Gerald Abut

ICS 432 Final Project

4/28/2015

All test done on a VM running 32bit Ubuntu and 2 GB of ram through command line. Each version was executed 10 times to get the average. Although it was acceptable to have a slight difference in output due to the floating-point calculations, I made sure that the results stayed consistent. That is, that the total stellar mass was always equal to 3833250.74163964.

1. **Original:**
   1. Code Modification: None
   2. Reasoning: Benchmarking to compare with my versions of the code.
   3. Issues: N/A
   4. Results:
      1. 104.98 seconds
      2. 103.71 seconds
      3. 109.89 seconds
      4. 104.15 seconds
      5. 104.53 seconds
      6. 105.77 seconds
      7. 104.11 seconds
      8. 104.88 seconds
      9. 107.57 seconds
      10. 107.57 seconds

**Average: 105.72**

* 1. Improvements: N/A

1. **Version 0:**
   1. Code Modification:
      1. Code clean up using indentations for clarity.
      2. Set the timer.
      3. Use pragma in main using 1 thread and declaring a whole block of code as critical.
      4. Pragma was used in the inner most loop.
   2. Reasoning:
      1. This is mainly to test the effects of using pragma on the code. I’ve tested it on 2 parallel threads and above but found out that using only one thread produced the fastest results. Therefore I continued to use 1 thread for the other versions.
      2. Using pragma on the inner most loop condenses the work done by the thread.
   3. Issues: Because pragma was declared in the innermost loop, it may produce race conditions.
   4. Results:
      1. 90.47 seconds
      2. 95.14 seconds
      3. 94.07 seconds
      4. 91.80 seconds
      5. 91.80 seconds
      6. 93.71 seconds
      7. 90.75 seconds
      8. 90.54 seconds
      9. 94.10 seconds
      10. 95.51 seconds

**Average: 92.79**

* 1. Improvements: Yes by about a little over 12 seconds

1. **Version 1:**
   1. Code Modification:
      1. Modified the function process\_SN() to use pragma.
   2. Reasoning:
      1. Learning about how helper threads can improve performance, I decided to use them on a call function.
   3. Issues: None
   4. Results:
      1. 44.90 seconds
      2. 43.75 seconds
      3. 44.00 seconds
      4. 42.32 seconds
      5. 44.02 seconds
      6. 44.49 seconds
      7. 42.88 seconds
      8. 43.01 seconds
      9. 42.83 seconds
      10. 43.24 seconds

**Average: 43.54**

* 1. Improvements: Yes by a little over a factor of 2

1. **Version 2:**
   1. Code Modification:
      * Added a pragma to the following functions:
        1. process\_SN\_end\_subsection()
        2. copy\_into\_star1()
        3. copy\_into\_star2()
        4. copy\_star2\_into\_star1()
        5. copy\_temp\_into\_star2()
        6. create\_columns()
   2. Reasoning:
      * Same reasoning as version 1
   3. Issues: Did not improve performance and there was a slight slowdown in performance. I suspect shared variable issues.
   4. Results:
      1. 41.88 seconds
      2. 44.05 seconds
      3. 45.36 seconds
      4. 43.76 seconds
      5. 44.78 seconds
      6. 43.97 seconds
      7. 44.52 seconds
      8. 43.70 seconds
      9. 43.53 seconds
      10. 42.67 seconds

**Average: 43. 82**

* 1. Improvements: No

1. **Version 3:**
   1. Code Modification:
      * Removing variables and using literals instead
      * Less block of code for critical sections
   2. Reasoning:
      * Removing variable initialization is one less step to do
      * This also removes false dependencies
      * No need for a thread to be blocking in non-critical areas
   3. Issues:

* Some variables were kept in tacked to maintain correctness
* I’ve also tried to consolidate the same calculations into one variable but it did not produce the results that I wanted and it actually slowed the execution.
* I also attempted to use #define constants but that too also slowed down execution.
  1. Results:
     1. 39.87 seconds
     2. 40.21 seconds
     3. 40.41 seconds
     4. 39.51 seconds
     5. 40.11 seconds
     6. 41.10 seconds
     7. 39.95 seconds
     8. 42.15 seconds
     9. 40.54 seconds
     10. 41.74 seconds

**Average: 36.51**

* 1. Improvements: Yes by a little over 7 seconds and almost by a factor of 3

1. **Version 4:**
   1. Code Modification:
      * Removed unnecessary critical sections
      * Comment out unused variables
      * Replaced calculations that used the #define constants with literal numbers
      * Unravel for-loops into many for-loops that have individual processes rather than a single iteration doing many things. The functions are listed below:
        1. copy\_into\_star1()
        2. copy\_into\_star2()
        3. copy\_star2\_into\_star1()
   2. Reasoning:
      * There was only 1 thread in use at a time so a pragma critical section was not necessary.
      * Less work within a single iteration of a for-loop helps speed things up.
   3. Issues:
      * I’ve also tried using pragma sections but that did not help in performance
      * copy\_temp\_into\_star2() function was kept intake because when I unraveled it to many for loops, it would crash due to a memory out of bounds error. This was due to incorrect placement of pragma in main
      * Only a slight improvement in performance
   4. Results:
      1. 33.74 seconds
      2. 34.42 seconds
      3. 35.55 seconds
      4. 35.03 seconds
      5. 34.84 seconds
      6. 34.29 seconds
      7. 34.52 seconds
      8. 34.61 seconds
      9. 33.84 seconds
      10. 34.54 seconds

**Average: 34.54**

* 1. Improvements: Yes by barely a factor of 3

1. **Version 5:**
   1. Code Modification:
      * Extended the techniques previously used above to parts of the code that I missed from version 4
      * Used memcpy to replace some for loop
      * Removed data shared arrays that are dynamically allocated and just share the index instead
      * Loop unrolling in these cpp files
        1. mersenne.cpp
        2. SFH.cpp
      * Replaceed for-loops in these functions to use memcpy() instead:
        1. copy\_into\_star1()
        2. copy\_into\_star2()
        3. copy\_star2\_into\_star1()
      * Used more literals to replace #define constants
      * Put pragma in outermost for-loop and adjusted the shared variables.
   2. Reasoning:
      1. Using gprof, I noticed that the randomization function were the slowest to execute, therefore, I used Loop unrolling in the mersenne.cpp and SFH.cpp to optimize execution time for randomization.
      2. Replacing a for-loop with memcpy() takes less process time than looping over an entire array an element at a time because it only takes a single process rather than n processes.
      3. Adjusted pragma and shared variables to prevent race conditions which fixed the issue with version 4’s copy\_temp\_into\_star2()
   3. Issues:
      1. Attempting to unravel the for-loop that has if statements in SFN.cpp gave inaccurate results so I did not unravel that part of the code.
   4. Results:
      1. 21.87 seconds
      2. 22.12 seconds
      3. 22.15 seconds
      4. 22.74 seconds
      5. 21.89 seconds
      6. 22.44 seconds
      7. 22.50 seconds
      8. 22.30 seconds
      9. 22.46 seconds
      10. 22.26 seconds

**Average: 22.27**

* 1. Improvements: Yes by a factor slightly greater than 4

Summary

Parallelizing the code helped in the overall speed up however I’m still baffled at why a single thread would have better performance than multiple threads since in parallelization, all a thread needs to do is wait for the other threads to finish and doesn’t bump each other along their execution (at least within the designated pragma). I’m assuming this is why a powerful single core machine is preferred over multicore. Another technique that I used that also baffled me was that substituting variables for calculations sped execution a bit. I would have thought that more calculations would be more costly but I’d say that it has to do with something within the assembly level or is compiler specific. Unraveling for-loops also helped optimize the code because it didn’t have to do costly jumps in the assembly level. What I do understand well is that the less processes to do the work the better the execution time. A great example of this was when I replaced the for-loops with the memcpy function. There was a significant improvement in execution time because memcpy took away having to iterate through each individual element in the array. If I had more time, a pthread version of the code would have been interesting.