EECS 578 - Project Outline

1. Project title

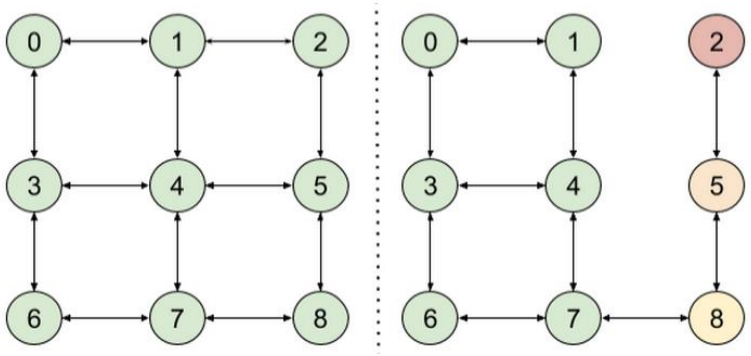
Efficient execution of MapReduce applications on irregular NoC topology

2. Team name

HAMM

3. Problem to be addressed

Hard faults in a network on chip (NoC) interconnect can cause links to break, creating an irregular network topology and potential bottlenecks. Since MapReduce applications create considerable network traffic they are sensitive to the network topology. Nodes communicating on congested links will experience communication delays causing them to finish later than well connected nodes. For example, in figure 1, each node in the regular topology (left) completes the mapping phase of MapReduce at approximately the same time. Conversely, nodes 2, 5, and 8 in the irregular topology (right), might finish much later than other nodes when given the same number of jobs. The overall execution time depends on the node that finishes last. As a result, the total execution time will be prolonged by the slow node.



Regular Topology Irregular Topology

Figure 1. Regular versus Irregular topology

4. Why does this problem matter?

In order to take advantage of the potential speedup of CMPs, parallel programming is necessary. Due to its simplicity, MapReduce is becoming an increasingly common parallel programming model for NoC based CMPs. However, continuously shrinking transistors are causing permanent faults in NoCs and creating a bottleneck in the execution of MapReduce applications.

5. Idea/solution to be investigated by the project

Nodes periodically collect information regarding the current topology and use it to determine their connectivity to the rest of the network. Based on their connectivity, the MapReduce framework determines the amount of loads for each node. The Phoenix++ framework allows opportunities to load balance during mapping and shuffling stages. During the mapping phase, the input data can be arbitrarily divided among nodes. During the shuffling phase, a hashing function can be configured to balance the amount of intermediate data destined for each nodes. Dynamically balancing loads this way should improve the overall execution time.

6. How do you plan to develop the project?

We will run two versions of Phoenix++ on Gem5 to generate network traces for each MapReduce applications. One version will be the unmodified Phoenix++, while the other will be modified to implement the load balancing mechanism described above. The unmodified version will generate network traces for the baseline model while the modified Phoenix++ generates network traces for the fault aware load balance model. We will then run the network traces on the faulty 2D meshes using Booksim.

7. How do you plan to evaluate the project's results?

We will compare the average network latency of the baseline model with the fault aware load balanced model using Booksim. Additionally, we will estimate the area overhead of our proposed model.

8. Timeline