Questions on the Midpoint Method for the Parallelization of Particle Simulations

John L. Weatherwax[*]

October 4, 2006

2006-09-28: The code at this point is finished. The parallel and the serial version produce the same results for several timesteps and many particles. I will now begin to run experiments to characterize the parallel algorithms speedup.

2006-09-17/2006-09-24/2006-09-26: I now have code that updates the data structures in each home processor when particles move from one home box to another. A major realization when working with the current implementation is that it would be much better to have the data structures in each home box consist of linked lists rather than statically defined arrays. This will be implemented if I have time. I have performed the following debugging checks and found my serial code and parallel codes to agree

1. I have checked that when we have *zero* forces both the serial and the parallel version of the code agree for one timestep.
2. I have checked that when we have *zero* forces both the serial and the parallel version of the code agree for two and more timesteps. Note: this was checked again on 2006-09-24.
3. I just checked that for two particles that stay in the same home for the real force calculation box each algorithm gives identical results.
4. The force calculation works for several timesteps and agrees exactly (to all precision) with the serial version.

2006-09-03: I now have code that exchanges particles when performing the particle exports *and* when particle ownership should be handed over to neighboring processor. The final step before completion of an implementation of this algorithm is to update the data structures after receiving all particles that have been evicted from their neighbors. Once this is done I'll swing into full scale debugging to check that logically everything I have done makes sense. I plan on doing this with an $O(N^2)$ serial version of range limited force. This can be found at

[*]wax@alum.mit.edu

```
Code/Testing/sequential_short_range.c
```

I hope to have this done in a couple of weeks.

2006-08-04: It seems the code to send particles from one processor to another is now working. Next I will implement the code to compute gravitational short range forces between all imported particles. **TODO:** I have cheated in the implementation by assuming that each particle is of unit mass. I should certainly go back and fix this. It is becoming more and more clear that I'm going to have to overlap the communication with the computation in this implementation of the midpoint method to further boost performance. This should be done also.

2006-07-31: I have implemented the routines to determine which particles each home box should export. This is performed in the routine `determineExportRegion.c`. A plot of the particles that will be exported (for a 2x2 grid) to their neighboring processors (for a very dense particle number) and implying the algorithm correctness is given below. Now I need to implement the message passing that will pass the particles tagged to be exported to each of the neighbors. This I'll do with a series of alternating `MPI_Send` and `MPI_Recv` sweeps horizontally, vertically, and diagonally. Note it might be faster to use addition MPI primitives like `MPI_Sendrecv` instead of the paired `MPI_Send` and `MPI_Recv`.

2006-07-16: One assumption in the code is that each individual boxes only requires particle imports from its *nearest neighbors.* This simplifies the coding of the particle imports since I don't have to consider how many neighboring boxes around each individual home box need to be searched for particles to import. The number of boxes required to be searched would depend in a complicated way on the interaction radius, the periodic boundary conditions, and would make the message passing more difficult to implement. This would only be a problem in situations where there were a great number of processors (dense virtual grid) and a long interaction radius.

2006-06-03:

I think I have completed a first attempt at the code required to create data structures for an implementation of a two dimensional version of the midpoint method. A high level overview of the structures I am using to solve this problem can be found in the file `global_grid.h`. I am using a virtual Cartesian grid with the processors numbered from top to bottom, left to right. Each processor will store a set of local particle positions and velocities, in addition to any static data required to calculate such things as box export regions. For the determination of the export regions, I'm currently only keeping the spatial locations of the corners of the individual boxes and this brings me back to my first question.

The code begins by setting up the above static (meaning it is constant for the entire simulation) data structure before it performs any iterations. A high level view of the code can be seen in the routine `midpoint.c`, which also contains the main function. At this point before I do any more I'd like to ask you to look over what I have thus far and then to talk to you specifically about how to determine the export region for each box with the structures

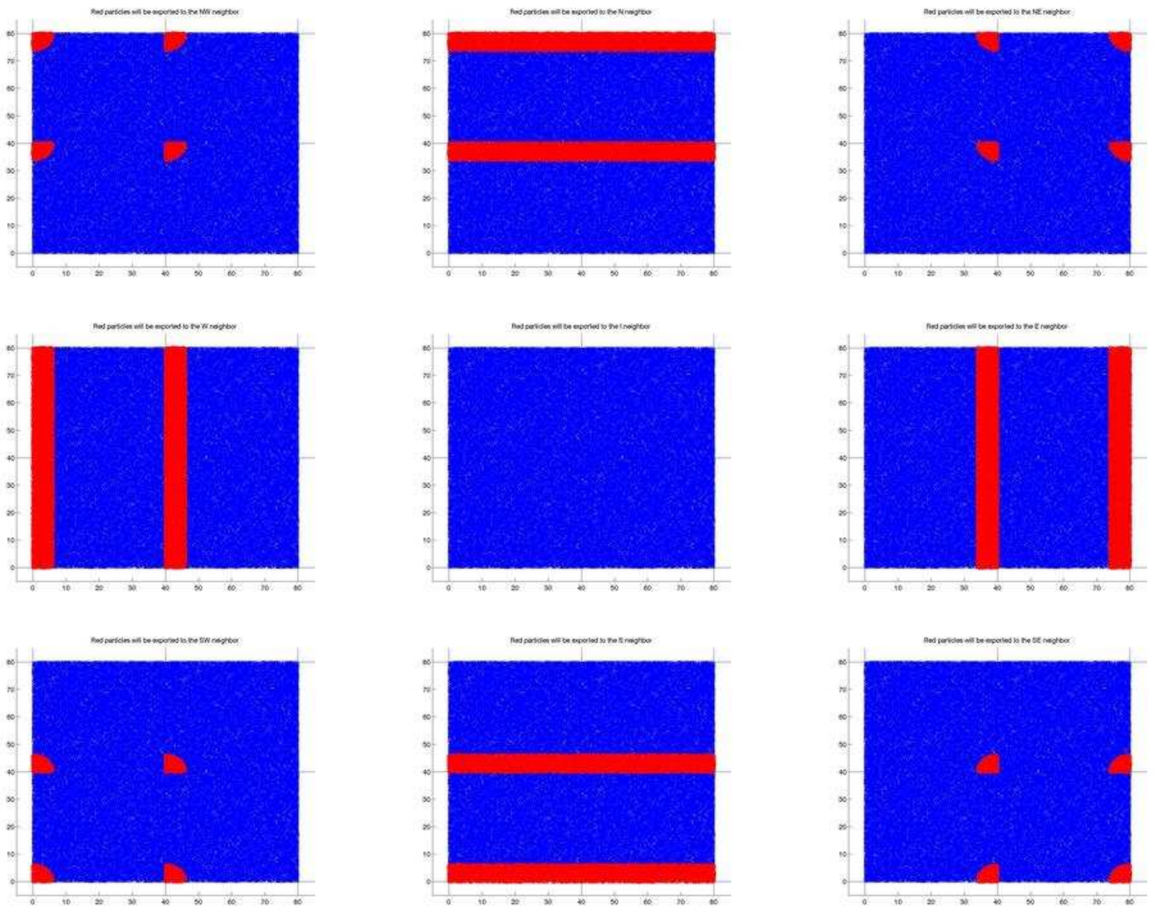

Figure 1: A plot of the particles that will be exported to their neighboring processors. In this example there are four processor arranged in a 2x2 grid. We display from top to bottom, left to right, the particles that will be exported to the Northwest (NW), the North (N), the Northeast (NE), the West (W), the Identity (I), the East (E), the Southwest (SW), the South (S), the Southeast (SE). Not that the identity is simply a placeholder obviously no particles should be exported since each processor already has them.

I have defined and or ones that I should add.

Question: Do I still need to do particle export on the particles that border the outer edge of the domain? I mean the particles that when exported would exercise the periodic boundary conditions.

Question: What are the requirements/constraints of the periodic grid?

2006-05-23:

From the first reading of your paper "The Midpoint Method for Parallelization of Particle Simulations", I've constructed a few notes in trying to better understand everything and to help guide our upcoming discussion. As I tried to write this method in my own words questions came up and what follows is rather like a literary "stream of consciousness. As it stands now here is my understanding of the midpoint method. I should note that currently I've only had time to read up to the section entitled "Midpoint-Ensured Methods".

- First each processor determines its "export region", the set of particles to be sent from their home box to another processor who will perform the computation of the pairwise interactions. Every particle with linear dimension closer than $R/2$ from a known processor edge boundary (box edge) will be exported by this processor. Computationally, this step is simply a particle loop within each processor, creating data structures to hold the particles that will be exported to the neighboring processors. For instance in Figure 4 (a) from your paper each processor would have 8 data structures containing the set of particles to be exported to its neighboring processors.
- Second, (the particles spatial locations) this data is then sent to the neighboring processors.
- Next, each processor computes internally the pairwise forces between all received particles.

  Question: When it happens that two particles are near the spatial domains edge does each neighboring processor import them *both*? I would think yes but then after import each processor would compute the midpoint of every corresponding pair, determine if it falls within the current processor and only compute the interaction force if it does. This would require an $O(N_l^2)$ computation to compute all of the midpoints of every pair but maybe this is not too computationally expensive since the number of local particles N_l in each processor is expected to be relatively small, compared to the total number of particles in the system. There are more advanced methods that could be used to reduce this calculation down, but this logic is fine.
- Next, with these newly computed forces on each particle we package up all the forces that go to a neighboring home box and then send them to their home box.
- Next, each processor then updates the position of the particles it is responsible for.

- Finally, each processor then computes which particles have moved outside of their controlling domain and these particles are package up and transmitted to the neighboring processor with another set of send receive type calls.

- The above loop is repeated for the next timestep.

  Question: Where would the long range charge assignment to underlying mesh go in the above looping?

  Answer: This could be done after several loops of the short range forces.

  Question: Do I have too many point to point MPI sends/receives in the above algorithm, i.e. can it be optimized further someway by delaying the send and receives?

  Answer: There are always ways to modify code to make it run differently and still be correct. What you have is one correct way.

  Question: How does one do data sends in sweeps, i.e. from each processor to its left/right neighbor?

  Answer: This is easily done using MPI Cartesian communicators. For example the MPI command `MPI_Cart_shift` performs returns the ranks of source and destination processes for a subsequent call to `MPI_Sendrecv` to shift data in a coordinate direction.