15-418 Project Proposal

Ashwin Adulla, aadulla

Title: Modeling Interconnection Networks

Summary: I plan on performing a study that analyzes how well various interconnection network topologies (i.e. mesh, ring, fat-tree), flow control algorithms (i.e. worhmhole, virtual channel), and routing algorithms (i.e. deterministic, adaptive) perform with regards to specific metrics (i.e. throughput, delay, contention). Such an analysis can provide insight into how certain combinations of topologies and algorithms affect the overall performance of message transmission and how that performance scales with the number of nodes on a massively parallel system.

Background: In various parallel processing applications where multiple processing and memory elements are required to communicate with each other, interconnections serve an important role in ensuring data is transmitted efficiently with minimal latency and maximal throughput. As these parallel systems scale up, such as by increasing the core count within a single processor or by adding more nodes to a supercomputer, properly designed interconnection networks are becoming more and more important to the overall performance of the system.

An interconnection network can be described by 3 attributes: a) topology, b) flow control algorithm, and c) routing algorithm.

A. The topology defines the physical hardware of the network and how all the nodes are hooked to the switches via links.

B. The flow control algorithm determines how a message is broken up into smaller chunks of data such as packets or flits, and then how those smaller chunks of data are transmitted along a path from source to destination. There are two design choices when it comes to flow control algorithms: circuit-switched and packet-switched. In circuit-switched, the full path from source to destination is reserved ahead of time and data is deterministically transmitted along that path. In packet-switched, no full path is predetermined, and instead the chunk of data dynamically determines where to move next during transmission.

C. The routing algorithm determines how the path from source to destination is generated either before the transmission in the case of circuit-switching or during the transmission in the case of packet-switching. Similar to flow control, there are two design choices when it comes to routing algorithms: non-adaptive/static and adaptive/dynamic.

With such a large design space, there is yet no clear answer as to what combination of topology, flow control, and routing works best given the target application. This project will explore this design space and will analyze under what conditions certain combinations perform better than others.

Challenge: This problem is challenging because of the large design space available and the accuracy of simulation required to quantitatively determine how well different interconnection networks perform. Given an interconnection network and a user defined environment that specifies the behavior of the messages being transmitted, there is no predefined formula available that will tell us what the average throughput, latency, or contention of the network is. The only way to determine these metrics is by simulating the network under the workload. The issue here is that the simulation must accurately imitate the workload where the only factors that determine the output metrics is the interaction between the messages and the network and not the overhead required to run the simulation.

To simulate multiple messages being transmitted as once as is often the case in a network, we could naively spawn multiple threads to each transmit their own message. However, if we were to then simply measure the wall clock time taken for a thread to move from its source node to destination node, then that measurement would be skewed by the memory accesses the thread had to make to travel from node to node. In addition, depending on how long certain memory accesses take, the thread may not accurately a message being transmitted and may skew the overall simulation. For instance, when transmitting message A and message B on a real network, their paths could intersect, causing them to contend for the same node at the same time. When simulating these 2 messages with thread A and thread B, we could imagine a case where thread A's memory accesses may take longer than thread B's memory accesses, thus causing thread B to move at a faster rate from node to node than thread A. As a result, we may never see thread A and thread B actually colliding with each other where their paths intersect.

A simple solution to fix this would be to synchronize all the threads after each move (i.e. all threads make a move in lockstep), however this is extremely costly and will significantly increase the overall simulation time (i.e. the communication to computation ratio will be very high). Thus, special care needs to be taken to ensure that the simulation remains accurate to the hardware without the additional overhead required to do so.

Resources: I will be using the GHC/Latedays clusters to run the simulations, however I would also like access to larger machines with a higher core count to simulate larger networks with more threads. Specifically, I was looking at using a few of the Amazon AWS EC2 instances with upwards of 32 virtual cores.

There are numerous references online regarding interconnection networks and their design/implementation. Here are a few of the following papers I have been reading up on:

https://etd.lib.metu.edu.tr/upload/12609136/index.pdf

http://www.diva-portal.org/smash/get/diva2:11666/FULLTEXT01.pdf

https://www.hindawi.com/journals/jece/2012/509465/

There are a few existing network simulators such as NOXIM and TOPAZ which I plan on starting with to understand how they go about running a simulation and gathering metrics during it. I cannot simply use these simulators for the purposes of this project however because they do not use parallel simulation and the built-in network topologies and routing algorithms are specific to certain application areas such as radio networks or wireless networks.

Platform Choice: I will be using C++ and mainly the POSIX threads library for my implementation. I plan on using POSIX threads because it will provide me with finer-grain control over each thread which will become necessary in the simulation. I will be running majority of my simulations on the GHC/Latedays machines because they can reliably support multiple threads, however for larger simulations, I will need to use larger machines such as Amazon AWS EC2 instances.

Goals and Deliverables:

A. Goals I plan to achieve
    a. A fully functioning network simulator that can support multiple topologies (3+), flow control (3+), and routing algorithms (3+) and can produce accurate performance metrics
    b. A completed analysis of 10+ popular/widely-used network designs in various communication environments that demonstrates which networks are better suited to which environments
    c. A model of how scaling up or down an existing network in terms of node count affects certain performance metrics
B. Goals I hope to achieve
    a. Add more topologies, flow control algorithms, and routing algorithms
    b. A visual heatmap to highlight certain areas of contention/dense traffic in the network as it is being simulated
    c. A GUI for the user to define his environment rather than in a config file

As a baseline, I will be implemented a lockstep simulator (i.e. barrier synchronization across threads after each move) to validate that my simulator can produce accurate results. I then will be optimizing beyond this baseline to decrease the overall simulation time while ensuring that the metrics it generates are within a certain degree of precision to the ones generated by the baseline.

For my final report, I will be presenting several graphs/tables that show the analysis of various network designs and I also hope to show a live demo of the simulator in action.

Schedule:

April 12 – April 18:

- Finalize base topologies and algorithms and begin working on implementing
- Finalize the criteria the user will define and then how that drives the simulation
- Begin working on simulator code

April 19 – April 25:

- Have a working prototype of the baseline simulator and perform basic analysis
- Begin working on optimizing the computation to communication ratio amongst threads
- Test simulator on GHC and Latedays machines (ensuring that the metrics output do not vary from machine to machine)
- Stress test baseline simulator on EC2 instances with large simulation size

April 26 – May 3:

- Compare optimized simulator to base simulator
- Perform tests on GHC/Latedays/EC2 instances and ensure validity of optimized simulator metrics to baseline metrics
- Compile findings and analysis into final report