Statement of Research Interests

Thomas F. Wenisch

I am broadly interested in computer architecture with particular emphasis on multi-core/multiprocessor systems, memory systems, and performance evaluation methodology. In particular, my future research interests are shaped by the emerging trend towards *multi-core* systems. Semiconductor industry observers now predict that, instead of processor clock frequencies, cores per die will double every two years [1]. My research agenda focuses on the question of how to leverage these cores to continue the trend of exponentially improving system performance.

My past research investigates memory system design and accelerated performance evaluation for uni- and multiprocessor systems. My dissertation research proposes *Temporal Memory Streaming*, a novel memory system design paradigm that exploits repetition in memory access patterns to transfer related data in groups. Furthermore, I am a primary contributor to the *SimFlex* project, which proposes statistically-rigorous computer system performance evaluation methodologies. In this statement, I briefly summarize the contributions of my prior research, and detail my future research plans.

Dissertation Research – Memory System Design

Technological advancements in semiconductor fabrication have led to an abundance of transistors, faster clock speeds, and unprecedented processor performance. In contrast, scaling trends in DRAM technology have favored improving density over access latency. As a result, modern processors spend much of execution time stalled on long-latency memory accesses. The conventional approach to latency tolerance—enlarging the on-chip cache hierarchy as transistor budgets scale—is providing diminishing returns because today's multi-megabyte caches already capture available locality. Commercial server applications present a particular challenge for memory system design because current prefetching/streaming approaches are often ineffective on the irregular data structures and dependent miss chains characteristic of these applications. To further improve server performance, architects must design mechanisms that issue memory requests earlier and with greater parallelism in the face of complex access patterns.

The key shortcoming of conventional cache hierarchies is the absence of information within the hierarchy about programs' inherent memory access patterns. Despite their complexity, commercial applications nonetheless execute repetitive code sequences, which give rise to recurring access sequences over frequently-traversed data structures. As a result, memory access addresses are often correlated—that is, accesses that occur near one another in time once are likely to recur together again, a property I refer to as *temporal correlation*. By recording temporally-correlated cache miss addresses and using the recorded information to predict future misses, irregular yet repetitive miss patterns can be predicted.

In my dissertation research, I propose Temporal Memory Streaming (TMS) [2], a memory system design paradigm where data are transferred from main memory according to *streams*—ordered sequences of temporally-correlated addresses. Streams are constructed dynamically in hardware by recording the sequence of off-chip memory requests issued by a program. In applications with producer-consumer communication, streams can be constructed by recording the producer's writes, which tend to match the order of the consumer's reads [3]. When the memory system predicts that a previously-observed access sequence will repeat (e.g., by observing requests for the first addresses of a stream), it uses the recorded stream to transfer the data in advance of explicit processor requests. TMS is particularly effective for traversals of the linked-data structures characteristic of commercial server applications, as the recorded stream allows otherwise-dependant memory accesses to proceed in parallel.

In addition to my dissertation research on TMS, I have collaborated with fellow graduate student Stephen Somogyi to investigate further approaches to memory streaming. Our recent work proposes Spatial Memory Streaming (SMS) [4], which constructs streams from repetitive data structure layouts instead of repetitive access sequences. Whereas TMS primarily targets pointer-chasing access patterns, SMS is most effective in scans over fixed-layout records (e.g., sequential scan of a database table).

Performance Evaluation Methodology

Much of computer architecture research, including my dissertation research, relies on software simulation to measure dynamic performance metrics (e.g., clocks-per-instruction) of a proposed design. Unfortunately, with the ever-growing size and complexity of modern microprocessors, detailed software simulators are four or more orders of magnitude slower than their hardware counterparts. The resulting low simulation throughput is especially prohibitive for large-scale multiprocessor systems because the simulation turnaround for these systems grows at

least linearly with the number of processors. Slow simulation has barred researchers from investigating complete benchmarks and input sets or realistic system sizes on detailed simulators.

The SimFlex project at Carnegie Mellon makes simulation-based studies of large-scale multiprocessor systems tractable by applying rigorous statistical sampling methods to computer system simulation [5]. By measuring only a sample of an application's execution, the SimFlex approach provides ten-thousand-fold reductions in simulation time while providing quantified measures of statistical confidence with each estimated performance metric. Moreover, because it allows each of the individual performance measurements comprising a sample to be performed independently, the SimFlex approach enables thousand-way parallelism over a cluster of simulation hosts to further reduce simulation turnaround time.

Two primary challenges must be addressed to minimize computer system simulation turnaround through statistical sampling. First, we must choose a representative subset of each application's execution for measurement. Our research demonstrates that the nature of performance variability across measurement granularities favors a large sample (e.g., 10,000 measurements) of brief (e.g., 1000-instruction) execution windows to minimize total simulation [6]. Second, we must construct accurate initial system state for the large number of fine-grain performance measurements. Rapid construction is challenging because the state of many structures (e.g., main memory, cache hierarchy) reflect the execution history of millions of instructions. For these structures, we create checkpoints of warm state using simplified simulation models [7]. We amortize the cost of checkpoint construction by reusing them over many experiments.

In addition to the challenges of evaluation methodology, many important applications, such as commercial databases and web servers, are difficult to simulate because they interact frequently with the operating system and peripheral devices. To enable study of these applications, we have developed the *Flexus* full-system simulation framework. Flexus is a family of component-based computer architecture simulators that enable full-system timing-accurate simulation of uni- and multiprocessor systems running unmodified commercial applications and operating systems. To accelerate checkpoint construction for large-scale installations of these applications, we are developing the *ProtoFlex* FPGA-based emulation platform. ProtoFlex is pioneering *co-simulation*, where the majority of execution occurs in FPGA emulation, while infrequent/complex operations (e.g., I/O) are handled by transferring CPU state from the FPGA to software simulation.

My role in the SimFlex project is two-fold. First, in collaboration with Roland Wunderlich, I developed the sampling and checkpoint-based state construction methodologies we describe in our publications. Second, I led the development of the Flexus simulation infrastructure. Flexus is used by several research groups at Carnegie Mellon, and has been the primary infrastructure used in the graduate computer architecture courses since the spring of 2005. Furthermore, Flexus is publicly available and has been adopted at several top academic and industrial research labs outside of Carnegie Mellon.

Future Research Directions

Over the past several years, a paradigm shift has occurred in the microprocessor design industry. In the past, improvements in silicon manufacturing technology have enabled steady increases in transistor density, processor clock frequency, and processing speed. Industry projections indicate that transistor density will continue increasing for at least another decade. Unfortunately, power and thermal constraints have slowed the march of clock speed. Moreover, design complexity, verification effort, and scalability issues in centralized structures impede further performance improvement in monolithic designs.

Instead of designing increasingly-complex monolithic architectures, processor manufacturers have turned to *multi-core* architectures, where several processor cores are integrated on a single die. The multi-core design paradigm provides scalability while easing complexity and validation challenges. While the trend towards multi-core design began in servers (e.g., IBM Power 5, Sun Niagara), the same silicon manufacturing pressures are already leading to multi-core desktop and embedded processors (e.g., Freescale MC5510 dual-core microcontroller).

Although the multi-core paradigm improves design scalability, it leaves us with two new challenges: (1) The programmer, compiler, and/or run-time system must identify parallel tasks to run on each core. (2) The hardware must provide for efficient communication between cores. While both these challenges exist in traditional multiprocessors, our need to find solutions has become more immediate; our ability to keep cores occupied with useful work will soon become the limiting factor in continued application performance improvement.

To address the new challenges of the multi-core era, I intend to pursue the following research directions:

Architectural support for programmability/debuggability. Several factors make parallel programming more difficult then sequential programming. First, parallel execution introduces new classes of programming errors (e.g., atomicity and synchronization bugs) that are not possible in sequential code. Second, application performance often depends heavily on microarchitectural details, such as coherence granularity, the relative die position of

communication cores, or constructive/destructive sharing patterns. Finally, to allow for maximum concurrency among memory accesses, many systems burden programmers with non-intuitive memory access ordering rules (i.e., relaxed consistency models) that can cause naïve synchronization primitives to fail in surprising ways.

To combat these factors, future multi-core systems must be designed to maximize ease of programmability and minimize the effort of locating bugs. Systems must detect and report atomicity violations and data races, and assist programmers in reconstructing non-deterministic execution sequences that lead to a failure. Performance counter hardware must be enhanced to track and report the performance effects unique to the multi-core environment. Ideally, applications or the run-time system will respond dynamically to this performance feedback. Finally, architects, program language theorists, and compiler writers must collaborate to simplify the parallel programming interface (e.g., speculatively-relaxed implementations of sequential consistency or transactional memory).

All of these approaches rely on a small number of key architectural mechanisms. First, multi-core systems must provide support for introspection and monitoring. Hardware must enable one core to observe and respond to correctness/performance bugs in a neighboring core. Second, systems should support speculative execution and rollback (e.g., through coarse-grain checkpoints). Support for deep speculation allows systems to provide high-level guarantees (e.g., sequentially-consistent ordering or atomic execution) while hiding synchronization overhead in the common case that data races do not occur.

Architectural opportunities of disruptive technologies. Today's microarchitectural designs are being shaped by the challenges of deep-submicron silicon manufacturing: limited pins, increasing prevalence of transient faults, increased leakage, manufacturing variability, and power dissipation as a first-order design constraint. Industry and academia are both actively engaged in addressing these challenges at the device, circuit, and architectural levels. However, beyond these well-documented trends, a variety of disruptive technologies that may radically alter the tradeoffs guiding current designs are likely to mature within the next decade. It is the responsibility of academia to begin considering the opportunities and implications of these technologies now.

Two potential examples of disruptive technologies are 3D die stacking and optical transceivers on silicon CMOS. 3D stacking allows several dies to be bonded in a vertical stack through low latency and high bandwidth connections. Moreover, each die in a stack may be manufactured with distinct processes (e.g., optimized for logic or DRAM). 3D stacking enables drastic increases in on-chip memory, and potentially allows sensors, MEMS or other devices with unique manufacturing requirements to be integrated with high performance processors. Silicon optical transceivers enable on-chip and off-chip optical interconnection networks that increase communication bandwidth by two orders of magnitude over electronic signaling. These drastic increases in communication bandwidth and on-chip storage allow us to consider architectural mechanisms that are prohibitive in current technology (e.g., highly aggressive memory prefetching/streaming, memory update logging/rollback, high-bandwidth atomic/bulk memory writes). Furthermore, exploration of radical architectures based on disruptive technologies will require continued innovation in performance evaluation methodology.

Selected References

[1] D. A. Patterson. "President's Message: Computer Science Education in the 21st Century." *Communications of the ACM*, vol. 49 no. 3, Mar. 2006.

[2] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. "Temporal Streaming of Shared Memory." *32nd International Symposium on Computer Architecture (ISCA)*, Jun. 2005.

[3] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, C. Gniady, A. Ailamaki, and B. Falsafi. "Store-Ordered Streaming of Shared Memory." *14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2005.

[4] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi and A. Moshovos. "Spatial Memory Streaming." *33rd International Symposium on Computer Architecture (ISCA)*, Jun. 2006.

[5] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. "SimFlex: Statistical Sampling of Computer System Simulation." *IEEE MICRO Special Issue on Computer Architecture Simulation and Modeling*, vol. 26, no. 4, Jul./Aug. 2006.

[6] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling." *30th International Symposium on Computer Architecture (ISCA)*, Jun. 2003.

[7] T. F. Wenisch, R. E. Wunderlich, B. Falsafi and J. C. Hoe. "Simulation Sampling with Live-Points." *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2006.