Computer Architecture Assignment 5

Rachel Hwang

November 26, 2013

Book exercises

- 5.2.1b
  4 integers may be stored on a 16-byte cache line.

- 5.2.2b
  References to B[I][0] exhibit temporal locality. The same B[I][0] element is being accessed every 8 iterations.

- 5.2.3b
  'I' is the variable in the inner-most loop, so references to 'A[J]' iterating over 'I' exhibit spatial locality, since at each iteration we are accessing the next element in the array, which is continguous in memory.

- 5.2.4b
  In this code snippet $I \cdot J = 64000$ elements are accessed in A and $I = 8$ elements are accessed in B for a total of 64008 ints. This is $\frac{64008 \cdot 4\ bytes}{16\ bytes} = 16002$ cache lines.

- 5.2.5b
  As in the previous example, references to B(I,0) exhibit temporal locality. The same B(I,0) element is being accessed every 8 iterations.

- 5.2.6b
  References to A(I, J) and B(I,0) exhibit spatial locality. Like in the previous code snippet, this is because the inner-most loop moves along the dimension that these matrices are contiguous along in memory.

- 5.4.1b
  The cache line size is 16 words.

- 5.4.2b
  The cache has 64 entries.

- 5.4.3b
  Data storage bits $=16\ words \cdot 4\ bytes\ \cdot 8\ bits = 512$. So the ratio is $\frac{512+21}{512} = 1.04$.

- 5.4.4b
  No data is evicted, since we are starting from power-on, but six initially-empty blocks are filled.

- 5.4.5b
  The hit ratio is $\frac{6}{12}$, so 1:2.

- 5.4.6b

index	tag	data
0	0	mem[0:63]
2	0	mem[128:191]
3	0	mem[192:255]
16	0	mem[1024:1087]
34	0	mem[2176:2239]
48	0	mem[3072:3135]

- 5.6.1b
  The miss rate is $\frac{1}{16}$. The miss rate is not particularly sensitive to the working size set, just the cache line size. According to the 3C model, the misses in this case are compulsory misses.

- 5.6.2b
  When the cache line size is 16 bytes, the miss rate is 1 in 8.
  When the cache line size is 64 bytes, the miss rate is 1 in 32.
  When the cache line size is 128 bytes, the miss rate is 1 in 64.

  This working set takes advantage of spatial locality.

- 5.6.3b
  With pre-fetching, there is only one miss, so the miss rate is 1 in 16,000.

- 5.6.4b
  The number of misses = 1.35· instructions ·(% misses · B).
  The number of cycles wasted = $misses \cdot 20$·B).
  So we want to minimize the percentage of total cycles that are wasted cycles, this is $1.35 \cdot 20 \cdot$ (% misses · B). Given the above miss rates, the optimal block size is 8 bytes.

- 5.6.5b
  Similar to the previous problem, we are minimizing $1.35 \cdot (24 + B) \cdot (\%misses)$. Given the above miss rates, the optimal block size is 8 bytes.

- 5.6.6b
  For constant miss latency, the optimal block size is 128 bytes, since this has the lowest miss rate, so the percentage of total cycles that are wasted is the lowest.

Lab exercises

- 1.a
  For the clustalw programs:
  Number of instructions executed: 117761198
  Number of L1 cache references executed: 347066 + 63658 + 39417731 + 232186 + 8428026 + 35100 = 48523767

- 1.b
  Fraction of the total number of executed instructions:
  Loads: 10174075/48524766 ≈ .20
  Stores: 6862665/48524766 ≈ .14
  Branches: 1999947/48523767 ≈ .04

- 1.c
  The CPI for the overall machine is 1/1.38648 ≈ 0.72.

- 1.d
  The contribution of the memory hiearchy to the CPI is the number of wasted cycles,

  $\frac{Missrate*25clocks}{Instructions} = ((63658 + 35100 + 232186) * 25)/117771197 \approx .0703$

- 2.a
  For $64k\_64b\_8way$, the hit rate is $(39417731 + 8428026 + 347066)/48523767 \approx 0.9933$.

  For $128k\_64b\_8way$, there were a total of 48732272 cache references, so the hit rate is $(8445990 + 331948 + 39894869)/48732272 \approx 0.9987$.

For $256k\_64b\_8way$, there were a total of 48678113 cache references. The hit rate was $(316199 + 8439491 + 39889695) = 48678113 \approx 0.9993$.

- 2.b
  Given the numbers above, the variation is very small for each of these cache sizes, less than 0.01. For each doubling in size, the improvement in hit rate is very minimal. Because the improvement in hit rate is so small, increasing the cache size would likely not offer much improvement in overall performance.

- 3.a
  For $64k\_64b\_8way$, the hit rate is $(39417731 + 8428026 + 347066)/48523767 \approx 0.9933$

  For $64k\_128b\_8way$, there were a total of 48305322 cache references, so cache hit rate was $(39623818 + 161400 + 8356096)/48304322 \approx 0.9966$.

  For $64k\_256b\_8way$, there were a total of 49274458 cache references. The hit rate was $(84228 + 7544505 + 41533949)/49274458 \approx 0.9977$.

- 3.b
  Once again, the variation between these hit rates is very small, less than 0.01. However, given how small the improvements are with a larger block size, because we are increasing the miss penalty, the overall impact of a cache miss is actually worse with a larger cache block. It's not worth the small gain.

- 4.a
  For $64k\_64b\_directmapped$, there were a total of 48235615 cache references, so the hit rate was $(38930579 + 329596 + 8247350) = 48235615 \approx 0.985$.

  For $64k\_64b\_2way$, there were a total of 48061622 cache references, so the hit hit rate is $(287098 + 8359183 + 38938575) = 48061622 = 0.990$

  For $64k\_64b\_4way$, there were a total of 51432586 referecenes, so the hit rate was $(329983 + 42456625 + 8392396)/51432586 = 0.9951$.

  For $64k\_64b\_8way$, as calculated above, hit rate is 0.9933.

  For $64k\_64b\_fullyassociative$, there were a total of 50029483 references, so the hit rate is $(362807+8370605+40788145) = 50029483 \approx 0.9898$.

- 4.b Variation is larger here than in previous problems, but still pretty small. The fact that the 4-way associative performs the best is confusing.

- 4.c
  I would use a 4-way associative set if this higher associativity were free, since this seemed to yield the best performance. In the case that each higher level of associativity instead costs an additional 10% clock time, I would instead go with a directly-mapped cache. My reasoning is that given a 25 cycle miss penalty, and given the we observed a cache rate of 0.990, ($\approx$ 330944 misses) for a 2-way associative set, if the CPI is 0.72, then there were .72*117771197 = 84795261 cycles, which yields about a .098 increase in the execution time. Even in the ideal case where we manage to somehow get a cache hit rate of 100%, giving us a speedup of 10%, it's not worth the cost of increased associativity, so we're better off with the direct-mapped cache.

- 5.a
  Data memory reads: 339649914
  Bytes read: 2537594686
  Bytes written: 541640062
  Total bytes interacted: 2577244603

- 5.b
  Bytes read per instruction: 21.54

Bytes written per instruction: 4.60
Bytes interacted with: 21.88.

We observed a total of .72*117771197 = 84795261 cycles. This means there were 39649917/84795261 = .468 bytes read per cycle. 541640064/84795261 = 6.39 bytes written per cycle and 2577244605/84795261 = 30.4 bytes interacted with per cycle. This gives us 30.4 bytes $\cdot 2 \cdot 10^9$ = 60.8 Gbps

- 5.c
  There was a total of 6320512 bytes read from L2.

  The total ratio is 1 - 6320512/2577244605 = 0.9975

- 5.e
  L2 misses: 16664
  Number of bytes: 1066496 bytes
  Given these figures, the L2 cache achieves a 1066496/6320512 = 0.169 reduction.

Dearest Grader,

I am very sorry I was unable to finish problem 6 or 5.d. I am out of time. :(

Cheers,
-Rachel