Nathan Zacharjak a1704695

Objective

Here is an analysis of various versions of a program based on "The N Body Problem". Basically, this program simulates the force of gravity on several particles at once, each particle exerting a force on the other at any given timestep. What we're interested in is how each of these program versions perform. There are 4 separate C programs representing the 4 different versions, the first is a "basic" version that has a simpler implementation but is expected to have some performance drawbacks. The other 3 versions are more interesting, they split the computation of the forces of the particles into 2 "phases" or parallel for loops, but these 2 phases are scheduled differently. The 2nd version has a block schedule for both phases, the 3rd version has a cyclic schedule for the first phase and a block schedule for the second phase, the 4th version has a cyclic schedule for both phases.

Results

The data below shows the runtime of each program in seconds, with the number of threads requested in the first column, and the program version in the first row. Each program was run 3 times, and the average of those times taken as the final runtime.

"./[program_name] [thread_number] 400 1000 0.01 100 g" was used to run each of the programs 3 times, for 1, 2, and 4 threads.

Parameters for each run:

400 – particles
1000 – timesteps
0.01 - seconds for each timestep (what was recommended for auto-generated initial particle conditions using the g parameter as per the program's comments)
100 - positions/velocities are output every 100th timestep
g - automatically generate initial states for each of the particles

Number of threads	reduce_1 (Basic)	reduce_2 (Reduced Block Scheduling)	reduce_3 (Reduced 1st phase Cyclic)	reduce_4 (Reduced All Cyclic)
1	2.88	1.68	1.67	1.67
2	1.50	1.26	0.882	0.883
4	0.818	0.759	0.481	0.495

Discussion

The first, basic version of the program is clearly the worst performing for all amounts of threads. Compared to the best performing reduced version of the program, it is up to twice as slow. Even for 1 thread, it is 72% slower than any version of the reduced program. As for the different reduced versions of the program, they perform identically with 1 thread, but both versions with cyclic scheduling perform better after that with 2 or 4 threads. With 2 threads, the cyclic versions perform 42% better than the block partitioning version. It isn't until there are 4 threads that the all-cyclic version starts to become slower than the 1st phase only cyclic version. It is only 3% faster, but the performance between these 2 versions was identical up until this point, which should only become more pronounced as more threads are used to run the program.

Nathan Zacharjak a1704695

Conclusion

To conclude, the 3rd version of the program, with a cyclic partitioning for the first phase of calculating but not the second, seems to perform the fastest. It is only slightly better than the all-cyclic version, which may be used instead for similar to identical performance for low amounts of threads. However, due to their implementation, all of the reduced versions of the program come with the trade-off that they use more memory than the basic versions of the program. With a large enough particle count, the memory use may become impractical, and the basic version of the program would have to be used instead.