Parallel Programming with MPIJava

Solomon Lane

Table of Contents

Introduction

In preparation for working on AgentTeamwork, my goal this term was to familiarize myself with parallel programming and MPIJava. To this end I parallelized two programs that were, in the words of Professor Fukuda, not embarrassingly parallel. This also provided the opportunity to evaluate the performance improvements of parallelization as well as the communication overhead.
The first program is a molecular dynamics simulation of a 2-D Lennard-Jones fluid implemented by Daniel V. Schroeder. The second program is a two-dimensional wave simulation based on the Schroedinger Equation. The author of the Java implementation is not known to me. But the source came from http://www.geocities.jp/supermisosan/secondwaveequationjava.html

In addition to parallelizing these applications I also added modifications to improve their user friendliness. I back ported these features to the original versions as none of these features affected the core algorithms.

Molecular Dynamics

This simulation calculates the force affection/collision among N particles[1] using an arbitrary force cut-off of 3 units. I created two parallel versions of the program using different strategies. The first version simply divided the particles between the number of nodes available for the program to run on. Because each particle is potentially affected by every other particle, each node needed knowledge of all the particles in order to independently calculate the new values for it's particles. This divides the O(n^2) algorithm by the number of nodes(p) resulting in O(n^2/p). However, the parallelization adds communication overhead(h) because each node must be updated with the new values of all the particles for each calculation, resulting in O(n^2/p + h)
The second parallelization approach partitions the simulation space into stripes and assigns one node to each stripe. Each node is only responsible for calculating the force for the molecules in it's stripe. Because each molecule has to be compared to molecules within the force cutoff radius, each stripe must be at least as wide as the force cutoff radius. Each node must be aware of the molecules in it's neighbors stripes.

With N molecules and p nodes each stripe will average N/p molecules which must be compared with each molecule in the current stripe plus each molecule in the left and right neighbors which is roughly 3(N/p). This changes the O(n^2) algorithm to O(N/p*3N/p) = O(3N^2/p^2). Adding communication overhead(h) we have O(3N^2/p^2 + h).

Wave2D (Schroedinger Equation)

Wave2D starts with a cube located in a simulation space, and simulates the destruction of the cube, a wave caused by the destruction, and the dissemination of the wave[2]. I modified the program to take arguments for the simulation and cube size, and to automatically generate and display the results.
The main computation algorithm is O(n^2) . The parallelization approach partitions the simulation space and assigns one node to each partition. Each node independently calculates the change in its space. Each portion of space is affected by the space adjacent to its edge so each node needs knowledge of the edge that is shares with it's neighbor. This adds communication overhead(h) for each computation cycle. This approach reduces the computational complexity of the algorithm to O(N/p*N + h) = O(N^2/p + h).

Performance Evaluation

For the purposes of evaluating performance I ran each simulation 10 times and took the average times.

For the MD variants, drawing to the screen was disabled to remove that overhead, allowing a more accurate comparison of the algorithms. Runtime was divided into initialization time and computation time. Again to allow for a more accurate comparison of the actual algorithms.

Molecular Dynamics Performance

MD simulations where run for 10,000 computation cycles with either 100 molecules or 625 molecules. The simulations where run on the serial version of the program to create a baseline, and on each of the parallel versions with 4 nodes and 9 nodes.

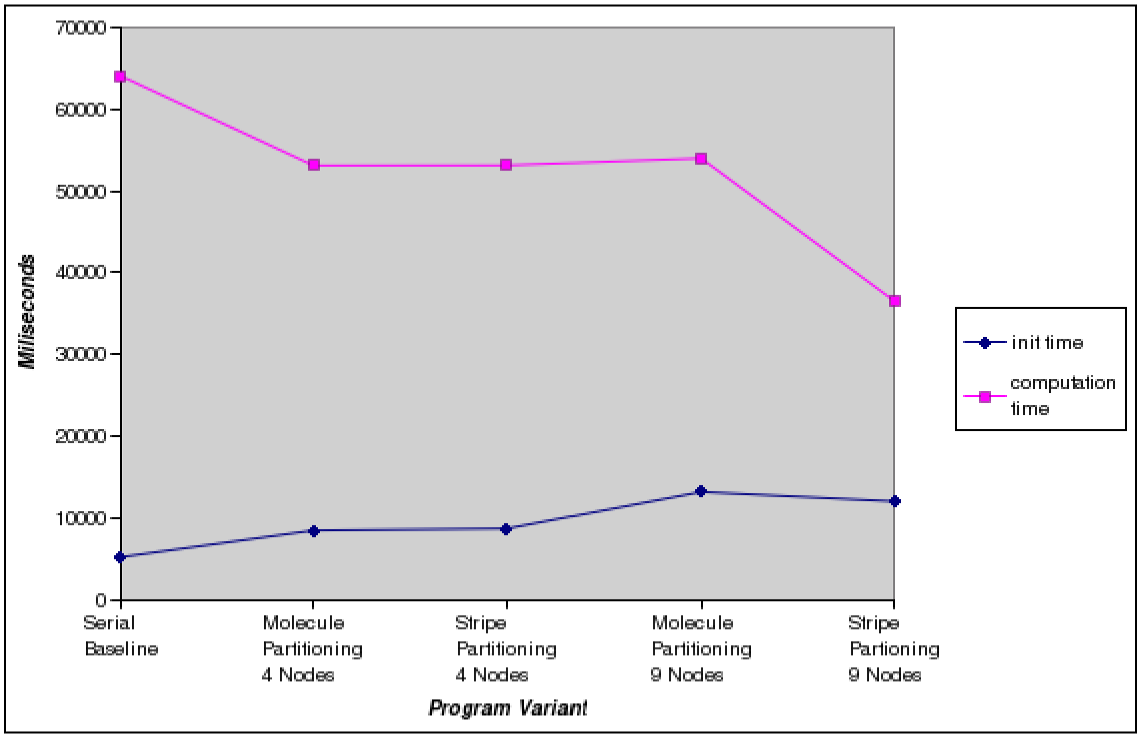

Chart 1: Molecular Dynamics Simulation - 625 Molecules/10,000 Cycles

As chart 1 illustrates initialization time increases with parallelization and computation time decreases. Computation time continues to decrease when increasing from 4 nodes to 9 nodes as would be expected. However, for the Molecule Partitioned version, the decrease is rather small reflecting the increased communication overhead with additional nodes. In the case of the Stripe Partitioned version the decrease in computation time is more impressive, despite the increased communication overhead because each additional node has greater affect on the overall complexity of the algorithm with this partitioning approach.

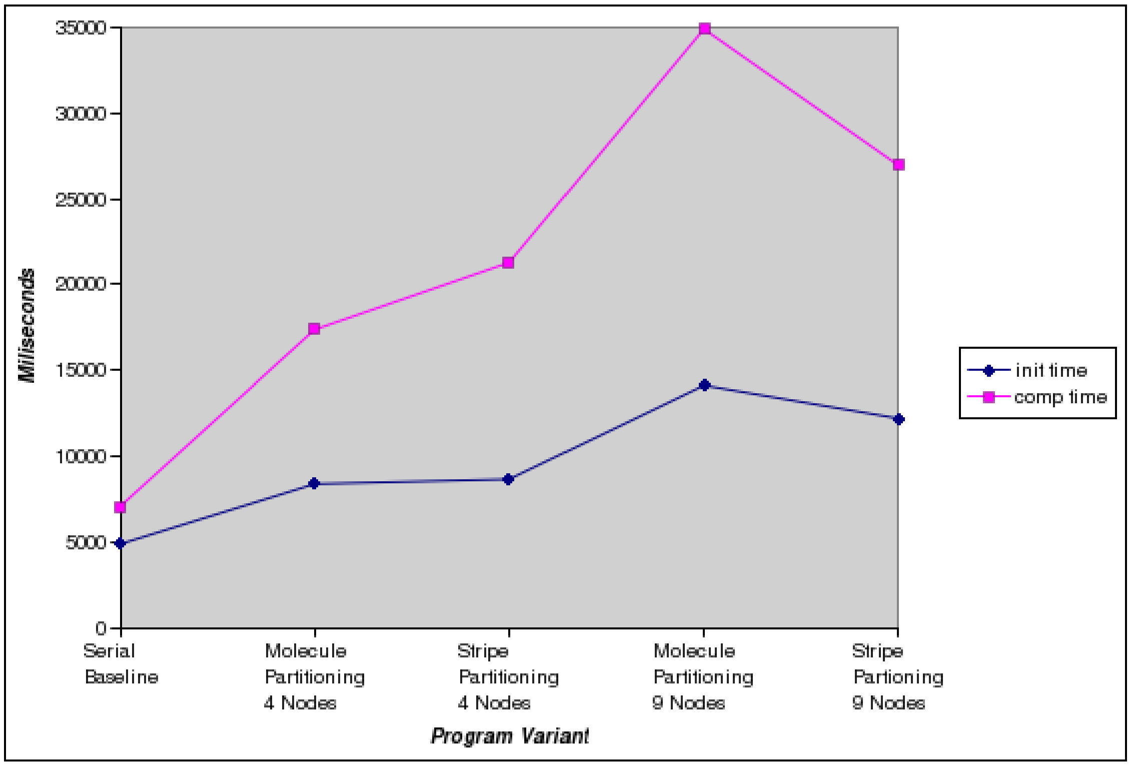

Chart 2: Molecular Dynamics Simulation - 100 Molecules/10,000 Cycles

Chart 2 shows the same simulation with a relatively small number of molecules. This illustrates that the algorithmic performance increases do not outweigh the communication overhead with small values of N.

Wave2D Performance

The Wave2D image processing time should be identical for each version of the program and it was disabled when running these simulations. Each simulation was run for 1,000 computation cycles. The simulation space was set to either 100 or 1,000 units. The simulations where run on the serial version of the program to create a baseline, and on the parallel version with 4 nodes and 9 nodes.

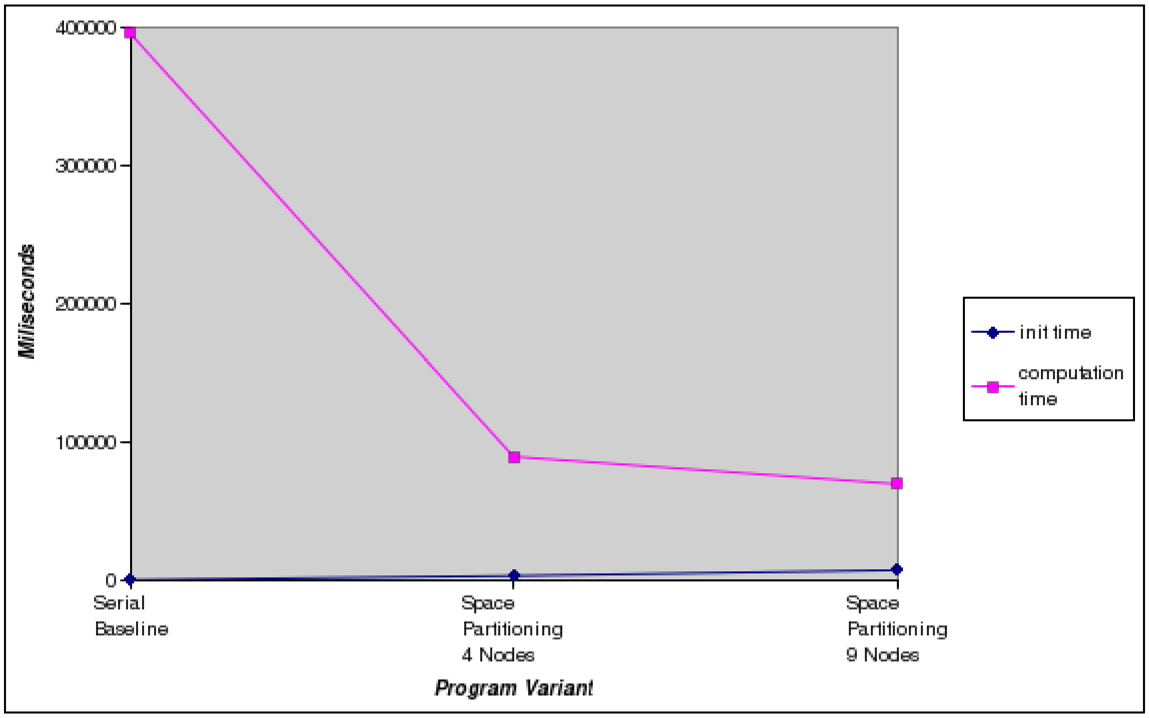

Chart 3: Wave2D Simulation - 1000 unit^2 simulation space/1,000 cycles

Chart 3 illustrates the dramatic performance improvements with the space partitioned algorithm. Communication overhead appears minimal with 4 nodes and the small amount of data that each node needs to exchange with its neighbors on each cycle. However the rate of improvement diminishes when the number of nodes is increased to nine. This indicates that the increased communication overhead for additional nodes increases at a greater rate regardless of the amount of data that is being communicated.

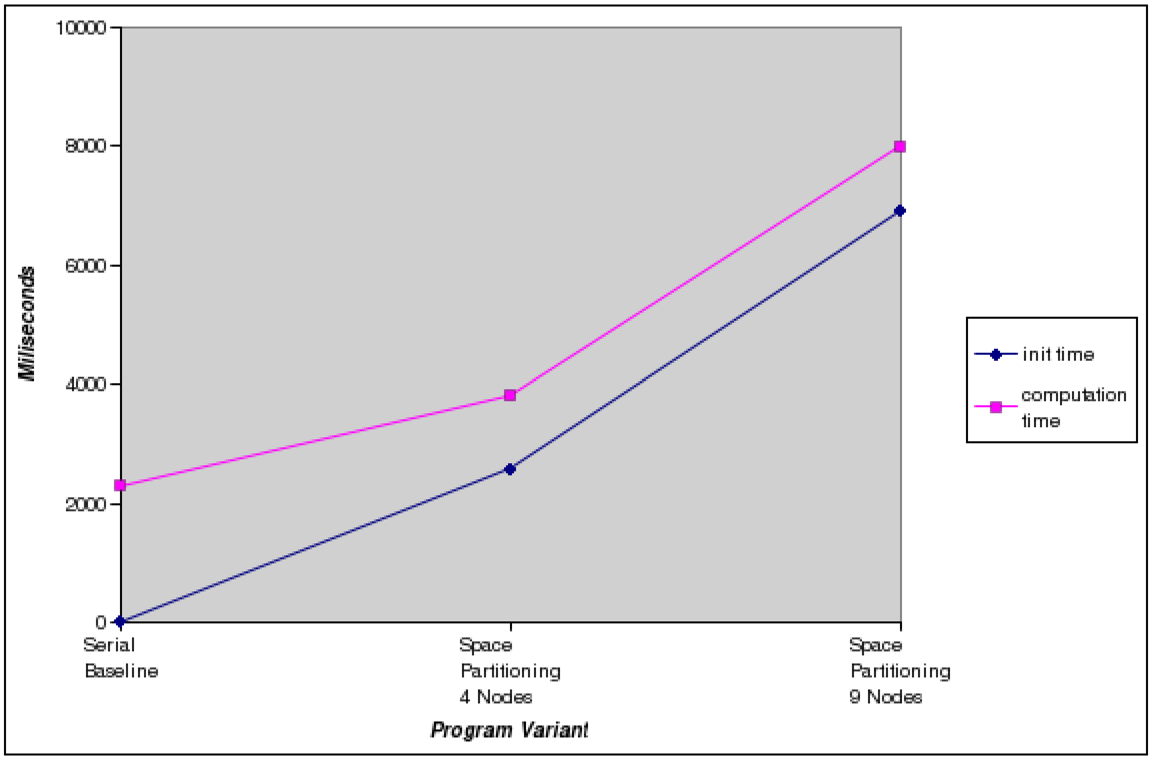

Chart 4: Wave2D Simulation - 100 unit^2 simulation space/1,000 cycles

Chart 4 shows the same simulation with a small simulation space. The space partitioned algorithm still reduces computation time despite communication overhead, but it doesn't offset the increased initialization time.

Conclusions

Parallelization with Mpijava improved the performance of these algorithms once they scaled to larger values of N. The performance improvements are most dramatic when communication overhead is minimized. There seems to be sweet spot with regards to the number of nodes after which we see diminishing returns in performance improvement even when communication is limited to small amounts of data.

Citations

[1] Professor Fukuda, “CSS499 statement of work”
[2] Professor Fukuda, “CSS499 statement of work”

Appendix 1 – Molecular Dynamics Simulation Details

Serial Baseline

source code: /home/uwagent/MA/applications/Molecules/mpj/MD625.java
details: Line 154 was commented out so the program would not draw to the screen
Simulation 1 execution: java MD
Lines 45-48 set the following runtime parameters:

```
static      int N = 625;             // number of molecules
static final int pixelDiameter = 6;  // molecule diameter in pixels
static final int boxWidth = 100;       // in units of the molecular diameter
static final int boxHeight = 100;
```

Simulation 2 execution: java MD
Lines 49-52 set the following runtime parameters:

```
static      int N = 100;             // number of molecules
static final int pixelDiameter = 10;  // molecule diameter in pixels
static final int boxWidth = 40;       // in units of the molecular diameter
static final int boxHeight = 40;
```

Molecule Based Partitioning

source code: /home/uwagent/MA/applications/Molecules/mpj/MD.java.finalMolecule
details: Line 319 was commented out so the program would not draw the results to the screen
Simulation 1 execution:
4 Nodes: prunjava 4 MD -n 625 -d 6 -w 100 -h 100 -r 10000
9 Nodes: prunjava 4 MD -n 625 -d 6 -w 100 -h 100 -r 10000

Simulation 2 execution:
4 Nodes: prunjava 4 MD -n 100 -d 10 -w 100 -h 40 -r 10000
9 Nodes: prunjava 4 MD -n 100 -d 10 -w 100 -h 40 -r 10000

Stripe Based Partitioning

source code: /home/uwagent/MA/applications/Molecules/mpj/MD.java.finalStripes
details: Line 449 was commented out so the program would not draw the results to the screen
Simulation 1 execution:
4 Nodes: prunjava 4 MD -n 625 -d 6 -w 100 -h 100 -r 10000
9 Nodes: prunjava 4 MD -n 625 -d 6 -w 100 -h 100 -r 10000
Simulation 2 execution:
4 Nodes: prunjava 4 MD -n 100 -d 10 -w 100 -h 40 -r 10000
9 Nodes: prunjava 4 MD -n 100 -d 10 -w 100 -h 40 -r 10000

Appendix 2 – Wave2D Simulation Details

Serial Baseline

source code: /home/uwagent/MA/applications/Wave2D/mpj/Wave2D.java.gui
Simulation 1 execution:
java Wave2D -n 1000 -xmin 400 -xmax 600 -ymin 400 -ymax 600 -r 1000
Simulation 2 execution:
java Wave2D -r 1000

Space Based Partitioning

source code: /home/uwagent/MA/applications/Wave2D/mpj/Wave2D.java.final
Simulation 1 execution:
4 nodes – prunjava 4 Wave2D -n 1000 -xmin 400 -xmax 600 -ymin 400 -ymax 600 -r 1000
9 nodes - prunjava 9 Wave2D -n 1000 -xmin 400 -xmax 600 -ymin 400 -ymax 600 -r 1000
Simulation 2 execution:
4 nodes – prunjava 4 Wave2D -r 1000
9 nodes – prunjava 9 Wave2D -r 1000