Research and Teaching Statement

Christoforos (Christos) Kozyrakis
http://csl.stanford.edu/~christos

1. Research Summary

Computing systems are now an essential part of our infrastructure for communication, commerce, government, education, scientific discovery, and entertainment. Users expect computers to provide ever increasing performance, while operating securely and in a resource efficient manner. In the past few years, however, performance improvements for single processor systems have nearly stopped, the volume of security attacks has grown, and energy consumption has become a major limitation for embedded and high-end systems.

The broad goal of my research is to architect efficient systems that meet user expectations. My basic approach is to work across the hardware-software interface and introduce new functionality and abstractions that can overcome the current limitations. I also build full-system prototypes in order to understand the practicality of hardware designs and measure the usefulness of the new functionality with real-world software.

Towards this goal, I have developed practical implementations of transactional memory, a technology that helps programmers turn the increasing number of processors in multi-core chips to scalable application performance. I have also created a robust security framework that uses dynamic information flow tracking to protect deployed software from a wide range of attacks. Finally, I have worked on tools to measure and analyze the energy efficiency of modern systems. This work has been presented in some of the top hardware symposiums (e.g., ISCA, ASPLOS, HPCA) and systems conferences (e.g., PLDI, OSDI, SIGMOD, POPL).

1.1 Transactional Memory

The industry-wide turn towards multi-core chips means that mainstream software developers must deal with the complexity of parallel programming. Transactional memory (TM) is a promising technology that can address the difficulty of synchronizing concurrently executing tasks. TM provides programmers with an intuitive abstraction that allows them to simply define portions of their code that should execute as atomic and isolated transactions. Concurrency management, including thorny issues such as races, deadlocks, or priority inversion, is the responsibility of the system and is hidden from programmers.

My research has covered most aspects of TM technology, from hardware to programming models and applications. The major conclusion is that for TM to deliver on its potential for high performance with simple parallel code, hardware support is needed. Hardware-assisted TM systems can be cost effective and sufficiently flexible to support innovation in parallel programming models. The following paragraphs summarize the specific results:

Hardware Support for TM: Software-only TM systems introduce significant per-thread overheads that cancel the benefits of parallel execution. We have developed Transactional Coherence

and Consistency (TCC), a hardware TM system that implements lazy data versioning and optimistic conflict detection using additional tag bits in cache memories [18, 22, 7, 20]. TCC eliminates per-thread overheads and provides robust performance across a range of contention scenarios. TCC enforces memory coherence and consistency in a bulk manner at the boundaries of transactions, which reduces hardware complexity and improves scalability.

TM Hardware Virtualization: Programs that use TM should not be limited by the physical constraints of hardware. We developed XTM, a mechanism that virtualizes all aspects of TM system (transaction length, dataset size, and nesting depth) [11]. XTM uses the existing virtualization mechanisms in the OS to support unbounded TM in a manner that is transparent to applications.

Hardware/Software Interface for TM: A critical issue for TM systems is interoperability with programming environments and operating systems. We proposed a comprehensive hardware/software interface for TM based on three mechanisms: two-phase commit, transactional handlers, and support for nested transactions [21]. We showed that it is adequate to support managed languages [6], composable libraries [5], and uses of TM for security and reliability [14, 9].

Hybrid TM System: Hardware TM systems introduce significant complexity in the cache hierarchy. We developed SigTM, a hybrid system that supports TM without modifications to the caches [4]. Using a set of hardware Bloom filters for conflict detection, it provides a 2x performance advantage over software-only systems. It also implements strong atomicity, a feature that simplifies interactions between code with transactions and other portions of a program.

Programming with Transactions: We have integrated transactions in two parallel programming models. Atomos is a Java variant with constructs for implicit transactions used to synchronize and coordinate parallel threads [6, 2]. OpenTM introduces transactions into the OpenMP model for master-slave parallelism [1]. It provides programming constructs for non-blocking synchronization and speculative parallelization using transactions.

Programmability Tools: We have developed tools to help write, debug, and optimize programs. Transactional collection classes use semantic concurrency control to provide thread-safe data structures that are composable and perform well [5]. TAPE builds upon TM hardware to automatically identify the points in the code responsible for the most significant performance losses [8].

Parallel Applications: Application development and analysis has helped us identify the common case behavior to optimize for [12], understand the advantages and shortcomings of TM [10], and evaluate programming constructs [28]. We also developed the STAMP benchmark suite that exercises a range of features of TM systems and is an effective tool for future TM research [3].

1.2 Secure Systems

Apart from transactions, memory tags can also support abstractions that improve the security of computing systems. Despite significant efforts, deployed software is still vulnerable to an ever-increasing set of attacks ranging from buffer overflows to SQL injections. My research has focused on creating a comprehensive security framework using dynamic information flow tracking (DIFT). The main idea of DIFT is to track the flow of untrusted data during program execution and continuously check for unsafe uses. Using memory tags to taint untrusted data and support programmable security policies, we developed a security framework with the following features: robust (protects against all known types of attacks), flexible (can be adjusted for future attacks), practical (works for all types of executable binaries), end-to-end (protects applications and the OS), and fast (no noticeable slowdown). The following paragraphs summarize the specific results:

Hardware Support for Security: We developed Raksha, a system with hardware support for DIFT [19]. Hardware performs fine-grain taint propagation and checks as programs run, while a

software manager sets security policies and invokes further analysis when needed. Unlike previous DIFT systems, Raksha supports multiple concurrently active policies and can protect user code as well as the OS. To demonstrate the practicality of Raksha, we developed a full-system prototype based on a SPARC processor and the Linux OS.

Protection Against High-level Attacks: We developed robust DIFT policies to detect high-level attacks such as SQL injection, command-line injection, and directory traversals [19]. Raksha is the first DIFT system that protects unmodified application binaries from such attacks.

Robust Protection Against Buffer Overflows: Existing security tools, including DIFT tools, provide partial protection against buffer overflows or generate many false positives [15]. We developed a new set of DIFT policies that robustly detect buffer overflows without false positives [16]. To avoid the difficult problem of recognizing input validation that haunts previous approaches, the policies simply protect legitimate pointers from being overwritten with untrusted data. We have demonstrated that the policies are practical and robust even in the case of stripped binaries.

End-to-end Protection: Using the Raksha full-system prototype, we have demonstrated end-to-end protection of all code against a comprehensive list of attacks [16]. We use multiple security policies to concurrently protect against control and data based memory corruptions, format-string exploits, and high-level attacks. This is also the first DIFT system to dynamically protect the whole Linux kernel from memory corruption attacks and user/kernel pointer dereferences.

Hardware Enforcement of Application-level Policies: We recently developed Loki, a variant of Raksha with generalized support for tagged physical memory [31]. We used Loki for hardware enforcement of application security policies such as restrictions on data access, movement, and use. This allowed us to reduce the trusted code base of a secure OS by a factor of two.

1.3 Energy Efficient Systems

Beyond performance and security challenges, power consumption emerges as the major limitation for data-centers in terms of operational cost, scalability, reliability, and environmental impact. My research has focused on providing tools to support the two complementary ways towards higher energy efficiency in data centers: adaptively managing the power consumption as the workload varies and building energy efficiency into the original design of the system components.

Non-intrusive Power Models for Servers: Power models are a fundamental component of an adaptive management system. We have developed a non-intrusive modeling approach that can accurately estimate the power consumption of servers using OS-level utilization metrics and hardware counters [27, 17]. Unlike models based on processor activity alone, our models are portable and accurate across a wide range of systems and workloads [25].

Benchmarking Energy: Benchmarks and metrics are necessary in order to analyze power consumption and guide the design of new systems. We designed JouleSort, the first full-system energy efficiency benchmark for servers [26]. It uses external sort as its workload in order to stress all server components (processor, memory, and I/O). In addition to the measurement rules, JouleSort defines a metric that balances performance and energy consumption and avoids basic pitfalls in energy benchmarking.

Energy Efficient Servers: Using JouleSort, we have analyzed a wide range of servers and identified key trends in energy efficiency across generations of systems and components. Based on the lessons, we built a prototype server that is three times more energy efficient than commercial systems [26]. Our server used notebook components (processors and disks) that have been designed to balance performance and power consumption.

1.4 Prototypes

What we have to learn, we learn by doing it[1]. All my projects have developed full system prototypes to gain better insights into the research challenges. For TM, we built an 8-core TCC prototype that boots Linux using the BEE2 FPGA board [30, 23]. Forty attendees at an ISCA'07 tutorial used it to write and optimize transactional programs. We prototyped a full-featured, Linux workstation for Raksha using a SPARC processor and an FPGA board. We have shared the design with colleagues in industry and academia. JouleSort is now part of the official Sort benchmark and our prototype server is the 2008 benchmark winner.

We have also shared our work with the wider community as open source software. We have released the Phoenix environment for MapReduce programming on multi-core systems [24], the OpenTM compiler for transactional programming [1], and the STAMP benchmarks suite [3].

1.5 Students and Collaborators

Faculty are in the business of producing students. I have been fortunate to work with a great set of PhD students. Ahmad Zmily, Suzanne Rivoire, JaeWoong Chung, Sewook Wee, and Chi Cao Minh have completed their degrees. Austen McDonald has defended his thesis and will graduate shortly. My group now includes five PhD students that work on parallelism, security, and energy efficiency.

My research has benefited greatly from collaborators within and outside Stanford. In particular, I have worked closely with Kunle Olukotun on transactional memory and Partha Ranganathan on energy efficiency. Along with researchers from several top universities, we recently created RAMP, an effort to build the research infrastructure for scalable multi-core systems [29].

1.6 Future Directions

Looking forward, two technical challenges capture my attention. The first challenge is *scalability*. Even the simplest computer system will soon comprise hundreds of components such as processors, memories, and I/O devices. Hence, we should now consider it as a distributed system and re-engineer all hardware and software layers of the stack accordingly. This work will include identifying the right abstractions for the hardware-software interface, developing scalable memory models and communication schemes [19], creating a layered runtime environment to adaptively manage heterogeneous resources, and supporting monitoring features that allow us to understand and optimize the behavior of the system.

The second challenge is that of *energy efficiency*. The experience with energy benchmarking and our prototype server indicates that the large potential in revisiting hardware architecture for energy efficiency. The evolutionary process from the early PCs to stand-alone servers has not necessarily led to an optimal organization, balance, and component mix for energy efficiency. Recognizing that modern computers are part of a larger system, a data-center or a client-server model, allows for further, end-to-end optimizations using hardware or software techniques. It is also necessary to develop the proper abstractions for reasoning, accounting, optimizing, and enforcing constraints on energy consumption across the layers of the system stack (hardware, operating system, and applications).

Even thought it is hard to predict the future, I expect my research to focus on these challenges, moving along the hardware-software interface and the various layers of the system stack as needed.

[1] Aristotle, *Ethica Nicomachea*, ca. 325 BC.

2. Teaching Summary

I enjoy teaching at all levels and take curriculum development seriously. My overarching goal is to provide students with an intuitive understanding of the principles of computing systems. Hence, I focus on fundamental concepts (e.g., layers of abstractions, separation of policy and mechanism, end-to-end system approach) and basic techniques (e.g., parallelism, pipelining, caching, indirection, speculation, amortization), using modern and classic systems as illustrative examples.

When I joined Stanford, there were two regularly offered architecture courses, taught mostly by visitors and focusing on a dated, processor-centric view of the field. I worked with senior colleagues to restructure the graduate curriculum which now includes four regular courses on systems architecture, parallel architecture, processor design, and interconnect networks. I designed and teach two of them. Along with the two new undergraduate classes on digital systems, the curriculum now provides Stanford students with a comprehensive coverage of computer architecture.
I teach three classes on a yearly basis:

1. **Digital Systems II** (EE108b, 85 to 130 students) is a junior class on the design of processor-based digital systems (instruction set architecture, low-level mapping of programming constructs, pipelining, caching, OS support, virtual memory, and I/O techniques). In the labs, students progressively implement a pipelined processor with caches using FPGA boards.
2. **Computer Systems Architecture** (EE282, 80 to 135 students) is a new mezzanine class with a system focus. It covers advanced hardware topics (memory hierarchy, main memory systems, I/O acceleration), hardware-software interactions (virtualization and performance optimization), cluster architecture and programming, power efficiency and reliability. In the programming assignments, the students use their understanding of small-scale and large-scale systems to optimize data-intensive applications on them.
3. **Advanced Processor Architecture** (EE382a, 20 to 35 students) is a new course that trains future researchers in computer architecture. It covers advanced ILP techniques, multithreading, multi-core chips, and designs for data-level parallelism. Students perform a research project on topics ranging from architectural support for security to heterogeneous systems.

I strongly believe that teaching computer systems is incomplete without practical experience with hands-on projects. All my classes include labs or significant projects. The student projects in EE382a typically lead to a couple of conference publications each year, one of which won the best paper award in HPCA'07 [13]. I also encourage undergraduate students to get involved with research through independent studies or REU projects. One of my advisees won the 2005 CS undergraduate thesis award in Computer Science and a Stanford Firestone medal for his research.

I have also been active in teaching activities beyond the Stanford campus. Along with colleagues from Intel, I have created an extensive tutorial on transactional memory technology. We have taught this material at four international conferences so far. I also taught a course on transactional memory at the 2008 ACACES summer school for computer architecture, where I received the highest student ratings from the twelve instructors at the school.

My future plans include creating a course that provides an integrated view of the computing infrastructure (data-center hardware, networking, client-server software). My goal is to provide students with a understanding of crosscutting issues across system layers such as end-to-end efficiency, security, and reliability. I am also interested in courses that can attract undergraduates to computing related majors. I will work with colleagues on courses that expose students the basic concepts as they build and deploy exciting systems such as cell phones or digital media centers.

References

[1] W. Baek, C. Cao Minh, M. Trautmann, C. Kozyrakis, and K. Olukotun. The OpenTM Transactional Application Programming Interface. In *Proceedings of the 16th Intl. Conference on Parallel Architecture and Compilation Techniques (PACT)*, Brasov, Romania, Sept. 2007.

[2] N. Bronson, C. Kozyrakis, and K. Olukotun. Feedback-Directed Barrier Optimization in a Strongly Isolated STM. In *Proceedings of the 36th Symposium on Programming Languages Principles (POPL)*, Savannah, GA, Jan. 2009.

[3] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In *Proceedings of The IEEE Intl. Symposium on Workload Characterization (IISWC)*, Seattle, WA, Sept. 2008.

[4] C. Cao Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis, and K. Olukotun. An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees. In *Proceedings of the 34th Intl. Symposium on Computer Architecture (ISCA)*, San Diego, CA, June 2007.

[5] B. Carlstrom, A. McDonald, M. Carbin, C. Kozyrakis, and K. Olukotun. Transactional Collection Classes. In *Proceedings of the ACM SIGPLAN Conference on Principles and Practice of Parallel Computing (PPoPP)*, San Jose, CA, Mar. 2007.

[6] B. Carlstrom, A. McDonald, H. Chafi, J. Chung, C. Cao Minh, C. Kozyrakis, and K. Olukotun. The ATOMOS Transactional Programming Language. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, pages 1–13, Ottawa, Canada, June 2006.

[7] H. Chafi, J. Casper, B. Carlstrom, A. McDonald, C. Cao Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A Scalable, Non-blocking Approach to Transactional Memory. In *Proceedings of the 13th Intl. Symposium on High-Performance Computer Architecture (HPCA)*, Phoenix, AZ, Feb. 2007.

[8] H. Chafi, A. McDonald, C. Cao Minh, J. Chung, B. Carlstrom, L. Hammond, C. Kozyrakis, and K. Olukotun. TAPE: a Transactional Application Profiling Environment. In *Proceedings of the 19th Intl. Conference on Supercomputing (ICS)*, Boston, MA, June 2005.

[9] J. Chung, W. Baek, N. Bronson, J. Seo, C. Kozyrakis, and K. Olukotun. ASeD: Availability, Security, and Debugging using Transactional Memory (Brief Announcement). In *Proceedings of the 20th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA)*, Munich, Germany, June 2008.

[10] J. Chung, C. Cao Minh, B. Carlstrom, and C. Kozyrakis. Parallelizing SPECjbb2000 with Transactional Memory. The Workshop on Transactional Memory Workloads (WTW), `http://freya.cs.uiuc.edu/WTW/`, June 2006.

[11] J. Chung, C. Cao Minh, A. McDonald, T. Skare, H. Chafi, B. Carlstrom, C. Kozyrakis, and K. Olukotun. Tradeoffs in Transactional Memory Virtualization. In *Proceedings of the 12th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, Oct. 2006.

[12] J. Chung, H. Chafi, C. Cao Minh, A. McDonald, B. Carlstrom, C. Kozyrakis, and K. Olukotun. The Common Case Transactional Behavior of Multithreaded Programs. In *Proceedings of the 12th Intl. Symposium on High Performance Computer Architecture (HPCA)*, Austin, TX, Feb. 2006.

[13] J. Chung, M. Dalton, H. Kannan, and C. Kozyrakis. Thread-Safe Dynamic Binary Translation Using Transactional Memory. In *Proceedings of the 14th Intl. Symposium on High-Performance Computer Architecture (HPCA)*, Salt Lake City, UT, Feb. 2008.

[14] J. Chung, J. Seo, W. Baek, C. Cao Minh, C. Kozyrakis, and K. Olukotun. Improving Software Concurrency with Hardware-Assisted Memory Snapshot (Brief Announcement). In *Proceedings of the 20th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA)*, Munich, Germany, June 2008.

[15] M. Dalton, H. Kannan, and C. Kozyrakis. Deconstructing Hardware Architectures for Security. The 5th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD), `http://www.ece.wisc.edu/~wddd2006`, June 2006.

[16] M. Dalton, H. Kannan, and C. Kozyrakis. Real-World Buffer Overflow Protection for Userspace and Kernelspace. In *Proceedings of the 17th Usenix Security Symposium*, San Jose, CA, July 2008.

[17] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system Power Analysis and Modeling for Server Environments. The Workshop on Modeling Benchmarking and Simulation (MOBS), http://www-mount.ece.umn.edu/~jjyi/MoBS2006/, June 2006.

[18] L. Hammond, B. Carlstrom, V. Wong, M. Chen, B. Hertzberg, B. Carlstrom, M. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional Memory Coherence and Consistency (TCC). In *Proceedings of the 11th Intl. Symposium on Computer Architecture (ISCA)*, Munich, Germany, June 2004.

[19] J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis. Comparing Memory Systems for Chip Multiprocessors. In *Proceedings of the 34th Intl. Symposium on Computer Architecture (ISCA)*, San Diego, CA, June 2007.

[20] C. Manovit, S. Hangal, A. McDonald, H. Chafi, C. Kozyrakis, and K. Olukotun. Testing Implementations of Transactional Memory. In *Proceedings of the 15th Intl. Conference on Parallel Architecture and Compilation Techniques (PACT)*, Seattle, WA, Sept. 2006.

[21] A. McDonald, J. Chung, B. Carlstrom, C. Cao Minh, C. Kozyrakis, and K. Olukotun. Architectural Semantics for Practical Transactional Memory. In *Proceedings of the 33rd Intl. Symposium on Computer Architecture (ISCA)*, Boston, MA, June 2006.

[22] A. McDonald, J. Chung, H. Chafi, C. Cao Minh, B. Carlstrom, L. Hammond, C. Kozyrakis, and K. Olukotun. Characterization of TCC on Chip Multiprocessors. In *Proceedings of the 14th Intl. Conference on Parallel Architecture and Compilation Techniques*, Saint Louis, MO, Sept. 2005.

[23] N. Njoroge, J. Casper, S. Wee, Y. Teslyar, D. Ge, C. Kozyrakis, and K. Olukotun. ATLAS: A Chip-Multiprocessor with Transactional Memory Support. In *Proceedings of the Conference on Design Automation and Test in Europe (DATE)*, Nice, France, Apr. 2007.

[24] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 13th Intl. Symposium on High-Performance Computer Architecture (HPCA)*, Phoenix, AZ, Feb. 2007.

[25] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A Comparison of High-Level Full-System Power Models. The Workshop on Power Aware Computing and Systems (HotPower), http://www.usenix.org/events/hotpower08/, Dec. 2008.

[26] S. Rivoire, M. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A Balanced Energy-Efficiency Benchmark. In *Proceedings of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)*, Beijing, China, June 2007.

[27] S. Rivoire, M. Shah, R. Ranganathan, C. Kozyrakis, and J. Meza. Models and Metrics to Enable Energy-Efficiency Optimizations. *IEEE Computer*, 40(12):39–49, Dec. 2007.

[28] T. Skare and C. Kozyrakis. Early Release: Friend or Foe. The Workshop on Transactional Memory Workloads (WTW), http://freya.cs.uiuc.edu/WTW/, June 2006.

[29] J. Wawrzynek, D. Patterson, M. Oskin, S. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic. RAMP: Research Accelerator for Multiple Processors. *IEEE MICRO*, pages 46–57, Mar. 2007.

[30] S. Wee, J. Casper, N. Njoroge, Y. Tesylar, D. Ge, C. Kozyrakis, and K. Olukotun. A Practical FPGA-based Framework for Novel CMP Research. In *Proceedings of the 15th ACM/SIGDA Intl. Symposium on Field Programmable Gate Arrays (FPGA)*, Montery, CA, Feb. 2007.

[31] N. Zeldovich, H. Kannan, M. Dalton, and C. Kozyrakis. Hardware Enforcement of Application Security Policies. In *Proceedings of the 8th Usenix Sympoisum on Operating Systems Design and Implementation*, San Diego, CA, Dec. 2008.