Computer Architecture (263-2210-00L), Fall 2017
HW 5: Multicore Execution, Runahead Execution, Prefetching

Instructor: Prof. Onur Mutlu
TAs: Hasan Hassan, Arash Tavakkol, Mohammad Sadr, Lois Orosa, Juan Gomez Luna

Assigned: Saturday, Nov 26, 2017
Due: **Saturday, Dec 13, 2017**

- **Handin - Critical Paper Reviews (1).** You need to submit your reviews to `https://safari.ethz.ch/review/architecture/`. Please check your inbox. You should have received an email with the password you can use to login to the paper review system. If you have not received any email, please contact comparch@lists.ethz.ch. In the first page after login, you should click in "Architecture - Fall 2017 Home", and then go to "any submitted paper" to see the list of papers.
- **Handin - Questions (2-6).** Please upload your solution to the Moodle (`https://moodle-app2.let.ethz.ch/`) as a single PDF file. **Please use a typesetting software (e.g., LaTeX) or a word processor (e.g., MS Word, LibreOfficeWriter) to generate your PDF file. Feel free to draw your diagrams either using an appropriate software or by hand, and include the diagrams into your solutions PDF.**

1 Critical Paper Reviews [200 points]

Please read the following handout on how to write critical reviews. We will give out extra credit that is worth 0.5% of your total grade for each good review.

- Lecture slides on guidelines for reviewing papers. Please follow this format. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=onur-comparch-f17-how-to-do-the-paper-reviews.pdf`
- Some sample reviews can be found here: `https://safari.ethz.ch/architecture/fall2017/doku.php?id=readings`

(a) Write a one-page critical review for the first paper of the following list and at least **one** of the other 3 papers:

- O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors," HPCA 2003. `https://people.inf.ethz.ch/omutlu/pub/mutlu_hpca03.pdf`
- M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive Insertion Policies for High Performance Caching" ISCA 2007. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=p381-qureshi.pdf`
- N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," ISCA 1990. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=1-jouppi.pdf`
- D. Josephand and D. Grunwald, "Prefetching using Markov Predictors," ISCA 1997. `https://safari.ethz.ch/architecture/fall2017/lib/exe/fetch.php?media=18-2-joseph-prefetching.pdf`

2 Running Ahead [200 points]

Consider the following program, running on an in-order processor with no pipelining:

```
LD   R1   ←  (R3)      // Load A
ADD  R2   ←  R4, R6
LD   R9   ←  (R5)      // Load B
ADD  R4   ←  R7, R8
LD   R11  ←  (R16)     // Load C
ADD  R7   ←  R8, R10
LD   R12  ←  (R11)     // Load D
ADD  R6   ←  R8, R15
```

Assume that all registers are initialized and available prior to the beginning of the shown code. Each load takes 1 cycle to execute, and each add takes 2 cycles to execute. Loads A through D are all cache misses. In addition to the 1 cycle taken to execute each load, these cache misses take 6, 9, 12, and 3 cycles, respectively for Loads A through D, to complete. For now, assume that no penalty is incurred when entering or exiting runahead mode.

Note: Please show all your work for partial credit.

(a) For how many cycles does this program run *without* runahead execution?

(b) For how many cycles does this program run *with* runahead execution?

(c) How many additional instructions are executed in runahead execution mode?

(d) Next, assume that exiting runahead execution mode incurs a penalty of 3 cycles. In this case, for how many cycles does this program run *with* runahead execution?

(e) At least how many cycles should the runahead exit penalty be, such that enabling runahead execution decreases performance? Please show your work.

(f) Which load instructions cause runahead periods? Circle all that did:

Load A Load B Load C Load D

For each load that caused a runahead period, tell us if the period generated a prefetch request. If it did not, explain why it did not and what type of a period it caused.

(g) For each load that caused a runahead period that did not result in a prefetch, explain how you would best mitigate the inefficiency of runahead execution.

(h) If all useless runahead periods were eliminated, how many additional instructions would be executed in runahead mode?

How does this number compare with your answer from part (c)?

(i) Assume still that the runahead exit penalty is 3 cycles, as in part (d). If all useless runahead execution periods were eliminated (i.e., runahead execution is made *efficient*), for how many cycles does the program run with runahead execution?

How does this number compare with your answer from part (d)?

(j) At least how many cycles should the runahead exit penalty be such that enabling *efficient runahead execution* decreases performance? Please show your work.

3 Prefetching [150 points]

An architect is designing the prefetch engine for his machine. He first runs two applications A and B on the machine, with a stride prefetcher.

Application A:

```
uint8_t a[1000];
sum = 0;
for (i = 0; i < 1000; i += 4)
{
    sum += a[i];
}
```

Application B:

```
uint8_t a[1000];
sum = 0;
for (i = 1; i < 1000; i *= 4)
{
    sum += a[i];
}
```

i and sum are in registers, while the array a is in memory. A cache block is 4 bytes in size.

(a) What is the prefetch accuracy and coverage for applications A and B using a stride prefetcher. This stride prefetcher detects the stride between two consecutive memory accesses and prefetches the cache block at this stride distance from the currently accessed block.

(b) Suggest a prefetcher that would provide better accuracy and coverage for
i) application A?
ii) application B?

(c) Would you suggest using runahead execution for
i) application A. Why or why not?
ii) application B. Why or why not?

4 More Prefetching [200 points]

A processor is observed to have the following access pattern to cache blocks. Note that the addresses are **cache block addresses**, not byte addresses. This pattern is repeated for a large number of iterations.

```
Access Pattern P:   A,   A + 3,   A + 6,   A,   A + 5
```

Each cache block is 8KB. The hardware has a fully associative cache with LRU replacement policy and a total size of 24KB.

None of the prefetchers mentioned in this problem employ confidence bits, but they all start out with empty tables at the beginning of the access stream shown above. Unless otherwise stated, assume that 1) each access is separated long enough in time such that all prefetches issued can complete before the next access happens, and 2) the prefetchers have large enough resources to detect and store access patterns.

(a) You have a stream prefetcher (i.e., a next-N-block prefetcher), but you dont know the prefetch degree (N) of it. However, you have a magical tool that displays the coverage and accuracy of the prefetcher. When you run a large number of repetitions of access pattern P, you get 40% coverage and 10% accuracy. What is the degree of this prefetcher (how many next blocks does it prefetch)?

(b) You didnt like the performance of the stream prefetcher, so you switched to a PC-based stride prefetcher that issues prefetch requests based on the stride detected for each memory instruction. Assume all memory accesses are incurred by the *same* load instruction (i.e., the same PC value) and the initial stride value for the prefetcher is set to 0.

Circle which of the cache block addresses are prefetched by this prefetcher:

```
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
```

Explain:

(c) Stride prefetcher couldnt satisfy you either. You changed to a Markov prefetcher with a correlation table of 12 entries (assume each entry can store a single address to prefetch, and remembers the most recent correlation). When all the entries are filled, the prefetcher replaces the entry that is least-recently accessed.

Circle which of the cache block addresses are prefetched by this prefetcher:

```
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
```

```
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
```

Explain:

(d) Just in terms of coverage, after how many repetitions of access pattern P does the Markov prefetcher from part (c) start to outperform the stream prefetcher from part (a), if it can at all? Show your work.

(e) You think having a correlation table of 12 entries makes the hardware too costly, and want to reduce the number of correlation table entries for the Markov prefetcher. What is the minimum number of entries that gives the same prefetcher performance as 12 entries? Similar to the last part, assume each entry can store a single next address to prefetch, and remembers the most recent correlation. Show your work.

(f) Your friend is running the same program on a different machine that has a Markov prefetcher with 2 entries. The same assumptions from part (e) apply.

Circle which of the cache block addresses are prefetched by the prefetcher:

```
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
A,   A + 3,   A + 6,   A,   A + 5
```

Explain:

(g) As an avid computer architect, you decide to update the processor by increasing the cache size to 32KB with the same cache block size. Assume you will be only running a program with the same access pattern P for a large number of iterations (i.e., one trillion), describe a prefetcher that provides smaller memory bandwidth consumption than the baseline without a prefetcher.

Explain:

5 Markov Prefetchers vs. Runahead Execution [100 points]

(a) Provide two advantages of runahead execution over Markov prefetchers.

(b) Provide two advantages of Markov prefetchers over runahead execution.

(c) Describe one memory access pattern in which runahead execution performs better than Markov prefetchers. Show pseudo-code.

(d) Describe one memory access pattern in which runahead execution performs worse than Markov prefetchers. Show pseudo-code.

6 Parallel Speedup [200 points]

You are a programmer at a large corporation, and you have been asked to parallelize an old program so that it runs faster on modern multicore processors.

(a) You parallelize the program and discover that its speedup over the single-threaded version of the same program is significantly less than the number of processors. You find that many cache invalidations are occuring in each core's data cache. What program behavior could be causing these invalidations (in 20 words or less)?

(b) You modify the program to fix this first performance issue. However, now you find that the program is slowed down by a global state update that must happen in only a single thread after every parallel computation. In particular, your program performs 90% of its work (measured as processor-seconds) in the parallel portion and 10% of its work in this serial portion. The parallel portion is perfectly parallelizable. What is the maximum speedup of the program if the multicore processor had an infinite number of cores?

(c) How many processors would be required to attain a speedup of 4?

(d) In order to execute your program with parallel and serial portions more efficiently, your corporation decides to design a custom heterogeneous processor.

- This processor will have one large core (which executes code more quickly but also takes greater die area on-chip) and multiple small cores (which execute code more slowly but also consume less area), all sharing one processor die.
- When your program is in its parallel portion, all of its threads execute **only** on small cores.
- When your program is in its serial portion, the one active thread executes on the large core.
- Performance (execution speed) of a core is proportional to the square root of its area.
- Assume that there are 16 units of die area available. A small core must take 1 unit of die area. The large core may take any number of units of die area n^2, where n is a positive integer.
- Assume that any area not used by the large core will be filled with small cores.

(i) How large would you make the large core for the fastest possible execution of your program?

(ii) What would the same program's speedup be if all 16 units of die area were used to build a homogeneous system with 16 small cores, the serial portion ran on one of the small cores, and the parallel portion ran on all 16 small cores?

(iii) Does it make sense to use a heterogeneous system for this program which has 10% of its work in serial sections?
Why or why not?

(e) Now you optimize the serial portion of your program and it becomes only 4% of total work (the parallel portion is the remaining 96%).

(i) What is the best choice for the size of the large core in this case?

(ii) What is the programs speedup for this choice of large core size?

(iii) What would the same programs speedup be for this 4%/96% serial/parallel split if all 16 units of die area were used to build a homogeneous system with 16 small cores, the serial portion ran on one of the small cores, and the parallel portion ran on all 16 small cores?

(iv) Does it make sense to use a heterogeneous system for this program which has 4% of its work in serial sections?
Why or why not?