Seth H. Pugsley

Contact Information

School of Computing, University of Utah.
50 S. Central Campus Dr. Rm 3190
Salt Lake City, UT 84112

(801) 641-1057
pugsley@cs.utah.edu
http://www.cs.utah.edu/~pugsley

Research Interests

Computer Architecture: Memory Hierarchies, Hardware Prefetching, DRAM organization, Cache management, Cache coherence, Processing-in-Memory.

Education

University of Utah, Salt Lake City, USA

Ph.D. Candidate in Computer Science — **June 2008 - Present**
Advisor: Rajeev Balasubramonian — (GPA 3.712)

B.S. Computer Science — **June 2005 - May 2008**
(GPA: 3.685)

Refereed Publications

HASP 2014 *Memory Bandwidth Reservation in the Cloud to Avoid Information Leakage in the Memory Controller*
Akhila Gundu, Gita Sreekumar, Ali Shafiee, Seth Pugsley, Hardik Jain, Rajeev Balasubramonian, Mohit Tiwari.
Workshop on Hardware and Architectural Support for Security and Privacy (Held in conjunction with ISCA 2014)

IEEE Micro 2014 *Comparing Different Implementations of Near Data Computing with In-Memory MapReduce Workloads*
Seth H. Pugsley, Jeffrey Jestes, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, Feifei Li.
IEEE Micro Special Issue on Big Data, 2014

ISPASS 2014 *NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads*
Seth H. Pugsley, Jeffrey Jestes, Hui Hui Zhang, Alper Buyuktosunoglu, Vijayalakshmi Srinivasan, Feifei Li, Rajeev Balasubramonian, Al Davis.
2014 IEEE International Symposium on Performance Analysis of Systems and Software

HPCA 2014 *Sandbox Prefetching: Safe, Run-Time Evaluation of Aggressive Prefetchers*
Seth H. Pugsley, Chris Wilkerson, Zeshan Chishti, Troy Chuang, Robert Scott, Aamer Jaleel, Shih-Lien Lu, Kingsum Chow, Rajeev Balasubramonian.
The 20th IEEE International Symposium on High Performance Ccomputer Architecture (HPCA-20)

PACT 2010 *SWEL: Hardware Cache Coherence Protocols to Map Shared Data onto Shared Caches*
Seth H. Pugsley, Josef Spjut, David Nellans, Rajeev Balasubramonian.
19th International Conference on Parallel Architectures and Compilation Techniques (PACT-19), Vienna, September 2010.

UCAS 2009 *Optimizing a Multi-Core Processor For Message Passing Workloads*
Niladrish Chatterjee, Seth Pugsley, Josef Spjut, Rajeev Balasubramonian.
5th Workshop on Unique Chips and Systems, Boston, December 2009. (Held in conjunction with ISPASS 2009)

PACT 2008 *A Scalable and Reliable Communication for Hardware Transactional Memory*
Seth H. Pugsley, Manu Awasthi, Niti Madan, Naveen Muralimanohar, Rajeev Balasubramonian.
17th International Conference on Parallel Architectures and Compilation Techniques (PACT-17), Toronto, October 2008

Non-Refereed Technical Reports

UUCS-Tech-Report *USIMM: the Utah Simulated Memory Module*
Niladrish Chatterjee, Rajeev Balasubramonian, Manjunath Shevgoor, Seth H. Pugsley, Aniruddha N. Udipi, Ali Shafiee, Kshitij Sudan, Manu Awashti, Zeshan Chishti.
Technical Report UUCS-12-002, February 2012

UUCS-Tech-Report *Scalable, Reliable, Power-Efficient Communication for Hardware Transactional Memory*
Seth H. Pugsley, Manu Awashti, Niti Madan, Naveen Muralimanohar, Rajeev Balasubramonian.
Technical Report UUCS-08-001, January 2008

UUCS-Tech-Report *Commit Algorithms for Scalable Hardware Transactional Memory*
Seth H. Pugsley, Rajeev Balasubramonian.
Technical Report UUCS-07-016, August 2007

Professional Experience

Intel Corporation, Hillsboro, OR, USA
Graduate Intern Technical **May 2012 - May 2013**
Managers: Dr. Kingsum Chow (Intel SSG) and Dr. Shih-Lien Lu (Intel Labs)

Worked to find microarchitectural opportunities to benefit Java-based enterprise-class server applications, focusing on hardware prefetching techniques.

University of Utah, Salt Lake City, UT, USA
Undergraduate and Graduate Research Assistant **January 2008 - Present**

Developed simulation models for studying several aspects of computer architecture including:

- Hardware Transactional Memory
- Cache Coherence
- Caching Filling and Replacement Strategies
- On-chip Networks
- DRAM Systems
- Processing-in-Memory Architectures
- Hardware Prefetchers

Projects

Near-Memory Processing for MapReduce Workloads

While Processing-in-Memory has been investigated for decades, it has not been embraced commercially. A number of emerging technologies have renewed interest in this topic. In particular, the emergence of 3D stacking and the imminent release of Micron's Hybrid Memory Cube device have made it more practical to move computation near memory. However, the literature is missing a detailed analysis of a killer application that can leverage a Near Data Computing (NDC) architecture. This paper focuses on in-memory MapReduce workloads that are commercially important and are especially suitable for NDC because of their embarrassing parallelism and largely localized memory accesses. The NDC architecture incorporates several simple processing cores on a separate, non-memory die in a 3D-stacked memory package; these cores can perform Map operations with efficient memory access and without hitting the bandwidth wall. This paper describes and evaluates a number of key elements necessary in realizing efficient NDC operation: (i) low-EPI cores, (ii) long daisy chains of memory devices, (iii) the dynamic activation of cores and SerDes links. Compared to a baseline that is heavily optimized for MapReduce execution, the NDC design yields up to 15X reduction in execution time and 18X reduction in system energy, while also reducing the power budget.

For this project, I extended the Simics and USIMM simulators to model an NDC system, and I ported MapReduce kernels to C to work with our simulation infrastructure. **(Published at ISPASS 2014)**

Sandboxed Evaluation of Hardware Prefetchers

Memory latency is a major factor in limiting CPU performance, and prefetching is a well-known method for hiding memory latency. Aggressive prefetching can waste scarce resources such as memory bandwidth and cache capacity, therefore it is important to employ prefetching mechanisms that use these resources prudently, while still prefetching required data in a timely manner. In this work we propose a new mechanism to determine at run-time the appropriate prefetching mechanism for the currently running program, called Sandbox Prefetching. Sandbox Prefetching evaluates primitive offset prefetchers at run-time by adding prefetch address to a Bloom filter, rather than actually fetching the data into the cache. Subsequent cache accesses are tested against the contents of the Bloom filter to see if the aggressive prefetcher under evaluation could have accurately prefetched the data, while simultaneously testing for the existence of prefetchable streams. Real prefetches are performed according to which evaluated prefetchers are most accurate. Sandbox Prefetching improves performance across tested workloads by 47.6% compared to not using any prefetching, and by 18.7% compared to the Feedback Directed Prefetching technique. Performance is also improved by 1.4% compared to the Access Map Pattern Matching Prefetcher, while using considerably less logic and storage overheads.

For this project, I developed the concept of using a sandboxed Bloom filter environment to safely evaluate high risk/reward prefetchers without polluting caches or wasting memory bandwidth. **(Published at HPCA 2014)**

History-based Cache Insertion Policies

The last level cache is the CPU's last opportunity to avoid a costly off-chip main memory access, so maintaining a high quality last level cache can have large implications for system performance and energy use. A typical cache insertion and replacement policy consists of maintaining a few bits for each cache line denoting whether it was recently used or not, and not-recently used (NRU) cache blocks are evicted to make room for new cache blocks. We have flexibility in cache block insertion to decide what the initial value of these NRU bits will be, and making the right decision can have a large impact on the quality of the cache, by determining which cache blocks are more likely to be evicted in the near future. In this work we use a novel history structure in several different ways to make this determination for each cache block insertion, and another method for using this history stucture to influence cache block eviction victim selection. By using these methods we see a maximum performance improvement of 60%, and an average performance improvement of 11% over a baseline NRU cache insertion policy across all benchmarks tested.

For this project, I developed the idea of a decaying bloom history, which discretely ages out old members of a set, and we used this to track various cache behaviors. I also developed several cache insertion and replacement heuristics using decaying bloom histories.

Reclaiming lost DRAM locality in a many-core environment

Future multi-core processors will place increasing demands on the main memory system. As the number of cores increases, there is greater interference between threads and loss of locality within DRAM row buffers. Further, there is an increase in conflicts for banks, which leads to idle cycles on the data bus and wasted bandwidth. These problems are expected to worsen in PCM-based main memory systems because of the longer bank access times. This paper considers a number of approaches to overcome these problems. In essence, bank parallelism and row buffer locality can be promoted by increasing the number of banks and improving the scheduling of requests. We consider five optimizations: (i) page coloring to reduce inter-thread interference, (ii) partitioning a DRAM chip into more banks, (iii) partitioning a DIMM into more rank subsets, (iv) incorporating a row buffer cache, and (v) intelligent scheduling of writes. We thus evaluate a vast design space of bank management mechanisms for DRAM and PCM systems that range from software approaches (page coloring) to area-neutral system organizations (rank subsetting) to non-area-neutral hardware innovations. We are thus able to compare different approaches and show that with the right combination of optimizations, memory efficiency can be improved beyond that of traditional single-core systems at a minor cost.

For this project, I developed and modeled the page coloring address mapping scheme, I modeled heterogeneous rank subsetting, and I modeled the row buffer caches, as well as assisted in the evaluations of these schemes and others in various combinations.

Data Sharing Characterization for Low-complexity Cache Coherence

Snooping and directory-based coherence protocols have become the de facto standard in chip multiprocessors, but neither design is without drawbacks. Snooping protocols are not scalable, while directory protocols incur directory storage overhead, frequent indirections, and are more prone to design bugs. In this paper, we propose a novel coherence protocol that greatly reduces the number of coherence operations and falls back on a simple broadcast-based snooping protocol when infrequent coherence is required. This new protocol is based on the premise that most blocks are either private to a core or read-only, and hence, do not require coherence. This will be especially true for future large-scale multi-core machines that will be used to execute message-passing workloads in the HPC domain, or multiple virtual machines for servers. In such systems, it is expected that a very small fraction of blocks will be both shared and frequently written, hence the need to optimize coherence protocols for a new common case. In our new protocol, dubbed SWEL (protocol states are Shared, Written, ExclusivityLevel), the L1 cache attempts to store only private or read-only blocks, while shared and written blocks must reside at the shared L2 level. These determinations are made at runtime without software assistance. While accesses to blocks banished from the L1 become more expensive, SWEL can improve throughput because directory indirection is removed for many common write-sharing patterns. Compared to a MESI based directory implementation, we see up to 15% increased performance, a maximum degradation of 2%, and an average performance increase of 2.5% using SWEL and its derivatives. Other advantages of this strategy are reduced protocol complexity (achieved by reducing transient states) and signicantly less storage overhead than traditional directory protocols.

For this project, I developed the SWEL and R-SWEL cache coherence policies, and modeled them in our simulator, along with a fleshed out MESI scheme as a baseline.
(Published at PACT 2010)

Transactional Memory Lazy Commit Algorithms

In a hardware transactional memory system with lazy versioning and lazy conict detection, the process of transaction commit can emerge as a bottleneck. This is especially true for a large-scale distributed memory system where multiple transactions may attempt to commit simultaneously and coordination is required before allowing commits to proceed in parallel. In this paper, we propose novel algorithms to implement commit that are more scalable in terms of delay and are free of deadlocks/livelocks. We show that these algorithms have similarities with the token cache coherence concept and leverage these similarities to extend the algorithms to handle message loss and starvation scenarios. The proposed algorithms improve upon the state-of-the-art by yielding up to a 7X reduction in commit delay and up to a 48X reduction in network messages for commit. These translate into overall performance improvements of up to 66% (for synthetic workloads with average transaction length of 200 cycles), 35% (for average transaction length of 1000 cycles), and 8% (for average transaction length of 4000 cycles). For a small group of multi-threaded programs with frequent transaction commits, improvements of up to 8% were observed for a 32-node simulation.

For this project, I developed the lazy transactional memory commit algorithm schemes, and assisted in the development of a trace-based simulator that evaluated their performance on a CMP. **(Published at PACT 2008)**

Formal Verification of the SWEL Cache Coherence Protocol (Course proejct)

This project involved evaluating the claims made in the SWEL paper that SWEL and R-SWEL are both less complex than a traditional MESI protocol, and therefore easier to design, verify, and build. This was accomplished using the Murphi modeling language and model checker. I found that R-SWEL produced apprixmately half as many states as MESI, and SWEL produced approximately one quarter as many states as MESI.

Design of a MIPS-based processor with a G-share Branch Predictor (Course project)

This project involved running through the entire ASIC design flow, from design specifications to GDS II tapeout, using the Cadence DF II tool suite. The performance and accuracy of the branch-predictor was estimated. A 15-cell standard cell library was also designed and characterized.

Honors

HP Distinguished Graduating Senior (University of Utah, 2008). Awarded to one CS student per year for academic performance and undergraduate research.

First place in faculty-judged senior project software engineering contest (University of Utah, 2008).

Skills

Languages : C, Java, Verilog, 8085/86/51/MIPS Assembly

Full system simulators : Simics

Other simulators : Pin-based simulators, USIMM

CAD Tools : Cadence SOC Encounter, ICC CCAR, DFII Virtuoso, Abstract, Spectre; Synopsys Design Compiler

Service

Reviewer HiPC 2009, HPCA 2014

Coursework

Graduate : Computer Architecture, Parallel Computer Architecture, Advanced Computer Architecture and Memory Systems, Digital VLSI Design, Operating Systems, Compilers, Formal Methods in System Design, Embedded Systems

References

Dr. Rajeev Balasubramonian
School of Computing
University of Utah
rajeev@cs.utah.edu

Dr. Erik Brunvand
School of Computing
University of Utah
elb@cs.utah.edu

Dr. Feifei Li
School of Computing
University of Utah
lifeifei@cs.utah.edu

Dr. Al Davis
School of Computing

University of Utah
ald@cs.utah.edu