Assignment 9 - Semil Jain (u1417989)

1) Consistency Models (40 points)

Assuming Before these threads start executing, A and B are both initialized to zero.
Let indicate each instruction with an alphabet

```
Thread 1
B = 100;       -> a
A = 150;       -> b

Thread 2
if ((B+A) > 80)       -> A
then A = A - B;       -> B
print A;              -> C
```

According to sequential consistency we assume: • Within a program, program order is preserved • Each instruction executes atomically • Instructions from different threads can be interleaved arbitrarily

Execution order can be 5C2 options = 10 orders

Output for each order is:

- **ABCab = 0**
- **ABaCb = 0**
- **ABabC = 150**
- **AaBCb = 0**
- **AaBbC = 150**
- **AabBC = 150**
- **aABCb = -100**
- **aABbC = 150**
- **aAbBC = 50**
- **abABC = 50**

In the above code is lock is acquired for the entire duration of execution by each thread and then released at the end of code. We have two possible scenarios. Either Thread 1 acquired the lock first and starts executing. In this case Thread 2 has to wait for the entire code execution to complete. Similarly vice versa.

Case 1: Thread 1 acquires lock first
abABC: 50

Case 2: Thread 2 acquires lock first
ABCab: 0

2) Consistency Models (20 points)
Relaxed Consistency Model:
This is a method to achieve relatively high performance along with maintaining the principles of a Sequential Consistent model. Here the hardware does some reorderings which are also needed to be made aware to the programmer in order to achieve this.

The code can consist of two types of region: A region where race condition exists, we need to be careful to maintain consistency and The region without race conditions where the hardware can apply its optimizations freely. For the hardware to be aware of such regions, It is the programmers job to mark this regions explicitly so that the Hardware can turn of any optimizations while executing a race condition code to preserve consistency.

The demarcations are done by the use of fence instructions. These instructions indicate that Everything upto that point needs to be finished first before executing the next set of instructions.

An example of this would be that before entering a critical section we can acquire locks and release them after the section ends. We can surround the acquire and release lock instructions with the fence instructions so that before entering the critical region all the instructions above it are completed. Similarly Before releasing the lock, all instructions upto that point have committed before moving forward. The hope is that the racy condition sections are less in the code and we can achieve relatively high performance while maintaining sequential consistency.

3) Synchronization (20 points)
LL-SC (Load linked & Stored Conditional):
LL-SC is an implementation of atomic read-modify-write with high flexibility. Here we make sure that the data is loaded and in the end the data is stored conditionally. In between these 2 instructions various computations can happen on the loaded data and not necessarily atomically (It is not ensured that all these operations happen sequentially without intervention). But while Storing the data in the end we do have a condition that whether the operations happened atomically and if they did we do the store or else start the process again. This happens through a notification system.

LL: While loading the data at the start in a register it maintains a table which stores the address from where the data was read. If someone is to make any changes at this location, this should be notified (coherence protocols) so that it can be tracked in the table.

SC: After all the computations after the Load are completed we move forward to start storing the data into the location. Before storing, the table is checked for any other instructions that might have modified it. If not, the data is stored, indicating “effective” atomicity.

Benefit of LL-SC over test-and-test-and-set:
-> Since Load and store happen independently, Even if the SC fails there is less coherence traffic on the bus as compared to test-and-test-and-set.
-> The hardware implementation here is relatively simpler as we don’t need to maintain atomicity for every instruction and just need to check the effective atomicity at the end.
-> Gives more flexibility to the Programmer as no immediate load store happens.

4) Networking (10 points)
West-First routing algorithm better than Dimension-Order routing and Fully Adaptive routing

Dimension order routing which is an example of Deterministic routing makes sure there are no deadlocks as we move along just the dimensions determined to reach our destination. The performance is not that great.

Fully Adaptive routing has great performance but can lead to deadlocks as it allows turns in all the directions.

West-First Algorithm lies between the above two methods and can be considered as a modified adaptive algorithm with the ability to turn into two dimensions removed out of the total of 8 turns. The two turns removed are such that the signal can never move to the west. So in order to reach the destination we make sure that the signal needs to go as west as possible (Slightly ahead from the destination) from the source, so that it does not require any west turns to reach its destination. This modified adaptive routing is a combination of both mentioned Algos above and are better than them as We make sure to have high performance and No deadlocks.

5) Networking (10 points)

There are two trade-offs to consider while choosing a network, namely, Performance and Cost.

Performance can be measured in terms of the bisection bandwidth and the average latency to traverse between two nodes.

Cost can be measured in terms of the total links in the network or total number of ports per router/switch. In other words how many neighbors do a node have that they may directly talk to.

Considering a 64 node network and given the choice between Torus and Hypercube:

Performance:
For a 2d torus having a 8x8 grid of nodes, On bisecting we get the bisection bandwidth to be 16 that is at any given point 16 signals can be transferred through the center.

For hypercube: Considering 64 nodes, each node can talk to 6 other nodes. The bisection bandwidth rises to 32.

Considering k-ary D-cube, Torus has 8^2 and Hypercube has 2^6.

On computing all the metrics like No. Of switches, links, pins per node, complexity, avg distance, diameter, bisection bandwidth using the formula in the slides for the above dimension and k. We can observe that as dimensions increase the complexity and cost affecting parameters increase also the performance metrics improve.

In conclusion, If We are solely needing performance Hypercube is the way to go, whereas if cost is a major factor Torus should be considered. It is all on the trade-offs that we want.