Ryan Anderson

Beowulf Cluster Report

4/20/18

**Hardware**

My Beowulf Cluster consisted of 4 Raspberry Pi 3 Model B’s. These computers have a 1.2GHz 64-bit quad-core ARMv8 CPU, with 1 GB RAM. Each Pi was networked using a Cat5e Ethernet cable, with speeds up to 1000 Mbps. The networking was accomplished using a 5-port Ethernet switch from Netgear. DCHP software on my Windows 10 machine allowed the network to be routed through the Ethernet switch. Runtime comparisons were made between this Beowulf Cluster and my Windows 10 machine, which contains a 2.50GHz 64-bit quad-core Intel Core i7-6500U CPU, and 8GB of RAM.

**Software**

The 3 algorithms that I implemented were Heapsort, Mergesort, and traditional matrix multiplication. Each Pi in my Beowulf Cluster was running the Raspbian OS—a fork of Debian—with the MPI software suite installed. In each of my programs, the overall task was divided according to the number of workers in the cluster. The results were then reassembled into the final result. For instance, in matrix multiplication with four Pi’s (1 master and 3 workers), a square matrix of size 10 will be partitioned into 2 3x10 matrices, and 1 4x10 matrix. These partitions are sent to each worker where they will multiply their partitions with the entire second matrix and send the resultant data back to the master. In my two sorting algorithms using MPI, the arrays to be sorted are also partitioned before being sent to the worker nodes. Heapsort works by building a heap before the sort, so when it is partitioned, each node builds its own heap. This additional overhead likely contributes to the program’s slower run times. Bubblesort is also partitioned, but there is no additional overhead; the nodes merely start and stop sorting from a specified index. On my laptop, all three of my algorithms were executed using a single thread each. Finally, I wrote a basic shell script to execute each program 50 times. Every execution would output the elapsed time that the program took to execute the calculations section, and I copied all 50 times to an Excel spreadsheet where I averaged and graphed them.

**Results**

Figure 1.

The Java and C implementations were run on my laptop, and the MPI implementation was on the Beowulf Cluster. In Figure 1., the MPI implementation is considerably slower. I hypothesize that this is because my laptop’s processor is much more powerful than the Pi’s, and the added cores of the Beowulf Cluster still cannot match the higher speed and hardware optimization on my laptop.

Figure 2.

Heapsort is an extremely fast sorting algorithm, but it is slower on the cluster than on my laptop. I hypothesize that the slowdown is both for the same reason as the previous example, and because of the additional Heapsort-specific overhead (building the heap) mentioned earlier. The reduced speed of the Pi’s combined with the added overhead required to distribute tasks adds too much time to the execution. I also hypothesize that adding more nodes will eventually reduce the speed below my laptop’s run time.

Figure 3.

Bubblesort is the worst algorithm, according to everyone. Surprisingly, the Beowulf Cluster executed this sorting method much faster than my laptop could. I hypothesize that the nature of Bubblesort (swapping until sorted) fits better with the architecture of the Pi’s processor, or the distribution of MPI, and the fact there this is no additional overhead involved in each node’s sorting range is what caused the MPI implementation on the cluster to run faster than the C implementation on my laptop.

**Conclusion**

The Beowulf Cluster with MPI, by its nature, a fast and efficient way to compute. I hypothesize that my cluster, with only four nodes, does not have enough computing power to rival a powerful laptop like the one I used in this project. Most of the Beowulf Cluster that I researched for this project had 10 or more nodes. Beowulf Clusters are meant to simulate super computers while maintaining affordability, so a cluster with only four nodes, each with middling processing power, will not deliver the speeds required for more realistic model sizes.