18-742 Project Statement (Fall 2012)

Important Dates

Proposal due: September 25, 2012, 11:59pm Eastern Time
Milestone 1 report/presentation due: October 17, 2012, 11:59pm Eastern Time
Milestone 1 meeting/presentations: TBD
Milestone 2 report/presentation due: November 16, 2012, 11:59pm Eastern Time
Milestone 2 meeting/presentations: TBD
Final presentation/posters: TBD (Tentatively, Final Exam Time)
Final report due: TBD (Tentatively, last day of final exams)
All reports and proposals should be submitted via blackboard as pdf files.

Introduction

In this project, you will improve the state-of-the-art and understanding in computer architecture by developing new architecture techniques or designing extensions to techniques presented in class or a recent architecture conference. The purpose of this project is to propose, conduct, and generate publication-quality research and to improve the state of the art. The project report will be in the form of a research article similar to those we have been reading and discussing in class. High-quality projects and reports will likely have a chance to be submitted to a top computer architecture conference (ISCA, MICRO, ASPLOS, HPCA) or systems (OSDI, SOSP, USENIX, USENIX Security, DSN, Hot*) conference.

The project will account for 40% of your final course grade. All components affect the grade.

Advice

When in doubt, meet with us during appointment or office hours. Conducting research is a time-consuming experience and choosing a topic you are excited about is an important part of it. Carefully choose your topic and write up your proposal after doing extensive literature search and consulting with the TAs and me.

Proposal

The proposal is a written two-to-three page document including at least the following:

1. The Problem: What is the problem you are trying to solve?
   a. Define clearly.
2. Novelty: Why has previous research not solved this problem? What are its shortcomings?
   a. Describe/cite all relevant works you know of and describe why these works are inadequate to solve the problem.
3. Idea: What is your initial idea/insight? What new solution are you proposing to the problem? Why does it make sense? How does/could it solve the problem better?
4. Hypothesis: What is the main hypothesis you will test?
5. Methodology: How will you test the hypothesis/ideas? Describe what simulator or model you will use and what initial experiments you will do.

6. Plan: Describe the steps you will take. What will you accomplish by Milestone 1, 2, and Final Report? Give 75%, 100%, 125% and moonshot goals.

All research projects can be and should be described in this fashion.

Milestone 1

In this milestone, you need to show preliminary experiments for your ideas. The format will be a short written report or a short presentation.

Milestone 2

The purpose of this milestone is to ensure that you are well ahead in executing your research plan. The format will be a short written report or a short presentation.

Final Report

You will hand in a report in the conference submission style. **Your final report should be formatted and written as if you are submitting the report to a top computer architecture conference.** The report should include the following:

- A descriptive title and author names
- Abstract
- Introduction (Problem and Motivation)
- Related Work and its Shortcomings
- Description of the Techniques (designed and evaluated)
- Evaluation Methodology
- Results and Analyses
- Conclusion/Summary
- Lessons Learned, Shortcomings, and Future Work
- Acknowledgements (if any)
- References (cited throughout the paper)

The page limit is 10 double-column, single-spaced pages. Make sure your document is spell-checked and grammatically sound.

Advice: The key characteristic that distinguishes a "good" research paper from a "bad" one is the existence of "clearly explained insight." I expect the highest quality of writing as well as clear descriptions of the ideas and techniques from the final report. Even if your research results in a "negative result" (not all research ideas pan out to be useful – in fact few of them do) writing it in a very insightful and intuitive manner can make the research very powerful and important. So, please do spend a significant amount of time and effort to ensure that your report is insightful and explains the ideas very clearly (possibly with examples and good analogies).

Final Presentation/Poster

In the last week of classes or during the final exam time we will hold presentation session(s) and a poster session in which teams get to present their projects and findings orally. More information on this will follow.

Best Projects

The top projects in class will be selected for submission to a computer systems conference for publication. In the past a number of papers from 740/741/742 have become full-blown research projects. For example, the TCM paper on fair and high-performance memory scheduling that appeared in MICRO 2010 and was selected to IEEE Micro Top Picks Issue was a 742 project in Spring 2010. The ATLAS paper on multiple memory controller scheduling that appeared in HPCA 2010 was a 741 project in Spring'09. Similarly, the HotNets 2010 paper on Application-Aware Congestion Control in On-Chip Networks was a class project in 741, Spring 2009. Other projects that appeared in top conferences include the SMARTS paper on simulation sampling that appeared in ISCA 2003, and the Spatial Pattern Prediction paper that appeared in HPCA 2004, and subsequently in ISCA 2006.

Evaluation Techniques and Infrastructures

You are welcome to use any infrastructure, tools, and benchmarks/applications that satisfy the needs of your project. However, you will need to justify why you use the infrastructure you choose. There are many simulation infrastructures available to perform experimental computer architecture research. Some of the tools you might find useful are:

- Internal CMU simulators (ask us)
- Simplescalar (contains timing models)
- PTLSim (contains timing models)
- GEMS (contains timing models)
- Pin dynamic binary instrumentation tool (no timing models)
- Simics (very crude timing models)
- Garnet (interconnection network simulator)
- Flexus (contains timing models)

These and other simulators can be found at http://www.cs.wisc.edu/arch/www/tools.html

Note that your proposed project does not *have to* use a simulator for evaluation. You can design real hardware, FPGA implementations, or even provide theoretical analyses. You only need to convincingly justify that your methodology is valid and satisfactory for the purposes of the project you propose.

Suggested (Sample) Research Project Topics

Many of the project topics discussed below are new ideas and are strictly confidential. Please do not distribute this list. If you make strides in any of these topics (and if you write it up well), you will likely be able to publish a good paper in a top computer architecture or systems conference.

While I provide this list for your benefit, I am open to any ideas you might have for a research project. In fact, you are encouraged to do a project that combines computer architecture with your area of expertise/strength (e.g., machine learning, theory, databases, etc). Please get in touch with me to discuss your proposal and ideas early if you are considering doing another architecture-related research project not listed here. I am open to project ideas in compilers, dynamic optimization, operating systems, parallel programming, circuits, and logic design as long as the ideas have interactions with computer architecture. I am also very open to entertaining ideas related to those you have been working on in your existing research areas, as long as you can show strong relationship to computer architecture.

New Projects Added in Fall 2012

Please see the hard-copy handout distributed in class.

New Projects Added in Spring 2011

1. **Analysis of critical sections in realistic workloads and development of an Amdahl's Law based analytical performance model that takes into account critical sections.** This analysis can be extended to include other synchronization mechanisms in programs such as barriers.

2. **Analysis of Javascript workloads and exploration of architectural alternatives to efficiently execute them.** This includes analysis of commonly used javascript applications (e.g., gmail) and design of rumtimes and architectures that can dynamically adapt to javascript program characteristics. Some references to start from:

   "Checked Load: Architectural Support for JavaScript Type-Checking on Mobile Processors", HPCA 2011 (to appear).

   "A Limit Study of JavaScript Parallelism", IISWC 2010.

3. **Memory controller designs for parallel applications and heterogeneous multicore systems (e.g., CPU+GPU).** Includes characterization of heterogeneous multicore applications.

4. **Parallelization of next-generation genome sequence analysis algorithms on GPUs, SIMD engines, and FPGAs (pick one or two).** Next-generation genome sequencing algorithms have significantly more computation requirements due to the very large amount of data throughput available from sequencing machines. Speeding them up with specialized and parallel architectures can lead to interesting results. A few interesting programs to understand and design GPU/SIMD algorithms and specialized hardware for are:

   Micro Read Fast Alignment Search Tools, **http://mrsfast.sourceforge.net/**

New Projects Added in Spring 2010

1. **Memory controller design for Phase Change Memory systems: Handling writes and providing QoS to different applications.** Much recent research has looked into designing main memory systems with Phase Change Memory technology. This technology has two important characteristics: 1) write latency is significantly higher than read latency, 2) writes cause a memory cell to wear out, and each cell has limited endurance. In the presence of these properties, how should a memory controller that controls PCM main memory be designed? In particular, how does such a controller perform prioritization between different threads/applications? For inspiration and starters, see:

   Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing. (pdf, slides)
   Moinuddin K. Qureshi, Michele Franceschini and Luis Lastras
   Appears in the International Symposium on High Performance Computer Architecture (HPCA) 2010.

   Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger,
   "Architecting Phase Change Memory as a Scalable DRAM Alternative"
   Proceedings of the 36th International Symposium on Computer Architecture (ISCA), pages 2-13, Austin, TX, June 2009. Slides (pdf)

   Scalable High-Performance Main Memory System Using Phase-Change Memory Technology. (pdf, slides)
   Moinuddin K. Qureshi, Viji Srinivasan and Jude A. Rivers
   Appears in the International Symposium on Computer Architecture (ISCA) 2009.

2. **Improving Data Locality in Multi-Core Systems using Core/Thread Migration.** Shared distributed caches are a scalable way of designing many-core systems. Much research has investigated how to attract recently/frequently used data to the core/thread that needs it. However, there is an alternative option: attract the thread(s) that need the data to the cores closeby the L2 banks that hold the data. The latter can provide significant improvements especially when many threads share a significant amount of data, by reducing the ping-ponging of shared dsata between different L2 cache banks or by reducing the replication of shared data. The goal of this project is to design realistic algorithms to accomplish this task. Talk to me for more information.

3. **Cycle Accounting and Slowdown Estimation in Multi-Core Memory Systems and Interconnects.** Understanding how much a thread slows down when it is run together with others compared to when it is run alone is important to account for how well the thread is performing in a shared-resource system. This cycle/slowdown accounting can be used for many purposes, e.g. enforcing slowdown fairness, performing prioritization based on which threads are "doing well," billing customers based on actual resources used. However, this is a tough task. The goal of this project is to develop mechanisms that estimate the slowdown incurred by threads in multi-core memory systems, including interconnects and memory controllers. SMT processors with realistic memory systems could also be investigated for this project. Some references:

   Per-Thread Cycle Accounting in SMT Processors
   Stijn Eyerman and Lieven Eeckhout
   Proceedings of ASPLOS 2009, pp 133-144, March 2009

   Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt,
   "Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems"
   to appear in in Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems ***(ASPLOS)***, Pittsburgh, PA, March 2010.

   Onur Mutlu and Thomas Moscibroda,
   "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors"
   Proceedings of the 40th International Symposium on Microarchitecture ***(MICRO)***, pages 146-158, Chicago, IL, December 2007. Slides (ppt)

4. **Prioritization policies for multithreaded applications in on-chip networks.** Several recent studies examined how to prioritize among different applications' requests in on-chip network packet schedulers. However, the question of "How to prioritize between different threads of a single application?" remains. The purpose of this project is to answer this question in a rigorous manner and develop a

comprehensive scheduling policy. The policy will be extended to hanle multiple multithreaded applications. For inspiration, please see:

Abhishek Bhattacharjee, Margaret Martonosi. "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors", *International Symposium on Computer Architecture (ISCA 2009)*, June 2009.(pdf)

Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions PDF
Q. Cai, J. Gonzalez, R. Rakvic, G. Magklis, P. Chaparro and A. Gonzalez. *17th Intl Conf on Parallel Architectures and Compilation Techniques* (PACT-17), Toronto, Canada, Oct 2008.

Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das, "Application-Aware Prioritization Mechanisms for On-Chip Networks" *Proceedings of the 42nd International Symposium on Microarchitecture (MICRO)*, pages 280-291, New York, NY, December 2009. Slides (pptx)

5. **Resource management in shared multi-core memory systems for multithreaded applications.** Cache, NoC, memory controller, memory management algorithms in many-core systems mainly focused on multiple different applications. The purpose of this project is to design caching, NoC, and memory controller management algorithms that intelligently manage resources between different threads of the **same** application. The policy will be extended to handle multiple multithreaded applications. For inspiration, please see:

   Abhishek Bhattacharjee, Margaret Martonosi. "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors", *International Symposium on Computer Architecture (ISCA 2009)*, June 2009.(pdf)

   Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions PDF
   Q. Cai, J. Gonzalez, R. Rakvic, G. Magklis, P. Chaparro and A. Gonzalez. *17th Intl Conf on Parallel Architectures and Compilation Techniques* (PACT-17), Toronto, Canada, Oct 2008.

   Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. (pdf, slides) Moinuddin K. Qureshi and Yale N. Patt.
   Appears in the International Symposium on Microarchitecture (MICRO) 2006.

6. **Request prioritization in on-chip networks: prefetches, writebacks, coherence messages, load/store demands, multiple applications, multiple threads.** While researchers have investigated how to prioritize requests from different applications in on-chip network packet schedulers, not much research has considered how to prioritize between different types of requests, e.g. prefetches, writebacks, coherence requests, load demand requests, store demand requests (while having multiple threads and multiple applications). The purpose of this project is to understand how a packet scheduler should prioritize between different types of requests and threads. For inspiration, take a look at the following papers:

   Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das, "Application-Aware Prioritization Mechanisms for On-Chip Networks" *Proceedings of the 42nd International Symposium on Microarchitecture (MICRO)*, pages 280-291, New York, NY, December 2009. Slides (pptx)

   Chang Joo Lee, Onur Mutlu, Veynu Narasiman, and Yale N. Patt, "Prefetch-Aware DRAM Controllers" *Proceedings of the 41st International Symposium on Microarchitecture (MICRO)*, pages 200-209, Lake Como, Italy, November 2008. Slides (ppt)

7. **Request prioritization in memory controllers: prefetches, writebacks, coherence messages, load/store demands, multiple applications, multiple threads.** Same as the above project except the focus is on memory controllers. The tradeoffs in DRAM controllers are very different, though.

8. **Page migration/management algorithms for hybrid Phase Change Memory + DRAM Systems.** If we have two types of memory (PCM, and DRAM), which have different read/write latencies, read/write energy, and wearout mechanisms, how should we dynamically manage virtual memory pages? Can we dynamically keep track of which pages benefit from being in one type of memory vs. the other? Based on this information, what is an algorithm that migrates pages from one type of memory to another? For example, pages that are written to frequently should probably be placed in DRAM instead of PCM. On the other hand, pages that are not latency-critical should likely be placed into PCM. What is a mechanism that achieves a good balance to maximize system throughput or energyxdelay product (or some other interesting metric)? For starters:

   Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative" *Proceedings of the 36th International Symposium on Computer Architecture (ISCA)*, pages 2-13, Austin, TX, June 2009. Slides (pdf)

Scalable High-Performance Main Memory System Using Phase-Change Memory Technology. (pdf, slides)
Moinuddin K. Qureshi, Viji Srinivasan and Jude A. Rivers
Appears in the International Symposium on Computer Architecture (ISCA) 2009.

G. Dhiman, R. Ayoub, T. Simunic Rosing, "PDRAM: A hybrid PRAM DRAM main memory system," DAC 09.

9. **3D memory systems: memory controller and cache design.** Talk to me. For starters, you can take a look at:

   Gabriel H. Loh
   3D-Stacked Memory Architectures for Multi-Core Processors (pdf)
   In the 35th ACM International Symposium on Computer Architecture (ISCA), pp. 453-464, June 21-25, 2008, Beijing, China.

   Dong Hyuk Woo, Nak Hee Seong, Dean L. Lewis, and Hsien-Hsin S. Lee. "An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth." In Proceedings of *the 16th International Symposium on High-Performance Computer Architecture*, Bangalore, India, January, 2010. [pdf] [slides]

Old Project List from Spring 2009

1. **QoS and Fairness in Bufferless Interconnection Networks:** Bufferless on-chip networks have been proposed to save energy and area consumption due to large buffers used in routing [see Moscibroda and Mutlu tech report and ISCA 2009 submission] (ask me for copies). These networks use “deflection routing” to deflect network packets instead of buffering them when there is contention for a particular output port. Because these networks do not make use of per-flow buffers, they cannot distinguish (or so far have not distinguished) between different flows. One important open problem that needs to be solved to make such networks viable is to design efficient techniques that can prioritize different flows and ensure bandwidth or QoS guarantees to different flows. For example, thread A could be allocated 50% of the network bandwidth, whereas thread B could be allocated 10% (similar allocations can be promised in terms of latency). How does the routing mechanism guarantee this allocation when no routing buffers are employed? What are the modifications needed for the routing algorithms? Is there a need for an “admission control” policy that ensures that some bandwidth allocations will not be violated? How is this policy designed and how can it be implemented? Can the classic “network fair queueing” concepts be used and modified in bufferless routing? Please talk with Onur Mutlu, the TAs, or Yoongu Kim about this problem. The following papers could help you get started on this topic:

Moscibroda and Mutlu, "*A Case for Bufferless Routing in On-Chip Networks*," MSR Technical Report 2008 (extended version submitted to ISCA 2009)
http://www.ece.cmu.edu/~omutlu/pub/tr-buffless.pdf

Lee, Ng, and Asanovic, "*Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks*," ISCA 2008.
http://www.icsi.berkeley.edu/pubs/arch/qos-isca2008.pdf

For a significantly extended and more detailed version of the first paper with better experimental methodology, please contact me.

2. **Preserving bank parallelism in multiple multi-core memory controllers:** When multiple cores share a single memory controller, the "memory level parallelism" of each core can be destroyed because the memory controller can service each core's concurrent requests serially in the banks rather than in parallel. Mutlu and Moscibroda proposed a solution to this problem in ISCA 2008 in their "Parallelism-aware batch scheduling (PAR-BS)" paper. The basic idea in that paper is to service requests in "batches" and within each batch to service one thread's requests after another (in a shortest job first manner) such that bank-level parallelism of each thread is maintained as much as possible (threads are ranked in a shortest-job-first manner to accomplish this). This approach works well with a single memory controller. The purpose of this project is to explore how this approach can be extended and made to work with *multiple* memory controllers (i.e. when there are multiple memory controllers shared by cores, where each memory controller controls a statically assigned portion of DRAM memory). One solution is to simple employ PAR-BS independently in each controller without any communication. However, this approach might not work well because the ranking of threads might be different in each controller. Another solution is to design a meta-controller that determines the thread ranking and batching in a global manner for all controllers (and enforces it). These two solutions are opposite extremes in a continuum and alternate designs are possible. How does such a meta-controller work? What is the hardware algorithm used for ranking and batching? When does a batch start and end? Are the batches aligned across different controllers? Your goal is to examine the tradeoffs in designing multiple memory controllers to preserve memory-level parallelism and experimentally evaluating different designs. Talk with Onur Mutlu, the TAs, or Yoongu Kim about this problem. The following paper is a prerequisite for performing this project:

   Mutlu and Moscibroda, "*Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems*," in ISCA 2008.
   http://www.ece.cmu.edu/~omutlu/pub/parbs_isca08.pdf

3. **Approximate computing: application analysis and microarchitecture design:** Processors are designed to produce correct results. However, not all software algorithms require absolute correctness. For example, when you are watching a video, you might not notice it if a pixel is rendered in a wrong color. Can we take advantage of this fact to relax the design constraints of processors such that we can simplify the design and/or improve performance? For example, can the processor ignore some cache misses and assume that a cache miss always returns some value (without checking the correctness of the returned value)? This would reduce the performance impact of some cache misses because the processor can simply supply a (possibly predicted) data value for that cache miss rather than fetching the needed data from main memory. Similarly, can the processor ignore some dependencies between store and load instructions? This could

simplify the design of the load-store queue which ensures that memory dataflow is correctly maintained in a sequential program. As a final example, can the processor ignore some instructions, for example some branch mispredictions? It might be that executing the wrong path produces acceptable output and the processor could save time by not recovering from a branch misprediction? How and when are these aggressive optimizations possible to do?

In this project, you will examine applications and determine the feasibility of the aforementioned (and similar) ideas. The ideal tool for this project is perhaps the Pin dynamic binary instrumentation tool, where you can modify the data values of any memory instruction and examine the output of the program. The following papers could help you get started on this topic (pay attention to the methodology used in these papers – that will come in very handy in doing your research project):

Wang, Fertig, and Patel, "*Y-Branches: When You Come to a Fork in the Road, Take It,*" in PACT 2004.
http://www.crhc.uiuc.edu/ACS/pub/branchflip.pdf

Li and Yeung, "*Application-Level Correctness and its Impact on Fault Tolerance*," in HPCA 2007.
http://maggini.eng.umd.edu/pub/faults-HPCA2007.pdf

This project is very open ended. You are encouraged to stretch your imagination and think about what aspects of hardware or software can be simplified if you can avoid the "strict correctness" requirement. While I would prefer you to pick an optimization enabled by "relaxed correctness" and examine it in detail (i.e. when is this optimization possible, how should the code be written to make it possible, what are the benefits, what are the downsides, etc), you can examine multiple optimizations (e.g. ignoring both cache misses and memory dependencies) I am open to software versions of this project (i.e. what can you relax in the software if things do not need to be strictly correct) as well, if you can specify the project reasonably well.

4. **On-chip Security (denial of service):** Shared resources among threads/cores in a multi-core multi-threaded system presents a vulnerability: if the sharing of resources is left uncontrolled, one thread can deny service to another (or all others) and cause starvation or possibly in severe cases deadlock. In fact, malicious programs can be written to destroy the performance of other programs by exploiting the behavior of controllers controlling access to shared resources. Previous research demonstrated this problem in shared memory controllers (see Moscibroda and Mutlu, USENIX Security 2007), trace caches shared by multiple threads in SMT processors (see Grunwald and Ghiasi, MICRO 2002), and shared power management (see "Heat Stroke" paper by Hasan and Vijaykumar in HPCA 2005). The goal of this project is to demonstrate the possibility and degree of such denial of service attacks in other shared system resources (and hopefully suggest/design techniques to mitigate such attacks) in multi-core and multi-threaded processors. The shared resources you might want to examine include but are not limited to:
   - *On-chip interconnects* (buses, 2D mesh designs, rings)
   - Flash memories
   - On-chip thermal/power controllers (e.g. thermal management mechanism of shared memory controllers)

- Shared floating-point units?
- Shared controller buffers on other chips (e.g. in a distributed shared memory system like AMD Athlon)
- Shared disk

Your goal is to 1) demonstrate the problem in a cycle-accurate simulator or (preferably) in a real system using microbenchmarks as well as real applications, 2) quantify the degree of the problem (how bad the denial of service can be), 3) describe how malicious applications can be designed, 4) suggest and possibly evaluate ways of solving the problem. In my opinion, examining on-chip interconnects is the most promising, but you might find other resources you are interested in. The following papers could get you started thinking in the right direction:

Moscibroda and Mutlu, "*Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems*," USENIX SECURITY 2007.
http://www.ece.cmu.edu/~omutlu/pub/mph_usenix_security07.pdf

Woo and Lee, "*Analyzing Performance Vulnerability due to Resource Denial-of-Service Attack on Chip Multiprocessors*," CMP-MSI 2007.
http://arch.ece.gatech.edu/pub/cmpmsi07.pdf

Grunwald and Ghiasi, "*Microarchitectural denial of service: insuring microarchitectural fairness*," MICRO 2002.
http://portal.acm.org/citation.cfm?doid=774861.774905

Hasan et al., "*Heat Stroke: Power-Density-Based Denial of Service in SMT*," HPCA 2005.
http://cobweb.ecn.purdue.edu/~vijay/papers/2005/heatstroke.pdf

Lee, Ng, and Asanovic, "*Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks*," ISCA 2008.
http://www.icsi.berkeley.edu/pubs/arch/qos-isca2008.pdf

5. **Fault tolerance: Architectural techniques for efficiently recovering from detected hardware design bugs:** Techniques have been proposed to detect at run-time hardware design bugs that were not caught during the testing of a microprocessor. These techniques all assume that once a bug is detected, some recovery technique can recover from the effects of the bug and place the system into a consistent state. However, specific mechanisms for accomplishing this recovery have not been researched or designed in detail. The goal of this project is to design techniques that enable efficient recovery from the unwanted effects of a processor design bug. The project has three components, any of which can be proposed as independent projects:
    - As a first step, rigorously analyze and classify real design bugs to gain insights into their behavior. What makes a bug detectable? What makes it recoverable? What types of bugs should be detected and recovered from? What mechanisms are needed for different types of bugs? Initial analysis of simple logic bugs by Constantinides, Mutlu, and Austin [MICRO 2008] suggests mechanisms for different bug types (e.g., algorithmic vs. logic) should be very different. At the end of your analysis, you might end up with different classes of bugs based on their recoverability.

- Develop new logic that can efficiently recover from the effects of a detected bug. What are the recovery techniques for design bug types? For example, an algorithmic bug might almost always require good reconfiguration support, but some logic bugs can be avoided with simple re-execution.
- A realistic evaluation of both the coverage and performance impact of design bug detection/recovery mechanisms.

This project is a relatively low-level project which will very likely be enhanced by careful and detailed examination of the RTL source code of an open source processor. The experimental methodology can also be significantly enhanced with FPGA implementation of a simple open-source processor and injection of different types of design bugs (I recommend this only if you have previous experience in FPGA implementations – Michael Papamichael and Eric Chung can help you with this). The following papers can help you get started on this problem:

Constantinides, Mutlu, and Austin, "*Online Design Bug Detection: RTL Analysis, Flexible Mechanisms, and Evaluation*," MICRO 2008.
http://www.ece.cmu.edu/~omutlu/pub/hwbugs_micro08.pdf

Wagner, Bertacco, and Austin, "*Using Field-Repairable Control Logic to Correct Design Errors in Microprocessors*," IEEE TCAD 2008.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04358502

Sarangi, Tiwari, and Torrellas, "*Phoenix: Detecting and Recovering from Permanent Processor Design Bugs with Programmable Hardware*," MICRO 2006.
http://iacoma.cs.uiuc.edu/iacoma-papers/micro06_phoenix.pdf

Narayanasamy, Carneal, and Calder, "*Patching Processor Design Errors*," ICCD 2006.
http://www.cse.ucsd.edu/~calder/papers/ICCD-06-HWPatch.pdf

Man-Lap Li et al., *"Understanding the propagation of hard errors to software and implications for resilient system design"*, ASPLOS 2008.
http://portal.acm.org/citation.cfm?id=1346281.1346315&coll=ACM&dl=ACM&CFID=18308174&CFTOKEN=57845610

6. **Asymmetric multi-core designs:** An asymmetric multi-core system consists of one or more powerful cores augmented with a large number of small cores. Large cores have been proposed to use serialization due to the serial portions of programs (think Amdahl's law) and serialization due to contention in critical sections (see Suleman et al. ASPLOS 2009). The basic idea of the latter work is to execute a critical section in a large core such that the likelihood of another thread waiting for that critical section to complete is reduced. This work did not examine the following:
   - When is it profitable to ship a critical section to a large core? Your job is to develop a cost-benefit model that decides whether it is beneficial to ship a critical section to a large core or it is better to execute it on a small core. The cost-benefit model can include dynamically-changing information about the characteristics of critical sections, contention for them, as well as speedup obtained by executing the critical section.
   - When is it profitable to execute the "serial bottleneck" on the large core? This is a simpler problem because in the "serial bottleneck" there are no threads running.

However, previous work always assumed that it is better to execute the serial bottleneck on the large core. Your task is to challenge this assumption and develop a model for deciding when the serial bottleneck should be shipped to the large core.

- The effect of running multiple applications on the same chip. If multiple applications are running concurrently, then how should the operating system or hardware allocate the large core? What are the decision mechanisms that are needed to optimize overall system throughput while respecting fairness? What happens if one application is executing in a critical section and another is executing its serial bottleneck? How does the hardware decide which application should use the large core? What are the required cost-benefit analyses or heuristics that can efficiently determine the allocation decision to be made by the OS/hardware?
- ***Very open ended:*** For what other purposes can a large core be used in an asymmetric multi-core system? If you have ideas and are ready for a more open-ended project, I encourage you to work on this problem. Talk with me if you would like some hints.

Understanding the following papers (especially the first one) would help a lot with this project:

Suleman et al., "*Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures*," ASPLOS 2009. http://www.ece.cmu.edu/~omutlu/pub/acs_asplos09.pdf

Hill and Marty, "Amdahl's Law in the Multicore Era," http://www.cs.wisc.edu/multifacet/papers/ieeecomputer08_amdahl_multicore.pdf

Suleman et al., "*Asymmetric Chip Multiprocessors: Balancing Hardware Efficiency and Programmer Efficiency*," UT-Austin HPS Technical Report, 2007. http://projects.ece.utexas.edu/hps/pub/TR-HPS-2007-001.pdf

7. **Temporal and Spatial Streaming in Disks:** Temporal and Spatial streaming have been used to accurately prefetch data in memory systems. The key idea in these approaches is to use the fetch of a stream of data blocks as an indicator that a larger stream will need to be fetched in the future. The goal of this project is to examine the feasibility of these approaches within the context of I/O systems and apply it to prefetching disk blocks rather than cache blocks (remember that the latency of a disk access is actually much larger than the latency of a memory access – hence the need for a memory hierarchy). You are expected to analyze and characterize the access behavior of real applications to the disk and develop temporal/spatial streaming (or similarly inspired) techniques to efficiently prefetch disk blocks into the buffer cache before they are needed. In addition to me and the TAs, you are welcome to talk with Stephen Somogyi about this project. Actual disk traces from Windows can be used for this project (talk to me about this). The following papers are required reading if you are interested in this project:

Wenisch et al., "*Temporal Streaming of Shared Memory*," ISCA 2005,
http://www.eecs.umich.edu/~twenisch/papers/isca05.pdf

Somogyi et al., "*Spatial Memory Streaming*," ISCA 2006,
http://www.eecg.toronto.edu/~moshovos/research/sms.pdf

Good characterization of memory streams:
Wenisch et al., "*Temporal streams in commercial server applications*," IISWC 2008,
http://www.iiswc.org/iiswc2008/Papers/010.pdf

8. **QoS aware (fair) memory controllers in the presence of prefetching:** Previous work on fair multi-core memory controllers did not take into account the existence of prefetch requests. They assumed every request was a load request (and sometimes store request as well). This is a shortcoming because most real systems employ aggressive stream or stride-based prefetching which can affect the fairness in the memory system. Your task in this project is to understand the effect of memory prefetching on memory system fairness. Based on this understanding, you will extend existing fair memory control schemes (such as "Stall time fair memory scheduling, MICRO 2007" or "Parallelism-aware batch scheduling, ISCA 2008" to take into account prefetch requests (and to take into account store requests better). Alternatively you can devise a new scheme from the ground up that takes into account prefetch requests. Talk to me or Yoongu Kim about this problem for more detail. Some required reading:

   Mutlu and Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," in ISCA 2008.
   http://www.ece.cmu.edu/~omutlu/pub/parbs_isca08.pdf

   Mutlu and Moscibroda, "Stall-time Fair Memory Access Scheduling for Chip Multiprocessors," in MICRO 2007.

   http://www.ece.cmu.edu/~omutlu/pub/stfm_micro07.pdf

   Lee et al., "Prefetch-aware DRAM Controllers," in MICRO 2008,
   http://www.ece.cmu.edu/~omutlu/pub/prefetch-dram_micro08.pdf

9. **Fair/QoS-aware caching in the presence of prefetching:** Previous work on fair multi-core caches did not take into account the existence of prefetch requests. They assumed every request was a demand request. This is a shortcoming because most real systems employ aggressive stream or stride-based prefetching which can affect the fairness in shared caches. Your task in this project is to understand the effect of memory prefetching on shared cache fairness. Based on this understanding, you will extend existing fair caching schemes (such as "Utility based cache partitioning, MICRO 2006" or "Fair cache sharing and partitioning, PACT 2004" to take into account prefetching. Alternatively you can devise a new caching scheme from the ground up that takes into account prefetch requests. Some reading that can help:

Qureshi and Patt, "*Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches*," in MICRO 2006, http://users.ece.utexas.edu/~qk/papers/util.pdf

Kim, Chandra, and Solihin, "*Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture*," PACT 2004, http://www.ece.ncsu.edu/arpers/Papers/faircaching.pdf

10. **Improving memory fairness and memory-level parallelism via dynamic physical-to-bank (or virtual-to-physical) address mapping:** In the main memory system, programs interfere with each other because the data they need are mapped to the same banks or same channels (see publications on fair memory controllers). One way to reduce these "collisions" among different programs is to detect them dynamically and change the bank mapping of frequently-colliding data elements by adjusting either 1) virtual-to-physical page mappings at the OS layer or 2) physical-to-bank mappings at the memory controller. The purpose of this project is to examine such strategies to reduce unfairness and inter-thread collisions in the memory system as well as to improve intra-thread bank-level parallelism. You will develop algorithms and mechanisms that detect collisions among different threads and dynamically change the mappings such that collisions (bank, channel, or bus) are minimized across the entire memory system. Note that changing a page mapping incurs overhead of data transfer so your mechanism will likely have to take into account this overhead cost of mapping adjustments. You will examine both OS-level and hardware-level (OS-transparent) techniques to perform page migration.

    Similar techniques can be employed to improve the performance of a single thread. The memory controller can keep track of bank conflicts incurred by accesses from the same thread. If accesses are serialized too often because they go to the same bank, the memory controller can change the bank mapping of the colliding pages to ensure that accesses to those pages are serviced in parallel. A possibly separate but related project idea is to develop techniques to improve the bank-level parallelism of a single thread.

11. **Prefetching in a distributed cache system:** Non-uniform access caches whose banks are distributed across a 2D-mesh network were proposed in past research to reduce average cache access time. Ideally, the data that will be accessed soon should be placed in a cache bank that is close to the processor that needs to access the data soon. To accomplish this, you will design targeted prefetching techniques that prefetch data between the cache banks of a NUCA (non-uniform cache access) cache. The first step is to evaluate some existing prefetching mechanisms within the context of NUCA, document and describe their shortcomings, and develop better prefetching techniques that consider the movement of data between the banks. You can also develop new interconnection networks that enable better prefetching in a NUCA cache. The key question to keep in mind is "If there are multiple processors sharing the NUCA cache, how should prefetching techniques be modified to account for this sharing?" Talk to Nikos Hardavellas if you are interested in this topic. The following publications could help you develop the required background:

    Kim, Burger, and Keckler, "*An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches*," ASPLOS 2002. http://www.cs.utexas.edu/users/ckkim/papers/nuca_dist.pdf

Beckmann, Marty, and Wood, "*ASR: Adaptive Selective Replication for CMP Caches*," MICRO 2006, http://www.cs.wisc.edu/multifacet/papers/micro06_asr.pdf

12. **Benchmark development (Pointer-intensive Applications):** Computer architecture development and research critically depends on the quality of benchmarks. Dhrystone, SPEC, synthetic benchmarks, Livermore Loops, TPC benchmarks, etc have shaped the way in which architectures have evolved (and have been criticized as well). In this project, your task is to develop a modern, *pointer-intensive (preferably parallel) benchmark suite* that stresses the memory system. The benchmarks you will use should be realistic and representative (as much as possible) of real workloads. You will characterize the applications, describe the way in which pointer-based data structures are actually used, and suggest techniques to improve their memory system performance based on this characterization. Previously developed pointer-intensive benchmark suite (Olden) mainly consists of very small kernels, which are somewhat outdated. A good new suite developed for pointer-intensive workloads with good characterization (especially of the memory system, data sharing, and branching behavior) and analysis could be made public and possibly used by a large number of researchers. I suggest you start studying the source code as well as publications related to that suite:

    Rogers et al., "*Supporting dynamic data structures on distributed memory machines*," ACM TOPLAS 1995.

    Woo et al., "*The SPLASH-2 Programs: Characterization and Methodological Considerations*," ISCA 1995. http://www.csd.uoc.gr/~hy527/papers/splash2_isca.pdf

    Luk and Mowry, "*Compiler-based prefetching for recursive data structures*," ASPLOS 1996. http://www.ckluk.org/ck/luk_papers/luk_asplos96.pdf

    Mutlu, Kim, Patt, "*Address-Value Delta (AVD) Prediction: A Hardware Technique for Efficiently Parallelizing Dependent Cache Misses*," IEEE TC 2006. http://www.ece.cmu.edu/~omutlu/pub/mutlu_ieee_tc06.pdf

    Bienia et al., "*The PARSEC Benchmark Suite: Characterization and Architectural Implications*," PACT 2008. http://parsec.cs.princeton.edu/publications/bienia08characterization.pdf

13. **Benchmark development (Artificial Intelligence and Robotics Related Applications):** See above (project 12) for motivation. AI and Robotics related programs have not been specifically analyzed in computer architecture or workload characterization research. Your task will be to analyze AI/robotics-related applications and develop/characterize a benchmark suite that represents the important AI/robotics applications in use today. You will also motivate the uses of such applications in real workloads (This motivates the development of your benchmark suite and argues why it is important). Characterization of the applications should be insightful: it should provide insights into the memory, branching, sharing, locking, scaling behavior of applications and provide hints into how to design a system that executes these applications well. Intel's RMS (Recognition, Mining, and Synthesis) suite contains some AI-flavored kernels, so I would suggest you take a look at that suite. Other publications that would be of interest are:

Woo et al., "*The SPLASH-2 Programs: Characterization and Methodological Considerations*," ISCA 1995. http://www.csd.uoc.gr/~hy527/papers/splash2_isca.pdf

Bienia et al., "*The PARSEC Benchmark Suite: Characterization and Architectural Implications*," PACT 2008. http://parsec.cs.princeton.edu/publications/bienia08characterization.pdf

14. **Bufferless Interconnection Networks - Implementation:** Bufferless routers are a promising way to interconnect cores and caches in an energy- and area-efficient manner. The goal of this project is to design a bufferless (or lightly buffered) pipelined router from scratch on an FPGA and demonstrate its performance, power, and QoS characteristics using traces fed into the implementation (or even real applications). You will design the router pipeline and deal with the timing issues arising from the lack of buffers in routing. The router will be aggressively clocked. Once the basic design is complete, you will extend it with enhancements to improve the performance of the bufferless router (for example, reduce the number of pipeline stages or add a small amount of buffering to improve bandwidth efficiency in the presence of congestion). You will implement both flit-level packet-switched and wormhole-based versions of bufferless routing (as well as any other versions you choose to develop as long as you justify their benefits). The end goal is to demonstrate the feasibility and characteristics of bufferless routing in a 2D-mesh topology. The following publications should get you started:

    Moscibroda and Mutlu, "*A Case for Bufferless Routing in On-Chip Networks*," MSR Technical Report 2008 (extended version submitted to ISCA 2009) http://www.ece.cmu.edu/~omutlu/pub/tr-buffless.pdf

    Konstantinidou and Snyder, "*The Chaos Router*," IEEE TC 1994, http://portal.acm.org/citation.cfm?id=191594

    For a significantly extended and more detailed version of the first paper with better experimental methodology, please contact me.

15. **Predictive DRAM row buffer management:** Some memory access patterns benefit from keeping the row buffer open, others benefit from keeping it closed (or proactively opening rows). Design a prediction mechanism that more intelligently manages the DRAM row buffers compared to simply using a static (and rigid) open-row or closed-row policy. Your mechanism can proactively determine when a row should be closed; proactively predict which rows should be opened, etc. You can get inspiration from prefetching mechanisms to design your own row-buffer management policy. In your design, you should keep in mind the complex parallelism and locality issues present in the DRAM system. You will also need to examine interactions of your developed row buffer management policy with the scheduling policy of the memory controller (they will very likely not be independent of each other). Your management policy needs to take into account the existence of multiple cores sharing the DRAM banks; however, I would suggest starting with a single core. See memory controller related publications for inspiration and background:

Rixner et al. "*Memory Access Scheduling*", ISCA 2000.
http://www.cs.rice.edu/~rixner/rixner_isca27.pdf

16. **Bufferless Interconnection Networks – Parallel Applications:** Previous work on bufferless routing in on-chip networks did not consider parallel applications. Your task in this project is to evaluate the effectiveness of bufferless networks with parallel applications such as SPLASH-2, NAS, and PARSEC benchmarks. Based on your analyses, you will develop new bufferless (or lightly buffered) routing algorithms that can better accommodate the communication properties of shared-memory parallel applications. One important question you might need to answer is: How can a bufferless network efficiently support broadcast/multicast as well as invalidation/coherence traffic?

    Moscibroda and Mutlu, "A Case for Bufferless Routing in On-Chip Networks," MSR Technical Report 2008 (extended version submitted to ISCA 2009)
    http://www.ece.cmu.edu/~omutlu/pub/tr-buffless.pdf

    For a significantly extended and more detailed version of the first paper with better experimental methodology, please contact me.

17. **Cost-benefit analysis driven prefetching in multi-core systems with shared resources:** In a multi-core system, each core contains one or more (hardware) prefetchers to reduce memory access time. These prefetchers share valuable system resources such as memory bandwidth, cache bandwidth, and shared cache space. Existing systems do not take into account the interactions between the prefetchers on different cores. The prefetcher of one core can deny service to demands of another core by hogging the DRAM or cache bus, or it can displace useful cache blocks of another core in the shared cache. As a result, system performance can degrade compared to employing no prefetching. In this project you will design mechanisms to control the interference caused by prefetchers of cores in the shared memory system resources. The basic idea is to develop a cost-benefit model based on which the hardware can adjust the aggressiveness of each prefetcher. Each prefetcher (at an aggressiveness level) provides some benefit but also incurs some cost on system performance. You will devise a model that predicts this cost-benefit based on prefetcher accuracy, coverage, timeliness, pollution, etc (and other possible characteristics). Based on the outcome of the model, the hardware should decide on an aggressiveness level for each prefetcher (say, for the next program phase or time interval). Note that previous research (under submission) used heuristics based approaches to achieve the same goal, but not a model-based approach as described here. Previous research on adjusting the aggressiveness of a single-core prefetcher also used heuristics (Srinath et al. HPCA 2007).
    The following papers would be good to gain background on this topic:

    Ebrahimi, Mutlu, and Patt, "*Coordinated Management of Multiple Prefetchers in Multi-Core Systems*," under submission to ISCA 2009. Email me for a copy and discussion.

    Ebrahimi, Mutlu, and Patt, "*Techniques for Bandwidth-Efficient Prefetching of Linked Data Structures in Hybrid Prefetching Systems*," HPCA 2009.
    http://www.ece.cmu.edu/~omutlu/pub/bandwidth_lds_hpca09.pdf Section 4 is especially relevant.

Srinath, Mutlu, Kim, and Patt, "*Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers*," HPCA 2007. http://www.ece.cmu.edu/~omutlu/pub/srinath_hpca07.pdf

18. **Transactional memory - intelligent contention management:** In transactional memory, if a data conflict is detected between two transactions, usually one of the transactions is aborted, and the other is allowed to continue. This is called contention management. Existing contention management policies usually do not take into account the importance and length of the thread that is executing the transaction. As a result they might make choices that are suboptimal in terms of system throughput as well as QoS. The goal of this project is to design a "thread-aware" contention management policy for transactional memory. The policy is supposed to maximize system thread throughput (how?) while ensuring non-starvation/fairness and being flexible enough to satisfy different QoS requirements (such as thread priorities). You will design both the software and hardware support required to support such a policy. See the following papers for related work:

    Scherer and Scott, "*Advanced Contention Management for Dynamic Software Transactional Memory*," PODC 2005. http://www.cs.rice.edu/~wns1/papers/2005-PODC-AdvCM.pdf

    Chafi et al. "*A Scalable, Non-blocking Approach to Transactional Memory*," HPCA 2007. http://csl.stanford.edu/~christos/publications/2007.scalable_tcc.hpca.pdf

19. **Denial of Service via Wearout in Flash and Phase-Change Memory Systems:** Flash memory and Phase Change Memory are promising non-volatile memories. Unfortunately, these memories have one shortcoming: the storage cell wears out with each write operation. As a result, over time, the ability of a storage cell to store digital data degrades and diminishes. To improve the lifetime of such memories, flash controller architects have developed techniques called "wear-leveling" which tries to equalize the number of write operations to each memory cell. Many wear-leveling techniques use a level of indirection (between the physical address and the actual bank address) to migrate blocks that have been written to too much. The goal of this project is to design malicious programs that can wear out Flash and PCM memories very quickly by subverting the defense mechanisms used by the wear-leveling techniques. You will study existing wear-leveling techniques and show in a real Flash-based system how to write a program that achieves this purpose. Then, you will develop better and more robust wear-leveling techniques that are not vulnerable to malicious programs that can quickly wearout memories. The problem becomes more severe if PCM memory is used as an alternative to DRAM memory. You will demonstrate the attacks and defenses you develop via simulation of a modern memory system employing PCM memory instead of DRAM as the main memory of the system. Talk to me if you are interested in this project.

20. **Application scaling analysis on multi-core engines (suggested by Brian Gold @ CMU):** The goal is to study OS and application scaling on multi-core architectures. Example applications include web servers, database servers, etc. You are expected to analyze and report on contention for shared resources (e.g., locks) and propose solutions to alleviate contention. Build analytic models of performance as number of cores scales, memory hierarchy changes, etc. and test models in both real hardware and simulated machines. Talk to me and Brian Gold about this project.

21. **Improving Idle Power Management in Servers (suggested by Brian Gold @ CMU):** PowerNap is a recently proposed system for cutting server idle power: entire servers (CPUs, memory, disk, etc.) are put into a low power state as soon as the OS task queue is empty. Implement key software components of PowerNap (see paper link below), and report on any practical limitations of this proposed work. In particular, implement suspend and resume mechanisms in the Linux scheduler and report on average wake and sleep transitions [NOTE: requires a server w/ solid-state disk, a modern CPU, and power-saving DRAM]. Measure performance impact (response time) and energy savings on SPECweb, and compare with analytic models in paper:
http://www.ece.cmu.edu/~bgold/papers/meisner-asplos09.pdf
Talk with me and Brian Gold if you want to pursue this topic.

22. **Analytical modeling of CPU temperature (suggested by Brian Gold @ CMU):** Temperature modeling is notoriously difficult, with state-of-the-art tools requiring multiple, time-consuming simulation runs. Recent (unpublished) work done here at CMU has developed analytic models of server CPU temperature based on queuing theory. Validate these models by performing experiments on real hardware running SPECweb, and compare with results using conventional simulation tools (HotSpot from UVA). Queuing theory experience desirable, but not necessary. Talk with me and Brian Gold if you want to pursue a similar direction.

23. **Hardware support for tolerating data races in multithreaded programs (suggested by Darko Kirovski @ Microsoft Research):** ToleRace is a runtime system that allows programs to detect and even tolerate asymmetric data races. Asymmetric races are race conditions where one thread correctly acquires and releases a lock for a shared variable while another thread improperly accesses the same variable. In this project we want to explore the type of hardware support that would enable the benefits provided by the ToleRace oracle described in Ratanaworabhan et al. The following paper is required reading if you would like to work on this project:

    Ratanaworabhan et al., "*Detecting and Tolerating Asymmetric Races*," PPOPP 2009.
    http://www.csl.cornell.edu/~paruj/papers/ppopp09.pdf

24. **Feature selection for data predictors/prefetchers (suggested by Nikos Hardavellas @ CMU):** Data prefetchers hide the data access latency by predicting future accesses and issuing requests ahead of time. Typically, the prediction is based on some recorded history stored at a table. When the current set of events match a historic event, the next entry in the history table is used as a prediction. For example, the history table entries could be data miss addresses (as in Markov Prefetchers, see Joseph et al. "Prefetching using Markov predictors" ISCA 1997, http://portal.acm.org/citation.cfm?id=264207). If the current miss matches entry "k" in the history table, the predictor may forecast that the next miss will be at an address that is the same as the address in entry "k+1" in the history table. In general, the entries in the history table are more complex than in our simplistic example, and are a combination of execution attributes. However, which execution attributes constitute that history is typically decided in an ad-hoc manner. A history table entry is usually a combination of some bits of the miss address and some bits of the PC (program counter) that generates the reference, but it could be any other attribute (e.g., recent branch predictor outcomes, the last n frame pointers, the previous k data misses, the previous k data misses with some reordering, the previous misses in the same page, general purpose register values,

any combination of the previously mentioned attributes, etc). The goal of this project is to investigate which set of attributes forms the best predictor to forecast the address of the k_th miss ahead of the current one (for example using principal component analysis or independent component analysis techniques, or time series forecasting). We are interested in the k_th miss so that there is enough time for the predictor to issue the prefetch request and get back the data from the memory system before the processor requests that data.

A similar analysis can be done for any other type of predictor (e.g., a branch predictor). The fundamental question this project tries to answer is: "*Which pieces of on-chip information are the most important to use to correctly predict the next branch (or the next address to be prefetched)?"* Talk to Nikos Hardavellas and me if you are interested in this problem. Some preliminary reading that could be useful:

For data prefetching:

Nesbit and Smith, "*Data Cache Prefetching Using a Global History Buffer*," HPCA 2004.

Somogyi et al., "*Spatial Memory Streaming*," ISCA 2006.
http://www.eecg.toronto.edu/~moshovos/research/sms.pdf

For branch prediction:

Yeh and Patt, "*Alternative Implementations of Two-level Adaptive Branch Prediction*," ISCA 1992, http://portal.acm.org/citation.cfm?id=139709

Gonzalez and Gonzalez, "*Control-Flow Speculation through Value Prediction for Superscalar Processors*" PACT 1999
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00807406

25. **Finite directories for private multi-core caches (suggested by Nikos Hardavellas @ CMU):** Future processors are expected to form a "tiled" architecture, with each tile housing a core and a portion of the on-chip cache. A common way to manage such a cache is to treat the cache "slice" within each tile as a cache private to the core within the tile. Because the cores actively share data, the cache must employ a coherence mechanism to maintain data correctness. Such a mechanism is typically directory based, with each entry in the directory keeping information about the sharers and the state of a particular cache block. In a traditional directory-based coherence mechanism, each tile is responsible for keeping information about a subset of the addresses in the address space. Usually the addresses are assigned to tiles in a round-robin fashion. Every cached block is assigned to a unique directory tile, and that tile has an entry for that block in its directory storage. In the extreme case, the entire on-chip cache is full (has only valid blocks) and every block has an address that maps to the same directory tile. To maintain correctness, a conventional directory-based scheme should provide the resources even for the extreme case. This could result in a directory that occupies several times larger chip area than the cache itself, which is a very inefficient use of the available die area. It is possible that a multi-core processor with private caches can maintain high performance and correctness with a significantly smaller directory that covers the common case, coupled with either a backup mechanism that engages when a request misses in the directory (e.g., broadcast to locate the requested block), or a prevention mechanism that

guarantees that when a request misses in the directory, the block is not on chip (e.g., when evicting a directory entry, evict the respective block from the cache). The goal of this project is to identify the ideal directory size across a range of workloads, and the ideal backup or prevention mechanism, for a range of workloads. Talk to Nikos Hardavellas and me if you are interested in this problem. Some papers to get you started:

Chaiken et al., "*Directory-Based Cache Coherence in Large-Scale Multiprocessors*," IEEE Computer, 1990
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=55500&isnumber=2005

Chapter 8 of "*Parallel Computer Architecture*" by Culler & Singh

Zhang and Asanovic, "*Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors*," ISCA 2005
http://pages.cs.wisc.edu/~isca2005/papers/06A-01.PDF

26. Need more topics?

One way of finding research topics is to read recent interesting papers published in top conferences and critically evaluating them. If you find a shortcoming in a paper that is written on an important problem and have an idea to solve the problem that is left unsolved, feel free to talk to me and the TAs. Some topics you might want to think about or search for are as follows **(if you need more detail on any of these topics, talk with me and/or the TAs)**:

- Applications of machine learning to the design of architectural controllers, e.g.
    - Cache replacement and insertion policies
    - More efficient and intelligent memory controllers. See the following paper:
        - Self-Optimizing Memory Controllers: A Reinforcement Learning Approach (http://www.ece.cmu.edu/~omutlu/pub/rlmc_isca08.pdf)
    - Thread switching policies in fine-grained multithreading systems. See the following paper:
        - Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow (http://www.ece.ubc.ca/~aamodt/papers/wwlfung.micro2007.pdf)
- Supporting Quality of Service and fairness in machine learning based memory controllers See the following paper:
    - Self-Optimizing Memory Controllers: A Reinforcement Learning Approach (http://www.ece.cmu.edu/~omutlu/pub/rlmc_isca08.pdf)
- Secure architectures and hardware support for security. For example,
    - Memory encryption
    - Extension of virtual memory mechanisms
- Hardware support for thread scheduling
    - In GPU architectures, see the following paper:
      Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow (http://www.ece.ubc.ca/~aamodt/papers/wwlfung.micro2007.pdf)
- Hard error tolerant architectures
- Architectural support for operating systems

- Shared resource management in multicore and multithreaded systems. Some resources of interest are:
    - Caches shared among different cores/threads
    - Interconnect shared among different cores/threads
    - Processor pipeline shared among different threads in a fine-grained multithreaded or simultaneously multithreaded system
- Cache replacement and insertion techniques
- Memory controllers for 1000-core systems
- Architectural support for garbage collection

    How does garbage collection affect memory system performance of the application? What techniques can we design to mitigate the negative effects of garbage collection?
- Memory bandwidth aware cache replacement policies
- Providing QoS guarantees in shared on-chip caches
- Thermal management of the DRAM system
- Topics in transactional memory – a variety of open-ended research problems exist in this area