Parallelization of an Axially Symmetric Steady Flow Program*

Norm Beekwilder
Andrew S. Grimshaw

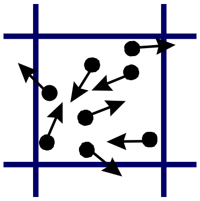
Technical Report CS-98-10

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
Email: { norm | grimshaw }@virginia.edu

Abstract: Description of the modifications made to an axially symmetric steady flow program to convert it from a sequential code to a parallel code. Some of the difficulties encountered and performance results are discussed.

Problem Description: The code that was parallelized was a particle in cell code from the Material Science Department that simulated an axially symmetric steady state flow of a gas. The purpose of this code is to see how and where particles will collide with a surface at the end of the gas jet. This simulates part of the chip manufacturing process in an attempt to increase the yield of a production line. Using the sequential code even medium sized problems where taking in excess of a week to complete and it was estimated that the largest desired problem size could take as long as a year to complete. Furthermore the largest simulations would most likely have been too large for the available memory resources of the target machine. The sequential code uses the following basic algorithm:

- All the particles are moved checking only to see if they have collided with a surface or have left the simulation area.
- Particles are indexed by cell.
- Pairs of particles in each cell are selected to collide.



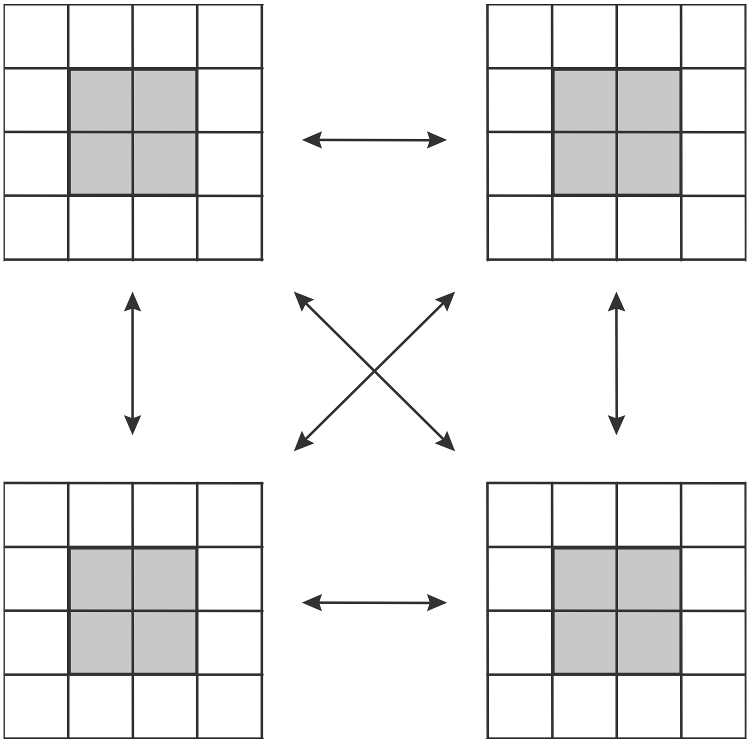
Close up of a cell in the simulation.

The simulations that were used for testing and scaling results would run for 500 to 7000 time steps and contain between 1.5 and 2.6 million particles depending on the size of the problem. Although we only varied the number of cells in the simulation other factors that would be varied are temperature, pressure, and jet velocity. All of these factors will impact the number of time steps required for the simulation to complete. The data structures used by the sequential code were one-dimensional arrays to track the position and velocity of the particles. The sequential code also used an index array so that cells would know where to find the particles that they contained.

Approach: The data structures were decomposed into a two dimensional grid of patches with several cells per patch. Dividing a regular grid into several sub-grids is a standard way to decompose particle in cell codes where most operations are local to a cell. Each patch uses the same basic algorithm, which is as follows.

* This work was partially funded by NSF grant CDA-9724552

- The particles are moved as in the initial algorithm with an additional check to see if they have left the patch.
- All particles that have left the patch are sent to the appropriate neighbor patch.
- The remaining particles are index and collisions for the interior cells of the patch are calculated.
- Particles form neighboring cells are received and indexed.
- Collisions are calculated for the boundary cells of the patch.



Above is a sample 2x2 decomposition, where each 4x4 block is a patch that runs on its own CPU. The squares in the patch are the cells of the original simulation. Shaded cells are interior cells, which calculate collisions while particles are transferred between patches. The arrows between patches represent communications that must occur each time step. Being able to overlap the communications with the computation of interior cells is an important optimization. A typical message send will be measured in milliseconds while machine instructions are measured in nanoseconds thus hundreds of thousands of instructions can be executed in the time it takes for a single message to be sent. The more computation that can overlap with communication the better the application will scale to large numbers of hosts. The Message Passing Interface (MPI) [1] was used for communications between patches. MPI is a standard that allows processes to communicate with each other in a portable manner. There are several implementations of the standard available spanning most platforms.

Another modification to the code was the random number generator. It was replaced with a Fibonacci lag random number generator that was slightly faster than the original random number generator and was able to produce an independent random sequence of numbers for each patch. The majority of random number generators are in fact completely deterministic, it is important to ensure that each processor does not end up using the same seed. Otherwise each process in the simulation would generate the same number at the same time. However even using a linear congruential generator with different seeds is not always good enough since that will merely cause each process to start at different points in the same sequence of numbers. Although this would not be a problem if only a few numbers were required it could cause problems when a large numbers of random numbers are require. This would occur when two process overlap in the sequence of numbers that are used, thus two processes would be generating the same numbers but perhaps ten thousand numbers out of step. Such a problem would be very hard to detect and yet could invalidate the results of the simulation. The advantage of a Fibonacci lag generator is that each process can be seeded with its own sequence of random numbers thus ensuring that such problems can not occur.

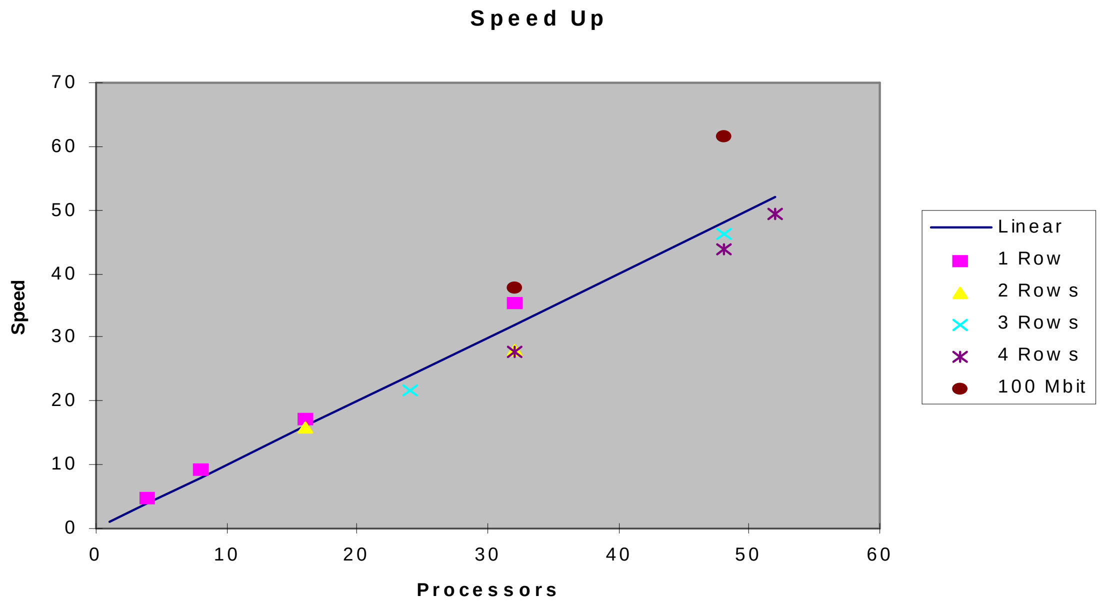
Problems Encountered: The initial approach had a slightly different ordering of events. In this approach we first moved all the particles in the boundary cells and then sent any particles that moved to neighboring patches. Then we moved the particles in the interior cells. Finally we received particles from the neighboring patches. This minimized the number of changes that had to be made to the original source. However, it was possible for the application to add an unmoved particle to a boundary cell and then move it to a neighboring patch after the boundary move. Since there was no way to prevent this and no way to guarantee that a particle once added would not leave the patch, I had to abandon this approach.

There were also some problems with the sample parameters for the simulation. The rules that had been specified for the simulation to obtain good results were as follows. Particles should not move more than 1 cell per time step. This is in order to ensure good physics in the simulation. Each cell should contain at least 10 particles. In order to ensure that there would be a sufficient number of particles in the cell for the collision step. In all the sample configurations the first constraint was not met for both the sequential and parallel cases. Many particles were moving more than one cell at a time some were moving as far as 2.5 cells per time step. As a result I had to drastically reduce the amount of simulated time per time step in both versions of the code. The second constraint was also not met, for both sequential and parallel versions of the code, in all but the smallest of problem sizes. This was due to the simulation using a geometric progression for the grid size along the Y-axis. As a result the initial rows of cells were nearly empty while the final rows would have hundreds of particles. This problem was the worst in the largest problem size I encountered where nearly 65% of the grid did not satisfy this constraint. I investigated fixing this problem by adjusting the number of real particles each simulated particle represented. However in the two largest problem sizes the machine did not have sufficient swap space to run the sequential program. In any case the results would have been useless for timing purposes so this problem was not corrected in the simulations.

Performance: The simulation yielded excellent scaling results especially with a one-dimensional decomposition of the problem. Timing results were gathered by running a simulation 5 times and then comparing the best time to the best sequential time. All binaries were compiled using g77 with O2 optimizations turned on and were run on 533MHz DEC Alpha with 128M of main memory and a 2M level 3 cache. The table below represents the results of the timing runs. The column headers denote how the problem was decomposed and the total number of processors is shown in parentheses. Note that in the two cases marked with an asterisk the simulation was run on a 100Mbit network while the other cases had only a 10Mbit network. The highlighted cell in each column is the best time and was used to calculate the speed up. One will notice that in all of the one-dimensional decompositions there was super-linear speed up this is due to a poor cache hit rate in the collision step. During the collision step two particles are selected from a cell to collide; however, their location in the array of particles is effectively random. As the problem gets decomposed into smaller and smaller pieces the percentage of the total number of particles that will fit in the cache goes up and thus so dose the cache hit rate for the parallel version.

	Sequential	4x1 (4)	8x1 (8)	16x1 (16)	8x2 (16)	8x3 (24)	32x1 (32)
1	41:41:31	9:05:36	5:17:37	2:28:07	2:44:05	2:05:52	1:19:21
2	41:53:14	8:59:48	4:48:18	2:39:07	2:37:46	1:56:17	1:18:19
3	42:18:11	9:08:24	4:36:57	2:30:56	2:46:18	1:56:01	1:19:34
4	42:13:46	9:05:00	4:28:00	2:34:42	2:45:11	1:55:36	1:10:48
5	42:16:26	9:12:03	4:34:36	2:26:39	2:53:48	1:56:59	1:17:56
Average	42:04:38	9:06:10	4:45:06	2:31:54	2:45:26	1:58:09	1:17:12
Speed Up	1.0	4.6	9.3	17.0	15.8	21.6	35.3
	16x2 (32)	8x4 (32)	16x3 (48)	12x4 (48)	13x4 (52)	32x1 (32)*	48x1 (48)*
1	1:30:21	1:43:17	0:55:33	0:58:10	0:53:06	1:06:26	0:49:10
2	1:29:35	1:32:08	0:59:45	0:57:59	0:54:32	1:09:37	0:46:43
3	1:38:09	1:31:28	0:56:53	1:01:00	0:50:35	1:09:32	0:40:41
4	1:43:18	1:34:21	0:54:28	0:57:12	0:54:36	1:09:39	0:40:37
5	1:29:53	1:29:46	0:58:20	0:59:48	0:52:08	1:09:30	0:44:31
Average	1:34:15	1:34:12	0:57:00	0:58:50	0:52:59	1:08:57	0:44:20
Speed UP	27.9	27.8	45.9	43.7	49.4	37.6	61.5

The reason that the two dimensional decompositions did not perform as well as the one dimensional cases is because the Y-axis uses a geometric progression which resulted in a load imbalance between the rows of patches. A secondary factor is that the two dimensional case requires that 4 times as many messages are sent as in the one dimensional case.



Observations: The hardest part of this project was not deciding how to parallelize the application; the hardest part was stepping through the source code and determining what the various variables were and if they needed to be converted to operate on patch coordinates or could remain on global coordinates.

The choice of using a geometric progression for one of the dimensions of the grid presents difficulties in the simulation. The reason that was stated for using such a grid was that it would yield better resolution along the axis of symmetry, which is the region that they are most interested in. However, this resulted in the cells being under populated in the region of interest. Also the collision step is the most expensive part of the simulation thus a large portion of the running time will go into computing cells of low interest since the larger cells have many more collisions. Furthermore the load imbalance in the 2 dimensional decomposition would not happen on a uniform grid. These three factors cause me to believe that the simulation would yield better results in less time if it were to use a uniform grid. If the resolution along the axis of symmetry is insufficient then reducing the size of all the cells on the Y-axis would probably be a better way of increasing the resolution than using a geometric progression.

Further Enhancements: There are two further optimizations that could be made to the code. First each time step currently involves five messages being sent to each neighboring patch in the grid. These messages could be combined into a single send. Currently most if not all of the communications are amortized by the interior cell collision step so the benefits of doing this are small given the size of the problems that will be simulated. The other enhancement is to improve the load balancing in the 2 dimensional decomposition. This would involve specifying the total number of cells for each row and column of patches to use. Instead of the current approach which is to divide the cells as evenly as possible. The need for this enhancement would be eliminated if a uniform grid were used, since then each cell would have roughly the same number of particles.

References:

[1] Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard,” June 1995.