This document is intended to aid my letter writers. It contains brief summaries of my research contributions and their impacts, organized by topic. It also describes some other noteworthy achievements from my time at Carnegie Mellon. I organize the document in three sections: memory system research, performance evaluation methodology research, and other achievements.

Memory System Research

Memory Streaming

Temporal address correlation. Many application classes, in particular commercial server applications, incur significant stalls on modern systems because of their large footprints and frequent chains of dependant memory accesses. Moreover, because of data structure complexity, current prefetching/streaming approaches that rely on simple heuristics (e.g., strides) are ineffective on the complex access patterns characteristic of these applications. Despite their complexity, these 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 at one point in program execution are likely to recur together again. This phenomenon—which I call *temporal address correlation* [WSH+05a]—is the fundamental observation underlying my dissertation research.

Temporal Memory Streaming. To exploit temporal address correlation, I propose Temporal Memory Streaming (TMS) [WSH+05a], 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. 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.

Store-ordered Memory Streaming. TMS constructs streams from repetitive miss sequences. Similar streams can be constructed from other correlations among data that indicate the order of future accesses. In particular, in applications with producer-consumer sharing, the order in which shared data are consumed by one processor is correlated to the order in which they were produced by another. Based on this producer-consumer temporal address correlation, I proposed Store-ordered Memory Streaming (with fellow students Hardavellas, Kim, and Somogyi) to eliminate coherence misses in producer-consumer style applications [WSH+05b].

Spatial Memory Streaming. 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) [SWA+06], 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).

Speculatively-relaxed Memory Consistency.

Store misses pose a significant performance challenge on shared-memory multiprocessors. Even under relaxed memory models, store delay remains a bottleneck for commercial applications because of limited store buffering and frequent synchronization. To minimize these stalls, researchers have proposed systems that speculatively relax ordering and roll back execution if a memory race exposes an ordering violation. Such systems improve performance because races are rare in well-behaved concurrent applications. However, past designs either provide insufficient buffering for speculative state or require high-overhead and non-scalable hardware.
In a manuscript that is currently under review [WAFM07], I propose two mechanisms that, together, enable store-wait–free implementations of any memory consistency model:

Atomic sequence ordering. To eliminate ordering-related stalls, I propose *atomic sequence ordering*. Atomic sequence ordering exploits the observation that if a memory access sequence appears atomic, then it does not violate memory consistency constraints regardless of the order accesses are performed. Atomic sequence ordering dynamically groups memory accesses into atomic sequences and relaxes ordering within and across sequences. Atomic sequence ordering allows the system to hide ordering-related stalls under speculative execution in the common case that there is no data race. Hence, systems can provide the programmer with the appearance of sequential consistency with performance rivaling or exceeding fully-relaxed memory models.

Scalable store buffer. The scalable store buffer separates the store order tracking and value forwarding functions of a conventional store buffer into separate structures. It places private/speculative values directly into the L1 cache, thereby eliminating the need for associative search. The scalable store buffer provides the buffering required by atomic sequence ordering with a tractable hardware implementation.

Coherence Activity Prediction.

I collaborated with Stephen Somogyi, Nikos Hardavellas, and Jangwoo Kim to investigate several prediction-based approaches to accelerate cache coherence activity [SWH+04]. We evaluated two general classes of coherence predictors:

Downgrade prediction (DGP). DGP predicts the last store to a cache block prior to a read by another node, allowing initiation of a downgrade (dirty to shared) transition ahead of explicit coherence requests. We show that program counter trace-based DGP is effective for commercial applications, correctly predicting one half to three quarters of last stores.

Consumer set prediction (CSP). CSP predicts the set of sharers of a newly-downgraded cache block based on the blocks prior sharing history. Although CSP is effective for scientific applications, which often have stable sharing patterns, we find it to be ineffective in commercial applications. In commercial applications, past sharing history is a poor indicator of the next node that will access the predominantly migratory shared data.

SimFlex — Performance Evaluation Methodology

Simulation Sampling.

SMARTS. Software-based microarchitecture simulators are many orders of magnitude slower than the hardware they simulate. Hence, most microarchitecture design studies draw their conclusions from drastically truncated benchmark simulations that are often inaccurate and misleading. With Roland Wunderlich, I proposed the Sampling Microarchitecture Simulation (SMARTS) framework to enable fast and accurate measurement of full benchmarks [WWFH03][WWFH06b][WWFH06c][WWF+06]. SMARTS prescribes a statistically-sound procedure for measuring a minimal benchmark subset to achieve a desired quantifiable confidence in estimates. Our work shows that the nature of performance variability across measurement granularities typically favors a large sample (e.g., 10,000 measurements) of brief (e.g., 1000-instruction) execution windows to minimize total simulation. According to Google Scholar, the original SMARTS publication [WWFH03] has been cited over 193 times (as of 1 Sep. 2008).

Stratified random sampling. Simple random sampling, as proposed in the SMARTS framework, yields accurate performance estimates with rigorous measures of statistical confidence. However, simple random sampling does not exploit the often repetitive behaviors of benchmarks, collecting many redundant measurements. Several studies propose exploiting program phase information to avoid these redundant measurements. We evaluate stratified random sampling, a sample design that eliminates redundant measurement of program phases while still computing statistical confidence for each result [WWFH04]. Our oracle limit study demonstrates substantial opportunity to reduce measurement by more than an order of magnitude through stratification. Unfortunately, we find that current phase detection approaches cannot identify phases at sufficiently-fine granularity to realize this opportunity.

Sampling for throughput applications. In contrast to uniprocessor applications, we cannot define a sampling population in terms of the dynamic instruction stream for multiprocessor applications. Instruction interleaving across processors varies over multiple application runs, and can cause changes in the dynamic instruction stream as races (e.g., for locks) resolve differently. Hence, for multiprocessor applications, we propose a new definition of the sampling population as the set of all reachable instruction interleavings and their occurrence probabilities [WWF+06]. This new definition allows us to extend the SMARTS measurement framework to multiprocessor *throughput applications* (e.g., transaction processing or web serving), where new transactions/requests are inserted into the system at random. The random nature of arriving work enables efficient construction of an unbiased sample in simulation. The practical impact of this research is that we can obtain performance results for commercial multiprocessor applications in a few hours of simulation rather than the CPU-years required without sampling.

Constructing warm simulation state.

Functional warming. The primary challenge in realizing the drastic acceleration promised by simulation sampling lies in rapidly constructing correct initial state for the large number of fine-grain performance measurements. This challenge is typically called the *warming problem*. Our first solution to this problem uses a simplified simulation model to maintain architectural and selected microarchitectural state while fast-forwarding between measurements, an approach we call *functional warming* [WWFH03][WWFH06c]. Functional warming produces accurate microarchitectural state two orders of magnitude faster than detailed simulation.

Live-points. Although functional warming provides accurate results, it does not realize the full potential of statistical sampling—the time spent fast-forwarding (99% of experiment turnaround time) still grows with benchmark length. Moreover, this warming approach precludes using the parallelism offered by compute clusters or multiprocessor simulation hosts to reduce turnaround time. Instead, we propose *live-points* as a replacement for functional warming to provide reduced simulation turnaround time, proportional to sample size, without sacrificing accuracy [WWFH05][WWFH06a][WWFH06b][WWFH06c]. A live-point stores the necessary data to reconstruct warm state for a brief simulation sampling execution window. Moreover, checkpoint-based sampling lets individual performance measurements be independent. Thus, a 1,000-checkpoint sample allows 1,000-way simulation parallelism. Our proof-of-concept implementation of live-points for uniprocessor simulation (based on SimpleScalar 3.0) produces performance estimates for SPEC CPU 2000 benchmarks in an average of only 91 seconds, faster than native hardware execution.

Full-system multiprocessor simulation.

Flexus. I led development of the Flexus simulation infrastructure [HSW+04][WWF+06]. 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. Flexus's component-based design allows simulation model detail to be tailored to the needs of a specific research hypothesis, while isolating code to ease development effort. Flexus includes designed-in support for simulation sampling, enabling rapid experimentation. Flexus is in use 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, including the University of Toronto, the University of Illinois at Urbana Champaign, the University of Minnesota, the University of Florida, and Intel Research Pittsburgh.

Other Achievements

Fellowships. For three of my six years at Carnegie Mellon, my tuition and stipend were fully supported by merit-based fellowships. Upon admission to Carnegie Mellon in 2001, I was awarded a Laboratory of Computer Systems Fellowship by the ECE department. In 2003, I was awarded the Intel PhD Research fellowship by the Intel Foundation. Finally, in 2004, my studies were supported by the Lamme/Westinghouse Graduate Fellowship.

Tutorials at ISCA & MICRO. In collaboration with Roland Wunderlich, I delivered tutorials at MICRO 2005 and ISCA 2006 that describe the SimFlex performance evaluation methodology research and Flexus simulation infrastructure. In total, nearly forty academic and industrial researchers attended the tutorials.

Funded NSF proposals. While at Carnegie Mellon, I drafted two NSF proposals. First, I drafted a proposal describing the multiprocessor aspects of the SimFlex performance evaluation methodology, titled "CSR—SMA: Fast and Accurate Simulation of Scalable Computer Systems." This proposal was awarded funding by NSF in 2005. Second, I drafted a proposal describing the unification of the temporal and spatial memory streaming research into Spatio-temporal Memory Streaming, which has also been awarded funding.

Languages. I am a native speaker of both English and German, and have a BA in German in addition to my engineering degrees. Throughout high school, I studied Spanish, and passed the International Baccalaureate Spanish exam as part of my IB diploma. At Carnegie Mellon, I studied Japanese for my first four semesters, achieving an intermediate level of proficiency before focusing my time exclusively on research.

Patents. Prior to coming to Carnegie Mellon, I worked as a software developer at American Power Conversions (maker of uninterruptible power supplies for IT equipment) in West Kingston, RI. While there, I was a co-inventor on two patent applications: (1) a security scheme to protect the integrity of HTTP traffic with Java-script-based MD5 hashes, (2) a system for cloning and delivering configuration information to network-attached uninterruptible power supplies. The former has been granted as US patent # 7,100,054.

Community leadership. I have been heavily-involved in the Engineering Graduate Organization (EGO) during my time at Carnegie Mellon. In 2004, I served as the lead organizer for the department's annual semi-formal Winter Party, an event serving more than 400 guests with an approximate budget of $35,000. In 2006, I served as EGO's Vice President.

References

[WWFH03] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling." *Proc. of the 30th International Symposium on Computer Architecture (ISCA)*, Jun. 2003.

[HSW+04] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, A. Nowatzyk. "SimFlex: A Fast, Accurate, Flexible Full-System Simulation Framework for Performance Evaluation of Server Architecture." *ACM SIGMETRICS Performance Evaluation Review (PER)*, Vol. 31, No. 4, Mar. 2004.

[SWH+04] S. Somogyi, T. F. Wenisch, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. "Memory Coherence Activity Prediction in Commercial Workloads." *Proc. of the 3rd Workshop on Memory Performance Issues (WMPI)*, Jun. 2004.

[WWFH04] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "An Evaluation of Stratified Sampling of Microarchitecture Simulations." *Proc. of the 3rd Workshop on Duplicating, Debunking, and Deconstructing (WDDD),* Jun. 2004.

[WWFH05] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe. "TurboSMARTS: Accurate Microarchitecture Simulation Sampling in Minutes." (Short paper) *Proc. of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS),* Jun. 2005.

[WSH+05a] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. "Temporal Streaming of Shared Memory." *Proc. of the 32nd International Symposium on Computer Architecture (ISCA)*, Jun. 2005.

[WSH+05b] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, C. Gniady, A. Ailamaki, and B. Falsafi. "Store-Ordered Streaming of Shared Memory." *Proc. of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2005.

[WWFH06a] T. F. Wenisch, R. E. Wunderlich, B. Falsafi and J. C. Hoe. "Simulation Sampling with Live-Points." *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2006.

[WWFH06b] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe. "Statistical Sampling of Microarchitecture Simulation." *Proc. of the 2006 Workshop on the NSF Next Generation Software Program (NGS),* Apr. 2006.

[SWA+06] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi and A. Moshovos. "Spatial Memory Streaming." *Proc., of the 33rd International Symposium on Computer Architecture (ISCA)*, Jun. 2006.

[WWFH06c] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. "Statistical Sampling of Microarchitecture Simulation." *ACM Transactions on Modeling of Computer Systems (TOMACS),* vol. 16, no. 3, Jul. 2006.

[WWF+06] 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.

[WAFM07] T. F. Wenisch, A. Ailamaki, B. Falsafi and A. Moshovos. "Mechanisms for Store-wait–free Multiprocessors." *Proc. of the 34th International Symposium on Computer Architecture (ISCA)*, Jun. 2007.