Jeff Naughton, referring to conference paper reviewing, said in a recent invited talk that "bad reviewing" is "sucking the air out of our community."

The "hypercriticality" claim is not just based on anecdotes; we actually have data that supports it. Proposals submitted to the Computer and Information Science and Engineering (CISE) Directorate of the U.S. National Science Foundation (NSF) are rated, on the average, close to 0.4 lower (on a 1-to-5 scale) than the average NSF proposal. In his blog entry, King discussed the harmful effects of such harshness.

What is the source of this harshness within our discipline? Here one can only speculate. Let me offer two possible explanations. My first theory refers to the intrinsic nature of our discipline. Computing systems are notoriously brittle. Mistyping one variable

What is the source of this harshness within our discipline?

name can lead to a catastrophic failure. Computing embodies the principle of "For lack of a nail, the kingdom was lost." This makes us eternally vigilant, looking for the slightest flaw. In our eternal hunt for flaws, we often focus on the negative and lose perspective of the positive.

My second theory refers to the sociology of our field. We typically publish in conferences where acceptance rates are 1/3, 1/4, or even lower. Reviewers read papers with "reject" as the default mode. They pounce on every weakness, finding justification for a decision that, in some sense, has already been made. It is particularly easy to be harsh when reviewing proposals. If the proposal is not detailed enough, then the proposer "does not have a clear enough plan of research," but if the proposal is rich in detail, then "it is clear that the proposer has already done the work for which funding is sought."

What is to be done? Remember, we are the authors and we are the reviewers. It is not "them reviewers;" it is "us reviewers." Hillel the Elder, a Jewish scholar, 30 B.C.–10 A.D., said "What is hateful to you, do not do to your fellow." This is known as the Silver Rule in moral philosophy. The Golden Rule, which strengthens the Silver Rule, asserts "do unto others as you would have them do to you." Allow me to rephrase this as the Golden Rule of Reviewing: "Write a review as if you are writing it to yourself." This does not mean that we should not write critical reviews! But the reviews we write must be fair, weighing both strengths and weaknesses; they must be constructive, suggesting how the weaknesses can be addressed; and, above all, they must be respectful.

After all, these are the reviews that we would like to receive!

Moshe Y. Vardi, EDITOR-IN-CHIEF

Don't Ignore Security Offshore, or in the Cloud

MOSHE Y. VARDI'S Editor's Letter "Globalization and Offshoring of Software Revisited" and Dave Durkee's "Why Cloud Computing Will Never Be Free" (both May 2010) failed to address security risks. Vardi's headline promised an update on the questions raised by increased globalization of outsourced software development. Though I knew his main focus was on the economic impact of global outsourcing, I was still disappointed there was no mention of the security challenges posed by the global supply chain for software. Such challenges have prompted the U.S. Departments of Defense and Homeland Security, the SAFECode consortium, and numerous other organizations to commit significant effort to combating threats posed by software of unknown pedigree and provenance, including individual and state-sponsored "insider threats" (such as implanted malicious logic, backdoors, and exploitable vulnerabilities), particularly when developed offshore. See the Government Accountability Office's *Defense Acquisitions: Knowledge of Software Suppliers Needed to Manage Risks* (<http://www.gao.gov/new.items/d04678.pdf>) and the *Report of the Defense Science Board Task Force on Mission Impact of Foreign Influence on DOD Software* (<http://www.acq.osd.mil/dsb/reports/ADA486949.pdf>). Though both focus on software used by DoD, the security issues apply to any organization that relies on outsourced software for critical business or mission functions.

Meanwhile, in an otherwise admirable assessment of the strengths and weaknesses of the cloud computing model of outsourced IT-as-a-service, Durkee likewise failed to mention potential consequences of cloud providers not protecting outsourced computing infrastructure against hackers and malicious code. For example, when discussing transparency, he overlooked the fact that no cloud provider allows its customers to implement intrusion detection or security monitoring ex-

tending into the management-services layer behind virtualized cloud instances. Moreover, these customers have learned not to expect their providers to deliver detailed security-incident, vulnerability, or malware reports.

The management-service layer provides a back channel through which the content of each cloud instance is accessible, not only by providers, but by any attacker able to hack into or implant a kernel-level rootkit. Once "in," the attacker is positioned to exploit the back channel to manipulate or even make full copies of all cloud instances hosted on the compromised platform. Even if customers manage to get their providers to agree to service-level agreements (SLAs) stipulating a high level of vigilance, reporting, and protection below the cloud-instance layer, the management-services layer remains an inherent weakness that should concern anyone looking to host "in the cloud" the kinds of critical applications Durkee explored.

Karen Mercedes Goertzel,
Falls Church, VA

Author's Response:

I strongly agree with Goertzel's sentiment and appreciate her raising this very important issue. The executive summary of the 2006 Globalization and Offshoring Report said: "Offshoring magnifies existing risks and creates new and often poorly understood or addressed threats to national security, business property and processes, and individuals' privacy. While it is unlikely these risks will deter the growth of offshoring, businesses and nations should employ strategies to mitigate them." The Report's Chapter 6, "Offshoring: Risks And Exposures," covered the risks at length.

Moshe Y. Vardi, Editor-in-Chief

Author's Response:

As with performance and uptime, cloud security is determined by the necessity of meeting the terms of SLAs as demanded by customers. As they mature, they will demand even more from their providers'

SLAs by agreeing to industry-standard audits and certifications that ensure they get the security they need, a topic that is a great starting point for another article.

Dave Durkee, Mountain View, CA

Up in the Air

Describing the network effects of a cloud strategy, particularly when it involves SaaS platform efficiency, in his "Technology Strategy and Management" Viewpoint "Cloud Computing and SaaS as New Computing Platforms" (Apr. 2010), Michael Cusumano said that major cloud hosts, including Amazon, Google, and Salesforce, generally rely on detailed SLAs to guarantee security and other parameters for their hosted customers. However, many such hosts, including Amazon SimpleDB and Google Apps, agree to SLAs involving only, perhaps, performance degradation limits and availability of a given service. If cloud-related SLAs fail to include more specific parameters, the cloud infrastructure risks closing itself to new, innovative services due to its lack of dependable guarantees.

Burkhard Stiller and **Guilherme Machado**, Zürich, Switzerland

Diversity Factor

Richard Tapia's inspiring Viewpoint "Hiring and Developing Minority Faculty at Research Universities" (Mar. 2010) said that looking for the next Gauss or Turing is not necessarily the key criterion in all CS faculty searches. I have sometimes sensed confusion between the notion that research excellence drives academic success (it does and should) and what might be called the "additive argument," or belief that maximizing the potential research stature of every new hire automatically maximizes a department's overall excellence in research. I read Tapia's section on reexamining search criteria to mean this is not always the case. I concur, convinced that the effects of talent are not simply additive.

It ought to go without saying that the goal of diversity of gender or ethnic origin does not generally conflict with excellence in research. For instance, in recent years my department has interviewed several women candidates who were uniformly superior to their male counterparts.

However, in specific faculty searches it may be that the potential research stature of a certain white male candidate is perceived as exceeding that of a certain female or minority candidate. The latter may be stellar, but the former's intellectual light shines just a bit brighter. If the discrepancy is comparable to the rather high level of uncertainty inherent in measuring a candidate's potential, some may invoke the additive argument.

However, this argument seems to rest on two questionable assumptions: departmental excellence (however measured) is the arithmetic sum of the individual levels of excellence of its faculty members; and the success of an individual researcher is independent of the surrounding environment.

Both are wrong. Excellence in research (individually or across a department) is a nonlinear function of interdependent factors. For instance, in a department that makes itself attractive to a broader pool of graduate students through the composition of its faculty, all researchers benefit from the resulting potentially improved quality of the department's student body. This also holds when attracting new colleagues, including so-called superstars. When female or minority candidates are at, say, the top of the list in a particular search, they (like everybody else) also consider a department's environment when choosing which job offer to accept. Moreover, a more welcoming, collegial, diverse faculty often leads to better and more frequent collaboration, as well as to more vibrant research.

The question is not whether to compromise between excellence and diversity but how best to foster excellence, with diversity a part of the equation.

Carlo Tomasi, Durham, NC

Wrong Side of the Road

In his Editor's Letter "Revisiting the Publication Culture in Computing Research" (Mar. 2010), Moshe Y. Vardi

said computer science is "the only scientific community that considers conference publications as the primary means of publishing our research results," asking, "Why are we the only discipline driving on the conference side of the 'publication road?'"

As an old timer, I can say that in the early days, there was a belief (conceit might be a better word) that the field's pace of discovery was happening so quickly that only conferences, with subsequent prompt publication of proceedings, could communicate results in a timely manner. As a corollary, the traditional peer-reviewed published literature review fell behind, as it was relieved of temporal pressure through the published proceedings.

These days, the pace of discovery in the biological sciences, including molecular biology, genomics, and proteomics, far exceeds that of computer science. Yet the gold standard of publication in archival journals continues. It is the ultimate irony that computer science, along with various disciplines in the physical sciences, employs the tools developed by computer scientists to ensure timely dissemination of research results through the online editions of their publications. *Science*, *Nature*, *Cell*, and other leading journals routinely present their most important articles in online form first. If, perhaps, computer science would make greater use of its own tools, the shoemaker's children would no longer go barefoot, and published proceedings would fade into its proper historical niche.

Stuart Zimmerman, Houston, TX

More to Celebrate in RDBMS History

Gary Anthes offered good reporting but also some serious errors concerning pre-RDBMS history in his news article "Happy Birthday, RDBMS!" (May 2010), saying "In 1969, an ad hoc consortium called CODASYL proposed a hierarchical database model built on the concepts behind IMS. CODASYL claimed that its approach was more flexible than IMS, but it still required programmers to keep track of far more details than the relational model did."

Please compare with the following basic facts as reported in Wikipedia: "In 1965 CODASYL formed a List Pro-

cessing Task Force. This group was chartered to develop COBOL language extensions for processing collections of records; the name arose because Charles Bachman's IDS system (which was the main technical input to the project) managed relationships between records using chains of pointers. In 1967 the group renamed itself the Data Base Task Group and in October 1969 published its first language specifications for the network database model, which became generally known as the CODASYL Data Model."

The Integrated Data Store (IDS) has been in continuous productive use since 1964, running first on GE 200 computers. In 1966, it began supporting a nationwide, 24/7, order-entry system (OLTP). And in 1969, running on the GE 600, it began supporting a shared-access (OLTP) database, complete with locks, deadlock detection, and automatic recovery and restart.

IBM did not release its IMS/360 (Information Management System) based on the hierarchical data model until September 1969 when future relational databases were still just a gleam in Ted Codd's eye.

B.F. Goodrich received the IDS source code from GE in 1964, renaming it the Integrated Database Management System, or IDMS, when rewritten for the IBM 360 (1969–1971). IDMS was acquired (1973) and marketed worldwide by Cullinane (later Cullinet). IDMS was acquired (1989) by CA (formerly Computer Associates), which still actively supports it worldwide on more than 4,000 IBM mainframes. British Telecom and the Brazilian government are the best-known IDMS users, rated, in 2005, the second- and third-largest OLTP systems in the world.

For more, please see the refereed papers on IDS, IMS, IDMS, and other DBMS products in *IEEE Annals of the History of Computing* (Oct.–Dec. 2009) special issue on "Mainframe Software: Database Management Systems." A future issue is planned to cover more recent RDBMS history.

Charles W. (Charlie) Bachman,

Lexington, MA, ACM Turing Award 1973

Communications welcomes your opinion. To submit a Letter to the Editor, please limit your comments to 500 words or less and send to letters@cacm.acm.org.

© 2010 ACM 0001-0782/10/0700 \$10.00



Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & digital library order form

Priority Code: ACACM11

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

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 acmhhelp@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 _____

In the Virtual Extension

Communications' *Virtual Extension* brings more quality articles to ACM members. These articles are now available in the ACM Digital Library.

IT Programs in High Schools: Lessons from the Cisco Networking Academy Program

Alan R. Dennis, Thomas M. Duffy, and Hasan Cakir

The authors studied 5,392 students taking a course from the Cisco Networking Academy at 764 high schools across the U.S. to understand the factors that influenced their achievement and confidence. Surprisingly, school characteristics (inner city vs. suburban, rich vs. poor) had virtually no impact. What mattered most was instruction quality and an individual student's ability, motivation, and, unfortunately, gender. This style of program, with a strong centralized curriculum, local customization, standards-based testing, and strong teacher support overcame the traditional educational barriers to enable each student to rise to his or her own level of ability and motivation.

Creating the Experience Economy in E-Commerce

Wei-Lun Chang, Soe-Tsy Yuan, and Carol W. Hsu

The potential economic value of experience-oriented offerings has been demonstrated in the physical marketplace. This study suggests the widespread use of the Internet allows the experience economy to be extended to the virtual marketplace. The growing practice of online collaborative design demonstrates the potential for providing the experience economy via the Internet. The authors propose expanding the existing practices by incorporating the concept of collaborative pricing into the design of experience offerings, as demonstrated in their iCare platform. The article is intended to motivate further research into the development of the experience economy in e-commerce.

How Distributed Data Mining Tasks Can Thrive as Knowledge Services

Domenico Talia and Paolo Trunfio

Through a service-oriented approach we can support distributed business intelligence tasks in clouds and grids. Those services can implement all the tasks in data mining and in knowledge discovery processes such as data selection, data analysis, and knowledge representation. The authors explore architectures and services for distributed knowledge discovery such as the Knowledge Grid, the Weka4WS toolkit, and mobile data mining services. The article describes a strategy and a model based on the use of services for the design of distributed knowledge discovery applications and discusses how grid and cloud frameworks can be developed as a collection of services and how they can be used to support knowledge discovery processes using the SOA model.

ERP: Drilling for Profit in the Oil and Gas Industry

Jorge A. Romero, Nirup Menon, Rajiv D. Banker, and Mark Anderson

The article presents research that applies to a new approach toward understanding ERP implementation. Rather than looking at ordinary measures of firm performance, the authors examine strategic performance measures that can only be utilized if one delves into data that is not found on the financial statements. It is one of the first studies to show the sources of profitability after an ERP implementation and will help managers understand the strategic and managerial implications of ERP implementation.

Why Do People Tag? Motivations for Photo Tagging

Oded Nov and Chen Ye

Given the growing popularity of tags as a means of sharing and organizing large amounts of data, it is critical for developers and managers of collaborative content-sharing systems such as Flickr, YouTube, and del.icio.us to understand what motivates users to tag. The authors examine individual-level motivations using a newly developed scale as well as the social presence driver and the number of user photos. The findings suggest that both social presence and individual-level motivation affect users' tagging level.

Using ESI Discovery Teams to Manage Electronic Data Discovery

John C. Ruhnka and John W. Bagby

Many organizations face litigation threats with associated crippling costs, staff-time demands, and adverse financial impacts. "Discovery," the legal and operational process governing the evidentiary use of electronically stored information (ESI)—including email messages—plays a central role in the cost of litigation as well as in potential outcomes. Multidiscipline ESI "discovery teams" containing key IT, legal, and operational players involved in this complex process can more effectively manage the "litigation hold" process, and can better assess the potential costs of alternative strategies in collection, identification, verification, recovery, and production of relevant electronic data sought as evidence.

Application Service Providers: Market and Adoption Decisions

Yurong Yao, Edward Watson, and Beverly K. Kahn

Deciding whether, how, and with whom to outsource an organization's applications is important. Key factors influence the Application Service Provider (ASP) decision-making process and the ultimate organizational success. The authors examine the current ASP market and recommend evaluation criteria by looking at hosting by "Pure ASPs" (small companies who purely provide hosting services) and "ISVs" (independent software vendors, who develop software and host their own applications). These ASPs provide either vertical (within a single industry sector) or horizontal (across industries) applications. Several adoption cases are presented to explain the recommendations.

The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.



Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/1785414.1785419

<http://cacm.acm.org/blogs/blog-cacm>

Software Development and Crunch Time; and More

Ruben Ortega discusses developers and crunch time; Mark Guzdial considers the impact of open source practices on computing education; and Daniel Reed writes about the technological shift from computational paucity to computational plethora.



Ruben Ortega's "Why Do Software Developers Tolerate 'Crunch Time'?"

<http://cacm.acm.org/blogs/blog-cacm/70922>

Given the increased risk of burnout for an extended "crunch time," why do developers put up with it?

For software developers, crunch time is a period prior to a major product milestone when team members are asked to put in extra effort to get a product finished by a specific delivery date. Practically, this can be a horrific period of 80-plus hour weeks that goes on for months as the team scrambles to deal with bugs, last-minute feature requests and modifications, and milestones. For game companies and large Internet retailers in particular, the mantra of "Christmas never slips" means that crunch time occurs during the summer so products can be released by the fall and be available between Thanksgiving and Christmas. Recently, the wives of

Rockstar Games posted an open letter to the company's management about the impact of the crunch time on their lives. The company was demanding 6–7 days a week, with 12–16 hour days. The impact included mental, physical, and emotional strain on the employees and their families.

Reading the discussions on Slashdot about Rockstar Games' working conditions highlighted that this problem is industrywide. As a developer and manager, I have worked on a number of projects at various startups that involved periods of crunch time that lasted longer than I thought was realistic. When I was a young engineer working on Amazon Auctions, doing the all-nighter was a badge of honor. Eventually, I discovered most of the code you write during those A.M. hours will likely be thrown away. After a few crunch times, I learned to be a better self-advocate, and was able to sensibly set expectations of what combination of features, quality, and testing I could deliver by a given date. When

I made the transition from developer to manager, I was glad to have had the experience so I could advocate for my teams. Although I couldn't always get rid of crunch times, I worked to keep their durations as short as possible.

Why do developers put up with crunch time? I believe the reason is as simple as "progress." "Progress" was the factor that was most important to 12,000 workers, according to two researchers who analyzed the workers' diary entries.

As long as the workers believe they are making headway in delivering their product, they are getting an intrinsic reward that motivates them to work more. If you have a team making progress on a delivery, the combined effort of the team will self-reinforce and encourage them all for their efforts. On Amazon Auctions, I worked on implementing search for the system and would nap while another team member would deliver new catalog content. By the time I returned, we would integrate our code, which would result in a complete auction search results. The work was rewarding despite our working through weekends to complete the project. The progress was beautiful and easy to see. One day the system had mockups for search results and the next day the results would be feeding from live data. The intrinsic reward of making progress and working with the team to deliver helped combat the potential for burnout.

It is unrealistic to deliver any project without going through some crunch time. Although progress helps to motivate employees during those periods,

ineffective project planning can lead to an egregious amount of time where progress alone will not be enough to sustain the employees' motivation. If excessive crunch time continues to occur, the employees—the company's most valuable resource—should work to either change the organization or they will be compelled to move to a more supportive company. The books *Peopeware* and *Slack: Getting Past Burnout, Busywork and Total Efficiency* are great reminders on why we should work hard to take care of our teams.



**Mark Guzdial's
"The Impact of Open
Source on Computing
Education"**

[http://cacm.acm.org/blogs/
blog-cacm/72144](http://cacm.acm.org/blogs/blog-cacm/72144)

We had a Georgia Tech alum, Mike Terry (now at Waterloo) visit us a couple weeks ago. Mike's research is on usability practices in open source. I got a chance to chat with Mike, and we talked about the impacts of open source on computing education, such as high school students getting started with computing by working in open source development. Overall, though, I came away concerned what the growth of open source development means for the future of computing education.

At a time when we are trying to broaden participation in computing, open source development is even more closed and less diverse than commercial software development. It is overwhelmingly white, Asian, and male. Some estimates suggest that less than 1% of open source developers are female.

Many kids and parents worry that all computer science jobs are being offshored and that it's not worth studying computing. As more and more of the software we use daily is created via open source development, I wonder if kids and parents will hear the message, "Most software developers work for free, or at least have to work for free for years before they can become professional and get paid for their work." Of course, that's not true. Neither is it true that all IT jobs are being offshored, but that's still what some people believe.

One of our challenges in computing education is convincing people that computing is broad and about more than programming. Open source

values code above all, or as Linux's originator Linux Torvalds said, "Talk is cheap. Show me the code." We're trying to convince students that talk is *also* valuable in computing.

Finally, Mike's talk was about how common usability practices are rare in open source development. Of course, that's a concern in itself, but it's particularly problematic for newcomers. When students develop toward being professionals, they frequently engage in a process that educators call legitimate peripheral participation (LPP). It's LPP when you start out in a company picking up trash (doing something legitimate on the periphery), and in so doing, figure out what happens in the company. Students can get started in software development at a company by doing tasks that aren't directly about writing software, but are about the whole enterprise. These legitimate peripheral tasks serve as a stepping stone into the process, like writing documentation or running subjects in usability testing. If you don't have usability testing, you don't have that path into the process. Breaking into an open source development process is hard, and that keeps more students out than invites them in.

I wrote on this topic in my regular blog, and was surprised at the response. I learned that it is not acceptable to criticize religion, Santa Claus, or open source development—it's a "good" that should just be accepted as such. I disagree. Open source development does generate enormous good, but it could do *more* good if it improved its practices. It's hard to change open source development, because of its distributed nature. Open source developers should worry about the messages they send future developers, especially if they hope to grow and attract the development talent pool.



**Daniel Reed's
"Paucity to Plethora:
Jevons Paradox"**

[http://cacm.acm.org/blogs/
blog-cacm/72373](http://cacm.acm.org/blogs/blog-cacm/72373)

Those of us of a certain age remember when the university computer (note the singular) was a scientific and engineering shrine, protected by computer operators and secure doors. We acolytes extended offerings of FORTRAN, ALGOL, or COBOL via punched card decks, hoping for the

blessings that accrued from a syntactically correct program that compiled and executed correctly.

The commonality across all our experiences was the need to husband computer time and plan job submissions carefully, particularly when one's job might wait in the queue for six to 10 hours before entering execution. I distinctly remember spending many evenings laboriously examining my latest printout, identifying each syntax error and tracing the program flow to identify as many logic errors as possible before returning to the keypunch room to create a new punched card deck.

Because computing time was scarce and expensive, we devoted considerable human effort to manual debugging and optimization. Today, of course, my wristwatch contains roughly as much computing power as that vintage university mainframe, and we routinely devote inexpensive computing time to minimize human labor. Or do we?

Yes, we routinely use WIMP interfaces for human-computer interaction, cellular telephony is ubiquitous, and embedded computers enhance everyday objects. However, I suspect much of computing is still socially conditioned by its roots in computational paucity to fully recognize the true opportunity afforded by computational plethora.

Many of us are wed to a stimulus-response model of computing, where humans provide the stimulus and computers respond in preprogrammed ways. In a world of plethora, computing could glean the work, personal, and emotional context, anticipating information queries and computing on behalf rather than in response. My computer could truly become my assistant.

In economics, the Jevons paradox posits that a technological increase in the efficiency with which a resource can be used stimulates greater consumption of the resource. So it is with computing. I believe we are just at the cusp of the social change made possible by our technological shift from computational paucity to computational plethora. □

Ruben Ortega is an engineering director at Google. **Mark Guzdial** is a professor at the Georgia Institute of Technology. **Daniel Reed** is vice president of Technology Strategy & Policy and the eXtreme Computing Group at Microsoft.

© 2010 ACM 0001-0782/10/0700 \$10.00



DOI:10.1145/1785414.1785420

David Roman

In Case You Missed It

The most popular content on *Communications'* site is something many readers know nothing about. The BLOG@CACM (<http://cacm.acm.org/blogs/blog-cacm>) is original online material that does not appear in the monthly magazine except in abbreviated form (see page 10). Eleven entries from this blog were among the site's 100 most popular articles in the first 14 months following its makeover, and seven landed in the top 50 (see below). That's more articles than from any single section of the monthly magazine.

These frontrunners show the strength and diversity of the blogs. Most were penned by regular contributors, but a couple were filed by guest bloggers from a major ACM conference. *Communications* is always looking for new bloggers: guests as well as new experts.

The blogs generate a strong share of comments. In most of the cases cited here, the author is an active participant in the discussions, responding to questions, refining points, and bringing an immediacy and level of engagement the magazine cannot match.

While it is clear that many readers are finding this content, it is also clear that many are not. A recent survey found that 40% of readers didn't know or had no opinion about the BLOG@CACM, and 61% said the same about the site's blogs overall (it also publishes a Blogroll at <http://cacm.acm.org/blogs/blogroll>). If you are in the 'don't know' category, here's what you've missed. If you are interested in blogging for the BLOG@CACM as a regular contributor or from a conference, email blog@cacm.acm.org.

Title/URL	Author/URL	# of User Comments
How We Teach Introductory Computer Science is Wrong http://cacm.acm.org/blogs/blog-cacm/45725	Mark Guzdial http://www.cc.gatech.edu/~guzdial/	15
The End of a DBMS Era (Might Be Upon Us) http://cacm.acm.org/blogs/blog-cacm/32212	Michael Stonebraker http://www.csail.mit.edu/user/1547	6
The 'NoSQL' Discussion Has Nothing to do With SQL http://cacm.acm.org/blogs/blog-cacm/50678	Michael Stonebraker http://www.csail.mit.edu/user/1547	7
Errors in Database Systems, Eventual Consistency, and the CAP Theorem http://cacm.acm.org/blogs/blog-cacm/83396	Michael Stonebraker http://www.csail.mit.edu/user/1547	10
Extreme Agility at Facebook http://cacm.acm.org/blogs/blog-cacm/51564	E. Michael Maximilien http://www.maximilien.com/homepage/about_me.html	4
What is a Good Recommendation Algorithm? http://cacm.acm.org/blogs/blog-cacm/22925	Greg Linden http://glinden.blogspot.com/	10
Clay Shirky: Doing work, or Doing Work? http://cacm.acm.org/blogs/blog-cacm/72609	Michael Bernstein http://people.csail.mit.edu/msbernst/	4

ACM Member News

ALAIN CHESNAIS ELECTED ACM PRESIDENT



Alain Chesnais, owner of Visual Transitions, a Toronto-based consulting company specializing in

computer graphics and social networks, was elected president of ACM in the May 2010 general election.

A longtime ACM volunteer, Chesnais views the key challenges facing ACM as "our becoming a truly international organization and attracting younger members into the organization." Specifically, Chesnais wants to both strengthen ACM's presence in China and India and "to do much more in terms of expanding our online presence to better cater to the needs of younger researchers and practitioners."

The ACM General Election results include:

PRESIDENT

Alain Chesnais,
Visual Transitions

(term: July 1, 2010–June 30, 2012)

VICE PRESIDENT

Barbara G. Ryder, Virginia Tech

(July 1, 2010–June 30, 2012)

SECRETARY/TREASURER

Alexander L. Wolf,
Imperial College London

(July 1, 2010–June 30, 2012)

MEMBERS AT LARGE

Vinton G. Cerf, Google

(July 1, 2010–June 30, 2014)

Salil Vadhan, Harvard University

(July 1, 2010–June 30, 2014)

A policy passed by the ACM Council in October 1980 calls for publication of the number of votes polled by each candidate.

President

Alain Chesnais	5,277
Joseph A. Konstan	3,344

Vice President

Barbara Ryder	5,743
Norman Jouppi	2,936

Secretary/Treasurer

Alexander L. Wolf	5,371
Carlo Ghezzi	3,162

Members at Large

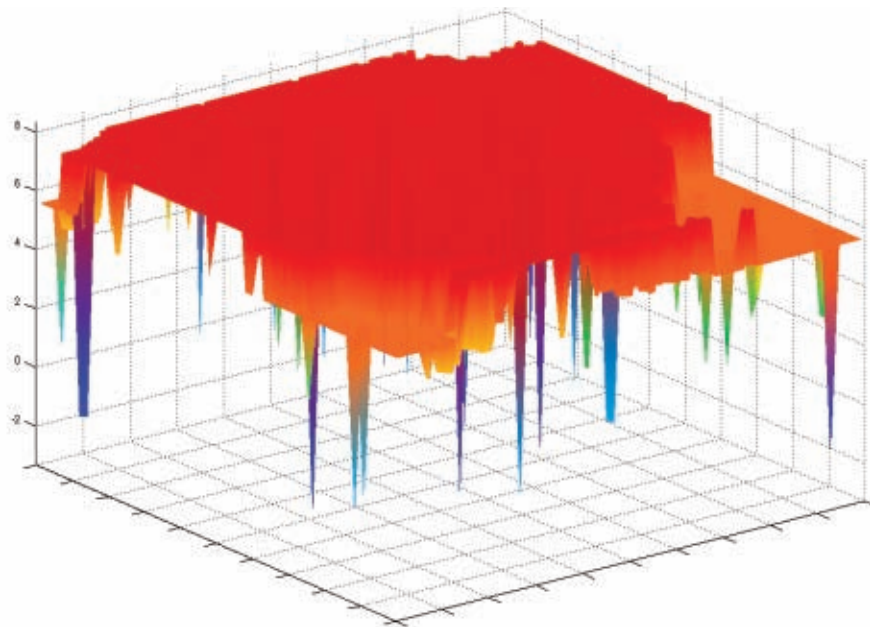
Vinton Cerf	6,485
Salil Vadhan	4,113
Satoshi Matsuoka	3,445
Fei-Yue Wang	2,664

Sharing Computational Perspectives

Computer scientists are now making intellectual contributions to a wide range of other disciplines, including evolutionary theory, physics, and economics.

WHY BOTHER WITH sex? Venturing forth in search of a mate can be dicey business, and even if you succeed, you can pass on only half of your genes, which are randomly combined with half of your mate's, to your offspring. For more than a century, biologists have suggested that mixing up genes is exactly what sexual reproduction contributes to evolution, making it easier for novel gene combinations to appear. But the downside, also long understood by biologists, is that what gene mixing gives, it also takes away: Sexual reproduction can break up a winning gene combination as easily as it can create one.

Nonetheless, among plants, fungi, and animals, sexual reproduction is far more popular than asexual reproduction, a fact for which biologists have no thoroughly satisfactory explanation. In the December 2008 issue of *Proceedings of the National Academies of Science*, computer scientist Christos Papadimitriou and biologist Adi Livnat of the University of California, Berkeley teamed with biologists Jonathan Dushoff of McMaster University and Marcus Feldman of Stanford University to offer



In the Avida artificial life software program, bits of code act as organisms. As new computationally beneficial genotypes evolve, their fitness rises and they sweep through the population (hence the general height and color increase over time).

a mathematical model that sheds some light on this puzzle. They imagined a simple system with two genes, each of which comes in three versions, and with different fitnesses (that is, survival values) assigned to the nine possible

combinations of the two genes. From a starting population in which all gene combinations are equally likely, the model found that asexual reproduction, which gives offspring the same genes as their parents, quickly led to the domina-

tion of the particular gene combination with the greatest fitness. But sexual reproduction, which switches genes from one generation to the next, leads to a different outcome: the gene versions that dominate are those that give the greatest average fitness no matter which other gene version they are paired with.

Sex, in other words, promotes genes that are good mixers with other genes, and a versatile gene may be more valuable, from an evolutionary perspective, than one that works very well in some combinations but much worse in others. Hence, the new model provides a specific and quantitative accounting of one useful aspect of sex.

Today, computer scientists are doing far more than helping other scientists run their numerical models more effectively. The theory of computation has become such a sophisticated science in its own right that computer scientists are now making intellectual contributions to a wide range of other disciplines, including evolutionary theory, physics, and economics.

Evolutionary Theory

Over the years, evolutionary theory has drawn on ideas from engineering, mathematics, and physics, so it's hardly surprising that computer scientists are now also making substantial contributions. "Evolution is such a broad topic [that] it really benefits from a range of approaches," says Sally P. Otto, an evolutionary biologist at the University of British Columbia. One noteworthy example of collaboration by biologists and computer scientists Otto cites is the

Evolutionary theory has drawn on ideas from engineering, mathematics, and physics. Now computer scientists are also making substantial contributions.

artificial life software platform Avida, in which bits of code act as "organisms" able to replicate and evolve. If the model is setup so that digital organisms gain more processing speed by being able to add two numbers, for example, then the ability to add appears and proliferates, even though it was not specifically programmed in.

But if computer science is addressing the puzzle of sex, then sex has also done something for computer science, in the form of genetic algorithms that address optimization problems by evolving candidate solutions in a manner inspired by genetic recombination. However, these algorithms are not always good at reaching the solution a computer scientist wants. "Nature's favorite trick yields bad optimizers," says Papadimitriou, and it is this mystery that got him

started on the research that led him to conclude that sex promotes mixability, not fitness. That distinction illustrates a difference in perspective; to a computer scientist, an algorithm has performed its job once it has produced a solution. To a biologist, on the other hand, a "solution" is a stable configuration that doesn't just appear once, but survives and prospers over many generations. Because evolution is so complex and, by definition, an ongoing process, "biologists have given up on having complete solutions," Otto says.

Solving difficult problems, of course, is a basic aim of computer science, whose defining characteristic is "conquering complexity," says Papadimitriou. The theory of computational complexity, which classifies problems by the resources needed to solve them, was pioneered by computer scientists in the face of general indifference from traditional mathematicians. The discovery of depth in computation, says Papadimitriou, allowed computer science to give birth to big questions, bestowing it with an intellectual respectability that mathematicians now fully embrace.

Insights from computer science have illuminated the mathematical properties of networks, broadly defined, leading to applications ranging from sociology to physics. Networks can exhibit phase transitions—changes from one kind of macroscopic behavior to another—that share deep similarities to phase transitions in physics, such as a liquid freezing into a solid. Belief propagation, for example, is an algorithm for performing statistical inferences that

Data Management

The Digital Universe Keeps Expanding

Between 2010 and 2020, the amount of digital information created and replicated in the world will grow to 35 trillion gigabytes as the major types of media—print, radio, TV, and voice—make the transition from analog to digital, according to a new IDC report *The Digital Universe Decade—Are You Ready?*

Written by John F. Gantz and David Reinsel, the report notes the amount of digital information created and

replicated in the world, which IDC calls "the Digital Universe," grew by 62% last year to nearly 800,00 petabytes, despite a global recession. This year, IDC expects the Digital Universe will grow almost as fast to 1.2 million petabytes. This steady growth means the digital universe in 2020 will be 44 times as big as it was in 2009. Meanwhile, the number of information containers—packets, files, images, and so on—in 2020

will be 25 quintillion.

"Although the amount of information in the Digital Universe will grow by a factor of 44, and the number of containers or files will grow by a factor of 67 from 2009 to 2020, the number of IT professionals in the world will grow only by a factor of 1.4," according to IDC.

The relevant issues that must be considered, IDC says, include "developing tools for research and discovery of information as

the Digital Universe expands, including finding ways to add structure to unstructured data through metadata, automatic content tagging, and pattern recognition"; deploying tools for new levels of information management and prioritized storage; and deploying tools and expertise for security and privacy projection for a growing portion of the Digital Universe in hybrid physical/virtual environments."

—Jack Rosenberger

turns out to contain concepts closely related to fundamental principles of statistical thermodynamics, so that knowledge gained by physicists has influenced how computer scientists understand some types of computation.

Economics and Game Theory

Early on, computer scientists appreciated game theory and economic modeling. That connection dates to at least the 1940s and the pioneering work of John von Neumann, but has grown much deeper in recent decades. Economists talk of Pareto optimums, in which no participant in a system can perform better without making another participant worse off, and of Nash equilibria, in which no participant can make a unilateral change of strategy that will bring an advantage, and they have proved theorems showing that, under certain assumptions, Pareto optimums and Nash equilibria must exist. But the existence of such a state is no guarantee that a realistic system can actually attain it. For example, computer scientists have shown that Nash equilibria are computationally intractable problems, meaning not only that mathematical models of economic markets are hard to solve, but that the extent to which human behavior can move a market toward a Nash equilibrium becomes questionable.

Moreover, Pareto optimality and Nash equilibria do not necessarily connote social or political desirability. Nash equilibrium represents the best that participants in a market can do by pursuing a wholly selfish strategy. Papadimitriou has coined the term “price of anarchy” for the fact that Nash equilibrium often gives participants a poorer outcome than the best they could have obtained through collaboration, while work by other computer scientists not only shows how to calculate this price, in certain models, but also indicates that the price may not be too high.

There are challenges, however, in getting academic disciplines to collaborate, says Lance Fortnow, a professor of electrical engineering and computer science at Northwestern University. “Computer scientists want to impress other computer scientists,” Fortnow says, and economists possess the same attitude. But cooperation between the two fields is growing, and an increasing number of economists see computer science as of-

fering ideas they can draw from, not only in the technical matter of making mathematical models work efficiently, but also as a source of insight into the behavior and properties of those models.

Fortnow notes that companies like Google, Microsoft, and Yahoo! have understood the importance of economics and computer science in developing products and services, and have assembled teams in which both types of scientists work side by side. This mixing of talents is happening at some universities, and the National Science Foundation is considering a new program in economics and computer science.

Certain ideas of computer scientists are catching on among economists, says Fortnow. He cites the auctioning of sections of the radio spectrum by the Federal Communications Commission as the type of economic game in which “computational issues get nasty,” because different players want different parts of the spectrum for a variety of exclusive purposes. Computer science can explain how complexity manifests itself in such a situation, and can suggest mechanisms to make an auction more orderly and useful. Thus far, however, it’s hard to make the case that computer science has changed economic theory in any significant ways but, Fortnow urges, “give it time!” **□**

Further Reading

Otto, S.P.

The evolutionary enigma of sex, The American Naturalist 174, S1, July 2009.

Lenski, R.E., Ofria, C., Pennock, R.T., and Adami, C.

The evolutionary origin of complex features, Nature 423, 139, May 8, 2003.

Yedidia, J., Freeman, W.T., and Weiss, Y.

Understanding belief propagation and its generalizations, Exploring Artificial Intelligence in the New Millennium,

Lakemeyer, G., Nebel, B. (eds), Morgan Kaufmann Publishers, San Francisco, CA, 2003.

Fortnow, L. and Gasarch, B.

Computational Complexity blog
<http://blog.computationalcomplexity.org/>

Kalai, E., Jackson, M.O., Lehrer, E., Palfrey, T.R., and Parkes, D.C. (eds.)

Games and Economic Behavior

<http://www.elsevier.com/wps/locate/geb>

David Lindley is a freelance science writer in Alexandria, VA.

© 2010 ACM 0001-0782/10/0700 \$10.00

Milestones

SIGSOFT Awards

The ACM Special Interest Group on Software Engineering (SIGSOFT) recently presented its highest awards to several computer scientists whose contributions in research, education, and service have shaped the development of software engineering and its ability to solve complex computational problems.

Eric Gamma, Richard Helm, Ralph Johnson, and (posthumously) John Vlissides received the 2010 SIGSOFT Outstanding Research Award for contributions to the practice of software engineering. Their landmark book *Design Patterns* explored the capabilities and pitfalls of object-oriented programming and catalogued 23 specific solutions to common design problems. Gamma is technical director of the Software Technology Center of Object Technology International in Zurich, Switzerland. Helm is a partner and managing director of the Boston Consulting Group in Sydney, Australia. Johnson is on the faculty of the University of Illinois Urbana-Champaign department of computer science. Vlissides, a researcher at IBM T.J. Watson Research Center, passed away in 2005.

The 2010 SIGSOFT Influential Educator Award was presented to Leon Osterweil, a computer science professor at the University of Massachusetts Amherst, for pioneering work in graduate-level education in software engineering. He launched the New Software Engineering Faculty Symposium in 1991 at the International Conference on Software Engineering to encourage new software engineering faculty early in their careers.

Mary Lou Soffa, who chairs the department of computer science at the University of Virginia, received the 2010 SIGSOFT Distinguished Service Award for her extensive service in the software engineering community and her commitment to strengthening ties with her colleagues across the programming languages community. Soffa has served on the community’s leading editorial boards, advisory and steering committees, and conferences.

Censored!

Countries use Internet censorship to dominate the political dialogue, but also to create favorable conditions for government-controlled businesses.

GOVERNMENTS ATTEMPTING to strangle access to sensitive information are nothing new. Throughout history, authorities have seized printing presses, jammed radio broadcasts, and blocked television programming in order to control hearts and minds. “In many countries, censorship is a deeply entrenched practice,” notes Andrew Lih, visiting professor of new media at the University of Southern California’s Annenberg School for Communication and Journalism.

However, in the age of the Internet, the stakes are higher and the challenges associated with controlling information are greater. According to Reporters Without Borders, a dozen countries—including China, Iran, and Saudi Arabia—are on its annual “Enemies of the Internet” list, which is based on the number of citizens arrested, harassed, or threatened in the previous year for their online activities and on how the nations monitor the Internet and limit access. Eleven other countries made Reporters Without Borders’ “Under Surveillance” list, and even nations that trumpet freedom sometimes rely on censorship techniques. In addition, Reporters Without Borders states that 118 bloggers and other netizens are currently residing in jails because of content they have posted.

In fact, a growing number of governments use methods—including Domain Name System blocking, Internet Protocol blocking, or Uniform Resource Locator keyword filters (see “How Censorship Works” on right)—to make popular Web sites, such as Facebook, YouTube, and Twitter, inaccessible to their citizens. Some also force Google and other search engines to self-censor their results. In most instances, the goal is to control the political dialogue, but authorities also use these techniques to create favorable conditions for govern-



The headquarters of Google China before the search engine company exited the country earlier this year and started routing searches from China through its Hong Kong site.

ment-controlled businesses and others operating in their country.

Not surprisingly, citizens of these countries are increasingly turning to software tools that circumvent blocks and filters through the use of proxy servers or virtual private networks (VPNs). “Governments impose blocks and restrictions and people try and often succeed in finding ways around them,” observes Vadim Isakov, scholar in residence at Ithaca University’s School of Communications.

A New Era of Openness?

When the Internet achieved a tipping point of popularity in the mid-1990s pundits argued that it would usher in a new era of openness. For the most part, this prediction has proved true. However, the challenges associated with filtering and blocking content haven’t stopped many governments from imposing restrictions. China, Turkmenistan, Uzbekistan, Saudi Arabia, and Iran are among the most aggressive

censors, says Isakov.

Yet even countries such as France, Germany, Poland, Thailand, and Turkey have turned to censorship—sometimes forcing Google to restrict access to sites, files, and reports. The aforementioned European countries, for example, ban materials that support Nazi causes, and Thailand won’t allow unflattering material—including videos—of its monarch. At the same time, India has ordered Google to remove content that the government flags as “indecent, immoral, or threatening the public order.”

Google recently brought the issue of Internet censorship to the forefront due to its troubled relationship with the Chinese government. In 2006, the search engine provider introduced a China-based Google.cn search page with censored results. Many observers criticized Google for bowing to pressure from the Chinese government. By early 2009, China began blocking Google’s YouTube site and other on-

line services. Finally, in January 2010, Google indicated it wasn't willing to censor search results and was considering pulling out of China.

After a series of hacker attacks originating from China in February and unsuccessful negotiations with the Chinese government, Google closed its China search site and began rerouting searches from China through its Hong Kong site. However, any searches conducted from China come back censored. While many applauded the decision to halt censored service in China, Google also was criticized by Chinese officials and citizens.

In fact, no nation has received as much public scrutiny as China and its so-called "Great Firewall." Only a few fiber-optic cables—think of them as checkpoints—manage data flowing into the country. Nevertheless, 99% of content flows through untouched, Lih says. In most instances, there's no easy or definable way to determine exactly what's being blocked and what citizens can access.

That's because China, like many countries, blocks sites sporadically and in no systematic way. "For one thing, there's no official policy or admission that the practice is taking place in China. For another, there's no documentation," Lih says. The net effect for those browsing the Web or attempting to connect to a blocked service is a "Connection Reset" message that the requested Web page or service is unavailable. "It looks as though you've encountered technical difficulties or a

temporary disruption," Lih says.

Worse, the interruptions are erratic and unpredictable. A news site such as CNN or BBC, for example, might be available at one moment but disappear seconds later if a story about a sensitive subject, such as Tibet or Tiananmen Square, is published.

Observers say these interruptions, particularly in China, wane with the timing of important events attracting foreigners, such as the Olympics or a large international business conference. In addition, hotels, which are typically the places where foreign visitors stay or work, and Internet service providers alter service and tweak routers in order to provide open access to visitors. This provides a very different experience of the country's Internet accessibility for foreign visitors compared to that of the average citizen.

A few countries have taken far more draconian measures. In North Korea, for example, Internet access is almost nonexistent, with only a few high-ranking government officials and foreign diplomats able to use it. Saudi Arabia has blocked more than 400,000 Web sites about religion, women's issues, Israel, and a slew of other topics. And Uzbekistan blocks all content centering on government corruption, criticism of the authoritarian regime, ethnic strife, and human rights.

Yet, what sometimes appears to be censorship is actually rooted in economics, Lih says. "Governments block certain services in order to give their own companies a competitive advan-

Nanotechnology

Molecular 'Robots' Advance

A multidisciplinary team from Columbia University, Arizona State University, the University of Michigan, and the California Institute of Technology (Caltech) have created and programmed "robots" the size of a single molecule that can independently move across a nano-scale track. First reported in *Nature*, the development marks an important advancement in the fields of molecular computing and robotics, and could contribute to the development of molecular robots that can sense their environment, repair individual human cells, or assemble nanotechnology products.

The project was led by Milan N. Stojanovic, a faculty member in the division of experimental therapeutics at Columbia University, and included Erik Winfree, associate professor of computer science at Caltech; Hao Yan, professor of chemistry and biochemistry at Arizona State University; and Nils G. Walter, professor of chemistry at the University of Michigan in Ann Arbor.

In recent years, scientists have worked to create robots that can reliably perform useful tasks, but at a molecular level. This involves reprogramming DNA molecules to perform in specific ways, and often involves "researchers at the interface of computer science, chemistry, biology, and engineering," says Mitra Basu, a program director at the National Science Foundation, which partially funded the project.

Now, Stojanovic's research team has created molecular robotic "spiders" that can move autonomously through a two-dimensional landscape and act in basic robotic ways, demonstrating that they are capable of starting motion, walking, turning, and stopping.

While the field of molecular robotics is just emerging, it is possible these tiny devices may have important medical applications. "This work one day may lead to effective control of chronic diseases such as diabetes or cancer," Basu says.

How Censorship Works

Governments' techniques for denying access to online information include:

Domain Name Service block

Name lookup fails or an Internet service provider redirects it to another site.

Internet Protocol (IP) block

This approach forbids packets to a specific host based on IP address. It usually results in a "timed out" error message.

Uniform Resource Locator (URL) keyword block

A sensitive word or specific context contained in the destination Web page triggers a URL block. Images and links may fail to load. This occurs frequently when using Google and other search engines.

Web content keyword triggers block

Specific keywords result a "Connection Interrupted" error. These blocks are often temporary and difficult to replicate. They appear to be a technical Internet problem.

tage," he notes. As a result, numerous YouTube and Twitter knockoffs exist in some countries, including China, and other countries block services such as Skype in order to protect government-run telephone services and other businesses.

Breaching Censors' Walls

Increasingly, students, dissidents, journalists, bloggers, human rights advocates, and others are challenging Internet censorship. In the digital age, they're fighting back with an arsenal of tools such as proxy servers that circumvent filtering by masking the user and altering the way data flows to a Web server or by VPNs that tunnel through to other less censorious countries.

It's a cat-and-mouse game, to be sure. As individuals begin downloading and using proxy servers, it's common for government censors to detect the activity and block downloads, as well as the proxies. However, the same applications frequently become available at alternative sites and through peer-to-peer services. Moreover, new proxy servers continually spring up. But VPNs represent a different challenge, and most censoring countries are hesitant to block them because they're essential for commerce and used heavily by foreign business leaders and diplomats.

Some services, like Tor, a free program offered by the nonprofit Tor Project, circumvent censorship by routing communications across a distributed network of relays located around the world. These VPN tunnels make it possible to access blocked pages and sites,

China, like many countries that practice Internet censorship, blocks Web sites sporadically and in no systematic way.

such as Facebook or YouTube. They also allow journalists and dissidents to publish Web sites and other services without revealing their location.

Another free VPN application, AnchorFree Hotspot Shield, enables users to view otherwise censored Web sites by converting the http protocol to an encrypted https protocol and providing users with a virtual identity. "Although the product was originally intended to serve as a universal privacy and security offering, a growing number of people are looking for a way to bypass controls and access information freely," says AnchorFree CEO David Gorodyansky.

Both Tor Project and AnchorFree's Web sites are blocked in China. Nevertheless, Gorodyansky claims that usage in China has doubled since the Chinese government began blocking the site last summer. Potential users download Tor and AnchorFree Hotspot Shield by visiting mirror sites or

by sending the companies an email message and receiving a message with a Windows- or Mac-compatible file. In addition, individuals share applications with USB flash drives and through peer-to-peer services.

The battle over censorship is likely to intensify as the world becomes more Web-centric. Despite tools for piercing and circumventing firewalls, authorities are constantly searching for new and better ways to filter and block traffic, Lih notes. "The only reason that authorities aren't more aggressive in tracking down those who circumvent restrictions is that it simply isn't worth the trouble," he says. "There isn't a critical mass of population that's dangerous to the government." ■

Further Reading

Deibert, R.J., Palfrey, J.G., Rohozinski, R., Zittrain, J. (Eds.)

Access Denied: The Practice and Policy of Global Internet Filtering, MIT Press, Cambridge, MA, 2008.

Fallows, J.

"The Connection Has Been Reset," *The Atlantic*, March 2008.

Farrar, L.

Cashing in on Internet censorship, CNN, February 19, 2010, <http://www.cnn.com/2010/TECH/02/18/internet.censorship.business/index.html>.

Reporters Without Borders

Enemies of the Internet, Reporters Without Borders, March 12, 2010, http://www.rsf.org/IMG/pdf/Internet_enemies.pdf.

Samuel Greengard is an author and journalist based in West Linn, OR.

© 2010 ACM 0001-0782/10/0700 \$10.00

Social Media

CMU Researchers Analyze Twitter Sentiments

A computer analysis of people's sentiments in a billion Twitter messages during 2008 and 2009 yielded measures of consumer confidence and presidential job approval similar to those of public opinion polls, according to Carnegie Mellon University researchers.

The findings suggest that analyzing the text in tweets could be an inexpensive, rapid means of gauging public opinion on

some subjects, says Noah Smith, assistant professor of language technologies and machine learning at Carnegie Mellon. However, the tools for extracting public opinion from the text of social media are crude and social media remain in their infancy, Smith says, so the extent to which these methods could supplement or replace traditional public opinion polls is unknown.

The findings were presented

in May at the Association for the Advancement of Artificial Intelligence's International Conference on Weblogs and Social Media in Washington, D.C.

Smith and colleagues used simple text-analysis techniques to identify messages that pertained to the economy or politics and then found words in the text that indicated if the writer expressed positive or negative sentiments. The

Twitter-derived consumer sentiment measurements were much more volatile day-to-day than the polling data, but when the researchers "smoothed" the results by averaging them over a period of days, the results often correlated closely with the polling data, says Brendan O'Connor, a graduate student in Carnegie Mellon's Language Technologies Institute and first author of the study.

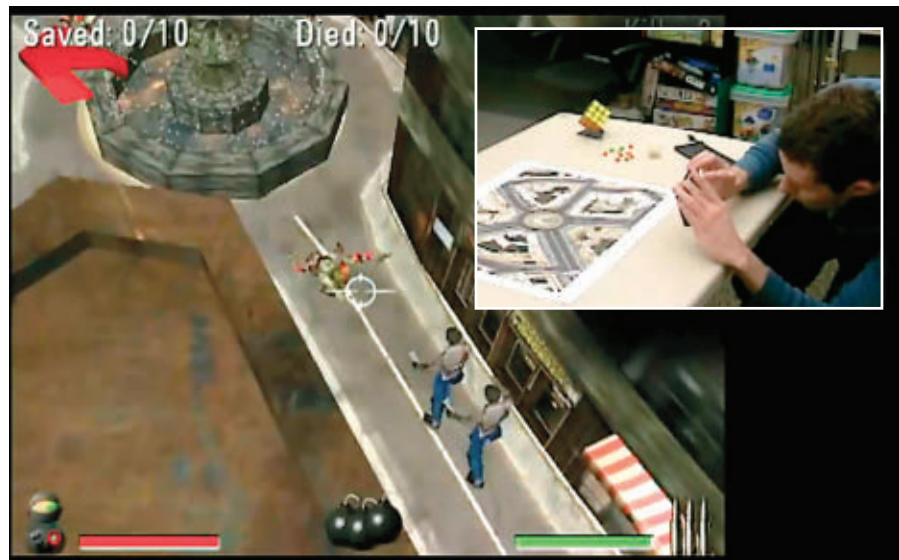
Mainstreaming Augmented Reality

Advancements in computer vision, object recognition, and related technologies are leading to new levels of sophistication in augmented-reality applications and presenting new ways for humans to relate to the natural world.

SINCE THE EMERGENCE of the first augmented-reality applications 20 years ago, the field has drawn a great deal of interest and enthusiasm, not only from researchers working in computer science at the cutting edge of graphics technologies, but also from leaders in aerospace, medicine, the military, and a wide range of other industries and government sectors. In augmented reality (AR), a real-world setting or set of objects is augmented by a computer-generated overlay. Advancements in computer vision, object recognition, and related technologies are increasing the level of sophistication of that overlay, and presenting entirely new ways for humans to relate to the natural world.

While a great deal of research is being conducted in this area, given the promise of the technology to have a major impact in industrial and consumer applications, significant challenges remain, such as the accuracy of Global Positioning System- (GPS-)

Augmented-reality applications are increasingly compact and powerful, and many of them require nothing more than a current-generation smartphone.



An augmented reality game called ARhrrrr! developed at Georgia Tech and the Savannah College of Art and Design. In the game, the graphics are tightly registered to a physical game board using an image-based feature tracker developed at Graz University.

or compass-based AR applications, the bulkiness of head-mounted displays, and other issues endemic to the sciences and systems upon which AR technologies rely. Still, researchers developing AR systems continue to build increasingly compact and powerful applications, many of which require nothing more than a current-generation smartphone.

Examples of mobile AR applications include Layar, a “reality browser” that retrieves point-of-interest data on the basis of GPS, compass, and camera view, and GraffitiGeo, an application that lets users read and write virtual Twitter-style comments on the walls of restaurants, movie theaters, and cafes. Both applications are available for the iPhone platform. Another example is Goggles, a Google-created application that allows users to search the Web on Android phones simply by capturing photos of landmarks or oth-

er objects. The technology also allows users to point the phone’s camera at local storefronts to retrieve business information automatically with GPS and compass data.

While the number of such mobile applications is increasing rapidly, AR evangelists say a killer app will be needed to make AR technologies truly catch on in the consumer space. Given the enormous popularity of Web-based social networking, for example, one killer app might come in the form of a mobile facial-recognition application that can automatically link the humans to their social-network profiles. One company, The Astonishing Tribe, has demonstrated an AR interface concept, called Recognizr, to show the possibilities of doing just that.

Another approach to mainstreaming AR is in gaming. One researcher working in this area is Blair MacIntyre, who directs the Augmented Environ-

ments Lab at the Georgia Institute of Technology. MacIntyre says his current work in AR is driven mainly by the desire to understand how to create compelling AR experiences, interfaces, and tools. To that end, he and his team build games and study them, focusing on everything from interactivity and visualization techniques to the feel of game mechanics to the social experiences they foster.

"I'm very driven to create tools and platforms that will give a broad range of people the ability to experiment with the technology," says MacIntyre. "Just as we didn't know what the Web would be used for until people with real problem- and design-driven goals started trying to create applications, the same will be true for AR."

Tightly Registered AR Games

For now, MacIntyre is focusing on what he calls tightly registered AR games, in which the graphics appear to be locked onto the real world. In the *ARhrrrr!* game, for example, a handheld device's graphics are aligned with the physical game board using an image tracker to determine where the camera on the handheld is located, relative to the board. The system pulls video from the camera, runs it through a vision library, and returns an estimate about the game

The mainstreaming of augmented reality now largely depends on the ability to manufacture and sell the technology profitably, says Steven K. Feiner.

board's relative position. Using that information, the handheld draws graphics in the camera's view of the board. Those graphics remain locked in place over a wide range of movement by the player.

"We found that if the graphics are unambiguously aligned with features in the world, game players treat the combined physical-virtual view as one merged space," he says. "As a result, they can refer to virtual content smoothly and unambiguously, and can collaborate or compete as they would on a physical board game."

MacIntyre says the biggest challenge he faces is with the limitations

of the vision-based tracking technology that signals to the phone what the camera's relation is to the world. "We are constantly struggling with the tension between what we want the games to do and what is technically possible to know about the world and to track and interact with," he says. Because accuracy is directly related to the quality of the inputs, MacIntyre and his team use vision-based tracking technology instead of less-accurate alternatives such as handheld-based GPS, compass, and accelerometer sensors, which might work for large-scale AR applications but lack the precision needed for tightly registered games.

Another researcher working in this area is Steven K. Feiner, director of the Computer Graphics and User Interfaces Laboratory at Columbia University. Feiner began his work in AR by exploring how the technology might be used to assist in maintenance and repair, and has directed projects ranging in focus from restaurant guides and gaming to integrating technical instructions directly into a task domain. "Our overarching goal is to design user interfaces that help people be better at whatever they do," Feiner says, noting that his general approach in these AR projects is to build dynamic virtual worlds that are overlaid on and geo-

Employment

U.S.'s Bright CS Job Forecast

The job outlook for U.S. college students majoring in computer science is very favorable, according to *The Market For Computing Careers*, a report by Joel Adams, a professor of computer science at Calvin College. Adams' report contains an analysis of data from the U.S. Bureau of Labor Statistics, Computing Research Association's Taulbee Survey, and *U.S. News & World Report*.

"The U.S. Bureau of Labor Statistics predicts that computing will be one of the fastest-growing U.S. job markets in science, technology, engineering, and mathematics (STEM) for the foreseeable future," according to the report, with "nearly three out of four

new science or engineering jobs in the U.S. going to be in computing." Of these new computing jobs, 27% will be in software engineering, 21% in computing networking, and 10% in systems analysis.

The "demand for software engineers, network administrators, systems analysts, and other computing-related professionals is exploding, but fewer and fewer students are choosing to study what is needed to get these jobs," the report says. The U.S. Bureau of Labor Statistics, for example, predicts, nearly 140,000 new job openings in computing per year through 2018, with less than 50,000 CS graduates vying for those jobs.

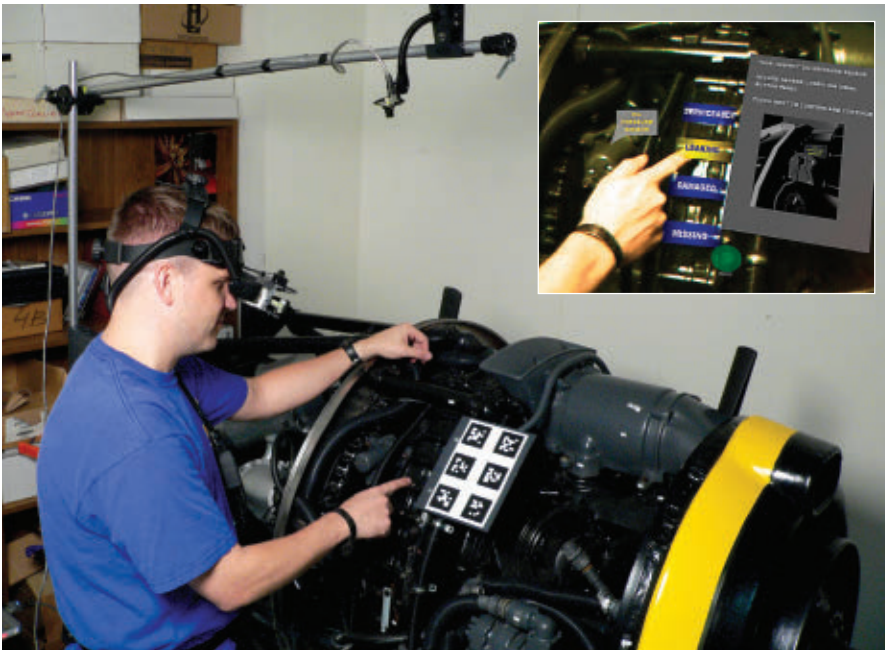
Meanwhile, as fewer students enter CS, the salaries for software engineers, network administrators, and systems analysts "are climbing." According to *U.S. News & World Report*, the median salary for a software engineer ranged from \$85,000-\$92,000 in 2008, with the best-paid 10% of software engineers earning more than \$136,000.

"I think the most surprising thing [in the report] is that the U.S. Bureau of Labor Statistics is projecting more than four times as many new jobs in computing than in all the traditional (non-software) engineering areas combined," Adams said in an email interview. "A second surprise was their projection of

more than twice as many new computing jobs per year than there are computing graduates at present. The third surprise was that computing is the only STEM discipline where the demand for graduates outstrips the supply.

"Calvin College is in Michigan, which has, I believe, the highest unemployment of any state, but we are already seeing the effects of this imbalance," says Adams. "This past semester, we received an average of three requests per week from local businesses seeking students with significant computing skills. We don't have nearly enough students to meet such demand."

—Jack Rosenberger



An augmented-reality application developed at Columbia University. The “opportunistic controls” shown in this image are virtual buttons on a raised portion of an aircraft engine housing, providing haptic feedback for a maintenance technician.

metrically registered with a user’s perception of the physical world, providing information that would otherwise be invisible.

In one example of this approach, Feiner and his team explored how users could more effectively control those AR applications in which it’s necessary to select and adjust certain physical operating parameters, all without using physical controls and without diverting attention from the task at hand. One of Feiner’s graduate students, Steve Henderson, developed a solution to this problem. Called “opportunistic controls,” the technology locates virtual controls, such as buttons, sliders, knobs, and so forth, on top of physical elements of the task environment. For example, the AR system might place a virtual button on a protruding bolt or a virtual rotary knob on a rotating cable connector.

To create an opportunistic control, the system overlays a physical object with a 3D widget and associates it with a hand gesture. The depiction of the widget is rendered in a head-worn display, while the hand gestures are recognized through computer-vision algorithms performed in real time on video captured from an overhead camera. A separate camera captures the video so the control can be operated even when the user is not looking

directly at it. The shape of the physical objects associated with the controls can help the user distinguish them by touch, as with conventional controls.

Making a Profit

With these and other AR technologies growing increasingly robust and reliable, Feiner says the mainstreaming of AR technology now largely depends on the ability to manufacture and sell the technology profitably. He says he remains convinced that future AR technology will not be a mere novelty; instead, he says, it will be one of the fundamental user interface paradigms through which humans interact with the world. In the future envisioned by Feiner, AR technology will be housed not only in comfortable eyewear, but also in handheld or stationary see-through displays, in projected displays, and even some in surfaces that are themselves displays.

Georgia Institute of Technology’s MacIntyre shares a similar view of the future in which humans are constantly immersed in a mixed physical-virtual world. The major challenge on the path to a future of ubiquitous AR technology is, of course, to develop the complete infrastructure, from the necessary technologies to track where users are and what they are looking at to the privacy and security infrastruc-

ture to ensure that users can trust the system, and also to ensure that user privacy and safety are not violated. “I think the technology has a long way to go before we can experience such constant immersion,” says MacIntyre. “But we will begin getting a taste of it in the very near future.”

With the goal of nudging the research community in that direction, one of MacIntyre’s projects is a standards-based platform for mobile AR, the aim of which is to do for AR what the early decoupled client-server architecture did for the Web. In contrast to cloud computing, AR applications currently require dedicated programs running on client devices. MacIntyre’s idea is to create a general-purpose AR browser and a corresponding collection of cloud-based technologies to allow anyone with a server to create and deploy mobile AR apps without requiring users to install anything.

“We need to start developing open standards for AR applications, so a wide variety of people, companies, and organizations can create and deploy these applications,” MacIntyre says. “I believe these application environments and open standards will have the biggest impact on the blossoming of AR as a widely used technology.” ■

Further Reading

Barfield, W., and Caudell, T. (eds.) *Fundamentals of Wearable Computers and Augmented Reality*, Lawrence Erlbaum, Mahwah, NJ, 2001.

Bimber, O., and Raskar, R. *Spatial Augmented Reality: Merging Real and Virtual Worlds*, A.K. Peters, Natick, MA, 2005.

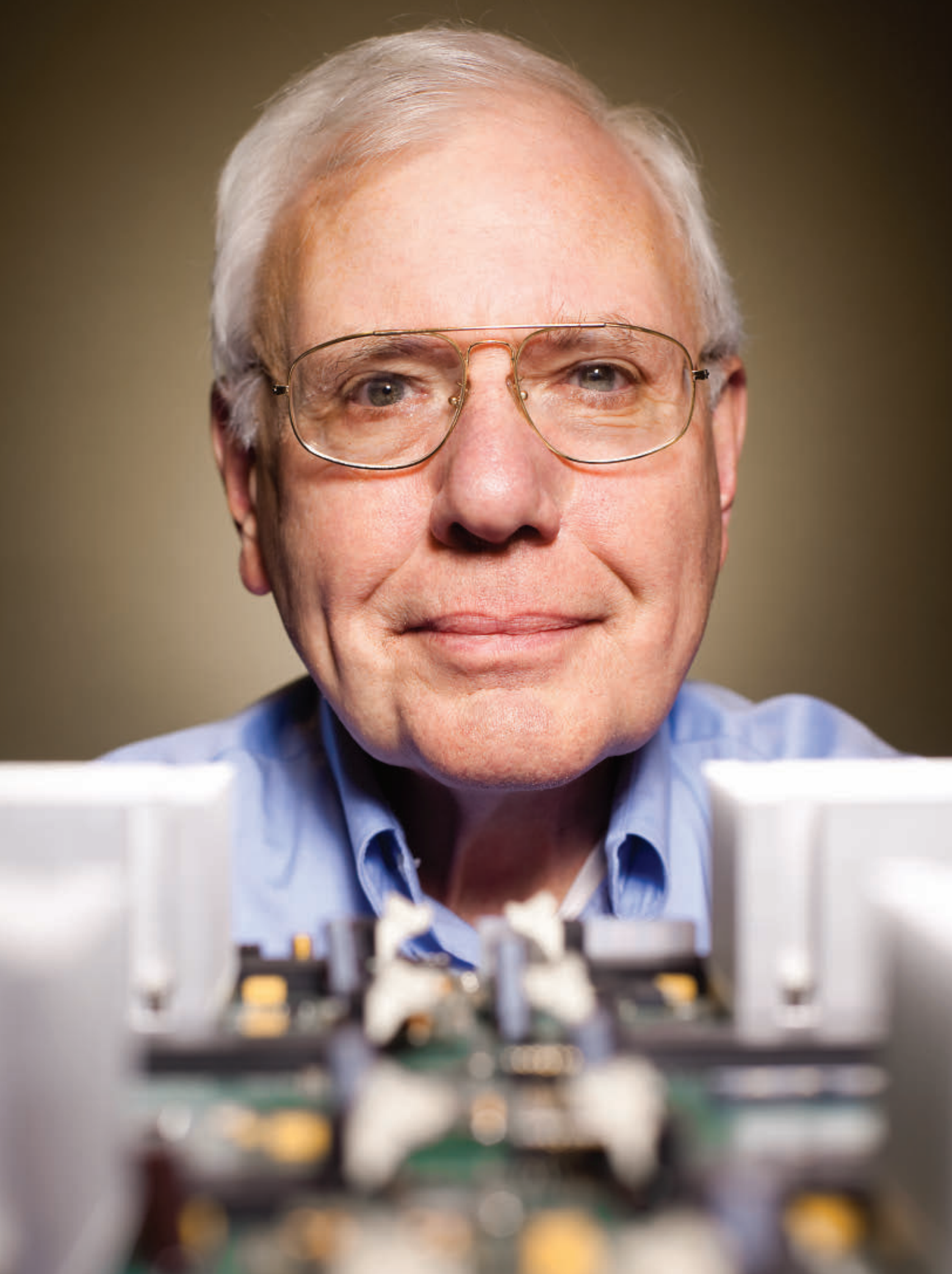
Feiner, S.K. The importance of being mobile: some social consequences of wearable augmented reality systems, *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, 1999.

Hainich, R.R. *Augmented Reality and Beyond*, Booksurge, Charleston, SC, 2009.

Haller, M., Billingham, M., and Thomas, B. *Emerging Technologies of Augmented Reality: Interfaces and Design*, Idea Group Publishing, Hershey, PA, 2006.

Based in Los Angeles, Kirk L. Kroeker is a freelance editor and writer specializing in science and technology.

© 2010 ACM 0001-0782/10/0700 \$10.00



Committed to Success

Charles P. Thacker talks about the importance of simplicity, reusable tools, thinking broadly, and his practice of Tom Sawyering.

IN OUR AGE of hyperspecialization, it's often said that no one can be a Renaissance man. Indeed, Charles P. Thacker, winner of the 2009 ACM A.M. Turing Award, insists he isn't one. But, he notes, "I can lurk at a lot of different levels. I have designed chips, I can design logic, I can design systems, and I can write software up to and including user interfaces."

While "lurking" in these distinct areas for four decades, Thacker has led the design of an astonishing array of technologies, from personal computers to networking technology to tablet PCs. (An interview with Thacker, "From Single Core to Multicore," appears on p. 112.) Today, he is best known for his invention of the Alto, a personal computer, at Xerox Palo Alto Research Center (PARC) in 1970.

"The Alto was the world's first personal computer," says David Patterson, a professor of computer science at the University of California, Berkeley and a computer hardware pioneer himself. "It included everything we think of as being in a PC today: a high-quality graphical user interface, networked computing, laser printing, and the mouse. It, in turn, enabled the invention of software at PARC that shapes our world today: window systems, WYSIWYG editing, drawing and painting, email clients, graphical CAD tools, and clients for file and print servers.

"For those who were not around at the time, it's hard to put into perspective what a breakthrough this was and how much it shaped the computer industry," Patterson adds. "The notion that you would build a powerful computer for just one person was a radical one."

Thacker also co-designed Ethernet local area network technology at PARC in the 1970s and the Firefly multipro-

Charles Thacker with the circuit board of his latest project—the BEE3 computer-architecture hardware platform.

"Complexity is the enemy of computer science, and it behooves us, as designers, to minimize it."

cessor workstation and fault-tolerant networks at Digital Equipment Corporation in the 1980s. "These things have a common thread," Thacker says, "which is they are part of a distributed system—they don't stand in isolation." The Alto was a "nice" single-user machine, he says, but its "real power" was unleashed by networking.

Thacker cites several secrets for his decades of continual success: strive for simplicity, build a kit of reusable tools, insist on sound specifications, think broadly, and make sure your collaborators also succeed.

Of simplicity, he says, "A lot of people think mastering complexity is the goal. But once you have gotten your Master of Complexity merit badge, you don't have to keep winning it. Complexity is the enemy of computer science, and it behooves us, as designers, to minimize it."

Thacker is accomplishing exactly that in his role as a Technical Fellow at Microsoft Research. He's designing simple multicore computers, using single field-programmable gate arrays. The computers are used to conduct research in multicore systems, and are much faster than simulators written in software and much cheaper than building real multicore chips, he says. Their simplicity makes it easy for Microsoft and university researchers to evaluate

different system designs and methods of programming systems with multiple processor cores.

Early in his career, Thacker built his own computer-aided design tool, which he has rewritten several times to take advantage of new programming languages. His toolkit also includes reusable algorithms and software modules developed by himself and others.

Thacker says he wrote specifications for the Alto before designing it, a step that is too often ignored today. "What software engineers frequently do is sit down with a list of features to add to a system," he says. "That's quite different from a specification, because the features might be relatively undefined."

Thacker points to the phenomenal success of IBM's System/360 line of mainframes and says, "The best specification ever written, in my view, was the *System/360 Principles of Operation*, which described the interface between the hardware and software."

As for advice for young computer scientists, Thacker says, "Try to be broad. Learn more math, learn more physics." He modestly calls himself "a jack of all trades and a master of some."

Thacker has succeeded, in part, by working with "really smart guys," says Gordon Bell, a Microsoft principal researcher. Thacker, he says, "basically cordons off a nice-sized, hard-but-doable, and really-useful-to-be-solved problem, and then he works with a small team to carry it out."

Thacker says it's important in cross-specialty projects to motivate teammates. "I've been fairly successful at what I call Tom Sawyering," he says. "It's the idea that if you want to get your fence painted, you trade something of value with the people with whom you work. You have to be committed to their success as well as your own." **□**

Gary Anthes is a technology writer and editor based in Arlington, VA.

© 2010 ACM 0001-0782/10/0700 \$10.00

Eric Brewer: Change Agent

Eric Brewer's latest project involves designing and deploying low-cost wireless infrastructure in developing regions.

ERIC BREWER HAS received plenty of accolades during his career, but his latest award is the first one that has moved him to tears.

Brewer, a professor of computer science at the University of California, Berkeley, is the recipient of the 2009 ACM-Infosys Foundation Award in the Computing Sciences for his contributions to the design and development of highly scalable Internet services. Brewer says his emotion about being named the winner stems partially from the fact that the ACM-Infosys Foundation Award considers accomplishments across the entire field of computer science. He says, with a bit of understatement, "That's a pretty big group."

It might be impossible to overstate Brewer's influence upon making computer science accessible to that "pretty big group." His research on cluster computing in the early 1990s led to the concept of scalable servers capable of simultaneously serving millions of users. His pioneering work as CEO of Inktomi in the mid- to late 1990s greatly advanced Internet search and improved network performance.

Brewer's latest project, called TIER (Technology and Infrastructure for Emerging Regions), which focuses on designing and deploying low-cost wireless infrastructure in the developing world, may have the same disruptive impact on traditional views of economic development policy that his earlier work had on computing architecture.

"The traditional model of economic development has been very top-down—'Take \$100 million to build a dam,' " Brewer notes. "That's had mixed success and is also very expensive. But things that have worked bottom-up, in particular cell phones, didn't have any top-down funding at all."



Eric Brewer, winner of the 2009 ACM-Infosys Foundation Award in the Computing Sciences.

Brewer is literally putting his \$150,000 ACM-Infosys Foundation Award where his mouth is by investing it in TIER, which has projects under way in Cambodia, Ghana, Philippines, and elsewhere. And, just as his work on clusters in the early 1990s served as a bridge between

"There aren't many ways to affect a billion people. I like the idea that computer science can do it."


contemporary research that explored clusters as supercomputers and the nascent ubiquitous demand for networked data, he hopes the TIER project will help blend the discipline of computer science with the economic and social benefits, such as improved public health, presented by low-cost wireless networks.

"Is computer science open minded enough to allow this kind of work to count?" he asks. "That's not a given, and a lot of my talks in the last five years have been evangelizing why this topic should be inside the fold."

Brewer, who made and lost a billion dollars during the dot-com bubble and burst, says the experience raised his aspirations about how he could influence the world, and that innovative ideas, particularly those nurtured in the risk-tolerant environment of tenured scientists, don't have to be backed by a large bankroll.

"When I had a billion dollars," Brewer says, "I was thinking about what to do with it, and surely would love to have it back, but when I lost it, I did realize that money wasn't the only way to try to affect all the people I wanted to affect. I think it's harder to do without the money, but it's certainly more replicable—it's something that everyone can do."

In particular, Brewer says, the vast possibilities offered by inexpensive cloud computing and the bootstrap-to-titan ethos of modern computer science means almost limitless opportunities for today's students and scientists.

"There aren't many ways to affect a billion people," he says. "I like the idea that computer science can do it." 

Gregory Goth is an Oakville, CT-based writer who specializes in science and technology.

© 2010 ACM 0001-0782/10/0700 \$10.00

Visions of the Future

ACM joined forces with the British Computer Society to deliver its first academic research conference in Europe.

DELEGATES FROM AFRICA, Europe, and North America gathered at the University of Edinburgh in April to discuss the latest research in computer science and listen to innovative project proposals for the U.K. Computing Research Committee's Grand Challenges program.

Professor Dame Wendy Hall, president of ACM and professor of computer science at the University of Southampton, opened the ACM-BCS Visions of Computer Science 2010 conference, alongside British Computer Society President Elizabeth Sparrow.

Hall discussed the importance of diversifying ACM beyond the U.S. and, after welcoming more than 100 conference delegates, handed over the proceedings to computer scientist Michael Foreman of the School of Informatics at the University of Edinburgh. Foreman paid tribute to the recently deceased Robin Milner, eminent computer scientist, co-creator of the Grand Challenges, and ACM A.M. Turing Award winner, and proposed a Milner symposium next year to celebrate the scientist's work.

Ross Anderson, professor of security engineering at the University of Cambridge, delivered the first keynote speech, "The Dependability of Complex Socio-Technical Systems." Anderson described the evolutionary convergence of branches of knowledge, including philosophy, mathematics, and economics, into computing, and questioned how it should advance. "We are responsible for everything," said Anderson, "and must deal with the global-scale socio-technical systems that are emerging and will be the way the world works."

The second keynote speaker, Nicolò Cesa-Bianchi, professor of computer science at the University of Milan, discussed "The Game-Theoretic Approach to Machine Learning and Adaptation." To consider whether game



The ACM-BCS Visions conference was held at the University of Edinburgh Informatics Forum.


theory could complement or surpass statistics in the analysis of algorithms that learn and adapt, Cesa-Bianchi presented research that replaces statistics to describe an interaction between a learning agent and a changing environment with a repeated game between an agent and environment. This approach, he says, is particularly appropriate to machine learning in arbitrary and adversarial environments.

"Things have changed, and our work has become more interdisciplinary," notes Wendy Hall.

The other keynote speakers were Jon Kleinberg, professor of computer science at Cornell University, who presented "Exploring the Structure of Online Social Networks: the Roles of Positive and Negative Links in Network Interaction," and Barbara Liskov, a professor at the Massachusetts Institute of Technology and ACM A.M. Turing Award winner, who discussed "The Power of Abstraction."

Among the conference's sessions covering subjects from ubiquitous systems to theoretical computing and the digital economy, one proved particularly timely. As a massive cloud of ash from Iceland's Eyjafjallajökull volcano shut down air traffic across Europe, Ken Anderson, associate professor of computer science at the University of Colorado, outlined a vision for technology-mediated support for public participation in mass emergencies and disasters.

Following Visions 2010, Hall introduced the Grand Challenges session. "Things have changed," said Hall, "and our work has become more interdisciplinary, feeding into areas such as health care, climate change, and security. We need to make evolutionary, not revolutionary, change, but a new list of Grand Challenges will emerge."

Eighteen proposals for the Grand Challenges, a program supported by the U.K. Computing Research Committee, were added to nine existing projects, with a decision on the proposals expected over the summer. The candidates included a project using software engineering to achieve zero-carbon buildings by 2019, a program to develop information and communication technologies for a global population of nine billion people in 2050, and five proposals about health care and independent living. 

Sarah Underwood is a technology writer based in Teddington, U.K.

© 2010 ACM 0001-0782/10/0700 \$10.00

Introducing:

XRDS

The ACM Magazine for Students

XRDS delivers the tools, resources, knowledge, and connections that computer science students need to succeed in their academic and professional careers!

The All-New *XRDS: Crossroads* is the official magazine for ACM student members featuring:

- › Breaking ideas from top researchers and PhD students
- › Career advice from professors, HR managers, entrepreneurs, and others
- › Interviews and profiles of the biggest names in the field
- › First-hand stories from interns at internationally acclaimed research labs
- › Up-to-date information on the latest conferences, contests, and submission deadlines for grants, scholarships, fellowships, and more!



Also available

The All-New XRDS.acm.org

XRDS.acm.org is the new online hub of *XRDS* magazine where you can read the latest news and event announcements, comment on articles, plus share what's happening at your ACM chapter, and more. Get involved by visiting today!

XRDS.acm.org



Association for
Computing Machinery

Advancing Computing as a Science & Profession

V viewpoints

DOI:10.1145/1785414.1785427

Mari Sako

Technology Strategy and Management Outsourcing Versus Shared Services

Choosing between outsourcing and shared services has significant implications for long-term corporate strategy.

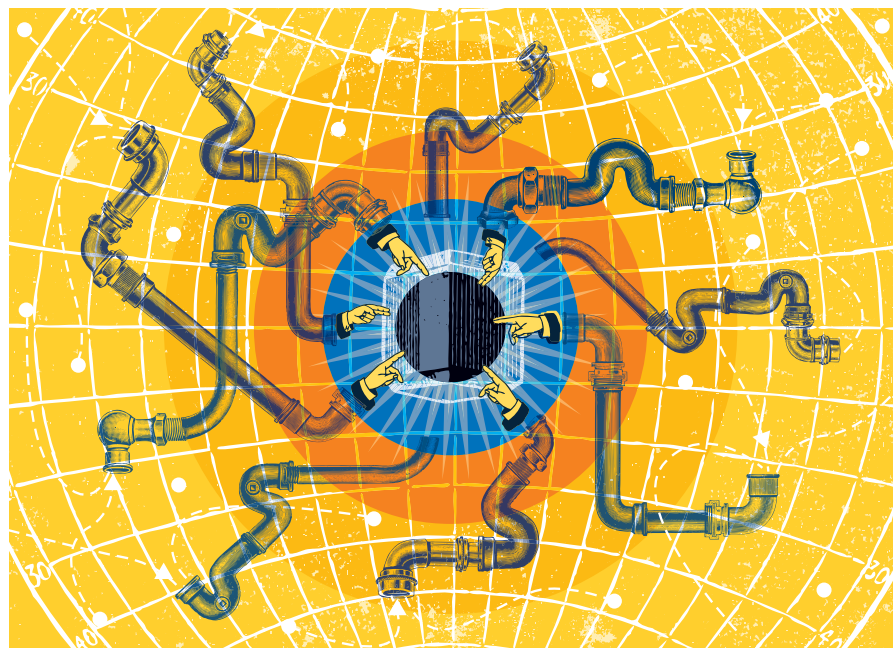
TWO DIAMETRICALLY OPPOSED perspectives continue to coexist in IT and other business service functions. One camp argues in favor of shared services, wherein the IT organization becomes the internal service provider to the rest of the company. The other camp promotes outsourcing: the delivery of IT services all done under one roof but with that roof located somewhere other than at the company. Proponents do not agree on which is better. This column examines the background that led to the adoption of these practices and the reasons for this disagreement.

Most readers know about IT outsourcing, and many readers are aware of H. Ross Perot, the Texan who left IBM to found Electronic Data Systems (EDS) in 1962. His idea was to offer technology as a service by not merely equipping, but also operating, customer data centers. Following his success, others joined to extend outsourcing

from IT services to a wide array of business services in finance and accounting, human resources, procurement, and logistics. Technology—a combi-

nation of computers, software, and networks—underpins these support operations.

The 1990s saw the birth of mega-



deals, such as the seven-year \$600 million human resource outsourcing (HRO) deal between the oil giant BP and a start-up provider, Exult.^{1,5} But is it really only third-party providers that can offer superior service at lower cost? Is it not possible for a company to perform equally well in its internal shared services by reducing duplication of processes and facilities and by reorganizing and sharing assets? I argue that a company can achieve similarly good performance levels in its internal shared services operation, under certain circumstances.

Companies outsource for several reasons. Cost saving may be the ultimate reason. But the means by which this is achieved vary, from introducing new technology, improving service quality, transforming fixed investments into variable costs, to freeing management time to focus on core competencies. Outsourcing is about the make-or-buy decision, and the term applies to a broad range of procurement activities in manufacturing (for example, automobile companies purchasing transmission components) or services (for example, a retailer sourcing TV advertising). But when applied to business (including IT) services, two essential features are highlighted.

Outsourcing as Corporate Restructuring

One feature is that outsourcing of business services combines two decisions. One is the make-or-buy decision concerning the corporate boundary; the other is the restructuring of the internal corporate hierarchy. Corporations' organization structure can be a source of competitive advantage. As recounted by the business historian Alfred Chan-

Existing corporate structure affects the firm's choice between outsourcing and shared services.

dlar, modern corporations have been restructuring constantly to align structure to strategy.³ In reality, it is easier said than done for a global corporation to design and implement an appropriate multidimensional matrix structure to meet the competing demands of different products, corporate functions, customers, and countries.⁴ The creation of shared services and outsourcing are both part of this search for an appropriate organizational design, giving primacy to corporate functions over other considerations, often accompanying mergers and acquisitions (M&A). M&A create duplicated functions previously belonging to two separate corporate entities. Attempts at eliminating the waste of duplication trigger the creation of shared services. More likely than not, such streamlining requires some central direction from the corporate headquarters. Without such centralized control, the intended standardization and efficiency gains may not be realized.

The contrasting experiences at Procter & Gamble (P&G) and Unilever illustrate this point.^a Under A.G. Lafley's leadership, P&G created an internal global shared services unit in 1999 as part of the 'Organization 2005' restructuring initiative. It gave itself five years to pull all essential corporate functions—finance and accounting, human resources, and later IT—away from regional and divisional companies into a single Global Business Services (GBS) operation. A statement by the GBS head, Mike Power, that “everyone has done something, but no one has done everything” illustrates the radical nature of this consolidation. Central direction from P&G headquarters in Cincinnati was essential in deploying SAP-based ERP systems throughout the company before such reorganization took place. By the time P&G's shared services were outsourced, their operations were drastically transformed and streamlined. This experience led Filippo Passerini, Chief Information Officer, to remark: “we believe that there is a sequence in

this process. You outsource only when you are internally optimized.”

In short, centralized firms are better placed to move quickly to efficient shared services. Conversely, divisional autonomy is likely to get in the way of implementing standardized processes. So, what if a corporate headquarters does not have central control? This is where outsourcing comes in as a first port of call. Unilever, the Anglo-Dutch firm, has lived with characterization as a loose federation of national companies with strong country managers who had little interest in global shared services. The human resource (HR) function did, however, consider outsourcing at the global level, and regarded the fragmentation of IT infrastructure as a hindrance in implementing it. With such cultural and technical barriers to creating in-house shared services, global HR outsourcing to Accenture was used as a trigger to transform HR processes, in a “throw it over the fence” or “lift and shift” approach, with an expectation of rapid cost reductions through scale economies, labor arbitrage, and increasing return on assets. Unilever could not have transformed without outsourcing, and outsourcing was an integral concomitant of transforming the organization. Outsourcing is indeed a corporate turnaround trigger. In such applications of outsourcing, both the risks and potential rewards are high.

Thus, existing corporate structure affects the firm's choice between outsourcing and shared services. Moreover, the creation of internal shared services first before outsourcing leads to greater retention of capabilities in-house; by contrast, a path to outsourcing, without an interim step of internal shared services, engenders greater reliance on suppliers' capabilities. In fact, the first path is about “selling” shared services assets and capabilities to providers, while the second path is about “buying in” such capabilities from providers. Some argue that in immature markets without competent providers, firms have no choice but to create internal shared services. There may well be a timing effect, with pioneers opting for the introduction of shared services and followers opting for outsourcing. But can shared services be an end point, without proceeding to outsourcing?

a See detailed comparisons in H. Gospel and M. Sako, “The Unbundling of Corporate Functions: The Evolution of Shared Services and Outsourcing in Human Resource Management.” *Industrial and Corporate Change*, (Mar. 2010), 1–30.

Outsourcing as Relational Contracting: Trust and Incentive

The answer to the question of shared services as an end point without proceeding to outsourcing depends on the second key feature of business service outsourcing, and that is the nature of “relational contracting.” For example, in an M&A deal the signing of the contract closes the deal. By contrast, in outsourcing, such “closing” is just the start of a long-term collaborative relationship between two firms. In order for such a relationship to operate well, it relies on trust and incentives. In fact, incentive and trust may be structured better in outsourcing in some cases, but in internal shared services in other cases.

Organizational economists define a relational contract as a contract that is incomplete (due to difficulty of full specification) and informal (due to difficulty in third-party enforcement, for example in courts).^{2,6} A multiyear business service outsourcing deal is a perfect example of a relational contract. It is typically incomplete due to two reasons. First, future contingencies are difficult to specify in the face of unknown market conditions and new technology in several years’ time. Second, quality of business services to be delivered is sometimes difficult to describe and verify. Indeed, both parties may wish to retain a certain degree of post-contractual flexibility. If a contract is incomplete, then it is also difficult for a third party to enforce it. Parties to the contract must rely on alternative enforcement mechanisms. One of these mechanisms is to rely on trust as a social norm to work things out through discussion, with social sanctions in the event of untrustworthy or unethical behavior.^b This works well in stable business communities, and where parties are chosen for their “cultural fit.”

Another enforcement mechanism is incentive alignment. Service Level Agreements (SLAs) are employed to secure high performance in outsourcing and shared services. With stable processes, performance is easily measured, and the bonus and penalty regime gives a good incentive for provid-

There may well be a timing effect, with pioneers opting for the introduction of shared services and followers opting for outsourcing.

ers to perform well. But the credibility of the client firm to commit to paying a bonus is different for external and internal providers. Outsourcing faces “high-powered” incentives, with the client able to credibly threaten to terminate the contract when the provider underperforms; with an internal SLA, it is not easy to do anything if the internal shared services center does not perform. And what’s worse, the internal operation is often a cost center rather than a profit center. At the same time, whenever a provider is offering standardized processes that could be delivered to more than one client, SLA acts as a powerful incentive to perform well for a specific client offering the bonus. By contrast, with processes that are customized for a specific client, SLA does not create as powerful an incentive.

To summarize, the following is the implication from a perspective based purely on incentives. Outsourcing works best to make an external provider truly accountable for performance, whenever processes are standardized and stable for easy SLA specification. By contrast, an internal shared service is a better option in cases where processes are either customized or being transformed. The incentive-based argument highlights the fact that parties must rely more on other mechanisms such as trust if outsourcing is applied to processes requiring customization or transformation. Thus, an optimal degree of contractual incompleteness—somewhere between a “blank check” and a “nail it all down” level of detail—depends on the task at hand (service delivery versus

transformation) as well as the availability of incentive alignment mechanisms and trust.

Conclusion

The jury is still out on whether or not outsourcing or shared services is ultimately the best service delivery model. I have argued in this column that, amid all the management fads and fashion, there is more than one way to do things, and that each way has its merits and demerits, with associated risks and rewards.

Outsourcing and shared services are both part of organization redesign to give primacy to the efficiency of corporate functions. Compared to shared services, outsourcing is a combination of decisions about the firm’s external boundary and its internal structure. Outsourcing may take place at different stages in corporate activities, either as an initial trigger to bring about fundamental restructuring in a “big bang” mode or a next step after a period of internal process transformation. Relational contracts, if well designed, can service the maintenance of high-powered incentives to ensure the delivery of high-quality service. However, the firm may wish to retain internal shared services without outsourcing if it anticipates instituting further business changes in structure and scope of business services. The choice between outsourcing versus shared services is not simply a matter of timing (in mature versus immature markets). It is more crucially a matter of long-term corporate strategy. ■

References

1. Adler, P.S. Making the HR outsourcing decision. *MIT Sloan Management Review* (2003), 53–60.
2. Baker, G.R., Gibbons, R., and Murphy, K.J. Relational contracts and the theory of the firm. *Quarterly Journal of Economics* 117, 1 (Jan. 2002), 39–84.
3. Chandler, A. *Strategy and Structure*. MIT Press, Cambridge, MA, 1962.
4. Galbraith, J.R. *Designing Matrix Organizations that Actually Work: How IBM, Procter & Gamble, and Others Design for Success*, Jossey-Bass, San Francisco, CA, 2009.
5. Lawler III, E.E. et al. *Human Resources Business Process Outsourcing: Transforming How HR Gets Its Work Done*. Jossey-Bass, San Francisco, CA, 2004.
6. Macneil, I.R. Contracts: Adjustments of long-term economic relations under classical, neoclassical, and relational contract law. *Northwestern University Law Review*, 74 (1978), 854–906.

Mari Sako (mari.sako@sbs.ox.ac.uk) is Professor of Management Studies in Said Business School at the University of Oxford, U.K.

Copyright held by author.

^b There is a large body of work on the topic of trust in business relations.

Computing Ethics

Work Life in the Robotic Age

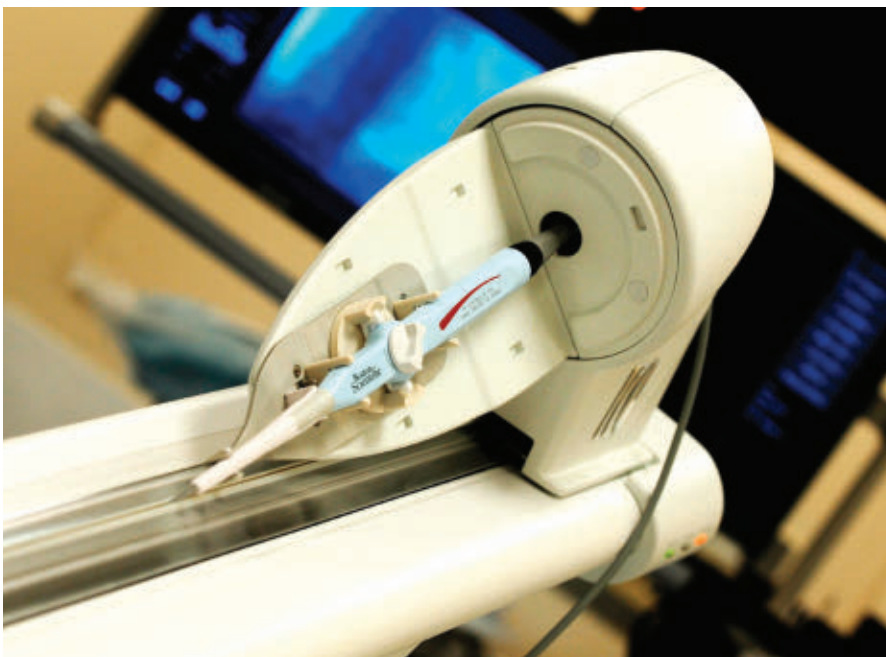
Technological change results in changes in expectations, in this case affecting the workplace.

ROBOTS ARE BEING designed to perform a broader array of work-related tasks. Global economic hardships may be (temporarily) causing the demand for industrial robots to decline,⁴ but improvements in artificial intelligence and the drive for efficiency will likely encourage companies to develop and use increasing amounts of robotic workers. Though the justification for automation is often couched in the language of liberation, this oversimplifies the complexities associated with technological

change. Merely because technology is well designed from an engineering perspective, it does not follow that society's problems are solved. This is not to say that efforts to create robotic workers must stop, but the robotics community must be diligent in dealing with emerging ethical issues. Design pathways must be selected that either mitigate or prevent the negative consequences of using robots in the workplace. Otherwise, troubling historical occurrences, such as the decimation of certain segments of the work force, might be repeated.

With each significant technological change, visions of how improved and efficient our lives will become are typically offered. To some degree, the promise that we will be “liberated” from performing repetitive and mundane tasks has held true. Most of us do not mourn the passing of having to wash clothes or dishes by hand. Yet expectations in both our personal and professional lives tend to shift correspondingly, which in many ways counterbalances the “liberating” features that technology offers. Ruth Schwartz Cowan recognized years ago that the introduction of electronic devices into the home did not free women from the burden of doing household chores. As Cowan states, “What a *strange paradox* that in the face of so many labor-saving devices, little labor appears to have been saved!”¹ In short, increasing expectations absorbed all of the extra time that was supposed to be freed up.

Similarly, we need to seriously consider how the increased use of robots will alter workplace expectations. For instance, if robots can help surgical procedures to be completed more rapidly, will demands on surgeons increase so they will have to perform more procedures per day? Expectations in terms of what it means to be a “good” professional are also likely to change, especially if a robot's error rate is lower than a human's. Briefly put, we should be wary of predictions that robots will be our liberators considering how the



A view of a robot arm used in world's first remote heart operation performed at Glenfield Hospital in Leicester, U.K., on April 28, 2010.

typical workweek does not seem to be getting shorter or less demanding in the digital age.

The U.S. military is enjoying the benefits of robots since they can complete “dull, dirty, or dangerous” tasks, and their labor is very useful in the civilian realm as well. Yet automation can eliminate job opportunities and usually causes the demographics of the work force to be significantly altered in a relatively short amount of time. Employers find robots to be rather enticing since they do not receive benefits or request vacation time. Through the design choices they make, scientists and engineers play a key role in determining the kinds of employment practices that can and will transpire.

Employment Impacts and Implications

If categories of jobs do indeed vanish as a result of robots, will the relevant skills of displaced workers be transferred to another application or will those skills be rendered obsolete? This concern is not unique to robots. But what may be a new variation now is that the jobs available to humans may be drastically reduced as computers, the Internet, and robots replace humans in employment sectors that used to be thought of as immune to automation. At present, it is fairly difficult for people to find work that is not connected in some way to these technologies. This development might not be conducive to the flourishing of each person’s respective talents, and robots are likely to exacerbate this situation. Also, the type of skills that will be in demand if and when the robotic age takes hold might be obvious in some ways but not so apparent in others.^a

The impact of robotic workers can and will extend beyond the elimination of labor-intensive jobs, which captures a key reason why the ethical dimensions of robots seem to be drawing increased attention. It is not only possible to eliminate “dangerous” and “boring” work but at least some jobs requiring specialized expertise, such

a For example, in *Wired for War*, P.W. Singer discusses how cooks might have more job security than military pilots because they can prepare food in creative ways. In the civilian realm, he reassures hairstylists by suggesting their specific abilities may keep them employed; Penguin Press, NY, 2009, 130–132.

Scientists and engineers should reflect on their ethical responsibilities to communicate with the public about a robot’s capabilities and limitations.

as being a surgeon, may start to disappear. A decade ago, Bill Joy, the co-founder of Sun Microsystems, famously warned against this.² Even if we don’t share Joy’s apprehension about the future of robotics, we can still appreciate the perils of trying to replace “uniquely human” abilities such as critical thinking and intuition.

To illustrate this point, we can look at the robots being created to assist with the health care needs of elderly populations. An outgrowth of this effort is that it could subtly or perhaps dramatically change how nursing homes function. In principle, robots could free up the time of nursing home staff; for example, a robotic assistant can provide medication reminders or warnings if a resident is in danger. Such a robotic counterpart might enable human workers to be more caring and productive. However, nursing homes and other care facilities will be tempted to downsize their human staff when a robot is “hired” instead of freeing up human staff to give more time to residents.³ Since many nursing home residents in the U.S. and elsewhere already do not get enough care and individualized attention, this is a very troubling possibility. Theoretically, an increased emphasis on in-home care could for example lead to the creation of other types of jobs but we should be skeptical about this. Financial considerations, the drive for efficiency, and overconfidence in technology are strong driving forces that can push humans “out of the loop.”

On a related note, reliance on automation may exacerbate a common human tendency to shift our attention to a

different task when we believe (perhaps falsely) that we can trust someone or something else to deal with the task at hand.^b Returning to the issue of health care, will nursing home staff be less attentive if a robotic assistant is placed in a resident’s room? The more reliable we think automated systems are, the more likely it is our attention will stray. What complicates matters is that this type of behavioral shift might not be consciously detected. Hence, it would be wise to temper the confidence that users place in robots and other automated systems, especially when people could be significantly harmed. This could be accomplished in part by ensuring that risks are transparently presented to users. To that end, scientists and engineers should reflect on their ethical responsibilities to communicate with the public about a robot’s capabilities and limitations, and not merely leave it to marketers, sales departments, and others to fill this role.

Conclusion

Ethical concerns about integrating robots into the workplace are becoming increasingly pronounced. Again, the intention here is not to stop innovation. Rather, the hope is to inform the design process. Ideally, the robotics community will select design pathways that mitigate the associated concerns and thereby enhance the public’s lives. ■

b Placing too much confidence in technology, often at the expense of other sources of information, seems to be a growing problem with GPS in automobiles; see for example, “Is your GPS navigator a friend or foe?” *The Sydney Morning Herald*, (Jan. 12, 2010); <http://www.smh.com.au/executive-style/gadgets/is-your-gps-navigator-a-friend-or-foe-20100112-m4ei.html>

References

1. Cowan, R.S. *More Work For Mother: The Ironies Of Household Technology From The Open Hearth To The Microwave*. Basic Books, 1983, 44.
2. Joy, B. Why the future doesn’t need us. *Wired* 8, 4 (Apr. 2000).
3. Sparrow, R. and Sparrow, L. In the hands of machines? The future of aged care. *Minds and Machines* 16, 2 (May 2006), 141–161.
4. Tabuchi, H. In Japan, machines for work and play are idle. *The New York Times* (July 12, 2009); <http://www.nytimes.com/2009/07/13/technology/13robot.html>

Jason Borenstein (borenstein@gatech.edu) is the director of Graduate Research Ethics Programs in the School of Public Policy at Georgia Tech in Atlanta, GA.

The author would like to thank Rachele Hollander, Keith W. Miller, and the anonymous reviewers for their helpful insights and guidance.

Copyright held by author.



Pamela Samuelson

DOI:10.1145/1785414.1785429

Legally Speaking

Should the Google Book Settlement Be Approved?

Considering the precedent that could be established by approval of the controversial Google book settlement.

THE COURTROOM WAS packed for the long-awaited hearing about the proposed settlement of the *Authors Guild v. Google* lawsuit on February 18, 2010. Class action lawsuits cannot in the U.S. be settled without a judicial determination that the proposed settlement is “fair, reasonable, and adequate” to the class on whose behalf the case was brought.

Judge Denny Chin heard five hours of oral argument about the proposed settlement not only from lawyers representing Google, the Authors Guild, and the Association of American Publishers (AAP) who negotiated it, but also from the U.S. Department of Justice (DOJ), five non-party supporters, and 21 objectors or opponents, of which I was one. Judge Chin announced at the outset of the hearing that he would not rule on the matter that day.

Because the DOJ has spoken out strongly against the settlement—along with the governments of France and Germany and hundreds of others from the U.S. and abroad—the settlement is facing an uphill battle. An appeal seems likely; so whatever Judge Chin decides, the case is far from over.

This column describes the genesis of the lawsuit and reasons the proposed settlement is so contentious. It presents my argument that the settlement should not be approved without substantial modifications to address concerns of academic authors whose books will make up a substantial por-

tion of the Google Book Search (GBS) corpus of out-of-print books that Google would be able to commercialize if the settlement is approved.

The Authors Guild Lawsuit and the Proposed Settlement

In the fall of 2005, the Authors Guild and three of its members sued Google for copyright infringement because Google was scanning in-copyright books from the collection of the University of Michigan Library. The Guild members claimed to represent the interests of a class consisting of persons holding a U.S. copyright interest in one or more books in Michigan’s library. Five trade publishers brought a similar suit one month later.

After 30 months of negotiations, the litigants announced in October 2008 a proposed settlement of the now-com-

bined lawsuits. The class on behalf of which the litigants now propose to settle consists of all owners of U.S. copyright interests in books published in the U.K., Canada, and Australia and books registered with the U.S. Copyright Office.

The only issue in litigation in the *Authors Guild* case is whether Google’s scanning of in-copyright books for purposes of making snippets of their contents available in response to Google Book Search (GBS) user queries is copyright infringement or fair use.

If the settlement resolved only that dispute (for example, with Google offering \$60 per book for past scanning in exchange for a license to make snippet-displays), approval would almost certainly be granted.

The settlement is controversial because it would give Google a license to commercialize all out-of-print, but still in-copyright books owned by class members as long as Google provides 63% of the revenues from its commercialization efforts to a newly created Book Rights Registry, which would be charged with locating rights holders and paying them money from Google’s commercialization of their books.

At first blush, the GBS settlement looks like a win-win-win. The public would get access to up to 20% of most out-of-print books in response to user queries and full text access in public libraries and higher education settings, either through public access terminals or institutional subscriptions to a database of millions of out-of-print

What should be done about orphan works is a public policy issue that should be decided by Congress, not private parties or the courts.

books. Copies of the books would also be available for individual purchase. Publishers and authors would get paid for the new market Google created for out-of-print books. Google would not only make some money from its 37% share of GBS revenues, but would also be able to make “non-display uses” of books for purposes such as refining its search technologies.

The DOJ agrees that the public would benefit from the enhanced public access to millions of books that would attend approval of the settlement. Yet it has reluctantly concluded that Judge Chin lacks power to approve this settlement because it goes so far beyond the issues actually in litigation that it is “a bridge too far.” The GBS settlement abuses the class action process because the litigants took the occasion of a lawsuit on one narrow issue and used it to dramatically restructure the market for digital books.

The DOJ would endorse a settlement that required class members to opt-in to Google’s commercialization plans. But Google has insisted that the settlement’s opt-out approach (that is, Google gets to commercialize the books unless the copyright owner comes forward to say no) is essential for establishing the new marketplace it envisions.

My Objection to the GBS Settlement

I was one of the 26 non-party speakers to whom Judge Chin granted five minutes to present their views. After introducing myself and noting that I had filed two letters objecting to specific terms of the GBS settlement, the latest one on behalf of 150 academic authors, I made the following points:

► Most of the books that will be regulated by the settlement agreement are out-of-print books from the collections of major research libraries such as the University of California, and most of these books were written by scholars for scholarly audiences.

► Many scholars own copyright interest in their books at least for electronic versions. Many have clauses in their contracts that allow author reversion rights upon the book going out of print. These books will be core parts of the institutional subscription database that will be licensed to universities such as UC Berkeley.

► In the past year I have spoken to many colleagues at UC Berkeley and elsewhere about the proposed settlement. When I asked them whether they would be willing to allow their out-of-print books to be made available on an open-access basis, to a person, they have said yes. Academic authors also tend to believe that orphan books should be available on an open access basis.

► Orphan books are not a trivial matter. The *Financial Times* has estimated the number of U.S.-published books likely to be orphans as between 2.8 to five million. These books will form a substantial part of the institutional subscription database to which my university and others are expecting to subscribe.

► The Plaintiffs have characterized open access advocacy as “a prime example of...parochial self-interests.” They also stated that the interests of open access advocates “plainly are *in-*

imical to the class.” (As if the word “inimical” wasn’t strong enough by itself, they italicized the word to emphasize just how inimical they think open access advocacy really is.)

These statements show that the Authors Guild has not fairly represented the interests of academic authors who are members of the author subclass.

It also bears mentioning that academic authors would not have brought this lawsuit against Google because we tend to think that scanning books to make snippets is available is fair use. If this case goes back into litigation instead of being settled, I will be writing briefs in support of Google, not in support of the Authors Guild.

But it’s not just me and the 150 people who signed the supplemental academic author objection letter who endorse open access. Last August a letter was sent to the court on behalf of the UC Academic Council, which represents 16,000 faculty members at



Author Susan Davis, representing the National Writers Union, arrives for the Feb. 18, 2010 hearing in New York about the proposed settlement of the *Authors Guild v. Google* lawsuit.



ACM's *interactions* magazine explores critical relationships between experiences, people, and technology, showcasing emerging innovations and industry leaders from around the world across important applications of design thinking and the broadening field of the interaction design. Our readers represent a growing community of practice that is of increasing and vital global importance.

interactions
<http://www.acm.org/subscribe>



If this settlement agreement is approved, Google may feel free to go out and scan other copyrighted works.

the University of California, expressing concern that open access preferences of academic authors would not be respected by the Plaintiffs.

More important, though, is the open access recommendation of the U.S. Copyright Office in its report on orphan works. The Office considered and rejected an escrow model for orphan works akin to that in the amended settlement agreement. Once the orphan status of a work has been determined, the Copyright Office thought the work should be available for free use. Congress has modeled its orphan works legislation on the Office's recommendation. What should be done about orphan works is a public policy issue that should be decided by Congress, not private parties or the courts.

It is far more consistent with the utilitarian principles of copyright law to allow orphan books to be made available on an open access basis once we know that they are, in fact, orphaned. This is important to academic authors because what the Plaintiffs want to do is maximize revenues for the millions of orphan books that will be in the institutional subscription database. This is why I have asked for some meaningful constraint on price hikes as part of the settlement agreement.

There is a fundamental difference in perspective between the Plaintiffs and academic authors about what books are really about. For the Plaintiffs, books are commodities to be exploited for maximum revenues.

Books for academics are more like a slow form of social dialogue. The books from the past open the conversation that scholars pick up and carry on. The books we write further that conversa-

tion, and set the stage for the conversation to be carried on by our successors.

The set of objections I made on behalf of academic authors should not be swatted down one by one, as they were in the Plaintiff's Objection memo, but viewed as important component parts of the cultural ecology of knowledge in academic communities. This ecosystem will be impaired if the ecosystem envisioned in the settlement agreement is adopted instead of the one that has long prevailed and should prevail in the future for academic communities.

Setting a Precedent?

While I could live with the GBS settlement if it was amended as suggested in my letters, I worry very much about the precedent that would be set by approval of this particular settlement.

Google's founders say the company's goal is to organize all of the world's information. As we all know, books are not the only type of work that contains the world's information. I have been wondering for some time which sector of the copyright industry will be next to have its works scanned by Google for inclusion in its search database.

If this settlement agreement is approved, Google may feel free to go out and scan other copyrighted works. And if their rights holders object, the pragmatic response might well be: we could litigate about this, but I have a good fair use defense, and it would be expensive and ugly to litigate, so why don't we just reach a deal on my terms right now? Approval of the settlement would give Google unfair leverage in such negotiations.

But beyond that, I think that approval of this settlement would encourage other class action lawsuits that would then seek to justify their efforts to remake copyright law by saying: Congress is too dysfunctional to address this problem, so we must be allowed to do it through a class action settlement. This is just bad public policy. □

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

A transcript of the fairness hearing, along with all documents filed with the court, is available at <http://www.thepublicindex.org>.

Copyright held by author.

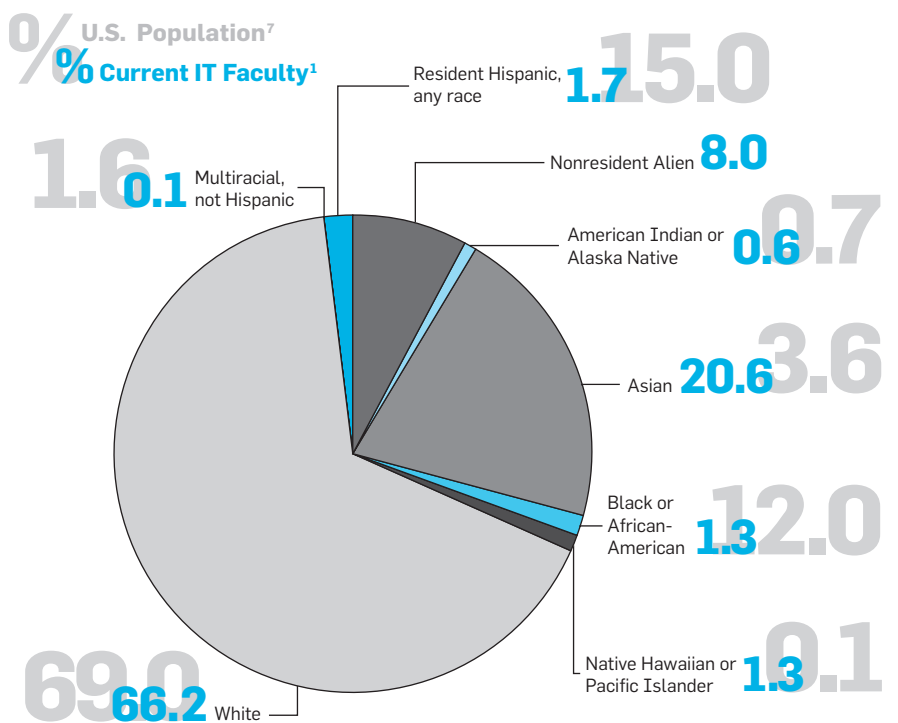
Broadening Participation Cultivating Cultural Diversity in Information Technology

Introducing CMD-IT, a new center focused on synergistic activities related to ethnic minorities and people with disabilities.

THE FIELD OF information technology has had a major impact on society.^a A panel of eight judges from the Wharton School at the University of Pennsylvania recently identified 20 top innovations from the past 30 years; half were tied to the field of IT (examples include the Internet, mobile phones, email, microprocessors, office software, and Internet-based social networking).² Given the significant impact of computing on society, it is important that all cultures, especially underrepresented cultures, are fully engaged in the field to ensure that everyone benefits from the advances in computing. The lack of cultural diversity is especially evident with respect to the following ethnic groups—African Americans, Hispanics, and Native Americans—as well as people with disabilities. CMD-IT was developed to focus on ethnic minorities and people with disabilities for which the link across these different communities is that of understanding a particular culture.

The demographics of the field's faculty shape the demographics of the student population.⁶ African Americans comprise 1.3% of the faculty in IT, but make up 12% of the U.S. population; Hispanics comprise 1.7% of the faculty in IT, but make

Ethnicity of Current IT Faculty



Current IT faculty data from the CRA Taulbee Report, U.S. population data from the U.S. Census Bureau.

up 15%, of the U.S. population.^{1,7,b} By 2020, ethnic minorities are projected to constitute almost 32% of the U.S. population.⁷ People with disabilities comprise 18% of the U.S. population aged five years and older, but the per-

centage of such people in the IT field is far lower.⁷ A diverse student population requires a diverse faculty for many reasons, including incorporation of diverse perspectives in development of student programs and curricula that are engaging to students from all groups.

Culture is manifested in practices that emerge from prolonged participation within specific communities.⁵

a The term "information technology" includes computing, computer science, computer engineering, and other specific subspecialties.

b The data from the CRA Taulbee Survey focuses on computer science and computer engineering, but the numbers are representative of the broader IT field.

According to Mintzes and Wanderse: “Our perceptions of objects and events in the natural world are strongly dependent on our store of prior knowledge we view the world through a pair of ‘conceptual goggles.’”³ These goggles are heavily influenced by culture. The process for seeing the value of diverse perspectives and diverse cultures begins early, usually through learning about multiple cultures in the school system where many social perspectives are formed. Teachers who understand the historical origins and present circumstances of different social groups help students to understand these issues as well. Understanding must go beyond what Moll and Gonzalez call “tangible surface markers,” such as dance, food, language, folklore, and ethnic heritage festivals. Understanding takes into account the everyday lived experiences of diverse cultures represented by students and their families.⁴ Teachers should enter their students’ households and communities as “learners,” seeking to understand the ways in which people make sense of their everyday lives. Teachers of multiple cultures should have direct experience in the communities they discuss.

Higher education and the professional workplace have a number of organizations that serve the important role of providing support mechanisms and programs to increase the participation of particular cultures in science and engineering, including IT. These organizations include the Society for Advancement of Chicanos and Native Americans in Science (SACNAS), the National Society of Black Engineering (NSBE), the American Indian Science and Engineering Society (AISES), and the National Federation of the Blind (NFB). We announce a new, complementary effort in which groups, companies, and organizations focused on underrepresented cultures in IT have a forum to develop synergistic activities and leverage from each other—the Center for Minorities and People with Disabilities in IT (CMD-IT), launched in March 2010.

CMD-IT was created by five people experienced with enhancing diversity within the IT field: Ron Eglash (Rensselaer Polytechnic Institute), Ann Gates (University of Texas in El Paso),

Richard Ladner (University of Washington), Bryant York (Portland State University), and the author. CMD-IT facilitates synergistic activities among industry, established organizations, and local projects related to ethnic minorities and people with disabilities in IT. The organization grew out of an NSF-sponsored meeting on Diversity in IT held at Texas A&M University in April 2008. That meeting identified the following goal for CMD-IT: *To ensure that underrepresented groups are fully engaged in information technologies, and to promote innovation that enriches, enhances, and enables these communities such that more equitable and sustainable contributions are possible by all communities.* That goal is made operational through the following objectives:

- ▶ Provide a *united voice*, spoken by many, that identifies the major issues facing African Americans, Native Americans, Hispanics, Pacific Islanders, and people with disabilities in the IT field.

- ▶ Provide a *resource* for information and statistics related to programs, organizations, and alliances focused on African Americans, Native Americans, Hispanics, Pacific Islanders, and people with disabilities in the IT field.

- ▶ Provide *leadership initiatives* that promote leadership among students, faculty, and professionals from the underrepresented groups.

- ▶ Facilitate *national-scale projects* that involve collaborations between established programs and organizations, with measurable goals focused on engagement and enrichment.

Success with these objectives will facilitate national awareness of cultural issues pertaining to IT and promote effective sharing of best practices and ideas for increasing cultural diversity in IT. The intent of the objectives is already manifested in a project supported by the Broadening Participation in Computing (BPC) Program in NSF’s Computer and Information Science and Engineering (CISE) Directorate. This project, “Incorporating Cultural Tools for Math and Computing Concepts into Boys and Girls Clubs,” is gaining prominence in the U.S. by leveraging regional and national organizations for volunteers required to scale two successful, local projects to

the national level. This project enlists the Boys and Girls Clubs of America to extend to national scale two local efforts—Culturally Situated Design Tools (CSDTs) and African American Distributed Multiple Learning Styles Systems City Stroll (AADMLSS-City Stroll). These two efforts use alternative approaches to educational material involving math and computing to provide a better fit to different cultural orientations and perceptions. This national effort in the U.S. intends to develop an institutional pipeline for K–12 students to enter undergraduate programs in IT, and extend new math education tools to include computing. The project includes an evaluation component to help determine the circumstances under which these tools are most useful. Early commitments to facilitate this effort have been obtained from the Hispanic Association of Colleges and Universities (HACU), the National Technical Association (NTA), and the STARS Alliance.

Currently, CMD-IT is establishing communities of practice, which consists of representatives from industry, organizations, and projects focused on cultural diversity in IT. Further, CMD-IT is developing national-scale projects and initiatives, and establishing partnerships and providing resources for improved understanding of different cultures. Readers are encouraged to learn more about CMD-IT at <http://www.cmd-it.org/>. ■

References

1. CRA Taulbee Report, <http://www.cra.org>
2. Korkki, P. Internet, mobile phones named most important inventions. *New York Times*, (Mar. 7, 2009).
3. Mintzes, J. and Wanderse, J.H. Reform and innovation in science teaching: A human constructive view. In J.J. Mintzes, J.H. Wanderse, and J.D. Novak, Eds., *Teaching Science for Understanding: A Human Constructivist View*, Academic Press San Diego, CA, 1997.
4. Moll, L.C. and Gonzalez, N. Teachers as social scientists: Learning about culture from household research. In P.M. Hall, Ed., *Race, Ethnicity, and Multiculturalism: Policy and Practice*, Routledge, 1997.
5. Rogof, B. *The Culture of Human Development*. Oxford University Press, New York, 2003.
6. Umbach, P.D. The contribution of faculty of color to undergraduate education. *Research in Higher Education* 47, (2006).
7. U.S. Census Bureau; <http://www.census.gov>

Valerie E. Taylor (taylor@cse.tamu.edu) is the holder of the Royce E. Wisenbaker Professorship and Department Head of the Department of Computer Science at Texas A&M University.

Viewpoint

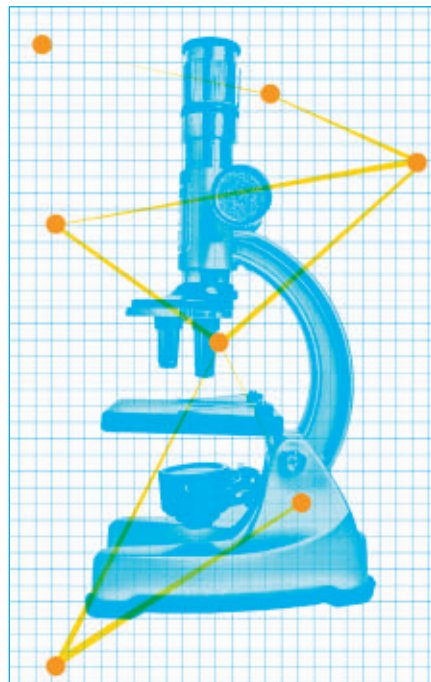
Is Computer Science Truly Scientific?

Reflections on the (experimental) scientific method in computer science.

WE ARE SORRY to inform you that your paper has been rejected, due to the lack of empirical evidence supporting it.” It may well be the case that some of us, in the course of our academic lives have received or will receive—perhaps more than once—a communication similar to the previous sentence. It seems there is a widespread idea that a work only deserves to be qualified as “scientific” if it is supported by “empirical evidence” (from the Greek *empeiria*, experience). In this column I will present some arguments (and attempt to convince the reader) that this stance is completely insufficient, and to recover a place in our academic lives for a kind of research that is more speculative than experimental in character. Of course, I do not intend to question the legitimacy of experimental research, but rather to argue that a harmony must exist between the two. However, this harmony seems to be particularly menaced in current computer science research. This is a paradoxical situation, since computer science is rooted both in speculative sciences such as mathematics and experimental sciences such as physics.

Radical Empiricism

Indeed, it is very easy to criticize this prevailing, radical empiricism: the idea that “only those propositions that are obtained through experience are scientific, and thus acceptable as



true,” is not supported itself by any kind of empirical evidence. Therefore, radical empiricism must be rejected as self-contradictory. Besides, the history of computer science provides us with empirical arguments against empiricism and shows us a very different picture, as I will discuss later. In other words, if radical empiricism is preached, it is not due to empirical or experience-based reasons, but because of other kinds of not-so-clear, to-be-discovered motives. However, given the extraordinarily important role that empirical evidence has in science (it is not without reason we speak of

the *experimental-scientific* method), it would be very superficial to remain with such facile criticism, without trying to go deeper into the question.

Learning from experience—formulating general rules on the basis of particular cases—is generally known as *induction*. Scientific inductivism expressed itself during the 20th century mainly through the philosophical stance known as Verificationism, to which Falsificationism was opposed (I will try to ensure these two are the last *-isms* mentioned in this column, so that the reader can proceed without having to make marginal notes).

Verificationism upholds an optimistic thesis: induction is possible. That is, it is possible to formulate true general laws on the basis of particular experiences. This optimism provides the foundation for the most generalized attitude among scientists, which precisely leads them to seek the confirmation of their theories in experience. The big problem of induction is to determine whether it truly has a rational foundation, since the mere fact that particular cases are repeated does not warrant the positing of a general law. Unless we admit a priori that regularities cannot be casual: there must be some kind of rationality in the universe that is within reach of the human mind. Sarcastic critics of Verificationism will likely recall the old story told by Bertrand Russell about that “inductive turkey,” which after months of repeated experiences (most regular, indeed) came to

the firm conclusion that the man who fed it every morning in the farmyard would continue to do so until the end of times, with all his affection...

Falsificationism, by contrast, as set forth mainly in the writings of Karl Popper, considers in a rather pessimistic way that induction is not possible; we cannot aspire to prove the truth of any scientific theory; scientific hypotheses are no more than mere conjectures that are provisionally accepted until a new experience appears to refute them (what Popper calls “falsification”). This stance is informed by a commendable skepticism that has helped to give it credit among scientists, too. But the truth is that, if taken to its ultimate consequences (beyond the point Popper himself would have taken it), Falsificationism becomes absurd: scientists do not devote themselves to formulating and provisionally accepting *whatever theory*, and then to looking for counterexamples that refute it.

On the contrary, scientists strive to verify hypotheses as much as to refute them, and they only accept hypotheses that are reasonable from the start and that have a huge explanatory power. What this “reasonability” might be, this “explanatory power,” or even the “simplicity and elegance” that no doubt have influenced great scientists in the formulation of their hypotheses and theories (consider Galileo, Newton, Einstein...), is an arduous problem for the Philosophy of Science that cannot be addressed here. I only wish to point out that neither Verificationism nor Falsificationism can give a full account of the reality of scientific activity in all its magnitude. And that both, considered as methodological stances, refer to something that is beyond factual experience. Paying attention only to empirical evidence is not acceptable, especially if the consideration of *correctness of reasoning* is set aside, since, at least, empirical evidence must be adequately *interpreted* with good reasons. Experimentation without the guide of speculative thinking is worthless.

Truth and Relevance

We have demonstrated that empiricism is insufficient. There cannot be a complete scientific activity that consists solely of proving theories by

means of experiments: first, theories must be formulated and developed, and their explanatory power must be demonstrated, so that the investment of human and material resources in the experiments, which may be very costly, can be justified; then, the experiments that will prove or refute the theories must be carried out. Moreover, experimental verification may say something about the *truth* of a theory, but it can say nothing about its *relevance*, that is, its interest to the scientific community or society as a whole.

In this respect, we should be careful to distinguish between experimentation of a theory and its *practical application*: the latter is particularly important in engineering, but developing a practical application does not properly constitute an experimental verification, according to inductive criteria, of the theory that supports it. For example, showing with adequate reasons that a certain design pattern solves a recurrent programming problem demonstrates its applicability without the need of experiments and statistics; the rationale of the pattern, instead, is indispensable. The potential utility of a theory may be enormous, and should be fully acknowledged, but it is not at all an inductive proof—a verification. Conversely, having an empirical validation is not the same as having a practical application.

Lessons from History

Having demonstrated that empiricism is insufficient in and of itself, can we at least say it is *necessary*? That is, should we consider it an essential part of *every* scientific activity? From the scientific point of view, is a purely speculative-theoretical work acceptable without empirical support? In order to answer this question, I will formulate another one: What do we learn from history? In

Experimentation without the guide of speculative thinking is worthless.

particular, and to focus on the area of major interest for the readers of this magazine: Who are the founders of computer science?

Consider some fundamental names: Turing (computation theory and programmable automata), von Neumann (computer architecture), Shannon (information theory), Knuth, Hoare, Dijkstra, and Wirth (programming theory and algorithmics), Feigenbaum and McCarthy (artificial intelligence), Codd (relational model of databases), Chen (entity-relationship model), Lamport (distributed systems), Zadeh (fuzzy logic), Meyer (object-oriented programming), Gamma (design patterns), Cerf (Internet), Berners-Lee (WWW)... Are their contributions perhaps distinguished by their experimental character? Aren't they mainly, or even solely, speculative investigations (yet with enormous possibilities for practical application), whose fundamental merit has been to light the way for the rest of the scientific community, by performing, so to speak, a work of *clarification and development of concepts*? Would they have been able to publish their work according to the “experimentalistic” criteria that currently prevail?

Having a look at the list of Turing Awards¹ or at the most cited computer science papers in CiteSEER² is very instructive. However, given the current standards for reviewing, many of those papers would never have been published. They would have come up against journal reviewers who would have rejected such works, considering them too speculative or theoretical, as has been humorously described in fictitious reviews.⁴

The attentive reader will have noticed that I am *inductively* justifying, from the experience of history, that many of the best works in computer science (the most cited ones, to accept the present identity between “most cited” and “best,” which is of course a very debatable one indeed) do not have a fundamentally experimental character, but rather a theoretical and speculative one. Nevertheless, I am afraid the “recalcitrant empiricist” will not let him or herself be convinced even by this argument...because, in the end, his or her conviction is not grounded in empirical arguments.

It may well happen that we are suffering the “swinging pendulum” effect. In the past, computer science was not so focused on experimentalism. But recently the pendulum has swung too far toward this side, and we should push it the other way. Maybe periodic swings are even helpful for science, and we should not try to stop them completely. After all, science tends to be a self-correcting system, because ultimately truth will win out, no matter how painful the process of discovery might be for those of us toiling in the trenches. As the great American philosopher and logician Charles S. Peirce put it, “the essence of truth lies in its resistance to being ignored.”³

What Distinguishes the Scientific Method?

Now then, if their experimental character is not what primarily distinguishes scientific works, what does? In my view, the distinguishing feature of the scientific method is its “public,” “social” character. I do not mean by this—far from it—that scientific truth is established by consensus, but that research results must be demonstrable to others. This, after all, is the aim of scientific publications (no matter how much these publications, and especially the *number* of publications, serve other, less “avowable” purposes). The enemy of the scientific method is not speculative reasoning, but the appeal to some kind of Cartesian-shaped “intuitive evidence,” enclosed within the individual, and which is neither communicable nor submitted to the community of researchers; the enemy is the acceptance of ideas because they are “clear and distinct” *for me*, regardless of whether or not they are “clear and distinct” *for others*.

Summing up, what the scientist looks for is to follow a way toward knowledge that can be followed by other researchers; the goal is to “convince” the scientific community of the validity of certain research results. Yet there are several possible ways to convince. Must all scientific works be reasoned and demonstrable? Yes, of course. Must they be empirically verifiable? That depends. Not all branches of science are equal; not all kinds of research are equal. If it would be absurd to try to axiomatically demonstrate

What the scientist looks for is to follow a way toward knowledge that can be followed by other researchers.

the failure probability law of a microchip as a function of its temperature; it would be equally absurd to require an experimental verification of the axioms of fuzzy logic.

Conclusion

Experience and speculation must go hand in hand in the way of science. Some investigations will have a basically experimental character, while others will be primarily speculative, with a wide gradation between these two extremes. As long as all are demonstrable, we should not consider some to be more worthy of respect than others. If the pendulum has swung too far toward the experimentalistic side of computer science, we should now push it a bit toward the speculative field, so that the whole picture gets corrected. Thus, I would like to call upon researchers who might feel inclined toward speculative matters—and even more upon those *in charge of research*—neither to close the door nor give up on this kind of scientific activity, which is so essential for the progress of knowledge. **□**

References

1. Association for Computing Machinery, Turing Awards; <http://awards.acm.org/homepage.cfm?awd=140>
2. CiteSeerX, Most Cited Computer Science Articles; <http://citeseerx.ist.psu.edu/stats/articles>
3. Peirce, C.S. Why study logic? In C. Hartshorne, P. Weiss and A. W. Burks, Eds., *The Collected Papers of Charles Sanders Peirce, Volumes 1–8*, Harvard University Press, Cambridge, MA, 1931–1958.
4. Santini, S. We are sorry to inform you. *IEEE Computer* 38, 12 (Dec. 2005), 126–128; <http://portal.acm.org/citation.cfm?id=1106763>

Gonzalo Génova (ggenova@inf.uc3m.es) is an associate professor of Software Engineering at Universidad Carlos III de Madrid.

Copyright held by author.

Calendar of Events

July 19–23

The 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Geneva, Switzerland, Sponsored: SIGIR, Contact: Fabio Crestani, Email: fabio.crestani@unisi.ch

July 21–23

4th International Workshop on Parallel and Symbolic Computation, Grenoble, France, Contact: Moreno Mac, Email: moreno@csd.uwo.ca

July 24–25

The 15th International Symposium on Web3D Technology, Los Angeles, CA, Sponsored: SIGGRAPH, Contact: Marcelo Knorich Zuffo, Email: mkzuffo@lsi.usp.br

July 24–25

ACM Symposium on Applied Perception in Graphics & Visualization 2010, Los Angeles, CA, Sponsored: SIGGRAPH, Contact: Diego Gutierrez, Email: diegog@unizar.es

July 24–26

International Conference on Queueing Theory and Network Applications, Beijing, China, Contact: Wang Jinting, Email: jtwang@btju.edu.cn

July 25–28

The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington DC, Sponsored: SIGKDD and SIGMOD, Contact: Balaji R. Krishnapuram, Email: balaji.krishnapuram@gmail.com

July 26–28

Principles and Practice of Declarative Programming, Hagenberg, Austria, Contact: Temur Kutsia, Email: kutsia@risc.uni-linz.ac.at

Distinguished Members Advice to Members Seeking ACM Distinction

ACM'S DISTINGUISHED MEMBER Recognition Program was initiated in 2006 to recognize those members with at least 15 years of professional experience who have made noteworthy contributions to the computing field. Since this program is relatively new, and has been undergoing changes, there may be many ACM members unfamiliar with the requirements for this grade. As co-chairs for the Distinguished Members committee, we have seen many submissions fail, not because of the quality of the candidates, but due to the lack of adequate information regarding the submission. We hope this column will help produce more effective nominations.

The ACM Distinguished grade consists of three categories: Educator, Engineer, and Scientist. Each category comes with a unique set of criteria, therefore alleviating any confusion or competition between grade levels. The committee ensures that candidates are assessed by experts knowledgeable of the contributions in their category. To the extent possible, candidates are judged by their peers: scientists by scientists, engineers by engineers, and educators by educators. There is no reason for an engineer or an educator to feel ineligible if their CV does not include an extensive list of publications, nor a scientist if he or she has never managed a large project. Indeed, we estimate approximately 10% of ACM's membership qualifies as Distinguished Members.

It is important to create a nomination package suitable for the category.

We estimate approximately 10% of ACM's membership qualifies as Distinguished Members.

Of course, many people will have contributed to more than one category; it is perfectly acceptable to list all major *professional* contributions and activities. However, the submission should focus on making the case for one particular category. The clincher should be contributions as a practitioner, or contributions that advance practice in the relevant category. A scientist practices science by doing research and publishing the results; an engineer by developing products; an educator by teaching. Thus, a member teaching engineering, but not practicing it, might better qualify as an educator than an engineer—unless this person has significantly contributed to the advancement of engineering as a discipline. A member doing research on teaching computer science, but not distinguished as a teacher, might better qualify as a scientist, unless that member has contributed significantly to the advancement of CS education.

The committee cannot independently assess the quality of each sub-

mission; candidates come from many different countries and professional backgrounds of which the committee members may have a limited knowledge. Therefore, the committee puts much weight on the endorsements that support the submission. Strong endorsements are essential for a successful submission.

To Nominate or to Self-Nominate

It has been our experience that nominating a colleague for this distinction succeeds more often than self-nominations. A major reason for this track record is that nominations composed by someone other than the candidate are likely to have stronger endorsements. The nominator should first check that endorsers are supportive of the nomination. Answers are likely to be more sincere if the nominator is not the candidate.

The choice of endorsers is crucial. The committee tends to trust the judgment of endorsers who are recognized authorities in their field, such as ACM Fellows. In fact, the Distinguished Member guidelines recommend that two of the endorsers be ACM Fellows. The nomination package should also include endorsers who are intimately familiar with the work of the candidate and can provide firsthand testimony of its importance. Endorsers in the first category can focus on qualitative assessment of the candidate's merit; endorsers in the second category should focus on providing factual information on the candidate's professional activities and their impact.

A nomination invites scrutiny if all endorsements come from the same institution. As a rule we expect that candidates will have had an impact beyond the boundaries of their own organization. Such candidates should be able to find endorsers outside their organization.

A strong endorsement will provide a personal angle—facts known to the endorser that will enable the committee to better judge the material in the nomination package. Such insights often help explain the significance of the nominee's contributions.

Only the Strong Survive

It is critical to note that content-free endorsements will not prevail. On occasion, the endorsements are reminiscent of the model recommendation letter composed by Benjamin Franklin:

"Sir: The bearer of this, who is going to America, presses me to give him a letter of recommendation, though I know nothing of him, not even his name ... As to this gentleman, I must refer you to himself for his character and merits, with which he is certainly better acquainted than I can possibly be. I recommend him, however, to those civilities which every stranger, of whom one knows no harm, has a right to..."

Endorsements that carry little weight include:

- ▶ Endorsements with no text attached.
- ▶ Perfunctory endorsements that say only something like "I know John Smith and he satisfies, in my opinion, the criteria for Distinguished Engineer."
- ▶ Endorsements that merely repeat text from the nomination.


What comprises a great nomination package? Depending on the contributions (packages will vary), successful nominations tend to have the following qualities:

Educator. The committee looks for someone whose work as an educator has had an impact on other educators as well as students. The package should outline the nominee's impact both within and outside their institution. Letters of support should include at least one person in a significant leadership role at the nominee's institution, and one person who can speak to their contributions in the broader community.

Engineer. The ideal nominee is someone who has led an engineering and/or product effort, and who ultimately delivered a result (typically a product and/or patents) that has demonstrated impact in their area of expertise. The package should outline the nominee's technical contribution. Letters of support should include at least one person who is well known in their technical area, and at least one letter from someone outside the nominee's institution or company.

Scientist. The committee seeks a candidate who is a recognized leader in the research field. The nomination should include a brief description of the field, leadership examples, and why the nominee's contribution is important. Letters of support should include at least one person well known in the nominee's research area. A letter, from a person at a different institution, could address the broader impact of the research.

As ACM is an international organization, the committee receives nominations from around the world. Unfortunately, we do not have representatives from every country, and at times it is difficult to assess the impact of the contributions. Successful nominations for scientists, for example, often include endorsements that illustrate participation and leadership in international research communities. For engineers, we look for products with broad recognition beyond country boundaries. An endorsement from an ACM Fellow or Distinguished Member also helps to calibrate contributions across borders.

It is said that "success has many fathers, while failure is an orphan." Nominations to advanced ACM membership grades reverse this adage: A success reflects on the unique contributions of the nominee; failures can be due to a weak case, a weak nomination, weak endorsements, or faulty judgment by committee members. Nominators can improve their odds by following the advice noted here, and by carefully following the instructions on the ACM Web site (<http://plone.acm.org/membership/distinguished>). 

Marc Snir and Telle Whitney are co-chairs of the ACM Distinguished Members Committee.

© 2010 ACM 0001-0782/10/0700 \$10.00



The image shows the cover of the journal 'ACM Transactions on Reconfigurable Technology and Systems'. The cover features a large, stylized diamond shape composed of smaller colored diamonds (purple, blue, green, yellow) in the top right corner. The title 'ACM Transactions on Reconfigurable Technology and Systems' is prominently displayed in blue and green. Below the title, there is a list of articles with their authors and page numbers. At the bottom of the cover, it says 'Distinguished by Computing Machinery' and 'Association for Computing Machinery'.

This quarterly publication is a peer-reviewed and archival journal that covers reconfigurable technology, systems, and applications on reconfigurable computers. Topics include all levels of reconfigurable system abstractions and all aspects of reconfigurable technology including platforms, programming environments and application successes.

www.acm.org/trets
www.acm.org/subscribe

 Association for Computing Machinery

Article development led by [acmqueue](http://queue.acm.org)
queue.acm.org

**Maybe it's Fortran.
Or maybe it just doesn't matter.**

BY EUGENE LOH

The Ideal HPC Programming Language

THE DARPA HIGH Productivity Computing Systems (HPCS) program sought a tenfold productivity improvement in trans-petaflop systems for high-performance computing (HPC). This article describes programmability studies undertaken by Sun Microsystems in its HPCS participation. These studies were distinct from Sun's ongoing development of a new HPC programming language (Fortress) and the company's broader HPCS productivity studies, though there was certainly overlap with both activities.

These programmability studies began with a focus on programming languages, but the focus quickly shifted to other topics. Existing languages—notably Fortran, which is arguably still the primary language in HPC—proved remarkably adequate. Programming challenges stem mostly from other factors.

What if programming did not mean having to learn a language someone else devised and then wrestling with the limitations of that language, its compilers,

and computers to implement your task? What if it meant, in a sense, the opposite? You could write your program in whatever way was most expressive for you, without regard for language rules imposed by someone else. Then it would be someone else's job to define the programming language that would make sense of what you wrote, write the compilers to digest the program, and build the computers that would efficiently run the task you specified.

We undertook such an exercise to get a feel for what an "ideal" programming language for HPC applications might look like. Our approach was to take existing HPC programs and have someone rewrite them in whatever way suited that individual, not bound by the constraints of any existing computer, compiler, or language. Rather, he was invited to write whatever seemed most expressive. We might not be able to compile or run these programs, but we could at least see what the writer wanted.

Almost immediately, we were struck by what we were seeing. Of course, the rewritten code was much more compact and readable than the original, but, surprisingly, the "ideal" programming language was basically Fortran.

My first job here is to convince you that this finding is not ridiculous. I admit, the experiment was biased in that we were starting with existing code, mostly written in Fortran, and used a human subject who was not only familiar with Fortran but indeed embraced it. The main point, however, is less that every programmer would have ended up preferring Fortran and more that the problems with the original source code have more to do with reasons other than the limitations of existing programming languages. We look at some of these reasons here.

The DARPA HPCS program also sponsored the development of new programming languages: Chapel from Cray, Fortress from Sun, and X10 from IBM. Proponents of those languages would show early on how rewriting



The U.S. Department of Energy's Jaguar supercomputer took home a trio of gold medals—for speed, sustainable memory bandwidth, and for FFT execution—at the recent HPC Challenge competition.

familiar HPC benchmarks in the new languages could reduce source-code volume substantially—tenfold reductions were not surprising—but rewriting these benchmarks even in Fortran achieved similar source-code reductions and corresponding improvements in expressivity.

New programming languages still have much to offer, for example, in the areas of expressing concurrency and especially data distribution. It's just that the bloat we see in current HPC source code stems not so much from inadequacies in current languages as from other factors.

What We Did

We rewrote a number of HPC benchmarks and applications using modern Fortran in a way that took into ac-

count the human costs of software development: programmability and associated characteristics such as readability, verifiability, and maintainability. These are important considerations; although copy-and-paste is a fast way of writing lines of code, it degrades readability and increases maintenance costs.

Part of this effort included working with the Sun HPCS productivity group to quantify programmer productivity in general and to study human subjects in our rewriting exercises in particular. A human subject's activities could be observed passively with the Hackstat telemetry tool or actively via interviews or having the subject keep a journal. The team included a cultural anthropologist who guided these observations.

In this article, we focus on the output of the rewriting activity, examining the rewritten HPC programs and causes of source-code bloat. The particular HPC test codes used here are the NPBs (NAS Parallel Benchmarks) CG, MG, and BT; the plasma fusion application GTC; and the 3D hydrodynamics code SPPM.

A key metric was the number of source lines of code (SLOC). This is admittedly a crude and often deceptive metric, but it served as a convenient starting point for quantifying readability and expressivity of source code.

Since the generated computer programs were in Fortran, they could be compiled and run. Thus, we were able to study their performance relative to the original code, test automatic par-

allelization with currently available tools, and speculate on the potential for improvements in autoparallelization.

Table 1 lists SLOC and performance comparisons between original and rewritten versions of some of the HPC codes we studied.

Author Feedback

We saw remarkably large reductions in source-code volume. The smallest reduction was in GTC, which already used relatively modern Fortran constructs, had relatively little MPI (Message Passing Interface) parallelism (distributed-memory), and had computation and I/O formatting that the human subject was uncomfortable modifying.

We saw various indications that the rewritten programs not only had fewer lines of code, but also were easier to read, verify, and modify. It was not simply our judgment, however, that concluded that expressivity could be improved tremendously. In the case of GTC, we solicited feedback on the rewritten program from one of the application's maintainers. Here are selected comments:

At first glance, I was impressed by how small and compact the code had become. I always thought that GTC was as small as it could get, but I was obviously wrong. I was also pleasantly surprised

to discover that the programming language was still standard Fortran 90/95, and not a totally new language.

The new code is clear, concise, and easy to read.

The fact that all the MPI calls and OpenMP directives have been removed makes the physics represented in the code easier to follow.

[The rewrite introduced elegant] code reuse in CHARGE and PUSH.

But there was this warning:

[Expect a] performance hit unless the compiler can perform very good interprocedural optimization and/or automatic inlining.

This warning arose because there were many transformations from continuous (ζ, r, θ) coordinates to discretized mesh indices. The readability and maintainability of the source code benefited greatly from encapsulating these many transformations into a few functions, but the performance suffered from the extra procedure calls and loss of many specializations and optimizations of the transformations.

Single-CPU Performance

Much of HPC is performance, including parallelization. The code we examined showed many familiar HPC characteristics: loop unrolling, vectorization, cache blocking, multithreading, data distribution, and so on. One

might argue that, while it may be possible to reduce code volume dramatically, the cost in overall performance would be intolerable.

We were pleasantly surprised that single-CPU performance degradation wasn't too bad in general. Indeed, for NPB CG, most of the work is performed by low-level sparse-matrix routines, and overall performance really didn't change at all. We expect similar results whenever the computationally intensive kernels—sparse-matrix routines, dense matrix multiplies, FFTs (fast Fourier transforms) among others—are performed in library or other well-tuned kernels.

In other cases we saw slowdowns but expected to recover much of the performance with judicious, tactical (few-line) optimizations. For example, the rewrite of the NPB MG code saw a 2x speedup by converting stencil operations from array syntax to (arguably more readable) DO loops. In GTC, one section of code ran four times faster when the Fortran MODULO intrinsic was replaced by a suitable substitute. Such optimizations, of course, place one on a slippery slope. Code bloat creeps back in, and maintainability of the code degrades. Indeed, even performance can suffer. We have seen cases where simplifying the source code by removing “optimizations” actually improved performance, presumably because the “optimizations” originated on sufficiently different hardware or targeted sufficiently different compilers.

Meanwhile, the battle to deliver good performance on expressive HPC source code must still be waged. Compiler optimizations must be augmented with ongoing hardware improvements. There is much work to be done on latency-hiding techniques such as prefetch, chip multithreading, and scout threads. To some extent, this simply moves the pressure from memory latency to memory bandwidth; thus, some system designers tackle other problems such as efficient use of partial cache lines.

Parallelization

HPC parallelization often falls into two categories: finding concurrency and distributing data. Finding concurrency is much simpler than dis-

Table 1. HPC code comparisons.

Code Name	Lines of Code			Performance Slowdown
	Before	After	Reduction	
NPB CG	839	81	10x	1x
NPB MG	1701	150	11x	2x–6x
NPB BT	4234	594	7x	2.7x
GTC	6736	1889	3.6x	2.7x
sPPM	13606	1358	10x	2x

Table 2. Possible pseudocode for the ADI algorithm.

```

INTEGER NX, NY, NZ, X, Y, Z
DIMENSION MYDATA(NX, NY, NZ)
FORALL ( X = 1:NX, Y = 1:NY ) CALL UPDATE(MYDATA(X, Y, :))
FORALL ( X = 1:NX, Z = 1:NZ ) CALL UPDATE(MYDATA(X, :, Z))
FORALL ( Y = 1:NY, Z = 1:NZ ) CALL UPDATE(MYDATA(:, Y, Z))

```

tributing data. Our guarded optimism regarding existing languages extends even to parallelization if, by that, we mean finding concurrency. If data distribution is needed to achieve high-end performance, however, new programming languages or constructs seem that much more crucial.

HPC seldom uses locks. More typically, concurrency is related to data-parallel loops—for example, time stepping all particles or grid elements concurrently. Meanwhile, clusters of commodity computers have become the price-performance winners in HPC. Therefore, parallelization also involves the decomposition of data over cluster nodes. Nodes share data in HPC typically through explicit message passing, for example with MPI.

Consider the alternating direction implicit (ADI) method for solving partial differential equations. Specifically, consider a 3D rectangular grid, such as that shown in the accompanying figure on the right. Physically, the information on any grid cell propagates throughout the 3D volume, ultimately influencing all other cells. Computationally, we can restrict data propagation along only the x -axis in one phase of computation, later along the y -axis, and finally along the z -axis. Ultimately, the computed physics should remain unchanged.

Such an algorithm organizes computation along “pencils” of cells. For example, in the first phase, all cells in an X -aligned pencil can be updated based solely on data values within this pencil. Indeed, all X -aligned pencils can be updated concurrently; then all Y -aligned pencils; and finally, all Z -aligned pencils. If there are N^3 elements in the grid, then there are N^2 pencils in each of the X , Y , or Z phases. That is to say, there is considerable

concurrency. The BT and sPPM codes both are organized like this, as are multidimensional FFTs. The pseudo-code might look like Table 2.

Each subroutine call can be made concurrently with all other calls in the same FORALL statement. There is, however, no way of distributing the elements of MYDATA onto multiple processors so that each processor has all the data it needs for all stages of computation. If a particular processor “owns” MYDATA(X,Y,Z), then to process an X -aligned pencil of data it needs all MYDATA($:,Y,Z$) values. Then, to process Y -aligned pencils of data, it needs all MYDATA($,:,Z$) values. Finally, to process Z -aligned pencils of data, it needs all MYDATA($,:,:$) values.

Therefore, while concurrency in this example is rife, distributed-memory systems face a great challenge both in exchanging data between processors explicitly and in distributing data so that such costly exchanges are minimized.

Similar issues arise even in shared-memory systems. It may be possible for all processors to access all elements in place, but these accesses must be coordinated, whether to prevent race conditions or to deal with cache coherency. Even shared-memory systems benefit from spatial locality since processors can then deal with complete cache lines.

If we focus on the relatively easier problem of concurrency, we could in the long term help keep the HPC programmer from having to parallelize explicitly. We would benefit from improvements in software. Existing compilers already identify some opportunities for automatic parallelization. This includes progress on autoscoping—that is, automatically analyzing source code to determine the usage

(private, shared, read-only shared, replicated, and so on) of variables so that a loop could be parallelized. Automatic analysis would be aided by whole-program or interprocedural analysis.

Runtime management of concurrency would also help. Loops might be nested, or loop iterations might be unbalanced. Loop counts and processor counts might not be known until runtime. Static analysis alone cannot balance computational loads or judge the balance between fine-grained parallelism (for maximum concurrency) and coarse-grained parallelism (to amortize the costs of parallelization).

Simpler concurrency for the HPC programmer would also benefit from hardware improvements. Large, globally addressable memories help. Processors run faster with cached data, however, so coherency must be managed. Hardware can support concurrency with atomic operations, transactional memory, and active messages.

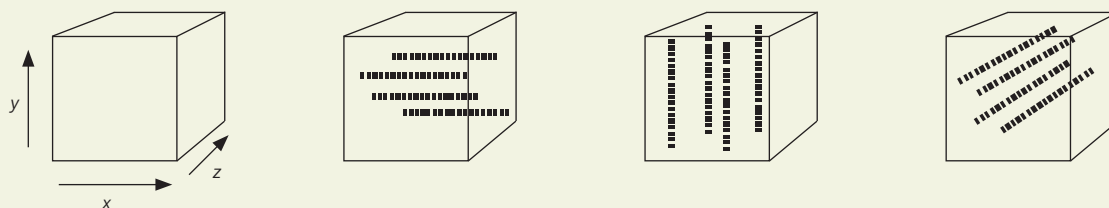
While concurrency seems relatively simpler, managing data distribution seems a much more difficult task. This is one area where new programming languages could really offer help.

For example, the NPB BT benchmark has a cousin, BT I/O, which adds I/O to the test. This offers a test of adaptive maintenance—that is, adding functionality to an already written program. The comparison was almost a joke: setting up I/O in the original, distributed-memory version of the code added 144 source lines, while the rewritten, shared-memory version needed only one extra line!

Algorithmic Complexity

Performance and parallelization are not the only pressures causing large

3D grid showing pencils of cells.



source code. Another issue is that the ideas the computational scientist wants to express are rather low level. For example, the fusion code GTC models the Lorenz force, which a physicist could succinctly write as

$$F = q(E + v \times B)$$

but which the computational physicist transforms into many pages of bewildering equations and commensurately large volumes of computer code. Since charged particles travel in very tight spirals in plasmas, the computational physicist starts by transforming to a “guiding-center” formulation. Then coordinates are transformed to align with the magnetic fields in a tokamak. Such transformations introduce considerable complexity, but they also improve the numerical properties and performance of the code by several orders of magnitude, an advantage that cannot be overcome just by buying more computer equipment.

Generally, computational scientists remove high-frequency components, discretize grids, use sophisticated time stepping, introduce crucial approximations, expand terms, transform coordinates, add dissipative terms and upwind differencing to control numerical stability, and otherwise turn a few simple equations into pages of mind-numbing algorithms that represent the essence of what they’re trying to do computationally. To forgo that algorithmic complexity would increase computational cost by many unaffordable orders of magnitude. A computational scientist’s bread and butter is not simply the equations of mathematical physics (the Lorenz force, Schrödinger’s equation, Navier-Stokes equations, among others), but algorithmic specifications that make computation possible within a particular set of conditions. Fortran is rather good at expressing computational rules. Modern Fortran with array syntax, generic interfaces, optional arguments, recursive subroutines, MODULES, array-valued functions, and other features, is even more so. The ability to have typeset mathematical syntax, as with Fortress or Mathematica, would also be nice.

Other areas that seem to have complexity that would be difficult to express regardless of the programming language include high-level algorithmic control flow and detailed I/O formatting.

Implementing Band-Aids

Source-code volume also expands as a result of limitations—even defects—in current software and hardware. One example is portability. HPC programmers must account for different vendors, MPI implementations, threading models, compilers, Fortran-C interoperability conventions, default word sizes, and so on. In other cases, code takes pains to reproduce particular floating-point numerics (regardless of whether those numerics are right). Inconsistent library availability, whether a result of licensing or installation and bundling issues, also is an issue: while libraries offer all sorts of functionality, HPC code often has its own random-number generators, matrix multipliers, sparse-matrix support, linear solvers, and FFTs to ensure these capabilities will be available regardless of where the application is run.

HPC code sometimes also implements capabilities that might be better provided by tools. Examples include performance instrumentation, debugging code, and checkpointing.

Source code also reflects workarounds to transient bugs or to limitations in compilers. An example is Fortran array syntax. We have found many instances where array syntax allows much higher-level programming. Many programmers, however, have avoided the elegant syntax because its implementation in many compilers is immature. Arguably, developing new programming languages would exacerbate rather than solve such a problem.

Despite our rosy view of existing programming languages, we admit encountering areas where language improvements would have been nice. Type inference, including the inference of array extents, would allow one to forgo tedious boilerplate declarations. Better support of stencils (computations on grids where each element is updated based on nearby elements) is useful for HPC.

Specification, Verification, and Validation

We started with the software development model in which a computer program starts from a written specification. Then, it must be verified (checked against the spec) and validated (checked that it fulfills its intended purpose over some range of parameters).

It is possible that the program is written without verifiability in mind. Here is a striking example from the NPB BT code.

```
rhs(2,i,j,k) = rhs(2,i,j,k) +
dx2tx1 *
(u(2,i+1,j,k) -
2.0d0*u(2,i,j,k) +
u(2,i-1,j,k)) +
xxcon2*con43 * (up1 -
2.0d0*uijk + um1) -
tx2 * (u(2,i+1,j,k)*up1 -
u(2,i-1,j,k)*um1 +
(u(5,i+1,j,k)-
square(i+1,j,k)-
u(5,i-1,j,k) + square(i-
1,j,k))*
c2)
```

This code is basically supposed to implement the following from the NPB1 specification.

$$[RHS2] = \dots - \left(\frac{\partial}{\partial \xi} \right) \left([u^{(2)}]_2 / u^{(1)} + \phi \right) + \left(\frac{\partial^2}{\partial \xi^2} \right) \left(d_{\xi}^{(2)} u^{(2)} + (4/3)k_3 k_4 [u^{(2)} / u^{(1)}] \right)$$

There is little correspondence between the source code and the specification it is supposed to implement. This is not so much a limitation of the programming language but of human intention. Here is how we rewrote the code, with the purpose of improving readability and verifiability.

```
RHS2 = RHS2 - deriv(1,1,u2**2/
u1+phi)
RHS2 = RHS2 + deriv(2,1,dx2*u2
+ 4*k3*k4/3*u2/u1)
```

It is more likely, however, that there isn’t even a spec to verify the code against. When we attempted to verify GTC and asked for a specification for the application, we received this somewhat humorous reply: “There is one physicist at the lab who actually

went through the code line by line and took some notes. Unfortunately, these notes are not in electronic format, and worse...they're in Chinese."

There may have been a specification originally, but the source code evolved over time, while the spec was never updated. To mitigate the divergence of spec and source code, we looked at making source code, even Fortran, as readable as possible and interleaving source code with specification or documentation. We tried an implementation of the HPCS graph analysis benchmark, SSCA #2, where the "source code" was HTML from which a script could extract Fortran code to compile and execute. This approach to having a single artifact to maintain, instead of disjointed specification and source code, is similar to ideas found in Mathematica notebooks, Donald Knuth's *Literate Programming*, and Scientific WorkPlace.

Validation is also difficult. One must compare results in particular parameter regimes to results that might be known from analysis or predecessor codes. Since validation is so expensive and depends so critically on experienced scientific understanding and intuition, over most of an HPC application's lifetime one simply checks software modifications by comparing results with an earlier version of the code. Whereas the science is meaningful to only limited precision (say, 1% or even 10%), checking numerical results in HPC usually means checking fickle floating-point arithmetic out to the least significant digit. We found cases, for example, where we refrained from changing the source code because changing $((2\pi)^k)/N$ to $2*((\pi^k)/N)$ or changing $X*(1/\text{deltat})$ to X/deltat changed floating-point results subtly. We do not know if the results were more accurate or less, only that they were slightly different. These differences prevented us from making the source code more readable or run faster.

Programmers' Priorities

Project deadlines force software to be written quickly. Many expedient writing styles, however, cause programs to become longer and therefore more difficult to read, understand, verify, and maintain. Meanwhile, many pro-

gramming habits develop in a culture of fast prototyping, avoiding advanced language features since their support is immature and focusing instead on the last drop of performance. As presented in Donn Seeley's ACM *Queue* article, "How Not to Write Fortran in Any Language" (December/January 2004/2005), examples of poor programming practices abound.

Programming for verifiability is often not a priority, as the BT example illustrated.

As another example, in SPPM we found thousands of lines of code for handling boundary conditions. The rewritten code used only about a dozen lines. There are many reasons for this astounding reduction, but one issue is that the original code attempted to fill in "ghost cells" only when their values would be needed. (Ghost cells are replicas of real computational cells, where such replication can simplify the handling of boundary conditions.) In the rewrite, we would routinely fill in all ghost cells. Eliminating checks on whether such updates were needed facilitated the programming logic immensely, with nearly no overall performance loss in the cases we studied. In HPC, the mind-set is usually to program for performance rather than programmability even before establishing whether a particular section of code is performance sensitive or not.

The ISO/IEC standard on software maintenance adopted the term *perfective maintenance*. Modifying source code simply to improve its maintainability, however, often receives scant attention when other objectives—such as fixing defects, implementing new features, tuning performance, and migrating to new platforms—clamor for attention.

The NPB BT source code takes hundreds of lines of code to compute the time derivative dU/dt to form the right-hand side. This computation appears to have been implemented from scratch twice, once in file `rhs.f` and again in `exact_rhs.f`. Even if this duplication of effort was overlooked originally, perfective maintenance should weed out such redundancy to benefit the generations of HPC workers who have had to look at this source code since it was first written—provid-

ed, of course, that this is important for the software's owners.

Where Do We Go From Here?

Repeating some of these programmability studies on larger HPC programs would be interesting. In particular, it would be nice to move from self-contained programs that are small enough for one person to have written—what DeRemor and Kron would term "programming in the small"—to larger pieces of software, written by many people and where interfaces among many parts are important ("programming in the large"). Like nature, source code looks different at different scales: from fast prototyping, to self-contained applications, to multi-decade legacy code. Further work to relate source-code characteristics empirically to human productivity metrics would also be interesting.

Most of all, the HPC community on all fronts: language development, compiler maturity, hardware innovations, HPC software development practices, and even procurements and competitive benchmarking.

When we start with an existing language, however, we benefit from available compilers, systems, reference codes, experience, and programmers. ■

Related articles on queue.acm.org

How Not to Write Fortran in Any Language

Donn Seeley

<http://queue.acm.org/detail.cfm?id=1039535>

Languages, Levels, Libraries, and Longevity

John R. Mashey

<http://queue.acm.org/detail.cfm?id=1039532>

Eugene Loh (eugene.loh@oracle.com) is a principal software engineer at Oracle Corporation, Redwood City, CA, where his work focuses on performance of MPI-based HPC applications. He has also worked on programmability, performance, and productivity studies as part of Sun's HPCS activities.

Article development led by [acmqueue](http://acmqueue.queue.acm.org)
queue.acm.org

Heat maps are a unique and powerful way to visualize latency data. Explaining the results, however, is an ongoing challenge.

BY BRENDAN GREGG

Visualizing System Latency

WHEN I/O LATENCY is presented as a visual heat map some intriguing and beautiful patterns can emerge. These patterns provide insight into how a system is actually performing and what kinds of latency end-user applications experience. Many characteristics seen in these patterns are still not understood, but so far their analysis is revealing systemic behaviors that were previously unknown.

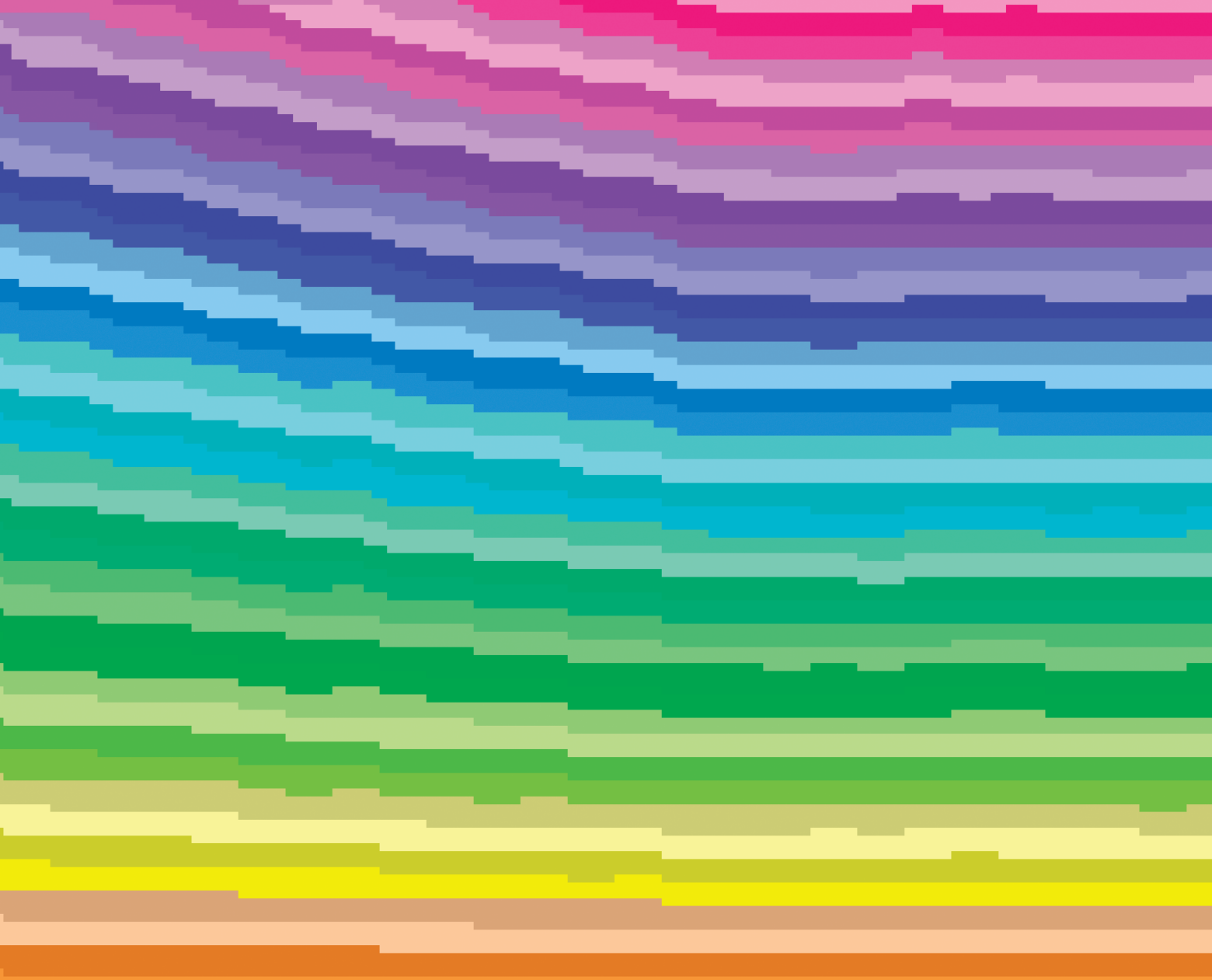
Latency is time spent waiting and has a direct impact on performance when induced by a synchronous component of an application request. This makes interpretation straightforward—the higher the latency, the worse the performance. Such a simple interpretation is not possible for many other statistics types that are commonly examined for performance analysis, such as utilization, IOPS (I/O per second), and throughput. Those statistics are often better suited for capacity planning and

for understanding the nature of workloads. For identifying performance issues, however, understanding latency is essential.

For application protocols measured from the application server, latency can refer to the time from when a request was received to when the completion was sent—for example, the time for a Web server to respond to HTTP GETs or a file server to respond to NFS (network file system) operations. Such a measurement is extremely important for performance analysis since the client and end users are usually waiting during this time.

For resource components such as disks, latency can refer to the time interval between sending the I/O request and receiving the completion interrupt. High disk latency often translates to application performance issues, but not always: file systems may periodically flush dirty cached data to disks; however, the I/O is asynchronous to the application. For example, the Oracle Solaris ZFS file system periodically flushes transaction groups to disks, causing a spike in average disk latency. This does not reflect the file-system latency experienced by ZFS consumers, since the average disk latency includes asynchronous writes from the transaction flush. (This misconception would be alleviated somewhat if read and write latency were observed separately, since the transaction flush affects write latency only.)

While it's desirable to examine latency, it has been historically difficult or impossible to measure directly for some components. For example, examining application-level latency server side may have involved instrumenting the application or examining network packet captures and associating request to response. With the introduction of DTrace,¹ however, measuring latency at arbitrary points has become possible for production systems—and in real time.



Latency Heat Maps

Given the ability to trace latency at arbitrary points of interest, the problem becomes effective visual presentation of this data. Busy systems can be processing hundreds of thousands of I/O events per second, each one providing a completion time and I/O latency. One approach is to summarize the data as average and maximum latencies per second, which can be presented as line graphs. While this would allow average latency to be examined over time, the actual makeup or distribution of that latency cannot be identified beyond a maximum, if provided.

To examine a distribution over time, visualizations such as heat maps may be used. The use of heat maps in system observability tools has been infrequent, with some appearances to map the access pattern of disk I/O. An exam-

ple of this is taztool (1995), which displays a heat map showing time on the *x*-axis and disk I/O offset on the *y*-axis, allowing random and sequential disk I/O patterns to be identified by visualizing the location of disk I/O.⁵

To visualize the distribution of latency over time, a heat map can be created with time on the *x*-axis and latency on the *y*-axis. The heat map is a color-shaded matrix of pixels, where each pixel represents a particular time and latency range. The amount of I/O that occurs in that time and latency range is shown by the color shade of the pixel: darker colors for more I/O, lighter colors for less. Apart from showing the latency distribution, the heat map also conveys details on maximum and average latency by looking for the pixel with the highest latency and where the darkest colors are grouped.

For the latency heat map to be most effective, the time and latency ranges represented by each pixel should be sufficiently large to allow multiple I/O operations to fall within them. This allows darker shades to be selected and patterns shown by different shades to be observed. If the ranges are too small, many of the pixels may represent only one I/O, and much of the heat map may appear in the same color shade; it may also reduce the likelihood that adjacent pixels are shaded, and the heat map may look more like a scatter plot.

The range of possible color shades from light to dark may be applied to each heat map generated. This can be applied linearly: the pixel with the most I/O is assigned the darkest color, and all other pixels are given a shade that is scaled from the darkest I/O count. A drawback with this approach is that

important details may appear washed out. Latency deviating from the norm is particularly important to examine, especially occurrences of high latency. Since these may represent only a small fraction of the workload—perhaps less than 1%—the color shade may be very light and difficult to see. A false color palette can be applied instead to highlight these subtle details, given the trade-off that the color shades then cannot be used to gauge relative I/O counts between pixels.

A particular advantage of heat-map visualization is the ability to see outliers. For the latency heat map these may be occasional I/O operations with particularly high latency, which can cause significant performance issues. If the y -axis scale is automatically picked to display all data, outliers are easily identified as occasional pixels at the top of the heat map. This also presents a problem: a single I/O with high latency will rescale the y -axis, compressing the bulk of the data. When desired, outliers can be eliminated so that the bulk of the I/O can be examined in detail. An automatic approach can be to drop a percentage (say, 0.1%) of the highest-latency I/O from the display, when desired.

To generate latency heat maps, data

is collected for each I/O event: the completion time and I/O latency. This data is then grouped into the time/latency pixels for the heat map, and the pixels are shaded based on their I/O counts. If the original I/O event data is preserved, heat maps can be regenerated for any time and latency range, and of different resolutions. A problem with this is the size of the data: busy production systems may be serving hundreds of thousands of I/O events per second. Collecting this continually for long intervals, such as days or weeks, may become prohibitive—both for the storage required and the time to process and generate the heat maps. One solution is to summarize this data to a sufficiently high time and latency resolution and to save the summarized data instead. When displaying heat maps, these summaries are resampled to the resolution desired.

Heat Map Explained

Latency heat maps were implemented as part of an Oracle system-observability tool called Analytics. The implementation allows them to be viewed in real time and continually records data with a one-second granularity for later viewing. This is made possible and optimal by DTrace, which has the ability

to trace and summarize data in-kernel to a sufficient resolution and to return these summaries every second to user-land. The user-land software then resamples the summarized data to produce the heat maps.

The heat map in Figure 1, an example screenshot from Analytics, shows the latency distribution of an NFS read workload and the effect on NFS latency when using an additional layer of flash-memory-based cache. This cache layer was enabled at 19:31:38, which has been centered on the x -axis in this screenshot. Explaining this heat map in detail will show how effective this visualization is for understanding the role of these system components and their effect on delivered NFS latency.

In this screenshot, a panel is displayed to the left of the heat map to show average IOPS counts. Above and below the panel the “Range average:” and “8494 ops per second” show the average NFS I/O per second for the visible time range (x -axis). Within the panel are averages for latency ranges, the first showing an average of 2,006 NFS IOPS between 0 and 333 μ s. Each of these latency ranges corresponds to a row of pixels on the heat map.

For the time before 19:31:38, the system served NFS reads from one of two locations: a DRAM-based cache or disk storage. If the requested I/O was not in the DRAM cache, then it was retrieved from disk instead. In the heat map, two levels of latency can be seen. These correspond to:

- ▶ DRAM hits, shown as a dark line at the bottom of the heat map
- ▶ Disk hits, shown as a shaded cloud of latency from 2ms and higher

This is as expected. DRAM hits have very low latency and are shown in the lowest-latency pixel. This pixel represents latencies between 0 and 333 μ s, which is the resolution limit of the currently displayed heat map. Since the recorded data has a higher resolution, this heat map can be redrawn with different vertical scales to reveal finer details. By zooming to the lower latencies the DRAM hits were found to be mostly in the range of 0 to 21 μ s.²

The latency for disk hits has a wide distribution, from about 2ms to the top of the displayed heat map at 10 ms. The returned latency for disk I/O includes rotation, seek, and bus I/O transfer

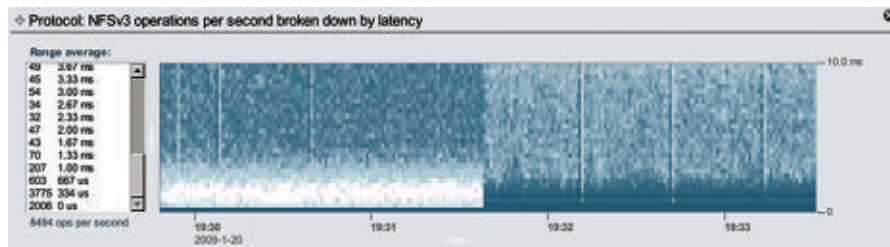


Figure 1. NFS latency when enabling SSD-based cache devices.

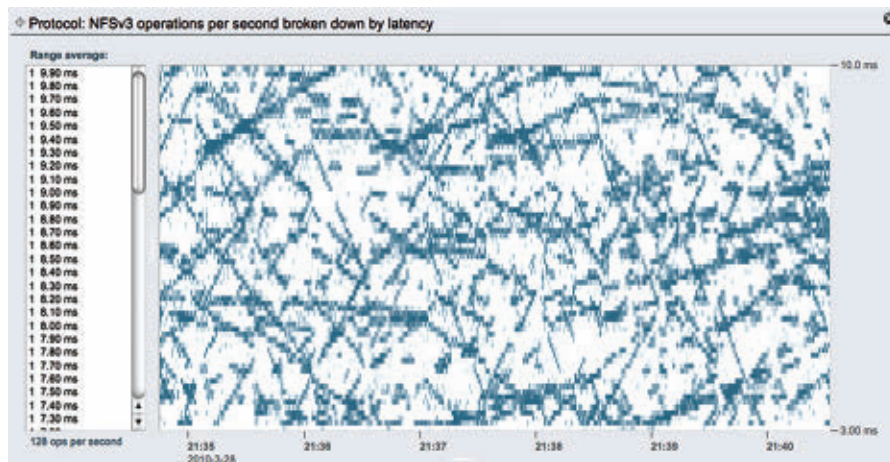


Figure 2. Synchronous writes to a striped pool of disks.

times. As the disks were accessed with a random I/O pattern, rotational latency alone can add up to 8.3 ms, the time for a full rotation on these disks. This rotational latency is presumed responsible for much of the random pattern seen in the heat map.

The heat map before 19:31:38 also identifies a latency range where I/O is less frequent: the lighter band seen from 334 μ s to about 2ms, between DRAM hits and disk hits. This latency gap has been addressed by the hybrid storage pool⁴ in ZFS⁶ by adding a flash-memory-based layer of cache. Flash memory is slower than DRAM and faster than disks, and was incorporated in this NFS server in the form of SSDs (solid state drives). NFS reads may then be served from one of three locations, in order of preference: the DRAM-based cache, the flash-memory-based cache, or disk storage.

Enabling the flash-memory-based cache occurred at 19:31:38, after which three levels of latency can be seen:

- ▶ DRAM hits, shown as a dark line at the bottom of the heat map
- ▶ SSD hits, having a latency of less than about 2ms
- ▶ Disk hits, which have become lighter with the addition of the extra cache layer, since fewer requests are reaching disk

This heat map shows that a flash-memory-based cache had reduced latency for I/O that would otherwise be served from disk. All three system components were visualized, with their latency ranges and the distribution of latency within that range. It also shows that disk I/O still occurs, although at a reduced rate. This is all useful information provided by the heat-map visualization. Imagine presenting this data as a line graph of average latency instead: the only information visible would be a small reduction in average latency when the cache was enabled (small since the average would be dominated by the high number of DRAM hits).

Most heat maps are well understood like this one. What follows are heat maps we have discovered that were not expected and that exhibit interesting patterns that are not fully understood.

The Icy Lake

The workload and target are simple:

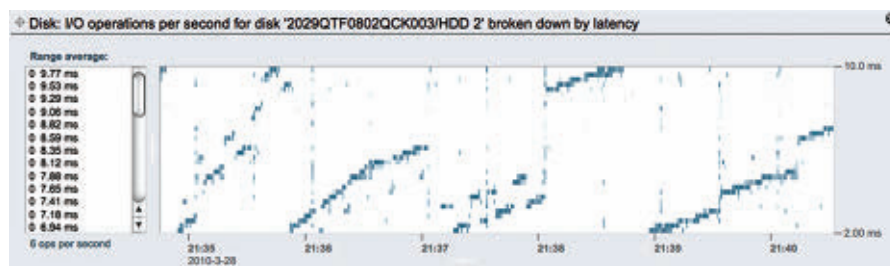


Figure 3. Single-disk latency from striped pool.

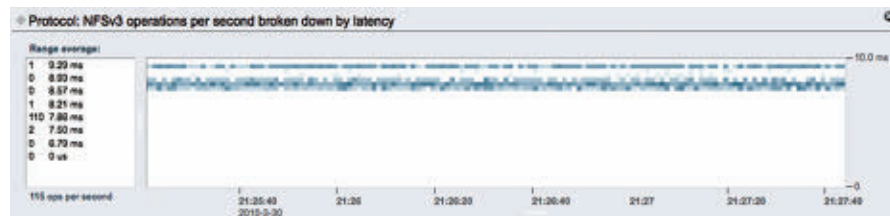


Figure 4. Synchronous write latency to single-disk pool.

A single client has a single thread performing sequential synchronous 8KB writes to an NFS share. The NFS server has 22 x 7,200RPM disks as part of a ZFS striped pool.

Since these are synchronous writes, the NFS request cannot complete until the data is written onto stable storage. No flash-memory-based log devices were used for this test, so latency is expected to be high, as the data must be written to the 7,200RPM disks.

You may expect the latency to be distributed randomly between 0 and 10 ms and for the heat map to appear as white noise. The actual result is shown in Figure 2.

Instead of a random distribution, latency is grouped together at various levels that rise and fall over time, producing lines in a pattern that became known as the icy lake. This was unexpected, especially considering the simplicity of the workload.

This behavior could not be identified from average or maximum latency alone—imagine compressing the y-axis information into a single line graph. This would also be challenging to identify when examining every I/O event, such as by tracing at the disk level using the DTrace-based iosnoop tool, because of the sheer volume of the data (thousands of lines of output).

The first step in understanding this pattern is to check if each of the 22 disks contributed distinct lines. Figure 3 shows the disk I/O latency from a single disk, which confirms that each disk is contributing lines to the pattern.

The next step is to investigate why some lines increase and some decrease. An increase could result from an application requesting I/O in lock-step with disk rotation and ZFS writing sectors along tracks on disk, increasing disk-rotation latency with each I/O (although this doesn't explain how latency could increase for some disks and decrease for others).

To simplify matters, the test was repeated with a single disk pool. Figure 4 shows that most of the NFS latency was between 7.86ms and 8.20ms, which is close to the 8.33ms rotation speed of the 7,200RPM disk. The disk I/O offsets were examined (using both Analytics and iosnoop), which showed that ZFS was writing the 8KB I/O sequentially across the disk. The reason for the smaller NFS latency may be the client and network latency: once one I/O completes, the disk continues to turn while the NFS completion is sent to the client; the client processes it, requests the next write, and then the next write is requested to the disk. By the time this has happened, the disk has rotated a little and so doesn't require a full rotation to write out the next offset. This would explain most of the I/O shown in the heat map; however, the reason for the line at the top is still unknown (it shows an average of one I/O per second from 9.29ms to 9.64ms and is made clearly visible by the false color palette used by Analytics).

ZFS serves synchronous writes by writing to ZILs (ZFS intent logs), which are later grouped and flushed to disk

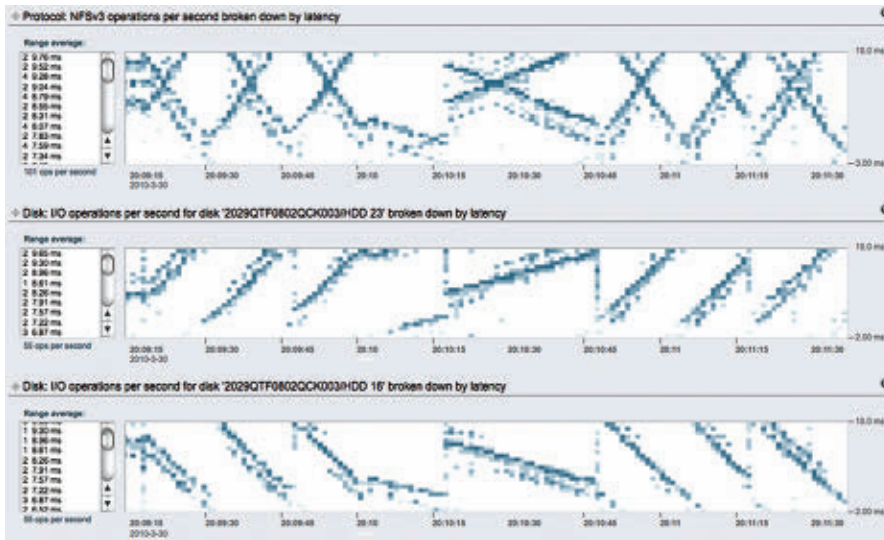


Figure 5. Synchronous-write latency to a two-disk pool.

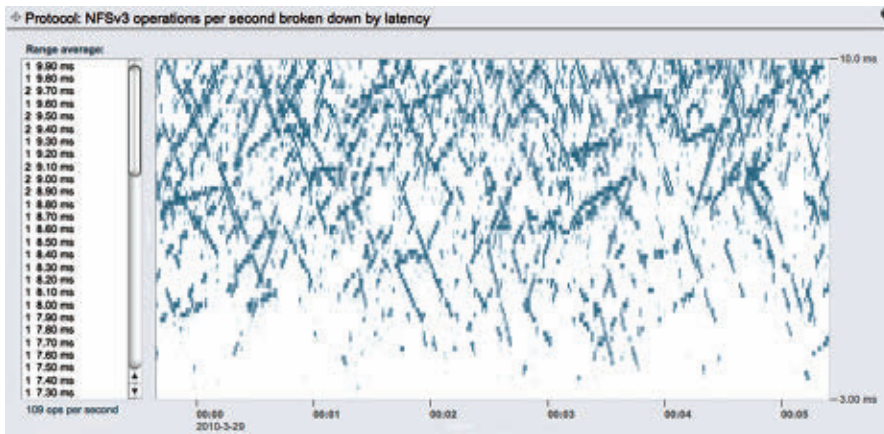


Figure 6. Synchronous writes to a mirrored pool of disks.

as a TXG (transaction group). The ZIL is expected to be written sequentially, and so the heat map is also as expected (with the exception of the line at the top). This will differ for a two-disk striped pool, since ZFS will have a ZIL on each disk and write to them in round-robin fashion. This was tested, and Figure 5 shows the resultant NFS latency on a two-disk pool and the disk I/O latency from each disk in the pool. The reason for increasing and decreasing latency can now be theorized: as the latency on one disk increases, the other disk continues to turn and by the time a request is issued has a corresponding smaller latency. The heat map in Figure 2 is an extension of this, with 22 disks instead of two.

The reason for the slope in the first place has still not been pinpointed. The disks are writing to a steadily increasing offset, which is expected to be placed in a sequential manner along

a disk track (it's up to the disk what it really does with it). If the starting point of rotation were fixed, the rotational latency to each write would steadily increase as the disk turns farther to reach the increasing offset (until a full revolution is reached). The starting point isn't fixed, however; rather it is the end point of the previous I/O, which includes the starting offset and I/O size. Since each offset delta and I/O size is the same, the rotational latency to the next I/O should also be the same. As with the single-disk pool analysis, the slope may actually be a result of client and network latency while the disks continue to rotate. The reason the slope changes is also unknown, as seen in Figure 5 between 20:10:00 and 20:10:45.

This workload was tested on other storage configurations such as mirroring, single-, and double-parity RAID. Figure 6 shows this workload to a mirrored pool of 22 disks. Here the ZIL

is mirrored across pairs of disks, and writes to stable storage are not considered completed until the ZIL exists on both sides of the mirror; thus, the NFS I/O latency is from the slowest disk in the pair. This has given the heat map a bias toward the higher latencies. A similar and greater effect was seen for single- and double-parity RAID (their heat-map screenshots are not included here).

To summarize what we know about the icy lake: lines come from single disks, and disk pairs cause increasing and decreasing latency to occur. The actual reason for the latency difference over time that seeds this pattern has not been pinpointed; what causes the rate of increase/decrease to change (change in slope seen in Figure 5) is also unknown; and, the higher latency line seen in the single-disk pool (Figure 4) is also not yet understood. Visualizing latency in this way clearly poses more questions than it provides answers.

The Rainbow Pterodactyl

As with the icy lake, the rainbow pterodactyl is another simple workload that has produced a surprisingly complex pattern. This time disk I/O latency is examined on a system with 48 disks across two JBOD (just a bunch of disks) enclosures. A local workload was executed to investigate I/O bus throughput by adding disks one by one with sequential 128KB reads, while looking for knee points in the throughput graph. The latency was expected to be consistent for the I/O size used and appears as a narrow line, perhaps with a slight increase as contention increased on I/O subsystem buses (which include HyperTransport, PCIe, and SAS). When one of those buses reaches saturation, the latency is expected to increase much more sharply. Therefore, only two features were expected: a gradual increase with consistent latency and later a sharp increase with latency becoming less consistent because of contention.

Figure 7 shows the throughput graph and latency heat map from this test. A new disk was added to the workload every two seconds, and a knee point in the disk throughput plot can be seen at 17:55. Finding this knee point was the original intent of this experiment; it was the latency heat map

that was eye-catching, however. We named it the rainbow pterodactyl.

The disk I/O bytes graph (the rainbow) shows three features: the initial rise, followed by a decreased slope, and then decay. By corresponding the disk graph with the heat map, characteristics in the heat map can be seen to occur at certain disk counts. The heat map shows the following features.

The “beak” occurs from disk one to disk eight. The reason for two levels of latency is not fully understood, but an experiment has provided a clue: if the same data is read repeatedly to ensure disk-cache hits, then only one line is seen with low latency. The two-line pattern happens when these disks are read sequentially, suggesting that the second line is for disk-cache misses. Analyzing this further is difficult with standard tools: input to the disk and its returned latency can be traced, but there is no visibility into disk internals such as the operation of the disk data controller.

When the ninth disk is added, the beak turns into the “head.” The disks are attached using two SAS cables, each x4 ports, providing eight SAS ports in total. Accessing the ninth disk may be causing contention on those ports in the SAS controller and the corresponding random latency pattern. When the disks are attached using a single x4 SAS cable, the beak-to-head transition occurs at the fifth disk.

A “bulge” forms at the top of the head between disks 9 and 12, showing slightly increased latency. The reason for this is not certain, though it may be from increasing contention for the SAS ports. Nor is the reason known for the reduced latency that forms the “neck” at disks 13 and 14.

Approximately between disks 15 and 20 is the “wing.” This sudden increase in latency causes the knee point in the disk-throughput graph. The source for this contention is not known, although another disk-scaling experiment using a single x4 SAS cable to a single JBOD produced a wingless pterodactyl.

From about disk 20 onward, while disks continue to be added, latency continues to rise and becomes less consistent. This is expected to be PCIe-gen1 bus contention on the SAS controller card.

All of these features are made visible by the heat map, yet are completely unknown by the individual I/O events that form the input: they provide only completion times and I/O latency, while the disk count is increased. The heat map has imaged the I/O subsystem from this data, showing components that are suspected to be disk caches, SAS ports, and the PCIe bus.

To summarize the rainbow pterodactyl: little is known with accuracy, and much more investigation is needed. What this does show is how deep a simple visualization can become.

Latency Levels

For the rainbow pterodactyl, I/O bus

throughput was tested by stepping a sequential disk-read workload. This was repeated on a different system with a more powerful I/O subsystem, and it was found that sequential disk reads from all available disks could not reach I/O bus saturation (no knee point). To see if a limit could be found, the workload was changed to read the same 128KB from each disk repeatedly, so that each could provide more throughput only by returning from its cache. The result is shown in Figure 8.

A knee point was reached between 15:39 and 15:40, although it is difficult to see in the disk bytes graph. At this point, a level of increased latency appears; a little later, there is

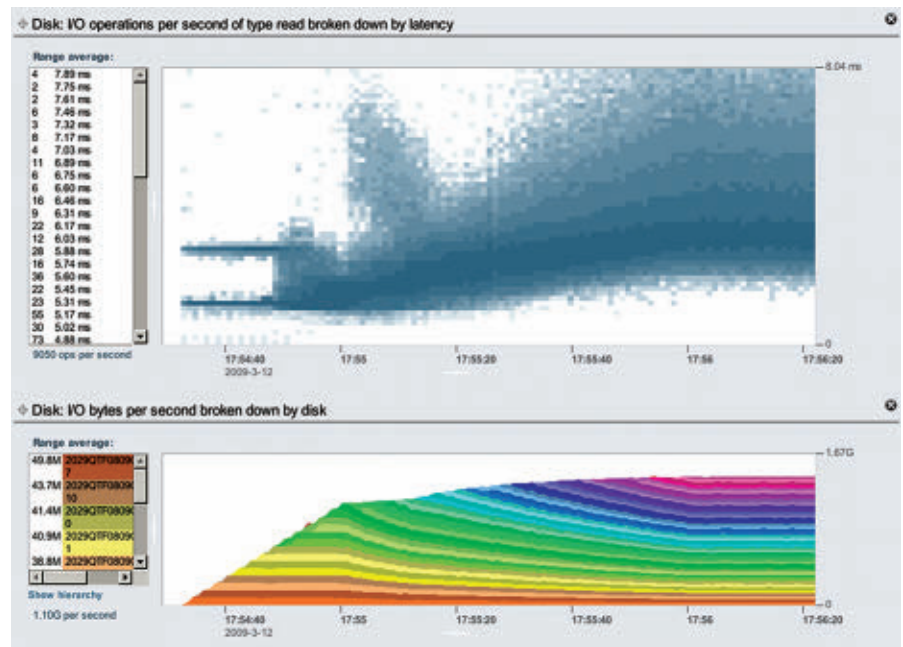


Figure 7. Sequential disk reads, stepping disk count.

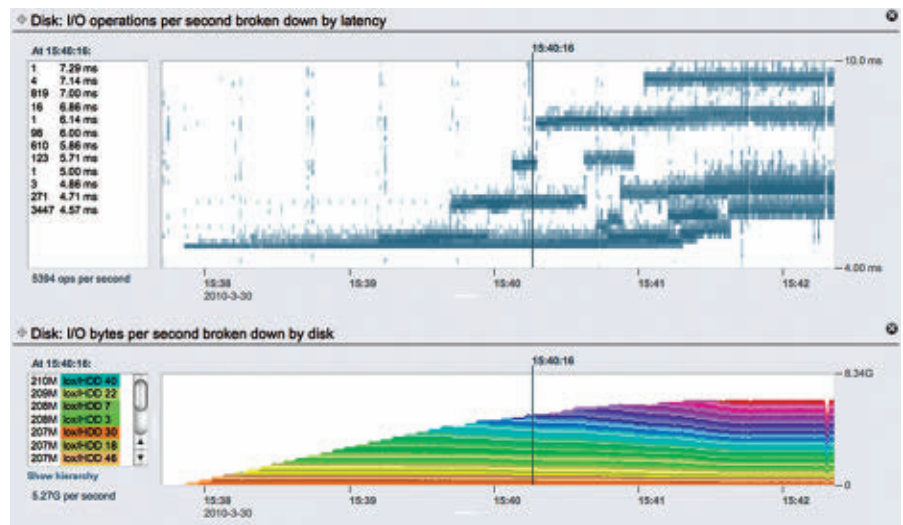


Figure 8. Repeated disk reads, stepping disk count.

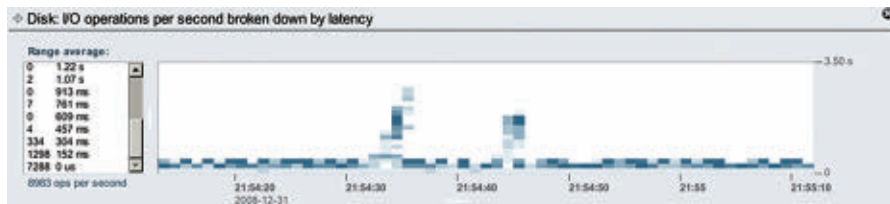


Figure 9. High-latency I/O.

another level (which was selected in this screenshot). At various points it appears as though a latency level has been promoted to a higher level. This was recently discovered and so far is not clearly understood. It is provided here as another example of unexpected details that latency heat maps have exposed.

Shouting at JBODs

Although not as beautiful as the previous examples, the story behind the next heat map has gained some notoriety and is worth including to stress that this was a latency heat map that identified the issue.

The system included several JBODs with dozens of disks and was performing a streaming write workload. I discovered that if I shouted into the JBODs as loud as I could, the disks returned I/O with extremely high latency. Figure 9 shows the heat map from this unusual test.

The heat map shows two spikes in latency, corresponding to each of my shouts. We videotaped this discovery and uploaded it to YouTube, where I describe the effect as disk vibration.³ It has since been suggested that this is better described as shock effects, not vibration, because of the volume of the shouts.

The affected disk I/O shown in the heat map has very high latency—more than one second. If average latency were tracked instead, a few high-latency I/O events may be drowned out on a system performing more than 8,000 faster I/O events at the same time. The lesson from this experience was how well latency heat maps could identify this perturbation.

Other Applications

The previous examples showed latency heat maps for systems deploying the ZFS file system, accessed over NFS. Latency heat maps are also applicable for other local and remote file system

types (for example, UFS, HFS+, CIFS), where characteristics can be identified and interpreted in similar ways. For example, UFS (Unix file system) as deployed on Solaris executes a thread named *fsflush* to periodically write dirty data to disk. This can update the UFS cylinder group blocks that are spaced across the disk, resulting in high-latency I/O resulting from seek and rotational latency. On older versions of Solaris the interval between writing was five seconds (`tune _t_ fsflushr`), so on a latency heat map of disk I/O this may be easy to identify, appearing as bursts of high latency spaced five seconds apart.

The heat-map visualization can also be applied to other metrics, apart from latency. I/O size can be visualized as a heat map with size (bytes) on the y-axis, allowing any particularly large or small I/O to be identified, either of which is interesting for different reasons. I/O location can be visualized as a heat map (as mentioned earlier) with offset on the y-axis, allowing random or sequential I/O to be identified.

Utilization of components can also be visualized as a heat map showing the percent utilization of individual components, instead of displaying an average percent utilization across all components. Utilization can be shown on the y-axis, and the number of components at that utilization can be shown by the color of the heat-map pixel. This is particularly useful for examining disk and CPU utilization to check how load is balanced across these components. A tight grouping of darker colors shows load is balanced evenly, and a cloud of lighter pixels shows it isn't.

Outliers are also interesting: a single CPU at 100% utilization may be shown as a light line at the top of the heat map and is typically the result of a software scalability issue (single thread of execution). A single disk at 100% utilization is also interesting and can be the result of a disk failure. This

cannot be identified using averages or maximums alone: a maximum cannot differentiate between a single disk at 100% utilization and multiple disks at 100% utilization, which can happen during a normal burst of load.

All of the heat maps mentioned here have been implemented in Analytics. Along with the I/O-latency heat map, the utilization heat maps are proving to be especially useful for quickly identifying performance issues.

Conclusion

Presenting latency as a heat map is an effective way to identify subtle characteristics that may otherwise be missed, such as when examining per-second average or maximum latency. Though many of the characteristics shown in this article are not understood, now that their existence is known we can study them and over time identify them properly. Some of the heat maps, such as the rainbow pterodactyl, are also interesting examples of how deep and beautiful a simple visualization can be. □

Related articles on queue.acm.org

Hard Disk Drives: The Good, the Bad and the Ugly

Jon Elerath

<http://queue.acm.org/detail.cfm?id=1317403>

Hidden in Plain Sight

Bryan Cantrill

<http://queue.acm.org/detail.cfm?id=1117401>

Fighting Physics: A Tough Battle

Jonathan M. Smith

<http://queue.acm.org/detail.cfm?id=1530063>

References

- Cantrill, B. 2006. Hidden in plain sight. *ACM Queue* 4 1 (Feb. 2006), 26–36.
- Gregg, B. DRAM latency; Feb. 6, 2009; http://blogs.sun.com/brendan/entry/dram_latency.
- Gregg, B. Shouting in the datacenter; <http://www.youtube.com/watch?v=tDacjrSCeq4>.
- Levanthal, A. Flash storage memory. *Commun. ACM* 51, 7 (July 2008), 47–51.
- taztool; <http://www.solarisinternals.com/si/tools/taz/index.php>.
- ZFS; <http://en.wikipedia.org/wiki/ZFS>.

Brendan Gregg (brendan.gregg@oracle.com) is a principal software engineer at Oracle, and works on performance analysis and observability in the Fishworks advanced development team. He is also the creator of the DTraceToolkit and is the co-author of "Solaris Performance and Tools."



Think you've mastered the art of server performance? Think again.

BY POUL-HENNING KAMP

You're Doing It Wrong

WOULD YOU BELIEVE me if I claimed that an algorithm that has been on the books as “optimal” for 46 years, which has been analyzed in excruciating detail by geniuses like Knuth and taught in all computer science courses in the world, can be optimized to run 10 times faster?

A couple of years ago, I fell into some interesting company and became the author of an open source HTTP accelerator called Varnish, basically an HTTP cache to put in front of slow Web servers. Today Varnish is used by Web sites of all sorts, from Facebook, Wikia, and Slashdot to obscure sites you have surely never heard of.

Having spent 15 years as a lead developer of the FreeBSD kernel, I arrived in user land with a detailed knowledge of what happens under the system calls. One of the main reasons

I accepted the Varnish proposal was to show how to write a high-performance server program.

Because, not to mince words, the majority of you are doing that wrong.

Not just wrong as in not perfect, but wrong as in *wasting half, or more, of your performance*.

The first user of Varnish, the large Norwegian newspaper VG, replaced 12 machines running Squid with three machines running Varnish. The Squid machines were flat-out 100% busy, while the Varnish machines had 90%

Figure 1. Comparison of runtime speeds of binary heap and B-heap.

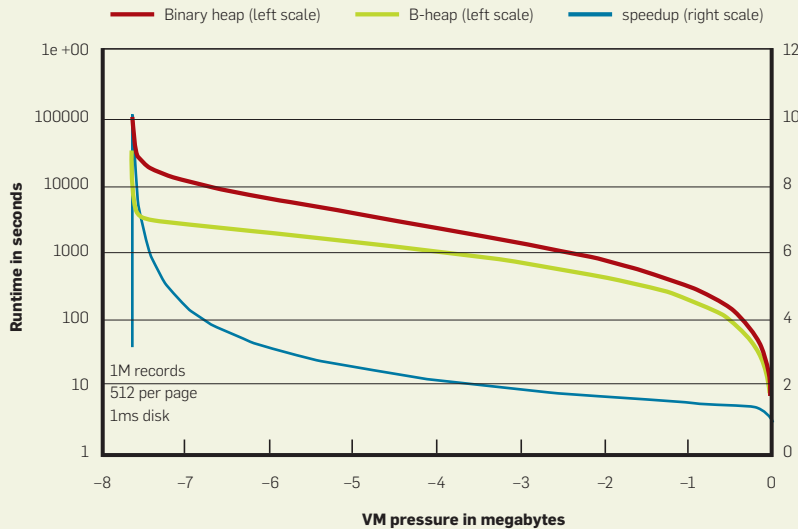
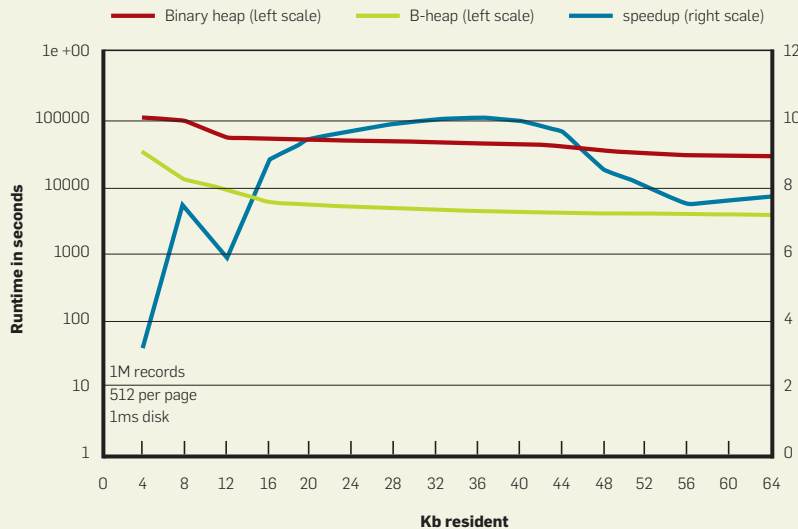


Figure 2. Close-up comparison of binary-heap and B-heap runtime speeds.



of their CPU available for twiddling their digital thumbs.^a

The short version of the story is that Varnish knows it is not running on the bare metal but under an operating system that provides a virtual-memory-based abstract machine. For example, Varnish does not ignore the fact that memory is virtual; it actively exploits it. A 300GB backing store, memory mapped on a machine with no more than 16GB of RAM, is quite typical.

^a This pun is included specifically to inspire Stan Kelly-Bootle.

The user paid for 64 bits of address space, and I am not afraid to use it.

One particular task inside Varnish is expiring objects from the cache when their virtual life-timers run out of sand. This calls for a data structure that can efficiently deliver the smallest keyed object from the total set.

A quick browse of the mental catalog flipped up the binary-heap card, which not only sports a $O(\log_2(n))$ transaction performance, but also has a meta-data overhead of only a pointer to each object—which is important if you have over 10 million objects.

Careful rereading of Knuth confirmed that this was the sensible choice, and the implementation was trivial: “Ponto facto, Cæsar transit,” and so on.

On a recent trip by night train to Amsterdam, my mind wandered, and it struck me that Knuth might be terribly misleading on the performance of the binary heap, possibly even by an order of magnitude. On the way home, also on the train, I wrote a simulation that proved my hunch right.

Before any fundamentalist CS theoreticians choke on their coffees: don’t panic! The P vs. NP situation is unchanged, and I have not found a systematic flaw in the quality of Knuth et al.’s reasoning. The findings of CS, as we know it, are still correct. They are just a lot less relevant and useful than you think—at least with respect to performance.

The oldest reference to the binary heap I have located, in a computer context, is J.W.J. Williams’ article published in the June 1964 issue of *Communications of the ACM*, entitled “Algorithm Number 232—Heapsort.”^{2,b} The trouble is, Williams was already out of touch, and his algorithmic analysis was outdated even before it was published.

In an article in the April 1961 issue of *Communications*, J. Fotheringham documented how the Atlas Computer at Manchester University separated the concept of an address from a memory location, which for all practical purposes marks the invention of virtual memory (VM).¹ It took quite some time before VM took hold, but today all general-purpose, most embedded, and many specialist operating systems use VM to present a standardized virtual machine model (such as POSIX) to the processes they herd.

Of course, it would be unjust and unreasonable to blame Williams for not realizing that Atlas had invalidated one of the tacit assumptions of his algorithm: only hindsight makes that observation possible. The fact is, however, 46 years later most CS-educated professionals still ignore VM as a matter of routine. This is an embar-

^b How wonderful must it have been to live and program back then, when all algorithms in the world could be enumerated in an 8-bit byte.

rassment for CS as a discipline and profession, not to mention wasting enormous amounts of hardware and electricity.

Performance Simulation

Enough talk. Let me put some simulated facts on the table. The plot in Figure 1 shows the runtime of the binary heap and of my new B-heap version for one million items on a 64-bit machine.^c (My esteemed FreeBSD colleague Colin Percival helpfully pointed out the change I have made to the binary heap is very much parallel to the change from binary tree to B-tree, so I have adopted his suggestion and named my new variant a B-heap.^d)

The x-axis is VM pressure, measured in the amount of address space not resident in primary memory, because the kernel paged it out to secondary storage. The left y-axis is runtime in seconds (log-scale), and the right Y-axis shows the ratio of the two runtimes: (binary heap/B-heap).

Let's get my "order of magnitude" claim out of the way. When we zoom in on the left side in Figure 2, we see there is indeed a factor 10 difference in the time the two algorithms take when running under almost total VM pressure: only 8 to 10 pages of the 1,954 pages allocated are in primary memory at the same time.

Did you just decide that my order of magnitude claim was bogus because it is based on only an extreme corner case? If so, you are doing it wrong, because this is pretty much the real-world behavior seen.

Creating and expiring objects in Varnish are relatively infrequent actions. Once created, objects are often cached for weeks if not months, and therefore the binary heap may not be updated even once per minute; on some sites not even once per hour.

In the meantime, we deliver giga-

^c Page size is 4KB, each holding 512 pointers of 64 bits. The VM system is simulated with dirty tracking and perfect LRU page replacement. Paging operations set to 1 millisecond. Object key values are produced by random(3). The test inserts one million objects, then alternately removes and inserts objects one million times, and finally removes the remaining one million objects from the heap. Source code is at <http://phk.freebsd.dk/B-Heap>.

^d Does *Communications* still enumerate algorithms, and is eight bits still enough?

Figure 3. Close-up of the effect of VM pressure on binary-heap and B-heap runtime speeds.

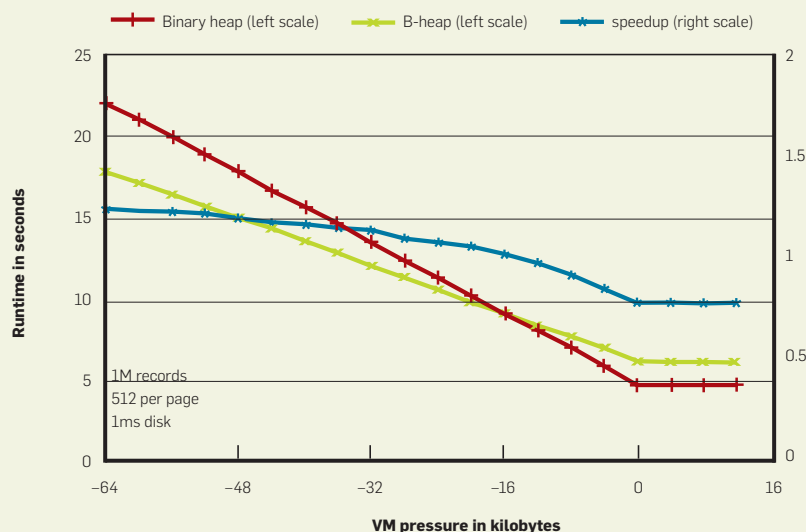
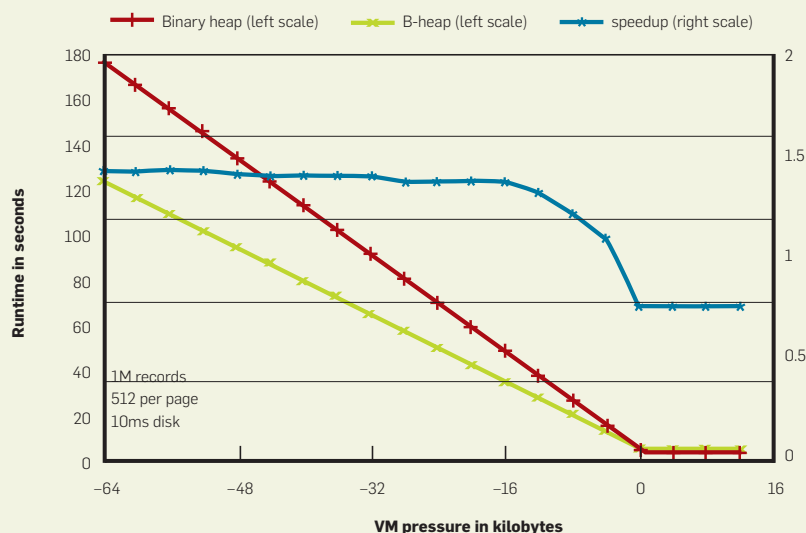


Figure 4. Comparisons of runtime speeds of binary heap and B-heap on a mechanical disk.



bytes of objects to clients' browsers, and since all these objects compete for space in the primary memory, the VM pages containing the binheap that are not accessed get paged out. In the worst case of only nine pages resident, the binary heap averages 11.5 page transfers per operation, while the B-heap needs only 1.14 page transfers. If your server has solid state drives (SSD), that is the difference between each operation taking 11 or 1.1 milliseconds. If you still have rotating platters, it is the difference between 110 and 11 milliseconds.

At this point, is it wrong to think, "If it runs only once per minute, who cares, even if it takes a full second?"

We care because the 10 extra pages needed once per minute loiter in RAM for a while, doing nothing—until the kernel pages them back out again, at which point they get to pile on top of the already frantic disk activity, typically seen on a system under this heavy VM pressure.^e

^e Please don't take my word for it: applying queuing theory to this situation is a very educational experience.

Next, let us zoom in on the other end of the plot (Figure 3). If there is no VM pressure, the B-heap algorithm needs more comparisons than the binary sort, and the simple parent-to-child / child-to-parent index calculation is a tad more involved: so, instead of a runtime of 4.55 seconds, it takes 5.92 seconds—a whopping 30% slower; almost 350 nanoseconds slower per operation.

So, yes, Knuth and all the other CS dudes had their math figured out right.

If, however, we move left on the

curve, then we find, at a VM pressure of four missing pages (= 0.2%) the B-heap catches up, because of fewer VM page faults; and it gradually gets better and better, until as we saw earlier, it peaks at 10 times faster.

That was assuming you were using an SSD, which can do a page operation in 1 millisecond—pretty optimistic, in particular for the writes. If we simulate a mechanical disk by setting the I/O time to a still-optimistic 10 milliseconds instead (Figure 4), then B-heap is 10% faster as soon as the kernel steals just a single page from our

1,954-page working set and 37% faster when four pages are missing.

So What is a B-Heap, Anyway?

The only difference between a binary heap and a B-heap is the formula for finding the parent from the child, or vice versa.

The traditional $n \rightarrow \{2n, 2n+1\}$ formula leaves us with a heap built of virtual pages stacked one over the next, which causes (almost) all vertical traversals to hit a different VM page for each step up or down in the tree, as shown in Figure 5, with eight items per page. (The numbers show the order in which objects are allocated, not the key values.)

The B-heap builds the tree by filling pages vertically, to match the direction we traverse the heap (Figure 6). This rearrangement increases the average number of comparison/swap operations required to keep the tree invariant true, but ensures that most of those operations happen inside a single VM page and thus reduces the VM footprint and, consequently, VM page faults.

Two details are worth noting:

- Once we leave a VM page through the bottom, it is important for performance that both child nodes live in the same VM page, because we are going to compare them both with their parent.

- Because of this, the tree fails to expand for one generation every time it enters a new VM page in order to use the first two elements in the page productively.

In our simulated example, failure to do so would require five pages more.

If that seems unimportant to you, then you are doing it wrong: try shifting the B-heap line 20KB to the right in figures 2 and 3, and think about the implications.

The parameters of my simulation are chosen to represent what happens in real life in Varnish, and I have not attempted to comprehensively characterize or analyze the performance of the B-heap for all possible parameters. Likewise, I will not rule out that there are smarter ways to add VM-clue to a binary heap, but I am not inclined to buy a ticket on the Trans-Siberian Railway in order to find time to work it out.

Figure 5. Binary-heap tree structure.

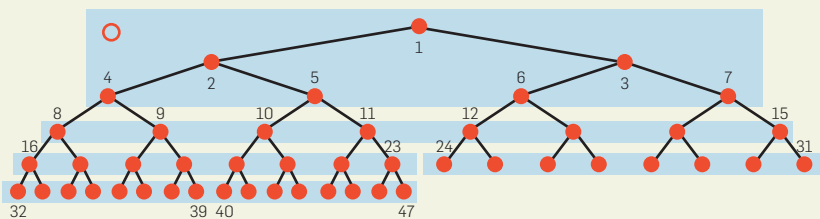


Figure 6. B-heap tree structure.

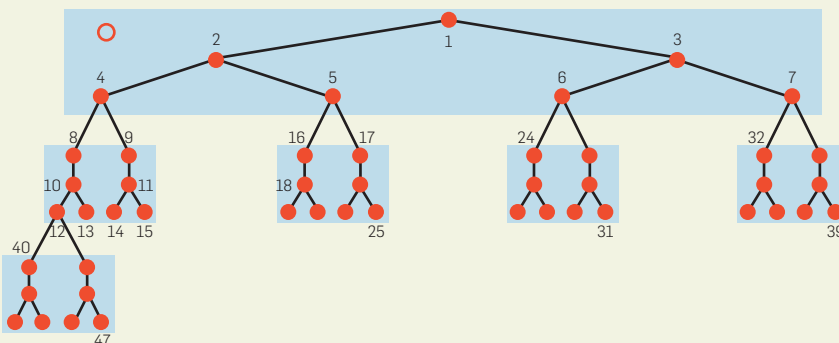
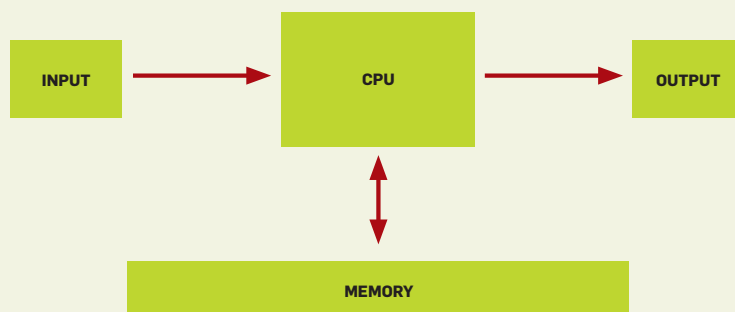


Figure 7. Outdated computer model.



The order of magnitude of difference obviously originates with the number of levels of heap inside each VM page, so the ultimate speedup will be on machines with small pointer sizes and big page sizes. This is a pertinent observation, as operating system kernels start to use super-pages to keep up with increased I/O throughput.

So Why are You, and I, Still Doing it Wrong?

An (in)famous debate, “Quicksort vs. Heapsort,” centered on the fact that the worst-case behavior of the former is terrible, whereas the latter has worse average performance but no such “bad spots.” Depending on your application, that can be a very important difference.

We lack a similar inquiry into algorithm selection in the face of the anisotropic memory access delay caused by virtual memory, CPU caches, write buffers, and other facts of modern hardware.


Whatever book you learned programming from, it probably had a figure within the first five pages diagramming a computer much like the one shown in Figure 7. That is where it all went pear shaped: that model is totally bogus today.

Amazingly, it is the only conceptual model used in computer education, despite the fact that it has next to nothing to do with the execution environment on a modern computer. And just for the record: by modern, I mean VAX 11/780 or later.


The past 30 or 40 years of hardware and operating-systems development seems to have only marginally impinged on the agenda in CS departments’ algorithmic analysis sections, and as far as my anecdotal evidence, it has totally failed to register in the education they provide.

The speed disparity between primary and secondary storage on the Atlas Computer was on the order of 1:1,000. The Atlas drum took two milliseconds to deliver a sector; instructions took approximately two microseconds to execute. You lost around 1,000 instructions for each VM page fault.

On a modern multi-issue CPU, running at some gigahertz clock fre-




Most CS-educated professionals still ignore VM as a matter of routine. This is an embarrassment for CS as a discipline and profession, not to mention wasting enormous amounts of hardware and electricity.



quency, the worst-case loss is almost 10 million instructions per VM page fault. If you are running with a rotating disk, the number is more like 100 million instructions.^f

What good is an $O(\log_2(n))$ algorithm if those operations cause page faults and slow disk operations? For most relevant datasets an $O(n)$ or even an $O(n^2)$ algorithm, which avoids page faults, will run circles around it.

Performance analysis of algorithms will always be a cornerstone achievement of computer science, and like all of you, I really cherish the foldout chart with the tape sorts in Volume 3 of *The Art of Computer Programming*. But the results coming out of the CS department would be so much more interesting and useful if they applied to real computers and not just toys like ZX81, C64, and TRS-80. 

^f And below the waterline there are the flushing of pipelines, now useless and in the way, cache content, page-table updates, lookaside buffer invalidations, page-table loads, etc. It is not atypical to find instructions in the “for operating system programmers” section of the CPU data book, which take hundreds or even thousands of clock cycles, before everything is said and done.

Related articles on queue.acm.org

Thread Scheduling in FreeBSD 5.2

Marshall Kirk McKusick and George V. Neville-Neil

<http://queue.acm.org/detail.cfm?id=1035622>

Flash Storage Today

Adam Leventhal

<http://queue.acm.org/detail.cfm?id=1413262>

High Performance Web Sites

Steve Souders

<http://queue.acm.org/detail.cfm?id=1466450f>

References

1. Fotheringham, J. Dynamic storage allocation in the Atlas Computer, including an automatic use of a backing store. *Commun. ACM* 4, 19 (Apr. 1961), 435–436.
2. Williams, J. W. J. Algorithm 232—Heapsort. *Commun. ACM* 7, 6 (June 1964), 347–348.

Poul-Henning Kamp (phk@FreeBSD.org) has programmed computers for 26 years and is the inspiration behind bikeshed.org. His software has been widely adopted as “under the hood” building blocks in both open source and commercial products. His most recent project is the Varnish HTTP accelerator, which is used to speed up large Web sites such as Facebook.

DOI:10.1145/1785414.1785438

Innate understanding of concurrency helps beginners solve CS problems with multiple processes executing at the same time.

BY GARY LEWANDOWSKI, DENNIS J. BOUVIER, TZU-YI CHEN, ROBERT MCCARTNEY, KATE SANDERS, BETH SIMON, AND TAMMY VANDEGRIFT

Commonsense Understanding of Concurrency

Computing Students and Concert Tickets

CONSTRUCTIVIST LEARNING THEORY says learning is grounded in and constructed from prior understanding and belief. In order to explore the prior understanding and belief of beginning computer science students, the Commonsense Computing Project (<http://commonsensecomputing.cs.siue.edu>) asks them to answer CS questions on the first day of their first course. Here, we report on the related commonsense knowledge as demonstrated in the area of concurrency. Long considered an advanced topic in the CS curriculum, concurrency is rarely^{5,16} considered appropriate for introductory students; on the other hand, it is increasingly at the forefront of CS; obvious instances

involve multicore processors on the desktop, distributed computing resources for computationally and data-intensive problems, and network-based games. Whether through implicit structures in programming languages or explicit structures in design, university students must demonstrate an understanding of the various approaches to and implications of concurrency.

Given the importance of the topic, what relevant knowledge and abilities do students without prior computing instruction (“beginners”) bring to their first class? Do they differ from more advanced students with no explicit teaching about concurrency?

Along with their commonsense understanding of concurrency, this article explores the solutions they provided on the first day of an introductory CS course to a problem devised by Yifat Ben-David Kolikant of Hebrew University of Jerusalem² to more advanced students on the first day of a concurrency course in an Israeli high school. We’ve used the responses to address two questions:

- ▶ Are beginners able to recognize the key concurrency issue regarding use of a shared resource? and
- ▶ How do answers given by beginners compare with answers given by more advanced students, as reported by Ben-David Kolikant?

Of the 66 students in our 2006 study, 97%, or 64, could identify a race condition and 71% provided a solution we considered reasonable. The

» key insights

- **Beginner CS students demonstrate commonsense knowledge about important CS topics that can be leveraged to improve instruction.**
- **Constructivist education begins with commonsense knowledge, helps students discover its utility, and provides tools for more complex approaches rooted in CS knowledge.**
- **Beginner CS students demonstrate intuition about concurrency roughly equivalent to experienced CS students beginning a concurrency course.**



most common technique they reported for avoiding race conditions was subdividing the resources. Our study provides a basis for a constructivist approach to teaching concurrency, allowing instructors to build on these ideas. Moreover, the study was independent of technological assumptions, making it relevant regardless of technology or pedagogical practice used.

Background

We have conducted a number of commonsense computing projects over the past five years, all with the same basic goal of identifying the commonsense knowledge beginners bring to the study of CS. Our foundation was the constructivist theory about how

people learn, starting with what they already know and building knowledge on that foundation, rather than receiving it passively from an instructor. Each learner’s background, culture, and previous knowledge define his/her starting point. Bransford et al.⁴ argued that learning must engage students’ preconceptions to be effective.

The importance of constructivism is recognized in the computing-education community. Ben-Ari¹ compared it with other fields, highlighting several differences; for example, in CS education, students need a model of the computer, and the computer then provides an “effective ontological reality,” or verification of whether a program works or not.

Students’ prior understanding and

beliefs about a particular topic are also considered preconceptions, as explored by several researchers:

► Miller¹⁰ analyzed “natural language” programs by students who previously had not taken a formal programming course, covering the idea of writing computer programs in natural language, and found that a number of standard programming concepts showed up in these natural-language descriptions;

► Onorato and Schvaneveldt¹¹ also looked at natural-language descriptions of a programming task, comparing survey subjects drawn from a variety of student categories: naïve, with no programming experience; beginner, currently taking a first programming course; and expert, with significant programming experience. Along with the differences between experts and novices, they also found differences between the naïves and the beginners, though both groups lacked programming experience;

► Studying the misconceptions of novice programmers, Bonar and Soloway³ focused on preprogramming knowledge, calling it “step-by-step natural language programming knowledge”; they distinguished it from knowledge of the programming language Pascal the students were learning in their introductory course. They found that many of the observed bugs in novice-programmer-written code could be explained by a mismatch in students’ knowledge in these domains; and

► Gibson and O’Kelly⁸ looked at a variety of search problems (with pre-college students) and Towers-of-Hanoi problems (with beginners), finding both groups showed “algorithmic understanding” of how to solve them.

In our own prior work beginning in 2005, we sought to identify student preconceptions that could be leveraged in teaching beginning computing concepts. To investigate student preconceptions about sorting, we asked 118 beginners to describe in words how they would sort a list of numbers into ascending order.¹⁴ A majority (69%) described a coherent algorithm, with many giving versions of selection or insertion sort. However, most treated numbers as strings, manipulating them digit by

Yifat Ben-David Kolikant’s cinema-tickets assignment.

Cinema Tickets Problem

A ticket office sells movie tickets for a certain cinema. The next client always gets the best-available ticket. Software decides the next-best-available seat and prints the ticket.

Assumptions:

- The movie is screened only once;
- This is the only office selling tickets for the movie;
- Each client can buy only one ticket; and
- Many people are waiting to buy tickets.

The software defines several procedures:

Function	Input: Hall
BestAvailableSeat()	Return value: Best-available seat in the Hall; -1 if no seat is available.
Procedure	Input: Seat is the place of an available seat in the Hall.
Mark AvailableSeat(Seat)	Output: The place of the Seat is marked as taken.
Procedure	Input: Seat is the place of an available seat in the Hall.
PrintTicket(Seat)	Output: A ticket for place Seat is printed.

A client is handled through these steps:

```
Seat <= BestAvailableSeat()
If Seat <> -1 then
  MarkAvailableSeat(Seat)
  PrintTicketSeat(Seat)
```

Since waiting in line takes too long, the Hall owners added another ticket office. Both offices are to be open at the same time and sell tickets for the same screening. Each office has its own printer for printing the tickets it sells; ignore money issues. The system must be developed by specifying:

1. Required hardware (screens, printers, keyboards) and how it is to be distributed in the system; and
2. Pseudocode for the system’s software (selling tickets through two offices); the above procedures can be used, with no need to redefine them.

digit. Many used iteration; to our surprise, most iteration involved post-test loops. In 2008,¹³ we described our investigation of students' commonsense knowledge of debugging, giving beginners one of four questions designed to elicit their knowledge of debugging strategies. The questions asked them to describe the advice they would give people whose lights did not turn on when they flipped the switch; how they would locate the moment things go wrong in the children's game "telephone"; how they would find a Starbucks if they were in a strange city where they did not speak the language; or an experience of their own involving troubleshooting.

In general, we found beginners had less commonsense knowledge of debugging than of sorting, and some of their preexisting knowledge did not serve them well. For example, real-world fixes are often easy to undo, unlike programming changes. Likewise, the real world is nondeterministic in ways CS1 programs generally are not; for example, if your car doesn't start, should you immediately turn the key a second time?

A substantial body of work in computing and in other scientific disciplines, including physics and mathematics, involves misconceptions, or incorrect concept understandings, that must be replaced with correct ones. Clancy⁶ provided a survey of this work in CS, and the National Academy's Committee on Undergraduate Science Education⁷ gave a more general overview. Smith et al.¹⁵ challenged this view in the context of math and science education, arguing that misconceptions are limited mental models that can be built onto, ultimately providing a correct understanding of the concept being studied.

Most relevant to the commonsense project discussed here is Ben-David Kolikant's 2001 work examining student preconceptions about concurrency.² She collected data from about 135 Israeli 12th-grade students in six classes at three different schools. The students previously studied computing and at the time of the study were taking an advanced class in concurrent and distributed computing.

At the beginning of the course she

gave them a critical-section problem in which multiple agents share a common resource—two ticket offices selling tickets for the same movie. As shown in the figure, the question was posed in a detailed pseudocode format. It expects a relatively sophisticated answer, including a hardware system specification for supporting multiple machines servicing sales requests, pseudocode for the solution, and an explanation of the answer and how it avoids duplicate ticket sales. Student answers were graded as part of the course.

Ben-David Kolikant divided the responses into two major categories: centralized, involving a solution in which communication and control for the solution are centralized; and decentralized, implicitly or explicitly involving the sellers communicating with one another to achieve a concurrent solution.

The centralized solutions involved three subcategories: (C1) with a central entity, essentially a master computer making all decisions; (C2) in which the solutions involved an assumption either of a constant rate of operations or of operations happening in a particular order; and (C3) in which solutions assume the sellers have private resources, each selling tickets for a separate area of the theater. Ben-David Kolikant described categories C2 and C3 as solutions that "attempted to solve a similar, but different problem."² She divided decentralized solutions into (D1) in which communication is implicit and (D2) in which communication is explicit.

Ben-David Kolikant's discussion focused on three aspects that together contribute to the solutions she evaluated, describing them as "(a) the algorithmic goal of the system, (b) synchronization goals, and (c) reasonableness."²

The algorithmic goal was the problem the system was intended to solve, in this case, selling the ticket for the best available seat. The cinema-ticket problem was well constrained, with student responses all sharing this goal.

The synchronization goals of the cinema-ticket problem were "to coordinate...access to a common resource (the database) in order to avoid sell-

ing the same ticket twice"² and "the prevention of the interleaving of the access to the database."² Ben-David Kolikant found that students did not identify the problem of interleaving in their written answers. Follow-up interviews indicated they assumed the key actions were inseparable, writing, "They assume that the two critical actions of checking and updating the database are always executed successively."² As a result, they assumed the key issue was communication, making sure all sellers were aware of the seats already sold. One-third of the 135 students presented centralized solutions, and the rest presented decentralized solutions.

A final aspect of the students' solutions Ben-David Kolikant considered was "reasonableness," or solving the problem "in a reasonable or realistic way...according to the context of the problem."² She found that students were influenced by their real-world experience with concurrency and networks. For example, C3 solutions were workable real-world solutions but did not ensure buyers would receive the best seat available.

Our 2006 study modified Ben-David Kolikant's problem for presentation to beginners, allowing us to explore commonsense concurrency knowledge and the difference in this knowledge between advanced students and beginner students.

Methodology

Here, we describe our process of data collection and analysis:

Data collection. In the first week of the first semester of the 2006–2007 school year, in a CS1 class, students were randomly assigned one of two tasks: the concurrency task explored here completed by 66 students from five different institutions; the other was unrelated to the study. The participating students were all beginners between the ages of 18 and 20. All but eight completed the questions online (outside of class) by typing English answers into a text box. Eight students (all at the same institution) completed the questions on paper in a laboratory setting. All were given credit for completing the assignment, though solution quality was not evaluated for credit. (The participant/subject iden-

Table 1. Institutional breakdown of respondents; n = number of students answering the concurrency question.

Institution Characterization	n	Class Characteristics
Private, on West Coast of U.S. with approximately 3,200 undergraduates and a school of engineering	25	CS1 serving all engineering, with most of class electrical engineering majors with prior MATLAB course
Public research, on West Coast of U.S. with approximately 27,000 undergraduates and a school of engineering	20	CS1 serving mostly computer science majors
Public research, on East Coast of U.S. with approximately 21,000 undergraduates and a school of engineering	10	CS1 serving computer science and engineering, computer science, computer engineering, and electrical engineering
Public regional, on East Coast of U.S. with approximately 2,000 undergraduates	8	CS1 serving mostly computer science majors
Private liberal arts, on East Coast of U.S. with approximately 1,600 undergraduates	3	CS1 serving computer science majors and minors and math majors

tifiers we cite here are renumbered and do not reflect institutional affiliation; the institutions' characteristics, which vary significantly, are summarized in Table 1.)

The task. Students were asked to address the following: Suppose we sell concert tickets by phone in the following way: When a customer calls and asks for a number (n) of seats, the seller (1) finds the n best seats available; (2) marks those n seats as unavailable; and (3) deals with customer-payment options (such as credit- and debit-card number) or sends the tickets to the will call window for pickup. Now suppose more than one seller is working at the same time. What problems would we see, and how could we avoid them?

Our goal was to determine whether the students would note the race condition between two sellers and suggest solutions to resolve it. There are several differences between this task and Ben-David Kolikant's counterpart. We modified the question to refer to concert tickets, since our students likely had more experience ordering concert tickets. We also removed the restriction that each customer be able to buy only one ticket. And finally, due to our students' lack of background, we phrased the question less technically, asking for responses in English paragraphs, with-

out pseudocode or detailed hardware and software specifications.

Analysis. After collecting the data, we read through all 66 responses for a general sense of their content. We then organized categorizations for the responses based on the categorizations used by Ben-David Kolikant.²

We considered all decentralized solutions in a single category (D), as the less-explicit question led students to provide less-explicit descriptions of this strategy, making it unclear if the communication was explicit or implicit. We also considered C2 and C3 answers (along with C1 and D) as "reasonable" solutions, because our tick-

Table 2. Number of concurrency solutions and problems identified by students; n = 66.

Accomplishment	Percent of students
Number of solutions provided	
One solution	70%
Two solutions	20%
Three or more solutions	10%
Problems identified	
Sell seat more than once	97%
Other	41%
Provided reasonable solution to concurrency problem	71%

et-ordering problem was more open-ended and did not ask students to consider hardware issues that would be necessary to make a solution work.

Several types of responses were considered "non-reasonable." Some were ambiguous (AS), neither centralized nor decentralized. In others (BS), the response could not reasonably be said to solve the problem. Some responses (NS) did not offer a solution. And some responses (NP) provided solutions to problems that were not our central focus, though they may have been interesting, with some even involving concurrency. We recorded the problems (and the suggested solutions) that were not our central focus so we could examine them for commonalities across students.

Unlike Ben-David Kolikant's participants, the students in our study often gave more than one possible solution to a problem, and we counted and coded each of them. After determining the proper categories, five of us coded the data, resolving coding conflicts through discussion.

Results

Here, we offer a sense of the student solutions by viewing them from three perspectives: per student, per categorizations used by Ben-David Kolikant,² and through a qualitative look, with characteristic examples of responses highlighting important aspects of the responses.

Per student. We addressed three questions: How many solutions did the student provide? Was the problem identified? Did the student's solution seem reasonable?

The 66 students in the study collectively produced a total of 97 identifiable solutions (see Table 2). Due to the open-ended nature of the solution we requested, many discussed multiple issues they saw in the problem statement (not all necessarily concurrency-related) or outlined several solutions to the concurrency problem of trying to sell the same seat to more than one person at a time. The majority (70% of the 66) of the students discussed only a single solution, but 20% identified two solutions, and 10% identified three or more solutions, with six solutions being the most identified by any one student.

Of the 66 students, 97%, or 64, identified our main problem of interest—that it may be possible to sell a given seat to more than one person—good evidence that even novice computing students are able to identify this critical concurrency issue. Additionally, six students explicitly noted the problems of interleaving access to a database that could result in one customer reserving but another customer buying the tickets.

In all, 71%, or 47, of the 66 students (73% of those identifying the main problem) identified at least one “reasonable” solution. Moreover, many of the beginners gave more than one type of answer. Of the 47 students who gave a reasonable solution, 12 (or 26%) gave both centralized and decentralized solutions.

Per categorization of solution. As many students provided multiple solutions, it is useful to look at the diversity of responses (see Table 3) out of the total number of solutions provided (97). We found that 69%, or 67, of them were reasonable solutions to the multiple-seat-selling problem; 31%, or 30, were not reasonable solutions; the majority of them (16) involved students describing the problem (often correctly) but without a solution.

Of the 67 reasonable solutions, 55%, or 37, described a centralized solution where the selling entities passed the responsibility of making a seat assignment to some central resource, and 45%, or 30, described a method by which individual sellers made decisions about seat assignments as individual entities.

The centralized solutions were further broken down into three categories, as defined by Ben-David Kolikant: (C1) with a central entity, essentially a master computer making all decisions; (C2) in which the solutions involved an assumption either of a constant rate of operations or of operations happening in a particular order; and (C3) in which solutions assumed that sellers have private resources, each selling tickets for a separate area of the theater. Of the reasonable solutions, 10%, or 7, were of type C1, relying on implicit communication between “dummy” sellers passing the request to a master computing resource to make assign-

ments. For example, one study participant said, “These problems might be avoided by having a computer system automatically (to the second) input the seat reservation for that customer.” [ID438] (Each participant was given an anonymous label.)

Of the 97 reasonable solutions, 13%, or 13 were of type C2, using the same master resource but requiring explicit ordering of communication or steps in the process, including lock-stepping and pipelining the process. An example of a C2 solution: “In order to avoid this [possibility of selling the same seat twice], we could set up the database so only one person could access the database at a time. This would slow sales significantly but is the safest setup.” [ID440]

By far the most popular centralized solution (31%, or 30 responses) was also the most restrictive (C3) and involved distributing or dividing resources, either by portioning out seats in the concert venue to different sellers or serializing or otherwise pipelining the selling process.

Qualitative results. A qualitative analysis of the responses is a useful way to examine other interesting aspects of the 97 student solutions, revealing the range of approaches taken, as well as the depth of student understanding of the computational issues.

Algorithmic goals. Most students (58) did not further refine the goal of their algorithms, either explicitly or implicitly using a goal of “best seats

available” in their responses. Some elaborated further, explaining why they chose a particular solution or the focus of their algorithms.

A number of students were concerned less about choosing seats than about handling seats that are given up. For example, one said, “If the seats are marked unavailable as soon as they are requested by the customer, other sellers cannot access the seats for their own customers at that time. This is a bad thing [less-than-optimal solution] because the better seats reserved by the first customers could potentially still be open should the customers change their minds about the purchase or if payment information cannot be validated. If the seats are marked unavailable, and the payment does not come through for whatever reason, the seats would remain unavailable and be empty during the concert.” [ID415]

Another said, “One obvious problem is two sellers giving up the same seats at the same time.” [ID405]

Some participants changed their algorithmic goals when they realized they could not simply reserve and sell as a single atomic action. For example, one said, “In a very unlikely situation, the sellers could mark the seats ‘unavailable’ at the same time. However, in a more likely situation, one of them would mark seats as unavailable and the other would mark and see that the seats are unavailable, but that seller was not the one reserving them. Then there will be multiple

Table 3. Solution breakdown by type: column 2: considered 97 solutions; column 3: considered 67 reasonable (C or D) solutions.

Category	Of all solutions	Of reasonable solutions
Reasonable solutions (centralized and decentralized)	69%	—
<i>Centralized</i>	38%	55%
C1	7%	10%
C2	9%	13%
C3	22%	31%
<i>Decentralized</i>	31%	45%
Not reasonable solutions	23%	—
Bad solution	5%	—
No solution	16%	—
Ambiguous	1%	—
Solved different problem	8%	—

tickets sent to will call for the same seats.” [ID412]

Citing concurrency, other students expressed concern about dealing with group sales, with one saying, “First of all, there could be an issue of finding group seating. Finding n best available seats will not necessarily do, if they have to be n best available seats together. In such a situation, each seat should be labeled with how many seats are available in front, behind, and to the left and right of it.” [ID425]

Another said, “Scalpers and other ticket-selling agencies will buy tickets and sell them at an increased price; with no limit on n , the number of tickets the caller is purchasing, a single caller could buy every ticket to the concert. This problem is easily fixed by putting an upper limit on n of eight to 12 tickets; large groups can call a special hotline and speak to an operator to purchase more.” [ID430]

The notion of “best seat” attracted further attention, with one participant handling the double-selling problem by dividing the seats among sellers. Most algorithmic attention was then focused on the following problem, as the participant described it: “First of all, there would no longer be a first-come-first-serve basis, and problems would arise over who actually occupies the good seats first. If it’s only one seller, she would be able to take one customer at a time. Two or more sellers would make it hard to decide which seller’s customer actually received the seats first.” [ID431]

Identifying the main synchronization problem. The degree to which study participants were able to identify the problem varied, with most giving a fairly standard “Sellers could mark the seats unavailable at the same time.” [ID412] or “There could be double-booking.” [ID106] Saying this scenario is unlikely was not uncommon, with some participants identifying computers or technology as the source of the problem. For example, one said, “One computer may be operating slower than another, causing the seats one seller saw to be taken by another seller.” [ID406]

Others might not have addressed technology specifically but did identify the key concept of time, with one saying, “One major issue is when, and

how long, it takes to mark a seat as unavailable.” [ID410]

A few others gave more detailed problem descriptions that hinted at the kind of analysis that will eventually be needed to construct a genuinely effective solution, including recognition of the interleaving problem. For example, one said, “The first, most obvious problem is overlap. If all sellers are working at the same time, then the system might display to Seller A that certain seats are open, when, in fact, they have already been reserved by Seller B. Thus A will have to find different seats, which might have, in the intervening time, been reserved by Seller C.” [ID 417]

Identifying other problems. As noted when we discussed algorithmic goals, study participants identified group sales as a particularly knotty problem, and payment and how to cancel orders were big issues. For example, one said, “Another problem would be if the seats are marked unavailable before they are sold, customers might change their minds before payment and possibly hinder the sale of the seats to other customers who might have wanted them at the same time.” [ID420]

Another said, “If a customer cancels an order, how is the information transmitted to the other seller in a reasonable amount of time? If the customer doesn’t pay for the tickets at will call, what happens to them?” [ID 422]

And another said, “At the moment of receiving the payments for the tickets, problems might come up, such as miscommunication between the sellers and charging the customer double.” [ID419]

Participants also mentioned reliability, with one saying, “The computers may malfunction, and the seller may not be able to key in the requested seats.” [ID406]

Centralized solutions. The three variants of the centralized solutions reflected significant distinguishing characteristics. For example, C1 solutions relied on implicit communication between sellers and a central system making the reservation or selection on their behalf. A common characteristic of these implicit communications, as exemplified by ID438

cited earlier, concerning automatic customer ordering, was that they be “fast.” Some were less specific but still involved implicit communication. For example, one participant said, “The program would have to temporarily mark seats being looked at during a transaction as unavailable so vendors couldn’t sell seats simultaneously.” [ID313]

Others were more specific, imposing additional restrictions while prompting doubt as to the participant’s true understanding of concurrency. In one case, we saw evidence of an attempt to move the potential point of concurrent access in an expressed solution. “The only real way to avoid this and still have multiple sellers is to run the booking on a computer network with a master list of available seats. The process would then go something like this: A caller requests n seats. The master list can be ordered in such a way that it fills the seats front to back, left to right; when a seller requests n seats, it gives the next n seats on the list. Cancelled seat orders are inserted at the top of the available list, in order of precedence. The seller can reserve the seats, ask if they are acceptable to the customer, and, if they are, proceed with the transaction. This would avoid double-booking, because during the time the seller offers the seats to the customer, they are withheld from the list, and the other sellers drawing from the list would not have access to those seats.” [ID130]

C2 solutions differed from C1 in their use of explicit communication with a centralized resource making seat assignments, sometimes identified as a database. The quote from ID440 earlier on explicit ordering is an example of this explicit communication.

Another variant of explicit communication involved a particular ordering required to ensure a safe process, including lockstepping and pipelining. For example, one participant said, “These problems [multiple bookings] might be avoided if instead of multiple people selling tickets and being involved in every step of the process, the selling process was divided between two employees. This way, while the second employee was


taking care of the payment of the first caller, the first employee could start to deal with the next sale, then transfer the call to the payment employee.” [ID120]

Another variant does not require a computer solution for concurrency at all, with one participant saying, “A possible solution to this problem [multiple sales of the same ticket] would involve a stagger-start approach when more than one worker is on the phone. For example, when the first caller calls, worker A picks up the phone right away. The second caller calls right after the first has called. Worker B then waits until the phone rings three times, then picks up and starts the process.” [ID121]


C3 solutions (the most common) involved distributing resources in a way that avoids simultaneous access. The most common resource was the seats to be sold. Some participants commented on potential problems with this approach, with one saying, “Perhaps if each vendor were responsible for a section of the concert hall, finding the best seats within their section would solve this problem. But this solution also means some vendors will fill the ‘good’ seats in their section faster, and certain customers won’t get the absolute best seats they could. Chances are good, however, that customers wouldn’t be aware that there are better seats available, rationalizing that the concert filled up quickly.” [ID303]

Other participants were more specific about the technique they would use—assigning a seller to a particular type of seat—to distribute resources. One said this could simplify or un-complicate things, pointing out a possible benefit: “One way we could fix this problem [of multiple sellers of the same ticket] would be to assign a section of seats to each seller. This way no seats would be sold twice, and it would be more organized. One seller would be in charge of one price and one section, making the selling of seats faster and more efficient.” [ID404]

Decentralized solutions. Decentralized solutions are distinguished from centralized solutions by whether sellers themselves make decisions and seat assignments are based on com-



Our motivation was the constructivist theory about how people learn, starting with what they already know and building knowledge on that foundation, rather than receiving it passively from an instructor.



munications with other sellers. If they are, the solution is decentralized. A common decentralized answer could reference a shared resource (such as a database or document), but sellers make decisions individually based on the resource, rather than on a centralized entity.

One participant said, “To resolve this issue [of multiple sellers of the same seats], there should be communication between the sellers. Ideally, they would mark the seats as unavailable on the same documents, so there could never be any doubling.” [ID101]

Other examples of nonspecific communication among distributed sellers included participants saying, “drawing off of the same information that was updated with each transaction” [ID304], “inform the other sellers of this by some form of communication” [ID437], and “using a program that is constantly updated” [ID434].

Speed was a common theme in the proposed solutions, with words like “instantaneously” [ID425], “instantly” [ID426] [ID402], “constantly” [ID434], “continuously” [ID410], and “real time” [ID417].

Some responses were more specific about how communication must occur, with some realizing the problem might not be completely solved. For example, one participant said, “A much easier way would be to use a computer program that networks each seller. This way, each seller has access to each available seat. As soon as a booking is made, it will automatically register on every seller’s screen, and the chance of there being a double booking will be close to impossible.” [ID323]

One participant provided explicit communication directives, saying, “I would change the order of operations so the two or more people booking seats would be required to check with each other while booking so as not to book the same seats, in that way adding another step and alleviating the two problems [booking the same seats and selling more tickets than are available].” [ID409]


Noncomputing-oriented solutions. Several solutions were distinguished by their noncomputing and nontechnological approaches, though they could be classified as either central-

ized or decentralized. For example, one participant said, “We could mark the same seat map with different colored markers for each sell.” [ID106] This participant’s solution is decentralized, since one can imagine individual sellers, each with their own colored marker, racing to mark off seats on a large map.


Another said, “This problem [of selling the same seat twice] could be avoided by allowing only one vendor at a time into the concert hall. But this would be unreasonable if the hall were too large or too many vendors were working to reserve seats. Perhaps if each vendor were responsible for a section of the hall, finding the best seats within their section, this problem would be solved.” [ID303] This response provides two solutions that eliminate potential conflicts, the first by enforcing exclusive access between the selection and the marking of seats, the other by dividing the resource (seats). Here, we could imagine sellers with cellphones dashing around the hall, placing markers on actual, physical seats. Note that scaling issues are mentioned in the proposed solution.

Common errors. The two most common errors in the student-proposed solutions were in thinking that a problem could be solved with a faster system and in devising solutions that simply moved concurrency to another point in the algorithm. As noted when we described decentralized solutions, the surveyed students suggested speed was necessary to avoid many problems. For example, one said, “To avoid this problem we could have very high ‘refresh’ rates or have a way of reserving n seats, as the process is still going through.” [ID 423] Another said, “As soon as n seats are marked unavailable, even before payment processing, the seats need to be marked unavailable. This way, another seller cannot try to reserve a seat that is already ‘reserved.’ ...the system (and screen) would need to be refreshed every time a reservation is made.” [ID 425]

Many proposed solutions moved the point of concurrency. In one, it was moved to a preview step, with the participant saying, “One more solution would be to have the com-



Of the 66 students, 64 identified our main problem of interest—that it may be possible to sell a given seat to more than one person—good evidence that even novice computing students are able to identify this critical concurrency issue.



puter show the n seats as unavailable as soon as any seller has them up on their screens. With this system, only one seller could see these seats as available at a time. If one seller (A) pulls up n seats for a customer, then another seller (B) searches for the best seats, and the seats seller A was looking at would not be shown to seller B.” [ID122]

Another retargeting of the point of concurrency was to a graphical interface, with another saying, “Creating a visual representation of the concert hall through a computer would alleviate this problem. Sellers would mark a certain number of seats for their clients, letting other sellers know which seats are purchased (potentially) and which seats are free for booking.” [ID413]

Another realized his/her distributed, graphical interface only hid the concurrency problem, so proposed a novel solution that apparently used the inherent randomness of human interaction to deal with the problem of multiple sellers claiming the same seats at the same time, saying, “Sellers would each have their own computers, and all of them would be connected, so once a seat is claimed, all other sellers would see it. If two sellers happen to click at the same time, a separate window opens, and both will have to try again.” [ID402]

One interpretation of this solution is that a separate window would open when the computer detects a conflict and forces the sellers to back off and retry, assuming it unlikely that the sellers would try again at precisely the same moment. However, this idea leaves many concurrency issues unresolved, including how the conflict is detected and whether or not other sellers could still get in and reserve the seat(s) targeted by the two original sellers.

Discussion

The proposed solutions of beginner students provide evidence of CS problem-solving skills through their ability to identify the problem and suggest reasonable, though relatively unsophisticated, solutions. By replicating the Ben-David Kolikant study, our study provided additional perspective in both analysis and results.

In addition to finding that her categorizations of student-proposed solutions and goals can be used to analyze beginner responses, our study helped compare beginners and students with significant CS skills but no significant experience with concurrency. This comparison gave a sense of how much sophistication was gained by the more experienced students and how much problem-solving sensibility was already available to students on the day they enter their first CS course.

We focused on two themes—reasonableness and synchronization goals—in the Ben-David Kolikant study:

Reasonableness. Using student interviews, Ben-David Kolikant found that students sometimes solved a simpler problem than the one she assigned, referring to these solutions as not fulfilling the goal of being “reasonable” solutions because they assumed something unrealistic, given the problem description. We saw the same phenomenon in our student solutions. Their wording sometimes made it clear the proposed C2 (constant rate or ordering of operations) and C3 (division of resources) solutions were “easy” or “simple.”

Some students in our study showed their ability to reason about the quality of their solutions. For example ID303, quoted earlier in the context of centralized solutions, recognized the limitations of his/her solution, without suggesting a better one. ID122 noted that moving the point of concurrency in his/her solution as superior to flawed, possibly incorrect alternatives, saying, “The obvious solution would be to fire one seller and have just one working at a time (only kidding). But the thing to do would be to ‘assign’ each seller a section of the hall where the concert is taking place. One seller would have control of half the seats, and the other the other half. There would be no conflicting seats. Or they could just switch systems to general admission. One more solution would be to have the computer show n seats as unavailable as soon as any sellers pull them up on their screens—a first-come-first-served system.” [ID 122]

Synchronization goals. A third of the solutions in the Ben-David Ko-

likant study were centralized. Our study involved an even larger percentage of centralized solutions (55%, or 36). This was consistent with Resnick,¹² whose studies in the mid-1990s found that managing a centralized solution to a problem is easier than managing entities in a decentralized way. Moreover, despite the proliferation of decentralized entities, particularly the Internet, in the 15 years since Resnick’s studies, the students in our study were still more inclined to pursue a centralized solution. Ben-David Kolikant noted that Resnick believed increased exposure to decentralized entities would increase the likelihood of using decentralization, but we found no evidence this happened. (All participants in our study were experienced Internet users.)

The larger percentage of centralized solutions in our study stemmed from two factors: First, 82% of our centralized solutions were C2 or C3 solutions; only 33% of Ben-David Kolikant’s centralized solutions were C2 or C3 solutions. As Ben-David Kolikant brought up in her discussion of reasonableness, students often made simplifying assumptions, not because they are reasonable, but because they allow them to more easily solve the problem posed to them. Given our more open-ended problem description, it seemed more “reasonable” to students to suggest C2 or C3 solutions to our problem about concurrency in concert-ticket sales. And second, given our less well-defined problem statement and reduced direct engagement with the ticket-sales problem in the classroom setting, our students were not necessarily prodded to consider or outline a more complex decentralized solution the same way Ben-David Kolikant’s students might have been.

Consistent with Ben-David Kolikant’s study, we found students concentrated on sharing information across sellers rather than preventing interleaving access to the database, with only six discussing the interleaving of operations. However, it may well be that the nature of our task was simply not as suggestive of database issues as the pseudocode in Ben-David Kolikant’s problem statement, particularly given the lack of comput-

ing experience by our CS1 students.

While Ben-David Kolikant was able to show students “exaggerate the grain of an atomic action” by assuming that checking and updating the database is atomic, we found that a description of even this level of granularity was present in only the most explicit and detailed of our student-generated solutions. Many of the responses did not clarify this level of description of granularity of interaction, leaving it ambiguous as to how well they really understood the concurrency issue at hand.

Algorithmic goal of solution. While Ben-David Kolikant’s study posed a well-constrained problem with a clear algorithmic goal, our study included an unconstrained problem description, and our student participants provided a number of goals for their algorithms, with 97% identifying the main goal of not letting the same seat be sold to two different customers, indicating a commonsense ability to identify concurrency conflicts.

Many of our students (41%, or 27) identified goals beyond not selling the same seat, including group sales of a large block of tickets, letting seats be reserved without being paid for, identity theft, choosing seats by price rather than by best available, payment-transaction delays, and data tracking and storage. Two of these problems—group ticket sales and reserving without selling to allow seats to be made available again—are notable for how they influenced approaches to achieving the main algorithmic goal. From an instructional point of view, these additional goals noted by students suggest they may need help prioritizing from among the goals they find in open-ended questions, unless the goals are given explicitly.

Concurrency techniques. Most students introduced techniques directly related to concurrency, with solutions including centralized techniques and distributed techniques. Within them, they introduced scaling, locks, pipelining, and methods for distributing resources. Even in proposed solutions with errors, we found concepts ripe for leverage. For example, many of our students chose to “pass the buck” by pushing concurrency from buying a seat to reserving it. Others

gave solutions that assumed a system fast enough to eliminate race conditions. Though incorrect, they provide a starting point for understanding atomic operations and the interleaving of instructions.

Conclusion

In both their correct and incorrect preconceptions, the 66 students in our 2006 study apparently began their first computing course with essentially the same level of intuition as they began their first course involving concurrency. This similarity suggests students do not gain a deeper understanding of concurrency as they advance through the curriculum. As we have no data indicating that taking *nonconcurrency* courses provides skills that help one learn concurrency-related material more quickly, it may be there is no advantage to delaying the introduction of concurrency. Given the prevalence of concurrency and its increasing relevance at all levels and applications of CS, we suggest it may be wise to include concurrency earlier in the curriculum.

No matter which course introduces concurrency, the problem and student-proposed solutions in our study suggest ways to leverage student preconceptions. For example, instructors could conduct an exercise like this and choose student-generated solutions for further discussion to explicitly address common errors. In particular, they could:

- ▶ Emphasize the real-world nature of the problem, pointing out related concurrency problems;
- ▶ Demonstrate that race conditions come up even in “fast” systems;
- ▶ Use responses that pass the buck (appearing to solve the concurrency problem by moving it to another operation without actually solving it) to help discuss the notion of atomic operations;
- ▶ Use a centralized solution to discuss interleaving instructions and pipelining technique; and
- ▶ Use responses that do not scale well to discuss scaling.

Here, we’ve addressed a rich set of student responses that represent a starting point for asking students to critique proposed solutions, emphasizing concepts we know are at the

edge of their preconceptions. All such concepts could be discussed early or at least introduced in CS1.

Instructors can use the question we asked—What’s the best way to organize the sale of tickets to a popular concert?—to help identify student preconceptions about CS. By documenting beginner-student preconceptions, instructors gain leverage for using a constructivist model, building on this commonsense knowledge through student preconceptions. Our study did not depend on a particular technology, pedagogy, or philosophy and can be replicated to study how or if students’ commonsense knowledge is changing.

Acknowledgments

This material is based in part on work supported by the National Science Foundation under grants DUE-0736343, DUE-0736700, DUE-0736572, DUE-0736738, DUE-0736859, and DUE-0736958. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the NSF. We thank the many students who responded to the questions used in this and our other studies; Sally Fincher, Josh Tenenber, and the NSF (through grant DUE-0243242) who provided us with workspace at the SIGCSE conferences in 2006 and 2007; Renee McCauley, Sara Miner More, and Suzanne Westbrook for help with fall 2006 data collection and others who collected data for us in other commonsense projects; and the anonymous reviewers for their comments and suggestions. An earlier description of the study, with expanded analysis and discussion, is in the *Proceedings of the Third International Computer Science Education Research Workshop*.⁹

References

1. Ben-Ari, M. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching* 20, 1 (Mar. 2001), 45–73.
2. Ben-David Kolikant, Y. Gardeners and cinema tickets: High schools’ preconceptions of concurrency. *Computer Science Education* 11, 3 (Sept. 2001), 221–245.
3. Bonar, J. and Soloway, E. Preprogramming knowledge: A major source of misconceptions in novice programmers. In *Studying the Novice Programmer*, E. Soloway and J. Spohrer, Eds. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989, 325–354.
4. Bransford, J.D., Brown, A.L., and Cocking, R.R., Eds. *How People Learn: Brain, Mind, Experience, and School*.

- National Academy Press, Washington, D.C., expanded edition, 2000.
5. Bruce, K.B. and Danyluk, A. Event-driven programming facilitates learning standard programming concepts. In *Companion to the 19th annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (Vancouver, BC, Oct. 24–28) ACM Press, New York, 2004, 96–100.
6. Clancy, M. Misconceptions and attitudes that interfere with learning to program. In *Computer Science Education Research*, S. Fincher and M. Petre, Eds. Taylor and Francis Group, London, 2004, 85–100.
7. Committee on Undergraduate Science Education. *Science Teaching Reconsidered: A Handbook*. National Academy Press, Washington, D.C., 1997.
8. Gibson, J.P. and O’Kelly, J. Software engineering as a model of understanding for learning and problem solving. In *Proceedings of the First International Workshop on Computing Education Research* (Seattle, Oct. 1–2), ACM Press, New York, 2005, 87–97.
9. Lewandowski, G., Bouvier, D., McCartney, R., Sanders, K., and Simon, B. Commonsense computing (Episode 3): Concurrency and concert tickets. In *Proceedings of the Third International Workshop on Computing Education Research* (Atlanta, Sept. 15–16). ACM Press, New York, 2007, 133–144.
10. Miller, L. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal* 20, 2 (June 1981), 184–215.
11. Onorato, L. and Schvaneveldt, R. Programmer/nonprogrammer differences in specifying procedures to people and computers. Chapter 9 in *Empirical Studies of Programmers*, E. Soloway and S. Iyengar, Eds. Ablex Publishing, Norwood, NJ, 1986, 128–137.
12. Resnick, M. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press, Cambridge, MA, 1994.
13. Simon, B., Bouvier, D., Chen, T.-Y., Lewandowski, G., McCartney, R., and Sanders, K. Commonsense computing (Episode 4: Debugging). *Computer Science Education* 18, 2 (June 2008), 117–133.
14. Simon, B., Chen, T.-Y., Lewandowski, G., McCartney, R., and Sanders, K. Commonsense computing: What students know before we teach (Episode 1: Sorting). In *Proceedings of the Second International Workshop on Computing Education Research* (Canterbury, U.K., Sept. 9–10), ACM Press, New York, 2006, 29–40.
15. Smith, J., diSessa, A., and Roschelle, J. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *Journal of Learning Sciences* 3, 2 (Apr. 1994), 115–163.
16. Stein, L.A. Interactive programming: Revolutionizing introductory computer science. *ACM Computing Surveys* 28, 4es (Dec. 1996), 103.

Gary Lewandowski (lewandow@cs.xu.edu) is a professor in and chair of the Department of Mathematics and Computer Science at Xavier University, Cincinnati, OH.

Dennis J. Bouvier (djb@acm.org) is an associate professor in the Department of Computer Science at Southern Illinois University, Edwardsville, IL.

Tzu-Yi Chen (tzuyi@cs.pomona.edu) is an associate professor in and chair of the Department of Computer Science at Pomona College, Claremont, CA.

Robert McCartney (robert@cse.uconn.edu) is an associate professor in the Department of Computer Science and Engineering at the University of Connecticut, Storrs, CT.

Kate Sanders (ksanders@ric.edu) is a professor in the Department of Math and Computer Science at Rhode Island College, Providence, RI.

Beth Simon (bsimon@cs.ucsd.edu) is a lecturer with potential for security of employment in the Department of Computer Science and Engineering at the University of California, San Diego.

Tammy VanDeGrift (vandegri@up.edu) is an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Portland, Portland, OR.

Interactive computer graphics would rival word-processing and presentation programs for everyday communications.

BY TAKEO IGARASHI

Computer Graphics for All

COMPUTER GRAPHICS IS a commodity. Sophisticated computer-generated imagery is everywhere—feature films, TV programs, video games, even cellphones—but most of it is created by professionals. Few people actually create computer graphics in their daily lives because most authoring tools are designed for professionals or dedicated amateurs following intensive training. This is unfortunate, because computer graphics could be a powerful communication tool for everyone.

Consider desktop publishing. Centuries ago, only a small number of professionals worked in the printing industry. When computer-based printing emerged as an alternative in the late 20th century, it, too, was initially limited to professionals. However, the widespread use of personal computers and easy-to-use graphical user interfaces quickly made high-quality printing accessible to the general public. Today, just about everyone uses word processors on a daily basis to create documents that communicate ideas to friends and colleagues. Computer graphics has not yet achieved such mass-market appeal.

Unlike with traditional physical me-

dia, consumers today create electronic content to share through the Internet. Most media are still text-based, as with email, blogs, and Twitter, but more and more include images, videos, animations, and other multimedia con-

» key insights

- **Computer-graphics authoring should be accessible to the general public.**
- **Designing these systems starts with what is natural to humans rather than what is natural to a computer.**
- **Most traditional research focuses on experts' high-end use of technology; here, our main target is the casual use of technology by nonprofessionals.**

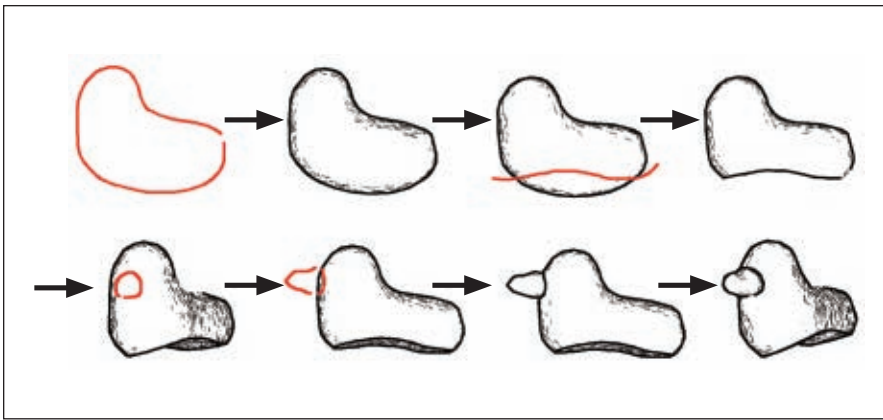


Figure 1. Modeling session in Teddy. Users create 3D models using simple sketching operations.



Figure 2. Screenshot of Teddy and sample 3D models created through the Teddy system.

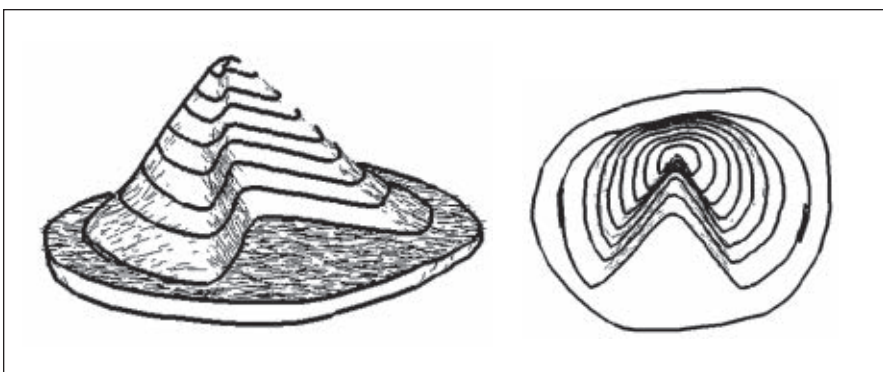


Figure 3. Using Teddy to teach the concept of contour lines.

tent. End users constructing 3D models are also supported by a number of systems, including Google's SketchUp (<http://sketchup.google.com/>) and modeling tools in *SecondLife* and *Spore*. However, these systems use scaled-down versions of traditional interfaces and still require a certain amount of skill. This article introduces research efforts at the University of Tokyo and Brown University to make computer-graphics authoring accessible to more casual users. To achieve this goal, the author and his collaborators developed easy-to-use prototype systems to create expressive computer graphics more quickly than with traditional interfaces. Examples are sketch-based 3D modeling, clothing manipulation, animation by performance, and 2D shape manipulation. We discuss the user interfaces and technical aspects of these prototype systems, as well as the lessons learned from their development, offering ideas for future research directions.

Most of our work is highly interactive and difficult to explain in written words and still images; please see demonstration videos and prototype systems at <http://www-ui.is.s.u-tokyo.ac.jp/~takeo>.

Sketching 3D Models

Creating a 3D model in a computer (not necessarily on a screen) is the first step in most 3D computer-graphics applications yet is also the most difficult. Traditional interfaces for 3D modeling programs trace their origins to traditional pencil-and-paper professional drafting. Users place vertices in 3D space by specifying x -, y -, and z -coordinates in a three-view interface, then create polygonal faces (individual polygonal sides of a polyhedron) by connecting these vertices. Alternatively, users start with a simple primitive (such as a sphere or cylinder) and modify it by editing individual vertices and edges. Many editing tools (such as free-form deformation and Boolean operations among solids) are available for designing complicated shapes from simple primitives. Although they might be appropriate for trained professionals designing precise models, they are generally too difficult for first-time users trying to quickly generate meaningful models.

The sketching interface is emerging as an alternative modeling method. Users draw 2D lines on the screen; the system then generates a 3D model automatically, inferring missing depth information. Sketching interfaces for 3D scenes consisting of simple primitives were first introduced in the SKETCH system,¹⁸ allowing users to perform complicated 3D editing operations in a single camera view by combining heuristics. A similar approach is used in commercial systems (such as Google's SketchUp). However, these systems are designed for sketching simple shapes defined by relatively few parameters. Designing them requires specialized training.

Our group at the University of Tokyo developed the Teddy system¹¹ to address this problem, allowing users to quickly generate interesting 3D freeform models (such as creating a teddy bear by drawing the silhouette of the desired shape) (see Figure 1). The user's strokes are in red; the system infers and draws everything else. The user first draws the silhouette of the base primitive, and the system generates the corresponding 3D geometry. The user then draws a stroke across the model, and the system cuts the model at the line. The user can also add parts to the base model by drawing two strokes; Figure 2 shows several 3D models created this way.

We do not expect Teddy to replace traditional 3D modeling tools. Rather, it will create new 3D modeling applications that are useful to nonexperts, including children, who want to play with 3D graphics for fun. Introduced at the SIGGRAPH conference in 1999, Teddy is used in several current commercial video games to permit players to create their own characters. Using it is a useful way for experts to express their ideas quickly in early design phases. A commercial 3D modeling package, Shade (available only in Japan, <http://shade.e-frontier.co.jp/>), includes an extension to Teddy as a plug-in for generating rough sketches. Finally, and most important, Teddy is useful for communicating 3D concepts face to face. In a classroom, for example, a teacher could quickly draw a model of bacteria, showing its cross section to explain its internal structure. In a hospital, a medical doctor could draw a model of a stomach to help explain a patient's stomach disease.

In 2003, to test the idea, we conducted a trial in a high school geography class in Chiba, Japan. Teaching 3D concepts (such as mountains and valleys), a geography teacher would have difficulty explaining them using traditional 2D media like a blackboard. Sketching in 3D can help address this problem. A convincing example is the teaching of contour lines using the Teddy system (see Figure 3) in which the teacher first shows a 3D model of a mountain, then draws several horizontal lines in the side view, saying the lines indicate equal height intervals. The teacher then changes the viewpoint to show the mountain and the lines from the top. This way, students understand the relationship between the closed lines on the map (contour lines) and the 3D geography, not just mountains, ridges, and valleys.

Clothing Manipulation

3D characters must also be dressed properly. The computer-graphics re-

search community has actively investigated the physical simulation of cloth, today producing realistic cloth simulations. However, the initial simulated-cloth configuration must be set manually, and the user interface for manipulating cloth is primitive. A typical approach is to place rigid cloth patches around the target body, combining 3D translation and rotation before starting the simulation—a tedious process. Moreover, users have difficulty changing the way the garment is worn once they've placed it on a character. Standard systems allow users to freely move individual vertices through direct manipulation, but it causes a large local distortion (stretching), making it difficult to achieve global movement.

In 2001, our group at Brown University developed clothing-manipulation techniques to address these issues.¹⁰ To put a garment on a character, users first draw free-form marks on both the garment and the character to indicate positional correspondence (see Figure 4).

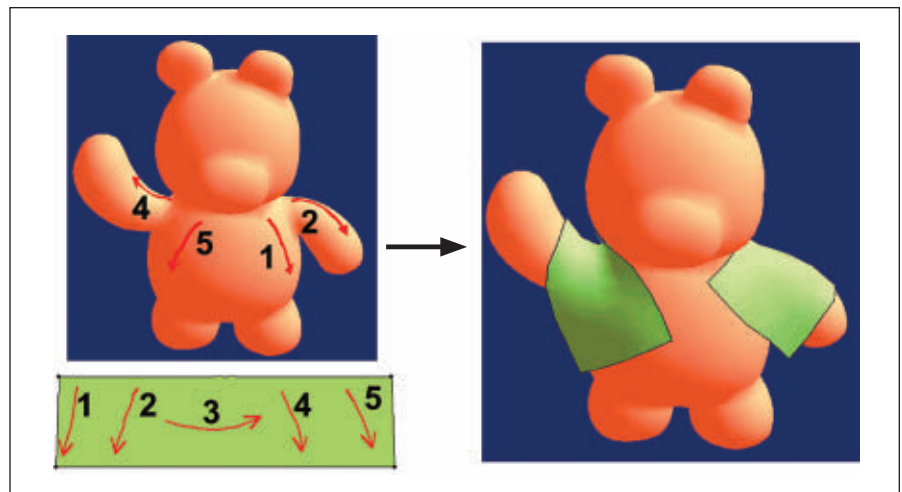


Figure 4. Users draw marks on the character and cloth; the system then places the cloth on the character.

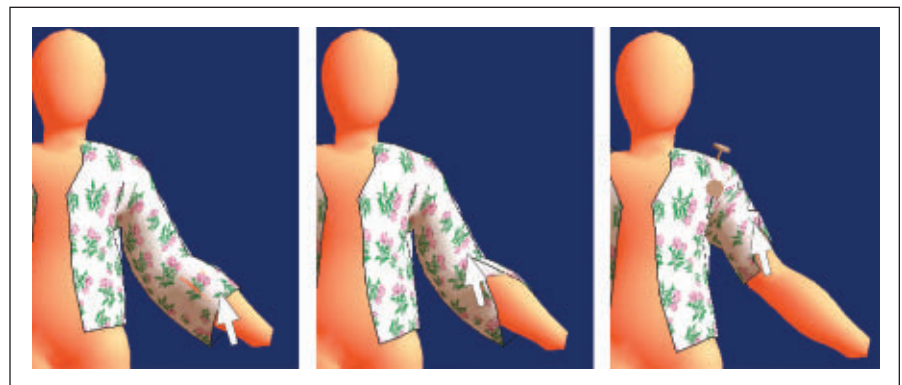


Figure 5. Dragging the cloth onto the character: left, before dragging; center, the result of traditional vertex dragging; right, the result of our clothing-manipulation method.

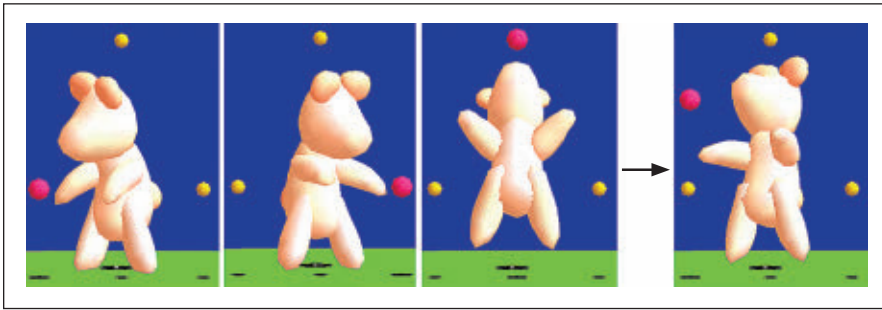


Figure 6. Spatial keyframing. Users specify three key poses (left), then freely control the character by dragging the red ball (right).

The system then places the garment on the character so the marks on the garment match the corresponding marks on the character. The system uses a simple relaxation process during placement to prevent stretching and squashing, even if the lengths of the corresponding marks are different. Working with only a few strokes, users are able to place reasonably complicated garments on any character.

Once a garment is on a character, users can grab any point of the garment and drag it onto its surface (see Figure 5). Unlike standard vertex dragging, in which a single vertex is moved while relying on subsequent simulation to move other vertices, this dragging operation moves all vertices of the cloth mesh directly, causing global movement. To achieve global movement, the movement vector of the dragged vertex is propagated to the complete cloth mesh along the surface of the character.

This technique allows even novice users to quickly test many different ways of dressing virtual characters. We also expect it to be useful for designing real garments as well. The cloth representation and simulation are limited in the prototype system implemented in 2001, but the basic user interface should still be applicable to today’s more sophisticated cloth representation.

The technical contribution of this work is the behavior of the cloth material in response to user input. It not only follows physical principles (such as gravity and collision) but behaves proactively to assist a user’s design process; for example, the cloth automatically unfolds local folds based on the assumption that users do not want to see accidental local folds unless they explicitly require them. Such built-in intelligent behavior of passive materials can be useful in other domains; we

are now testing it in knot- and hairstyle-design systems.

Performance-Driven 3D Animation

Keyframing is the most popular method for designing character animation. The user specifies the pose of the character at each time point, and the system interpolates the key poses at runtime. Though many other methods (such as motion capture and procedural animation) are available, keyframing is by far the most popular approach due to its simplicity and versatility. But manually defining so many keyframes is tedious. Moreover, novice users experience great difficulty designing natural-looking motion through discrete sets of poses. The result tends to look mechanical while lacking the rich textures seen in the motion of living things.

A live demonstration is the simplest approach to designing motion, in which a user moves the target character in real time and the system records the motion, like dancing a teddy bear in front of a video camera. However, moving a character with many joints is difficult when using a standard input device like a mouse. Though possible to demonstrate the motion of each joint one at a time,⁶ synchronizing individual motions is difficult.

Our group at Brown University developed a spatial keyframing method to address this problem.⁹ With it, the user first sets a group of key poses in the 3D or 2D space; a pose is associated with a position in a space. The user then moves the cursor in that space, and the system sets the character pose by blending the key poses around the cursor position (see Figure 6). The user is thus able to design interesting whole-body character motion (such as juggling and dancing) by recording simple cursor movements.

This technique is well suited to defining expressive motion (such as to show joy or sadness). The resulting motion is much more alive than motion generated through traditional keyframing because the motion directly mirrors the operator’s natural hand movement. However, motion dominated by physical factors (such as jumping and running) is better supported by physical approaches.⁷

We expect spatial keyframes to be a useful intermediate representation for 3D characters. Current 3D character representation consists of geometry, texture, rigs, and possibly predefined motions. Users who want to define a new motion must specify individual poses one at a time. Specialized tools include Maya’s set-driven keys (http://caad.arch.ethz.ch/info/maya/manual/UserGuide/Animation/KeyframeMoPath/03_understanding_key.doc5.html) and the Waldo input device (<http://www.character-shop.com/waldo.html>), though neither is designed for the blending of key poses. By providing predefined spatial keyframes (a set of natural poses) for a character, users can create new motion very quickly by moving the control cursor. This can make it much easier for inexperienced users to make characters move at will.

2D Shape Manipulation

In the physical world, one can hold an object (such as a teddy bear) with two hands and freely manipulate it through rotating, stretching, squashing, and bending motions. Standard 2D drawing programs provide poor support for such shape manipulation, allowing only simple editing operations (such as scaling and rotation). Not only do these operations require a complicated combination of tools, the result for the user simply doesn’t feel like manipulating a physical entity.

In 2004, our group at the University of Tokyo developed a novel manipulation technique to address this problem.⁸ Users are thus able to select arbitrary points as handles on a 2D shape, then freely manipulate the shape by moving the handles (see Figure 7). They can relocate the shape by setting a single handle to rotate, stretch, and squash the shape. Users also swing a head or stretch an arm by setting handles on the corresponding positions. The

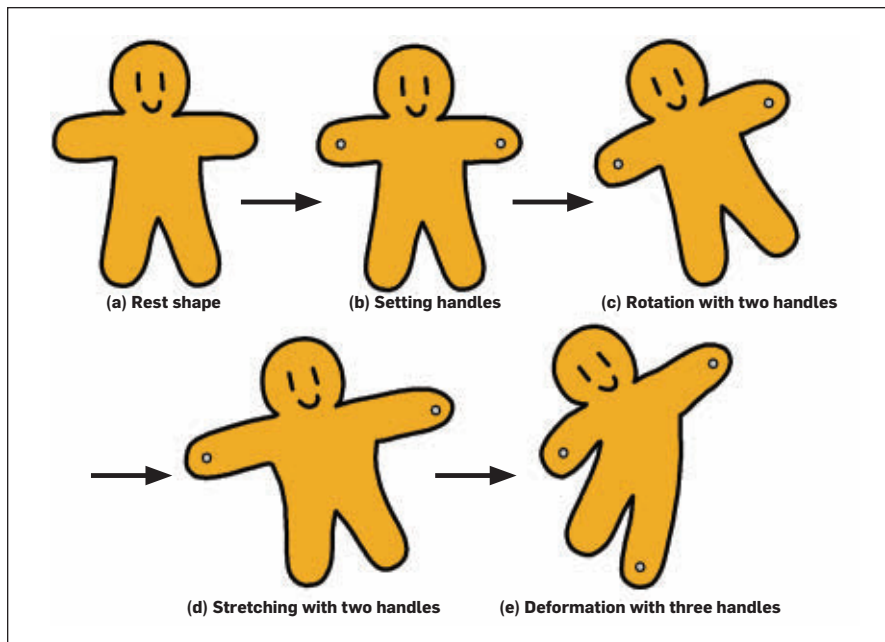


Figure 7. As-rigid-as-possible shape manipulation. Users place handles on the drawing, then manipulate it by moving the handles.

shape deforms naturally in response to user input; for users it feels like they're manipulating a physical object.

Traditional computer-based methods for shape manipulation are roughly divided into three categories:

*Skeleton.*¹³ The user embeds a skeletal structure inside the shape and controls it to deform the shape. However, embedding a skeleton in each shape is tedious, and the approach does not work for stretching and squashing;

*Spatial deformation.*¹⁴ The user defines a spatial mapping using several control points, then deforms the shape according to the mapping function. However, mapping functions do not consider the rigidity of the shape and result in unnatural deformation; and

Physics-based. This approach simulates the deformation process of physical material.¹² However, the computation is not fast or stable enough to provide real-time feedback to a global deformation caused by user operations.

Our method takes a completely geometric approach, defining an energy function that measures the amount of geometric deformation, then minimizes it using an optimization method. We designed the energy to give a closed-form solution to the problem. In it, the system obtains the deformation by solving two large sparse linear-matrix equations in sequence, a very fast and perfectly stable approach.

It is also particularly useful for creating 2D animations. Traditional animation artists assemble many slightly different drawings to create an animation. In our shape-manipulation system MovingSketch (<http://www-ui.is.s.u-tokyo.ac.jp/~takeo/research/rigid/movingsketch/index.html>), users create an interesting animation by drawing a character and recording the manipulation process. Using a multi-touch input device,¹⁶ they grab a character with both hands and manipulate it to create an animation (see Figure 8). We tested the

system with children in an educational TV program in Japan and found that even elementary-school students could quickly generate reasonably interesting animations.

Lessons Learned

Each of these projects addresses a specific problem, with technical contributions being rather independent. However, emerging from them are common guidelines for designing a compelling user experience:

Natural to humans. First, start with what is natural to a human rather than with what is natural to a computer. The computer represents a 3D model with a collection of 3D points and their connections; traditional computer-aided-design systems ask users to provide this information directly. Advanced systems represent a model with a sequence of editing operations, but most of them still require that users be aware of points and faces. Similarly, a computer represents a 2D drawing with its position and orientation. Traditional drawing systems ask users to directly control these parameters; that is, traditional systems expose the underlying representation to the user directly. Though it is the most straightforward way to implement a system, the result means difficulty for novice users. That's why we start by identifying the most natural operations for a human referring to real-world examples, then developing an algorithm that maps

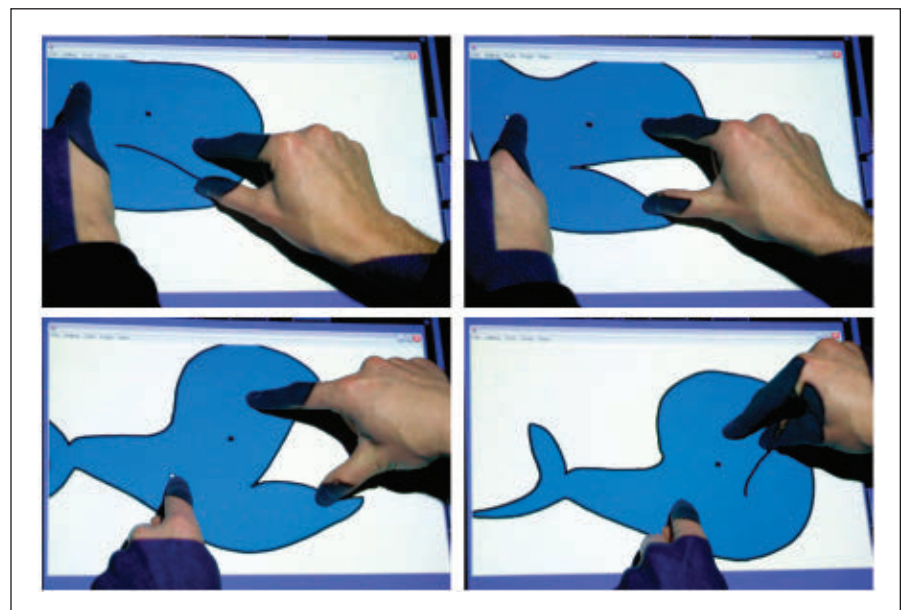


Figure 8. Bimanual manipulation of a drawing.

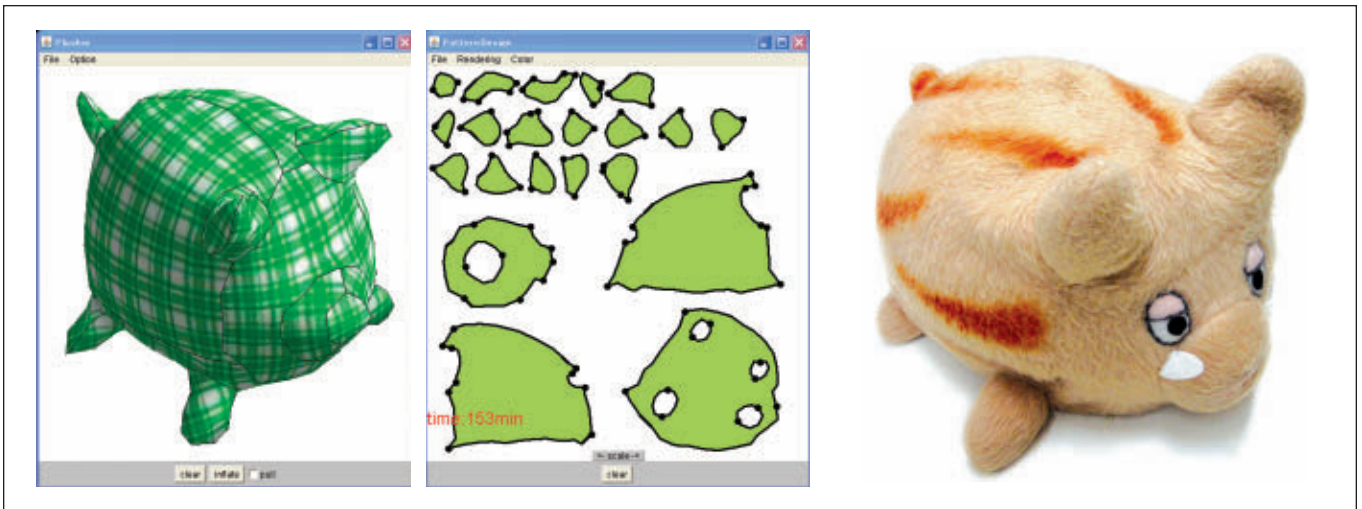


Figure 9. Screenshot of the Plushie system and plush toy designed with the system.

them to computer representations. In the case of 3D modeling, we learned from sketching activity on real paper. For 2D animation, we learned from children playing with a real plush toy using both hands.

Instant feedback. Instant feedback is critical to real-time interaction. It allows for graceful learning through casual trial and error while supporting creative exploration through rapid experimentation. To provide a rich and comfortable user experience, three opportunities for executing computation should be used: One is computation during mouse dragging, a computation that must be very fast (on the order of 0.1 seconds). The second is computation right after a mouse click or dragging; it can be somewhat slower (about a full second). And the third is running a computation in the background while the user is looking at a result. The sketching interface is effective because it gives a system the opportunity to execute a heavyweight computation right after the completion of a sketch (mouse release) that would otherwise be too time-consuming during a mouse drag. In 2D animation, the system computes time-consuming matrix factorization when a pin is added or removed, applying fast back-substitution during dragging. The clothing-manipulation system exploits idle time to refine the cloth configuration.

Right target task. System designers must choose and focus on the right target task to achieve the first two goals. Developers try to address a range of tasks, overloading the interface with too many functions, as in professional

systems like Maya. In theory, including more functions could expand the range of user options but also require intensive training and reduce what casual users are able to do in the system. Carefully limiting functional scope, designers provide an optimized interface and algorithm for the task in exchange for losing some rarely used functions. Teddy is designed for rotund models (such as stuffed animals), freeing users from having to specify depth information each time. The clothing-manipulation system simplifies the interface and accelerates the computation by focusing on the cloth on the surface of the body. System designers are better off tapping user creativity than constraining it with many predefined functions. A simple, well-designed interface allows users to apply their imaginations to complete tasks beyond the system designer's original assumptions, as in terrain sketching with Teddy.

We would also like to share some general lessons learned after the original publication of these research results in 1999.¹¹ First, even though a sketching interface does lower the threshold, 3D modeling remains difficult. The main difficulty is control of 3D rotation with a 2D input device. We observed that many test users failed to rotate a model to the desired orientation. It is therefore desirable to give users rotation-free modeling methods or a significantly easier rotation interface. Second, though we tried to extend these techniques to support more advanced modeling operations, we were unsuccessful. The more operations we

tried to support, the more complicated the interface became, and the original advantages disappeared. We therefore explored new application domains instead of focusing on the same problem. Finally, the tools we've outlined here were generally better received by users with no prior experience in 3D modeling or animation. Users who had previously worked with computer graphics had their own preferred tools and did not show much interest in Teddy. Those without prior experience saw great potential. We encourage researchers working on similar problems to not be intimidated by negative reactions from existing users but to try finding new users outside the existing user community. This will ultimately expand application of computer graphics.

Future Directions

In addition to improving the tools discussed here, we plan to work on other aspects of computer-graphics authoring in the future, including two notable problems:

Designing interactive behaviors. Interactivity is an important aspect of computer-generated media. Not only do users passively watch predefined imagery, they also interact with computer imagery (such as by poking a character) to observe its response. The systems we've introduced here are all interactive as authoring tools, but the content they produce is noninteractive; 3D models and 2D animation created this way do not respond to user input. End-user design of interactive behavior is an exciting but challenging research direction.

Several earlier research efforts sought to achieve end-user design of interactive behavior. One involved making traditional programming (scripting) accessible to casual users through a highly visual editing environment.³ In the system, users write a program using simple drag-and-drop operations without making syntax errors. Another involved using programming by demonstration⁵ for character animation¹⁷; in it, users demonstrate the desired interactive behavior of a character, with the system learning the pattern from the demonstration. Programming with visual replacement rules is a promising approach for defining a character's interactive behavior.⁴ The user specifies before-and-after pairs; at runtime, the system compares the scene configuration with the *before* patterns, replacing it with *after* patterns when the match is identified.

Though these experiments produced interesting initial results, designing the arbitrary interactive behavior of a virtual agent is often prohibitively difficult. We are particularly interested in teaching interactive behavior to physical agents (robots). End-user programming for robot behavior has been tested in some systems¹ but is still limited to basic motions. Programming by demonstration for robots has also been reported but is used mainly for acquiring physical-manipulation skills.² Techniques developed in the user-interface-research community that should be applicable to human-robot interaction represent an interesting research direction.

Designing real-world objects. The systems outlined here were all designed for virtual representations; one can produce interesting graphics on the computer screen but cannot touch or use them in the real world. Then there's development of end-user tools for designing physical objects (such as furniture and clothing). The idea is to help people custom-design the things they will use instead of having to buy manufactured products in stores. Objects designed by users themselves should satisfy their needs more directly and produce greater satisfaction.

Unlike professional designers, the typical consumer generally lacks the professional knowledge needed to de-

sign physical objects. Inexperienced consumers could easily create a bag that is not sturdy enough or a chair that cannot stand up. One promising approach is to introduce physics into the modeling process. Traditional modeling systems ignore physics, possibly producing physically inappropriate results, as in, say, objects that penetrate one another. It might be possible to help users avoid these issues by considering physical principles within a modeling system.

In 2006, our first such experiment involved a design system for plush toys.¹⁵ Users would interactively draw a sketch on the screen, and the system would then automatically generate a 3D plush toy model, as in the Teddy system. In addition, the system simultaneously generated a 2D cloth pattern corresponding to the 3D geometry, allowing the user to create a physical plush toy by cutting the cloth according to the generated pattern (see Figure 9). Internally, the system first generated a 2D cloth pattern, then ran a simple physical simulation to predict the 3D shape of the resulting toy. This way, even young children would be able to design their own plush toys just by sketching.

The idea of 3D modeling with physical simulation is very powerful. We expect future modeling systems to consider various physical constraints in the background (such as collisions and stability), freeing users from low-level physical concerns and allowing them to concentrate on more important high-level design concerns. We plan to test this idea in a number of target domains, including furniture and clothing design.

Conclusion

This article introduced our efforts to make computer-graphics authoring accessible to the general public, making it as much a daily communication tool as word processing and presentation applications. What most defines our research is its focus on end users. This opens up new application possibilities for existing technologies while posing unique technological challenges for interface researchers and developers. We look forward to more computer-science researchers participating in this fertile field.

Acknowledgments

I would like to thank Satoshi Matsuo-ka, Hidehiko Tanaka, John F. Hughes, Tomer Moscovich, Yuki Igarashi, Maneesh Agrawala, and Masahiko Inami for their contributions and comments. ■

References

- Baum, D. and Zurche, R. *Definitive Guide to Lego Mindstorms*. Apress, New York, 2000.
- Billard, A., Calinon, S., Dillmann, R., and Schaal, S. Robot programming by demonstration. In *Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, New York, 2008, 1371–1394.
- Cooper, S., Dann, W., and Pausch, R. Teaching objects first in introductory computer science. In *Proceedings of the ACM Technical Symposium on Computer Science Education* (Reno, NV, Feb. 19–22). ACM Press, New York, 2003, 191–195.
- Cypher, A. and Smith, D.C. KidSim: End-user programming of simulations. In *Proceedings of the ACM Conference on Computer Human Interaction* (Denver, May 7–11). ACM Press, New York, 1995, 27–34.
- Cypher, A. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, 1993.
- Dontcheva, M., Yngve, G., and Popović, Z. Layered acting for character animation. In *Proceedings of ACM SIGGRAPH* (San Diego, CA, July 27–31). ACM Press, New York, 2003, 409–416.
- Fang, A.C. and Pollard, N.S. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics* 22, 3 (July 2003), 417–426.
- Igarashi, T., Moscovich, T., and Hughes, J.F. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics* 24, 3 (July 2005), 1134–1141.
- Igarashi, T., Moscovich, T., and Hughes, J.F. Spatial keyframing for performance-driven animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, July 29–31). ACM Press, New York, 2005, 107–115.
- Igarashi, T. and Hughes, J.F. Clothing manipulation. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (Paris, Oct. 27–30). ACM Press, New York, 2002, 91–100.
- Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: A sketching interface for 3D freeform design. In *Proceedings of ACM SIGGRAPH* (Los Angeles, Aug. 8–13). ACM Press, New York, 1999, 409–416.
- James, D.L. and Pai, D.K. ArtDefo: Accurate real-time deformable objects. In *Proceedings of ACM SIGGRAPH* (Los Angeles, Aug. 8–13). ACM Press, New York, 1999, 65–72.
- Lewis, J.P., Corder, M., and Fong, N. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH* (New Orleans, July 23–28). ACM Press, New York, 2000, 165–172.
- MacCracken, R. and Joy, K.I. Freeform deformations with lattices of arbitrary topology. In *Proceedings of ACM SIGGRAPH* (New Orleans, Aug. 4–9). ACM Press, New York, 1996, 181–188.
- Mori, Y. and Igarashi, T. Plushie: An interactive design system for plush toys. *ACM Transactions on Graphics* 26, 3 (July 2007).
- Rekimoto, J. SmartSkin: An infrastructure for freehand manipulations on interactive surfaces. In *Proceedings of ACM Conference on Human Computer Interaction* (Minneapolis, Apr. 20–25). ACM Press, New York, 2002, 113–120.
- Young, J.E., Igarashi, T., and Sharlin, E. Puppet Master: Designing reactive character behavior by demonstration. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation* (Dublin, July 7–9). ACM Press, New York, 2008, 183–191.
- Zelevnik, R.C., Herndon, K.P., and Hughes, J.F. SKETCH: An interface for sketching 3D scenes. In *Proceedings of ACM SIGGRAPH* (New Orleans, Aug. 4–9). ACM Press, New York, 1996, 163–170.

Takeo Igarashi (takeo@acm.org) is an associate professor in the Department of Computer Science in the Graduate School of Information Science and Technology at The University of Tokyo.

A new era of theoretical computer science addresses fundamental problems about auctions, networks, and human behavior.

BY TIM ROUGHGARDEN

Algorithmic Game Theory

THE WIDESPREAD ADOPTION of the Internet and the emergence of the Web changed society's relationship with computers. The primary role of a computer evolved from a stand-alone, well-understood machine for executing software to a conduit for global communication, content-dissemination, and commerce. The algorithms and complexity theory community has responded to these changes by formulating novel problems, goals, and design and analysis techniques relevant for modern applications.

Game theory, which has studied deeply the interaction between competing or cooperating individuals, plays a central role in these new developments. Research on the interface of theoretical computer science and game theory—an area now known as *algorithmic game theory* (AGT)—has exploded over the past 10 years. The primary research themes in AGT differ from those in classical microeconomics and game theory in important, albeit predictable, respects. Firstly in application areas: Internet-like networks and nontraditional auctions motivate much of the work in AGT. Secondly in its quantitative engineering approach: AGT research typically models applications

via concrete optimization problems and seeks optimal solutions, impossibility results, upper and lower bounds on feasible approximation guarantees, and so on. Finally, AGT usually adopts reasonable (for example, polynomial-time) computational complexity as a binding constraint on the feasible behavior of system designers and participants. These themes, which have played only a peripheral role in traditional game theory, give AGT its distinct character and relevance.

Here, we touch on the current dominant research trends in AGT, loosely following the organization of the first book in the field.³⁰ We focus on contributions of the algorithms and complexity theory community; see two recent articles in *Communications*^{18,40} and the references therein for alternative perspectives on computer science and game theory.

Algorithmic Mechanism Design

Algorithmic mechanism design studies optimization problems where the underlying data—such as the values of goods and costs of performing a task—is initially unknown to the algorithm designer, and must be implicitly or ex-

>> key insights

- Many modern computer science applications involve autonomous decision-makers with conflicting objectives. Current research in algorithms and complexity theory uses game theory as an important tool for modeling and reasoning about such applications.
- One application domain is auctions, including the single-item auctions of eBay and Amazon; the sponsored search auctions of Google, Yahoo!, and Microsoft; and the combinatorial auctions used by governments to sell wireless spectrum. A second application is large networks, where the goal is to understand how such networks form, how network users behave, and what kind of design and management strategies ensure good network performance.
- Recent results that determine the computational complexity of computing a Nash equilibrium cast doubt on the concept's ability to predict the outcome of rational behavior.



ILLUSTRATION BY CELIA JOHNSON

explicitly elicited from self-interested participants. Auction settings are canonical examples, where the private data is the willingness to pay of the bidders for the goods on sale, and the optimization problem is to allocate the goods to maximize some objective, such as revenue or overall value to society. A “mechanism” is a protocol that interacts with participants and determines a solution to the underlying optimization problem.

There is a complex dependence between the way a mechanism employs elicited data and participant behavior. For example, consider the sale of a single good in a sealed-bid auction with several bidders. In a “first-price” auction, the selling price is the bid of the winner (that is, the maximum bid). Bidders naturally shade their bids below their maximum willingness to pay in first-price auctions, aspiring to achieve

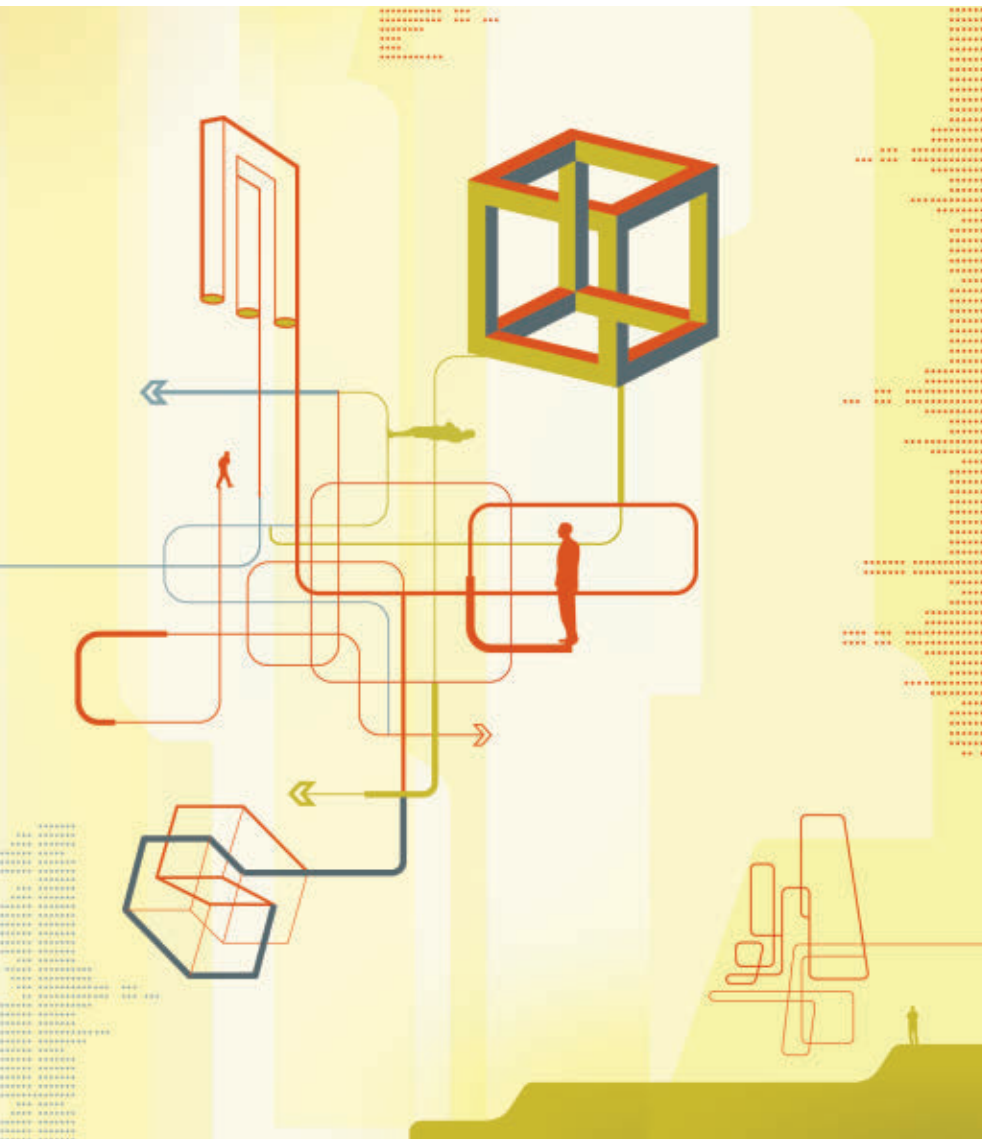
the lowest-possible price subject to winning the auction. Determining how much to shade requires guessing about the behavior of the other bidders. A different auction is the “second-price” auction, in which the selling price is only the *second-highest* bid. A famous result of Vickrey⁴³ is that every participant of a second-price auction may as well bid its true value for the good: intuitively, a second-price auction optimally shades the bid of the winner on its behalf, to the minimum alternative winning bid. eBay and Amazon auctions are similar to second-price auctions in many (but not all) respects; see Steiglitz⁴² for a detailed discussion. Keyword search auctions, such as those run by Google, Yahoo!, and Microsoft, are more complex variants of second-price auctions with multiple heterogeneous goods, corresponding to the potential ad slots on a search results

page. Lahaie et al.³⁰ provide an overview of theoretical work on search auctions.

While the economic literature on mechanism design is quite mature,²⁰ computer scientists have initiated a number of new research directions. We concentrate here on the emphasis in algorithmic mechanism design on complexity bounds and worst-case approximation guarantees, as first proposed by Nisan and Ronen.²⁹ Additional aspects including prior-free revenue-maximization, distributed (or Internet-suitable) mechanism design, and online (or real time) mechanism design are discussed in Nisan et al.³⁰

The technical core of this part of algorithmic mechanism design is the following deep question:

(Q1) To what extent is “incentive-compatible” efficient computation fundamentally less powerful than



“classical” efficient computation?

To translate this question into mathematics, reconsider the Vickrey (second-price) auction for selling a single good. Each bidder i has a private willingness-to-pay v_i and submits to the auctioneer a bid b_i . The auction comprises two algorithms: an *allocation algorithm*, which picks a winner, namely the highest bidder; and a *payment algorithm*, which uses the bids to charge payments, namely 0 for the losers and the second-highest bid for the winner. We argued intuitively that this auction is *truthful* in the following sense: for every bidder i and every set of bids by the other participants, bidder i maximizes its “net value” (its value for the good, if received, minus its payment, if any) by bidding its true private value: $b_i = v_i$. Moreover, no false bid is as good as the truthful bid for all possible bids by the

other participants. Assuming all bidders bid truthfully (as they should), the Vickrey auction solves the *social welfare maximization* problem, in the sense that the good is allocated to the participant with the highest value for it.

More generally, an allocation algorithm x is *implementable* if, for a judiciously chosen payment algorithm π , coupling x with π yields a truthful mechanism: every participant is guaranteed to maximize its payoff by reporting its true preferences. For a single-good auction, the “highest-bidder” allocation algorithm is implementable (as we have seen); the “second-highest bidder” allocation algorithm is not (a straightforward exercise). Thus some but not all algorithms are implementable.

We can mathematically phrase the question (Q1) as follows: *Are implementable algorithms less powerful than*

arbitrary algorithms for solving fundamental optimization problems?

Understanding this question involves two interrelated goals: characterization theorems and approximation bounds.

(G1) Usefully characterize the implementable allocation algorithms for an optimization problem.

(G2) Prove upper and lower bounds on the best-possible solution quality achieved by an implementable algorithm, possibly subject to additional constraints such as polynomial running time.

The second goal quantifies the limitations of implementable algorithms via an approximation measure; the most commonly used such measure is the worst-case ratio, over all possible inputs, between the objective function value of the algorithm’s solution and the optimal objective function value. The first goal aims to reformulate the unwieldy definition of implementability into a more operational form amenable to both upper and lower approximation bounds. Both goals, and especially (G1), seem to grow more complex with the number of independent parameters required to describe the private information of a participant.

Versions of (G2) pervade modern algorithmic research: for a given “constrained computational model,” where the constraint can be either computational (as for polynomial-time approximation algorithms) or information-theoretic (as for online algorithms), quantify its limitations for optimization and approximation. Goal (G1) reflects the additional difficulty in algorithmic mechanism design that even the “computational model” (of implementable algorithms) induced by strategic constraints is poorly understood. For example, determining whether or not a given algorithm is online is intuitively far easier than checking if one is implementable.

Single-Parameter Mechanism Design. This two-step approach is vividly illustrated by the important special case of *single-parameter* problems, where goal (G1) has been completely resolved. A mechanism design problem is single-parameter if the possible outcomes are real n -vectors ω and each participant i has an objective function of the form $v_i \omega_i$ for a private real number v_i ;

(the “single parameter”). The numbers ω_i and v_i can be thought of as the quantity received and the value per-unit of a good, respectively. A single-item auction is the special case in which each ω is either a standard basis vector or the all-zero vector. Keyword search auctions are also single-parameter, under the assumptions that every advertiser cares only about the probability ω_i of a click on its sponsored link and has a common value v_i for every such click.

An algorithm for a single-parameter problem is *monotone* if a greater bid begets a greater allocation: increasing the value of a bid (keeping the other bids fixed) can only increase the corresponding value of the computed ω_i . For example, the “highest bidder” allocation algorithm for a single-good auction is monotone, while the “second-highest bidder” allocation algorithm is not. In general, monotonicity characterizes implementability for single-parameter problems.

Myerson’s Lemma.²⁷ *An allocation algorithm for a single-parameter mechanism design problem is implementable if and only if it is monotone.*

Myerson’s Lemma is a useful solution to the first goal (G1) and reduces implementable algorithm design to monotone algorithm design. For example, consider the following “rank-by-weighted bid” allocation algorithm for a keyword search auction. Advertisers’ bids are sorted in decreasing order, possibly after scaling by advertiser-specific relevance factors, and ad slots are populated in this order. Assuming that the probability of a click is higher in higher slots, every such algorithm is monotone: increasing one’s bid can only increase one’s position in the ordering, which in turn leads to an only higher probability of a click. Thus, Myerson’s Lemma guarantees an analog of the second-price rule that extends the allocation algorithm into a truthful auction.^a

Despite our thorough understanding of goal (G1), question (Q1) remains open for single parameter problems. A

Truthful mechanisms are—by design—strategically degenerate in that the best course of action of a participant does not depend on the actions taken by others.

single-parameter scheduling problem proposed by Archer and Tardos¹ had been the most natural candidate for differentiating between the optimization power of monotone and arbitrary polynomial-time algorithms, but Dhangwatnotai et al.¹⁴ recently gave a (randomized) polynomial-time monotone algorithm for the problem with approximate guarantee as good as the best-possible polynomial-time algorithm (assuming $P \neq NP$).

Multiparameter Mechanism Design. Many important mechanism design problems are not single-parameter. *Combinatorial auctions*,¹¹ in which each participant aims to acquire a heterogeneous set of goods and has unrelated values for different sets, are a practical and basic example. Combinatorial auctions are used in practice to sell wireless spectrum (where the goods are different licenses), with auction designs by theoretical economists generating billions of dollars of revenue over the past decade.¹¹ Their complexity stems from “complements,” meaning goods that are more useful when purchased in tandem (for example, spectrum licenses for small but adjacent regions); and “substitutes,” meaning goods that are partially redundant (for example, two different but functionally identical licenses for the same region). Each bidder in a combinatorial auction has, in principle, an exponential number of private parameters—one private value for each subset of goods.

Multiparameter mechanism design is complex and our current understanding of goals (G1) and (G2) is primitive for most problems of interest. There are natural optimization problems for which there is a provable gap between the best-possible worst-case approximation ratio of implementable and arbitrary polynomial-time deterministic algorithms. This fact was first proved by Lavi et al.;²³ more recently, Papadimitriou et al.³³ showed that this gap can be as large as a polynomial in the number of bidders. Because of its importance and abundance of open questions, multiparameter mechanism design has been a hotbed of activity over the past few years. See Roughgarden³⁵ for a survey of the primary research threads, including upper and lower approximation bounds for polynomial-time welfare maximization for combinatorial auc-

a Modern search engines use allocation algorithms that are similar to rank-by-weighted bid algorithms. By historical accident, they use a slightly different pricing rule than that advocated by Myerson’s Lemma, although the two pricing rules lead to comparable outcomes and revenue at equilibrium. For details, see Lahaie et al.³⁰

tions, and work toward multiparameter analogs of Myerson’s Lemma.

Quantifying Inefficiency and the Price of Anarchy

The truthful mechanisms examined earlier are—by design—strategically degenerate in that the best course of action of a participant (that is, truth-telling) does not depend on the actions taken by the others. When a designer cannot specify the rules of the game and directly dictate the allocation of resources—or when there is no central designer at all—dependencies between different participants’ optimal courses of action are generally unavoidable and preclude exact optimization of standard objective functions. This harsh reality motivates adopting an equilibrium concept—a rigorous proposal for the possible outcomes of a game with self-interested participants—and an approximation measure that quantifies the inefficiency of a game’s equilibria, to address the following basic question:

(Q2) When, and in what senses, are game-theoretic equilibria guaranteed to approximately optimize natural objective functions?

Such a guarantee implies that the benefit of imposing additional control over the system is small, and is particularly reassuring when implementing an optimal solution is infeasible (as in a typical Internet application).

Routing with Congestion. There are now numerous answers to question (Q2) in different models. We describe one by Roughgarden and Tardos,^{37,39} for a model of “selfish routing” originally pro-

posed for road traffic (see Beckmann⁴) and subsequently adapted to communication networks (see Bertsekas and Tsitsiklis⁵). This was the first general approximation bound on the inefficiency of equilibria; the idea of quantifying such inefficiency was explored previously in a scheduling model.²²

Consider a directed graph with fixed traffic rates between various origin-destination pairs in which the traffic chooses routes to minimize individual cost; see also Figure 1. Here, we assume that the traffic comprises a large number of selfish users, each of negligible size, such as drivers on a highway or packets in a network. Edge costs are congestion-dependent, with the continuous, nondecreasing function $c_e(x)$ denoting the per-unit cost incurred by traffic on edge e when x units of traffic use it. In an equilibrium, each user travels along a minimum-cost path from its origin to its destination, given the congestion caused by the traffic. These selfish routing games are strategically non-trivial in that the minimum-cost path for a given user generally depends on the paths chosen by the others.

For example, in a “Pigou-like network” (Figure 1a), r units of selfish traffic autonomously decide between parallel edges e_1 and e_2 that connect the origin s to the destination t . Suppose the second edge has some cost function $c_2(\cdot)$, and the first edge has a constant cost function c_1 everywhere equal to $c_2(r)$. Such networks are strategically trivial, just like the truthful mechanisms noted earlier: the second edge’s cost is never larger than that of the

first, even when it is fully congested. For this reason, all traffic uses the second edge at equilibrium. This equilibrium does not generally minimize the average cost of all users. For example, if $r = 1$ and $c_2(x) = x$ as in Figure 1a, the average cost at equilibrium is 1, while splitting the traffic equally between the two edges yields a routing with average cost $3/4$. The latter traffic pattern is not an equilibrium because of a “congestion externality”: a selfish network user routed on the first edge would switch to the second edge, indifferent to the fact that this switch (slightly) increases the cost incurred by a large portion of the population. Similarly, in the Braess’s Paradox⁷ network of Figure 1b, the average cost at equilibrium is 2 (with all traffic on the zig-zag path), while a benevolent dictator could route the traffic at average cost $3/2$ (by splitting traffic between the two two-hop paths).^b

The *price of anarchy* (POA) of a selfish routing network is the ratio of the average user cost at equilibrium and in an optimal routing— $4/3$ in both of the networks in Figure 1. The closer the POA is to 1, the lesser the consequences of selfish behavior. Replacing the cost function of the second edge in Figure 1a by $c_2(x) = x^d$ for large d shows that the POA can

b This network is called a “paradox” because removing the intuitively helpful zero-cost edge—depriving users of one of their options—recovers the optimal solution as an equilibrium, thereby decreasing the cost incurred by all users. Analogously, cutting a taut string in a network of strings and springs that carries a heavy weight can cause the weight to levitate further off the ground!¹⁰

Figure 1. Two selfish routing networks with price of anarchy $4/3$. One unit of selfish traffic travels from s to t . At equilibrium, all traffic travels on the bottom path and the zig-zag path, respectively. In an optimal solution, traffic is split equally between the two edges and between the two two-hop paths, respectively.

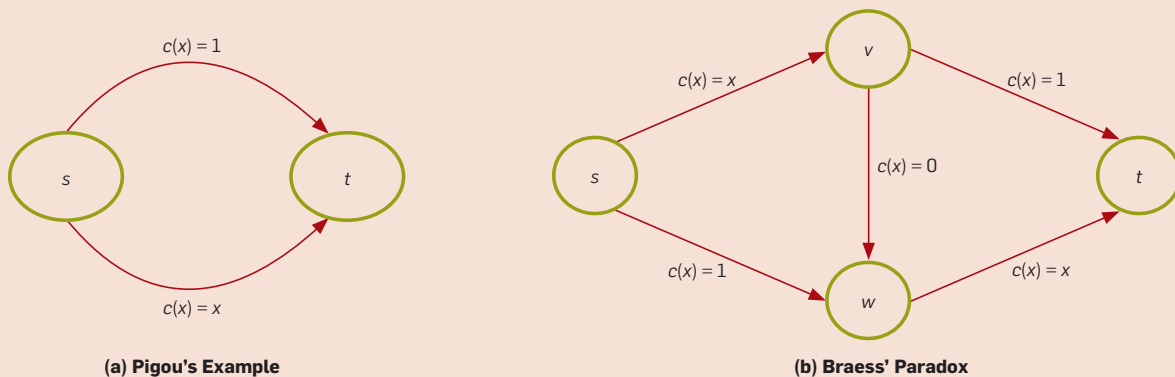
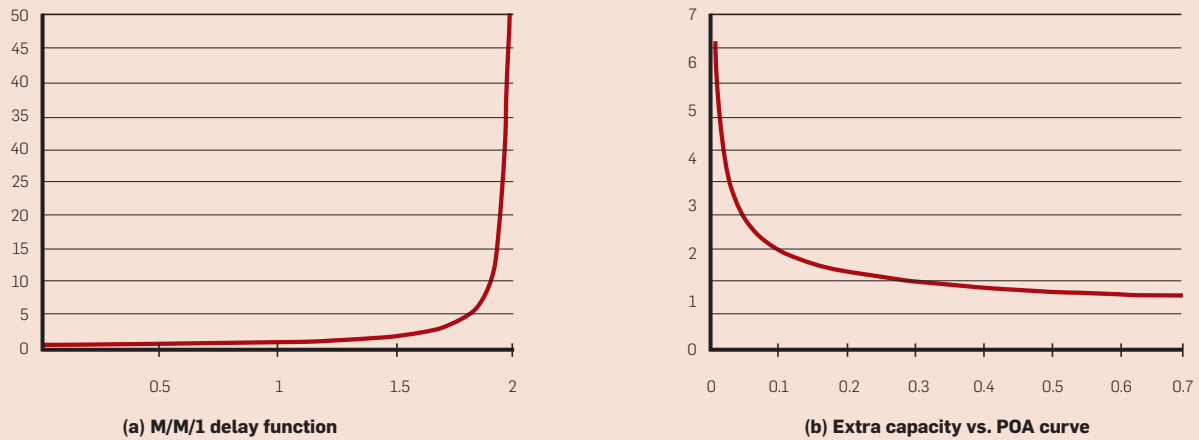


Figure 2. Modest overprovisioning guarantees near-optimal routing. (a) displays the per-unit cost $c(x) = 1/(u - x)$ as a function of the load x for an edge with capacity $u = 2$. (b) shows the worst-case price of anarchy as a function of the fraction of unused network capacity.



networks, and suggests that the POA is governed by the “degree of nonlinearity” of the cost function c_2 . A key result formalizes and extends this intuition to arbitrary networks: among all networks with cost functions lying in a set C (for example, bounded-degree polynomials with nonnegative coefficients), the largest-possible POA is achieved already in Pigou-like networks.³⁷ Conceptually *complex topologies do not amplify the worst-case POA*. This reduction permits the easy calculation of tight bounds on the worst-case POA for most interesting sets C of cost functions. For example, the POA of every selfish routing network with affine cost functions (of the form $c_e(x) = a_e x + b_e$ for non-negative a_e, b_e) is at most $4/3$, with a matching lower bound provided by the examples in Figure 1.³⁹ See Nisan et al.³⁰ for a recent survey detailing these and related results.

These POA bounds provide a theoretical justification for a common rule of thumb used in network design and management: overprovisioning networks with extra capacity ensures good performance. Precisely, suppose every edge e of a network has a capacity u_e and a corresponding cost function $c_e(x) = 1/(u_e - x)$; see Figure 2a. (If $x \geq u_e$, we interpret the cost as infinite.) This is the standard M/M/1 queueing delay function with service rate u_e . We say that a network is β -overprovisioned for $\beta \in (0, 1)$ if, at equilibrium, at least a β fraction of each edge’s capacity remains unused. The following is a tight bound on the POA for such networks; the bound

Theorem (Consequence of Roughgarden³⁷) *The POA of every β -overprovisioned network is at most*

$$\frac{1}{2} \left(1 + \frac{1}{\sqrt{\beta}} \right)$$

Thus even 10% extra capacity reduces the worst-case price of anarchy of selfish routing to roughly 2.

Further Aspects of Quantifying Inefficiency. We have barely scratched the surface of recent work on equilibrium efficiency analyses. For an overview of work on some other application domains, including resource allocation, scheduling, facility location, and network design, see Nisan et al.³⁰

An important emerging trend in this area is to prove POA-type bounds under increasingly weak assumptions on the rationality of participants. Recall in algorithmic mechanism design, our only assumption was that participants will make use of a foolproof strategy (one that dominates all others), should one be available. Here, we implicitly assumed that selfish participants can reach an equilibrium of a game without such foolproof strategies, presumably through repeated experimentation. This much stronger assumption has been addressed in two different ways in the recent literature. The first is to formally justify it by positing natural experimentation strategies and proving that they quickly reach a (possibly approximate) equilibrium; see Chien and Sinclair⁹ and the references therein for a sampling of such

like guarantees that apply “on average,” even when such experimentation strategies fail to converge to an equilibrium. Remarkably, such approximation bounds hold in interesting classes of games, including in selfish routing networks. See Awerbuch et al.,² Blum et al.,⁶ and Goemans et al.¹⁹ for initial formalizations of this approach. Roughgarden³⁶ recently proved the general result that, under fairly weak conditions, POA bounds for equilibria extend automatically to the results of repeated experimentation.

Complexity of Equilibrium Computation

Equilibrium concepts—most famously the Nash equilibrium²⁸—play a starring role in game theory and microeconomics. If nothing else, a notion of equilibrium describes outcomes that, once reached, persist under some model of individual behavior. In engineering applications we generally demand a stronger interpretation of an equilibrium, as a credible prediction of the long-run state of the system. But none of the standard equilibrium notions or the corresponding proofs of existence suggest how to arrive at an equilibrium with a reasonable amount of effort. This fact motivates the following questions.

(Q3) When can the participants of a game quickly converge to an equilibrium? More modestly, when can a centralized algorithm quickly compute an equilibrium?

These questions are interesting for two reasons. First, algorithms for equilibrium computation can be useful practically, for example in game-playing and for multi-agent reasoning.⁴¹ Second, assuming that players can invest only polynomial computation in playing a game, resolving the complexity of computing an equilibrium concept has economic implications: a polynomial-time algorithm is an important step toward establishing the concept's credibility, while an intractability result casts doubt on its predictive power.

There has been a frenzy of recent work on these questions, for many different fundamental equilibrium concepts. Perhaps the most celebrated results in the area concern the *PPAD*-completeness of computing mixed-strategy Nash equilibria in finite games with two or more players.^{8,12} To briefly convey the spirit of the area with a minimum of technical fuss, we instead discuss the complexity of converging to and computing pure-strategy Nash equilibria in a variant of the routing games discussed earlier. We then discuss the key differences between the two settings. For work on the complexity of computing other equilibrium concepts, such as market, correlated, and approximate Nash equilibria, and for a discussion of equilibrium computation in extensive-form, compact, randomly generated, and stochastic games, see Nisan³⁰ and Roughgarden³⁸ and the references therein.

Pure Nash Equilibria in Network Congestion Games. In the *atomic* variant of selfish routing, there are a finite number k of players that each control a non-negligible amount of traffic (say one unit each) and choose a single route for it. Each edge cost function $c_e : \{1, 2, \dots, k\} \rightarrow \mathbb{R}^+$, describing the per-player cost along an edge as a function of its number of users, is non-decreasing. An outcome (P_1, \dots, P_k) —a choice of a path P_i for each player i —is a *pure-strategy Nash equilibrium (PNE)* if each player simultaneously chooses a best response: a path with minimum possible cost, given the paths chosen by the other players. For instance, consider Pigou's example (Figure 1a) with the constant cost on the upper edge raised from 1 to 2. If there are two players (with origin s and destination t), then there are three

Equilibrium concepts play a starring role in game theory. If nothing else, a notion of equilibrium describes outcomes that, once reached, persist under some model of individual behavior.

PNE: one with both players on the lower link, and two in which each link is used by a single player. In every case, a deviating player would incur cost 2 and be no better off than in the equilibrium.

Best-response dynamics is a simple model of experimentation by players over time: while the current outcome is not a PNE, choose an arbitrary player that is not using a best response, and update its path to a best response. The update of one player usually changes the best responses of the others; for this reason, best-response dynamics fails to converge in many games (such as “Rock-Paper-Scissors”). In an atomic selfish routing network, however, every iteration of best-response dynamics strictly decreases the potential function

$$\Phi(P_1, \dots, P_k) = \sum_{e \in E} [c_e(1) + c_e(2) + \dots + c_e(x_e)],$$

where x_e denotes the number of paths P_i that contain edge e , and is thus guaranteed to terminate, necessarily at a PNE.^{26,34} Does convergence require polynomial or exponential time? Can we compute a PNE of such a game by other means in polynomial time?

Assume for the moment that the problem of computing a PNE of an atomic selfish routing network is not solvable in polynomial time; how would we amass evidence for this fact? An obvious idea is to prove that the problem is *NP*-hard. Remarkably, a short argument^{21,25} shows that this is possible only if $NP = coNP$! Intuitively, solving an *NP*-hard problem like satisfiability means to either exhibit a satisfying truth assignment of the given Boolean formula or to correctly determine that none exist. Computing a PNE of an atomic selfish routing game appears easier because the latter situation (of there being no PNE) can be ruled out a priori—the “only” challenge is to exhibit a solution in polynomial time.^c

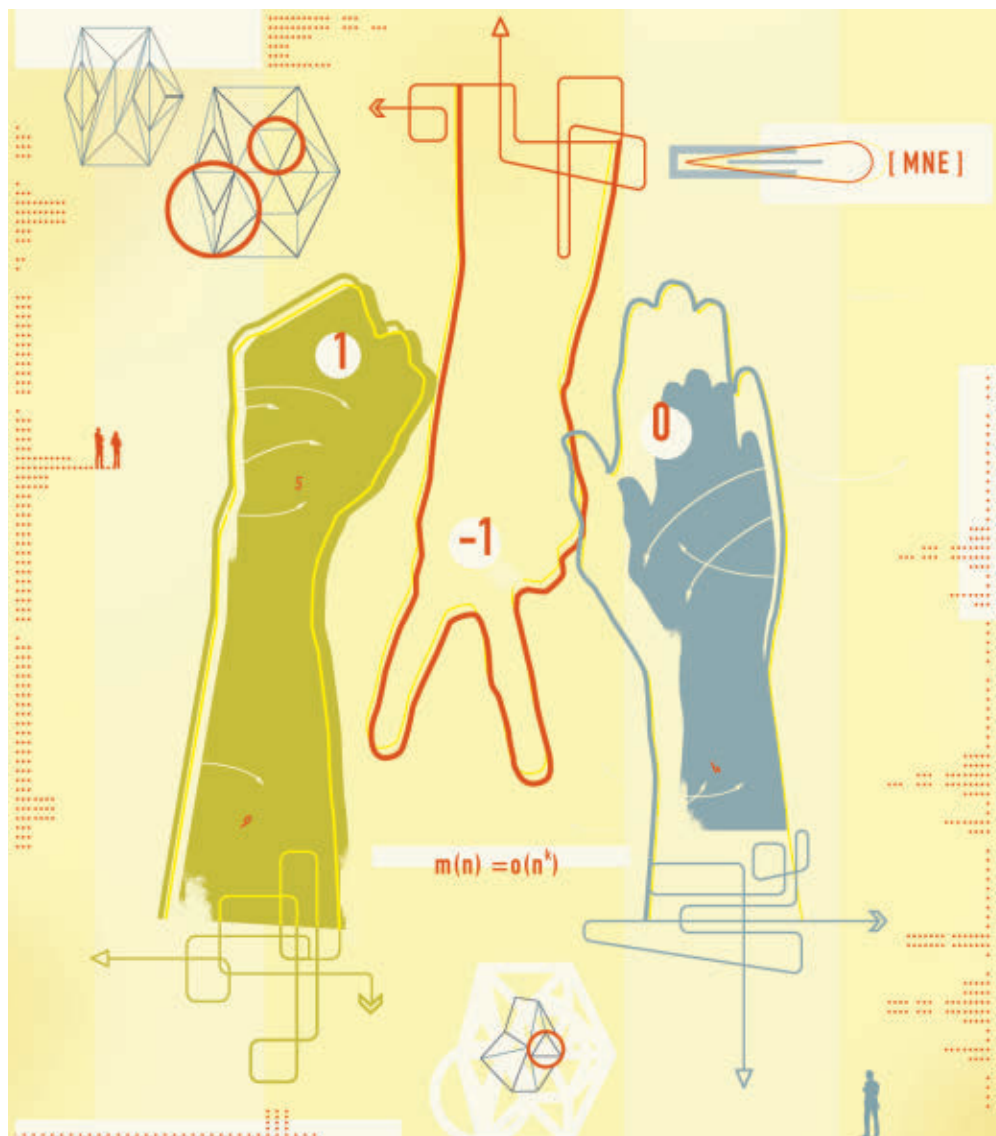
To motivate the definition of the appropriate complexity class, recall that problems in the class *NP* are characterized by short and efficiently verifiable witnesses of membership, such as

^c The complexity classes *P* and *NP* are usually defined for decision problems, where the answer sought is a simple “yes” or “no.” Here we refer to the similar but more general *search* versions of *P* and *NP*, where for a “yes” instance, the deliverables include a correct solution.

satisfying truth assignments or Hamiltonian cycles. There is thus a generic “brute-force search” algorithm for NP problems: given an input, enumerate the exponentially many possible witnesses of membership, and check if any of them are valid. Computing a PNE of an atomic selfish routing game appears to be easier than an NP -hard problem because there is a *guided search* algorithm (namely, best-response dynamics) that navigates the set of possible witnesses and is guaranteed to terminate with a legitimate one. At worst, computing a PNE might be as hard as all problems solvable by such a guided search procedure. This is in fact the case, as we formalize here.

What are the minimal ingredients that guarantee that a problem is solvable via guided search? The answer is provided by the complexity class PLS (for “polynomial local search”).²¹ A PLS problem is described by three polynomial-time algorithms: one to accept an instance and output an initial candidate solution; one to evaluate the objective function value of a candidate solution; and one that either verifies local optimality (for some local neighborhood) or else returns a neighboring solution with strictly better objective function value. To solve a PLS problem means to compute a local optimum, by local search or by other means. For example, computing a PNE of an atomic selfish routing game can be cast as a PLS problem by adopting the potential function as an objective function, and defining two outcomes to be neighbors if all but one player choose the same path in both. Local minima then correspond to the PNE of the game. A problem in PLS is then *PLS-complete* if every problem in PLS reduces to it in polynomial time, in which case the complete problem is solvable in polynomial time only if every problem in PLS is.

The problem of computing a PNE of an atomic selfish routing network is PLS -complete.¹⁷ It is therefore polynomial-time solvable if and only if $P = PLS$. In the spirit of the P vs. NP question, it is generally believed that $P \neq PLS$ but researchers seem far from a resolution in either direction. Since PLS contains several important problems that have resisted all attempts at a computationally efficient solution, PLS -hardness is



viewed as strong evidence that a problem will not be solved in polynomial time (at least in the near future).

Mixed-Strategy Nash Equilibria and PPAD. A *mixed strategy* is a probability distribution over the pure strategies of a player. In a *mixed-strategy Nash equilibrium* (MNE), every player simultaneously chooses a mixed strategy maximizing its expected payoff, given those chosen by the others. For example, in “Rock-Paper-Scissors,” with each player receiving payoff 1 for a win, 0 for a draw, and -1 for a loss, the only MNE has each player randomizing uniformly over its three strategies to obtain an expected payoff of 0. Nash proved that every game with a finite number of players and strategies has at least one MNE.²⁸ Computing an MNE of a finite game is a central equilibrium computation problem.

We focus on the two-player (“bimatrix”) case, where the input is two $m \times n$ payoff matrices (one for each player) with integer entries; with three or more players, the problem appears to be harder in a precise complexity-theoretic sense.¹⁵ We emphasize that the two payoff matrices are completely unrelated, and need not be “zero-sum” like in Rock-Paper-Scissors. (When the two payoff matrices sum to a constant matrix, an MNE can be computed in polynomial time via linear programming; see for example, Nisan³⁰ for details.)

There is a non-obvious “guided search” algorithm for two-player games called the *Lemke-Howson algorithm*;²⁴ see von Stengel³⁰ for a careful exposition. This algorithm is a path-following algorithm in the spirit of local search, but it is not guided by

an objective or potential function and thus does not prove that computing an MNE of a bimatrix game is in *PLS*. In conjunction with our earlier reasoning, however, the Lemke-Howson algorithm shows that the problem is not *NP*-hard unless $NP = coNP$.²⁵

A complexity class that is related to but apparently different from *PLS* is *PPAD*, which stands for “polynomial parity argument, directed version”. This class was defined in Papadimitriou³² to capture the complexity of computing MNE and related problems, such as computing approximate Brouwer fixed points. Its formal definition parallels that of *PLS*, with a *PPAD* problem consisting of the minimal ingredients necessary to execute a Lemke-Howson-like path-following procedure (again easily phrased as three polynomial-time algorithms). A problem in *PPAD* is *PPAD*-complete if every problem in *PPAD* reduces to it in polynomial time; the complete problem is then polynomial-time solvable only if all problems in *PPAD* are. Since *PPAD* contains several well-studied problems that are not known to be solvable via a polynomial-time algorithm, a proof of *PPAD*-completeness can be interpreted as a significant intractability result.

A few years ago, the problem of computing an MNE of a bimatrix game was shown to be *PPAD*-complete.^{8, 12} Thus, if $P \neq PPAD$, there is no general-purpose and computationally efficient algorithm for this problem, and in particular there is no general and tractable way for players to reach a Nash equilibrium in a reasonable amount of time. This hardness result casts doubt on the predictive power of the Nash equilibrium concept in arbitrary games. See Chen⁸ and Daskalakis et al.¹² for the details of this tour de force result and Daskalakis et al.¹³ for a high-level survey of the proof.

Future Directions

The rapid rate of progress in algorithmic game theory has been nourished by deep connections with other areas of theoretical computer science and a consistent infusion of new motivating applications. There remains a surfeit of important open research directions across all three of the AGT areas surveyed here, such as develop-

ing theory for the design and analysis of mechanisms for multi-parameter problems, for minimizing the inefficiency of equilibria (for example, via a mediating network protocol), and for the computation of approximate equilibria. See Roughgarden³⁵ and the concluding sections of many chapters in Nisan³⁰ for more details and many concrete open problems.

A broad challenge, mentioned also in Shoham’s recent *Communications* article,⁴⁰ is to develop more appropriate models of agent behavior. All of the results described in this article, even the welfare guarantee of the simple second-price auction, depend on some kind of behavioral assumptions about the participants. Such assumptions are required to address modern applications, yet are largely foreign to the theoretical computer science mindset, which is characterized by minimal assumptions and worst-case analysis. But a number of new types of worst-case guarantees, coupled with novel behavioral models, have already begun to sprout in the AGT literature. For example: mechanism implementation in undominated strategies³ and in ex post collusion-proof Nash equilibrium;³¹ the price of total anarchy;^{6,36} and the complexity of unit-recall games.¹⁶ We expect these are only the vanguard of what promises to be a rich and relevant theory. □

This work is supported in part by NSF CAREER Award CCF-0448664, an ONR Young Investigator Award, an AFOSR MURI grant, and an Alfred P. Sloan Fellowship.

References

1. Archer, A. and Tardos, É. Truthful mechanisms for one-parameter agents. *FOCS '01*, 482–491.
2. Awerbuch, B., Azar, Y., Epstein, A., Mirrokni, V.S., and Skopalik, A. Fast convergence to nearly optimal solutions in potential games. *EC '08*, 264–273.
3. Babaioff, M., Lavi, R., and Pavlov, E. Single-value combinatorial auctions and algorithmic implementation in undominated strategies. *JACM* 56, 1 (2009).
4. Beckmann, M.J., McGuire, C.B., and Winsten, C.B. *Studies in the Economics of Transportation*. Yale University Press, 1956.
5. Bertsekas, D.P. and Tsitsiklis, J.N. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.
6. Blum, A., Hajiaghayi, M., Ligett, K., and Roth, A. Regret minimization and the price of total anarchy. *STOC '08*, 373–382.
7. Braess, D. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12 (1968), 258–268.
8. Chen, X., Deng, X., and Teng, S.-H. Settling the complexity of computing two-player Nash equilibria. *JACM* 56, 3 (2009).
9. Chien, S. and Sinclair, S. Convergence to approximate Nash equilibria in congestion games. *SODA '07*, 169–178.
10. Cohen, J.E. and Horowitz, P. Paradoxical behavior of mechanical and electrical networks. *Nature* 352, 8 (1991), 699–701.

11. Cramton, P., Shoham, Y., and Steinberg, R., Eds. *Combinatorial Auctions*. MIT Press, 2006.
12. Daskalakis, C., Goldberg, P.W., and Papadimitriou, C.H. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing* 39, 1 (2009), 195–259.
13. Daskalakis, C., Goldberg, P.W., and Papadimitriou, C.H. The complexity of computing a Nash equilibrium. *Commun. ACM* 52, 2 (Feb. 2009) 89–97.
14. Dhangwatnotai, P., Dobzinski, S., Dughmi, S., and Roughgarden, T. Truthful approximation schemes for single-parameter agents. *FOCS '08*, 15–24.
15. Etessami, K. and Yannakakis, M. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing* 39, 6 (2010) 2531–2597.
16. Fabrikant, A. and Papadimitriou, C.H. The complexity of game dynamics: BGP oscillations, sink equilibria, and beyond. *SODA '08*, 844–853.
17. Fabrikant, A., Papadimitriou, C.H., and Talwar, K. The complexity of pure Nash equilibria. *STOC '04*, 604–612.
18. Feigenbaum, J., Parkes, D.C., and Pennock, D.M. Computational challenges in e-commerce. *Commun. ACM* 52, 1 (Jan. 2009), 70–74.
19. Goemans, M.X., Mirrokni, V.S., and Vetta, A. Sink equilibria and convergence. *FOCS '05*, 142–151.
20. Jackson, M.O. A crash course in implementation theory. *Social Choice and Welfare* 18, 4 (2001), 655–708.
21. Johnson, D.S., Papadimitriou, C.H., and Yannakakis, M. How easy is local search? *J. Computer and System Sciences* 37, 1 (1988), 79–100.
22. Koutsoupias, E. and Papadimitriou, C.H. Worst-case equilibria. *STACS '99*, 404–413.
23. Lavi, R., Mu’alem, A., and Nisan, N. Towards a characterization of truthful combinatorial auctions. *FOCS '03*, 574–583.
24. Lemke, C.E. and Howson, J.T., Jr. Equilibrium points of bimatrix games. *SIAM J. 12*, 2 (1964), 413–423.
25. Megiddo, N. and Papadimitriou, C.H. On total functions, existence theorems and computational complexity. *Theoretical Computer Science* 81, 2 (1991), 317–324.
26. Monderer, D. and Shapley, L.S. Potential games. *Games and Economic Behavior* 14, 1 (1996), 124–143.
27. Myerson, R. Optimal auction design. *Mathematics of Operations Research* 6, 1 (1981), 58–73.
28. Nash, J.F., Jr. Equilibrium points in *N*-person games. In *Proceedings of the National Academy of Science* 36, 1 (1950), 48–49.
29. Nisan, N. and Ronen, A. Algorithmic mechanism design. *Games and Economic Behavior* 35, 1–2 (2001), 166–196.
30. Nisan, N., Roughgarden, T., Tardos, É., and Vazirani, V.V., Eds. *Algorithmic Game Theory*. Cambridge University Press, 2007.
31. Nisan, N., Schapira, M., Valiant, G., and Zohar, A. Best-reply mechanisms. Working paper, 2009.
32. Papadimitriou, C.H. On the complexity of the parity argument and other inefficient proofs of existence. *J. Computer and System Sciences* 48 3 (1994), 498–532.
33. Papadimitriou, C.H., Schapira, M., and Singer, Y. On the hardness of being truthful. *FOCS '08*, 250–259.
34. Rosenthal, R.W. A class of games possessing pure-strategy Nash equilibria. *International J. Game Theory* 2, 1 (1973), 65–67.
35. Roughgarden, T. Algorithmic game theory: Some greatest hits and future directions. *TCS '08*, 21–42.
36. Roughgarden, T. Intrinsic robustness of the price of anarchy. *STOC '09*, 513–522.
37. Roughgarden, T. The price of anarchy is independent of the network topology. *J. Computer and System Sciences* 67, 2 (2003), 341–364.
38. Roughgarden, T. Computing equilibria: A computational complexity perspective. *Economic Theory* 42, 1 (Jan. 2010), 193–236.
39. Roughgarden, T. and Tardos, É. How bad is selfish routing? *JACM* 49, 2 (2002), 236–259.
40. Shoham, Y. *Computer science and game theory*. *Commun. ACM* 51, 8 (Aug. 2008), 75–79.
41. Shoham, Y. and Leyton-Brown, K. *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations*. Cambridge University Press, 2008.
42. Steiglitz, K. *Snipers, Shills, and Sharks: eBay and Human Behavior*. Princeton University Press, 2007.
43. Vickrey, W. Counterspeculation, auctions, and competitive sealed tenders. *J. Finance* 16, 1 (1961), 8–37.

Tim Roughgarden (tim@cs.stanford.edu) is an assistant professor in the computer science and management science and engineering departments at Stanford University, Stanford, CA. He is the recipient of ACM’s 2009 Grace Murray Hopper Award.

research highlights

P. 88

**Technical
Perspective
A Solid Foundation
for x86 Shared
Memory**

By Hans-J. Boehm

P. 89

**x86-TSO: A Rigorous and
Usable Programmer's Model
for x86 Multiprocessors**

By Peter Sewell, Susmit Sarkar, Scott Owens,
Francesco Zappa Nardelli, and Magnus O. Myreen

P. 98

**Technical
Perspective
Technology Scaling
Redirects Main
Memories**

By Mary Jane Irwin

P. 99

**Phase Change Memory
Architecture and the Quest
for Scalability**

By Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger

Technical Perspective

A Solid Foundation for x86 Shared Memory

By Hans-J. Boehm

MULTITHREADED PROGRAMS THAT communicate through shared memory are pervasive. They originally provided a convenient way for an application to perform, for example, a long compute task while remaining responsive to an interactive user. Today they are the most obvious route to using multiple available processor cores for a single task, so the user can benefit from the increased number of available cores.

Unfortunately, a surprising amount of confusion surrounds the basic rules obeyed by shared memory. If a variable is updated by one thread, when will the new value become visible to another thread? What does it mean, if anything, to have two threads updating the same variable at the same time? Do all threads have to see updates in a consistent order?

The confusion surrounding these issues has resulted in many intermittent software bugs, often in low-level libraries that affect large numbers of applications. On at least one occasion, it has resulted in a pervasively used, but usually incorrect, programming idiom. (Try searching for “double-checked locking”).

This problem arises at different levels. At the programming language level, there must be clear rules for the programmer’s use of shared variables. Compilers and low-level libraries must enforce these rules by relying on corresponding hardware properties for memory-access instructions—the subject of the following paper.

Most of the early work on shared-memory semantics focused on the instruction set level and trade-offs with hardware optimization. Roughly concurrently, some older programming language designs, notably Ada 83, made very credible attempts to address the language-level issue. Unfortunately none of this prevented major problems in the most widely used languages from escaping attention until very recently. The Java specification

was drastically revised around 2005,² and has still not completely settled.^{1,3} Similarly, the C and C++ specifications are being revised to finally address similar issues.¹ As a result, hardware architects often could not have a clear picture of the programming language semantics they needed to support, making a fully satisfactory resolution of the hardware-level issues more difficult or impossible.

The recent work on shared variables in programming languages highlighted some remaining questions about hardware memory models. For example, Java’s `volatile` requires that updates of all such fields must become visible in the same order to all other threads. Is it even possible to enforce that on common hardware at reasonable cost? What instruction sequences do compilers need to generate? Until 2007 or so, the answers to such questions remained unclear, and sometimes unraised, on several major architectures. X86 is probably the most visible of these architectures.

Raising these issues resulted in extended discussions involving the machine architects. As a result of this process, Intel and AMD have released a sequence of specifications for x86 shared-memory accesses. These are far more precise than they were at the start of the process, and directly address many interesting test cases that arose during the discussions. However, they are still not a precise mathematical description that could be used to, for example, prove that an implementation of Java `volatile` is correct.

The x86-TSO model fills that gap, by providing precise mathematical and empirically accurate models of x86 shared memory as it is visible to user programs. These include an operational model presented here in a very intuitive fashion. In the process of making the model precise and directly testing it against existing implementations, Sewell et al. expose new issues

not currently addressed by the manufacturers’ specifications, while also confirming their model is compatible with existing implementations.

The examples here are interesting, not just because it may be surprising that there are tiny program fragments (often with only four instructions) for your desktop computer whose meaning is still open to debate, but also because these same small examples are often at the core of important algorithms or software. The first example is an abstraction of Dekker’s mutual exclusion algorithm first described in 1965. The same property is important to many modern lock-free algorithms. We already mentioned the importance of a consistent write visibility ordering for Java `volatiles`. The upcoming C and C++ standards introduce an approximate analog, `atomic` variables, aspects of which also rely on the issues surrounding single-variable coherence (examples `n5` and `n4b` in the paper) being resolved exactly as suggested by x86-TSO. The paper itself discusses the impact of these issues on Linux spin-lock code.

X86-TSO describes the behavior of x86 memory accesses, which depends on the behavior of processors produced by multiple vendors, and chipsets produced by an even larger number. This helped to put academic researchers into the best position to write such a specification. ■

References

1. Adve, S. and Boehm, H.-J. Memory models: A case for rethinking parallel languages and hardware. To appear in *Commun. ACM*.
2. Manson, J., Pugh, W., and Adve, S. The Java memory model. In *Proceedings of the Symposium on Principles of Programming Languages*, 2005.
3. Sevcik, J. and Aspinall, D. On validity of program transformations in the Java memory model. In *ECOOP 2008*, 27–51.

Hans-J. Boehm (Hans.Boehm@hp.com) is a member of Hewlett-Packard’s Exascale Computing Lab, Palo Alto, CA.

x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors

By Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen

Abstract

Exploiting the multiprocessors that have recently become ubiquitous requires high-performance and reliable concurrent systems code, for concurrent data structures, operating system kernels, synchronization libraries, compilers, and so on. However, concurrent programming, which is always challenging, is made much more so by two problems. First, real multiprocessors typically do not provide the sequentially consistent memory that is assumed by most work on semantics and verification. Instead, they have relaxed memory models, varying in subtle ways between processor families, in which different hardware threads may have only loosely consistent views of a shared memory. Second, the public vendor architectures, supposedly specifying what programmers can rely on, are often in ambiguous informal prose (a particularly poor medium for loose specifications), leading to widespread confusion.

In this paper we focus on x86 processors. We review several recent Intel and AMD specifications, showing that all contain serious ambiguities, some are arguably too weak to program above, and some are simply unsound with respect to actual hardware. We present a new *x86-TSO* programmer's model that, to the best of our knowledge, suffers from none of these problems. It is mathematically precise (rigorously defined in HOL4) but can be presented as an intuitive abstract machine which should be widely accessible to working programmers. We illustrate how this can be used to reason about the correctness of a Linux spinlock implementation and describe a general theory of data-race freedom for x86-TSO. This should put x86 multiprocessor system building on a more solid foundation; it should also provide a basis for future work on verification of such systems.

1. INTRODUCTION

Multiprocessor machines, with many processors acting on a shared memory, have been developed since the 1960s; they are now ubiquitous. Meanwhile, the difficulty of programming concurrent systems has motivated extensive research on programming language design, semantics, and verification, from semaphores and monitors to program logics, software model checking, and so forth. This work has almost always assumed that concurrent threads share a single sequentially consistent memory,²¹ with their reads and writes interleaved in some order. In fact, however, real multiprocessors use sophisticated techniques to achieve high performance: store buffers, hierarchies of local cache, speculative execution,

etc. These optimizations are not observable by sequential code, but in multithreaded programs different threads may see subtly different views of memory; such machines exhibit *relaxed*, or *weak*, *memory models*.^{6,7,17,19}

For a simple example, consider the following assembly language program (SB) for modern Intel or AMD x86 multiprocessors: given two distinct memory locations x and y (initially holding 0), if two processors respectively write 1 to x and y and then read from y and x (into register EAX on processor 0 and EBX on processor 1), it is possible for both to read 0 *in the same execution*. It is easy to check that this result cannot arise from any interleaving of the reads and writes of the two processors; modern x86 multiprocessors do not have a sequentially consistent semantics.

SB

Proc 0	Proc 1
MOV [x]←1 MOV EAX←[y]	MOV [y]←1 MOV EBX←[x]
Allowed Final State: Proc 0:EAX=0 ∧ Proc 1:EBX=0	

Microarchitecturally, one can view this particular example as a visible consequence of store buffering: if each processor effectively has a FIFO buffer of pending memory writes (to avoid the need to block while a write completes), then the reads from y and x could occur before the writes have propagated from the buffers to main memory.

Other families of multiprocessors, dating back at least to the IBM 370, and including ARM, Itanium, POWER, and SPARC, also exhibit relaxed-memory behavior. Moreover, there are major and subtle differences between different processor families (arising from their different internal design choices): in the details of exactly what non-sequentially-consistent executions they permit, and of what memory barrier and synchronization instructions they provide to let the programmer regain control.

For any of these processors, relaxed-memory behavior exacerbates the difficulties of writing concurrent software, as systems programmers cannot reason, at the level of abstraction of memory reads and writes, in terms of an intuitive concept of global time.

This paper is based on work that first appeared in the *Proceedings of the 36th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2009, and in the *Proceedings of the 22nd International Conference on Theorem Proving in Higher-Order Logics (TPHOLS)*, 2009.

Still worse, while some vendors' architectural specifications clearly define what they guarantee, others do not, despite the extensive previous research on relaxed memory models. We focus in this paper on x86 processors. In Section 2, we introduce the key examples and discuss several vendor specifications, showing that they all leave key questions ambiguous, some give unusably weak guarantees, and some are simply wrong, prohibiting behavior that actual processors do exhibit.

For there to be any hope of building reliable multiprocessor software, systems programmers need to understand what relaxed-memory behavior they *can* rely on, but at present that understanding exists only in folklore, not in clear public specifications. To remedy this, we aim to produce mathematically precise (but still appropriately loose) programmer's models for real-world multiprocessors, to inform the intuition of systems programmers, to provide a sound foundation for rigorous reasoning about multiprocessor programs, and to give a clear correctness criterion for hardware. In Section 3, we describe a simple x86 memory model, x86-TSO.²⁷ In contrast to those vendor specifications, it is unambiguous, defined in rigorous mathematics, but it is also accessible, presented in an operational abstract-machine style. To the best of our knowledge it is consistent with the behavior of actual processors. We consider the relevant vendor litmus tests in Section 3.2 and describe some empirical test results in Section 3.3.

Relaxed memory behavior is particularly critical for low-level systems code: synchronization libraries, concurrent data structure libraries, language runtime systems, compilers for concurrent languages, and so on. To reason (even informally) about such code, such as the implementation of an OS mutual exclusion lock, one would necessarily depend on the details of a specific model. Higher-level application code, on the other hand, should normally be oblivious to the underlying processor memory model. The usual expectation is that such code is in some sense *race free*, with all access to shared memory (except for accesses within the library code) protected by locks or clearly identified as synchronization accesses. Most memory models are designed with the intention that such race-free code behaves *as if* it were executing on a sequentially consistent machine. In Section 4, we describe an implementation of spin locks for x86, from one version of the Linux kernel, and discuss informally why it is correct with respect to x86-TSO. In Section 5, we define a precise notion of data race for x86 and discuss results showing that programs that use spin locks but are otherwise race-free (except for the races within the lock implementation) do indeed behave as if executing on a sequentially consistent machine.²⁶

To support formal reasoning about programs, a memory model must be integrated with a semantics for machine instructions (a problem which has usually been neglected in the relaxed-memory literature). In previous work^{31, 83} we describe a semantics for core x86 instructions, with several innovations. We take care not to over-sequentialize the memory accesses within each instruction, parameterizing the instruction semantics over parallel and sequential combinators. A single definition, with all the intricacies of flag-setting,

addressing modes, etc., can then be used to generate both an event-based semantics that can be integrated with memory models, and a state-based semantics for sequential programs; the latter enables us to test the semantics against implementations. We also build an instruction decoding function, directly from the vendor documentation, to support reasoning about concrete machine code.

The intended scope of x86-TSO is typical user code and most kernel code: we cover programs using coherent write-back memory, without exceptions, misaligned or mixed-size accesses, "nontemporal" operations (e.g., MOVNTI), self-modifying code, or page-table changes. Within this domain, and together with our earlier instruction semantics, x86-TSO thus defines a complete semantics of programs.

Relaxed memory models play an important role also in the design of high-level concurrent languages such as Java or C++0x, where programs are subject not just to the memory model of the underlying processor but also to reorderings introduced by compiler optimizations. The Java Memory Model²⁴ attempts to ensure that data-race free (DRF) programs are sequentially consistent; all programs satisfy memory safety/security properties; and common compiler optimizations are sound. Unfortunately, as shown by Ševčík and Aspinall,³³ the last goal is not met. In the future, we hope that it will be possible to prove correctness of implementations of language-level memory models above the models provided by real-world processors; ensuring that both are precisely and clearly specified is a first step towards that goal.

2. ARCHITECTURE SPECIFICATIONS

To describe what programmers can rely on, processor vendors document *architectures*. These are loose specifications, claimed to cover a range of past and future processor implementations, which should specify processor behavior tightly enough to enable effective programming, but without unduly constraining future processor designs. For some architectures, the memory-model aspects of these specifications are expressed in reasonably precise mathematics, as in the normative Appendix K of the SPARC v.8 specification.² For x86, however, the vendor architecture specifications are informal prose documents. Informal prose is a poor medium for loose specification of subtle properties, and, as we shall see, such documents are almost inevitably ambiguous and sometimes wrong. Moreover, one cannot test programs above such a vague specification (one can only run programs on particular actual processors), and one cannot use them as criteria for testing processor implementations. In this section, we review the informal-prose Intel and AMD x86 specifications: the Intel 64 and IA-32 Architectures Software Developer's Manual (SDM)⁵ and the AMD64 Architecture Programmer's Manual (APM).³ There have been several versions of these, some differing radically; we contrast them with each other, and with what we have discovered of the behavior of actual processors. In the process we introduce the key discriminating examples.

2.1. Pre-IWP (before Aug. 2007)

Early revisions of the Intel SDM (e.g. rev. 22, Nov. 2006) gave an informal-prose model called "processor ordering,"

unsupported by any examples. It is hard to see precisely what this prose means, especially without additional knowledge or assumptions about the microarchitecture of particular implementations. The uncertainty about x86 behavior that at least some systems programmers had about earlier IA-32 processors can be gauged from an extensive discussion about the correctness of a proposed optimization to a Linux spinlock implementation.¹ The discussion is largely in microarchitectural terms, not just in terms of the specified architecture, and seems to have been resolved only with input from Intel staff. We return to this optimization in Section 4, where we can explain why it is sound with respect to x86-TSO.

2.2. IWP/AMD3.14/x86-CC

In August 2007, an Intel White Paper⁴ (IWP) gave a somewhat more precise model, with 8 informal-prose principles P1–P8 supported by 10 examples (known as litmus tests). This was incorporated, essentially unchanged, into later revisions of the Intel SDM (including rev. 26–28), and AMD gave similar, though not identical, prose and tests in rev. 3.14 of their manual³, vol. 2, §7.2 (AMD3.14). These are essentially causal-consistency models,⁹ and they allow different processors to see writes to independent locations in different orders, as in the IRIW litmus test¹¹ below.⁸ AMD3.14 allows this explicitly, while IWP allows it implicitly, as IRIW is not ruled out by the stated principles. Microarchitecturally, IRIW can arise

IRIW

Proc 0	Proc 1	Proc 2	Proc 3
MOV [x]←1	MOV [y]←1	MOV EAX←[x] MOV EBX←[y]	MOV ECX←[y] MOV EDX←[x]
Forbidden Final State: Proc 2:EAX=1 ∧ Proc 2:EBX=0 ∧ Proc 3:ECX=1 ∧ Proc 3:EDX=0			

from store buffers that are shared between some but not all processors.

However, both require that, in some sense, causality is respected, as in the IWP principle “P5. *In a multiprocessor system, memory ordering obeys causality (memory ordering respects transitive visibility).*”

We used these informal specifications as the basis for a formal model, x86-CC,³¹ for which a key issue was giving a reasonable interpretation to this “causality,” which is not defined in IWP or AMD3.14. Apart from that, the informal specifications were reasonably unambiguous—but they turned out to have two serious flaws.

First, they are arguably rather weak for programmers. In particular, they admit the IRIW behavior above but, under reasonable assumptions on the strongest x86 memory barrier, MFENCE, adding MFENCES would not suffice to recover sequential consistency (instead, one would have to make liberal use of x86 LOCK’d instructions).^{31, §2.12} Here, the specifications seem to be much looser than the behavior of implemented processors: to the best of our knowledge,

⁸ We use Intel assembly syntax throughout except that we use an arrow ← to indicate the direction of data flow, so MOV [x]←1 is a write of 1 to address x and MOV EAX←[x] is a read from address x into register EAX. Initial states are all 0 unless otherwise specified.

and following some testing, IRIW is not observable in practice, even without MFENCES. It appears that some JVM implementations depend on this fact, and would not be correct if one assumed only the IWP/AMD3.14/x86-CC architecture.¹⁵

Second, more seriously, x86-CC and IWP are unsound with respect to current processors. The following example, n6, due to Paul Loewenstein [personal communication, Nov. 2008] shows a behavior that is observable (e.g., on an Intel Core 2

n6

Proc 0	Proc 1
MOV [x]←1 MOV EAX←[x] MOV EBX←[y]	MOV [y]←2 MOV [x]←2
Allowed Final State: Proc 0:EAX=1 ∧ Proc 0:EBX=0 ∧ [x]=1	

duo) but that is disallowed by x86-CC and by any interpretation we can make of IWP principles P1, 2, 4 and 6.^{27, A.5}

To see why this could be allowed by multiprocessors with FIFO store buffers, suppose that first the Proc 1 write of [y]=2 is buffered, then Proc 0 buffers its write of [x]=1, reads [x]=1 from its own store buffer, and reads [y]=0 from main memory, then Proc 1 buffers its [x]=2 write and flushes its buffered [y]=2 and [x]=2 writes to memory, then finally Proc 0 flushes its [x]=1 write to memory.

The AMD3.14 manual is not expressed in terms of a clearly identified set of principles, and the main text (vol. 2, §7.2) leaves the ordering of stores to a single location unconstrained, though elsewhere the manual describes a micro-architecture with store buffers and cache protocols that strongly implies that memory is coherent. In the absence of an analogue of the IWP P6, the reasoning prohibiting n6 does not carry over.

2.3. Intel SDM rev. 29–34 (Nov. 2008–Mar. 2010)

The most recent substantial change to the Intel memory-model specification, at the time of writing, was in revision 29 of the Intel SDM (revisions 29–34 are essentially identical except for the LFENCE text). This is in a similar informal-prose style to previous versions, again supported by litmus tests, but is significantly different to IWP/x86-CC/AMD3.14. First, the IRIW final state above is forbidden,^{5, Example 8–7, vol. 3A} and the previous coherence condition: “P6. *In a multiprocessor system, stores to the same location have a total order*” has been replaced by: “*Any two stores are seen in a consistent order by processors other than those performing the stores*” (we label this P9).

Second, the memory barrier instructions are now included. It is stated that reads and writes cannot pass MFENCE instructions, together with more refined properties for SFENCE and LFENCE

Third, same-processor writes are now explicitly ordered: “*Writes by a single processor are observed in the same order by all processors*” (P10) (we regarded this as implicit in the IWP “P2. *Stores are not reordered with other stores*”).

This revision appears to deal with the unsoundness, admitting the n6 behavior above, but, unfortunately, it is still problematic. The first issue is, again, how to interpret “causality”

as used in P5. The second issue is one of weakness: the new P9 says nothing about observations of two stores by those two processors themselves (or by one of those processors and one other). The following examples (which we call n5 and n4b) illustrate potentially surprising behavior that arguably violates coherence. Their final states are not allowed in x86-CC, are not allowed in a pure store-buffer implementation or in x86-TSO, and we have not observed them on actual processors. However, the principles stated in revisions 29–34 of the Intel SDM appear, presumably unintentionally, to allow them. The AMD3.14 vol. 2, §7.2 text taken alone would allow them, but the implied coherence from elsewhere in the AMD manual would forbid them. These points illustrate

n5

Proc 0	Proc 1
MOV [x]←1 MOV EAX←[x]	MOV [x]←2 MOV EBX←[x]
Forbidden Final State: Proc 0:EAX=2 ∧ Proc 1:EBX=1	

n4b

Proc 0	Proc 1
MOV EAX←[x] MOV [x]←1	MOV ECX←[x] MOV [x]←2
Forbidden Final State: Proc 0:EAX=2 ∧ Proc 1:ECX=1	

once again the difficulty of writing unambiguous and correct loose specifications in informal prose.

2.4. AMD3.15 (Nov. 2009)

In November 2009, AMD produced a new revision, 3.15, of their manuals. The main difference in the memory-model specification is that IRIW is now explicitly forbidden.

Summarizing the key litmus-test differences, we have the

	IWP/x86-CC	3.14	29–34	3.15	Actual processors
IRIW	√*/√	√	×	×	Not observed
n6	×*/×	√*	√*	√*	Observed
n5/n4b	×*/×	×*	√*	×*	Not observed

following, where √ and × entries are explicit in the specification text and starred entries indicate possible deductions, some of which may not have been intended.

There are also many non-differences: tests for which the behaviors coincide in all three cases. We return to these, and go through the other tests from the Intel and AMD documentation, in Section 3.2.

3. OUR x86-TSO PROGRAMMER'S MODEL

Given these problems with the informal specifications, we cannot produce a useful rigorous model by formalizing the “principles” they contain, as we attempted with x86-CC.³¹ Instead, we have to build a reasonable model that is consistent with the given litmus tests, with observed processor behavior, and with what we know of the needs of programmers, the vendors’ intentions, and the folklore in the area.

We emphasize that our aim is a *programmer’s model*, of the allowable behaviors of x86 processors as observed by assembly programs, not of the internal structure of processor implementations, or of what could be observed on hardware interfaces. We present the model in an abstract-machine style to make it accessible, but are concerned only with its external behavior; its buffers and locks are highly abstracted from the microarchitecture of processor implementations.

The fact that store buffering is observable, as in the SB and n6 examples, but IRIW is not (and IRIW is explicitly forbidden in the SDM revs. 29–34 and AMD3.15), together with additional tests that prohibit many other reorderings, strongly suggests that, apart from store buffering, all processors share the same view of memory. Moreover, different processors or hardware threads do not observably share store buffers. This is in sharp contrast to x86-CC, where each processor has a separate view order of its memory accesses and other processors’ writes. To the best of our knowledge, for the usual write-back memory, no other aspects of the microarchitecture (the out-of-order execution, cache hierarchies and protocols, interconnect topology, and so on) are observable to the programmer, except in so far as they affect performance.

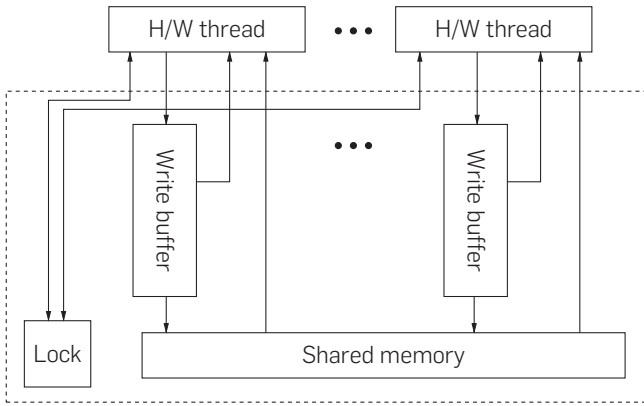
This is broadly similar to the SPARC Total Store Ordering (TSO) memory model,^{2,32} which is essentially an axiomatic description of the behavior of store-buffer multiprocessors. Accordingly, we have designed a TSO-like model for x86, called x86-TSO.²⁷ It is defined mathematically in two styles: an abstract machine with explicit store buffers and an axiomatic model that defines valid executions in terms of memory orders; they are formalized in HOL4²⁰ and are proved equivalent. The abstract machine conveys the programmer-level operational intuition behind x86-TSO; we describe it informally in the next subsection. The axiomatic model supports constraint-based reasoning about example programs (e.g. by our `memevents` tool in Section 3.3); it is similar to that of SPARCV8,^{2,App.K} but we also deal with x86 CISC instructions with multiple memory accesses and with x86 barriers and atomic (or *LOCK’d*) instructions. The x86 supports a range of atomic instructions: one can add a LOCK prefix to many read–modify–write instructions (ADD, INC, etc.), and the XCHG instruction is implicitly LOCK’d. There are three main memory barriers: MFENCE, SFENCE and LFENCE.

3.1. The abstract machine

Our programmer’s model of a multiprocessor x86 system is illustrated in Figure 1. At the top of the figure are a number of hardware threads, each corresponding to a single in-order stream of instruction execution. (In this programmer’s model there is no need to consider physical processors explicitly; it is the hardware threads that correspond to the Proc N columns in the tests we give.) They interact with a storage subsystem, drawn as the dotted box.

The state of the storage subsystem comprises a shared memory that maps addresses to values, a global lock to indicate when a particular hardware thread has exclusive access to memory, and one store buffer per hardware

Figure 1. x89-TSO block diagram.



thread.

The behavior of the storage subsystem is described in more detail below, but the main points are:

- ▶ The store buffers are FIFO and a reading thread must read its most recent buffered write, if there is one, to that address; otherwise reads are satisfied from shared memory.
- ▶ An MFENCE instruction flushes the store buffer of that thread.
- ▶ To execute a LOCK'd instruction, a thread must first obtain the global lock. At the end of the instruction, it flushes its store buffer and relinquishes the lock. While the lock is held by one thread, no other thread can read.
- ▶ A buffered write from a thread can propagate to the shared memory at any time except when some other thread holds the lock.

More precisely, the possible interactions between the threads and the storage subsystem are described by the following *events*:

- ▶ $W_p[a]=v$, for a write of value v to address a by thread p
- ▶ $R_p[a]=v$, for a read of v from a by thread p
- ▶ F_p , for an MFENCE memory barrier by thread p
- ▶ L_p , at the start of a LOCK'd instruction by thread p
- ▶ U_p , at the end of a LOCK'd instruction by thread p
- ▶ τ_p , for an internal action of the storage subsystem, propagating a write from p 's store buffer to the shared memory

For example, suppose a particular hardware thread p has come to the instruction INC [56] (which adds 1 to the value at address 56), and p 's store buffer contains a single write to 56, of value 0. In one execution we might see read and write events, $R_p[56]=0$ and $W_p[56]=1$, followed by two τ_p events as the two writes propagate to shared memory. Another execution might start with the write of 0 propagating to shared memory, where it could be overwritten by another thread. Executions of LOCK;INC [56] would be similar but bracketed by L_p and U_p events.

The behavior of the storage subsystem is specified by the following rules, where we define a hardware thread to be *blocked* if the storage subsystem lock is taken by another hardware thread, i.e., while another hardware thread is

executing a LOCK'd instruction.

1. $R_p[a]=v$: p can read v from memory at address a if p is not blocked, there are no writes to a in p 's store buffer, and the memory does contain v at a .
2. $R_p[a]=v$: p can read v from its store buffer for address a if p is not blocked and has v as the newest write to a in its buffer.
3. $W_p[a]=v$: p can write v to its store buffer for address a at any time.
4. τ_p : if p is not blocked, it can silently dequeue the oldest write from its store buffer and place the value in memory at the given address, without coordinating with any hardware thread.
5. F_p : if p 's store buffer is empty, it can execute an MFENCE (note that if a hardware thread encounters an MFENCE instruction when its store buffer is not empty, it can take one or more τ_p steps to empty the buffer and proceed, and similarly in 7 below).
6. L_p : if the lock is not held, it can begin a LOCK'd instruction.
7. U_p : if p holds the lock, and its store buffer is empty, it can end a LOCK'd instruction.

Technically, the formal versions of these rules²⁷ define a labeled transition system (with the events as labels) for the storage subsystem, and we define the behavior of the whole system as a parallel composition of that and transition systems for each thread, synchronizing on the non- τ labels as in CCS.²⁵

Additionally, we tentatively impose a progress condition, that each memory write is eventually propagated from the relevant store buffer to the shared memory. This is not stated in the documentation and is hard to test. We are assured that it holds at least for AMD processors.

For write-back cacheable memory, and the fragment of the instruction set that we consider, we treat LFENCE and SFENCE semantically as no-ops. This follows the Intel and AMD documentation, both of which imply that these fences do not order store/load pairs which are the only reorderings allowed in x86-TSO. Note, though, that elsewhere it is stated that the Intel SFENCE flushes the store buffer.^{5, vol.3A, §11.10}

3.2. Litmus tests

For our introductory SB example from Section 1, x86-TSO permits the given behavior for the same reasons as set forth there. For each of the examples in Section 2 (IRIW, n6, and n5/n4b), x86-TSO permits the given final state if and only if it is observable in our testing of actual processors, i.e., for IRIW it is forbidden (in contrast to IWP and AMD3.14), for n6 it is allowed (in contrast to IWP), and for n5/n4b it is forbidden (in contrast to the Intel SDM rev. 29–34). For all the other relevant tests from the current Intel and AMD manuals the stated behavior agrees with x86-TSO. We now go through Examples 8–1 to 8–10 from rev. 34 of the Intel SDM, and the three other tests from AMD3.15, and explain the x86-TSO behavior in each case.

For completeness we repeat the Intel SDM short

descriptions of these tests, e.g. “stores are not reordered with other stores,” but note that “not reordered with” is not defined there and is open to misinterpretation.^{27, §3.2}

EXAMPLE 8-1. STORES ARE NOT REORDERED WITH OTHER STORES.

Proc 0	Proc 1
MOV [x]←1 MOV [y]←1	MOV EAX←[y] MOV EBX←[x]
Forbidden Final State: Proc 1:EAX=1 ∧ Proc 1:EBX=0	

This test implies that the writes by Proc 0 are seen in order by Proc 1’s reads, which also execute in order. x86-TSO forbids the final state because Proc 0’s store buffer is FIFO, and Proc 0 communicates with Proc 1 only through shared memory.

EXAMPLE 8-2. STORES ARE NOT REORDERED WITH OLDER LOADS.

Proc 0	Proc 1
MOV EAX←[x] MOV [y]←1	MOV EBX←[y] MOV [x]←1
Forbidden Final State: Proc 0:EAX=1 ∧ Proc 1:EBX=1	

x86-TSO forbids the final state because reads are never delayed.

EXAMPLE 8-3. LOADS MAY BE REORDERED WITH OLDER STORES. This test is just the SB example from Section 1, which x86-TSO permits. The third AMD test (amd3) is similar but with additional writes inserted in the middle of each thread, of 2 to x and y respectively.

EXAMPLE 8-4. LOADS ARE NOT REORDERED WITH OLDER STORES TO THE SAME LOCATION.

Proc 0
MOV [x]←1 MOV EAX←[x]
Required Final State: Proc 0:EAX=1

x86-TSO requires the specified result because reads must check the local store buffer.

EXAMPLE 8-5. INTRA-PROCESSOR FORWARDING IS ALLOWED. This test is similar to Example 8-3.

EXAMPLE 8-6. STORES ARE TRANSITIVELY VISIBLE.

Proc 0	Proc 1	Proc 2
MOV [x]←1	MOV EAX←[x] MOV [y]←1	MOV EBX←[y] MOV ECX←[x]
Forbidden Final State: Proc 1:EAX=1 ∧ Proc 2:EBX=1 ∧ Proc 2:ECX=0		

x86-TSO forbids the given final state because otherwise the Proc 2 constraints imply that y was written to shared memory before x. Hence the write to x must be in Proc 0’s store buffer (or the instruction has not executed), when the write to y is initiated. Note that this test contains the only mention of “transitive visibility” in the Intel SDM, leaving its meaning unclear.

EXAMPLE 8-7. STORES ARE SEEN IN A CONSISTENT ORDER BY OTHER PROCESSORS. This test rules out the IRIW behavior as described in Section 2.2. x86-TSO forbids the given final state because the Proc 2 constraints imply that x was written to shared memory before y whereas the Proc 3 constraints imply that y was written to shared memory before x.

EXAMPLE 8-8. LOCKED INSTRUCTIONS HAVE A TOTAL ORDER. This is the same as the IRIW Example 8-7 but with LOCK’ d instructions for the writes; x86-TSO forbids the final state for the same reason as above.

EXAMPLE 8-9. LOADS ARE NOT REORDERED WITH LOCKS.

Proc 0	Proc 1
XCHG [x]←EAX MOV EBX←[y]	XCHG [y]←ECX MOV EDX←[x]
Initial state: Proc 0:EAX=1 ∧ Proc 1:ECX=1 (elsewhere 0)	
Forbidden Final State: Proc 0:EBX=0 ∧ Proc 1:EDX=0	

This test indicates that locking both writes in Example 8-3 would forbid the nonsequentially consistent result. x86-TSO forbids the final state because LOCK’ d instructions flush the local store buffer. If only one write were LOCK’ d (say the write to x), the Example 8-3 final state would be allowed as follows: on Proc 1, buffer the write to y and execute the read x, then on Proc 0 write to x in shared memory then read from y.

Proc 0	Proc 1
XCHG [x]←EAX MOV [y]←1	MOV EBX←[y] MOV ECX←[x]
Initial state: Proc 0:EAX=1 (elsewhere 0)	
Forbidden Final State: Proc 1:EBX=1 ∧ Proc 1:ECX=0	

EXAMPLE 8-10. STORES ARE NOT REORDERED WITH LOCKS. This is implied by Example 8-1, as we treat the memory writes of LOCK’ d instructions as stores.

Proc 0	Proc 1
MOV [x]←1 MFENCE MOV EAX←[y]	MOV [y]←1 MFENCE MOV EBX←[x]
Forbidden Final State: Proc 0:EAX=0 ∧ Proc 1:EBX=0	

TEST AMD5.

For x86-TSO, this test has the same force as Example 8.8, but using MFENCE instructions to flush the buffers instead of LOCK’ d instructions. The tenth AMD test is similar. None of the Intel litmus tests include fence instructions.

In x86-TSO adding MFENCE between every instruction would clearly suffice to regain sequential consistency (though obviously in practice one would insert fewer barriers), in contrast to IWP/x86-CC/AMD3.14.

3.3. Empirical testing

To build confidence that we have a sound model of the behavior of actual x86 processors we have tested the

correspondence between them in various ways.

First, for the memory model, we have a `litmus` tool that takes a litmus test (essentially as given in this paper) and builds a C program with embedded assembly to run the test repeatedly to try to produce all possible results, taking care to synchronize the different threads and with some randomization of memory usage. We have run these on the Intel and AMD processors that we have access to. The results can be compared with the output of a `memevents` tool that takes such tests and computes the set of all possible executions allowed by the x86-TSO model. We use a verified witness checker, extracted from the HOL4 definition of the model, to verify that any executions found are indeed allowed.

The results correspond exactly for all the tests given here and others we have tried, including `amd3`, `n1`,³¹ `n7`,²⁷ the single-XCHG variant of Example 8–9, and an unfenced variant of RWC.¹¹ In general, though, there may be tests where x86-TSO allows some final state that cannot be observed in practice, perhaps because `litmus` does not drive the processor into the correct internal state (of store buffers, cache lines, etc.) to exhibit it, or perhaps because the particular implementations we tested cannot exhibit it. For example, we have only seen `amd3` on a four-processor ($\times 2$ hyperthread) machine and only very rarely, 4 out of $3.2e9$ times. Testing, especially this black-box testing of a complex and time-dependent system, is obviously subject to the usual limitations; it cannot conclusively prove that some outcome is not possible.

Second, for the behavior of individual instructions, we have an `x86sem` tool that generates random instances of instructions, runs them on an actual machine, and generates a HOL4 conjecture relating the memory and register state before and after. These conjectures are then automatically verified, by a HOL4 script, for the 4600 instances that we tried.

4. A LINUX x86 SPINLOCK IMPLEMENTATION

In Section 2.1, we mentioned the uncertainty that arose in a discussion on a particular optimization for Linux spin-locks.¹ In this section, we present a spinlock from the Linux kernel (version 2.6.24.7) that incorporates the proposed optimization, as an example of a small but nontrivial concurrent programming idiom. We show how one can reason about this code using the x86-TSO programmer’s model, explaining in terms of the model why it works and why the optimization is sound—thus making clear what (we presume) the developer’s informal reasoning depended on. For accessibility we do this in prose, but the argument could easily be formalized as a proof.

The implementation comprises code to acquire and release a spinlock. It is assumed that these are properly bracketed around critical sections and that spinlocks are not mutated by any other code.

On entry the address of spinlock is in register EAX and the spinlock is unlocked iff its value is 1	
acquire: LOCK;DEC	[EAX] ; LOCK’d decrement of [EAX]
JNS	enter ; branch if [EAX] was ≥ 1
spin: CMP	[EAX],0 ; test [EAX]
JLE	spin ; branch if [EAX] was ≤ 0
JMP	acquire ; try again
enter:	; the critical section starts here
release: MOV	[EAX] $\leftarrow 1$

A spinlock is represented by a signed integer which is 1 if the lock is free and 0 or less if the lock is held. To acquire a lock, a thread atomically decrements the integer (which will not wrap around assuming there are fewer than 2^{31} hardware threads). If the lock was free, it is now held and the thread can proceed to the critical section. If the lock was held, the thread loops, waiting for it to become free. Because there might be multiple threads waiting for the lock, once it is freed, each waiting thread must again attempt to enter through the LOCK’d decrement. To release the lock, a thread simply sets its value to 1.

The optimization in question made the releasing MOV instruction not LOCK’d (removing a LOCK prefix and hence letting the releasing thread proceed without flushing its buffer).

For example, consider a spinlock at address x and let y be another shared memory address. Suppose that several threads want to access y , and that they use spinlocks to ensure mutual exclusion. Initially, no one has the lock and $[x] = 1$. The first thread t to try to acquire the lock atomically decrements x by 1 (using a LOCK prefix); it then jumps into the critical section. Because a store buffer flush is part of LOCK’d instructions, $[x]$ will be 0 in shared memory after the decrement.

Now if another thread attempts to acquire the lock, it will not jump into the critical section after performing the atomic decrement, since x was not 1. It will thus enter the spin loop. In this loop, the waiting thread continually reads the value of x until it gets a positive result.

Returning to the original thread t , it can read and write y inside of its critical section while the others are spinning. These writes are initially placed in t ’s store buffer, and some may be propagated to shared memory. However, it does not matter how many (if any) are written to main memory, because (by assumption) no other thread is attempting to read (or write) y . When t is ready to exit the critical section, it releases the lock by writing the value 1 to x ; this write is put in t ’s store buffer. It can now continue after the critical section (in the text below, we assume it does not try to reacquire the lock).

If the releasing MOV had the LOCK prefix then all of the buffered writes to y would be sent to main memory, as would the write of 1 to x . Another thread could then acquire the spinlock.

However, since it does not, the other threads continue to spin until the write setting x to 1 is removed from t ’s write buffer and sent to shared memory at some point in the future. At that point, the spinning threads will read 1 and restart the acquisition with atomic decrements, and another thread could enter its critical section. However, because t ’s write buffer is emptied in FIFO order, any writes to y from within t ’s critical section must have been propagated to shared memory (in order) before the write to x . Thus, the next thread to enter a critical section will not be able to see y in an inconsistent state.

5. DATA-RACE FREEDOM

To make a relaxed-memory architecture usable for large-scale programming, it is highly desirable (perhaps essential)

to identify programming idioms which ensure that one can reason in terms of a traditional interleaving model of concurrency, showing that any relaxed memory execution is equivalent to one that is possible above a sequentially consistent memory model. One common idiom with this property is *data-race freedom*. Informally, a program has a data race if multiple threads can access the same location (where at least one is writing to the location) without a synchronization operation separating the accesses. Programs where every shared access is in a critical section are one common example of DRF programs.

A variety of relaxed models, both for processors and for programming languages, have been proved to support sequentially consistent semantics for DRF programs.^{8, 9, 10, 12, 16, 23} Saraswat et al.³⁰ call supporting sequentially consistent semantics for DRF programs the “fundamental property” of a relaxed memory model, and indeed memory models have sometimes been defined in these terms.⁶ However, for a processor architecture, we prefer to define a memory model that is applicable to arbitrary programs, to support reasoning about low-level code, and have results about well-behaved programs as theorems above it.

The details of what constitutes a data race, or a synchronization operation, vary from model to model. For x86-TSO, we define two events on different threads to be *competing* if they access the same address, one is a write, and the other is a read (for aligned x86 accesses, it is not necessary to consider write/write pairs as competing). We say that a program is data race free if it is impossible for a competing read/write pair to execute back-to-back. Critically, we require this property only of sequentially consistent executions (equivalently, the x86-TSO executions where store buffers are always flushed immediately after each write).

We have proved that x86-TSO supports interleaving semantics for DRF programs. However, this theorem alone is not often useful, because most programs *do* contain data races at this level of abstraction. For example, the read in the spin loop of Section 4’s spinlock races with the write in the release. We have, therefore, identified an extended notion of data race freedom that the spinlock code does satisfy, and we have used it to prove that, for well-synchronized programs using the spinlock, every x86-TSO execution has an equivalent sequentially consistent execution.²⁶

Thus, the relaxed nature of x86-TSO is provably not a concern for low-level systems code that uses spinlocks to synchronize. Extending this result to other synchronization primitives, and to code compiled from high-level languages, is a major topic for future work.

6. RELATED WORK

There is an extensive literature on relaxed memory models, but most of it does not address x86. We touch here on some of the most closely related work.

There are several surveys of weak memory models, including those by Adve and Gharachorloo,⁶ Luchango,²³ and Higham et al.¹⁹ The latter, in particular, formalizes a range of models, including a TSO model, in both operational and axiomatic styles, and proves equivalence results. Their axiomatic TSO model is rather closer to the operational style


than ours is, and is idealized rather than x86-specific. Park and Dill²⁸ verify programs by model checking them directly above TSO. Burckhardt and Musuvathi^{13, APP. A} also give operational and axiomatic definitions of a TSO model and prove equivalence, but only for finite executions. Their models treat memory reads and writes and barrier events, but lack instruction semantics and LOCK’d instructions with multiple events that happen atomically. Hangal et al.¹⁸ describe the Sun TSOtool, checking the observed behavior of pseudo-randomly generated programs against a TSO model. Roy et al.²⁹ describe an efficient algorithm for checking whether an execution lies within an approximation to a TSO model, used in Intel’s Random Instruction Test (RIT) generator. Loewenstein et al.²² describe a “golden memory model” for SPARC TSO, somewhat closer to a particular implementation microarchitecture than the abstract machine we give in Section 3, that they use for testing implementations. They argue that the additional intensional detail increases the effectiveness of simulation-based verification. Boudol and Petri¹² give an operational model with hierarchical write buffers (thereby permitting IRIW behaviors), and prove sequential consistency for DRF programs. Burckhardt et al.¹⁴ define an x86 memory model based on IWP.⁴ The mathematical form of their definitions is rather different to our axiomatic and abstract-machine models, using rewrite rules to reorder or eliminate memory accesses in sets of traces. Their model validates the 10 IWP tests and also some instances of IRIW (depending on how parallel compositions are associated), so it will not coincide with x86-TSO or x86-CC. Saraswat et al.³⁰ also define memory models in terms of local reordering, and prove a DRF theorem, but focus on high-level languages.

7. CONCLUSION

We have described x86-TSO, a memory model for x86 processors that does not suffer from the ambiguities, weaknesses, or unsoundnesses of earlier models. Its abstract-machine definition should be intuitive for programmers, and its equivalent axiomatic definition supports the `memevents` exhaustive search and permits an easy comparison with related models; the similarity with SPARCv8 suggests x86-TSO is strong enough to program above. This work highlights the clarity of mathematically rigorous definitions, in contrast to informal prose, for subtle loose specifications.

We do not speak for any x86 vendor, and it is, of course, entirely possible that x86-TSO is not a good description of some existing or future x86 implementation (we would be very interested to hear of any such example). Nonetheless, we hope that this will clarify the semantics of x86 architectures as they exist, for systems programmers, hardware developers, and those working on the verification of concurrent software.

Acknowledgments

We thank Luc Maranget for his work on `memevents` and `litmus`, Tom Ridge, Thomas Braibant and Jade Alglave for their other work on the project, and Hans Boehm, David Christie, Dave Dice, Doug Lea, Paul Loewenstein, and Gil Neiger for helpful remarks. We acknowledge funding from EPSRC grants EP/F036345 and EP/H005633 and ANR grant ANR-06-SETI-010-02. 

References

- Linux kernel mailing list, thread "spin_unlock optimization (i386)", 119 messages, Nov. 20–Dec. 7, 1999, <http://www.gossamer-threads.com/lists/engine?post=105365;list=linux>. Accessed 2009/11/18.
- The SPARC Architecture Manual, V. 8*. SPARC International, Inc., 1992. Revision SAV080SI9308. <http://www.sparc.org/standards/V8.pdf>.
- AMD64 Architecture Programmer's Manual (3 vols)*. Advanced Micro Devices, Sept. 2007. rev. 3.14.
- Intel 64 architecture memory ordering white paper, 2007. Intel Corporation. SKU 318147-001.
- Intel 64 and IA-32 Architectures Software Developer's Manual (5 vols)*. Intel Corporation, Mar. 2010. rev. 34.
- Adve, S. Gharachorloo, K. Shared memory consistency models: A tutorial. *IEEE Comput.* 29, 12 (Dec. 1996), 66–76.
- Adve, S.V., Boehm, H.-J. Memory models: A case for rethinking parallel languages and hardware. *C. ACM*. To appear.
- Adve, S.V., Hill, M.D. A unified formalization of four shared-memory models. *IEEE Trans. Parallel Distrib. Syst.* 4, 6 (1993), 613–624.
- Ahamad, M., Neiger, G., Burns, J., Kohli, P., Hutto, P. Causal memory: Definitions, implementation, and programming. *Distrib. Comput.* 9, 1 (1995), 37–49.
- Aspinall, D., Ševčík J. Formalising Java's data race free guarantee. In *Proc. TPHOLS, LNCS 4732* (2007), 22–37.
- Boehm, H.-J., Adve, S. Foundations of the C++ concurrency memory model. In *Proceedings of PLDI* (2008).
- Boudol, G., Petri, G. Relaxed memory models: An operational approach. In *Proceedings of POPL*, 2009.
- Burckhardt, S., Musuvathi, M. Effective program verification for relaxed memory models. Technical Report MSR-TR-2008-12, Microsoft Research, 2008. Conference version in *Proceedings of CAV, LNCS 5123* (2008).
- Burckhardt, S., Musuvathi, M., Singh, V. Verifying compiler transformations for concurrent programs, Jan. 2009. Technical report MSR-TR-2008-171.
- Dice, D. Java memory model concerns on Intel and AMD systems. http://blogs.sun.com/dave/entry/java_memory_model_concerns_on, Jan. 2008.
- Friedman, R. Consistency conditions for distributed shared memories. *Israel Institute of Technologie*, 1994.
- Gharachorloo, K. Memory consistency models for shared-memory multiprocessors. *WRL Res. Rep.* 95, 9 (1995).
- Hangal, S., Vahia, D., Manovit, C., Lu, J.-Y.J., Narayanan, S. TSOtool: A program for verifying memory systems using the memory consistency model. In *Proceedings of ISCA* (2004), 114–123.
- Higham, L., Kawash, J., Verwaal, N. Weak memory consistency models part I: Definitions and comparisons. Technical Report 98/612/03, Department of Computer Science, *The University of Calgary, January*, 1998. Full version of a paper in PDCS 1997.
- The HOL 4 system. <http://hol.sourceforge.net/>.
- Lampert, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comput.* C-28, 9 (1979), 690–691.
- Loewenstein, P.N., Chaudhry, S., Cypher, R., Manovit, C. Multiprocessor memory model verification. In *Proceedings of AFM (Automated Formal Methods)* (Aug. 2006). FLCo workshop. <http://fm.csl.sri.com/AFM06/>.
- Luchango, V.M. *Memory consistency models for high-performance distributed computing*. PhD thesis, MIT, 2001.
- Manson, J., Pugh, W., Adve, S. The Java memory model. In *Proceedings of POPL* (2005).
- Milner, R. *Communication and Concurrency*. Prentice Hall International, 1989.
- Owens, S. Reasoning about the implementation of concurrency abstractions on x86-TSO. In *Proceedings of ECOOP* (2010). To appear.
- Owens, S., Sarkar, S., Sewell, P. A better x86 memory model: x86-TSO. In *Proceedings of TPHOLS, LNCS 5674* (2009), 391–407. Full version as Technical Report UCAM-CL-TR-745, Univ. of Cambridge.
- Park, S., Dill, D.L. An executable specification and verifier for relaxed memory order. *IEEE Trans. Comput.* 48, 2 (1999), 227–235.
- Roy, A., Zeisset, S., Fleckenstein, C.J., Huang, J.C. Fast and generalized polynomial time memory consistency verification. In *CAV* (2006), 503–516.
- Saraswat, V., Jagadeesan, R., Michael, M., von Praun, C. A theory of memory models. In *Proceedings of PPoPP* (2007).
- Sarkar, S., Sewell, P., Zappa Nardelli, F., Owens, S., Ridge, T., Braibant, T., Myreen, M., Alglave, J. The semantics of x86-CC multiprocessor machine code. In *Proceedings of POPL 2009* (Jan. 2009).
- Sindhu, P.S., Frailong, J.-M., Cekleov, M. Formal specification of memory models. In *Scalable Shared Memory Multiprocessors*, Kluwer, 1991, 25–42.
- Ševčík, J., Aspinall, D. On validity of program transformations in the Java memory model. In *ECOOP* (2008), 27–51.

Peter Sewell (<http://www.cl.cam.ac.uk/~pes20/>), University of Cambridge.

Susmit Sarkar (<http://www.cl.cam.ac.uk/~ss726/>), University of Cambridge.

Scott Owens (<http://www.cl.cam.ac.uk/~so294/>), University of Cambridge.

Francesco Zappa Nardelli (<http://www.moscova.inria.fr/~zappa/>), INRIA.

Magnus O. Myreen (<http://www.cl.cam.ac.uk/~mom22/>), University of Cambridge.

© 2010 ACM 0001-0782/10/0700 \$10.00

Announcing ACM's Newly Improved Career & Job Center!

Are you looking for your next IT job? Do you need Career Advice?
Visit ACM's newly enhanced career resource at:

<http://www.acm.org/careercenter>



The ACM Career & Job Center offers ACM members a host of benefits including:

- A highly targeted focus on job opportunities in the computing industry
- Access to hundreds of corporate job postings
- Resume posting keeping you connected to the employment market while letting you maintain full control over your confidential information
- An advanced Job Alert system that notifies you of new opportunities matching your criteria
- Career coaching and guidance from trained experts dedicated to your success
- A content library of the best career articles compiled from hundreds of sources, and much more!

The ACM Career & Job Center is the perfect place to begin searching for your next employment opportunity!

Visit today at <http://www.acm.org/careercenter>



Association for Computing Machinery

Advancing Computing as a Science & Profession

Technical Perspective

Technology Scaling Redirects Main Memories

By Mary Jane Irwin

AS PREDICTED BY Intel's Gordon Moore in 1965, based on his observation of the scaling of several generations of silicon technology at the time, the number of transistors that can be integrated on one die continues to double approximately every two years. Amazing to some, Moore's Law has prevailed for 45 years and is expected to continue for several more generations. Transistor feature size and die integration capacity projections from the International Technology Roadmap for Semiconductors (ITRS) roadmap is shown in the accompanying table here.

These faster and more abundant transistors have been exploited by computer engineers to build processors that double in performance about every two years. Up until the beginning of this decade, that was done through faster clock speeds and clever architectural enhancements. Many of these architectural enhancements were directed at tackling the "memory wall," which still plagues us today. Early in this decade, we ran into the "power wall" that dramatically slowed the increase in clock speeds. Since then, we are still seeing performance double^a every two years, but now it's through having more cores (running at only modestly faster clock rates) on one die since technology scaling provides all of those additional transistors.

Another key component on the motherboard affected by technology scaling is the main memory, traditionally built out of dynamic random access memory (DRAM) parts. DRAMs have been doubling in capacity every two to three years while their access latency has improved about 7% per year. However, processors speeds still leave main memories in the dust—with the processors having to wait 100 or more cycles to get information back from

a But one only really gets double the performance if they can figure out how to keep all of those cores busy.

Projections for transistor size and die integration capacity.

Year	2004	2006	2008	2010	2012
Feature size (nm)	90	65	45	32	22
Integration Capacity (BT)	2	4	6	16	32

main memory—hence, the focus by architects on cache memory systems that tackle this "memory wall." And multi-core parts put even more pressure on the DRAM, demanding more capacity, lower latencies, and better bandwidth.

As pointed out in the following paper by Lee, Ipek, Mutlu, and Burger, DRAM memory scaling is in jeopardy, primarily due to reliability issues. The storage mechanism in DRAMs, charge storage and maintenance in a capacitor, requires inherently unscalable charge placement and control. Flash memories, which have the advantage of being nonvolatile, have their own scaling limitations. Thus, the search for new main memory technologies has begun.

The authors make a case for phase change memories (PCMs) that are nonvolatile and can scale below 40nm. PCMs store state by forcing a phase change in their storage element (for example, chalcogenide) to a high resistance state (so storing a "0") or to a low resistance state (so storing a "1"). Fortunately, programming current scales linearly. However, PCMs do not come without their disadvantages: read and, especially, write latencies several times slower than DRAMs, write energies several times larger than DRAMs, and, like Flash, a limited lifetime directly related to the number of writes to a memory location.

This paper is a wonderful illustration of the way computer architects can work around the limitations of the technology with clever architectural enhancements—turning lemons into lemonade. By using an area-neutral

memory buffer reorganization, the authors are able to reduce application execution time from 1.6X to only 1.2X relative to a DRAM-based system and memory array energy from 2.2X to 1.0X also relative to a DRAM-based system. They use multiple, narrower memory buffers, which reduces the number of expensive (in terms of both area and power) sense amplifiers and focus on application performance rather than the performance of an individual memory cell.

The authors also describe their investigation of the trade-offs between buffer row widths and the number of rows. To tackle the PCM's lifetime limitation, the authors propose using partial writes to reduce the number of writes to the PCM by tracking dirty data from the L1 caches to the memory banks. With this approach, they can improve PCM lifetimes from hundreds of hours to nearly 10 years, assuming present $1\text{E}+08$ to $1\text{E}+12$ writes per bit for a 32nm PCM cell.

The paper concludes with some suggestions as to how the use of a nonvolatile main memory would change the computing landscape: instantaneous system boot/hibernate, cheaper checkpointing, stronger safety guarantees for file system. Now, if only someone could figure out a way to dramatically improve memory to processor bandwidth. □

Mary Jane Irwin (mji@cse.psu.edu) is Evan Pugh Professor and A. Robert Noll Chair in Engineering in the Department of Computer Science and Engineering at Penn State University, University Park, PA.

Phase Change Memory Architecture and the Quest for Scalability

By Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger

Abstract

Memory scaling is in jeopardy as charge storage and sensing mechanisms become less reliable for prevalent memory technologies, such as dynamic random access memory (DRAM). In contrast, phase change memory (PCM) relies on programmable resistances, as well as scalable current and thermal mechanisms. To deploy PCM as a DRAM alternative and to exploit its scalability, PCM must be architected to address relatively long latencies, high energy writes, and finite endurance.

We propose architectural enhancements that address these limitations and make PCM competitive with DRAM. A baseline PCM system is 1.6× slower and requires 2.2× more energy than a DRAM system. Buffer reorganizations reduce this delay and energy gap to 1.2× and 1.0×, using narrow rows to mitigate write energy as well as multiple rows to improve locality and write coalescing. Partial writes mitigate limited memory endurance to provide more than 10 years of lifetime. Process scaling will further reduce PCM energy costs and improve endurance.

1. INTRODUCTION

Memory technology scaling drives increasing density, increasing capacity, and falling price-capability ratios. Memory scaling, a first-order technology objective, is in jeopardy for conventional technologies. Storage mechanisms in prevalent memory technologies require inherently unscalable charge placement and control. In the nonvolatile space, Flash memories must precisely control the discrete charge placed on a floating gate. In volatile main memory, DRAM must not only place charge in a storage capacitor but must also mitigate subthreshold charge leakage through the access device. Given these challenges, solutions for scaling DRAM beyond 40nm are unknown.¹⁷

PCM provides a nonvolatile storage mechanism amenable to process scaling. During writes, an access transistor injects current into the storage material and thermally induces phase change, which is detected as a programmed resistance during reads. PCM, relying on analog current and thermal effects, does not require control over discrete electrons. As technologies scale and heating contact areas shrink, programming current scales linearly. This PCM scaling mechanism has been demonstrated in a 32nm device prototype.¹⁵ As a scalable DRAM alternative, PCM could provide a clear roadmap for increasing main memory density and capacity.

These scalability trends motivate a transition from charge memories to resistive memories. To realize this transition for PCM, we must overcome PCM disadvantages relative to DRAM. Access latencies, although tens of nanoseconds, are several times slower than those of DRAM. At present technology nodes, PCM writes require energy intensive current injection. Moreover, writes induce thermal expansion and contraction within the storage element, degrading injection contacts and limiting endurance to hundreds of millions of writes per cell at current processes. These limitations are significant, which is why PCM is currently positioned only as a Flash replacement; in this market, PCM properties are drastic improvements. For a DRAM alternative, however, we must architect PCM for feasibility in main memory within general-purpose systems.

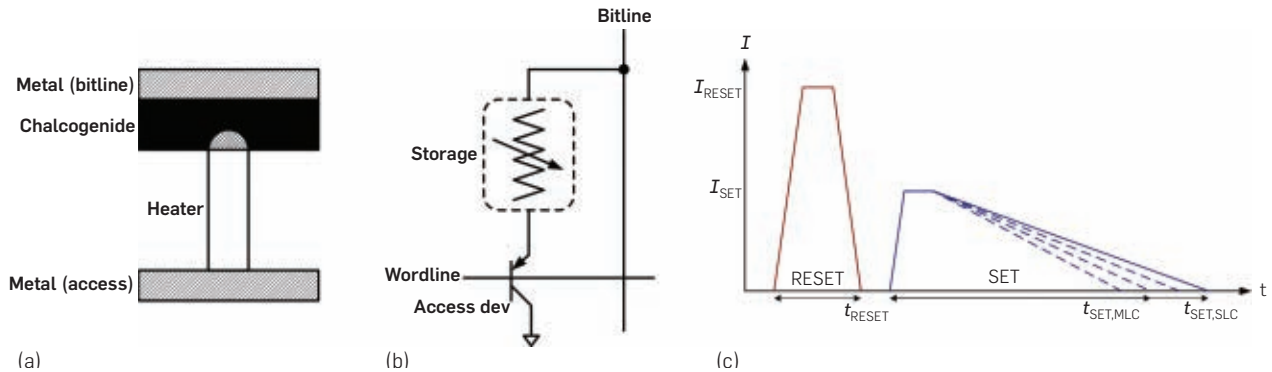
Current prototype designs are not designed to mitigate PCM latencies, energy costs, and finite endurance. This paper rethinks PCM subsystem architecture to bring the technology within competitive range of DRAM. Drawn from a rigorous survey of PCM device and circuit prototypes published within the last 5 years and comparing against modern DRAM memory subsystems, we propose:

- **Buffer Reorganization:** Narrow buffers mitigate high energy PCM writes. Multiple buffer rows exploit locality to coalesce writes, hiding their latency and reducing their energy. Effective PCM buffering reduces application execution time from 1.6× to 1.2× and memory array energy from 2.2× to 1.0×, relative to DRAM-based systems.
- **Partial Writes:** Partial writes track data modifications and write only modified cache lines or words to the PCM array. We expect write coalescing and partial writes to deliver an average memory module lifetime of 11.2 years. PCM endurance is expected to improve by four orders of magnitude when scaled to 32nm.¹⁷

Collectively, these results suggest PCM is a viable DRAM alternative, with architectural solutions providing competitive performance, comparable energy, and feasible lifetimes.

A previous version of this article appears in *Proceedings of the 36th International Symposium on Computer Architecture* (June 2009). Parts of this article appear in *IEEE Micro Top Picks from the Computer Architecture Conferences of 2009* (January/February 2010).

Figure 1. Phase change memory. (a) Storage element with heating resistor and chalcogenide between electrodes. (b) Cell structure with storage element and BJT access device. (c) Reset to an amorphous, high resistance state with a high, short current pulse. Set to a crystalline, low resistance state with moderate, long current pulse. Slope of set current ramp down determines the state in MLC.



2. PCM TECHNOLOGY

Given the still speculative state of PCM technology, researchers have made several different manufacturing and design decisions. We survey device and circuit prototypes published within the last 5 years.¹⁰

2.1. Memory cell

As shown in Figure 1a, the PCM storage element is comprised of two metal electrodes separated by a resistive heater and a chalcogenide, the phase change material. $\text{Ge}_2\text{Sb}_2\text{Te}_5$ (GST) is most commonly used, but other chalcogenides may offer higher resistivity and improve the device's electrical characteristics. Nitrogen doping increases resistivity and lowers programming current while GS may offer faster phase changes.^{4,8}

As shown in Figure 1b, PCM cells are 1T/1R devices, comprised of the resistive storage element and an access transistor. Access is typically controlled by one of three devices: field-effect transistor (FET), bipolar junction transistor (BJT), or diode. In future, FET scaling and large voltage drops across the cell may adversely affect reliability for unselected wordlines.¹⁴ BJTs are faster and expected to scale more robustly without this vulnerability.^{3,14} Diodes occupy smaller areas and potentially enable greater cell densities, but require higher operating voltages.¹¹

Phase changes are induced by injecting current into the resistor junction and heating the chalcogenide. Current and voltage characteristics of the chalcogenide are identical regardless of its initial phase, which lowers programming complexity and latency.⁹ The amplitude and width of the injected current pulse determine the programmed state as shown in Figure 1c.

2.2. Operation

The access transistor injects current into the storage material and thermally induces phase change, which is detected as a programmed resistance during reads. Logical data values are captured by the resistivity of the chalcogenide. A high, short current pulse increases resistivity by abruptly discontinuing current, quickly quenching heat generation, and freezing the chalcogenide into an amorphous state (i.e., reset). A moderate, long current pulse reduces resistivity by ramping down current, gradually cooling the chalcogenide,

and inducing crystal growth (i.e., set). Requiring longer current pulses, set latency determines write performance. Requiring higher current pulses, reset energy determines write power.

Prior to reading the cell, the bitline is precharged to the read voltage. If a selected cell is in a crystalline state, the bitline is discharged with current flowing through the storage element and access transistor. Otherwise, the cell is in an amorphous state, preventing or limiting bitline current.

Cells that store multiple resistance levels might be implemented by leveraging intermediate states, in which the chalcogenide is partially crystalline and partially amorphous.^{3,13} Smaller current slopes (i.e., slow ramp down) produce lower resistances and larger slopes (i.e., fast ramp down) produce higher resistances. Varying slopes induce partial phase transitions changing the size or shape of the amorphous material produced at the contact area, giving rise to resistances between those observed from the fully amorphous or the fully crystalline chalcogenide. The difficulty and high latency of differentiating between a large number of resistances may constrain such multilevel cells (MLC) to a small number of bits per cell.

Wear and Endurance: Writes are the primary wear mechanism in PCM. When injecting current into a volume of phase change material, thermal expansion and contraction degrades the electrode-storage contact, such that programming currents are no longer reliably injected into the cell. Since material resistivity is highly dependent on current injection, current variability causes resistance variability. This greater variability degrades the read window, the difference between programmed minimum and maximum resistance.

Write endurance, the number of writes performed before the cell cannot be programmed reliably, ranges from $1\text{E}+04$ to $1\text{E}+09$. Write endurance depends on process and differs across manufacturers. Relative to Flash, PCM is likely to exhibit greater write endurance by at least two to three orders of magnitude; Flash cells can sustain only $1\text{E}+05$ writes. The ITRS roadmap projects improved endurance of $1\text{E}+12$ writes at 32nm .¹⁷ With wear reduction and leveling techniques, PCM write limits may not be exposed to the system during a memory's lifetime.

2.3. Process scaling

PCM scaling reduces required programming current injected via the electrode-storage contact. As the contact area decreases with feature size, thermal resistivity increases and the volume of phase change material that must be cooled into an amorphous state during a reset to completely block current flow decreases. These effects enable smaller access devices for current injection. Pirovano et al. outline PCM scaling rules,¹⁴ which are confirmed empirically in a survey by Lai.⁹ Specifically, as feature size scales linearly ($1/k$), contact area decreases quadratically ($1/k^2$). Reduced contact area causes resistivity to increase linearly (k), which causes programming current to decrease linearly ($1/k$).

Operational issues arise with aggressive PCM technology scaling. As contact area decreases, lateral thermal coupling may cause programming currents for one cell to influence the states of adjacent cells. Lai's survey of PCM finds these effects negligible in measurement and simulation.⁹ Temperatures fall exponentially with distance from programmed cell, suggesting no appreciable impact from thermal coupling. Increasing resistivity from smaller contact areas may reduce signal strength (i.e., smaller resistivity difference between crystalline and amorphous states). However, these signal strengths are well within the sense circuit capabilities of modern memory architectures.⁹

2.4. Array architecture

As shown in Figure 2, PCM cells might be hierarchically organized into banks, blocks, and subblocks. Despite similarities to conventional memory architectures, PCM-specific issues must be addressed. For example, PCM reads are non-destructive whereas DRAM reads are destructive and require mechanisms to replenish discharged capacitors.

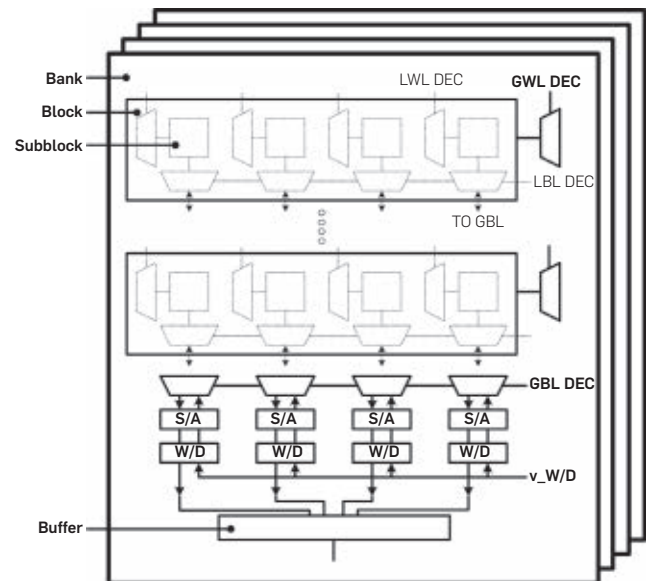
Sense amplifiers detect the change in bitline state when a memory row is accessed. Choice of bitline sense amplifiers affects array read access time. Voltage sense amplifiers are cross-coupled inverters which require differential discharging of bitline capacitances. In contrast, current sense amplifiers rely on current differences to create a differential voltage at the amplifier's output nodes. Current sensing is faster but requires larger circuits.¹⁸

In DRAM, sense amplifiers serve a dual purpose, both sensing and buffering data using cross-coupled inverters. In contrast, we explore PCM architectures with separate sensing and buffering; sense amplifiers drive banks of explicit latches. These latches provide greater flexibility in row buffer organization by enabling multiple buffered rows. However, these latches incur area overheads. Separate sensing and buffering enables multiplexed sense amplifiers. Multiplexing also enables buffer widths narrower than the array width, which is defined by the total number of bitlines. Buffer width is a critical design parameter, determining the required number of expensive current sense amplifiers.

3. A DRAM ALTERNATIVE

We express PCM device and circuit characteristics within conventional DDR timing and energy parameters, thereby quantifying PCM in the context of more familiar DRAM parameters to facilitate a direct comparison.¹⁰

Figure 2. Array architecture. A hierarchical memory organization includes banks, blocks, and subblocks with local, global decoding for row, column addresses. Sense amplifiers (S/A) and word drivers (W/D) are multiplexed across blocks.



We evaluate a four-core chip multiprocessor using the SESC simulator.¹⁶ The 4-way superscalar, out-of-order cores operate at 4.0GHz. This datapath is supported by 32KB, direct-mapped instruction and 32KB, 4-way data L1 caches, which may be accessed in two to three cycles. A 4MB, 8-way L2 cache with 64B lines is shared between the four cores and may be accessed in 32 cycles.

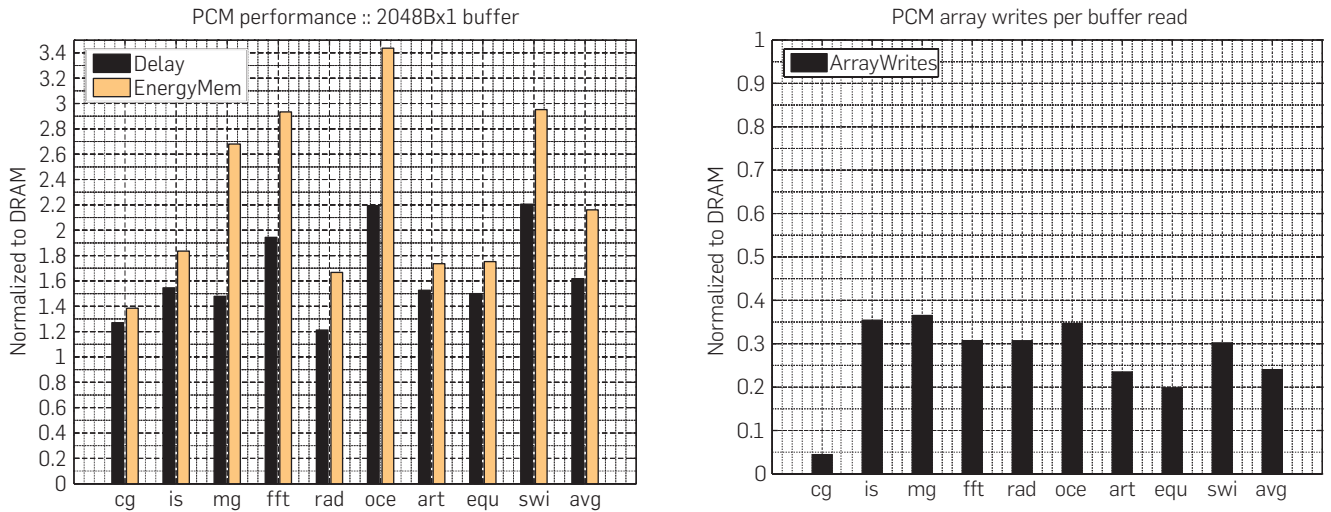
Below the caches is a 400 MHz SDRAM memory subsystem modeled after Micron's DDR2-800 technical specifications.¹² We consider one channel, one rank, and four $\times 16$ chips per rank to achieve the standard 8B interface. Internally, each chip is organized into four banks to facilitate throughput as data are interleaved across banks and accessed in parallel. We model a burst length of eight blocks. The memory controller has a 64-entry transaction queue.

We consider parallel workloads from the SPLASH-2 suite (fft, radix, ocean), SPEC OpenMP suite (art, equake, swim), and NAS parallel benchmarks (cg, is, mg).^{1, 2, 19} Regarding input sets, we use 1 M points for FFT, 514 \times 514 grid for ocean, and 2 M integers for radix. SPEC OpenMP workloads run MinneSpec-Large data set and NAS parallel benchmarks run with Class A problem sizes. Applications are compiled using gcc and Fortran compilers at the -O3 optimization level.

3.1. Baseline comparison

We consider a PCM baseline architecture, which implements DRAM-style buffering with a single 2048B-wide buffer. Figure 3a illustrates end-to-end application performance when PCM replaces DRAM as main memory. Application delay increases with penalties relative to DRAM ranging from 1.2 \times (radix) to 2.2 \times (ocean, swim). On average, we observe a 1.6 \times delay penalty. The energy penalties are larger, ranging from 1.4 \times (cg) to 3.4 \times (ocean), due to the highly expensive array writes required when buffer contents

Figure 3. PCM as a DRAM alternative. (a) Application delay and memory energy. (b) Percentage of buffer evictions that require array writes.



are evicted. On average, we observe a 2.2× energy penalty.

The end-to-end delay and energy penalties are more modest than the underlying technology parameters might suggest. Even memory-intensive workloads mix computation with memory accesses. Furthermore, the long latency, high energy array writes manifest themselves much less often in PCM than in DRAM; nondestructive PCM reads do not require subsequent writes whereas destructive DRAM reads do. Figure 3b indicates only 28% of PCM array reads first require an array write of a dirty buffer.

To enable PCM for use below the lowest level processor cache in general-purpose systems, we must close the delay and energy gap between PCM and DRAM. Nondestructive PCM reads help mitigate underlying delay and energy disadvantages by default. We seek to eliminate the remaining PCM-DRAM differences with architectural solutions. In particular, the baseline analysis considers a single 2048B-wide buffer per bank. Such wide buffering is inexpensive in DRAM, but incurs unnecessary energy costs in PCM given the expensive current injection required when writing buffer contents back into the array.

3.2. Buffer organization

We examine whether PCM subsystems can close the gap with DRAM application performance and memory subsystem energy. To be a viable DRAM alternative, buffer organizations must hide long PCM latencies, while minimizing PCM energy costs.

To achieve area neutrality across buffer organizations, we consider narrower buffers and additional buffer rows. The number of sense amplifiers decreases linearly with buffer width, significantly reducing area as fewer of these large circuits are required. We utilize this area by implementing multiple rows with latches much smaller than the removed sense amplifiers. Narrow widths reduce PCM write energy but negatively impact spatial locality, opportunities for write coalescing, and application performance. However, these penalties may be mitigated by the additional buffer rows.

We consider buffer widths ranging from the original

2048B to 64B, which is the line size of the lowest level cache. We consider buffer rows ranging from the original single row to a maximum of 32 rows. At present, we consider a fully associative buffer and full associativity likely becomes intractable beyond 32 rows. Buffers with multiple rows use a least recently used (LRU) eviction policy implemented in the memory controller.

3.3. Buffer design space

Buffer reorganizations impact the degree of exploited locality and energy costs associated with array reads and writes. Figure 4 illustrates the delay and energy characteristics of the buffer design space for an average of memory-intensive benchmarks. Triangles illustrate PCM and DRAM baselines, which implement a single 2048B buffer. Circles illustrate various buffer organizations. Reorganizing a single, wide

Figure 4. Pareto analysis for PCM buffer organizations.

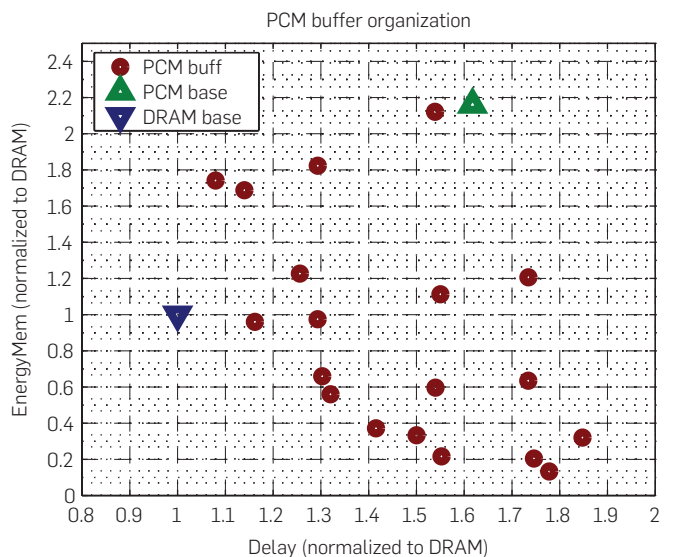
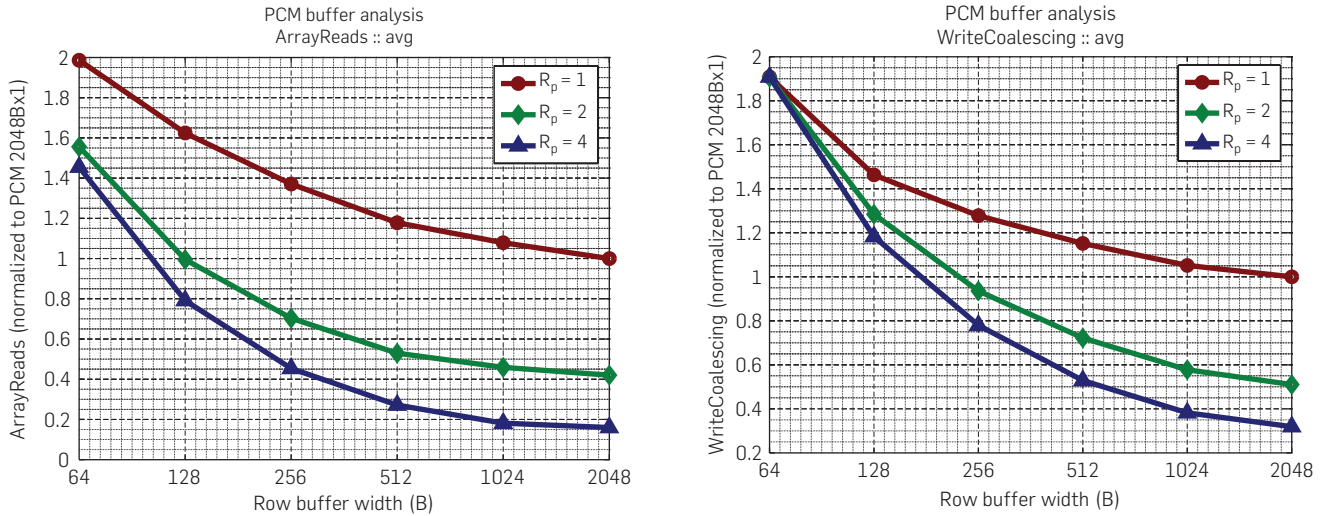


Figure 5. Memory subsystem trends from PCM buffer organizations. (a) Array reads increase sublinearly with buffer width. (b) Array write coalescing opportunities increase with buffer rows.



buffer into multiple, narrow buffers reduce both energy costs and delay. Examining the Pareto frontier, we observe Pareto optima shift PCM delay and energy into the neighborhood of the DRAM baseline. Among these Pareto optima, we observe a knee that minimizes both energy and delay; this organization uses four 512B-wide buffers to reduce PCM delay, energy disadvantages from $1.6\times$, $2.2\times$ to more modest $1.2\times$, $1.0\times$.

The number of array reads is a measure of locality. Figure 5a shows the number of array reads increases very slowly as buffer width decreases exponentially from 2048B to 64B. For a single buffered row ($R_p = 1$), a $32\times$ reduction in buffer width produces only a $2\times$ increase in array reads, suggesting very little spatial locality within wide rows for the memory-intensive workloads we consider. The single row is evicted too quickly after its first access, limiting opportunities for spatial reuse. However, we do observe significant temporal locality. A 2048B-wide buffer with two rows ($R_p = 2$) requires $0.4\times$ the array reads as a 2048B-wide buffer with only a single row ($R_p = 1$).

Writes are coalesced if multiple writes modify the buffer before its contents are evicted to the array. Thus the number of array writes per buffer write is a metric for write coalescing. Figure 5b illustrates increasing opportunities for coalescing as the number of rows increase. As the number of rows in a 2048B-wide buffer increases from one to two and four rows, array writes per buffer write fall by $0.51\times$ and $0.32\times$, respectively; the buffers coalesce 49% and 68% of memory writes. Coalescing opportunities fall as buffer widths narrow beyond 256B. Since we use 64B lines in the lowest level cache, there are no coalescing opportunities from spatial locality within a 64B row buffered for a write. Increasing the number of 64B rows has no impact since additional rows exploit temporal locality, but any temporal locality in writes are already exploited by coalescing in the 64B lines of the lowest level cache.

Thus, narrow buffers mitigate high energy PCM writes and multiple rows exploit locality. This locality not only improves performance, but also reduces energy by exposing additional opportunities for write coalescing. We evaluate

PCM buffering using technology parameters at 90nm. As PCM technology matures, baseline PCM latencies may improve. Moreover, process technology scaling will drive linear reductions in PCM energy.

3.4. Scaling comparison

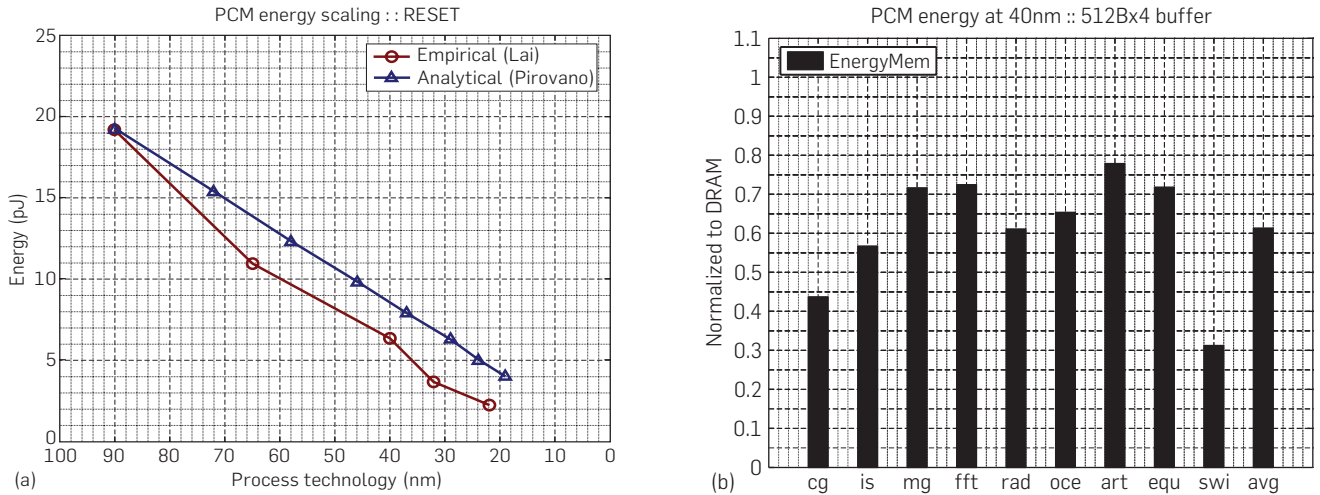
DRAM scaling faces many significant technical challenges as scaling attacks weaknesses in both components of the one transistor, one capacitor cell. Capacitor scaling is constrained by the DRAM storage mechanism, which requires maintaining charge on a capacitor. In future, process scaling is constrained by challenges in manufacturing small capacitors that store sufficient charge for reliable sensing despite large parasitic capacitances on the bitline.

The scaling scenarios are also bleak for the access transistor. As this transistor scales down, increasing subthreshold leakage will make it increasingly difficult to ensure DRAM retention times. Not only is less charge stored in the capacitor, that charge is stored less reliably. These trends impact the reliability and energy efficiency of DRAM in future process technologies. According to ITRS, "manufacturable solutions are not known" for DRAM beyond 40nm.¹⁷

In contrast, ITRS projects PCM scaling mechanisms will extend to 32nm, after which other scaling mechanisms might apply.¹⁷ Such PCM scaling has already been demonstrated with a novel device structure fabricated by Raoux.¹⁵ Although both DRAM and PCM are expected to be viable at 40nm technologies, energy scaling trends strongly favor PCM with a $2.4\times$ reduction in PCM energy from 80 to 40nm as illustrated in Figure 6a. In contrast, ITRS projects DRAM energy falls by only $1.5\times$ at 40nm, which reflects the technical challenges of DRAM scaling.¹⁷

Since PCM energy scales down faster than DRAM energy, PCM subsystems significantly outperform DRAM subsystems at 40nm. Figure 6b indicates PCM subsystem energy is 61.3% that of DRAM averaged across workloads. Switching from DRAM to PCM reduces energy costs by at

Figure 6. PCM Scalability. (a) Reset energy scaling from a survey of empirical prototypes by Lai and an analytical analysis by Pirovano et al. (b) Memory energy projections for 40 nm.



least 22.1% (art) and by as much as 68.7% (swim). Note this analysis does not account for refresh energy, which would further increase DRAM energy costs. Although ITRS projects constant retention time of 64ms as DRAM scales to 40nm,¹⁷ less effective access transistor control may reduce retention times. If retention times fall, DRAM refresh energy will increase as a fraction of total energy costs.

4. MEMORY LIFETIMES

In addition to architecting PCM to offer competitive delay and energy characteristics relative to DRAM, we must also consider PCM wear mechanisms. To mitigate these effects, we propose partial writes, which reduce the number of writes to the PCM array by tracking modified data from the L1 cache to the memory banks. This architectural solution adds a modest amount of cache state to reduce the number of bits written. We derive an analytical model to estimate memory module lifetime from a combination of fundamental PCM technology parameters and measured application characteristics. Partial writes, combined with an effective buffer organization, increase memory module lifetimes to a degree that makes PCM in main memory feasible.

4.1. Partial writes

Partial writes track data modifications, propagating this information from the L1 cache down to the buffers at the memory banks. When a buffered row is evicted and contents written to the PCM array, only modified data is written. We consider partial writes at two granularities: lowest level cache line size (64B) and word size (4B).

These granularities are least invasive since modified words are tracked by store instructions from the microprocessor pipeline. In contrast, bit-level granularity requires knowledge of previous data values and expensive comparators. We analyze a conservative implementation of partial writes, which does not exploit cases where stores write the same data values already stored.

Partial writes are supported by adding state to each cache

Table 1. Endurance model parameters.

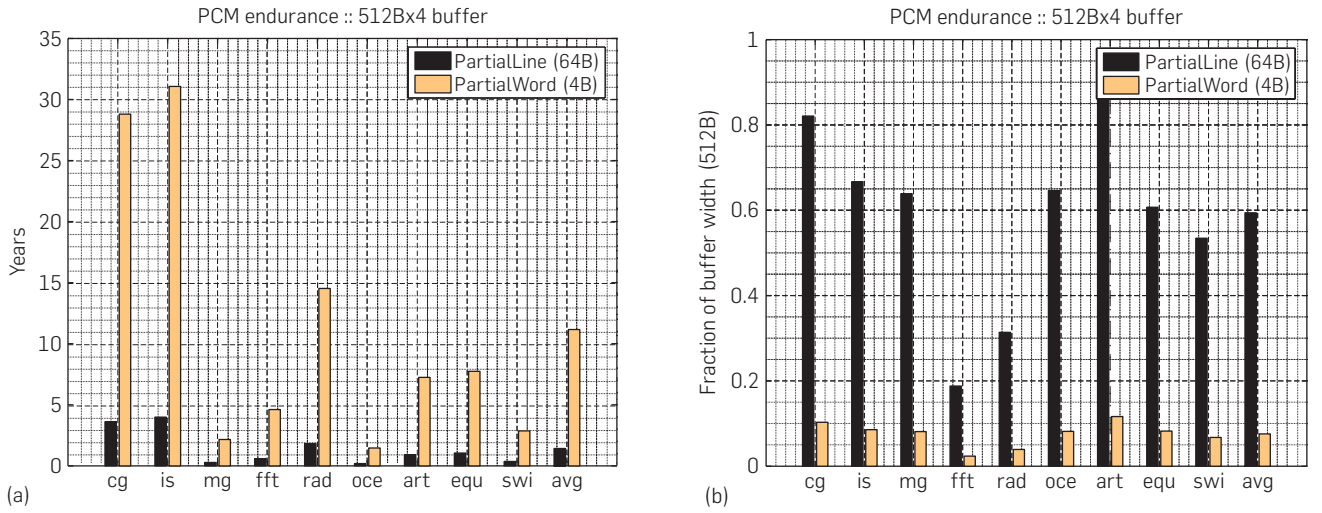
Endurance		
\hat{W}	Writes per second per bit	Equation 1
\hat{L}	Memory module lifetime (s)	Equation 1
E	Write endurance	$1E + 08$
Memory Module		
C	Logical capacity (Gb)	2
Memory Bus Bandwidth		
f_m	Memory bus frequency (MHz)	400
M_f	Processor frequency multiplier	10
B	Burst length (blocks)	8
Application Characteristics		
N_w, N_r	Number of writes, reads	sim
T	Execution time (cy)	sim
Buffer Characteristics		
W_{pt}, R_p	Buffer width (B), rows	512, 4
N_{wb}, N_{wo}	Buffer, array writes	sim
δ	Fraction of buffer written to array	sim

line, tracking stores using fine-grained dirty bits. At the dirty line granularity, 64B modifications are tracked beginning at the lowest level cache and requires only 1b per 64B L2 line. At the dirty word granularity, 4B modifications are tracked beginning at the L1 cache with 8b per 32B L1 line and propagated to the L2 cache with 16b per 64B L2 line. Overheads are 0.2% and 3.1% of each cache line when tracking dirty lines and words, respectively.

4.2. Endurance

Equation 1 estimates the write intensity observed by a memory module driven with access patterns observed in our memory-intensive workloads. Table 1 summarizes the model parameters. The model estimates the number of writes per second \hat{W} for any given bit. We first estimate memory bus occupancy, which has a theoretical peak command bandwidth of $f_m \cdot (B/2)^{-1}$. Each command requires $B/2$ bus cycles to transmit its burst length B in a DDR interface, which prevents commands from issuing at memory bus speeds f_m . We then scale this peak bandwidth by application-specific utilization. Utilization is computed by

Figure 7. PCM Endurance. (a) PCM memory module lifetimes. (b) Fraction of buffer modified (δ).



measuring the number of memory operations $N_w + N_r$, and calculating the processor cycles spent on these operations $(B/2) \cdot M_f$. The processor is M_f faster than f_m . The time spent on memory operations is divided by total execution time T .

$$\hat{W} = \underbrace{\frac{f_m \cdot (N_w + N_r) \cdot (B/2) \cdot M_f}{B/2}}_{\text{memBusOcc}} \times \underbrace{\frac{N_w}{N_w + N_r}}_{\text{writeIntensity}} \times \underbrace{8W_p \cdot \delta \cdot \left(\frac{N_{wa}}{N_{wb}}\right)}_{\text{bufferOrg}} \times \underbrace{\frac{1}{C}}_{\text{capacity}} \quad (1)$$

Since only a fraction of memory bus activity reaches the PCM to induce wear, we scale occupancy by write intensity to estimate the number of write operations arriving at the row buffers. In the worst case, the entire buffer must be written to the array. However, not all buffer writes cause array writes due to coalescing. N_{wa}/N_{wb} measures the coalescing effectiveness of the buffer, which filters writes to the array. Lastly, partial writes mean only the dirty fraction δ of a buffer's $8W_p$ bits are written to the array. Assuming ideal wear-leveling, writes will be spread across the C bits in the module. Given writes per second \hat{W} and characterized endurance E , a bit will fail in $\hat{L} = E/\hat{W}$ seconds.

In a baseline architecture with a single 2048B-wide buffer, average module lifetime is approximately 1050 h as calculated by Equation 1. For our memory-intensive workloads, we observe 32.8% memory bus utilization. Scaling by application-specific write intensity, we find 6.9% of memory bus cycles are utilized by writes. At the memory banks, the single 2048B buffer provides limited opportunities for write coalescing, eliminating only 2.3% of writes emerging from the memory bus. Frequent row replacements in the single buffer limit opportunities for coalescing.

Figure 7 illustrates significant endurance gains from reorganized buffers and partial writes. 64B and 4B partial writes improve endurance to 1.4 and 11.2 years, respectively. Buffers use partial writes so that only a fraction of the

buffer's bits is written to the array. As shown in Figure 7, only 59.3% and 7.6% of the buffer must be written to the array for 64B and 4B partial writes.

4.3. Density versus endurance

PCM cells are presently larger than DRAM cells. Measuring cell size in square feature sizes, which makes the discussion independent of process technology, PCM cells are 1.5–2.0 \times larger than DRAM cells.

In particular, $8F^2$ DRAM cells provide a sufficiently wide pitch to enable a folded bitline architecture, which is resilient against bitline noise during voltage sensing. However, manufacturers often choose the density of $6F^2$ DRAM cells. The narrow pitch in $6F^2$ designs preclude folded bitlines, increasing vulnerability to noise and requiring unconventional array designs. For example, Samsung's $6F^2$ implements array blocks with 320 wordlines, which is not a power of two, to improve reliability.⁵

In contrast, PCM cells occupy between $6F^2$ and $20F^2$.¹⁰ Part of this spread is due to differences in design and fabrication expertise for the new technology. However, we also observe a correlation between cell size and access device (e.g., the $6F^2$ cell uses the relatively small diode). We favor larger BJTs for their low access times. Cells with BJTs occupy between $9F^2$ and $12F^2$.

Given 9–12 F^2 PCM cells and $6F^2$ DRAM cells, two-bit multilevel PCM cells are necessary to be competitive with respect to density. Two-bit MLC provide an effective density of 4.5–6.0 F^2 per bit. However, MLC suffer from lower endurance. Process and manufacturing set the read window, which quantifies the difference between the lowest and highest programmed resistances in single-level cells. By programming the cell to intermediate resistances within the same read window, MLC inherently require a larger number of logical states that each occupy a narrower region of the read window. Thus, wear more quickly impacts the ability to differentiate these resistances.

4.4. Assumptions and qualifications

Considering only memory-intensive workloads, this analysis

is conservative. PCM subsystems would more likely experience a mix of compute and memory-intensive workloads. Expected lifetimes would be higher had we considered, for example, single-threaded SPEC integer workloads. However, such workloads are less relevant for a study of memory subsystems. Moreover, within memory-intensive workloads, we would expect to see a mix of read and write intensive applications, which may further increase lifetimes.

Scalability is projected to improve PCM endurance from the present $1E+08$ to $1E+12$ writes per bit at 32nm with known manufacturable solutions.¹⁷ This higher endurance increases lifetime by four orders of magnitude in our models. ITRS anticipates $1E+15$ PCM writes at 22nm although manufacturable solutions are currently unknown.

5. CONCLUSION

The proposed memory architecture lays the foundation for exploiting PCM scalability and nonvolatility in main memory. Scalability implies lower main memory energy and greater write endurance. Furthermore, nonvolatile main memories will fundamentally change the landscape of computing. Software cognizant of this newly provided persistence can provide qualitatively new capabilities. For example, system boot/hibernate will be perceived as instantaneous; application checkpointing will be inexpensive⁷; file systems will provide stronger safety guarantees.⁶ Thus, we take a step toward a new memory hierarchy with deep implications across the hardware–software interface. C

References

1. Aslot, V., Eigenmann, R. Quantitative performance analysis of the SPEC OMPM2001 benchmarks. *Sci. Program.* 11, 2 (2003).
2. Bailey, D. et al. NAS parallel benchmarks. In *Technical Report RNR-94-007, NASA Ames Research Center*, March 1994.
3. Bedeschi, F. et al. A multi-level-cell bipolar-selected phase-change memory. In *International Solid-State Circuits Conference*, 2008.
4. Chen, Y. et al. Ultra-thin phase-change bridge memory device using GeSb. In *International Electron Devices Meeting*, 2006.
5. Choi, Y. Under the hood: DRAM architectures: 8F2 vs. 6F2. *EE Times*, February 2008.
6. Condit, J. et al. Better I/O through byte-addressable, persistent memory. In *Symposium on Operating System Principles*, Oct 2009.
7. Dong, X. et al. Leveraging 3D PCRAM technologies to reduce checkpoint overhead in future exascale systems. In *Conference on Supercomputing*, Nov 2009.
8. Horii, H. et al. A novel cell technology using N-doped GeSbTe films for phase change RAM. In *Symposium on VLSI Technology*, 2003.
9. Lai, S. Current status of the phase change memory and its future. In *International Electron Devices Meeting*, 2003.
10. Lee, B., Ipek, E., Mutlu, O., Burger, D. Architecting phase change memory as a scalable DRAM alternative. In *International Symposium on Computer Architecture*, June 2009.
11. Lee, K.-J. et al. A 90nm 1.8V 512Mb diode-switch PRAM with 266 MB/s read throughput. *J. Solid State Circuit.* 43, 1 (Jan 2008).
12. Micron. 512Mb DDR2 SDRAM component data sheet: MT47H128M4B6-25. In www.micron.com, Mar 2006.
13. Nirschl, T. et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In *International Electron Devices Meeting*, 2008.
14. Pirovano, A. et al. Scaling analysis of phase-change memory technology. In *International Electron Devices Meeting*, 2003.
15. Raoux, S. et al. Phase-change random access memory: A scalable technology. *IBM J. Res. Dev.* 52, 4/5 (Jul/Sept 2008).
16. Renau, J. et al. SESC simulator. In <http://sec.sourceforge.net>, 2005.
17. Semiconductor Industry Association. Process integration, devices & structures. *International Technology Roadmap for Semiconductors*, 2007.
18. Sinha, M. et al. High-performance and low-voltage sense-amplifier techniques for sub-90 nm sram. In *International Systems-on-Chip Conference*, 2003.
19. Woo, S. et al. The SPLASH-2 programs: Characterization and methodological considerations. In *International Symposium on Computer Architecture*, June 1995.

Benjamin C. Lee (bclee@stanford.edu), Stanford University.

Engin Ipek (ipek@cs.rochester.edu), University of Rochester.

Onur Mutlu (onur@cmu.edu), Carnegie Mellon University.

Doug Burger (dburger@microsoft.com), Microsoft Research.

© 2010 ACM 0001-0782/10/0700 \$10.00



You've come a long way.
Share what you've learned.



ACM has partnered with MentorNet, the award-winning nonprofit e-mentoring network in engineering, science and mathematics. MentorNet's award-winning **One-on-One Mentoring Programs** pair ACM student members with mentors from industry, government, higher education, and other sectors.

- Communicate by email about career goals, course work, and many other topics.
- Spend just **20 minutes a week** - and make a huge difference in a student's life.
- Take part in a lively online community of professionals and students all over the world.



Make a difference to a student in your field.
Sign up today at: www.mentornet.net
Find out more at: www.acm.org/mentornet

MentorNet's sponsors include 3M Foundation, ACM, Alcoa Foundation, Agilent Technologies, Amylin Pharmaceuticals, Bechtel Group Foundation, Cisco Systems, Hewlett-Packard Company, IBM Corporation, Intel Foundation, Lockheed Martin Space Systems, National Science Foundation, Naval Research Laboratory, NVIDIA, Sandia National Laboratories, Schlumberger, S.D. Bechtel, Jr. Foundation, Texas Instruments, and The Henry Luce Foundation.

Cal Poly State University Lecturer

COMPUTER SCIENCE - Part-time lecturer positions are available in the Computer Science Department at Cal Poly, San Luis Obispo, CA during the 2010-2011 academic year. Post-Baccalaureate degree in computer science, mathematics, engineering, or related area required. Candidates must have demonstrated teaching ability as well as evidence of ongoing professional development and competence. To apply, please visit www.calpolyjobs.org, complete a required online faculty application, and apply to requisition #102089. Review Begin Date: June 1, 2010. EEO

DOCOMO Communications Laboratories USA, Inc. Research Engineer

We currently look for 5-10 years experience researcher/engineer to work in distributed computing, data analysis, and optimization. The candidate must have a strong implementation skill for Java Developer Apache Hadoop -Computer Storage System. We expect the candidate must be self motivated and have experience of system design and algorithm during the work. Strong background in the following areas is also required: statistical algorithm, signal processing, and graph theory. Strong programming experience in Java, statistical programming languages like R, and Open Source technologies are required. Experience with implementation on distributed computing platforms like MapReduce or Hadoop on Linux systems is highly required.

Applicants must have the relevant authorization to work in a U.S. company.

Please indicate Code:

OOP-RE-ACM

Email Address:

oop_recruit@docomolabs-usa.com

Apply URL:

<http://www.docomolabs-usa.com/>

Drakontas Technical Project Leader

Manage and develop software projects involving web services/SOA in law enforcement and military domain. Hands-on a must. Expert in Linux, XML, IP networks, mobile technologies. Help write research grant proposals. MS or PhD in CS. Apply at: <http://www.drakontas.com/careers>

Louisiana State University Assistant/Associate Professor (Computational Biology/Tenure-track position)

Department of Computer Science/Center for Computation and Technology: The Louisiana State University Department of Computer Science

(<http://csc.lsu.edu>) & Center for Computation & Technology (CCT) (<http://www.cct.lsu.edu>), invite applications for an Assistant/Associate Professor (tenure-track) faculty position in Computational Biology, broadly defined. The CCT offers an innovative and interdisciplinary research environment for advancing computational sciences, including a highly competitive computing environment with access to 100 TFlops of computing resources in conjunction with the Louisiana Optical Network Initiative (LONI) (<http://institute.loni.org/>). Suitable candidate may be appointed a LONI Institute Fellow. LSU is part of the national TeraGrid. Required Qualifications: (Both Levels) Ph.D. or equivalent degree; a successful track record of productive research and extramural funding; a computer scientist or computational biologist whose research involves areas such as: biomolecular dynamics and structure-based drug design, systems biology and interaction networks, metabolomics, evolutionary genomics, or metagenomics analyses; experience with developing the methods, infrastructure and algorithms to take advantage of high-performance and distributed computing and other advances in computing. Responsibilities: establishes a vigorous, extramurally funded research program; contributes to undergraduate and graduate teaching; contributes to a growing interdepartmental program in computational biology at LSU. We encourage applications from women and minorities. Rank and salary will be commensurate with qualifications and/or experience. An offer of employment is contingent on a satisfactory pre-employment background check. Application deadline is July 15, 2010 or until a candidate is selected. Apply online at: www.lsusystemcareers.lsu.edu. Position #034189. LSU SYSTEM IS AN EQUAL OPPORTUNITY/EQUAL ACCESS EMPLOYER

Princeton University Computer Science Lecturer

Part- and full-time Lecturer positions. The Department of Computer Science seeks applications from outstanding teachers to assist the faculty in teaching our introductory course sequence.

The primary requirements of the job are to teach recitation sections and to participate in overall management of the introductory sequence. Other responsibilities include supervising graduate student teaching assistants and developing and maintaining online curricular material, classroom demonstrations, and laboratory exercises.

Candidates should have an exceptional record of classroom instruction and curricular innovation. An advanced degree in computer science is preferred.

For general application information and to self-identify visit: <https://jobs.princeton.edu>
Requisition Number: 1000207. You may apply online on the Department's website at: <http://www.cs.princeton.edu/jobs/lecturerposition>

We will not accept applications from the

Princeton jobs site.

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations.

Shell Principal Researcher, Computation and Modeling

CAN YOU SAY NO TO NO? If you answered yes, then we should talk. We're Shell Global Solutions and we're looking for individuals who find more ways to say "yes" instead of "no."

As a **Principal Researcher, Computation and Modeling**, you'll have the opportunity to seek out, evaluate, discover and invent cutting-edge emerging technologies on the interface of engineering, the physical sciences and computer science for the Royal Dutch Shell (RDS) business.

In short, the perfect candidate for our Principal Researcher position is the type of person who turns impossible into possible almost every day.

► PhD in Computer Science, Physics, Engineering, Mathematics or related discipline.

► 10 or more years of post-doctorate, hands-on, in-depth technical and project lead experience in the area of simulation, computation, and modeling.

► Demonstrated professional recognition via peer reviews, papers, books, teaching, training and more.

► Legal authorization to work in the US on a full-time basis.

With technically innovative people—and state-of-the-art equipment and facilities—Shell is a worldwide leader in the development and application of technology. At Shell, we're committed to satisfying the world's need for energy with economically, socially and environmentally responsible solutions. We'll provide you with the resources to put your ideas into action, worldwide opportunities to advance your career, and outstanding benefits and rewards that support your quality of life. Join us and let's make a real difference together.

Apply URL: <http://www.shell.us/careers>

No phone inquiries will be accepted. Shell is an Equal Opportunity Employer.

Universidad de Chile Faculty Positions in Advanced Networks and Future Internet Services

The Electrical and Computer Engineering Departments of the Universidad de Chile have opened two full-time faculty positions. Their main duties will be teaching and research at the Universidad de Chile in Santiago.

A PhD in a related field is required. Spanish is not required but the candidate should be willing to learn the language.

Details at:

http://www.dcc.uchile.cl/networks_position

University of North Dakota
Department of Computer Science
Assistant to Associate Professor

The Department of Computer Science at the University of North Dakota is recruiting for a nine-month tenure-track position at the Assistant or Associate Professor level with an anticipated start date of January 1, 2011. Salary ranges from 76,000 to 93,206.

The responsibilities include teaching at the undergraduate and graduate levels, supervising graduate students (MS and PhD), and developing and maintaining an active research program. Associate Professor candidates should have a good publication and funding record, proven ability to establish an independent research program, and be open to participation in interdisciplinary programs of study. Industry experience and/or post-doctoral experience will be considered an asset.

The position requires a Ph.D. in Computer Science or a related field. The department is seeking outstanding candidates with a research focus in one of the following areas: (a) simulation and modeling or (b) high-performance computing.

Our department strives to maintain a collegial and supportive atmosphere. UND hires on the basis of merit and is committed to employment equity. We strongly encourage candidates with diverse backgrounds and experiences to apply. The University of North Dakota is an AAEO Employer.

Applicants for the position must submit their curriculum vitae, a teaching statement, a research statement, and the names of at least three references. The teaching statement should include a record of teaching interests and experience. Screening begins immediately and continues until the position is filled. Applications may be sent electronically in PDF format to: Ronald Marsh, Ph.D., Associate Professor and Chair
www.cs.und.edu

If you have questions about the application process, please contact Ronald Marsh at rmarsh@cs.und.edu



**ADVERTISING IN
 CAREER OPPORTUNITIES**

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@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: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. 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 30 days.

For More Information Contact:

**ACM Media Sales
 at 212-626-0686 or
acmm mediasales@acm.org**



Los Alamos National Laboratory - a premier national security research institution delivering innovative science and engineering solutions for the nation's most crucial and complex problems - has the following opening available:

R&D MANAGER 4

Our Computer, Computational and Statistical Sciences Division seeks an accomplished professional to provide technical leadership and management for the Applied Computer Science group, CCS-7. A significant growth area, CCS-7 is at the forefront of technology and algorithm design/development for cutting-edge computer architectures. Your role will be to develop and execute a diverse portfolio of work in support of multiple programmatic efforts, including nuclear weapons, various Office of Science, energy and work-for-others programs. Additionally, you'll attract, develop and motivate a diverse scientific workforce to deliver on our mission and achieve success.

Requires a Bachelor's degree in Science or Engineering, demonstrated experience building teams and leading technical projects in a diverse scientific environment, advanced expertise in relevant technical areas and ability to obtain a DOE Q clearance.

For the complete job description and to apply online, visit www.lanl.gov/jobs and apply to job number 219494.



www.lanl.gov/jobs



Announcement of an open position at the Faculty of Informatics,
 Vienna University of Technology, Austria

**FULL PROFESSOR (TENURED)
 of Dependable Systems**

The open position and its research group are affiliated with the Institute of Computer Engineering. The professor is expected to maintain the leading role of this institute in the international domain of dependable embedded-systems research for the next decades.

The successful candidate will have an outstanding research record in the field of dependable embedded systems, with expertise in at least two of the following sub-fields:

- Distributed systems
- Real-time systems
- Design and programming languages
- Safety and security

We offer excellent working conditions in an attractive research environment in a city with an exceptional quality of living.

A more detailed announcement and information on how to apply can be found at:

<http://www.informatik.tuwien.ac.at/DS.pdf>

Application Deadline: **September 30, 2010**



The Hong Kong Polytechnic University is the largest government-funded tertiary institution in Hong Kong in terms of student numbers. It offers programmes at Doctorate, Master's, Bachelor's degrees and Higher Diploma levels. It has a full-time academic staff strength of around 1,400. The total consolidated expenditure budget of the University is in excess of HK\$4 billion per year.

DEPARTMENT OF COMPUTING

Head of Department of Computing (Ref. 99726)

Founded in 1974, the Department of Computing of The Hong Kong Polytechnic University was amongst the first in Hong Kong to offer education in computing and information technology. As a provider of the most versatile education in computing, information technology and information systems, our undergraduate programmes are designed in ways that can incorporate new applications, meet challenges and respond to trends.

Apart from nurturing professional talents for society, the Department excels in research and innovation. Leveraging its research strengths, we make significant contributions to the economy with research services that lead to technology transfer, consultancy projects and commercialisation. Please visit the website at <http://www.comp.polyu.edu.hk/> for more information about the Department. The Department of Computing is a constituent of the Faculty of Engineering, information of which is available at <http://www.polyu.edu.hk/feng/>.

The University is now inviting applications and nominations for the post of Head of Department of Computing. The successful candidate will be appointed as Chair Professor or Professor and hold a concurrent headship appointment.

The position calls for an academic leader with responsibilities of ensuring the smooth and successful operation and sustainable development of the Department. Reporting directly to the Dean of Faculty, the appointee will be required to provide effective leadership in the development of long-term strategies and plans of the Department, and provide support to staff members in the Department for the accomplishment of strategic objectives with high quality standards. Other responsibilities include inspiring excellence in teaching, research and services; fostering strong partnerships and collaborations with external organizations and strengthening the international network of the Department and the University; ensuring optimal deployment of human, financial and other resources in the Department; and implementing an effective mechanism to acquire donations and other forms of sponsorship to support the University's pursuits and long-term development.

Applicants should have (a) outstanding academic qualifications at doctoral level in Computer Science, Information Technology, Information Systems or related disciplines, evidence of eminent scholarship and substantial relevant experience in a senior academic position; (b) a strong track record of achievement in teaching, research, professional services and leadership at a senior level; (c) demonstrated ability to build up a strong team of faculty members with different research and cultural backgrounds; (d) effective interpersonal communication and resources management skills, and excellent adaptability to changes and challenges; (e) good knowledge of the higher educational environment, preferably including Hong Kong; and (f) a global perspective, and proven ability to promote collaboration across departments and institutions. Experience in fund-raising will be an additional advantage.

Remuneration and Conditions of Service

Terms of appointment and remuneration package are negotiable and highly competitive.

Application

Applicants are invited to send detailed curriculum vitae with names and addresses of two referees to the **Human Resources Office, 13/F, Li Ka Shing Tower, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong [Fax: (852) 2764 3374; E-mail: hrstaff@polyu.edu.hk], quoting position applied for and reference number. Recruitment will continue until the position is filled. Initial consideration of applications will commence in September 2010. Candidature may be obtained by nomination. The University reserves the right not to fill this post or to make an appointment by invitation. General information about the University is available on the University's World Wide Web Homepage <http://www.polyu.edu.hk> or from the Human Resources Office [Tel: (852) 2766 5040]. Details of the University's Personal Information Collection Statement for recruitment can be found at <http://www.polyu.edu.hk/hro/jobpics.htm>.**

For further details about the University, please visit www.polyu.edu.hk

Take Advantage of ACM's Lifetime Membership Plan!

- ◆ **ACM Professional Members** can enjoy the convenience of making a single payment for their entire tenure as an ACM Member, and also be protected from future price increases by taking advantage of **ACM's Lifetime Membership** option.
- ◆ **ACM Lifetime Membership** dues may be tax deductible under certain circumstances, so becoming a Lifetime Member can have additional advantages if you act before the end of 2009. (Please consult with your tax advisor.)
- ◆ Lifetime Members receive a certificate of recognition suitable for framing, and enjoy all of the benefits of **ACM Professional Membership**.

Learn more and apply at:

<http://www.acm.org/life>



Association for
Computing Machinery

Advancing Computing as a Science & Profession

Call for Nominations

The ACM Doctoral Dissertation Competition

Rules of the Competition

ACM established the Doctoral Dissertation Award program to recognize and encourage superior research and writing by doctoral candidates in computer science and engineering. These awards are presented annually at the ACM Awards Banquet.

Submissions

Nominations are limited to one per university or college, from any country, unless more than 10 Ph.D.'s are granted in one year, in which case two may be nominated.

Deadline

Submissions must be received at ACM headquarters by **October 31, 2010** to qualify for consideration.

Eligibility

Each nominated dissertation must have been accepted by the department between October 2009 and September 2010. Only English language versions will be accepted. Please send a copy of the thesis in PDF format to emily.eng@acm.org.

Sponsorship

Each nomination shall be forwarded by the thesis advisor and must include the endorsement of the department head. A one-page summary of the significance of the dissertation written by the advisor must accompany the transmittal.

Publication Rights

Each nomination must be accompanied by an assignment to ACM by the author of exclusive publication rights. (Copyright reverts to author if not selected for publication.)

Publication

Winning dissertations will be published by Springer.

Selection Procedure

Dissertations will be reviewed for technical depth and significance of the research contribution, potential impact on theory and practice, and quality of presentation. A committee of five individuals serving staggered five-year terms performs an initial screening to generate a short list, followed by an in-depth evaluation to determine the winning dissertation.

The selection committee will select the winning dissertation in early 2011.

Award

The Doctoral Dissertation Award is accompanied by a prize of \$20,000 and the Honorable Mention Award is accompanied by a prize of \$10,000. Financial sponsorship of the award is provided by Google.

For Submission Procedure

See <http://awards.acm.org/html/dda.cfm>



[CONTINUED FROM P. 112] documents, made us think it would be a good idea to pursue the question of how to build what came to be called the paperless office.

You're talking about Robert Taylor, who managed the Computer Systems Laboratory at PARC.

Taylor was not a technologist—he was a psychologist by training. But he was an extremely effective leader. The other thing was, he knew everyone in computing because he had run ARPA's Information Processing Techniques Office. So when he was hired to staff the lab at PARC, he knew where to go.

Did you have a sense of how revolutionary Alto was as you worked on it?

Oh, yes, we knew it was revolutionary. We built it with the very first semiconductor dynamic RAM, the Intel 1103, which was the first memory you could buy that was less than a tenth of a cent a bit. As a result, we realized we could build a display that was qualitatively better than what we had at the time. We had character generator terminals, and some of them were quite nice. But they were limited in various ways, whereas the Alto had the property that anything you could represent on paper, you could put on the screen. We knew that was going to be a big deal.

You were also involved in the invention of the Ethernet.

The Ethernet grew out of the realization I had of how to provide a network for the Alto. We had been studying the ALOHA network, a radio network that was used to connect the various Hawaiian Islands. The limitation was that when a transmitter started to transmit, it could no longer receive anything. One night I was lying in bed thinking about the problem when I had this sudden realization that if you used a more limited media, say, the coaxial cables used in cable television, the transmitter could not only hear what it transmitted, it could also tell whether what it thought it put on the wire was the same as what actually got put on the wire.

So if another transmitter was interfering, it could drop back and retransmit later.

“The Alto had the property that anything you could represent on paper, you could put on the screen. We knew that was going to be a big deal.”

That idea was refined by Bob Metcalfe and Dave Boggs into what we knew as the Ethernet. Of course, the Ethernet in those days was quite different than it is today.

You joined Microsoft in 1997 to help establish the company's research lab in Cambridge, England, and were later involved in the development of the tablet PC, a subject that's much in the news of late.

The line of thinking about tablets actually started at DEC [Digital Equipment Corporation]. We built a tablet called Lectrice back in the early 1990s, primarily as an electronic book reader. When I returned to the U.S. from my two-year assignment in Cambridge, I was working with a group in Redmond that was trying to build an electronic book reader. That didn't work out too well, but it evolved into the idea of building a tablet PC. Of course the view there was it would be great to have a device that didn't require a keyboard.


What do you make of the persistence of the keyboard in spite of the alternatives that now exist?

Typing is so much faster than virtually any other way of entering information into a computer, so I don't expect that to change. There's only one thing that can be better, and that's to use a different set of muscles—the tablet allows you to do that. You're holding a stylus and writing or drawing with it, and the interaction can be faster.

More recently, you've been working on multicore systems.

I've been using field-programmable gate arrays (FPGAs) to explore multicore architectures. For a long time, it was impossible for academic researchers—or even people working in industrial labs—to design their own chips. It's now possible to build a nontrivial multicore computer, with something on the order of 15 cores, on a single FPGA on a board that's available for \$750.

What does the future hold for you in terms of research? Are you tempted to go back and continue working on the tablet PC?

If I have a good idea I might go back to it. But right now I'm quite happy with what I'm doing, and there's a considerable amount of work to do in this area. So I think I'm set for the next few years. 

Leah Hoffmann is a Brooklyn, NY-based technology writer.

© 2010 ACM 0001-0782/10/0700 \$10.00



Publish yourself at Google Code University

Share what you know with students, teachers, and other computer scientists at Google Code University. It's the online exchange where great computing minds publish tutorials, lesson plans and test exercises — under Creative Commons, for the common good. You'll help others learn (and look good doing it).

Learn more:
<http://code.google.com/edu/>

Q&A

From Single Core to Multicore

Charles P. Thacker discusses the legendary Alto personal computer, the invention of the Ethernet, and his current research on multicore architectures.

CHARLES P. THACKER, a Technical Fellow at Microsoft, is the winner of the 2009 ACM A.M. Turing Award for his pioneering contributions to computer architecture and networks, as well as his current work on multicore computing. (A profile of Thacker, “Committed to Success,” is on p. 22.) We spoke with him about the technological highlights of his career, beginning with his work at Xerox Palo Alto Research Center (PARC) in 1970.

Let’s talk about the development of the Alto, the first computer to incorporate a bitmap display and a graphical user interface.

The Alto was actually the second machine we built at PARC—the first one was a time-sharing machine. We wanted a PDP-10 because that was the standard machine that the ARPA [Advanced Research Projects Agency] research community used, but it would

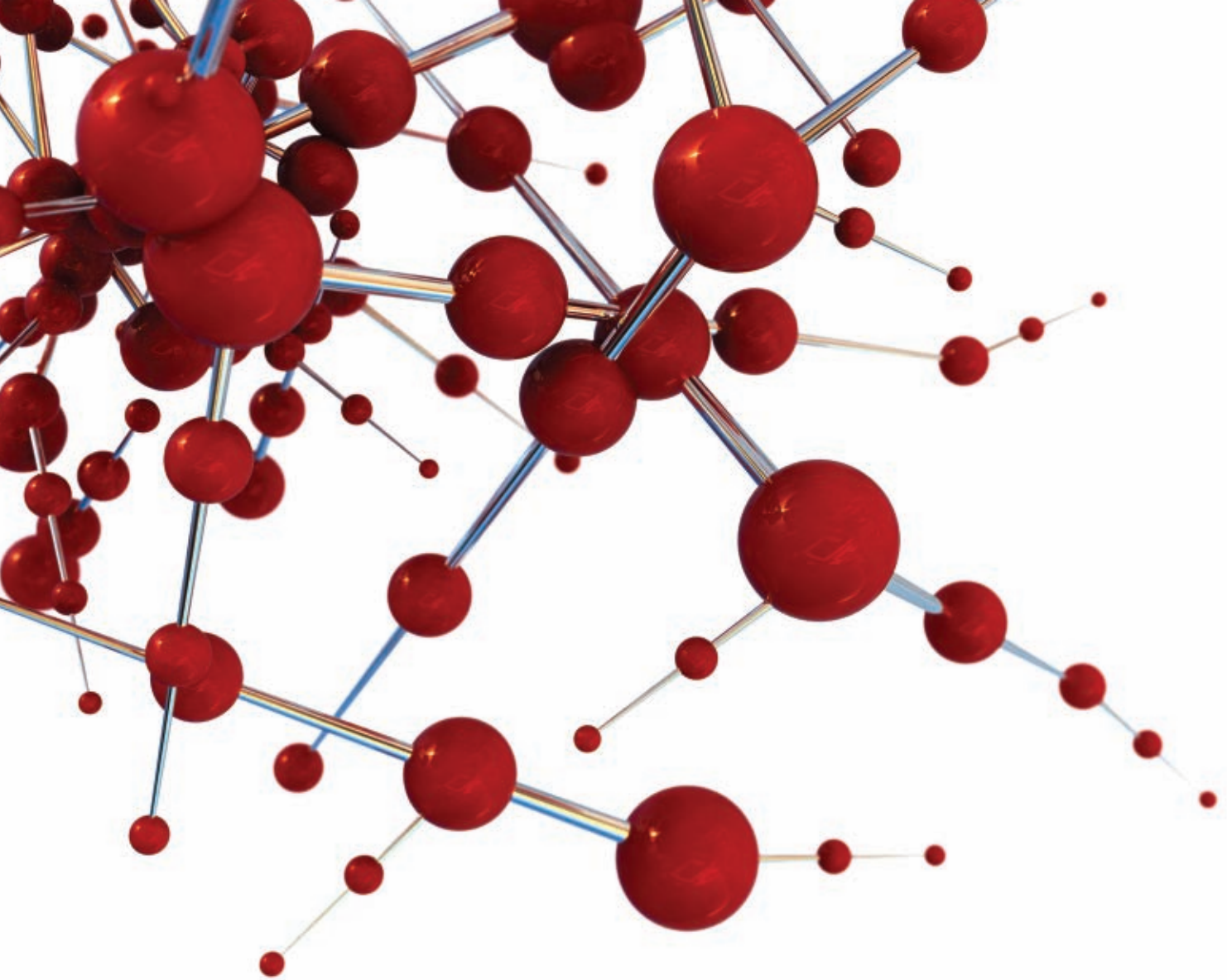
have been unseemly for us to buy one because Xerox had just bought a computer company that made a competing machine.

So you decided to build one instead.

Bob Taylor had continuously told us, “Computers are for people. They’re personal devices.” That, coupled with the fact we were in this company that handled [CONTINUED ON P. 111]



Microsoft Technical Fellow and 2009 ACM A.M. Turing Award winner Charles P. Thacker in front of the Charles Babbage Difference Engine No. 2 at the Computer History Museum, Mountain View, CA.



**CONNECT WITH OUR
COMMUNITY OF EXPERTS.**

www.reviews.com

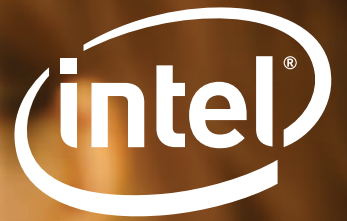


Association for
Computing Machinery

Reviews.com

They'll help you find the best new books
and articles in computing.

Computing Reviews is a collaboration between the ACM and Reviews.com.



Think Parallel....

It's not just what we make.
It's what we make possible.

Advancing Technology Curriculum
Driving Software Evolution
Fostering Tomorrow's Innovators

Learn more at: www.intel.com/thinkparallel

