Project 4:
Research Project

Version 1.0

ECE 721: Advanced Microarchitecture
Spring 2016, Prof. Rotenberg

1. Important Dates

Teams formed and proposal due	Monday, March 7 (5:00 PM)
Instructor gives feedback by this time	Monday, March 14 (5:00 PM)
Project presentations (in class)	April 12, 14, 19, 21
Paper and deliverables due	Monday, May 2 (5:00 PM)

2. Logistics

For this course you are required to propose and carry out a research project.

2.1. Experimental Framework

The research project will be implemented in 721sim. Use of other simulation frameworks will require prior approval from the instructor and must be justified. Phase 2 of the project (Phases 1 and 2 are explained in Section 3) may also explore the use of FabScalar RTL, compilers / dynamic optimizers, *etc*.

2.2. Teams

Given the scope of both phases of the project, it is expected that students will work in groups of two. Working alone will require prior approval from the instructor. A group of three students will only be permitted if there is a multidisciplinary aspect (*e.g.*, compiler + microarchitecture, microarchitecture + RTL) or other unique requirement.

3. Project Description: Phases 1 and 2

The project is comprised of two phases. In Phase 1, teams will implement a Trace Processor microarchitecture in 721sim. In Phase 2, teams will implement something unique with that microarchitecture. A list of example ideas is provided below.

3.1. Phase 1: Trace Processor

Modify 721sim to implement a Trace Processor microarchitecture[1]. The microarchitecture must be modeled *structurally*, in the same way that pipeline registers, ILP-extracting resources (PRF, Active List, IQ, LQ/SQ, etc.), and so forth, are explicitly modeled in 721sim. One aspect, that is

[1] The trace processor PHD thesis has a lot more depth than the MICRO-30 paper and also had more time to "stew": http://people.engr.ncsu.edu/ericro/publications/thesis_ericrotenberg.pdf

modeled implicitly in 721sim, is the bypass network. For the trace processor's hierarchical bypasses – particularly the global bypasses – an explicit/structural model may be required.

Tactically, implementation and testing of the trace processor frontend (trace pre-processing and T$, trace predictor, global rename stage, etc.) and backend (PEs, centralized LQ/SQ) can proceed somewhat independently: a trace processor frontend can be tested with a conventional wide superscalar backend (except for hierarchical renaming); a trace processor backend can be tested with an idealized wide fetch unit (there is a simulator flag for perfect fetch) plus idealized pre-renaming.

As with the current 721sim, the Trace Processor must be fully parameterized:

- number of PEs
- issue width per PE and FU types per lane
- LRF size
- GRF size
- global bypass topology: number of global bypasses and number of global bypasses accessible by a given PE
- global bypass latency (# cycles between producer of live-out value and consumer of live-in value)
- trace selection constraints: maximum trace length, maximum number of live-ins (global RMT read port constraint), maximum number of live-outs (global RMT write port constraint), maximum number of local registers (LRF size constraint)
- trace cache parameters
- trace predictor parameters
- free PEs OOO or in-order
- others that we haven't thought of, as they come up (when in doubt, ask)

3.2. Phase 2: Unique Enhancement or Execution Model within the Trace Processor

In Phase 2, your team will implement an enhancement or new execution model within the Trace Processor. Here are some example ideas:

1. Dynamic Hammock Predication (DHP) [1]: Pre-process traces to implement DHP within traces. Because the join will naturally be within the trace, the insertion of CMOV's within the trace will ensure that live-outs are insulated from selective squashing within the PE. Thus, I expect the trace-level active list, GRF, etc. – the overall top-level structure – to be unimpacted by DHP. Its effects are localized to within a PE. One could add another optimization on top of DHP, which is to predict the predicate of CMOVs to speculatively forward values to CIDD instructions, and implement selective re-execution in the overall Trace Processor (as in MICRO-30 paper [2]). Predicate prediction is a form of exploiting control independence.
2. Control independence in trace processors: Please reference the MICRO-32 paper on this topic [3].
3. Value prediction in trace processors: Background material: MICRO-30 paper [2], class notes on value prediction and selective re-execution strategies, literature on different value predictors, and consultation with instructor. Aside from value predictors and selective re-

execution strategies, an interesting facet that deserves attention is using live-in value predictability and criticality to influence the trace selection policy.

4. Store and load handling: Implement a distributed data cache and LQ/SQ system. Initial proposed implementation (© Eric Rotenberg, Feb. 2016):
    a. Small L0 D$ (non-speculative), SQ, and LQ per PE (or per N PEs), with coherence among L0 D$'s implemented via a centralized L1 D$ directory. Stores drain from a PE's SQ to its L0 D$ only when it becomes the head (non-speculative) PE. Drained stores write-through to L1 D$ to trigger coherence actions. [*Non-speculative private L0 D$'s makes this different than related work on banked caches and speculative versioning caches [5] in TLS processors. Needs rigorous related work search.*]
    b. Explicit synchronization of *predicted-dependent* stores and loads across PEs via store sets, coupled with store-to-load (and store-to-store) forwarding of store addresses and values among PEs. The store tag (for load wake-up) is followed a fixed number of cycles later by the store address and value. Thus, leverage the existing register tag/value forwarding framework – global bypasses – for store-load forwarding. Sending the store address, in addition to the value, enables filtering out false dependencies.
    c. Missed true dependencies (load mispredictions) must be detected. So far, (a) and (b) above make no mention of a store searching all younger loads or loads searching all older stores across PEs. Instead, the store-set predictor is exploited to create point-to-point dataflow, but it is speculative. A true dependency may be missed. Potential solution: drained stores (committed stores in head PE) trigger coherence across L0 D$'s; each speculative PE's LQ snoops this coherence traffic on the L0-L1 cache bus; a conflicting load that was not synchronized with the store is marked as mispredicted (or rather its PE/trace is marked as mispredicted in the trace-level active list); can refine this to filter out false mispredictions. BTW, I guess this really constitutes stores searching all younger loads, but it feels more like conventional consistency model verification in multi-core processors; moreover it is off the critical path with good store-set prediction. Another possibility is using a bloom filter to limit the number of LQ searches, bulk memory disambiguation [6], etc.
5. Multiscalar [4] features on top of Trace Processor:
    a. Decouple trace length from IQ size within the PE, to allow for large traces. This will open up more opportunity for things like DHP and control independence exploitation.
    b. Pair (a) with localized (within PE) fetching of instructions. I.e., local (per-PE) trace caches. The global (top-level) trace cache has only live-in/live-out information to facilitate global renaming.
    c. Instead of a GRF, implement a ring of ARFs like in multiscalar processors.
6. Loop Acceleration: The trace(s) comprising an inner-loop body may fit within some or all of the PEs. Pin these traces to the PEs and disable the frontend. Each PE has multiple LRFs, for multiple instances of the same trace. Multiple instances of a trace correspond to that same trace from multiple loop iterations. Couple with DHP to increase coverage of loops with embedded control-flow. A key challenge is creating multiple instances of the trace's live-outs: either serialize iterations, or provide multiple GRF instances or partial-GRF instances; exploit opportunities to reuse physical registers such as idempotency.

[1] A. Klauser, T. Austin, D. Grunwald, and B. Calder. Dynamic hammock predication for non-predicated instruction set architectures. *1998 International Conference on Parallel Architectures and Compilation Techniques*, pp. 278-285, Oct. 1998.

[2] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. E. Smith. Trace processors. *30th International Symposium on Microarchitecture (MICRO-30)*, pp. 138-148, Dec. 1997.
[3] E. Rotenberg and J. E. Smith. Control Independence in Trace Processors. *32nd International Symposium on Microarchitecture (MICRO-32)*, pp. 4-15, November 1999.
[4] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar. Multiscalar processors. *22nd International Symposium on Computer Architecture (ISCA-22)*, pp. 414-425, June 1995.
[5] S. Gopal, T. N. Vijaykumar, J. E. Smith, and G. S. Sohi. Speculative Versioning Cache. *4th International Symposium on High-Performance Computer Architecture (HPCA-4)*, p. 195, Jan. 1998.
[6] L. Ceze, J. Tuck, J. Torrellas, and C. Cascaval. Bulk Disambiguation of Speculative Threads in Multiprocessors. *33rd International Symposium on Computer Architecture (ISCA-33)*, pp. 227-238, May 2006.

3.3. Special Projects

If you have a special project in mind, please contact the instructor as soon as possible to get permission (or not). Special projects may include RTL based projects (*e.g.*, an alternate Phase 2 might be to also implement the baseline trace processor in FabScalar RTL), combined microarchitecture-compiler projects, microarchitectures conceived by the student(s), etc.

4. Proposal

Submit a proposal (3-6 pages) by the due date indicated above. The proposal must include the following.

- **Project title and author(s)**
- **Project summary**
  Describe the project at a high level. Summarize the Phase 2 features. Summarize objectives.
- **Key outcomes of project**
  Examples:
  - A working structural simulator of a baseline trace processor.
  - A detailed quantitative comparison (IPCs) of trace processors and equally-provisioned superscalar processors. Insight into the IPC impact of distributing a wide-issue processor among PEs.
  - A novel working implementation of DHP in a trace processor.
  - Insight into the achievable speedups of a realistic implementation of DHP in a trace processor.
- **Detailed implementation plan**
  If using a C++ simulator (*e.g.*, 721sim), RTL design, and/or other infrastructure, be specific about all the files, functions, and classes / data structures that will need to be modified or added, the nature of the modifications, *etc*. A **detailed implementation plan** will serve three purposes:
  (1) You will get a head-start in the research project and have a big-picture view of your proposed microarchitecture before coding.
  (2) Both the project team and instructor will be able to gauge the complexity of the project, and increase or decrease the complexity to make it sufficient and reasonable. With a detailed implementation plan, implementation tasks can also be assigned to team members.

(3) By proposing a detailed implementation plan, I can give detailed feedback. For example, I can point out critical omissions and unforeseen complexity, suggest major simplifications, and suggest additional facets to make the project sufficient in scope.

Examples:

- Create a PE class that has an issue queue, execution lanes, LRF, GRF replica, forwarding queues for global result buses, …
- Create a trace cache class that has a trace cache, trace miss unit (BP, I$), trace pre-processing unit (pre-renaming, …), …
- Modify the fetch stage (processor class, processor::fetch(), fetch.cc) to use the trace cache class and trace predictor class.. We anticipate the following specific changes to function processor::fetch(): ...

- **Detailed experimental plan**

  Enumerate the different classes of experiments you will perform to evaluate your microarchitecture. For each class of experiment, describe what insight you are trying to gain; describe and justify the baseline for comparison; propose a hypothesis that will be confirmed or refuted by the class of experiment; *etc*. Where warranted, propose "sensitivity studies": these are studies performed after the "main" experiments, to understand the sensitivity of primary results to parameter variations. In all classes of experiments, indicate what metric or metrics will be plotted (IPC, speedup, cycle time, energy, power, mispredictions per thousand instructions, *etc*.).

  Examples:

  - Perform an extensive design space exploration of the baseline trace processor. Measure IPCs for different trace processor configurations, varying the following parameters: …
  - Compare IPCs of a trace processor and an equally-provisioned superscalar processor to understand the IPC impact of a distributed microarchitecture. Vary the following parameters of the trace processor and provision the superscalar equivalently: …
  - Compare IPC with and without DHP. Vary key parameters that affect DHP coverage, such as trace length. Vary other parameters of the trace processor for sensitivity studies.

- **Contingency plan**

  Some outcomes of the project may involve risk. Later, if you determine these outcomes are unachievable *for reasons beyond your control*, have a backup plan for redirecting the research to alternative outcomes.

  Example:

  - "We may find that the multithreading libraries installed on the Pentium-4 machine have certain limitations that prevent us from doing X. In particular, we are unsure of aspect Y which we assume works as follows.... In preparing this proposal, we did a feasibility study including examination of available software and found no conflicts with the project outcomes. However, if problems do arise that are beyond our control, we will restrict our experiments on the Pentium-4 to basic workload characterization and implement X in the simulator instead."

  Note that thorough feasibility studies (done while preparing your proposal) will reduce the likelihood of contingencies.

- **Assignment of tasks** (for group projects only)

  If working in a team, tentatively assign tasks in the detailed implementation plan to team members. The project complexity and scope must justify the number of students collaborating on the project. Moreover it is important that all students contribute to implementation. It is alright for some tasks to be implemented jointly, for example, some

students like to apply pair-programming as a way to review code as it is being written; if pair-programming, team members must alternate typing.
Likewise, tentatively assign tasks in the detailed experimental plan.

- **References**
  In your project summary, and possibly in your detailed implementation plan, it is appropriate and recommended that you cite any relevant papers in the literature. Include citations in the body of your proposal and a bibliography section (references) at the end.

5. Presentation

5.1. Logistics

The last four lectures of the semester are devoted to project presentations. The classroom is equipped with a PC and projector.

- Your presentation must be in Powerpoint format.
- Use Wolfware to hand in the Powerpoint file. It is due by 9 AM the day of your presentation. 1 point (out of 5) will be deducted for every 1 hour you are late. The instructor will ensure the file is available from the classroom PC, and will open and test it before class begins.
- Presentations will be 15 minutes in duration. The allotted time includes time for questions (~ 2-5 minutes). Obviously, due to schedule constraints, your project does not need to be complete to give the presentation. All group members must take a turn at presenting. The time limit will be strictly enforced. It is wise to practice your talk, time it, and trim it down to the allotted time. (1 slide per minute is a good rule-of-thumb, *e.g.*, 10 minute presentation should have 10 or fewer slides. This is just a guideline for staying within your time budget.)

5.2. Guidelines for presentation content

Content Guideline #1: (adapt according to your specific project)

- **First**, describe the problem or challenge being addressed.
- **Second**, state your approach to the problem. (*Not* implementation, just the approach that you are going to explore.)
- **Third**, describe anticipated project outcomes. There are two essential outcomes for any project: (1) a working implementation of X, and (2) an understanding of the performance benefit of X. Other supporting outcomes can be framed accordingly.
- **Fourth**, describe your implementation in a conceptual, thoughtful, and illustrative way, like an ECE 721 lecture on PRF management, LQ/SQ operation, CPR, trace cache, *etc*. The instructor did *not* explain a mechanism or microarchitecture with a C++/RTL code dump or with raw, uninterpreted descriptions. Instead, the instructor explained mechanisms and microarchitectures conceptually, focused on key aspects so that students understand how something works, interpreted papers, and used diagrams.
- If you have multiple implementation alternatives in mind, please provide coverage of each one.

Content Guideline #2:

Please try to have at least one experimental result by the time of your presentation. This can be anything of an empirical nature.

- In this semester's project, it is expected that at least Phase 1 be completed by the time of the presentation, so there should be adequate experimental results for Phase 1.
- Also try to have at least one experimental result for Phase 2. For example, if you are implementing DHP, even if the implementation is not complete, it is possible to gather data on the lengths of predicted and alternate paths of if-then-else branches, the total misprediction contribution (%) by these branches versus all branches, the distribution of the number of CMOVs required per predicated branch, *etc*.

6. Paper

Your paper is my only "window" into your submitted project. Your final implementation and experiments may be excellent, but if the paper is poorly written, superficial, and not comprehensive, I will not be able to gauge or fully appreciate how good your project is. So write a clear, insightful, and comprehensive paper, that reflects the full extent of your project's contributions.

Another important tip is that, an implementation should be explained conceptually and thoughtfully, as you have experienced in the conference papers that we read throughout the semester. The explanation should NOT be a raw description of code that was modified or added (the submitted code, itself, is the design documentation). Raw, uninterpreted descriptions do not really educate the reader on how a mechanism or microarchitecture works.

The paper should contain the following sections.

- Title and authors
- Abstract
- Introduction (motivation, main idea, contributions, outcomes, *etc*.)
- Related work
- One or more technical sections (ideas, designs, microarchitecture, system, methods, *etc*.)
- Experimental method (simulator or other platform, CAD tools (if applicable) and synthesis/P&R/*etc*. methodology, configurations, benchmarks, metrics, *etc*.)
- Results section (experimental results, graphs, analysis, observations, *etc*.)
- Conclusions (summarize ideas, designs, contributions, outcomes, *etc*.)
- Future work
- References
- Appendix (only for group projects): The research and paper will be graded as a whole and members will receive that grade multiplied by a "contribution factor" from 0.5 (no contribution) to 1 (roughly equal contribution). Accordingly, include an appendix (1 page maximum) indicating contributions made by each student toward the research and preparation of the paper. Also include a proposed contribution factor for each team member, which may be adjusted by the instructor based on the evidence.

7. Grading

Proposal (5 points)

- Project summary (1/2 point)
- Key outcomes of project (1/2 point)
- Detailed implementation plan (2 points)
- Detailed experimental plan (1 point)
- Contingency plans & other (1 point)

Presentation (5 points AND required to receive a grade for your project)

Research & Paper (90 points)

- Organization & overall content (10 points). See Section 6 for requirements.
- Writing quality (10 points). Emphasis is on clarity. Good grammar is necessary but not sufficient.
- Analysis (10 points). Graphs (applicable in most cases), explanation of results.
- Key outcomes (60 points). Original outcomes or modified outcomes of equal significance.
  30 points - No satisfactory outcomes but significant effort
  40 points - Achieved some outcomes satisfactorily
  50 points - Achieved all outcomes satisfactorily
  60 points - Excelled in all outcomes

Here's the bottom-line: you need to impress me with your project to receive a good grade. A mediocre project will receive a mediocre grade. Carry out the research and write the paper as if you were writing a paper for submission to a top computer architecture conference. You have numerous examples to guide you: see the papers that we read throughout the course.