

Milo M. K. Martin

Research and Teaching Statement

May 17, 2009

Modern computing systems are amazingly complicated artifacts. Today's hardware and software systems combine billions of transistors and millions of lines of code to form the computational and communication infrastructure critical for commerce, government, education, and entertainment. Because of the increasing sophistication of these computing systems, creating correct, high-performance, robust, and secure systems has become an almost insurmountable challenge. One result of this trend is that the design and validation of both hardware and software is becoming more expensive and taking longer, yet buggy and insecure systems are still widespread. With the advent of ubiquitous multi-core chips, these systems are increasingly parallel and concurrent, adding design and validation complexity to already complex systems.

Research agenda. My research focus is on practical techniques to meet the challenge of creating multi-core hardware and software systems that are correct, fast, robust, and secure. Towards this goal, I have been exploring two primary research thrusts to address different hardware and low-level software aspects of this larger problem. The first thrust is **making multicore programming easier**, which includes three concrete projects: (1) exploring the semantics and simple hardware implementation of transactional memory [1, 2, 3, 4], (2) programmer-friendly, adaptive, latency-tolerant, and verifiable multicore hardware architectures [5, 6, 11, 12, 14], and (3) methods for architecture-aware verification of multithreaded software [7, 8]. The second primary thrust is exploring hardware/software approaches for **safe and secure low-level programming via memory-safe implementations of the C/C++ programming languages** [9, 13]. The goal is to make C/C++ as safe and secure as modern languages such as Java, thus detecting memory usage errors and preventing all memory-based security vulnerabilities such as buffer overflows. In addition to these two main research thrusts, my other projects include developing micro-architectural techniques for scalable handling of in-window memory operation re-ordering [15, 16, 17], exploring architectural implications of terabit networking [18], and demonstrating and detecting malicious hardware [10].

Educational agenda. The final way in which I am tackling these problems is by exposing the next generation of computer scientists and engineers to the techniques and approaches needed to tackle the increasing sophistication and parallel nature of modern computing system. As such, my educational focus on techniques for dividing and conquering complexity, on the art of designing simple and robust systems, and on experimental thinking is closely aligned with these research goals.

Collaborators. This overall agenda has led to projects that cross traditional boundaries, which in turn has resulted in collaborations with researchers in formal verification (Rajeev Alur), compilers (E Lewis), programming languages (Steve Zdancewic), networking (Jonathan Smith), systems security (Sam King of University of Illinois), and with computer architecture faculty at Penn (Amir Roth) and elsewhere (Tom Wenisch of University of Michigan).

Indications of research impact. Since joining Penn, my research results have been published in all of the most prestigious venues in computer architecture (ISCA, MICRO, ASPLOS, HPCA, and IEEE Micro's annual "Top Picks") and in top venues in compilers (PLDI) and formal verification (CAV). I've been selected to serve on six program committees for top-tier publication venues (ISCA 2007 & 2008, MICRO 2007 & 2009, ASPLOS 2010, Top Picks 2009). My research has been financially supported by NSF (including a CAREER award), DARPA, industrial grants from Intel, equipment donations from Sun Microsystems, and a highly selective IBM Graduate Fellowship received by one of my PhD students. I have presented my research at leading companies and top universities in the field. Distributions of my software tools have been used in over fifty academic papers and by thousands of researchers worldwide. Overall, I have published two dozen papers in highly selective venues. According to Google Scholar, my publications have collectively received over one thousand citations.

Research

A. Easier Multicore Programming

A.1 Transactional Memory

Hardware-based transactional memory has great potential for simplifying multithreaded programming by allowing programmers to specify regions of the program that must appear to execute atomically and serially, but actually execute concurrently.

Implications of transactional semantics. Although transactional constructs are promising, their semantics can be surprisingly subtle. My research has explored some of these subtle semantic issues of transactional memory [2, 3], showing that the same atomic guarantees that give transactions their power also have unexpected side effects. For example, deadlock may result when directly translating lock-based critical sections into transactions [2] or sequentially composing transactional code [3]. This work was the first to discuss the interaction between non-transactional code and code within transactions. We introduced the terms *weak atomicity* and *strong atomicity*, which are now standard terminology, and sparked a vibrant debate on what is the right tradeoff between strong semantics and efficient implementation. This work has been widely cited (well over a hundred citations according to Google Scholar) in both the hardware and software transactional memory communities.

Challenge: implementing hardware-based transactional memory. In addition to semantic challenges, hardware-based transactional memory presents significant implementation roadblocks. The basic hardware mechanisms for the common-case execution of transactions (log-based version management and cache-based conflict detection) are well-understood, simple, effective, and being rapidly incorporated into industrial designs (*e.g.*, Azul’s Vega and Sun’s Rock). However, these basic mechanisms alone are insufficient when transactions are large (*i.e.*, touch more data than fits in the data cache), long (*i.e.*, execute for longer than a scheduling quantum), or irrevocable (*i.e.*, perform arbitrary system calls or I/O). For transactional memory to fulfill its potential as a general-purpose synchronization mechanism, it must handle such cases correctly, efficiently, and without unduly complicating the hardware.

ONETM. We have developed a simple low-overhead hardware approach for providing unrestricted transactional execution (*i.e.*, transactions of unbounded size and duration and can contain I/O and system calls). Our system, ONETM [1], has two synergistic components. First, we make the fast case of non-overflowed transactions more common by exactly tracking the block’s permissions (just a few bits) *without* the overhead of caching the block’s data (hundreds of bits) [1]. This hundred-fold increase in encoding efficiency allows transactions to grow significantly larger before triggering an overflow. Second, we aim to provide a simple implementation for handling the remaining uncommon cases by introducing the concept of a singular “unrestricted” transaction that can overflow and perform I/O and system calls, and it does so without fully serializing the system. When a transaction overflows its cache or attempts to perform an irrevocable operation, it transitions to an unrestricted mode of transactional execution. Only one transaction is allowed to execute in this mode, but non-transactional code and normal-mode transactions are allowed to continue to execute concurrently [1]. Transactions in this mode of execution win all conflicts, and thus never abort due to conflicts. This non-abortable nature allows the transaction to perform I/O and arbitrary system calls without serializing the system [4].

This work performed in collaboration with Dr. E Lewis and students Colin Blundell and Joe Devietti. This work forms the foundation of my NSF CAREER award and Colin Blundell’s PhD dissertation.

A.2 Programmer-Friendly and Designer-Friendly Multicore Architectures

Data sharing and synchronization-induced memory consistency overheads are significant impediments to the creation of efficient and scalable multicore programs. Furthermore, the performance-opacity of today’s multicore processors leads to fragile performance, impedes scaling, makes performance analysis and tuning notoriously difficult, and generally places a greater burden on software developers.

To lighten the burden on multicore software developers, we strive to create multicore systems that are *programmer friendly*. Specifically, we have explored: (1) minimizing the performance penalties of data sharing via low-latency cache coherence protocols [12, 14], (2) providing robust performance by dynamically learning sharing patterns and adapting to interconnection network congestion [14], and (3) tolerating the latency impact of memory consistency on stores, memory fences, and atomic operations using deep beyond-the-window speculation [5].

These same architectural proposals are also *designer friendly* in that they explicitly strive for elegant simplicity. For example, our proposals identify clearly stated safety [12, 14] and forward progress invariants [14], explore the verifiability of our proposals [6, 11, 12], and advocate the decoupling of coherence from consistency [5, 11]. More specifically, TOKENCMP [12] proposed a multi-level cache coherence protocol with the same verification and design complexity as a simpler single-level protocol. *Token tenure* [14] is a simpler and more scalable mechanism for ensuring forward progress in token counting protocols. Our latency-tolerant INVISIFENCE design [5] employs carefully designed deep speculation mechanisms, which use efficient block-granularity tracking of speculative state and provide single-cycle commit and abort (which in turn further simplifies the design by obviating the need for multiple in-flight checkpoints).

A.3 Architecture-Aware Verification of Multicore Software

Modern hardware architectures allow the reordering of memory operations in ways that are visible to software running on other cores in a multicore system. The exact reorderings allowed are defined by the system's memory consistency model. In well-synchronized programs, such reorderings are hidden by explicit synchronization. However, in highly optimized concurrent data structures (such as lock-free data structures), the programmer must explicitly prevent such reorderings by inserting memory ordering fences. Because such concurrent code is notoriously difficult to reason about, we have developed a method for checking bounded test cases for concurrent data structures under weakly-ordered memory consistency models [7]. Our CHECKFENCE [8] work further developed this approach by checking C source code, employing specification mining, and using range analysis. CHECKFENCE found several bugs in previously published algorithms. This work resulted from an on-going collaboration with Prof. Rajeev Alur, and it formed the basis of Sebastian Burckhardt's PhD dissertation.

B. Secure and Safe Low-Level Programming via Hardware Support for Memory-Safe C

Challenge: making C as secure and safe as Java. Many of today's security vulnerabilities and memory corruption bugs stem from a fundamental flaw in the C programming language: its lack of memory accesses checking. Modern languages such as Java avoid such problems by enforcing memory safety. Unfortunately, most low-level systems code that exists today is written in C. However, retrofitting C code with the same memory safety guarantees is challenging because of C's pointer arithmetic, sub-object pointers, arbitrary casts, and programmer-visible memory layout, and conflation of array and non-array pointers. As a result, prior proposals either suffer from high performance overhead, require program source code modifications, have incompatible data layout, or are incomplete.

HardBound: hardware-assisted enforcement of spatial safety. We first focused on hardware support to provide complete spatial safety (bounds checking of array and pointer accesses) for C programs while avoiding the performance and compatibility limitations mentioned above. Our proposal, HARDBOUND [9], adds direct hardware support for a bounded pointer primitive. The compiler inserts code to establish the initial bounds of pointers, and the hardware does the rest. HARDBOUND is fast (with less than a 10% slowdown) because it provides dedicated hardware to propagate and check the base/bound metadata. To avoid memory layout incompatibilities, HARDBOUND transparently records of all the base/bound metadata in a disjoint shadow space. Finally, HARDBOUND uses metadata encoding tricks (*e.g.*, using fewer metadata bits for the common case of small objects) to reduce memory overhead.

SoftBound: highly compatible software-only spatial safety. Based on our experiences with HARDBOUND, we realized that the same techniques could be applied even *without* hardware support. This realization led to our development of SOFTBOUND [13], a software-only compiler transformation that inherits many of the same advantages of HARDBOUND. Unlike the in-line “fat” pointer encoding used by prior software-only pointer-based approaches, SOFTBOUND’s key contribution is its use of disjoint base/bound metadata to both preserve memory layout compatibility and shelter the metadata from corruption by arbitrary type casts. Without HARDBOUND’s extensive hardware support, SOFTBOUND’s runtime overheads are higher. However, SOFTBOUND’s runtime overhead is low enough (less than 2x) for use during all stages of software development and testing. Furthermore, if configured to check only memory writes (the main culprit for security vulnerabilities), the overhead is less than 25% on average, which is likely low enough for use when deploying security-critical software.

Next steps. We are currently investigating the addition of special-purpose instructions to create an intermediate design point with the performance efficiency of HARDBOUND with less invasive hardware changes. We are also exploring similar metadata approaches for detecting temporal errors. This work is being done in collaboration with Prof. Steve Zdancewic and graduate students Joe Devietti, Colin Blundell, Santosh Nagarakatte, and Jianzhou Zhao.

C. Other Research

Scalable Processor Microarchitectures via Streamlined Memory Forwarding

To obtain higher performance, modern processors allow speculative reordering of memory load and store operations by employing power-hungry and non-scalable structures (the store and load queues) to speculatively perform in-window store-to-load forwarding and detect any ensuing mis-forwardings. To mitigate the performance and energy implications of these structures, we first proposed store-queue index prediction (SQIP) to avoid searching the store queue when executing loads [15]. Instead of searching the entire queue, a highly-accurate distance-based forwarding predictor identifies which store (if any) communicates with the load, accessing just the single location that corresponds to that store. In a second paper, we proposed eliminating the store and load queues completely [16] by augmenting register renaming to speculatively connect the store’s producer to the load’s consumers, obviating the need for the store queue as an intermediary. Filtered in-order load re-execution combined with effective address regeneration in the commit pipeline verifies the speculation and eliminates the load queue. This NSF-supported project was joint work with graduate student Tingting Sha and Prof. Amir Roth, and it was selected as an IEEE Micro “Top Pick” from the 2006 architecture conferences [17].

Architectural Implications of Terabit Networking

In collaboration with Prof. Jonathan Smith, I participated in a DARPA-funded study of the systems impact of next-generation terabit LAN and WAN networking [18]. My role included identifying architectural implications and challenges such as efficiently interfacing with highly-integrated multicore CPUs and GPUs, memory bandwidth limitations, energy concerns, and the networking capacity demands of high-bandwidth solid-state storage technologies.

Demonstrating, Detecting, and Defending Against Malicious Hardware

I’m collaborating with Prof. Sam King (of University of Illinois) and Prof. Jonathan Smith on BlueChip, a NSF-funded project on demonstrating, detecting, and defending against malicious hardware. This project brings together a cross-disciplinary team of computer architects and computer security researchers to addresses attacks on the most basic building blocks of modern computing systems. Our initial foray focuses on preventing rogue designers from inserting malicious circuitry. We’re exploring design-time checking of HDL source code using a synthesis of code coverage and dataflow tracking to identify stealthy circuits [10].

Teaching & Education

In my complementary role as an educator, I have: taught several courses (three different required undergraduate courses, one “core” graduate course, and two graduate seminars), received strong teaching evaluations (overall weighted average of 3.4 out of 4), taught an average of 44 students per course, advised students at all levels (undergraduate, masters, and PhD students), supervised senior projects, and contributed to the development of course materials and infrastructure used at Penn and beyond.

CIS/CSE 240: Introduction to Computer Systems

E Lewis and I completely redesigned and co-taught “CSE240: Introduction to Computer Systems” in 2004. This 100-student sophomore-level course is one of the required courses at the heart of all Penn computer science majors (BAS and BSE in CS, as well as BSE in Digital Media Design), and it serves as the starting point for other systems courses in the major (operating systems, networking and security, databases, and more advanced computer hardware courses).

The new course design uses a “no magic” bottom-up approach starting with transistors and progressing through digital logic, assembly-level programming, and C programming. Throughout, the connections between the layers are stressed, and related topics in operating systems, compilers, and more advanced architectures are introduced. The highlight of the course is the mid-semester project in which students write a simple game in assembly language (for example, over the years we have used Tetris, Breakout, and Snake). Since its inception, this redesign has remained a key part of the curriculum.

CIS/CSE 372: Digital Systems Organization and Design Lab

I transformed “CSE 372: Digital Systems Organization and Design Lab”, the required lab component of the junior-level computer architecture course (required for the BSE in CS) for its Spring 2006 offering. I added a weekly lab lecture to cover core material and discuss the software design tools, design methodology, validation approaches, and other advanced topics in digital design. More importantly, the course used reconfigurable hardware prototyping boards with real I/O devices rather than just simulating the hardware design in software. I worked closely with the TAs to create infrastructure for the lab, including design tool tutorials and hardware designs for interacting with the keyboard and video monitor devices. The lab course consists of several projects in which the students build parts of a processor, with the end result being a processor that can play the simple graphical games introduced in CSE240.

Other Courses

I have taught CIS/CSE371 (the junior-level computer organization course lecture) and CIS501 (graduate-level computer architecture course and one of six “core” courses in the CIS graduate program). My offerings of these courses were based upon Prof. Amir Roth’s format and lecture slides. My revisions to these courses include additional material on multicore processors (501 and 371) and an in-class research paper discussion component (501). In addition, I have taught two graduate seminars.

Course Materials and Infrastructure

As part of redesigning CSE240 and CSE371, we developed “PennSim”, an architectural simulator for use in these courses. PennSim was first developed with the help of a Penn masters student, and has been repeatedly extended by me and others to support more multiple instruction set architectures, new hardware devices, simple pipeline and cache simulation, and a GUI code debugger. Beyond its use in multiple Penn courses, PennSim has also been used at University of Wisconsin.

In addition, Amir Roth and I have shared our lecture slides for CIS371 and CIS501 with professors at other universities. As a result, versions of these slides are in active use at several universities, including: University of Washington, Duke, University of Wisconsin, and University of Michigan.

References

- [1] C. Blundell, J. Devietti, E. C. Lewis, and M. M. K. Martin. Making the Fast Case Common and the Uncommon Case Simple in Unbounded Transactional Memory. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, June 2007.
- [2] C. Blundell, E. C. Lewis, and M. M. K. Martin. Deconstructing Transactional Semantics: The Subtleties of Atomicity. In *Proceedings of Workshop on Duplicating, Deconstructing, and Debunking*, June 2005.
- [3] C. Blundell, E. C. Lewis, and M. M. K. Martin. Subtleties of Transactional Memory Atomicity Semantics. *IEEE TCCA Computer Architecture Letters*, 5(2), Nov. 2006.
- [4] C. Blundell, E. C. Lewis, and M. M. K. Martin. Unrestricted Transactional Memory: Supporting I/O and System Calls within Transactions. Technical Report CIS-06-09, Department of Computer and Information Science, University of Pennsylvania, Apr. 2006.
- [5] C. Blundell, M. M. K. Martin, and T. Wensch. InvisiFence: Performance-Transparent Memory Ordering in Conventional Multiprocessors. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, June 2009.
- [6] S. Burckhardt, R. Alur, and M. M. K. Martin. Verifying Safety of a Token Coherence Implementation by Parametric Compositional Refinement. In *Proceedings of the Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, Jan. 2005.
- [7] S. Burckhardt, R. Alur, and M. M. K. Martin. Bounded Model Checking of Concurrent Data Types on Relaxed Memory Models: A Case Study. In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV)*, pages 489–502, Aug. 2006.
- [8] S. Burckhardt, R. Alur, and M. M. K. Martin. CheckFence: Checking Consistency of Concurrent Data Types on Relaxed Memory Models. In *Proceedings of the SIGPLAN 2007 Conference on Programming Language Design and Implementation*, June 2007.
- [9] J. Devietti, C. Blundell, M. M. K. Martin, and S. Zdancewic. Hardbound: Architectural Support for Spatial Safety of the C Programming Language. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2008.
- [10] M. Hicks, S. T. King, M. M. K. Martin, and J. M. Smith. Untrusted Computing Base: Demonstrating and Detecting Malicious Hardware. In submission, Mar. 2009.
- [11] M. M. K. Martin. Formal Verification and its Impact on the Snooping versus Directory Protocol Debate. In *Proceedings of the International Conference on Computer Design*, Oct. 2005.
- [12] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood. Improving Multiple-CMP Systems Using Token Coherence. In *Proceedings of the 11th Symposium on High-Performance Computer Architecture*, Feb. 2005.
- [13] S. Nagarakatte, J. Zhao, M. M. K. Martin, and S. Zdancewic. SoftBound: Highly Compatible and Complete Spatial Memory Safety for C. In *Proceedings of the SIGPLAN 2009 Conference on Programming Language Design and Implementation*, June 2009.
- [14] A. Raghavan, C. Blundell, and M. M. K. Martin. Token Tenure: PATCHing Token Counting Using Directory-Based Cache Coherence. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 47–58, Nov. 2008.
- [15] T. Sha, M. M. K. Martin, and A. Roth. Scalable Store-Load Forwarding via Store Queue Index Prediction. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 159–170, Nov. 2005.
- [16] T. Sha, M. M. K. Martin, and A. Roth. NoSQ: Store-Load Communication without a Store Queue. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 285–296, Dec. 2006.
- [17] T. Sha, M. M. K. Martin, and A. Roth. NoSQ: Store-Load Communication without a Store Queue. *IEEE Micro*, 27(1), Jan/Feb 2007.
- [18] J. M. Smith and M. M. K. Martin. Terabit Edge Research Activity (TERA). Technical Report AFRL-RY-WP-TR-2008-1254, Air Force Research Laboratory, Oct. 2008.