

Parallelization of Particle-Particle, Particle-Mesh Method within N-Body Simulation

Nicholas W. Nocito
 Kean University, NJCSTM
 1000 Morris Avenue
 Union, NJ 07083
 nociton@kean.edu

ABSTRACT

The N-Body problem has become an intricate part of the computational sciences, and there has been rise to many methods to solve and approximate the problem. The solution potentially requires on the order of N^2 calculations each time step, therefore efficient performance of these N-Body algorithms is very significant [5]. This work describes the parallelization and optimization of the Particle-Particle, Particle-Mesh (P3M) algorithm within GalaxSeeHPC, an open-source N-Body Simulation code. Upon successful profiling, MPI (Message Passing Interface) routines were implemented into the population of the density grid in the P3M method in GalaxSeeHPC. Each problem size recorded different results, and for a problem set dealing with 10,000 celestial bodies, speedups up to 10x were achieved. However, in accordance to Amdahl's Law, maximum speedups for the code should have been closer to 16x. In order to achieve maximum optimization, additional research is needed and parallelization of the Fourier Transform routines could prove to be rewarding. In conclusion, the GalaxSeeHPC Simulation was successfully parallelized and obtained very respectable results, while further optimization remains possible.

General Terms

Performance, algorithms, design

Keywords

N-Body, Message Passing Interface, Particle-Mesh, Fourier Transform

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright ©JOCSE, a supported publication of the Shodor Education Foundation Inc.

1. INTRODUCTION

The N-body problem is an example of an algorithm that in simple to explain to students, but that quickly grows in complexity and need for resources. Dealing with the potential interactions between particles in a distribution, the N-body problem would ideally compute all possible interactions, and if every N objects interacts with every other (N-1) objects, this results in $N(N-1)$ total possible interactions, growing as N^2 in complexity. This category of problems has applications in many fields such as astrophysics, molecular dynamics, fluid dynamics, and plasma physics.[2] This paper applies the N-Body problem to the simulation of celestial bodies in space, particularly in the case of the "universe in a box" problem where the space being calculated is assumed to be one unit cell out of an infinite expansion.

This research opportunity arose through my selection into the Blue Waters Undergraduate Petascale Education Program. In order to best prepare for the experience we received two weeks of intensive training at the National Center for Supercomputing Applications (NCSA), via the University of Illinois at Champagne Urbana. During the training we were exposed to the world of high performance computing and its architectures and applications. We gained experience in shared-memory parallelism with OpenMP, and distributed system parallelism with MPI. In addition we were exposed to modern applications in computing using GPU architectures. At the time of my research I was a rising junior in Kean University's Center for Science Technology & Mathematics program, majoring in Computational Applied Mathematics. My research and study was done under the mentoring of my advisor, Dr. David Joiner, Kean University. Prior to this summer I had taken multiple mathematics courses, a calculus-based physics course, and a basic java computer programming course.

2. BACKGROUND

2.1 The N-Body Problem

The N-Body problem is a classic physics problem dealing with the interactions of particles. When we are dealing with more two or more particles, each potentially interacts with every other particle, and each force pair is equal and opposite giving $N(N-1)/2$ unique forces. Unfortunately, the scaling then leans towards N^2 which quickly accumulates when dealing with problems of large N. The initial conditions m_i, r_i, v_i are the mass, coordinates, and

velocity of each particle, respectively. Because of the singularity in the forces of close interactions a cutoff radius, sometimes call a shield radius, is typically used to reduce the computational error due to close interactions. Alternatively, a softening radius can be added in the calculation of the potential. Both of these are options in GalaxSeeHPC [1]

Shield Radius:

$$a_i = \frac{\vec{F}_i}{m_i} = -G \sum_{j=1, j \neq i}^N \frac{\alpha_{ij} m_j (\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j|^3} \tag{2.1}$$

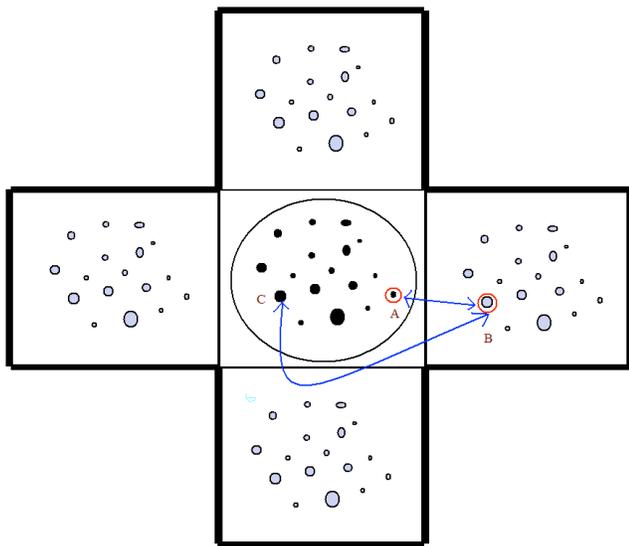
$$\alpha_{ij} = \begin{cases} 1 & |r_i - r_j| > r_{cutoff} \\ 0 & otherwise \end{cases}$$

Softened Potential:

$$a_i = \frac{\vec{F}_i}{m_i} = -G \sum_{j=1, j \neq i}^N \frac{m_j (\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j + \epsilon|^3} \tag{2.2}$$

2.2 Particle-Mesh Method

In the case of the “universe-in-a-box” problem, the N-body problem is solved in a system with a “wrapped” geometry, assuming that we are solving for a typical unit cell out of an infinite expanse. The wrapped geometry in gravitational dynamics is typically used to solve for large scale structure, where solving for the entire universe is not practical, and one assumes that there is mass immediately outside of the box that would affect the evolution of the system. Such methods are also used in molecular dynamics calculations, where it is necessary to maintain a consistent bath of solvent molecules around a protein being studied. The Particle-Particle Particle-Mesh method allows for the use of a spectral technique assuming a periodic solution.



(2.3) Illustration of a “wrapped” world as is pertains to the code. The distance between particle A and B represents the “ghost” distance. Particle B does not exist because we are dealing with a “wrapped” world. Therefore, the “real” distance is between particle A and C.

Poisson’s Potential Energy equation is given by:

$$\nabla^2 \Phi = 4\pi G \rho \tag{2.4}$$

The algorithm for the density population first begins with incrementing through the array of coordinates and setting the initial density grid to zero at each particle. Next we must find the nearest grid point for each body of mass and “wrap” if needed. After locating the nearest point the range around that grid point is then calculated and we ensure that coordinates are broken by the wrapping of any particles. The final part of the density distribution is incremented through each of the bodies and updating the density array with the values for the particle mesh.

Through a Fourier Transform, the solution to Poisson’s equation becomes:

$$\hat{\Phi} = \frac{-G}{\pi |k|^2} \hat{\rho} \tag{2.5}$$

In practice, an additional function is used to smooth out shorter range forces and control the scale of the long range forces. This function, called the influence function, is typically taken to be the Fourier transform of the density function of a single particle in the system during the density grid population [4].

$$\hat{\Phi} = \frac{-\hat{\lambda}(k)G}{\pi |k|^2} \hat{\rho} \tag{2.6}$$

The basic procedure of the PM method is as follows: [6]

- Transform particle mass into density distribution
- Solve Poisson equation using FFT algorithm
- Calculate the force fields and interpolate to find forces
- Integrate to obtain positions and velocities
- Update time step

The Particle-Particle, Particle-Mesh (P3M) is a hybrid method for approximating the solution to the N-body problem. PM methods suffer from an inability to predict short range forces accurately, as any nearby effects are “smoothed” out over nearby gridpoints. The P3M method adds a nearest neighbor “particle-particle” interaction that essentially divides the forces into short-range forces, and long-range forces.[7] Short-range forces are calculated using the direct force method which is the brute force calculation using equation 1.1 from above. The longer range forces are then calculated using the Particle-Mesh approximation.[6] Getting an accurate solution efficiently requires careful selection of the size scale of the density function

for each object in the density population, the influence function, and the radius used to determine nearest neighbors. The hybrid P3M method ideally results in a scaling of $N \log(N)$.[4]

2.3 GalaxSeeHPC

GalaxSeeHPC is a N-Body simulation source code. It requires an UNIX-like environment, or a Cygwin-like environment for windows applications. The code provides many options of different force calculation techniques. You may chose the direct force method, the Fourier Transform based P3M method or the Barnes-Hut tree-based method. For the purpose of this paper we will only discuss in detail the P3M method.

2.4 Message Passing Interface

With the occurrence of larger and larger problems needed to be solved, comes the use and application of supercomputers. The most widespread accepted parallel programming language is the Message Passing Interface, or MPI.[8] Users of C or Fortran can use MPI to pass messages between the nodes of the cluster. This communication can spread out the computer’s work and drastically alter the overall performance. If a code has to little work or data the addition of MPI could potentially hurt performance. Also, one must take into consideration the amount of communication the nodes will have to have with each other. Too much communication can also produce harmful effects on a code.

3. PROFILING AND DESIGN

3.1 Profile

In order to attempt to optimize the performance of the code we first began profiling GalaxSeeHPC under many different conditions. The code was examined for various values of N, Final Time, and grid size (the resolution of the mesh, higher the resolution the more accurate). For use of GalaxSeeHPC it is best to have the grid size set to a power of two; for example, 16, 32, 64, etc.

The implementation of the P3M algorithm in GalaxSeeHPC follows three steps in the force calculation, population of a density grid, FFT solution of Poisson’s equation on that grid, and interpolation of forces from the Poisson solution back to individual points, along with directly calculated nearest neighbor corrections. Shown below is the percentage of the wall time (as determined through the use of both hard coded timers and gprof) taken up by the density population step. Note that for smaller grid sizes, population of the density grid dominates the time required for a force calculation—compared to the time required for the FFT calculation for larger grid sizes. This suggests that for lower resolution P3M grids, speedups on the order of 20 to 50 times should be obtainable through parallelizing the density population alone.

GRID SIZE	N = 1,000	N = 5,000	N = 10,000
16	.97	.97	.96

32	.93	.94	.94
64	.73	.88	.91
128	.27	.56	.68
256	.08	.18	.24

(3.1) This figure portrays the percentage of total wall time which the creation of the density grid was responsible for.

3.2 Parallelization

By examining the code further, clearly the last part of the density creation loop had the highest potential for parallelization. The statement we parallelized is said to *embarrassingly parallel*. A code is said to be *embarrassingly parallel* if there are no channels between tasks and each process can perform its duties without communication with any other processes.[8] These types of algorithms are usually the easiest to parallelize because of this lack this ability to do work without interaction.

The next step after defining the subset of code to be parallelized is design. A round-robin technique was used, where each node does some work, and returns to do more based on its rank and world size. The rank is each processor’s own unique ID number, so that we have a way of distinguishing between nodes. The world size is the total amount of processors initialized at run time. The following three functions are the three most basic and important functions when using the Message Passing interface.

MPI_Init (&argc, &argv);

MPI_Comm_rank (MPI_COMM_WORLD, &rank);

MPI_Comm_size (MPI_COMM_WORLD, &world_size);

Mpi_Init is the function initializes the use of Message Passing Interface. The MPI_Comm_rank and MPI_Comm_size determine the processor id’s and total number of proccessors, respectively. Rank and world_size are arbitrary variable names which hold the IDs and total size. Now that we have MPI initialized we can parallelize the code.

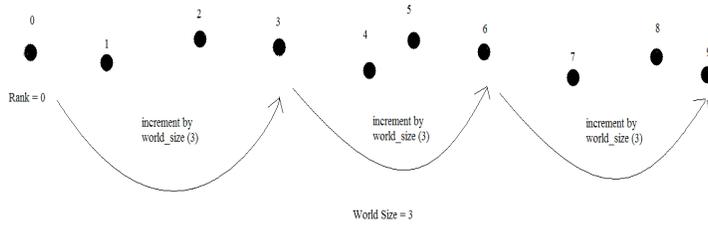
The initial parameters for the loop in the density grid creation was as follows:

```
For(l = 0; l < theModel->n; l++)
```

This is a very basic loop incrementing by one and looping through all the bodies of mass in the model. In using the round robin technique we change the loop to the following:

```
For( l= rank; l<theModel-n; l += size)
```

Now each processor will begin its loop at its rank (ID) and increment by the world size. The following diagram depicts the round robin use of MPI.



(3.3) Depicts the round robin technique in a group of 3 nodes. Each node begins at its own ID and increments by the total amount of nodes being used.

This type of parallelization is also referred to as data decomposition. This is defined as when multiple tasks have same responsibility but work on different sets of data.[3] As opposed to functional decomposition where each task has its own set of responsibilities.[3] Now the data is broken up, but we still need to make sure every process receives the results of every other process so that have the results of the entire density grid.

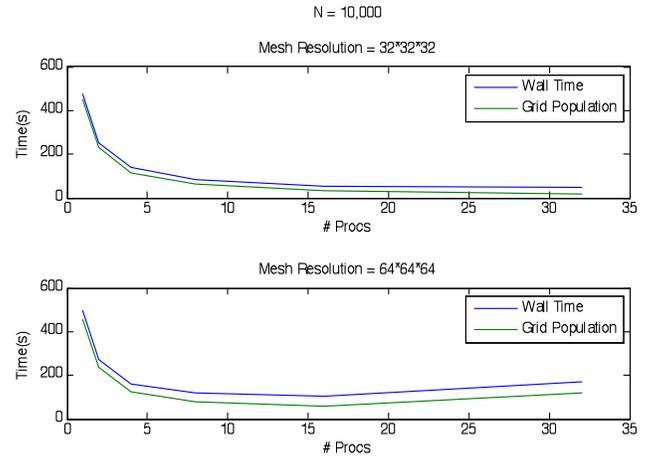
For this we will use one of MPI's reduction commands. MPI_Allreduce is a collective communication, so each processor will receive the reduction results.[7] Below is the command along with its necessary parameters.

```
MPI_Allreduce(
    Void* send_buffer, // = the send buffer
    Void* recv_buffer = the receive buffer
    Int cnt = the number of elements to reduce
    MPI_Datatype dtype = Element type
    MPI_Op op = Reduction Operator
    MPI_Comm comm. = Communicator
);
```

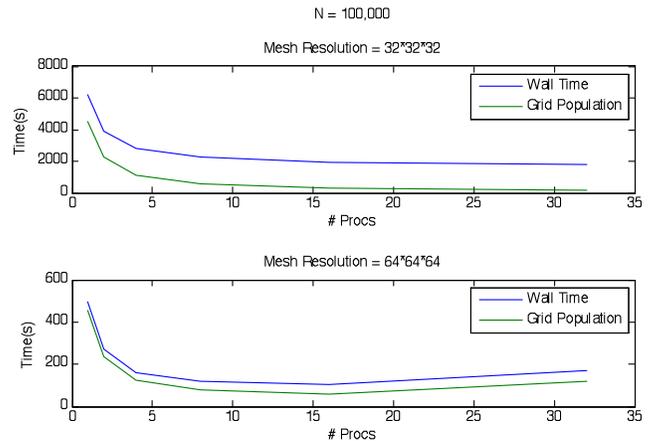
After reducing to a temporary buffer we can transfer all the results back to each processors density array. The result is all the nodes sharing the results of the density grid creation with each other.

4. RESULTS

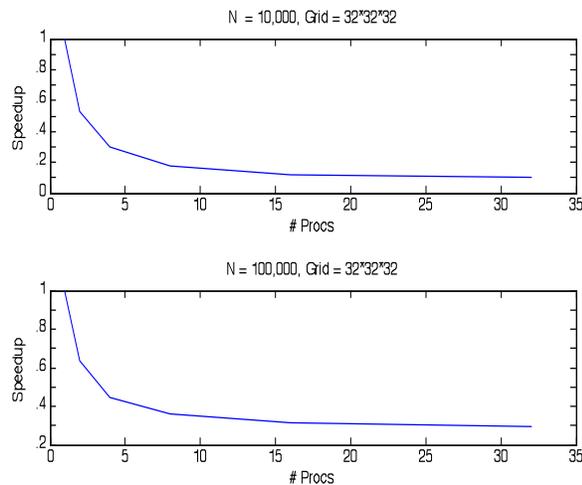
After successfully profiling and parallelizing the code we were able to see very significant results. Below are a few graphs represents the increased speed up we received by running the code with various amounts of nodes.



(4.1) Above are the optimization results when simulating a galaxy of 10,000 stars. Below are the results when simulating 100,000 stars.



In the supercomputing world there exists Amdahl's law which accurately predicts the maximum speedup of parallelized code. Amdahl's law assumes we are trying to solve any problem of constant size as quickly as possible, and can determine potential speedup achievable as you increase the number of processors. [7] The equation is simply that the maximum speedup is equal to one over the fraction of the time spent in code that must be run serially. For instance, if 10% of your compute time is from code which can not be parallelized, then the maximum performance you can achieve is 1/.10 or a speed up of 10x.



(4.2) Above are the Speedups of the Wall-times of GalaxSeeHPC as a function of the # of processors. The speedup value is calculated by divided the wall-time of P processors by the wall time for one processor.

By examining graph (4.2) we see that our parallelized code for 10,000 bodies maxes out at around .1, which results in speedup of 10x. However, when we are dealing with 100,000 bodies we only see a max of .29 with a speedup of 3.45x. We can take our results from the above table (3.1), and apply them to Amdahl's Law. Therefore, for N = 10,000 bodies and a grid size of 32*32*32, the maximum achievable speedup = $(1) / (1-.94)$ or 16x. Although we received a very impressive speedup of 10x for N = 10,000, we weren't able to totally optimize within accordance of Amdahl's Law's projected maximum speedup of 16x.

If we examine table (3.1) further we notice as the grid resolution becomes larger and larger (which increases accuracy), the percentage of total time used in creating the density grid decreases. This is because as the grid sizes increase so does the time in calculating the Fourier Transform by the FFTW algorithm. Therefore, our next task in fully optimizing performance of GalaxSeeHPC should be the profiling and parallelization of the 3 dimensional Fourier Transform of the gravitational potentials.

5. CONCLUSION

5.1 Analysis

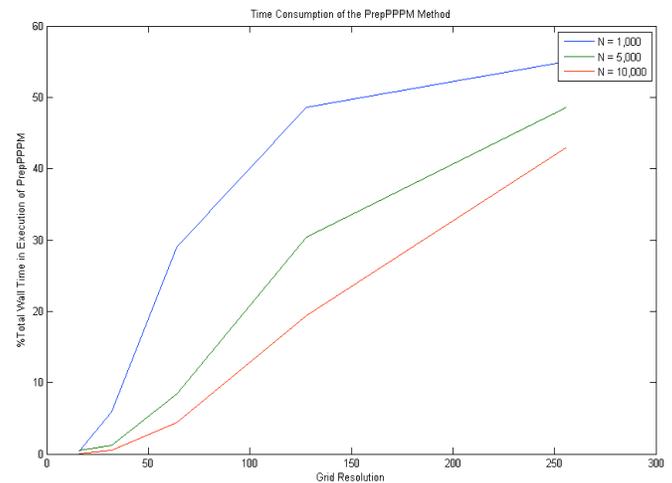
Our analysis shows that the current parallelization of GalaxSeeHPC using the P3M method scales in accordance with expectations, however, limited by the time required to perform the FFT, which as of yet has not been implemented in parallel. The results substantiate that the population of the density grid consumes much CPU time, and that parallelization of the algorithm can lead to improved performance.

Howbeit we achieved speedups of up to 10x; we must not overlook that only one algorithm in the code was parallelized. To obtain maximum performance we must continue to examine the

code for potential methods possible to execute in parallel. As stating above, the next step would be optimizing the FFTW algorithm, within GalaxSeeHPC's PrepPPPM method. There is now Message Passing algorithms available for the FFTW open source libraries. Below is data which validates that in order to reach maximum attainable speedups, implantation of the MPI-based Fourier Transforms would be the next sensible phase.

#Procs	1	2	4	8	16	32
%Total Time	3%	6%	11%	17%	22%	25%

(5.1) Above represents the direct proportionality of the percentages of total wall time spent in PrepPPPM, and the number of processors used in execution, for N = 10,000, and a Grid Resolution of 32*32*32.



(5.2) The above graph is the Percentage of total time spent in PrepPPPM as a function of the Grid Resolution. The time in computing the Fourier Transform increases as the mesh size increases.

Use of the GalaxSeeHPC Simulation will require a significant amount of CPU time, and computing power. Time constraints and computing resources necessary will depend on the actual problem set. For instance, computing a simulation of a galaxy of 100,000 stars on one core, over the course of 15 billions years and a time step of .1 million years would require a tremendous amount of CPU time. Our earlier results, when running 100,000 stars for 100 time steps, required just over 6000 seconds to run. Therefore running at 15 billion years with a time step of .1 would require approximately 1500x more CPU time, which results in over 2584 hours!

5.2 Reflections

Through Blue Waters' education program I received an invaluable educational experience. This section describes the impact this research internship had on my learning experience, and how to potentially use this paper in undergraduate education so that others may gain knowledge in high performance computing applications.

Firstly, one must master the basic mathematics and physics concepts behind the simulation. The N-Body is calculated by means of Newtown's basic physics equations depicted in (2-1). This governing equation should not take much mathematics knowledge to comprehend, and I already had prior experience with these physics concepts. The most abstract concept for me, which is most likely to be the one misunderstood by students, is the use of Fourier Transforms. A Fourier Transform in a very basic sense is the transformation of one equation from the time domain to the frequency domain. The result of the transform will depict the frequencies of the original equation. Initially I had no knowledge of Fourier techniques, therefore in order to properly understand the algorithms of GalaxSeeHPC; I had to master the concepts behind Fourier transforms and their applications. Therefore, future students who wish to benefit from this paper must first comprehend the concepts behind the simulation.

I also had to familiarize myself with MPI and ways of thinking *parallel* in order to formulate the best algorithm. The training provided by the NCSA allowed me to efficiently discover algorithms in the code which would benefit by the application of MPI and its parallel routines. However, before I actually implemented any code changes, I had to first properly model and profile the code. I began executing the code for many different sets of initial conditions, and used Microsoft Excel and MATLAB in order to make sense of these results. I also learned how to efficiently use gprof (GNU Profiler) which aided in shedding light on which method calls within the simulation actually were taking up the most time. However, in order to analyze the code as accurate as possible, I also implemented my own timing methods into GalaxSeeHPC. Students must learn how to correctly profile and analyze in order to locate the algorithms which are best-fit to parallelize.

I was also fortunate enough to work in a team environment throughout my research. During my training I had a chance to collaborate with the other gifted students who were chosen as Blue Waters Petascale interns, and continued to use them as resources throughout my research by using OpenStudy, an online study group. I also collaborated with my professors at Kean's NJCSTM and some of our graduate students. There is always more than one way of implementing a problem in parallel and having a team environment is very useful in discovering the optimal solution.

This research experience can potentially be replicated by professors for use in their research, as well as in undergraduate education. The code itself can be used in classrooms, and can execute scalable N-Body simulations for different boundary conditions. Using the parallelized P3M algorithm within GalaxSeeHPC, students and researchers can experience speedups of 10-20x, and will have the ability to execute simulations for very large N. Also students may benefit from the research

methods I used and how to successfully analyze algorithms, and implement high performance computing techniques. There are many different paths one could take in furthering this research, depending on their field of study, and intended teaching application. For instance, a computer scientist may consider analyzing the Barnes-Hut method for parallelization, an alternative N-body approximation using tree data structures. However, a physicist may be more interested in using the parallelized P3M methods in order to conduct research on the organization of large scale universes, and how their formation.

6. ACKNOWLEDGMENTS

Thanks to Dr. David Joiner of Kean University, for the constant support and mentorship throughout the process. Also thanks to SHODOR and the NCSA for the support and education experience.

7. REFERENCES

- [1] Aarseth, S., *Gravitational N-Body Simulation*, Cambridge Monographs of Mathematical Physics, 2003.
- [2] Aluru, S., Prabhu, G.M., and Gustafson, J. *Truly Distribution Independent Algorithms for the N Body Problems*, Ames Laboratory Department of Energy, Ames, Iowa, <http://www.scl.ameslab.gov/Publications/Gus/N-Body/N-Body.html>.
- [3] Binstock, A., *Data Decomposition: Sharing the Love and the Data*, <http://software.intel.com/en-us/articles/data-decomposition-sharing-the-love-and-the-data/>, November 4th, 2009
- [4] Board, J, and Toukmaji, A., *Ewald Summation Techniques in Perspective: A Survey*, Computer Physics Communications Volume 95, Issues 2-3, June 1996, Pages 73-92, 1999
- [5] Graps, A., *N-Body Particle Simulation Methods*, <http://www.amara.com/papers/nbody.html>
- [6] Shodor, *GalaxSeeHPC*, <http://shodor.org/petascale/materials/UPModules/NBody/>, 2010
- [7] Splinter, R., *A Nested Grid Particle-Mesh Code for High Resolution Simulations of Gravitational Instability in Cosmology*, <ftp://asta.pa.uky.edu/cosmology/ngpms.ps>
- [8] Quinn, M., *Parallel Programming in C with MPI and OpenMP*, 2004