Daniel Eisenberg
CSc 422
Parallel Project: N-Bodies Problem

This report details sequential and parallel solutions to the n-bodies problem in two dimensions in a closed system with elastic collisions. The overhead of a dissemination barrier and the time spent in barriers during execution will be evaluated. False sharing will also be investigated.

The sequential solution takes advantage of the symmetry of gravitational forces in its calculations. The solution is adaptive with respect to time granularity in an effort to avoid undetected collisions. (Undetected collisions can still occur when objects are moving sufficiently fast and are sufficiently close together that the ratio of distance to speed does not evaluate to a finite number.) Specifically, the smallest value of the ratio of the distance between the edges of two objects to the sum of their speeds is used as the next time granularity provided the objects are moving closer to one another. The initial time increment argument serves as the maximum time granularity; although increasing this value could potentially provide a significant increase in efficiency at a cost of precision. The solution executes until the total elapsed simulated time exceeds the product of the number of steps argument with the initial time increment argument.

The parallel solution is modeled closely on the sequential solution. The symmetry of gravitational forces is again exploited using the fact that the total force exerted upon an object is the sum of the individual forces; this eliminates redundant calculations at the cost of additional memory while simultaneously avoiding an unnecessary critical section. There is one critical section owing to the adaptive time granularity. When one worker has reason to update the time step, it must do so atomically. Additionally, timing barriers requires a critical section to ensure that only the first worker to enter the barrier tracks the time spent in the barrier.

There are a number of test cases used to evaluate the correctness of the solutions. These are preserved in the Constants.java file.

There are two momentum transfer tests in one dimension for which objects behave as expected. The first involves one body moving toward another body which is at rest. When these collide the second body will begin travelling toward another body at rest. After execution, the first two bodies are at rest (disregarding gravitational forces) while the third has the velocity of the first, as expected. The second involves two objects moving toward two different objects at rest which are located between them. The moving objects collide with these objects which then collide with one another before finally colliding with the first two objects.

There is one simple collision of two objects with radius 1 in two dimensions:

$$\begin{aligned}(x_1, y_1) &= (1, 3);\\ v_1 &= (-1, -3)\\ (x_2, y_2) &= (-1, -3);\\ v_2 &= (1, 1)\end{aligned}$$

After 1 second, we expect these objects to be located at $(0, 0)$ and $(0, -2)$, respectively. At this point they will collide.

Following the collision, we expect the objects to have the following velocities:

$$\begin{aligned}x_2 - x_1 &= 0\\ y_2 - y_1 &= -2\end{aligned}$$

$$\begin{aligned}v_{1f} &= \left(\frac{0 + 0 + (-1)(-2)^2 + 0}{0 + (-2)^2}, \frac{0 + (1)(-2)^2 + 0 + 0}{4}\right)\\ &= \left(\frac{-4}{4}, \frac{4}{4}\right)\\ &= (-1, 1)\\ v_{2f} &= \left(\frac{0 + 0 + (1)(-2)^2 + 0}{4}, \frac{0 + (-3)(-2)^2 + 0 + 0}{4}\right)\\ &= \left(\frac{4}{4}, \frac{-12}{4}\right)\\ &= (1, -3)\end{aligned}$$

The solution bears out these results (if gravitational effects are disregarded).

One test case involves four objects which are symmetrically equidistant from a common central point and at rest. We would expect the components of the gravitational forces which pull a given object away from the central point to cancel one another and for each object to uniformly move toward the central point. This is how the solution behaves.

There is a falling bodies test case which attempts to approximate a small object thrown directly up from the surface of the Earth. This case behaves mostly as expected (the small object reverses its velocity extremely quickly), although a collision will not be detected until after the small object moves significantly below the Earth's surface, and the collision algorithm will not evaluate a collision correctly because the objects have significantly different masses.

There is one aspect of the parallel solution which could be viewed as a bug: The final collision detection in which the objects are moved over a very small time interval involves resetting the force values of the objects, which in turn causes approximations of the objects' next positions to be nondeterministic. However, the effect of the gravitational forces will rarely affect objects' positions to any noticeable degree where objects have the same mass (the collision algorithm assumes objects share a mass and this will generally be the case). Meanwhile, rectifying this detail would require a fourth barrier at each iteration.

Timing tests were performed first for the sequential solution, then for each of the following with 1 to 16 workers:

(1) the parallel solution with padded force vectors as well as reversed barrier semaphore declaration, so each worker has one row containing a semaphore for each stage, to investigate false sharing.

(2) a version of the parallel solution with only force vector padding.

(3) a version of the parallel solution without any of these adjustments.

The parameters of the test were 300 objects at random positions with random velocities (all random values were in the range [-100, 100]) generated with the seed 99, over a period of 1000 seconds at a maximum time granularity of 1 second. The tests were performed on harvill in the Gould-Simpson 930 lab.

The timing results were as follows:

sequential
36 seconds 149 milliseconds
36 seconds 138 milliseconds
36 seconds 184 milliseconds

workers	padding w/ barrier change (1)	padding (2)	no padding (3)
1	37 seconds 500 milliseconds	37 seconds 317 milliseconds	37 seconds 247 milliseconds
	37 seconds 438 milliseconds	37 seconds 284 milliseconds	37 seconds 229 milliseconds
	37 seconds 309 milliseconds	37 seconds 295 milliseconds	37 seconds 241 milliseconds
2	35 seconds 313 milliseconds	34 seconds 814 milliseconds	35 seconds 629 milliseconds
	31 seconds 880 milliseconds	22 seconds 820 milliseconds	35 seconds 869 milliseconds
	34 seconds 641 milliseconds	33 seconds 320 milliseconds	24 seconds 102 milliseconds
3	26 seconds 196 milliseconds	29 seconds 914 milliseconds	23 seconds 248 milliseconds
	26 seconds 477 milliseconds	27 seconds 678 milliseconds	24 seconds 472 milliseconds
	25 seconds 575 milliseconds	26 seconds 780 milliseconds	26 seconds 731 milliseconds
4	22 seconds 78 milliseconds	22 seconds 147 milliseconds	20 seconds 181 milliseconds
	22 seconds 806 milliseconds	23 seconds 892 milliseconds	21 seconds 752 milliseconds
	23 seconds 348 milliseconds	21 seconds 608 milliseconds	22 seconds 246 milliseconds
5	20 seconds 450 milliseconds	20 seconds 694 milliseconds	19 seconds 90 milliseconds
	20 seconds 516 milliseconds	20 seconds 503 milliseconds	18 seconds 969 milliseconds
	20 seconds 581 milliseconds	20 seconds 507 milliseconds	19 seconds 92 milliseconds
6	18 seconds 17 milliseconds	17 seconds 750 milliseconds	16 seconds 397 milliseconds
	17 seconds 609 milliseconds	17 seconds 652 milliseconds	16 seconds 420 milliseconds
	17 seconds 616 milliseconds	17 seconds 607 milliseconds	16 seconds 372 milliseconds
7	15 seconds 921 milliseconds	15 seconds 735 milliseconds	14 seconds 613 milliseconds
	15 seconds 806 milliseconds	15 seconds 831 milliseconds	14 seconds 632 milliseconds
	15 seconds 873 milliseconds	15 seconds 767 milliseconds	14 seconds 819 milliseconds
8	14 seconds 163 milliseconds	14 seconds 95 milliseconds	13 seconds 226 milliseconds
	14 seconds 242 milliseconds	14 seconds 44 milliseconds	13 seconds 225 milliseconds
	14 seconds 88 milliseconds	14 seconds 64 milliseconds	13 seconds 229 milliseconds
9	18 seconds 776 milliseconds	18 seconds 753 milliseconds	17 seconds 950 milliseconds
	19 seconds 91 milliseconds	19 seconds 101 milliseconds	18 seconds 102 milliseconds
	18 seconds 834 milliseconds	18 seconds 762 milliseconds	17 seconds 773 milliseconds
10	20 seconds 34 milliseconds	19 seconds 469 milliseconds	18 seconds 882 milliseconds
	19 seconds 480 milliseconds	19 seconds 388 milliseconds	18 seconds 888 milliseconds
	19 seconds 507 milliseconds	19 seconds 417 milliseconds	18 seconds 883 milliseconds
11	20 seconds 346 milliseconds	20 seconds 482 milliseconds	20 seconds 113 milliseconds
	20 seconds 406 milliseconds	20 seconds 376 milliseconds	20 seconds 51 milliseconds
	20 seconds 431 milliseconds	20 seconds 299 milliseconds	21 seconds 495 milliseconds
12	20 seconds 559 milliseconds	20 seconds 622 milliseconds	20 seconds 247 milliseconds
	20 seconds 499 milliseconds	20 seconds 491 milliseconds	20 seconds 180 milliseconds
	20 seconds 527 milliseconds	20 seconds 504 milliseconds	20 seconds 266 milliseconds
13	21 seconds 232 milliseconds	21 seconds 172 milliseconds	20 seconds 785 milliseconds
	21 seconds 239 milliseconds	21 seconds 231 milliseconds	20 seconds 836 milliseconds
	21 seconds 189 milliseconds	21 seconds 256 milliseconds	20 seconds 867 milliseconds
14	21 seconds 321 milliseconds	21 seconds 242 milliseconds	20 seconds 977 milliseconds
	21 seconds 437 milliseconds	21 seconds 621 milliseconds	20 seconds 979 milliseconds
	22 seconds 811 milliseconds	21 seconds 520 milliseconds	21 seconds 57 milliseconds
15	22 seconds 148 milliseconds	22 seconds 203 milliseconds	21 seconds 237 milliseconds
	22 seconds 22 milliseconds	22 seconds 72 milliseconds	21 seconds 291 milliseconds
	22 seconds 129 milliseconds	22 seconds 124 milliseconds	21 seconds 127 milliseconds
16	21 seconds 230 milliseconds	21 seconds 329 milliseconds	20 seconds 691 milliseconds
	21 seconds 154 milliseconds	21 seconds 218 milliseconds	20 seconds 698 milliseconds
	21 seconds 192 milliseconds	21 seconds 292 milliseconds	20 seconds 672 milliseconds

The increase in performance as the number of workers increases seems to be approximately linear for this test case. Between 3 and 8 workers, adding a worker scales the time of the calculation to about 90%.

These results demonstrate that any false sharing is negligible; rather, attempts to prevent false sharing could instead be interfering with cache optimizations by the JIT compiler, which would explain the slightly slower performance of these variations.

We now turn to an investigation of barrier efficiency. The averaged results for each of the parallel solutions are provided here for each number of workers up to 8; beyond 8 workers, barrier performance is nearly uniform among the 3 solutions and degrades from 56% of computation time at 9 workers to 86% of computation time at 16 workers.

Time spent in barriers (in milliseconds)

solution	# of workers							
	1	2	3	4	5	6	7	8
(1)	11	757	6712	6377	8544	7373	7474	2175
(2)	8	768	9046	6385	8473	6751	7234	2015
(3)	10	656	10326	8408	9028	5845	6899	1997

Average time spent in each barrier (in milliseconds)

solution	# of workers							
	1	2	3	4	5	6	7	8
(1)	0.000246	0.0164	0.145	0.138	0.185	0.160	0.162	0.0471
(2)	0.000180	0.0167	0.196	0.138	0.183	0.146	0.157	0.0437
(3)	0.000210	0.0142	0.224	0.182	0.196	0.127	0.150	0.0433

Percent of computation time spent in barriers

solution	# of workers							
	1	2	3	4	5	6	7	8
(1)	0.0303%	2.24%	25.7%	28.0%	41.7%	41.6%	47.1%	15.3%
(2)	0.0224%	2.69%	32.1%	28.1%	41.2%	38.2%	45.9%	14.3%
(3)	0.0260%	2.07%	41.4%	39.3%	47.4%	35.7%	47.0%	15.1%

These data show that the overhead of the dissemination barrier is extremely small for 2 or 8 workers. This should not be surprising as a dissemination barrier functions for any number of processes, but redundant information is shared when the number of processes is not a power of 2. What is somewhat surprising is that the dissemination barrier does not demonstrate minimal overhead for 4 processes.

Also worth noting is that solutions (1) and (2) spend similar amounts of time in barriers despite having different organization of barrier semaphores. We might expect (2) and (3) to spend the same amount of time in barriers as they are executing identical barrier code. However, the additional computation time resulting from the padded force values in solutions (1) and (2) could explain this: If one process almost always arrives at the barrier first because of the padded force values, this would explain these similarities inside the barrier, provided that the difference in barrier execution time is essentially independent of the organization of the barrier semaphores in memory.

Interpreting these results suggests that a fourth solution which does not pad force values but which does reverse the indexing of barrier semaphores may be worth investigating. In fact, such a solution (ParallelCollisionsBarrier.java) does show very slight improvement over solution (3) in most cases, as the following data demonstrates.

Computation time

# of workers							
1	2	3	4	5	6	7	8
37s 221ms	20s 738ms	25s 694ms	21s 980ms	19s 24ms	16s 363ms	14s 846ms	13s 151ms
37s 205ms	34s 153ms	27s 571ms	15s 983ms	18s 894ms	16s 215ms	14s 568ms	13s 164ms
37s 214ms	20s 642ms	26s 115ms	19s 745ms	18s 784ms	16s 230ms	14s 596ms	13s 114ms

Time spent in barriers (in milliseconds)

# of workers							
1	2	3	4	5	6	7	8
9	586	11479	5176	8799	5856	6899	1961

Average time spent in each barrier (in milliseconds)

# of workers							
1	2	3	4	5	6	7	8
0.000166	0.0127	0.249	0.112	0.191	0.127	0.149	0.0425

Percent of computation time spent in barriers

# of workers							
1	2	3	4	5	6	7	8
0.0251%	2.39%	43.3%	25.9%	46.6%	36.0%	47.0%	14.9%

These timing results also demonstrate that this final solution can occasionally run much faster than the other solutions while not lagging behind in other executions. This suggests that reversing the indexing of the barrier semaphores is a recommended strategy.

We have established that padding or reorganizing variables to avoid false sharing may or may not improve performance; that the overhead of a dissemination barrier is minimal for 2 or 8 workers; and that performance gains seen by increasing the number of workers are consistant.

This project has undergone many changes since the original sequential solution was written. Each test case uncovered new bugs. Most such bugs were in the collision detection algorithm. The initial version of the parallel solution did not correctly reset the next time increment to the maximum time increment and, as a result, was extremely slow as compared to the sequential version.

The development process of the parallel solutions brought into focus nondeterministic execution orders; time/space trade-offs; and the importance of minimizing or eliminating critical sections, as well as the conditions necessary for critical sections to be eliminated.