Computer Science 605.611

Problem Set 12 Answers

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.

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)?
The single processor consumes (78+40+62) billion cycles = 180 billion cycles. This corresponds to 180 billion cycles/(2 billion cycles/sec) = 90 seconds.

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)?
One processor executes part A in 78 billion cycles while the other processor executes part B. Part B is completed after 40 billion cycles. The second processor could then execute part C. Hence the minimum time to execute the program corresponds to Max(78, (40+62)) = 102 billion cycles. So it takes 102/2 = 51 seconds. Doubling the number of processors does not cut the execution time in half.

c) (5) What is the minimum time required to execute the entire program (all 3 parts) if all 4 processors are used?
In this case, parts A, B and C could each be executed in parallel by three separate processors. Therefore the minimum time to complete the program would correspond to the longest part (78 billion cycles). The required execution time = 78/2 = 39 seconds.

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?
Execution time on single-core = 20 + 4*12 = 68 seconds.

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?
Execution time on 4-core system = 20 + 12 = 32 seconds
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:

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.

From part a) the total execution time for the program on a single core is 68 seconds. The sequential part requires 20 seconds and the parallel part accounts for the remaining 48 seconds.
Hence f = 48/68 and N=4
Speedup = 4/[48/68 + 4*20/68] = 4/[128/68] = 4*68/128 = 272/128 = 2.125
Which is the same speedup computed in part b).

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?

The vector product requires multiplying the corresponding elements from the two vectors, this takes 100 cycles on a single processor. The matrix contains 1000*1000 = 1000000 elements. So, multiplying the matrix by a scalar takes 1000000 cycles. Hence a single processor takes 1000100 cycles.

With 500 processors, the vector product could be computed in just one cycle by using a different processor for each element in the vector product (the other 400 processors would be idle). For the matrix product, 500 processors could multiply the scalar by one half of the elements in each of the rows of the matrix at the same time.
Hence multiplying the scalar times all 1000 elements in each row of the matrix takes 2 cycles. So multiplying the scalar times all 1000 rows takes 1000*2 = 2000 cycles. Therefore the sum of the times for the vector and matrix products is 2001 cycles.
This yields a speedup = 1000100/2001 = 499.8

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.

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?

Each set contains 2 lines. There are 8192/2 = 4096 sets in the cache. The variable X resides within memory block 0x400804 which maps to set 0x0804 = 2052 in P1's cache.
Since this is the first read from this block, the state of the line in P1's cache changes from invalid (I) to exclusive (E) and there is no change to P2's cache.

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 variable Y resides within memory block 0x400804 as well and maps to set 2052 in P2's cache. The state of the line containing a copy of this memory block in P2's cache changes from I to M. The line containing a copy of the same block in P1's cache changes from E to I.

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?

The 2 GHz clock rate corresponds to a 0.5ns cycle time, so the extra 200 ns corresponds to 400 clock cycles. Instructions that do not reference remote memory take 2*0.5ns = 1ns to complete.
Speedup = T1/T2 where T1 is the execution time with the 200ns remote memory access penalty and T2 is the execution time with the 10 ns remote memory access penalty.
T1 = IC*1ns + IC*0.002*200ns = IC*(1.0ns + 0.4) = IC*1.4ns
T2 = IC*1ns + IC*0.002*10ns = IC*1.02ns
Therefore the speedup = 1.4/1.02 = 1.3725

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 $1/8^{th}$ 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).

Reading an array element that is not already in the cache requires 2 cycles to detect the miss plus 400 cycles to load the memory block containing the element into cache while returning the element to the CPU.
Each write of an array element is a hit, since elements are only written after they have been read. So 2 cycles are required to write the element into cache. Updating the first element in each line

requires 2 + 400 + 2 = 404 cycles. Updating each element that is already in the cache requires 4 cycles (2 cycles for the read hit plus 2 cycles to write the updated element).

So updating all elements in a line requires 404 + (N-1)*4 cycles where N is the number of elements in the line. That is, 404+255*4 = 1424 cycles per line.

Each block contains 1024/4 = 256 elements. The starting address 0x10084000 corresponds to the beginning address of a memory block. The total array requires 134217728/256 = 524288 memory blocks. Updating the first 65536 blocks fills the cache for the first time and takes a total of 65536*1424 = 93323264 cycles.

Thereafter, each block loaded into the cache must replace a line already in the cache.
Each replacement adds 400 cycles to the total time.
Hence the next 524288-65536 = 458752 blocks requires 400+1424 = 1824 cycles to update the array elements contained in the block.
The grand total number of cycles required to update the entire array on the single core system = 93323264 + 458752*1824 = 930086912 cycles

With the 8-core system, each core updates 1/8 of the array elements (=524288/8 or 65536 elements). So each core only has to fill its cache once and none of the lines have to be replaced. It takes 65536 * 1424 = 93323264 cycles to update the 8 subsets of array elements in parallel.

This corresponds to a speedup of 930086912 /93323264 = 9.97

This is greater than the linear speedup of 8 and is referred to as “superlinear” speedup.

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.

Seven bits per set are needed; one bit is used to divide the 8-way system into two 4-way subsystems, and a separate 3-bit group is used for each of the two 4-way subsystems. The minimum number of pseudo-LRU bits required per set = 1 +2*3 = 7.

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)?

With the MESI protocol, the first time that a memory block is loaded into any cache its state is changed from invalid to exclusive. The exclusive state indicates that the newly loaded line resides in no other caches so a write to the line does not have to be broadcast to other caches. This reduces the traffic on the shared bus. Without the exclusive state, the state of the newly loaded line would change from invalid to shared. So the shared state would now indicate that the line is in the local cache and may be in one or more other remote caches. A write to the cache line would have to be broadcast on the bus to alert the other caches just in case they contain a copy of the line.

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.

Even though the line has already been written to locally in P2's cache, the write from a remote processor (P1) could be to a different location within the line than the location that was written to by P2. Hence a back-off signal should be sent to the remote processor (P1) and P2's modified local cache line is written back to memory and marked as invalid. The remote processor (P1) can then complete its write operation. So P2's cache line will be invalid after P1 completes its write.

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 1048576 lines each of which is 512 bytes in size. What is the minimum total number of MSI state bits needed for the entire cache?

The state must be recorded for each cache line. Since there are three states (M,S, I), at least 2 bits are needed to encode the state. Therefore the total number of MSI bits required for the cache is 2*2097152= 4194304.