Partners: Isaac Plunkett

Briana Liosatos

Instructor: Dr.Homer

CSc 422 Project 1 Report

MAKE TO PDF

Introduction

The problem that is going to be solved is the NBody problem. Bodies are placed in a space and given their own x and y velocities. Each body exerts a force (gravity) on the other bodies in the space. When one body gets close to another body, the forces between the two causes the bodies to rapidly approach each other, resulting in a collision. When a collision happens, the two bodies change the direction of their velocity vectors and move away from one another at a faster speed.

Programs

There are two versions of the NBody problem. A sequential and a parallel version using threads. Both were written in Java using the Eclipse software.

Each version takes the same command line arguments. The first argument is how many workers or threads are to be used. For the sequential this is set to 0, and 1 for the parallel. The second argument is the number of bodies that are to be used for the simulation. For both versions, number of bodies is set to 2. The third parameter is the size of the bodies (their diameter), which is set to 7 for both versions. The fourth parameter is the number of timesteps, set to 1000. Timesteps is how many times the loop for changing the position and velocity of the bodies is called. The fifth parameter is if the GUI should be displayed, the default is set to true. The sixth parameter is to set the size of the bodies to random sizes. This is set to false to start and the bodies have a size restraint of being a minimum of 10 and a maximum of 50. The last parameter is to make the threads sleep (used so the user can see the updates in the GUI). If sleep is not set the program runs so fast that the user won’t be able to see a window or the collisions before the simulation is over. Sleep is initially set to false.

In the parallel version, the barrier that is used is a dissemination barrier. A 2-D array of semaphores is created to stop threads at the correct stage of the program.

Verification

We used print statements and the GUI to help determine if the program was working correctly. JUnit tests were also written to test for the movement of the bodies for one timestep (pass of the loop that performs the physics on the bodies) and the collision between two bodies.

By also using the GUI we were able to see that not all of the collisions are elastic (after the bodies hit they bounce away from each other). On special occasions, when three or more bodies collide they tend to cluster together until another body collides with the cluster, when this happens the bodies tend to break up. We were unable to find a solution to this problem.

Timing Experiments/ Other Experiments

We used the ‘harbor’ cs computer to do our timing experiments.

Figure 1 contains the data for the sequential and the parallel programs tested with 50 bodies, body size of 1o, and the number of time steps is 10,000 with no GUI. Each test was run 4 times and the average was calculated. Graph 1 contains the graphical version of these results.

As part of our other experiments we tested the parallel version with 1 to 32 threads.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name of test.  GUI not tested | Test 1 (ms) | Test 2  (ms) | Test 3 (ms) | Test 4  (ms) | Average  (ms) |
| sequential | 213 | 213 | 214 | 213 | 213.25 |
| parallel 1 | 218 | 210 | 216 | 215 | 214.75 |
| parallel 2 | 212 | 211 | 205 | 204 | 208 |
| parallel 3 | 261 | 265 | 261 | 261 | 262 |
| parallel 4 | 255 | 244 | 254 | 248 | 250.25 |
| parallel 5 | 387 | 379 | 377 | 385 | 382 |
| parallel 6 | 380 | 369 | 389 | 371 | 377.25 |
| parallel 7 | 400 | 373 | 374 | 382 | 382.25 |
| parallel 8 | 390 | 382 | 409 | 410 | 397.75 |
| parallel 9 | 543 | 588 | 556 | 592 | 569.75 |
| parallel 10 | 587 | 572 | 619 | 622 | 600 |
| parallel 11 | 686 | 741 | 660 | 735 | 705.5 |
| parallel 12 | 758 | 771 | 765 | 719 | 753.25 |
| parallel 13 | 772 | 766 | 790 | 839 | 791.75 |
| parallel 14 | 849 | 834 | 790 | 854 | 831.75 |
| parallel 15 | 846 | 850 | 812 | 841 | 837.25 |
| parallel 16 | 891 | 978 | 878 | 914 | 915.25 |
| parallel 17 | 1342 | 1228 | 1284 | 1292 | 1286.5 |
| parallel 18 | 1323 | 1318 | 1310 | 1336 | 1321.75 |
| parallel 19 | 1326 | 1323 | 1395 | 1360 | 1351 |
| parallel 20 | 1474 | 1459 | 1446 | 1425 | 1451 |
| parallel 21 | 1543 | 1561 | 1514 | 1528 | 1536.5 |
| parallel 22 | 1563 | 1567 | 1593 | 1627 | 1587.5 |
| parallel 23 | 1531 | 1603 | 1720 | 1573 | 1606.75 |
| parallel 24 | 1787 | 1818 | 1735 | 1823 | 1790.75 |
| parallel 25 | 1778 | 1888 | 1775 | 1770 | 1802.75 |
| parallel 26 | 2026 | 1914 | 2028 | 1951 | 1979.75 |
| parallel 27 | 1961 | 1962 | 1930 | 1938 | 1947.75 |
| parallel 28 | 2023 | 2069 | 2015 | 2012 | 2029.75 |
| parallel 29 | 2069 | 2141 | 1998 | 2028 | 2059 |
| parallel 30 | 2110 | 2302 | 2133 | 2100 | 2161.25 |
| parallel 31 | 2087 | 2106 | 2045 | 2125 | 2090.75 |
| parallel 32 | 2280 | 2280 | 2257 | 2242 | 2264.75 |

Image 1. Results for the sequential program and 1-32 threads for the parallel program.

Graph 1. Graph of the results found in Image 1.

The question was wanted to know is how the overall time changes when more threads are added to the parallel version of the NBody problem. From our results, we gathered that the more threads were used to move 50 bodies, the longer the program took to execute. We believe this is because some threads are waiting in the barrier for a majority of the time because they have less bodies to calculate the new velocities and positions. Unlike the sequential version, where the program just loops over a body array to calculate the physics aspects without having to wait for other processes to finish their calculations before moving on to the next step.

For the other extra experiment, a GUI was created. The fifth parameter for the program turns the GUI on or off. The same exact code was run for if there was a GUI or not, to insure correct time comparisons. Since we were unable to time the GUI performance on the lab computers remotely, we did a timing test with the GUI on a local machine. The software that was used to run the tests locally was Eclipse. Our results for the GUI timing for the sequential version are shown in Image 2. The sequential program was run 3 times (with and without GUI) to produce the average time the sequential version took to complete run the simulation.

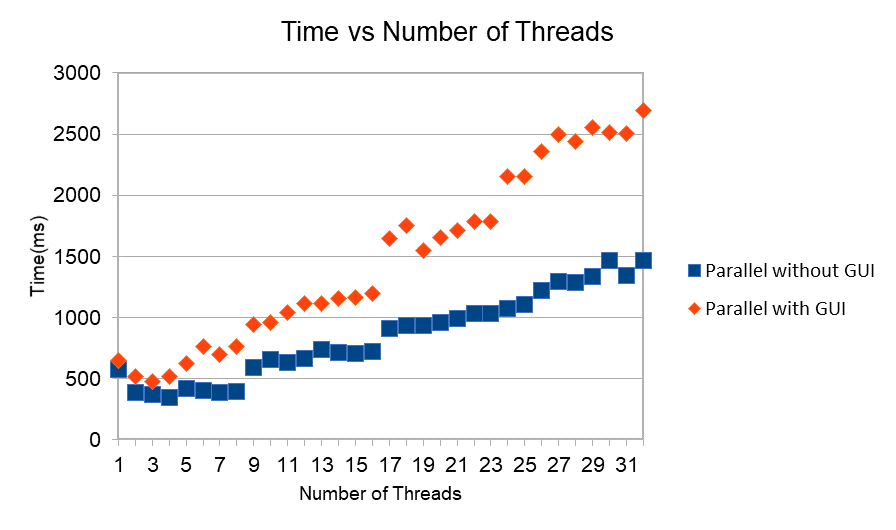
|  |  |  |
| --- | --- | --- |
| sequential NO GUI 1 |  | sequential GUI 1 |
| 570 milliseconds |  | 628 milliseconds |
|  |  |  |
| sequential NO GUI 2 |  | sequential GUI 2 |
| 566 milliseconds |  | 629 milliseconds |
|  |  |  |
| sequential NO GUI 3 |  | sequential GUI 3 |
| 557 milliseconds |  | 626 milliseconds |
|  |  |  |
| sequential NO GUI Average |  | sequential GUI Average |
| 564.3333333 milliseconds |  | 627.6666667 milliseconds |
|  |  |  |

Image 2. The timing of the sequential program with and without the GUI.

The parallel program was run with 1 to 32 threads. This timing was also run three times and the average was taken. For this test, the GUI was turned off on the local machine to give a more accurate comparison between the times between the GUI being on and off, since both versions were run on the same machine. Image 3 is the results of each test run and the averages of the GUI being turned off and on. Graph 2 is the graphical representation of these results.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Thread number and No GUI | Time (millisec) | Time (millisec) | Time (millisec) | Average Time (millisec) |  | Thread number and GUI | Time (millisec) | Time (millisec) | Time (millisec) | Average Time (millisec) |
| parallel 1 no GUI | 570 | 580 | 566 | 572 |  | parallel 1 with GUI | 645 | 643 | 643 | 643.666667 |
| parallel 2 no GUI | 386 | 388 | 381 | 385 |  | parallel 2 with GUI | 505 | 495 | 530 | 510 |
| parallel 3 no GUI | 390 | 356 | 354 | 366.6666667 |  | parallel 3 with GUI | 476 | 477 | 465 | 472.666667 |
| parallel 4 no GUI | 361 | 324 | 347 | 344 |  | parallel 4 with GUI | 508 | 510 | 510 | 509.333333 |
| parallel 5 no GUI | 417 | 426 | 394 | 412.3333333 |  | parallel 5 with GUI | 613 | 598 | 659 | 623.333333 |
| parallel 6 no GUI | 381 | 423 | 390 | 398 |  | parallel 6 with GUI | 665 | 890 | 710 | 755 |
| parallel 7 no GUI | 381 | 391 | 381 | 384.3333333 |  | parallel 7 with GUI | 704 | 692 | 691 | 695.666667 |
| parallel 8 no GUI | 379 | 409 | 379 | 389 |  | parallel 8 with GUI | 779 | 740 | 746 | 755 |
| parallel 9 no GUI | 603 | 609 | 553 | 588.3333333 |  | parallel 9 with GUI | 924 | 996 | 896 | 938.666667 |
| parallel 10 no GUI | 775 | 583 | 598 | 652 |  | parallel 10 with GUI | 955 | 933 | 967 | 951.666667 |
| parallel 11 no GUI | 650 | 616 | 611 | 625.6666667 |  | parallel 11 with GUI | 1089 | 1018 | 997 | 1034.66667 |
| parallel 12 no GUI | 732 | 620 | 622 | 658 |  | parallel 12 with GUI | 1186 | 1099 | 1047 | 1110.66667 |
| parallel 13 no GUI | 815 | 690 | 688 | 731 |  | parallel 13 with GUI | 1158 | 1128 | 1050 | 1112 |
| parallel 14 no GUI | 695 | 750 | 687 | 710.6666667 |  | parallel 14 with GUI | 1263 | 1104 | 1097 | 1154.66667 |
| parallel 15 no GUI | 677 | 679 | 758 | 704.6666667 |  | parallel 15 with GUI | 1225 | 1112 | 1141 | 1159.33333 |
| parallel 16 no GUI | 764 | 691 | 707 | 720.6666667 |  | parallel 16 with GUI | 1211 | 1137 | 1225 | 1191 |
| parallel 17 no GUI | 900 | 908 | 908 | 905.3333333 |  | parallel 17 with GUI | 1629 | 1621 | 1674 | 1641.33333 |
| parallel 18 no GUI | 943 | 978 | 860 | 927 |  | parallel 18 with GUI | 1695 | 1742 | 1809 | 1748.66667 |
| parallel 19 no GUI | 919 | 929 | 943 | 930.3333333 |  | parallel 19 with GUI | 1619 | 1503 | 1513 | 1545 |
| parallel 20 no GUI | 971 | 942 | 963 | 958.6666667 |  | parallel 20 with GUI | 1609 | 1581 | 1765 | 1651.66667 |
| parallel 21 no GUI | 1005 | 988 | 976 | 989.6666667 |  | parallel 21 with GUI | 1933 | 1626 | 1576 | 1711.66667 |
| parallel 22 no GUI | 1042 | 1035 | 1017 | 1031.333333 |  | parallel 22 with GUI | 2045 | 1674 | 1639 | 1786 |
| parallel 23 no GUI | 1013 | 1040 | 1025 | 1026 |  | parallel 23 with GUI | 1778 | 1648 | 1935 | 1787 |
| parallel 24 no GUI | 1056 | 1045 | 1115 | 1072 |  | parallel 24 with GUI | 2220 | 2101 | 2138 | 2153 |
| parallel 25 no GUI | 1107 | 1096 | 1104 | 1102.333333 |  | parallel 25 with GUI | 2362 | 1952 | 2131 | 2148.33333 |
| parallel 26 no GUI | 1219 | 1202 | 1234 | 1218.333333 |  | parallel 26 with GUI | 2381 | 2432 | 2256 | 2356.33333 |
| parallel 27 no GUI | 1240 | 1391 | 1244 | 1291.666667 |  | parallel 27 with GUI | 2331 | 2679 | 2471 | 2493.66667 |
| parallel 28 no GUI | 1250 | 1371 | 1231 | 1284 |  | parallel 28 with GUI | 2495 | 2374 | 2446 | 2438.33333 |
| parallel 29 no GUI | 1299 | 1268 | 1428 | 1331.666667 |  | parallel 29 with GUI | 2384 | 2709 | 2556 | 2549.66667 |
| parallel 30 no GUI | 1468 | 1459 | 1474 | 1467 |  | parallel 30 with GUI | 2653 | 2495 | 2401 | 2516.33333 |
| parallel 31 no GUI | 1440 | 1322 | 1258 | 1340 |  | parallel 31 with GUI | 2386 | 2583 | 2551 | 2506.66667 |
| parallel 32 no GUI | 1538 | 1536 | 1325 | 1466.333333 |  | parallel 32 with GUI | 2737 | 2679 | 2657 | 2691 |
|  | Run 1 | Run 2 | Run 3 | average |  |  | Run 1 | Run 2 | Run 3 | average |
|  |  |  |  | Parallel without GUI |  |  |  |  |  | Parallel with GUI |

Image 3. Results of the GUI being on and off



Graph 2. Graphical representation of the results (shown in Image 3).

Our question for the GUI is if providing a graphical interface causes a significant difference between the two programs and the number of threads used. The sequential version was faster with creating a GUI (627.667 milliseconds) compared to the parallel version with only one thread creating a GUI (643.667 milliseconds). We believe this is because the thread version must go through a barrier, even though there are no other threads present. As for when there is in increase in the number of threads used to execute the problem, the time goes up almost linearly. This is because the more threads executing the program means the threads spend more time waiting in the barrier for the other threads to finish a specific stage in the code. The increase in time is also due to the computer having to redraw the images, every time a timestep is completed.

Conclusion

Overall, the sequential program is the fastest version of the NBody program. The barrier in the parallel version causes the thread version to be slower because of the waiting for a signal. The more threads that are used to execute the program, increases the overall time. We learned that even with more threads, which should be faster, the sequential version is the fastest, even when a GUI is being created.

describe what you have learned from this project