COE 501: Computer Architecture

Problem Set 3: Memory Hierarchy

Solution

1) (10 pts) The transpose of a matrix interchanges its rows and columns. Here is the code:

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    output[j][i] = input[i][j];
```

Both the input and output matrices are stored in row-major order. Assume that you are executing $N \times N$ double-precision (8 bytes per element) matrix transpose on a processor with 16 KB D-Cache, which is 2-way set-associative, and 64-byte blocks. The D-Cache is a write-back with write-allocate policy on a write miss.

a) (4 pts) Assume each set in the D-Cache stores one block from the input matrix and a second block from the output matrix. How many sets exist in the D-Cache? What is the maximum value of N such that both the input and output matrices can fit in the 16-KB D-Cache?

Number of Sets = 16 KB / 2 ways / 64 bytes = 128 sets [2 points]

The cache has 2 ways. The input matrix is stored in the first way, while the output matrix is stored in the second way. Each way has 8 KB. Each matrix element is 8 bytes. Therefore, a maximum of 1024 elements of the input and output matrices can be placed in the cache. Therefore, $N^2 = 1024$ and $N = 32$. [2 points]

b) (6 pts) A compulsory cache miss occurs when a block is referenced for the first time. Given that N=16, how many cache misses are caused in the 16 KB 2-way set associative cache? If each cache miss stalls the processor for 8 cycles (assuming hit in L2 cache) then what is the total number of stall cycles for matrix transpose when N=16? What is the impact on the CPI if the execution CPI = 1.1 (excluding cache misses)? Assume six instructions are fetched and executed per inner loop iterate plus 2 instructions per outer loop iterate.

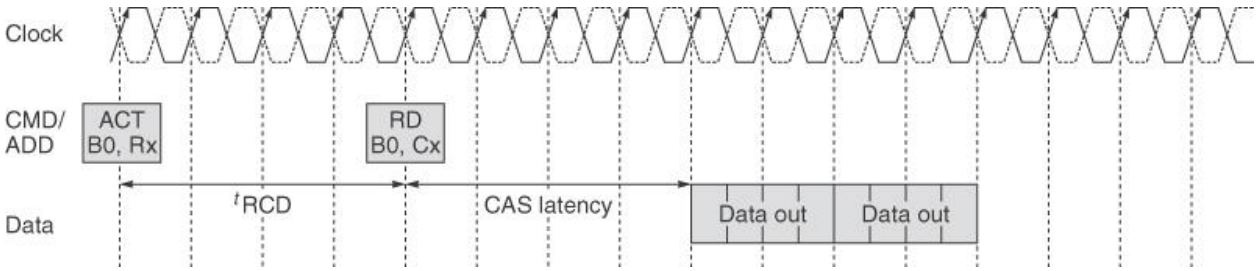
Number of compulsory cache misses = Number of input and output matrix blocks fetched into the cache. The first reference to a block causes a compulsory miss. However, references to the remaining block elements do not generate cache misses due to spatial locality. Each block has 64 bytes = 8 matrix elements.

Compulsory misses = 2 matrices × (16×16 elements) / 8 per block = 64 misses. [2 points]

Number of stall cycles = 64 misses × 8 cycles = 512 cycles. [2 points]

Total instruction count = 6 × 16 × 16 + 2 × 16 = 1568 instructions.
Stall cycles per instruction = 512 / 1568 = 0.326
Effective CPI = 1.1 + 0.326 = 1.426 [2 points]

2) (10 pts) A sample DDR SDRAM timing diagram is shown below. The cache block size is 64 bytes and each transfer consists of 8 bytes. Two transfers = 16 bytes are done per bus cycle.



a) (5 pts) Assume that tRCD = 5 and CL = tCAS = 5 bus cycles. In addition, for each requested block (64 bytes) we automatically prefetch a second adjacent block (another 64 bytes). How much time is required from the presentation of the ACTIVE command until the two blocks are read from memory? Assume that a DDR2-800 DIMM is used (800 Millions of transfers per second at double data rate). Show your answer in bus clock cycles as well as in nanoseconds.

Each block = 64 bytes = 8 Transfers = 4 bus cycles (double data rate)
To transfer two blocks (one requested + one prefetched) = 16 transfers = 8 bus cycles
Time required = tRCD + tCAS + tBurst (for 2 blocks) = 5 + 5 + 8 = 18 bus cycles [3 points]
DDR2-800 ➔ 800 Millions of transfers per second ➔ Bus frequency = 400 MHz
Bus cycle = 1000 / 400 = 2.5 ns
Time required = 18 × 2.5 = 45 ns [2 points]

b) (5 pts) What is the latency when using the DDR2-800 DIMM of a burst read = 64 bytes requiring a bank ACTIVE command versus one to an already open row (no ACTIVE command)? The on-chip latency of a cache miss through levels 1 and 2 and back, not including the DRAM access is 20 ns. Add the time required to process a cache miss through L1 and L2 caches.

Time required with ACTIVE command (one block only) = tRCD + tCAS + tBurst (for 1 block)
= 5 + 5 + 4 bus cycles = 14 bus cycles x 2.5 ns = 35 ns [2 points]
Time required to open row (no ACTIVE command) = tCAS + tBurst (for 1 block)
= 5 + 4 bus cycles = 9 bus cycles x 2.5 ns = 22.5 ns [2 points]
Total time to process a cache miss (with ACTIVE command) = 20 + 35 = 55 ns [0.5 pts]
Total time to process a cache miss (no ACTIVE command) = 20 + 22.5 = 42.5 ns [0.5 pts]

3) (12 pts) A processor with in-order execution runs at 1 GHz and has an execution CPI of 1.2 without counting memory stall cycles. The only instructions that access memory are loads (20% of all instructions) and stores (6% of all instructions). The memory system is composed of an I-cache and a D-cache that each has a hit time of 1 clock cycle.

The I-cache has a 3% miss rate. The D-cache is write-back with a 5% miss rate for reads and a 2% miss rate for writes. It takes 15 ns on average to access and transfer a block from the unified L2 cache into the I-cache or D-cache. Of all memory references sent to the L2 cache in the system, 20% miss in the L2 cache and require main memory access. It takes 50 ns on average to access and transfer a block from the main memory into the I-cache or D-cache.

a) (3 pts) What is the average memory access time for instruction fetching?
b) (3 pts) What is the average memory access time for data reads?
c) (3 pts) What is the average memory access time for data writes?
d) (3 pts) What is the overall CPI, including memory stall cycles?

Solution:

a) Average memory access time for instruction fetching =
I-cache hit time +
I-cache miss rate × average access time to L2-cache +
I-cache miss rate × L2-cache miss rate × average access time to main memory =
1 ns + 0.03 × 15 ns + 0.03 × 0.2 × 50 ns = 1.75 ns

b) Average memory access time for data reads =
D-cache hit time +
D-cache read miss rate × average access time to L2-cache +
D-cache read miss rate × L2-cache miss rate × average access time to main memory =
1 ns + 0.05 × 15 ns + 0.05 × 0.2 × 50 ns = 2.25 ns

c) Average memory access time for data writes =
D-cache hit time +
D-cache write miss rate × average access time to L2-cache +
D-cache write miss rate × L2-cache miss rate × average access time to main memory =
1 ns + 0.02 × 15 ns + 0.02 × 0.2 × 50 ns = 1.5 ns

d) From part (a), Stall time per instruction caused by I-cache misses
= 1.75 ns – 1 ns (hit time) = 0.75 ns = 0.75 cycles per instruction (1 GHz ➔ 1 ns cycle)

From part (b), Stall time per instruction caused by D-cache read misses
= 0.2 (load frequency) × (2.25 ns – 1 ns hit time) = 0.25 ns = 0.25 cycles per instruction

From part (c), Stall time per instruction caused by D-cache write buffer
= 0.06 (store frequency) × (1.5 ns – 1 ns) = 0.03 ns = 0.03 cycles per instruction

Overall CPI (including stall cycles caused by cache misses and write buffer stalls)
= 1.2 (execution CPI) + 0.75 (caused by I-cache misses) + 0.25 (caused by D-cache misses) + 0.03 (caused by Write buffer stalls) = 2.23

4) (9 pts) Cache organization is often influenced by the desire to reduce the cache's energy consumption. For that purpose we assume that the cache is physically distributed into a data array (holding the data), tag array (holding the tags), and replacement array (holding information needed by replacement policy). In addition, each one of these arrays is physically distributed into multiple sub-arrays (one per way) that can be individually accessed. For example, a four-way set associative LRU cache has four data sub-arrays, four tag sub-arrays, and four replacement sub-arrays. The replacement sub-arrays are accessed once every access when the LRU replacement is used. However, it is not needed when Random replacement is used. For a specific cache, it was determined that the accesses to the different arrays have the following energy consumption:

Array	**Energy Consumption Per Way Accessed**
Data Array	20 energy units
Tag Array	5 energy units
Replacement Array	1 energy unit

Estimate the energy consumption for the following configurations. The cache is 4-way set associative. Main memory access and cache refill (although important) are not considered here. Provide answers for the LRU and Random replacement policies.

a) (3 pts) A cache read hit. All arrays are read simultaneously.

b) (3 pts) The cache access is now split across two cycles. In the first cycle, all tag sub-arrays are accessed. In the second cycle, only the sub-array whose tag matched will be accessed. Repeat part (a) for a cache read hit.

c) (3 pts) Repeat part (b) for a cache read miss, assuming that no data array access in the second cycle for a cache miss.

Solution:

a) Read Hit (simultaneous access):
LRU: 4 parallel accesses for data/tag/replacement arrays = 4 × (20+5+1) = 104 energy units
Random: 4 parallel accesses for data/tag arrays = 4 × (20+5) = 100 energy units

b) Read Hit (split access)
LRU: 4 parallel accesses for tag/replacement arrays + one access to hit data sub-array
= 4 × (5+1) + 20 = 44 energy units
Random: 4 parallel accesses for tag arrays + one access to hit data sub-array
= 4 × 5 + 20 = 40 energy units

c) Read Miss (split access)
LRU: 4 parallel accesses for tag/replacement arrays = 4 × (5+1) = 24 energy units
Random: 4 parallel accesses for tag arrays = 4 × 5 = 20 energy units

5) (9 pts) The following table shows parameters of a virtual memory system:

Virtual Address	Maximum Physical Memory	Page Size	Page Table Entry
50 bits	64 GB	16 KB	4 bytes

a) (2 pts) For a single-level page table, how many page table entries are needed? How much physical memory is needed for storing the page table?

Page size = 16 KB ➔ Page offset = 14 bits
Virtual page number = 50 bits – 14 bits = 36 bits
Number of page table entries = 2^{36}
Page Table Size = $2^{36} \times 4 = 2^{38}$ bytes = 256 GBytes

b) (3 pts) Using a multilevel page table can reduce the physical memory allocation of page tables. How many levels of page tables will be needed, given the size of the page table at any level is the size of a page?

Size of page table = Size of a page = 16 KB
Each page table entry is 4 bytes ➔ Number of entries per page table = $2^{14} / 4 = 2^{12}$ entries
12 bits are needed to index the page table at any level
Virtual page number = 36 bits ➔ Three page table levels are needed.

A virtual address is shown below:

L3 page table index <12 bits>	L2 page table index <12 bits>	L1 page table index <12 bits>	Page Offset <14 bits>

c) (2 pts) The architect wants to support a large page size, what will be the best choice for a large page size and why?

Small page size = 16 KB. It is mapped by the L1 page table.
Since there are 2^{12} entries in the L1 page table each pointing to a 16 KB page, the architect can define a large page size = 2^{12} x 2^{14} = 2^{26} bytes (64 Mbytes).
The large page size is pointed directly by the L2 page table.

d) (2 pts) The architect wants to design a 64 KB cache that should be indexed in parallel with the TLB (address translation). The cache should be physically tagged and should not have any aliasing problem. What should be the minimum associativity of the cache to avoid aliases?

To avoid aliases, the cache should be indexed using the page offset bits only.
For a direct mapped cache, the cache size cannot exceed 2^{14} = 16 Kbytes.
To increase the cache capacity (but without causing aliases) the associativity of the cache should be increased.
To design a 64 KB cache the minimum associativity should be 64 KB / 16 KB = 4.