# MOLECULAR-DYNAMICS SIMULATIONS USING SPATIAL DECOMPOSITION AND TASK-BASED PARALLELISM

by

CHRIS MANGIARDI

A thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science (MSc) in Computational Sciences

The Faculty of Graduate Studies
Laurentian University
Sudbury, Ontario, Canada

# Abstract

Molecular Dynamics (MD) simulations are an integral method in the computational studies of materials. This thesis discusses an algorithm for large-scale MD simulations using modern multi- and many-core systems on distributed computing networks. In order to utilize the full processing power of these systems, algorithms must be updated to account for newer hardware, such as the many-core Intel Xeon Phi co-processor.

The hybrid method is a data-parallel method of parallelization which combines spatial decomposition using the Message Passing Interface (MPI) to distribute the system onto multiple nodes, along with the cell-task method used for task based parallelism on each node. This allows for the improved performance of task based parallelism on single compute nodes in addition to the benefit of distributed computing allowed by MPI.

Results from benchmark simulations on Intel Xeon multi-core processors, and Intel Xeon Phi coprocessors are presented. Results show that the hybrid method provides better performance than either spatial decomposition or cell-task methods alone on single nodes, and that the hybrid method outperforms the spatial decomposition method on multiple nodes, on a variety of system configurations.

# Acknowledgements

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca) and Compute/Calcul Canada.

Computations were made on the supercomputer Guillimin from McGill University, managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), NanoQuébec, RMGA and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT).

A special thank you to Aaron Langille, for the constant nagging for me to get my work done. Told you I'd put you in here.

To Lorrie Fava for taking the time to review this work, and for the suggestions.

To Dr. R. Meyer, whose experience and knowledge helped me through many problems along the way. If not for his support, and the opportunity he's given me, this work would not have been possible.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Molecular dynamics is a computer simulation method that is widely used in computational physics, chemistry and material sciences. The method is described in detail by Allen and Tildesley [1], and Frenkel and Smit [2]. Since molecular dynamics is frequently used to perform large-scale simulations on high-performance computers, it is important to develop molecular dynamics algorithms that make the best use of modern computing architectures.

Molecular dynamics simulations compute the trajectories of a set of interacting particles. This requires a model for the forces (or interactions) between particles. The simulation is able to examine interactions between small items such as atoms, or large items such as planets, so long as an appropriate interaction model is supplied. If a short-range force model is used and the simulated system is sufficiently homogeneous, the spatial decomposition method [3] provides an effective means for the parallelization of the simulations. For inhomogeneous systems, an alternative parallelization scheme named the cell-task method has recently been proposed for shared-memory systems [4, 5, 6].

This thesis discusses a hybrid method that uses a two-level approach for the parallelization of molecular dynamics simulations. The first level is based on the spatial decomposition method and is implemented with the Message-Passing Interface (MPI) [7]. The second parallelization level employs the cell-task method for the parallelization of the workload within the spatial domains and is implemented with Intel's Threading Building Blocks Library [8].

The primary rationale for the implementation of the hybrid method is that it extends the range of the cell-task method to more than one compute node. However, even on a single node, the hybrid approach can be advantageous. While the cell-task method is more efficient for inhomogeneous systems, the situation is less clear for homogeneous systems where spatial decomposition works well. In this case, the situation depends on system details since the overhead of the task management in the cell-task method competes with the communication overhead of the spatial decomposition method. Further, the spatial decomposition approach may have a slight advantage through a more localized memory access pattern. A hybrid approach can lead to performance enhancements as it allows the use of both the cell-task method and spatial decomposition.

This thesis examines the hybrid approach, and compares it to the spatial decomposition and cell-task methods, in order to gauge the performance gains. This is important within molecular dynamics simulations, in order to reduce the amount of time required to simulate ever larger systems, without the need to wait weeks, or months for the results.

This simulation code used in this thesis to implement the hybrid method has previously been used in several research projects and run for many millions of CPU hours. Some examples of simulations running for weeks or months are described in [9, 10, 11]. The purpose of this work is to take a simulation, and through hybridization improve the performance in order to reduce the overall running time of the simulation. Not only is hybridization relevant to the simulation being worked on, it can also be expanded to other fields, such as computationally expensive simulations, image processing, and many other areas.

# Chapter 2

# Background

This chapter will introduce some background information on molecular dynamics simulations. This will include

- the main computational tasks,
- traditional parallelization methods,
- interaction potentials, and
- related work.

## 2.1 Main Computational Tasks

Molecular dynamics simulations employ three main computational tasks which account for the largest portion of time spent within the simulation. These tasks include

- the update of particle positions and velocities,
- the calculation of forces on particles, and
- for short range forces, the calculation and construction of neighbour lists.

These tasks account for roughly 90% of the time spent in a molecular dynamics simulation, with the force calculations taking the most amount of time out of the three, as determined by using performance profiling on a single core. The update of particle positions and velocities as well as the force calculations are completed at every time step, however the construction of neighbour lists is only done after a specified number of time steps.

Molecular dynamics simulations integrate Newton's equations of motion in order to calculate the trajectories of particles. By taking advantage of Newton's third law of motion, which states that for every exuded force there's an equal and opposite force applied, the number of calculations can effectively be cut in half. In molecular dynamics simulations, a particle's neighbours consist of nearby particles which exude a force upon the particle. The force calculated on a particle by one of its neighbouring particles can be applied as a negative force to the neighbour.

Molecular dynamics simulations must calculate the forces on particles within the simulation system in order to calculate the velocity and position of each particle. For molecular dynamics simulations, these forces can be calculated using different interaction potentials, described in section 2.3. For short range forces, a particle's force upon another is considered to be zero once the distance between particles exceeds a certain point. Depending upon the interaction potential used, the force between particles may approach zero as the particles get to this specified distance, or a sharp cut-off may be used where the force simply drops to zero at the specified distance. The force calculation process is described in section 3.6.2.

Neighbour lists are created by selecting a particle and looping over all other particles and determining if that particle is within a certain range. This distance is slightly larger than the specified cut-off for short range forces as the neighbour lists are not rebuilt at every time step. This allows for particles which move within range of another particle's range of influence to have an effect on each other, and reduces the need to rebuild the neighbour lists at every time step. This must be completed for all particles within the simulation. For optimization purposes, if a particle $i$ is in the neighbour list of particle $j$ then particle $j$ will not be in the neighbour list of particle $i$, since by taking advantage of Newton's third law the calculations of forces are only required on one particle which updates the other particle.

## 2.2 Traditional Parallelization Methods

Several techniques currently exist for the parallelization of molecular dynamics simulations. Each method is similar in that they each attempt to distribute the workload in order to improve the performance of the simulation, but differ in key aspects. These differences offer advantages and disadvantages to the different parallelization techniques. This section will briefly introduce each parallelization technique, and their advantages and disadvantages.

### 2.2.1 Particle Decomposition

Particle decomposition is a method which divides the particles within the simulation system evenly amongst the available processing nodes [3]. Each node is then responsible for updating positions, calculating forces, and updates of their own particles. This has an advantage of an even distribution of particles between nodes on any type of simulation system, and helps to improve the performance and results in good load balancing.

This parallelization technique, however, does not guarantee that particles and their neighbours are on the same node. This therefore requires additional communication between nodes in order to transfer all relevant data required to perform the simulation accurately, although typically all positions are broadcast to all nodes. This additional communication causes processing overhead which can degrade the performance of the simulation.

### 2.2.2 Force Decomposition

Force decomposition is different than most traditional parallelization techniques of decomposition. Instead of dividing the system by particles, the system is partitioned into a block matrix [3]. Since forces must be calculated on all particles in the system, the forces can be written as a matrix $F$ where each element $F_{ij}$ is the force of particle $j$ on particle $i$. This matrix can then be blocked into sub-matrices and each block can be distributed to a compute node, which calculates the forces contained within its block of the matrix.

Force decomposition has an advantage of removing some communication between nodes, as each node only requires knowledge of the particles which they process, although since particles can appear on multiple nodes there is still some required communication. This method tends to work well for long range forces where the other decomposition methods would require an excessive amount of communication.

Conversely, it would be difficult for the force decomposition method to take advantage of Newton's third law, and hence would require more computations. While simulations can be optimized for this, it would require increased communication between nodes, which takes away from its advantages.

### 2.2.3 Spatial Decomposition

The spatial decomposition [3] method involves decomposing the simulation system into the number of spatial domains specified, which represent three-dimensional sections of the simulated system, and are processed by individual compute nodes. These domains are of equal physical size and shape to each other, and the method can be applied in the $x$, $y$, and $z$ directions.

Throughout the simulation, in order to maintain accuracy, information must be communicated between the compute nodes. This information includes, but is not limited to, the particle locations, velocities, and accelerations. This information is used by other spatial domains to compute the forces applied on its own particles which interact with particles in neighbouring domains. Further, the entire particle must be transmitted to another compute node once the particle leaves that node's spatial domain.

This type of decomposition works well with short range forces as particles and their neighbours are generally contained within the same spatial domain, and hence on the same compute node. Particles which appear near the borders of such spatial domains, however, will have particles in neighbouring domains and hence those particles' information will need to be exchanged. This is generally a minimal amount of communication compared to particle or force decomposition, although as more domains are added, the amount of data communicated is also increased. This is offset, however, by the further parallelization of the simulation.

For systems which have uneven distributions of particles, spatial decomposition may perform poorly. As particles are distributed to processors based upon their location within the simulation system, if certain spatial domains contain fewer particles than others, then a load imbalance will occur, which can degrade the performance of the simulation.

### 2.2.4 Cell-Task Method

System-level thread pool parallelization implementations can use basic approaches, such as parallel for loops, or more advanced parallelization techniques such as task based parallelism. An inherent problem with parallelization of a molecular dynamics simulation using threading is race conditions. Race conditions occur when two or more threads are attempting to update a single particle's data. This can be avoided by using *critical* sections or *atomic* updates; however, these methods would require thread blocking, which often degrades the performance of the system.

Throughout this work, a task is used to describe small portions of the simulation system which require processing in order to calculate the forces on particles which occur within those portions. These portions of the simulation system are made up of one or more cells, which are sub-domains of the full simulation system, typically the width of the specified cut-off distance. These tasks are used to calculate the forces on the particles and to construct neighbour lists.

The cell-task method [4] avoids this issue by using a dynamic scheduling algorithm which schedules tasks in a fashion which prevents two tasks which contain the same particles from running simultaneously. In order to reduce the number of cores waiting for work, the simulation system is split into thousands of smaller tasks, which allows for several hundred tasks to run simultaneously without causing cache coherency issues. This method is highly dependent upon short-range interactions for the purposes of building the task lists. Details of this process are described by Meyer [4].

Since the dynamic scheduler prevents two tasks which access the same particle from running simultaneously, critical sections can be avoided. In order to avoid the load-balancing issue which can occur with spatial decomposition, this method's dynamic scheduler will automatically run the next available task once a previous task is complete. Further, tasks which have no particles within can simply be skipped. Despite these advantages, in order for the system to remain accurate, the task lists must be updated when the neighbour-lists are rebuilt, which adds overhead to the process.

This method's dynamic scheduler works well for inhomogeneous systems due to its automated load balancing. However, this method has a disadvantage of requiring the creation of the dynamic scheduler, which is an intensive process. Further, this method is limited to a single compute node due to its use of thread pool parallelization.

## 2.3 Interaction Potentials

This section will briefly describe the three interaction potentials used for this work. Each of the potentials offers varying degrees of processor and memory strain, and are each therefore important to test with, to determine where limitations may lie.

$$E_i = F\left(\sum_{i \neq j} \rho(r_{ij})\right) + \frac{1}{2}\sum_{i \neq j} \phi(r_{ij}) \tag{2.1}$$

The interaction models follow a general pattern for the interactions of particles which can be found in the embedded atom method (EAM), which is shown in Equation 2.1 [12]. While none of the potentials used throughout this work use EAM, the format remains the same.

### 2.3.1 Lennard-Jones Potential

The Cutted Lennard-Jones potential is used for a silver system in this work. This potential is the simplest potential used, with reduced calculations involving $\phi$, which is shown in Equation 2.2.

$$\phi(r_{ij}) = \begin{cases} 4\varepsilon \times \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + c\right) & \text{if } r \leq r_{cut} \\ 0 & \text{if } r > r_{cut} \end{cases} \tag{2.2}$$

In this, $\varepsilon$ represents the energy scale, $\sigma$ a scale where the resulting value from the function equals zero, $r$ represents the distance between particles $i$ and $j$, and $c$ represents the shift in the function in order to have the equation approach zero at $r_{cut}$, which is typical in the cutted version of the Lennard-Jones potential. The simplicity of the potential allows for fast calculations which removes strain upon the processor but also creates a large strain on the memory. This means that this potential will be more affected by memory access latencies, and less so by processor speeds. This type of potential, due to its simplicity, cannot be fitted for properties of any real materials; however it can still be used for examination of general phenomena.

### 2.3.2 Tight-Binding Potential

The Tight-Binding potential is often used to simulate many metals in molecular dynamics simulations. Throughout this work, it is used on several copper systems. This potential has moderate calculations, which can be seen in Equations 2.3 and 2.4.

$$\rho(r_{ij}) = \begin{cases} C_{elec} \times e^{Z_{elec} \times r} & \text{if } r \leq r_{cut} \\ 0 & \text{if } r > r_{cut} \end{cases} \tag{2.3}$$

In these, $r$ represents the distance between the particles $i$ and $j$, $r_{cut}$ represents the cut-off distance, $C_{elec}$ and $Z_{elec}$ are constants which describe the electronic system parameters, $C_{rep}$ and $Z_{rep}$ represent the Born-Mayer repulsion parameters, and $F'$ is the derivative of the embedding function for particle $i$ with respect to *rho*. The values for the energy potential are included in the calculation of $\phi$ and are stored in appropriate variables.

$$\phi(r_{ij}) = \begin{cases} \frac{2 \times Z_{rep} \times \left( C_{rep} * e^{Z_{rep} * r} \right) + F' \times \left( Z_{elec} \times C_{elec} \times e^{Z_{elec} * r} \right)}{r} & \text{if } r \leq r_{cut} \\ 0 & \text{if } r > r_{cut} \end{cases} \tag{2.4}$$

The tight-binding potential is important to examine due to its usefulness on many types of materials. Further, its moderate calculations puts some strain on memory accesses, but also upon the processor, both of which can affect the performance of simulations.

### 2.3.3 Mendelev Potential

The Mendelev potential is used for this work for the examination of the interactions between iron atoms. This potential is far more complex than the tight binding potential in both the $\rho$ and $\phi$ calculations. The formulas for these calculations are complex and hence are omitted, but can be found in the Appendix of Ref. [13]. These calculations use up to 15 cases in a piece-wise fashion, which makes vectorization difficult, further increasing the processor strain.

The complexity of these calculations reduces the strain upon the memory access of the simulation, but increases the strain upon the processor. This can show potential areas of interest in the simulation's limitations.

## 2.4 Related Work

A hybrid approach to molecular-dynamics simulations using the cell-task and spatial decomposition methods has not been previously been developed, to the best knowledge of the author. This section will introduce some related methods which use different parallelization techniques.

A method similar to the cell-task method used for this work was developed by Ackland et al. [14]. This method uses thread locking and unlocking to prevent race condition issues while maintaining a thread-pool parallelization scheme. In this method, a parent thread locks the cell of particle $i$ which it is calculating the forces upon, which prevents any other thread from accessing particles within this cell. It also locks the surrounding cell of its neighbouring particle $j$, and unlocks that cell when the neighbouring particle is no longer in that cell. This requires a thread lock on only one neighbouring cell at a time, compared to preventing access on 26 neighbouring cells used by the cell-task method, which prevents any task from running which may use the same particles as another task.

A similar hybrid method was used by Pal et al. [15], which used MPI and OpenMP, using the particle decomposition method. Further, the work done by Pal does not take advantage of Newton's Third Law of motion. This means that the work required by this hybrid method is effectively double that required through the proposed hybrid method with spatial decomposition. The results do indicate the hybrid method proposed by Pal does outperform using particle decomposition alone, by approximately a factor of two on six dual-quad-core compute nodes, totalling forty-eight cores.

A recent publication from Needham et al. [16] uses the Intel Xeon Phi co-processor, discussed in 3.5.1, with the AMBER Molecular Dynamics software package in an offload fashion. For offload processing, only a small porting of the code is passed onto the co-processor, as opposed to the entire program being run by the co-processor. Needham's method uses both spatial decomposition to distribute the workload onto two host processors, and then offloads work to the Xeon Phi co-processor which uses a threading method for some computations. For the simulation systems tested by Needham, his hybrid method achieved a speedup factor of up to 1.62, compared to a baseline run using only the two host processors.

Early tests with the Xeon Phi using the LAMMPS software done by Willmore [17] favoured negatively, compared to using only host processors or in combination with the host, in fact creating slower simulations. Work done by Plotnikov [18] and Gopalan [19], however, show improved performance when using both the host and co-processor in conjunction with each other.

Little information is given on the work done by either Plotnikov or Gopalan, and only some information is given in relation to the work done by Willmore. Willmore did use the USER-OMP [20] package with LAMMPS, which nests OpenMP parallelization inside MPI parallelism, creating a hybrid method on its own. MPI is used for the spatial decomposition done by LAMMPS, and OpenMP is used for thread level parallelization of the pair calculations over atoms in an "MPI-like Approach."

Willmore compared his results of both the many-core and multi-core machines with which his tests were run. Consistently, Willmore was able to show that the performance of the Xeon Phi was significantly slower than that of the multi-core machine. He also found that using 1, 2, and 4 Xeon Phis did improve results, however going beyond 4 Xeon Phis resulted in slower performance. Further, Willmore only tested on the many-core architecture using smaller systems, which included a 2 million atom system.

# Chapter 3

# Methodology

The methodology involved in this work consists of modifications to an existing molecular-dynamics simulation program. This chapter will describe

- the simulation program being used,
- the configurations used for testing,
- the hardware the simulation is running on,
- the simulation program components and modifications required to the simulation, and
- the analysis process.

## 3.1 Simulation Program

The program used for this work is a molecular dynamics (MD) simulation program written in C and C++ used in the research of material sciences. The existing code base contains the cell-task implementation using Intel Threading Building Blocks, as well as an unused spatial decomposition method implemented with the Message Passing Interface (MPI) from a previous iteration of the program code. This simulation method's time intensive parts have been SIMD vectorized [21] to optimize CPU performance. The focus of this work is placed upon the combination and improvement of the existing spatial decomposition code with the cell-task code, with no changes to other sections of the code which are not directly related to these approaches.

The simulation can use multiple short-ranged types of potentials (many-body and pair-potential), and uses a cut-off distance, $r_{cut}$, specified in a configuration file. For many-body potentials, the main time consuming section of the simulation employs several steps in order to calculate the values required to appropriately simulate the interaction between particles. In the first step a localization function, $\rho$, is calculated for each atom and its neighbours.

The second step is to calculate the embedding function $F$ and its derivative $F'$, after which a separate function, $\phi$, is used to calculate the pair-potential and complete the calculation of the forces on the atoms within the current domain. This information is used to update the atoms' accelerations, velocities, and locations at the end of each simulation step. The process is described in detail in section 3.6.2.

## 3.2 Spatial Decomposition and the Message Passing Interface

For this work the Message Passing Interface (MPI) is used. MPI is a commonly used interface allowing for applications to be parallelized on a single compute node, or multiple compute nodes. The main advantage of the MPI interface is its ability to distribute work loads amongst multiple compute nodes, which allows for the full processing power of each compute node to be dedicated towards a portion of the simulation's system. However, as most compute nodes today use multiple cores this can lead to several problems.

While MPI is fully capable of distributing work loads amongst cores on each compute node, the simulation system must first be broken down into smaller domains using the spatial decomposition method. This may not always have adverse effects, however for systems which have particles which are not equally distributed this can lead to severe load balancing issues in which some CPU cores may have few to no particles to process, whilst others have thousands or even millions of particles. Further, as the simulation system is split into smaller domains, there is an increased overhead in the communication. An advantage to MPI is its inherent use of multiple processes. Since MPI launches distinct applications on single or multiple compute nodes, each application has its own section of memory dedicated to it. This allows for better read and write times within memory, which is generally more localized.

An inherent disadvantage of MPI is the inter-process communication required. In order to properly simulate the particle interactions of the system, particle information such as locations and velocities must be communicated between the processes. While the number of particles and their respective data is limited, due to the use of short-range interactions, there is still a significant amount of data that needs to be transferred. Further, once a particle gets moved outside of the current node's domain, as determined by the spatial decomposition method, the entire particle structure needs to be moved to another domain, which further increases the amount of inter-process communication.

In general, the number of particles requiring transport to other nodes is limited, however on fluidic systems the particles are allowed to freely move about, which increases the number of particles requiring transport. This can have a detrimental effect on the simulation's performance for these systems. While this is reduced by only transporting particles when the neighbour-lists are rebuilt, the effect is still noticeable.

The spatial-decomposition method's main use is within the force calculations. During these steps, described in section 3.6.2, each processor-core is responsible for calculating forces on particles in its domain. This allows for simultaneous computations on multiple particles during the most time-intensive section of the simulation.

## 3.3 Task-Based Parallelism and Intel Threading Building Blocks

Intel's Threading Building Blocks (TBB) [8] is a C++ library allowing for system-level parallelization using a thread-pool shared memory machine, and is used in this simulation for the cell-task method. The main advantage to TBB, and system-level parallelization in general, is its use of shared memory, which removes communication between nodes.

TBB has a limitation in that it is only able to run on a single computer node; hence you can only have as many threads running simultaneously as the number of processor cores available on the system. Conversely, MPI is limited to the total number of processor cores on each machine within a cluster, allowing for many more simultaneous processes to run in parallel. This limitation is a rationale for this work, in order to distribute the cell-task method onto other nodes, thereby making more processors available for the simulation.

## 3.4 Simulation Configurations

The simulation configurations consist of all particles within the material being simulated. The files for the configurations consist of the particle locations, velocities, accelerations, the type of particle, and other relevant information. This information is required in order to properly simulate the system. For the purposes of testing, a variety of different configurations were used in order to ensure that the changes work well on multiple types of systems. Most systems used for testing use a periodic boundary condition, meaning that when a particle goes out of the simulation area on one end, it will re-enter on the other end of the system. The $Cu(Spheres)$ system, however, does not use periodic boundary conditions, and the honeycomb systems are periodic in only the $x$ and $y$ directions, but not $z$. This section introduces the simulation configurations used, and graphical representations of some of the systems. The systems shown were created with a custom software provided by the supervisor.

These configurations consist of both homogeneous and inhomogeneous systems. Homogeneous systems consist of particles which are evenly distributed throughout the system, and is the most common type of configuration used for this work. Several homogeneous systems are used for testing including bulk, liquid, spherical, and honeycomb shaped systems. A bulk system consists of particles arranged in a periodic fashion encompassing the entire simulation area, an example of such a system can be seen in Figure 3.1. While a spherical system and a honeycomb system also have a regular crystalline arrangement of atoms, they do not encompass the entire system, and instead only take up part of the system, however in the sections of the system they do encompass, they would be closely packed, as seen in Figures 3.3 and 3.4. A liquid system is similar to that of the bulk system, however its particles are capable of moving more freely within the system, compared to a solid form.

Inhomogeneous systems, however, consist of an inconsistent distribution of particles within the system. This can be seen in the porous system of Figure 3.2, wherein the particles are spread throughout the system. Having a porous system, with holes throughout the system, does not make the system inhomogeneous, instead the inhomogeneity is caused by the uneven distribution of particles within the space. The sphere and honeycomb system are also considered to be inhomogeneous, however their particles are arranged in a fashion which can more easily allow for the even distribution of particles to nodes.

### 3.4.1 Copper (Cu) Systems

Three separate bulk copper systems were used consisting of different numbers of particles. The smallest, *Cu63*, consists of 1,000,188 particles, followed by the mid-size, *Cu105*, with 4,630,500 particles, and lastly a larger system, *Cu126*, consisting of 8,001,504 particles. The different sizes allow for the observation of growing the simulation size on the effect of time required to complete the simulation. The layout of these can be seen in Figure 3.1. This type of bulk system is well suited for spatial decomposition due to the even distribution of particles within the system.

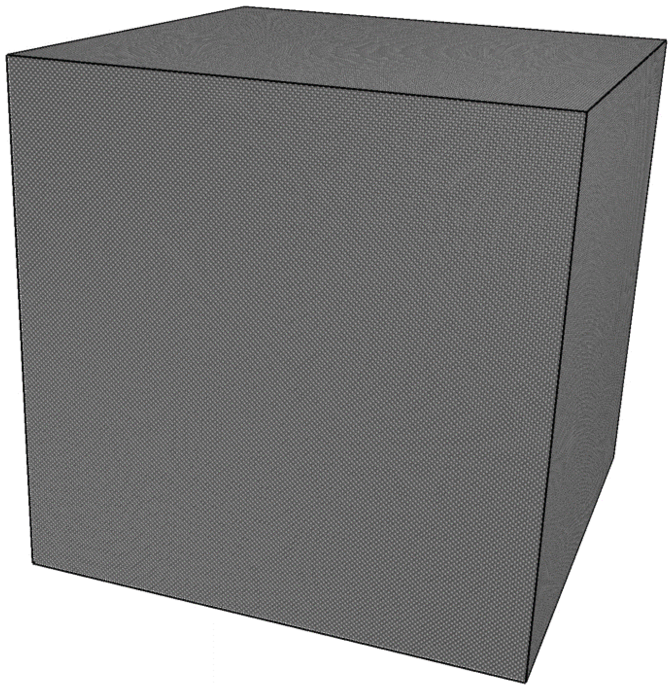

Figure 3.1: The *Cu63* bulk copper system, consisting of approximately 1 million atoms.

Another copper system used, *Cu (porous)*, is unlike the other systems, as it is a porous system consisting of 1,992,220 particles which fill only a fraction of the available volume of space. This type of porous system is well suited for the cell-task method, due to the inhomogeneity, as seen in Figure 3.2, which degrades the performance of the spatial decomposition method, caused by the uneven distribution of particles amongst spatial domains.

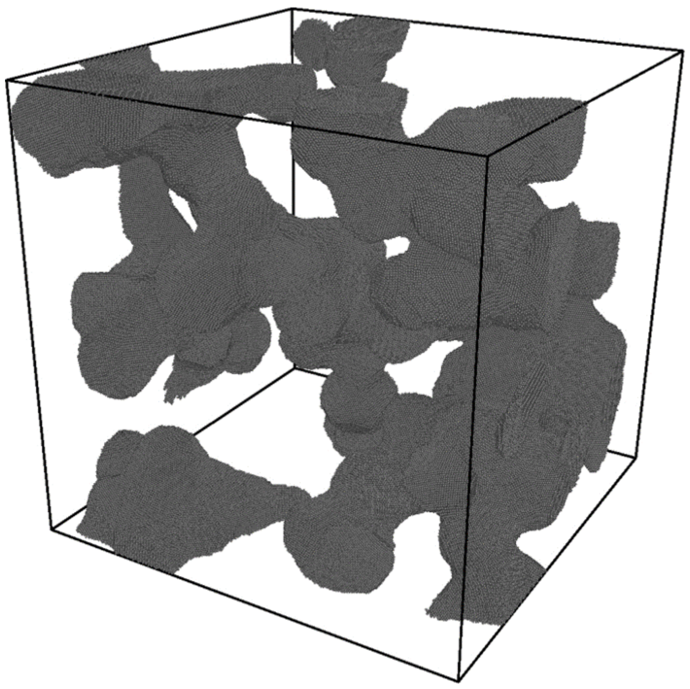

Figure 3.2: The *Cu (porous)* system, consisting of approximately 2 million atoms.

Another set of systems consisting of two separate spheres of copper were also used for testing, *Cu (spheres)*. These two systems contain 2,354,302 and 9,261,150 particles respectively. This type of system has an advantage in removing all communication in the between the two spheres, as the two sets of copper spheres' particles do not interact with each other. The smaller of these systems is depicted in Figure 3.3.

Lastly, four additional systems which are configured in a honeycomb pattern were also used. These four systems consist of 2,023,216 (*Cu (honeycomb, 2x4)*), 4,046,432 (*Cu (honeycomb, 4x4)*), 8,089,576 (*Cu (honeycomb, 4x8)*), and 16,179,152 (*Cu (honeycomb, 8x8)*) particles. As each of these systems have the same structure, just doubling the sizes, these are important to determine what happens as system sizes are doubled as well as the number of processors working on the simulation – in the best case, this would be a linear growth, meaning that time and speedup should

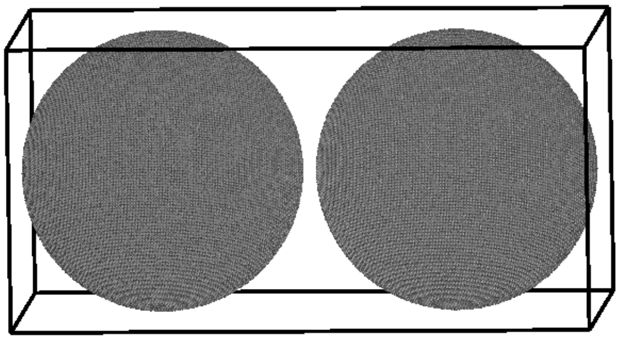

Figure 3.3: The *Cu (spheres)* system, consisting of approximately 2.3 million atoms.

be constant. The smallest of these systems, *Cu (honeycomb, 2x4)*, is depicted in Figure 3.4.

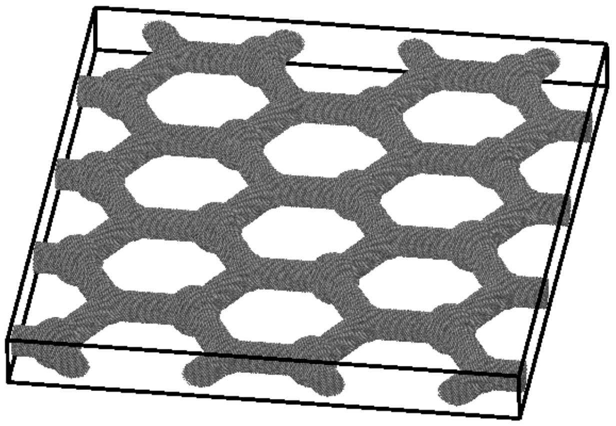

Figure 3.4: The *Cu (honeycomb, 2x4)* system, consisting of approximately 2 million atoms.

The copper systems each use a vectorized implementation [21, 6] of the Tight-Binding Potential [22]. This type of potential is of medium complexity, which allows for a large portion of the time spent to be focused on the calculations, but also has some focus on the communication time required. Each of these systems are set to 300 degrees Kelvin, approximately room temperature.

### 3.4.2 Iron (Fe) System

One system consisting of iron atoms, *Fe (bulk)*, containing 4,000,752 particles is used with the Mendelev potential [13]. This system was used since the potential has more complex force calculations, and is therefore less susceptible to memory access latencies than the Tight-Binding potential which is simpler and therefore more affected by memory access speed. Comparison of the iron system to the bulk copper systems therefore allows to judge the influence of the memory access speed on the results. This system can be depicted similarly to that of the bulk copper systems. This iron system is also set to 300 degrees Kelvin.

### 3.4.3 Silver (Ag) System

The last type of system used is a liquid silver system, *Ag (liquid)*, which employs the Lennard-Jones potential [1]. This type of potential is relatively simple, and hence has fast force calculations, making it highly sensitive to memory access speeds. This system also consists of 4,000,752 particles and is depicted in Figure 3.5; however, unlike the other systems, it is heated to 6,210.59 degrees Kelvin, in order to turn the silver system into a liquid. This liquid state increases the movement of particles, and in turn increases the communication between spatial domains as particles move between domains more frequently.

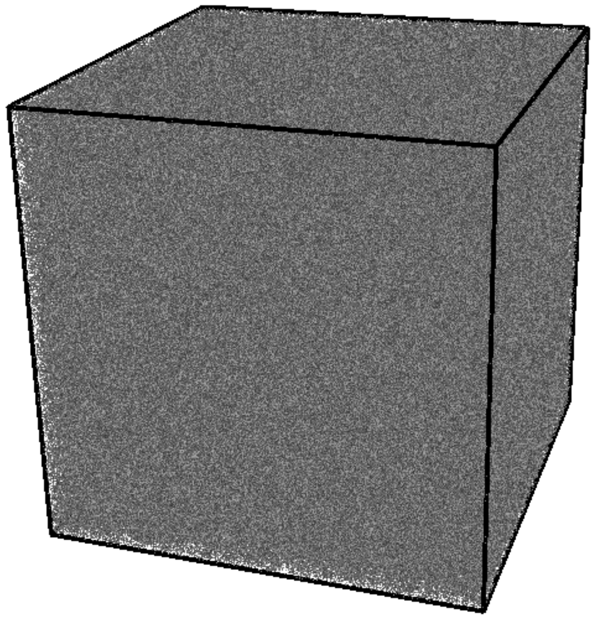

Figure 3.5: The Ag *(liquid)* system, consisting of approximately 4 million atoms.

## 3.5 Hardware

Two separate machine types were used for the purposes of this work. Table 3.1 contains a brief overview of each test system, with each containing dual processors. The two systems were chosen in order to test the methods on current multi-core technology, and the up-and-coming many-core processor technology. The Xeon Phi is an example of the many-core technology, and is described in detail in section 3.5.1.

The multi-core processor uses shared memory for inter-process communication, even amongst the processors on each node; however, the Xeon Phi processors use shared memory only for inter-process communication on the individual Phi. The Xeon Phi relies on the PCIe bus to communicate to the second Xeon Phi located on the same node. Both the multi-core processor and Xeon Phi systems use QDR InfiniBand to communicate with processors located on other nodes. The Xeon Phi, due to it being a co-processor, is located on a node which also contains a host processor which can also be used.

| | **Multi-Core** | **Xeon Phi** |
|---|---|---|
| **Processors** | 2 × Intel Xeon E5-2680 @ 2.7GHz | Intel Phi 5110P @ 1.053GHz |
| **Number of cores (per processor)** | 8 | 60 |
| **Hardware Threads (per core)** | 1 | 4 |
| **Virtual Threads (per hardware thread)** | 2 | 1 |
| **Total Threads** | 32 | 240 |
| **Total Memory** | 64GiB | 8GiB |
| **Connection Type** | QDR InfiniBand | QDR InfiniBand |

Table 3.1: Brief overview of the systems' hardware.

Tests done on the multi-core Xeon processor consider both processors on the same compute node to be one processor, as the communication between them still uses a shared memory architecture. For the Xeon Phi, however, each Xeon Phi is considered to be a separate system, as they do not communicate through shared memory.

### 3.5.1 Intel Xeon Phi

The Intel Xeon Phi is a coprocessor which runs alongside the host system's processor, in order to accelerate performance. It consists of 60 physical processor cores, L1 and L2 cache, GDDR5 memory controllers with on-board memory, and PCIe logic in order to be connected to the system through a PCI Express slot. The Xeon Phi processor is based on the Pentium P54c processor, with modifications to allow for 64-bit instructions, vector units, and multi-threading.

Due to the Xeon Phi being based upon the Pentium P54c processor, the Xeon Phi is an in-order processor, meaning that new instructions will be delayed until all previous instructions which the new instruction depends upon are completed. This can lead to pipeline stalls, and in effect slower processing. The Xeon Phi attempts to make up for this by using four hardware threads per core for simultaneous multi-threading.

The Xeon Phi uses time-multiplexed multi-threading with a thread picker to determine which instruction should be sent to the Arithmetic Logic Unit (ALU), based on a round-robin mechanism to avoid threads which are inactive due to memory stalls, waiting for previous instructions to complete, or other conditions which are required to be completed prior to further instructions being processed [23].

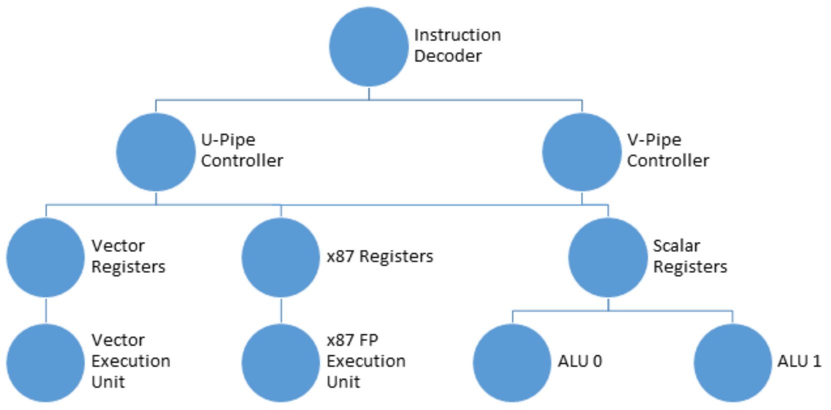

Figure 3.6: Intel Xeon Phi Core Micro-architecture.

The Xeon Phi uses two separate pipelines on each core, the U-pipe and the V-pipe, although the V-pipe can only execute a subset of the U-pipe instructions. However, as both pipes can communicate with the ALUs, each core is able to execute two separate instructions per clock cycle [23].

The coprocessor's cores each consist of three separate processing units, the scalar processor, the x87 floating point unit (FPU), and the vector processing unit (VPU), as seen in Figure 3.6. The FPU works as normal in other systems, capable of doing arithmetic on floating point values, however, the scalar unit consists of two separate ALUs in order to complete two operations per clock cycle. The VPU has a separate pipeline, seen in Figure 3.7, once the instruction reaches the write back stage of the core pipeline.

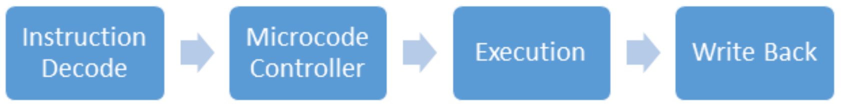

Figure 3.7: Intel Xeon Phi Core Pipeline Stages.

### Intel Xeon Phi Vector Processing Unit

Due to the number of arithmetic calculations required for molecular dynamics simulations, the Vector Processor Unit (VPU) of the Xeon Phi becomes extremely important. The VPU receives its data from the L1 cache through a dedicated 512-bit bus, and is capable of communicating to the core to stall as necessary.

The vector processor is a 512-bit Single Instruction Multiple Data (SIMD) unit, capable of working on up to 16 single precision (SP) floating point values, or 8 double precision (DP) floating point values simultaneously. The unit can process one load and an operation in the same cycle, with ternary operands - two source, and one destination. Each VPU consists of 8 UALUs, each containing 2 SP ALUs and one DP ALU.

Once the vector operation instructions reach the write back stage of the main pipeline, the VPU's pipeline takes over (Figure 3.8). In its first stage, E, the VPU detects any dependencies the current instruction requires, and will stall as necessary. The VC1/VC2 stage then completes shuffle and load conversions. In the V1-V4 stages the 4-cycle multiply/add instructions are executed, followed lastly by the write back stage, where the vector register data is written back to the cache.

At full capacity, the VPU pipeline has a 1-cycle throughput, with a 4-cycle latency. In essence, the VPU will run at maximum efficiency when four threads are running on each core, to account for the 4-cycle latency.

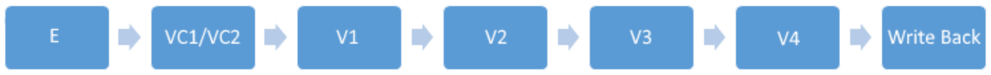

Figure 3.8: Intel Xeon Phi Vector Pipeline Stages.

The VPU also contains an Extended Math Unit (EMU) for single precision transcendental functions, such as exponential and square root functions. These instructions utilize Quadratic Minimax Polynomial approximations, and use lookup tables for fast approximations of the transcendental functions. However, utilizing the EMU has additional penalties for some functions in the latency, thereby decreasing the throughput for these functions to 2 cycles or greater, depending upon the function being used [24], see Table 3.2 for some examples.

| Instruction | Latency (cycles) | Throughput (cycles) |
|---|---|---|
| Exp | 8 | 2 |
| Log | 4 | 1 |
| Power | 16 | 4 |
| Sqrt | 8 | 2 |
| Div | 8 | 2 |

Table 3.2: Intel Xeon Phi Extended Math Unit Latency and Throughput.

### Intel Xeon Phi Limitations

While the Intel Xeon Phi coprocessor has the ability to perform many operations per second, it does have limitations due to its design. The design of the Xeon Phi's instruction decoder is for a two-cycle pipeline unit; hence, if only one thread is being used per core, then only 50% of the core's peak performance can be achieved. It is therefore beneficial, and required to get the best performance, to use two or more threads on each processor core.

The coprocessor also has a clock frequency of only 1.053 GHz, far lower than most modern processors. This lower frequency means that fewer operations are completed per second; however, the Many Integrated Cores (MIC) architecture of the Xeon Phi overcomes this limitation by containing 60 cores per processor. On a per-thread comparison, modern processors will outperform the Xeon Phi, however, at peak performance the Xeon Phi is capable of reaching 2.1 trillion floating point operations per second, far above that of modern processors [23].

Due to the Xeon Phi being based upon an older Pentium P54c processor, instructions are processed in order, as opposed to modern processors which use out of order architectures, as well as speculative execution. Compared to modern architectures, the in order execution negatively affects the performance of the Xeon Phi. The Xeon Phi also contains 8 GiB of on-board memory for the operating system in addition to mounting a home directory. This limits the amount of data capable of being stored on the system, thereby limiting the size of the configuration systems that can be utilized.

### Comparison with GPUs

A Graphics Processing Unit (GPU) is often used to improve performance of computationally expensive tasks, due to its many-core design being well suited for vectorized instructions. A GPU, however, has a limitation in being used only in an *offload* fashion - meaning that only some instructions are processed by the GPU, with the majority of instructions still processed on the host processor. While the Xeon Phi is cappable of being run in an offload mode, it is still a general purpose processor, so it does not have this limitation, and can process all instructions required for the program.

GPU programming also requires additional work for programmers in order to make their application compatible with the GPU. Depending upon the application, this could result in an extensive amount of code being added or re-written to use the application programming interface (API) for the GPU. Further, some of these APIs are proprietary and specific to certain brands of GPUs. The Xeon Phi again has the advantage of not requiring programmers to re-write their code to work for the specialized hardware, and instead only requires informing the compiler to generate the proper machine code for the instruction set.

Conversely, a GPU is advantageous in its specialization and focus towards vectorized instructions. GPUs can have hundreds or thousands of processor cores, each carrying out the same instruction on different pieces of data, which can have substantial benefits in sections of the code which perform the same operations on multiple pieces of data. Since MD simulations do require the same computations just on different data, GPUs can be applied to MD simulations as well.

For the purposes of this work, however, the Xeon Phi's general purpose design is better suited, as it allows the use of different parallelization techniques, as opposed to the limited vectorized approach which is used by GPUs.

## 3.6 Implementation

This section introduces the main components of the simulation code and some of the changes that were made to the existing code base, in order to accommodate the hybrid implementation. Some of these changes include the creation of an outer grid for the task scheduler, modified buffers for transferring data to nodes, and modifications to the force calculations.

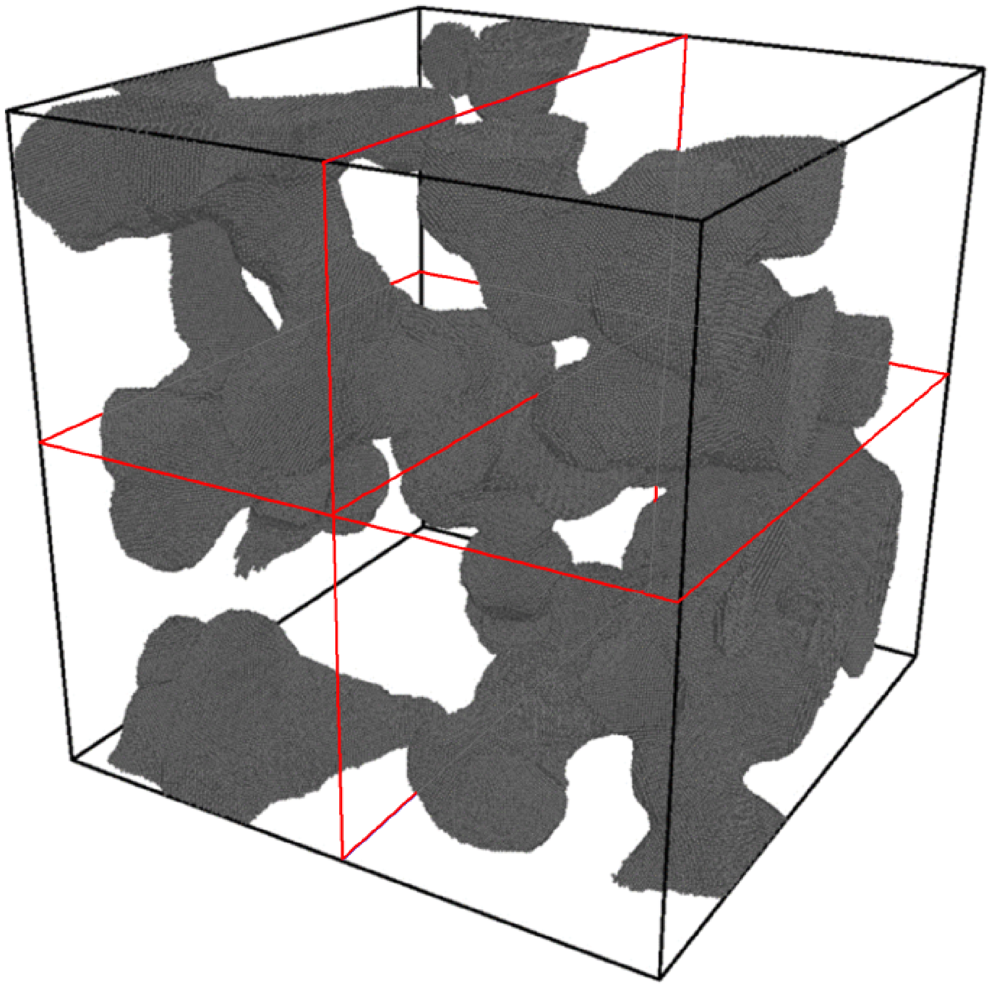

Figure 3.9: Example of the first step of dividing the $Cu(porous)$ system using the spatial decomposition method. The lines shown in red show a division of the system in a $2 \times 2$ formation.

The proposed hybrid method combines both the spatial decomposition method and the task based method. The simulation system is split into the specified number of domains and the particles are distributed according to the spatial decomposition method, shown in Figure 3.9, after which the cell-task method further divides the domains into smaller sections, shown in Figure 3.10, which can be scheduled and processed. This allows for each domain to be simulated by its own processor, and from there each processor core can work on separate tasks.

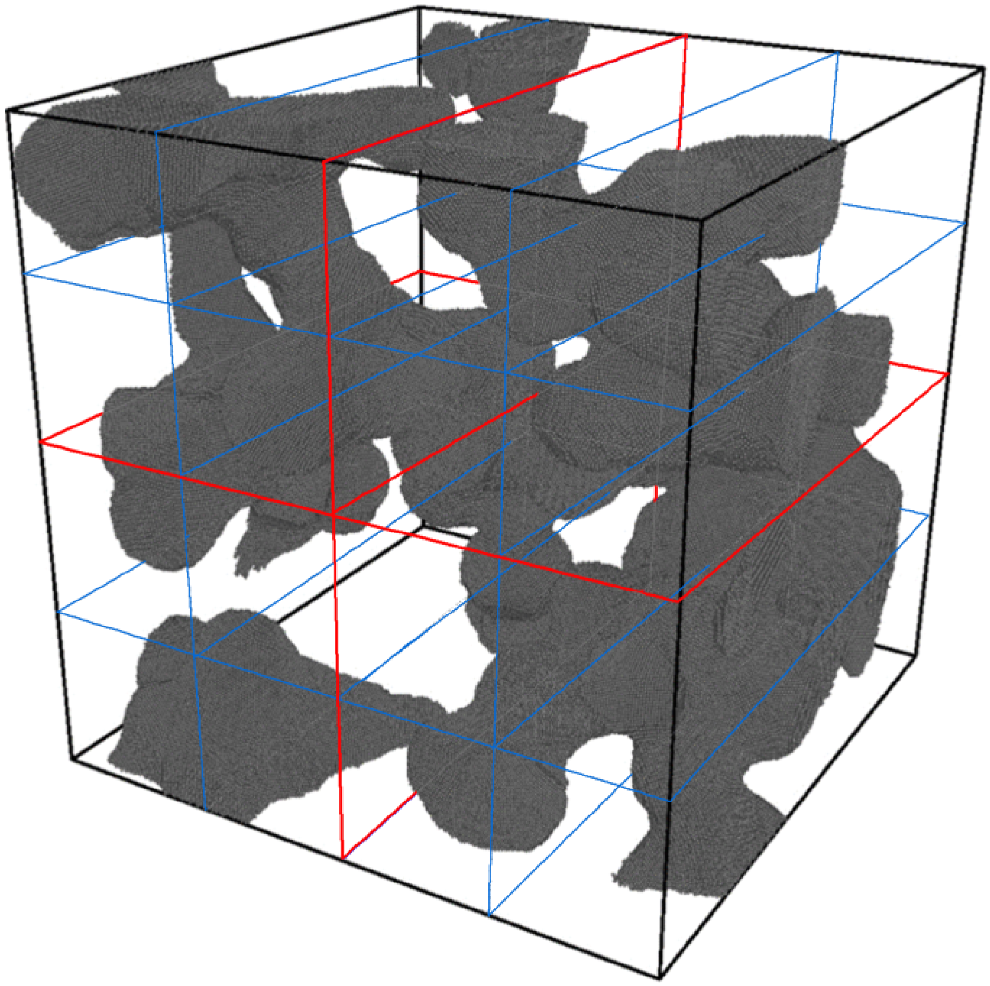

Figure 3.10: Example of the second step of dividing the $Cu(porous)$ system using the cell-task method. The lines shown in red show a division of the system in a $2 \times 2$ formation, followed by the blue lines signifying the divisions used by the cell-task method. Note that the cell-task splits the system into significantly more sections than shown.

This overcomes the limitation on the number of cores available that is inherent to the task based method. Further, the dynamic scheduling helps reduce load balancing issues inherent in the spatial decomposition method. This method uses both the Message Passing Interface and the Threading Building Blocks library in order to facility the parallelization. The MPI interface is used for the communication between processes, whereas TBB is used for threading within each process.

### 3.6.1 The Outer Grid

In the task based implementation, the simulation is split into thousands of smaller cells in a grid pattern. Depending on the simulation configuration settings, these cells are sometimes grouped together to reduce the total number of tasks. The scheduler then dynamically schedules each task in order to calculate the interactions upon each atom within the cells. The scheduler is designed to prevent any two running tasks from accessing the same particle structure, in order to prevent cache coherency issues or thread locking. A rudimentary algorithm as pseudo-code for the inner grid creation for the cell task method can be found in Algorithm 3.1, and the inner and outer grids creation for the hybrid method can be found in Algorithm 3.2.

**Algorithm 3.1** Pseudo-code of the creation of the inner grid for the cell task parallelism method

```
for $i \leftarrow 0$ to $innerGrid.size()$ do
    $innerGrid[i] \leftarrow nullptr$
end for
for all $particles$ do
    $dist \leftarrow distance\ from\ center$
    $idx \leftarrow calculateIndex(dist)$
    $innerGrid[idx]\ appends\ part$
end for
```

A secondary grid, known as the outer grid, is required for this work, which contains only the cells which contain particles which interact with particles in neighbouring domains. Limiting this grid to only relevant particles reduces the need to process each particle within the current domain, which is not necessary for the simulation. This outer grid works alongside the inner grid, which contains all particles located on the current node and is used to schedule the force calculations for all particles interactions occurring within the current sub-domain.

The creation of the outer grid is done when the neighbour-lists are rebuilt. During the construction of the inner grid, which is done using a parallel loop over a range of particles, a thread local copy of the domain data and its surrounding 26 neighbours is created. A thread local copy is used in order to prevent race conditions with the global data, and to prevent the use of thread locking – this prevents degradation of the performance and still allows for the parallel construction of the grid. This domain data is effectively a linked list of particles which occur on the node. It is split into 27 sections, where particles are sorted based on the neighbouring domains with which they communicate.

This data is filled with information in a step by step process. To begin, the current particle is examined in order to determine if the particle is considered to have passed outside the current domain limits. In this case, the particle has to be transferred to a neighbouring domain. Due to a generally minimal number of particles being transported to other domains, this storage of these particles is done using thread locking; however this does not significantly affect the performance for solid systems.

If the particle is not considered to have moved beyond the bounds of the domain, then the particle's current location is used in order to determine which of the surrounding 26 neighbouring domains the particle interacts with – this is for the transmission of coordinates and other relevant information to the other domains. Once this has been completed for all particles within the current domain, the thread local data is combined into one larger structure, by consecutively appending the links of each domain data to the next.

At this point, the outer grid is resized and all values within it are set to null pointers. This ensures that previous values, if any, are not accidentally reused when they should not be. The particles which have moved beyond the domain boundary then need to be exchanged with the appropriate domains. Due to the order of particle data being highly important in order to ensure that data is sent to the appropriate neighbouring domains, combined with a generally minimal amount of particles being transferred, this sending of the particles is done in a serial fashion. The particles are removed from the current list of particles in the current domain, then placed into a buffer. Once this has been completed for all particles in the domain requiring transport, the data is then sent to the appropriate neighbouring domains. The receiving of particles, unlike sending, is done in parallel. The particles are appended to the current domain's list of particles, then added to both the domain structure, and the inner grid. This can be done in parallel due to the incoming particle order not being important.

The domain data can now be finalized in order for appropriate data to be sent to appropriate nodes. A list of communication buffers is added to the buffer list of the domain structures. This is then followed by the creation of a list of particles, which is of equal size to the domain structure's buffers. This list is kept in the same order which the particles need to be sent to its neighbouring domains, although this does require atomic updates.

In order to send the types and coordinates, the send buffers are first filled with the appropriate data from the particles stored in the previously mentioned list of particles. This is done in order to fill the buffers in parallel in order to reduce the amount of time required preparing the buffers. Once the buffers have been filled, the data can be sent to the surrounding domains. This is done in an attempt to reduce the time required for the communication process.

The next step is to exchange the numbers of particles which are near enough to the borders of the sub-domain to interact with particles in neighbouring domains, which is necessary to allocate the appropriate send and receive buffers for all domains. At the same time, the particle coordinates are also sent to the corresponding domains. Once this is complete, the types (i.e. their chemical elements) of the particles are also exchanged. The types only need to be exchanged when the neighbour-lists are rebuilt, as the types of the particles cannot change, and the order of the particles can only change when the neighbour-lists are rebuilt.

Once the types have been sent, the outer grid can then be constructed. This construction is again done in parallel using threads. The particles are placed into the appropriate grid cell by using the coordinates received from the current domain's surrounding neighbours. Once the construction of the inner and outer grids is complete the inner scheduler can be created. This inner scheduler is the key component of the cell-task method which is used to schedule tasks in a way such that no two tasks which interact with the same particle are scheduled simultaneously. This is an important aspect, as it is used in the creation of the inner and outer neighbour-lists. The scheduler creation is detailed by Meyer [4].

An outer neighbour-list is needed to keep track of a particle's neighbours which occur in other spatial domains. This outer neighbour-list is built with the inner grid's scheduler. At this point, the

**Algorithm 3.2** Pseudo-code of the creation of the inner and outer grids for the hybrid parallelism method

```
for $i \leftarrow 0$ to $innerGrid.size()$ do
    $innerGrid[i] \leftarrow nullptr$
end for
for all $buffers$ do
    $reset\ buffer$
end for
for all $domains$ do
    $reset\ domain$
end for
for all $particles$ do
    $dist \leftarrow distance\ from\ center$
    if $particle\ is\ out\ of\ domain$ then
        $store\ to\ send\ to\ proper\ domain$
    else
        $dir \leftarrow associated\ domain$
        $link\ particle\ to\ domain[dir]$
        $idx \leftarrow calculateIndex(dist)$
        $append\ particle\ to\ innerGrid[idx]$
    end if
end for
for $i \leftarrow 0$ to $outerGrid.size()$ do
    $outerGrid[i] \leftarrow nullptr$
end for

$ExchangeDeadParticles()$
$Insert\ Buffers\ Into\ Areas$
$ExchangeParticleNumbers()$
$ExchangeParticleTypes()$

for all $Received\ particles$ do
    $dist \leftarrow distance\ from\ center$
    $idx \leftarrow calculateIndex(dist)$
    $outerGrid[idx]\ appends\ part$
end for
```

outer grid scheduler has not been created, however, it is beneficial to continue using the parallel code in order to improve performance.

The particles from all surrounding domains which interact with the particles in the current domain are first loaded into a buffer. Then, for each particle within the domain, an outer neighbour-list is generated using the particles currently in the buffer, if, and only if, the surrounding particle is within the range of the current particle. A counter is used to keep track of the number of particles added to the outer neighbour-list for each particle – if there are none, then the value is set to zero. These particle lists are then appended to the appropriate particle structures, which indicates where their outer neighbour-lists are.

The outer grid is then used to link all particles interacting with other domains. This allows for pointers to the next particle which contains links to particles in other domains to be added to the grid. The location of the particle is used to assign the particle to a specific cell within the grid. The outer scheduler can now be created, using the same methods as the inner scheduler, except it uses the outer grid to only schedule particles which have neighbours in other domains.

The schedulers are required because they ensure that no two tasks are running in parallel which would affect the same particle. Several functions in the simulation code take advantage of Newton's Third Law of Motion, which requires accessing and modifying the particle being examined and all of its neighbours. By taking advantage of this, the number of calculations is effectively cut in half.

### 3.6.2 Force Calculations

The calculations of the forces are dependent upon the type of interaction model being used. By following the pattern found in Equation 2.1, the potentials used throughout this work can all follow the same steps. The basic algorithm for many-body potentials can be found in the pseudo-code implementation of the cell task method in Algorithm 3.3 and the hybrid method in Algorithm 3.4.

For a many-body potential, such as the Mendelev or Tight-Binding Potentials, the coordinates of the particles near the border of the domain must first be exchanged with neighbouring domains. The $\rho$ function is then executed for all particles within the current domain, which gives information

**Algorithm 3.3** Pseudo-code of the force calculations of many-body potentials for the cell task method

```
for all particles do
    calculate $\rho$
end for
for all particles do
    calculate $F$ and $F'$
end for
for all particles do
    calculate $\phi$
end for
```

about how many and how close neighbours are to the current particle, which is summed up into a single value. These calculations are done using the tasks created for the inner grid in order to do the calculations in parallel.

The program will then wait until it receives particle data from its neighbouring domains. This wait is generally kept minimal due to the sending of data prior to the calculations of the $\rho$ function. Once the particle data is received from all neighbours, the calculation of $\rho$ is again done, this time using the outer scheduler. These values are again added to the single resulting value from the previous calculations.

These results are used to calculate the embedding function $F$. The derivatives of $F$ are then computed, which gives $F'$. This is done in parallel over blocks of particles within the entire domain, but does not require the scheduler as they only affect the particle currently being examined, and not their neighbours. Once this is complete, the results are sent to the neighbouring domains for particles which interact with those neighbours. For the Lennard-Jones potential, the above calculations are not necessary. To stick with the model in Equation 2.1, we simply set the result of $F$ to zero, and the above functions are no longer necessary, other than the initial distribution of particle coordinates.

The next step is to calculate the forces on each particle, which is done in the $\phi$ function. The force calculations use the results from the derivative of the embedding function. This again uses the inner scheduler for task based parallelism. The function further calculates the derivatives of the

**Algorithm 3.4** Pseudo-code of the force calculations of many-body potentials for the hybrid method

```
$ExchangeParticleCoordinates()$
for all $particles$ do
    $calculate\ \rho\ of\ inner\ particles$
end for
$WaitForReceive()$
for all $particles$ do
    $calculate\ \rho\ of\ outer\ particles$
end for
for all $particles$ do
    $calculate\ F\ and\ F'$
end for
for all $particles$ do
    $calculate\ \phi\ of\ inner\ particles$
end for
$WaitForReceive()$
for all $particles$ do
    $calculate\ \phi\ of\ outer\ particles$
end for
```

forces, and the pair potential. Once the calculations of $\phi$ are complete, the simulation will again wait until the results from surrounding domains are received. Once this data has been received, the simulation will continue with the calculation of the forces generated upon the particles within the domain from particles in the neighbouring domains, using the outer scheduler.

The simulation will then combine data from all threads to calculate the total potential energy of each domain, which is then summed over all domains. The results of each of these steps is then utilized to update the positions and velocities of all particles within each of the domains.

## 3.7 Analysis

On both hardware systems, a variety of combinations of threads and MPI ranks are used, in order to gauge and compare performance gains. The analysis is based upon the strong scaling speedups, $S$, as measured against baseline runs using a single MPI rank and a single thread. In addition to the simulations using the hybrid method, all MPI runs were carried out using a single thread per MPI rank, and all threaded runs on the multi-core system using a single MPI rank. The speedup is calculated using Equation 3.1, where $t_{base}$ is the time taken for the baseline run, and $t_{current}$ is the time taken for the current run of the simulation.

$$S = \frac{t_{base}}{t_{current}} \tag{3.1}$$

On the multi-core systems tests were done using one, two, four, eight, and sixteen MPI ranks with varying numbers of threads to the total number of cores available per system. On the Xeon Phi coprocessor tests employed one, two, four, eight, sixteen, thirty, sixty, one-hundred twenty, and two-hundred forty MPI ranks with varying numbers of threads. Tests were run using the same configurations on multiple compute nodes. With the multi-core processors, up for 4 compute nodes were used; whereas with the Xeon Phi nodes only one compute node was used with both Xeon Phis within the node, with the exception of the Copper Honey Comb systems which used up to 4 compute nodes, and 8 Xeon Phis.

All tests were run for 1000 time steps, each step being 2 femtoseconds long ($10^{-15}$ seconds), with the neighbour-lists regenerated every 10 time steps. Each set of tests were run five times, taking the average time of each, in order to more accurately measure the time required to perform the simulation on a given system.

The wall clock time is chosen to measure the time required for the tests to complete, and does not include the time required to read and write configurations files, and instead consists of only the time required to simulate the actual interactions of the particles. This time was chosen for two reasons: firstly, the wall clock time is the perceived time of the user and therefore is the most significant. Secondly, the purpose of this thesis is focused on the improvements of using a hybrid parallel method, and therefore sections of the code which are unaffected by the changes are removed from the timing. Lastly, the percentage of time required for reading and writing files depends on the simulation length, meaning that as simulations are run longer, the percentage of time spent reading and writing the configuration files is reduced.

The hybrid method, which combines both the spatial decomposition and task based method, is able to use varying numbers of threads and MPI ranks. When a single thread is utilized, a serial scheduler is used, as opposed to the more complex dynamic scheduler when using more than one thread. A serial schedule removes the need to schedule cells in an order which prevents them from simultaneously accessing the same particle, and instead schedules cells consecutively. This type of run was used for the spatial decomposition method. Further, for the task based runs, the same simulation was run with only one MPI rank, in which the communication times are insignificant to the speedups – although it does affect the wall clock time. This was done in order to ensure that all three methods use the same code base, in order to more accurately judge the speedup of each method.

### 3.7.1 Expectations

The proposed hybrid method is able to take advantage of both spatial decomposition and cell-task parallelism techniques. It is reasonable to assume that using the hybrid method will allow for better performance on distributed systems, due to the allowance of the cell-task method on multiple compute nodes. For a single compute node, improvements should still be seen in the memory localization of MPI, but also using the improvements of task based parallelism.

The maximum attainable speedups will be limited by Gustafson's law, which states that the maximum attainable speedup is limited not only by the process of parallelization, but the amount of serialized code [25]. In this regard, the serial communication between compute nodes on a distributed network will be a limiting factor. On single nodes, this is also an issue in the fact that one MPI rank may complete its work prior to other ranks completing their work. A significant portion of serial work, including the reading and writing of the configuration file, and the initial distribution of particles is not counted towards the time taken, as this thesis is interested in the gains on the calculation-heavy sections of the code.

The speedups for the hybrid method are expected to be on par with those of the task based parallelism method, but to outperform the spatial decomposition method. For bulk systems, this is expected to be a slight advantage, but not a large gain due to spatial decomposition performing well on these systems. However, for the porous systems the gain is expected to be more significant, due to the spatial decomposition method's load balancing issues for these types of systems. Further, on the many-core Xeon Phi system, the gain is expected to be even larger compared to the spatial decomposition method due to the system not being well designed for MPI. This also means that the expected improvements of the hybrid method will be lower than that of the task based method on these systems.

# Chapter 4

# Single Node Results

This section examines the results of the hybrid, cell-task, and spatial decomposition methods, and compares the timings against a baseline one-rank one-thread timing, in order to measure their speedup factors. For ease of reading, the results are separated into two sections, those for the multi-core processor, and those for the Xeon Phi processor. This allows us to more accurately see the speedup factors resulting from each architecture. Some results are shown using graphs, which were created using xmgrace, for illustrative purposes. Each section also contains tables which show the best results achieved.

Initial tests were completed on the multi-core Xeon processor using a single compute node and on the many-core Xeon Phi processor using a single processor within the compute node. This was done in order to gauge the effectiveness of the hybrid method on these systems. If the overhead associated with the hybrid method is significantly more than that of the spatial decomposition method, then it could potentially mean that the spatial decomposition method would also outperform the hybrid method on multiple nodes.

Previous results from Meyer [4] indicate that the cell-task method outperforms the spatial decomposition method on single node systems. It is expected that the hybrid method, with its use of the cell-task method will also outperform the spatial decomposition method; however will likely not perform as well as the cell-task method alone. This is because of the communication overhead of the spatial decomposition method potentially degrading the performance.

## 4.1 Multi-Core Processor Results

For the multi-core Xeon Processor, only five systems were used: $Cu63$, $Cu105$, $Cu(porous)$, $Fe(bulk)$, and $Ag(liquid)$. The large copper system ($Cu126$) is not used here, as the differences in sizes can be seen between the $Cu63$ and $Cu105$ systems. Further, the $Cu(spheres)$ and $Cu(honeycomb)$ systems have properties which are of specific relevance to the multi-node Xeon Phi (see Chapter 5.2), and hence are also not included.

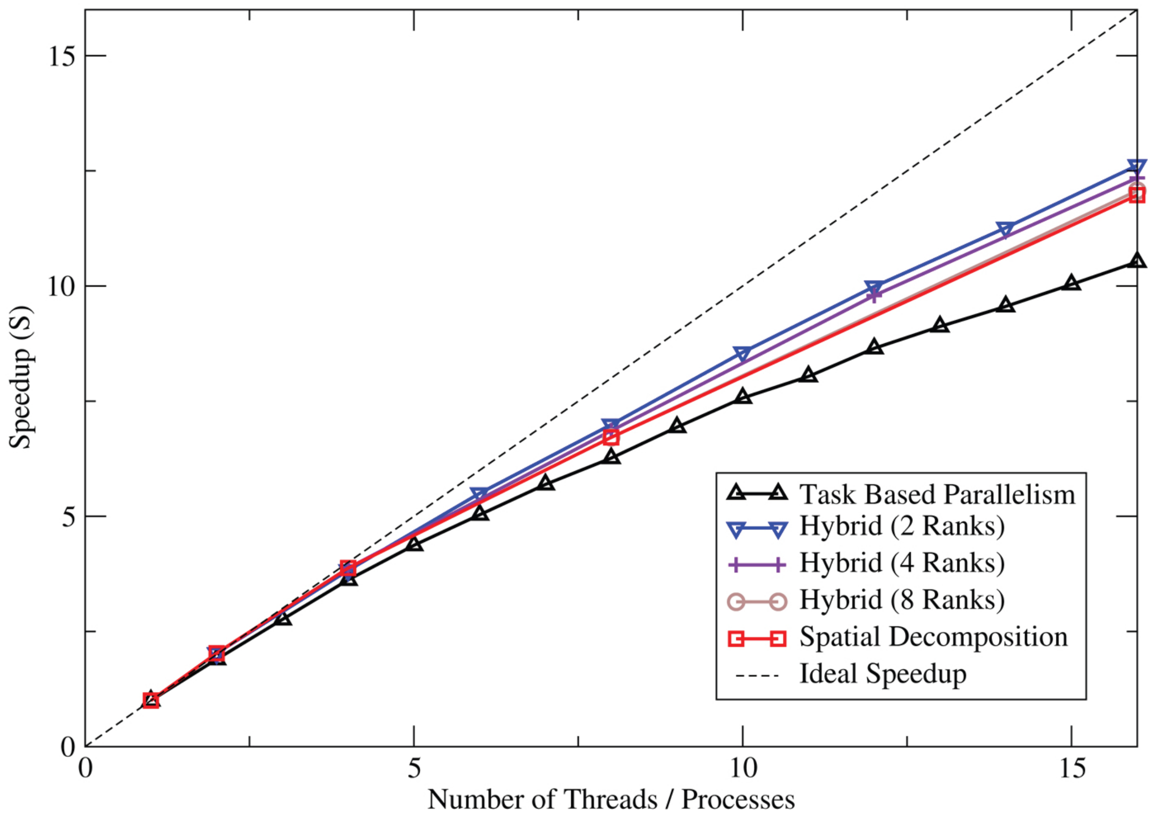

Figure 4.1: The speedups of the three different methods using various numbers of threads and ranks for the $Cu63$ system on a single multi-core processor. The hybrid method results shown in the graph include the number of ranks used for each result in the legend.

Figure 4.1 shows the speedups using varying numbers of threads and ranks for the three methods on the bulk $Cu63$ system consisting of approximately one-million particles – similar results were obtained for the bulk $Cu105$ system consisting of approximately four-million six-hundred-thousand particles, except that the $Cu105$ system had slightly better speedups for all methods, see Table 4.1. Due to the cluttering caused by showing all hybrid runs, all proceeding graphs will exclude those results, and instead only show the best results achieved with the hybrid method.

On both the $Cu63$ and $Cu105$ systems the task based method is outperformed in terms of speedup by the Spatial Decomposition method, which is contrary to results obtained by Meyer [4, 5]. The differences in the results from this work and previous work is likely due to the updated code being more efficient and better parallelized than the original spatial decomposition method used by Meyer [4, 5]. Some changes included updated communication code, and a vectorized potential being used. Another potential reason for this discrepancy is memory speeds, which can have drastic effects on the simulation [5]. This does show that results may vary between runs and compute nodes.

Further, the hybrid method with two MPI ranks was able to achieve the best performance increase for these two systems. This is likely due to the localized memory access associated with the spatial decomposition method, in combination with using fewer ranks than the spatial decomposition method, which reduces the communication and other overhead associated with MPI.

Figure 4.2 shows the speedups for the three methods on the inhomogeneous $Cu(porous)$ system consisting of approximately two-million particles. The results from this system are significantly different than the bulk homogeneous systems of $Cu63$ and $Cu105$ (see Table 4.1). In this case, the inhomogeneity of the porous system significantly degrades the performance of the spatial decomposition method. This degradation is expected due to the uneven distribution of particles amongst processor cores.

The task based method does not suffer from this disadvantage due to its use of a dynamic scheduler which is able to automatically fill idle cores with additional work while there are tasks remaining. The hybrid method is again able to outperform either of these methods using two MPI ranks, each with eight threads. This is likely due to the more localized memory access of MPI, combined with the dynamic scheduling of the task based method.

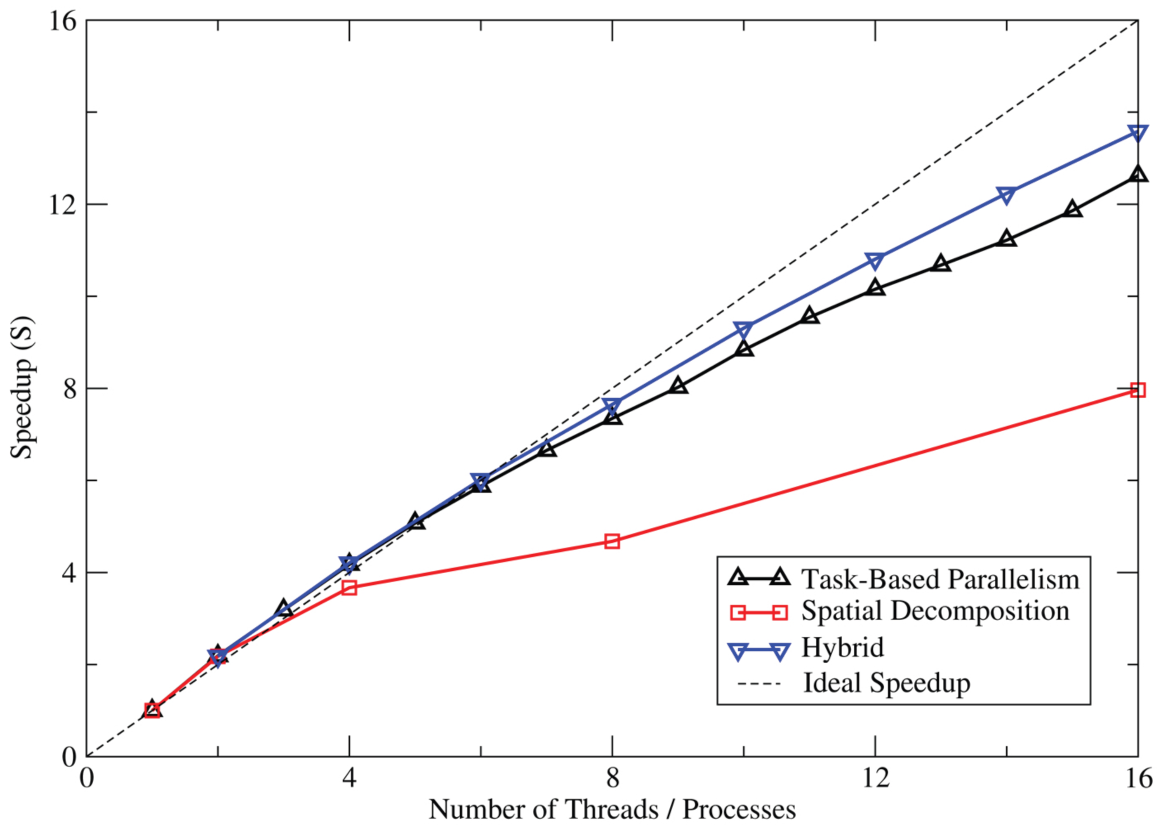

Figure 4.2: The speedups of the three different methods using various numbers of threads and ranks for the $Cu(porous)$ system on a single multi-core processor. The hybrid method shown in the graph utilizes two MPI ranks.

The $Fe(bulk)$ system consisting of approximately four million particles performs well using all three methods, as seen in Table 4.1, although the cell-task method does have worse speedups compared to the other two methods, only achieving a factor of 10.4 when using all sixteen threads. The homogeneity of the system works well for the spatial decomposition method which was able to achieve a speedup factor of 13.1 using sixteen MPI ranks. The hybrid method achieved a factor 13.2 using two MPI ranks each with eight threads, which has no statistical difference compared to the spatial decomposition method, based on the standard deviations of the results.

More interestingly, Figure 4.3 shows the results of the $Ag(liquid)$ system with the three dif-

| **System** | **Spatial Decomposition Speedup (Ranks)** | **Task-Based Parallelism Speedup (Threads)** | **Hybrid Speedup (Ranks $\times$ Threads)** |
|---|---|---|---|
| Cu63 | 12.0 (16) | 10.5 (16) | **12.6** (2 $\times$ 8) |
| Cu105 | 13.1 (16) | 11.3 (16) | **13.5** (2 $\times$ 8) |
| Cu (porous) | 8.0 (16) | 12.6 (16) | **13.6** (2 $\times$ 8) |
| Fe (bulk) | 13.1 (16) | 10.4 (16) | **13.2** (2 $\times$ 8) |
| Ag (liquid) | **7.6** (16) | 7.4 (16) | 4.7 (8 $\times$ 2) |

Table 4.1: Best parallel speedup factors in simulations involving different systems on the multi-core processor. Best speedups for each system are bolded for emphasis.

ferent methods. For the multi-core processors, this is the only case where the hybrid method had its best performance not using two ranks, and instead uses eight ranks, each with two threads – this results in only two data points within the graph. It can still be seen, however, that the hybrid method performed significantly worse than the other two methods.

All three methods perform worse on this system than any other system – this is likely due to two factors. For the spatial decomposition and hybrid method, they are both strongly affected by the communication overhead of MPI. For this system, which is a liquid, the particles are more freely allowed to move about, hence there is a higher than normal amount of particles which move between domains. This results in a higher than normal amount of communication between domains, in order to transport the particles which have moved outside the domain boundaries. Further, the cell-task method, along with the hybrid method, are limited by another factor. Due to the relatively simple and quick calculations using the Lennard-Jones potential, these methods are more limited by memory access. These simpler calculations also further emphasize the times taken to create the neighbour-lists and scheduler. Further, the serialized removal of particles which have passed beyond the domain's boundaries could also have a crucial affect on the performance.

The results from the single node multi-core processor show promise for the hybrid method under all cases, other than the $Ag(liquid)$ system. It can be seen that in most cases two MPI ranks in combination with eight threads for the hybrid method achieves the best performance on these systems, again with the exclusion of the $Ag(liquid)$ system. This shows that the memory management

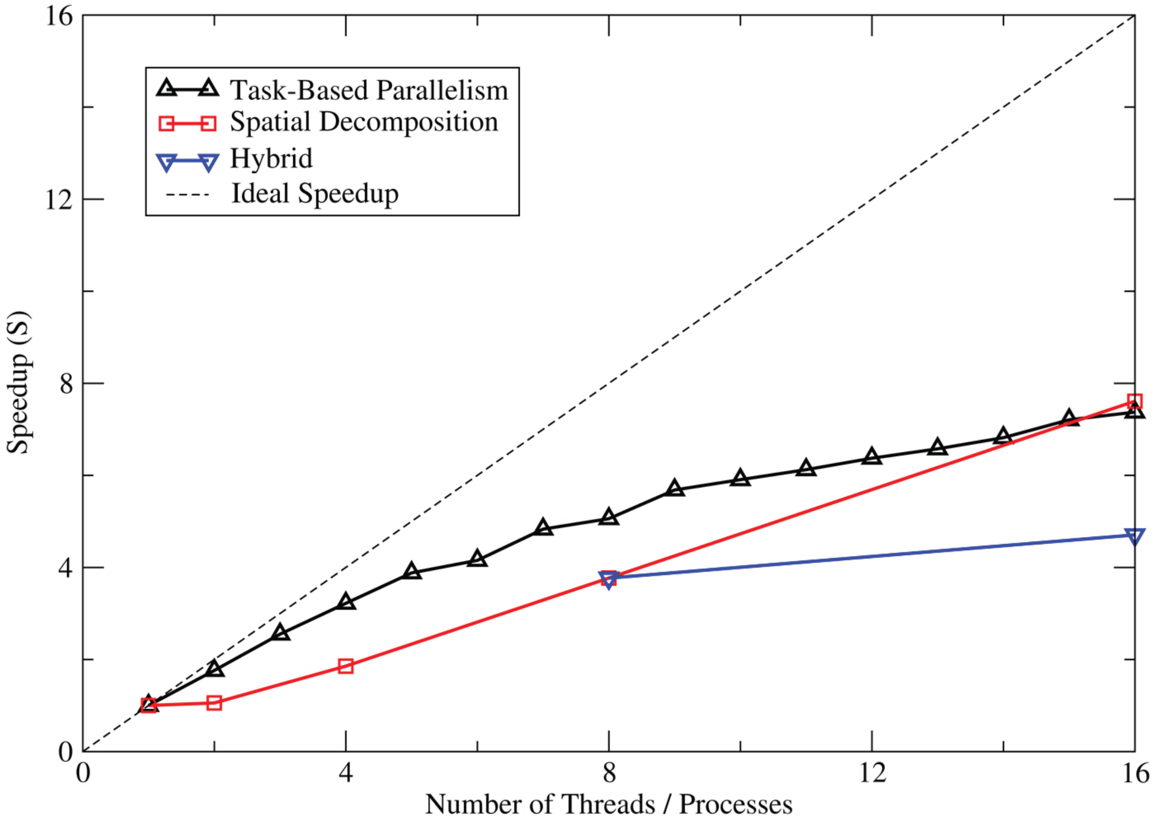

Figure 4.3: The speedups of the three different methods using various numbers of threads and ranks for the $Ag(liquid)$ system on a single multi-core processor. The hybrid method shown in the graph utilizes two MPI ranks.

associated with the spatial decomposition method is advantageous to the hybrid method.

## 4.2 Xeon Phi Processor Results

For the Xeon Phi system, in addition to the five systems used for the multi-core processor, the eight million particle bulk $Cu126$ system was also tested. As seen in Figure 4.4, after a certain point the speedups of the cell-task method are reduced significantly for the $Cu63$ system. This is due to the fact that the scheduler is limited to running only tasks which do not interact with any particles which are used by any other running tasks. For the $Cu63$ system, this limits the amount of tasks to around 216 simultaneously running, which cannot use the full Xeon Phi processor. When more threads are requested than there are tasks available, this leaves some cores idle, and waiting for work. For this reason, the $Cu126$ system was added, for comparative purposes with the $Cu105$ system.

Despite this, results still show that the task based method is able to achieve the best performance of all three methods, with the hybrid method, using thirty ranks each with eight threads, still outperforming the spatial decomposition method – this is contrary to what is seen on the multi-core processors. The Xeon Phi, with its many-core architecture, is not well designed for MPI communications, and is instead designed more for a threaded approach, which can degrade performance of the spatial decomposition and hybrid methods.

Results for the $Cu105$ and $Cu126$ systems are similar, as seen in Table 4.2. Results from $Cu126$ are shown in Figure 4.5, which again shows that both the task based and hybrid methods outperform the spatial decomposition method. The hybrid and cell-task methods achieve similar speedups, with the hybrid method achieving slightly better for the $Cu126$ system, and the cell-task slightly better for the $Cu105$ system.

It is interesting to note that for these two systems a near ideal speedup factor is attained with the task based method up to sixty threads – the number of physical cores available for the system. This is shared in the hybrid system, up to a combined forty-eight threads and ranks. This linear growth matches with the maximum speedups factors, until the speedups change to a less linear fashion.

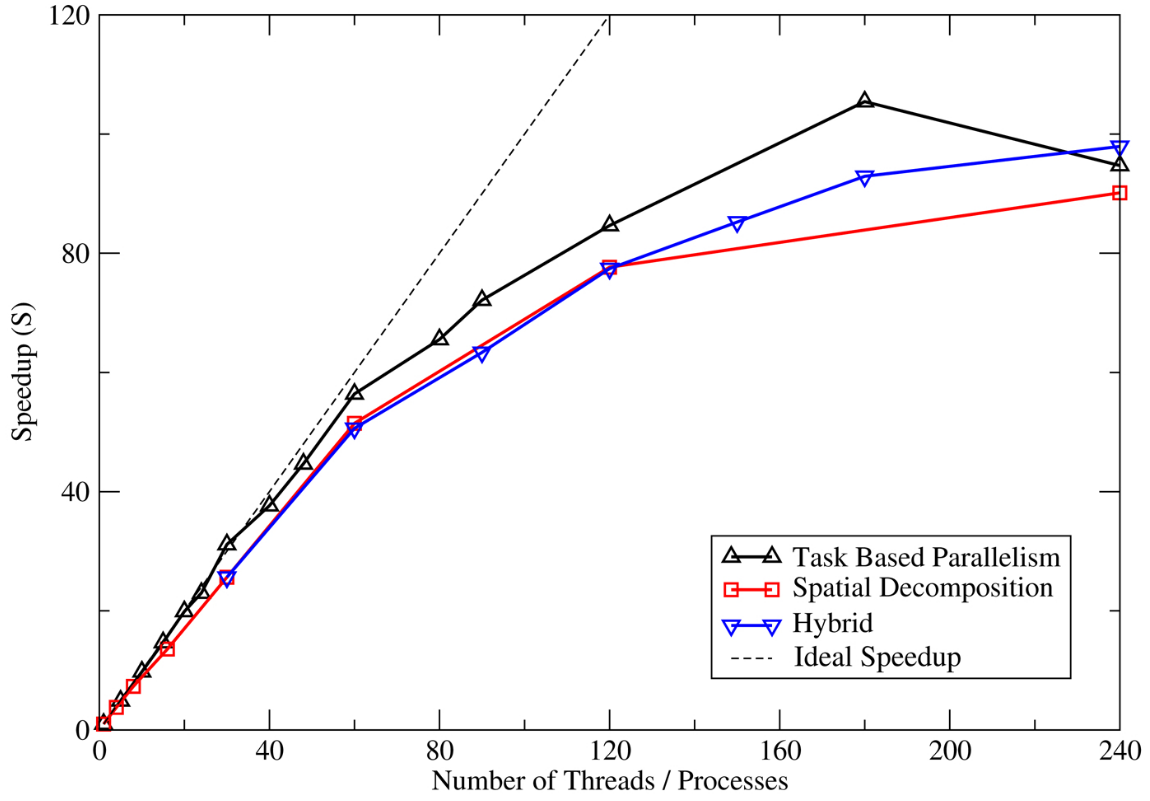

Figure 4.4: The speedups of the three different methods using various numbers of threads and ranks for the $Cu63$ system on a single Xeon Phi processor. The hybrid method shown in the graph utilizes two MPI ranks.

Again, with the $Cu(porous)$ system, the improvements of the spatial decomposition method are significantly worse than that of the other methods. Figure 4.6 shows the large disparity between the spatial decomposition and the two other methods. Unlike with the multi-core processor, the cell-task method is able to outperform the hybrid method, which achieved its best speedup factor with two ranks, and ninety threads per rank.

For this system, the inhomogeneity drastically affects the spatial decomposition method, which is left with a significant amount of cores being idle as they wait upon other cores to finish their assigned work. This does not affect the cell-task method, which uses its dynamic scheduler to

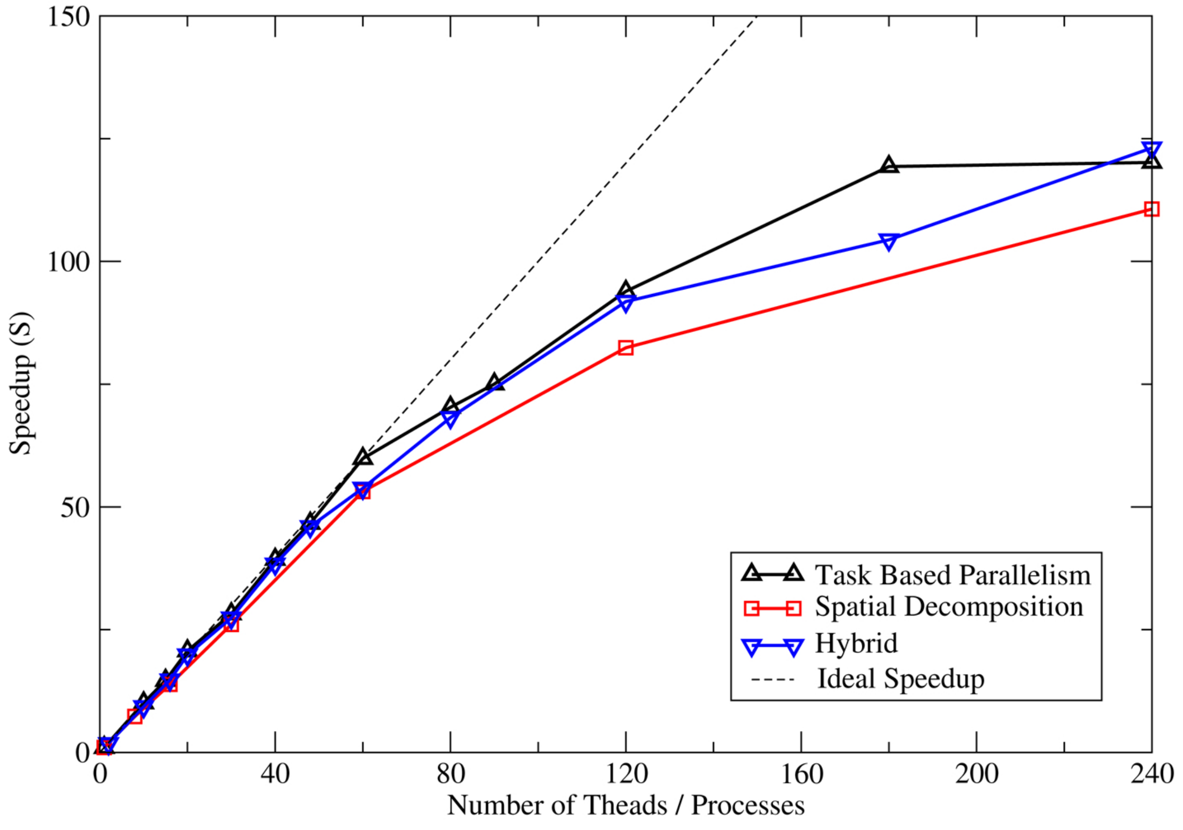

Figure 4.5: The speedups of the three different methods using various numbers of threads and ranks for the $Cu126$ system on a single Xeon Phi processor. The hybrid method shown in the graph utilizes two MPI ranks.

reduce the load imbalance. Further, the hybrid method is able to skip over these empty tasks as well, however one domain of the system may still have more particles than the other, which forces one processor to wait upon the other to complete.

Interestingly, using the $Cu(porous)$ system is the only time in which the hybrid method attains a better performance by not using all processor threads, and instead uses a total of only one hundred and eighty spread across two MPI ranks. The cell-task method again reaches its best speedup at one-hundred-and-eighty threads, as with the $Cu63$ system.

As with running on the multi-core processors, the $Fe(bulk)$ system again performs very sim-

| System | Spatial Decomposition Speedup (Ranks) | Task-Based Parallelism Speedup (Threads) | Hybrid Speedup (Ranks $\times$ Threads) |
|---|---|---|---|
| Cu63 | 90.1 (240) | **105.4** (180) | 97.9 (30 $\times$ 8) |
| Cu105 | 104.1 (240) | **122.7** (240) | 120.7 (2 $\times$ 120) |
| Cu126 | 110.7 (240) | 120.2 (240) | **123.1** (2 $\times$ 120) |
| Cu (porous) | 24.1 (240) | **86.9** (180) | 79.8 (2 $\times$ 90) |
| Fe (bulk) | 103.4 (240) | 119.8 (240) | **121.3** (8 $\times$ 30) |
| Ag (liquid) | 86.2 (240) | 72.3 (240) | **93.4** (120 $\times$ 2) |

Table 4.2: Best parallel speedup factors in simulations involving different systems on the Xeon Phi processor. Best speedups for each system are bolded for emphasis.

ilarly to that of the $Cu105$ system, and in turn the $Cu126$ systems. All three methods are able to achieve good speedup factors, with the spatial decomposition still lagging behind the other methods. The other two methods attain very similar results, with the hybrid method edging ahead of the cell-task method.

Figure 4.7 shows the results attained using the $Ag(liquid)$ system on the Xeon Phi processor. This has significantly different results for the hybrid method compared to those found using the multi-core processor. Once more, the hybrid method achieved its best performance using only two threads, however this time with one-hundred-and-twenty MPI ranks, using the entire Xeon Phi processors. The speedup of the hybrid method is able to surpass both the spatial decomposition and task based parallelism methods' speedups. The different behaviours of this system across the different processor architectures is not fully-understood, and requires more research; however, it is most likely due to the increased number of MPI-ranks for the hybrid method, which allows for more communication to occur in parallel, in combination with the more localized memory access of MPI, since the Lennard-Jones potential is extremely sensitive to memory delays.

It can still be seen that using the hybrid method on the Xeon Phi processor is able to achieve better speedups in all cases compared to the spatial decomposition method. Compared to the cell-task method, in some instances the hybrid achieves better speedups, and in others not. This is dependent on the test system being used, including factors such as homogeneity and number of

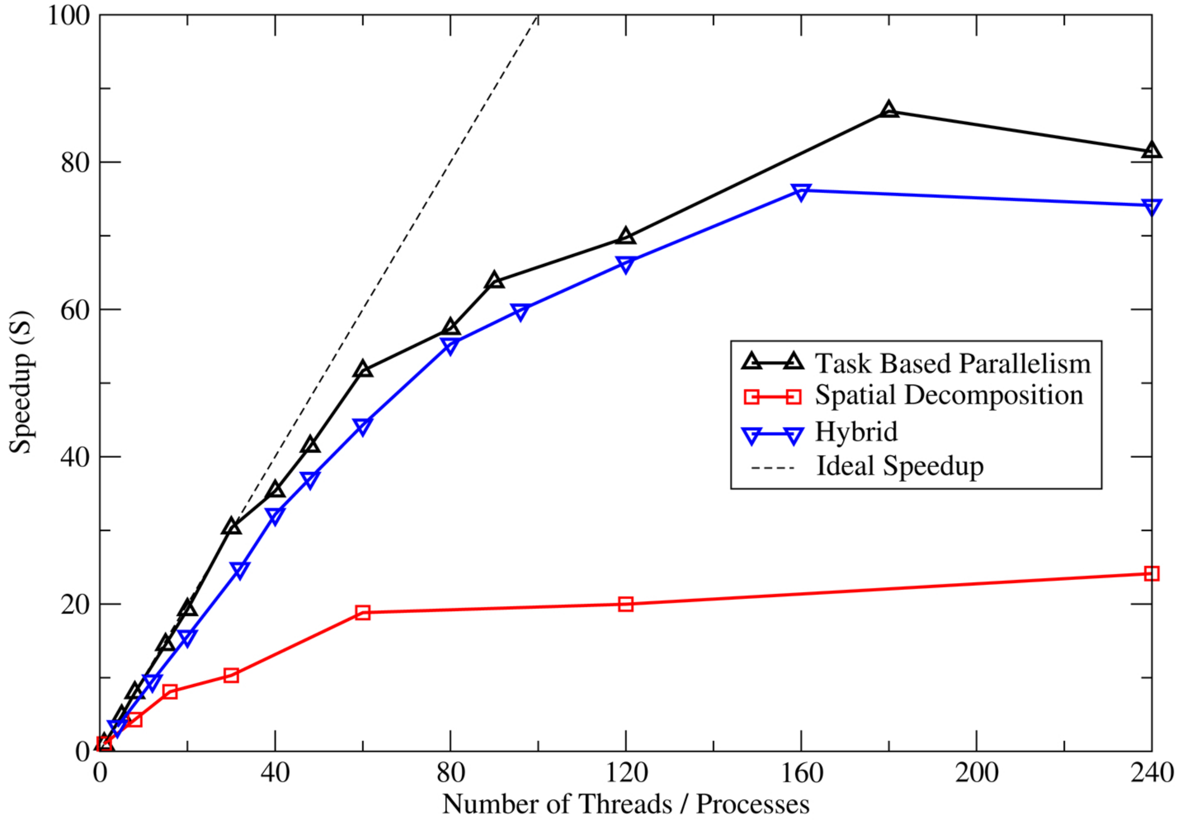

Figure 4.6: The speedups of the three different methods using various numbers of threads and ranks for the $Cu(porous)$ system on a single Xeon Phi processor. The hybrid method shown in the graph utilizes two MPI ranks.

particles, but does not necessarily depend upon the type of potential being used.

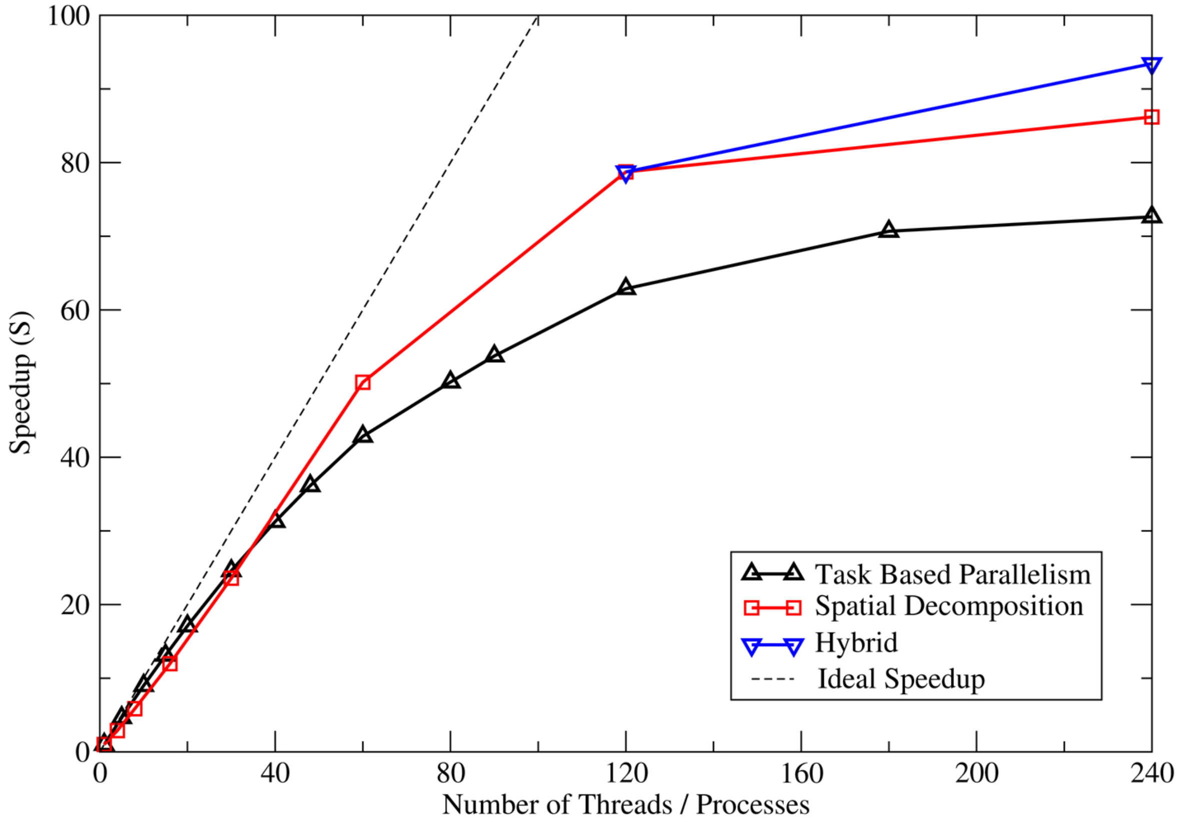

Figure 4.7: The speedups of the three different methods using various numbers of threads and ranks for the $Ag(liquid)$ system on a single Xeon Phi processor. The hybrid method shown in the graph utilizes one hundred and twenty MPI ranks, and hence only has two data points.

# Chapter 5

# Multi Node Results

The main purpose of this work is to bring the task based parallelism method onto separate nodes with the hybrid method. From the results in Chapter 4, it can be seen that the hybrid method does result in better speedups than other methods on some test systems, which shows promise for moving towards multiple compute nodes. Since both the spatial decomposition method and the hybrid method will share the same factor of being, in part, limited to the bandwidth of the network, and the hybrid method outperforming the spatial decomposition on some test systems, it is expected to see the hybrid method again performing well on more than one compute node.

This section will examine the hybrid method against the spatial decomposition method. The results displayed throughout this section will include two results from the hybrid method, one for using one MPI rank per node, and one which represents the best speedup achieved on that test system. Similarly to Chapter 4, the results are again split into two sections, one for each architecture, in order to facilitate easier reading. Further, an additional section is added for symmetric processing which uses both the many-core processor and the Xeon Phi processor located on each computer node.

For this type of set-up, using the spatial decomposition method alone results in small sub-domains being assigned to each processor core. Conversely, the hybrid method with one rank per node assigns larger sub-domains per compute node, and each processor core on the node will process various cells of the assigned sub-domain. For spatial decomposition with inhomogeneous sparse systems, the small sub-domains may result in some cores having relatively few or no particles compared to other cores. As such, the hybrid method with its larger sub-domains is expected to significantly outperform the spatial decomposition method for inhomogeneous systems. Further, as the hybrid method was able to outperform the spatial decomposition method in most cases on a single computer node, it is expected that this outcome will carry over to multiple nodes.

## 5.1 Multi-Core Processor Results

The same five systems were used for the multi-node tests as were used for the single node tests, $Cu63$, $Cu105$, $Cu(porous)$, $Fe(bulk)$, and $Ag(liquid)$. A summary of the results obtained for two and four multi-core nodes can be found in Table 5.1 and Table 5.2 respectively.

Figure 5.1 displays a comparison between the spatial decomposition method and two results of the hybrid method – one with a single rank per node, and one for the best results achieved (two ranks per node) – for the $Cu63$ test system using four multi-core nodes. Both the spatial decomposition method and hybrid method perform well on this test system, with the hybrid method's best performance edging out that of the spatial decomposition method. Similar results were attained using two multi-core nodes.

These results do show that the combination of ranks and threads do make an impact upon the performance of the hybrid method. This is important to keep in mind, as finding the correct balance between ranks and threads is necessary to achieve the best performance. Similar trends are also seen in the results for the $Cu105$ system, using both two and four multi-core nodes. Results for the $Cu105$ system with two multi-core nodes is shown in Figure 5.2. The difference being only in the speedup factor being greater in the $Cu105$ system than in the $Cu63$ system, which is also observed utilizing only one compute node. For both the $Cu63$ and $Cu105$ systems, the results show that the speedup factor continues to increase as more compute nodes are added. This bodes well for the hybrid method, as it shows that the results continue to improve and outperform the spatial decomposition method.

Figure 5.3 shows that on the porous system, the inhomogeneity still drastically affects the spatial decomposition method, but does not affect the hybrid method as severely. It is noted however, that only on this system using four multi-core nodes does using one rank per node produce better performance than two ranks per node. This shows that there is a point when the system division causes load balancing issues due to the inhomogeneity.

When using two compute nodes, however, the hybrid method has its best performance at two

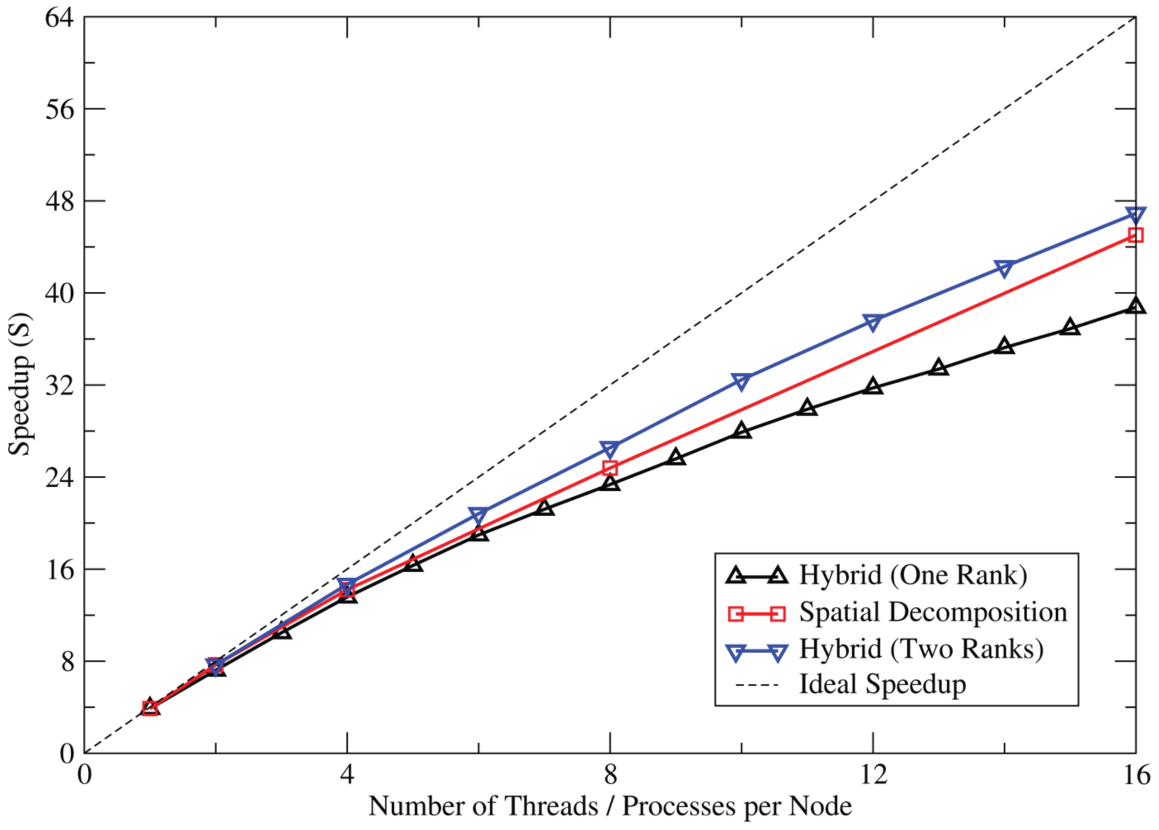

Figure 5.1: The speedups of two different methods using various numbers of threads and ranks for the *Cu*63 system on a four multi-core nodes. The hybrid methods number of ranks are displayed in the legend.

ranks per compute node. This further emphasizes the splitting of the system into domains creates imbalances on the nodes. This does indicate that splitting the system into sub-domains does provide an advantage, however splitting into too many sub-domains will affect the performance. This is an important system to examine, however it should be noted that another porous system with a different shape may yield different results. This is simply due to the distribution of particles over nodes, and not simply in the fact that it is a porous system – the inhomogeneity is the larger factor at play. As with four compute nodes, the spatial decomposition method has the lowest speedup factor for this test system.

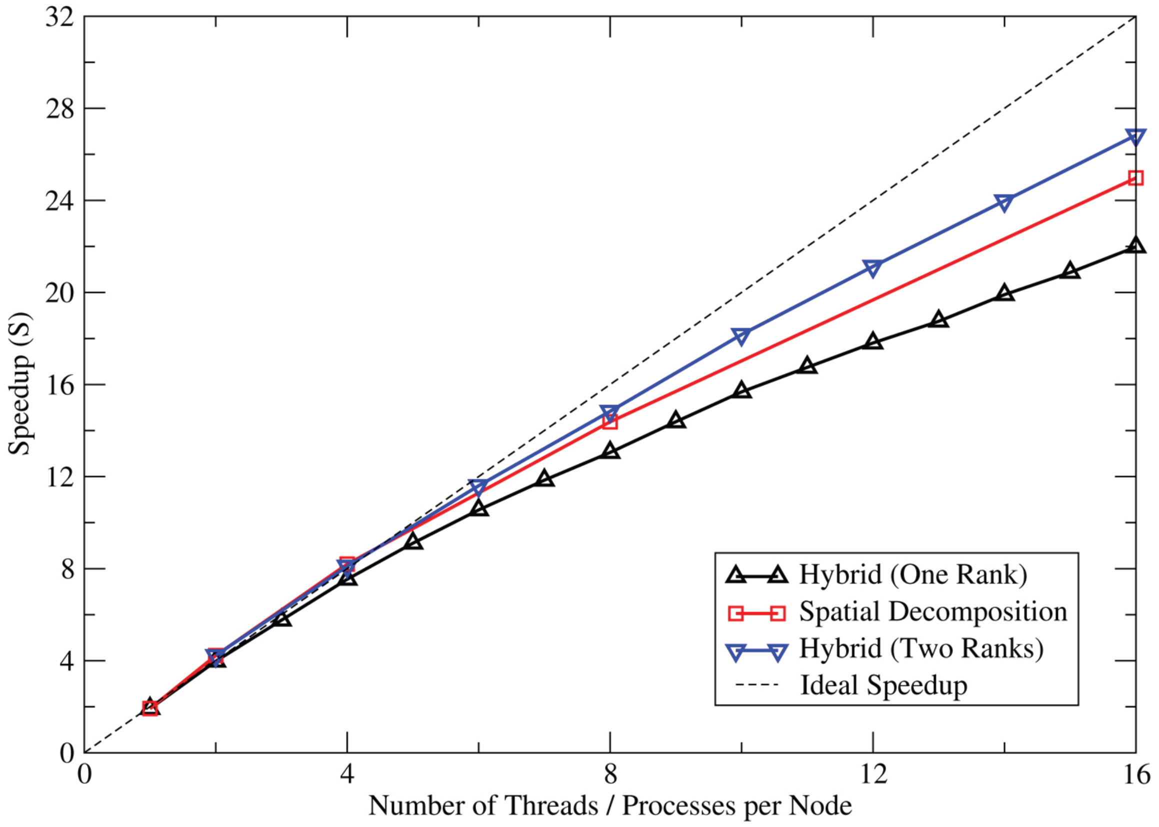

Figure 5.2: The speedups of two different methods using various numbers of threads and ranks for the $Cu105$ system on a two multi-core nodes. The hybrid methods number of ranks are displayed in the legend.

As with the other bulk systems, the results for $Fe(bulk)$ show that the hybrid method and spatial decomposition method both work well across multiple compute nodes. Similarly to using a single compute node, there is no statistically significant difference between the hybrid method and the spatial decomposition method, in terms of speedup factors. This is somewhat surprising, as it was expected that the improvements of the hybrid method would take advantage of reduced memory demand associated with the more complex algorithm of the Mendelev potential. However, the longer compute times associated with this potential also reduces the relative impact of communication overhead. This shows that the speedup factor is also dependent upon the system, in addition

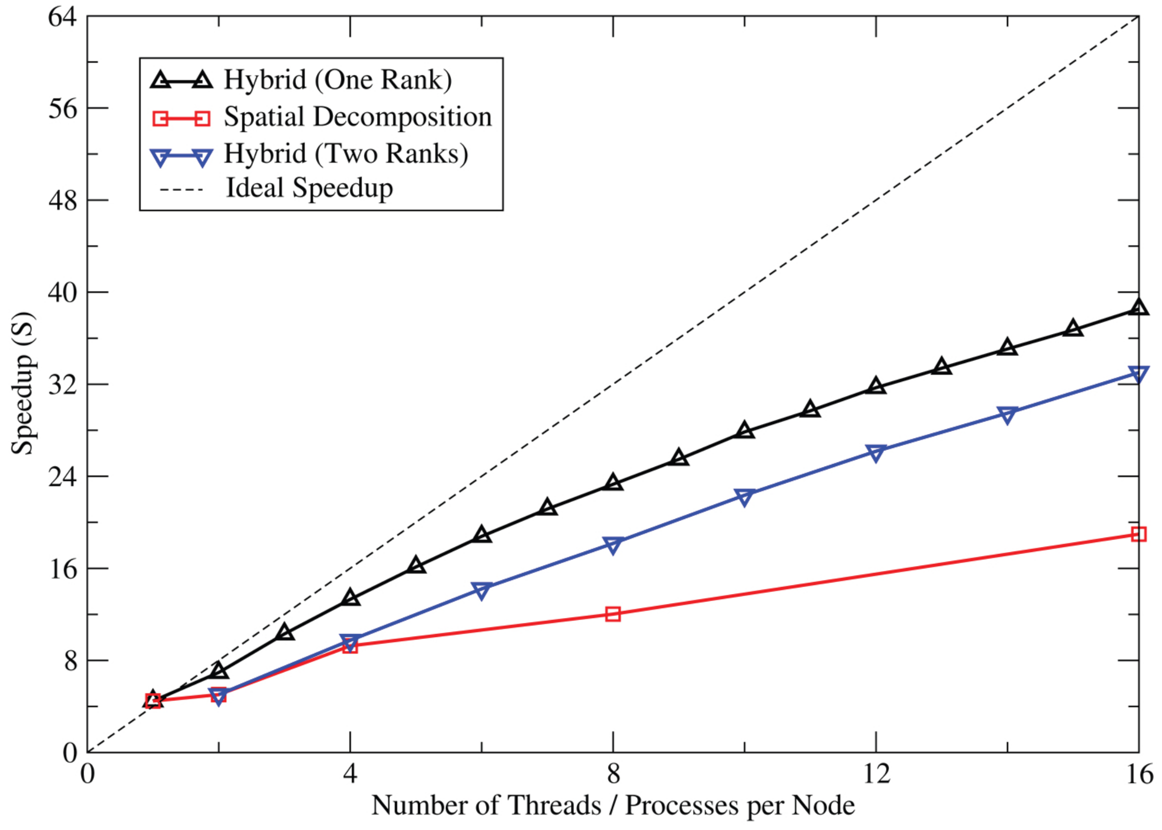

Figure 5.3: The speedups of two different methods using various numbers of threads and ranks for the $Cu(porous)$ system on a four multi-core nodes. The hybrid methods number of ranks are displayed in the legend.

to the potential on the multi-core processors.

Lastly, the $Ag(liquid)$ system, as seen in Figure 5.4, again shows that the hybrid method performs poorly compared to the spatial decomposition method for this system setup. Again, better performances were attained by adding more MPI ranks to the hybrid method.

It should be noted, that due to only using the Lennard-Jones potential on a liquid system, it is difficult to determine whether the performance of the hybrid method on this system is limited due to the simpler calculation of the Lennard-Jones potential, or in the overhead associated with the movements of particles between domains caused by the liquid system. This warrants further study

| **System** | **Spatial Decomposition Speedup (Ranks)** | **Hybrid Speedup Single Rank (Threads)** | **Best Hybrid Speedup (Ranks $\times$ Threads)** |
|---|---|---|---|
| Cu63 | 23.2 (16) | 20.3 (16) | **24.4** (2 $\times$ 8) |
| Cu105 | 25.0 (16) | 22.0 (16) | **26.8** (2 $\times$ 8) |
| Cu (porous) | 11.4 (16) | 22.7 (16) | **24.9** (2 $\times$ 8) |
| Fe (bulk) | 25.6 (16) | 20.3 (16) | **25.9** (2 $\times$ 8) |
| Ag (liquid) | **17.0** (16) | 1.4 (16) | 11.1 (8 $\times$ 2) |

Table 5.1: Best parallel speedup factors in simulations involving different systems on two multi-core nodes. The number of ranks and threads displayed are per compute node, not totals. Best speedups for each system are bolded for emphasis.

in order to determine where the limitation occurs.

| **System** | **Spatial Decomposition Speedup (Ranks)** | **Hybrid Speedup Single Rank (Threads)** | **Best Hybrid Speedup (Ranks $\times$ Threads)** |
|---|---|---|---|
| Cu63 | 45.0 (16) | 38.7 (16) | **46.9** (2 $\times$ 8) |
| Cu105 | 47.9 (16) | 42.7 (16) | **51.9** (2 $\times$ 8) |
| Cu (porous) | 19.0 (16) | **38.5** (16) | 33.0 (2 $\times$ 8)[1] |
| Fe (bulk) | 50.3 (16) | 29.6 (16) | **50.4** (2 $\times$ 8) |
| Ag (liquid) | **36.1** (16) | 2.6 (16) | 26.5 (8 $\times$ 2) |

Table 5.2: Best parallel speedup factors in simulations involving different systems on four multi-core nodes. The number of ranks and threads displayed are per compute node, not totals. Best speedups for each system are bolded for emphasis.

Apart from the poor performance with the $Ag(liquid)$ test system, the hybrid method does perform well on the other test systems both on single and multiple compute nodes. This shows that the hybrid method is a viable option for molecular dynamics simulation on multi-core nodes. Further, since in most cases the hybrid method outperforms the spatial decomposition method when using multiple compute nodes, the hybrid method can improve performance and reduce run times of these simulations.

[1]Second best performance, due to single rank per node having best results.

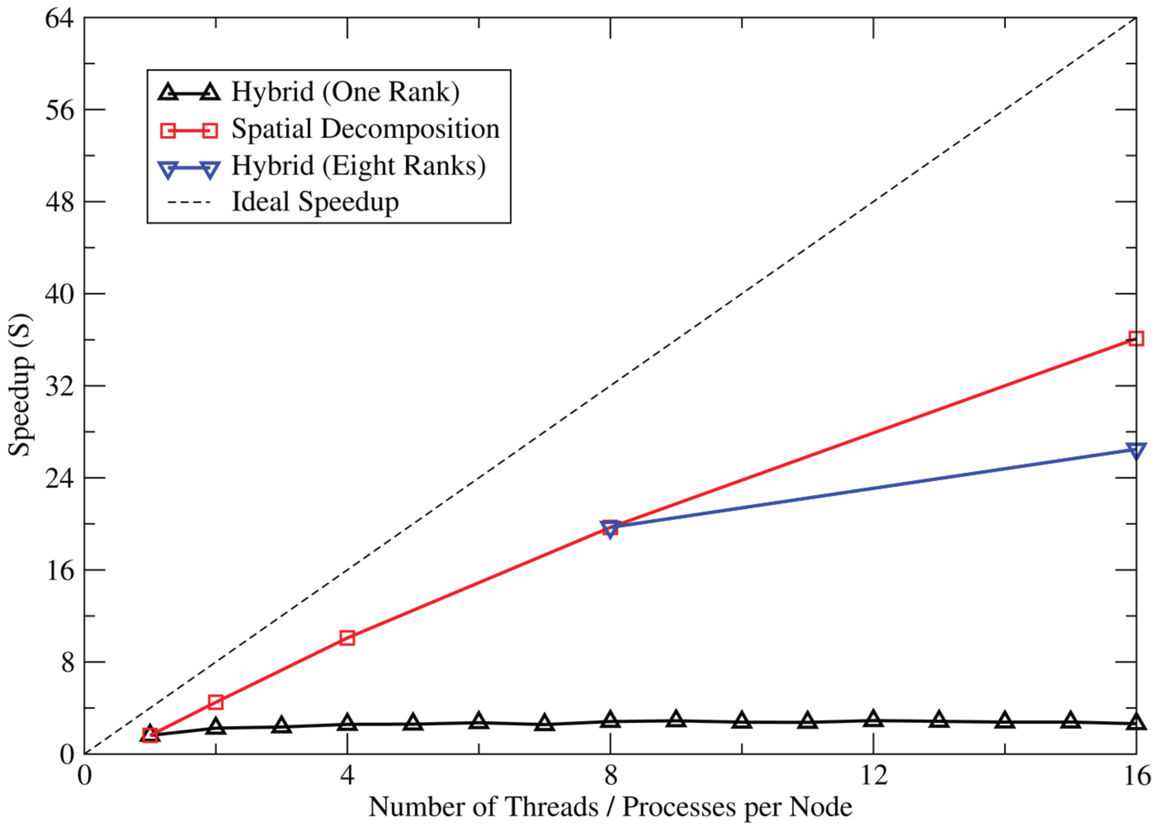

Figure 5.4: The speedups of two different methods using various numbers of threads and ranks for the $Ag(liquid)$ system on a four multi-core nodes. The hybrid methods number of ranks are displayed in the legend.

## 5.2 Xeon Phi Processor Results

The Xeon Phi for multi-node uses the same six simulation systems as its many-core equivalent tests, in addition to the copper sphere and honeycomb systems. A summary of the main six test systems is shown in Table 5.3. On two Xeon Phis, which are interconnected via the system bus, the speedups of the hybrid method are still greater than that of the spatial decomposition method in all cases. For the $Cu63$ system, the speedups are limited due to the small size of the test system which allows for only marginal speedups, although in this case the hybrid method still attains speedups three times that of the spatial decomposition method.

Similarly, for the *Cu*105 system the hybrid method again drastically outperforms the spatial decomposition method by more than a factor of three, and with the *Cu*126 system by more than a factor of four. This clearly shows that the spatial decomposition method is not well suited for use on multiple Xeon Phi processors. For the *Cu*(*porous*) test system, the difference between the hybrid and spatial decomposition is the most extreme, with the hybrid method attaining speedups over six and a half times better than the spatial decomposition method. This is not unexpected due to the inhomogeneity of the system.

With the *Fe*(*bulk*) system, the best speedup attained by the hybrid method is achieved with a single MPI rank per processor, each using one-hundred-and-eighty threads. Again though, the hybrid method attains significantly better speedups than the spatial decomposition method, by nearly a factor of four. The last of these test systems, *Ag*(*liquid*), attains significantly poorer performance than desired, although due to the amount of communication this was not entirely unexpected. In this case, the hybrid and spatial decomposition methods perform similarly, unlike the rest of the test systems.

| **System** | **Spatial Decomposition Speedup (Ranks)** | **Hybrid Speedup Single Rank (Threads)** | **Best Hybrid Speedup (Ranks $\times$ Threads)** |
|---|---|---|---|
| Cu63 | 14.7 (16) | 46.3 (180) | **47.1** (2 $\times$ 90) |
| Cu105 | 21.6 (16) | 70.8 (180) | **71.9** (4 $\times$ 40) |
| Cu126 | 19.5 (16) | 74.8 (240) | **78.4** (4 $\times$ 40) |
| Cu (porous) | 15.2 (30) | 91.9 (180) | **101.3** (2 $\times$ 90) |
| Fe (bulk) | 21.9 (16) | **84.8 (180)** | 83.9 (4 $\times$ 60)[2] |
| Ag (liquid) | 16.6 (16) | 5.2 (180) | **18.4** (16 $\times$ 10) |

Table 5.3: Best parallel speedup factors in simulations involving different systems on two Xeon Phi processors. The number of ranks and threads displayed are per compute node, not totals. Best speedups for each system are bolded for emphasis.

What can be seen immediately with all systems is the drastic drop in performance by utilizing both Xeon Phis – with the exception of the *Cu*63 system, this drop was rather unexpected. With the *Cu*63 system the drop was anticipated due to its inability to use the full processing power of

[2]Second best performance, due to single rank per node having best results.

even a single Xeon Phi, however, on the larger $Cu105$ and $Cu126$ systems, the drop in performance was unexpected, and extremely detrimental. The drop in performance for these two systems using the hybrid method is nearly fifty in the speedup factors between single and multi-node results. A drop of a speedup factor of forty is also seen in the $Fe(bulk)$ system, with an even more significant drop seen in the $Ag(liquid)$ system. The only system which does not follow this trend is the $Cu(pourous)$ system.

The $Cu(porous)$ system, which is the only system to improve performance with more than one Xeon Phi, also has the least amount of communication between its domains. This is due to the porous nature of the system, and particles having fewer neighbours in surrounding domains. Conversely, the $Ag(liquid)$ system, which has the most communication due to it being a liquid system, has the most significant drop in performance when going to two Xeon Phis. This shows that the performance when utilizing the Xeon Phi is most likely drastically affected by the communications on these systems, despite being connected via the system bus. This is a crucial and debilitating limitation of the Xeon Phis, if this were shown to be true.

As such, the two Sphere systems were introduced. These sphere systems consist of two separate spheres which are not connected, and each of the spheres' particles do not interact with the other sphere's particles. This is to ensure that there is minimal communication between the domains – although the communication is not entirely eliminated, as the system must still check to ensure that there's no information coming from the other sphere.

| **System** | **Best Hybrid Speedup Single Node (Ranks $\times$ Threads)** | **Best Hybrid Speedup Two Nodes (Ranks $\times$ Threads)** |
|---|---|---|
| Cu (Spheres) small | 119.8 (16 $\times$ 15) | 225.3 (1 $\times$ 240) |
| Cu (Spheres) large | 126.8 (16 $\times$ 15) | 230.5 (1 $\times$ 240) |

Table 5.4: Best parallel speedup factors in simulations involving the copper sphere systems on one and two Xeon Phi coprocessors. The number of ranks and threads displayed for two Xeon Phi processors are per compute node, not totals.

As can be seen from Table 5.4, when going from one to two Xeon Phis with the sphere systems the speedup factor increases accordingly. When using two Xeon Phis the best speedup factor for these test systems is when each Xeon Phi is working on exactly one Sphere – this is expected, as this is where the communication between the two domains would be negated. This clearly shows that the communication over the system bus is the most debilitating factor when using the two Xeon Phis. While this is important to note, the two distinct spheres is an ideal system, although not practical. As such the honeycomb systems were introduced. These honeycomb systems do have communication between domains, although it is limited compared to the bulk systems.

For these systems, we are more interested in seeing how doubling the system size and doubling the number of processors affects the time required to complete the simulation. As such, we are measuring the *weak scaling parallel efficiency* [26] associated with moving to larger systems compared to the smallest system $Cu(honeycomb(2x4))$. The results from these tests can be seen in Table 5.5. The *weak scaling parallel efficiency* is the parallel efficiency incurred by increasing the problem size linearly with the number of processors. In an ideal system, this would be 100% although generally anything above 80% may be considered to be good for parallelization.

| **System** | **Number of Processors** | **Best Hybrid Efficiency (Ranks $\times$ Threads)** |
|---|---|---|
| Cu (honey comb (4x4)) | 2 | 86.8% (8 $\times$ 30) |
| Cu (honey comb (4x8)) | 4 | 86.5% (8 $\times$ 24) |
| Cu (honey comb (8x8)) | 8 | 82.9% (8 $\times$ 24) |

Table 5.5: Best parallel efficiency in simulations involving the copper honeycomb systems on Xeon Phi coprocessors compared to $Cu(honeycomb(2x4))$ on single Xeon Phi processor. The number of ranks and threads displayed for Xeon Phi processors are per compute node, not totals.

While there is overhead associated in the communication for these systems, as was expected, the overhead is comparatively small, and the simulation still maintains a parallel efficiency above 80% in each case. When using more than two Xeon Phi coprocessors, the processors are connected via QDR InfiniBand between compute nodes, but still use the system bus for the two Xeon Phis located on each compute node. This shows that the hybrid method does indeed perform well on multiple Xeon Phi coprocessors, and can be used in order to further improve performance upon these new many-core systems. However, the hybrid method, as well as the spatial decomposition method, is severely limited in the communications between multiple Xeon Phis, which needs to be reduced.

## 5.3 Symmetric Mode Results

The symmetric mode involves using both the Xeon Phi coprocessor and the multi-core host processor, which are both located in each compute node. This would allow for the full processing power of each compute node to be taken advantage of. This is advantageous, as the Xeon Phi co-processor must sit alongside the host processor, and without using the host processor in addition to the Xeon Phis, the host processor would be wasted. Only a limited number of tests were done with this type of set-up, using only one or two ranks on the Xeon Phi, and using most of the threads available. On the host processor all sixteen threads were used for these tests. The reason for the limited tests is due to noticing slower than expected speeds, as well, this set-up is testing of a concept as opposed to a full-fledge test.

| **System** | **Hybrid Speedup One Rank Per Processor (Threads)** | **Hybrid Speedup Two Ranks On Xeon Phi (Threads)** |
|---|---|---|
| Cu105 | **95.0** (240) | 81.9 (120) |
| Cu (porous) | **86.0** (180) | 70.4 (90) |
| Fe (bulk) | **135.7** (240) | 115.6 (120) |
| Ag (liquid) | 12.7 (180) | **14.7** (90) |

Table 5.6: Parallel speedup factors in simulations involving different systems on host processor and Xeon Phi coprocessor. The host processor utilized 16 threads in each case; the threads shown for two ranks on Xeon Phi is the number of ranks per MPI rank. Best speedups for each system are bolded for emphasis.

The results summarized in Table 5.6 show that in all cases, evenly distributing the system between the host processor and the Xeon Phi coprocessor attained the best results, with the exception of the $Ag(liquid)$ system. This is not entirely unexpected, as although the Xeon Phi does have the ability to run 240 simultaneous threads as compared to the 16 of the host, it does have a significantly lower clock frequency. The lower clock frequency of the Xeon Phi means that the host processor may be able to complete its work on its domain prior to the Xeon Phi completing work on its own. Although a quick test demonstrated that running two thirds of the system on the host also decreased performance. This shows that a correct balance of the system between host and

Xeon Phi processors must be determined prior to running the full simulation.

Preliminary results do indicate that evenly dividing the simulation system between the host processor and the Xeon Phi coprocessor produces better results than a division wherein the coprocessor has double the number of particles compared to the host. While this is merely a preliminary and incomplete set of tests, it does emphasize a need to determine the correct distribution of particles prior to running the simulation. These results do show that in some cases the performance is improved compared to just using a single Xeon Phi, such as with the *Cu (porous)* and *Fe (bulk)* systems, however it has the opposite effect for the *Cu105* and *Ag (liquid)* systems.

# Chapter 6

# Discussion

From the results seen in Chapters 4 and 5, we can see that the hybrid method outperforms spatial decomposition in most cases. However, on the Xeon Phi, the multi-node results are far less than expected.

The hybrid method is able to brings the advantages of the cell-task parallelism method to multiple compute nodes, which is the main goal of this work. In general, for bulk systems the hybrid method performs well on both single node and multi-node set-ups using the multi-core Xeon processor, although only occasionally are the speedups significantly better than the spatial decomposition method. On the porous system the speedup of the hybrid method is much more significant than that of the spatial decomposition method, due to the uneven distribution of particles onto nodes. Conversely, for the liquid system the spatial decomposition method is able to outperform the hybrid method.

This seems to indicate the methods' speedups are more dependent upon the consistency of the systems as opposed to the potentials being used. It should be noted that the total time required for the simulation to complete is dependent upon the potentials being used, with the Lennard-Jones potential taking the least time with its simple computations, and the Mendelev potential taking the most time due to its more complex algorithm.

The more interesting discussion arises from the Xeon Phi system with its many-core architecture. The many-core architecture is designed to use threading as opposed to message passing, which can be seen in the results of the single compute nodes. When using inhomogeneous systems the high number of simultaneous running processes can be debilitating to the spatial decomposition method, although it does not affect the hybrid method nearly as severely, due to the reduced number of MPI ranks. The effect on the spatial decomposition method is caused by the variations in the workload and particle distributions increasing as domains become smaller.

When using the symmetric mode only a limited number of tests were done, with even distribution of the system between host and coprocessor, or with the Xeon Phi coprocessor having more of the system to work on. It would be interesting to see if these results would be consistent with different distributions of the system between host and coprocessor.

With multiple Xeon Phi processors the main bottleneck is the system bus, which is surprising due to the generally fast speeds of this type of medium. This can be seen in both the symmetric mode and when using multiple Xeon Phis. When the amount of data required in the communication process on the test systems is reduced then the amount of overhead is in turn reduced and the system is able to perform well on multiple Xeon Phis. This is an important limitation, although it is not debilitating, as most bulk systems are not necessarily of too much interest, and instead the honeycomb systems provide a more current field of interest.

It should be noted that for the communication between Xeon Phis the TCP protocol was used. This was initially used due to limitations of the supercomputing network, and all tests completed carried through with using the TCP protocol. The TCP protocol has overhead associated with it which can further affect the performance associated in the communication aspect of the simulation. By using other protocols, such as TMI, the efficiency may be improved, and initial results seen in Table 6.1 show that this does reduce the communication overhead; however, this has limitations in the number of concurrent ranks which can be run, due to the MPI implementation. Further, this seems to be highly dependent upon the number of ranks and threads being used. By using different protocols results may in improved, although this is not conclusive and requires further study.

From Table 5.5 we can see that there is a fair size drop in the parallel efficiency between $4 \times 8$

| System | TCP Efficiency (Ranks $\times$ Threads) | TMI Efficiency (Ranks $\times$ Threads) |
|---|---|---|
| Cu (honey comb (8x8)) | 55.7% (1 $\times$ 180) | **65.3%** (1 $\times$ 180) |
| Cu (honey comb (8x8)) | **82.9%** (8 $\times$ 24) | 77.3% (8 $\times$ 20) |

Table 6.1: Comparison between the weak parallel efficiency associated with the TCP and TMI protocols using one rank per Xeon Phi, across 8 Xeon Phi processors.

and $8 \times 8$ copper honeycomb systems, which is not seen between the $4 \times 4$ and $4 \times 8$ systems. This may be an effect of the communication being different between divisions of systems, i.e. a $4 \times 8$ division is not the same as a $8 \times 4$ division. It would be interesting to see whether this continues further as the systems again increase in size, however, due to the limitation of only having 8GB of on board memory for the Xeon Phi, the largest system we could test was the $8 \times 8$ system. This is another limitation which must be accounted for. One possible solution to this problem is to read the initial configuration file from a host processor, which distributes all particles to the Xeon Phi nodes, before the simulation begins. This is an aspect which has not currently been examined, but should be looked into in order to overcome this limitation in system size.

Further things which must be accounted for is to determine what the largest factor at play is for the speedup factors. From the results, it indicates that the composition of the material is the largest factor, as opposed to the potential being used. It would be beneficial to examine the Lennard-Jones potential in a solid bulk form, as well as porous and honeycomb systems. As well the Tight-Binding and Mendelev potentials could be tested in a liquid form. It would be interesting to see how the $Ag(liquid)$ system were to perform if the thread locking which occurs for particles requiring transport to other domains were removed. This system has many particles which require transport, and the performance may be hampered by the thread locking.

It should also be noted, and is important to take into account, the combination of MPI ranks and TBB threads for the hybrid method. While on the multi-core systems, this is fairly consistent from the results, on the many-core Xeon Phi systems, this is not as clear. It may require testing the simulation using short runs in order to first determine the best combination prior to running the simulation for longer terms.

Also, throughout this work simulation systems which were used consisted of only a single type of material: copper, iron, or silver. While using multiple types of materials within each system would likely not affect the speedup factors, it should still be tested in order to show conclusively the effects. In order to solidify the results of the hybrid method, further tests are required which would determine the effects of the above. These tests are unlikely to affect the results, nevertheless they should still be done in order to remove any doubt associated with the hybrid method's performance surpassing that of the spatial decomposition method.

As mentioned in Chapter 4, the results obtained for this work show that the spatial decomposition method outperformed that of the cell-task method on bulk copper systems, which is contrary to previous results found by Meyer [4, 5]. Although this observation is interesting, the reasons for it are not entirely known. The most likely reason for this, is the introduction of the cell-task method into the spatial decomposition method. While for these tests a serial scheduler was used, meaning that tasks are scheduled in sequence, it may still result in better memory management for the calculations. The SIMD instructions are also now used in the spatial decomposition method, which required changes to the access patterns of particle data, which further emphasises changes to the memory management.

Another reason for the discrepancy may be in the combination of the cell-task method with the spatial decomposition method. For the tests done for this paper, all methods used the same program using different configurations. The task based parallelism technique, therefore, does incur some overhead from the MPI calls required by the spatial decomposition method, even though only one spatial domain exists. While a quick test showed that this does not affect the speedups compared to a purely task based approach, it is possible that it does in fact affect the speedups. Further tests comparing the speedups of a purely task based approach without MPI calls, and the task based approached used throughout this work may be required in order to determine if the use of the MPI calls affected the performance. Nevertheless, the main goal is to show the different speedups between the hybrid and spatial decomposition methods, and is not heavily focused on the cell-task method. This work has still shown that the hybrid method is able to provide better speedups than that of the spatial decomposition method.

Compared to the work done by Pal [15], this method is not able to double the performance using the hybrid method except in certain conditions, such as the porous system on the Xeon Phi. While this does appear at first glance to be negative, it should be noted that the two sets of results are different enough to make them difficult to compare directly, since Pal only used two tests cases involving significantly fewer particles than used throughout this work, and used significantly different hardware, further making it difficult to compare the two methods.

The proposed method also does take advantage of Newton's Third Law of motion, and hence requires significantly less work. Further, the spatial decomposition method requires less communication between nodes. Without having more detailed information from Pal's method using similar systems to those tested for this work, it is impossible to state which would perform better; however, taking into account the aforementioned benefits, it is likely that the proposed hybrid method would outperform Pal's method.

Ackland does not give explicitly the speed-ups achieved through his work [14]. By examining his graphs, it can be deduced that his method achieved approximately a speed-up factor of 11 using 16 threads for an iron system consisting of 31,250 atoms. For the iron system tested for this thesis, speed-ups exceeding a factor of 13 were obtained using 16 threads for the system consisting of 4,000,752 atoms. Tests were, however, run on different hardware which prevents conclusively stating that the proposed hybrid method was able to outperform Ackland's method.

It may be possible to further improve the hybrid method by combining it with the use of graphical processing units (GPUs). Since GPUs are only able to execute a small fraction of the instructions required for the MD simulation, which has already been massively parallelized, the gains from GPUs vector units may be small compared to using the GPU on a serial version of the code. However, due to the computationally intensive sections of code having already be vectorized, the GPU instructions may be implemented with relative ease.

This work has shown that the Xeon Phi's performance is severely impacted by the utilization of the system bus to communicate between nodes. It is likely, although not for certain, that the communication required between a Xeon Phi and a GPU would also negatively impact the performance of the simulation. This would not be the case, however, for the multi-core processors. It would be interesting to see what gains may be achieved by the introduction of a GPU to the hybrid method, and whether it be a positive or negative effect on the many- and multi-core architectures.

# Chapter 7

# Conclusion

The hybrid method, in general, performs well and can outperform the spatial decomposition and cell-task parallelism methods on a variety of test systems. This applies to both the multi- and many-core processor architectures seen in the Xeon and Xeon Phi processors. This shows that the hybrid method is a viable candidate which can be used in conjunction with, or in place of the cell-task or spatial decomposition methods.

Despite this, it should still be noted that the other methods will still outperform the hybrid method under certain conditions. However, in these cases the hybrid method is able to switch to a purely spatial decomposition or cell-task method, thereby making it unique and well adaptable to different situations. This does mean that some initial short run tests are required in order to first determine the number of MPI ranks and TBB threads which get the best performance out of the simulation. There is no way around this limitation, and the user would be required to determine the best rank/thread combination.

On a single compute node multi-core processor the hybrid method is able to outperform the spatial decomposition method in all systems, with the exception of the liquid silver system. Further, the hybrid method speedups are significantly more than those of the spatial decomposition method for the porous copper system. This observation further extends to that of two and four compute nodes, still with the multi-core processor. This goes along with the expectations for the hybrid and spatial decomposition methods.

What was not expected, is that the hybrid method provides better speedups on all systems with a single multi-core compute node compared to the cell-task parallelism method, again with the exception of the liquid silver system. The expectation was that the two systems would perform on par with each other, however the results obtained show that the hybrid method, and in some cases the spatial decomposition method, is able to outperform the cell-task method; this may be due to more localized memory access associated with MPI.

With the many-core Xeon Phi processors, the speedups of the hybrid method surpass those of the spatial decomposition method in all cases using both a single Xeon Phi, or two Xeon Phis, due to the Xeon Phis not being well designed for MPI, and instead being designed for threading. The threading-focused design also did impact the performance of the hybrid method compared to the cell-task method on some systems. Surprisingly, the hybrid method was able to outperform the cell-task method on three out of the six main test systems on the Xeon Phi.

The results from the two Xeon Phis did show a detrimental performance drop on the main six systems. By using further tests, it is determined that this is a result of the communication between the two processors. This was not originally anticipated due to the system bus generally being a fast medium. By reducing the communication between the two Xeon Phis, the results significantly improve. This has hence allowed for the use of 16+ million particle systems to be simulated within a day. This can further be expanded to larger systems if the Xeon Phi were to allow for larger configurations, although due to hardware limitations this was the largest system tested.

This work has shown that the hybrid method is able to produce better speedups than the spatial decomposition method and the cell-task method in most cases. This make the hybrid method a viable alternative for molecular dynamics simulations.

# Bibliography

[1] M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids*. Oxford: Clarendon, 1987.

[2] D. Frenkel and B. Smit, *Understanding Molecular Simulation*. San Diego, CA: Academic Press, 2002.

[3] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, Mar. 1995.

[4] R. Meyer, "Efficient parallelization of short-range molecular dynamics simulations on many-core systems," *Physical Review E*, vol. 88, p. 053309, Nov. 2013.

[5] R. Meyer, "Efficient parallelization of molecular dynamics simulations with short-ranged forces," *Journal of Physics: Conference Series*, vol. 540, p. 012006, Oct. 2014.

[6] R. Meyer and C. M. Mangiardi, "Parallelization of Molecular-Dynamics Simulations Using Tasks," *MRS Proceedings*, vol. 1753, pp. mrsf14–1753–nn10–09, Feb. 2015.

[7] Message Passing Interface Forum.

[8] TBB official website.

[9] R. Meyer, L. J. Lewis, S. Prakash, and P. Entel, "Vibrational properties of nanoscale materials: From nanoparticles to nanocrystalline materials," *Physical Review B*, vol. 68, 2003.

[10] R. Meyer and D. Comtesse, "Vibrational density of states of silicon nanoparticles," *Physical Review B*, vol. 83, 2011.

[11] A. Grünebohm, A. Hucht, R. Meyer, D. Comtesse, and P. Entel, "Simulation of Cluster Sintering, Dipolar Chain Formation and Ferroelectric Nanopartiulate Systems," *Nanoparticles from the Gas Phase*, pp. 139–159, 2012.

[12] M. S. Daw and M. Baskes, "Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals," *Physical Review B*, vol. 29, no. 12, pp. 6443–6453, 1984.

[13] M. I. Mendelev, S. Han, D. J. Srolovitz, G. J. Ackland, D. Y. Sun, and M. Asta, "Development of new interatomic potentials appropriate for crystalline and liquid iron," *Philosophical Magazine*, vol. 83, pp. 3977–3994, Dec. 2003.

[14] G. Ackland, K. D'Mellow, S. Daraszewicz, D. Hepburn, M. Uhrin, and K. Stratford, "The MOLDY short-range molecular dynamics package," *Computer Physics Communications*, vol. 182, pp. 2587–2604, July 2011.

[15] A. Pal, A. Abhishek, R. Soumyendu, and B. Baidury, "Performance metrics in a hybrid MPI-OpenMP based molecular dynamics simulation with short-range interactions," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2203–2214, Mar. 2014.

[16] P. Needham, A. Bhuiyan, and R. Walker, "Extension of the AMBER molecular dynamics software to Intel's Many Integrated Core (MIC) architecture," *Computer Physics Communications*, vol. 201, pp. 95–105, Apr. 2016.

[17] F. Willmore, "Early Experiences With the Intel Xeon Phi and MIC Architecture," *AIChE Annual Meeting*, Nov. 2013.

[18] M. Plotnikov, "GROMACS for Intel Xeon Phi Coprocessor," 2014.

[19] R. Gopalan, "NAMD* for Intel Xeon Phi Coprocessor," 2014.

[20] A. Kohlmeyer, "Implementation of Multi-level Parallelism in LAMMPS for Improved Scaling on PetaFLOP Supercomputers," 2011.

[21] C. M. Mangiardi, "Acceleration of molecular-dynamics simulation through simd instructions," 2013. Undergraduate thesis. Unpublished.

[22] F. Cleri and V. Rosato, "Tight-binding potentials for transition metals and alloys," *Physical Review B*, vol. 48, pp. 22–33, July 1993.

[23] R. Rahman, "Intel xeon phi core micro-architecture," 2013.

[24] R. Rahman, "Intel xeon phi coprocessor vector microarchitecture," 2013.

[25] J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, pp. 532–533, 1988.

[26] SHARCNET, "Measuring parallel scaling performance," 2016.