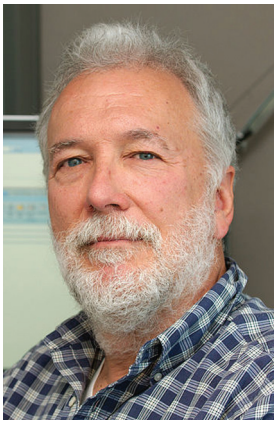

Reflections from the 2013 Eckert-Mauchly Award Recipient

James Goodman
University of Auckland

I thank Editor in Chief Erik Altman and Associate Editor in Chief Lieven Eeckhout for inviting me to share my experiences leading to the Eckert-Mauchly Award. Nobody wins this award on his own (sadly, this gender reference is correct: it has been awarded 35 times, all to men). Many people contributed to the achievements that led to the award, and my biggest fear in writing this is that for lack of space I will leave too many unrecognized.

A meandering path

Coming of age at the beginning of the Vietnam War, I spent my twenties simultaneously protesting the war and supporting it by assisting the U.S. Navy in deploying computers and navigation systems. This deep internal conflict had many consequences and affected important decisions later in life. But at Texas Instruments I learned a lot about how real computers worked from Tom Stringfellow, Quitman Liner, and many others.

Working as an engineer while a graduate student in the 1970s, I was involved in the development of third-party memory systems attaching to IBM mainframes. I was fortunate that upstart Intel allowed me a 30-hour work week during more than five years' employment. I learned about memory systems by examining detailed designs of third-party memory systems for the IBM System/370 family, struggling particularly with the problems of how to hold off a processor designed for core memory or static RAM when a DRAM required refresh and how to extend a transaction look-aside buffer to support a larger memory than the manufacturer had intended. I learned much about caches from "Mr. Cache," Alan J. Smith at the University of California, Berkeley. And I learned a lot about how to conduct research from my advisor Al Despain, along with Carlo Sequin and David Patterson, the last of whom has provided me with sage advice throughout my career, despite being three years my junior.

Near the end of my studies at Berkeley, as Intel shifted emphasis from memory chips to microprocessors, I moved to the microprocessor group, where the P1 and P2 (80186 and 80286) were being designed. The state of the art was the single-board computer, with multiple processors sharing their memory by communicating through a backplane Multibus. Although computers costing more than a half-million dollars had caches, the microprocessors did not, but Intel understood Moore's law. Extending the single-board model to computers with caches led directly to the cache coherence problem, which I discussed with Jack Klebanoff, leading me to think about using Multibus broadcast commands to keep the caches consistent. Thus, I got an early start on the problems of cache consistency for the coming generation of microprocessors capable of shared-memory multiprocessing.

At the age of 36, with three degrees in engineering but not yet having authored a published paper, I completed a PhD and took a position as assistant professor in the Department of Computer Sciences at the University of Wisconsin–Madison. Having investigated architectural support for databases in my dissertation, I had listed research interests as "computer architecture and databases." I was blessed with a colleague, David DeWitt, who tactfully advised me that I was more likely to succeed in architecture than databases. Over the next two decades, I enjoyed a rich and productive environment with colleagues Andy Pleszkun, Jim Smith, Guri Sohi, Mark Hill, and David Wood. It was my natural inclination to work on "small science" projects, in part because big science seemed inaccessible to one determined to eschew support from military sources. My embrace of small science also persuaded me that not every interesting idea was worth publishing, and that I would best succeed by sifting and winnowing ideas before publishing them. Perhaps this was less about great insight than simple laziness, but to this day my list of publications over 45 years is barely one per year.

In early collaboration with Jim Smith and Andy Pleszkun on a "decoupled architecture," I learned a lot about compilers and simulation when Honesty Young designed an early compiler for PIPE, a decoupled architecture.[1] Within

0272-1732/14/$31.00 © 2014 IEEE Published by the IEEE Computer Society

the same project, Wei-chung Hsu recognized that conflicts between register allocation and code scheduling could be handled best by doing both at once, resulting in work[2] I'm very proud to claim despite my minimal contribution.

Working with Mary Vernon and me, Steve Scott did some excellent work evaluating the Scalable Coherent Interface (SCI) ring.[3] Working with IEEE standards committees (Futurebus and SCI) over the next 10 years, I learned a lot about cache coherence protocols, and with Stefanos Kaxiras, developed a strong belief that such protocols could be extended without falling back to a scalable—but slow— directory-based scheme. Gradually it emerged that caches were highly effective for the sharing of data, particularly if things could get out of order, but that locks and critical sections could exhibit disastrous memory behavior for cache-based memory systems.

Steve Scott also came up with the brilliant notion of pruning caches, exploring the novel concept of a distributed directory (or cache) that remembered regions of the network where a line was *not* cached. The concept has since come up repeatedly in my work toward scalable, non-directory-based cache consistency, but this work is rarely referenced, perhaps because it ended up in an IEEE journal[4] after multiple conference rejections.

I delved into locks and memory ordering, working with Phil Woest and Mary Vernon to propose the concept of building hardware queues to avoid many of the problems associated with spinlocks. We initially called this Queue-on-Sync-Bit (QOSB, pronounced "Cosby"), but soon renamed it Queue-on-Lock-Bit ("Colby," after the Wisconsin town responsible for a common cheese). This work[5] inspired Michael Scott and John Mellor-Crummey to propose the popular MCS lock, a software-built queue that captured much of the benefit of QOLB.[6] Meanwhile, Alain Kagi and Doug Burger analyzed the potential for QOLB,[7] concluding that it could be effective, but required sophisticated and disciplined programming, as if programming SMPs wasn't hard enough. With Guri Sohi arguing that speculation could be exploited in many ways, Alain had the insight that hardware could deduce when lock contention was occurring, creating "Implicit QOLB," a queue similar to that of QOLB but without assistance from the programmer.[8] A key notion was that performance could be improved by delaying a response to a request for a cache line containing a lock, allowing the thread holding the lock a brief opportunity to complete execution of the critical section.

Once we recognized the benefit of speculation regarding critical sections, other ideas quickly followed. For example, recognizing that certain memory locations could be associated with a given lock suggested the possibility of *speculative push,* passing cache lines modified within the CS at the time the now-available lock was replaced.[9]

Ravi Rajwar extended the notion of speculating about critical sections one step further, recognizing that a CS without data conflicts need not acquire the lock and therefore can share the cache line containing the lock, permitting concurrent execution of critical sections protected by a common lock.[10] I only realized how counterintuitive this was when I described it to knowledgeable colleagues who initially insisted this was impossible since the programmer explicitly invokes mutual exclusion.

I'm delighted to claim partial credit for this breakthrough, though my primary contributions were presenting Ravi with the context and insisting—over his objections—on calling it Speculative Lock Elision (SLE). Like many brilliant insights, this seems obvious in retrospect, and after we disclosed the idea in 2000 I expected it would soon appear in new processors. It soon appeared in software implementations with limited hardware support in Azul Systems, and experimental software-only versions have been widely discussed. Sun Microsystems introduced support for software-hinted lock elision in their experimental processor Rock, through instructions that explicitly begin speculative execution rather than acquire the lock. But only in the past two years—a full decade later—has this capability appeared in commercial products.

After a sabbatical in 2000 to 2001 at Intel, I discovered the frustration of company secrecy preventing the disclosure of interesting new ideas. Herbert Hum and I conceived a novel cache-coherence protocol intended to exploit the higher bandwidth opportunities present with the emerging transmitter equalization (pre-emphasis clocking) technology, and further increasing the speed advantage of point-to-point networks over buses. The starting point was the notion of a broadcast coherence protocol, with the introduced problem being event ordering, conveniently avoided by a bus. Although our original MESIF Coherence Protocol evolved some before becoming a critical part of QPI source snooping in Nehalem, we were initially prohibited by Intel from publishing the idea, then had it rejected twice by ISCA[11,12] because of limits on what we could disclose.

In 2003, after 23 years at Wisconsin and with an empty nest, I took up a position in computer science at the University of Auckland in New Zealand. Fuad Tabba and I collaborated extensively with Mark Moir on transactional memory issues, and Fuad experimented with a Rock prototype, demonstrating that a hybrid TM system—best-effort hardware support for transactions, falling back to software when necessary—is a promising approach to supporting transactional memory.[13]

Considering the incredible advances made in computer architecture over my career, it is easy to suggest that architecture thrived because of Moore's law and dies with it—where could it possibly go from here? But we still don't know how to build general-purpose parallel computers that are easy to program. I believe there are yet many opportunities to contribute to the goal of truly scalable systems that can be programmed by unsophisticated users.

Computer Architecture lives! MICRO

References

1. H.C. Young and J.R. Goodman, "A Simulation Study of Architectural Data Queues and Prepare-To-Branch Instruction," *Proc. IEEE Int'l Conf. Computer Design (ICCD): VLSI in Computers*, 1984, pp. 544-549.
2. J.R. Goodman and W.-C. Hsu, "Code Scheduling and Register Allocation in Large Basic Blocks," *Proc. 2nd Int'l Conf. Supercomputing*, 1988, pp. 442-452.
3. S.L. Scott, J.R. Goodman, and M.K. Vernon, "Performance of the SCI Ring," *Proc. 19th Ann. Int'l Symp. Computer Architecture* (ISCA 92), 1992, pp. 403-414.
4. S.L. Scott and J.R. Goodman, "Performance of Pruning-Cache Directories for Large-Scale Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 5, 1993, pp. 520-534.
5. J.R. Goodman, M.K. Vernon, and P.J. Woest, "A Set of Efficient Synchronization Primitives for a Large-Scale Shared-Memory Multiprocessor," *Proc. 3rd Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 1989, pp. 64-75.
6. J.M. Mellor-Crummey and M.L. Scott, "Synchronization without Contention," *Proc. 4th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 1991, pp. 269-278.
7. A. Kägi, D. Burger, and J.R. Goodman, "Efficient Synchronization: Let Them Eat QOLB," *Proc. 24th Ann. Int'l Symp. Computer Architecture* (ISCA 97), 1997, pp. 170-180.
8. R. Rajwar, A. Kägi, and J.R. Goodman, "Improving the Throughput of Synchronization by Insertion of Delays," *Proc. 6th Int'l Symp. High Performance Computer Architecture*, 2000, pp. 168-179.
9. R. Rajwar, A. Kägi, and J.R. Goodman, "Inferential Queueing and Speculative Push," *Int'l J. Parallel Processing*, vol. 32, no. 3, 2004, pp. 225-258.
10. R. Rajwar and J.R. Goodman, "Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution," *Proc. 34th Ann. ACM/IEEE Int'l Symp. Microarchitecture*, 2001, pp. 294-305.
11. J.R. Goodman and H.H.J Hum, *MESIF: A Two-Hop Cache Coherency Protocol for Point-to-Point Interconnects (2004)*, tech. report, Univ. of Auckland, https://researchspace.auckland.ac.nz/bitstream/handle/2292/11593/MESIF-2004.pdf?sequence=7.
12. J.R. Goodman and H.H.J. Hum, *MESIF: A Two-Hop Cache Coherency Protocol for Point-to-Point Interconnects (2009)*, tech. report, Univ. of Auckland, https://researchspace.auckland.ac.nz/bitstream/handle/2292/11594/MESIF-2009.pdf?sequence=6.
13. F. Tabba et al., "NZTM: Non-Blocking Zero-Indirection Transactional Memory," *Proc. 21st Ann. Symp. Parallelism in Algorithms and Architectures* (SPAA 09), 2009, pp. 204-213.

James Goodman is a professor in the Department of Computer Science at the University of Auckland and an Emeritus Professor in the Computer Sciences Department at the University of Wisconsin-Madison. Contact him at goodman@cs.wisc.edu.