CSC 656
Homework #???
by
Andrew Hutzel

Exercise 5.1

5.1.1) How many 32-bit integers can be stored in a 16-byte cache block?

16*8=128 and we have 32 bit integers so… 128/32 = 4

We can store four 32 bit integers in a 16 byte block.

5.1.2) References to which variables exhibit temporal locality?

a. Temporal

Memory address is referenced at the CURRENT time, the same address will be referenced again soon…

b. Spatial

If a memory address is referenced at the current time, an address close to it will be referenced soon

Knowing this we want to think of a particular sequence such as

A[0][0]=B[0][0]+A[0][0]
A[1][0]=B[0][0]+A[0][1]
A[2][0]=B[0][0]+A[0][2]
Etc

The variables that exhibit temporal locality are: I, J, and B[I][0].

5.1.3) References to which variables exhibit spatial locality?

<u>Refer to previous example and we find that the variables that exhibit spatial locality are A[I][J] and B[I][0]. Note B[I][0] exhibits both localities.</u>

5.1.4) How many 16-byte cache blocks are needed to store all 32-bit matrix elements being referenced?

Consider what the program is doing before answering… The goal of this algorithm is too take a matrix with 8 rows and 8000 columns and transforming it into a 8000 row and 8 column array.

Briefly consider the access sequence here:

A(1,1) = B(1,0) + A(1,1)
A(1,2) = B(1,0) + A(2,1)
A(1,3) = B(1,0) + A(3,1)
…
A(1,8000) = B(1,0) + A(8000,1)
A(2,1) = B(2,0) + A(1,2)
A(2,2) = B(2,0) + A(2,2)
…
A(2,8000) = B(2,0) + B(8000,2)
…
A(8,1) = B(8,0) + A(1,8)
A(8,2) = B(8,0) + A(2,8)
…
A(8,8000) = B(8,0) + A(8000,8)

Knowing this we can now answer the question….

Matrix A has 8x8000=64000 elements
Total elements from start to finish is 64,000x2 = 128,000
Matrix elements per cache BLOCK is 4 (Given in 5.1.1)
Total cache blocks 128,000 / 4 = 32,000

But we're missing one key point, the upper left 8x8 elements are going to overlap. Cache isn't required for this we can remove these therefore we have 8x8/4=16 cache blocks…

Matrix B has 8x1 = 8 elements
Matrix elements per cache block is 4 (5.1.1)
Total cache blocks 8 / 4= 2
Total 16 byte cache blocks gives us 32,000-16+2 ⇒ ***31,986 cache blocks are needed.***

5.1.5) Which variables exhibit temporal locality?

Variables I, J, and B[I][0] have temporal locality.

5.1.6)Which variables exhibit spatial locality?

The variables that exhibit spatial locality are A[I][J] and **B[I][0]**.

5.2.1) Below is a list of 32-bit memory address references, given as word addresses.

3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

So I was really confused by this question at first and then after looking at it for a little while I was able to figure out what it was.

16 one-word block, via direct mapped cache (2^0 = 1 word block, but we have 16 blocks 2^4 = 4 index bits meaning we have 28 bits left for the tag...

Address		Tag		Index		Hit or miss	
Decimal	Binary	Binary	Decimal	Binary	Decimal	Hit	Miss
3	0000 0011	0000	0	0011	3		✓
180	1011 0100	1011	11	0100	4		✓
43	0010 1011	0010	2	1011	11		✓
2	0000 0010	0000	0	0010	2		✓
191	1011 1111	1011	11	1111	15		✓
88	0101 1000	0101	5	1000	8		✓
190	1011 1110	1011	11	1110	14		✓
14	0000 1110	0000	0	1110	14		✓
181	1011 0101	1011	11	0101	5		✓
44	0010 1100	0010	2	1100	12		✓
186	1011 1010	1011	11	1010	10		✓
253	1111 1101	1111	15	1101	13		✓

5.2.2)

For each of these references, identify the binary address, the tag,
and the index given a direct-mapped cache with two-word blocks and a total size of 8
blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

8 2-word blocks, direct mapped cache

Address		Tag		Index		Hit or miss	
Decimal	Binary	Binary	Decimal	Binary	Decimal	Hit	Miss
3	0000 0011	0000	0	001	1		✓
180	1011 0100	1011	11	010	2		✓
43	0010 1011	0010	2	101	5		✓
2	0000 0010	0000	0	001	1	✓	
191	1011 1111	1011	11	111	7		✓
88	0101 1000	0101	5	100	4		✓
190	1011 1110	1011	11	111	7	✓	
14	0000 1110	0000	0	111	7		✓
181	1011 0101	1011	11	010	2	✓	
44	0010 1100	0010	2	110	6		✓
186	1011 1010	1011	11	101	5		✓
253	1111 1101	1111	15	110	6		✓

5.2.3 <§§5.3, 5.4> You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of 8 words of data: C1 has 1-word blocks, C2 has 2-word blocks, and C3 has 4-word blocks. In terms of miss rate, which cache design is the best? If the miss stall time is 25 cycles, and C1 has an access time of 2 cycles, C2 takes 3 cycles, and C3 takes 5 cycles, which is the best cache design?

There are many different design parameters that are important to a cache's overall performance. Below are listed parameters for different direct-mapped cache designs.

Cache Data Size: 32 KiB
Cache Block Size: 2 words
Cache Access Time: 1 cycle

Address		Tag		*C1*		*C2*		*C3*	
DEC	BIN	DEC	BIN 5 bits	BIN	H/M	BIN	H/M	BIN	H/M
3	011	0	00000	011	M	01	M	0	M
180	10110100	22	10110	100	M	10	M	1	M
43	*00101011*	5	00101	011	M	01	M	0	M
2	10	0	00000	010	M	01	M	0	M
191	10111111	23	101111	111	M	11	M	1	M
88	*01011000*	11	01011	000	M	00	M	0	M
190	*10111110*	23	10111	110	M	11	H	1	H
14	*00001110*	1	00001	110	M	11	M	1	M
181	*10110101*	22	10110	101	M	10	H	1	M
44	*00101100*	5	00101	100	M	10	M	1	M
186	*10111010*	23	10111	010	M	01	M	0	M
253	*11111101*	31	11111	101	M	10	M	1	M

So we can figure out from the 12 given values what MISS rate we have currently.
C1:
12/12 Misses so 100% miss rate
C2:
10/12 Misses so 83% miss rate
C3:
11/12 Misses so 92% miss rate

So we can deduce that C2 is the best!

Performance:
C1:
12 accesses means 12 x2 = 24 cycles
12 misses means 12 x 25 = 300 cycles
Total: 324 Cycles

C2:
12 accesses means 12 x 3 = 36 cycles
10 misses means 10 x 25 = 250 cycles
Total: 286 cycles

C3:
12 accesses means 12 x5 = 60 cycles
11 misses 11 x 25 = 275 cycles
Total: 335 cycles

And so we're able to figure out that C2 requires the fewest clock cycles, at 286.

5.2.4 [15] <§5.3> Calculate the total number of bits required for the cache listed above, assuming a 32-bit address. Given that total size, find the total size of the closest direct-mapped cache with 16-word blocks of equal size or greater. Explain why the second cache, despite its larger data size, might provide slower performance than the first cache.

So we know the total bits can be calculated using:
Total bits= (2^index bits) * (valid bits + tag bits + (data bits * 2^offset bits))

Total bits = 2^15 (1+14+(32*2^1))
=2588672 bits

Total bits = 2^13 (1+13 (32*2^4))
=4308992

The next smallest cache with the 16 word block and a 32 bit addr is a total of 215892 bits, which is smaller than the first cache. The larger cache despite its data size might provide slower performance than the first cache because it requires a longer access time which in turn leads to lower performance.

5.2.5 [20] <§§5.3, 5.4> Generate a series of read requests that have a lower miss rate on a 2 KiB 2-way set associative cache than the cache listed above. Identify one possible solution that would make the cache listed have an equal or lower miss rate than the 2 KiB cache. Discuss the advantages and disadvantages of such a solution.

To have the 2KB, 2-way SA cache have a lower miss rate, we want to find addresses that will map to the same block in the larger direct mapped cache. While these blocks will all map to the same set, because there are 2-ways we can tolerate 2 blocks to the same set w/o replacing a block. Thus if we come back and access the first block we should get a hit in the set associative cache where we will get a miss in the direct mapped cache.

EX: (CONTINUED ON NEXT PG)

Address	Block in 64 KB	H/M	Set in KB	H/M
0x00000	Block 0	M	Set 0 Way 0	M
0x10000	Block 0	M	Set 0 Way 1	M
0x00000	Block 0	M	Set 0 Way 0	H

Some of the advantages, based off of this theoretical example provided is that the user doesn't need to alter the address much as the set bits are the same and the mapping between blocks of cache with set associative cache will be easier. The disadvantages is that this makes the method complex and achieving read hits much harder. To make matters worse this would make the series of read requests approach two different methods for different types of caches which again would add to the complexity of the problem.

Address	Block in 64 KB	H/M	Set in 2 KB	H/M
0x00000	Block 0	M	Set 0 - Way 0	M
0x00400	Block 0x80=128	M	Set 0 - Way 1	M
0x00800	Block 0x100=256	M	Set 0 - Way 0	M/evict 00000
0x00000	Block 0	M	Set 0 - Way 1	M

5.2.6 [15] <§5.3> The formula shown in Section 5.3 shows the typical method to index a direct-mapped cache, specifically (Block address) modulo (Number of blocks in the cache). Assuming a 32-bit address and 1024 blocks in the cache, consider a different indexing function, specifically (Block address[31:27] XOR Block address[26:22]). Is it possible to use this to index a direct-mapped cache? If so, explain why and discuss any changes that might need to be made to the cache. If it is not possible, explain why.

It is in fact POSSIBLE to use the function block address[31:27] XOR block address[31:27] to index a direct-mapped cache. BUT, we can only achieve this if we modify the cache since XOR will result in the loss of the five tag bits we would usually have for a 32 bit address. Knowing this, we can only use this function to INDEX a direct-mapped chance if we use different tag bits so the address can be identified in the cache.

5.3⇒

5.3.1)What is the cache block size (in words)?

Offset: 5 bits = 2^5 = 32 byte
Words: 32/4 = **8 words per block**

5.3.2) How many entries does the cache have?

Entries in the cache is given by the index so 5 bits… 2^5 = **32 entries**

5.3.3) What is the ratio between total bits required for such a cache implementation over the data storage bits?

Each block with 32 bytes of data = (32*8) = 256 bits + 22 bits of tag + 1 valid bit
= 279/256
= **1.089**

5.3.4) How many blocks are replaced?
(N means No and Y means yes….)

Addr	Index	Offset	H/M	Replace?	Last Value?
0	00000	00000	M	N	N
4	00000	00100	H	N	N
16	00000	10000	H	N	N
132	00100	00100	M	N	N
232	00111	01000	M	N	Y
160	00101	00000	M	N	Y
1024	00000	00000	M	Y	N
30	00000	11110	M	Y	N
140	00100	01100	H	N	N
3100	00000	11100	M	Y	Y
180	00101	10100	H	N	Y
2180	00100	00100	M	Y	Y

Notice that there are 4 values for 'replace' that mark yes and 4 values in 'last value' that also mark yes. This can directly mean there have been 4 blocks replaces (incomplete blocks might mean we floor the number? 1.5⇒ 1 block replaced?)

5.3.5) What's the hit ratio?
4/13 = 33%

5.3.6)List the final state of the cache, with each valid entry represented as a record of <index,tag,data>

<00100,0010,mem[2176]>
<00101,00000,mem[160]>
<00000,0011,mem[3072]>
<00111,0000,mem[224]>

5.6 In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that memory accesses are 36% of all instructions. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

5.6.1 [5] <§5.4> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

Clock rate = (1/cycle time) and (cycle time = L1 hit time)

P1: 1/0.66 ns ⇒ 1.52 Ghz
P2: 1/0.9 ns ⇒ 1.11 Ghz

5.6.2 [5] <§5.4> What is the Average Memory Access Time for P1 and P2?

a) Average Memory Access Time For P1 = L1 hit time + (L1 miss rate * Memory Access Time)

=> (AMAT FOR P1) = 0.66 + (8%*70) = 0.66 + 5.6 = 6.26 ns

b) Average Memory Access Time For P2 = L2 hit time + (L2 miss rate * Memory Access Time)

=> (AMAT FOR P2) = 0.90 + (6%*70) = 0.90 + 4.2 = 5.1 ns

5.6.3 [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster?
For the next three problems, we will consider the addition of an L2 cache to P1 to presumably make up for its limited L1 cache capacity. Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

a) Total CPI For P1 = Base CPI + [(Memory Access Time*L1 miss rate)/L1 Hit time] * Number of memory instructions

=> Total CPI For P1 = 1 + [(70*8%)/0.66]*0.36 = 4.054

b) Total CPI For P2 = Base CPI + [(Memory Access Time*L1 miss rate)/L1 Hit time] * Number of memory instructions

=> Total CPI For P2 = 1 + [(70*6%)/0.90]*0.36 = 2.68

As Total CPI For P2 is less than the Total CPI For P1, Processor P2 is Faster.

5.6.4 [10] <§5.4> What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

Average Memory Access Time For P1 with the addition of a L2 Cache

= L1 hit time + L1 Miss rate*(L2 Hit time + L2 Miss rate*Memory Access Time)

= 0.66 + 8%*(5.62+95%*70)

= 0.66 + 8%*(5.62+66.5)

= 0.66 + 8%*72.12 = 0.66 + 5.76 = 6.4296 ns

As the average memory access time has increased now, the AMAT has become worse on adding an extra L2 Cache.

5.6.5 [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

Total CPI = Base CPI + Number of memory instructions * [L1 miss rate * (L2 Hit in Cycles + L2 Memory Miss In Cycles)]

= 1 + 0.36*[0.08*(5.62+0.95*70)/0.66] = 1 + 0.36*[0.08*109.27] = 4.14

5.6.6 [10] <§5.4> Which processor is faster, now that P1 has an L2 cache? If P1 is faster, what miss rate would P2 need in its L1 cache to match P1's performance? If P2 is faster, what miss rate would P1 need in its L1 cache to match P2's performance?

Now processor P2 is only faster though P1 has a L2 Cache.

P2 time = 2.68*0.9 = 2.41 ns/instruction

For P1 to be faster than P2, 0.66 * (1+0.36*X*106) < 2.41

Solve to get X which is:-

X = 0.069 = 69% Miss Rate

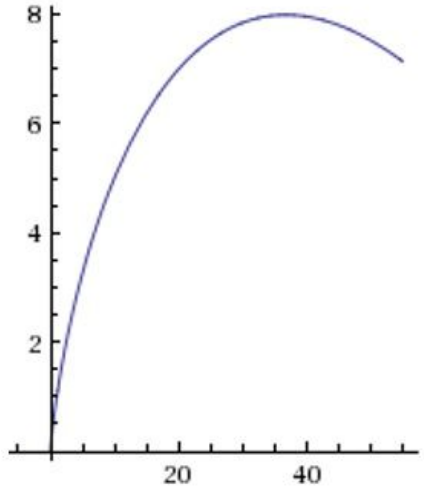
Exercise 6.3

Many computer applications involve searching through a set of data and sorting the data. A number of efficient searching and sorting algorithms have been devised in order to reduce the runtime of these tedious tasks. In this problem we will consider how best to parallelize these tasks.

6.3.1 [10] <§6.2> Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value *X* in a sorted N-element array A and returns the index of matched entry:

Assume that you have Y cores on a multi-core processor to run BinarySearch. Assuming that Y is much smaller than N, express the speedup factor you might expect to obtain for values of Y and N. Plot these on a graph.

- Binary search ideal for parallism
- Increasing Y beyond 2 or 3 would have no benefits
- ideally we can
    On core 1: perform the comparison between low and high
    On core 2: perform the computation for mid
    On core 3: perform the comparison for A[mid]
- Without additional instructions/reforming of the code, no speedup would occur …and communication between cores is not free



6.3.2 [5] <§6.2> Next, assume that Y is equal to N. How would this affect your conclusions in your previous answer? If you were tasked with obtaining the best speedup factor possible (i.e., strong scaling), explain how you might change this code to obtain it.

Answer:

- Create threads to compare the N elements to the value X and perform these in parallel
- Then, we can get ideal speed-up (Y)
- Entire comparison can be completed in the amount of time to perform a single computation

Exercise 6.4
Consider the following piece of C code:

```
for (j=2;j<1000;j++)
    D[j] = D[j−1]+D[j−2];
```

The MIPS code corresponding to the above fragment is:

```
        addiu   $s2, $zero, 7992
        addiu   $s1, $zero, 16
loop:   l.d     $f0, -16($s1)
        l.d     $f2, -8($s1)
        add.d   $f4, $f0, $f2
        s.d     $f4, 0($s1)
        addiu   $s1, $s1, 8
        bne     $s1, $s2, loop
```

Instructions have the following associated latencies (in cycles):

6.4.1 [10] <§6.2> How many cycles does it take for all instructions in a single iteration of the above loop to execute?

2 x 6 + 1 x 4 + 1 x 1+ 1 x 2 + 1 x 1 = 20 cycles, assuming 1 cycle for bne, The two addiu take 4 cycles, but are not part of the loop iteration

6.4.2 [10]<§6.2> When an instruction in a later iteration of a loop depends upon a data value produced in an earlier iteration of the same loop, we say that there is a *loop carried dependence* between iterations of the loop. Identify the loop-carried dependencies in the above code. Identify the dependent program variable and assembly-level registers. You can ignore the loop induction variable j.

D[j] and D[j-1] are loop carried dependencies, D[j] in the current iteration is the D[j-1] in the next iteration and D[j-1]- in the current iteration is the D[j-2] in the next iteration. These variables are stored in registers $f2 and $f4.

6.4.3 [10] <§6.2> Loop unrolling was described in Chapter 4. Apply loop unrolling to this loop and then consider running this code on a 2-node distributed memory message passing system. Assume that we are going to use message passing as described in Section 6.7, where we introduce a new operation send (x, y) that sends to node x the value y, and an operation receive( ) that waits for the value being sent to it. Assume that send operations take a cycle to issue (i.e., later instructions on the same node can proceed on the next cycle), but take 10 cycles be received on the receiving node. Receive instructions stall execution on the node where they are executed until they receive a message. Produce a schedule for the two nodes assuming an unroll factor of 4 for the loop body (i.e., the loop body will appear 4 times). Compute the number of cycles it will take for the loop to run on the message passing system.

Node 1:

```
ADDIU $s2, $zero, 7992
ADDIU $s1, $zero, 16
L.D f0, –16($s1)
L.D f2, –8($s1)
loop:
ADD.D $f4, $f0, $f2
ADD.D $f5, $f4, $f0
Send (2, $f4)
Send (2, $f5)
S.D $f4, 0($s1)
S.D $f5, 8($s1)
Receive($f6)
ADD.D $f7, $f6, $f5
ADD.D $f0, $f7, $f6
Send (2, $f7)
Send (2, $f0)
S.D. $f6, 16($s1)
S.D $f7, 24($s1)
S.D $f0 32($s1)
Receive($f2)
S.D $f2 40($s1)
ADDIU $s1, $s1, 48
BNE $s1,$s2,loop
ADD.D $f4,$f2,$f0
ADD.D $f5,$f4.$f2
ADD.D $f7, $f6, $f5
 S.D $f4, 0($s1)
S.D $f5, 8($s1)
S.D $f6, 16($s1)
```

Node 2

```
ADDIU $s3, $s0, $zero
loop:
Receive ($f8)
Receive ($f9)
ADD.D $f10, $f9, $f8
Send(1, $f10) Receive ($f8)
Receive ($f9)
ADD.D $f10, $f9, $f8
Send(1, $f10)
Receive ($f8)
Receive ($f9)
ADD.D $f10, $f9, $f8
Send(1, $f10)
Receive ($f8)
Receive ($f9)
ADD.D $f10, $f9, $f8
Send(1, $f10)
ADDIU $s3, $s3, 1
BNE $s3, 83, loop
```

The loop will take *1463 cycles* , the unrolled loop runs more optimally with the given instruction latency.

6.4.4 [10] <§6.2> The latency of the interconnect network plays a large role in the efficiency of message passing systems. How fast does the interconnect need to be in order to obtain any speedup from using the distributed system described in Exercise 6.4.3?

Answer)

The message passing process in the following way

send(x, y) where "y" sends a message to x, which takes one cycle to do it.

receive() it waits for the value to be sent which takes ten cycles,

Therefore, the receive instruction made have more stalling in the execution. Plus you can only execute the received instruction once the message is received. As a result, the time it takes by the receive to execute should be the same time for the sent instruction.

The loop network should respond in a single cycle to obtain a speedup.

Exercise 6.7

Consider the following portions of two different programs running at the same time on four processors in a symmetric multicore processor (SMP). Assume that before this code is run, both x and y are 0.

Core 1: x = 2;
Core 2: y = 2;
Core 3: w = x + y + 1;
Core 4: z = x + y;

6.7.1 [10] <§6.5> What are all the possible resulting values of w, x, y, and z? For each possible outcome, explain how we might arrive at those values. You will need to examine all possible interleavings of instructions.

Case 1: (Core order of execution: 3→2→1→4)
Core 3 runs and w = 1
Core 2 runs and y = 2
Core 1 runs and x = 2
Core 4 runs and z = 4
Case 2: (Core order of execution: 1→2→3→4)
Core 1 runs and x = 2
Core 2 runs and y = 2
Core 3 runs and w = 5
Core 4 runs and z = 4
Case 3: (Core order of execution: 4→3→2→1)
Core 4 runs and z = 0
Core 3 runs and w = 1
Core 2 runs and y = 2
Core 1 runs and x = 2
Case 4: (Core order of execution: 2→3→1→4)
Core 2 runs and y = 2
Core 3 runs and w = 3
Core 1 runs and x = 2
Core 4 runs and z = 4
Case 5: (Core order of execution: 3→4→2→1)
Core 3 runs and w = 1
Core 4 runs and z = 0
Core 2 runs and y = 2
Core 1 runs and x = 2
Case 6: (Core order of execution: 1→4→3→2)
Core 1 runs and x = 2
Core 4 runs and z = 2
Core 3 runs and w = 3
Core 2 runs and y = 2
Case 7: (Core order of execution: 3→4→1→2)
Core 3 runs and w = 1

Core 4 runs and z = 0
Core 1 runs and x = 2
Core 2 runs and y = 2
Case 8: (Core order of execution: 2→4→1→3)
Core 2 runs and y = 2
Core 4 runs and z = 2
Core 1 runs and x = 2
Core 3 runs and w = 5
Case 9: (Core order of execution: 2→4→3→1)
Core 2 runs and y = 2
Core 4 runs and z = 2
Core 3 runs and w = 3
Core 1 runs and x = 2
Case 10: (Core order of execution: 3→2→4→1)
Core 3 runs and w = 0
Core 2 runs and y = 2
Core 4 runs and z = 2
Core 1 runs and x = 2

6.7.2 [5] <§6.5> How could you make the execution more deterministic so that only one set of values is possible?

To make the execution more deterministic so that only one set of values are possible, you need to make one specific order of the cycle decided. Then the result should be ambiguous so that the operations that may be inconsistent can be executed atomically so that it will result in the single values for X, Y, Z.

Exercise 6.20

Assume a quad-core computer system can process database queries at a steady state rate of requests per second. Also assume that each transaction takes, on average, a fixed amount of time to process. The following table shows pairs of transaction latency and processing rate.

For each of the pairs in the table, answer the following questions:

6.20.1 [10] <§6.11> On average, how many requests are being processed at any given instant?

Average Transaction Latency	Maximum Transaction Processing Rate	Average Number of Requests
1 ms	5000/sec	1.25
2 ms	5000/sec	2.5
1 ms	10,000/sec	2.5
2 ms	10,000/sec	5

6.20.2 [10] <§6.11> If move to an 8-core system, ideally, what will happen to the system throughput (i.e., how many queries/second will the computer process)?

If the computer is moved to an 8-core system, ideally it would double the maximum transaction rate by doubling the number of cores.

6.20.3 [10] <§6.11> Discuss why we rarely obtain this kind of speedup by simply increasing the number of cores.

This kind of speedup is rarely obtained because of the memory contention on the shared memory system.