Computer Science 605.611

Problem Set 12

1. A certain program consists of three independent parts A, B and C that can execute in parallel on separate processors within a multiprocessor system. None of the parts can be further subdivided and each part must execute on a single processor. Part A requires 78 billion cycles to complete, part B requires 40 billion cycles to complete and part C requires 62 billion cycles. The program is executed on an SMP system with 4 identical processors all running at the same 2 GHz clock rate.

2 * 10 ^ 9 GHz -> 5 * 10 ^ -10 s / instruction

a) (5) How long does it take to execute the entire program (all 3 parts) if only one of the processors is used to execute the program (the other 3 processors remain idle)?

Executed in series, A + B + C = 78 + 40 + 62 = 180 billion cycles
180,000,000,000 * 5 * 10 ^ -10 = 90s

b) (5) What is the minimum time required to execute the entire program (all 3 parts) if only two of the processors are used to execute the program (the other 2 processors remain idle)?

A can be executed in parallel with B and C
Executing B and C in series, B + C = 40 + 62 = 102 billion cycles
102,000,000,000,000 * 5 * 10 ^ -10 = 51s

c) (5) What is the minimum time required to execute the entire program (all 3 parts) if all 4 processors are used?

Executing A, B, and C in parallel takes 78 billion cycles (A is the limiting process)
78,000,000,000,000 * 5 * 10 ^ -10 = 39s

2. Another different program consists of a purely sequential part that requires 20 seconds to execute on a single-core processor. The sequential part must execute first and is followed by a parallel part containing 4 independent indivisible tasks each of which takes 12 seconds and can only run on a single processor.

a) (5) What is the total execution time for the entire program on the single-core processor?

20 + 12 (4) = 68s

b) (5) Compared to the single-core processor, what is the actual speedup achieved by running the program on a 4-core version of the processor?

20 + 12 = 32 s
Speedup = 68 / 32 = 2.125

c) (5) What speedup is predicted by Amdahl's law for the program running on the 4-core system compared to the single-core system?
Recall that for multi-processor or multi-core systems Amdahl's law states:

$$\text{Speedup} = N / [f + N(1-f)]$$

where f is the fraction of the total execution time accounted for by the parallel part and N is the number of cores on the multi-core system.

$f = 48/68$
$N = 4$
Speedup = $4 / [(48/68) + 4(1-(48/68))] = 2.125$ (Amdahl's law works!)

3. (10) A different program is to compute two products: one is a vector product of two vectors each containing 100 elements; the other product is produced by multiplying all of the elements of a matrix by a single scalar constant. The matrix has 1000 rows and 1000 columns. The vector product must be computed first and each of its elements is the product of the corresponding elements in the two vectors. That is, P[i] = A[i]*B[i] where P is the product vector and A and B are the two vectors that are multiplied. Multiplying one vector element by another takes 1 cycle and multiplying the scalar by a single matrix element takes one cycle. The program is run first on a single processor. What speedup is provided for the same program if 499 additional identical processors are included so that the system has a total of 500 processors and there are no memory conflicts or other dependencies?

Multiplying 100 elements in a vector with 100 elements in another vector is 100 operations, requiring 100 cycles. Multiplying a matrix with 1000 rows and 1000 columns will take 1000000 operations and 1000000 cycles.

Running with just one processor will take 100 + 1000000 cycles = 1,000,100 cycles

Running on multiple processors, we can tackle the first and second tasks in parallel, and the individual tasks in series. Thus, the first task will require 100 processors and take 1 cycle. The second task will use all 500 processors and take 1000000 / 500 = 2000 cycles. Thus the whole operation will take 2001 cycles.

The speedup is 1000100 / 2001 = 499.8 (very close to maximum speedup)

4. A dual processor SMP system includes an L1 data cache for each processor and employs the MESI protocol to maintain cache consistency. Each cache is a 2-way set associative copy-back cache that contains a total of 8192 cache lines each of which is 256 bytes in size. The lines or ways within each empty set are filled in the order Way0, Way1, Way2 then Way3. A write-allocate policy is used for each cache. One process, P1, runs on the first processor at the same time that another process, P2, runs on the other processor. The first access made is when P1 reads a variable X that resides in memory at address 0x400804C0 and contains an initial value of 80. After P1 reads X, P2 writes the value 156 into a variable Y located at memory address 0x400804F8. All caches are initially empty, so each cache line starts out in the invalid (I) state.

256 * 8 = 2 ^ 11
0x400804F8 and 0x400804C0 lie on the same cache line

a) (5) After P1 first reads X, what is the MESI state of the line containing X in P1's cache and how is P2's cache affected? That is, what change (if any) occurs in the MESI state for the affected line or lines in the two caches?

The MESI state of the line containing X in P1 cache is set to E – Exclusive. The MESI state of that line in P2's cache is not affected, as that line has not been loaded yet.

b) (5) After P1 first reads X and P2 then writes Y, what is the MESI state of the line containing Y in P2's cache and how is P1's cache affected? That is, what change (if any) occurs in the MESI state for the affected line or lines in the two caches?

The MESI state of the line containing Y in P2's cache is set to S – Shared. The line containing X in P1's cache should be changed from E to I – Invalid, since the value is outdated compared to what is in memory.

5. (15) A program runs on a superscalar uniprocessor with a 2 GHz clock rate within a NUMA (non-uniform memory access) system. The number of instructions executed in the program is IC. Each instruction that does not reference remote memory takes 2 cycles. Each instruction that does reference remote memory incurs an additional 200 ns penalty over those that do not reference remote memory. The percentage of program instructions that access remote memory is 0.2%. What speedup is achieved for the program if the remote memory access penalty is somehow reduced from 200 ns to 10 ns per access?

2 GHz = 0.5 ns / cycle
Assume IC = 500 (round number)
500 instructions * (2 cycles/instruction) * (0.5 ns/cycle) + 200 ns * (0.2% * 500) =
1000 cycles * (0.5 ns/cycle) + 200 ns * 1 = 700 ns

500 instructions * (2 cycles/instruction) * (0.5 ns/cycle) + 10 ns * (0.2% * 500) =
1000 cycles * (0.5 ns/cycle) + 10 ns * 1 = 510 ns

speedup = 700 / 510 = 1.37

or without assuming IC = 500:

$$speedup = \frac{IC * 2 * 0.5 + 200 * 0.002 * IC}{IC * 2 * 0.5 + 10 * 0.002 * IC}$$

$$speedup = \frac{1 + .4}{1 + \ .02} = 1.37$$

6. (15) A uniprocessor system accesses memory over a 32-bit bus. The processor has a direct mapped write-back (as opposed to write-through) data cache that operates in look-through mode and employs a write-allocate policy. The data cache contains 65536 lines, each of which is 1024 bytes in size. Detecting a cache miss, or performing a cache read, or performing a cache write for the data cache each takes 2 CPU clock cycles. Loading a memory block into the data cache or writing a data cache line back to memory takes 400 CPU clock cycles. Recall that for a cache miss, the data item that is needed from the memory block is transferred to the CPU in parallel with loading the memory block containing the data item into cache.

Each of the 134217728 four-byte elements in an integer array are read and updated by the code shown below. The address of the array in memory is 0x10084000.

```
            lui     $8,0x1008       #point to the first element to be processed
            ori     $8,0x4000
            lui     $4,0x0800       # number of elements = 134217728 (=0x08000000)
loop:       lw      $12,0($8)       # get next element
            sub     $12,$0,$12      # negate by subtracting from 0
            addi    $8,$8,4         # point to next element
            sw      $12,-4($8)      # update the element that was read
            bgtz    $4,loop         # repeat if more elements remain
            addi    $4,$4,-1        # decrement loop control variable
```

In an attempt to speedup the processing of the array, the code is executed on an 8-core system in which each core has a separate data cache identical to the data cache for the uniprocessor. A speedup factor of 8 for the 8-core system compared to the uniprocessor is defined as "linear" speedup.

What is the actual speedup provided by the 8-core system based on just the time required to read and update all of the array elements? Ignore the time required by the instructions that do not reference memory as well as any possible memory conflicts and the flushing of the caches once the code completes. Each of the 8 cores processes a separate contiguous subset of elements by executing a separate copy of the code sequence. Each subset corresponds to 1/8th of the array. Assume that all data caches are initially empty and that the final cache flush when the program ends is handled by the operating system (so it does not contribute to the code's execution time).

Each line is 1024 bytes, thus each line holds 256 elements.
The cache can hold 65536 * 256 = 16777216 array elements, which is 1/8th of the array
Stepping incrementally through the array elements causes a data cache miss on the initial reference to the first element in a cache line. All 256 lines in the cache are referenced twice (read then write), thus for every 512 references there will be 1 cache miss and 511 cache writes. This is the

case for both the uniprocessor and 8-core system. Since the 8-core system operates perfectly in parallel if we ignore all non-memory calls, the 8 core system will complete in 1/8th the time, thus providing a speedup of 8, which is linear.

7. (5) Use our previously described 3-bit pseudo-LRU replacement algorithm for a 4-way set associative cache as a guide for designing a pseudo-LRU replacement algorithm for an 8-way set associative cache. What is the minimum total number of pseudo-LRU bits required for each set in the 8-way set associative cache? Describe how the pseudo-LRU bits are used with the 8-way set associative cache. You do not need to specify the actual bit patterns.

As there are 3 bits used to track the pseudo-LRU replacement for a 4-way set associative cache, there must be 7 bits used to track the pseudo-LRU replacement for an 8-way set associative cache. Each bit essentially represents one branch point in a binary decision tree. When a way is accessed (read or written) each of the bits that represent branches in the binary tree are updated to reflect that their branch was accessed most recently. When one wants to overwrite a way, one traverses the binary tree and at each branch determine which side (left or right) was accessed *less* recently, thus arrives at the pseudo-LRU way to be overwritten.

8. Answer the following questions about cache coherency protocols in a dual-processor system.

a) (5) If each processor has a separate cache, what is the main advantage of the 4-state MESI snoopy cache coherency protocol compared to a snoopy cache coherency protocol that uses only 3 states M, S and I (modified, shared and invalid)?

A protocol which only utilizes M, S, and I excludes the E – exclusive state. This state indicates that the value in cache both matches memory and is exclusive to this cache. Having this state is advantageous in comparison to the MSI protocol in the case of a write hit. For both the MESI and MSI protocols, the value needs to be written to the cache and to memory, however in this case the MESI protocol does not need to scan the other caches in the system to see if/where the other instances of that data is, as it knows that the data is exclusive to that cache.

b) (5) Suppose that processor P1 (in a dual-processor system) writes to a memory block. A copy of this same memory block is already in a modified line within the local cache of processor P2. What actions should be taken for the modified line within P2's cache? Explain your answer.

According to the table:

Bus Transaction	Action by local cache
Read hit	E➔ S, M➔ S or no change if S
Read miss	no change
Write hit	E ➔ I, S ➔ I, M ➔ I
Write miss	no change

A line that is previously M and has a write hit is changed to state I – invalid in the MESI protocol. No change is needed to the cache line.

c) (5) Assume that the 3-state MSI cache coherency protocol is used with a single 2-way set associative cache containing a total of 2097152 lines each of which is 512 bytes in size. What is the minimum total number of MSI state bits needed for the entire cache?

A 3 state MSI cache coherency protocol needs at least 2 bits per line to determine the 3 states the line can be in. A cache containing 2097152 lines would then need 2097152 * 2 bits to store the state, for a total of 4,194,304 bits.