CMU 18-447 Introduction to Computer Architecture, Spring 2013
HW 7 Solutions: Coherence, Prefetching, Parallelism and Topologies

Instructor: Prof. Onur Mutlu
TAs: Justin Meza, Yoongu Kim, Jason Lin

1 Prefetching

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.

Solution:
Application A's prefetch accuracy is 248/249 and coverage is 248/250.
Application A accesses a[0], a[4], a[8], ... a[996]. It has 1000/4 = 250 accesses to memory.
The first two accesses to a[0] and a[4] are misses. After observing these two accesses, the prefetcher learns that the stride is 4 and starts prefetching a[8], a[12], a[16] and so on until a[1000] (on the access to a[996], a[1000] is prefetched; however, it is not used). In all, 249 cache blocks are prefetched, while only 248 are used.
Hence, the prefetch accuracy is 248/249 and coverage is 248/250.

Application B's prefetch accuracy is 0 and coverage is 0.
Application B accesses a[1], a[4], a[16], a[64] and a[256]. It has five accesses to memory.
However, there isn't a constant stride between these accesses, as the array index is multiplied by 4, rather than being incremented/decremented by a constant stride. Hence, the stride prefetcher cannot prefetch any of the accessed cache blocks and the prefetch accuracy and coverage are both 0.

(b) Suggest a prefetcher that would provide better accuracy for
i) application A?
Solution:
A next block prefetcher would always prefetch the next cache block. Hence, the cache block containing a[4] would also be prefetched. Therefore, the prefetch accuracy would be 249/250 and coverage would be 249/250.

ii) application B?

Solution:

Most common prefetchers such as stride, stream, next block would not improve the prefetch accuracy of application B, as it does not have an access pattern for which prefetchers are commonly designed. Some form of pre-execution, such as runahead execution or dual-core execution could help improve its prefetch accuracy.

(c) Would you suggest using runahead execution for

i) application A. Why or why not?

Solution:

No. While runahead execution could still provide good prefetch accuracy even for a regular access pattern as in application A, it might not be possible to prefetch all cache blocks before they are accessed, as runahead execution happens only when the application is stalled on a cache miss. A stride prefetcher or next block prefetcher, on the other hand could possibly prefetch all cache blocks before they are accessed, as prefetching happens in parallel with the application's execution.

ii) application B. Why or why not?

Solution:

Yes. Application B's memory access pattern does not have a regular access stride. Commonly used prefetchers are not designed to prefetch an access pattern like this. However, runahead execution could potentially prefetch some cache blocks, especially as the address of the cache blocks does not depend on the data from a pending cache miss.

2 More Prefetching

You and your colleague are tasked with designing the prefetcher of a machine your company is designing. The machine has a single core, L1 and L2 caches and a DRAM memory system.

We will examine different prefetcher designs and analyze the trade-offs involved.

- For all parts of this question, we want to compute prefetch accuracy, coverage and bandwidth overhead after the prefetcher is trained and is in steady state. Therefore, **exclude the first six requests from all computations**.
- If there is a request already outstanding to a cache block, a new request for the same cache block will not be generated. The new request will be merged with the already outstanding request in the MSHRs.

(a) You first design a stride prefetcher that observes the last three cache block requests. If there is a constant stride between the last three requests, it prefetches the next cache block using that stride.

You run an application that has the following access pattern to memory (these are cache block addresses):

```
A A+1 A+2 A+7 A+8 A+9 A+14 A+15 A+16 A+21 A+22 A+23 A+28 A+29 A+30...
```

Assume this pattern continues for a long time.

(i) Compute the coverage of your stride prefetcher for this application.

Solution:
0%.

After each group of three requests, a prefetch is triggered due to a detected stride, but the prefetched block is always useless; none of the demand requests are covered by this prefetch.

(ii) Compute the accuracy of your stride prefetcher for this application.

Solution:
0%. (see above).

(b) Your colleague designs a new prefetcher that, on a cache block access, prefetches the next N cache blocks.

(i) The coverage and accuracy of this prefetcher are 66.67% and 50% respectively for the above application. What is the value of N?

Solution:
$N = 2$.

After (for example) the access to block 14, the prefetcher prefetches blocks 15 and 16. After 15, it prefetches 16 (merged with the prefetch that was already issued) and 17. After 16, it prefetches 17 and 18. Hence, two out of every three demand accesses are covered (66.7%), and half of prefetches are useful (50%).

(ii) We define the bandwidth overhead of a prefetcher as

$$\frac{\text{Total number of cache block requests with the prefetcher}}{\text{Total number of cache block requests without the prefetcher}} \tag{1}$$

What is the bandwidth overhead of this next-N-block prefetcher for the above application?

Solution:
5/3.

For every group of accesses to three consecutive cache blocks, two extra blocks are prefetched. For example, cache blocks 14, 15 and 16 are fetched. In addition to these, blocks 17 and 18 are prefetched.

(c) Your colleague wants to improve the coverage of her next-N-block prefetcher further for the above application, but is willing to tolerate a bandwidth overhead of at most 2x. Is this possible? Why or why not?

Solution:
No.

To get better coverage, the prefetcher must prefetch into the next group of 3 strided requests from the previous group, because the full group of 3 is already prefetched by the first. For instance, on an access to A+14, A+15 and A+16 are already prefetched. To improve coverage, A+21 (which is the first of the next group of 3 strided requests) should be prefetched. However, this would require prefetching the four cache blocks in between A+16 and A+21 (A+17, A+18, A+19, A+20). This increases the bandwidth overhead beyond 2x.

(d) What is the minimum value of N required to achieve a coverage of 100% for the above application? Remember that you should exclude the first six requests from your computations.

Solution:
$N = 5$ (so that A+16 prefetches A+21, then A+21 prefetches A+22, A+23, etc.)

What is the bandwidth overhead at this value of N?

Solution:
7/3.

(e) You are not happy with the large bandwidth overhead required to achieve a prefetch coverage of 100% with a next-N-block prefetcher. You aim to design a prefetcher that achieves a coverage of 100% with a 1x bandwidth overhead. Propose a prefetcher design that accomplishes this goal. Be concrete and clear.

Solution:
1. A prefetcher that learns the pattern of strides: 1, 1, 5, in this case. This can be accomplished by keeping the last three strides and a confidence counter for each pattern of last three strides.

2. A *two-delta stride prefetcher* could record up to two different strides Δ_1 and Δ_2, and the number of strides Δ_1 that are traversed before a stride of Δ_2 is traversed. For this sequence, the prefetcher would learn that $\Delta_1 = 1$, $\Delta_2 = 5$, and that two strides of Δ_1 occur followed by one stride of Δ_2.

3 Markov Prefetchers vs. Runahead Execution

(a) Provide two advantages of runahead execution over markov prefetchers.
Solution:

(i) Does not require recording of miss addresses

(ii) More accurate, as it executes actual code.

(iii) (Mostly) Uses pre-existing hardware.

(iv) Less bandwidth wasted. Markov will issue N prefetches/miss.

(v) Can prefetch compulsory misses (i.e. does not need to see an address previously as a miss to be able to prefetch it)

(b) Provide two advantages of markov prefetchers over runahead execution.
Solution:

(i) Is not limited by the branch prediction accuracy as it is not pre-execution based.

(ii) Can prefetch dependent misses.

(iii) Does not need to execute instructions to generate prefetches.

(c) Describe one memory access pattern in which runahead execution performs better than markov prefetchers. Show pseudo-code.
Solution:
If the access pattern did not repeat, runahead execution would perform better, as markov prefetchers cannot avoid compulsory misses.

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

(d) Describe one memory access pattern in which runahead execution performs worse than markov prefetchers. Show pseudo-code.
Solution:
Linked list traversal that happens multiple times. Runahead cannot prefetch dependent misses whereas markov is able to trigger the miss for the next node when the previous node is visited (assuming that the linked list was traversed previously and the correlation table still contains the correlations between node addresses).

```
while (node != NULL) {
...
node = node->next;
}
```

4 Pre-Execution

A machine's designer is trying to use thread-based pre-execution to speed up an application A's performance. The machine has two cores. Each core has its own private L1 and L2 cache. The cores share an L3 cache and the memory. The designer tries to improve application A's performance by i) running a pruned speculative thread on a separate core and ii) running a pruned speculative thread on a separate thread context on the same core.

(a) Give one reason why executing the speculative thread on a separate core could provide better performance than executing the speculative thread on a separate thread context on the same core.
Solution:
The main program thread runs without sharing any compute resource with the speculative thread. This reduces the interference between the two threads.

(b) Give one reason why executing the speculative thread on a separate thread context on the same core could provide better performance than executing the speculative thread on a separate core.
Solution:
Running on the same core can ensure that blocks are prefetched directly into the core's L1 cache, increasing the benefit of prefetching.

(c) The designer finds that executing the speculative thread on a separate core provides better performance for application A. Now, the core executing the speculative thread is hit by gamma rays and a bit in its register file flips. How does this affect the correctness of application A's execution?
Solution:
Correctness is not affected as the thread running on the core is speculative and does not affect program output.

(d) The designer finds a way to parallelize application A and splits its work into two threads He expects that running the two threads on the two cores would provide better performance than running a speculative thread on one of the cores and the single-threaded version of the program on the other core, as in (c). However, he is surprised to see that the opposite is true. Why do you think this is the case?
Solution:
Parallelizing an application does not always improve performance, for the same reasons listed in question 2 above. Especially if the application were memory-limited, parallelizing the application's computation would not yield much performance improvement. On the other hand, pre-execution could potentially prefetch data and reduce the amount of time spent stalling for memory.

(e) When the two threads of application A are executing on the two cores, the second core is hit by gamma rays and a bit in its register file flips. How does this affect the correctness of application A's execution?
Solution:
Correctness could potentially be affected in this case, as the thread run on the second core is non-speculative (if the specific register were used and its value affects program output).

5 Cache Coherence

(a) What is the advantage of the MESI protocol over the MSI cache coherence protocol?
Solution:
This state allows a cache/processor to write to a location (known to be the only clean copy) without notifying any other processor/cache.

(b) A designer has built a directory-based cache coherent system using the MESI invalidation protocol. Under certain workloads, however, the system performs very poorly, and after a simulation run and closer examination, the designer finds that there is a constant stream of invalidation requests between four of the nodes for one particular cache block. Why does this happen?
Solution:
The cache block ping pongs between the four nodes.

(c) Where and how is the problem best solved?
Solution:
If due to true sharing, rewrite code to reduce sharing or use synchronization primitives that reduce communication. If not possible, consider using an update-based coherence protocol.
If due to false sharing, change data layout via compiler or code rewrite to eliminate false sharing.

6 Coherence Protocols

Suppose we have a multiprocessor system with 512 processors. Each processor has a 1 Megabyte private writeback cache with 64-byte cache blocks. The main memory size is 1 Gigabyte.

(a) If we implement a snoopy bus based MESI cache coherence protocol, how many bits of state do we need in the entire system for the coherence protocol implementation?

Solution:
2^{24} bits.

There are 2^{23} cache blocks in total (2^{20} bytes per cache, 2^9 caches, hence 2^{29} bytes in private caches; divided by 2^6 bytes per cache block). Each cache block requires two bits of state (M, E, S, or I). Hence, 2^{24} bits.

Where do these bits reside?

Solution:
In the private cache tag stores.

(b) If we instead implement a directory based cache coherence protocol *as we discussed in class*, how many bits of state do we need in the entire system for the coherence protocol implementation?

Solution:
$2^{24} \times 513 + 2^{24}$ bits.

We still need the MESI state bits in private caches. Then, we must compute the directory storage space. There are 2^{24} cache blocks in main memory, and each cache block needs one bit per processor and one exclusive bit (513 bits), hence $2^{24} \times 513$ bits in the directory in total.

Where do these bits reside?

Solution:
Directory bits: in cache directory. MESI bits: in private caches.

(c) Which of the above two protocols would you choose for this system? Why?

Solution:
Directory.

A bus is not scalable to 512 processors, but a directory is.

Although the snoopy bus-based system has much lower storage requirements, a bus cannot provide enough bandwidth to sustain the demand of 512 processors. Hence, a directory-based system (built using a scalable interconnect) is more appropriate.

7 Parallel Speedup

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)?

Solution:
Cache ping-ponging due to (inefficient or poorly-designed) data sharing.

(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?

Solution:
10.
Use Amdahl's Law: for n processors, Speedup(n) $= 1/(0.1 + \frac{0.9}{n})$.
As $n \to \infty$, $Speedup(n) \to 10$.

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

Solution:
6.
Let $Speedup(n) = 4$ (from above) and solve:

$$4 = 1/(0.1 + \frac{0.9}{n})$$
$$0.25 = 0.1 + \frac{0.9}{n}$$
$$0.15 = \frac{0.9}{n}$$
$$n = 6$$

.

(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?

Solution:
4 units.
For a given large core size of n^2, then the large core yields a speedup of n on the serial section, and there are $16 - n^2$ small cores to parallelize the parallel section. Speedup is thus $1/(\frac{0.1}{n} + \frac{0.9}{16-n^2})$. To maximize speedup, minimize the denominator. One can find that for $n = 1$, the denominator is 0.16. For $n = 2$, the denominator is 0.125. For $n = 3$, the denominator is 0.1619 (this can be approximated without a calculator: 0.0333 plus $0.90/7 > 0.12$ is greater than 0.15, thus worse than $n = 2$.) Hence, $n = 2$ is optimal, for a large core of $n^2 = 4$ units.

(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?

Solution:
6.4
Speedup is $1/(0.10 + \frac{0.90}{16}) = 6.4$.

(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?

Solution:
Yes.
The serial portion of the program is large enough that speedup of the serial portion with the large core speedup outweighs loss in parallel throughput due to the large core.

(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?

Solution:
A large core of 4 units.
Similar to above, check speedup for a large core of execution speed n units and $16 - n^2$ small cores. Speedup is $1/(\frac{0.04}{n} + \frac{0.96}{16-n^2})$. Minimize the denominator to maximize speedup. For $n = 1$, the denominator is 0.104. For $n = 2$, the denominator is 0.1, and for $n = 3$ it is 0.1504. Thus, as before, use a large core of 4 units.

(ii) What is the program's speedup for this choice of large core size?

Solution:
10.
Speedup is $1/(\frac{0.04}{2} + \frac{0.96}{12}) = 10$.

(iii) What would the same program's 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?

Solution:
10.
Speedup is $1/(0.04 + \frac{0.96}{16}) = 10$.

(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?

Solution:
No.
Heterogeneous system provides no performance benefit over the homogeneous system but is more complex to design.

8 Topologies

Suppose you would like to connect 625 processors, and you are considering three different topologies: bus, point-to-point network, and mesh.
Describe one disadvantage of each:

(i) A Single Bus.

Solution:
Very high bus contention (with 625 processors).

(ii) A Point-to-Point Network.

Solution:
Many individual wires (between every pair of nodes).

(iii) A 25x25 Mesh.

Solution:
High complexity.

Which one would you choose? Why?
Solution:
A 25x25 Mesh: it is performance-scalable to 625 processors and not cost-prohibitive.