Mark Randles
CS417
Lab 3
2004-03-31

The results of this lab where unique. The purpose of the lab was to explore a bit about OpenMP on a SMP machine, and to compare the results gained from the OMP program vs. the results gain from the MPI implementation of the same program. The actual coding of the program was pretty straight forward, and the results gained yielded some unique results.

The coding of the program, as mentioned above, was rather straight forward. Since OpenMP is a set of compiler directives and limited library routines, it is quite easy to parallelize code with. First a single processor application was created, to which the OMP code was added. The directions for the project were such that we weren't allowed to use OMPs native partitioning support, and had to partition the dataset by ourselves in user land code. This was easily done, and implemented well.

The OMP code was run for 5-15 parallel thread, incrementing by 2 processes. The parent process did not do any actual work, except to initialize the original arrays, and to print a bit of status information. The times for the execution of the code was determined by the UNIX time program. The directions for the project were for the project to report it's own time, but there was little difference between the two time, so the UNIX time output was used as it was the more accurate of the two.

Also run was the MPI implementation of the same problem, from a previous lab. Also created was a version of the program where the dataset partitioning was handled by OMP using the directives provided. The same data was collected for this program as well. The data from all runs was tabulated in a spreadsheet and graphed as speedup vs. processors, each on a separate chart.

Surprisingly the OMP programs were slower then the MPI version. Much of this can be contributed to the fact that in a MPI cluster the variables are operated on in that processors cache and they don't need to be synced across the threads. However in a OMP system each thread needs to sync its values continuously which generates more overhead. This could change if the OMP version of the code was optimized so more of the calculations were private to the threads so none of the processors cache syncing overhead would be incurred.