Jonathon Davis

Scott Source

Project 1

Program Overview:

The program can be created by running the following command

*Make all*

The console version of the program is run when calling the main method in Body.class

*Java Body numThreads numBodies radius mass maxIterations*

numThreads (int): is the number of threads that will be run in the program

numBodies (int): is the number of bodies that will be created for the simulation

radius (int) or (min-max): is the radius of each object, can be a number or range,

mass (int) or (min-max): is the mass of each object, can be a number or a range

maxIterations (long): The number of calculations the simulation will preform

The GUI version of the program can be run when calling the main method in GUI.class

*The gui will pop up a window asking for the arguments, if the window is closed a new*

*simulation can be ran by going to Settings->New Simulation*

Program Summery:

The n-bodies problem, which is a problem where bodies are created that resemble planets, has two major versions, the parallel and sequential versions of the program. The two versions were created to measure the performance between sequentially executed code, and executing code in parallel. The data showed that the parallel version was on average faster than the sequential. This was the expected result as the work is divided among multiple processors.

The timing tests were conducted on the Cambridge machine with the number of threads being the only difference; for two tests one being 500 bodies (figure 1) and another 1000 bodies (figure 2). The timing starts after all the bodies are created, and ends when the number of iterations inputted is complete. This means that each time the program was run, it was given similar start conditions. The data showed that running the program in parallel is faster than running the program sequentially. However the increase in performance was only around 30% when the number of threads was increased to a total of 2. Then the programs performance began to deteriorate as more threads were created. It is important to note that due to the programs gui, the program technically ran on more than the inputted number of threads, even in the sequential version. This is because the GUI, Simulation, and WorkerThreads each ran in their own thread. For example when the program starts, the main thread will create a thread to handle the GUI, which will create a thread to manage the simulation, which will then create a WorkerThread to actually compute the simulation. Because of these extra threads, the performance of the simulation begins to drop off, even when the number of inputted threads is less than the number of processors. Each test was done three times to find a consistency (see Figure 3). The average of the three tests were used in the graphs of this report.

Figure 1

Figure 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iterations | Size (radius) | Mass | Threads | DeltaT | TimeEnd (Total Time) milliseconds | Bodies |
| 100 | 100 | 100 | 1 | 0.5 | 15703 | 1000 |
| 100 | 100 | 100 | 1 | 0.5 | 11080 | 1000 |
| 100 | 100 | 100 | 1 | 0.5 | 13351 | 1000 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **13378** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 1 | 0.5 | 5896 | 500 |
| 100 | 100 | 100 | 1 | 0.5 | 5985 | 500 |
| 100 | 100 | 100 | 1 | 0.5 | 6420 | 500 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **6100.333333** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 2 | 0.5 | 11971 | 1000 |
| 100 | 100 | 100 | 2 | 0.5 | 10727 | 1000 |
| 100 | 100 | 100 | 2 | 0.5 | 10582 | 1000 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **11093.33333** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 2 | 0.5 | 5623 | 500 |
| 100 | 100 | 100 | 2 | 0.5 | 4586 | 500 |
| 100 | 100 | 100 | 2 | 0.5 | 5516 | 500 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **5241.666667** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 3 | 0.5 | 12319 | 1000 |
| 100 | 100 | 100 | 3 | 0.5 | 13559 | 1000 |
| 100 | 100 | 100 | 3 | 0.5 | 12105 | 1000 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **12661** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 3 | 0.5 | 6157 | 500 |
| 100 | 100 | 100 | 3 | 0.5 | 5223 | 500 |
| 100 | 100 | 100 | 3 | 0.5 | 5629 | 500 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **5669.666667** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 4 | 0.5 | 12999 | 1000 |
| 100 | 100 | 100 | 4 | 0.5 | 12128 | 1000 |
| 100 | 100 | 100 | 4 | 0.5 | 13174 | 1000 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **12767** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 4 | 0.5 | 4836 | 500 |
| 100 | 100 | 100 | 4 | 0.5 | 5222 | 500 |
| 100 | 100 | 100 | 4 | 0.5 | 5130 | 500 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **5062.666667** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 8 | 0.5 | 12744 | 1000 |
| 100 | 100 | 100 | 8 | 0.5 | 11560 | 1000 |
| 100 | 100 | 100 | 8 | 0.5 | 12498 | 1000 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **12267.33333** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 8 | 0.5 | 5365 | 500 |
| 100 | 100 | 100 | 8 | 0.5 | 5584 | 500 |
| 100 | 100 | 100 | 8 | 0.5 | 4812 | 500 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **5253.666667** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 16 | 0.5 | 11950 | 1000 |
| 100 | 100 | 100 | 16 | 0.5 | 13477 | 1000 |
| 100 | 100 | 100 | 16 | 0.5 | 12129 | 1000 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **12518.66667** |  |
|  |  |  |  |  |  |  |
| 100 | 100 | 100 | 16 | 0.5 | 6299 | 500 |
| 100 | 100 | 100 | 16 | 0.5 | 5351 | 500 |
| 100 | 100 | 100 | 16 | 0.5 | 4819 | 500 |
|  |  |  |  |  |  |  |
|  |  |  |  | Average | **5489.666667** |  |
|  |  |  |  |  |  |  |

Figure 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | Average |  |
| 1 | 500 | 73755 | **75599.7** |  |
| 1 | 500 | 74293 |  |  |
| 1 | 500 | 78751 |  |  |
|  |  |  |  |  |
| 2 | 500 | 91827 | **91728** |  |
| 2 | 500 | 92994 |  |  |
| 2 | 500 | 90363 |  |  |
|  |  |  |  |  |
| 3 | 500 | 93279 | **94174** |  |
| 3 | 500 | 95361 |  |  |
| 3 | 500 | 93882 |  |  |

Figure 4