Gary Tang
ICS 432
5/3/15

Results

The goal of the assignment was to increase the performance of Mike Gowanlock's Galactic Habitable Zone software. The software was written in C and C++. Any method that would increase the software's performance was allowed in this assignment. These methods included sequential optimization, compiler optimization, parallelization, and more. The host machine for performance testing was a Apple Mac Mini 2012 with 2.5 GHz Intel Core i5 processor, 4 GB RAM and OS X Yosemite 10.10.3. Utilizing Virtual Box 4.3.16, Ubuntu 14.04 was set up as the testing environment. This environment was given 2 CPU cores and 2 GB RAM.

The program was tested using input 0.18. All versions of the program used the same input. Performance was measure by execution time. Each version of the program was executed 10 times and the average execution time was taken. Gprof was used to profile all program versions. The original program, v0, took 96.1s to run. Profiling v0 revealed the program spent 73.1% of the time executing the running_total() method. Version 1, v1, used local variables to simplify the equation in the running_total's for loop and other equations that executed running_total() within itself. Performance dropped 25.4%. v2 tried to optimize the main() 's for loops. Local variables were used to store the array values that were checked in the for loops. As a result, performance dropped 10.1%. v3 attempted to parallelize the running_total() with openmp. The resulting program took over 10 minutes to execute and was not tested further. v31 attempted to parallelize the main(). The segmentation fault errors resulted and were not solved. v4 tried to optimize process_sn(), the method with the second highest execution time percentage at 13.1%. process_sn() spent the most time calling get_star_range_upper(), get_star_range_upper_cell_left(), and get_star_range_upper_right(). These methods were called repeatedly inside for loops. Local variables were used to store these values. This reduced the amount of times the methods were called. As a result, execution time was reduced to 31.6s and performance increased 67.2% from v0. v5 attempted to optimize generate_birth_date(), the other method process_sn() spent time calling. This method was located in the SFH.cpp file. generate_birth_date() spent the most time calling all SFR_Gyr() methods and called each method twice. Local variables were used to store values returned from the SFR_Gyr() methods. The two for loops were broken down and all 14 iterations were written out. As a result, performance increased 6.1% from v4. v6 attempted to parallelize process_sn(). The race conditions were not solved and the resulting output was not correct. v5 included v2's optimizations. v7 is v5 without the v2 optimizations. The removal resulted in 4.7% performance increase from v5 and execution time reduced to 21.2s. v8 is v4 without the v2 optimizations. As a result, performance increased 5.9% from v4 and execution time reduced to 25.9s. v7 and v8 had the greatest improvement. These two versions were then compiled using optimizations; o1, o2, o3, and ofast. v7 with compiling

optimizations did not reduce execution time and remained at 21.2s. v84, v8 with the ofast optimization, reduced execution time to 25.6s.

v7 had the greatest performance increase of 78.9% from v0. The execution time was reduced to 21.2s from 96.1s. v7 with compiling optimizations had an increase of 0.1% from v7. The performance enhancements described above were from sequential optimizations. Further performance improvements can be seeked with code parallelization. The code parallelization attempts resulted in incorrect output or slower performance. Enhancing the program to use distributed memory coding can also be explored.

Performance Data

	Total Execution Time (s)	Avg. Execution Time (s)	Improvement % from v0
v0	960.9	96.1	N/A
v1	1205.2	120.5	-25.4%
v2	1058.3	105.8	-10.1%
v3	NA see report		
v31	NA see report		
v4	315.6	31.6	67.2%
v5	257.3	25.7	73.2%
v6	Incorrect output see report		
v7	212.4	21.2	77.9%
v71	212.1	21.2	77.9%
v72	212.3	21.2	77.9%
v73	213.3	21.3	77.8%
v74	211.6	21.2	78.0%
v8	258.6	25.9	73.1%
v81	258.1	25.8	73.1%
v82	257.9	25.8	73.2%
v83	259.7	26.0	73.0%
v84	256.4	25.6	73.3%