

COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

05/2015 VOL.58 NO.05

Optimization as Motion Selection Principle in Robot Action



Privacy Behaviors
After Snowden

Decoding
Femininity in
CS in India

Managing Your
Digital Life

Life of IP

Is 'Good Enough'
Computing Good
Enough?

Incentivizing Quality
and Impact in
Computing Research



Photo: Andrés Lucero | Design: Andrés Lucero, Rodrigo Zuloaga

ITS 2015

NATURE MEETS INTERACTIVE SURFACES



Deadlines

Tutorials and Workshops
15 May 2015

Full and Short Papers
3 July 2015

Doctoral Symposium
3 July 2015

Demos and Posters
28 August 2015

Interactive Surfaces increasingly pervade our everyday life, appearing in various sizes, shapes, and application contexts offering a rich variety of ways how to interact with them.

ITS 2015 welcomes original, high-quality research and industry contributions that advance the state-of-the-art in the area of interactive tabletops and surfaces. We embrace innovations in a wide variety of areas including design, software, hardware, understanding of use, and applications or deployments of interactive surfaces.

Join us in Madeira, Portugal

15-18 November 2015

its2015.org



THE ACM A. M. TURING AWARD

by the community ♦ from the community ♦ for the community



ACM and Google congratulate

MICHAEL STONEBRAKER

For fundamental contributions to the concepts and practices underlying modern database systems.



"The efficient and effective management of Big Data is crucial to our 21st century global economy," said Google Senior Vice President of Knowledge Alan Eustace. "Michael Stonebraker invented many of the architectures and strategies that are the foundation of virtually all modern database systems."

Alan Eustace
Vice President of Knowledge
Google Inc.

Google™

For more information see <http://research.google.com/>

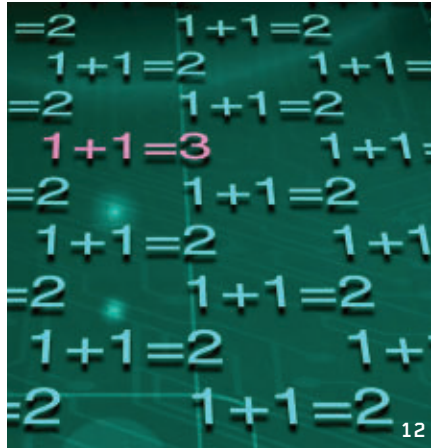
Departments

- 5 **Editor's Letter**
Incentivizing Quality and Impact in Computing Research
By Moshe Y. Vardi
-
- 7 **Cerf's Up**
Cascade Failure
By Vinton G. Cerf
-
- 8 **Letters to the Editor**
Abolish Software Warranty Disclaimers
-
- 10 **BLOG@CACM**
Introducing Young Women to CS, and Supporting Advanced Research Environments
Joel Adams talks about starting a chapter of Girls Who Code, while Daniel A. Reed considers an issue with cyberinfrastructure.
-
- 25 **Calendar**
-
- 87 **Careers**

Last Byte

- 88 **Upstart Puzzles**
Strategic Friendship
By Dennis Shasha

News



- 12 **Is "Good Enough" Computing Good Enough?**
The energy-accuracy trade-off in approximate computing.
By Logan Kugler
-
- 15 **Putting the Data Science into Journalism**
News organizations increasingly use techniques like data mining, Web scraping, and data visualization to uncover information that would be impossible to identify and present manually.
By Keith Kirkpatrick
-
- 18 **Robots with a Human Touch**
Empowering smart machines with tactile feedback could lead to tremendous new applications.
By Gregory Mone

Viewpoints

- 20 **Economic and Business Dimensions**
Life of IP
Seeking to balance intellectual property protection with incentives for investment in innovation.
By Michael Schrage and Marshall Van Alstyne
-
- 24 **Education**
What Are We Doing When We Teach Computing in Schools?
Research on the cognitive, educational, and policy dimensions of teaching computing is critical to achieving "computer literacy."
By Sally Fincher
-
- 27 **Law and Technology**
Oracle v. Google: A High-Stakes Legal Fight for the Software Industry
Copyrighting APIs threatens significant harm to both consumers and the software industry.
By Christopher Jon Sprigman
-
- 30 **Viewpoint**
Teach Foundational Language Principles
Industry is ready and waiting for more graduates educated in the principles of programming languages.
By Thomas Ball and Benjamin Zorn
-
- 32 **Viewpoint**
Managing Your Digital Life
Everyone should be able to manage their personal data with a personal information management system.
By Serge Abiteboul, Benjamin André, and Daniel Kaplan



Practice



42

36 **There Is No Now**
Problems with simultaneity
in distributed systems.
By Justin Sheehy

42 **Parallel Processing with Promises**
A simple method of writing
a collaborative system.
By Spencer Rathbun

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

Contributed Articles



56

48 **Privacy Behaviors After Snowden**
Despite continuing media coverage,
the public's privacy behaviors
have hardly changed.
By Sören Preibusch

56 **Decoding Femininity in
Computer Science in India**
The benefits in school and
the job market so far outweigh
any potential gender bias that
few women are deterred.
By Roli Varma and Deepak Kapur

Review Articles



64

64 **Optimization as Motion Selection
Principle in Robot Action**
Robots move to act. While actions
operate in a physical space, motions
begin in a motor control space.
So how do robots express actions
in terms of motions?
*By Jean-Paul Laumond, Nicolas Mansard,
and Jean Bernard Lasserre*



Watch the authors discuss
this work in this exclusive
Communications video.

Research Highlights

76 **Technical Perspective**
Programming Multicore Computers
By James Larus

77 **Can Traditional Programming
Bridge the Ninja Performance Gap
for Parallel Computing Applications?**
*By Nadathur Satish,
Changkyu Kim, Jatin Chhugani,
Hideki Saito, Rakesh Krishnaiyer,
Mikhail Smelyanskiy,
Milind Girkar, and Pradeep Dubey*

About the Cover:



While movement is a
fundamental characteristic
of living systems,
incorporating that ability
into robotic systems
relies on the strength
and expertise of many
scientific domains.
This month's cover
story (p. 64) details
the expressive power of
optimal motion in robot
action modeling. Cover
illustration by Peter
Crowthier Associates.



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

Executive Director and CEO
John White
Deputy Executive Director and COO
Patricia Ryan
Director, Office of Information Systems
Wayne Graves
Director, Office of Financial Services
Darren Ramdin
Director, Office of SIG Services
Donna Cappel
Director, Office of Publications
Bernard Rous
Director, Office of Group Publishing
Scott E. Delman

ACM COUNCIL

President
Alexander L. Wolf
Vice-President
Vicki L. Hanson
Secretary/Treasurer
Erik Altman
Past President
Vinton G. Cerf
Chair, SGB Board
Patrick Madden
Co-Chairs, Publications Board
Jack Davidson and Joseph Konstan
Members-at-Large
Eric Allman; Ricardo Baeza-Yates;
Cherri Pancake; Radia Perlman;
Mary Lou Soffa; Eugene Spafford;
Per Stenström
SGB Council Representatives
Paul Beame; Barbara Boucher Owens;
Andrew Sears

BOARD CHAIRS

Education Board
Mehran Sahami and Jane Chu Prey
Practitioners Board
George Neville-Neil

REGIONAL COUNCIL CHAIRS

ACM Europe Council
Fabrizio Gagliardi
ACM India Council
Srinivas Padmanabhuni
ACM China Council
Jiaquan Sun

PUBLICATIONS BOARD

Co-Chairs
Jack Davidson; Joseph Konstan
Board Members
Ronald F. Boisvert; Nikil Dutt; Roch Guerrin;
Carol Hutchins; Yannis Ioannidis;
Catherine McGeoch; M. Tamer Ozsu;
Mary Lou Soffa

ACM U.S. Public Policy Office

Renee Dopplick, Director
1828 L Street, N.W., Suite 800
Washington, DC 20036 USA
T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association
Lissa Clayborn, Acting Executive Director

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

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

STAFF

DIRECTOR OF GROUP PUBLISHING
Scott E. Delman
cacm-publisher@cacm.acm.org

Executive Editor
Diane Crawford
Managing Editor
Thomas E. Lambert
Senior Editor
Andrew Rosenbloom
Senior Editor/News
Larry Fisher
Web Editor
David Roman
Rights and Permissions
Deborah Cotton

Art Director
Andrij Borys
Associate Art Director
Margaret Gray
Assistant Art Director
Mia Angelica Balaquiot
Designer
Iwona Usakiewicz
Production Manager
Lynn D'Addesio
Director of Media Sales
Jennifer Ruzicka
Public Relations Coordinator
Virginia Gold
Publications Assistant
Juliet Chance

Columnists
David Anderson; Phillip G. Armour;
Michael Cusumano; Peter J. Denning;
Mark Guzdial; Thomas Haigh;
Leah Hoffmann; Mari Sako;
Pamela Samuelson; Marshall Van Alstyne

CONTACT POINTS

Copyright permission
permissions@cacm.acm.org
Calendar items
calendar@cacm.acm.org
Change of address
acmhelp@cacm.acm.org
Letters to the Editor
letters@cacm.acm.org

WEBSITE
<http://cacm.acm.org>

AUTHOR GUIDELINES
<http://cacm.acm.org/>

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY
10121-0701
T (212) 626-0686
F (212) 869-0481

Director of Media Sales
Jennifer Ruzicka
jen.ruzicka@hq.acm.org
Media Kit acmm mediasales@acm.org

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

EDITORIAL BOARD

EDITOR-IN-CHIEF
Moshe Y. Vardi
eic@cacm.acm.org
NEWS
Co-Chairs
William Pulletyblank and Marc Snir
Board Members
Mei Kobayashi; Kurt Mehlforn;
Michael Mitzenmacher; Rajeev Rastogi

VIEWPOINTS
Co-Chairs
Tim Finin; Susanne E. Hambrusch;
John Leslie King
Board Members
William Aspray; Stefan Bechtold;
Michael L. Best; Judith Bishop;
Stuart I. Feldman; Peter Freeman;
Mark Guzdial; Rachelle Hollander;
Richard Ladner; Carl Landwehr;
Carlos Jose Pereira de Lucena;
Beng Chin Ooi; Loren Terveen;
Marshall Van Alstyne; Jeannette Wing

PRACTICE

Co-Chairs
Stephen Bourne
Board Members
Eric Allman; Charles Beeler; Bryan Cantrill;
Terry Coatta; Stuart Feldman; Benjamin Fried;
Pat Hanrahan; Tom Limoncelli;
Kate Matsudaira; Marshall Kirk McKusick;
Erik Meijer; George Neville-Neil;
Theo Schlossnagle; Jim Waldo

The Practice section of the CACM Editorial Board also serves as the Editorial Board of [queue](http://queue.acm.org).

CONTRIBUTED ARTICLES

Co-Chairs
Al Aho and Andrew Chien
Board Members
William Aiello; Robert Austin; Elisa Bertino;
Gilles Brassard; Kim Bruce; Alan Bundy;
Peter Buneman; Peter Druschel;
Carlo Ghezzi; Carl Gutwin; Gal A. Kaminka;
James Larus; Igor Markov; Gail C. Murphy;
Bernhard Nebel; Lionel M. Ni; Kenton O'Hara;
Sriram Rajamani; Marie-Christine Rousset;
Avi Rubin; Krishan Sabnani;
Ron Shamir; Yoav Shoham; Larry Snyder;
Michael Vitale; Wolfgang Wahlster;
Hannes Werthner; Reinhard Wilhelm

RESEARCH HIGHLIGHTS

Co-Chairs
Azer Bestavros and Gregory Morrisett
Board Members
Martin Abadi; Amr El Abbadi; Sanjeev Arora;
Dan Boneh; Andrei Broder; Doug Burger;
Stuart K. Card; Jeff Chase; Jon Crowcroft;
Sandhya Dwaeekadas; Matt Dwyer;
Alon Halevy; Maurice Herlihy; Norm Jouppi;
Andrew B. Kahng; Henry Kautz; Xavier Leroy;
Kobbi Nissim; Mendel Rosenblum;
David Salesin; Steve Seitz; Guy Steele, Jr.;
David Wagner; Margaret H. Wright

WEB
Chair
James Landay
Board Members
Marti Hearst; Jason I. Hong;
Jeff Johnson; Wendy E. MacKay

ACM Copyright Notice

Copyright © 2015 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

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

Subscriptions

An annual subscription cost is included in ACM member dues of \$99 (\$40 of which is allocated to a subscription to *Communications*); for students, cost is included in \$42 dues (\$20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is \$100.

ACM Media Advertising Policy

Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting <http://www.acm-media.org> or by contacting ACM Media Sales at (212) 626-0686.

Single Copies

Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@acm.org.

COMMUNICATIONS OF THE ACM

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

POSTMASTER

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

Printed in the U.S.A.



Association for
Computing Machinery





Moshe Y. Vardi

DOI:10.1145/2753507

Incentivizing Quality and Impact in Computing Research

Over the past few years, the computing-research community has been conducting a public conversation on its publication culture. Much of that conversation has taken place

in the pages of *Communications*. (See <http://cra.org/scholarlypub/>.) The underlying issue is that while computing research has been widely successful in developing fundamental results and insights, having a deep impact on life and society, and influencing almost all scholarly fields, its publication culture has developed certain anomalies that are not conducive to the future success of the field. A major anomaly is the reliance of the fields on conferences as the chief vehicle for scholarly publications.

While the discussion of the computing-research publication culture has led to general recognition that the “system is suboptimal,” developing consensus on how the system should be changed has proven to be exceedingly hard. A key reason for this difficulty is the fact the publication culture does not only establish norms for how research results should be published, it also creates expectations on how researchers should be evaluated. These publication norms and research-evaluation expectations are complementary and mutually enforcing. It is difficult to tell junior researchers to change their publication habits, if these habits have been optimized to improve their prospects of being hired and promoted.

The Computing Research Association (CRA) has now addressed this issue head-on in its new Best Practice Memo: “Incentivizing Quality and Impact: Evaluating Scholarship in Hiring Tenure, and Promotion,” by Batya Friedman and Fred B. Schneider (see <http://cra.org/resources/bp-memos/>). This memo may

be a game changer. By advising research organizations to focus on quality and impact, the memo aims at changing the incentive system and, consequently, at changing behavior.

The key observation underlying the memo is that we have slid down the slippery path of using quantity as a proxy for quality. When I completed my doctorate (a long time ago) I was able to list four publications on my CV. Today, it is not uncommon to see fresh Ph.D.’s with 20 and even 30 publications. In the 1980s, serving on a single program committee per year was a respectable sign of professional activity. Today, researchers feel that unless they serve on at least five, or even 10, program committees per year, they would be considered professionally inactive. The reality is that evaluating quality and impact is difficult, while “counting beans” is easy. But bean counting leads to inflation—if 10 papers are better than five, then surely 15 papers are better than 10!

But scholarly inflation has been quite detrimental to computing research. While paradigm-changing research is highly celebrated, normal scientific progress proceeds mainly via careful accumulation of facts, theories, techniques, and methods. The memo argues that the field benefits when researchers carefully build on each other’s work, via discussions of methods, comparison with related work, inclusion of supporting material, and the like. But the inflationary pressure to publish more and more encourages speed and brevity, rather than

careful scholarship. Indeed, academic folklore has invented the term LPU, for “least publishable unit,” suggesting that optimizing one’s bibliography for quantity rather than for quality has become common practice.

To cut the Gordian knot of mutually reinforcing norms and expectations, the memo advises hiring units to focus on quality and impact and pay little attention to numbers. For junior researchers, hiring decisions should be based not on their number of publications, but on the quality of their top one or two publications. For tenure candidates, decisions should be based on the quality and impact of their top three-to-five publications.

Focusing on quality rather than quantity should apply to other areas as well. We should not be impressed by large research grants, but ask what the actual yield of the funded projects has been. We should ignore the *h*-index, whose many flaws have been widely discussed, and use human judgment to evaluate quality and impact. And, of course, we should pay no heed to institutional rankings, which effectively let newspapers establish our value system.

Changing culture, including norms and expectations, is exceedingly difficult, but the CRA memo is a very promising first step. As a second step, I suggest a statement signed by leading computing-research organizations promising to adopt the memo as the basis for their own hiring and promotion practices. Such a statement would send a strong signal to the computing-research community that change is under way!

Follow me on Facebook, Google+, and Twitter.

Moshe Y. Vardi, EDITOR-IN-CHIEF

Copyright held by author.

SHAPE THE FUTURE OF COMPUTING. JOIN ACM TODAY.

ACM is the world's largest computing society, offering benefits and resources that can advance your career and enrich your knowledge. We dare to be the best we can be, believing what we do is a force for good, and in joining together to shape the future of computing.

SELECT ONE MEMBERSHIP OPTION

ACM PROFESSIONAL MEMBERSHIP:

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

ACM STUDENT MEMBERSHIP:

- Student Membership: \$19 USD
- Student Membership plus ACM Digital Library: \$42 USD
- Student Membership plus Print *CACM* Magazine: \$42 USD
- Student Membership with ACM Digital Library plus Print *CACM* Magazine: \$62 USD

- Join ACM-W:** ACM-W supports, celebrates, and advocates internationally for the full engagement of women in all aspects of the computing field. Available at no additional cost.

Priority Code: CAPP

Payment Information

Name

ACM Member #

Mailing Address

City/State/Province

ZIP/Postal Code/Country

Email

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc., in U.S. dollars or equivalent in foreign currency.

- AMEX VISA/MasterCard Check/money order

Total Amount Due

Credit Card #

Exp. Date

Signature

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

Return completed application to:
ACM General Post Office
P.O. Box 30777
New York, NY 10087-0777

Prices include surface delivery charge. Expedited Air Service, which is a partial air freight delivery service, is available outside North America. Contact ACM for more information.

Satisfaction Guaranteed!

BE CREATIVE. STAY CONNECTED. KEEP INVENTING.



Association for
Computing Machinery

1-800-342-6626 (US & Canada)
1-212-626-0500 (Global)

Hours: 8:30AM - 4:30PM (US EST)
Fax: 212-944-1318

acmhelp@acm.org
acm.org/join/CAPP



Vinton G. Cerf

DOI:10.1145/2749417

Cascade Failure

Most readers of this column are likely familiar with the notion of a *cascade failure* in which one failure triggers another and the process continues gaining strength until it produces

a catastrophic outcome. A particularly good example of this is the Northeast power failure of 2003^a about which was written:

“The blackout’s primary cause was a software bug in the alarm system at a control room of the FirstEnergy Corporation, located in Ohio. A lack of alarm left operators unaware of the need to redistribute power after overloaded transmission lines hit unpruned foliage, which triggered a race condition in the control software. What would have been a manageable local blackout cascaded into widespread distress on the electric grid.”

There are plenty of other examples of this kind, including the reactor meltdown in the Japanese Fukushima area in 2011^b in which a series of failures produced a negative reinforcement cycle ending in disastrous consequences. That this was triggered by the massive 9.0 Tohoku earthquake and subsequent tidal wave underscores the undeniable importance of trying to imagine the worst-case scenarios and asking how to design for their mitigation.

Ironically, my topic for this column is *not* about these kinds of massive failures but, rather, more subtle scenarios we might not immediately identify as triggers with serious consequences. This whole line of reasoning started when I had to replace a

tiny battery in one of my hearing aids. I have to do this every couple of days. The aids are in-the-ear type, so they are small, and the battery powering them must fit exactly into the enclosure provided. By good fortune and design, these are standard batteries made by a number of suppliers and can be found typically in most drugstores and hardware shops in the U.S. and probably elsewhere, at least in the so-called developed countries. So, what’s my point?

It occurred to me that if these battery makers might someday decide to abandon the product, these expensive hearing aids would be worthless. The more I thought about this, the more I began to think about the problems of specialization and obsolescence.

I have owned a lot of printers over time. For all practical purposes, they all used ink cartridges specially designed to be incompatible with other models. I recall railing about this to HP management once, and was told, “How do you think we make up for selling the printers cheaply? We lock you into our ink cartridges.” Of course, there *are* third-party suppliers of cartridges and even refill kits, but consumers are often warned about potential risks. It seems as if new models take different formats for the cartridges so even if you stocked up on older cartridges, the next printer you buy may not be compatible with them. This column is *not* a rant against HP or any other maker of printers, but an observation about the dependencies we are building into our increasingly

complex infrastructure that may come to bite us from time to time.

Anyone with a collection of DVD or CD-ROM disks will appreciate that some new laptops do not even have readers for these, let alone 5¼" or 3½" floppies. We cannot rely in the long term on specialized format devices being available. Some battery formats have had very long availability (D, C, AA, AAA, AAAA batteries), but others may be much harder to come by (often by design). The same can be said of various memory devices including USB memory sticks. Some interfaces change and adapters are needed to cope with older formats. There are analogues of incompatibility in the communications protocol world (for example, implementing IPv6 in addition to IPv4) where adaptation is impossible or awkward and clumsy at best.

So, what’s the point, you may ask? Essentially, I think it is worth some effort to pay attention to the nature of these dependencies, their scope, and the potential side effects when the supply of particular devices runs out. Preparing for that eventuality seems wise. There may be no tactic that saves the day forever. Stocking up on batteries may not work, especially if they are not rechargeable or will not hold a charge for years. I am sure you can think of many other examples.

It is impressive to what degree ingenuity can keep things running. Consider the 1950s automobiles in Cuba, or the possibility of some parts being made in 3D printers. Still, the main point is we may make big investments in things that are dependent on small but crucial items only to have those investments rendered useless for lack of a small component. Maybe I should have titled this column “for want of a nail...”

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

Copyright held by author.

a http://en.wikipedia.org/wiki/Northeast_blackout_of_2003

b http://en.wikipedia.org/wiki/Fukushima_Dai-ichi_nuclear_disaster

Abolish Software Warranty Disclaimers

CARL LANDWEHR WAS right to suggest in his Viewpoint “We Need a Building Code for Building Code” (Feb. 2015) that there should be a building code for software. Lawmakers should thus re-examine the common practice of allowing warranty disclaimers for commercial software. The software development industry claims it is simply too difficult to build correct software. But such a position looks increasingly absurd. A smartphone derives most of its functionality from gigabytes of software, though it is regarded as a device; an inability to make calls or take photos is rightly covered by consumer-protection laws. Smartphones are only the most obvious case; innumerable consumer products, including televisions and automobiles, depend crucially on software yet are protected by warranties. The only software subject to warranty disclaimers is, incredibly, actual software products.

As a first step toward abolishing warranty disclaimers, lawmakers should make software companies liable for faults caused by egregious programming malpractice. The most obvious example is buffer overruns and, more generally, failure to validate input data, whether from a user or from a file. An example of the latter is when a program crashes due to one of the possibly thousands of fonts on the machine being corrupt. Passwords stored in clear and other gross lapses of security engineering should be explicitly forbidden. Never forget the astonishing case of the Diebold voting machine, as reported by Douglas W. Jones of the University of Iowa in 2003, saying, “The encryption key F2654hD4 is present, in plain view, in the source code.”¹

In addition to forbidding specific instances of programming malpractice, a software building code should include the obligation to use static analysis tools and other techniques to improve the practice of software development.

Lawrence C. Paulson,
Cambridge, England

The most obvious example is buffer overruns and, more generally, failure to validate input data, whether from a user or from a file.

Reference

1. Jones, D.W. The case against the Diebold AccuVote TS. Presented at the USACM Workshop on Voter-Verifiable Election Systems (Denver, July 28, 2003); <http://homepage.cs.uiowa.edu/~jones/voting/dieboldacm.html>

Verify a Computation by Checking the Result

The title of Michael Walfish’s and Andrew J. Blumberg’s article “Verifying Computations without Reexecuting Them” (Feb. 2015) suggested a much more practical problem than the one these authors actually solved. When most users want to verify a computation, all they really want to do is check the result. They are not asking whether the algorithm is correct or if every step of a computation was in accordance with the algorithm. They want to know if the computed result is accurate enough to satisfy their requirements. This is often much easier than computing the result and far easier than verifying the correctness of a program; for example, verifying that a computed square root is close enough to the actual root is far easier than computing the root. One need only multiply the result by itself and compare the product with the

original number. Similarly, solving a set of equations may require a complex algorithm, but confirming that a proposed solution works is easy; one substitutes the values for the variables and evaluates the equations.

In general, the specification of a program is a predicate. If a program can evaluate the predicate, it can check the result. In many common situations, evaluating the predicate is a simpler process than finding the answer.

David Lorge Parnas, Ottawa, Canada

First Simplify Software

Vinton G. Cerf’s Cerf’s Up column “A Long Way to Have Come and Still to Go” (Jan. 2015) invited comments on progress in programming. Forty years of deliberately perpetuated excess complexity has caused great harm. Progress toward simplicity has been obstructed by anxieties about being simplified out of jobs, corporate revenue, and elite status. Computer languages used today are deficient compared with a 1973 design called “Prose” distributed within IBM based on each of the following seven simplicity-related criteria:

Flexibility of data structures;

Simplicity of expression. For example, a computation like `6 = count every state where populatn of some city of it > 1000000` requires at least twice those 14 tokens (words and symbols) when using today’s wide-purpose languages compared to what was operational in 1982. There is no enduring technical reason for using a wide-purpose technical language that fails to express this computation with comparable simplicity;

Generality of language applicability;

Modularity of fundamental information building blocks. Including iteration connections in the “atomic structure” of information is probably essential. They resolve a historical dichotomy between languages with flexible data and those with simple plural expressions;

Fluency of precise expression;

Durability of core language design; and

Potential for precise human-to-human communication. Simplification can extend the usefulness of precise language broadly into technical communication and education.

Workers and policymakers in software, education, cybersecurity, and public safety have shown almost no publicly visible aspiration to language technology by any of these criteria as advanced as was designed in the early 1970s at IBM and implemented for internal use a decade later at Digital Equipment Corporation. For more on these deficiencies, see “Inexcusable Complexity for 40 Years” at <http://users.rcn.com/eslowry>.

Students everywhere are taught how to arrange pieces of information by educators unaware of elementary pieces of information well designed for easy arrangement. Sound leadership could easily fix this deficiency. From airfoils to zippers, fundamental technological structures have received meticulous attention, but serious study of easily arranged information building blocks has been resisted for at least 30 years.

Thorough elimination of extraneous complexity can produce fluency, language generality, and convergence of core semantic design, especially the design of information building blocks, leading toward the practicality of an enduring lingua franca for technical literacy. Computer science curriculum planning would benefit from considering possible student reaction to learning about the deficiencies in current languages.

In its 2013 annual report, IBM reported 88.8% gross profit on \$25 billion in software revenue. Complexity in software discourages customers from switching vendors, thus keeping these numbers high. This is just one example of the magnitude of incentives to keep the world ignorant of serious simplification.

Leadership tolerance for the effort to clarify these issues has seemed meager for decades, while the technical literature has ignored thorough simplification. IEEE-USA publicly acknowledges the long delay in simplification but in a muted way. There

is thus an urgent need for organizations that value post-1974 competence in software simplicity to make their positions known.

Edward S. Lowry, Bedford, MA

Python for Beginners? Try Perl 6

Esther Shein’s news article “Python for Beginners” (Mar. 2015) tracked with the kind of articles on Python popularity I have been reading online for several years. Despite Python’s growing popularity, Perl has remained my scripting language of choice since 1993, even as I awaited Perl 6 (<http://perl6.org/>) to address the shortcomings of Perl 5. Perl 6 is usable today, and I have begun translating my latest Perl 5 code to Perl 6. I wonder what Shriram Krishnamurthi, the professor of computer science at Brown University quoted in the article, would think of Perl 6, since most, if not all, major Python shortcomings he mentioned are indeed addressed quite well in Perl 6.

Perl 6 would make an excellent replacement for Python as an introductory teaching language, whether in college or in middle school; I even recently proposed using it for an introductory scripting course in a summer science camp sponsored by the Okaloosa STEMM (Science, Technology, Engineering, Math, and Medicine) Academy (<http://www.okaloosaschools.com/stemm/>).

Tom Browder, Niceville, FL

Correction

The name “Blaise” inadvertently added [in brackets] in the context of the Pascal programming language in Esther Shein’s news article (Mar. 2015) was removed from digital versions of the article but was not caught in time to be removed from the print edition.

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

© 2015 ACM 0001-0782/15/05 \$15.00

Coming Next Month in **COMMUNICATIONS**

**A Profile of ACM’s
A.M. Turing
Award Recipient
Michael Stonebraker**

**Future Internets Escape
the Simulator**

**The Science of Managing
Data Science**

**Reliable Cron
Across the Planet**

**IllumiRoom:
Immersive Experiences
Beyond the TV Screen**

**Evaluation with
Ground Truth in
Social Media Research**

Colored Petri Nets

**Created Computed
Universe**

**Forgetting Made
(Too) Easy**

**An Interview with
U.S. CTO Megan Smith**

**The Computer Science
of Concurrency:
The Early Years
The Turing Award
Lecture from
2013 Recipient
Leslie Lamport**

Also, the latest news about parking and pocketsensing, capturing complex structures, and e-citizenship.

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:10.1145/2742480

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

Introducing Young Women to CS, and Supporting Advanced Research Environments

Joel Adams talks about starting a chapter of Girls Who Code, while Daniel A. Reed considers an issue with cyberinfrastructure.



Joel Adams
"Launching a New Girls Who Code Chapter"

<http://bit.ly/17zt0X2>

February 1, 2015

Our CS department recently launched a new Girls Who Code (GWC, <http://girlswhocode.com>) chapter. GWC is an after-school-club outreach that strives to introduce girls in grades 6–12 to computer science. Its goals include helping girls develop algorithmic thinking skills that can open up computing-related careers, helping them see how CS is relevant in today's world, making friends with other girls interested in technology, and having fun!

To achieve these goals, GWC provides a project-based curriculum with three levels. In each level, the girls use different technologies to complete a series of challenges, each of which introduces one or more new level-appropriate computing concepts. The curriculum is organized by semesters, and in

addition to weekly challenges, the girls work their way through projects each month and semester.

Our new chapter has generated some media coverage which led to inquiries from others interested in launching a GWC chapter, so I thought it might be helpful to enumerate the steps needed to launch a chapter:

1. Find an organization willing to **host** the meetings; it should have a space with computers and projection facilities. A computer lab designed for instruction can work well, at a high school, college/university, library, or non-profit organization. The Calvin College Department of Computer Science (<http://cs.calvin.edu/>) is our host organization.

2. Find a person from that organization willing to serve as the chapter's **advisor**. This person will be the liaison between the chapter and the organization, reserving space for meetings, arranging for access, and other local logistics. I am our chapter's advisor, so this was easy.

3. Find one or more **instructors**, volunteers knowledgeable about software development (having taken at least three college-level CS courses) and, ideally, women. I contacted several of our recent female CS grads working locally as software developers. One volunteered to be the lead instructor, and she recruited a female co-worker to help her.

4. Find a club **president**, who will submit the online application to launch a new GWC chapter, and will help to recruit other girls. This should be a female student (ideally high school, but middle school could work) proactive and motivated to learn about technology. I recruited an outstanding high school junior who had worked for me in our Imaginary Worlds Camps (<http://alice.calvin.edu/iwc/>).

5. Schedule **meeting times** in consultation with the instructor(s) and president. Find a time convenient for as many participants as possible. We chose to meet in the evenings, because many girls participate in other activities after school. The GHC curriculum requires eight hours of contact time per month, so we scheduled our club to meet for two hours each Monday evening.

6. Publicize the club to recruit **students**. We contacted local high schools, companies, community partners, and related outreach efforts (such as BitCamp, <http://bit.ly/1alfnwx>) to spread the word.

In #4 above, I mentioned our Imaginary Worlds Camps, one-week computing animation camps I have been directing each July since 2003. A week is too

short a time to build mentoring relationships with the students; students come for a week, learn some things, and we never see the majority of them again.

One of the most promising things about GWC is its potential to build sustained mentoring relationships between our instructors and these young women. Compared to a one-week camp, I believe the GWC weekly format holds much greater potential to mentor these young women and get them excited about technology. This is what motivated me to launch our GWC chapter.

It took time to get our club off the ground. After deciding in summer 2014 to pursue this, I recruited our club president and lead instructor in August, then the president had to submit the new chapter application. After that was accepted, I had to complete a form outlining my responsibilities as advisor, and our instructor had to complete a qualifying quiz to show GWC she had the necessary computer science knowledge. After GWC approved our new chapter, we could decide when to meet, set a launch date, and so on. We then began publicizing the chapter and recruiting students. As this was happening, our instructors worked through instructor-training materials GWC provides for its curriculum.

GWC recommends starting new chapters in January or September. We set our launch date for the second week of January, and our instructors created an online signup form to get some idea of how many girls they could expect (and how many snacks to buy). Twenty-one girls signed up, and 23 girls showed up for our first meeting! Attendance has stabilized at about 20. GWC recommends a minimum of five girls for a chapter, and a student:instructor ratio no greater than 20:1.

The girls in our chapter are enjoying themselves, and their parents are very appreciative. I received this note from a parent the day after our launch:

My girls, surly and whining, didn't want to go to GWC last night. Two hours later, smiling, excited, they couldn't stop talking. "I can't wait to work more in Scratch, you should see what I made; I think Scratch is already on my computer." Thank you for making this possible.

We think GWC holds life-changing potential for the young women in our chapter. While we hope some of them

pursue computing-related careers, we believe the algorithmic thinking skills they acquire will be beneficial no matter the career path they choose.

GWC is just one of many organizations seeking to catalyze change. If you have direct experience with other organizations that seek to bring greater diversity to our technical workforce, tell us about your experience. If you are interested in launching your own GWC chapter, I strongly encourage you to contact GWC and do so. Good luck!



Daniel A. Reed
"Lessons from Winnie the Pooh: Sustainable Cyberinfrastructure"

<http://bit.ly/1uIMju3>

Nov. 16, 2014

In his 1928 children's book *The House at Pooh Corner* (<http://bit.ly/1BZaN2L>), A.A. Milne introduces us to Tigger, a tiger (and friend of Winnie-the-Pooh) with unusual tastes and behavior, who lives in the Hundred Acre Wood (<http://bit.ly/1vBYQh9>). Disney brought Tigger to animated life in the 1968 movie *Winnie the Pooh and the Blustery Day* (<http://bit.ly/1JQmz3C>), in which Tigger sings the song "The Wonderful Things about Tigger," noting, "the most wonderful thing about Tiggers is I'm the only one. Yes, I'm the only one."

All too often, I find myself thinking about Tigger during conversations about cyberinfrastructure (<http://bit.ly/1FYGPNI>), the computing, storage, networks, software, and staff who support research data archives and computing systems. Why, you might ask? It is because we find it difficult to *replicate and sustain* cyberinfrastructure, whether at the international, national, regional, or campus levels, for the long periods needed to preserve digital artifacts and to reap the benefits of cross-disciplinary, longitudinal data fusion and multidisciplinary modeling. Each deployed system is like Tigger; it is the only one, unique in its characteristics and capabilities.

Paradoxically, the rapid ferment in computing is both a benefit and a hindrance. The benefits of change are obvious, yielding a cornucopia of new consumer devices and rich cloud services; an unprecedented scale and richness of data, made possible by high-resolution scientific instruments, social me-

dia and e-commerce interactions, and a dizzying array of environmental and biological sensors; and higher-performance computing (HPC) systems of ever-rising performance and capability. Every year, what was new and exciting quickly becomes passé, obviated by new technological innovations.

The hindrance of computing change is less obvious, but real. Any deployment of data archives or computing systems is quickly obviated by newer technology with greater capacity and lower cost. However, the consequences of this change are far more serious and deleterious than mere technological obsolescence. They shape our research psychology, lessen our commitment to continued reinvestment and sustainability, and minimize the most important aspect of sustainable cyberinfrastructure, the cadre of experienced staff with institutional and historical context who operate that infrastructure.

In this, we in computing are unusual, if not unique. Most other capital investments presume a 10-, 20-, or even 30-year capital depreciation schedule. Rare is the computer system still in operation after five years. Rarer still is the research data preserved for use by other disciplines when the data ceases to have value to either the research team that produced it or cognate disciplines.

The latter is worrisome, because the longitudinal value of research data often accrues to disciplines other than those where it was created. This is equally true for scientific research and for multidisciplinary attacks on wicked problems (<http://bit.ly/1EuVjI2>) in environmental sustainability and climate change, healthcare and aging populations, economic disruption, and social change.

Tigger is not the right role model for sustainable cyberinfrastructure. We need a new model that judiciously balances investment between the "next big thing" and sustainable cyberinfrastructure, for the latter is itself a crucial enabler of discovery.

As Winnie-the-Pooh would say, "Oh, bother!"

Joel Adams is a professor at Calvin College. Daniel A. Reed is Vice President for Research and Economic Development, University Computational Science and Bioinformatics Chair, and professor of Computer Science, Electrical and Computer Engineering, and Medicine at the University of Iowa.

© 2015 ACM 0001-0782/15/05 \$15.00

Is “Good Enough” Computing Good Enough?

The energy-accuracy trade-off in approximate computing.

DEMAND FOR INFORMATION technology continues to grow, and computers have become an integral part of life. Most U.S. households now own multiple computing devices: 58% of American adults own a smartphone and 42% own a tablet computer, according to Pew Research. Ongoing technological developments and the Internet of Things mean more aspects of our lives are being computerized and connected, requiring ever more processing of data. Despite advances in reducing the power consumption of devices and in enhanced battery technology, today’s computers continue to increase their energy use as the amount of computation increases, at a time when energy efficiency is being encouraged and demands on battery life increasingly scrutinized.

At the same time, as the drive to portability continues and hardware components become smaller, the amount of power they require to operate cannot be reduced to the same degree.

Increasingly tightly packed electronics cannot dissipate the proportionally larger amount of heat they generate, which can cause them to overheat and fail. This heat energy has



to be dissipated to protect the electronics, which requires yet more energy.

One solution to the problems of such diminishing returns and of smaller devices overheating due to the growing proximity of individual components is to reduce the amount

of energy used by a computing device by introducing less accuracy into its processing—a tactic known as approximate computing, where error in computation is acceptable. Reducing the energy gap between a 0 state and a 1 state takes less energy to switch

from one to another, but it increases the probability of a spurious switch.

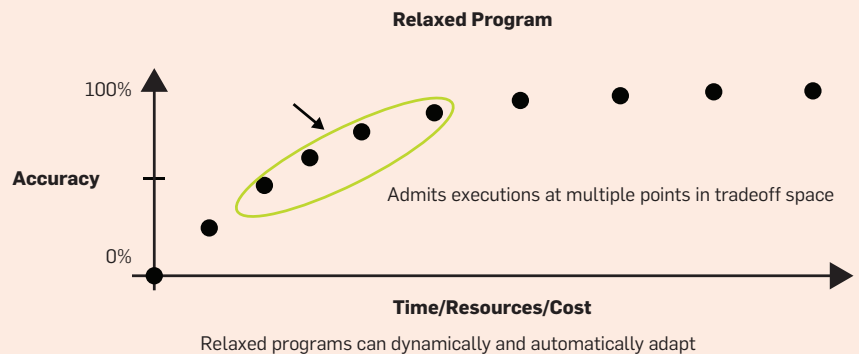
What Is Approximate Computing?

Historically, computer platform design has been a quest for ever-increasing accuracy, following the principle that every digital computation must be executed correctly. As Hadi Esmaeilzadeh and colleagues put it in their paper “General-purpose code acceleration with limited-precision analog computation,” “[c]onventional techniques in energy-efficient computing navigate a design space defined by the two dimensions of performance and energy, and traditionally trade one for the other. General-purpose approximate computing explores a third dimension—error—and trades the accuracy of computation for gains in both energy and performance.” They use machine learning-based transformations to accelerate approximation-tolerant programs.

V.K. Chippa and colleagues in Purdue’s Integrated Systems Laboratory are exploring scalable effort design to achieve improved efficiency (power or performance) at the algorithm, architecture, and circuit levels while maintaining an acceptable (and frequently, nearly identical) quality of the overall result. Chippa et al. (2013) acknowledged that “applications are often intrinsically resilient to a large fraction of their computations being executed in an imprecise or approximate manner,” described as approximate computing, or “good-enough” computing, with the aim of increasing efficiency/reducing energy consumption. The idea is that error-tolerant processes can be run on less-reliable hardware that operates faster, uses less energy, and/or is less likely to burn up.

Approximation is not a new idea, as it has been used in areas such as lossy compression and numeric computation; in fact, John von Neumann wrote a paper on it in 1956 (Probabilistic logic and the synthesis of reliable organisms from unreliable components, *Automata Studies* (C.E. Shannon and J. McCarthy, Eds.), Princeton University Press). According to a Computing Community Consortium blog post on the U.S. Defense Advanced Research Projects Agency (DARPA) 2014 Information Science and Technology (ISAT) Targeted

The cost–accuracy trade-off.



Source: <http://people.csail.mit.edu/mcarbin/>

Approximate Computing workshop, a number of researchers are working in this area.

How It Works

The main stages of design for approximate computing are (1) identifying those elements of an application that can tolerate error, (2) calculating the extent of error that can be tolerated, (3) discovering performance or energy savings, and (4) executing the instruction.

1. Where can errors be tolerated?

First, the kernels where error can be tolerated need to be identified. It is hugely time-consuming to identify all the combinations and their computational accuracy, plus the potential energy savings.

The early research of Sasa Misailovic and his colleagues at the Massachusetts Institute of Technology (MIT) Computer Science and Artificial Intelligence Laboratory focused on enabling programs to perform less work and therefore trade accuracy for faster, or more energy-efficient, execution. The team delivered compiler transformations that, for example, skipped regions of code that “are time-consuming, but do not substantially affect the accuracy of the program’s result,” says Misailovic.

At the 2013 Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) conference, the MIT team unveiled Rely (<http://mcarbin.github.io/rely/>), a language developed to indicate which instructions can be processed by less-reliable hardware, to a specified probability.

2. What is the tolerable error?

Introducing deliberate errors goes against the grain, but a certain degree of inaccuracy can be tolerated by the user in certain aspects of programming. One example is video rendering, where the eye and brain fill in any missing pixels. Other applications where a certain percentage of error can be tolerated without affecting the quality of the result as far as the enduser is concerned include:

- ▶ wearable electronics
- ▶ voice recognition
- ▶ scene reconstruction
- ▶ Web search
- ▶ fraud detection
- ▶ financial and data analysis
- ▶ process monitoring
- ▶ robotics
- ▶ tracking tags and GPS
- ▶ audio, image, and video processing and compression (as in Xbox and PS3 videogaming).

The common factor here is that 100% accuracy is not needed, so there is no need to waste energy computing it. But how much error is too much?

At last year’s OOPSLA conference, the same MIT team presented a system called Chisel (<http://groups.csail.mit.edu/pac/chisel/>), a simulation program that identifies elements of programming that can tolerate error, extending Rely’s analysis approach. Chisel can calculate how much error can be tolerated, evaluating the percentage of improperly rendered pixels at which the user will notice an error.

3. How can energy be saved?

As their contribution to the issue, Chippa

and other members of the Purdue group proposed the concept of Dynamic Effort Scaling, leveraging error resilience to increase efficiency. Recognition and mining (RM) are emerging computer processing capabilities anticipated on future multi-core and many-core computing platforms. To close the gap between the high computational needs of RM applications and the capabilities of the platforms, the Purdue group revealed at the International Symposium on Low Power Electronics and Design (ISLPED '14) its proposed energy-efficient **Stochastic Recognition and Mining (StoRM)** processor, which the group said will lead to energy savings with minimal quality loss.

Luis Ceze and his colleagues at the University of Washington (UW) have been working on approximate computing for more than five years, using a more coarsely grained approach to approximation than other researchers. One unique aspect of their work is hardware-software co-design for approximate computing. Control in modern processors accounts for a significant fraction of hardware resources (at least 50%), which fundamentally limits approximation savings. The UW team found that using limited-precision analog circuits for code acceleration, through a neural approach, is both feasible and beneficial for approximation-tolerant applications. The UW group's hardware model—SNNAP (systolic neural network accelerator in programmable logic)—assesses the effect of approximation output. It works with the neural network, accelerating approximate code, removing the need to fetch and decode individual instructions. Says Ceze, “Applications that we do well in the digital neural processing unit on FPGAs [field-programmable gate arrays] (that is, the SNNAP work) are financial analysis apps, robotics control systems, and computer vision. The analog version (<http://bit.ly/1zLkric>) also shows great promise in game physics engines and machine learning applications.”

In January, the UW group published in *Communications* on the technique of using neural networks as general-purpose approximate accelerators. Such a system chooses a block of approximate code and learns how it behaves using a neural net; then it involves the neural net, as opposed to executing the original code.

Chisel computes how much energy can be saved. By the simple expediency of allowing errors in processing, a computer's power consumption may be reduced by 9%–19%, according to the MIT research simulations. The amounts given by other researchers vary, but there are significant savings to be had.

4. How can the instruction be executed?

The mechanism in Rely is the use of an operator that indicates the instruction can be operated on unreliable hardware in order to save energy. Previously, the period had to be inserted manually, but Chisel inserts Rely's operators automatically, also guaranteeing maximized energy savings.

The Developer View

The counterintuitiveness of tolerating error is a common concern among developers, according to Ceze and his colleagues at UW.

The MIT team is developing rigorous approaches to help developers understand and control the approximation technique, in the process changing the perspective of many who were initially reluctant. “Some of these developers are excited by the promise of potential performance improvements and energy savings,” says Misailovic. “Others look to our techniques for a way to cope with future trends in the design of hardware circuits, which may require hardware to be less reliable. And still others see our techniques as providing novel ways to deal more effectively with software errors, which remain ubiquitous throughout our entire software infrastructure.”

Conclusion

The Rely and Chisel systems and others (such as Accept, which applies a variety of approximation techniques, including hardware acceleration; Flicker, which uses critical data partitioning to save refresh power, and Precimonious, which assists developers in tuning the precision of floating-point programs) have created the possibility of off-the-shelf programming that can identify where errors can be tolerated, indicate the degree of inaccuracy of computation that can be tolerated, calculate the energy that can be saved, and insert the operators that control

the computations. The savings in energy can be significant, with little noticeable loss in quality.

Research into approximate computing is still in its infancy. Error-tolerant applications combined with energy-efficient programming would seem to be the way forward, according to Baek and Chilimbi in an article about Green, their framework for energy-conscious programming.

The ongoing development of tools and frameworks continues to simplify the practical implementation of approximate computing in a number of ways, which means it would seem “good-enough” computing may be here for good. □

Further Reading

Baek, W., and Chilimbi, T.M. (2010).

Green: A framework for supporting energy-conscious programming using controlled approximation. *Proceedings of the PLDI* (pp. 198–209). <http://bit.ly/1vG9NOB>

Chippa, V.K., Venkataramani, S., Chakradhar, S.T., Roy, K., and Raghunathan, A. (2013, Nov.).

Approximate computing: An integrated hardware approach. *Asilomar conference on Signals, Systems and Computers, Pacific Grove, CA* (pp. 111–117). Washington, DC: IEEE Computer Society. <http://bit.ly/1DFIvKA>

Drobnis, A. (June 23, 2014).

ISAT/DARPA Workshop Targeted Approximate Computing. <http://bit.ly/1xZct9a>

Khan, A.I., Chatterjee, K., Wang, B., Drapcho, S., You, L., Serrao, C., Bakaul, S.R., Ramesh, R., and Salahuddin, S. (2014).

Negative capacitance in a ferroelectric capacitor. *Nature Materials*, December 15, 2014.

Moreau, T., Wyse, M., Nelson, J., Sampson, A., Esmailzadeh, H., Ceze, L., and Oskin, M. (2015).

SNNAP: Approximate computing on programmable SoCs via neural acceleration. *2015 International Symposium on High-Performance Computer Architecture*. <http://bit.ly/1DYWz9>

St. Amant, R., Yazdanbakhsh, A., Park, J., Thwaites, B., Esmailzadeh, H., Hassibi, A., Ceze, L., and Burger, D. (2014).

General-purpose code acceleration with limited-precision analog computation. *Proceedings of the 41st International Symposium on Computer Architecture*. <http://bit.ly/1wccH7D>

Logan Kugler is a freelance technology writer based in Tampa FL. He has written for over 60 major publications.

© 2015 ACM 0001-0782/15/05 \$15.00

Putting the Data Science into Journalism

News organizations increasingly use techniques like data mining, Web scraping, and data visualization to uncover information that would be impossible to identify and present manually.

THE KEY ATTRIBUTES journalists must have—the ability to separate fact from opinion, a willingness to find and develop strong sources, and the curiosity to ask probing, intelligent questions—are still relevant in today’s 140-character-or-less, ADHD-esque society. Yet increasingly, journalists dealing with technical topics often found in science or technology are turning to tools that were once solely the province of data analysts and computer scientists.

Data mining, Web scraping, classifying unstructured data types, and creating complex data visualizations are being utilized by news organizations to uncover data that would be impossible to compile manually, such as searching all Web retailers to find the average price of a particular item, given the vast number of potential sites to visit and the limited amount of time usually afforded to reporters on deadline. Additionally, the tools can be used to dig up or present data in a way that helps journalists generate story ideas, as well as presenting complex information to readers in ways they have not seen it presented before.

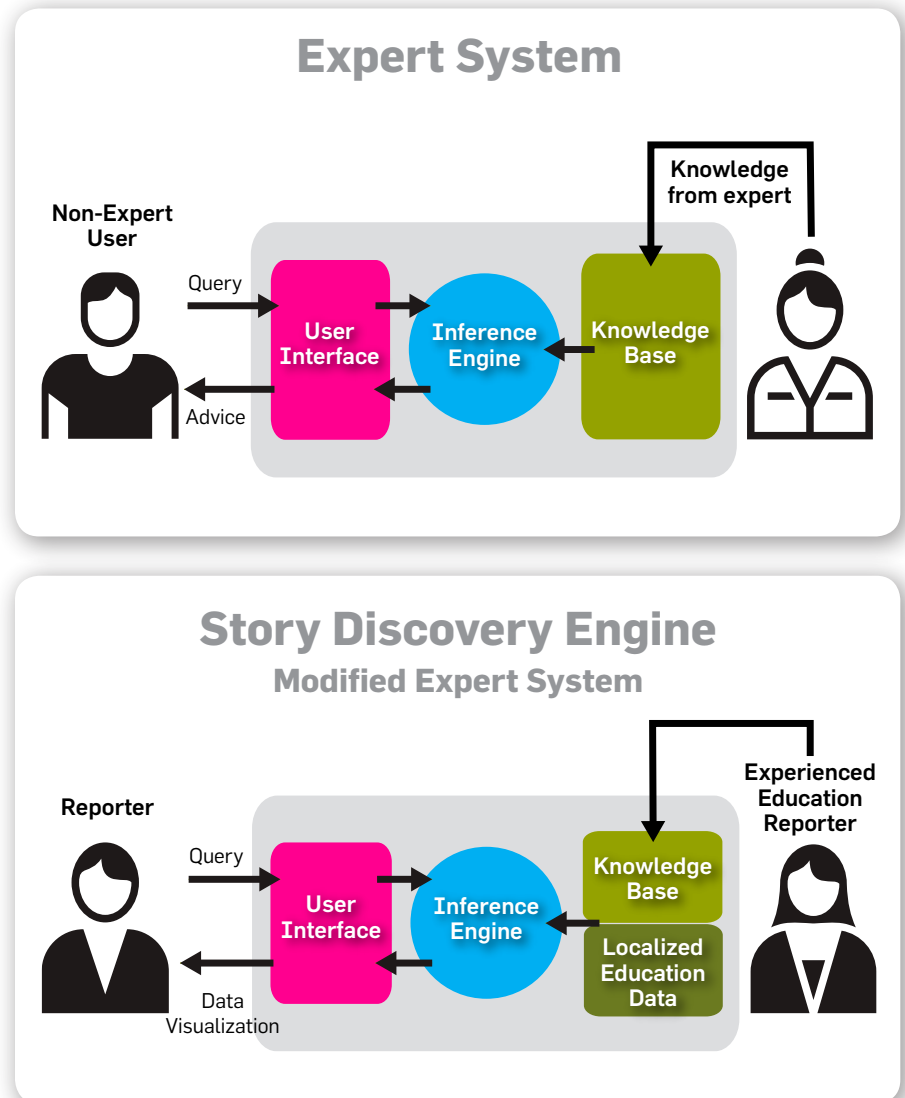
“There’s this whole realization that if news organizations are to attract an audience, it’s not going to be by spewing out the stuff that everyone else is spewing out,” says David Herzog, a professor at the University of Missouri and academic advisor of the National Institute of Computer-Assisted Reporting (NICAR), part of Investigative Reporters and Editors (IRE), a non-profit organization dedicated to improving the quality of investigative reporting. “It is about giving the audience information that is unique, in-depth, that allows them

to explore the data, and also engage with the audience,” he adds.

One of the most interesting examples of programming technology being used to augment the reporting process comes from reporter and Temple University journalism professor Meredith Broussard, who was

working on a series of stories for *The Atlantic* that focused on poor student test performance in the School District of Philadelphia.

As part of her investigative series, Broussard was trying to track down the number of textbooks in use in each public school in Philadelphia, and to



A depiction of a traditional expert system compared with the Story Discovery Engine. Graphic by Meredith Broussard and Marcus McCarthy.

see if the number matched the number of students enrolled. If there were not enough textbooks to go around, Broussard believed that could explain why so many students were performing poorly on tests directly tied to the curricula taught from those textbooks (a few main educational publishers design and grade the tests, and publish the books that students use to prepare for the tests).

“I wanted to do a story on whether there were enough textbooks in the Philadelphia schools,” based on the number of students enrolled, Broussard explains. However, “it turns out that it is a really complicated calculation, and there wasn’t software available to do that calculation for me.”

To support Broussard’s reporting, she built a data-analysis tool to crunch the complex data sets being analyzed for the article. To understand whether there really was one set of textbooks for each student, she needed to match the hundreds or thousands of student names associated with each school with book inventory lists (which could include various versions of each book and multiple titles within a set), and then repeat that process across the hundreds of schools within the district.

“The machine supercharges the reporter,” Broussard says, noting that humans are limited, in terms of time and processing power, in a way that machines are not. Broussard observes that one reporter generally can analyze a single data set, whereas a team of a reporter, analyst, and editor usually can handle two or three large datasets. “But I needed to crunch at least 15 datasets for the story I wanted to do, and there was no way to do it using existing technology, so I had to create new technology.”

In the end, Broussard’s analysis yielded some shocking results. The average Philadelphia public school had only 27% of the books in the district’s recommended curriculum, and at least 10 schools had no books at all, according to their own records. Some other schools had books that were extremely out of date.

The software tool used by Broussard was actually a prototype for the Story Discovery Engine, a tool that reporters can use to accelerate the process of finding investigative story ideas.

It takes a traditional expert model of assessing information (applying a reporter’s insight to a known knowledge base) and abstracts out high-level rules, then implements these rules in the form of database logic. When localized data sets are added, it then creates visualizations of the output that can show the reporter data anomalies or new avenues for exploration.

“You set up parameters, and then the system will spit out a visualization that will allow the reporter to come up with story ideas,” Broussard explains.

She is quick to note that no technology, not even the Story Discovery Engine, is designed to or could replace a human reporter’s instincts on what makes a good story. Says Broussard: “There’s no such thing as a machine that comes up with story ideas, and spits them out. You don’t actually want that, either, because computers are insufficiently creative. A human investigative journalist can look at the facts, identify what’s wrong with a situation, uncover the truth, and write a story that places the facts in context. A computer can’t. Investigative journalism plays an important role in preserving democracy; we need watchdog journalists to hold institutions accountable, to shine a light on misdeeds. The Story Discovery Engine allows investigative

“A human investigative journalist can look at the facts, identify what’s wrong with the situation, uncover the truth, and write a story that places the facts in context. A computer can’t.”

journalists to quickly uncover many stories on a topic as a way of getting broader and deeper coverage of important civic issues.”

Some of the traditional news sources that have embraced this new world of data visualization include *The New York Times*, *The Los Angeles Times*, *The Chicago Tribune*, and *The Washington Post*, each of which has dedicated a significant level of resources and staffing to augment traditional reporters, often creating entire teams of analysts, designers, and computer programmers to ensure the data uncovered during the course of researching a story can be presented in a way that is interactive and meaningful to readers, many of whom now expect to be able to interact with that story on a personal level.

One such example is the Dollars for Doctors project, a project conducted by ProPublica, a self-described “independent, non-profit newsroom that produces investigative journalism in the public interest.” The Dollars for Doctors project compiled the payments made to physicians by drug companies, and created an interactive tool that allows users to see exactly which payments were made to their doctor. The data itself was compiled from disclosures from 17 pharmaceutical companies, and then assembled into a single, comprehensive database.

Broussard says data visualizations and interactive tools are appealing to today’s news consumers because they involve the readers in the story and localize events occurring in the news.

“I don’t know where you live, so I can’t draft a story [specifically] for you and show you what’s happening in your neighborhood,” Broussard says. “What I can do, however, is create a news app that lets you search your own address and find what’s relevant to your neighborhood. Readers feel more engaged when they can put themselves into a story and find out what it means for their own lives.”

The use of data to generate stories or provide context or color within them isn’t new; reporters traditionally have had to dig through mountains of content to find statistics or data that helped drive or support a story. However, a key challenge faced by many reporters—particularly in the scientific field—is

quickly pulling together stories on deadline while trying to analyze a wide range of technical papers and reports.

That is where Science Surveyor, a project launched in May 2014 and funded by the David and Helen Gurley Brown Institute for Media Innovation at Columbia University, seeks to assist science journalists and improve reporting on deadline by using natural language processing algorithms to review and analyze scientific research papers, and then provide a more detailed picture of each paper's findings in terms of its views, funding source, and other key attributes that likely will help frame the paper and its results.

"It's meant to provide, on deadline, a more nuanced view [of] where a particular paper sits, is it part of a growing consensus or not, and how is it funded," explains Mark Hansen, director of the David and Helen Gurley Brown Institute for Media Innovation. Hansen says the tool is designed to sift through and analyze research papers, and then creates visualizations summarizing the papers' timeliness, sources of funding or support, and which are part of a consensus view and which are outliers in their findings. Then, reporters are able to integrate "a more nuanced view of the topic or a particular paper," Hansen says, noting that providing context for specific research findings is key to ensuring better science reporting.

Clearly, creating data visualizations, developing programs to scrape Web sites, or even creating a new application to mine specific databases are tasks that require programming skills,

Broussard says data visualizations and interactive tools are appealing to today's news consumers because they involve the readers in the story and localize events occurring in the news.

and not everyone is jumping on board.

"You still have division in newsrooms, where some people don't want to have anything to do with data," Herzog says. "Then you have people in the middle, with spreadsheet skills or some analysis, for enterprise stories. Then you'll have people who are really adept at learning new technologies, and who are pretty fearless at jumping in and trying to solve problems, and how to create a visualization."

It is the latter group of people that some of the top graduate journalism programs are seeking to attract. One such example is the Lede Program,

a post-baccalaureate certification program jointly offered by Columbia University's Graduate School of Journalism and Department of Computer Science. The Lede Program is designed to offer journalism students a background in data, code, and algorithms, each of which are becoming increasingly crucial to research and data-centric journalism.

"A good story is a good story, whether it has data or not," Hansen says. However, with noted journalism schools at the University of Missouri and Columbia University actively teaching data- and computer science-based skills, "my guess is that over time, the term 'data journalist' will disappear and it will just become journalism," Hansen says. **□**

Further Reading

Bainbridge, L.

Ironies of automation. New Technology and Human Error, J. Rasmussen, K. Duncan, J. Leplat (Eds.). Wiley, Chichester, U.K., 1987, 271–283.

The National Institute for Computer-Assisted Reporting
<http://ire.org/nicar/>

Broussard, M.

Artificial Intelligence for Public Affairs Reporting, Computational Journalism, <http://bit.ly/1DGxiGL>

The Data Journalism Handbook
<http://datajournalismhandbook.org/1.0/en/>

Laroia, A.

Web scraping: Reliably and efficiently pull data from pages that don't expect it, <http://bit.ly/1LNaCKk>

Keith Kirkpatrick is principal of 4K Research & Consulting, LLC, based in Lynbrook, NY.

© 2015 ACM 0001-0782/15/05 \$15.00

Milestones

ACM Bestows A.M. Turing, Infosys Awards

At press time, ACM announced the names of this year's recipients of two of its most prominent awards.

The A.M. Turing Award, ACM's most prestigious technical award, is given for major contributions of lasting importance to computing.

This year's ACM A.M. Turing Award is being presented to Michael Stonebraker, an adjunct

professor of computer science at the Massachusetts Institute of Technology (MIT), for "fundamental contributions to the concepts and practices underlying modern database systems."

The ACM-Infosys Foundation Award in the Computing Sciences recognizes personal contributions by young scientists and system developers to a contemporary innovation that, through its

depth, fundamental impact, and broad implications, exemplifies the greatest achievements in the discipline.

This year's ACM-Infosys Foundation Award is being presented to Dan Boneh, professor of computer science and electrical engineering at Stanford University and leader of the applied cryptography group there, for "the groundbreaking development of

pairing-based cryptography and its application in identity-based cryptography."

Both awards will be presented at the ACM Awards Banquet in San Francisco in June.

Communications will provide in-depth coverage of both award recipients in upcoming issues, beginning with interviews with Stonebraker in the June issue.

—Lawrence M. Fisher

Robots with a Human Touch

Empowering smart machines with tactile feedback could lead to tremendous new applications.

IN A NORTHEASTERN University lab in Boston, MA, a red and roughly humanoid robot named Baxter spots a USB cable dangling from a cord. Baxter, made by Boston's Rethink Robotics, reaches out and grasps the plug between two fingers, then slowly guides it downward. On the desk below, a USB socket is plugged into a power strip. A camera mounted on Baxter's arm helps the robot locate the socket; then the hand guides the USB plug into place, wriggling it briefly before completing the connection.

A job this delicate is difficult enough for people, let alone robots. Machines have long been able to use vision to identify the plug and socket, but to complete this task autonomously, Baxter relied on another type of feedback: a sense of touch. The robot was equipped with a new high-resolution GelSight tactile sensor. GelSight, developed by Massachusetts Institute of Technology (MIT) engineer Edward Adelson and put to work on Baxter with the help of Northeastern University roboticist Robert Platt, does not just tell the robot whether it is holding something in its hand; the sensor provides high-resolution feedback that allows the robot to identify the object from its shape and imprinted logo, then determine its orientation in space. As a result, Baxter can figure out whether it is holding the plug the right way and inserting it into the socket correctly.

Humans do the same thing when handling small objects; we rely on feel. In fact, the human hand is evidence that equipping robots with a sense of touch could be incredibly powerful, leading to new applications in surgery, manufacturing, and beyond. "We have in our own hands a proof that tactile sensing is very important," says Bill



Equipped with the high-resolution GelSight tactile sensor, a robot can grasp a cable and plug it into a USB port.

Townsend, CEO of robotics manufacturer Barrett Technology. "The question is how do we get from where we are with tactile sensors now to what we know can be possible based on what our own hands can do."

Feeling Around

Researchers have been working to develop powerful tactile sensors for decades, but few of these technologies have successfully transitioned out of the academic lab. "There have been lots and lots of designs, but nobody has ever come up with a really reliable tactile sensor," says Ken Goldberg, director of the Center for Automation and Learning for Medical Robotics at the University of California, Berkeley. "It's easy to build a camera and get high resolution at a low cost, but a sense of touch is extremely difficult.

It's just a very hard problem."

Platt hopes GelSight could be one possible solution. The sensor consists of a square of synthetic rubber coated on one side with metallic paint. The other side of the rubber is attached to a transparent piece of plastic equipped with light-emitting diodes (LEDs) and a small camera. The camera captures the LED light as it bounces off the layer of metallic paint, and a computer processes this data.

When the robot grips that USB plug, the layer of rubber and paint deforms as if the device had been pressed into a piece of clay. This change in the surface also changes the way the LED light is reflected. The computer interprets these alterations, generating a fine-grained picture of what the robot has in its hand. "It's able to feel the geometry of the surface it's touching in a very

precise way,” Platt explains. “If the sensor touches the back of your hand, you get an image showing that it sensed the hairs and the wrinkles in your skin. That kind of accuracy is really a first.”

Platt envisions a robot with this level of sensitivity accomplishing a range of fine-manipulation jobs, from assembling small parts in an electronics factory to changing the batteries in a smoke alarm. However, an effective tactile sensor also needs to be reliable, affordable, and robust enough to survive the wear and tear of the real world.

The Barrett Technologies robot hand illustrates still another challenge. One model of the BarrettHand features 24 touch-sensitive pads in each of its four fingers. When the robot grabs an object, researchers can analyze data from the force-torque sensors in the finger joints to register that item’s presence and guess its shape. The tactile pads provide even more finely grained detail, in part because they are distributed widely across the fingers. Yet Townsend says researchers struggle to incorporate that feedback; he suspects that may be because the software needed to make sense of that data has not yet been developed.

Modeling the World

Robotist Charlie Kemp and his team at the Georgia Institute of Technology in Atlanta have adopted a more software-centric approach to tactile sensing. They built their own sensors because they could not find any off-the-shelf models that met their particular needs, basing the hardware on a design by John Ulmen and Mark Cutkosky at Stanford University. Each sensor is about as wide as a Sharpie, and consists of alternating layers of conductive rubber, capacitive plates, foam, and steel mesh. Kemp and his team outfitted the entire arm of a humanoid named Cody with 384 of these tactile sensors, but they were not focusing on manipulating or identifying objects like Platt; instead, the researchers wanted their robot to use tactile feedback to learn.

The group set up several experiments in which the Cody robot was given a task, such as grasping or simply making contact with a small ball. The trick was that the ball was half-hidden in a thicket of house-

plants, tree trunks, or cinderblocks, or tucked behind a window with only select panes open. The traditional approach would be for the robot to study the scene visually, then make a plan and execute it, but with a cluttered setup like the one featuring the plants, that would have led to failure. “Based on just looking at the scene, the robot can’t tell what’s going on,” he explains. “It’s just this big patch of green leaves.”

In Kemp’s experiments, when the robot’s arm bumped against the cinderblock or the tree trunks, and the tactile sensors registered high forces from the pressure of these solid objects, Cody recognized it would not be able to move them, so it tried another route. When it brushed up against the fronds of the houseplants, the robot kept moving, making the assumption, given the low forces, that these were not real impediments. Finally, as it moved and made contact with the different objects, it started to build a virtual model of what to avoid and which paths were more promising. “Every place it makes contact, it alters its model a little,” Kemp says. “It’s making a map based on tactile feedback.”

Space, Health Care, and More

Kemp says that there is still plenty of work to be done on both the hardware and the software for tactile sensing, but addressing these challenges could lead to a whole new set of roles for robots. For example, at the Center for Automation and Learning for Medical Robotics, Goldberg has his eye on autonomous surgery, an application that will demand advances in both sensors and learning. Today’s da Vinci surgical robot does not include tactile feedback, so a doctor relies primarily on cameras when performing a remote operation. Surgeons would benefit from a sense of touch, but Goldberg says this feedback could also allow the machines to complete simple procedures on their own. In this case, developing the right sensor will only be the first step; the robots also will need to learn from their experiences. Goldberg pictures them uploading the data gathered during surgeries to the cloud, then allowing other robots to draw upon and learn from those experiences when per-

forming their own basic operations.

Several researchers mention that the U.S. National Aeronautics and Space Administration has taken an interest in tactile sensors as well. Platt notes that robots with tactile feedback could be used to build structures on distant planets in advance of an astronaut crew’s arrival; as with surgery, the added sensitivity would give the robots the ability to complete these jobs autonomously, without waiting for orders from human operators. Eduardo Torres-Jara, a computer scientist at Worcester Polytechnic Institute in Massachusetts, explains this was part of the problem with the Spirit rover on Mars; if the robot had been able to feel its way around, it might not have spent so long stuck in the Martian soil.

Whether these robots end up toiling in operating rooms, offices, factories, or homes, the researchers say tactile feedback will be critical if the machines are going to do real work. Kemp, for one, envisions a bright future for sensitive machines. “I’m optimistic that robots are going to be able to serve as 24/7 assistants for people from older adults to those with severe disabilities,” Kemp says. “And for those types of things, touch is going to be critical.”

Further Reading

Li, R., Platt R., Yuan, W., ten Pas, A., Roscup, N., Srinivasan, M., and Adelson, E.

Localization and Manipulation of Small Parts Using GelSight Tactile Sensing *IEEE Int’l Conf. on Intelligent Robot Systems (IROS)*, 2014.

Kemp, C., Edsinger, A., and Torres-Jara, E. Challenges for Robot Manipulation in Human Environments *IEEE Robotics and Automation*, March 2007, pgs. 20-29.

Jain, A., Killpack, M., Edsinger, A., and Kemp, C. Reaching in Clutter with Whole-Arm Tactile Sensing, *The International Journal of Robotics Research*, 2013.

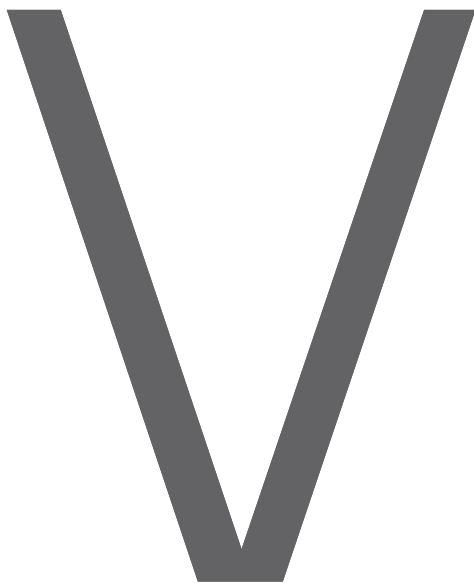
Khatib, O.

Mobile manipulation: The robotic assistant, *Robotics and Autonomous Systems Volume 26*, Issues 2–3, Feb. 28 1999, pp.175–183.

Automatic USB cable insertion using GelSight tactile sensing <http://bit.ly/1IyFHNA>

Gregory Mone is a Boston, MA-based writer and the author of the children’s novel *Dangerous Waters*.

© 2015 ACM 0001-0782/15/05 \$15.00



DOI:10.1145/2742689

Michael Schrage and Marshall Van Alstyne

Economic and Business Dimensions Life of IP

Seeking to balance intellectual property protection with incentives for investment in innovation.

INTELLECTUAL PROPERTY,” DECLARED Bill Gates, Microsoft founder and the world’s richest man, “has the shelf life of a banana.” Perhaps. But what if innovation in molecular biology, preservatives, or packaging could give bananas the unrivaled durability of Hostess Twinkies? Those intellectual property life extenders would not only preserve value, they would render Gates’s aphorism a stale anachronism.

The irony is that while any given intellectual property might be as perishable as a banana, IP protections are more like Twinkies. They last longer than might be healthy. The duration of such protections can—and do—privilege creators often at the cost of promoting progress in science and useful arts that the U.S. Constitution^a originally promised.¹

IP can be a platform for accelerating economic growth. If skills and know-how are complementary, then adding research to a knowledge base extends new resources to all in the network.⁴

^a Article 1, Section 8, Clause 8 of U.S. Constitution.



An exponential increase in patent applications, for example, indicates the growing interest in seeking competitive advantage. Likewise, the rise of code repositories on GitHub, founded in 2008, shows the opportunity provided by code sharing and recombination (see the accompanying figure). Global networks, digital, systems, and interoperable architectures that accumulate innovations just intensify policy debate about the rational balance between appropriate protection and fair use.

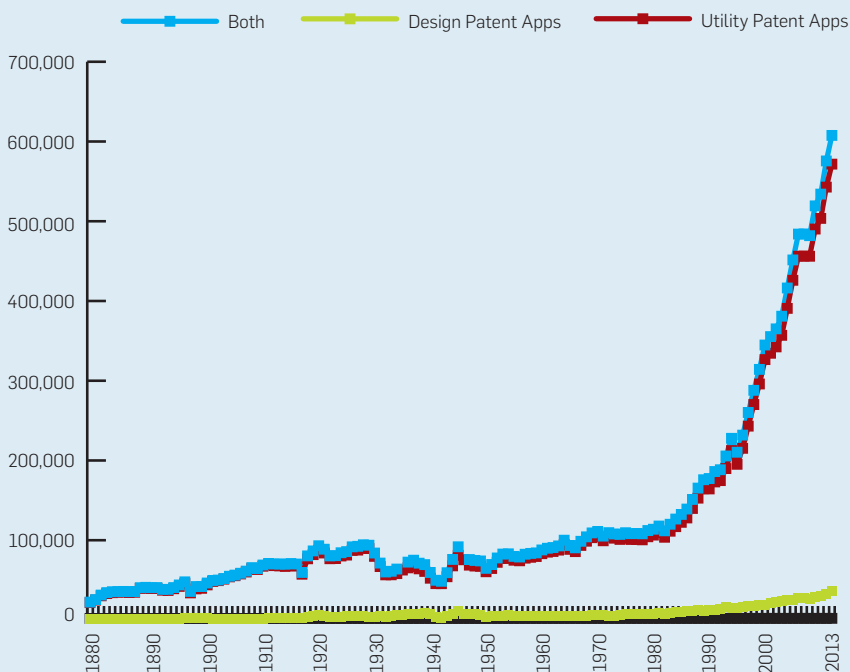
As the economics of innovation go exponential, IP protections can prevent more innovation than they protect. Stronger protection increases incentives to invent now but increases costs to invent later.³ Exponential growth then means marginal gains today have magnified costs tomorrow. When policy is more about protectionism than protection, it is inefficient and unwise. We need IP 2.0 policies that respect innovation realities.

Unfortunately, current legislation offers little coherent rationale for how long IP should be protected, whether for mobile apps, life-saving pharmaceuticals, novel materials, or Twinkified bananas. The lengths of IP protections appear divorced from basic economic principles of efficiency and effectiveness. Does policy respect how technological innovation really occurs? Not when there is too much regulatory fiat instead of market mediation. We learned this lesson when we stopped selling airwaves at fixed prices and made them more responsive to markets through spectrum auctions that promoted leaps in innovation. Such market mechanisms can make life—and innovation—better for both buyers and sellers. Innovation increasingly emerges both from “recombination” and “network effects.” Rigid, static, and unresponsive IP regimes that grant maximum protections for fixed lengths of time hurt innovators and customers alike. To paraphrase Clemenceau’s comment about war and generals, IP protections are too important to just leave to the lawyers. A networked world needs markets in IP protection.

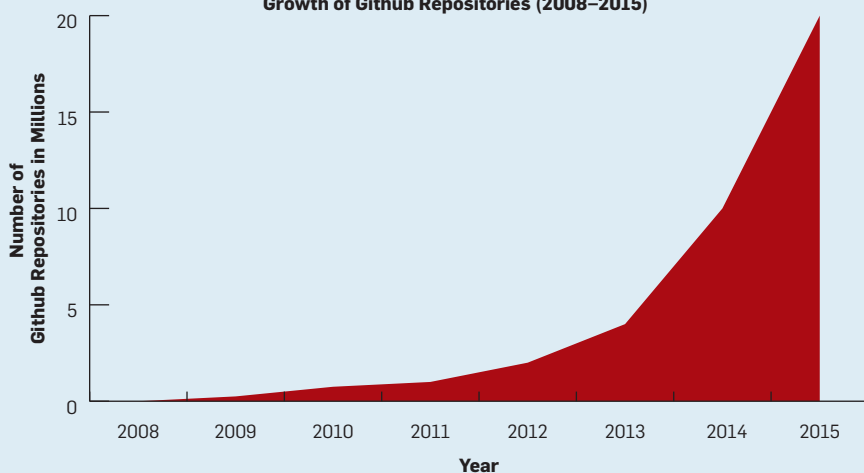
What Is to Be Done?

Economists typically remark that optimal IP policies balance complementary tensions: protections must

Patent applications* 1880–2013 and new software repositories** on GitHub show exponential growth.



Growth of Github Repositories (2008–2015)



* Source: USPTO http://www.uspto.gov/web/offices/ac/ido/oeip/taf/reports.htm#by_hist.

** Source: GitHub <https://github.com/blog/1724-10-million-repositories>; <https://github.com/about/press>.

be adequate to stimulate investment without conferring too much monopoly power.⁵ Balanced protection ensures future competition but not so much that an innovator can’t recover costs and still profit from their innovation investments. But what about opportunity costs associated with potential follow-on innovations in this era of APIs and greater interoperability? Balanced IP policies would recognize

and reward successful inventions that result from creative recombination of prior art. When ideas are combined with other ideas, opportunity costs can grow exponentially.⁷ For example, carbon nanotubes can be used not just for computing devices but also stronger-than-steel sporting goods, flame retardants, body armor, low-friction paints, electronic transistors, heat sinks, solar cells, hydrogen stor-

INTERACTIONS



ACM's *Interactions* magazine explores critical relationships between people and technology, showcasing emerging innovations and industry leaders from around the world across important applications of design thinking and the broadening field of interaction design.

Our readers represent a growing community of practice that is of increasing and vital global importance.



To learn more about us, visit our award-winning website <http://interactions.acm.org>

Follow us on Facebook and Twitter  

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



age, desalination, and delivery of cancer drugs. Innovation is cumulative; greater protection in one area can diminish and/or retard advancement in other areas.¹

What are the innovation ecosystem implications? Sooner or later, diminishing returns to individual innovation are overcome by the prospect of increasing returns to collective recombination. A smart and serious IP policy accounts for both market incentives and network effects. IP policy must balance protection with promotion, as the U.S. Constitution declared.

Proposal 1—Indefinitely Renewable Time Periods

Disney has a point in protecting Mickey Mouse—whose shelf life may well exceed even irradiated Twinkies. Certain rare works are so valuable that property rights motivate continued investment and careful husbandry. Landes and Posner, scholars of law and economics, note trademarks can be infinitely renewed to promote high quality and prevent brand confusion.² If IP terms are infinitely renewable most private works would move to the public domain as property rights owners stopped paying maintenance fees, while a few such as Mickey Mouse would receive continued investment and quality control from extended incentives. IP policies should not be confiscatory but neither should they represent protracted subsidies and free rides.

One-size-fits-all durations are too much like rent control for intellectual property just like those for physical property: government edicts, not markets, set terms. Such price fixing typically limits standards of living and curbs valuable investment.^b Dynamic IP 2.0 policies would be fairer, more efficient, and more effective than “rent control.” Inventors could place larger bets on innovation, knowing extension is possible if innovation succeeds. A renewable term would adjust easily across different types of markets such as those for designs, apps, and drugs. Renewal adjusts easily too within a market to designs that endure or fall out of fashion, to books that sell or get remaindered, and to drugs that do or die.

b <http://steshaw.org/economics-in-one-lesson/chap18p1.html>

Proposal 2—Exponential Renewal Fees

But infinity is a long time. If longer IP terms increase inventive effort but aggravate monopoly distortion, then infinity implies lots of work for lots and lots of distortion. Appropriate renewal fees must reflect opportunity costs. A testable, rational mechanism could help based on the simple exponential power of combinatorial innovation and network effects.

Suppose the fee starts at just \$1, then doubles every year. In year 20, the term under current patent law, the fee^c would be \$542,288. So at a 5% discount rate, the total net present cost would be \$436,796.^d Current USPTO practice applies renewal fees in years 3.5, 7.5, and 11.5, doubling only twice, and reaching a maximum of only \$7,400 for corporations.^e

The doubling period could also adjust by property type. For example, copyright fees might apply every third year. Under the 1831 law that allowed for a 42-year maximum copyright term, the final fee would have been \$8,192. This is quite reasonable after one has already profited from the work for more than two generations. In contrast, the far-less-reasonable Copyright Term Extension Act (CTEA) grants a term of “life plus 70.” Assuming an author lives for 35 years after producing a creative work, CTEA gives a term of 105 years. Doubling the fee every three years leads to a renewal cost in year 105 of \$17,179,869,184. Disney can give the geriatric Mickey a very long life but it becomes increasingly costly to imprison that life in a

c To start at \$1 rather than \$2, calculate 2^{19} in the twentieth year as \$542,288.

d $\sum_{i=0}^{19} (\frac{2}{1.05})^i = \$436,796$.

e <http://www.uspto.gov/web/offices/ac/qs/ope/fee010114.htm>

We need IP 2.0 policies that respect innovation realities.

tower of copyright. Exponentiation does its magic. In contrast to current practice, open renewal allows protection to adjust to true innovation value while exponential fees allow progress to adjust to true innovation costs.

The Point

Duration grows linearly while fees grow exponentially. Benefits and costs must cross. IP will enter the public domain at a time matched to economic value. And, markets—not regulators—decide.

Compare to What We Have

Patent terms date to colonial times. They represent a bargain that might not be rational today. Judge Giles Rich recounts that patent duration started as the length of two apprenticeships.⁶ Learning a trade as a blacksmith, cooper (barrel maker), or printer required seven years of training. If the apprentice invented anything of value, the master had the right to keep it. This was the master's bargain for having trained the apprentice; it also kept the master from being put out of business when the apprentice left. Congress extended patent duration to three apprenticeships in 1836. It later compromised at 17 years, between two and three terms, where the law stood for more than a century. U.S. law changed again in 1995, fixing the term at 20 years, to reconcile foreign and domestic practices. This change was not major, given a three-year average review cycle,^f because the term timing also changed to 20 years from date of filing from 17 years from date of issue.

Copyright law is no better. Current law provides a term of 70 years plus the life of the author as set by the 1998 Copyright Term Extension Act (CTEA). This law is waggishly known as the “Mickey Mouse Protection Act” based on the extensive lobbying efforts of the Walt Disney Company, which sought legislative action as copyright protection was due to expire. Sonny Bono championed Disney's cause in the U.S. House of Representatives as his own song “I Got You Babe” also received

f <http://patents.stackexchange.com/questions/5496/what-is-the-average-time-for-a-utility-patent-to-be-approved>

IP policy must balance protection with promotion, as the U.S. Constitution declared.

extended shelf life.

Many economists, including five Nobel laureates, opposed CTEA on the grounds it could not improve creation incentives retroactively on works already written. Further, CTEA could barely improve incentives prospectively for creators who would have long been dead. These economists filed an amicus brief accompanying the Supreme Court challenge to CTEA.^g No friend-of-the-court economists defended it. Yet the Supreme Court ruled 7 to 2 in favor of CTEA, addressing only the limited constitutional question of whether Congress had authority to extend the law retroactively and not whether it was foolish to do so.^h Politics won over prudence.

Benefits

The wisest social policy accounts for reasonable differences in IP duration for boot designs, smartphone apps, and lifesaving drugs. What is reasonable can vary dramatically across functions and across markets. Yet current policy is the same regardless of whether the boots are fads or timeless fashion, or the drugs are snake oil or streptomycin. The option to renew allows investments to adjust to market value. More options can create more innovations.

At the same time, proper policy must balance the promise of innovation against the problem of social costs. Opportunity costs rise increasingly as networks become more con-

g <https://cyber.law.harvard.edu/openlaw/eldredvashcroft/supct/amicus/economists.pdf>

h “... we are not at liberty to second-guess congressional ... policy judgments, however, debatable or ... unwise they may be.” Decision No. 01-618 in *Eldred vs. Ashcroft* (2003), 17.

ected and ideas can be recombined. Spillovers matter more in a world of bits than one of atoms. Exponentially increasing renewal fees move to match the patterns we observe in an economy of information. The proposed fee structure helps small inventors especially. Fees start very low and increase only as their ideas have time to prove their worth. Annual renewals also address the problem of orphan works—ones protected by copyright but whose authors cannot be found.

As a side benefit, such fees might also attack subterranean and non-practicing patents. Preventing others from using an idea if the owner does not use it becomes increasingly expensive. Geometric fees might not kill a patent troll, but the space under his bridge gets smaller and smaller with each passing year.

In any case, neither slaying patent trolls nor extracting greater fees from IP oligarchs is the point of our reform. The goal is to encourage inventors and IP holders to think more innovatively about how to profit from recombinant innovation and network effects. More efficient and more effective innovation creates opportunities for innovators and customers alike. Let IP protection be as dynamic as the market it serves. ■

References

1. Heller, M.A. and Eisenberg, R.S. Can patents deter innovation? The anticommons in biomedical research. *Science* 280, 5364 (1998), 698–701.
2. Landes, W.M. and Posner, R.A. Indefinitely renewable copyright. *The University of Chicago Law Review* (2003), 471–518.
3. Landes, W.M. and Posner, R.A. *The Economic Structure of Intellectual Property Law*. Harvard University Press, 2009.
4. Moenius, J. and Trindade, V. Networks, standards and intellectual property rights. Chapter 5 in *Frontiers of Economics and Globalization*, H. Beladi and E. Choi, Eds., (2007), 157–198.
5. Scherer, F.M. Nordhaus' theory of optimal patent life: A geometric reinterpretation. *The American Economic Review* (1972), 422–427.
6. Schrage, M. Archaic patent laws need to be rewritten. *Los Angeles Times* (Oct. 24, 1991).
7. Weitzman, M.L. Recombinant growth. *Quarterly Journal of Economics* (1998), 331–360.

Michael Schrage (schrage@mit.edu) is a research fellow at the Center for Digital Business at MIT Sloan School of Management and a visiting fellow in the Innovation and Entrepreneurship program at Imperial College in London.

Marshall Van Alstyne (mva@bu.edu) is a professor in the department of management information systems at Boston University and a research scientist at the MIT Center for Digital Business; Twitter: InfoEcon.

Education

What Are We Doing When We Teach Computing in Schools?

Research on the cognitive, educational, and policy dimensions of teaching computing is critical to achieving “computer literacy.”

LET ME TELL you a story. When my son was in fifth grade, we went to a parent-teachers’ evening. If you have ever been to one of these, you know the drill. You cross the school-gate line and trespass into classrooms that are normally forbidden territory. You gamely insert yourself into chairs designed for considerably smaller people. You walk around admiring nature tables, number lines, and colorful compilations of work accumulated over the school year. And as we wandered and admired we saw a colorful poster that said “SCIENCE: Gravity makes heavier things fall faster.” This is what happened next:

“That’s wrong.”

“It’s in the National Curriculum.”

“I don’t care: that’s wrong.”

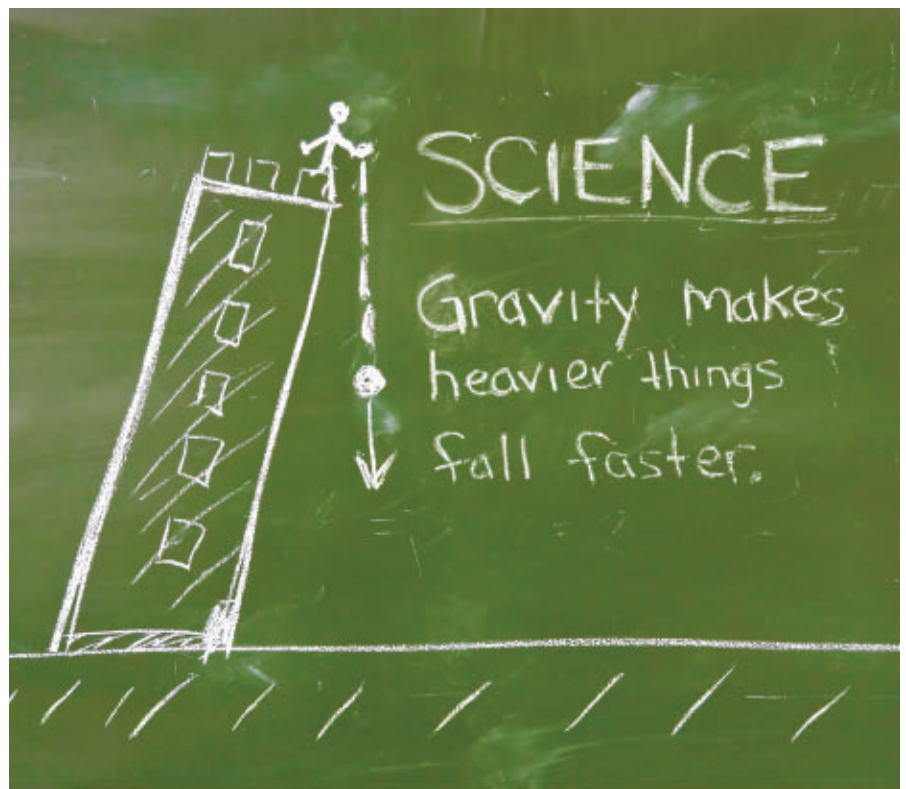
“But we did the experiment!”

“It’s wrong. You have to take it down. Now.”

At which point, conflict-averse, I walked away.

Every Teacher Must Teach

This is both a true story and a cautionary tale. It illustrates a considerable concern, which I believe we all should share, for the push to get computing taught in schools. Whenever subject matter is taught by non-specialists—and, of course, most school teachers are not computing specialists—disciplinary



understanding will be compromised. School teachers use materials created by other people and do the best they can to incorporate them into their classes. My son’s teacher did not deliberately set out to teach something wrong. She took materials available to her and tried to make them work. The U.K. National Curriculum said fifth-grade children should be taught about forces and mo-

tion. It said, “objects are pulled downwards because of the gravitational attraction between them and the Earth.”² Perhaps in “the experiment” they sent two children to the top of a slide, one with a heavy cricket ball and one with a light supermarket plastic bag. On the ground there were children with stop-watches. Someone shouted “Go!” The children dropped their objects. Other

children pressed their stopwatches. The ball fell to the ground. Some children shouted “Stop!” The wind caught the plastic bag and blew it to the other side of the playing field. Other children shouted “Stop!” Maybe they did it again, with different children. And then they went indoors, charted their results, and wrote the poster. Perhaps the children enjoyed the experience. They might have gone home and said, “we did science today.” The teacher had a good lesson. Except the poster remained on the wall, unchallenged, for the greater part of the school year and these 20 years later may still remain, unchallenged, in some minds: we learn wrong things just as hard as we learn correct things.

Every Child Must Learn

There is a current rhetoric that all children should be taught computer literacy. I think we would do well to pause and ask what it means to be “literate.” In traditional terms, we agree it is important that everyone should be able to read and write—and that if they cannot, they are disempowered, unable to participate in society. It is not, of course, easy. Children do not learn at the same rate, or in the same way, but being able to read is so important a skill that everyone must have it—even those with no talent for it, or appetite to learn. So how do we teach *everyone* to read? Historically, we have tried several things.

Simplifying the Spelling: Initial Teaching Alphabet

The “initial teaching alphabet” (ITA) was developed by James Pitman in the 1960s and introduced into U.K. schools. At root, it recognized that English spelling is extremely irregular (consider: *though*, *rough*, *bough*, *through*, and *thorough*) and this syntactical barrier made it difficult for children to learn to read. The ITA consisted of 44 “characters” that represented the sound of words, regularized spelling, and so simplified complexity: thus “Initial Teaching Alphabet” became *inifshul tēchīng alfubet*. Books and materials were produced to support this, so children could follow the adventures of corky’s enjnj or share the problems of the pengwin hω cōdn’t paddl

Did it work? In 1963–1964, a study compared 873 children who learned to read and write in the traditional way

with 873 children who were taught using the ITA.¹ The good news was that children learned to read faster with ITA: the bad news was that children could not transfer that learning to traditional books. A teacher recalls “I was in my second year’s teaching in Luton in 1968. ITA seemed to be a brilliant way of pushing the children on and they learned to read much earlier than usual. But—and the ‘but’ is enormous—some could not make the transition. I don’t think they’ll ever unlearn ITA spelling.”³

Simplifying the Vocabulary: “Look and Say”

Another approach took the view that it was not sensible to change the texts that children were presented with. In “look and say” children were taught to memorize high-frequency words as whole shapes: it was thought that 30 repetitions of a word would be enough for a child to learn it. So, books were produced that introduced careful sequences of words with considerable repetition. These stories featured docile children who played all day and had a curiously leaden way of talking to each other:

John, see the aeroplanes. One, two, three aeroplanes. I can see three aeroplanes. John said ‘See the aeroplane go up. See the aeroplane fly. The aeroplane can fly fast. Fly fast, big aeroplane’⁴

By the end of which the reader had seven exposures to *aeroplane*.

Did it work? Morag Stuart reports an experiment she conducted with colleagues in 2000 to see how easy it was for five-year-old beginning readers to store new, whole words in memory from repeated reading of whole word texts.⁵ They worked with 16 new words.

“After the children had seen and read each word 36 times, no child was able to read all 16 of them, and the average number of words read correctly was five. We were quite shocked by this.”

“When we tested children’s ability to read words they’d experienced more than 20 times in their school reading, on average they could read only one word correctly.”

Computing in Schools

So, those people who call for “computer literacy” must believe it is as important for our children to learn computation as it is for them to learn how to read. To

Calendar of Events

May 7–8

ANCS ‘15: Symposium on Architectures for Networking and Communications Systems, Oakland, CA, Co-Sponsored: Other Societies, Contact: Gordon Brebner, Email: gordon.brebner@xilinx.com

May 16–24

37th International Conference on Software Engineering, Florence, Italy, Sponsored: SIGSOFT, Contact: Antonia Bertolino, Email: antonia.bertolino@isti.cnr.it

May 18–21

CF’15: Computing Frontiers Conference, Ischia, Italy, Sponsored: ACM/SIG, Contact: Valentina Salapura, Email: salapura@us.ibm.com

May 19–22

MobiSys’15: The 13th Annual International Conference on Mobile Systems, Applications, and Services, Florence Italy, Sponsored: ACM/SIG, Contact: Gaetano Borriello, Email: gaetano@cs.washington.edu

May 20–22

GLSVLSI ‘15: Great Lakes Symposium on VLSI 2015, Pittsburgh PA, Sponsored: ACM/SIG, Contact: Alex Jones, Email: akjones@ece.pitt.edu

May 26–29

CCGrid ‘15: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, Co-Sponsored: Other Societies, Contact: Xian-He Sun, Email: sun@cs.iit.edu

May 31–June 4

SIGMOD/PODS’15: International Conference on Management of Data, Melbourne VIC Australia, Sponsored: ACM/SIG, Contact: Timoleon K. Sellis, Email: timos.sellis@rmit.edu.au

me, that seems unlikely ever to be the case. But, if that's their goal, I know it cannot be achieved by enthusiastic effort (however expert) happy to engage mainly the “gifted and talented” in forums of their own choosing, in voluntary “after-school” clubs, or via subscription services like Bitsbox, which sends out monthly coding projects for children with the educationally naive pitch “Learning to code takes time and practice. Sometimes it's just plain hard. But that's okay—we know kids are willing to work at learning hard things as long as they don't get bored along the way.”

Restricting the Syntax

In historical symmetry we can see a similar response to coding literacy as in traditional literacy. Block-based languages (like Blockly), microworlds (like Alice), and “Initial Teaching Environments” such as Scratch, reduce the syntactic complexity of coding to allow a more direct access to fluency. But they suffer the same problems as any “cut down” approach and the question of transition to “grown up” languages, remains as potent with them as it was for the ITA and reading. Are we teaching an ITC—an *Initial Teaching Computing*—that, in avoiding “obvious” difficulties, leads to problems later?

Restricting the API

Other approaches believe it is more appropriate to use real syntax, but constrain the environment to a particular (attractive) problem domain so learners become fluent in a constrained space. Event-driven environments (such as Greenfoot) or scaffolded systems (like Processing.js) aim for the learner to develop an accurate mental model of what their code is doing, and ultimately transfer that to other environments. Although whether they actually do so remains unclear: we may be restricting things in the wrong way.

Still others hold that coding—however approached—is insufficient for literacy and advocate a wider approach, taking in “computational thinking,” for instance as embedded in the framework of the “CS Principles”: Enduring Understandings, Learning Objectives, and Essential Knowledge.

What is resolutely held common with traditionally formulated literacy is that these approaches are unleashed

Are we teaching an ITC—an *Initial Teaching Computing*—that, in avoiding “obvious” difficulties, leads to problems later?

on classrooms, often whole school districts, even into the curriculum of entire countries—with scant research or evaluation. And without carrying the teachers. If we are to teach computing in schools we should go properly equipped. Alongside the admirable energy being poured into creating curricular and associated classroom materials, we need an accompanying set of considered and detailed programs of research, to parallel those done for previous literacies.

In complement to efforts in mathematical and natural language education we need to undertake *cognitive research* to discover how children acquire computational concepts asking questions (for example) as to whether there is a “best order” for the presentation of concepts, or whether pedagogically focused “initial programming environments” are a more productive way to learn than “real language” teaching. And, if so, under what conditions? This is not virgin territory, but the majority of previous work has been on learning in cognitively mature undergraduates, and that is unlikely to transfer directly.

In parallel, we need a program of *educational research* to support teachers, to ensure ideas work in real classrooms and with real teachers—and so we do not repeat cycles of error. At the moment, teachers are faced with a plethora of plausible approaches and no way to choose between them but the conviction (and charisma) of their inventors. A recent *Computing at School* magazine (Autumn 2014) is not short of ideas: *A four step scaffolding exemplar using Scratch... A simple project utilizing the python turtle library... Functional programming: an example in VB*. Each of these is a response to the need for teachers to

have something to teach, to be able to fill their lessons with engaging and useful material. But, at the same time, the evidence these are based on is solely “Do it like this! It works for me!”

Finally, we need *policy research* so we may effectively coordinate and disseminate practices at scale. It is not only individuals who can learn from research—districts, countries, and governments can, too.

Moral

It is tempting to think these are not true problems, that my opening story is an artifact of history, and that the attention of intelligent people, the support of professional organizations, and the commitment of money being applied to making our children computationally competent is sufficient for success. But I was recently talking with an industrialist who has a considerable commitment to outreach work. He was describing going into sessions for training school teachers (for children aged 5–11) and said “When I talk about computational thinking, they look horrified.”

And what happens when companies stop donating their staff's volunteer effort? When the spotlight of governmental attention passes on? Without the scaffold of *evidence* we risk condemning both these teachers, their pupils, and our large-scale efforts (with Scratch or Alice) to failure (like ITA or look-and-say), or to be successful only in certain localities under certain conditions.

When you frame a subject as *literacy*, the educational problems that entails are very different to the problems of subject experts enthusing an engaged minority. We should learn from educational history, and—this time—do the research. ■

References

1. Bell, M. The significance of the ITA experiment. *Journal of the Simplified Spelling Society*, 29 (2001).
2. Department for Education and Employment. *The National Curriculum: Handbook for Primary Teachers in England*. HMSO, 1999.
3. Lane, M. Educashunal lunacie or wizdom? 2001; <http://news.bbc.co.uk/1/hi/uk/1523708.stm>.
4. O'Donnell, M. In Munro, R., and Warwick, M., Eds. *Janet and John ...* James Nisbet and Company, Welwyn, 1959.
5. Stuart, M. Learning to read the words on the page. In Lewis, M. and Ellis, S., Eds. *Phonics: Practice, Research and Policy*. Paul Chapman, London, 2006.

Sally Fincher (S.A.Fincher@kent.ac.uk) is a professor of computing education and director of graduate studies at the University of Kent.

Copyright held by author.

Law and Technology

Oracle v. Google: A High-Stakes Legal Fight for the Software Industry

Copyrighting APIs threatens significant harm to both consumers and the software industry.

RECENTLY, THE U.S. Supreme Court has been hearing an unusually large number of intellectual property cases. And it has issued opinions, in cases like *Alice v. CLS Bank* and *ABC v. Aereo*, that have made important changes to the patent and copyright laws. But now the Supreme Court may wade into a case that could prove far more important than any of the Court's recent forays into intellectual property. That case, *Oracle v. Google*, involves a foundational question about the copyrightability of software. As I write this column, the Court has not yet decided whether to take the case, but that decision will almost certainly be made by the time this column appears. I will try to explain here why it is important the Court take this case, and, even more importantly, that it overturn the lower court's ruling holding that software APIs are copyrightable. If the Court leaves the lower court's ruling in place, both the software industry and consumers are very likely to pay a steep price. (*Communications* columnist Pamela Samuelson wrote expertly about this case in her March 2015 Legally Speaking column—take a look back at her column if you want the complete picture).



The Origins of Java

Back in the mid-1990s, Sun Microsystems and a group of collaborators—including, importantly, Oracle—developed the Java programming language. Sun’s goal in developing Java was clear—to overcome Microsoft’s Windows monopoly.

How? By adopting a strategy of openness. Windows is a proprietary, closed platform. Sun recognized that it could not displace Windows with another platform that worked in the same way. Instead, Sun made Java open and free for anyone to use.

Sun’s strategy was to create a “Java community,” and to declare all specifications would be decided in the open and all players would have equal access. The foundation of this strategy was a promise that the Java applications programming interface (API) would be open, so anyone could use Java to create cross-platform software. APIs are the parts of a computer program that other programs interact with. If you want programmers to write programs that are compatible with your software, you ensure they have access to your program’s APIs.

Having open and attractive APIs was a particular priority for Java. As Sun’s then-CEO Jonathan Schwartz explained, “[w]e wanted to basically build the biggest tent and invite as many people as possible.” That way, “when [programmers] at Oracle or SAP write an application, it can run on an IBM computer ... It can run on a Sun computer. It can run on any computer that runs Java. And that was our way of bypassing the monopoly.”

How things have changed. Oracle acquired Sun in 2010. Following the acquisition, Oracle has taken a very different path. It has taken steps to close the Java API that Sun formerly evangelized as open. And Oracle is relying on copyright law to do the job.

Oracle’s Copyright Strategy to Close the Java API

The focus of Oracle’s effort is a lawsuit the company filed against Google. In the lawsuit, which was filed shortly after Oracle’s acquisition of Sun, Oracle charges Google violated Oracle’s copyrights in Java when it included certain Oracle code in the implementation of Java the Internet search giant uses in

its Android mobile operating system.

It is important to understand what code Google copied from Oracle, and what Google did not copy. Java is composed of a set of pre-written programs that allow programmers to efficiently build software by assembling pieces of code that execute commonly used core operations. These chunks of code are called “methods,” and each method performs a function. The methods are grouped into “classes,” and classes are further grouped into “packages.”

The computer code for each method consists of “declaring code” and “implementing code.” The declaring code attached to any method is, in reality, nothing more than a name for that method. These names are structured in a particular format: `java.package.Class.method(input)`. An example would be `java.lang.Math.max(1,2)`. This bit of declaring code refers to a particular method (`max`), situated within the `Math` class and the `lang` package, that, when executed, returns the greater of two numbers.

The actual operation is executed by the method’s implementing code—the declaring code performs no function other than to instruct the computer to execute the implementing code for the method it names.

And yet the declaring code is key to the utility of Java. Programmers working in Java do not need to know the implementing code for any particular method they wish to use in software they are authoring. Instead, programmers simply input the declaring code. Using the name within software written in Java results in the software performing the function. In this way, a

What is the business motivation behind Oracle’s claim? The entire thing boils down to money.

programmer uses declaring code to operate the methods.

Java is an efficient way to build software, and over the years it has attracted a large community of developers who have become fluent in Java’s declaring code—that is, they have learned the names for many of the hundreds of methods available in the Java library. Which is why Google decided to use Java as the platform upon which developers build Android apps.

Google’s Android software contains 168 packages of methods. But Google did not copy any of Oracle’s implementing code. Instead, Google wrote or acquired its own implementing code for all the Java methods employed in Android. Oracle’s lawsuit focuses on declaring code. Google used the same names for the methods contained in 37 of the Java packages used in Android. Oracle says using the same names is copyright infringement.

What is the business motivation behind Oracle’s claim? The entire thing boils down to money. Oracle wants Google to pay to license Java rather than allow Google to author its own version of the implementing code. But Oracle cannot stop Google from writing its own implementing code, so they are using an indirect strategy. Oracle is trying to use copyright law to block Google’s ability to use the Java declaring code. If they succeed, Google will be in a very tight spot. Java developers know the declaring code—that is, the names—for the Java methods. If Oracle can force Google to use different names, it will, at a stroke, make thousands of developers suddenly illiterate in Google’s implementation of Java.

Or, to put it another way, think of the declaring code as a language—say, Spanish. Oracle wants to force developers to write for Android to learn to speak Portuguese. Which will make it much more difficult and expensive for developers to write apps for the Android platform. And that will give Oracle leverage to get millions in licensing fees from Google.

Oracle’s Copyright Claims Fail Their First Test

In the beginning, Oracle’s copyright claim did not fare so well. Federal District Judge John Alsup—a coder him-

self—dismissed Oracle’s suit. Judge Alsup found that programmers used Oracle’s declaring code to *operate* the Java methods. And, as a consequence, Oracle’s declaring code was an uncopyrightable “method of operation.”

As Judge Alsup noted, “methods of operation” are uncopyrightable under the section of the copyright law that distinguishes between copyrightable expression and uncopyrightable ideas. Methods of operation can sometimes be patented. But they are outside the jurisdiction of the copyright laws.

Oracle argued the declaring code was not a method of operation, but rather a “taxonomy”—that is, system for naming things. In Oracle’s view, taxonomies may be copyrighted if they are creative. And Java’s declaring code, Oracle argued, is creative. There is nothing inevitable, Oracle asserted, about the `java.package.Class.method(input)` structure of Java declaring code. Oracle could have chosen some other structure for declaring code. And because Oracle had choice in how to structure declaring code, its decisions regarding that structure were necessarily creative.

Judge Alsup accepted Oracle’s arguments, but held they did not matter—the Java declaring code was nonetheless an uncopyrightable method of operation. Here is the key piece of Judge Alsup’s opinion: “That a system or method of operation has thousands of commands arranged in a creative taxonomy does not change its character as a method of operation. Yes, it is creative. Yes, it is original. Yes, it resembles a taxonomy. But it is nevertheless a command structure, a system or method of operation—a long hierarchy of over six thousand commands to carry out pre-assigned functions. For that reason, it cannot receive copyright protection—patent protection perhaps—but not copyright protection.”

The Federal Circuit Rules for Oracle

Oracle appealed to the U.S. Court of Appeals for the Federal Circuit. The Federal Circuit has jurisdiction over patent, not copyright. But the case ended up before the Federal Circuit because in the beginning it included patent claims.

Keeping copyright away from APIs will help keep the software industry healthy and competitive.

The Court acknowledged that Java declaring code was a method of operation, but held that “the [copyright] protection accorded a particular expression of an idea” is not extinguished “merely because that expression is embodied in a method of operation.” In other words, in the Federal Circuit’s view, if the construction of a particular method of operation involves some creative choice, then it will be copyrightable.

Which, to put it bluntly, simply cannot be right. Not least because the Copyright Act makes clear that methods of operation are uncopyrightable “regardless of the form in which [they are] described, explained, illustrated, or embodied ...” Which means Java’s methods of operation are uncopyrightable even if the description of a particular method in declaring code involves creativity.

More generally, the construction of every method of operation involves some choice. But if every API involves creative choice, then every API would be, in the Federal Circuit’s view, copyrightable. And that would simply erase the copyright law’s exclusion of methods of operation. That is something no court has the power to do.

Will the Supreme Court Intervene?

Google has asked the Supreme Court to review the Federal Circuit’s decision. In a sign that Google’s request is being examined closely, the Court has asked the Department of Justice to weigh in on whether the Court should take the case. Regardless of what the DOJ says, the Court should go ahead and grant Google’s petition for certiorari. And they should reverse the Fed-

eral Circuit and declare software APIs uncopyrightable. Copyrighting APIs threatens serious harm to both the software industry and consumers.

To see this, think for a moment of the competition that open software APIs have encouraged. Open interfaces are the foundation of the Internet—nobody has to ask for permission or pay a license fee to transmit a message according to the Internet Protocol, or to build a server adhering to the Hypertext Transfer Protocol.

Similarly, the continuing vitality of Unix-style operating systems was made possible the fact that although much of the source code for AT&T’s Unix was copyrighted, the APIs were not. This allowed open source systems like Linux to use Unix interfaces, which made it easier for programs built on top of Unix to interact with Linux. The increased software compatibility that open interfaces provided paved the way for open source operating system competition with established proprietary platforms.

Keeping copyright away from APIs will help keep the software industry healthy and competitive. On the other hand, allowing companies like Oracle to use copyright to close APIs will block a lot of future software development. It will limit competition, and give big companies like Oracle the power to pick favorites and squash upstarts.

In view of the critical role open APIs play in ensuring software markets are competitive, the best outcome would be a Supreme Court holding that APIs, as methods of operation, are simply uncopyrightable. Note that such a holding would not be the end of intellectual property protection for APIs. Elements of APIs that are novel and meet the other criteria of patentability would still be eligible for patent protection. But Oracle and other software firms would be prevented from using the comparatively blunt instrument of copyright law to deny their competitors access to APIs wholesale. □

Christopher Jon Sprigman (christopher.sprigman@nyu.edu) is a professor of Law, New York University School of Law, and the co-director of its Engelberg Center on Innovation Law and Policy.

The author has licensed non-commercial reproduction and distribution of this work on terms that can be found at <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

Viewpoint

Teach Foundational Language Principles

Industry is ready and waiting for more graduates educated in the principles of programming languages.

THE NEED FOR more people to learn to program has received widespread attention recently (see, for example, www.code.org and its recent “Hour of Code” held during CS Education week in December of 2013 and 2014). While the ability to program has tremendous potential to support and channel the creative power of people, we should remember that programming languages continuously arise as the need to solve new problems emerges and that it is *language principles* that are lasting. As we discuss in this Viewpoint, language foundations serve an increasingly important and necessary role in the design and implementation of complex software systems in use by industry. Industry needs more people educated in language principles to help it deliver reliable and efficient software solutions to its customers.

Historically, many important principles of languages have arisen in response to the difficulties of designing and implementing complex systems. Garbage collection, introduced by John McCarthy around 1959 for the Lisp language, is now commonplace in modern programming languages such as Java and C#, as well as popular scripting languages such as Python and JavaScript.⁸ Dijkstra’s “Go To Statement Considered Harmful” *Communications* Letter to the Editor advocated the use of structured programming, which is enshrined in all modern programming languages.⁵ Type systems classify program expressions by



the kind of values they compute,¹² enabling compilers to prove the absence of certain kinds of errors and optimize code more effectively. Hoare’s assertion-al method provides a framework for establishing the correctness of programs.⁶

As the complexity of the systems we desire to build increases, new mechanisms to express programmer intent at a higher level are required in order to deliver reliable systems in a predictable manner. The design of new programming languages is driven by new classes of systems and the desire to make programming such systems within the reach of more people. New methods for expressing programmer intent take many forms including features of general-purpose languages, domain-specific languages, and formal specification languages used to verify properties of high-level designs.

Experiences with bugs like the recent TLS heartbeat buffer read overrun in OpenSSL (Heartbleed)¹⁰ show the cost

to companies and society of building fundamental infrastructure in dated programming languages with weak type systems (the C language in this case) that do not protect their abstractions. Several companies have developed new safe systems programming languages to address the challenge of programming scalable and reliable systems. Examples include the Go language from Google (<http://golang.org/>), the Rust language from Mozilla (<http://www.rust-lang.org/>), and the Sing# language from Microsoft (<http://singularity.codeplex.com/>). Such languages raise the level of programming via new type systems that provide more guarantees about the safety of program execution.

Domain-specific languages (DSLs) go further by restricting expressive power to achieve higher-level guarantees about behavior than would be possible with general-purpose languages. The SQL database query language is a classic example, based on the relational algebra,² which enables sophisticated query optimization.

DSLs continue to find application in industry. Google’s Map/Reduce data parallel execution model³ gave rise to a number of SQL-inspired DSLs, including Pig (<http://pig.apache.org/>) from Yahoo. The Spiral system from ETH Zurich (<http://www.spiral.net/>) generates very efficient platform-specific code for digital signal processing from declarative specifications of mathematical functions and optimization rules. Intel makes Spiral-generated code available

as part of its Integrated Performance Primitives library. Colleagues at Microsoft recently developed a DSL called P for programming asynchronous event-driven systems⁴ that allows a design to be checked for responsiveness—the ability to handle every event in a timely manner—using model checking.¹ Core components of the Windows 8 USB 3.0 device driver stack were implemented and verified using P.

Another important class of languages are specification languages, which allow the designers of systems and algorithms to gain more confidence in their design before encoding them in programs where it is more difficult to find and fix design mistakes. Recently, Pamela Zave of AT&T Labs showed the protocol underlying the Chord distributed hash table is flawed¹⁴; she modeled the protocol in the Alloy language⁷ and used the Alloy Analyzer tool to show that “under the same assumptions about failure behavior as made in the Chord papers, no published version of Chord is correct.” Emina Torlak and colleagues used a similar modeling approach to analyze various specifications of the Java Memory Model (JMM) against their published test cases,¹³ revealing numerous inconsistencies among the specifications and the results of the test cases.

Our recommendations are three-fold, visiting the three topics discussed in this Viewpoint in reverse order (formal design languages, domain-specific languages, and new general-purpose programming languages). First, computer science majors, many of whom will be the designers and implementers of next-generation systems, should get a grounding in logic, its application in design formalisms, and experience the creation and debugging of formal specifications with automated tools such as Alloy or TLA+. As Leslie Lamport says, “To designers of complex systems, the need for formal specs should be as obvious as the need for blueprints of a skyscraper.”⁹ The methods, tools, and materials for educating students about “formal specs” are ready for prime time. Mechanisms such as “design by contract,” now available in mainstream programming languages, should be taught as part of introductory programming, as is done in the introductory programming language


Many important principles of languages have arisen in response to the difficulties of designing and implementing complex systems.

sequence at Carnegie Mellon University.¹¹ Students who learn the benefits of principled thinking and see the value of the related tools will retain these lessons throughout their careers. We are failing our computer science majors if we do not teach them about the value of formal specifications.

Second, would-be programmers (CS majors or non-majors) should be exposed as early as possible to functional programming languages to gain experience in the declarative programming paradigm. The value of functional/declarative language abstractions is clear: they allow programmers to do more with less and enable compilation to more efficient code across a wide range of runtime targets. We have seen such abstractions gain prominence in DSLs, as well as in imperative languages such as C#, Java, and Scala, not to mention modern functional languages such as F# and Haskell.

Third, anyone who has a desire to design a new programming language should study type systems in detail; B.C. Pierce’s *Types and Programming Languages*¹² is a very good starting point. The fact that companies such as Microsoft, Google, and Mozilla are investing heavily in systems programming languages with stronger type systems is not accidental—it is the result of decades of experience building and deploying complex systems written in languages with weak type systems. To move our industry forward, we very much need language designers who come to the table educated in the formal foundations of safe programming languages—type systems.

Conclusion

Future applications and systems will increasingly rely on principled and formal language-based approaches to software development to increase programmers’ productivity as well as the performance and reliability of the systems themselves. Software developers will need a solid understanding of language principles to be effective in this new world and increased emphasis on these principles in computer science education is required. For readers interested in the topic of programming languages in education, we strongly urge consulting the work of the SIGPLAN Education Board (<http://wp.acm.org/sigplaneducationboard/>), which rewrote from scratch the “Knowledge Area” of “Programming Languages,” contained in the first public draft of the 2013 Computer Science Curricula Report (<http://cs2013.org>). 

References

1. Clarke, E.M., Grumberg, O., and Peled, D. *Model Checking*. MIT Press, 2001, I–XIV, 1–314.
2. Codd, E.F. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377–387.
3. Dean, J. and Ghemawat, S. MapReduce: A flexible data processing tool. *Commun. ACM* 53, 1 (Jan. 2010), 72–77.
4. Desai, A. et al. P: Safe asynchronous event-driven programming. In *Proceedings of the 2013 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013.
5. Dijkstra, E.W. Letters to the editor: Go To statement considered harmful. *Commun. ACM* 11, 13 (Mar. 1968), 147–148.
6. Hoare, C.A.R. An axiomatic basis for computer programming. *Commun. ACM* 12, vol. 10 (Oct. 1969), 576–580.
7. Jackson, D. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, Cambridge, MA, 2012.
8. Jones, R. *The Garbage Collection Handbook: The Art of Automatic Memory Management*. Chapman and Hall, 2012.
9. Lamport, L. Why we should build software like we build houses. *Wired* 25 (Jan. 2013).
10. OpenSSL Project. OpenSSL Security Advisory [07 Apr 2014]. (Apr. 7, 2014); http://www.openssl.org/news/secadv_20140407.txt.
11. Pfenning, F. Specification and verification in introductory computer science. Carnegie Mellon University; <http://c0.typesafety.net/>.
12. Pierce, B.C. *Types and Programming Languages*. MIT Press, 2002, I–XXI, 1–623.
13. Torlak, E., Vaziri, M. and Dolby, J. MemSAT: Checking axiomatic specifications of memory models. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2010.
14. Zave, P. Using lightweight modeling to understand Chord. *SIGCOMM Comput. Commun. Rev.* 42 (Mar. 2012), 49–57.

Thomas Ball (tball@microsoft.com) is a principal researcher and co-manager of the Research in Software Engineering (RISE) group at Microsoft Research, Redmond, WA.

Benjamin Zorn (zorn@microsoft.com) is a principal researcher and co-manager of the Research in Software Engineering (RISE) group at Microsoft Research, Redmond, WA.

Copyright held by authors.

Viewpoint

Managing Your Digital Life

Everyone should be able to manage their personal data with a personal information management system.

A TYPICAL PERSON today usually has data^a on several devices and in a number of commercial systems that function as data traps where it is easy to check in information and difficult to remove it or sometimes to simply access it. It is also difficult, sometimes impossible, to control data access by other parties. One might consider this an unavoidable price to pay in order to fully take advantage of the ever-increasing amount of available information. However, this situation is not only unsatisfactory because it requires users to trade privacy against convenience but also, because it limits the value we, as individuals and as a society, can derive from the data.

We live in a world where data is considered a vital asset *and* where most people consider they have little, if any, control over their personal data. This is surely detrimental to trust, innovation, and growth. In this world, we are also limited in leveraging all this existing information because it resides in isolated silos kept apart by technical incompatibilities, semantic fuzziness, organizational barriers, as well as privacy regulations. The situa-

a Data that we publish (for example, pictures), produce (for example, contacts), co-produce socially (for example, in social networks), data that organizations produce about us (for example, banks, public administrations), data about us captured by sensors (for example, GPS), and so forth.



tion gets worse as the number of data sources keeps growing.

Of course, users could choose to delegate all their information to a single company (some companies clearly dream of offering all the spectrum of information services). This would definitely make the users' lives easier in

the short run, but this would also make them totally dependent of that company. Although this is debatable, we will assume that, given a choice, most users would prefer to avoid such a solution.

Another possibility is to ask users to spend a few years of their lives studying to become computer wizards. We

can safely assume this is not what a large portion of the population craves. Is there another option? We believe there is one: the personal information management system (PIMS).

The Personal Information Management System

To understand the notion of personal information management system, we must consider today's context. Why do users "entrust" their data to services proposed by companies such as Google or Facebook? Because they enjoy using these services. Now, there are two facets to these services: they are supported by software with useful features, and they are executed on machines that are not managed by the user. What if we could separate these two facets? On one hand, a particular user would select, for each service, the best software developer or service provider that suits his or her needs. On the other hand, this user would choose a server where all these applications would run. This would therefore bring together, on a personal server, all this user's favorite applications and all the user's data that is currently distributed, fragmented, and isolated.

This is what a PIMS does. It may look like utopia. As we will see, it is not.

The PIMS system consists of a user's server, running the services selected by the user, storing and processing the user's data:

- ▶ The user pays for the server (possibly owns it) so the server does what the user wants it to do and nothing else.
- ▶ The user chooses the application code to deploy on the server.
- ▶ The server software, as well as that of the services, is possibly open source (which allows for code verification on behalf of the users of the service).
- ▶ The server resides in the cloud so it can be reached from anywhere.

Many different settings are possible. We do not need to specify a particular one. The user may own the server, or pay for a hosted server. The server may be a physical or a virtual machine. It may be physically located in the user's home (for example, a TV box) or not. It may run on a single machine or be distributed among several machines.

The PIMS centralizes the user's personal information. It is a *digital home*. The PIMS is also able to exert control

People have had relatively little concern so far about where their personal data goes, but this is changing for a number of reasons.

over information that resides in external services (for example, Facebook), and that only gets replicated inside the PIMS. These services' business models are based on our personal data, and PIMS will not prevent them from working in this way, so long as their customers agree; however, they will need to share their data with their users, who *may* want to use the data with competing platforms, or for whatever makes sense to them. PIMS do not prevent data sharing, they prevent unilateral data hoarding. The PIMS software provides the necessary support so the user always has access to his or her information and controls (to the extent this is possible) how information is accessed by the applications.

By centralizing access to an individual's information, the PIMS enables very useful new services that combine information from a wide variety of sources—those same silos that were prevented from collaborating together in an organizations-centric world—under the user's control and to serve his or her needs.

Is the PIMS a security risk? Of course, one could answer it is difficult to be more risky than today's large, interconnected corporate databases containing data about millions of customers, but this is hardly a comforting answer. A possible weakness is that PIMS security seems to rest on end users when individuals have repeatedly proved to be either disinclined or unable to apply even the minimal effort toward securing their systems. However:

- ▶ The PIMS is run by a professional operator and/or on secure hardware.

Thus, security is handled by the PIMS on behalf of end users better than it would ever be, for instance, on general-purpose personal devices such as PCs.

- ▶ The users have only to select security/privacy options under the guidance of the PIMS. The PIMS then reduces privacy risks, for example by monitoring accesses and access patterns, for all applications run within the PIMS environment.

Also, in a properly designed PIMS, each collection of user data is strongly isolated from that of others. So, in case security has been compromised, it has been so for a single user. Pirates will therefore be more attracted to other systems with lots of data and many users to attack.

PIMS will not resolve the security issues for protecting users' data. However, by providing a single entry point for specifying security/privacy rules, and with the support of the PIMS carefully designed with security in mind, we believe this model puts us in a better position to provide security as well as privacy to users.

Another main issue for regular users is clearly the management of their PIMS. This is where the cloud turns out to be essential. With the cloud, it is possible to have a company host the system for the users. (The user is a paying customer of that company and a contract protects the data privacy.)

PIMS Are Coming!

This may be observed from three different angles: society, technology, and industry.

Society is ready to move. People have had relatively little concern so far about where their personal data goes, but this is changing for a number of reasons:

- ▶ Clear-cut abuses of massive data gathering by both governments (NSA and its European counterparts) and corporations (credit bureaus, health corporations, and social networks come to mind).

- ▶ An increasing awareness by individuals of the asymmetry between what companies know about a person, and what the person actually knows about the companies (or even about herself): in Europe as well as the U.S., consumer surveys all indicate consumers are increasingly worried, not just about the se-

curity of their data, but also about what the organizations holding data about them are likely to do with the data.^b

► A growing resentment toward intrusive marketing, cryptic personalization, and creepy “big data” inferences: As an example, according to the Pew Internet and American Life Project, 65% of U.S. adults view personalized search as “bad” and 73% see it as a privacy invasion.^c

► An emerging understanding that personal data could be valuable to individuals as well as to corporations. “Quantified self” applications are a case in point: millions of people seem ready to spend \$100 or more for devices that help them keep track of their health, physical condition, sleep patterns, and so forth, all via data.

As a result, a series of initiatives are converging toward giving individual users not only more control over how others gather and use their personal data,

but more power to actually own and use this data to their own ends. These initiatives fall into several categories:

► *Privacy control*: In 2009, the User Managed Access Work Group proposed specifications^d to let an individual control the authorization of data sharing and service access made between online services on the individual’s behalf, and to facilitate interoperable implementations of the specs. The current revision of privacy regulations in Europe and elsewhere introduces new concepts such as “privacy by design” (for example, data minimization), opt-in, sticky privacy policies, the “right to be forgotten,” or data portability.

► *Information symmetry*: In the spirit of establishing a better symmetry between customers and vendors, Doc Searls and others have promoted the concept of Vendor Relationship Management^e (VRM) since 2006. VRM emerged from the idea that customers would benefit from having an inte-

grated view of their relationships with vendors, in the same way that vendors try to have an integrated view of their customers through CRM.

► *Information ownership and use by individuals*: In a 2011 report,² the World Economic Forum wrote: “In practical terms, a person’s data would be equivalent to their “money.” It would reside in an account where it would be controlled, managed, exchanged, and accounted for just like personal banking services operate today.” See also, for instance, the OpenPDS Project¹ at the MIT Media Lab.

These expectations have also recently led to important personal data disclosure initiatives, such as Smart Disclosure in the U.S. (where more than 40 million Americans can currently download and use their health data by using the same “Blue Button” on their health insurance provider’s website), MiData in the U.K., and MesInfos^f in France.

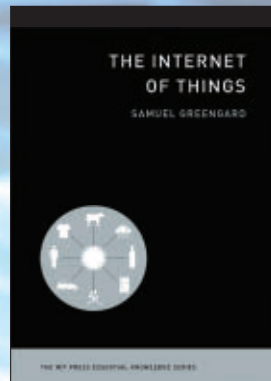
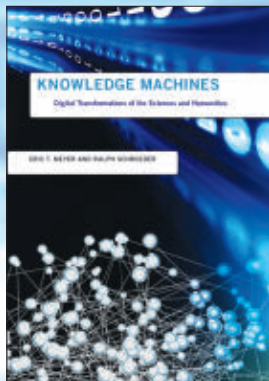
b As an example, see GFK Survey on Data Privacy and Trust, 2014: <http://www.gfk.com/trustsurvey/>.

c See <http://www.pewinternet.org/media-mentions/pew-report-65-view-personalized-search-as-bad-73-see-it-as-privacy-invasion/>.

d See <https://kantarainitiative.org/confluence/display/uma/Home>.

e Project VRM, Berkman Center for Internet and Society, Harvard University.

f MesInfos is a personal data disclosure experiment where several large companies (network operators, banks, retailers, insurers...) have agreed to share with a panel of customers the personal data they hold about them.



KNOWLEDGE MACHINES

Digital Transformations of the Sciences and Humanities

Eric T. Meyer and Ralph Schroeder

An examination of the ways that digital and networked technologies have fundamentally changed research practices in disciplines from astronomy to literary analysis.

Infrastructures series • 280 pp., 13 illus., \$40 cloth

THE INTERNET OF THINGS

Samuel Greengard

A guided tour through the Internet of Things, a networked world of connected devices, objects, and people that is changing the way we live and work.

The MIT Press Essential Knowledge series
184 pp., \$12.95 paper



BIG DATA, LITTLE DATA, NO DATA

Scholarship in the Networked World

Christine L. Borgman

“We live amidst a sea of data. Christine Borgman explores the depths and swells of that data and how they connect with scholarship and, more broadly, systems of knowledge. The result is an invaluable guide to harnessing the power of data, while sensitive to its misuses.”

—Jonathan Zittrain, Harvard University; Co-Founder, Berkman Center for Internet & Society; Director, Harvard Law School Library

400 pp., 7 illus., \$32 cloth

GREAT PRINCIPLES OF COMPUTING

Peter J. Denning

with Craig H. Martell

foreword by Vint Cerf

“This insightful book provides a structure to help us understand the breadth and depth of computing, which can enable us to place our knowledge in a coherent framework, and in turn can inform the design of curricula.”

—Tim Bell, Professor, Computer Science and Software Engineering, University of Canterbury

312 pp., 95 illus., \$30 paper

Technology is gearing up. Some people already use their own PIMS. They run a home server or rent a hosted server (in a 2013 market test, the French Web hosting company OVH rented 15,000 low-cost personal servers in just 10 days). They have at their disposal some rather primitive functionality, typically by developing scripts. A limiting factor is that, in order to use existing services, they have no choice but to relinquish some control over their data. For instance, if they want to partake in the social Web, they must trust their data to Facebook or others. However, by devoting time and effort and subject to these limitations, they can manage their own data and services to some extent.

This is not for everyone, though. One needs to be highly skilled and willing to devote a lot of time in order to achieve such a result today. But things are changing rapidly:

- ▶ Abstraction technologies are helping tame the complexity of servers.
- ▶ Open source technology is increasingly available for a large range of services.
- ▶ Hardware price is now very low and the price of machine hosting has dropped.

Research in PIMS is also increasingly active.⁸ A number of prototypes have been developed for storing and retrieving personal data: Lifestreams, Stuff-I've-Seen, Haystack, MyLifeBits, Connections, Seetrieve, Personal Dataspaces, or deskWeb. The tipping point appears close as indicated by a number of projects such as Mailpile (for mail), Lima (for Dropbox-like service hosted at home), Synologie or Iomega (personal NAS), SAMI of Samsung (personal data store), and a number of self-host PIMS such as YounoHost, Amahi, ArkOS, OwnCloud, or Cozy Cloud.

Large companies are getting in. PIMS also act as magnets to large companies, and in particular:

- ▶ Traditional companies that already have large amounts of personal information. These companies, including retailers, insurance companies, or banks, are increasingly disintermediated from their customers by pure Internet players. They can find in PIMS an opportunity to rebuild a direct

⁸ Personal information management, Wikipedia.

We are all experiencing a loss of control over our personal data. With PIMS, we can regain control.

interaction with these customers.

- ▶ Companies managing home appliances (notably Internet boxes) are natural hosts for personal information. Starting from data dedicated to specific usages, these boxes could evolve to become more generic and control increasing numbers of connected objects, services, and data.

PIMS should also be of interest to pure Internet players. Some of them (for example, Amazon), have a great amount of know-how in providing data services. They could seamlessly move to this new business. Others (for example, Facebook), centered on the management of information, cannot let such a wide field of information management grow without becoming involved. However, PIMS, as defined here, are very far from these companies' indirect business models based on personalized advertisement. So moving in this new market would require a major change for them, and in particular, the clarification of the relationship with users (represented by the PIMS) with respect to personal data monetization.

PIMS Enable New Functionalities

For users, perhaps the main reason to move to PIMS is these systems enable great new functionalities. Building on the integration of the user's data, PIMS can provide:

- ▶ Global search over the person's data with a semantic layer using a personal ontology (for example, the data organization the person likes and the person's terminology for data) that helps give meaning to the data;
- ▶ Automatic synchronization of data on different devices/systems, and glob-

al task sequencing to facilitate interoperating different devices/services;

- ▶ Exchange of information and knowledge between "friends" in a truly social way, even if these use different social network platforms, or no platform at all;
- ▶ Centralized control point for connected objects, a hub for the Internet of Things; and
- ▶ Data analysis/mining over the person's information.

Conclusion

Online services have become an essential part of our daily life. However, because of them, we are all experiencing a loss of control over our personal data. With PIMS, we can regain control. PIMS also enable a wide range of new functionalities. They point toward a new, powerful, yet more balanced way of creating user value as well as business value. They achieve all this without giving up on ubiquity, ease of use, or security. For these reasons, we believe their benefits are so clear that PIMS will be adopted massively in a near future. What remains to be seen is what shape this evolution will take, and how it will alter the relationships between new "personal cloud" players, home appliance and electronics providers, established online platforms, and current personal data holders.

Will we continue to move toward an Internet dominated by oligopolies, user profiling, and generalized surveillance? Will our lack of control over our data increasingly turn us into passive products of a global digital economy? PIMS may be the alternative to such an outcome. ■

References

1. de Montjoye, Y.-A., Wang S., and Pentland, A. On the trusted use of large-scale personal data. *IEEE Data Engineering Bulletin* 35, 4 (2012).
2. World Economic Forum. Personal Data: The Emergence of a New Asset Class (2011); <http://www.weforum.org/reports/personal-data-emergence-newasset-class>.

Serge Abiteboul (Serge.Abiteboul@inria.fr) is a senior researcher at INRIA and Distinguished Affiliated Professor at Ecole Normale Supérieure de Cachan in Paris, France.

Benjamin André (ben@CozyCloud.CC) is the co-founder and CEO at Cozy Cloud in Paris, France.

Daniel Kaplan (dkaplan@fing.org) is the founder and CEO of La Fondation Internet Nouvelle Génération (Fing).

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

Problems with simultaneity in distributed systems.

BY JUSTIN SHEEHY

There Is No Now

“NOW.”

The time elapsed between when I wrote that word and when you read it was at least a couple of weeks. That kind of delay is one we take for granted and do not even think about in written media.

“Now.”

If we were in the same room and instead I spoke aloud, you might have a greater sense of immediacy. You might intuitively feel as if you were hearing the word at exactly the same time that I spoke it. That intuition would be wrong. If, instead of trusting your intuition, you thought about the physics of sound, you would know that time must have elapsed between my speaking and your hearing. The motion of the air, carrying my word, would take time to get from my mouth to your ear.

“Now.”

Even if I held up a sign with that word written and we both looked at it, our perception of that image would not happen at the same moment, as the light carrying the information about the sign would take a different amount of time to reach each of us.

While some things about computers are “virtual,” they still must operate in the physical world and cannot ignore the challenges of that world. Rear Admiral Grace Hopper (one of the most important pioneers in our field, whose achievements include creating the first compiler) used to illustrate this point by giving each of her students a piece of wire 11.8 inches long, the maximum distance that electricity can travel in one nanosecond. This physical representation of the relationship between information, time, and distance served as a tool for explaining why signals (like my metaphorical sign here) must always and unavoidably take time to arrive at their destinations. Given these delays, it can be difficult to reason about exactly what “now” means in computer systems.

There is nothing theoretically preventing us from agreeing about a common idea of “now,” though, if we carefully plan ahead. (Relativity is not a problem here, though it is a tempting distraction. All of humanity’s current computing systems share a close enough frame of reference to make relativistic differences in the perception of time immaterial.) The Network Time Protocol (NTP),¹⁴ used for synchronizing the clocks between systems on the Internet, works in part by calculating the time that messages take to travel between hosts. Then, once that travel time is known, a host can account for it when adjusting its clock to match a more authoritative source. By providing some very precise sources (clocks based on continuous measurement of atomic radiation) in that network, we are able to use NTP to synchronize computers’ clocks to within a small margin of error. Each of the satellites making up the worldwide GPS contains multiple

0101010101110101010001

0011010101010

NOW

NOWNOWNOWNOWNOWN

OWNOWN

OWNOWNOWNOWN



of these atomic clocks (so a single clock failing does not make a satellite worthless), and the GPS protocols allow anyone with a receiver—as long as they can see enough satellites to solve for all of the variables—to determine not only the receiver’s own position, but also the time with excellent precision.

We have understood these protocols for a few decades, so it would be easy to believe we have overcome this class of problems and we ought to be able to build systems that assume our clocks are synchronized. Atomic clocks, NTP, and GPS satellites provide both the knowledge and the equipment to account for the time it takes for information to travel. Therefore, systems anywhere should be able to agree on a “now” and to share a common, single view of the progress of time. Then whole categories of hard problems in networks and computing would become much easier. If all of the systems you care about have the exact same perception of time, then many of these problems become tractable to solve even when some of the hosts involved are failing. Yet, these problems do still exist, and dealing with them is not only a continuously active area of research, but also a major concern when building practical distributed systems.

You might look at the mature mechanisms available for understanding time and believe that researchers and system builders are doing a huge amount of unnecessary work. Why try to solve problems assuming clocks may differ when we know how to synchronize? Why not instead just use the right combination of time sources and protocols to make the clocks agree, and move on past those problems? One thing makes this implausible and makes these problems not only important, but also necessary to face head on: everything breaks.

The real problem is not the theoretical notion of information requiring time to be transferred from one place to another. The real problem is that in the physical world in which computing systems reside, components often fail. One of the most common mistakes in building systems—especially, but not only, distributed-computing systems on commodity machines and networks—is assuming an escape from basic physical realities.

The speed of light is one such reality, but so is one that is more pernicious and just as universal: we cannot make perfect machines that never break. It is the combination of these realities—of asynchrony and partial failure—that together make building distributed systems a difficult pursuit. If we do not plan and account for failures in individual components, we all but guarantee the failure of combined systems.

One of the most important results in the theory of distributed systems is an impossibility result, showing one of the limits of the ability to build systems that work in a world where things can fail. This is generally referred to as the FLP result, named for its authors, Fischer, Lynch, and Paterson.⁸ Their work, which won the 2001 Dijkstra prize for the most influential paper in distributed computing, showed conclusively that some computational problems that are achievable in a “synchronous” model in which hosts have identical or shared clocks are impossible under a weaker, asynchronous system model. Impossibility results such as this are very important, as they can guide you away from going down a dead-end path when designing your own system. They can also provide a snake-oil detector, so you can feel justified in your suspicion about anyone who claims a product does something you know to be impossible.

A related result, better known than FLP by developers today, is the CAP theorem (for consistency, availability, and partition tolerance). This was first proposed informally by Eric Brewer⁵ and later a formal version of it was proven by Seth Gilbert and Nancy Lynch.⁹ From a distributed systems theory point of view, the CAP theorem is less interesting than FLP: a counterexample “beating” the formal version of CAP assumes an even weaker and more adversarial model of the world than FLP, and demands that even more be achieved within that model. While one is not exactly a subproblem of the other, FLP is a stronger, more interesting, and perhaps more surprising result. A researcher already familiar with FLP might find the CAP idea a bit boring.

It would be reasonable, though, to think perhaps the value of CAP’s existence is to be more approachable and more easily understood by those not

steeped in the literature of distributed systems. That would be laudable and worthwhile, but the past several years have shown (through dozens of articles and blog posts, many of which badly misunderstand the basic idea) the idea of CAP has, sadly, not been an easy way for today’s developers to come to terms with the reality of programming in a distributed and imperfect world. That reality, whether from the point of view of CAP or FLP or any other, is a world in which you must assume imperfection from the components you use as building blocks. (Any theoretical “impossibility result” such as CAP or FLP is inately tied to its system model. This is the theoretical model of the world that the result depends on. Any such result does not really say that some goal—such as consensus—is impossible in general, but rather it is impossible within that specific model. This can be extremely useful to practitioners in developing intuitions about which paths might be dead ends, but it can also be misleading if you only learn the result without learning the context to which the result applies.)

The real problem is things break. The literature referred to here, such as FLP, is all about dealing with systems in which components are expected to fail. If this is the problem, then why don’t we just use things that do not break, and then build better systems with components we can assume are robust?


Quite often in the past couple of years, the Spanner system from Google has been referenced as a justification for making this sort of assumption.⁶ This system uses exactly the techniques mentioned earlier (NTP, GPS, and atomic clocks) to assist in coordinating the clocks of the hosts that make up Spanner and in minimizing and measuring (but not eliminating) the uncertainty about the differences between those clocks. The Spanner paper, along with the system it documents, is often used to back up claims that it is possible to have a distributed system with a single view of time.

Despite the appeal of pointing at Google and using such an argument from authority, everyone making that claim is wrong. In fact, anyone citing Spanner as evidence of synchronization being “solved” is either lying or has


not actually read the paper. The simplest and clearest evidence against that claim is the Spanner paper itself. The TrueTime component of Spanner does not provide a simple and unified shared timeline. Instead, it very clearly provides an API that directly exposes uncertainty about differences in perceived time between system clocks. If you ask it for the current time, it does not give you a single value, but rather a pair of values describing a range of possibility around “now”—that is, TrueTime does the opposite of fixing this fundamental problem. It takes the brave and fascinating choice of confronting it directly and being explicit about uncertainty, instead of pretending a single absolute value for “now” is meaningful across a working distributed system.

Within the production environment of Spanner, clock drift at any moment is typically from one to seven milliseconds. That is the best Google can do after including the corrective effects of GPS, atomic clocks, eviction of the worst-drifted clocks, and more in order to minimize skew. Typical x86 clocks vary their speeds depending on all kinds of unpredictable environmental factors, such as load, heat, and power. Even the difference between the top and bottom of the same rack can lead to a variance in skew. If the best that can be done in a wildly expensive environment like Google’s is to live with an uncertainty of several milliseconds, most of us should assume our own clocks are off by much more than that.

Another claim often made in justifying the “just pretend it’s fine” approach in distributed systems design is that sufficiently high-quality equipment does not fail, or at least fails so rarely that you do not need to worry about it. This claim would be understandable, though incorrect, coming from the makers of such equipment, but it is usually the users of such gear, such as designers of distributed database software, who are heard making such claims. (The designers of GPS, which Spanner and others rely on, certainly did not subscribe to this kind of claim. There are almost 30 GPS satellites, and you need to see only four of them for the system to work. Each of those satellites has multiple redundant atomic clocks so it can continue functioning when one of them breaks.)



One of the most common mistakes in building distributed systems is assuming we can escape from basic physical realities.



One of the most common variants of such claims is “the network is reliable,” in the context of a local, in-data-center network. Many systems have undefined and most likely disastrous behavior when networks behave poorly. The people who want to sell you these systems justify their choice to ignore reality by explaining that such failures are extremely uncommon. By doing this, they do their customers a double disservice. First, even if such events were rare, wouldn’t you want to understand how your system would behave when they do occur, in order to prepare for the impact those events would have on your business? Second, and worse, the claim is simply a lie—so bald-faced a lie, in fact, that it is the very first of the classic eight fallacies of distributed computing.^{7,15} The realities of such failures are well documented in a previous ACM *Queue* article,³ and the evidence is so very clear and present that anyone justifying software design choices by claiming “the network is reliable” without irony should probably not be trusted to build any computing systems at all. While it is true that some systems and networks might not have failed in a way a given observer could notice, it would be foolish to base a system’s safety on the assumption it will never fail.

The opposite approach to this waving-off of physical problems is to assume that almost nothing can be counted on, and to design using only formal models of a very adversarial world. The “asynchronous” model that FLP was proved on is not the most adversarial model on which to build a working system, but it is certainly a world much more hostile than most developers believe their systems are running on. The thinking goes that if the world you model in is worse than the world you run in, then things you can succeed at in the model should be possible in the real world of implementation. (Note, distributed systems theorists have some models that are more difficult to succeed in, such as those including the possibility of “Byzantine” failure, where parts of the system can fail in much worse ways than just crashing or delaying. For a truly adversarial network/system model, you could see, for example, either the symbolic or the computational models

used by cryptographic protocol theorists. In that world, system builders really do assume your own network is out to get you.)

It is in this context, assuming a world that is a bit worse than we think we are really operating in, that the best-known protocols for consensus and coordination, such as Paxos,¹² are designed. For such essential building blocks for distributed systems, it is useful to know they can provide their most important guarantees even in a world that is working against the designer by arbitrarily losing messages, crashing hosts, and so on. (For example, with Paxos and related protocols, the most emphasized property is “safety”—the guarantee that different participating hosts will not ever make conflicting decisions.) Another such area of work is logical time, manifest as vector clocks, version vectors, and other ways of abstracting over the ordering of events. This idea generally acknowledges the inability to assume synchronized clocks and builds notions of ordering for a world in which clocks are entirely unreliable.

Of course, you should always strive to make everything, including networks, as reliable as possible. Concluding that constant goal with an achievable perfect end state is silly. Equally silly would be a purist view that only the most well-founded and perfectly understood theoretical models are sensible starting places for building systems. Many of the most effective distributed systems are built on worthwhile elements that do not map perfectly to the most common models of distributed computing. An exemplary such building block would be TCP. This nearly ubiquitous protocol provides some very useful properties, the exact set of which do not map exactly to any of the typical network models used in developing theoretical results such as FLP. This creates a quandary for the systems builder: it would be foolish not to assume the reality of something like TCP when designing, but in some cases that puts them in a tenuous position if they try to understand how exactly distributed systems theory applies to their work.

The Zab protocol, which forms the most essential part of the popular Apache ZooKeeper coordination software, is a fascinating example of an

attempt at walking this middle road.¹⁰ The authors of ZooKeeper knew about existing consensus protocols such as Paxos but decided they wanted their own protocol with a few additional features such as the ability to be processing many requests at once instead of waiting for each proposal to complete before starting the next one. They realized that if they built on TCP, they had the advantage of an underlying system that provided some valuable properties they could assume in their protocol. For example, within a single connected TCP socket, a sender producing message A followed by message B can safely assume that if the receiver reads B, then the receiver has previously read A. That “prefix” property is very useful, and one not present in the asynchronous model. This is a concrete example of the advantages available by looking at both the research in the field and the specific technology that is actually available to build on, instead of ignoring either.

When trying to be pragmatic, though, one must be careful not to let that pragmatism become its own strange kind of purity and an excuse for sloppy work. The Zab protocol as implemented inside ZooKeeper, the de facto reference implementation, has never been an accurate implementation of the Zab protocol as designed.¹³ You might call yourself a “pragmatist” and note that most other software also does not match a formal specification; thus you might say there is nothing unusual to worry about here. You would be wrong for two reasons. First, the role ZooKeeper and similar software is used for is not like other software; it is there precisely to provide essential safety properties forming a bedrock on which the rest of a system can make powerful assumptions. Second, if there are problems with the safety properties of a protocol like this, the appearance of those problems (while possibly very dangerous) can be subtle and easy to miss. Without a strong belief the implementation reflects the protocol as analyzed, it is not reasonable for a user to trust a system. The claim that a system’s “good-enough correctness” is proven by its popularity is nonsense if the casual user cannot evaluate that correctness.

All of this is not to pick on ZooKeeper. In fact, it is one of the highest-quality

pieces of open source software in its genre, with many excellent engineers continually improving it. Under recent analysis, it fares far better under duress than anything it competes with.¹ I have pointed at ZooKeeper only as an example of both the necessity and the pitfalls of taking a middle road with regard to theory and practicality. Mapping theory to practice can be extremely challenging.

Another example of this middle road is hybrid logical clocks¹¹ that integrate the general techniques of logical time with timestamps from physical clocks. This allows users to apply some interesting techniques based on having a view (imperfect, but still useful) of the physical clocks across a whole system. Much like Spanner, this does not pretend to create a single unified timeline but does allow a system designer to build on the best available knowledge of time as perceived and communicated across a cluster.

All of these different coordination systems—including Paxos, Viewstamped Replication, Zab/ZooKeeper, and Raft—provide ways of defining an ordering of events across a distributed system even though physical time cannot safely be used for that purpose. These protocols, though, absolutely can be used for that purpose: to provide a unified timeline across a system. You can think of coordination as providing a logical surrogate for “now.” When used in that way, however, these protocols have a cost, resulting from something they all fundamentally have in common: constant communication. For example, if you coordinate an ordering for all of the things that happen in your distributed system, then at best you are able to provide a response latency no less than the round-trip time (two sequential message deliveries) inside that system.

Depending on the details of your coordination system, there may be similar bounds on throughput as well. The designer of a system with aggressive performance demands may wish to do things correctly but find the cost of constant coordination to be prohibitive. This is especially the case when hosts or networks are expected to fail often, as many coordination systems provide only “eventual liveness” and can make progress only during times of minimal trouble. Even in those rare times when

everything is working perfectly, however, the communication cost of constant coordination might simply be too much.

Giving up strict coordination in exchange for performance is a well-known trade-off in many areas of computing, including CPU architecture, multithreaded programming, and DBMS design. Quite often this has turned into a game of finding out just how little coordination is really needed to provide unsurprising behavior to users. While designers of some concurrent-but-local systems have developed quite a collection of tricks for managing just enough coordination (for example, the RDBMS field has a long history of interesting performance hacks, often resulting in being far less ACID (that is, atomicity, consistency, isolation, durability) than you might guess²), the exploration of such techniques for general distributed systems is just beginning.

This is an exciting time, as the subject of how to make these trade-offs in distributed systems design is just now starting to be studied seriously. One place where this topic is getting the attention it deserves is in the Berkeley Orders of Magnitude (BOOM) team at the University of California Berkeley, where multiple researchers are taking different but related approaches to understanding how to make disciplined trade-offs.⁴ They are breaking new ground in knowing when and how you can safely decide not to care about “now” and thus not pay the costs of coordination. Work like this may soon lead to a greater understanding of exactly how little we really need synchronized time in order to do our work. If distributed systems can be built with less coordination while still providing all of the safety properties needed, they may be faster, more resilient, and more able to scale.

Another important area of research on avoiding the need for worrying about time or coordination involves conflict-free replicated data types (CRDTs). These are data structures whose updates never need to be ordered and so can be used safely without trying to tackle the problem of time. They provide a kind of safety that is sometimes called strong eventual consistency: all hosts in a system that have received the same set of updates, regardless of order, will have the same


state. It has long been known that this can be achieved if all of the work you do has certain properties, such as being commutative, associative, and idempotent. What makes CRDTs exciting is the researchers in that field¹⁶ are expanding our understanding of how expressive we can be within that limitation and how inexpensively we can do such work while providing a rich set of data types that work off the shelf for developers.

One way to tell the development of these topics is just beginning is the existence of popular distributed systems that prefer ad hoc hacks instead of the best-known choices for dealing with their problems of consistency, coordination, or consensus. One example of this is any distributed database with a “last write wins” policy for resolving conflicting writes. Since we already know that “last” by itself is a meaningless term for the same reason that “now” is not a simple value across the whole system, this is really a “many writes, chosen unpredictably, will be lost” policy—but that would not sell as many databases, would it? Even if the state of the art is still rapidly improving, anyone should be able to do better than this.

Another example, just as terrible as the “most writes lose” database strategy, is the choice to solve cluster management via ad-hoc coordination code instead of using a formally founded and well-analyzed consensus protocol. If you really do need something other than one of the well-known consensus protocols to solve the same problem they solve (hint: you don’t), then at a very minimum you ought to do what the ZooKeeper/Zab implementers did and document your goals and assumptions clearly. That will not save you from disaster, but it will at least assist the archaeologists who come later to examine your remains.

This is a very exciting time to be a builder of distributed systems. Many changes are still to come. Some truths, however, are very likely to remain. The idea of “now” as a simple value that has meaning across a distance will always be problematic. We will continue to need more research and more practical experience to improve our tools for living with that reality. We can do better. We can do it now.

Acknowledgments

Thanks to those who provided feedback, including Peter Bailis, Christopher Meiklejohn, Steve Vinoski, Kyle Kingsbury, and Terry Coatta. 

Related articles on queue.acm.org

If You Have Too Much Data, then “Good Enough” Is Good Enough

Pat Helland

<http://queue.acm.org/detail.cfm?id=1988603>

There’s Just No Getting around It: You’re Building a Distributed System

Mark Cavage

<http://queue.acm.org/detail.cfm?id=2482856>

The Network is Reliable

Peter Bailis and Kyle Kingsbury

<http://queue.acm.org/detail.cfm?id=2655736>

References

1. Aphyr. Call me maybe: ZooKeeper, 2013; <http://aphyr.com/posts/291-call-me-maybe-zookeeper>.
2. Bailis, P. When is “ACID” ACID? Rarely, 2013; <http://www.bailis.org/blog/when-is-acid-acid-rarely/>.
3. Bailis, P. and Kingsbury, K. The network is reliable. *ACM Queue* 12, 7 (2014); <http://queue.acm.org/detail.cfm?id=2655736>.
4. BOOM; <http://boom.cs.berkeley.edu/papers.html>.
5. Brewer, E.A. Towards robust distributed systems, 2000; <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>.
6. Corbett, J.C. et al. Spanner: Google’s globally distributed database. In *Proceedings of the 10th Symposium on Operating System Design and Implementation*, 2012; <http://research.google.com/archive/spanner.html>; <http://static.googleusercontent.com/media/research.google.com/en/us/archive/spanner-osdi2012.pdf>.
7. Deusch, P. Eight fallacies of distributed computing; <https://blogs.oracle.com/jag/resource/Fallacies.html>.
8. Fischer, M.J., Lynch, N.A. and Paterson, M.S. Impossibility of distributed consensus with one faulty process. *JACM* 32, 2 (1985), 374–382; <http://macs.citadel.edu/rudolph/csci604/ImpossibilityofConsensus.pdf>.
9. Gilbert, S. and Lynch, N. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant Web services. *ACM SIGACT News* 33, 2 (2002), 51–59; http://webpages.cs.luc.edu/~pld/353/gilbert_lynch_brewer_proof.pdf.
10. Junqueira, F.P., Reed, B.C. and Serafini, M. Zab: High-performance broadcast for primary backup systems, 2011; <http://web.stanford.edu/class/cs347/reading/zab.pdf>.
11. Kulkarni, S., Demirbas, M., Madeppa, D., Bharadwaj, A. and Leone, M. Logical physical clocks and consistent snapshots in globally distributed databases; <http://www.cse.buffalo.edu/tech-reports/2014-04.pdf>.
12. Lamport, L. The part-time parliament. *ACM Trans. Computer Systems* 16, 2 (1998), 133–169; <http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#lamport-paxos>.
13. Medeiros, A. ZooKeeper’s atomic broadcast protocol: Theory and practice; <http://www.tcs.hut.fi/Studies/T-79.5001/reports/2012-deSouzaMedeiros.pdf>.
14. NTP; <http://www.ntp.org>.
15. Rotem-Gal-Oz, A. Fallacies of distributed computing explained; <http://www.rgoarchitects.com/Files/fallacies.pdf>.
16. SyncFree; <https://syncfree.lip6.fr/index.php/publications>.

Justin Sheehy (@justinsheehy) is the site leader for VMware’s Cambridge MA R&D center, and also plays a strategic role as architect for the company’s Storage & Availability business. His three most recent positions before joining VMware were as CTO for Basho, Principal Scientist at MITRE, and Sr. Architect at Akamai.

Copyright held by owner.
Publication rights licensed to ACM. \$15.00

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

A simple method of writing a collaborative system.

BY SPENCER RATHBUN

Parallel Processing with Promises

IN TODAY'S WORLD, there are many reasons to write concurrent software. The desire to improve performance and increase throughput has led to many different asynchronous techniques. The techniques involved, however, are generally complex and the source of many subtle bugs, especially if they require shared mutable state. If shared state is not required, then these problems can be solved with a better abstraction called *promises* ([https://en.wikipedia.org/wiki/Promise_\(programming\)](https://en.wikipedia.org/wiki/Promise_(programming))). These allow programmers to hook asynchronous function calls together, waiting for each to return success or failure before running the next appropriate function in the chain.

Using the design pattern described in this article, programmers can build collaborative systems of threads or processes that are simple and easy to

reason about because of the abstraction provided by *promises*. As *promises* are available in many languages, this design pattern can be implemented in whichever language is most appropriate or even multiple languages for each process.

In the basic design, each piece of work requested of a worker creates a *promise* for that work. Dependent work hooks onto this *promise* and waits for it to complete before starting. If the work gets requested again, the new worker simply hooks onto the current *promise* chain and waits for the results with everything else. In short, the *promises* abstract the locking and allow the programmer to concentrate on the overlying pattern. By limiting the set of possible combinations the system can reach, handling all possible situations becomes simple, obvious, and removes the largest class of errors inherent to concurrent programs.

Developers often need to distribute work across a cluster of machines, processes, or threads. These distributed systems usually require complex code to ensure only one worker does each piece of work and that no work is missed, but it only gets worse when some of the work involved depends on previous pieces of work. How do these pieces synchronize themselves? When does a worker wait on another doing the work, and when does it take the work for itself?

This article shows how these and other problems easily resolve themselves without requiring a complex system of management, but instead use *promises* to create a state machine that simplifies the situation immensely. Unlike many distributed-programming algorithms, these workers collaborate to ensure only one of them does a unique unit of work and communicates with those workers that need that data. This eliminates the need for worker management, scales to an arbitrarily large cluster, and, most importantly, makes it easy to write and reason about the code.

What kind of work can be done in



such a system? The answer, as it happens, is any work that can be uniquely named. Using a simple data structure to keep the name of the work with the associated promise keeps any outstanding work from being reprocessed and allows new requests to hook onto the outstanding promise. This simply and easily coordinates among workers and presents a simple API for programmers in whichever language they prefer, so long as it supports promises. The API is especially useful in server software because the work could be request processing for a Web server, function computation for a MapReduce algorithm, or database queries.

Whatever is involved, it is unimportant to the system itself, so long as all of the workers use the same naming scheme. The simplest solution is to hash each piece of work to get this

unique name. This is effective for most types of work, because they are easily represented as text and hashed.

When a worker receives one of these pieces of work, it requests a promise from a central authority within the system. As the maintainer of these promises, the central authority abstracts the complexity from the workers. This makes for an easy interface for programmers of the system and provides a single place to add a distributed-locking algorithm, if one is required. Once the worker receives the promise, it checks if it should begin processing, based upon which end of the promise it received. Otherwise, it simply hooks up the appropriate function to resolve upon completion of the promise it received. Once the worker actually processing the request finishes, it sends the data down

the promise. This notifies all of the other workers with the results and allows them to process their completion handlers in parallel. This system lets programmers easily write concurrent processing systems without getting bogged down in the details of locking or complex lockless algorithms.

On a larger scale, if the system design calls for multiple servers or processes, the central authorities in each process must implement some kind of synchronization among themselves. Programmers must take care to determine where they fall in the CAP spectrum of such a distributed system. The promises do not, and cannot, solve the distributed consensus system and, indeed, do not even know such coordination is happening. The best solution is to abstract the problem to an outside source designed specifically for it. If

the work can be completed multiple times without detriment, then an available and partition-tolerant algorithm would be most appropriate. As much as possible, no worker will duplicate effort, but no disaster will occur in the edge case. For work that can happen only once, a consistent and partition-tolerant algorithm should be used instead for the locking mechanism. Some workers will not be able to process in the case of a network partition, but that is the trade-off for consistency.


Upon completion of this system, programmers no longer need to worry about the details of how to synchronize. They can expend their effort instead on the problem they actually need to solve.

What Are Promises?


Since promises provide the core capabilities used by this algorithm, it is important to understand them. Fundamentally, a promise has two components, herein called deferreds and futures. A deferred is the input side of a promise, the piece where work actually happens. Once the work is accomplished, the deferred is resolved with the result. If an error occurs, the deferred can be rejected with the error instead. In either case, the future is the output side of a promise, which receives the result. Futures can be chained, so one future can act as the deferred for other futures.

Every future takes two functions: success and failure. If the deferred gets resolved, then the success function is called with the resulting value. If the deferred gets rejected, then the failure function is called with the error instead. If a future does not have the appropriate function defined, then the value bubbles up to any attached futures.

Because futures can have an error function attached, they do not need to have error-handling code inside the success function. This avoids one of the problems of callbacks: mixing error handling with successful processing code. But what if the error is recoverable? Then there is an intermediate future with only an error function. If an error occurs, it can fix it and then resolve itself. This calls the next future's success function. If no error occurs, then the intermediate function simply passes the value to the next



This system lets programmers write concurrent processing systems without getting bogged down in the details of locking or complex lockless algorithms.



future's success function. In either case, the appropriate processing occurs. This allows the promise to work as an asynchronous equivalent to a try/catch block.

Promises also have another feature, unlike their cousin, the callback: a deferred can be resolved or rejected with a future. When this happens, a new promise is inserted into the chain before any futures waiting on the current step. An important use of this feature is recovery code in a failure function. By resolving it with a future, a failed attempt to get data can try again without the other futures waiting on the data knowing an error occurred. By using this technique, developers can encapsulate portions of a programming task into loosely coupled atomic functions whose ordering can easily be reasoned about, as shown in Figure 1.

States. Because promises must run either their success function or failure function, the overall system maintained by the central authority has only a few states that must be reasoned about. First, there is the waiting future. Once a worker begins processing, the future could be left in a waiting state forever, since its associated deferred has not been resolved. To deal with this, the central authority should expose a function to the worker with which it can receive periodic notifications as a health pulse. Even using these periodic notifications can still have loss, however. The central authority should sweep the currently outstanding work and reject any promises that have waited too long. This will trigger the retry code down the chain.

Once the promise gets resolved with the results of the work, it must start running through the future chains attached to it. During any of these steps, an error could occur. Luckily, promises transform thrown exceptions into calls for the next catch function. Passing these errors up the chain means they will get properly handled in the catch functions or passed along to the external callers. This prevents work from silently failing, or completing in a bad state.

Finally, if the work concludes, everything has functioned as desired, and the system has processed the request. Because promises abstract out

to these three states, each state can be handled in the code in an obvious manner. This prevents many of the subtle bugs that plague distributed systems.

Basic algorithm design. At its core, the collaboration algorithm follows five simple steps, as shown in Figure 2.

1. The unit of work gets an agreed-upon name so that multiple requests for the same work get the same future. Without an agreed-upon naming scheme, no worker could know if another is processing that work already.

2. The `deferred` is requested from the hash table of currently operating requests. If no `deferred` exists, one is created, added to the hash table, and returned to the requestor. If there is already a `deferred`, the associated future gets returned instead. Changing what gets returned in each case allows the worker to know if it should begin processing or simply wait.

3. A catch function is attached to the future. This catch function retries if an error occurs, running the original function. By doing so, the new attempt will catch and try again. This allows for infinite retries, or by tracking a count of retry attempts, it can fail up the chain instead of recovering. Since every worker dealing with this promise attaches its own catch in parallel, they all retry on failure. Any one of them might win the right to try again, providing a built-in communications system between the involved workers.

4. The work to do on the input gets attached so processing can occur once the data has arrived. It should not matter if multiple workers do the work at the same time, since their promises complete in parallel. Additionally, by returning this promise, work can be chained on preceding pieces of work getting completed.

5. Finally, if the worker received the `deferred`, then it should resolve the `deferred` once it has finished processing the work. If a failure occurs during processing, then the `deferred` with the error should be rejected, which will notify all workers they should retry.

What happens when the work consists of small composable pieces? Recursively requesting each piece of work means the `deferred` can be resolved by the future for the preceding piece. This chains the small pieces together, with a worker walking back-

ward down the chain of things it needs until it finds either the beginning or another worker it can take advantage of. As each promise is asynchronous, the worker no longer needs to keep hold of it, and as the futures get data and are ready to run, any worker can pick them up. If there is only one chain, it will necessarily process one step at a time, the same as if no system were surrounding it. The pieces cannot go faster because of multiple workers, as they depend on the previous pieces, but shared pieces or multiple requests can take advantage of the already-processing parts.

What Happens in a Distributed System?

This basic algorithm requires a more complex central authority if the architecture precludes sharing a promise globally—for example, running on separate servers, or an application written in a language that cannot share promises across processes. In these cases a locking mechanism is required to keep the distributed central authority synchronized for a worker processing a promise. A Redis server or a

chubby cluster can be used to provide this lock, since both systems are used for distributed locking. Care must be taken to determine the chosen system provides the necessary consistency or availability requirements. In addition, a method of broadcasting to the additional workers assists in speeding completion times. Instead of waiting for them to check up on the lock, a message can be passed indicating the work has been completed. After the internal `deferred` is received for the current process, a request must be made to the locking service.

If the locking service gives the lock to the requesting worker, it resolves the `deferred` as before. Another promise is added in parallel, however. On completion, this promise broadcasts a completion message to the distributed peers. The peers listen for such announcements and resolve their local `deferred` with the data by getting it either from a shared datastore or out of the broadcast message.

These steps must be added after the local `deferred` is requested. Since the local worker does not know if any other worker has already start-

Figure 1. Promise insertion.

```
// Pseudocode demonstrating future insertion for error recovery

var future = doSomeWork();

// error handler function

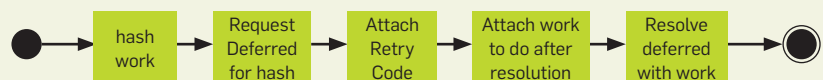
var retry = function(error) {
  console.log("Had an error: ", error);

  //get a new promise for the work
  var newFuture = doSomeWork();

  // attach error handler to new promise and
  // return chained set, inserting into the chain
  // actual code should not blindly retry, as that could
  // potentially cause an infinite loop
  return newFuture.catch(retry);
};

// attach error handler and then later work
// if an error occurs, the catch inserts the
// new future to try again
future.catch(retry).then(doMoreWork);
```

Figure 2. Basic cache algorithm.



ed, it must attempt to take the lock whenever it starts. Failure to take the lock indicates some other worker has already started processing. This self-regulates the workers across nodes. The steps detailed in Figure 3 show what happens.

Staying alive. In any case, the worker processing the work needs to keep its peers updated on its progress. To that end, a periodic notification is sent from the worker to the central authority, acting as a health pulse. If this pulse is absent, then the central authority triggers the retry code for the other workers, recognizing the death of the one doing the work. Using an already-established code path to indicate failure means there are fewer states to reason about, and the workers can be certain that if a lock is out for work, someone is processing or will be soon.

Additionally, promises protect the system from false death of a worker. If the worker fails to continue sending out the health pulse, another worker takes over the work, but the original worker continues processing the work and will probably complete before the original. The original deferred, however, has already been rejected by the missing health pulse, prompting another worker to take over. Promises can be completed only once, so when the original worker resolves the original deferred, nothing happens. All of the promises have had a new promise inserted into their chains—that is, they have all been told the original worker is defunct and a new one is processing. No one needs to know or care if the original worker has died or will complete its task eventually.

As Figure 4 shows, the system has two main states: working and waiting. If a worker succeeds in getting the deferred, it enters the working state. While there, the worker must send out notifications to the central authority indicating it is still alive. If the system uses a locking service, then the notification also updates the lock time to indicate work is ongoing. This health pulse prevents other workers from waiting forever. Without it, a failed worker would hang the system forever.

Workers in the waiting state will get the completed work and finish, or else they will get rejected and run

Figure 3. Cache algorithm with locking.

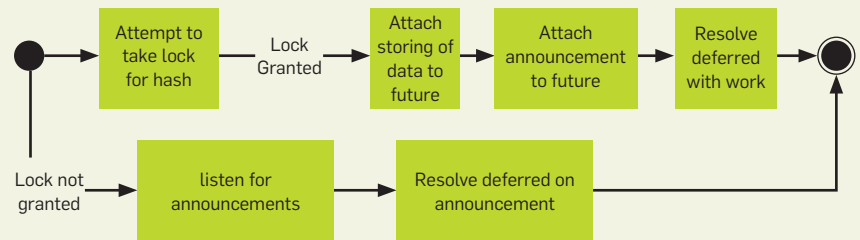
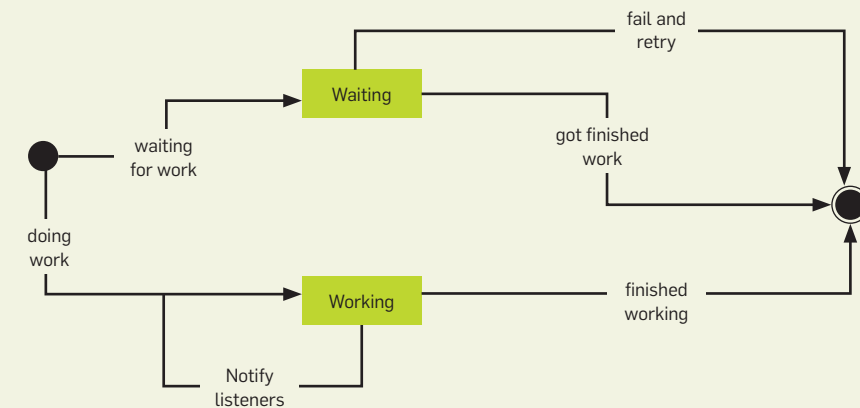


Figure 4. Cache notifier.



the retry code. If the central authority times out waiting for a worker's health pulse, then it checks with the locking service, if it exists. If there is no service or the lock time has failed to get an update, then the waiting workers have their deferred rejected with the retry error, causing all waiting workers to follow the basic error-recovery steps. A new worker will begin doing the work, and if the original worker has not actually failed, it will resolve the previously rejected promise. Resolving or rejecting a completed promise is a noop (no operation) and gets ignored, as in Figure 5.

Implementation experience. The design described here originally came out of the need to integrate a caching system for queries on top of a Mongo database cluster. Multiple Node.js processes spread across different servers would run queries for users simultaneously. Not only did the results need to be cached, but also large, long running queries would lock the Mongo cluster if run many times in parallel. Since each query was a pipeline of steps to take, the queries could also build upon each other. A query match-

ing certain documents, then sorting them, could have a grouping step added afterward.

Naturally, recursing backward through these steps and hashing the current segments provides a perfect naming scheme. If a user has already run the query, then the name can be looked up in the Redis database used to store the results and the query does not need to run. Otherwise, following the design described here, the appropriate promise is requested. If another user has already started the query, then no work needs to be done. This prevents overloading the Mongo database with the same queries continuously and gets the results back faster for multiple requests for the same query.

Working backward through the set of query segments, the worker can hook onto any already-cached or running queries. Upon getting the results for an intermediate section, however, additional Mongo query work needs to happen. The mungedb-aggregate library (<https://github.com/RiveraGroup/mungedb-aggregate>) provides a one-to-one set of operators allowing the worker to finish any addi-

Figure 5. Sample retry code.

```

promiseHealthCheck: function () {
  var self = this,
      keys = Object.keys(self.currentlyWaiting),
      currKey, i, recache, fail;
  // loop over the current waiting work
  for (i = 0; i < keys.length; i++) {
    currKey = keys[i];
    // set up a failure chain if we need to retry
    recache = Q.defer();
    fail = recache.promise.catch(this.cleanupAndNotify.bind(this, currKey));
    // if it is one of our local workers, check if it is still building
    else if (currKey in this.lastUpdateTime && this.lastUpdateTime[currKey]) {
      if ((Date.now() - this.lastUpdateTime[currKey]) < this.maxWaitTime) continue;
      else {
        // have someone else build this
        // resolve the promise with our failure chain
        // this cleans up, then sends a retry failure
        // to the workers
        this.currentlyWaiting[currKey].resolve(fail);
        recache.reject(Cache.RejectReason.RECACHE);
      }
    } else {
      // otherwise, check the distributed lock for last update time
      this.get(this.lock(currKey))
        .then(function (result) {
          // someone is caching, so check updateTime
          if ((Date.now() - result.lastUpdateTime) < self.maxWaitTime) {
            return result; // wait time is not exceeded, so do nothing
          }
          else {
            // dud lock
            self.currentlyWaiting[currKey].resolve(fail);
            recache.reject(Cache.RejectReason.RECACHE);
          }
        })
        .catch(function (err) {
          // no one has the lock
          self.currentlyWaiting[currKey].resolve(fail);
          recache.reject(Cache.RejectReason.RECACHE);
        });
    }
  }
  // set up the next health check
  setTimeout(this.promiseHealthCheck.bind(this), this.healthCheckMs);
}

```

tional query segments. Without this library, the whole query would have had to run in Mongo, and running composable segments would have been infeasible. The only useful thing would be to cache the results, and there are already solutions for doing so.

The Node.js process has its own central authority to keep a single promise for any concurrent requests for a single query. This functioned well when a single process was in use for small deployments. Multiple concurrent requests would chain together, and they would transparently collaborate on the work. Unfortunately, the Node.js promises could not cross the process and server boundary. Upon scaling out to multiple servers, this necessitated the development of the fully distributed version of

the system. Since Redis already had a place as the caching database, it made a natural abstraction for the distributed lock required. Redis was considered the best algorithm to use because it is available and partition-tolerant. If separate servers lost contact, then having each one run a single copy of the query was considered the right thing to do, as the results from the separate runs would be the same. The requestor always gets the results, and as much as possible the system minimizes the workload.

Once the implementation was started, the only major problem encountered was how to provide a health check for a currently processing worker. Luckily, the Q promise library (<https://github.com/>

kriskowal/q) provides a notify function to its implementation. This provides a convenient hook for the central authority to listen on. The listener function updates the remote Node.js processes through the update time in the Redis database.

Throughout the implementation's life in production, there has not been a single unhandled error. Even with many queries hitting the system at once, and Node.js processes dying abruptly while processing, the system has continued to work as designed. Unlike complex locking code about which programmers have difficulty reasoning, the promise chains provide a clear and unambiguous way to coordinate the queries. Upon reading the code, a programmer can clearly see it will work and which part is responsible for which task. Other systems may provide the same utility, but few are as easy to use and modify.

Conclusion

The system defined by these promises provides strong guarantees. Exceptions during resolution cause a rejection to flow through the promise, so work does not hang. Promises chain, so workers wait on previous work and then kick off to do their work on resolution. Notification listeners provide timeouts to prevent work from hanging. As few workers as possible will do the work, and the first one to succeed is the canonical answer provided to everyone. Most importantly, this algorithm provides a simple and correct method of writing a collaborative system. ▣

Q Related articles on queue.acm.org

Software and the Concurrency Revolution

Herb Sutter and James Larus

<http://queue.acm.org/detail.cfm?id=1095421>

A Conversation with Ray Ozzie

<http://queue.acm.org/detail.cfm?id=1105674>

Parallel Programming with Transactional Memory

Ulrich Drepper

<http://queue.acm.org/detail.cfm?id=1454464>

Spencer Rathbun is a senior software architect for Rivera Group. His specialty is designing big-data solutions for a customer base that focuses primarily within the software architecture domain. To this end, he specializes in architecting creative solutions for real-world problems using structured/semi-structured data.

Copyright held by owner.

Publication rights licensed to ACM. \$15.00

DOI:10.1145/2663341

Despite continuing media coverage, the public's privacy behaviors have hardly changed.

BY SÖREN PREIBUSCH

Privacy Behaviors After Snowden

MASS MEDIA HAVE been reporting on global-scale state surveillance following former NSA contractor Edward J. Snowden's exposure of PRISM in June 2013. Extensive, continuing news coverage makes this revelation a natural experiment. In a longitudinal study from May 2013 to January 2014, I examined the immediate and longer-term effects on Web use in the U.S. I combined evidence of privacy self-protection and behaviors indicating an interest in privacy. Users' interest rose after the PRISM revelation but returned to and even fell below original levels despite continuing media coverage. I found no sustained

growth in the user base of privacy-enhancing technologies (such as anonymizing proxies). The worldwide public revelation of PRISM affected individuals' interest less than other breaking news concerning sports and gossip. My results challenge the assumption that Web users would start to care more about their privacy following a major privacy incident. The continued reporting on state surveillance by the media contrasts with the public's quickly faded interest.

Many details about PRISM and related cyberintelligence initiatives, including Tempora and XKeyScore, are still unknown by the public,⁸ though news coverage following June 6, 2013 was thorough and steady.⁵ For the first time since 2009, the U.S. national newspaper *USA Today* reported on government surveillance on its front page,⁷ and *The Washington Post* ran a front-page article on the invasion of privacy for the first time in 2013,¹⁶ prompting contentious discussion of the revelations among privacy experts, as well as politicians and advocates, worldwide.

Mass media is able to set the public agenda through increased coverage of a topic.¹² However, little is known about how media salience of privacy might steer public opinion and behavior toward caring more about privacy. Previous media analyses suffered from unreliable and temporarily sparse measures of privacy concern¹⁷ or simply ignored the public's response.²¹ The PRISM

» key insights

- In June 2013, Snowden revealed worldwide government surveillance, a privacy shock also representing a natural experiment in the study of privacy behaviors.
- The effect on the public's interest in privacy and behaviors was limited and short-lived despite long-running, in-depth media coverage.
- High-resolution data from multiple sources shows the greatest privacy meltdown of our time had only a small impact on Web users beyond debates among journalists and academic researchers.



revelations blurred the distinction between privacy in state/citizen and company/customer relationships, as corporations were revealed to be accomplices in government surveillance. Some citizens reacted by modulating their consumption of Web-based services.

Mistrust by German Internet users of government and corporate data processing increased by nine percentage points in 2013¹ over 2011, but only a minority reported they had changed how they manage their personal data.² Google Trends suggested a chilling effect that varied by country on users' propensity to issue search queries that might "get them into trouble" with the U.S. government.¹¹ However, such public-opinion snapshots derived through ad hoc surveys were unable to capture consumers' volatile and ephemeral reactions. Annual polls provide only coarse temporal resolution, when continuous records of behaviors are needed to measure the effect of an event like the PRISM revelation. Privacy surveys also suffer from such shortcomings as respondents' lack of commitment, leading to post hoc rationalizations and socially desirable but unreliable answers.

Acquiring data on user behavior is difficult,⁴ and companies rarely disclose usage statistics of their Web-based services. When the Web search engine DuckDuckGo, which advertises its superior privacy practices, attributed a rise in its daily queries to the PRISM revelation, it did not include user counts.^{3,20}

PRISM also represents an opportunity to carry out a Web-scale longitudinal study of privacy behaviors, and this article offers the first high-resolution analysis using primary sources of data. I combined indicators from multiple primary data sources to explore the evolution of consumers' privacy behaviors. Privacy self-protection can be observed directly, and information-seeking activities can be interpreted as indicators of an interest in privacy.

The mainstream media reported the first details about the NSA's state surveillance programs on June 6, 2013, hereafter called "PRISM day"; the three preceding weeks are the reference period. Over the following six and 30 weeks, respectively, I analyzed the immediate extended reactions by daily



The continued reporting on state surveillance by the media contrasts with the public's quickly faded interest.



measurements of several key metrics:

Information seeking. Web search and browsing, as measured through visits to the privacy policy posted at Microsoft.com, one of the companies mentioned in the early media coverage, and also through visits to privacy-related Wikipedia pages, and privacy-related webpages in general;

Privacy-enhancing configurations. Privacy settings made in the Internet Explorer Web browser, as well as adoption of Tor and the Anonymox Firefox add-on, both connection-concealing mechanisms; and

New webpages. As a control variable, media coverage in terms of number of new webpages created around the topic of government surveillance and declines (following the initial reports) on the existence of PRISM.

Methodology

I analyzed user behavior for U.S. English-language consumers. The U.S. is the home of PRISM and related programs, as well as where the first related media coverage appeared, an obvious yet deliberate limitation of my study. Future work will examine differences across countries, but for this article, I avoid confounding factors of language and culture. The U.S. is the one country for which the most comprehensive data is available for Web searches, visits to corporate privacy policies and Wikipedia pages, and use of Tor/Anonymox. I considered the totality of Web users within the U.S., not a sample. As an exception, I examined general page-visit data and Internet Explorer privacy settings on a global sample of users, owing to a lack of country information. I thus made any comparisons with care.

I recorded Web search behavior for manually chosen PRISM-related keywords (see Figure 1) I then cross-checked against privacy-related keywords inferred from query reformulation, a standard technique in studying Web search behavior. I favored semi-manual selection over a necessarily retrospective data-driven approach that would deny the post-Snowden sea change on the public perception of privacy. U.S. user counts were from the totality of spell-corrected queries on Microsoft's search engine Bing (such as "privcay," which also counted).

Fluctuations in query-term popularity happen naturally in any year; for instance, daily search volume for the term “Wikipedia” varied by up to 33% during the reference period, but its relative share varied by only 0.01 percentage point. To accommodate for varying total search volumes, namely weekday/weekend, differences in general popularity, and varying search patterns across users, I recorded changes in the proportion of users initiating a given query relative to all users, indexed to the reference period. I used weeklong binning to even out weekday/weekend effects, so start days did not have to be aligned when comparing responses to different events.

I measured webpage visits to Microsoft’s privacy statement at <http://www.microsoft.com/privacystatement/> by its four main sections—Bing, Microsoft.com, Xbox, and other Microsoft products—that together accounted for more than 99% of all page views. I also measured general browsing to webpages on PRISM-related topics. I counted page visits if the URL or the title of the page included the keyword. I collected data from a sample of users consenting to share their browsing history, possibly leading to bias toward less-privacy-concerned users.

Tor users refresh their list of running relays on a regular basis. These requests to directory mirrors are useful for estimating the number of Tor users.²³ In line with the other analyses, I focused on U.S. users, who contributed the largest national user share (18%) in the reference period. Anonymox users install a Firefox add-on for easy anonymous browsing.¹⁵ Anonymox is among the top-10 “privacy and security” add-ons in the U.S. and the only one to advertise itself as avoiding government threats to privacy. I counted active users, again for the U.S., by daily update pings.

I assessed media coverage by counting online documents mentioning three keywords: Snowden, PRISM, and surveillance. Discovery was through a subset of the Web index maintained by Bing,¹³ with counts relative to all new documents. The term “privacy” produced false positives and was excluded, as many websites must now include (by law) a link to their “privacy” policy or statement.

Web Search Behavior Post-PRISM

Web search queries represent a lens into user behavior,⁴ serving as a proxy for user interests. For example, the Flu Trends analysis uses Web search as a precursor of medical conditions and behavior.⁶ The inaccuracies subsequently found by others in the Flu Trends models are due mainly to positive feedback loops⁹ that do not apply to my analysis. I measured interest in PRISM-related topics by the number of queries issued about them.

The composite measure of search behavior based on all relevant, automatically discovered “privacy”-related keywords showed neither an immediate ($p = 0.14$) nor longer-term trend ($p = 0.84$, F-test over linear regression).

Short-term evolution. Considering individual topics, Web search queries about the NSA, surveillance, and the government saw a modest spike on PRISM day, June 6, 2013 (+0.08% to +0.02%, all search volume numbers in percentage points). Searches for “Snowden” increased when that name was revealed two days later, June 8. Snowden and NSA were the only terms that continued to attract elevated search volume over the six weeks following PRISM day, with a steady downturn for “NSA.” There was only a slightly increased interest

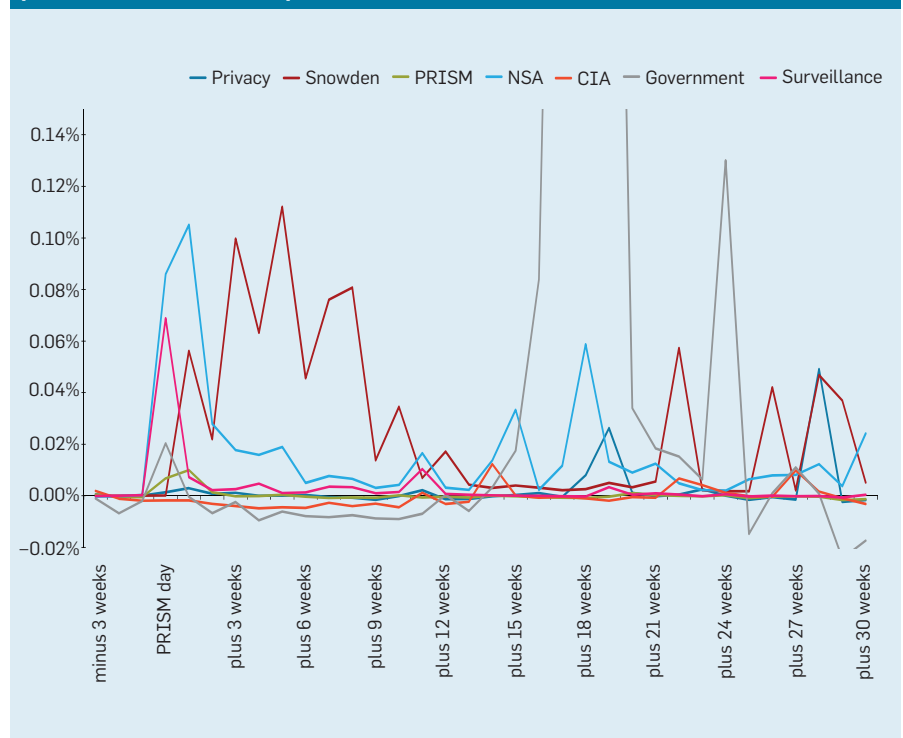
in “privacy,” with a maximum increase of 0.003% during the week following PRISM day. Searches for PRISM itself rose by 0.01% in the same week but fell to the original level afterward, as in Figure 1. The CIA, another U.S. federal agency involved in collecting foreign intelligence, did not attract any more queries, and related search volume decreased after PRISM day.

Whereas the celebrity Snowden remained popular among search users ($p < 0.0001$, t-test, before versus after PRISM day), privacy could not maintain a larger user base ($p = 0.4$). This observation was corroborated when I examined search for news results only. The average user base searching for “Snowden” rose by 0.007% when comparing before and after PRISM day, whereas news searches for privacy were unaffected, with less than 0.00001% change.

Searches for privacy-enhancing technologies saw a very small increase, with general encryption (at most +0.001% in week one), Tor, a system for users to conceal with whom they communicate (+0.002%, same week), and PGP, a system for encrypted messaging (+0.0003%, same week).

Longer-term evolution. Interpreting the longer-term evolution of search be-

Figure 1. Changes in search-term volume after PRISM day, June 6, 2013, in percentage points relative to reference period.



havior is more difficult, as the increasing time from the reference period introduced seasonal variations and other influential events. By week 11 following PRISM day, all terms I considered had a search volume elevated by no more than 0.02%, as in Figure 1. Snowden and NSA continued to spike in September/October and November/December 2013, respectively, up to +0.06%. I ob-

served no notable increases for PRISM, surveillance, or CIA.

Long after PRISM day, “privacy” spiked in weeks 19 and 28, with 0.03 and 0.05 percentage points, respectively, above the reference period, reaching levels unattained directly following the revelations. Besides continued media coverage about government surveillance, those weeks coincided

with media reports about Facebook removing a privacy setting and Google removing privacy enhancements in Gmail and Android. Likewise, the spike in “government” searches from week 15 to 21 (up to +0.50%) can be attributed to the U.S. government shutdown in October 2013.

Some variance of query volume may be explained by one-off versus ongoing information needs. Users wanting to keep up with Snowden’s whereabouts or with new revelations about the NSA would repeatedly issue these queries, whereas they may have sought only once background material on privacy or surveillance. Web-browsing behavior on Wikipedia and the general Web, discussed in the following sections, provides further evidence on one-off versus ongoing information seeking.

Benchmarking against other events.

The public revelation of PRISM seems to have had much less of an influence on searches related to privacy than on other events. I contrast it with three topical issues with societal impact, chosen manually for being globally recognized events happening shortly after related media reports. During the summer of 2013, mass media reported on Hassan Rouhani’s election as president of Iran, a geopolitical issue (June 15, 2013), the U.S. Open golf tournament, a four-day major sporting event (June 13–16), and the birth of Prince

Figure 2. Changes in Web search volume for PRISM-related topics compared to other events in international politics, sports, and gossip.

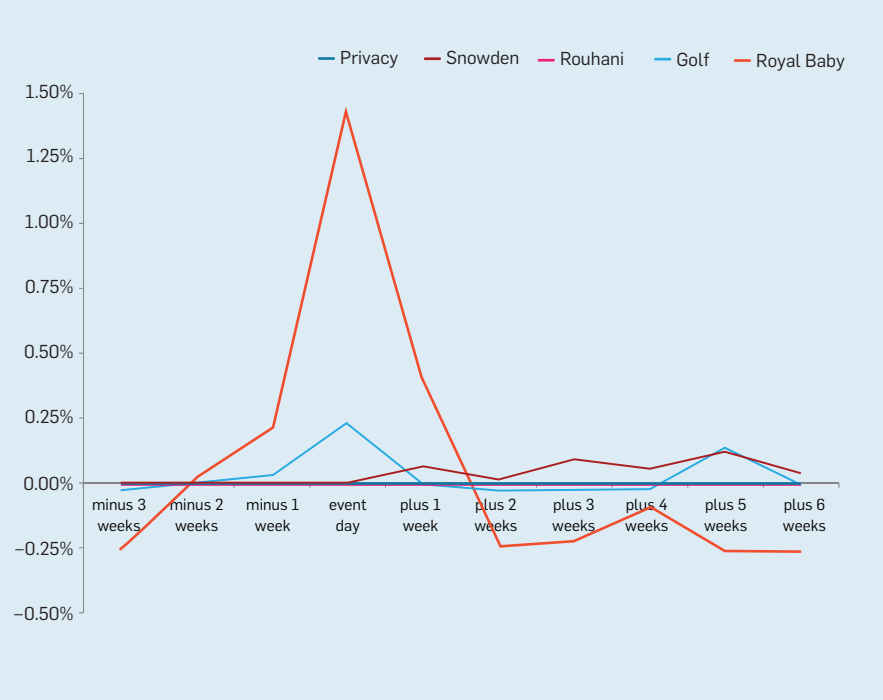
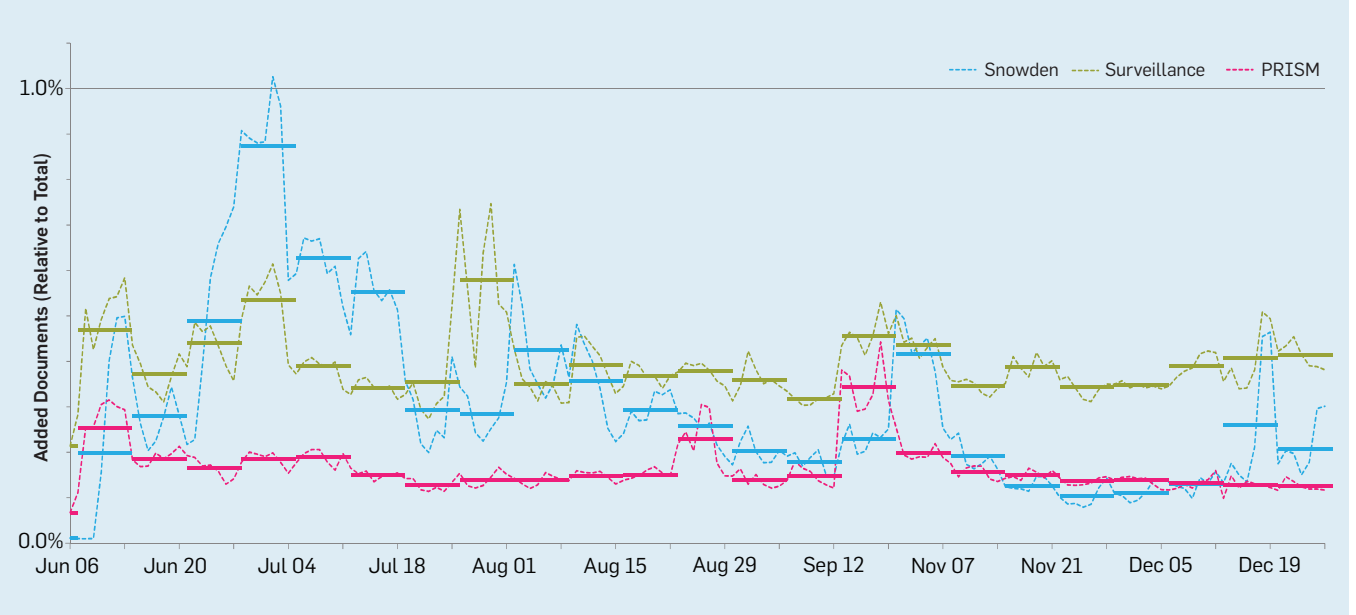


Figure 3. New online documents mentioning “Snowden,” “PRISM,” and “surveillance,” respectively, over the 30 weeks following the original revelation of PRISM, June 6, 2013; the horizontal lines are averages for each seven-day period following June 6; data for October and late December 2013 are missing due to a system failure in the system used to record the data.



George of Cambridge, the “royal baby,” (July 22) (see Figure 2).

The number of queries for Rouhani increased 0.0039% on the day of his election, falling behind interest in “PRISM” but surpassing “privacy” on PRISM day. However, interest in Rouhani faded even more quickly; for example, the volume of search queries for Rouhani was significantly higher in the week following his election compared to the three weeks preceding it ($p = 0.0001$) but no more significantly in the second week ($p = 0.3$). Golf searches peaked during the four-day U.S. Open tournament (+0.25%) and were elevated in the preceding week (+0.04%) but fell sharply and below the original user base thereafter.

The birth of the royal baby showed the greatest daily increase, with +1.5% on the day of his birth, July 22, and further still, +0.41%, in the following week; the proportion of users searching for the new heir fell thereafter. Interest may be underestimated, as numbers were high before the corresponding reference period; compared to one month earlier, more than one hundred times more users searched for the royal baby on the day he was born. On 12 days during the two weeks preceding his birth, at least 0.1% of all users searched for the royal baby, and search-volume share peaked at more than 2% on July 23. Among search terms relating to PRISM, NSA and Snowden attracted their own peak search volume share with 0.4% on July 7 and June 11, respectively.

Browsing Behavior Post-PRISM

Along with Web search, browsing behavior that manifests as information seeking indicates an interest in privacy topics. I thus counted the number of users who visited webpages about Snowden, PRISM, privacy, and surveillance, using the same temporal binning as before—weeklong intervals following PRISM day. Also as before, I assessed population numbers against the three weeks prior to PRISM day.

Users browsing webpages about Snowden increased by two orders of magnitude, by far the most growth among all topics considered in my survey. Snowden stayed popular during all six weeks following PRISM day ($p < 0.00001$, t-test). PRISM and surveil-



Snowden's revelations brought few new users to privacy-enhancing technologies.



lance attracted more users as well, increasing by 95% and 250%, respectively. Whereas PRISM was able to maintain its public interest (significantly elevated page visits for all six weeks, $p < 0.0001$), the number of users visiting webpages on surveillance decreased steadily ($\rho = -0.93$, $R^2 = 0.87$) during that time and was no longer different from the reference period in the fifth week ($p = 0.22$). There was no significant increase in the number of people visiting privacy-related webpages. Numbers increased slightly in the week following PRISM day (+4%) but fell below the original levels thereafter (up to -13%).

Looking at longer-term trends, only “Snowden” continued to attract a significantly larger audience. Visits to webpages about PRISM or privacy fell to or below their original levels, and numbers for surveillance-related webpages were sporadically above the reference period in August and November/December, albeit with no clear trend.


Wikipedia is a standard online reference that fulfills general-purpose information needs. The English Wikipedia pages on PRISM and Snowden were created June 7 and 9, 2013, respectively, and data does thus not exist for the reference period. The number of page views increased significantly ($p < 0.001$) for the encyclopedia entries on privacy and surveillance, by 23% and 75%, respectively. But by week two, page views for the “privacy” Wikipedia article had already fallen to and below the reference period, with a later increase in September 2013; 25 numbers for the “surveillance” article were back to original levels by week five. Although the “Edward Snowden” article was created after the reference period, the Wikipedia article statistics provide further insight into the question of one-off versus ongoing information-seeking behavior through Web search. There is no indication Snowden would attract readership on an ongoing basis, while privacy and surveillance would not. On the contrary, the weekly interest in Snowden shrank even more drastically (-80%) than privacy and surveillance (-13%), respectively, when comparing the immediate reactions after PRISM day to longer-term evolution.

The early media coverage on PRISM reported the NSA would be “tapping directly into the central servers of nine leading U.S. Internet companies,”⁵ including Microsoft, and individuals seeking information may have consulted the privacy statements published on the corporate Website. Although consumers rarely consult privacy policies, they may have suddenly become eager to learn the details of corporate practices, including data sharing “when required by law or to respond to legal process or legal requests, including from law enforcement or other government agencies.”¹⁴ Visits increased by up to 12% in the first six weeks following PRISM day and stayed significantly above the reference period for the entire extended range ($p < 0.01$) with an upward trend ($R^2 = 0.10$, $p = 0.09$, F-test). However, I observed significantly higher numbers only in the third week following PRISM day, with no significant increase in the week immediately after the revelations ($p = 0.29$). A seasonal effect cannot be ruled out, as data from the preceding year was unavailable due to changes in how visits are counted.


Privacy-Enhancing Technology Post-PRISM

The divergence between privacy attitudes on the one hand, expressed by interest and search, and behaviors on the other, is well documented¹⁸ and was corroborated for PRISM. For example, Facebook reported diminishing trust but no impact on frequency of use of its social networking service.²⁴

Tor is a privacy-enhancing technology that allows users to conceal their location and browsing habits by routing Web traffic through multiple relays. Tor markets itself as a protective measure against network surveillance; “Browse anonymously with Tor” was featured in the *Washington Post* as the first of “five ways to stop the NSA from spying on you.”¹⁰ Tor use increased significantly— $p < 0.01$, plus up to 10%—but only in weeks three to five following PRISM day. Tor remained a niche technology; the growth of its user population was small (up by a maximum of 15,000 users in week four) and tiny compared to September 2013 when misuse of the



At the peak on July 2, 2013, “Snowden” appeared in more than 1% of all newly found online documents.



Tor infrastructure by cybercriminals drove user numbers to quadruple.

Anonymox is an alternative browser add-on service for hiding one’s IP address. Anonymox use more than doubled in 2013 to more than 200,000 users at the end of the year; I corrected for this trend in my analysis. PRISM day represented a temporary high between May and July but itself generated no spike in use ($p = 0.24$ for the largest increase in week two).

Configuration of the privacy settings in all Web browsers represents a third indicator of behavioral changes following PRISM day. Whereas the configuration of a proxy still requires technical skill, browsers are designed to be configurable by ordinary users. I used data from a sample of Internet Explorer users who consented to share usage metrics and counted how many of them selected the “privacy” tab under Internet options in the preceding month. The prevalence of this behavior increased after PRISM day by up to 2.8 percentage points compared to the reference period, approaching significance ($p = 0.02$).

Configuring the privacy options is still among the lesser-used browser features, and I found no other significant increases over the extended period. In comparing Tor/Anonymox with Internet Explorer data, I noted use of anonymizing proxies is an ongoing privacy effort, whereas adjusting browser settings toward more privacy protection is a one-off operation. The available data could still overestimate the proportion of users; for example, I counted users who opened the privacy-settings tab, without necessarily making a change, let alone activating more restrictive settings. Moreover, the sample size for Internet Explorer telemetry data was small.

Evolution of Media Coverage

I interpreted the data on consumers’ privacy behaviors, described earlier, against media coverage. Apart from a dip in the second week following PRISM day, the terms included in the survey followed a consistent upward trend. For example, more documents mentioning Snowden and surveillance appeared in week six than in week one. At the peak on July 2, 2013, “Snowden” appeared in more than

1% of all newly found online documents. And on each of the following days, “PRISM” appeared in more than 240,000 daily new documents on average while competing with other initiatives (such as Tempora and XKeyScore) (see Figure 3).

Media coverage continued over the 30 weeks following PRISM day, with no noticeable downward trend for PRISM ($\rho = -0.00002$, F-test: $p < 0.0001$), surveillance ($\rho = 0.0000$, $p = 0.001$), or Snowden ($\rho = 0.0000$, $p = 0.004$). By the end of the study period—week 30 following PRISM day—the relative daily volume of documents about Snowden was 18 times as much as on June 6, 2013.

Conclusion

This article covers the first longitudinal study of the privacy behaviors of U.S. Web users as they might have been affected by Edward Snowden’s 2013 revelations about government surveillance and the general lack of communications privacy. I compared the use of privacy-enhancing technologies pre- and post-PRISM day, using Web search and browsing activity as proxies for interest in privacy and information-seeking behavior. My analysis of Web search behavior through Microsoft’s Bing search engine may have introduced a bias impossible to quantify, should it exist. However, external evidence suggests Bing may be more appealing to the privacy-aware,^{19,22} meaning the small increases I observed could still represent an overestimation.

I combined high-resolution data from primary sources that indicate the new public information on PRISM led to momentarily increased interest in privacy and protection. However, the spike was much less than for other news events (such as the royal baby and the U.S. Open golf tournament). It was also less than the increased interest following the removal of privacy-enhancing functions in Facebook, Android, and Gmail.

While media coverage of PRISM and surveillance was elevated for the 30 weeks following PRISM day, many privacy behaviors faded quickly. Visits to Microsoft’s corporate privacy policy page stayed high, but only certain privacy-related webpages kept

larger audiences—those on Snowden and surveillance—while Wikipedia articles about PRISM topics lost their increased readership. Snowden’s revelations brought few new users to privacy-enhancing technologies; anonymizing proxies experienced increased numbers through 2013, but PRISM did not add to the momentum. While Snowden himself was the only name to consistently attract large audiences, one may argue this interest had already separated from the erosion of privacy he had revealed.

Only longitudinal studies with high temporal resolution are able to reveal the influence of privacy invasions on people’s behavior. The paucity of such studies may be explained by the difficulty of obtaining expressive data. I had to rely on proxies (such as information-seeking behavior) for users’ interest in privacy that cannot be observed directly. My selection of data sources was partly pragmatic, aiming for rich data providing good coverage of the same population over an extended time.

I thus opted to focus on English-language users in the U.S.; this so-called “en-us” market is a standard geographic filter for many consumer services, allowing consistent scoping across different data sets. The trade-off between narrow but good data leads to an obvious limitation, as users may have chosen to pose as “en-us” in their software settings. Still, previous internal analyses indicate their proportion is negligible. I also plan to examine the effect PRISM had (and continues to have) in various countries. My results warrant contrasting the effect of governmental versus corporate wrongdoing in privacy issues. C

References

1. BITKOM (Federal Association for Information Technology). Internetautzer werden misstrauisch, July 25, 2013; http://www.bitkom.org/de/presse/8477_76831.aspx
2. Dierig, C., Fuest, B., Kaiser, T., and Wisdorff, F. *Die Welt* (Apr. 13, 2014); <http://www.welt.de/wirtschaft/article126882276/Deutsche-unterschaetzen-den-Wert-persoenerlicher-Daten.html>
3. DuckDuckGo. DuckDuckGo Direct queries per day (28-day average), July 2014; <https://duckduckgo.com/traffic.html>
4. Dumais, S., Jeffries, R., Russell, D.M., Tang, D., and Teevan, J. Understanding user behavior through log data and analysis. Chapter in *Ways of Knowing in HCI*, Springer, New York, 2014, 349–372.
5. Gellman, B. and Poitras, L. U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program. *The Washington Post* (June 7, 2013); <http://wapo.st/1888aNaq>
6. Ginsberg J. et al. Detecting influenza epidemics using

- search engine query data. *Nature* 457, 7232 (Feb. 19, 2009), 1012–1014.
7. Jackson, D., Davis, S., and Johnson, K. Obama’s spy plan includes Internet: NSA’s expansive reach revives an unsettled and vexing 9/11 debate. *USA Today* (June 7, 2013), A1.
8. Landau, S. Making sense from Snowden: What’s significant in the NSA surveillance revelations. *IEEE Security & Privacy* 11, 4 (July–Aug. 2013), 54–63.
9. Lazer, D., Kennedy, R., King, G., and Vespignani, A. The parable of Google flu: Traps in big data analysis. *Science* 343, 6176 (Mar. 14, 2014), 1203–1205.
10. Lee, T.B. Five ways to stop the NSA from spying on you. *The Washington Post* (June 10, 2013), <http://www.washingtonpost.com/blogs/wonkblog/wp/2013/06/10/five-ways-to-stop-the-nsa-from-spying-on-you/>
11. Marthews, A. and Tucker, C. *Government Surveillance and Internet Search Behavior*. SSRN Working Paper, Social Science Electronic Publishing, Inc., Rochester, NY, 2014.
12. McCombs, M.E. and Shaw, D.L. The agenda-setting function of mass media. *Public Opinion Quarterly* 36, 2 (Summer 1972), 176–187.
13. Microsoft. Bing Help, 2013; <http://onlinehelp.microsoft.com/en-us/bing/ff808447.aspx>
14. Microsoft. Microsoft.com Privacy Statement, 2013; <http://www.microsoft.com/privacystatement/en-us/core/default.aspx>
15. Mozilla. anonymoX: Add-ons for Firefox, 2014; <https://addons.mozilla.org/en-US/firefox/addon/anonymox/>
16. Nakashima, E. and Markon, J. Dozens of attacks foiled, NSA says. *Washington Post* (June 13, 2013), A1.
17. Phelps, J., Gonzenbach, W., and Johnson, E. Press coverage and public perception of direct marketing and consumer privacy. *Journal of Direct Marketing* 8, 2 (Spring 1994), 9–22.
18. Preibusch, S., Kübler, D., and Beresford, A.R. Price versus privacy: An experiment into the competitive advantage of collecting less personal information. *Electronic Commerce Research* 13, 4 (Nov. 2013), 423–455.
19. Protalinski, E. Microsoft confirms Google privacy campaign to promote Bing is aimed at Apple Safari users. *The Next Web* (Sept. 20, 2012); <http://thenextweb.com/microsoft/2012/09/20/microsoft-confirms-google-privacy-campaign-aimed-apple-safari-users/>
20. Rosenblatt, S. Escaping Google’s gravity: How small search engines define success. *CNET* (July 11, 2013); <http://cnet.co/15kOLCY>
21. Roznowski, J.L. A content analysis of mass media stories surrounding the consumer privacy issue, 1990–2001. *Journal of Interactive Marketing* 17, 2 (Apr. 2003), 52–69.
22. Stone, B. Facebook radically revamps its search engine. *Bloomberg Businessweek* (Jan. 15, 2015); <http://www.businessweek.com/articles/2013-01-15/facebook-radically-revamps-its-search-engine>
23. The Tor Project, Inc. *Tor Metrics Portal: Users 2013*; <https://metrics.torproject.org/users.html?graph=direct-users&country=us#direct-users>
24. Van Grove, J. Zuckerberg: Thanks NSA, now people trust Facebook even less. *CNET* (Sept. 18, 2013); http://news.cnet.com/8301-1023_3-57603561-93/zuckerberg-thanks-nsa-now-people-trust-facebook-even-less/
25. Wikipedia article traffic statistics, 2013; <http://stats.grok.se/>

Sören Preibusch (<http://preibusch.de/>) is a user experience researcher at Google, Mountain View, CA, and was at Microsoft Research, Cambridge, U.K., when this article was written.

Copyright held by author.

DOI:10.1145/2663339

The benefits in school and the job market so far outweigh any potential gender bias that few women are deterred.

BY ROLI VARMA AND DEEPAK KAPUR

Decoding Femininity in Computer Science in India

IN THE U.S., the share of bachelor's degrees awarded to women has declined by 10% since 2000; in 2011, women earned only 18% of bachelor's degrees awarded in CS.⁴ It is thus no surprise that much research has focused on the underrepresentation of women in CS education, often portraying CS as a man's field.^{1,2,5} However, this characterization is society-specific, not universal. Unlike in the U.S., women's participation in CS education in India has increased in the past 15 years in most nationally accredited institutes and universities;⁶⁻⁸ for instance, women constituted 42% of undergraduate students in CS and computer engineering in 2011 in India.³ They were and still are not the odd ones out, as the masculine perspective

might hold. Rather, they enroll in CS because men and women alike see CS as a woman-friendly field.

To understand why women in India are attracted to CS education, we carried out a qualitative study in 2007-2008, conducting in-depth interviews with 60 female undergraduates majoring in CS at two technical institutes and two universities granting four-year undergraduate degrees in CS. One campus is the top national technical institute, the other a well-known regional technical institute. To ensure minorities in India were included in the study, we included a third university that is historically Muslim and a fourth university that is predominantly Sikh. (Due to the University of New Mexico Institutional Review Board requirement that granted us permission to conduct interviews, we cannot disclose the names of the institutions.) We used random sampling to select 15 subjects at each. We recorded interviews that were transcribed and processed through the NVivo software package from QSR International for data analysis. Two independent coders coded the same data to ensure reliability. Here, we present key findings, along with frequency of response.

All students we interviewed were young unmarried women age 19 to 22 in their second-to-fourth year of CS studies. Other than being full-time students, none held a job at the time of the interview. A large majority of them characterized their family background as middle class or upper middle class. Almost 75% were born to Hindu families, with the majority from middle and

» key insights

- **CS is viewed as a liberating major and profession for and by women in India.**
- **Our study demonstrates how women's experience in CS differs according to sociocultural, economic, and historical context.**
- **It also outlines challenges women in India face to do well in CS education and seeking related employment.**



high castes; remaining students were born to Sikh and Muslim families. They had attended private or central government schools that used English as a main medium of instruction.

Early Exposure to Computers

We asked whether they had a computer at home when growing up, and only a few (8%) reported they did. However, personal access to a computer was available to a little over half of them (53%) in their adolescent years. Parents typically brought home a com-

puter for work or for an older sibling. If not available at home, the students had access to computers in cyber cafés. A significant percentage (39%) had no personal access to a computer until enrolling in the CS degree program at a university or institute. Of those with personal access to a computer, most used it for non-technical purposes—to watch movies, listen to music, play games, browse the Web, or chat with friends or relatives.

We asked them to describe the computer resources available to them

in their high schools (ninth and 10th grades) and plus-two schools (11th and 12th grades). Whereas one-third (32%) reported having no computer resources in school, the rest (68%) said their schools had some type of computer facilities but with limited access. They also complained about high student-to-computer ratios, lack of useful software, limited access to computer labs, slow Internet access, and frequent electric power outages. Some reported learning basic computer skills and Microsoft Word in school.

Enrollment in CS

As to how they became interested in pursuing a CS degree, they explained that as bright students, they were expected to choose between engineering and medical tracks after high school so picked the former. To be admitted to institutions of higher education in India, they had to sit for entrance and/or board exams; they chose their field of study based on ranks and/or marks they received in these exams. For over one-third (35%), a major motivation toward CS was family, directly or indirectly. Some (27%) said a CS degree offered promising career prospects, while others (18%) viewed computers as new social tools to help people. A few (15%) said they had developed their interest in computers as adolescents, without further specificity. The rest (5%) did not fit a category.

We also questioned them about the people who influenced their decision to study CS. A majority (62%) credited family members, not limited to, but mainly males, including fathers, brothers, cousins, and uncles. These men valued education immensely and saw CS as a proper field of study for women. The rest (38%) said either no one influenced them to opt for a CS major or were simply self-driven to pursue it.

Educational background. We asked whether their schools prepared them for undergraduate studies in CS. Over half (55%) said they were “not well prepared” because they did not have strong training in programming or CS in general. Over one-third (35%) felt “partially prepared” because they had taken basic computer classes. A few (10%) believed they were “fully prepared” because they were taught basic programming. Students in all three categories said their mathematics education was strong and thus helpful in engaging in CS at the undergraduate level. This was consistent with a majority of the students listing mathematics (77%) as their best subject in school, followed by science (20%) and computers (3%). Most said social studies and languages were their worst subjects.

Undergraduate CS Program

When discussing their CS courses, a majority (60%) gave a mixed or nega-



Another set of students (34%) emphasized the good work environment associated with CS jobs, viewing them as white-collar positions at a desk in a secure indoor office, possibly with air conditioning and coffee.



tive review; the rest (40%) responded positively. They said basic material was often omitted from course content, as it was assumed students had a basic level of proficiency in CS. This left many needing extra time with self-instruction to keep up. Overall, they said their courses were difficult but more rewarding than other courses in their institutions. They confronted this difficulty eagerly and embraced challenges. However, they did not enjoy the extra time required for class preparation and self-instruction; many also felt the material taught in the classroom was insufficient for homework and tests.

Teachers. The students reported favorably (51%), mixed (34%), and negatively (15%) when asked about their CS teachers. Overall, they said those teachers were knowledgeable, supportive, interactive, and motivating. Even demanding teachers with high expectations received praise or at least approval, as they were viewed as able to motivate their students to work hard outside of class. The students expressed aversion to teachers who came to class only to lecture and not care whether students understood the material. Other than the national technical institute, students from the other institutions expressed annoyance that teachers would often be replaced midway through a semester or that the teachers at their institution viewed the institution only as a stepping-stone to a better job elsewhere. Interestingly, none of the students mentioned the gender of the CS teachers, almost all of whom were men.

Advisors. Only students from the national technical institute reported academic advisors were part of their CS program. Some turned to their advisors for help choosing classes and schedules, though many found it helpful to discuss course choices with their senior and fellow students. Most students from other institutions reported relying on advice from peers and seniors. Some students at each institution claimed to be relatively self-reliant, choosing their courses independently.

Peers. A majority (63%) reported positively about their male and female peers, viewing them as friendly, helpful, intelligent, or even brilliant. Students discussed competition in a

positive light, explaining that the level of competition at their institutions was healthy and motivating. However, many (37%) gave somewhat mixed reviews of their peers, saying not all were good; some viewed themselves as technically superior to women and thus did not take women seriously when working on group projects. A few commented that the men were jealous of the women's presence in CS. Nonetheless, the students generally seemed to enjoy the presence of their classmates, getting along with them socially, as well as academically.

Computer resources. The students gave positive-to-mixed assessments of the computer resources available to them at their institutions, reporting facilities featuring specialties in Internet use and programming languages. Specific complaints were location-specific, centering on Internet speed and not enough computers. Such complaints typically came from the two universities.

CS culture. We asked the students to describe the typical perception of CS people at their institutions. An overwhelming percentage (90%) referred to CS people in a positive light, including highly regarded, respected, intelligent, smart, or extremely good within other technical disciplines. Some said the CS people love the minute details needed to make computer programs work. Only six students from the technical institutes used the term “geek” to describe a CS student. Interestingly, geeks were also perceived differently from the stereotypical U.S. view, citing creative or inventive and working hard at their computers to develop new and exciting programs and applications. There was general agreement among the students that CS people are social and enjoyable to be around.

Gender and CS

As to whether careers with a CS degree are attractive to women, all students we interviewed agreed enthusiastically. Over half (54%) identified high-paying job opportunities with a CS degree, as employment can be found in multiple sectors, fields, and locations, including in global companies. Due to the tremendous ongoing growth in the IT sector, they said they would be able to take a job soon after graduation and

that IT jobs paid higher salaries compared to other fields. Another set of students (34%) emphasized the good work environment associated with CS jobs, viewing them as white-collar positions at a desk in a secure indoor office, possibly with air conditioning and coffee. They contrasted this with the work environment at a construction site or factory. They compared CS jobs with regular daily hours against the arbitrary hours and locations associated with medicine, a popular field considered by some in India as more suitable for women. Few (12%) cited social and psychological benefits, including independence by evading marriage plans set by parents, self-confidence due to regularly exercising their minds with critical analysis, and higher social status from the ability to perform technical work.

Experience. Questioned about how female students are perceived in the CS major, a large majority (88%) responded positively, expressing feelings of achievement and privilege from studying CS. They talked about the big compliments they got from family and friends for studying such a hot, exciting, challenging major like CS at a reputable institution. A few went so far as to say if they were in commerce, social studies, or medicine, they would not be treated with such respect. A small group (12%) expressed concern only because their male peers had better prior exposure to computers and continued access to support structures, thus making the CS program somewhat more challenging for female students.

Considering the U.S. perspective, where women are underrepresented in CS, we asked why so few women study CS at their institution. Some students (32%) said it was not true, seeing near-parity at their institutions or closing gender imbalance in CS. A majority (68%) depicted a gender gap that developed from a mixture of social biases and structural obstacles. Students from universities said many Indians typically imagine engineering as a male-oriented field and medicine, commerce, and social studies as feminine fields. They further blamed Indian society, which places women's first priority as marriage and raising a family, instead of pursuing an indepen-

dent career. Students from the technical institutes generally said women lacked financial resources for good coaching to do well on the entrance exams that would enable them to opt for a CS major.

Obstacles. Asked to describe the obstacles women face completing their degree, almost half (47%) said structural obstacles. Due to time curfews imposed on girls at their hostels (typically 8 P.M.), they could not stay in the labs longer and thus had more difficulty completing group projects and networking with peers. Those who lived at home had even more restrictive time curfews. During placement interviews, employers also questioned whether women had their parents' permission to work, as well as their marriage plans. Many students (36%) mentioned societal gender perceptions that identified engineering as a male-oriented subject, that women were not supposed to study beyond 12th grade, and that women's role was as wives and raising children at home. A few students (17%) explained that women are often forced to compromise within their families, setting their desires aside to go along with parental decisions and family harmony.

Incidents. To report specific incidents faced exclusively by women in a CS major at their institutions, a small minority (17%) had nothing to say. However, an overwhelming percentage (83%) reported a range of bias. One-quarter (26%) mentioned activities reflecting biased treatment from teachers, namely unequal expectations for male and female students and singling out female students for questioning or reprimand. One-fifth of the students (21%) described “gendered” restrictions, from curfews in the hostels where they lived to being excluded from male hostels, thus hindering spontaneous networking and group projects. Travel for possible internship, additional training, and job interviews were also a problem for some, as it is not socially acceptable for a woman to travel alone in India. Another set of bias cited by some students (18%) was specific to home and family, as they received less financial support and emotional encouragement than their male siblings for

higher education. A few (15%) reported teasing, both on and off campus.

Confidence. Asked to compare themselves academically with their male colleagues, they were almost equally divided between those (40%) who ranked themselves equal with their male peers and those (37%) who considered their male classmates better than them. Almost one-fourth (23%) said they were, in fact, superior to their male peers. Those who thought they were either better or at the same level were typically enrolled at technical institutes. There was general agreement that gender was not the deciding factor in determining who does well in CS courses. Rather, they said there are more opportunities available to men than women, causing some men to perform better; according to some of the students, if women had the same opportunities, they would do even better than men. Students from the predominantly Sikh university said women regularly received top grades. Lack of access to extra outside coaching, inability to stay late in labs and travel more freely, and limited computer exposure were some of the factors students cited as resulting in gender imbalance in CS.

Attraction for men. Asked what attracts men to CS, a large majority of the students (63%) said men were motivated to enter CS came from the pull factors characterizing the field. A CS graduate is able to get a well-paying job, which is a major concern for many men, especially considering they are viewed as family providers. Also, many job opportunities are available both inside and outside India. The rest (37%) identified such push factors as parents providing financial support for men to be in a potentially well-paid technical field. No one mentioned the geek CS culture as a compelling attraction for male students.

Graduation and Future Prospects

Asked whether they had ever considered changing their major from CS to something else, an overwhelming percentage (90%) replied in the negative, showing considerable confidence in their decision to have selected CS as a major. It did not cross their minds to switch to another major due to the advantages, prospects,

or freedom they might gain from earning a CS degree. A small minority (10%) who considered other options did so only in the first year because they had no prior CS experience, thus making their coursework rather more difficult. However, by the second year these students were able to catch up and regain the confidence needed to complete a CS degree.

Asked whether they knew students who had switched majors from CS to another discipline, an overwhelming majority (81%) said they did not know others who had changed their major. They said the goal for many students was to be accepted into the CS program, so switching was held in low regard. Only some students (19%) from the two universities knew or had heard about someone who had dropped out to get married, join the Indian Air Force, or switch majors to electronics.

After graduation. After completing their CS degree, a majority of the students (65%) planned to take a job, mainly in the IT sector. Entering directly into the work force was appealing due to the opportunity so soon after graduation. Some said the additional flow of money from their jobs would ease their families' concern about the cost of their marriage. However, a few (13%) wanted to move directly into an MBA program or go on to graduate studies in CS. They viewed an MBA as providing them a broader range of employment options at an even higher pay scale than with a CS degree alone. The rest (22%) were undecided between pursuing a job or going further into higher education. They planned to make the final decision after evaluating their job placement options and possible admission to university graduate programs.

Asked to describe the most encouraging and most discouraging experience in their programs, they cited the success of their seniors and what their friends were doing after graduation. Most were inspired when students were able to find employment right after graduation from global companies. Some said they were motivated hearing about friends receiving summer internships at well-regarded companies. A few mentioned a peer's admission into a graduate program to study CS, as the popular trend was

to go for an MBA. On the other hand, many were discouraged by knowing someone unable to find a job after graduation. Some became dispirited over an overbearing workload. One-third of the interviewed students said nothing discouraged them or that since the CS program was the best, complaining was useless.

Marriage. With regard to the interviewed students' plans for marriage, a majority (63%) reported they planned to get married at some point. However, most had no desire to give up the hard-won independence they could expect with a CS job when married. A few also wanted to keep working after marriage but said the decision would have to be made jointly with their future husband and in-laws. Interestingly, a solid percentage (37%) said they were not contemplating marriage either at the time or in the future, wanting instead to work and travel, possibly abroad.


Discussion: CS as Women Friendly

Since India's independence in 1947, Indian women, especially in urban areas, have gone to school and been educated in increasing numbers. However, the main purpose of them getting educated was typically not to prepare for a job, unless their families had a pressing financial need. Being educated was and still is considered a societal benefit, further making women "good" prospective brides and mothers. Indian women typically prefer career opportunities in government, teaching, and medicine. Engineering was and is still viewed as suited mostly for men, as is evident from the low enrollment and graduation figures of women in engineering programs in India. With the economic liberalization and reformist policies implemented by the Indian central government in 1991 and subsequent growth of the Indian IT sector, female students in India have come to view an IT career as appealing. This has led to increased enrollment of women in CS majors in Indian universities and technical institutes. As our study demonstrates, a CS major is perceived as women friendly, not only by female students but by their families in traditional Indian society. Students majoring in CS in general, and especially women, are viewed


as intelligent by family and friends, boosting their confidence to enroll in a CS major and stay with it until graduation. Majoring in CS is viewed as leading to well-paid employment opportunities in a range of companies in India, as well as abroad. Such employment typically comes with a high pay scale compared to other jobs. Moreover, they would be doing mental labor with leading-edge technology in a secure office as opposed to manual labor in factories and construction sites. With secure high-paying jobs, women are likely to have some social independence from wives and mothers in traditional roles. In this sense, female students view CS as a liberating major and profession, further questioning the assumption that CS is universally a masculine discipline.

Interesting complexities also arise due to the perceived role of women in a patriarchal society. Although a CS degree offers female students the ability to claim higher social status and become economically independent in the future, they still need their parents' approval before marriage, and husbands' and in-laws' after marriage, for pursuing internships, taking a job, and having a successful career. Even employers ask prospective female graduates whether they have permission from their parents to work, along with their marriage plans. Parents seem to be okay with their daughters pursuing a CS degree, as it goes well with their perception of the type of work their daughters would do, that is, mental work performed indoors at a desk, with minimal interaction with males.

The Indian female students majoring in CS we interviewed said problems in recruitment and retention of women in CS education is not due to the nature of the field but rather to the Indian patriarchal society and its value system favoring men. They did not characterize CS as a field for male geeks, which is demonstrably the case in the U.S., but for both men and women seeking high-paying jobs and aspiring to be competent computer and technology users. The word geek was rarely mentioned in more than 100 hours of in-depth interviews. Though fascinated by computer technology and consumed by



For the Indian female students we interviewed, benefits of a CS career—high social status and independence due to a well-paid job—outweigh any bias from teachers and gender restrictions, even at institutions of higher education.



figuring out its inner workings, the CS major is not seen as obsessive to the extent of being anti-social, a demonstrably widespread perception in the U.S. Both men and women seek to be computer savvy to get into a well-paid technical field by virtue of being smart and hard working. Even with the gendered treatment of female students and related social obstacles, those we interviewed never seemed to question whether a CS major is appropriate for them; on the contrary, they feel their standing in their families and in society as a whole is elevated since only smart students at the top of their class are able to major in CS. For those we interviewed, benefits of a CS career—high social status and independence due to a well-paid job—outweigh any bias from teachers and gender restrictions, even at institutions of higher education.

These findings are in sharp contrast with the reasons often given for low U.S. enrollment and graduation figures of women in CS degree programs. Most important, the U.S. image of CS as a man's field, even though many people in it view themselves differently; it has been suggested that this perception of CS leads to excluding women from CS.^{1,2,5} However, our study shows Indian women have a very different story to tell, viewing CS as suited for all, especially women.

In the U.S., prior experience with computers is considered an important variable for generating interest in CS. However, few female students in India are regularly exposed to computers before enrolling in their institutions of higher education. They also found CS courses difficult initially, much like in the U.S. The reasons were different, however, due mainly to teachers not being considerate of lack of an adequate background of many Indian female students. Further, they could not work freely with male students on group projects since they had to return to hostels and homes to meet curfews. Despite such difficulties, these women have a strong desire to get a good job, so they continue to enroll and persist in CS because they view themselves as strong in mathematics and can learn on their own. Such perceptions give them confidence to pursue a CS major.

Another key issue helping explain CS being less attractive for female students in the U.S. is fewer female teachers in science and mathematics classes in schools, as well as a lack of female role models. These weaknesses are generally seen as reinforcing the attitude that these fields are not for women. Low numbers of female faculty in CS in U.S. universities is viewed as making many female students feel isolated.^{1,2,5} Our study shows that for Indian female students, these were hardly issues, especially in contrast to other social factors. Indian female students rarely had female mathematics and science teachers in high school, and the number of female faculty in CS-related departments in Indian universities and institutions is even smaller than in the U.S. This is not to suggest that more female teachers in high schools teaching science and mathematics, as well as more women faculty in CS, would not help increase female enrollment. Rather, the Indian female students we interviewed complained of a lack of resources and opportunities they felt they must overcome to do well in CS in India.

These students belonged to India's middle and upper castes. While this might have given them an advantage over their peers in lower castes, male students in the same universities and institutes also tend to come from middle and upper castes. It would be difficult to argue that female students did thus not feel inferior to male students in their ability and intelligence, given that India is a far more male-dominated society than is the U.S.; they also did not feel superior to male students. Since caste is still an overriding factor in Indian society, a study of perceived intelligence based on caste versus gender is an important research direction worth pursuing.


Conclusion

Our study shows that socioeconomic context must be taken into consideration to understand how gender interacts with CS education in India. Lack of interest by women in CS should not be viewed as a global phenomenon; women may not show interest in majoring in CS in the U.S. and perhaps other Western countries but definitely

Since caste is still an overriding factor in Indian society, a study of perceived intelligence based on caste versus gender is an important research direction worth pursuing.

go for it in India. Among all engineering-related disciplines, CS is viewed as the most attractive to Indian female students. Further, CS is viewed as a major pursued by intelligent students, helping boost their confidence, especially among women; prospects of a high-paying job leading to independence from family and parents motivate female students in CS to do well and complete a degree, an academic pursuit where strong mathematical skills are helpful. This fact contradicts many conventional assumptions, including that CS is a man's discipline and CS reliance on mathematical skills might be a hindrance to attracting female students.

Acknowledgments

This work was supported by the National Science Foundation under Grant 0650410. We thank Ila Kapur Varma for suggesting the title. 

References

1. Ahuja, M.K. Women in the information technology profession: A literature review synthesis and research agenda. *European Journal of Information Systems* 11, 1 (Mar. 2002), 20–34.
2. Cohoon, J.M. and Aspray, W., Eds. *Women and Information Technology: Research on Underrepresentation*. MIT Press, Cambridge, MA, 2006.
3. Government of India. *All India Survey on Higher Education*. New Delhi, 2012–2013; <http://india.gov.in/all-india-survey-higher-education-ministry-human-resource-development>
4. National Science Board. *Science and Engineering Indicators*. National Science Foundation, Arlington, VA, 2014; <http://www.nsf.gov/statistics/seind14/>
5. Singh, K., Allen, K.R., Scheckler, R., and Darlington, L. Women in computer-related majors: A critical synthesis of research and theory from 1994 to 2005. *Review of Educational Research* 77, 4 (Dec. 2007), 500–533.
6. Varma, R. Exposure, training and environment: Women's participation in computing education in the United States and India. *Journal of Women and Minority in Science and Engineering* 15, 3 (Sept. 2009), 205–222.
7. Varma, R. Computing self-efficacy in India. *Journal of Women and Minorities in Science and Engineering* 16, 3 (Sept. 2010), 257–274.
8. Varma, R. Indian women and mathematics for computer science. *IEEE Technology and Society Magazine* 30, 1 (Spring 2011), 39–46.

Roli Varma (varma@unm.edu) is the Carl Hatch Endowed Professor and Regents' Lecturer in the School of Public Administration at the University of New Mexico in Albuquerque and served on the ACM Job Migration Task Force (<http://www.acm.org/globalizationreport/>).

Deepak Kapur (kapur@cs.unm.edu) is a Distinguished Professor in the department of computer science at the University of New Mexico in Albuquerque and a recipient of the Conference on Automated Deduction Herbrand Award.

acm Inroads

PAVING THE WAY TOWARD EXCELLENCE IN COMPUTING EDUCATION



Association for
Computing Machinery

ACM/SIGCSE Seek New Editor for ACM Inroads

ACM and the Special Interest Group on Computer Science Education (SIGCSE) seek a volunteer editor-in-chief for its quarterly magazine *ACM Inroads*.

ACM Inroads serves professionals interested in advancing computing education on a global scale. The magazine—written by computing educators for computing educators—presents an array of thought-provoking commentaries from luminaries in the field together with a diverse collection of articles that examine current research and practices within the computing community. For more about *ACM Inroads*, see <http://inroads.acm.org/>

Job Description

The editor-in-chief is responsible for soliciting all editorial content for every issue. These responsibilities include: soliciting articles from prospective authors; managing the magazine's editorial board and contributors; creating new editorial features, special sections, columns and much more. For more information, see <http://inroads.acm.org/eic-search.cfm>

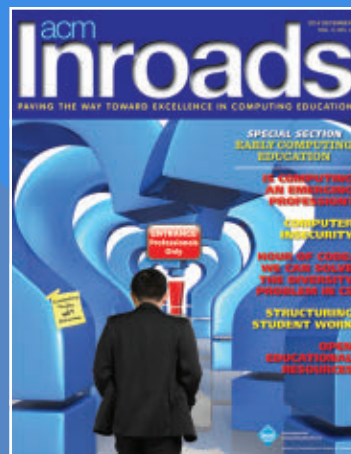
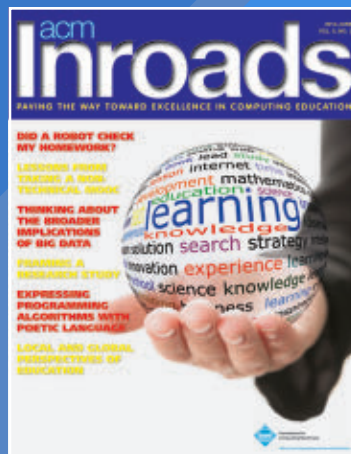
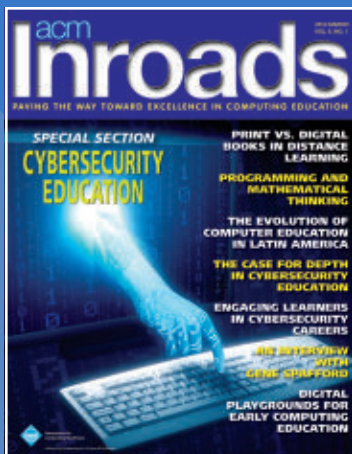
Eligibility Requirements

The EiC search is open to applicants worldwide. Experience in and knowledge about the issues, challenges, and advances in computing education a must.

This editorship commences on September 1, 2015. You must be willing and able to make a three-year commitment to this post.

Please send your CV and vision statement expressing the reasons for your interest in the position and your goals for *Inroads* to: eicsearch@inroads.acm.org

The ACM Publications Board will review all candidates.



Robots move to act. While actions operate in a physical space, motions begin in a motor control space. So how do robots express actions in terms of motions?

BY JEAN-PAUL LAUMOND, NICOLAS MANSARD,
AND JEAN BERNARD LASSERRE

Optimization as Motion Selection Principle in Robot Action

MOVEMENT IS A fundamental characteristic of living systems (see Figure 1). Plants and animals must move to survive. Animals are distinguished from plants in that they have to explore the world to feed. The carnivorous plant remains at a fixed position to catch the imprudent insect. Plants must make use of self-centered motions. At the same time the cheetah goes out looking for food.

Feeding is a paragon of action. Any action in the physical world requires self-centered movements, exploration movements, or a combination of both. By analogy, a manipulator robot makes use of self-centered motions, a mobile robot moves to explore the world, and a humanoid robot combines both types of motions.

Actions take place in the physical space. Motions originate in the motor control space. Robots—as any living system—access the physical space only indirectly through sensors and motors. Robot motion planning and control explore the relationship between physical, sensory, and motor spaces; the three spaces that are the foundations of geometry.³² How to translate actions expressed in the physical space into a motion expressed in motor coordinates? This is the fundamental robotics issue of inversion.

In life sciences, it is recognized that optimality principles in sensorimotor control explain quite well empirical observations, or at least better than other principles.⁴⁰ The idea of expressing robot actions as motions to be optimized was first developed in robotics in the 1970s with the seminal work by Whitney.⁴¹ It is now well developed in classical robot control,³⁷ and also along new paradigms jointly developed in multidisciplinary approaches.³⁵ Motion optimization appears to be a natural principle for action selection. However, as we explained in the companion article,²⁴ optimality equations are intractable most of the time and numerical optimization is notoriously slow in practice. The article aims

» key insights

- **For robots and living beings, the link between actions expressed in the physical space and motions originated in the motor space, turns to geometry in general and, in particular, to linear algebra. In life science the application of optimality principles in sensorimotor control unravels empirical observations. The idea to express robot actions as motions to be optimized has been developed in robotics since the 1970s.**
- **Among all possible motions performing a given action, optimization algorithms tend to choose the best motion according to a given performance criterion. More than that, they also allow the realization of secondary actions.**
- **Optimal motions are action signatures. How to reveal what optimality criterion underlies a given action? The question opens challenging issues to inverse optimal control.**



Figure 1. Stones and hammers do not move by themselves.

Movement is a prerogative of living (and robot) systems. Plants (and manipulator robots) move to bring the world to them via self-centered movements. Animals (and mobile robots) navigate to explore the world. Human (and humanoid) actions are built from both types of movement.



to provide a short overview of recent progress in the area. We first show how robot motion optimization techniques should be viewed as motion selection principles for redundant robots. In that perspective, we review results and challenges stimulated by recent applications to humanoid robotics. The remainder of the article is devoted to inverse optimal control as a means to better understand natural phenomena and to translate them into engineering. The question opens highly challenging problems. In that context, methods based on recent polynomial optimization techniques appear complementary to classical machine learning approaches.

Power and Limits of Linearization

Translating actions in terms of motions expressed in the robot control space has been expressed in many ways, from the operational space formulation¹⁹ to the task function approach,³⁴ to cite a few. The notion of task encompasses the notion of action expressed in the physical space. The task space may be the physical space (like for putting a manipulator end effector to some position defined in a world frame) or a sensory space (like tracking an object in a robot camera frame). The role of the so-called task function is to make the link between the task space and the control space.

Due to the underlying highly non-linear transformations, the inversion problem is very costly to solve (minutes or hours of computation for seconds of motion). To meet the time constraints

imposed by the control frequency of the robots, the problem is addressed only locally by considering the tangent spaces of both the task space and the configuration space. Such a linearization involves the Jacobian matrix³¹ and resorts to all the machinery of linear algebra. The linearization is particularly interesting as the tangent space of the configuration space gathers the configuration velocities that usually contain the robot control inputs. Dynamic extensions of this principle allow considering torque-based controls.¹⁹

The Jacobian matrix varies with the robot configuration, making the search for a trajectory nonlinear. However, for a given configuration, it defines a linear problem linking the unknown system velocity to the velocity in the task space given as references. From a numerical point of view, this problem is linear and can easily be solved at each instant to obtain the system velocity. The integration of this velocity from the initial configuration over a time interval draws a trajectory tending to fulfill the task. The velocity can similarly be applied in real time by the robot to control it toward the goal. The linear problem is re-initialized at each new configuration updated with the sensor measurements and the process is iterated. This iterative principle corresponds to the iterative descent algorithms (like the gradient descent or the Newton-Raphson descent), which are used to numerically compute the zero value of a given function. However, the method gives more: the sequence of descent itera-

tions, assuming small descent steps, is a discretization of the real trajectory from the initial configuration to the goal. The drawback of the instantaneous linearization is it provides no look-ahead capabilities to the control, which might lead the robot to a local minimum, typically when approaching non-convex obstacles. This is the well-known curse of linearization.

Motion Selection

The dimension of the task space can be equal, greater, or lower than the dimension of the control space. For the sake of simplification, let us consider the task space as a manifold that expresses the position of the end effector of a fully actuated manipulator. When the dimensions of both the task space and the configuration space are equal, each point in the task space defines a single configuration^a and the task function can be used to drive the robot to a unique configuration. There is no problem of motion selection. The Jacobian matrix is square invertible and solving the linear problem is easy. The task function approach was initially proposed in this context to define admissibility properties to connect two points of the configuration space while avoiding singularities.³⁴

Optimization is used as motion selection principle in the other cases. The choice of the optimization criterion determines the way to invert the Jacobian matrix, as we will explain.


When the task space has a larger dimension than the configuration space, it is not always possible to find a configuration satisfying the task target: the task function is not onto, that is, the Jacobian matrix has more rows than columns. It is then not possible to find a velocity in the configuration tangent space that corresponds to the velocity in the task tangent space. For instance, this is the case in visual servoing when many points should be tracked in a camera frame.⁴ This is also the case in simultaneous localization and mapping when optimizing the positions of the camera with respect

^a The non-linearities in the task function can generate a discrete set of configurations accomplishing the task, corresponding to several options. In such cases, the discussion holds, but only locally.


to the landmarks.¹³ Optimization is used to find a velocity that minimizes the error in the task tangent space. The problem is then to minimize the distance to the reference task vector. Generally, the reference task vector cannot be reached. In the special case, when the reference belongs to the image space of the Jacobian, the residual of the optimization is null. This is the case in visual servoing, when the target image has been acquired from a real scene with no noise.

On the contrary, if the dimension of the task space is smaller than the dimension of the configuration space, several configurations correspond to a single task. The task function is not one-to-one; that is, the Jacobian matrix has more columns than rows. For instance, in the case of a 30-joint humanoid robot picking a ball with its hand: the dimension of the task space is three while the dimension of the configuration space is 30. Several motions may fulfill the task. The system is said to be redundant with respect to the task. Optimization is then used as a criterion to select one motion among all the admissible ones. In that case, several vectors in the configuration tangent space produce the same effect in the task space. Equivalently, some velocities produce no effect in the task space. This subset of the configuration tangent space is the kernel of the Jacobian matrix and is called the null space of the task. Any velocity in the null space leaves the task unchanged. Adding up a given configuration tangent vector satisfying the task with the null space gives the vector space of all velocities satisfying the task. The minimization problem consists in selecting one sample in this space, according to some criteria, for example, the least-norm velocity.

In general, the task function may neither be onto nor one-to-one; that is, the Jacobian matrix is neither full row rank (its rows are not linearly independent) nor full column rank (that is, its columns are not linearly independent). In general, no vector in the configuration tangent space satisfies the task (since the transformation is not onto), and there is infinity of vectors that minimize the distance to the task vector in the task tangent space



It is possible to recognize actions from motion observation using reverse engineering techniques.



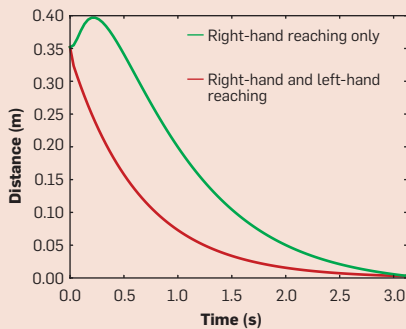
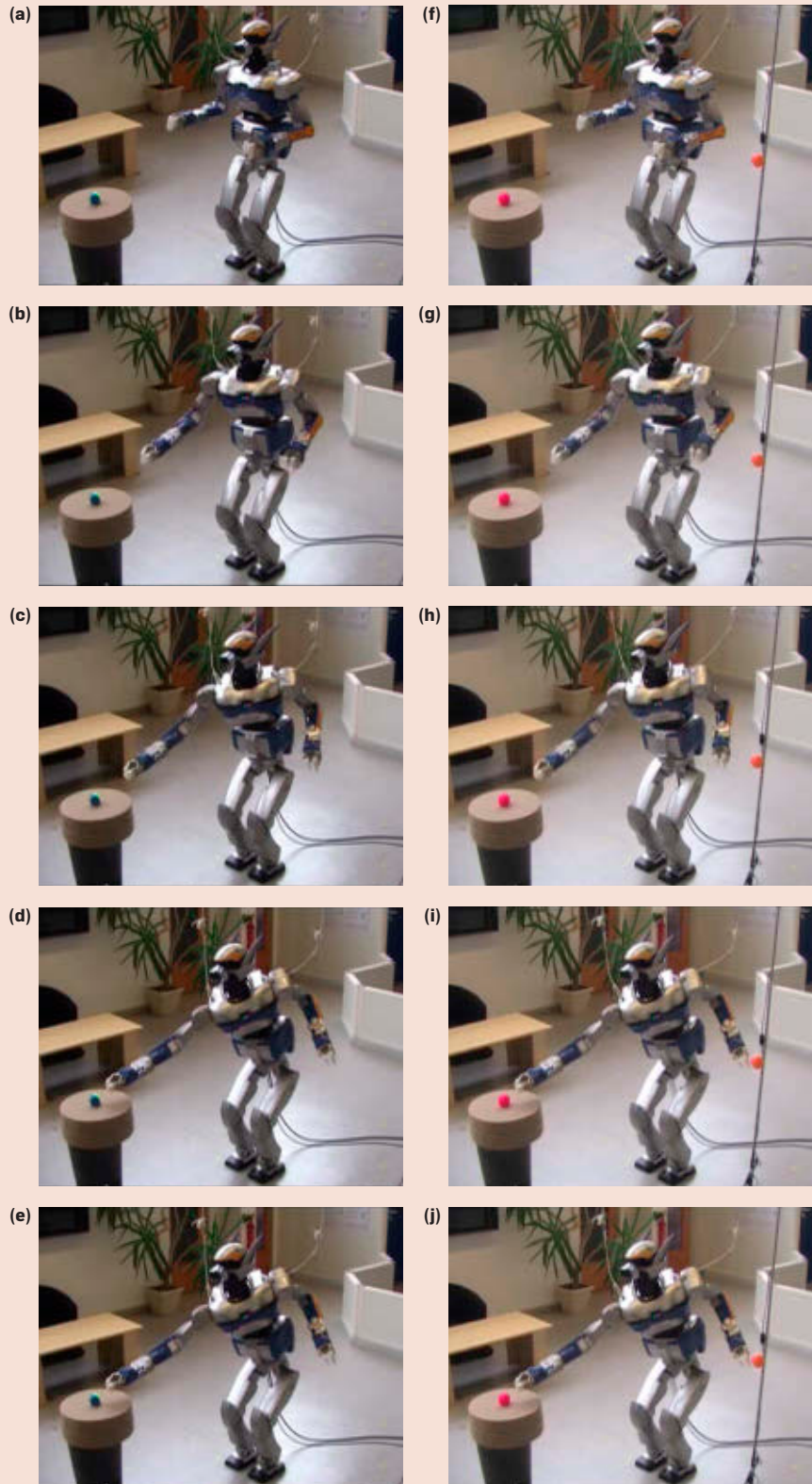
(since the transformation is not one-to-one). Therefore, the selection problem becomes a double minimization problem: we simultaneously minimize the distance to the task and the norm of the configuration velocity. A solution for this double minimization problem is given by the Moore-Penrose pseudo-inverse,² also called the least-square inverse. Notice that other minimization criteria may be considered in the same framework by changing the metrics in the tangent spaces. For instance, we can use weighted pseudo-inverses in which the components of the system input (columns of the Jacobian matrix) and the task residual (rows of the Jacobian) do not receive the same weight. As before the sum of the optimal vector with the null space gives the set of all solutions that are optimal for the first problem (smallest distance to the task) but only suboptimal for the second one (smallest velocity norm).

Optimization as Selection Principle

Stack of tasks for redundant systems. When a robot is redundant with respect to a task, it is interesting to allocate it a secondary task. This is the case for humanoid robots that can perform two tasks simultaneously. Consider two distinct task functions dealing with the positions of the right and left hands respectively. How to check if both tasks are compatible? A simple idea consists of ordering the two tasks. At each time step of the integration process, a vector of the configuration tangent space associated to the first task is selected. Then the secondary task is considered only within the restricted velocity set lying in the kernel of the first task. The reasoning that applies to the first task also applies to the projected secondary task: the task function may be onto, one-to-one, or neither of it. In particular, if it is not onto, the task is said to be singular (in the sense the Jacobian is rank deficient). Two cases can be distinguished. If the (not projected) secondary task function is not onto, then the singularity is said to be kinematic: it is intrinsically due to the secondary task. On the opposite, if the (not projected) secondary task function is onto, then the singularity is said to be algorithmic: because of a conflict with

Figure 2. Examples of motions generated by the stack of task.

Left column: The stack of tasks is composed of three constraints (both feet and center of mass should remain at a fixed position; all of them are always feasible and satisfied) and two tasks to control the gaze and the right hand. At the final configuration (e), the robot has reached the ball with its right hand and the ball is centered in the robot field of view. The left hand had moved only to regulate the position of the center of mass. Right column: The motion is similar but a task has been added to control the position of the left hand: the desired position imposed to the left hand is the final position reached by the left hand in the previous movement. The two motions look very similar, but their "meanings" are different. In the right motion (a–e), the left hand moves to regulate the balance; in the right motion (f–j), the left hand moves to reach a specific position.



The motions of the left hand look similar but they are not exactly the same. The curves show the distance function for the left hand going from its initial position to its final position in both cases. The curves are different. The exponential decreasing of the distance function (red curve) "signs" the presence of the left-hand reaching task.

Right-hand reaching while enforcing the balance: the left hand has moved only to correct the balance.

Simultaneous right-hand and left-hand reaching: the target imposed to the left hand is the final position reached in the previous (right-hand only) movement.

the main task, the secondary one becomes singular.⁵ Outside of algorithmic singularities, the two tasks are said to be compatible and the order between them is irrelevant.

The projection process can be iterated for other tasks, resulting in a so-called stack of tasks.²⁶ In doing so, the dimension of the successive null spaces decreases. The process stops either when all tasks have been processed, or as soon as the dimension of the null-space vanishes (see Figure 3). In the latter, we can still select the minimum-norm vector among those in the remaining null space.

The null space was first used in the frame of numerical analysis for guiding the descent of sequential optimization.³³ It was used in robotics to perform a positioning task with a redundant robot while taking care of the joint limits.²⁵ A generalization to any number of tasks was proposed in Nakamura,³⁰ and its recursive expression was proposed in Siciliano³⁸ (see also Baerlocher¹ in the context of computer animation).

Here, we limit the presentation to inverse kinematics, that is, computing the robot velocities from reference velocity task constraints. The same approach can be used in inverse dynamics, to compute the system torques¹⁹ (typically, joint torques, but also tendon forces or other actuation parameters) from homogeneous operational constraints (typically, reference forces or accelerations). In that case, the Euclidean norm is irrelevant, and weighted inverses are generally preferred to enforce minimum energy along the motion.

Stack of tasks, quadratic programming, and inequality constraints. A stack of tasks can be compared to quadratic programming: a quadratic program is an optimization problem that involves a set of linear constraints and a quadratic cost (for example, a linear function to be approximated in the least-square sense). It is then similar to a stack with two tasks: the first (with higher priority) would be the constraint; the secondary would be the cost to minimize. However, the similarity is not total: the constraint in a quadratic program is supposed to be admissible (at least one feasible solution exists), while it is not the case for the main task. Also, a stack of tasks can

be extended to more than two tasks.

Up to now, we have considered a task corresponds to equality in the configuration tangent space, to be satisfied at best in the least-square sense. Consider a region defined by a set of inequalities: the robot can move freely inside the region but should stay inside it; when the task becomes infeasible, it should minimize its distance to the region in the least-square sense. Such inequality constraints cannot be solved directly with the method described here.

Historically, the first solution has been to set a zero velocity in the task space when the inequality constraint is satisfied. This is the artificial potential field approach proposed by Khatib:¹⁸ the target region is described with a low or null cost, while the cost increases when approaching the limit of the region, following the behavior of the barrier functions used in the interior-point numerical algorithms. The gradient of the function then acts as a virtual force that pushes the robot inside the region when approaching the region boundaries while it has zero or very little influence inside the region.

For robot control, penalty functions are generally preferred to barrier functions to prevent bad numerical behavior when the robot is pushed to the limits. For a single task or when the inequality task has the lowest priority, the obtained behavior is always satisfactory: the robot does not have to move when the inequality is satisfied. However, it is difficult to enforce a hierarchy using this approach. The gradient-projection method²⁷ can be used if the inequality task has a secondary importance, in particular, when enforcing the robot constraints in a very redundant context (for instance, a three-dimensional reaching task performed by a six-joint robot arm). When the inequality task has the priority, the saturation of one boundary of the task region will correspond to the allocation of one degree of freedom,^b which is allocated to fix the velocity orthogonal to the boundary. This degree of freedom is thus not available anymore for any secondary task. Moreover, when freely moving inside the region (far from the bound-

aries), this degree of freedom can be used by the secondary tasks. In order to take advantage of the redundancy offered inside the region defined by the inequality constraints, the corresponding degrees of freedom should be dynamically allocated. If the inequality constraint is satisfied, the degree of freedom is left unallocated and can be used by a secondary task: the constraint is said to be inactive. If the constraint is violated, then the corresponding degree of freedom is used to satisfy the constraint at best: the constraint is said to be active.

The set of all active constraints is called the active set. Active-set-search algorithms are iterative resolution schemes searching over all possible active constraints. A candidate solution is computed at each iteration that fits the active constraints. Depending on the status of the active and inactive constraints with respect to the candidate solution, the active set is modified and the process is iterated. Active-set search algorithms are classical to solve inequality-constrained quadratic programs. The priority order between multiple coning objectives can be introduced, leading to hierarchical quadratic programs.⁹

Let us illustrate the stack-of-tasks framework from the worked out example of HRP2 humanoid robot performing two simultaneous reaching tasks, while respecting equilibrium constraints. All elementary tasks are embedded into a single global trajectory. We will see that the hierarchy introduced in quadratic programming induces a structure in the task vector space. Doing so, the global trajectory appears as a composition of elementary movements, each of them characterizing a given task (or subtask). Reverse engineering can then be used to identify the “meaning” of the motion, that is, the various tasks the motion is embedding.

Motion as Action Signature

We review here two practical applications of the stack of tasks on the humanoid robot, HRP2.^c The first one shows how to express complex actions while involving all body segments and respecting physical constraints.

b A degree of freedom is a linear combination of controls in the configuration tangent space.

c A detailed presentation appeared in Hak et al.¹²

The second application shows how it is possible to recognize actions from motion observation using reverse engineering techniques.

From action to motion: The optimization-based selection principle at work. The stack of tasks is a generic tool to generate and to control a robot's motion. Given an initial configuration, a motion is generated by adding a set of tasks into the stack and integrating the resulting velocity until the convergence of all active tasks. The stack of tasks can be used in various robotics scenarios. In humanoid robotics, classical tasks deal with reaching (expressed as the placement of an end effector), visual servoing (expressed as the regulation of the gaze on the position of an object in the image plane of the robot camera), or quasi-static balance (expressed as the regulation of the center-of-mass in such a way that its projection on the floor lies inside the support polygon of the feet).

For example, the motion in Figure 2 (left column) is generated by constraining the two feet and the center of mass to remain to their initial positions and by setting two tasks to control the right hand and the gaze both to the ball in front of the robot. The robot bends forward to reach the ball. In doing so, it pushes the center of mass forward. The left hand moves backward to compensate for this motion of the center of mass. The motion of the left hand does not answer a specific action. It is a side effect of the balance maintenance.

Setting new tasks or changing the desired value of the active tasks can easily modify the motion. For example, the motion in Figure 2 (right column) is generated by adding a task that regulates the position and orientation of the left hand to the final placement of the left hand in the first scenario. This new task is a reaching task: the left hand must reach a goal. The two movements of the left hand in both scenarios look very similar, but their meanings are different. In the first case, the motion is not intentional: the left hand moves to regulate the center-of-mass position; its motion is then a side effect of the other tasks. In the second case, the motion is intentional: the left-hand explicitly moves to reach a given target. A careful analysis of slight differences between the two left-hand

motions eliminates the ambiguity. This is made possible by a reverse-engineering approach.

From motion to action: A reverse-engineering approach of action recognition. The hierarchy artificially decouples the tasks of the stack in order to prevent any conflict between two different tasks. A side effect is the trajectory into a given active task space is not influenced by any other task. For example, on Figure 2 (right side) the stack of tasks enforces a decoupling for the left hand, which moves independently of the two other tasks. The trajectory in one task space then constitutes a signature of the activity of the task in the generation of the robot motion.

Consider the following problem: we observed the motion of a system whose possible controllers are known. Observing only the joint trajectory, the question is to reconstruct which of the possible controllers were active and which were the associated parameters. Recovering one task is easy: the configuration trajectory is projected in all the candidate task spaces using the corresponding task function. The best task is selected by fitting the projected trajectory with the task model (once more, the fitting and thus the selection is done by optimization).

However, if the stack of tasks artificially decouples the active tasks, some coupling between the candidate tasks may occur: for example, there are a lot of similarities between the trajectories of the wrist and the elbow due to their proximity in the kinematic chain. These similarities can lead to false positives in the detection. To avoid this problem, only the most relevant task is chosen first. The motion due to this task is then canceled by projecting the configuration trajectory in the null space of the detected task. The detection algorithm then iterates until all the tasks have been found, that is, until the remaining quantity of movement after successive projections is null.¹²

This detection algorithm can be used to disambiguate the two similar-looking motions performed in Figure 2, without using any contextual information. An illustration of the successive projections is given in Figure 3. The tasks are removed in the order given by the detection algorithm. The right-

hand task is removed first (second row), followed by the center of mass (third row): this cancels most of the motion of the left hand because the coupling between the three tasks is important; however a small part of left-hand movement remains. On the contrary, the head movement, which is nearly decoupled from the right-hand and center-of-mass, remains important. It is totally nullified after removing the gaze task (fourth row). The remaining motion of the left hand can only be explained by the left-hand task, which is detected active and then removed. Finally, the two feet constraints are detected and removed. The effect of the first foot removal (sixth row) is noticeable.

The algorithm achieves very good performances to recognize actions and to tell the differences between similar-looking robot motions. Beyond robotics, the method can be applied to human action recognition. However, it requires a critical prerequisite: the knowledge of the optimality principles grounding the motion generation of intentional actions. Indeed, the algorithm is based on action signatures that are the typical results of a particular cost function. The approach also requires a computational model of the coordination strategies used by the human to compose several simultaneous motion primitives. They are the promising routes for future researches combining computational neuroscience and robotics.

At this stage, we have seen how optimization principles and the notion of tasks help to ground a symbolic representation of actions from motions: an action is viewed as the result of an optimization process whose cost represents the signature of the action. The next section addresses the dual problem of identifying action signatures from motions.

Inverse Optimal Control

Let us introduce the section by a case study taken from humanoid robotics. Suppose we want a humanoid robot to walk as a human, that is, following human-like trajectories. So the question is: What are the computational foundations of human locomotion trajectories? In a first stage, we showed that locomotor trajectories are highly stereotypical across repetitions and subjects. The methodology is based

Figure 3. Successive projection of the motion after detecting each of the seven tasks.

From top to bottom: original movement; removing the right-hand task; removing the center-of-mass task; removing the gaze task; removing the left-hand task; removing the left foot task; removing the right-foot task. On the last row, all the tasks are canceled, the projected movement is totally nullified.



on statistical analysis of a huge motion capture data basis of trajectories (seven subjects, more than 1,500 trajectories).¹⁵ Next, in a second stage, it is assumed that human locomotion trajectories obey some optimality principle. This is a frequent hypothesis in human or animal motion studies. So the question is: Which cost functional is minimized in human locomotion? In practice, we consider the human and the robot obey the same model, that is, we know precisely the differential equation that describes the motions under some control action, and the constraints the state of the system should satisfy. The data basis of trajectories is available. Based on this knowledge, determining a cost functional that is minimized in human locomotion becomes an inverse optimal control problem.

Pioneering work for inverse optimization in control dates back to the 1960s in systems theory applied to economics.²⁹ Similarly, for optimal stabilization problems, it was known that every value function of an optimal stabilization problem is also a Lyapunov function for the closed-loop system. Freeman and Kokotovic¹⁰ have shown that the reciprocal is true: namely, every Lyapunov function for every stable closed-loop system is also a value function for a meaningful optimal stabilization problem.

In static optimization, the direct problem consists in finding in some set K of admissible solutions a feasible point x that minimizes some given cost function f .

We state the associated inverse optimization problem as follows: given a feasible point y in K , find a cost criterion g that minimizes the norm of the error $(g - f)$, with g being such that y is an optimal solution of the direct optimization problem with cost criterion g (instead of f). When f is the null function, this is the static version of the inverse optimal control problem. Pioneering works date back to the 1990s for linear programs, and for the Manhattan norm. For the latter, the inverse problem is again a linear program of the same form. Similar results also hold for inverse linear programs with the infinite norm. The interested reader will find a nice survey on inverse optimization for linear programming and combinatorial optimization problems in Heuberger.¹⁴

Note that, in inverse optimization, the main difficulty lies in having a tractable characterization of global optimality for a given point and some candidate cost criterion. This is why most of all the above works address linear programs or combinatorial optimization problems for which some characterization of global optimality is available and can sometimes be effectively used for practical computation. This explains why inverse (nonlinear) optimization has not attracted much attention in the past.

Recently, some progress has been made in inverse polynomial optimization; that is, inverse optimization problems with polynomial objective function and semi-algebraic set as feasible set of solutions.²² Powerful representation results in real algebraic geometry²¹ describe the global optimality constraint via some certificate of positivity. These can be stated as linear matrix inequalities (LMIs) on the unknown vector of coefficients of the polynomial cost function. The latter set is a convex set on which we can optimize efficiently via semi-definite programming,²³ a powerful technique of convex optimization. We can then show that computing an inverse optimal solution reduces to solving a hierarchy of semi-definite programs of increasing size.

Back to the inverse optimal control problem for anthropomorphic locomotion, we can consider a basis of functions to express the cost function candidate. The method proposed in Mombaur et al.²⁸ is based on two main algorithms: an efficient direct multiple shooting technique to handle optimal control problems, and a state-of-the-art optimization technique to guarantee a match between a solution of the (direct) optimal control problem and measurements. Once an optimal cost function has been identified (in the given class of basis functions), we can implement a direct optimal control solution on the humanoid robot. So far, the method is rather efficient at least on a sample of test problems. However, it requires defining a priori class of basis functions. Moreover, the direct shooting method provides only a local optimal solution at each iteration of the algorithm.

Thus, there is no guarantee of global optimality.

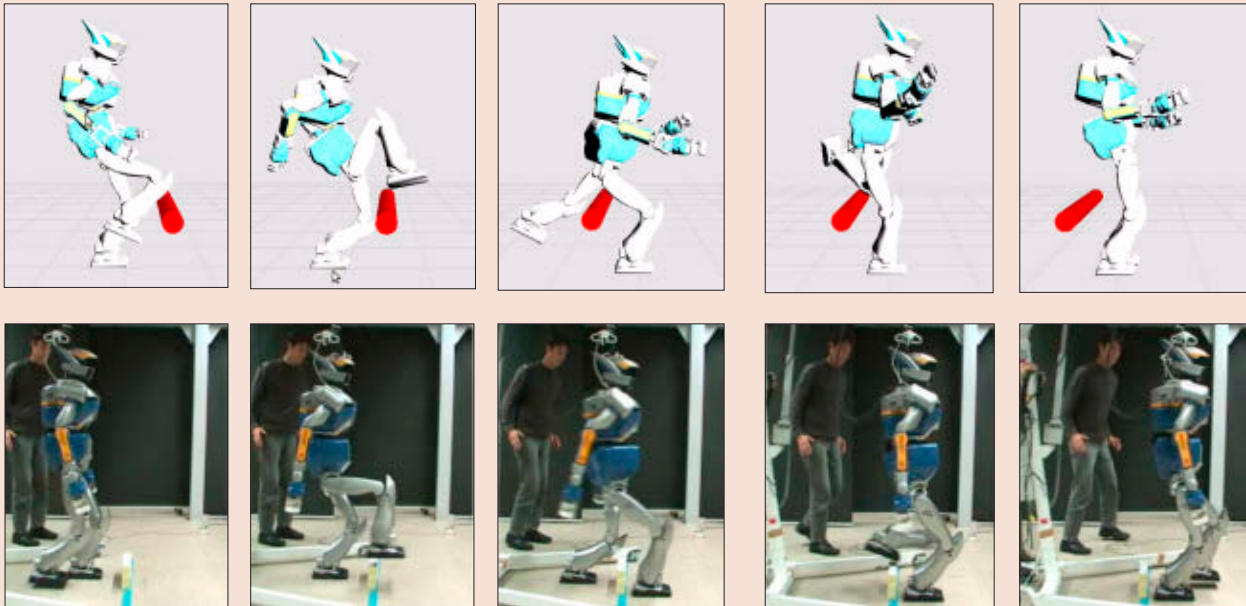
An alternative way to consider the

problem is to extend the methodology developed for inverse polynomial optimization²² to the context of inverse optimal control. Note that the Hamilton-Jacobi-Bellman (HJB) equation is the perfect tool to certify global optimality of a given state-control trajectory whenever the optimal value function is known. The basic idea is to use a relaxed version of the HJB-optimality equation as a certificate of global optimality for the experimental trajectories stored in the database. The optimal value function, which is generally assumed to be continuous, can be approximated on a compact domain by a polynomial. If we search for an integral cost functional whose integrand h is also a polynomial, then this certificate of global optimality can be used to compute h and an associated (polynomial) optimal value function by solving a semi-definite program. Proceeding as in Lasserre,²² we solve a hierarchy of semi-definite programs of increasing size. At each step of this hierarchy, either the semi-definite program has no solution or any optimal solution h is such that the trajectories of the database are global optimal solutions for the problem with polynomial cost function h as integrand. The higher in the hierarchy the better is the quality of the solution (but also at a higher computational cost)

Apart from polynomial optimization techniques, other approaches have been recently introduced with a geometric perspective of optimal control theory. Significant results have been obtained in the context of pointing motions:³ based on Thom transversability theory, the cost structure is deduced from qualitative properties highlighted by the experimental data. These can be, for instance, the characterization of inactivity intervals of the muscles during the motion. Such a qualitative approach has been also successfully applied to human locomotion.⁶

We have introduced the inverse optimal control problem from the perspective of biomimetic approaches to robot control. The question is: how to synthesize natural motion laws to deduce from them optimal control models for robots? We emphasized recent developments in inverse polynomial optimization. We should note this article is far from covering all the approaches to inverse optimal con-

Figure 4. Two stepping movements obtained with (top) a whole-body trajectory optimization³⁶ (courtesy from K. Mombaur) and (bottom) a linearized-inverted-pendulum based walking pattern generator¹⁷ (courtesy from O. Stasse.³⁹). The whole-body optimization enables the robot to reach higher performances but the numerical resolution is yet too slow to obtain an effective controller.



trol. Inverse optimal control is also an active research area in machine learning. In the context of reinforcement learning,^{16,20} inverse reinforcement learning constitutes another resolution paradigm based on Markov decision processes with spectacular results on challenging problems such as helicopter control.⁷ The method corpus comes from stochastic control (see Kober et al.²⁰ and references therein.)

Computation: A Practical or Theoretical Problem?

In computer animation optimization-based motion generation is experienced as giving excellent results in terms of realism in mimicking nature. For instance, it is possible to use numerical optimization to simulate very realistic walking, stepping, or running motions for human-like artifacts. These complex body structures include up to 12 body segments and 25 degrees of freedom.³⁶ At first glance, the approach a priori applies to humanoid robotics. Figure 4 (top) provides an example of the way HRP2 steps over a very large obstacle.

However, robotics imposes physical constraints absent from the virtual worlds and requiring computation performance. Biped walking is a typical example where the technological limi-

tation implies a search of alternative formulations. The bottleneck is the capacity of the control algorithm to meet the real-time constraints.

In the current model-based simulation experiments the time of computation is evaluated in minutes. Minute is not a time scale compatible with real time. For instance, computation time upper-bounds of a few milliseconds are required to ensure the stability of a standing humanoid robot. So taking advantage of general optimization techniques for the real-time control necessary requires building simplified models or to develop dedicated methods. The issue constitutes an active line of research combining robot control and numerical optimization.

An example is given by the research on walking motion generation for humanoid robots. The most popular walking pattern generator is based on a simplified model of the anthropomorphic body: the linearized inverted pendulum model. It was introduced in Kajita et al.¹⁷ and developed for the HRP2 humanoid robot. The method is based on two major assumptions: (1) the first one simplifies the control model by imposing a constant altitude of the center of mass, (2) the second one assumes the knowledge of the footprints. Assumption (1)

has the advantage to transform the original nonlinear problem into a linear one. The corresponding model is low dimensioned and it is possible to address (1) via an optimization formulation.⁸ With this formulation, assumption (2) is no longer required. The method then gives rise to an on-line walking motion generator with automatic footstep placement. This is made possible by a linear model-predictive control whose associated quadratic program allows much faster control loops than the original ones in Kajita et al.¹⁷ Indeed, running the full quadratic program takes less than 1ms with state-of-the-art solvers. More than that, in this specific context, it is possible to devise an optimized algorithm that reduces by 100 the computation time of a solution.⁸

An example of this approach is given in Figure 4 (bottom) that makes the real HRP2 step over an obstacle. The approach based on model reduction enables the robot to be controlled in real time. However, the reduced model does not make a complete use of the robot dynamics. The generated movement is less optimal than when optimizing the robot whole-body trajectory. Consequently, it is not possible to reach the same performances (in this case, the same obstacle height): the whole-body optimization enables the robot to reach

higher performances, but only offline. Reaching the same performance online requires either more powerful computers (running the same algorithms) or more clever algorithms.

Conclusion

The notion of robot motion optimality is diverse in both its definitions and its application domains. One goal of this article was to summarize several points of view and references spread out over various domains: robotics, control, differential geometry, numerical optimization, machine learning, and even neurophysiology.

The objective was to stress the expressive power of optimal motion in robot action modeling and to present current challenges in numerical optimization for real-time control of complex robots, like the humanoids. A second objective was to report recent issues in inverse optimal control. While its stochastic formulation is popular in machine learning, other paradigms are currently emerging in differential geometric control theory and polynomial optimization.

As testified in a companion article,²⁴ robotics offers rich benchmarks for optimal control theory. Due to real-time computation constraints imposed by effective applications, robotics also induces challenges to numerical optimization. The difficulty for roboticists is to find the right compromise between generality and specificity. General algorithms suffer from the classical curse of dimensionality that constitutes a bottleneck for robot control. Therefore, they may be used for offline motion generation, but they are inefficient for real-time applications. Real-time robot control requires very fast computations. It requires dedicated numerical optimization methods. We have seen how bipedal walking illustrates this tension between generality and specificity. Roboticists are today asking optimization theorists for more efficient algorithms, while they are developing at the same time a specific know-how to this end.

Last but not least, let us conclude by referring to a controversy introduced by neurophysiologist K. Friston. In a recent paper,¹¹ he asks the provocative question: “Is optimal control theory useful for understanding motor behavior or is it a misdirection?” He opposes

to optimal control the competitive notion of active inference. While the paper is mainly dedicated to motor control in life sciences, the issue is of crucial and utmost interest for roboticists and calls for a reinforcement of the cooperation between life and engineering sciences.

Acknowledgments

This article benefits from comments by Quang Cuong Pham, from a careful reading by Joel Chavas, and above all, from the quality of the reviews. The work has been partly supported by ERC Grant 340050 Actanthrope, by a grant of the Gaspar Monge Program for Optimization and Operations Research of the Fondation Mathématique Jacques Hadamard (FMJH) and by the grant ANR 13-CORD-002-01 Entracte. □

References

- Baerlocher, P. and Boulic, R. An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer* 6, 20 (2004), 402–417.
- Ben-Israel, A. and Greville, T. Generalized inverses: Theory and applications. *MS Books in Mathematics*. Springer, 2nd edition, 2003.
- Berret, B. et al. The inactivation principle: Mathematical solutions minimizing the absolute work and biological implications for the planning of arm movements. *PLoS Comput Biol.* 4, 10 (2008), e1000194.
- Chaumette, F. and Hutchinson, S. Visual servo control, Part I: Basic approaches. *IEEE Robotics and Automation Magazine* 13, 4 (2006), 82–90.
- Chiaverini, S. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Trans. on Robotics and Automation* 13, 3 (1997), 398–410.
- Chitour, Y., Jean, F. and Mason, P. Optimal control models of goal-oriented human locomotion. *SIAM J. Control and Optimization* 50 (2012), 147–170.
- Coates, A., Abbeel, P. and Ng, A. Apprenticeship learning for helicopter control. *Commun. ACM* 52, 7 (July 2009), 97–105.
- Dimitrov, D., Wieber, P.-B., Stasse, O., Ferreau, H. and Diedam, H. An optimized linear model predictive control solver. *Recent Advances in Optimization and its Applications in Engineering*. Springer, 2010, 309–318.
- Escande, A., Mansard, N. and Wieber, P.-B. Hierarchical quadratic programming. *The Intern. J. Robotics Research* 33, 7 (2014), 1006–1028.
- Freeman, R. and Kokotovic, P. Inverse optimality in robust stabilization. *SIAM J. Control Optim.* 34 (1996), 1365–1391.
- Friston, K. What is optimal about motor control? *Neuron* 72, 3 (2011), 488–498.
- Hak, S., Mansard, N., Stasse, O. and Laumond, J.-P. Reverse control for humanoid robot task recognition. *IEEE Trans. Sys. Man Cybernetics* 42, 6 (2012), 1524–1537.
- Harris, C. and Pike, J. 3d positional integration from image sequences. *Image and Vision Computing* 6, 2 (1988), 87–90.
- Heuberger, C. Inverse combinatorial optimization: A survey on problems, methods and results. *J. Comb. Optim.* 8 (2004), 329–361.
- H. Hicheur, H., Pham, Q., Arechavaleta, G., Laumond, J.-P. and Berthoz, A. The formation of trajectories during goal-oriented locomotion in humans. Part I: A stereotyped behaviour. *European J. Neuroscience* 26, 8 (2007), 2376–2390.
- Kaelbling, L., Littman, M. and Moore, A. Reinforcement learning: A survey. *J. Artificial Intelligence Research* 4 (1996), 237–285.
- Kajita, S. et al. Biped walking pattern generation by using preview control of zero-moment point. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation* (2003), 1620–1626.
- Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. *The Intern. J. Robotics Research* 5, 1 (1986), 90–98.
- Khatib, O. A unified approach for motion and force control of robot manipulators: The operational space formulation. *The Intern. J. Robotics Research* 3, 1 (1987), 43–53.
- Kober, J., Bagnell, J. and Peters, J. Reinforcement learning in robotics: A survey. *The Intern. J. Robotics Research* 32, 11 (Sept. 2013).
- Lasserre, J.B. *Moments, Positive Polynomials and Their Applications*. Imperial College Press, London, 2010.
- Lasserre, J.B. Inverse polynomial optimization. *Math. Oper. Res.* 38, 3 (Aug. 2013), 418–436.
- Lasserre, J.B. and Anjos, M., eds. *Semidefinite, Conic and Polynomial Optimization*. Springer, 2011.
- Laumond, J.-P., Mansard, N. and Lasserre, J.B. Optimality in robot motion (I): Optimality versus optimized motion. *Commun. ACM* 57, 9 (Sept. 2014), 82–89.
- Liégeois, A. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. Systems, Man and Cybernetics* 7 (1977), 868–871.
- Mansard, N. and Chaumette, F. Task sequencing for sensor-based control. *IEEE Trans. on Robotics* 23, 1 (2008), 60–72.
- Marchand, E. and Hager, G. Dynamic sensor planning in visual servoing. In *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (1998), 1988–1993.
- Mombaur, K., Truong, A. and Laumond, J.-P. From human to humanoid locomotion: An inverse optimal control approach. *Autonomous Robots* 28, 3 (2010).
- Mordecai, K. On the inverse optimal problem: Mathematical systems theory and economics, I, II. *Lecture Notes in Oper. Res. and Math. Economics* 11, 12 (1969).
- Nakamura, Y., Hanafusa, H. and Yoshikawa, T. Task-priority based redundancy control of robot manipulators. *The Intern. J. Robotics Research* 6, 2 (1987), 3–15.
- Paul, R. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, 1st edition, 1982.
- Poincaré, H. On the foundations of geometry (1898). *From Kant to Hilbert: A Source Book in the Foundations of Mathematics*. W. Ewald, Ed. Oxford University Press, 1996.
- Rosen, J. The gradient projection method for nonlinear programming. Part II. Nonlinear constraints. *SIAM J. Applied Mathematics* 9, 4 (1961), 514–532.
- Samson, C., Borngren, M.L. and Espiau, B. *Robot Control: The Task Function Approach*. Clarendon Press, 1991.
- Schaal, S., Mohajerian, P. and Ijspeert, A. Dynamics system vs. optimal control: A unifying view. *Prog. Brain Research* 165 (2007), 425–445.
- Schultz, G. and Mombaur, K. Modeling and optimal control of human-like running. *IEEE/ASME Trans. Mechatronics* 15, 5 (2010), 783–792.
- Siciliano, B., Sciavicco, L., Villani, L. and Oriolo, G. *Robotics: Modeling, Planning and Control*. Springer, 2009.
- Siciliano, B. and Slotine, J.-J. A general framework for managing multiple tasks in highly redundant robotic systems. In *Proceedings of the IEEE Int. Conf. on Advanced Robot.*, 1991.
- Stasse, O., Verrelst, B., Vanderborght, B. and Yokoi, K. Strategies for humanoid robots to dynamically walk over large obstacles. *IEEE Trans. Robotics* 25 (2009), 960–967.
- Todorov, E. Optimality principles in sensorimotor control. *Nature Neuroscience* 7, 9 (2004), 905–915.
- Whitney, D. Resolved motion rate control of manipulators and human prostheses. *IEEE Trans. Man-Machine Systems* 10, 2 (1969), 47–53.

Jean-Paul Laumond (jpl@laas.fr) is a CNRS director of research at LAAS, Toulouse, France.

Nicolas Mansard (nmansard@laas.fr) is a CNRS researcher at LAAS, Toulouse, France.

Jean Bernard Lasserre (lasserre@laas.fr) is CNRS director of research at LAAS, Toulouse, France.



Watch the authors discuss this work in this exclusive *Communications* video.

research highlights

P. 76

Technical Perspective Programming Multicore Computers

By James Larus

P. 77

Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?

By Nadathur Satish, Changkyu Kim, Jatin Chhugani,
Hideki Saito, Rakesh Krishnaiyer, Mikhail Smelyanskiy,
Milind Girkar, and Pradeep Dubey

Technical Perspective

Programming Multicore Computers

By James Larus

WHAT IS THE best way to program a parallel computer? Common answers are to have a compiler transform a sequential program into a parallel one or to write a parallel program using a parallel language or library.

In the early days of parallel computers, parallelizing compilers offered the tantalizing promise of running unmodified “dusty deck” sequential FORTRAN programs on the emerging parallel computers. Although research on these compilers led to many program analysis and representation innovations used in modern compilers, the resulting tools were not successful at parallelizing most applications, and developers turned instead to libraries such as pthreads and MPI.

In this approach, programs use parallel constructs; either explicitly parallel operations such as fork-join or implicitly parallel operations such as map and reduce. These abstractions in theory should encourage developers to think “parallel” and write parallel programs, but in practice, even with them, parallel programming is challenging because of new types of errors such as data races and the diversity of parallel machines (for example, message passing, shared memory, and SIMD).

So, what can a developer do to improve the performance of his or her code on a modern, parallel microprocessor with multiple cores and vector processing units? The following paper advocates an appealing division of labor between a developer and a compiler, with the human restructuring code and data structures and forcing parallel execution of some loops, thereby increasing the opportunities for the compiler to generate and optimize parallel machine code.

The results in this paper are quite striking. For 11 computationally intensive kernels, code developed in this manner performed within an average of 30% of the best hand-optimized code

and did not require the developer to use low-level programming constructs or to understand a machine’s architecture and instruction set.

But why is this division of labor necessary? Why are compilers unable to parallelize and vectorize these (relatively simple) programs? The authors allude to “difficult issues such as dependency analysis, memory alias analysis, and control-flow analysis.” In practice, compilers employ a large repertoire of local optimizations, each of which incrementally improves a small region of code. Large, pervasive restructurings that change how a program computes its result are outside of the purview of a traditional compiler. Until recent work on program synthesis, there has been little research on efficient techniques for exploring large spaces of possible transformations. Moreover, even for local optimizations, compilers are hamstrung by conservative program analysis, which at best only approximates a program’s potential behavior^a


^a Many program analyses, if fully precise, would allow solution of the Turing halting problem.

The following paper argues the restructurings and annotations, when performed by developers, should be part of every programmer’s repertoire for modern computers.

and must disallow optimizations that might adversely affect a program’s result.

This paper argues the restructurings and annotations, when performed by developers, are not difficult and should be part of every programmer’s repertoire for modern computers. These changes include transforming an array of structures into a structure of arrays, blocking loops to increase data reuse, annotating parallel loops, and adopting more parallel algorithms. Conceptually, none of these changes is difficult to understand—although finding a new algorithm may be challenging. However, these modifications can introduce errors into a program and can be complex to apply to a large application, where a data structure may be shared by many routines.

Of course, program optimization in general can have similarly pernicious effects on program structure and readability, so these concerns are not limited to parallel programs. Balanced against the challenge of directly writing a correct, high-performing parallel program, restructuring and annotation appear to be a reasonable methodology that produces maintainable programs. However, this approach would have little value if the resulting programs do not run significantly faster.

The paper’s principal contribution is to demonstrate this division of labor between human and compiler achieves its goal of effectively using hardware parallelism to improve performance. Mature, modern compilers—aided by restructuring and annotation—can produce extremely efficient parallel code. Neither compilers nor people are very good at achieving this goal on their own. 

James Larus is a professor and dean of computer and communications sciences at EPFL, Lausanne, Switzerland.

Copyright held by author.

Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?

By Nadathur Satish, Changkyu Kim,* Jatin Chhugani,* Hideki Saito, Rakesh Krishnaier, Mikhail Smelyanskiy, Milind Girkar, and Pradeep Dubey

Abstract

Current processor trends of integrating more cores with wider Single-instruction multiple-data (SIMD) units, along with a deeper and complex memory hierarchy, have made it increasingly more challenging to extract performance from applications. It is believed by some that traditional approaches to programming do not apply to these modern processors and hence radical new languages must be designed. In this paper, we question this thinking and offer evidence in support of traditional programming methods and the performance-versus-programming effort effectiveness of multi-core processors and upcoming many-core architectures in delivering significant speedup, and close-to-optimal performance for commonly used parallel computing workloads.

We first quantify the extent of the “Ninja gap,” which is the performance gap between naively written C/C++ code that is parallelism unaware (often serial) and best-optimized code on modern multi-/many-core processors. Using a set of representative throughput computing benchmarks, we show that there is an average *Ninja gap* of 24X (up to 53X) for a 6-core Intel® Core™ i7 X980 Westmere CPU, and that this gap if left unaddressed will inevitably increase. We show how a set of well-known algorithmic changes coupled with advancements in modern compiler technology can bring down the Ninja gap to an average of just 1.3X. These changes typically require low programming effort, as compared to the very high effort in producing Ninja code. We show equally encouraging results for the upcoming Intel® Xeon Phi™ architecture which has more cores and wider SIMD. We thus demonstrate that we can contain the otherwise uncontrolled growth of the Ninja gap and offer a more stable and predictable performance growth over future architectures, offering strong evidence that radical language changes are not required.

1. INTRODUCTION

Performance scaling across processor generations has previously relied on increasing clock frequency. Programmers could ride this trend and did not have to make significant code changes for improved code performance. However, clock frequency scaling has hit the power wall,¹⁶ and the free lunch for programmers is over.

Recent techniques for increasing processor

performance have a focus on integrating more cores with wider Single-instruction multiple-data (SIMD) units, while simultaneously making the memory hierarchy deeper and more complex. While the peak compute and memory bandwidth on recent processors has been increasing, it has become more challenging to extract performance out of these platforms. This has led to the situation where only a small number of expert programmers (“Ninja programmers”) are capable of harnessing the full power of modern multi-/many-core processors, while the average programmer only obtains a small fraction of this performance. We define the term “Ninja gap” as the performance gap between naively written parallelism unaware (often serial) code and best-optimized code on modern multi-/many-core processors.

There have been many recent publications^{8, 14, 17, 24} that show 10–100X performance improvements for real-world applications through optimized platform-specific parallel implementations, proving that a large Ninja gap exists. This typically requires high programming effort and may have to be re-optimized for each processor generation. However, these papers do not comment on the effort involved in these optimizations. In this paper, we aim at quantifying the extent of the Ninja gap, analyzing the causes of the gap and investigating how much of the gap can be bridged with low effort using traditional C/C++ programming languages.^a

We first quantify the extent of the Ninja gap. We use a set of real-world applications that require high throughput (and inherently have a large amount of parallelism to exploit). We choose throughput applications because they form an increasingly important class of applications⁷ and because they offer the most opportunity for exploiting architectural resources—leading to large Ninja gaps

^a Since measures of ease of programming such as programming time or lines of code are largely subjective, we show code snippets with the code changes required to achieve performance.

The original version of this paper was published in the *Proceedings of the 39th Annual International Symposium on Computer Architecture* (June 2012). IEEE Computer Society, Washington, D.C., 440–451.

* This work was done when these authors were at Intel.

if naive code does not take advantage of these resources. We measure performance of our benchmarks on a variety of platforms across different generations: 2-core Conroe, 4-core Nehalem, 6-core Westmere, Intel® Xeon Phi™^b, and the NVIDIA C2050 GPU. Figure 1 shows the Ninja gap for our benchmarks on three CPU platforms: a 2.4 GHz 2-core E6600 Conroe, a 3.33 GHz 4-core Core i7 975 Nehalem, and a 3.33 GHz 6-core Core i7 X980 Westmere. The figure shows that there is up to a 53X gap between naive C/C++ code and best-optimized code for a recent 6-core Westmere CPU. The figure also shows that this gap has been increasing across processor generations — the gap is 5–20X on a 2-core Conroe system (average of 7X) to 20–53X on Westmere (average of 25X). This is in spite of micro-architectural improvements that have reduced the need and impact of performing various optimizations.

We next analyze the sources of the large performance gap. There are many reasons why naive code performs badly. First, the code may not be parallelized, and compilers do not automatically identify parallel regions. This means that the increasing core count is not utilized in naive code, while the optimized code takes full advantage of it. Second, the code may not be vectorized, thus under-utilizing the increasing SIMD widths. While auto-vectorization has been studied for a long time, there are many difficult issues such as dependency analysis, memory alias analysis and control flow analysis which prevent compilers from vectorizing outer loops, loops with gathers (irregular memory accesses) and even innermost loops where dependency and alias analysis fails. A third reason for large performance gaps may be that the code is bound by memory bandwidth—this may occur, for instance, if the code is not blocked for cache hierarchies—resulting in cache misses.

Our analysis of code written by Ninja programmers show that such programmers put in significant effort to

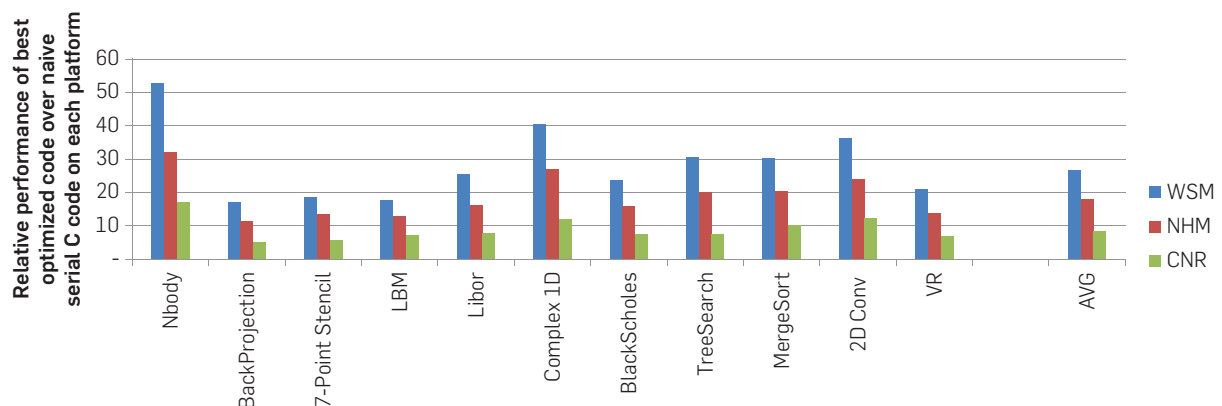
use threading technologies such as pthreads along with low level intrinsics for vectorization to obtain performance. This can result in very complex code especially when vectorizing irregular loops. In this work, we show that we can leverage recent compiler technologies that enable parallelization and vectorization of code with relatively low programmer effort. Parallelization can be achieved using OpenMP pragmas over the loop to be parallelized, and the programmer avoids writing complex threading code. For simple loops without dependencies, automatic loop parallelization is also possible—we assume the use of pragmas in this work. For vectorization, recent compilers such as the Intel® Composer XE 2011 version have introduced the use of a pragma for the programmer to force loop vectorization by circumventing the need to do dependency and alias analysis. This version of the compiler also has the ability to vectorize outer level loops, and the Intel® Cilk™ Plus feature¹¹ helps the programmer to use this new functionality when it is not triggered automatically.^c These features allow programmers to move away from using lower level intrinsics and/or assembly code and immensely boost performance. Using these features, we show that the Ninja gap **reduces to an average of 2.95X for Westmere**. The remaining gap is either a result of bandwidth bottlenecks in the code or the fact that the code gets only partially vectorized due to irregular memory accesses. This remaining gap, while relatively small, will however inevitably increase on future architectures with growing SIMD widths and decreasing bandwidth-to-compute ratios.

In order to bridge the remaining gap, programmer intervention is required. Current compilers do not automate changes at an algorithmic level that involve memory layout changes, and these must be manually performed by the programmer. We identify and suggest three critical algorithmic changes: **blocking** for caches, bandwidth/SIMD friendly **data layouts** and in some cases, choosing an

^b Intel, Xeon and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

^c For more complete information about compiler optimizations, see the optimization notice at <http://software.intel.com/en-us/articles/optimization-notice/>.

Figure 1. Growing performance gap between Naive serial C/C++ code and best-optimized code on a 2-core Conroe (CNR), 4-core Nehalem (NHM), and 6-core Westmere (WSM) systems.



alternative SIMD-friendly algorithm. An important class of algorithmic changes involves blocking the data structures to fit in the cache, thus reducing the memory bandwidth pressure. Another class of changes involves eliminating the use of memory gather/scatter operations. Such irregular memory operations can both increase latency and bandwidth usage, as well as limit the scope of compiler vectorization. A common data layout change is to convert data structures written in an Array of Structures (AOS) representation to a Structure of Arrays (SOA) representation. This helps prevent gathers when accessing one field of the structure across the array elements, and helps the compiler vectorize loops that iterate over the array. Finally, in some cases, the code cannot be vectorized due to back-to-back dependencies between loop iterations, and in those cases a different SIMD-friendly algorithm may need to be chosen.

Performing algorithmic changes does require programmer effort and insights, and we expect that education and training in parallel programming will play a big role in enabling programmers to develop better parallel algorithms. The payoffs are large—we show that **after performing algorithmic changes**, we have an **average performance gap of only 1.3X** between best-optimized and compiler-generated code. Moreover, this effort can be amortized across different processor generations and also across different computing platforms such as GPUs. Since the underlying hardware trends toward increasing cores, SIMD width and slowly increasing bandwidth have been optimized for, a **small and predictable** performance gap will remain across future architectures. We demonstrate this by repeating our experiments for the **Intel® Xeon Phi™ Knights Corner co-processor architecture**, a recent 86X based manycore platform. We show that the **Ninja gap is almost the same (1.2X)**. In fact, the addition of hardware gather support makes programmability easier for at least one benchmark. The combination of algorithmic changes coupled with modern compiler technology is an important step toward enabling programmers to ride the trend of parallel processing using traditional programming.

2. BENCHMARK DESCRIPTION

For our study, we analyze compute and memory characteristics of recently proposed benchmark suites,^{2, 3, 5} and choose a representative set of benchmarks from the suite of **throughput computing applications**. Throughput workloads deal with processing large amounts of data in a given amount of time, and require a fast response time for all the data processed. These include workloads from areas of High Performance Computing, Financial Services, Image Processing, Databases, etc.⁵ Throughput computing applications have plenty of data- and thread-level parallelism, and have been identified as one of the most important classes of future applications with *compute* and *memory characteristics* influencing the design of current and upcoming multi-/many-core processors. They also offer the most opportunity for exploiting architectural resources—leading to large Ninja gaps if naive code does not take advantage of increasing computational resources. We formulated a representative set of benchmarks described

below that **cover this wide range of application domains** of throughput computing. We capture the key computational kernels where most time is spent in throughput computing applications. As such, reducing Ninja gap in our benchmarks will also translate to the applications themselves.

1. NBody: NBody computations are used in many scientific applications, including the fields of astrophysics and statistical learning algorithms.¹ For given \mathcal{N} bodies, the basic computation is an $O(\mathcal{N}^2)$ algorithm that has two loops over the bodies, and computes and accumulates pair-wise interactions between them.

2. BackProjection: Backprojection is a commonly used kernel in performing cone-beam reconstruction of CT data.¹³ A set of 2D images are “back-projected” onto a 3D volume in order to construct the grid of density values. For each input image, each 3D grid point is projected onto the image, and the density from the neighboring 2×2 pixels is bilinearly interpolated and accumulated.

3. 7-Point Stencil: Stencil computation is used for a wide range of scientific disciplines.⁸ The computation involves multiple sweeps over a spatial input 3D grid of points, where each sweep computes the weighted sum of each grid point and its neighbors, and stores the computed value to the output grid.

4. Lattice Boltzmann Method (LBM): LBM is a class of computational fluid dynamics capable of modeling complex flow problems.²⁵ It simulates the evolution of particle distribution functions over a 3D lattice over many time-steps. For each time-step, at each grid point, the computation performed involves directional density values for the grid point and its face (6) and edge (12) neighbors (also referred to as D3Q19).

5. LIBOR Monte Carlo: The LIBOR market model⁴ is used to price a portfolio of swaptions. It models a set of forward rates as a log-normal distribution. A typical Monte Carlo approach would generate random samples for this distribution and compute the derivative price using a large number of independent paths.

6. Complex 1D Convolution: This is widely used in application areas like image processing, radar tracking, etc. It performs 1D convolution on complex 1D images with a large complex filter.

7. BlackScholes: The Black-Scholes model provides a partial differential equation (PDE) for the evolution of an option price. For European options, there is a closed form expression for the solution of the PDE.²⁰ This involves a number of math operations: computation of exponent, log, square-root, and division operations.

8. TreeSearch: In-memory tree structured index search is commonly used in commercial databases.¹⁴ This application involves multiple parallel searches over a tree with different queries, with each query tracing a path through the tree depending on the results of comparison to the node value at each tree level.

9. MergeSort: MergeSort is commonly used in the area of databases, etc.,⁶ and also shown to be the sorting algorithm of choice for future architectures.²² It sorts an array of \mathcal{N} elements using $\log \mathcal{N}$ merge passes over the complete array.

10. 2D 5×5 Convolution: Convolution is a common

image filtering operation used for effects such as blur, emboss, etc.¹⁵ For a given 2D image and a 5×5 spatial filter, each pixel computes and stores the weighted sum of a 5×5 neighborhood of pixels, where the weights are the corresponding values in the filter.

11. Volume Rendering: Volume Rendering is commonly used in the fields of medical imaging,²⁴ etc. Given a 3D grid, and a 2D image location, the benchmark spawns rays perpendicular to the image plane through the 3D grid, which accumulates the color and opacity to compute the final color of each pixel of the image.

Ninja Performance: Table 1 provides details of the representative dataset sizes for each of the benchmarks. There exists a corresponding best performing code for each, for which the performance numbers have been previously cited^d on different platforms than those used in our study. In order to perform a fair comparison, we **implemented and aggressively optimized (including the use of intrinsics/assembly code) the benchmarks**, and obtained *comparable performance* to the best reported numbers on the corresponding platform. This code was then executed on our platforms to obtain the corresponding best optimized performance numbers we use in this paper. Table 1 (column 3) show the best optimized (*Ninja*) performance for all the benchmarks on Intel® Core™ i7 X980. For the rest of the paper, **Ninja Performance refers to the performance numbers** obtained by executing this code on our platforms.

3. BRIDGING THE NINJA GAP

In this section, we take each of the benchmarks described in Section 2, and attempt to bridge the Ninja gap starting with naively written code with low programming effort. For a detailed performance analysis, we refer the reader to our ISCA paper.²³

Platform: We measured the performance on a 3.3 GHz 6-core Intel® Core™ i7 X980 (code-named Westmere, peak compute: 158 GFlops, peak bandwidth: 30 GBps). The cores feature an out-of-order super-scalar micro-architecture,

Table 1. Various benchmarks and the respective datasets used, along with best optimized (Ninja) performance on Core i7 X980.

Benchmark	Dataset	Best Optimized Performance
NBody ¹	10 ⁶ bodies	7.5×10^9 Pairs/sec
BackProjection ¹³	500 images on 1K ³	1.9×10^9 Proj./sec
7 Point 3D Stencil ¹⁷	512 ³ grid	4.9×10^9 Up./sec
LBM ¹⁷	256 ³ grid	2.3×10^8 Up./sec
LIBOR ¹⁰	10M paths on 15 options	8.2×10^5 Paths/sec
Complex 1D Conv. ¹²	8K on 1.28M pixels	1.9×10^8 Pixels/sec
BlackScholes ²⁰	1M call + put options	8.1×10^8 Options/sec
TreeSearch ¹⁴	100M queries on 64M tree	7.1×10^7 Queries/sec
MergeSort ⁶	256M elements	2.1×10^8 Data/sec
2D 5X5 Convolution ¹⁵	2K \times 2K Image	2.2×10^9 Pixels/sec
Volume Rendering ²⁴	512 ³ volume	2.0×10^8 Rays/sec

^d The best reported numbers are cited from recent top-tier publications in the area of Databases, HPC, Image processing, etc. To the best of our knowledge, there *does not exist* any faster performing code for any of the benchmarks.

with 2-way Simultaneous Multi-Threading (SMT). It also has 4-wide SIMD units that support a wide range of instructions. Each core has an individual 32 KB L1 and 256 KB L2 cache. The cores share a 12 MB last-level cache (LLC). Our system has 12 GB RAM and runs SuSE Enterprise Linux (ver. 11). We use the Intel® Composer XE 2011 compiler.

Methodology: For each benchmark, we attempt to first get good single thread performance by exploiting instruction and data level parallelism. To exploit data level parallelism, we measure the SIMD scaling for each benchmark by running the code with auto-vectorization enabled/disabled (-no-vec compiler flag). If SIMD scaling is not close to peak, we analyze the generated code to identify architectural bottlenecks. We then obtain thread level parallelism by adding OpenMP pragmas to parallelize the benchmark and evaluate thread scaling—again evaluating bottlenecks. Finally, we make necessary algorithmic changes to overcome the bottlenecks.

Compiler pragmas used: We use OpenMP for thread-level parallelism, and use the auto-vectorizer or recent technologies such as array notations introduced as part of the Intel® Cilk™ Plus (hereafter referred to array notations) for data parallelism. Details about the specific compiler techniques are available in Tian et al.²⁶ The compiler directives we add to the code and command line are the following:

- ILP optimizations: We use `#pragma unroll` directive before an innermost loop, and `#pragma unroll_and_jam` primitive outside an outer loop. Both optionally accept the number of times a loop is to be unrolled.
- Vectorizing at innermost loop level: If auto-vectorization fails, the programmer can force vectorization using `#pragma simd`. This is a recent feature introduced in Cilk Plus.¹¹
- Vectorizing at outer loop levels: This can be done in two different ways: (1) directly vectorize at outer loop levels, and (2) Stripmine outer loop iterations and change each statement in the loop body to operate on the strip. In this study, we used the second approach with array notations.
- Parallelization: We use the OpenMP `#pragma omp` to parallelize loops. We typically use this over an outer for loop using a `#pragma omp parallel for` construct.
- Fast math: We use the `-fimf-precision` flag selectively to our benchmarks depending on precision needs.

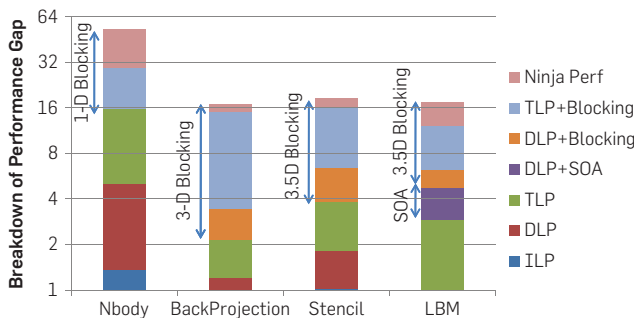
1. Nbody: We implemented Nbody on a dataset of 1 million bodies (16 MB memory). Figure 2 shows the breakdown of the various optimizations. The code consists of two loops that iterate over all the pairs. We first performed unrolling optimizations to improve ILP, which gives a benefit of 1.4X. The compiler **auto-vectorizes** the code well with no programmer intervention and provides a 3.7X SIMD scaling. We obtained a parallel scaling of 3.1X, which motivates the need for our algorithmic optimization of blocking the data structures to fit in L3 cache (**1-D blocking**, code in Section 4.1). Once blocking is done, we obtain an additional 1.9X thread scaling, and a 1.1X performance gap between compiled and best-optimized code.

2: BackProjection: We back-project 500 images of dimension 2048×2048 pixels onto a $1024 \times 1024 \times 1024$ 3D grid. Backprojection requires 80 ops per grid point. Both the image (16 MB) and volume (4 GB) are too large to reside in cache. Figure 2 shows that we get poor SIMD scaling of 1.2X from auto-vectorization. Moreover, parallel scaling is only 1.8X. This is because the code is **bandwidth-bound** (1.6 bytes/flop). We perform blocking over the 3D volume to reduce bandwidth (**3D blocking** in Figure 2). Due to spatial locality, the image working set reduces accordingly. This results in the code becoming **compute-bound**. However, due to gathers which cannot be vectorized on CPU, SIMD scaling only improved by additional 1.6X (total 1.8X). We obtained additional 4.4X thread scaling (total 7.9X), showing benefits of SMT. The net performance is 1.1X off the best-optimized code.

3: 7-Point 3D Stencil: Application iterates over a 3D grid of points, and performs 8 flops of computation per point. A 3D dataset with $512 \times 512 \times 512$ grid points is used. Figure 2 shows that we get a poor SIMD scaling of 1.8X from auto-vectorization (**bandwidth bound**). In order to improve the scaling, we perform both **spatial** and **temporal blocking** to improve the performance.¹⁷ The resultant code performs *four* time-steps simultaneously, and improves the DLP by a further 1.7X (net SIMD scaling of 3.1X—lower than 4X due to the overhead of repeated computation on the boundary). The thread scaling is further boosted by 2.5X (overall 5.3X). The net performance is within 10.3% of the best-optimized code.

4. Lattice Boltzmann Method (LBM): The computational pattern is similar to the stencil kernel. We used a $256 \times 256 \times 256$ dataset. Figure 2 shows that our initial code (SPEC CPU2006) does not achieve any SIMD scaling, and 2.9X core-scaling. The reason for no SIMD scaling is the AOS data layout that results in gather operations. In order to improve performance, we perform the following **two algorithmic changes**. Firstly, we perform an AOS to SOA conversion of the data. The resultant auto-vectorized code improves SIMD scaling to 1.65X. Secondly, we perform 3.5D blocking. The resultant code further boosts SIMD scaling by 1.3X, achieving a net scaling of 2.2X. The resultant thread scaling was further increased by 1.95X (total 5.7X). However, the

Figure 2. Breakdown of Ninja Performance Gap in terms of Instruction (ILP), Task (TLP), and Data Level Parallelism (DLP) before and after algorithm changes for NBody, BackProjection, Stencil, and LBM. The algorithm change involves blocking.



compiler generated extra spill/fill instructions, that resulted in performance gap of 1.4X.

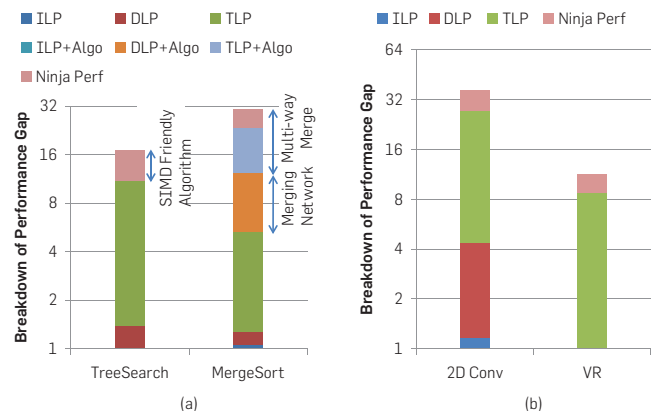
5: LIBOR: LIBOR⁴ has an outer loop over all paths of the Monte Carlo simulation, and an inner loop over the forward rates on a single path. Figure 3 shows performance benefit of only 1.5X from auto-vectorization, since the current **compiler only attempts to vectorize the inner loop**, which has back-to-back dependencies and can only be partially vectorized. To solve this issue, we performed an algorithmic change to convert layout from AOS to SOA. We use the array notations technology to express outer loop vectorization (code in Figure 7b). Performing these changes allowed the outer loop to vectorize and provides additional 2.5X SIMD scaling (net 3.8X). The performance is similar to the best-optimized code.

6. Complex 1D Convolution: We use an image with 12.8 million points, and a kernel size of 8K. The first bar in Figure 3 shows the performance achieved by the unrolling enabled by the compiler, which results in 1.4X scaling. The auto-vectorizer only achieves a scaling of 1.1X. The TLP achieved is 5.8X. In order to improve the SIMD performance, we perform a **rearrangement of data** from AOS to SOA format. As a result, the compiler produces efficient SSE code, and the performance scales up by a further 2.9X. Our overall performance is about 1.6X slower than the best-optimized numbers (inability of the compiler to block the kernel weights).

7. BlackScholes: BlackScholes computes the call and put options together. The total computation is 200 ops, while the bandwidth is 36 bytes. Figure 3 shows a SIMD speedup of 1.1X using auto-vectorization. The low scaling is primarily due to the AOS layout, which results in gather operations. To improve performance, we *changed the data layout* from AOS to SOA. As a result, the auto-vectorizer generated SVML (short vector math library) code, resulting in an increase in scaling by 2.7X (total 3.0X). The net performance is within 1.1X of the best performing code.

8. TreeSearch: The binary tree is laid out in a breadth-first fashion. The auto-vectorizer achieves a SIMD speedup

Figure 3. Breakdown of Ninja Gap for (a) Treesearch and Mergesort and (b) 2D convolution and VR. The benchmarks in (a) require rethinking algorithms to be more SIMD-friendly, while in (b) do not require any algorithmic changes.



of 1.4X (Figure 4a). This is because it operates simultaneously on 4 queries, and each query may traverse down a different path — resulting in a gather operation. In order to improve performance, we perform an **algorithmic change**, and traversed 2 levels at a time (similar to SIMD width blocking¹⁴). However, the compiler did not generate the described code sequence, resulting in a 1.55X Ninja gap.

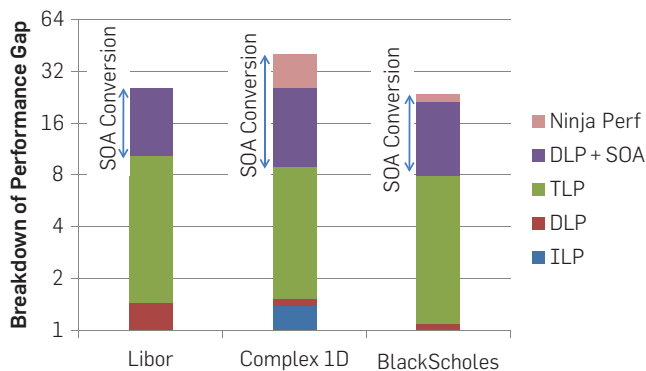
9. MergeSort: Our analysis is done for sorting an input array with 256 M elements. Figure 4a shows that we get a 1.2X scaling from auto-vectorization. This is due to gather operations for merging *four* pairs of lists. Parallel scaling is only 4.1X because the last few merge phases being bandwidth bound. In order to improve performance, we perform the following *two* **algorithmic changes**. Firstly, we implement merging of lists using a merging network⁶ (code in Section 4.1). Secondly, in order to reduce the bandwidth requirement, we perform multiple merge phases together. The parallel scaling of the resultant code further speeds up by 1.9X. The resultant performance is within 1.3X of the best-optimized code.

10. 2D Convolution: We perform convolution of a $2K \times 2K$ image with a 5×5 kernel. The code consists of four loops. Figure 4b shows that we obtained a benefit of 1.2X by loop unrolling. We implemented the two inner loops using the array notations technology. That enabled vectorization of the outer loop, and scaled 3.8X with SIMD width. The thread-level parallelism was 6.2X. Our net performance was within 1.3X of the best-optimized code.

11. Volume Rendering: As shown in Figure 4b, we achieve a TLP scaling of 8.7X (SMT of 1.5X). As far as DLP is concerned, earlier compiler versions did not vectorize the code due to various control-intensive statements. However, recent compilers vectorize the code using mask values for each branch instruction, and using proper masks to execute both execution paths for each branch. There is only a small 1.3X Ninja performance gap.

Summary: In this section, we analyzed each benchmark, and reduced the Ninja gap to within 1.1–1.6X by applying necessary algorithmic changes coupled with the latest compiler technology.

Figure 4. Breakdown of Ninja Performance Gap for Libor, Complex 1D convolution, and BlackScholes. All benchmarks require AOS to SOA conversion to obtain good SIMD scaling.



4. ANALYSIS AND SUMMARY

In this section, we identify the steps taken to bridge the Ninja gap with low programmer effort. The key steps taken are to first perform a set of well-known algorithmic optimizations to overcome scaling bottlenecks, and secondly to use the latest compiler technology for vectorization and parallelization. We now summarize our findings with respect to the gains we achieve in each step.

4.1 Algorithmic Changes

Algorithmic changes do require programmer effort and some insights, but are essential to avoid vectorization issues and bandwidth bottlenecks. Figure 5 shows the performance improvements due to set of well-known algorithmic optimizations that we describe below.

AOS to SOA conversion: A common optimization that helps prevent gathers and scatters in vectorized code is to convert data structures from Array-Of-Structures (AOS) to Structure-Of-Array (SOA). Separate arrays for each field allows contiguous memory accesses when vectorization is performed. AOS structures require gathers and scatters, which can impact both SIMD efficiency and introduce extra bandwidth and latency for memory accesses. The presence of a hardware gather/scatter mechanism does not eliminate the need for this transformation—gather/scatter accesses commonly need significantly higher bandwidth and latency than contiguous loads. Such transformations are advocated for a variety of architectures including GPUs.¹⁹ Figure 5 shows that for our benchmarks, AOS to SOA conversion helped by an average of 1.4X.

Blocking: Blocking is a well-known optimization that can help avoid bandwidth bottlenecks in a number of applications. The key idea is to exploit the inherent data reuse available in the application by ensuring that data remains in caches across multiple uses, both in the spatial domain (1-D, 2-D, or 3-D), and temporal domain.

In terms of code change, blocking involves a combination of loop splitting and interchange. The code snippet in Figure 6a shows an example of blocking NBody code. There are two loops (body1 and body2) iterating over all bodies. The *original code* on the top streams through the entire set of bodies in the inner loop, and must load the body2 value from memory in each iteration. The *blocked code* is

Figure 5. Benefit of three different algorithmic changes to our benchmarks normalized to code before any algorithmic change. The effect of algorithmic changes is cumulative.

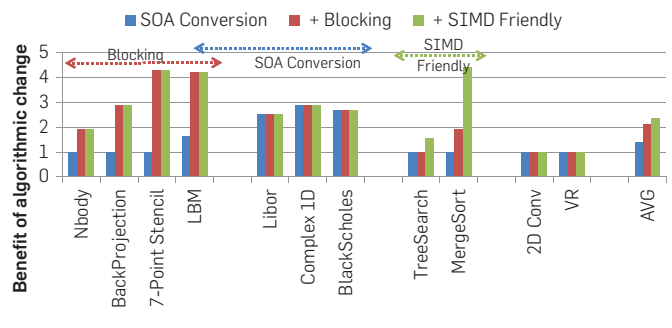
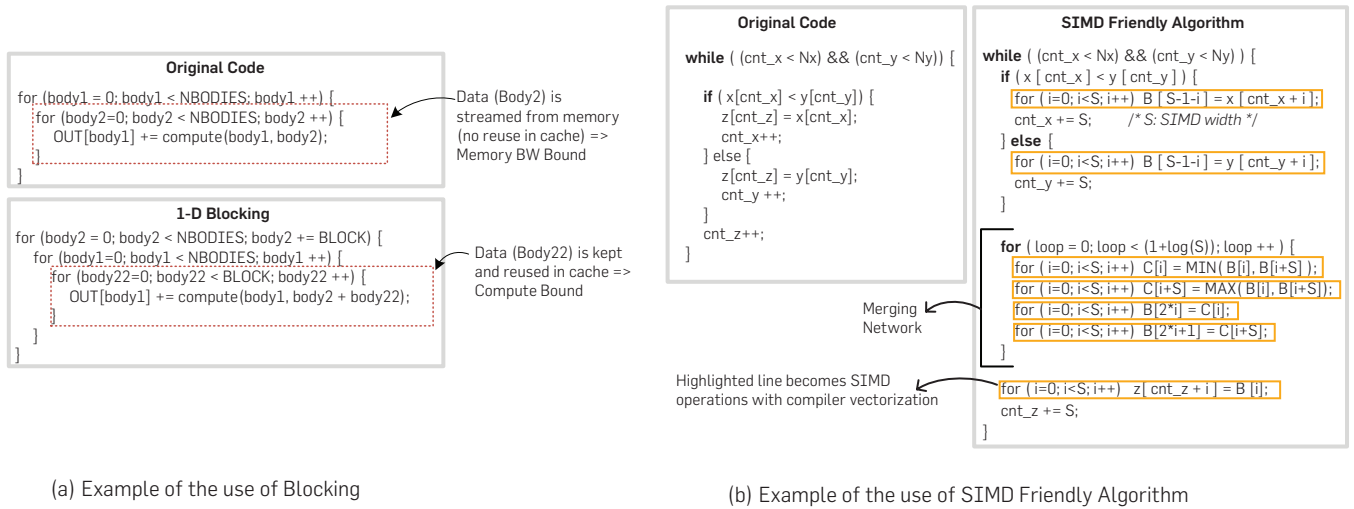


Figure 6. Code snippets showing algorithmic changes for (a) blocking in NBody and (b) SIMD-friendly MergeSort algorithm.



obtained by splitting the body2 loop into an outer loop iterating over bodies in multiple of *BLOCK*, and an inner body22 loop iterating over elements within the block. This code reuses a set of *BLOCK* body2 values across multiple iterations of the body1 loop. If *BLOCK* is chosen such that this set of values fits in cache, memory traffic is brought down by a factor of *BLOCK*. In terms of performance (Figure 5), we achieve an average of 1.6X improvement (up to 4.3X for LBM and 7-point stencil).

SIMD-friendly algorithms: In some cases, the naive algorithm cannot easily be vectorized either due to back-to-back dependencies between loop iterations or due to the heavy use of gathers and scatters in the code. A different algorithm that is more SIMD friendly may then be required. Figure 6b shows an example of this in MergeSort. The code on the left shows the traditional algorithm, where only two elements are merged at a time and the minimum written out. There are back-to-back dependencies due to the array increment operations, and hence the code cannot vectorize. The figure on the right shows code for a SIMD-friendly merging network,⁶ which merges two sequences of SIMD-width *S* sized elements using a sequence of min, max and interleave ops. This code auto-vectorizes with each highlighted line corresponding to one SIMD instruction. However, this code does have to do more computation (by a constant factor of $\log(S)$), but still yields a gain of 2.3X for 4-wide SIMD. Since these algorithmic changes involve tradeoff between total computation and SIMD-friendliness, the decision to use them must be *consciously taken by the programmer*.

Summary: Using well-known algorithmic techniques, we get an average of **2.4X performance gain** on 6-core Westmere. With increasing cores, SIMD width, and compute-to-bandwidth ratios, gains due to algorithmic changes will further increase.

4.2 Compiler Technology

Once algorithmic changes have been taken care of, we show the impact of utilizing the parallelization and vectorization

technology present in recent compilers in bridging the Ninja gap.

Parallelization. We parallelize our benchmarks using OpenMP pragmas typically over the outermost loop. OpenMP offers a portable solution that allows for specifying the number of threads to be launched, thread affinities to cores, specification of thread private/shared variables. Since throughput benchmarks offer significant TLP (typically outer for loop), we generally use a **omp parallel for** pragma. One example is shown in Figure 7a for complex 1D convolution. The use of SMT threads can help hide latency in the code—hence we sometimes obtain more than 6X scaling on our 6-core system.

Vectorization.

SSE versus AVX: Figure 8a shows the benefit from inner and outer loop auto-vectorization on Westmere, once proper algorithmic changes are made. We also compare it to the SIMD scaling for the manual best-optimized code, and show scaling on AVX (8-wide SIMD) in Figure 8b using a 4-core 3.4 GHz Intel® Core i7-2600 K Sandybridge system. We use the same compiler, and only change compilation flags to **-xAVX** from **-xSSE4.2**. In terms of performance, we obtain on average 2.75X SIMD scaling using compiled code, which is within 10% of the 2.9X scaling using best-optimized code. With 8-wide AVX, we obtain 4.9X and 5.5X scaling (again very close to each other) using compiled and best-optimized code.

Our overall SIMD scaling for best-optimized code is good for most of our benchmarks, with the exceptions being MergeSort, TreeSearch and BackProjection. As also explained in Section 4.1, we performed algorithmic changes in MergeSort and TreeSearch at the expense of performing more operations, resulting in lower than linear speedups. Backprojection does not scale linearly due to the presence of unavoidable gathers/scatters in the code. This limits SIMD scaling to 1.8X on SSE (2.7X on AVX) for backprojection.

Inner loop vectorization: Most of our benchmarks

Figure 7. Code snippets showing compiler techniques for (a) Parallelization and inner loop vectorization in complex 1D convolution and (b) Outer loop vectorization in LIBOR. Note that the code changes required are small and can be achieved with low programmer effort.

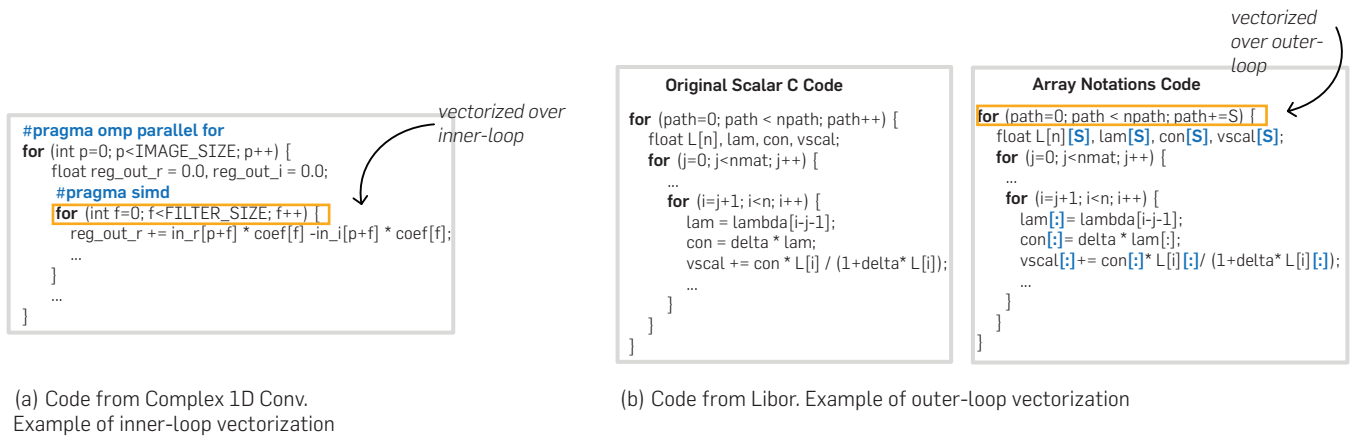
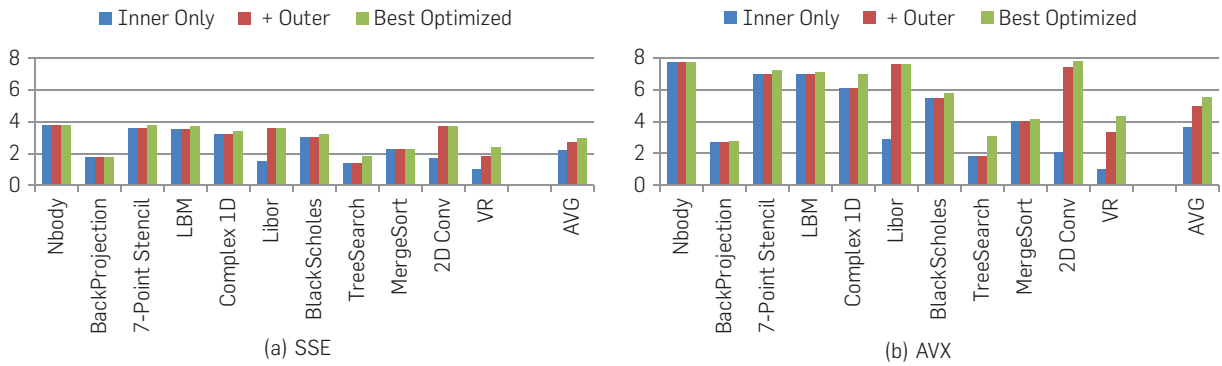


Figure 8. Breakdown of benefits from inner and outer loop vectorization on (a) SSE and (b) AVX. We also compare to the best-optimized performance.



vectorize over the inner loop of the code, either by using compiler auto-vectorization or `#pragma simd` when dependence or memory alias analysis fails. The addition of this pragma is a directive to the compiler that the loop must be (and is safe to be) vectorized. Figure 7a shows an example where this pragma is used. Our average speedup for inner loop vectorization is 2.2X for SSE (3.6X on AVX).

Outer loop vectorization: Vectorizing an outer-level loop is challenging: Induction variables need to be analyzed for multiple loop levels; and loop control flows such as zero-trip test and exit conditions have to be converted into conditional/predicated execution on multiple vector elements. The array notations technology helps the programmer avoid those complications without elaborate program changes. We gain benefits from outer-loop vectorization in LIBOR, where the inner loop is only partially vectorizable. We currently use array notations to vectorize the outer (completely independent) loop (Figure 7b). The scalar code is modified to change the outer loop index to reflect the vectorization, and compute results for multiple iterations in parallel. Note that the programmer declares arrays of size S (simd width), and $X[a:b]$ notation stands for accessing b elements of X , starting with index a . It

is usually straightforward to change scalar code to array notations code. This change results in high speedups of 3.6X on SSE (7.5X on AVX).

5. SUMMARY

Figure 9 shows the relative performance of best-optimized code versus compiler generated code before and after algorithm changes. We assume that the programmer has put in the effort to introduce the pragmas and compiler directives described in previous sections. There is a **3.5X average gap** between compiled code and best-optimized code **before we perform algorithmic changes**. This gap is primarily because of the compiled code being bound by memory bandwidth or low SIMD efficiency. After we perform algorithmic changes described in Section 4.1, this gap shrinks to avg. 1.4X. The only benchmark with a significant gap is TreeSearch, where the compiler vectorizes the outer loop with gathers. The rest show 1.1–1.4X Ninja gaps, primarily due to extra instructions being generated due to additional spill/fill instructions, loads/stores—these are hard problems where the compiler relies on heuristics.

Impact of Many-core Architectures: In order to test the Ninja gap on many-core platforms, we performed the same

experiments on the Intel® Xeon Phi™ “Knights Corner” co-processor (KNC), which has 60/61 cores on a single die, and each core features an in-order micro-architecture with 4-way SMT and 512-bit SIMD unit.

Figure 10 shows the Ninja performance gap for KNC as well as for Westmere. The average Ninja gap for KNC is only 1.2X, which is almost the same (slightly smaller) than the CPUs. The main difference between the two performance gaps comes from TreeSearch, which benefits from the hardware gather support on Xeon Phi, and is close in performance (1.1X) to the best-optimized code.

The remaining Ninja gap after algorithmic changes remains small and stable across KNC and CPUs, inspite of the much larger number of cores and SIMD width on KNC. This is because our algorithmic optimizations focused on resolving vectorization and bandwidth bottlenecks in the code. Once these issues have been taken care of, future architectures will be able to exploit increasing hardware resources yielding stable and predictable performance growth.

6. DISCUSSION

The algorithmic optimizations that we described in Section 4.1 are applicable to a variety of architectures including GPUs. A number of previous publications^{19,21} have discussed the optimizations needed to obtain best performance on GPUs. In this section, we show the impact of the same

algorithmic optimizations on GPU performance. We use the NVIDIA C2050 Tesla GPU for this study.

Although GPUs have hardware gather/scatters, best coding practices (e.g., the CUDA C programming guide¹⁹) state the need to avoid uncoalesced global memory accesses—including converting data structures from AOS to SOA for reducing latency and bandwidth usage. GPUs also require blocking optimizations, which refers to the transfer/management of data into the shared memory (or caches) of GPUs. Finally, the use of SIMD-friendly algorithms greatly benefits GPUs that have a wider SIMD width than current CPUs. The average **performance gain from algorithmic optimizations is 3.8X** (Figure 11)—higher than the 2.5X gain on CPUs, since GPUs have more SMs and larger SIMD width, and hence sub-optimal algorithmic choices have a large impact on performance.

7. RELATED WORK

There are a number of published papers that show 10–100X performance gains over previous work using carefully tuned code.^{1,8,14,17,24} Lee et al.¹⁵ summarized relevant hardware architecture features and a platform-specific software optimization guide on CPU and GPUs. While these works show the existence of a large Ninja performance gap, they do not describe the programming effort or how to bridge the Ninja gap.

In this work, we analyze the sources of the Ninja gap and use traditional programming models to bridge it

Figure 9. Relative performance between the best-optimized code, the compiler-generated code after algorithmic change, and the compiler-generated code before algorithmic change. Performance is normalized to the compiled code after algorithmic change.

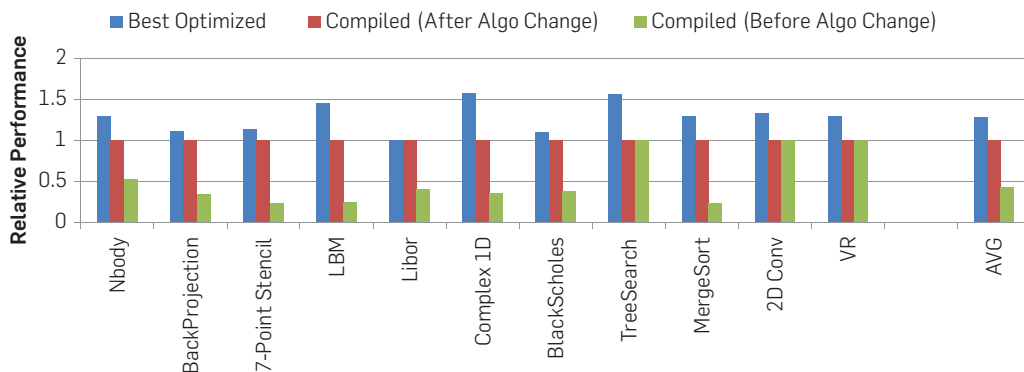


Figure 10. Gap between best-optimized and compiler-generated code after algorithmic changes for Intel® Xeon Phi™ and CPUs.

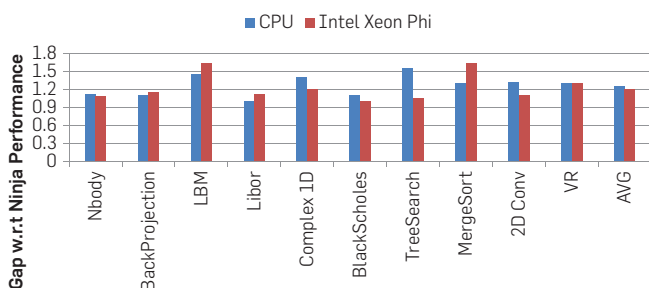
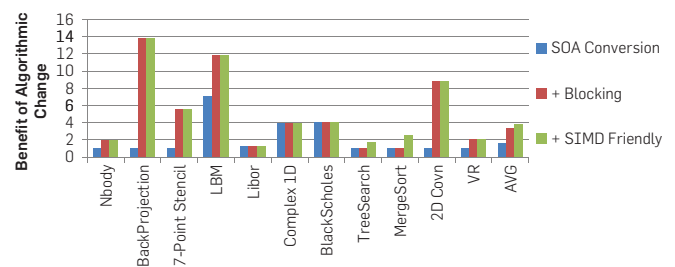


Figure 11. Benefit of the algorithmic changes described in Figure 6 on the NVIDIA Tesla C2050 GPU.




using low programmer effort. A previous version of this paper was published in Satish et al.²³ Production compilers have recently started to support parallelization and vectorization technology that have been published in compiler research. Examples of such technology include OpenMP for parallelization available in recent GCC and ICC compilers, as well as auto-vectorization technology,¹⁸ dealing with alignment constraints and outer loop vectorization. These technologies have been made available using straightforward pragmas and technology like array notations, a part of Intel® Cilk™ Plus.¹¹

However, naively written code may not scale even with compiler support since they are bottlenecked by architectural features such as memory bandwidth, gathers/scatters or because the algorithm cannot be vectorized. In such cases, algorithmic changes such as blocking, SOA conversion and SIMD-friendly algorithms are required. There have been various techniques proposed to address these algorithmic changes, either using compiler assisted optimization, using cache-oblivious algorithms or specialized languages. Such changes usually require programmer intervention and programmer effort, but can be used across a number of architectures and generations. For instance, a number of papers have shown the impact of similar algorithmic optimizations on GPUs.²¹

While our work focuses on traditional programming models, there have been radical programming model changes proposed to bridge the gap. Recent suggestions include Domain Specific Languages (a survey is available at Fowler⁹), the Berkeley View project² and OpenCL for programming heterogeneous systems. There have also been library oriented approaches proposed such as Intel® Threading Building Blocks, Intel® Math Kernel Library, Microsoft Parallel Patterns Library (PPL), etc. We believe these are orthogonal and used in conjunction with traditional models.

There is also a body of literature in adopting auto-tuning as an approach to bridging the gap.^{8, 25} Autotuning results can be significantly worse than the best-optimized code. For, for example, for autotuned stencil computation,⁸ our best-optimized code is 1.5X better in performance. Since our Ninja gap for stencil is only 1.1X, our compiled code performs 1.3X better than auto-tuned code. We expect our compiled results to be in general competitive with autotuned results, while offering the advantages of using standard tool-chains that can ease portability across processor generations.

8. CONCLUSION

In this work, we showed that there is a large Ninja performance gap of 24X for a set of real-world throughput computing benchmarks for a recent multi-processor. This gap, if left unaddressed will inevitably increase. We showed how a set of simple and well-known algorithmic techniques coupled with advancements in modern compiler technology can bring down the Ninja gap to an average of **just 1.3X**. These changes only require low programming effort as compared to the very high effort in Ninja code. 

References

- Arora, N., Shringarpure, A., Vuduc, R.W. Direct N-body Kernels for multicore platforms. In *ICPP* (2009), 379–387.
- Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., et al. *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report UCB/ECS-183, 2006.
- Bienia, C., Kumar, S., Singh, J.P., Li, K. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT* (2008), 72–81.
- Brace, A., Gatarek, D., Musiela, M. The market model of interest rate dynamics. *Mathematical Finance* 7, 2 (1997), 127–155.
- Chen, Y.K., Chhugani, J., et al. Convergence of recognition, mining and synthesis workloads and its implications. *IEEE* 96, 5 (2008), 790–807.
- Chhugani, J., Nguyen, A.D., et al. Efficient implementation of sorting on multi-core simd cpu architecture. *PVLDB* 1, 2 (2008), 1313–1324.
- Daly, W.J. The end of denial architecture and the rise of throughput computing. In *Keynote Speech at Design Automation Conference* (2010).
- Datta, K. Auto-tuning Stencil Codes for Cache-based Multicore Platforms. PhD thesis, EECS Department, University of California, Berkeley (Dec 2009).
- Fowler, M. *Domain Specific Languages*. 1st edn. Addison-Wesley Professional, Boston, MA 2010.
- Giles, M.B. *Monte Carlo Evaluation of Sensitivities in Computational Finance*. Technical report. Oxford University Computing Laboratory, 2007.
- Intel. A quick, easy and reliable way to improve threaded performance, 2010. software.intel.com/articles/intel-cilk-plus.
- Ismail, L., Guerchi, D. Performance evaluation of convolution on the cell broadband engine processor. *IEEE PDS* 22, 2 (2011), 337–351.
- Kachelriebe, M., Knaup, M., Bockenbach, O. Hyperfast perspective cone-beam backprojection. *IEEE Nuclear Science* 3, (2006), 1679–1683.
- Kim, C., Chhugani, J., Satish, N., et al. FAST: fast architecture sensitive tree search on modern CPUs and GPUs. In *SIGMOD* (2010), 339–350.
- Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., et al. Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. In *ISCA* (2010), 451–460.
- T. N. Mudge. Power: A first-class architectural design constraint. *IEEE Computer* 34, 4 (2001), 52–58.
- Nguyen, A., Satish, N., et al. 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In *SC10* (2010), 1–13.
- Nuzman, D., Henderson, R. Multi-platform auto-vectorization. In *CGO* (2006), 281–294.
- Nvidia. *CUDA C Best Practices Guide* 3, 2 (2010).
- Podlozhnyuk, V. Black-Scholes option pricing. *Nvidia*, 2007. http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/BlackScholes/doc/BlackScholes.pdf.
- Ryoo, S., Rodrigues, C.I., Bagsorkhi, S.S., Stone, S.S., Kirk, D.B., Hwu, W.M.W. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *PPoPP* (2008), 73–82.
- Satish, N., Kim, C., Chhugani, J., et al. Fast sort on CPUs and GPUs: A case for bandwidth oblivious SIMD sort. In *SIGMOD* (2010), 351–362.
- Satish, N., Kim, C., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., et al. Can traditional programming bridge the Ninja performance gap for parallel computing applications? In *ISCA* (2012), 440–451.
- Smelyanskiy, M., Holmes, D., et al. Mapping high-fidelity volume rendering to CPU, GPU and many-core. *IEEE TVCG*, 15, 6(2009), 1563–1570.
- Sukop, M.C., Thorne, D.T., Jr. *Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers*, 2006.
- Tian, X., Saito, H., Girkar, M., Preis, S., Kozhukhov, S., Cherkasov, A.G., Nelson, C., Panchenko, N., Geva, R. Compiling C/C++ SIMD extensions for function and loop vectorization on multi-core-SIMD processors. In *IPDPS Workshops* (Springer, NY, 2012), 2349–2358.

Nadathur Satish, Mikhail Smelyanskiy, and Pradeep Dubey, Parallel Computing Lab, Intel Corp.

Changkyu Kim, Google Inc.

Jatin Chhugani, Ebay Inc.

Hideki Saito, Rakesh Krishnaiyer, and Milind Girkar, Intel Compiler Lab, Intel Corp.

ACM Transactions on Reconfigurable Technology and Systems



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

Tallinn University of Technology (TUT), Estonia Full Professor in Information Society Technologies

Job description

Faculty of IT of Tallinn University of Technology (TUT), Estonia, seeks to fill the position of a Full Professor in Information Society Technologies (tenure-track position with dedicated funding from the Estonian Government).

The selected candidate will have an opportunity to join an effort by a multi-disciplinary research team in building up the e-Governance Competence Centre at TUT. We expect from the person fulfilling the new position the integration and advancement of technologies utilized by information society and e-government and also teaching and curricula development in that field. In building up the Competence Centre, our research team collaborates with the Estonian governmental ICT agencies. TUT runs also the International Master's Program in E-Governance Technologies and Services.

Estonia is a worldwide leader in digital society and e-government. The country is known as a pioneer in introducing digital services such as e-voting, e-signature, and e-taxation, and more recently, e-residency for foreigners.

TUT is a key enabler of e-Estonia. It is a public university established in 1918 with over 2000 staff and 14000 students. The Faculty of IT has more than 150 academic staff members and more than 2500 students.

Requirements

We expect from the successful candidate:

- ▶ A PhD in information systems, computer science, informatics, or the equivalent, and an active interest in e-governance.

- ▶ A proven track record of the highest quality research in relevant areas.
- ▶ A proven experience with industrial or e-governance applications.
- ▶ Excellent teaching skills and experience and a strong vision on academic education.
- ▶ Willingness to participate in curricula development.
- ▶ The vision and competences to initiate and manage research projects and supervise staff members and students.
- ▶ A proven track record in acquisition of research funds, a strong network and internationally recognized reputation.
- ▶ Successfully supervised doctoral students are an advantage.

Responsibilities

We are looking for an ambitious and inspiring person successfully fulfilling the following responsibilities:

- ▶ Applying information society technologies in areas such as information systems, cybersecurity, open data management, sociotechnical systems, public administration, and regulation.
- ▶ Teaching and designing courses in MSc curricula and particularly in the International MSc Curriculum in E-Governance Technologies and Services, and teaching courses at the PhD level.
- ▶ Preparing and running R&D projects funded by the EU and by the Estonian Research Agency and Enterprise Estonia.
- ▶ Contributing to consolidation and expansion of the department's international network via joint research projects with industry, governmental organisations, and universities.
- ▶ Supervising MSc and PhD students and post-doctoral researchers.

Information about the application process:
ttu.ee/itpositions

All documents must be submitted by August 15, 2015.

University of Texas at Tyler Assistant Professor, Computer Science

Applications are invited for Assistant Professor of Computer Science which is housed within the College of Business and Technology at the University of Texas at Tyler. The Department offers the BS and MS in Computer Science and the BS in Computer Information System.

Requirements include a Doctorate in CS or CIS or closely related field. All areas of computer science will be considered.

Send a letter of interest, summary of qualifications and a resume/CV in a single file to Dean James Lumpkin at kgaribaldi@uttyler.edu. Evaluation will begin immediately and continue until the position is filled. The University of Texas at Tyler is an Equal Opportunity Employer.

**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: 20th of the month/2 months prior to issue date. For latest deadline info, please contact: acmm mediasales@acm.org

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at: <http://jobs.acm.org>

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**



Dennis Shasha

DOI:10.1145/2743036

Upstart Puzzles

Strategic Friendship

CONSIDER THE FOLLOWING game (first posed to my close friend Dr. Ecco) played among several entities. Each entity E_i has a certain force F_i and a certain wealth W_i . A coalition of one or more entities has a combined force equal to the sum of the force of the individual entities. If a coalition C_1 has a force that exceeds the force of a coalition C_2 , and C_1 attacks C_2 , then C_2 is eliminated, and the wealth of the entities making up C_2 is distributed equally among the coalition members of C_1 , but the force of the coalition members in C_1 does not change. Note every member of a coalition must agree to attack for an attack to take place. If the force of C_1 is less than the force of C_2 , and C_1 attacks C_2 , then C_1 is eliminated. This will never happen, however, because we assume every entity wants to survive and increase its wealth. If the force of C_1 is equal to the force of C_2 , then an attack has no effect.

Starter warm-up 1. Suppose there are only two entities— E_1 and E_2 —and $F_1 > F_2$. What happens then?

Solution. E_1 attacks E_2 and takes its wealth; there is indeed no charity in this world.

1. Assume there are three entities— E_1 , E_2 , E_3 —with force 5, 4, 3 and wealth

10, 10, and 100, respectively. What then? See the figure here for a hint.

2. What would happen if there were three entities— E_1 , E_2 , E_3 —each with the same force, say, 2, but with wealth 1, 2, 3, respectively? How does wealth influence outcome?

3. What would happen if there were four entities— E_1 , E_2 , E_3 , E_4 —with force 5, 4, 3, 6 and wealth 10, 10, 12, and 20, respectively?

Stability among entities sometimes depends on wealth, as we have seen, but also on the willingness of an entity to take risk. Suppose there are four entities, each with force 1 and wealth 6. If, say, E_1 , E_2 , and E_3 form a coalition to defeat E_4 , they divide E_4 's wealth equally, but then one of them will be the target of the other two, based on their self-interest. We say an entity E is “risk ready” if it is willing to agree to an attack that might later expose E to an attack. Otherwise, we say E is “risk averse.”

The general upstart question is, given a configuration of risk-ready entities, no two of which have the same wealth, how would you test for its stability? If unstable, devise a formula or an algorithm to determine a stable configuration that can be reached and

has the property that the survivors between them would gain as much wealth from the vanquished as possible. Demonstrate any properties—uniqueness, comparison of total force before and after—that strike you as interesting.

For clever reader solutions to this, as well as to other, upstart challenge, see <http://cs.nyu.edu/cs/faculty/shasha/papers/cacmpuzzles.html>

Solutions that show what happens in this strategically unforgiving world:

1. Nothing happens because E_2 and E_3 form a coalition; E_1 never chooses to attack. E_3 never allows that coalition to attack E_1 , because once E_1 goes away, E_3 loses to E_2 . Similarly, E_2 never attacks E_3 while E_1 is still around, because without E_3 , E_2 would lose to E_1 . This configuration is “stable.”

2. Most likely, E_1 and E_2 would form a coalition to attack E_3 . When they do, the resulting configuration is stable.

3. There are several possibilities, because any three entities here would form a stable configuration, whereas no two entities are stable. But E_4 is the most attractive target due to its wealth. Any two of E_1 , E_2 , and E_3 could defeat E_4 , but most likely three are needed to defeat E_4 . Do you see why?

4. E_3 would then form a coalition with E_4 from the start, because if E_4 would be vanquished, then E_3 would definitely be next. ■

Despite a tempting target, a stable outcome.

If E_3 is eliminated, then E_2 will fall to E_1 ;

if E_1 is eliminated, then E_3 will fall to E_2 ;

and if E_2 is eliminated, then E_3 will fall to E_1 .

This configuration is stable, even though E_3 is such a tempting target.



E_1 ,
force 5,
\$10



E_2 ,
force 4,
\$10



E_3 ,
force 3,
\$100

All are invited to submit solutions and prospective upstart-style puzzles for future columns to upstartpuzzles@cacm.acm.org

Dennis Shasha (dennisshasha@yahoo.com) is a professor of computer science in the Computer Science Department of the Courant Institute at New York University, New York, as well as the chronicler of his good friend the omnihurist Dr. Ecco.

Copyright held by author.
Publication rights licensed to ACM. \$15.00

Blog Ubiquity

INFORMATION EVERYWHERE



The newest ACM forum.

Contributions that cover the vast information-rich world where computing is embedded everywhere.

ACM's *Ubiquity* is the online magazine oriented toward the future of computing and the people who are creating it.

We invite you to participate: leave comments, vote for your favorites, or submit your own contributions.

Captivating topics.

*Net Neutrality
and the Regulated
Internet*

*The End of Life
As We Know It*

*A Shortage of
Technicians*

*The Fractal
Software
Hypothesis*

*Your Grandfather's
Oldsmobile—NOT!*

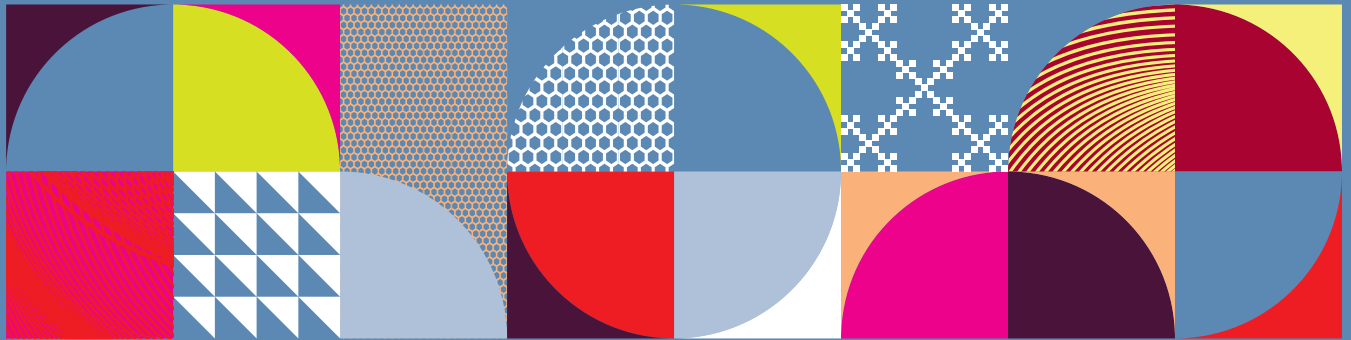
*Superscalar
Smart Cities*



Association for
Computing Machinery

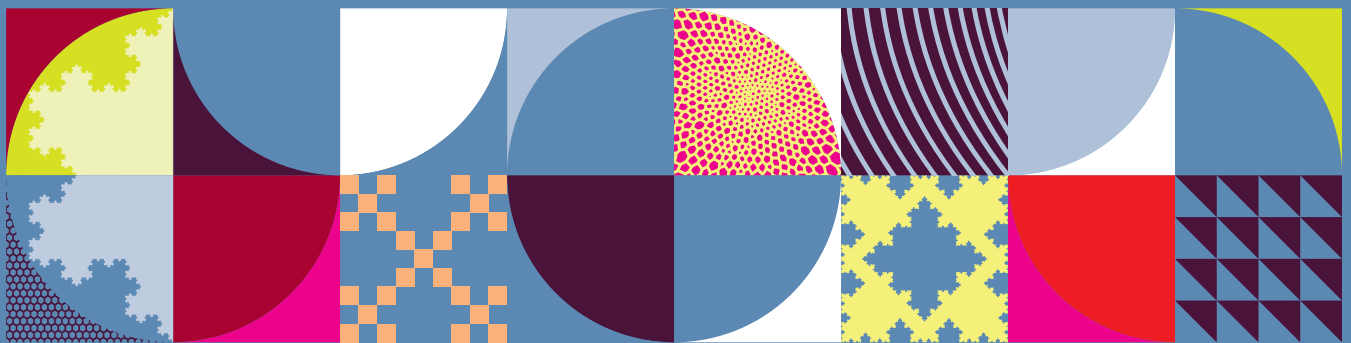
Visit us at
<http://ubiquity.acm.org/blog/>

Comingle with the diverse fabric of the computer graphics and interactive techniques community and discover an exhilarating patchwork of innovation that inspires powerful creation.



SIGGRAPH 2015

Xroads of Discovery



9-13 August 2015



Los Angeles Convention Center



Sponsored by ACM SIGGRAPH

The 42nd International Conference and Exhibition
on Computer Graphics and Interactive Techniques

s2015.siggraph.org

REGISTER TODAY!