Advanced Computer Architecture

Project Statement

Proposal due date Aban 17, 1395
Status Report: Azar 22, 1395
Final Report due date: Day 15, 1395

1 Introduction

In this project, you will innovate new designs or evaluate extensions to designs presented in class or a recent architecture conference. The purpose of this project is to 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 discussing in the class.

The project will account for **25% extra credit** to your final course grade. The project will be graded out of 100 points:

- proposal (10 points)
- status report (10 points)
- final report (80 points)
  - problem definition and motivation (15 points)
  - survey of previous related work (15 points)
  - description of design (15 points)
  - experimentation methodology (15 points)
  - analysis of results (20 points)

2 Proposal

You are encouraged to meet with me and/or the TAs (or send us an email) prior to submitting the proposal if you have any questions. When in doubt, meet with us. Please make an appointment to meet with us.

The proposal is a written two-page document including:

1. A problem definition and motivation.
2. A brief survey of related work with at least three papers you have found and read directly related to the topic (**ask me for pointers**). Conferences that primarily focus on architecture research are ISCA, ASPLOS, MICRO, HPCA, SIGMETRICS, ISLPED, and DSN. You can search the online pages and/or contact the TA for pointers. You will also find the World Wide Computer Architecture web page `http://arch-www.cs.wisc.edu/home` a good source of information.

3. A detailed description of your experimental setup including the tools and simulators that you will use, and modifications to the existing simulation environment.

4. Milestones for the status and final report. What do you expect to see in each milestone? Where do you plan to go based on your observations? You can draw a flow chart to clarify.

3 Status Report

You will hand in a two-page write-up describing your preliminary results. These results form the basis for the final outcome of your research — i.e., the results will not substantially change. You will spend the rest of the semester polishing the results with more extensive analysis and experimentation. Based on these results, explain your plans for the final milestone. If there are any changes to plans, you should bring them up in this report.

4 Final Report

Reports should be in the form of a conference submission including an abstract, an introduction, followed by a detailed description of the design, a methodology section, a results section, and a conclusions section. You may choose to write a related work section preceding the conclusions, or a background section following the introduction. Please make sure your document is spell-checked and is grammatically sound. You will need all the relevant citations to your work in a reference section at the end.

5 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 projects from similar courses at Sharif have become full-blown research projects including the Domino paper that will be submitted to ISCA 2017, the MemCache paper that will be submitted to IEEE CAL, and the EESOP paper that will be submitted to IEEE TC.

6 Research Project Topics

Some of the project topics below are undisclosed ideas and are confidential. Please do not distribute this list.

I will be open to any ideas you may have for a research project if you can convince me that it is worth pursuing. Otherwise, here is a list of possible projects.

1. *Linearizing Program Code.* L1 instruction fetch misses remain a critical performance bottleneck, accounting for up to 40% slowdown in server applications. While simple prefetchers (e.g. Next-Line) fall short of covering many misses, more advanced prefetchers can fully eliminate the instruction misses. Unfortunately, these sophisticated prefetchers have significant hardware overheads. So the processor vendors do not implement these instruction prefetchers

and prefer to implement simple Next-Line (or its variants, like next-2-lines) prefetchers. One way to increase the performance of these applications on general-purpose processors is to code them in a way, that Next-Line prefetcher can prefetch most of the instruction misses. For this reason, the programmer should linearize the program code. This way, code blocks which were temporarily correlated will be spatially ordered. However, this method trades off the ease of programming for performance, and as such, is not desirable. While the programmer may not code the application linearly, a compiler can change the code. Compiler linearizes the program code by simulating the behavior of the cache hierarchy and prefetcher of the target machine. Note that this stimulating approach does not require a detailed timing simulation of the processor. It requires only enough simulation of the cache hierarchy and the prefetcher for discovering temporarily-correlated code blocks. Study the feasibility of a compile-time method for accelerating server programs on general-purpose processors which have only simple instruction prefetcher(s). You are allowed to use extra hardware resources to maximize the benefit.

2. *Leveraging Belady's Algorithm for Better Replacement.* Prior work showed that learning from Belady's algorithm can improve the replacement policy. One solution can be executing Belady's replacement on previous cache accesses and learn the differences of Belady and LRU and apply the result to future replacements. For each set, dump the difference between LRU and Belady victim positions and measure the repetitiveness with Sequitur hierarchical compression. If the opportunity is high, propose an efficient implementation for emulating Belady to reaching a better replacement policy. Compare your design with Hawkeye (ISCA 2016) and Shepherd Cache (MICRO 2007).

3. *TAG Reduction/Elimination for DRAM Caches.* Emerging 3D die stacking technology interconnects a processor die with a stack of DRAM dies using high-density, low-latency through-silicon vias (TSVs). This technology virtually eliminates the memory bandwidth wall by providing orders of magnitude higher bandwidth, and exposes lower latency for stacked on-chip DRAM. Technological constraints, however, limit the stacked DRAM capacity to levels that are far lower than what modern server workloads demand (i.e., at most a few GBs). As such, most proposals for die stacking advocate using the stacked DRAM as a cache. One problem with a large cache is the size of the TAG. One solution to reduce the size of the TAG is to benefit from TLB and store the location of a block in the DRAM cache in the TLB. Study the effectiveness of TLB in reducing the size of the TAG in DRAM caches. Can we eliminate the TAG all together?

4. *Android App on Gem5.* Create a platform (including a multi-core processor) that resembles existing smart-phones using Gem5, pick a popular Android app, tune, run, and characterize the app on the platform. An integral aspect of this project is characterizing the processor's microarchitectural behavior, a-la Clearing the Clouds (ASPLOS 2012), and not just reporting raw performance numbers.

5. *Replacement Policy for Compressed Caches.* One way to increase the effective capacity of a processor's cache, without physically increasing the cache size, is to use data compression. In compressed caches, size of cache blocks varies depending on the compression ratio (i.e.,

some cache blocks are smaller and some of them are larger). In such cases, the Belady's optimal replacement policy does not necessarily select the optimal candidate for replacement. Propose an optimal replacement policy for compressed caches. Find a practical replacement policy that closely mimics the proposed optimal replacement policy. Compare and contrast your practical replacement policy with *ECM* (HPCA 2013).

6. *Architectural Support for Generalized Memory Tagging.* As we can't always write code carefully, we need to guard against malicious attacks on our carelessness. So, it turns out that if you read data from a network packet onto the stack way past the point you should, the packet data can overwrite the return address pointer on the stack to point to code inside the packet! Memory tagging is a technique that maintains bookkeeping information for the application memory space at run-time, and can avoid such problems. But accessing and updating the bookkeeping information incurs high performance/area/energy overheads. Fortunately, most of the time, a program is safe and updating the bookkeeping information is unnecessary. So one can greatly reduce the overhead of memory tagging by filtering most of the bookkeeping updates (See *BugSifter* at `infoscience.epfl.ch/record/187154/files/BugSifter_2012_1.pdf`). In this project, you are going to analyze the sources of overhead of memory tagging and evaluate the effectiveness of filtering as a way to tackle them.

   Note: You may need to use a binary instrumentation tool like Pin for this project.

7. *Design for Dark Silicon.* In the future, we will have dark silicon because we will continue to have denser chips but will not be able to reduce voltages. Design for dark silicon requires specializing different parts of a chip. Take a CloudSuite or smartphone benchmark (or any open-source datacenter or smartphone application), profile it to find out where you spend time, and then design an accelerator which is specialized to run the selected parts of the software (e.g., in a database management system, index traversal is a prime candidate for specialization: see *Meet the Walkers* in MICRO 2013). Doing so should improve energy efficiency by several orders of magnitude. Future server chips will likely include such specialized accelerators for reduced energy.

   Note: You need to profile the code using a software performance analysis tool like OProfile or VTune.

8. *Perforated Computing.* Computing has hit the energy wall. Approximation appears to be a promising approach to reducing energy. Many (e.g., Rinard at MIT, Burger at Microsoft Research, and Ceze at University of Washington) argue that computer systems are too exact and much of what is executed today on the system is not needed. Indeed much computation is done without attention to the fidelity/accuracy desired in the end result. If computation is done just right, one could save a lot of energy and do away with all the waste. Take one CloudSuite or smartphone benchmark (e.g., search), identify the opportunities for perforation to either speed up the workload or reduce energy by doing less. Can one improve efficiency by 10x? 100x?

   Note: You need to read the source code to identify approximable parts, and may want to profile the code using a software performance analysis tool like OProfile or VTune to identify highly-used segments of the code.

9. *Temporal Instruction Prefetchers.* L1 instruction fetch misses remain a critical performance bottleneck, accounting for up to 40% slowdowns in server applications. Whereas instruction footprints typically fit within last-level caches, they overwhelm L1 caches, whose capacity is limited by latency constraints. Past work has shown that server application instruction miss sequences are highly repetitive. By recording, indexing, and prefetching according to these sequences, many of L1 instruction misses can be eliminated. Evaluate *PIF* (MICRO 2011) and *RDIP* (MICRO 2013) with CloudSuite and OLTP workloads. Suggest changes to these algorithms to make them more suitable for CloudSuite and OLTP workloads.

Important Final Note: You are required to send me and the TA an email and let us know on which topic you want to write a proposal.