Computer Architecture HW5

Fabian Wüthrich

December 15, 2020

1 Critical Paper Reviews [1000 points]

see here

2 Parallel Speedup [200 points]

a) Data sharing between cores could cause cache ping ponging

b) We can calculate the speedup using Amdahl's law

$$\frac{1}{0.1 + \frac{0.9}{n}} \xrightarrow{N \to \infty} \frac{1}{0.1} = 10\text{x}$$

c) Using Amdahl's law we can solve the following equation for n

$$4 = \frac{1}{0.1 + \frac{0.9}{n}}$$

Thus, we need 6 processors to achieve a speedup of 4.

d) i) A large core of size n^2 yields a speedup of n for the serial portion of the code and $16 - n^2$ small cores are available to execute the parallel portion. Using Amdahl's law we get the following speedup

$$\frac{1}{\frac{0.1}{n} + \frac{0.9}{16 - n^2}}$$

To maximize the speedup, we need to minimize the denominator.

$n = 1$ denominator 0.16

$n = 2$ denominator 0.125

$n = 3$ denominator 0.1619

Thus, $n = 2$ is optimal so the large core requires 4 units.

ii) We can calculate the speedup using Amdahl's law

$$\frac{1}{0.1 + \frac{0.9}{16}} = 6.4\text{x}$$

iii) Yes, with the heterogeneous system we get a speedup of 8x whereas the homogeneous system achieves only 6x.

e) i) Now the speedup is

$$\frac{1}{\frac{0.04}{n} + \frac{0.96}{16 - n^2}}$$

and $n = 2$ still maximizes the speedup so we need 4 units for the large core.

ii) We can calculate the speedup using Amdahl's law

$$\frac{1}{\frac{0.04}{2} + \frac{0.96}{12}} = 10\text{x}$$

iii) We can calculate the speedup using Amdahl's law

$$\frac{1}{0.04 + \frac{0.96}{16}} = 10\text{x}$$

iv) No, the homogeneous system is as fast as the heterogeneous system and the homogeneous system is easier to design.

3 Asymmetric Multicore [400 points]

a) The large core occupies n^3 units so the number of small cores is $32 - n^3$.
Using the formula from the lecture notes we get

$$Speedup = \frac{1}{\frac{0.2}{n} + \frac{0.8}{32-n^3}}$$

We can approximate the total execution time as

$$t_n = t_{ser} + t_{par} = \frac{0.2}{n} + \frac{0.8}{32 - n^3}$$

and try different values for n

$t_1 = 0.2 + 0.03 = 0.23$

$t_2 = 0.1 + 0.03 = 0.13$

$t_3 = 0.07 + 0.16 = 0.23$

Thus, the optimal configuration is $n = 2$ and 24 small cores.

b)

$$\begin{aligned} E_{ser} &= t_{ser} \times (P_{large\_dynamic} + P_{large\_static} + P_{small\_static}) \\ &= 0.1 \times (6n + n + 0.5(32 - n^3)) \\ &= 2.6\, Joules \end{aligned}$$

$$\begin{aligned} E_{par} &= t_{par} \times (P_{large\_static} + P_{small_dynamic} + P_{small_static} \\ &= 0.03 \times (n + (32 - n^3) + 0.5(32 - n^3)) \\ &= 1.14\, Joules \end{aligned}$$

$$E_{total} = E_{ser} + E_{par} = 2.6 + 1.14 = 3.74\, Joules$$

c) i) If the large core participate in the computation of the parallel portion, t_{ser} stays the same and t_{par} decreases from $\frac{0.8}{32-n^3}$ to $\frac{0.8}{32-n^3+n}$ i.e. a speedup of $\frac{13}{12}$.

ii) E_{ser} is still 2.6 Joules. For the parallel portion we get

$$\begin{aligned} E'_{par} &= t'_{par} \times (P_{large\_static} + P_{large\_dynamic} + P_{small_dynamic} + P_{small_static} \\ &= 0.0307 \times (6n + n + (32 - n^3) + 0.5(32 - n^3)) \\ &= 1.54\, Joules \end{aligned}$$

and can calculate

$$E'_{total} = E_{ser} + E'_{par} = 2.6 + 1.54 = 4.14\, Joules$$

The overall increase is $\frac{4.14\, Joules}{3.74\, Joules} = 1.12x$.

iii) The performance increases by 1.08x whereas the energy consumption increases by 1.12x so the large core collaboration is not really an improvement.

d) We reduce the size of the large core i.e. $n = 1$. Let s be the new serial portion and t_2 the execution time calculated in (a).

$$t_2 > t'_{ser} + t'_{par} = \frac{s}{n} + \frac{1-s}{32-n^3}$$
$$0.13 > \frac{s}{1} + \frac{1-s}{31}$$
$$0.1 > s$$

The serial portion should be at most 10%.

e) The total power consumption is

$$P_{total} = \frac{E_{ser} + E_{par}}{t_{total}}$$

E_{par} is still $1.14\,Joules$ whereas $E_{ser} = 0.1 \times (D \times n + n + 0.5(32 - n^3) = 0.2D + 1.4\,Joules$.

Now we have to constrain the total power consumption to

$$P_{total} = \frac{0.2D + 1.4 + 1.14}{0.13} \leq 20\,Watts$$
$$D \leq 0.3$$

The current dynamic power consumption of the large core is $6n$ Watts and we have to decrease it to $0.3n$ Watts. Thus, we have to decrease the power consumption by at least 20x.

4 Runahead Execution [200 points]

a) To get 66 cache hits within 100 load instructions, every cache miss has to be followed by 2 cache hits. Thus, the runahead execution has to prefetch 2 cache blocks within the memory latency X. To get 2 times to the load instruction, the processor has to execute at least $2 \times (29\,ALU + 1\,BRANCH + 1\,LD) = 62$ instructions in runahead mode. If the processor executes more than 92 instructions, 3 cache blocks will be loaded which conflict with each other so the cache hits decrease. Therefore, the cache miss latency is $61 < X < 93$.

b) If the cache miss latency is below 31 cycles i.e. $X < 31$, the runahead execution cannot reach the load instruction which always incur a cache miss. If the cache miss latency is above 123 i.e. $X > 123$ 4 cache blocks are fetched and the first runahead block is evicted from cache (all blocks map to same set and regular block is not evicted).

c) If the runahead execution fetches 3 cache blocks, the cache is filled in an optimal way because with 4 blocks interference happens and with 2 blocks the cache is not fully used. Therefore, the minimum number of cache misses is 25.

5 Prefetching [200 points]

a) A cache block holds 4 elements of `a`.

Application A

Application A has the following access pattern

`a[0]`, `a[4]`, `a[8]`, ..., `a[996]`

and performs $1000/4 = 250$ accesses to memory.

The first two accesses are misses without a prefetch. Then the prefetcher observes a stride of 4 and starts prefetching the correct cache blocks until `a[1000]`. All prefetched blocks are used except `a[1000]` so the accuracy is $\frac{248}{249}$ and the coverage is $\frac{248}{250}$ because the first two misses were not prefetched.

Application B

Application B has the following access pattern

`a[1]`, `a[4]`, `a[16]` `a[64]` `a[256]`

The prefetcher is not able to find a constant stride across these accesses. Hence, the accuracy and coverage are both 0.

b) **Application A**

A next-line prefetcher would always prefetch the next cache line so `a[4]` would also be prefetched. Thus, accuracy and coverage would be both $\frac{249}{250}$.

Application B

Application B has an uncommon access pattern that is not supported by common prefetchers. Therefore, more sophisticated prefetchers such as a Markov prefetcher or runahead execution are required for accurate prefetches.

c) **Application A**

No, Application A has already a good prefetch metrics with a stride or next-line prefetcher so using runahead execution would be an overkill. What's more, the processor can enter runahead mode only when it's stalled whereas a stride/next-block prefetcher can run in parallel to the processor.

Application B

Yes, Application B has an uncommon access pattern which could be detected using runahead execution.

6 Cache Coherence [300 points]

a) The cache block `0x511100` in set 1 cannot be in state E in cache 2 and in state S in cache 3 because the MESI protocol doesn't allow that a cache block is in exclusive and shared state at the same time. The cosmic rays could have either switched the cache block from I to S in cache 3 or from S to E in cache 2.

b) The first path cannot lead to an incorrect execution because the store instruction of processor 3 brings the system back into a consistent state (cache 3 becomes M and cache 2 switches to I).

The second path could lead to an incorrect execution because the first instruction could read invalid data if the cosmic ray change was from I to S in cache 3 (cache 2 believes it has an exclusive copy and doesn't notify the other caches on a write).

c) (1) `st 0x522222C0` Processor 0

(2) `ld 0x5FF00000` Processor 1

(3) `ld 0x5FFFFFC0` Processor 0

(4) `st 0x5FF00000` Processor 3

7 Memory Consistency [300 points]

a) Sequential consistency guarantees that each thread executes the instructions in the order specified by the programmer. Across threads the ordering is ensured by the while-loop and the `flag` variable. T1 initializes `Y[0]` to 1 and T0 waits until the flag is set i.e. `Y[0]` is initialized. After the flag is set, T0 increments `Y[0]` so the final value is 2.

b) There are several possible instruction interleaving but the final value of `b` is either 0 or 1. If T0.1 executes before T1.0 the value is 0 or if T1.0 executes before T0.1 the value is 1.

c) Due to weak consistency the following orderings are possible:

T1.1 T0.3 T1.0 Final Value 1

T1.1 T1.0 T0.3 Final Value 2

T1.0 T1.1 T0.3 Final Value 2

d) A memory fence between T1.0 and T1.1 ensures the `Y[0]` is initialized before the flag is set.

A memory fence between T0.2 and T0.3 ensures that `Y[0]` is not incremented before the flag is set from T1.

8 BONUS: Building Multicore Processors [250 points]

a) The loop body is not parallelizable because `past` and `current` are stored in registers on the same core. Hence, we can ignore the cycle information. The 13th loop iteration is dependent on the 1st iteration so we can execute 12 consecutive iterations in parallel i.e. the serial portion of the program is $\frac{1}{12}$. Using Amdahl's law with an infinite number of cores we get a maximum speedup of 12x.

b) 12 cores i.e. one core per parallel iteration.

c) Assign iteration i to processor $i \,\%\, 12$.

d) No. On a single core machine one memory access is performed every 4 cycles so 12 iterations take 900 cycles. To achieve a 12x speedup, each core needs to complete it's work in 75 cycles which is impossible as a memory access alone requires 100 cycles.

e) The maximum speedup is achieved if every cache hit is used as best as possible. Thus, 3 cores are required to achieve almost a 3x speedup.

f) Core 1 runs iterations i+0, i+1, i+2, i+3

Core 2 runs iterations i+4, i+5, i+6, i+7

Core 3 runs iterations i+8, i+9, i+10, i+11