ECSE 420 - Parallel Computing
Theoretical Assignment 3
Fall 2015
Daniel Macario - 260503662

Q1. Performance Issues & Network Delay (20%)

Considering a ring network topology of 16 nodes (or processor), with hop latency of 10 ns. If a message communicates through a node in the network to the farthest point, for 80 times, what is the overall network delay? Suggest all possible ways in which you can reduce the network delay.

For the given network, let's assume that the message is communicated from node one. Assuming duplex communication channels, the farthest node would be node nine. There a total of 8 hops in this path.

$$h = Number\ of\ Hops\ Traversed = 8$$
$$f = Number\ of\ Messages = 80$$
$$t_h = link + switch\ latency\ per\ hop = 10\ ns$$

$$Network\ Delay = h \times f \times t_h$$
$$Network\ Delay = 8 \times 80 \times 10\ ns = 6400\ ns$$

In order to reduce the network delay, one could alter either f or h. To reduce f we can diminish inter-process communication or increase the message size to reduce the load on the network. We can reduce h by mapping processes to processors so that the topology of inter-process communication matches the physical topology of the network. Assuming we can alter the system, t_h can be reduced by shortening the links between the nodes, which translates to a smaller propagation time of the messages.

"Manycore" processors (multicore processors with tens or hundreds of cores) can be arranged in many different topologies. Compare these two topologies by filling the table below (writing +/- as advantages and disadvantages).

	Ring Network	Bus Network
Cache Coherency Complexity	-	+
Latency	-	+
Bandwidth	+	-
Wiring Cost	+	-

Q2. Cache Memory & Bus-Based Shared Memory Multiprocessor (10%)

Consider a bus-based shared memory multiprocessor system with write-through caches. It is constructed using 1.6 GHz processors, and a bus with a peak bandwidth of 50 Mega fetches/s. The caches are designed to support a hit rate of 90%. Only 15 % of program execution time is related to Read and Write commands. Considering each Read/Write instruction, in average, takes 2 clock cycles, What is the maximum number of processors that can be supported by this system?

First we must consider the cache hit rate in the computation. We can see that (1 - hit rate) x ProcessorSpeed determines the actual rate of bus requests from each processor. Therefore, the bus will be saturated when the inequality shown below is no longer true:

$$\frac{(Num\ Of\ Processors \times (1\ -\ Cache\ Hit\ Rate) \times Processor\ Frequency \times Time\ Spent\ in\ I/O)}{2} \leq Bus\ Bandwidth$$

$$\frac{(Num\ Of\ Processors \times (1\ -\ 0.9) \times 1.6{\times}10^{9} \times 0.15)}{2} \leq Bus\ Bandwidth$$

$$Num\ Of\ Processors \leq 4.167$$

Therefore, the maximum number of processors that the system can support is 4.

Q3. Memory Consistency (10%)

Suppose there are two process-P1 and process-P2 working on the same counter instance. Explain all possible outcomes from u and v for the following code.

int A,B,u,v = 0;

```
P1() {                      P2() {
        [A] = 1; //x1               [B] = 1 //x2
        u = [B]; //y1               v = [A] //y2
}                           }
```

The possible execution paths are demonstrated in the table below. The first possible outcome results from the sequence x1, y1, x2, y2; P1 executes both of its statements, and then P2 follows. The second possible outcome comes from the sequence x1, x2, y1, y2; a zig-zag pattern starting from P1. The third possible out come comes from the sequence x1, x2, y2, y1; also a zig-zag pattern but with P2 executing both statements in a sequence. The fourth outcome comes from the sequence x2, x2, x1, x2; P2 executes both of its statements followed by P1 executing both of its statement. The fifth possible

execution path comes from the sequence x2, x1, y1, y2; again, this is a zig-zag pattern starting from P2, but where P1 executes both of its statements in a sequence. The final outcome comes from the sequence x2, x1, y2, y1; a zig-zag pattern starting from P2.

Note that although there are six different execution paths, there are only three different combinations of values for u and v.

Outcome	Path	A	B	u	v
1	x1 -> y1 -> x2 -> y2	1	1	0	1
2	x1 -> x2 -> y1 -> y2	1	1	1	1
3	x1 -> x2 -> y2 -> y1	1	1	1	1
4	x2 -> y2 -> x1 -> y1	1	1	1	0
5	x2 -> x1 -> y1 -> y2	1	1	1	1
6	x2 -> x1 -> y2 -> y1	1	1	1	1

Q4. Illinois Protocol (35%)

Consider a bus-based shared memory multiprocessor system with 2 Processors, Illinois Protocol and write-back caches, initially empty, except P1 contains VarB. Both the processors access the shared variables A and B. For the following sequence of commands, write the state of all caches after each executed instruction. Considering the table below and in the last page, indicate bus transaction type and if there is memory access (by writing Yes/No) for each line. Calculate the overall data transfer for whole commands

Instruction	P0-A	P0-B	P1-A	P1-B	Memory Access	Bus Transaction
(initially)	I	I	I	E	-	-
P0 reads B	I	S	I	S	No	BusRd
P1 writes B	I	I	I	M	No	BusRdX
P0 reads A	E	I	I	M	Yes	BusRd
P0 writes A	M	I	I	M	No	-
P1 reads A	S	I	S	M	Yes	BusRd
P0 reads B	S	S	S	S	Yes	BusRd

BusRd = 4
BusRdX = 1

Overall data transfer = 4 * 64 + 1 * 64 = 320 (units are not specified in the table).

We now repeat the same exercise but using the MSI protocol. We assume that the initial state E of processors P1-B translates to state S in the MSI protocol.

Instruction	P0-A	P0-B	P1-A	P1-B	Memory Access	Bus Transaction
(initially)	I	I	I	S	-	-
P0 reads B	I	S	I	S	No	BusRd
P1 writes B	I	I	I	M	No	BusRdX
P0 reads A	S	I	I	M	Yes	BusRd
P0 writes A	M	I	I	M	No	BusRdX
P1 reads A	S	I	S	M	Yes	BusRd
P0 reads B	S	S	S	S	Yes	BusRd

BusRd = 4
BusRdX = 2

Overall data transfer = 4 * 64 + 2 * 64 = 384 (units not specified in the table)

Q5. Test & Set Lock (25%)

Consider a bus-based shared memory multiprocessor system with 2 Processors. Both the processors execute the same assembly code:

For the given sequence of commands, identify the critical section and provide the needed atomicity. Implement the proper lock-unlock operation using test&set instructions:

```
lock:   t&s $r0, 0(&lock)
        bnez $r0, lock

unlock: st $zero, 0(&lock)
```

Assume that both processors have the same $r2 initial values and different $r7 initial values stored in their local register file.

The table below demonstrates what the critical section of the code is; these lines are highlighted in red. The critical sections are wrapped using lock and unlock statements.

Line	P0	P1
0	addi $r1, #5, $r7	addi $r1, #5, $r7
1	sub $r1, $r1, $r2	sub $r1, $r1, $r2
new line	**lock1: t&s $r0, 0(&lock)**	**lock1: t&s $r0, 0(&lock)**
new line	**bnez $r0, lock1**	**bnez $r0, lock1**
2	ld $r5, 2($r2)	ld $r5, 2($r2)
new line	**st $zero, 0(&lock)**	**st $zero, 0(&lock)**
3	blt $r5, $r3, 8	blt $r5, $r3, 8
4	sub $r5, $r2, $r1	sub $r5, $r2, $r1
new line	**lock2: t&s $r0, 0(&lock)**	**lock2: t&s $r0, 0(&lock)**
new line	**bnez $r0, lock2**	**bnez $r0, lock2**
5	st $r5, 2($r2)	st $r5, 2($r2)
new line	**st $zero, 0(&lock)**	**st $zero, 0(&lock)**
6	sub $r5, $r6, $r5	sub $r5, $r6, $r5
7	addi $r6, #4, $r1	addi $r6, #4, $r1
8	addi $r6, #3, $r1	addi $r6, #3, $r1
9	addi $r6, #1, $r6	addi $r6, #1, $r6
new line	**lock3: t&s $r0, 0(&lock)**	**lock3: t&s $r0, 0(&lock)**
new line	**bnez $r0, lock3**	**bnez $r0, lock3**
10	ld $r4, 0($r1)	ld $r4, 0($r1)
new line	**st $zero, 0(&lock)**	**st $zero, 0(&lock)**
11	bgt $r4, $r2, 14	bgt $r4, $r2, 14
new line	**lock4: t&s $r0, 0(&lock)**	**lock4: t&s $r0, 0(&lock)**
new line	**bnez $r0, lock4**	**bnez $r0, lock4**
12	st $r4, 0($r1)	st $r4, 0($r1)
new line	**st $zero, 0(&lock)**	**st $zero, 0(&lock)**
13	addi $r5, #2, $r5	addi $r5, #2, $r5
14	addi $r5, #3, $r5	addi $r5, #3, $r5