

2012

## GPU-Accelerated Molecular Dynamics Simulation of Rigid Water

Byron Alexander Tasseff

*Let us know how access to this document benefits you*

Copyright ©2012 Byron Alexander Tasseff

Follow this and additional works at: <https://scholarworks.uni.edu/hpt>

---

**Offensive Materials Statement:** Materials located in UNI ScholarWorks come from a broad range of sources and time periods. Some of these materials may contain offensive stereotypes, ideas, visuals, or language.

# GPU-ACCELERATED MOLECULAR DYNAMICS SIMULATION OF RIGID WATER

A Thesis Submitted  
in Partial Fulfillment  
of the Requirements for the Designation  
University Honors

Byron Alexander Tasseff  
University of Northern Iowa  
May 2012

# Abstract

All simulations of physical systems employ the use of numerical algorithms and approximations to accurately mimic a system's behavior. In classical molecular dynamics (MD), there exists an important tradeoff between the desired accuracy and time scale of a simulation. To decrease computational load and preserve long time scales, MD simulations have often ignored long-range physical interactions among particles after some arbitrary cutoff distance. In many cases, as this cutoff distance is decreased, the accuracy of a simulation significantly degrades.

In biochemical systems with relatively large amounts of polar water molecules, the error introduced through the use of cutoff distances or other approximation schemes could be significant. The purpose of this research is to determine if long-range coulombic contributions from polar water molecules have a nontrivial impact on the accuracy of recent and historical MD simulations.

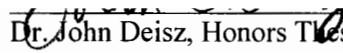
As a first step, we have constructed and partially validated a classical MD simulation of rigid (*i.e.*, fixed-geometry) water molecules. As future simulations will be highly computationally-intensive, the underlying code has been built to utilize the recently-realized power of the graphics processing unit (GPU).

This Study by: Byron Tasseff

Entitled: GPU-Accelerated Molecular Dynamics Simulation of Rigid Water

has been approved as meeting the thesis or project requirement for the Designation  
University Honors

2012-04-27  
Date

  
\_\_\_\_\_  
Dr. John Deisz, Honors Thesis Advisor

5/7/12  
Date

  
\_\_\_\_\_  
Jessica Moon, Director, University Honors Program

# Acknowledgments

There are a few individuals, directly and indirectly involved with this research, whom I offer my sincerest gratitude. First and foremost, it is necessary to thank my professor, academic adviser, and research adviser, Dr. John Deisz, for his scientific insight, expertise on computational modeling, and patience. For his contributions, as well as for the sake of clarity, I employ the use of “we” throughout this thesis. I also thank Jessica Moon and the UNI Honors Program for driving me to pursue this research more diligently and Dr. Paul Gray for allowing me a user account on his CUDA server. Finally, I am grateful to my mother, Colleen Collins, and girlfriend, Sarah Hedeem, for their loving support.

# Table of Contents

<b>List of Tables</b> . . . . .	<b>v</b>
<b>List of Figures</b> . . . . .	<b>vi</b>
<b>Chapter 1 Introduction</b> . . . . .	<b>1</b>
1.1 GPU Computing . . . . .	1
1.1.1 CUDA programming basics . . . . .	2
1.2 GPU-Accelerated Molecular Dynamics . . . . .	5
1.3 Motivation to Study Long-Range Interactions in Molecular Systems . . . . .	6
1.4 Research Objectives . . . . .	7
<b>Chapter 2 Model and methodology</b> . . . . .	<b>9</b>
2.1 Water Models . . . . .	9
2.1.1 TIP4P/2005 . . . . .	10
2.2 Parameters and Constants . . . . .	11
2.2.1 Reduced units . . . . .	11
2.2.2 Other parameters . . . . .	11
2.3 Initial Configuration . . . . .	13
2.3.1 Initializing unit cells . . . . .	13
2.3.2 Initializing atom positions . . . . .	14
2.3.3 Initializing molecular orientations . . . . .	16
2.4 Initial Time Derivatives of Motion . . . . .	19
2.5 Interaction Functions and Definitions . . . . .	20
2.5.1 Lennard-Jones interaction . . . . .	20
2.5.2 Coulomb interaction . . . . .	21
2.5.3 Translational acceleration . . . . .	21
2.5.4 Torque and rotational acceleration . . . . .	23
2.6 Integration . . . . .	24
2.6.1 Predictor-Corrector Technique . . . . .	24
<b>Chapter 3 Results</b> . . . . .	<b>26</b>
3.1 Energy Conservation . . . . .	26
3.2 Temperature equilibrium . . . . .	29
3.3 GPU benchmarking . . . . .	32
<b>Chapter 4 Summary and future work</b> . . . . .	<b>34</b>
4.1 Summary . . . . .	34
4.2 Future Work . . . . .	35
<b>References</b> . . . . .	<b>36</b>

# List of Tables

2.1	TIP4P/2005 water model parameters . . . . .	10
2.2	Comparison of reduced and standard units . . . . .	11
2.3	Description of the eight molecules inside the smallest possible orthorhombic unit cell for ice-Ih in terms of the cell lengths . . . . .	14
2.4	Unit vectors <b>c</b> and <b>b</b> describing water orientations at each of the four crystal sites . . . . .	15
2.5	Predictor-corrector integrator coefficients for $k = 4$ . . . . .	25

# List of Figures

1.1	Comparison of floating-point operations per second and memory bandwidth among historical GPUs and CPUs . . . . .	2
1.2	Hierarchy of CUDA threads and blocks . . . . .	3
1.3	Hierarchy of CUDA memory . . . . .	4
2.1	Schematic representation of the TIP4P family of water models . . . . .	11
2.2	Euler angle representation of a finite rotation . . . . .	16
2.3	Graphical representation of the Lennard-Jones potential . . . . .	20
3.1	Energy conservation for double precision simulations with 64 molecules over 6.25 picoseconds.	27
3.2	Energy conservation for floating point simulations with 64 molecules over 6.25 picoseconds . .	28
3.3	Comparison of translational and rotational temperatures obtained using a 300.15 K Berendsen thermostat in a system of 512 water molecules . . . . .	30
3.4	Phases of water generated using the Berendsen thermostat in a system of 512 molecules over 62.5 ps . . . . .	31
3.5	Performance of parallel GPU and serial CPU versions as a function of the number of molecules in a system . . . . .	32
3.6	GPU performance as a function of the number of molecules in a system . . . . .	33



# Chapter 1

## Introduction

### 1.1 GPU Computing

In high-performance computing, there exists a constant demand for increased processing power. Until recently, computers had used “single-core” central processing units (CPUs), or computer chips containing only one independent processor (or core), to execute program instructions. However, as the circuit density of computer chips begins to approach hard physical limits [1], the single-core processor may soon attain its maximum potential. The death of the single-core processor may be evidenced by the relatively recent manufacture of multiple-core CPUs. Major chip manufacturers no longer aim to substantially increase the speed of single-core processors but rather increase the number of processors per chip. Presently, personal computers have between two to eight independent processing cores present in their CPU.

Single-core processors perform program instructions in sequence. Multi-core processors, on the other hand, have the ability to run multiple instructions synchronously (or “in parallel”), increasing the speed of programs which may be decomposed into several data-independent functions. However, as multi-core processors commonly share pathways to system memory, conflicts for memory access may be relatively frequent. For this reason among others [1], it is difficult to attain the anticipated  $n$ -fold speedup from an  $n$ -core CPU.

As multi-core CPU technology remains in its infancy, there are several alternatives to obtain the processing power required by computationally-demanding applications. One such alternative is the use of computer clusters, which contain many instances of low-cost hardware distributed across high-speed local networks. However, such clusters usually attract a large number of users and are often quite difficult to build and maintain. Another alternative is the use of supercomputers, which share similar problems. A more recent, cost-effective solution has been the use of graphics processing units (GPUs) [2].

Unlike a CPU, GPUs devote themselves almost entirely to intensive data processing rather than caching (storing memory) or flow control (managing data transmission). Consequently, today’s GPUs contain hundreds of processing cores capable of executing many calculations simultaneously. These properties make

GPUs well-equipped to handle problems that can be interpreted as mutually exclusive, data-parallel computations. In such problems, as similar functions are executed on each data element, there is little need for sophisticated flow control. Also, as such applications are extremely computationally-intensive, delays due to memory access may be reduced by “hiding” data access and GPU-CPU data transfer operations behind large calculations [3, 4].

Figure 1.1 displays the very recent evolution of the GPU as a computational powerhouse. It has grown to be not only a powerful graphics engine but also a massively parallel, programmable processor with arithmetic bandwidth much greater than its CPU counterpart. The rapid increase of GPUs in both capability and programmability has allowed for the successful mapping of several computationally-demanding problems to the GPU [4]. Notably, the recent development of the NVIDIA® CUDA™ programming environment has allowed for easy access to processing units on NVIDIA® GPUs, thereby allowing for easily-obtained speedups in parallelizable programs.

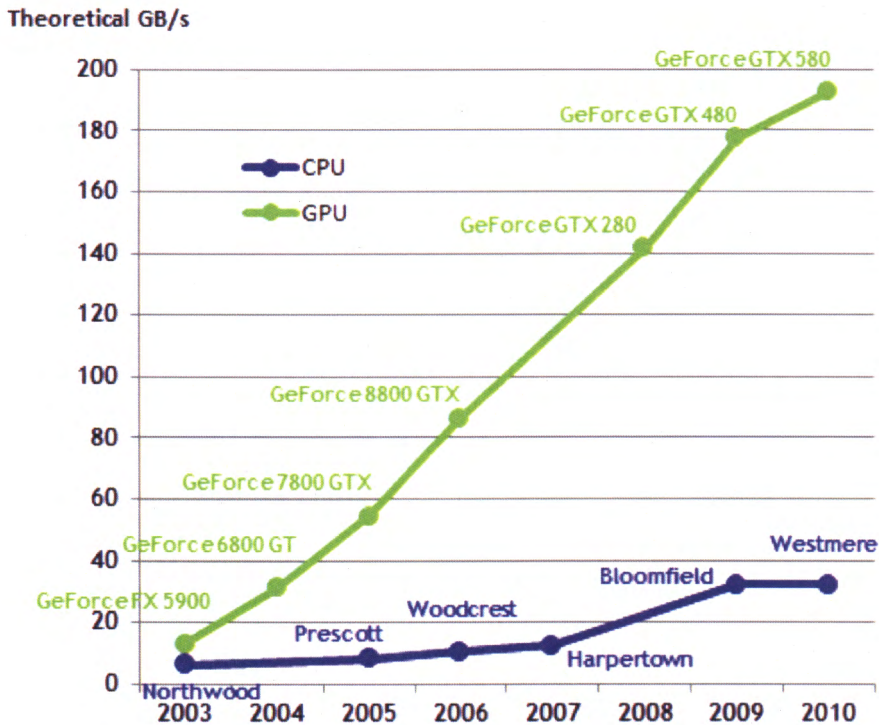


Figure 1.1: Comparison of floating-point operations per second and memory bandwidth among historical GPUs and CPUs [3].

### 1.1.1 CUDA programming basics

CUDA allows the programmer to create a parallel program partitioned into a hierarchy of *blocks* and *threads*. In a single-GPU parallel program, the GPU’s computational *grid* is first partitioned into a user-defined

number of blocks. Each block is then divided into a number of threads. These threads are the functional building blocks of the program; each thread points to data elements for the program to operate on in parallel. This computational hierarchy is more clearly illustrated in figure 1.2.

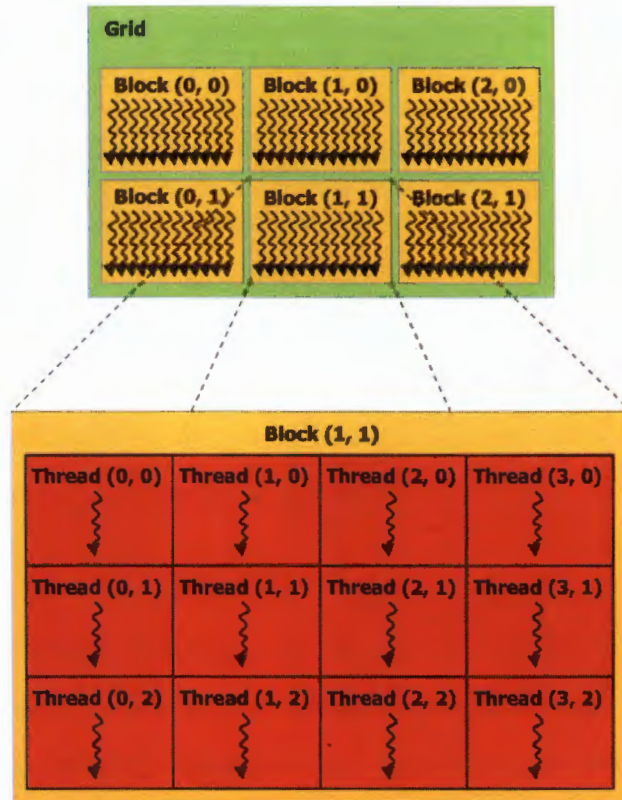


Figure 1.2: Hierarchy of CUDA threads and blocks [3].

CUDA *kernel* functions are then defined by the programmer. Kernels contain the instructions used to operate on data elements within each thread. When called, these functions are executed  $N$  times in parallel by  $N$  different threads. During execution, these threads may access data from three primary locations: private (local) memory, block memory, and global memory [3]. This memory hierarchy is illustrated in figure 1.3.

To serve as an example of CUDA syntax and structure, the *CUDA C Programming Guide* provides a sample program describing the addition of two  $N \times N$  matrices [3]:

```
// Kernel definition
__global__ void MatAdd( float A[N][N], float B[N][N], float C[N][N] )
{
    int i = threadIdx.x;
    int j = threadIdx.y;
```

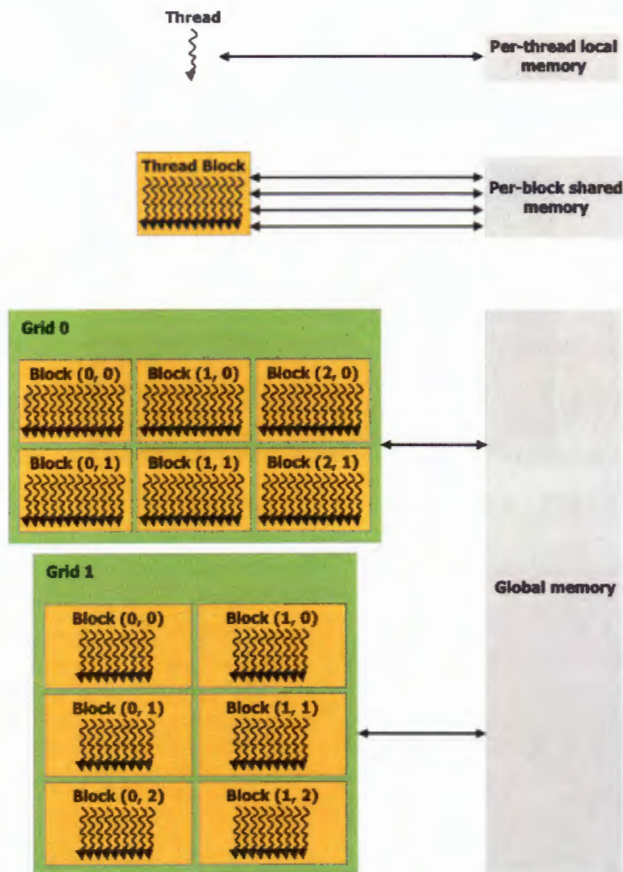


Figure 1.3: Hierarchy of CUDA memory defining the relationships of private memory, block memory, and global memory to threads, blocks, and grids [3].

```

C[i][j] = A[i][j] + B[i][j];
}

int main( )
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<< numBlocks, threadsPerBlock >>>(A, B, C);
    ...
}

```

In this program, `MatAdd` is defined as a kernel function which adds the elements of two  $N \times N$  matrices, **A** and **B**, and stores the result as matrix **C**. In the kernel,  $i$  and  $j$  represent the row and column positions of

each matrix element, respectively. Within the CPU program, the computational grid is first partitioned into one block. This block is then divided into  $N \times N$  threads; each thread points to one element of an  $N \times N$  matrix. The kernel function is then called by the CPU, and the corresponding elements of matrices **A** and **B** are added concurrently. This example serves as a good demonstration of CUDA's syntax and ease-of-use.

## 1.2 GPU-Accelerated Molecular Dynamics

In molecular systems with large numbers of particles, it is often impossible to determine properties of such systems analytically. Consequently, it is necessary to simulate these systems using numerical techniques. One such approach, molecular dynamics (MD), is regarded as one of the most powerful computational tools in biochemistry. In molecular systems with large numbers of particles, MD employs definitions of molecular mechanics force fields to estimate forces and potential energies among particles. MD then uses the laws of classical mechanics to integrate the motion of interacting particles over time [5]. Although MD proves useful in *visualizing* chemical and biological systems, it is more readily used to determine thermodynamic properties of large systems by computing statistical averages of various energetic quantities. These quantities can be used to characterize temperature-related phenomena (*e.g.*, phase change behavior), diffusion processes, viscosity, adsorption, pressure, and much more.

As similar force fields are computed for each particle in MD, such simulations may be highly parallelized. For example, one could assign each processor (or thread) a set of atoms or molecules upon which to operate. One might also assign each processor a number of interactions to compute. One may also assign each processor a fixed spatial region of the system to work with. MD simulations also do not require large amounts of memory; only position and energy information of each atom or molecule must be stored. Thus, such simulations are usually "large" in only two respects: the number of particles and number of timesteps [6].

As researchers in the field of MD continue to study increasingly large biomolecules and cellular processes occurring on longer time scales, the demand for increased processing power has continued to grow. Over the last decade, general-purpose MD codes such as LAMMPS [6], GROMACS [7], and NAMD [8] have been developed to run on distributed-memory computer clusters [9]. More recently, these and many other packages have begun to be ported to GPU-compatible code. Documented attempts to produce MD simulations that harness the power of GPUs have been very successful, in some cases providing a performance that is thirty to seven hundred times greater than that achieved with a single-core CPU [9, 10]. Thus, in building modern MD simulations, scientists should take advantage of the GPU's computational capabilities.

## 1.3 Motivation to Study Long-Range Interactions in Molecular Systems

All simulations of physical systems employ the use of numerical algorithms and approximations to accurately mimic a system’s behavior. In classical MD, there exists an important tradeoff between the desired accuracy and time scale of a simulation. To decrease computational load, MD simulations have often ignored long-range physical interactions among particles after some arbitrary “cutoff distance.” In many cases, as cutoff distance is decreased, the accuracy of a simulation significantly degrades [11, 12, 13].

In biochemical systems with relatively large amounts of polar water molecules, the error introduced through the use of cutoff distances or other approximation schemes could be significant. The aim of this research is to eventually determine if long-range coulombic contributions by polar water molecules have a nontrivial impact on the accuracy of biochemical MD simulations.

As computational technology has advanced, classical MD simulations containing tens of thousands of atoms with time spans on the order of nanoseconds have become commonplace. In all such simulations, atoms generate forces on one another. Historically, there was usually insufficient computational power to perform all  $N^2$  force calculations required by large systems. Thus, several simple approximation techniques were utilized [14]. It is well-known that the calculation of electrostatic forces is the most computationally-expensive component of MD simulations. In the past, to reduce this bottleneck, simulations had either neglected long-range coulombic interactions or truncated them at some cutoff distance [12]. More recently, rather than resorting to the use of arbitrary cutoffs, several methods for computing long-range interactions have been developed and employed. In particular, Ewald summation methods [15], fast particle mesh methods [16], and fast multipole methods [17] have been used [14]. However, these methods also carry degrees of uncertainty and are often quite difficult to implement.

In many biomolecular systems, we argue long-range electrostatic interactions deserve the utmost attention. For example, in biomembrane systems, phospholipid molecules that make up the lipid bilayer are polar and charged. Clearly, the long-range electrostatic interactions of phospholipids with polar water and other charged molecules could play a large role in determining the overall behavior of the system. Nonetheless, in simulations containing large amounts of such molecules, researchers had truncated interactions at some arbitrary distance (typically 1.5-2.0 nm). In 2004, Patra, *et al.* recognized all examined truncation distances (1.8-2.5 nm) led to major effects on bilayer properties, all of which were significantly inconsistent with experimental data. Patra, *et al.* concluded “the truncation of electrostatic interactions may lead to profound artifacts in the properties of lipid bilayer systems, and should be used with great care, if at all” [12].

The use of cutoffs in MD simulations of solvated peptides, or short amino acid chains surrounded by a water solvent, was also found to entail significant numerical artifacts. In one such study, three coulombic cutoff radii (0.6, 1.0, and 1.4 nm) were tested in a system containing a model peptide in the presence of over one thousand water molecules. Although the stability of an alpha-helical (coiled) structure was observed using the cutoff radius of 1.0 nm, this stability was peculiarly lost as the cutoff was increased to 1.4 nm. The study concluded “even 1.4 nm [was] too short for a cutoff” and “this truncation scheme seems questionable for molecular dynamics simulations of solvated biomolecules” [13].

In another study, long-range electrostatic interactions in a system of a highly charged oligonucleotide (a short nucleic acid polymer) immersed in aqueous solution were also found to be of large consequence. The two tested truncation methods in the study failed to accurately mimic experimental behavior. Nonetheless, by employing particle mesh methods for long-range interactions, the simulation was found to be stable and accurate [11].

Clearly, it is reasonable to assume that in biochemical systems with relatively large amounts of water, coulombic interactions associated with surrounding polar and/or charged molecules contribute greatly to the accuracy of MD simulations. Because many biochemical processes take place either in or surrounded by an aqueous environment, it is of value to consider and examine the effects of long-range electrostatic interactions from polar water molecules on the accuracy of biochemical MD studies. In such testing, it would be crucial to compare results with those gained via truncation methods and summation/mesh methods. It is our hope the results of such tests would lead to the adoption of more accurate water approximation schemes by the MD community.

## 1.4 Research Objectives

Commonly, MD simulations are developed using specialized, prebuilt software packages. Naturally, these packages can be either difficult or impossible to modify in order to suit a particular study. Thus, we have chosen to build our MD simulations entirely from scratch to allow for full customizability of all accuracy-related components of the code. With the recent advent of GPU programming, we will have sufficient computational power to more accurately estimate long-range water-molecule interactions in biochemical systems, allowing us to test our aforementioned predictions. Building such code from the ground up also serves an important pedagogical purpose.

We have begun by first developing a classical MD simulation of water molecules for the GPU. As we are primarily interested in simulating biomolecules in the presence of water, the use of a simple, rigid (*i.e.*, fixed-

geometry) water model has been employed in our simulations. Our current research aims to validate this water model's accuracy by comparing numerically-obtained properties of the system with experimentally-validated thermodynamic quantities and plots. In the process of continually developing our code, the GPU simulation will be further optimized and benchmarked, and more sophisticated algorithms will be implemented to decrease the simulation's computational load while maintaining sufficient accuracy.



# Chapter 2

## Model and methodology

### 2.1 Water Models

In MD, to characterize the physical properties of solvated molecules (*i.e.*, molecules surrounded by a solvent such as water), one must first accurately describe the water contained within the system. This can be accomplished by employing either *implicit* or *explicit* solvation methods. Implicit solvation methods represent water as a continuous medium, while explicit solvation methods represent water as discrete, interacting molecules. Although implicit solvation methods are relatively quick and have shown promise in the simulation of larger systems, they often fail to reproduce the finer details of many microscopic systems. For this reason, MD simulations most often employ explicit solvation methods [18].

In MD, explicit water models may be used to approximate the shape and interaction site distributions of discrete water molecules. Such models are most readily used in the simulation of water clusters, liquid water, and molecules immersed in or surrounded by an aqueous environment. A variety of explicit models have been proposed, each of which may be categorized by three features: the number of particles used to define the model, whether the molecule has a fixed or flexible geometry, and whether the model includes polarization effects. Water models describe the estimated electrostatic charge sites for coulombic potentials as well as the Lennard-Jones site which accounts for the Lennard-Jones potential. The Lennard-Jones potential approximates the interaction between a pair of electrically neutral atoms or molecules and is further described in section 2.5.

The goal of producing MD simulations in aqueous environments is to accurately characterize the physical properties of solvated systems. As molecules such as proteins and nucleic acids are typically highly charged, long-range electrostatic interactions must be properly accounted for. Thus, accurate descriptions of molecule-water interactions are necessary. As water-water interactions produce the highest computational cost for most MD simulations, it is important to use a water model which is both fast and accurate. More specifically, a good water model should reproduce statistical bulk water properties over a wide range of temperatures and physical states [18].

The simplest water models treat water molecules as rigid bodies and rely only on interactions with particles outside the molecule. Such models describe a water molecule as a system with  $n$  sites, with  $n$  often ranging from two to six. As the number of sites in a rigid water model increases, so usually does the model’s accuracy in describing bulk properties. Thus, different models may be used for different applications and approximations. For example, three-site models have three interaction “sites” corresponding to the three atoms of a water molecule. Each atom is assigned a charge, and the oxygen atom is assigned an additional Lennard-Jones site. A five-site model, on the other hand, may include representations of the lone pairs of oxygen as charged “dummy atoms” to improve the estimate of the molecule’s charge distribution [19].

### 2.1.1 TIP4P/2005

Four-site models place one negative charge on a dummy atom placed near the oxygen along the bisector of the HOH angle, intended to improve the charge distribution around the molecule. These models also place two positive charge sites on the two hydrogen atoms and one Lennard-Jones site on the oxygen atom. Historical four-site models did not reproduce bulk properties of water well, but more recent parameterizations have been developed for general-purpose MD simulations. The TIP4P model, in particular, was geared toward accurately reproducing the qualitative behavior of the entire phase diagram of water. However, the ice regime of the phase diagram was poorly estimated by this model [20].

The TIP4P/2005 model provides a substantial improvement on the TIP4P model and works well in simulations ranging in temperatures from 123 to 573 K and pressures up to 40,000 bar. This model provides impressive performance for a variety of thermodynamic states and has been found to be significantly more accurate than the TIP4P model [20]. Considering the general nature of future simulations, the choice of TIP4P/2005 as our working water model seems reasonable. Figure 2.1 and table 2.1 describe the parameterization of the TIP4P/2005 water model.  $r_{OM}$  defines the distance from the oxygen atom to the negatively charged dummy atom,  $M$ ;  $r_{OH}$  defines the distance from the oxygen atom to a hydrogen atom;  $\sigma$  defines the nearest-neighbor oxygen-oxygen separation distance;  $\epsilon/k_B$  defines the ratio of the depth of the Lennard-Jones potential well,  $\epsilon$ , to the Boltzmann constant,  $k_B$ ; and  $q_O$ ,  $q_H$ , and  $q_M$  represent the charges on the oxygen, hydrogen, and dummy atoms.

$r_{OH}$ (Å)	$\angle$ HOH (degrees)	$\sigma$ (Å)	$\epsilon/k_B$ (K)	$q_O$ (e)	$q_H$ (e)	$q_M$ (e)	$r_{OM}$ (Å)
0.9572	104.52	3.1589	93.2	0	0.5564	$-2q_H$	0.1546

Table 2.1: TIP4P/2005 water model parameters [20].

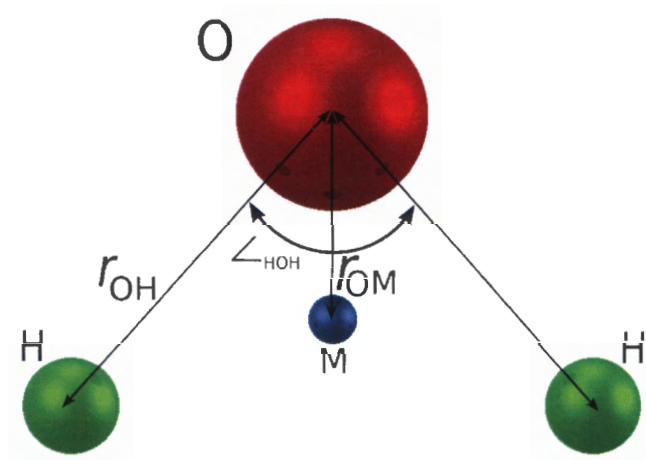


Figure 2.1: Schematic representation of the TIP4P family of water models (courtesy of Carl McBride).

## 2.2 Parameters and Constants

### 2.2.1 Reduced units

Physical quantities in MD are represented as dimensionless or reduced units, placing all values of interest near unity. By working in reduced units, simulation results are scaled to reasonable magnitudes, allowing for easy interpretation. Such scaling also results in minimal floating-point rounding errors accumulated during computation. In MD, the kcal-Ångstrom system often defines these units. In table 2.2, reduced units are compared with their SI counterparts:

System	Length	Mass	Time	Charge	Temp.	Velocity	Energy	Force
kcal-Ångstrom	Ångstrom	Dalton	picosecond	e	Kelvin	100 m/s	kcal/mol	kcal/(mol s)
SI	meter	kilogram	second	Coulomb	Kelvin	m/s	J	N

Table 2.2: Comparison of reduced (kcal-Ångstrom) and standard units.

### 2.2.2 Other parameters

In addition to the parameters defined by the TIP4P/2005 water model in table 2.1, a number of other constant numerical and physical parameters were initialized near the beginning of the program. In our simulation, variable parameters were defined by the user in `input.dat` and included the desired system temperature, timestep, size of system in the  $x$ ,  $y$ , and  $z$  directions, total number of timesteps, and number of GPU threads per block. In most cases, the timestep was defined as  $6.25 \times 10^{-4}$  picoseconds, or 0.625 femtoseconds. As we were required to describe both the acceleration of each molecule's center of mass as well as the inertia tensor of the TIP4P/2005 water molecule, we defined  $m_H$ , the mass of a hydrogen atom,

as 1.00794 Daltons and  $m_O$ , the mass of an oxygen atom, as 15.9994 Daltons. Of course, we defined the mass of the entire water molecule as  $m_{CM} = 2m_H + m_O$ . From these masses and the parameters listed in table 2.1, we defined the center of mass position relative to the oxygen atom as

$$r_{OCM} = \frac{1}{m_{CM}} \left( 2m_H r_{OH} \cos \frac{\theta}{2} \right), \quad (2.1)$$

where  $\theta$  is the HOH angle defined in table 2.1.

We also defined body-fixed unit vectors  $\hat{x}'$ ,  $\hat{y}'$ , and  $\hat{z}'$  attached to the water molecule. We assumed  $\hat{z}'$  to point along the HOH angle bisector toward the hydrogen atoms and  $\hat{x}'$  to point toward the leftmost hydrogen atom displayed in figure 2.1. Using these definitions, we described the body-fixed positions of the the atoms (with the center of mass as the origin) as

$$H_{1,x'} = r_{OH} \sin \frac{\theta}{2}, \quad H_{1,y'} = 0, \quad H_{1,z'} = r_{OH} \cos \frac{\theta}{2} - r_{OCM}, \quad (2.2)$$

$$H_{2,x'} = -r_{OH} \sin \frac{\theta}{2}, \quad H_{2,y'} = 0, \quad H_{2,z'} = r_{OH} \cos \frac{\theta}{2} - r_{OCM}, \quad (2.3)$$

$$O_{x'} = 0, \quad O_{y'} = 0, \quad O_{z'} = -r_{OCM}, \quad (2.4)$$

$$M_{x'} = 0, \quad M_{y'} = 0, \quad M_{z'} = -r_{OCM} + R_{OM}. \quad (2.5)$$

From these body-fixed positions, we described the diagonalized inertia tensor with elements  $I_{xx}$ ,  $I_{yy}$ , and  $I_{zz}$  as the following:

$$I_{xx} = m_O O_{z'}^2 + m_H H_{1,z'}^2 + m_H H_{2,z'}^2, \quad (2.6)$$

$$I_{yy} = m_O O_{z'}^2 + m_H (H_{1,z'}^2 + H_{1,x'}^2) + m_H (H_{2,z'}^2 + H_{2,x'}^2), \quad (2.7)$$

$$I_{zz} = m_H H_{1,x'}^2 + m_H H_{2,x'}^2. \quad (2.8)$$

We also defined the Boltzmann constant,  $k_B$ , as  $1.9872065 \times 10^{-3}$  (kcal/mol)/K, epsilon as  $k(\frac{\epsilon}{k}) = 0.1852076458$  kcal/mol, and the coulomb constant,  $\frac{1}{4\pi\epsilon_0} = k_e$ , as  $3.32063711 \times 10^2$  kcal Angstroms/e<sup>2</sup>. All parameters not defined in `input.dat` were defined as global device variables in `init_vars.cu`. Parameters declared in `input.dat` were later copied from CPU to GPU memory before entering the simulation's main loop.

## 2.3 Initial Configuration

### 2.3.1 Initializing unit cells

Simulations of ice-Ih (hexagonal ice) are of general interest to the molecular dynamics community. In particular, this representation of ice has been used widely in the simulation of ice surfaces, especially in the study of melting and freezing phenomena. Historically, these simulations had involved the use of large (48- or 96-molecule) periodic unit cells to describe an ice-Ih system with sufficient random hydrogen-bonding patterns. Hayward and Reimers instead described a method for generating sets of internally-consistent, minimal net-dipole water lattices from a small, eight-molecule unit cell [21].

In ice-Ih, each oxygen atom lies at the intersection of two hexagonal lattices. When hydrogen atoms are also considered, at least four water molecules are required to comprise a unit cell. The four-membered unit cell is hexagonal, and any number of these cells may be added to form a larger unit cell, allowing for more random hydrogen orientations. In our simulation, two of such unit cells were combined to form an eight-membered unit cell which had overall orthorhombic symmetry. As orthorhombic unit cells are advantageous in computer simulations, we considered these cells the basic building blocks of our simulation. Each cell was defined with spatial dimensions [21]

$$x_\ell^0 = \sigma\sqrt{8/3}, \quad y_\ell^0 = \sigma\sqrt{8}, \quad z_\ell^0 = 8\sigma/3, \quad (2.9)$$

where  $\sigma$  again represents the nearest-neighbor oxygen-oxygen separation. We constructed a “large” system of ice by replicating this building block by translation  $n_x$ ,  $n_y$ , and  $n_z$  times in the  $x$ ,  $y$ , and  $z$  directions, generating an ice structure containing a total of  $N = 8n_xn_yn_z$  water molecules. The system dimensions  $x_\ell$ ,  $y_\ell$ , and  $z_\ell$  were then described as

$$x_\ell = n_x x_\ell^0, \quad y_\ell = n_y y_\ell^0, \quad z_\ell = n_z z_\ell^0. \quad (2.10)$$

It was first necessary to describe the position of each unit cell with respect to the entire system. In our initialization procedure, we first assigned a thread index to each molecule by assigning the label `intid = threadIdx.x + blockIdx.x * blockDim.x`. The molecule index within an orthorhombic cell was then described as `id % 8`. We then defined the relative location of each unit cell using code similar to the following:

```
cell_index = id / 8;
k = cell_index / ( nx * ny );
```

```

cell_index = cell_index - k * nx * ny;
j = cell_index / nx;
i = cell_index - j * nx;

```

where  $i$ ,  $j$ , and  $k$  corresponded to integer labels of a unit cell in the  $x$ ,  $y$ , and  $z$  directions.

### 2.3.2 Initializing atom positions

After defining molecular locations both within a unit cell and with respect to the entire system, we initialized the positions of atoms. As the positions of atoms within each molecule could be easily related to the position of each oxygen atom, we first initialized the positions of oxygen atoms within the system. Hayward and Reimers defined the locations of each molecule within a unit cell in table 2.3.

Molecule	Site	$4x/x_\ell^0$	$6y/y_\ell^0$	$16z/z_\ell^0$
1	1	1	1	3
2	2	3	2	5
3	3	1	1	13
4	4	3	2	11
5	1	3	4	3
6	2	1	5	5
7	3	3	4	13
8	4	1	5	11

Table 2.3: Description of the eight molecules inside the smallest possible orthorhombic unit cell for ice-Ih in terms of cell lengths [21].

After assigning values of  $4x/x_\ell^0$ ,  $6y/y_\ell^0$ , and  $16z/z_\ell^0$  to each molecule in the system, we then initialized positions of the oxygen atoms using the following relations:

$$O_x = \left(\frac{4x}{x_\ell^0}\right)\left(\frac{x_\ell^0}{4}\right) + ix_\ell^0, \quad O_y = \left(\frac{6y}{y_\ell^0}\right)\left(\frac{y_\ell^0}{6}\right) + jy_\ell^0, \quad O_z = \left(\frac{16z}{z_\ell^0}\right)\left(\frac{z_\ell^0}{16}\right) + kz_\ell^0, \quad (2.11)$$

where  $i$ ,  $j$ , and  $k$ , again, represent a unit cell's indices with respect to the system in the  $x$ ,  $y$ , and  $z$  directions.

At each of the four sites in the unit cell, there existed six possibilities for the orientation of hydrogen atoms. Thus, there were a total of twenty-four unique water orientations. The possible arrangements for each site are listed in table 2.4 in terms of orthogonal unit vectors representing molecular-symmetric (c) and in-plane (b) axes [21]. These unit vectors were randomly assigned to each molecule, matching required site criteria. From these vectors as well as the position vector of the associated oxygen atom,  $\mathbf{O}$ , the positions of the two hydrogen atoms,  $\mathbf{H}_1$  and  $\mathbf{H}_2$ , were obtained using the water model's given O-H bond length,  $r_{OH}$ ,

and  $\angle HOH$ ,  $\theta$ , using the following relations:

$$\mathbf{H}_1 = \left( r_{OH} \cos \frac{\theta}{2} \right) \mathbf{c} + \left( r_{OH} \sin \frac{\theta}{2} \right) \mathbf{b} + \mathbf{O}, \quad (2.12)$$

$$\mathbf{H}_2 = \left( r_{OH} \cos \frac{\theta}{2} \right) \mathbf{c} - \left( r_{OH} \sin \frac{\theta}{2} \right) \mathbf{b} + \mathbf{O}. \quad (2.13)$$

The molecule's charged dummy atom and center of mass were then initialized using the following relations:

$$\mathbf{M} = (r_{OM})\mathbf{c} + \mathbf{O}, \quad (2.14)$$

$$\mathbf{CM} = (r_{OCM})\mathbf{c} + \mathbf{O}. \quad (2.15)$$

Site	Orientation	$c_x$	$c_y$	$c_z$	$b_x$	$b_y$	$b_z$
1	1	0	$(2/3)^{1/2}$	$(1/3)^{1/2}$	-1	0	0
	2	$-(1/2)^{1/2}$	$-(1/6)^{1/2}$	$(1/3)^{1/2}$	-1/2	$(3/4)^{1/2}$	0
	3	$-(1/2)^{1/2}$	$(1/6)^{1/2}$	$-(1/3)^{1/2}$	1/2	$-(1/12)^{1/2}$	$-(2/3)^{1/2}$
	4	$(1/2)^{1/2}$	$-(1/6)^{1/2}$	$(1/3)^{1/2}$	-1/2	$-(3/4)^{1/2}$	0
	5	$(1/2)^{1/2}$	$(1/6)^{1/2}$	$-(1/3)^{1/2}$	-1/2	$-(1/12)^{1/2}$	$-(2/3)^{1/2}$
	6	0	$-(2/3)^{1/2}$	$-(1/3)^{1/2}$	0	$(1/3)^{1/2}$	$-(2/3)^{1/2}$
2	1	$(1/2)^{1/2}$	$(1/6)^{1/2}$	$-(1/3)^{1/2}$	1/2	$-(3/4)^{1/2}$	0
	2	0	$-(2/3)^{1/2}$	$-(1/3)^{1/2}$	-1	0	0
	3	$(1/2)^{1/2}$	$-(1/6)^{1/2}$	$(1/3)^{1/2}$	-1/2	$(1/12)^{1/2}$	$(2/3)^{1/2}$
	4	$-(1/2)^{1/2}$	$(1/6)^{1/2}$	$-(1/3)^{1/2}$	-1/2	$-(3/4)^{1/2}$	0
	5	0	$(2/3)^{1/2}$	$(1/3)^{1/2}$	0	$-(1/3)^{1/2}$	$(2/3)^{1/2}$
	6	$-(1/2)^{1/2}$	$-(1/6)^{1/2}$	$(1/3)^{1/2}$	1/2	$(1/12)^{1/2}$	$(2/3)^{1/2}$
3	1	0	$(2/3)^{1/2}$	$-(1/3)^{1/2}$	-1	0	0
	2	$-(1/2)^{1/2}$	$-(1/6)^{1/2}$	$-(1/3)^{1/2}$	-1/2	$(3/4)^{1/2}$	0
	3	$-(1/2)^{1/2}$	$(1/6)^{1/2}$	$(1/3)^{1/2}$	1/2	$-(1/12)^{1/2}$	$(2/3)^{1/2}$
	4	$(1/2)^{1/2}$	$-(1/6)^{1/2}$	$-(1/3)^{1/2}$	-1/2	$-(3/4)^{1/2}$	0
	5	$(1/2)^{1/2}$	$(1/6)^{1/2}$	$(1/3)^{1/2}$	-1/2	$-(1/12)^{1/2}$	$(2/3)^{1/2}$
	6	0	$-(2/3)^{1/2}$	$(1/3)^{1/2}$	0	$(1/3)^{1/2}$	$(2/3)^{1/2}$
4	1	$(1/2)^{1/2}$	$(1/6)^{1/2}$	$(1/3)^{1/2}$	1/2	$-(3/4)^{1/2}$	0
	2	0	$-(2/3)^{1/2}$	$(1/3)^{1/2}$	-1	0	0
	3	$(1/2)^{1/2}$	$-(1/6)^{1/2}$	$-(1/3)^{1/2}$	-1/2	$(1/12)^{1/2}$	$-(2/3)^{1/2}$
	4	$-(1/2)^{1/2}$	$(1/6)^{1/2}$	$(1/3)^{1/2}$	-1/2	$-(3/4)^{1/2}$	0
	5	0	$(2/3)^{1/2}$	$-(1/3)^{1/2}$	0	$-(1/3)^{1/2}$	$-(2/3)^{1/2}$
	6	$-(1/2)^{1/2}$	$-(1/6)^{1/2}$	$-(1/3)^{1/2}$	1/2	$(1/12)^{1/2}$	$-(2/3)^{1/2}$

Table 2.4: Unit vectors  $\mathbf{c}$  and  $\mathbf{b}$  describing all six possible water orientations at each of the four crystal sites [21].

### 2.3.3 Initializing molecular orientations

The motion of a rigid body can be separated into two components: translation of the center of mass and rotation of particles about the center of mass. Thus, rigid body dynamics often requires two frames: the space-fixed frame and the body-fixed frame. In section 2.2.2, we described the body-fixed coordinate system as well as the body-fixed inertia tensor common to all water molecules. These two definitions proved useful in our treatment of molecules' rotational dynamics.

#### Euler angles

The orientation of a body-fixed coordinate system with respect to a space-fixed coordinate system may be described by three “Euler angles” ( $\alpha$ ,  $\beta$ ,  $\gamma$ ). These angles define three successive rotations from the space-fixed frame to the body-fixed frame. The first rotation occurs through an angle  $\alpha$  about the space-fixed  $z$  axis, resulting in a new coordinate system ( $\xi$ ,  $\eta$ ,  $\zeta$ ). This rotation is followed by a rotation of angle  $\beta$  around the new  $\eta$  axis, resulting in another coordinate system ( $\xi'$ ,  $\eta'$ ,  $\zeta'$ ). The final rotation is through an angle  $\gamma$  about the  $\zeta'$  axis, reproducing the body-fixed coordinate system ( $x'$ ,  $y'$ ,  $z'$ ) [22]. Euler angles and their derivations are more clearly illustrated in figure 2.2.

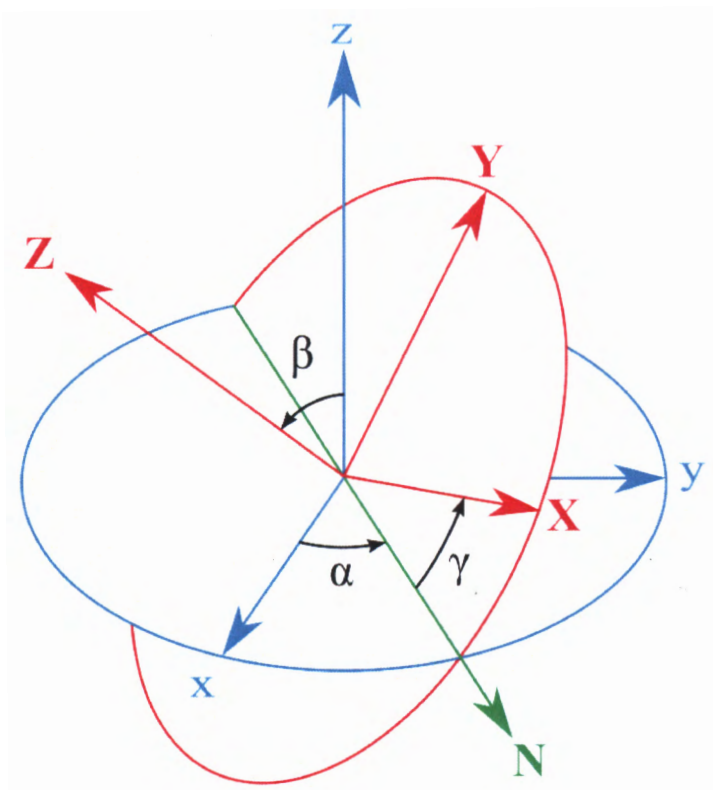


Figure 2.2: Euler angles representation of a finite rotation from a space-fixed frame (blue) to a rotated frame (red) (courtesy of Lionel Brits).



The initial Euler angles associated with each molecule were then determined using components of the unit vectors defined in table 2.4 [23]:

$$\alpha = \text{atan2}(c_x, -c_y), \quad \beta = \text{acos}(c_z), \quad \gamma = \text{atan2}(b_z, a_z), \quad (2.16)$$

where  $a_z$  is a component of the vector  $\mathbf{c} \times \mathbf{b}$ , *i.e.*,  $a_z = c_x b_y - c_y b_x$ .

Unfortunately, the use of Euler angles becomes unstable in integrating rotation as  $\beta$  approaches 0 or  $\pi$ . To avoid such inconvenience, we defined the initial orientations of molecules in terms of Euler angles, then expressed the rotational motion of the rigid body using Hamilton's *quaternions*, described in the following section.

### Quaternions

The simplest way to avoid the disadvantages of Euler angles is to instead use quaternions. Consider a rotation of a particle about a fixed point. In a space-fixed coordinate system, a vector  $\mathbf{r}$  may be rotated to  $\mathbf{r}'$  by applying the rotation

$$\mathbf{r}' = \mathcal{A}^T \mathbf{r}, \quad (2.17)$$

where  $\mathcal{A}$  is a  $3 \times 3$  matrix describing the rotation for all vectors  $\mathbf{r}$ . If  $\mathbf{r}(0)$  represents  $\mathbf{r}$  at  $t = 0$  in the space fixed system, a series of finite rotations may be represented by a single rotation if all rotations are made about a single point. Thus, we may write

$$\mathbf{r}(t) = \mathcal{A}^T(t) \mathbf{r}(0), \quad (2.18)$$

where  $\mathcal{A}(t)$  is the time-dependent rotation matrix with  $\mathcal{A}(0) = I$ , where  $I$  is the identity matrix [24]. More generally, we could write the above as

$$\mathbf{r}(t) = \mathcal{A}^T(t) \mathbf{r}_{ref}, \quad (2.19)$$

where the time-dependent space-fixed representation of the vector,  $\mathbf{r}(t)$ , is expressed as a time-dependent rotation of the reference state  $\mathbf{r}_{ref}$  such that the initial transformation matrix is no longer the identity matrix but rather the matrix that rotates the reference state to the orientation in the space-fixed axis at  $t = 0$ , *i.e.*,

$$\mathbf{r}(0) = \mathcal{A}^T(0) \mathbf{r}_{ref}. \quad (2.20)$$

In our case, the reference configuration is where the water molecule aligns with the space-fixed axis such that

$$\mathbf{b}_{ref} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{c}_{ref} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \quad (2.21)$$

The rotation of these vectors described the time-dependent alignment of a water molecule in the space-fixed system, *i.e.*,

$$\mathbf{b}(t) = \mathcal{A}^T(t) \mathbf{b}_{ref}, \quad \mathbf{c}(t) = \mathcal{A}^T(t) \mathbf{c}_{ref}. \quad (2.22)$$

In our case, we used quaternions to describe the rotation matrix  $\mathcal{A}$ . According to Rapaport [24],

$$\mathcal{A} = 2 \begin{pmatrix} \frac{1}{2} - q_2^2 - q_3^2 & q_1 q_2 + q_0 q_3 & q_1 q_3 - q_0 q_2 \\ q_1 q_2 - q_0 q_3 & \frac{1}{2} - q_1^2 - q_3^2 & q_2 q_3 + q_0 q_1 \\ q_1 q_3 + q_0 q_2 & q_2 q_3 - q_0 q_1 & \frac{1}{2} - q_1^2 - q_2^2 \end{pmatrix}, \quad (2.23)$$

and thus,

$$\mathcal{A}^T = 2 \begin{pmatrix} \frac{1}{2} - q_2^2 - q_3^2 & q_1 q_2 - q_0 q_3 & q_1 q_3 + q_0 q_2 \\ q_1 q_2 + q_0 q_3 & \frac{1}{2} - q_1^2 - q_3^2 & q_2 q_3 - q_0 q_1 \\ q_1 q_3 - q_0 q_2 & q_2 q_3 + q_0 q_1 & \frac{1}{2} - q_1^2 - q_2^2 \end{pmatrix}. \quad (2.24)$$

The representation of the rotation matrix in terms of quaternions proved useful since the time-dependent quaternion vector,

$$\mathbf{q}(t) \equiv \begin{pmatrix} q_0(t) \\ q_1(t) \\ q_2(t) \\ q_3(t) \end{pmatrix}, \quad (2.25)$$

could be described by a simple set of dynamical equations and integrated similarly to translational motion but also subject to the constraint

$$\sum_{m=0}^3 q_m^2 = 1. \quad (2.26)$$

Such a constraint was expected, as rotations may always be described via three independent parameters.

In terms of quaternions, then, we obtained the following expressions for the time-dependent vectors  $\mathbf{c}$

and  $\mathbf{b}$  describing the orientation of a water molecule:

$$\mathbf{c}(t) = \begin{pmatrix} c_x(t) \\ c_y(t) \\ c_z(t) \end{pmatrix} = \mathcal{A}^T(t) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2(q_1q_3 + q_0q_2) \\ 2(q_2q_3 - q_0q_1) \\ 2\left(\frac{1}{2} - q_1^2 - q_2^2\right) \end{pmatrix}, \quad (2.27)$$

$$\mathbf{b}(t) = \begin{pmatrix} b_x(t) \\ b_y(t) \\ b_z(t) \end{pmatrix} = \mathcal{A}^T(t) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2\left(\frac{1}{2} - q_2^2 - q_3^2\right) \\ 2(q_1q_2 + q_0q_3) \\ 2(q_1q_3 - q_0q_2) \end{pmatrix}. \quad (2.28)$$

Before dynamical equations were used to determine  $\mathbf{q}(t)$  and, thus,  $\mathbf{b}(t)$  and  $\mathbf{c}(t)$ , the initial quaternion vector was defined for the rigid body. This was necessary to accurately reflect the initial molecular orientation axes,  $\mathbf{b}(0)$  and  $\mathbf{c}(0)$ . To do so, we first used the Euler angle representation of the quaternion vector [24]:

$$\mathbf{q}(0) = \begin{pmatrix} q_0(0) \\ q_1(0) \\ q_2(0) \\ q_3(0) \end{pmatrix} = \begin{pmatrix} \cos(\beta/2) \cos \frac{1}{2}(\alpha + \gamma) \\ \sin(\beta/2) \cos \frac{1}{2}(\alpha - \gamma) \\ \sin(\beta/2) \sin \frac{1}{2}(\alpha - \gamma) \\ \cos(\beta/2) \sin \frac{1}{2}(\alpha + \gamma) \end{pmatrix}, \quad (2.29)$$

where the Euler angles  $(\alpha, \beta, \gamma)$  were defined via the conventions described in the previous subsection. In this case, the Euler angles represented the rotation from a reference state to the initial state.

Again, in the reference state,  $\mathbf{c}$  aligned with the positive  $z$ -direction,  $\hat{z}$ . Thus, we identified  $\mathbf{c}$  with the body-fixed  $z$  direction,  $\hat{z}'$ , as described by Euler angle conventions:  $\mathbf{c} = \hat{z}'$ . Likewise, we identified  $\mathbf{b}$  with the body-fixed  $x$  direction,  $\hat{x}'$ , *i.e.*,  $\mathbf{b} = \hat{x}'$ . The third axis in the body-fixed system was defined by the cross-product  $\hat{y}' = \hat{z}' \times \hat{x}'$ .

## 2.4 Initial Time Derivatives of Motion

Prior to beginning the simulation's main loop, the first time derivatives describing translational and rotational motion were initialized. For the sake of simplicity, we defined these to be zero, *i.e.*,

$$\dot{\mathbf{r}}_{CM}(0) = 0, \quad \dot{\mathbf{q}}(0) = 0. \quad (2.30)$$

## 2.5 Interaction Functions and Definitions

### 2.5.1 Lennard-Jones interaction

In the interaction of two neutral atoms, an attractive van der Waals force balances a repulsive force resulting from the overlapping of electron orbitals. A commonly used approximation for such an interaction is called the Lennard-Jones potential (also called the LJ potential or 12-6 potential) proposed by John Lennard-Jones in 1924 [25]. The most common expression for the Lennard-Jones potential is the following:

$$V_{LJ,ij} = 4\epsilon \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right], \quad (2.31)$$

where  $\epsilon$  is the depth of the Lennard-Jones potential well,  $\sigma$  is the distance between the two interacting Lennard-Jones particles at which the potential is zero, and  $r_{ij}$  is the distance between two interacting particles labeled  $i$  and  $j$ . The net Lennard-Jones contribution to the potential energy of an atom  $i$  in a system of  $N$  Lennard-Jones atoms is thus

$$V_{LJ,i} = 4\epsilon \sum_{i \neq j}^N \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right]. \quad (2.32)$$

Upon inspection of 2.31, it is of note that the Lennard-Jones potential consists of two parts: a strongly repulsive term at small distances and an increasingly weak attractive term at larger distances. The Lennard-Jones potential is displayed graphically in figure 2.3. As described in discussion of the TIP4P/2005 water

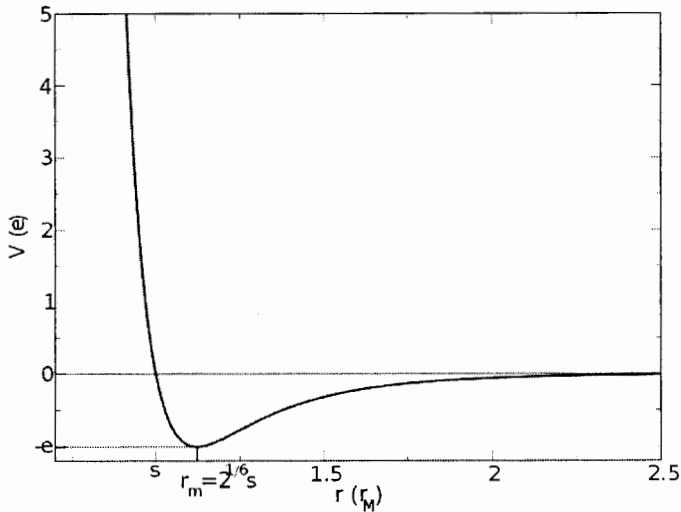


Figure 2.3: Graphical representation of the Lennard-Jones potential (courtesy of Olaf Lenz).

model, the Lennard-Jones site for each water molecule is placed on the molecule's oxygen atom. Thus, only

oxygen-oxygen interactions produce a nonzero Lennard-Jones potential in TIP4P/2005 water.

The Lennard-Jones force experienced by a Lennard-Jones particle  $i$  from an interaction with a Lennard-Jones particle  $j$  may be expressed as

$$\mathbf{F}_{LJ,ij} = (\mathbf{r}_i - \mathbf{r}_j) \frac{48\epsilon}{\sigma^2} \left[ \left( \frac{\sigma}{r_{ij}} \right)^{14} - \frac{1}{2} \left( \frac{\sigma}{r_{ij}} \right)^8 \right]. \quad (2.33)$$

Thus, the net Lennard-Jones force exerted by all  $N$  oxygen atoms  $j$  on one oxygen atom  $i$  may be expressed as

$$\mathbf{F}_{LJ,i} = \frac{48\epsilon}{\sigma^2} \sum_{i \neq j}^N (\mathbf{r}_i - \mathbf{r}_j) \left[ \left( \frac{\sigma}{r_{ij}} \right)^{14} - \frac{1}{2} \left( \frac{\sigma}{r_{ij}} \right)^8 \right]. \quad (2.34)$$

### 2.5.2 Coulomb interaction

Coulomb's law describes the electrostatic interaction between two electrically charged particles. The potential between two charges  $q_i$  and  $q_j$  separated by a distance  $r_{ij}$  is defined as

$$V_{C,ij} = k_e \frac{q_i q_j}{r_{ij}^2}, \quad (2.35)$$

where  $k_e$  is the coulomb proportionality constant. The net coulombic potential energy contribution of a charge  $i$  in a system of  $N$  discrete charges is thus

$$V_{C,i} = k_e \sum_{i \neq j}^N \frac{q_i q_j}{r_{ij}^2}. \quad (2.36)$$

The force experienced by a charged particle  $i$  from an interaction with a charged particle  $j$  may therefore be expressed as

$$\mathbf{F}_{C,ij} = k_e \frac{q_i q_j}{|r_i - r_j|^3} (\mathbf{r}_i - \mathbf{r}_j). \quad (2.37)$$

Thus, the net electrostatic force exerted by all  $N$  charged particles  $j$  on an charged particle  $i$  may be defined as

$$\mathbf{F}_{C,i} = k_e \sum_{i \neq j}^N \frac{q_i q_j}{|r_i - r_j|^3} (\mathbf{r}_i - \mathbf{r}_j). \quad (2.38)$$

### 2.5.3 Translational acceleration

The force on the center of mass of a rigid water molecule  $i$  may be described as the sum of all forces on the atoms of that molecule. Equation 2.34 describes the force experienced by an oxygen atom in a water molecule. Equation 2.38 describes the forces experienced by both hydrogen atoms as well as the negatively-

charged dummy atom in a water molecule. As each hydrogen atom experiences a force from all other hydrogen atoms in the system as well as all negatively charged dummy atoms, the net forces on the dummy atom and hydrogen atoms in our simulation were described as

$$\mathbf{F}_{H_{1,i}} = \sum_{j \neq i}^N \mathbf{F}_{H_{1,j} \rightarrow H_{1,i}} + \mathbf{F}_{H_{2,j} \rightarrow H_{1,i}} + \mathbf{F}_{M_j \rightarrow H_{1,i}}, \quad (2.39)$$

$$\mathbf{F}_{H_{2,i}} = \sum_{j \neq i}^N \mathbf{F}_{H_{1,j} \rightarrow H_{2,i}} + \mathbf{F}_{H_{2,j} \rightarrow H_{2,i}} + \mathbf{F}_{M_j \rightarrow H_{2,i}}, \quad (2.40)$$

$$\mathbf{F}_{M_i} = \sum_{j \neq i}^N \mathbf{F}_{H_{1,j} \rightarrow M_i} + \mathbf{F}_{H_{2,j} \rightarrow M_i} + \mathbf{F}_{M_j \rightarrow M_i}. \quad (2.41)$$

The net force on the oxygen atom was related to equation 2.34, *i.e.*,

$$\mathbf{F}_{O_i} = \sum_{j \neq i}^N \mathbf{F}_{O_j \rightarrow O_i}. \quad (2.42)$$

Clearly, for each molecule, a total of ten net force computations were required. In each net force computation, there were a total of  $N^2$  calculations required. Thus, each molecule required a total of  $10N^2$  computations per timestep.

The force on the center of mass of molecule  $i$  was then defined as

$$\mathbf{F}_{CM,i} = \mathbf{F}_{H_{1,i}} + \mathbf{F}_{H_{2,i}} + \mathbf{F}_{M_i} + \mathbf{F}_{O_i}. \quad (2.43)$$

The sum of potential energies for each molecule were also calculated, with rules similar to the addition of forces as above, providing a scalar net potential energy value for each molecule,  $U_i$ .

The translational acceleration of a molecule's center of mass resulting from the force on the center of mass was defined as

$$\mathbf{a}_{CM,i} = \frac{\mathbf{F}_{CM,i}}{m_M}. \quad (2.44)$$

This quantity was later used in the integration of  $\mathbf{r}_{CM,i}$ .

## 2.5.4 Torque and rotational acceleration

As each atom in a rigid water molecule rotates about the molecule's center of mass, describing the molecule's rotation required us to first compute the torque on each atom. The definition of the torque vector is

$$\boldsymbol{\tau} = \mathbf{r} \times \mathbf{F}, \quad (2.45)$$

where  $\mathbf{r}$  is the displacement vector (directed from the point where the torque is measured to the point where the force is applied) and  $\mathbf{F}$  is the force vector. For each atom surrounding a molecule's center of mass, we were thus required to calculate its corresponding torque vector. The displacement vector from the center of mass to an atom on the molecule was defined as

$$\mathbf{r}_{disp} = \mathbf{r}_{atom} - \mathbf{r}_{CM}. \quad (2.46)$$

We used equation 2.46 and equations 2.39-2.42 to describe the relative displacement vector and force vector of each atom. We then used 2.45 to describe the torque on an atom. Finally, we summed the resultant torques of all atoms within each molecule and determined the overall torque on the molecule, *i.e.*,

$$\boldsymbol{\tau}_{net} = \boldsymbol{\tau}_{H_1} + \boldsymbol{\tau}_{H_2} + \boldsymbol{\tau}_O + \boldsymbol{\tau}_M. \quad (2.47)$$

This space-fixed torque vector was then described in the body-fixed frame by computing its scalar products with the vectors describing molecular orientation, *i.e.*,

$$\tau'_{net,x} = \boldsymbol{\tau}_{net} \cdot \mathbf{a}, \quad \tau'_{net,y} = \boldsymbol{\tau}_{net} \cdot \mathbf{b}, \quad \tau'_{net,z} = \boldsymbol{\tau}_{net} \cdot \mathbf{c}. \quad (2.48)$$

These body-fixed descriptions of torque were necessary to integrate the rotational motion of each molecule.

At an instant in time, using quaternions, the body-fixed angular velocity of a molecule was described as [24]:

$$\begin{pmatrix} \omega'_x \\ \omega'_y \\ \omega'_z \\ 0 \end{pmatrix} = 2 \begin{pmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \\ q_0 & q_1 & q_2 & q_3 \end{pmatrix} \begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix}. \quad (2.49)$$

The body-fixed angular acceleration components were then defined as [24]:

$$\dot{\omega}'_x = \frac{\tau'_{net,x} + (I_y - I_z)\omega'_y\omega'_z}{I_x}, \quad (2.50)$$

$$\dot{\omega}'_y = \frac{\tau'_{net,y} + (I_z - I_x)\omega'_z\omega'_x}{I_y}, \quad (2.51)$$

$$\dot{\omega}'_z = \frac{\tau'_{net,z} + (I_x - I_y)\omega'_x\omega'_y}{I_z}. \quad (2.52)$$

Finally, the second time derivative of  $\mathbf{q}$  was defined as [24]:

$$\begin{pmatrix} \ddot{q}_0 \\ \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -q_1 & -q_2 & -q_3 & q_0 \\ q_0 & -q_3 & q_2 & q_1 \\ q_3 & q_0 & -q_1 & q_2 \\ -q_2 & q_1 & q_0 & q_3 \end{pmatrix} \begin{pmatrix} \dot{\omega}'_x \\ \dot{\omega}'_y \\ \dot{\omega}'_z \\ -2 \sum \dot{q}_m^2 \end{pmatrix}. \quad (2.53)$$

These relations were used in the description of rotational motion as defined by the integration of the time-dependent quaternions; such integration is described in the next section.

## 2.6 Integration

### 2.6.1 Predictor-Corrector Technique

Predictor-corrector (PC) methods are numerical techniques that make use of several data elements computed at one or more earlier timesteps in the simulation. As we were only interested in derivative quantities included in classical Newtonian dynamics, we used a PC integrator which only incorporated second-order time derivatives. The goal of the numerical integrator, of course, is to solve the second-order differential equation  $\ddot{Q} = f(Q, \dot{Q}, t)$ . In our integration scheme,  $P(\cdot)$  and  $C(\cdot)$  denote the formulae used in the predictor and corrector steps of the integration. The predictor step at a time  $t + \Delta t$  was described as an extrapolation of earlier values [5]:

$$P(Q) : Q(t + \Delta t) = Q(t) + \dot{Q}\Delta t + \Delta t^2 \sum_{i=1}^{k-1} \alpha_i f(t + (1-i)\Delta t), \quad (2.54)$$

$$P(\dot{Q}) : \frac{Q(t + \Delta t) - Q(t)}{\Delta t} + \Delta t \sum_{i=1}^{k-1} \alpha'_i f(t + (1-i)\Delta t). \quad (2.55)$$

After computing these predicted values, force and torque calculations, outlined in the previous chapter,



were estimated. Using these new values, the “corrected” quantities were computed [5]:

$$C(Q) : Q(t + \Delta t) = Q(t) + \dot{Q}\Delta t + \Delta t^2 \sum_{i=1}^{k-1} \beta_i f(t + (2-i)\Delta t), \quad (2.56)$$

$$C(\dot{Q}) : \frac{Q(t + \Delta t) - Q(t)}{\Delta t} + \Delta t \sum_{i=1}^{k-1} \beta_i' f(t + (2-i)\Delta t). \quad (2.57)$$

Note the predicted values did not appear in corrector formulae except in evaluating  $f$  (the accelerations). The coefficients,  $\alpha_i$ ,  $\alpha_i'$ ,  $\beta_i$ , and  $\beta_i'$  gave weight to previously-calculated values of  $f$  in determining  $Q$  and  $\dot{Q}$ . For the case of  $k = 4$ , which provides decent accuracy for short MD simulations, these coefficients are defined as the following:

$i$	1	2	3
$\alpha_i$	$\frac{19}{24}$	$-\frac{10}{24}$	$\frac{3}{24}$
$\alpha_i'$	$\frac{27}{24}$	$-\frac{22}{24}$	$\frac{7}{24}$
$\beta_i$	$\frac{3}{24}$	$\frac{10}{24}$	$-\frac{1}{24}$
$\beta_i'$	$\frac{7}{24}$	$\frac{6}{24}$	$-\frac{1}{24}$

Table 2.5: Predictor-corrector integrator coefficients for  $k = 4$  [5].

Using the predictor and corrector formulae, at the beginning of a simulation timestep, we first predicted  $\mathbf{r}_{CM}$  and  $\mathbf{q}$ . We then computed predictions for  $\dot{\mathbf{r}}_{CM}$  and  $\dot{\mathbf{q}}$ . From these quantities, we estimated the molecules’ associated unit vectors and atomic positions. We then calculated  $\ddot{\mathbf{q}}$  and  $\ddot{\mathbf{r}}_{CM}$  in the subroutine used to calculate forces. Using this new information, we applied the corrector steps to  $\mathbf{r}_{CM}$  and  $\mathbf{q}$ , then  $\dot{\mathbf{r}}_{CM}$  and  $\dot{\mathbf{q}}$ . Finally, we corrected the molecules’ unit vectors and atom positions.

# Chapter 3

## Results

### 3.1 Energy Conservation

The most fundamental validation tool of any physical simulation is the conservation of energy. The energy of a system of  $N$  molecules may be defined as the sum of its net potential and kinetic energy constituents, *i.e.*,

$$E_{net} = U_{net} + K_{net}. \quad (3.1)$$

Of course, the potential energy of a molecule may be defined as the sum of the potential energies of its constituent atoms. These potential energies were calculated using equations 2.32 and 2.36. Thus, the total potential energy of a molecule was considered as the sum of potential energies resulting from both Lennard-Jones and coulombic interactions. We defined the net potential energy of a system to be

$$U_{net} = \frac{1}{2} \sum_i^N U_i, \quad (3.2)$$

where the factor of a half compensated for double-counting.

The kinetic energy of a molecule was then divided into its translational and rotational kinetic energy constituents. The translational kinetic energy of each rigid water molecule was defined as

$$K_{trans,i} = \frac{1}{2} m_M v_M^2, \quad (3.3)$$

where  $v_M^2 = \mathbf{v}_M \cdot \mathbf{v}_M$ . The rotational kinetic energy of each molecule was defined as

$$K_{rot,i} = \frac{1}{2} \mathbf{I} \omega^2 = \frac{1}{2} (I_x \omega_x'^2 + I_y \omega_y'^2 + I_z \omega_z'^2). \quad (3.4)$$

Of course, the total kinetic energy of a molecule was defined as

$$K_{net} = K_{trans,i} + K_{rot,i}. \quad (3.5)$$

In testing the energy conservation of any physical simulation, as the timestep of a simulation decreases, so too should the error accumulated in energy over time. Thus, if  $E_{initial}$  and  $E_{final}$  represent the initial and final energies of the system, the absolute value of the difference between them, *i.e.*,  $|\Delta E|$  should become smaller as  $\Delta t$  is decreased.

Figure 3.1 displays the average molecular energy in a system of sixty-four water molecules over a period of 6.25 ps, with  $\Delta t$  ranging from 0.00015625 ps to 0.000625 ps. Clearly, as the timestep was decreased, energy conservation improved. However, it is of note that the simulations from which these curves were obtained used double-precision variables. As the use of double-precision variables is computationally expensive in MD and incompatible with many CUDA-capable devices, it is advantageous to employ lesser-precise floating-point variables.

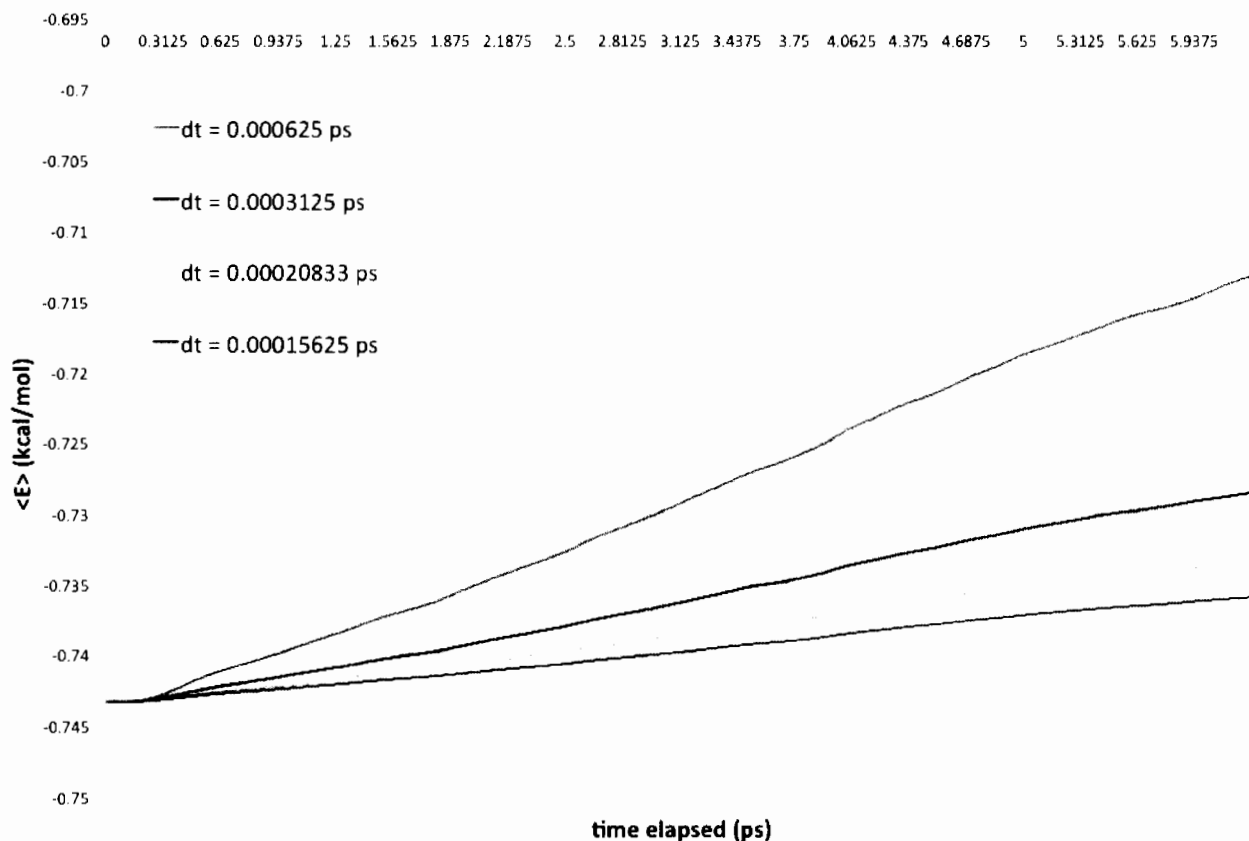


Figure 3.1: Energy conservation for double precision simulations with 64 molecules over 6.25 picoseconds.

Figure 3.2 displays the same situation as that described above, instead using floating-point variables.

Of course, the total kinetic energy of a molecule was defined as

$$K_{net} = K_{trans,i} + K_{rot,i}. \quad (3.5)$$

In testing the energy conservation of any physical simulation, as the timestep of a simulation decreases, so too should the error accumulated in energy over time. Thus, if  $E_{initial}$  and  $E_{final}$  represent the initial and final energies of the system, the absolute value of the difference between them, *i.e.*,  $|\Delta E|$  should become smaller as  $\Delta t$  is decreased.

Figure 3.1 displays the average molecular energy in a system of sixty-four water molecules over a period of 6.25 ps, with  $\Delta t$  ranging from 0.00015625 ps to 0.000625 ps. Clearly, as the timestep was decreased, energy conservation improved. However, it is of note that the simulations from which these curves were obtained used double-precision variables. As the use of double-precision variables is computationally expensive in MD and incompatible with many CUDA-capable devices, it is advantageous to employ lesser-precise floating-point variables.

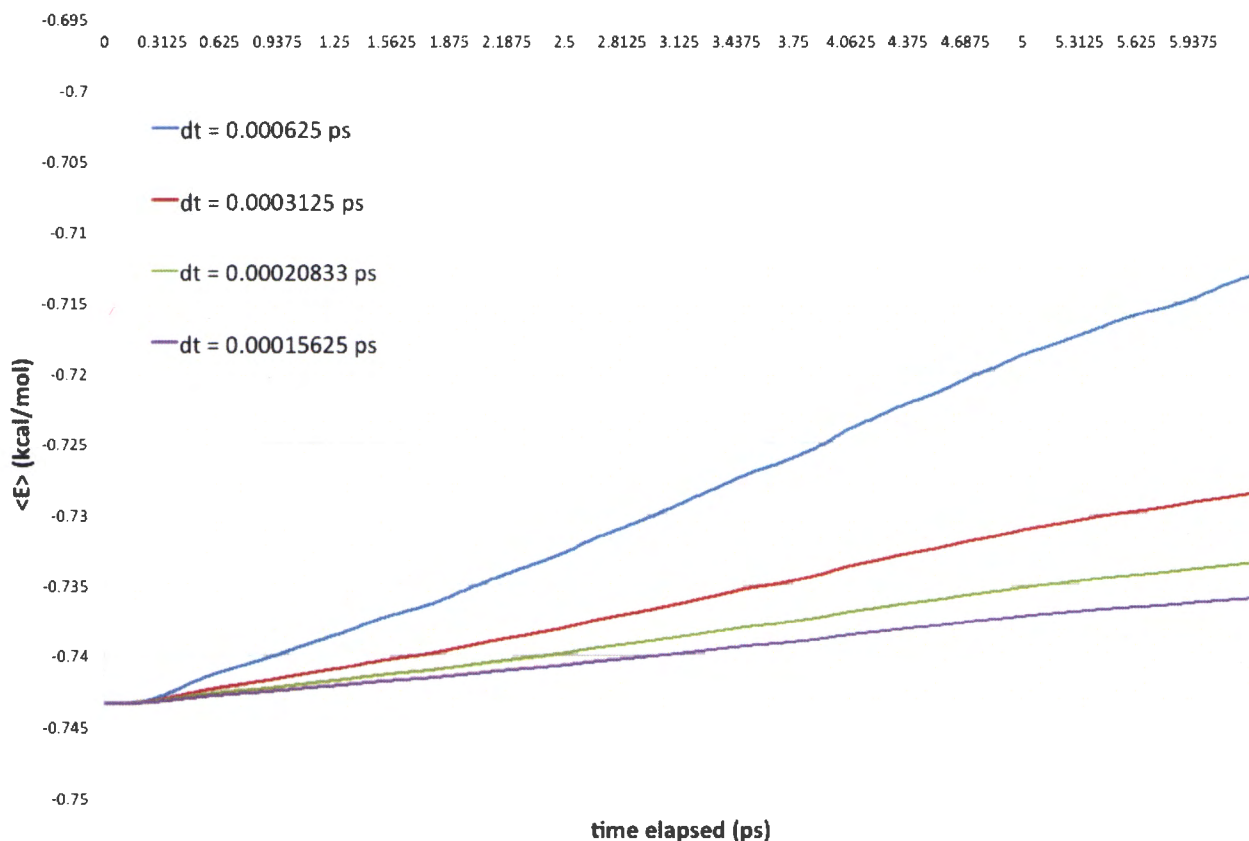


Figure 3.1: Energy conservation for double precision simulations with 64 molecules over 6.25 picoseconds.

Figure 3.2 displays the same situation as that described above, instead using floating-point variables.

Under these circumstances, as the timestep was decreased, overall energy conservation appeared to become *worse*. Especially notable was the case of  $\Delta t = 0.00015625$  ps, in which errors in energy increased relatively quickly. This peculiarity could have been a result of the increased accumulation of rounding errors as a consequence of a very small timestep. For example, a relatively “large” number multiplied by a relatively “small” number, such as a smaller timestep, could have resulted in *more significant* truncations than in the case of a larger timestep. Such a computation could have occurred in the prediction and/or correction stages of the simulation. Nonetheless, the energy curve corresponding to  $\Delta t = 0.000625$  ps appeared to be relatively stable. As most MD simulations use a  $\Delta t$  near this value, we deemed it appropriate to continue running simulations using this timestep.

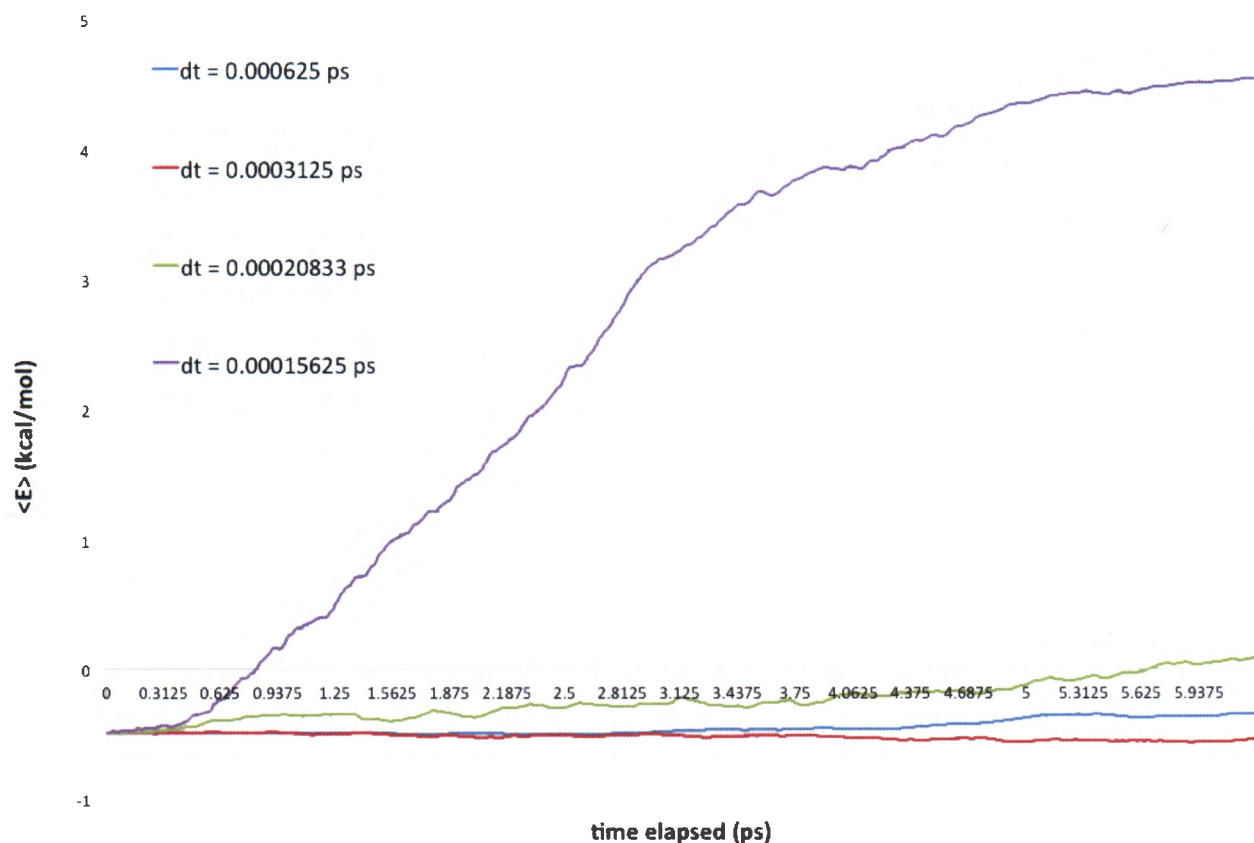


Figure 3.2: Energy conservation for floating point simulations with 64 molecules over 6.25 picoseconds.

## 3.2 Temperature equilibrium

As temperature may be extrapolated from the ensemble averages of the translational and rotational kinetic energies in an MD simulation, over a sufficient period of time [26],

$$\langle K_{trans} \rangle = \frac{1}{N} \sum_{i \neq j}^N K_{trans,i} = \langle K_{rot} \rangle = \frac{1}{N} \sum_{i \neq j}^N K_{rot,i} = \frac{3}{2} N k_B T, \quad (3.6)$$

where  $T$  is the equilibrium temperature of the system and  $N$  is the number of molecules.

Temperature variations can thus be implemented into an MD simulation by modifying translational and/or rotational motion of molecules at the end of a timestep, provided the system has been allowed to “age” before manipulation. As an example, after allowing an MD simulation to run for some time, one could begin to rescale the translational or rotational velocities of molecules until their average kinetic energies became proportional to the desired temperature. This “rescaling” would also result in the removal or addition of energy over time, usually coming from one of the two kinetic terms. After rescaling for a sufficient amount of time, equation 3.6 would be satisfied, and the system would exist in an equilibrium state with a relatively constant temperature [5].

The simplest mechanism used to rescale motion is by multiplying the translational velocities of particles by some rescaling factor  $\lambda$ . This rescaling factor is most easily defined as

$$\lambda = \sqrt{\frac{T_0}{T}}, \quad (3.7)$$

where  $T$  represents the average system temperature and  $T_0$  represents the desired system temperature. Each component of  $\mathbf{v}_{CM}$  could then be multiplied by the rescale factor after the corrector step, *i.e.*,

$$\dot{\mathbf{r}}_{CM} = \lambda \dot{\mathbf{r}}_{CM}. \quad (3.8)$$

As the system temperature,  $T$ , approaches the desired system temperature  $T_0$ ,  $\lambda$  approaches unity, and motion ceases to be significantly affected by the rescaling factor.

Another popular velocity scaling method in MD is the Berendsen thermostat [27]. Using this method, the rescaling factor is instead defined as

$$\lambda = \left[ 1 + \frac{\Delta t}{\tau_T} \left( \frac{T}{T_0} - 1 \right) \right]^{1/2}, \quad (3.9)$$

where  $\tau_T$  describes the strength of coupling of the system to a hypothetical thermal reservoir. As  $\tau_T$  increases,

this coupling becomes weaker. Thus, as  $\tau_T$  increases, it takes longer to reach the desired temperature  $T_0$ . In our simulation, we used a value of  $\tau_T$  equal to 0.01, indicating strong coupling between the system and reservoir.

Figure 3.3 represents the translational and rotational “temperatures” of a system containing 512 water molecules over a period of 62.5 ps (or 100,000 timesteps). The system was first allowed to age 6.25 ps. The Berendsen thermostat was then applied to continually rescale translational molecular velocities such that the average translational kinetic energy became proportional to 300.15 K. Clearly, the translational motion of atoms was rescaled successfully and quickly. However, over time, a loss in rotational kinetic energy was anticipated. Instead, this rotational kinetic energy fluctuated about a relatively large value corresponding to a higher temperature than desired. Thus equation 3.6 was not satisfied over a sufficient time scale. This implies there is an error in our current description of rotational motion which allows rotations to remain undamped even with significant changes to the system.

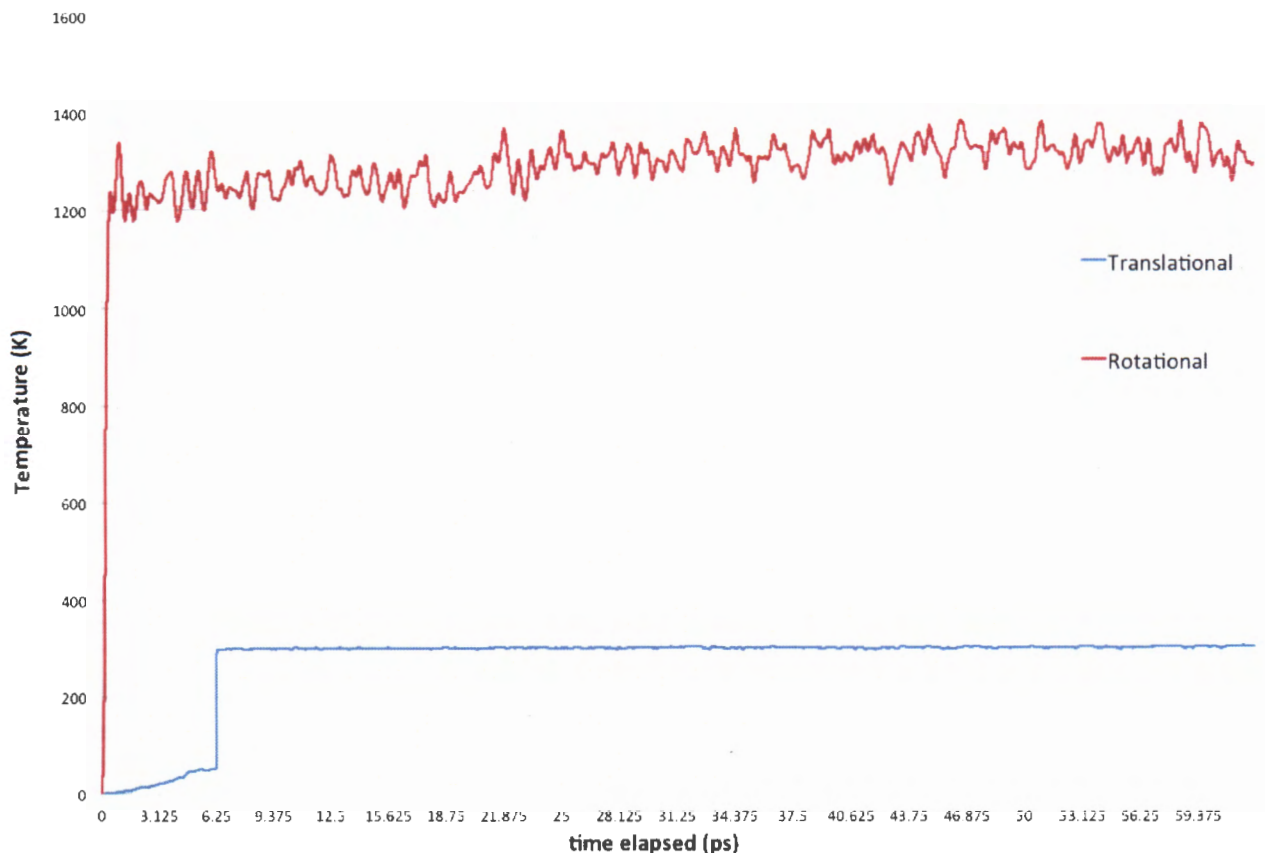


Figure 3.3: Comparison of translational and rotational temperatures obtained using a 300.15 K Berendsen thermostat in a system of 512 water molecules.

Figure 3.4 demonstrates the use of the Berendsen thermostat in a system of 512 molecules over 62.5 ps for

a variety of desired system temperatures. The molecules' oxygen atoms were colored according to their kinetic energy; blue and red indicate small and large kinetic energies (relative to other molecules), respectively. The solid phase of water was first simulated at 100.15 K. As the simulated system was small, the sublimation of a small number of surface molecules was expected and observed. However, rotational motion was not damped over time by the rescaling of translational motion. After a sufficient period of time, the formation of an ordered crystal of molecules was expected but not achieved. Liquid water and water vapor were then simulated at 330.15 and 1000.15 K. As expected, the gas phase became much more disordered over time compared to the liquid phase. However, rotational kinetic energies again did not equilibrate to the desired system temperatures. Although relatively correct qualitative behavior is observed for all three phases, errors in the description of rotational motion prevented the systems from being quantitatively accurate.

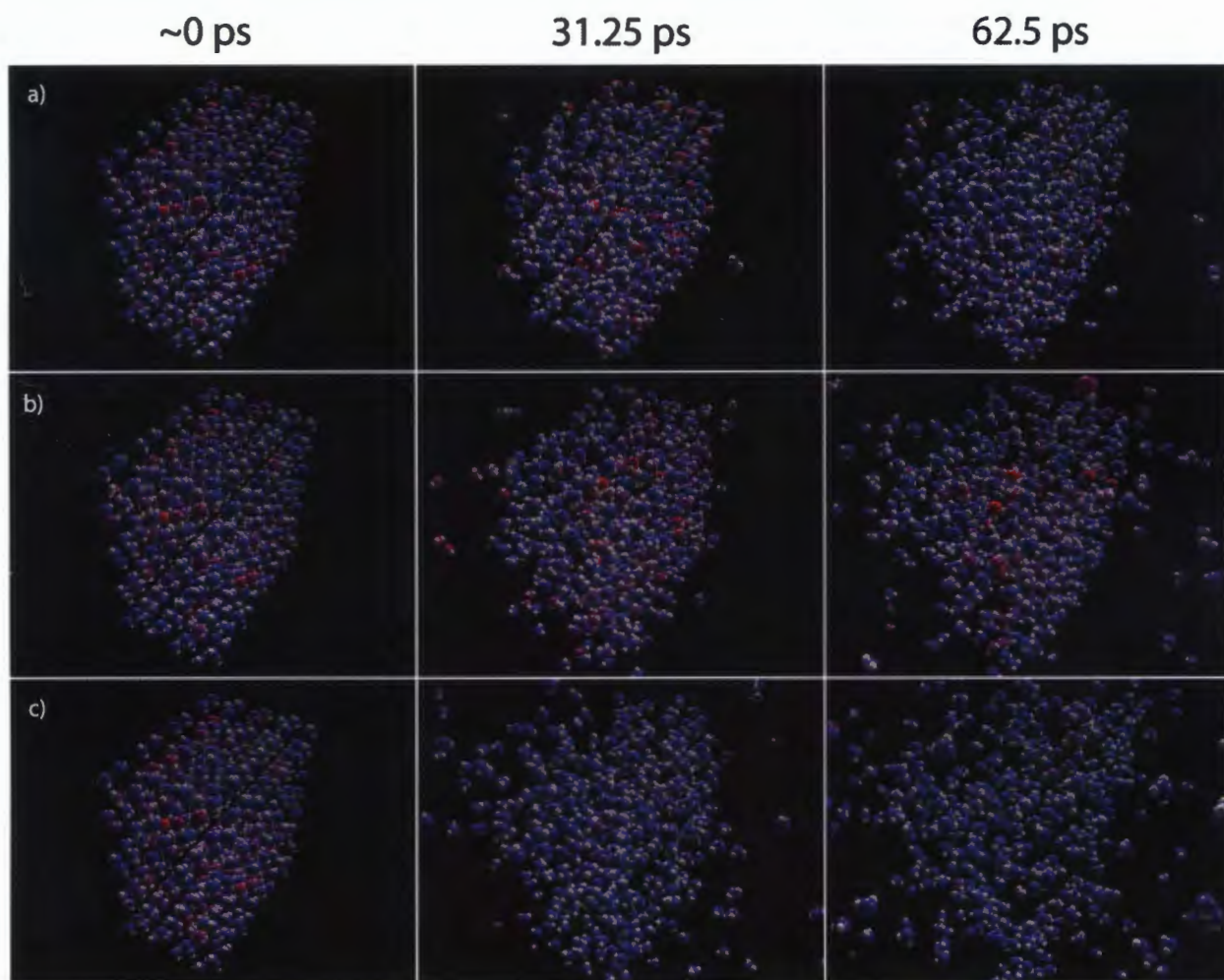


Figure 3.4: Phases of water generated using the Berendsen thermostat in a system of 512 molecules over 62.5 ps: a) “ice” at 100.15 K; b) “liquid water” at 330.15 K; c) “water vapor” at 1000.15 K.



### 3.3 GPU benchmarking

Aside from physical validation, it was also important to validate the benefit of using GPU programming for our computations. Figure 3.5 shows the comparison between CPU and parallel GPU versions of the simulation. The CPU version was run using an AMD Athlon™ 64 X2 Dual Core Processor 5800+; the GPU version was run using an NVIDIA® GeForce 8800 GTS. The average computation time required per timestep is displayed on the vertical axis of the figure; each of these values were computed over a total of 10,000 timesteps. These averages are displayed as a function of the number of water molecules present in a system.

In the CPU version, as the number of water molecules increased, the time required per iteration became long compared to the GPU version. From this figure, the importance of utilizing the GPU or other parallel processors for MD simulations with large numbers of molecules is clear. Figure 3.6 displays the benchmarking results of the GPU version alone. The jaggedness of this curve exists as a result of the partitioning of GPU blocks into different sizes for different numbers of molecules. Nonetheless, a roughly exponential shape is maintained, indicating that as the number of molecules increased, the time required for a simulation also significantly increased. In the future, it will be advantageous to employ more sophisticated interaction algorithms to reduce this  $\mathcal{O}(N^2)$  complexity.

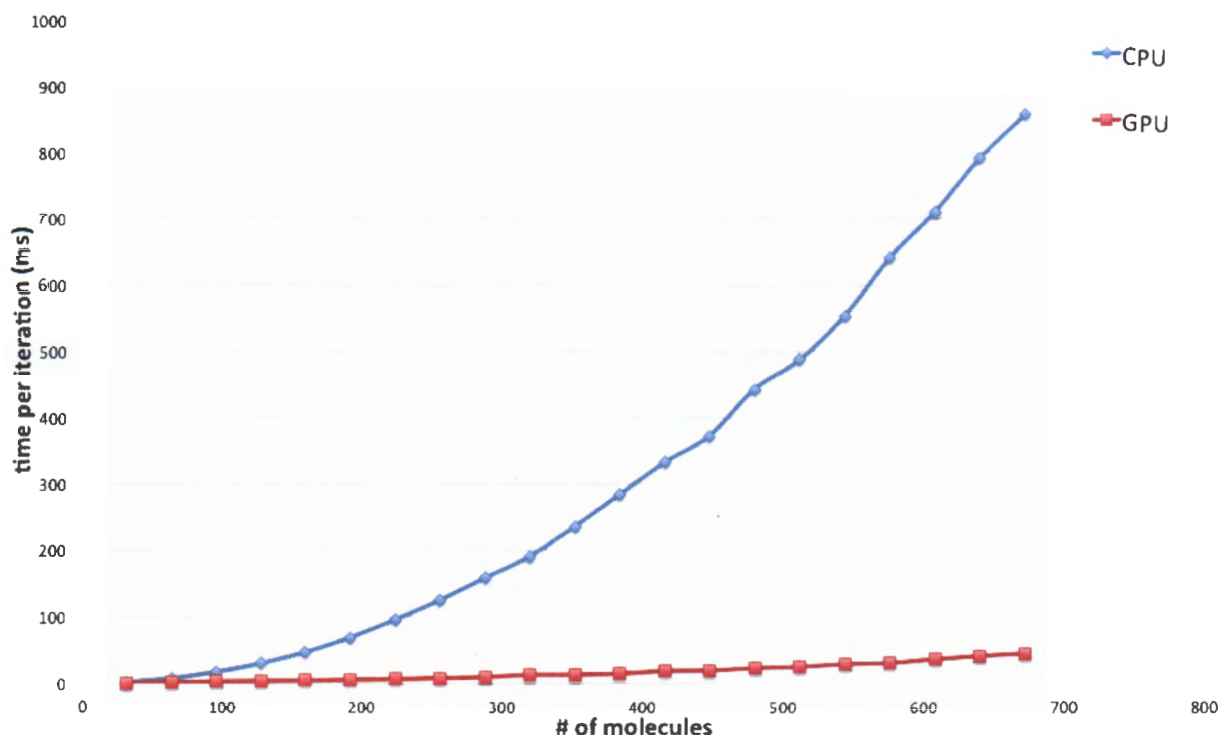


Figure 3.5: Performance for parallel GPU and serial CPU versions as a function of the number of molecules in a system.

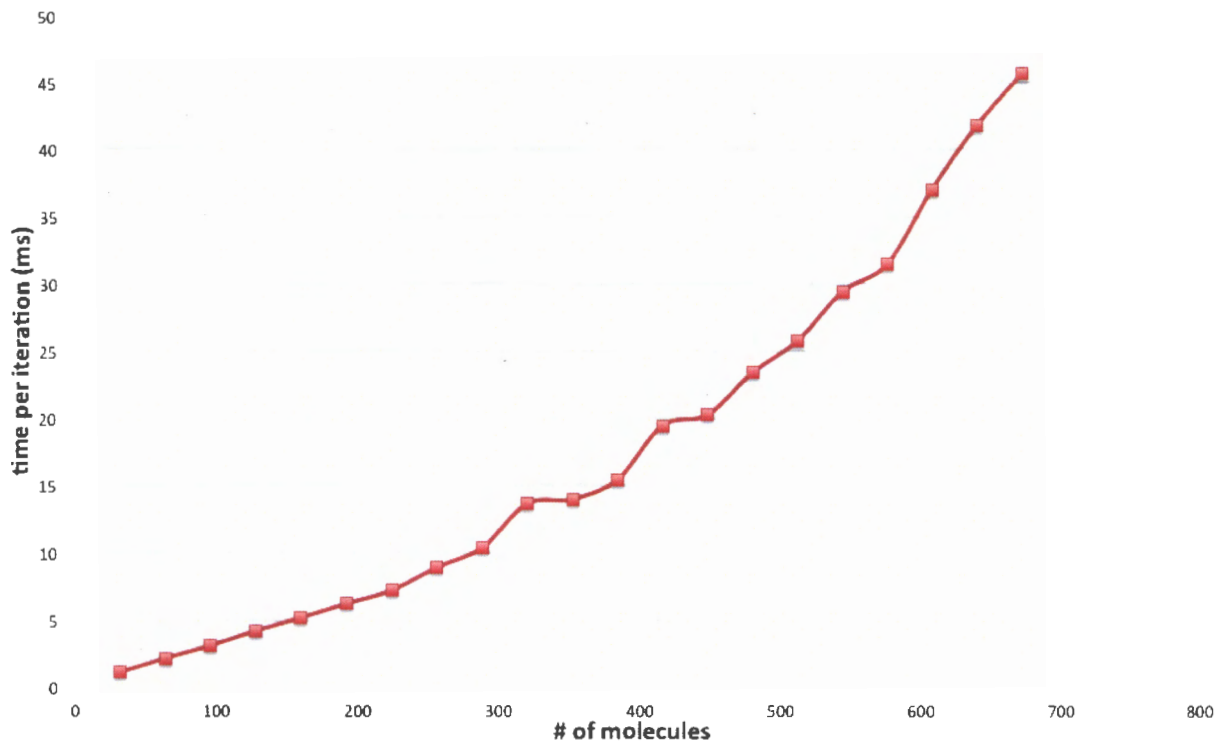


Figure 3.6: GPU performance as a function of the number of molecules in a system.

# Chapter 4

## Summary and future work

### 4.1 Summary

We have begun our introduction to molecular dynamics by first developing a classical MD simulation of rigid water molecules for the GPU. We employed the use of a simple, rigid water model (TIP4P/2005) in our simulations and extensively outlined the classical and statistical mechanical theory involved. We discussed the numerical techniques with which our simulation had been built and described the underlying GPU architecture which powers it. Finally, we partially validated our model by comparing the behavior of our simulations with anticipated behavior.

It is important to note the deficiencies present in the current version of our simulation. As displayed in figure 3.2, in the floating-point version of our simulation, as the timestep decreases, there exists ill-behaved energy conservation over time; this is contradictory to the results of the double-precision version as well as intuition. This may be a result of reduced units not near enough to unity, misdefined interaction rules, errors in our integration scheme, or errors in the handling of rotational motion. It may also be a result of other unforeseen problems.

It is also important to note the deficiencies exhibited when attempting to control the temperature of a system. Over a sufficient period of time, the kinetic energies associated with translational and rotational motions should converge. In the current version, although the translational energy may be scaled to correspond to a particular temperature, such rescaling does not result in the removal or addition of energy from the rotational kinetic component. To validate our simulation, it will first be crucial to observe the equilibration of translational and rotational kinetic energies onto user-defined temperatures. It will also be important to halt velocity rescaling at such an equilibrium and ensure the system stays in a stable state thereafter.

Nonetheless, with the grand scale of the project as well as the accomplishments seen over the past year, these issues appear relatively small and reparable. As the current program proves to be quite robust, these fixes will not noticeably change the current structure of the code. In summary, although there are currently small problems preventing further validation of the simulation, the solving of these problems as well as other

small additions will result in an MD simulation of rigid water comparable in completeness to any other.

## 4.2 Future Work

After solving the problems discussed in the previous section, it will be advantageous to employ boundary conditions in our simulation. Periodic boundaries, in particular, will allow systems to remain spatially homogenous. These strict boundaries will prohibit molecules from leaving the sample space we wish to study, allowing us to control the volume of a periodic box and thus the pressure of systems. Such pressure control is often packaged with more sophisticated temperature control mechanisms [5]. These new methods will most likely entail modifications to the periodic box size and molecular velocities only once every few timesteps rather than each timestep.

Through the addition of periodic boundaries as well as more sophisticated pressure and temperature controls, we will have the capabilities required to reconstruct several quantities and plots obtained in other TIP4P/2005 experiments, including water's phase diagram, compressibility, heat capacity, and other statistical properties. After validating our simulations agree with previous TIP4P/2005 experiments, we will be in a position to add simple ions and other molecules to systems of water. We hope to further validate our model by comparing the behaviors of these simple chemical systems with those described in other MD literature.

Once we are satisfied with the comparative performance of our simulations, we will then begin to test the effects of long-range interactions from polar water molecules on recent and historical MD simulations. By such a point, we will undoubtedly need to utilize increased processing power and/or other approximation schemes. We first hope to reconstruct our code such that GPU clusters (*i.e.*, systems of multiple GPUs) may be used in our simulations. We also hope to employ the use of the Barnes-Hut algorithm [28] and/or other long-range interaction approximation schemes to decrease computational load while maintaining sufficient accuracy.

It is our hope the results of future simulations will better guide the MD community toward the accurate characterization of long-range interactions of water in biomolecular systems. Moving toward increased accuracy while maintaining efficiency will have lasting effects on the many scientific domains currently employing MD techniques.

# References

- [1] A.C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh. Parallelism via multithreaded and multicore CPUs. *Computer*, 43(3):24–32, March 2010.
- [2] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. GPU cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, SC '04, pages 47–, Washington, DC, USA, 2004. IEEE Computer Society.
- [3] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007.
- [4] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.
- [5] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, New York, NY, USA, 1996.
- [6] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, March 1995.
- [7] Erik Lindahl, Berk Hess, and David van der Spoel. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling*, 7:306–317, 2001. 10.1007/s008940100045.
- [8] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kal, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.
- [9] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227:5342–5359, May 2008.
- [10] Mark S. Friedrichs, Peter Eastman, Vishal Vaidyanathan, Mike Houston, Scott Legrand, Adam L. Beberg, Daniel L. Ensign, Christopher M. Bruns, and Vijay S. Pande. Accelerating molecular dynamic simulation on graphics processing units. *Journal of Computational Chemistry*, 30(6):864–872, 2009.
- [11] Jan Norberg and Lennart Nilsson. On the truncation of long-range electrostatic interactions in DNA. *Biophysical Journal*, 79(3):1537–1553, 2000.
- [12] Michael Patra, Mikko Karttunen, Marja T. Hyvnen, Emma Falck, and Ilpo Vattulainen. Lipid bilayers driven to a wrong lane in molecular dynamics simulations by subtle changes in long-range electrostatic interactions. *Journal of Physical Chemistry B*, 108(14):4485–4494, 2004.
- [13] H. Schreiber and O. Steinhauser. Cutoff size does strongly influence molecular dynamics results on solvated polypeptides. *Biochemistry*, 31(25):5856–5860, 1992.
- [14] Celeste Sagui and Thomas A. Darden. Molecular dynamics simulations of biomolecules: Long-range electrostatic effects. *Annual Review of Biophysics and Biomolecular Structure*, 28(1):155–179, 1999.

- [15] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh Ewald: An  $N \log(N)$  method for Ewald sums in large systems. *Journal of Chemical Physics*, 98(12):10089–10092, 1993.
- [16] Ulrich Essmann, Lalith Perera, Max L. Berkowitz, Tom Darden, Hsing Lee, and Lee G. Pedersen. A smooth particle mesh Ewald method. *Journal of Chemical Physics*, 103(19):8577–8593, November 1995.
- [17] L Greengard and V Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325 – 348, 1987.
- [18] H. W. Horn, W. C. Swope, J. W. Pitera, J. D. Madura, T. J. Dick, G. L. Hura, and T. Head-Gordon. Development of an improved four-site water model for biomolecular simulations: TIP4P-Ew. *Journal of Chemical Physics*, 120:9665–9678, May 2004.
- [19] Michael W Mahoney and William L Jorgensen. A five-site model for liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions. *Journal of Chemical Physics*, 112(20):8910, 2000.
- [20] J. L. F. Abascal and C. Vega. A general purpose model for the condensed phases of water: TIP4P/2005. *Journal of Chemical Physics*, 123(23):234505, 2005.
- [21] J. A. Hayward and J. R. Reimers. Unit cells for the simulation of hexagonal ice. *Journal of Chemical Physics*, 106:1518–1529, January 1997.
- [22] W. Tang and University of Delaware. Department of Mechanical Engineering. *Molecular Dynamics Simulations of Carbon Nanotubes in Liquid Flow*. University of Delaware, 2007.
- [23] Gregory G. Slabaugh. Computing euler angles from a rotation matrix. Technical report, August 1999.
- [24] D.C Rapaport. Molecular dynamics simulation using quaternions. *Journal of Computational Physics*, 60(2):306–314, 1985.
- [25] J. E. Jones. On the determination of molecular fields: (II) From the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):pp. 463–477, 1924.
- [26] A. Rahman and F. H. Stillinger. Molecular dynamics study of liquid water. *Journal of Chemical Physics*, 55:3336–3359, October 1971.
- [27] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak. Molecular dynamics with coupling to an external bath. *Journal of Chemical Physics*, 81(8):3684–3690, October 1984.
- [28] J. Barnes and P. Hut. A hierarchical  $O(N \log N)$  force-calculation algorithm. *Nature*, 324:446–449, December 1986.