2009

A PARALLEL MOLECULAR DYNAMICS PROGRAM FOR SIMULATION OF WATER IN ION CHANNELS

Laxmi Mullapudi
Virginia Commonwealth University

School of Engineering
Virginia Commonwealth University

This is to certify that the thesis prepared by Laxmi Mullapudi entitled A PARALLEL MOLECULAR DYNAMICS PROGRAM FOR SIMULATION OF WATER IN ION CHANNELS has been approved by her committee as satisfactory completion of the thesis requirement for the degree of
Master of Science, Chemical and Life Science Engineering

Michael H. Peters, Ph.D, School of Engineering

Stephen S. Fong, Ph.D, School of Engineering

Vamsi K. Yadavalli, Ph.D, School of Engineering

Laura A. McLay, Ph.D, School of Humanities and Sciences

Frank Gupton, Ph.D, Chair of Chemical and Life Science Engineering, School of Engineering

Rosalyn Hobson, Ph.D, Associate Dean for Graduate Studies, School of Engineering

Russell D. Jamison, Ph.D, Dean, School of Engineering

F. Douglas Boudinot, Ph.D, Dean of the Graduate School

05/08/2009

A PARALLEL MOLECULAR DYNAMICS PROGRAM FOR SIMULATION OF WATER ATOMS IN ION CHANNELS

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science, Chemical and Life Science Engineering
at Virginia Commonwealth University.

by

LAXMI ANASUYA MULLAPUDI
B Tech Chemical Engineering, University College of Technology,
Osmania University India, 2006

Thesis Advisor: DR. MICHAEL H. PETERS
PROFESSOR, CHEMICAL AND LIFE SCIENCE ENGINEERING

Virginia Commonwealth University
Richmond, Virginia
May 2009

Acknowledgement

I thank my advisor, Dr Michael H. Peters, for, giving me this opportunity to work on a topic in the area of my interest and inspiring me with his insightful suggestions for carrying out the research successfully. I sincerely appreciate the technical help provided by Mr. John G. Laynne in debugging the parallel code.

I thank the staff and the faculty of the School of Engineering for cooperating with me during my research. I thank the Virginia Commonwealth University for providing me the facilities and financial assistance for my research work.

I thank my parents for raising me in an atmosphere that was conducive to enjoy the sheer experience of learning. I thank my loving husband for consistently supporting me in pursuing my interest.

I finally thank the people and the government of the United States of America for their gracious encouragement, accorded to a student from overseas, to pursue the dream of higher education.

Table of Contents

<u>List of Tables</u>

Page

List of Figures

Abstract

A PARALLEL MOLECULAR DYNAMICS PROGRAM FOR SIMULATION OF

WATER ATOMS IN ION CHANNELS.

By Laxmi Mullapudi, B Tech, Chemical Engineering

A Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science, Chemical and Life Science Engineering
at Virginia Commonwealth University.

Virginia Commonwealth University, 2009

Thesis Advisor: Dr. Michael H. Peters
Professor, Chemical and Life Science Engineering

With a modest beginning from developing a model of dynamics of hard liquid spheres (Alder *et al.*, 1957), molecular dynamics (MD) simulations have come to a point where complex biomolecules can be simulated with precision close to reality (Noskov *et al.*, 2007). In this context, a parallel molecular dynamics program for simulation of ion channels associated with cellular membranes has been developed. The parallel MD code developed is simple, efficient, and easily coupled to other codes such as the hybrid molecular dynamics/ brownian dynamics (MD/BD) code developed for the study of protein interactions (Ying *et al.*, 2005).

The Atom Decomposition (AD) Method was used in partitioning calculations on atoms to processors. One of the major impediments in using AD was the relatively large size of data that had to be communicated by the processes (Plimpton et al., 1995). Replicating only positions of atoms eased the congestion created by communication of both force terms and positions of atoms between processes. The performance of the code was tested on KcsA, a bacterial potassium channel. The program was written in Fortran 90 with parallel functions from the library of mpich-1.2.7. The idle time of processes was optimized by message driven ordering of communication.

The scaling of the parallel program with 2000 – 60,000 atoms was determined and compared with the results obtained from the serial program. As expected, the parallel program scaled better than the serial program as the number of atoms included in the simulation increased from 2000 - 60000. The performance of the parallel program was tested on 4-15 processes, for a system comprising 20,000 atoms. The results obtained were compared with results from the serial program. It was observed that the parallel program scaled better than the serial program as the number of processes increased from 4 to 15. When compared with serial program, which had application of Newton's Third Law in calculating force terms once per each pair of atoms, it was observed that the parallel program scaled better on 6-15 processes for a physical system comprising of 20,000 atoms.

CHAPTER 1 Introduction and Background Information

1.1 Theory of Molecular Dynamics

The dynamics of biological (bio) molecules range from local atomic fluctuations of the order, 0.01 angstroms/femto (10^{-15}) second, to molecular-scale motions of the order, 5 angstroms/micro (10^{-3}) second (Karplus et al., 2002). Although macroscopic properties can be calculated using experimental methods, computer simulations provide details of conformational, structural and thermodynamic changes of biomolecules and simultaneously track the atomic fluctuations. On the basis of the structure determined by X-ray crystallography and Nuclear Magnetic Resonance (NMR) studies, and with the advent of parallel programming on large-scale computing systems with hundreds of CPU's, Molecular dynamics (MD) method is being used extensively to discover time-dependent behavior of biomolecules.

MD method can be used to calculate the properties of a defined molecular system from equations of motion describing the displacements of individual atoms. The equations of motion are approximated by finite difference schemes and are solved numerically yielding changes in positions and velocities of atoms at each time step. A typical MD simulation is considered in three steps; (1) initialization (2) equilibration, and (3) production. (Karplus et al., 2002)

1) Initialization. The first step is specification of initial conditions at time zero. This is done by, assuming the initial atomic positions on a lattice and the velocities drawn from a Maxwellian distribution based on the system-temperature.

2) Equilibration. The next step is to allow the system to move to its equilibrium state.

3) Production. The third step is the actual calculation of the properties of the system along its trajectory in space field.

In canonical ensemble, the simulation can be conducted under constant temperature (T), constant number of particles (N) and constant volume (V) case. For a classical, Newtonian system, equation of motion can be written as,

$$\boxed{m_i \frac{d^2 r_i(t)}{dt^2} = -\frac{dv_i}{dr_i} = \sum_{i<j} F_i(r_{ij})} \text{-----------(1a)}$$

Where, 'm_i', 'r_i', 'v_i', are the mass, position and velocity of particle 'i' at time 't', 'F_i', the total force exerted by the system on the particle 'i', and 'r_{ij}', the distance between particles, 'i' and 'j'. Applying Finite Difference Scheme to equation (1a) gives the set of equations (1b).

$$\boxed{\begin{aligned} v_{n+\frac{1}{2}} &= v_n + \frac{\Delta t}{2m} F_n \\ r_{n+1} &= r_n + \Delta t v_{n+\frac{1}{2}} \\ F_{n+1} &= f(r_{n+1}) \\ v_{n+1} &= v_{n+\frac{1}{2}} + \frac{\Delta t}{2m} F_{n+1} \end{aligned}} \text{--------------(1b)}$$

Where, for a particle 'i', 'Δt', is the time-step; 'v_n', the initial velocity at time t=n; 'r_n', position at t=n and 'F_n', total force at time t=n. At time step t=n+1, velocity, position and force, on the particle are calculated as shown in (1b). This is known as Verlet Leap Frog algorithm of integration (Ying et al., 2003).

By the new set of positions obtained, the force components of atoms can be calculated from a potential energy function, which describes the physical system accurately. Amber (Ponder et al., 2003) and CHARMM (Brooks et al., 2003) are the most commonly used potential energy functions in simulation studies of biomolecules.

The fundamental drawback of MD is that it requires a relatively large computational effort in calculating force terms between atoms, raising the computational processing time required for simulating biomolecules to weeks and some times months. Parallel calculation of forces drastically reduces the simulation time. (Plimpton et al., 1995).

1.2 MPI- Message Passing Interface and Portable Parallel Programming

The speed of light and the effectiveness of heat dissipation impose physical limits on the speed of a single computer. Moreover, as performance of personal computer (PC) has increased and the prices have fallen significantly, it has become easier to acquire a cluster of PC' that are networked together, to develop suitable parallel codes to run on the cluster (Gropp et al., 1994).

The message passing model of parallel programming has a set of processes with local memory. The processes communicate with each other by sending and receiving messages. (Gropp et al., 1994)

MPI is a specific realization of message passing model. It is a library that specifies the names, calling sequences and results of subroutines/functions/classes. The programs developed using MPI library functions are compiled with ordinary compilers and linked with MPI library. Communication occurs when a portion of a process' address space is copied into another process' address space. Processes are identified by their ranks, which are integers from 0 to p-1 where, p is the total number of processes. Process is a software term, which refers to the address space, whereas the term processor is a hardware term representing a Central Processing Unit (Gropp et al., 1994).

1.3 Physical System for MD - Potassium Channel

The physical system focused for developing the parallel MD code was potassium channel pertaining to cell membranes. Potassium channels are made of protein atoms in an aqueous environment. The protein atoms on the internal surface of the channel are flexible and the water atoms inside the channel are diffusive in nature (Karplus et al., 2002).

Potassium channels let inorganic ions like Na+ and K+ pass in and out of the cell membranes. The passage of ions is crucial in intercellular communication and signal transduction. Potassium channel is gated and is selective to the passage of potassium ion. The recently discovered structure (PDB ID: 1BL8) of KcsA, a bacterial Voltage gated Potassium Channel from Streptomyces Lividians, shown in Figure 1, is frequently used for studying Potassium Channels. (Mackinnon et al,. 1998). X-ray diffraction studies of crystallized protein structure of KcsA showed that it has a wall made up of four identical protein monomers, which look like folded helices. The wall has an outer helix, inner helix and a central-filter region. The channel is filled with water molecules. (MacKinnon et al,. 1998)

MD is being used extensively in the field of potassium channels for determining the dynamics of the water atoms inside the channel, dynamics of the potassium ion, dynamics of the filter and free energy calculations on potassium ions. As there are around 6000 protein atoms and 60,000 water atoms in KcsA, parallel MD codes recently developed, like NAMD (Kale et al., 1996), are being used by the research community for simulating potassium channels. (Furini et al., 2009)

As a primary step of optimizing MD, the goal of the research was to decrease the cpu-time of simulation of water atoms in KcsA by parallel programming.

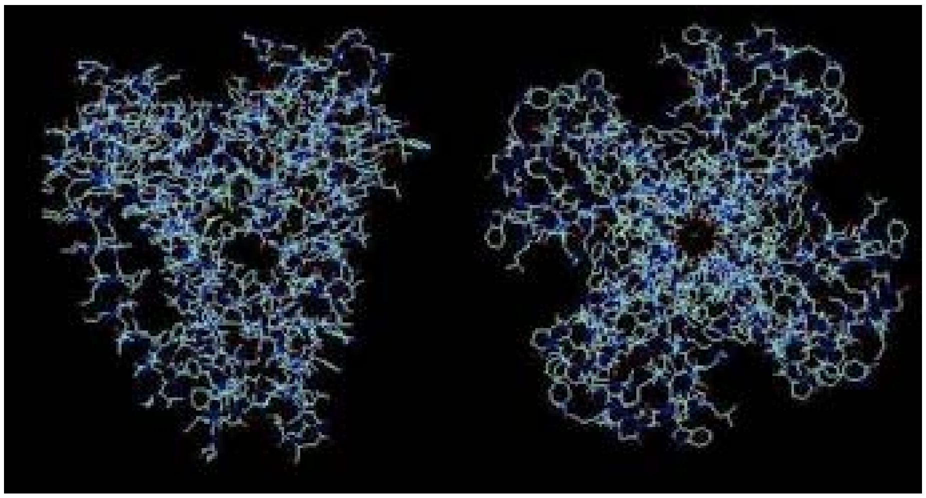


Figure 1 Structure of potassium channel 1BL8
Side and top view
(MacKinnon *et al*,. 1998), (Humphrey *et al*., 1995)

CHAPTER 2 Literature Review

The number of atoms in the physical system and number of time steps needed for simulation make MD computationally intensive. The fluctuations in positions of atoms are of the order of angstroms. Thousands of atoms must be included into calculations to simulate even a sub micron scale of physical system. The size of the time step is determined by the frequency of vibration of atoms, which is of the order of a femto second (Karplus et al., 2002). Thousands of time steps are necessary to simulate even picoseconds of real time. Because of these computational demands, optimizing MD calculations for clusters of processing units, has gained significance and the attention of researchers (Plimpton et al., 1995).

The force terms involved in MD calculations may either be long range or short range forces. For long range forces, such as columbic interactions, all charged atoms interact. For a system comprising N atoms, directly computing these forces scales with N2 and is computationally prohibitive for a large N. Algorithms like (1) Particle Mesh Algorithm (PME) which scales with f(M)N, where M is the number of mesh points and f(M) is a function of M, specific to the physical system (2) Hierarchical Method, which scales with Nlog(N), and (3) Fast Multi-Pole Method, which scales with N, have been developed to overcome the difficulty in calculating long range forces
(Plimpton et al., 1995).

Short -range interactions are less prohibitive because the ranges of influence of inter- atomic forces can be truncated using a cut-off distance, outside of which all interactions are ignored. In such cases, Neighbor List Algorithms are used to maintain a list of atoms, within the cut off radius from each atom (Plimpton et al., 1995). Linked Cell Method divides the physical system into 3D cells with length greater than cut-off radius. The force terms of atoms belonging to a particular cell are limited to that cell and its immediate neighbors only (Plimpton et al., 1995). Additional time-savings can be made by applying Newton's Third Law by computing force terms only once per each pair of interacting atoms.

Also, given the fact that MD computations are inherently parallel because of their explicit nature, there has been considerable effort by the researchers in the last few years to exploit this parallelism on various machines to improve the performance of MD programs. A predetermined set of atoms (Atom Decomposition), or a predetermined set of force calculations (Force Decomposition), or a single portion of the physical domain (Spatial Decomposition), is assigned to each processor. Most MD softwares being developed use one or a combination of these three methods of decomposing calculations to processors (Plimpton et al., 1995), (Murty et al., 1998), (Brown et al., 1997).

Earliest versions of CHARMM, GROMOS, Amber, EGO and Blue Matter (developed by IBM), use Atom Decomposition (AD). Other codes, such as

DL_POLY_3, NAMD2, GROMACS, MOLDY, latest version of CHARMM, NW Chem, PMD, SIGMA, ddgmq, and Euler Gromos, use Spatial Decomposition (SD) (Izaguirrre et al., 2004), (Zhestkov et al., 2008), (Refson et al., 1999).

Table 1 illustrates some of the earliest results using AD, Force Decomposition (FD), and SD on an Intel cluster with 32 and 64 processors.

Table 1 CPU-Seconds/time step for AD, FD and SD methods
N – Number of atoms, P- Number of processors. (Plimpton *et al*., 1995)

METHOD	**N**	**P=32**	**P=64**
AD	500	0.0111	0.0088
	2048	0.0446	0.0336
	4000	0.0807	0.0616
	6912	0.138	0.103
	10976	0.22	0.164
	16384	0.337	0.249
	32000	0.635	0.474
	50000	0.993	0.74
FD	500	0.0098	0.00695
	2048	0.0359	0.025
	4000	0.112	0.0759
	6912	0.18	0.122
	10976	0.521	0.349
	16384	0.828	0.544
	32000	1.75	1.1
	50000	NA	6.04
SD	500	0.0129	0.0106
	2048	0.0321	0.0189
	6912	0.0768	0.0436
	16384	0.161	0.0874
	50000	0.42	0.224

AD, FD and SD have been used extensively by many MD codes. NAMD (Kale et al., 1996), is one such software that is being used extensively in the research field of ion channels. Apart from SD, NAMD has other features, like Object Oriented Programming (OOPS), multiple-time step integration, message-driven scheduling of processes, full electrostatics-calculation and multiple threads of control. Table 2 has the earliest results reported by NAMD with a system containing 4885 atoms simulated for 20 time steps.

Table 2 Earliest results of NAMD
N-Number of atoms, P-Number of processes (Kale *et al* .1996)

N	P	Run time sec/20 steps
4885	1	118
4885	2	655
4885	4	411
4885	8	241

Table 3 shows the earliest results of NAMD, compared with the results of X-PLOR and CHARMM, for a system containing 32687 atoms simulated for 1000 time steps (Kale et al., 1996).

Table 3 Comparison of NAMD with X-PLOR and CHARMM
Run time/1000 time steps in minutes, N-Number of atoms, P-Number of processes. (Kale *et al* .1996)

N	P	NAMD	X-PLOR	CHARMM
32687	1	304.72	237.45	255.78
32687	2	163.88	125.38	157.27
32687	4	92.06	75.45	119.25
32687	8	50.65	46.38	64.18

Although many MD codes recently developed, like NAMD, are using SD, it is disadvantageous with physical systems with complex geometries. Division of such a system into uniform cells is a tedious and complicated process. AD however has a disadvantage of requiring global communication of data, but has an inherent advantage of being simple in load balancing and distribution of force calculations to processes (Plimpton et al., 1995). New efficient algorithms of global communication and faster ways of electronically communicating data between processors of a cluster are being developed making AD a natural choice for building novel MD simulations

Table 4 shows a comparison of CPU seconds per communication call, varying the data sizes, reported for three different algorithms; (1) long (2) short and (3) hybrid. ‘Broadcast’ sends data from the root process (rank=0) to the rest of the processes. ‘Collect’ collects data from all the processes to store the data in an array local to the root process. ‘Global sum to all’ is a combination of ‘Collect’ and ‘Broadcast’, where the root collects the data, and the collection is sent to the rest of the processes, such that all the processes have a copy of the entire data. Figure 2 shows the hybrid algorithm for performing a ‘Broadcast’ operation on an array X[x0,x1,x2,x3]. The root has to send X to all the other processes invoked by the parallel code. Each of the arrows indicates a pair of ‘Send’ and ‘Receive’ operations between a pair of processes. Conventional Broadcast sends X to each process, which requires nP operations, where ‘n’ is the number of elements in the array and ‘P’, the number of processes invoked. Hybrid algorithm shown in Figure 2, sends half of the elements of the array to the next process, until n individual elements are distributed evenly to the first n processes, after which, the first n nodes send their copies of the elements to the rest of the processes. The adjacent processes communicate with a pair of ‘Send’ and ‘Receive’ operations until all the processes have a copy of the array X (Bruck et al.. 1997).

Table 4 New efficient algorithms for communications (Bruck *et al.*, 1997)
Comparison of CPU time in seconds taken by each algorithm for reported communication

Operation	Algorithm	Data size 256 B	Data size 262144 B	Data size 1048576 B
Bcast	short	0.00128	0.12401	0.48549
	long	0.02939	0.05426	0.11965
	hybrid	0.00139	0.03932	0.09957
Collect	short	0.00338	0.07789	0.29512
	long	0.02672	0.04358	0.08838
	hybrid	0.00355	0.03074	0.07469
Global sum to all	short	0.00676	0.15933	0.07469
	long	0.05547	0.09729	0.60273
	hybrid	0.00696	0.07251	0.19747

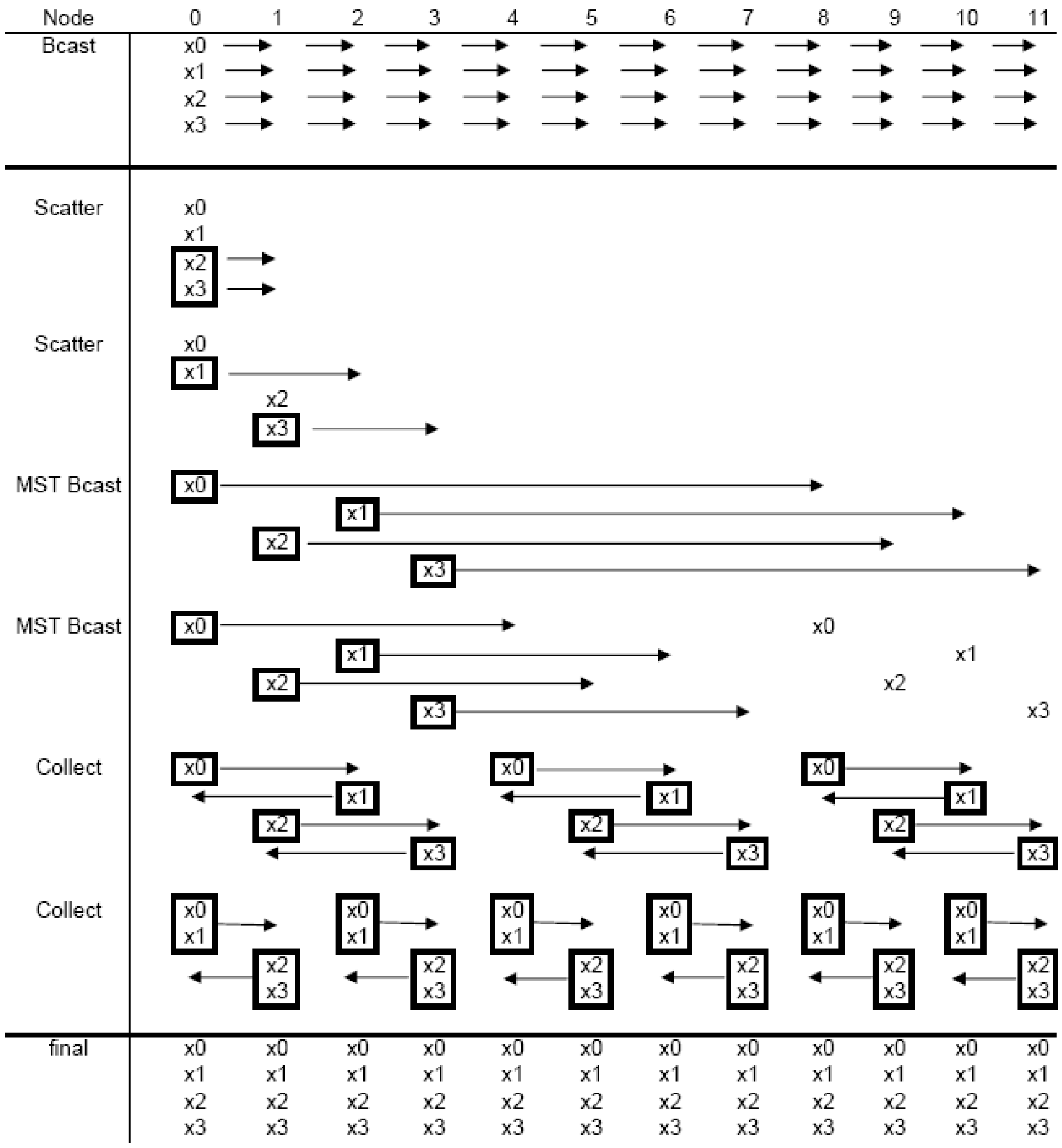


Figure 2 Hybrid algorithm for Broadcast
(Bruck *et al.*, 1997)

CHAPTER 3 Research Objectives

Although many algorithms, programs and models for MD simulations have been developed, there is still a huge potential for new developments that would address basic issues with all MD simulations. Customizing the code depending on the needs of the physical system is yet to be achieved in the field of computer simulation. Source codes declared by some of the softwares are filled with programming language intricacies- difficult to interpret and connect to the actual theory of MD. According to a study of parallel applications, published on 10/10/2006 in a newsletter for ‘Electronics, Design, Strategy’, the majority of parallel application prototypes (65 percent) are developed in very high level languages (VHLLs) such as MATLAB, Mathematica, Python, and R. A private organization, Simon Management Group, which offers business management solutions, conducted this study (www.simonmanagement.com).

The availability of computing resources is another major constraint in opting for a particular MD program. Also, debugging public domain and commercial MD softwares is a tedious process, as there are no simple debugging options normally available to users. The study conducted by Simon Management Group surveyed more than 500 users of parallel computing resources, to estimate the span of developing a typical parallel application, and revealed that 20% of the respondents’ projects consumed two to three years of their time.

In this scenario, where efficient but complicated hybrid MD/BD (Brownian Dynamics) codes exist; the transparency involved in using a straightforward parallel MD code could be the deciding-factor in its choice. As MD is an integral part of the more efficient hybrid multi-time-step MD/BD code (Ying et al., 2003), uncoupling the classic MD code from the hybrid code and making it fast and compact, gained the interest of this research.

The objective of this research is to develop a simple and efficient parallel MD code by cultivating a competent parallel scheme, establish a favorable communication system between the parallel processes calculating force terms and optimize the communication by reducing the idle time of the processes. The efficiency of the parallel MD code in cutting down the cpu-time, will be tested on water inside a potassium channel (KcsA). The scalability of the parallel program with 2000-60,000 atoms and 4-15 processes will be determined.

CHAPTER 4 Research Methods

4.1 Protein Data Bank

The structural aspects of the protein wall of KcsA were obtained from PDB file 1BL8 (Mackinnon et al., 1998). The text file thus obtained had 3D coordinates of atoms detected using X-ray Crystallography. The hydrogen atoms, being too small to be detected using X-ray crystallography, were missing. An example of the PDB format corresponding to the format given in Table 5 is,

```
ATOM      1  N   ALA A  23      65.191  22.037  48.576
```

Table 5 PDB format

S.No	Column	Field	Definition of the field
1	1-6	ATOM	Record name ATOM, HETATM
2	7-11	1	Serial number of the atom
3	13-16	N	Name of the Atom
4	18-20	ALA	Name of the residue
5	22	A	Chain identifier
6	23-26	23	Sequence number of the residue
8	31-38	65.191	Position Co ordinate x in Angstroms
9	39-46	22.037	Position Co ordinate y in Angstroms
10	47-54	48.576	Position Co ordinate z in Angstroms

4.2 Preparation of Input Files

The preparation of the system is shown in Figure 3. The missing hydrogen atoms were added to the PDB file wherever necessary using Accelrys Pro software. Masses were assigned to individual atoms and the centre of mass of the protein was determined. The origin was then shifted to the calculated centre of mass. The relative positions of the atoms were calculated and the partial atomic charges were added to the text file according to Amber 95 (Cornell et al., 1995). The new file obtained had around 5200 protein atoms and was named 'protdata'.

The Lennard-Jones (LJ) parameters describe the interaction potentials between atoms. The parameters were taken from AMBER 95 (Cornell et al., 1995). The input text file 'protdata' was read and the lj data of each atom was independently recorded into another text file. This file was named 'ljdata'.

SOLVATE 1.0 was used for introducing water as a solvent into the channel. The solute (protdata) was solvated in a cubic water box. Water atoms were placed starting from a minimum distance (overlap radius) from each atom. Layer by layer, water atoms were added until the specified thickness was met. The length of the box thus generated was 91.808 A0. The total number of water atoms in the box was approximately 60,000. The cubic model was chosen to make periodic boundary conditions easier. The positions of water atoms were recorded in a text file 'waterdata'.

The 3 files thus prepared, protdata, waterdata and ljdata, were input files to the main MD program 'potch.f'.

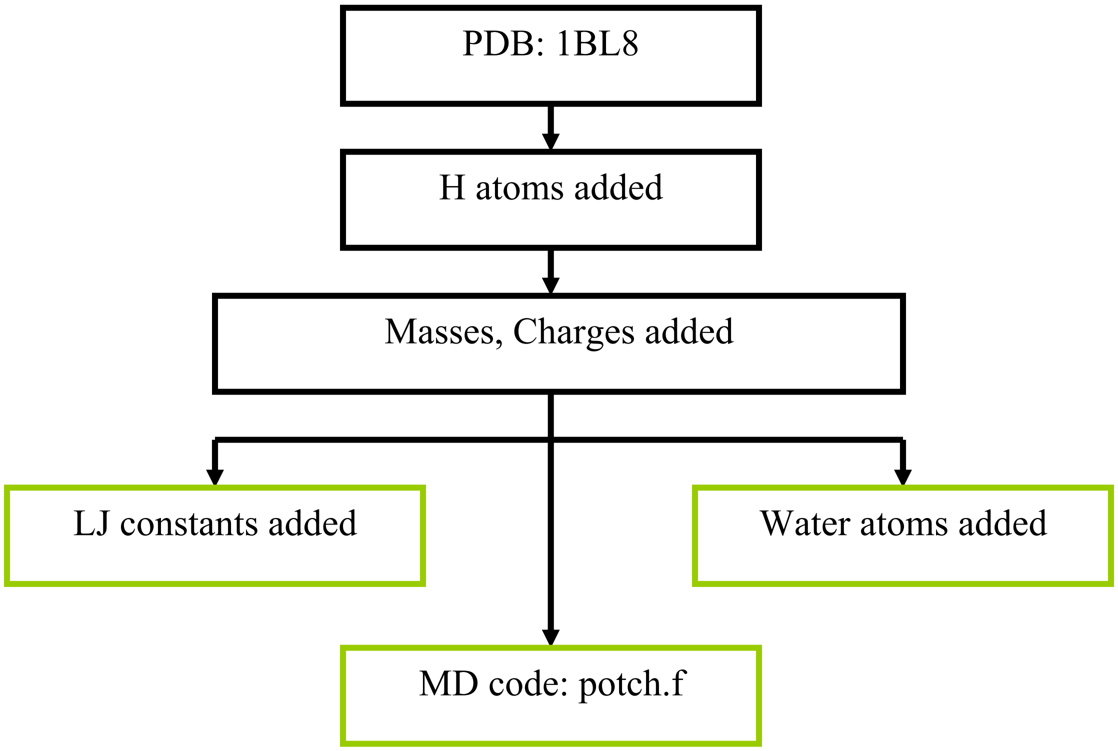


Figure 3 Flow chart, Preparation of Simulation System

4.3 MD Program potch.f

The structure of the program was divided into four parts,

1. Variable declaration. The data needed was declared by specifying the type of each variable and the length of each array.
2. Parameter Declaration. The parameters needed for the calculations were numerically declared. All the quantities were in SI units. The declared values were printed into text files. The list of the variables and their values are given in Table 6.
3. Reading and formatting input files. The input files, protdata, waterdata, ljdata, were read and the values were captured into the local variables declared in the program. The corrections needed for converting quantities to SI system of units were done. The water atoms were coded according to the type of the atom, icode = 1 for Oxygen and icode = 2 for Hydrogens. Modified Simple Point Charge (SPC) model of water was used. A number was assigned to each water molecule for identification, which was stored in the array 'imol'. The effective peptide diameter, peptide volume were calculated. The parameters are reported in Table-6. Scaling all the physical quantities avoided dealing with extremely large or extremely small values. The parameters were printed into separate files for checking.

Table 6 List of Simulation-Parameters,
SI units otherwise stated.

S.NO	Parameter	Value
1	Temperature of water	310.15
2	Mean fluid velocity	0.0
3	Boltzmann's constant	1.38048e-23
4	Molecular mass of water	2.99e-26
5	Relative H locations to O	(5.776e-11, 5.776e-11, 8.163e-11)
6	O-H Length (water)	0.1e-09
7	H-H Length (water)	0.16328e-09
8	Bulk water density (number/m3)	3.337e28
9	Dielectric constant of water	80.0
10	LJ constants for water (σ,ε) (nm,KL/mole)	(0.340, 0.680)
11	Cut off potential distance (nm)	0.85
12	Time step (femto seconds)	0.5
13	Number of water atoms in box	60000
14	Number of peptide atoms	5270
15	Pi	3.1415297

Actual MD. The details of the potential energy function used in the MD code are given in Figure 4. Verlet Leap Grog Algorithm of integration (Appendix B), conserving, number of atoms, volume and the temperature of the system, was implemented (Ying *et al*., 2003). Initial velocities of atoms were generated using Maxwellian distribution based on the temperature of the system. The random number generators used for this purpose were declared as subroutines and were defined within the program to make it portable. The generated velocities were scaled to the temperature of the system. The forces on atoms were initialized to zero. The time-step calculations of positions were done in a new loop over time, details of which are shown in Figure 5. The positions were updated after each time step. The periodic boundary conditions were applied and any atom that escaped the box was sent to the next periodic cell in that direction. The positions were printed into text files at this stage. The inter-molecular and intra-molecular forces of water atoms, the protein-water forces were dealt separately as shown in Figure 6. Modified SPC model was used for treating intra-molecular forces of water (Ying *et al*., 2003). Reaction Field theory was used for calculating electrostatic force components between atoms. (Ying *et al*., 2003). The calculated forces on each atom were totaled for calculating the combined force of all the other atoms. The three components of the force (fx,fy,fz) were calculated independently. The velocities of atoms at time step t, were determined from atomic displacements at time t+Δt. To keep temperature constant, velocity rescaling was used (Ying *et al*., 2003). The simulation was run for 1000 steps to allow water to equilibrate.

$$\boxed{U_{ij} = U_{bonded} + \sum_{i<j}^{atoms} \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \sum_{i<j}^{atoms} \frac{q_i q_j}{\varepsilon r_{ij}}}$$

Figure 4 Details of MD code

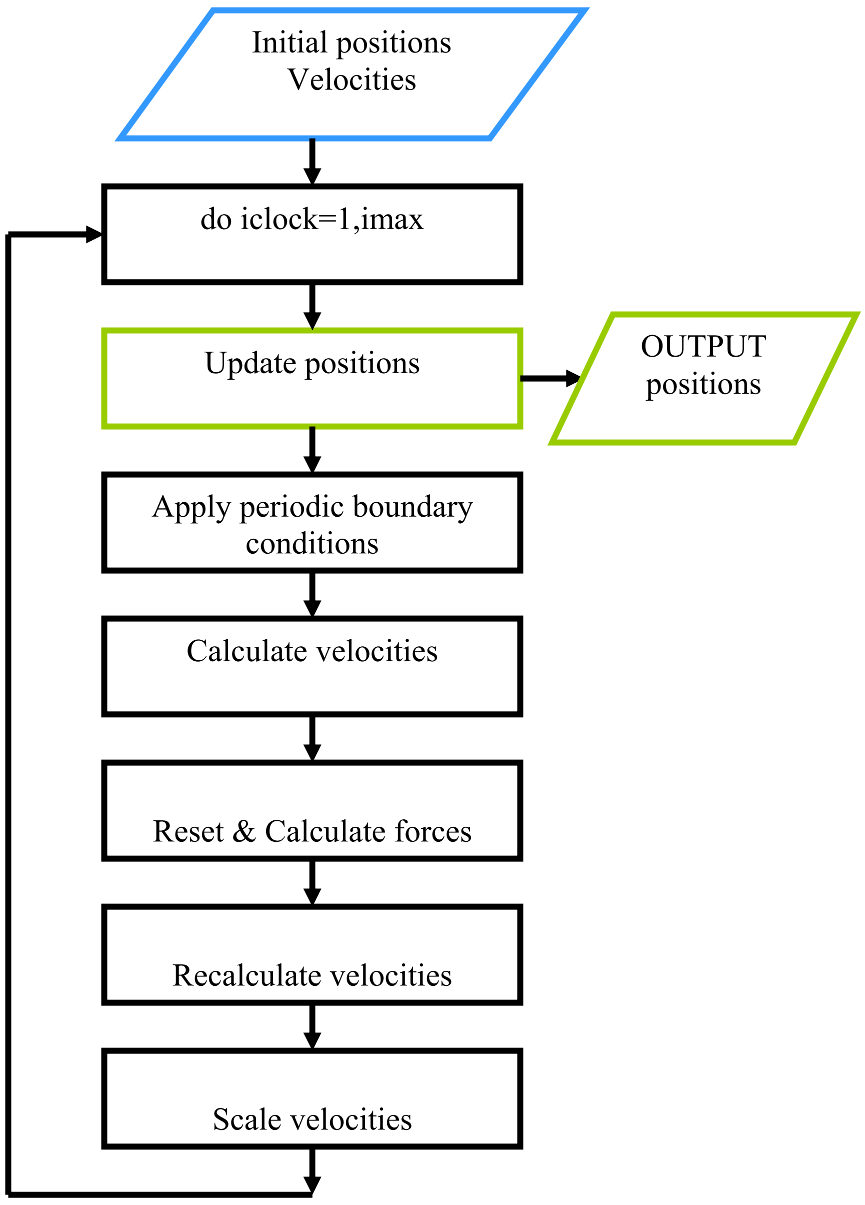


Figure 5 Flowchart, Loop over time

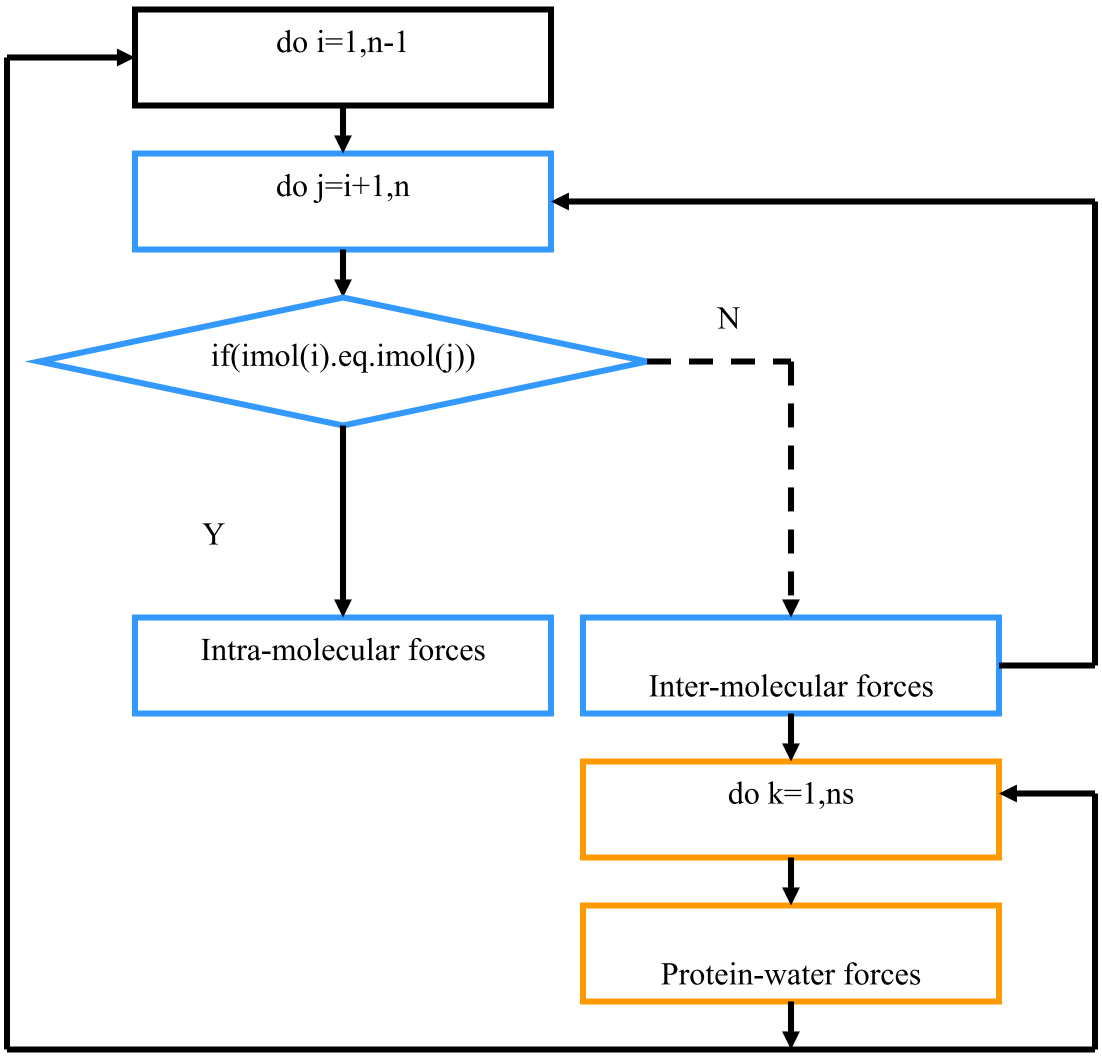


Figure 6 Flowchart, Force calculation

4.4 Parallel Schemes for MD

The literature classified parallel schemes being employed currently by most MD codes into three categories. All the schemes were described for a system of N atoms and P processors. Computation load and communication load on each processor were also discussed.

1. ATOM DECOMPOSITION METHOD (Plimpton et al., 1995) : The force calculations on a fixed group of atoms (N/P), go to fixed processors (P). Each processor calculates the total force on each atom in the group (N/P) allotted to it. The group of atoms mapped to each processor, remains unchanged with time. The computation load on each processor is of the order of O(N/P). The data that should be communicated between processors is of the order N, as all the processors need the positions of all the atoms for calculating the combined force on each atom. However, this reduces the N/N matrix of force elements to ((N/P)xN)) on each process. This method was used by earlier versions of CHARMM, GROMACS and the first version of Bluematter released by IBM
2. FORCE DECOMPOSITION (Plimpton et al., 1995) : The N^2 force elements are divided into blocks of size $(N/\sqrt{P})x(N/\sqrt{P})$, numbered starting from the first row to the last row. These numbers are formed into two strings (x, y) where x consists of row-wise generated numbers and y consists of column-wise generated numbers. A processor P(K), calculates the forces on elements in the

block numbered x(k), caused by the elements in the block numbered y(k).

Computation load for this method is of the order $O((N/P)+(N/\sqrt{P}))$.

Communication load is of the order $O(3N/\sqrt{P})$

3. SPATIAL DECOMPOSITION (Plimpton et al., 1995) : The simulation box is physically divided into 3-D shells and each shell is allotted to a processor. Atoms move through these shells as time proceeds. The size and shape of each shell depend on, N, P and the cut off radius used for interactions. As this method takes full advantage of cut off radius, the data needed for calculation of forces on atoms is local to the shell and a small layer of the shells surrounding each shell. Thus the communication load is of the order $O(6r(N/P)^{2/3})$, where r is the cut off radius. Computation load is of the order $O((N/P)+(6r(N/P)^{2/3}))$. The disadvantage of this method is complication involved in physically splitting the system into equal spaces to balance the load on each processor and communication of data between adjacent shells.

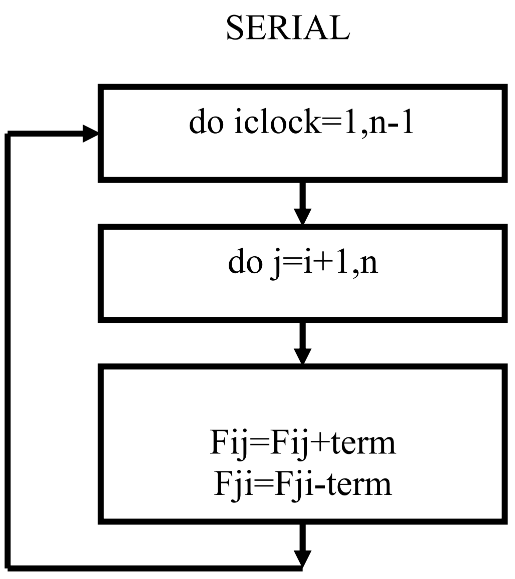
4.5 Choice of the Parallel Scheme

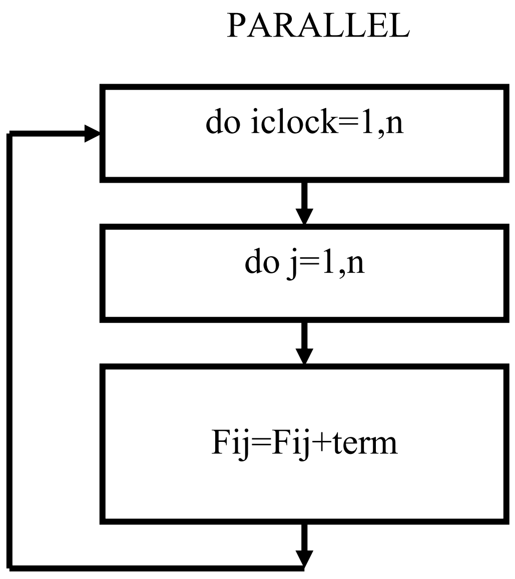
Given the limit over the number of nodes available, which was 15, the literature suggested that AD, FD, SD would give similar results in cutting down the execution time. AD, being simple to incorporate into the existing serial program, was chosen. There were two options in incorporating AD for a system of atoms, (1) Master Slave Model (MSM), (2) Traditional AD (TAD). In MSM, the root does not share the computation load and is responsible only for driving the communication between the rest of the processes to complete computation. Load distribution over processes is uneven in this case. TAD makes the root share the computation load thus guaranteeing load balance but has double the communication load of the MSM.

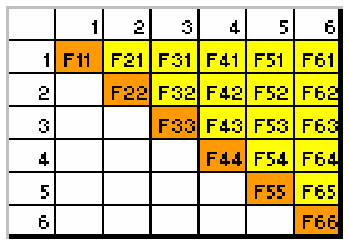
A hybrid algorithm of MSM and TAD was developed for incorporating the good features, load-balance and optimal communication. A slight over-load on the root was created by making it, share the computation load, and for driving the communication between processes. Placing a function after computation and communication leveled the slight imbalance thus created, such that all the processes approaching the function, would execute the next statement only when each process has called the function.

4.6 Preparation of potch.f for Introducing Parallel Scheme

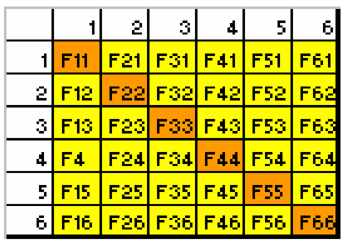
Newton’s Third Law was used in calculating F_{ji} as $-F_{ij}$ in the serial program. Considering a square matrix of force components shown in the Figure 7, only the upper triangular components are needed. In parallel environment, these components have to be inter-communicated such that the lower-triangular elements are derived from the upper triangular components. In a parallel code, this calls for a communication of the force term Fji between the process responsible for calculations on ‘i’ and the process responsible for calculations on atom ‘j’. Such communications were complicated to handle in a parallel environment due to imbalances over the number of ‘Send’ and ‘Recv’ functions that would result on each process. Hence Newton’s Third Law was not used in parallel program. Thus, intra-molecular forces for water molecules were most effectively calculated in serial mode following which inter-molecular and protein-water molecular forces were calculated in parallel.







	1	2	3	4	5	6
1	F11	F21	F31	F41	F51	F61
2		F22	F32	F42	F52	F62
3			F33	F43	F53	F63
4				F44	F54	F64
5					F55	F65
6						F66



	1	2	3	4	5	6
1	F11	F21	F31	F41	F51	F61
2	F12	F22	F32	F42	F52	F62
3	F13	F23	F33	F43	F53	F63
4	F4	F24	F34	F44	F54	F64
5	F15	F25	F35	F45	F55	F65
6	F16	F26	F36	F46	F56	F66

Figure 7 Flowchart, Application of Newton's Third Law

4.7 Computing Cluster

The parallel MD program was written in Fortran 90 with parallel functions from the library mpich-1.2.7. The program was run on 'Bach' (bach.vcu.edu), a linux cluster with 500 processors, 1 TB RAM and 73 GB of internal memory. The nodes of Bach use ethernet to communicate with each other. Every user of Bach was given a username and password for logging to the main node. All other nodes had identification numbers (node-id) (ex: bach45). All programs were compiled using 'mpif90' and linked to the parallel environment using 'mpirun'. After compilation a linux shell script with statements for executing the program on the assigned node, was submitted to the engine. The script was submitted to the system using the command, 'qsub'. The submission script used is given in the Appendix A. Each submission thus made, called 'job' had a distinct identification number (job-id) on Bach. Once a job was initiated, the command 'qstat' was used to check the status of the job, which showed job-id, submission time, and the node-id. All the jobs were submitted on the head node. The list of commands used on Bach is given in Table 7

Table 7 List of commands on Bach

Function	Command
Compiling	$ usr/global/mpich-1.2.7/bin/mpif90 -c filename.f
Linking	$ usr/global/mpich-1.2.7/bin/mpif90 –o filename filename.f
Linking to parallel library	This has to be included in the submission script submit.sh. (APPENDIX A) usr/global/mpich-1.2.7/bin/mpirun –np p filename (p = number of processes)
Submitting a job to cluster	$ qsub submit.sh
Checking the status of a job	$ qstat –u username

4.8 Parallel Programming Library mpich-1.2.7

Before any parallel schemes were employed in the main MD program potch.f, the parallel library mpich-1.2.7, and parallel functions were tested using elementary programs. The submission script is reported in the Appendix A. Communication between processes was tested using both blocking and non-blocking modes. The list of functions tested is shown in Table 8. All the functions are called using a tag, 'call'

'MPI_Init ( )' and 'MPI_Finalize ( )' are first 2 of the 3 important and basic functions which appear in all parallel programs. 'MPI_Init ( )' marks the beginning of parallel environment and has to be called right after the variable declaration. 'MPI_Finalize ( )' marks the end the parallel code. The arguments common to the three basic functions are the communicator 'MPI_COMM_WORLD', which is used through out the code for communicating data, and an error tag 'ierr', which stores a number indicating either success or failure of the function. The error tag is a common argument to all the parallel functions listed in the Table 8.

'MPI_Comm_rank ( )' has 3 arguments, an integer variable which stores the rank of the process calling it, 'MPI_COMM_WORLD' and 'ierr'. This has to be called after MPI_Init( ).

'MPI_Comm_size ( )', again, has 3 arguments, an integer variable which stores the total number of processes initiated by the command 'mpirun', the communicator, 'MPI_COMM_WORLD' and the error tag, 'ierr'.

‘MPI_Send ( )’ (Send) is used by a process to send a single element or a string of elements to another process. The first argument is the starting address of first element of the array, the second argument is an integer which has the number of elements to be sent, the third argument is the data type of the elements being sent, the fourth argument is the rank of the destination, the fifth argument is a tag which is used to match the ‘Send’ with the corresponding receiving call made by the destination, sixth argument is the communicator ‘MPI_COMM_WORLD’ and the seventh argument is the error tag ‘ierr’. The communication completes when the destination process receives the data. Until the communication is complete, the statement after ‘MPI_Send ( )’ is not executed. In other words, ‘MPI_Send ( )’ blocks the program until the receiving process calls the corresponding ‘MPI_Recv ( )’.

MPI_ISend ( ), contrastingly, executes the next statement of the code as soon as it is called.

‘MPI_Recv ( )’ (Recv) receives a single element of the array or a string of elements from the process sending the data. The first argument is the starting address of the first element of the array, the second argument is an integer which has the number of elements to be received, the third argument is the data type of the elements being received, the fourth argument is the rank of the sending process, the fifth argument is a tag which is used to match the ‘Recv’ with corresponding the ‘Send’ call, the sixth argument is an integer which stores the information of the data being received, the seventh argument is the communicator operating between sending and receiving processes and the eighth argument is the error tag.

‘MPI_Recv ( )’ is a blocking communication and does execute the next statement until the communication is complete. ‘MPI_IRecv ( )’ is a non-blocking version of the ‘Recv’ call. A blocking ‘Send’ can be paired with a non-blocking ‘Recv’. Similarly, a blocking ‘Recv’ can be paired with a non-blocking ‘Send’. Such combinations, when improperly used, have a disadvantage of creating a dead lock, where a communication cannot be completed. An example of such a dead lock is shown in Figure 8 where two calls of ‘Send’ are made with two different tags. The receiving process has ‘Recv’ statements in the reverse order of tags. This creates a deadlock where intended communications do not happen.

Table 8 List of basic mpich-1.2.7functions

Functions tested	Duty
MPI_Init( )	Initializes MPI environment
MPI_Comm_rank( )	Stores the ID of the process in the variable declared in its argument
MPI_Comm_size( )	Stores the total number of processes in the variable declared in its argument
MPI_Send( )	Sends the buffer declared, to its destination by specifying the communicator. The buffer is cleared after the communication ends.
MPI_Recv( )	Receives the buffer through communicator and stores it in variable specified in its argument.
MPI_ISend( )	Sends the buffer declared, to its destination by specifying the communicator. The buffer is cleared instantly.
MPI_IRecv( )	Receives the buffer through communicator and stores it in variable specified in its argument. The code is blocked until the communication ends.
MPI_Bcast( )	Broadcasts the buffer to all the processes.
MPI_Barrier( )	Blocks the code until all the processes call it.
MPI_Finalize( )	End of MPI.

```
if(myid.eq.0) then
  dest=1
   a=2
   b=3
   tag=0
   dtag=1
   call MPI_Send(a,1,MPI_INTEGER,dest,tag,MPI_COMM_WORLD,ierr)
   call MPI_Send(b,1,MPI_INTEGER,desr,dtag,MPI_COMM_WORLD,ierr)
 end if
if(myid.eq.1)  then
  tag=0
  dtag=1
  call MPI_Recv(b,1,MPI_INTEGER,0,dtag,MPI_COMM_WORLD,stat,ierr)
  call MPI_Recv(a,1,MPI_INTEGER,0,tag,MPI_COMM_WORLD,stat,ierr)
  print *,'a=',a,'b=',b
end if
```

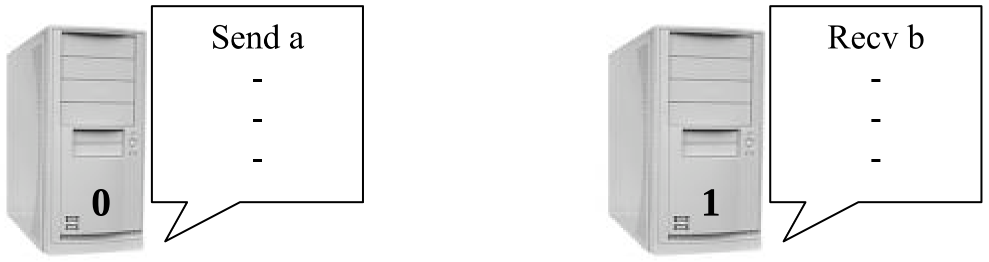


Figure 8 Deadlock created with blocking Send and Recv calls

4.9 Designing the Parallel Algorithm

After testing the basic functions of the parallel library mpich-1.2.7, the hybrid algorithm of TAD and MSM was designed. The parallel scheme employed is shown in the flow Chart of Figure 9. The first three parts of the code were common to all the processes. (1) variable declaration (2) fixing the parameters (3) reading and formatting inputs.

As shown in the flowchart in Figure 9, after the do loop on the time step, the positions were updated and broadcasted to all the processes by the root process. The force calculations on the first P atoms were taken care by the corresponding P processes. If ‘i’ was the id of the first atom assigned to the processor Pi, the group of atoms assigned to the process Pi was (i, i+P,i+2P,i+3P…). Thus load balance was achieved in this scheme. The calculated force components in x, y and z directions, on each atom were sent to the root. After the processes calculated forces components, the root received the forces, calculated velocities and updated positions. The positions were broadcast to all the processes for calculation of forces in the next time step.

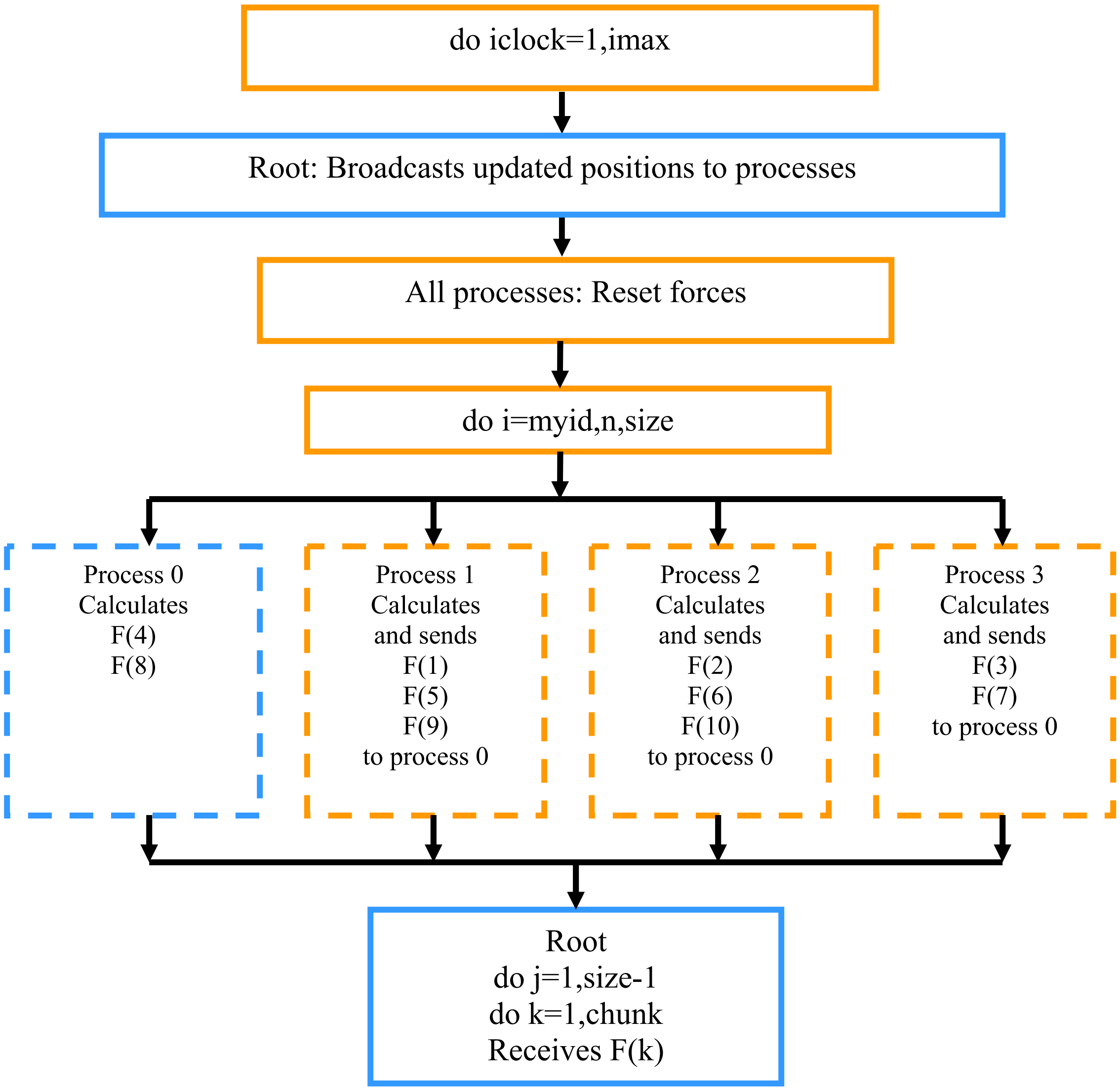

do iclock=1,imax
Root: Broadcasts updated positions to processes
All processes: Reset forces
do i=myid,n,size
Process 0 Calculates F(4) F(8)
Process 1 Calculates and sends F(1) F(5) F(9) to process 0
Process 2 Calculates and sends F(2) F(6) F(10) to process 0
Process 3 Calculates and sends F(3) F(7) to process 0
Root do j=1,size-1 do k=1,chunk Receives F(k)

Figure 9 Flowchart of the algorithm designed

4.10 Communications - Test

A model program ‘newcomtest.f’, listed in Appendix B, was developed for testing the communication adopted in the algorithm. As a part of the test program, positions and forces on atoms were declared as real data types. An initial loop declared all the positions of atoms as real numbers. Another loop calculated the force components over atoms as half the value of the declared positions. The force calculations were done in parallel and were sent to the root. The root received the force components and reassigned the values of individual force components to the corresponding atom-positions, which were broadcast to all the processes in the next time step. The positions of atoms were printed at each time step to check if their values halved each time.

4.11 Debugging the Test Program

The model program resulted in the p4_error and segmentation problems. It was presumed that the communication sandwiched in between computational steps caused segmentation errors. Each process calculated the total force on each atom allotted to it and communicated the value of the total force on each atom with the root. The root had to schedule ‘Recv’ statements according to the order of ‘Send’ calls made by processes, which was fixed by the program in the increasing order of the id's of the processes. With N force terms and corresponding N 'Send' calls made by the processes, scheduling the calls made by processes created congestion.

The algorithm was improved to reduce the number of 'Send' and 'Recv' statements keeping the total size of the data constant. This was achieved by gathering all the force components calculated by a process into 3 arrays, each for x,y,z components, with their corresponding atom id's into another array. After all the calculations on the set of allotted atoms were done, each process would send the arrays to the root. The root would receive arrays from each process, in the order of their ranks (id's) and retrieved force components based on the atom id's from the received arrays. This would reduce the number of 'Send' and 'Recv' calls from '6N' (N=number of atoms) to '6*(P-1)' (P=number of processes), keeping the size of the data constant.

Upon testing, the new model continued to give segmentation errors.

Table 9 List of errors in test progam.

ERROR	Interpretation
P4_error SIGEGV: Could not send to fd=5	Segmentation error: One of the processes died early, leaving the communication un successful (Algorithm should be reviewed)
P4_error SIGEGV: Could not write to fd=5	Segmentation error: One of the processes dies early, leaving the communication un successful (Algorithm should be reviewed)

It was supposed that bringing entire communication together could resolve the segmentation problem. The 'Broadcast' calls, which were originally scheduled before force calculations started, were shifted and placed after the 'Recv' calls from the root. The communication then functioned as a packet of 'Send' and 'Recv' calls, which sent, received and broadcast values after each process completed its calculations. Upon testing, the model 'newcomtest.f' functioned as anticipated.

4.12 Parallel MD Program, parmd.f

Following the developed algorithm and model, parallel MD program parmd.f was developed.

The first three parts of the code, (1) variable declarations (2) fixing the parameters (3) reading and formatting inputs, were kept common to all the processes. The force calculations on the first P atoms were taken care by the corresponding P processes. Pi calculated forces on, (i, i+P,i+2P,i+3P…) atoms. The calculated combined force component on each atom was stored in an array, local to each process. The ID's of the atoms were stored in another array in the order (i, i+P,i+2P,i+3P…). Thus each process had four arrays, tempfx, tempfy, tempfz, temp, three for the force components and one for the atom ID's. These were sent to the root after each process completed the calculations on the assigned N/P atoms.

The root received the arrays in the order of the ranks of the processes and stored them into a consorted array, which had total force components on each atom. The root, from the forces received, calculated velocities and positions from the retrieved force components. The root broadcast updated positions to all the processes. This step ended the communication packet and the loop over time. As a fresh time-step started, force calculations resumed using updated positions.

MPI_Barrier ( ) calls were used before and after the loop, which calculated the forces to guarantee that all the processes finish their calculations before communication start and all processes start fresh calculation of force components only after receiving the updated positions from the root.

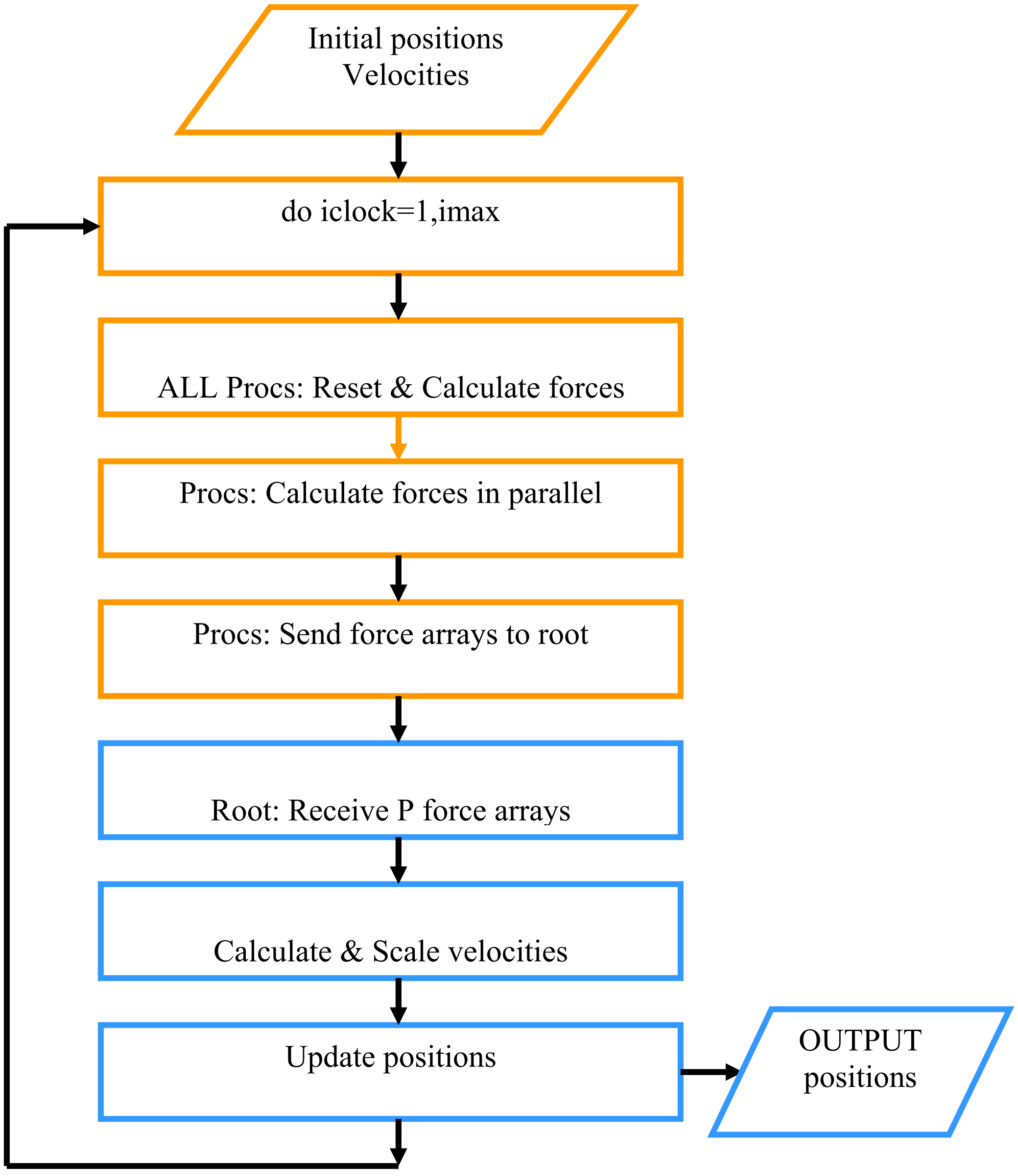
4.13 Debugging the Parallel MD program, parmd.f.

The parallel program had occasional reports of segmentation problems. Introducing message driven communication optimized the scheduling of 'Send' calls.

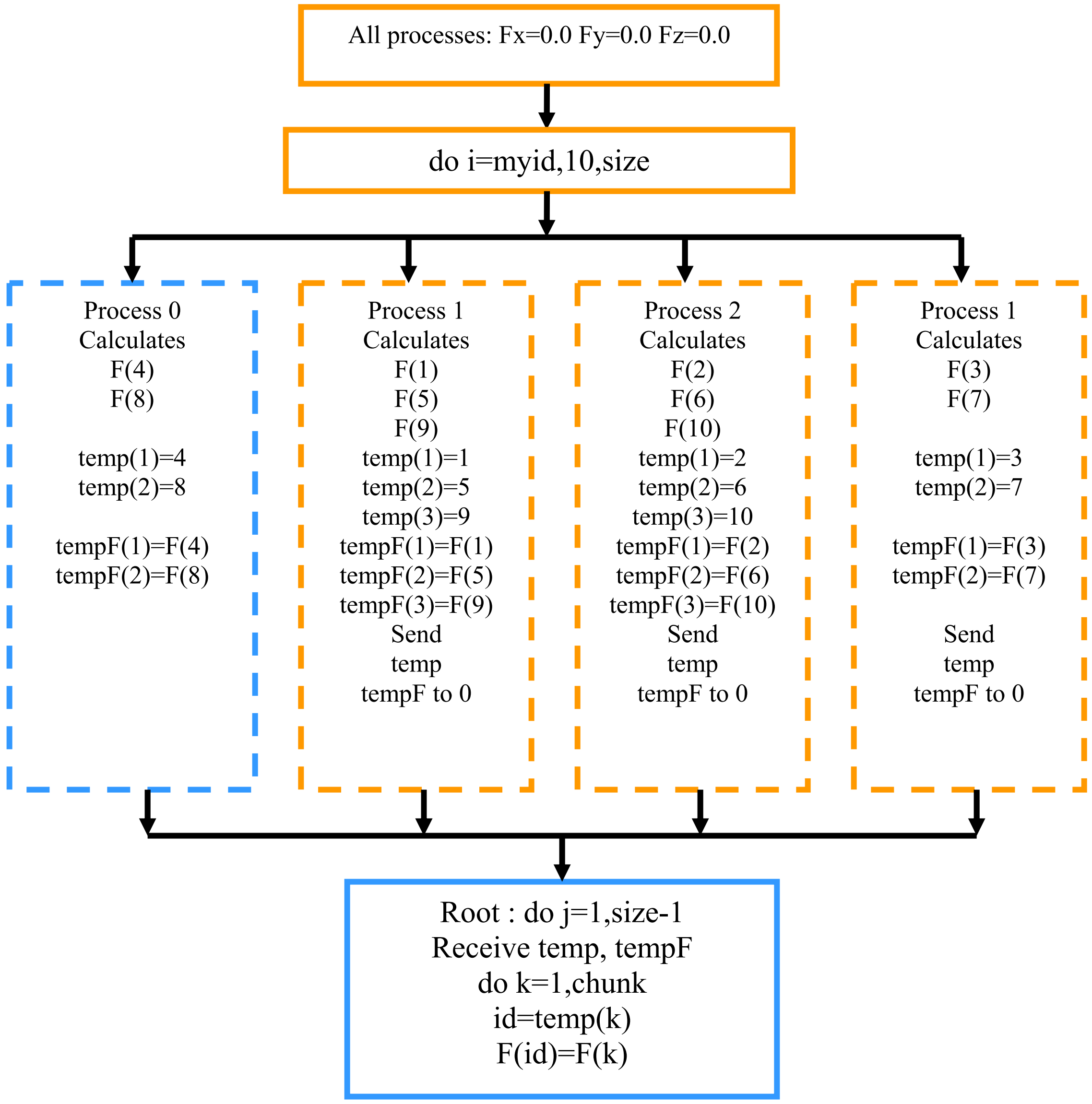
MPI_Any_Source ( ), was used in the 'Recv' calls made by the root. This would allow a 'Recv' statement to communicate with any process, without the condition imposed on it earlier to receive data from processes in the order of their ranks. This is shown in Figure 11. In other words, the order of processes calling 'Recv', which was fixed to be in the order of their ranks, was changed such that the process finishing calculations first would call the ‘Recv’ first, making it the FIFO (first in first out) form of the queue system. The rank of the process was then stored in a variable, which was used in the following ‘Recv’ statements such that all the ‘Recv’ statements were called by a single process at a time.

The overlap distance employed was increased to 0.3 angstroms. Any smaller quantities were corrected to 0.3. All the variables were declared to avoid errors occurring from data type of the variable. ‘Implicit none’ statement was used to assure there are no quantities left undeclared.

After the listed changes were made, the program functioned as anticipated. The results obtained were compared with serial program and were found to be the same. The final flowcharts of calculations inside loop over time and force calculations for a system of 10 atoms and 4 processes are presented in Figure 10 (a) and (b).


Initial positions
Velocities
do iclock=1,imax
ALL Procs: Reset & Calculate forces
Procs: Calculate forces in parallel
Procs: Send force arrays to root
Root: Receive P force arrays
Calculate & Scale velocities
Update positions
OUTPUT
positions

(a)



(b)

Figure 10
(a) and (b) :Calculations inside time-loop of parmd.f

```
c BARRIER for all processes to complete calculations
       call MPI_BARRIER(MPI_COMM_WORLD,ierr)
       print*,'BEFORE WAIT*****I AM******',myid

c***************************************************
c if process is not head, send calculated forces to head
c
          if(myid.ne.0) then
           call MPI_SEND(temp(1),sz,MPI_INTEGER,0,0,
     &      MPI_COMM_WORLD,ierr)
           call MPI_SEND(tempfx(1),sz,MPI_REAL,0,1,
     &      MPI_COMM_WORLD,ierr)
           call MPI_SEND(tempfy(1),sz,MPI_REAL,0,2,
     &      MPI_COMM_WORLD,ierr)
           call MPI_SEND(tempfz(1),sz,MPI_REAL,0,3,
     &      MPI_COMM_WORLD,ierr)
           endif
c********************************************************

c**********************************************************
c head stores collected forces into their proper places f
c head computes velocities
c
c Head revceives the calculated forces
           if(myid.eq.0) then
           k=1
           counter=1
           do 523 j=1,size-1
           call MPI_RECV(temp1(1),sz,MPI_INTEGER,MPI_ANY_SOURCE,
     &      0,MPI_COMM_WORLD,stats,ierr)
           iam=stats(MPI_SOURCE)
           call MPI_RECV(tx(1),sz,MPI_REAL,iam,1,
     &      MPI_COMM_WORLD,stats,ierr)
           call MPI_RECV(ty(1),sz,MPI_REAL,iam,2,
     &      MPI_COMM_WORLD,stats,ierr)
           call MPI_RECV(tz(1),sz,MPI_REAL,iam,3,
     &      MPI_COMM_WORLD,stats,ierr)
            do 524 i=1,sz
           temp3=temp1(i)
           if(temp3.ne.0) then
           fx(temp3) = tx(i)
           fy(temp3) = ty(i)
           fz(temp3) = tz(i)
           write(21,501)iclock,temp3,icode(temp3),fx(temp3)
 501       format(i4,1x,i8,1x,i6,1x,10f10.2)
          counter=counter+1
           endif
 524       continue
 523       continue
```

Figure 11 Reception and retrieval of forces with message driven scheduling of processes

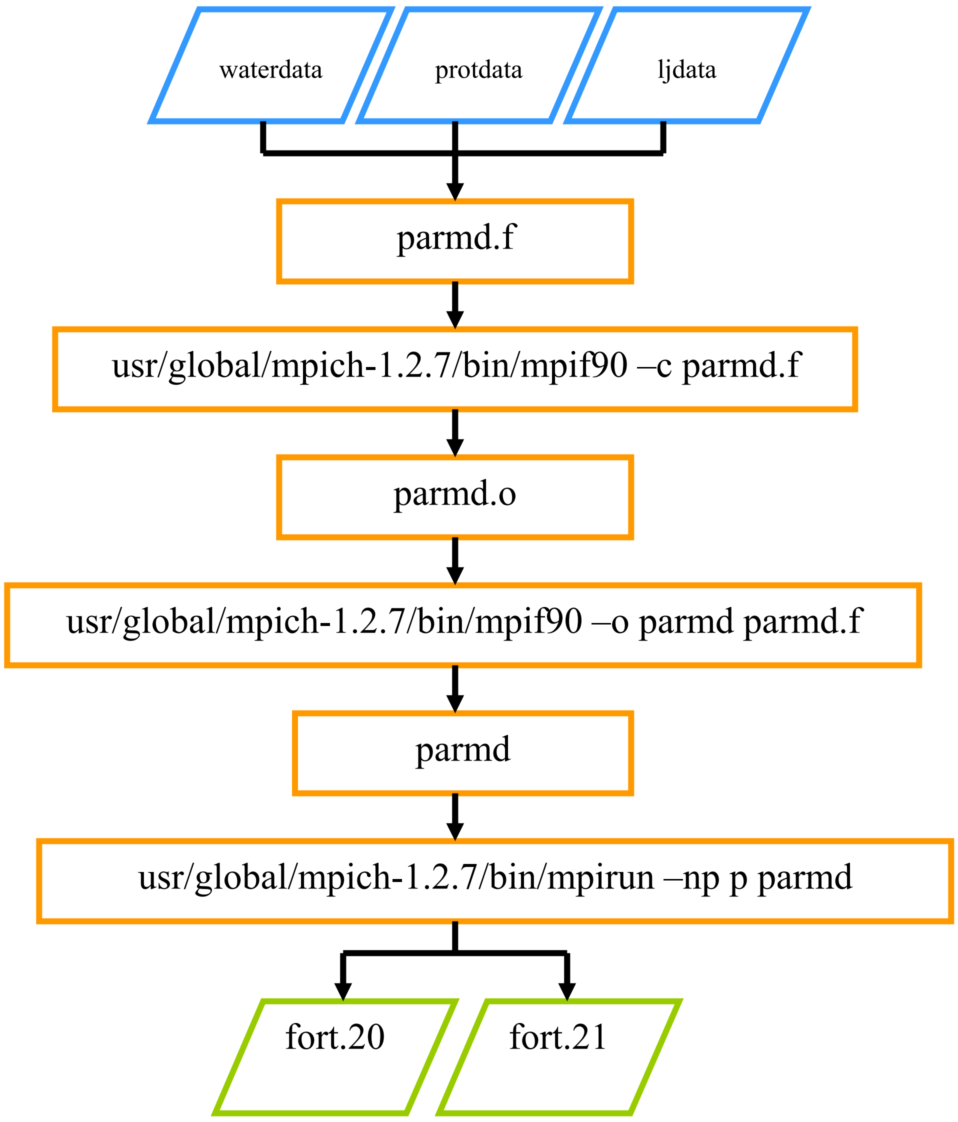

waterdata
protdata
ljdata
parmd.f
usr/global/mpich-1.2.7/bin/mpif90 –c parmd.f
parmd.o
usr/global/mpich-1.2.7/bin/mpif90 –o parmd parmd.f
parmd
usr/global/mpich-1.2.7/bin/mpirun –np p parmd
fort.20
fort.21

Figure 12 Flowchart, input files, execution and output files.

CHAPTER 5 Results and Discussions

5.1 Performance of the Parallel Program after Debugging.

The results from parallel program were consistent after debugging. The graph shown in Figure 13 was generated in a microsoft excel sheet using the data from cpu_time per time-step, for 10 time-steps, on the system comprising 20000 water atoms. The parallel program ran on 4 processors. The ideally parallel curve was generated by dividing each data point of the cpu_times recorded by the serial program by the number of processes declared for the parallel program. The lag observed between the curve generated by the parallel program and the ideal parallel curve was a result of the time elapsed in communication of data between processes.

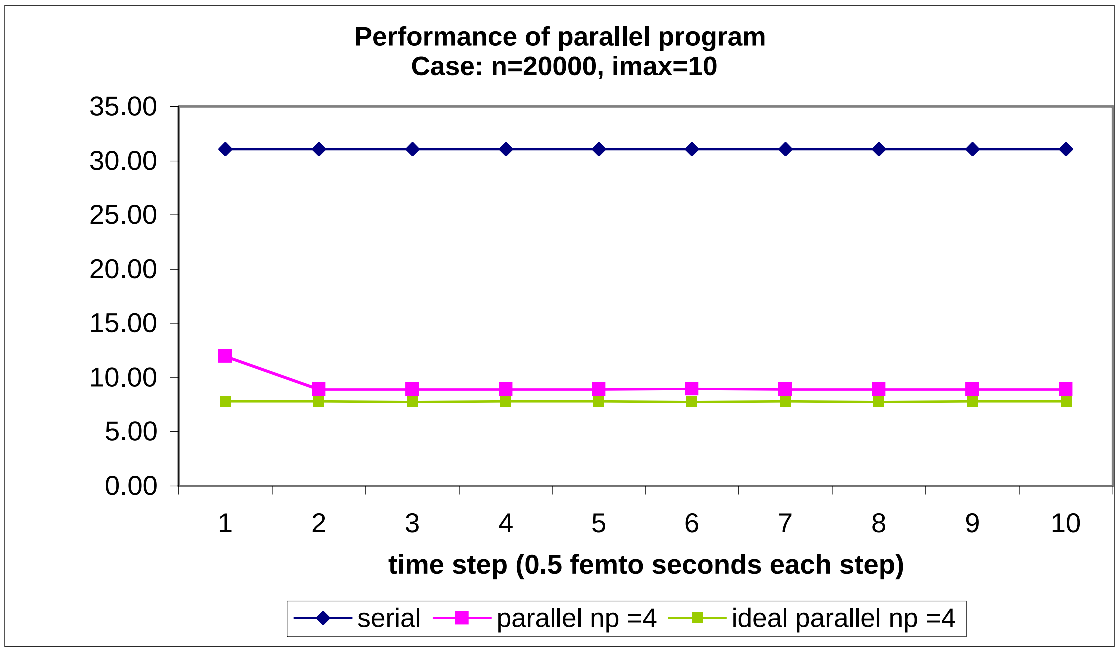


Figure 13 Graph showing comparison of potch.f and parmd.f

5.2 Scaling with Number of Atoms

The scaling of the parallel program with number of atoms, in terms of cpu_time per time step was determined for 2000-60000 atoms. However, the complete simulation of KcsA is only relevant for the case of 60,000 water atoms. The graph shown in Figure 14, representing the scaling of parallel and serial programs with number of atoms was generated in a microsoft excel sheet using means of cpu_time per time-step for 10 time steps on 4 processors for the parallel program. As the number of atoms increased beyond 20,000, there was almost four-fold reduction in the time taken by the parallel program when compared to the serial program. This was important because most of the biological systems had more than 20,000 atoms and needed 100,000 – 100,00,000 time for complete study of their dynamics. The ideal parallel curve was generated by dividing the cpu_times recorded by the serial program by number of processes declared for the parallel program. For the typical case of potassium channel being considered, a complete simulation, with 60,000 water atoms, a 40,000 time step simulation would run for almost 84 days with the serial program and 22 days with the parallel program developed.

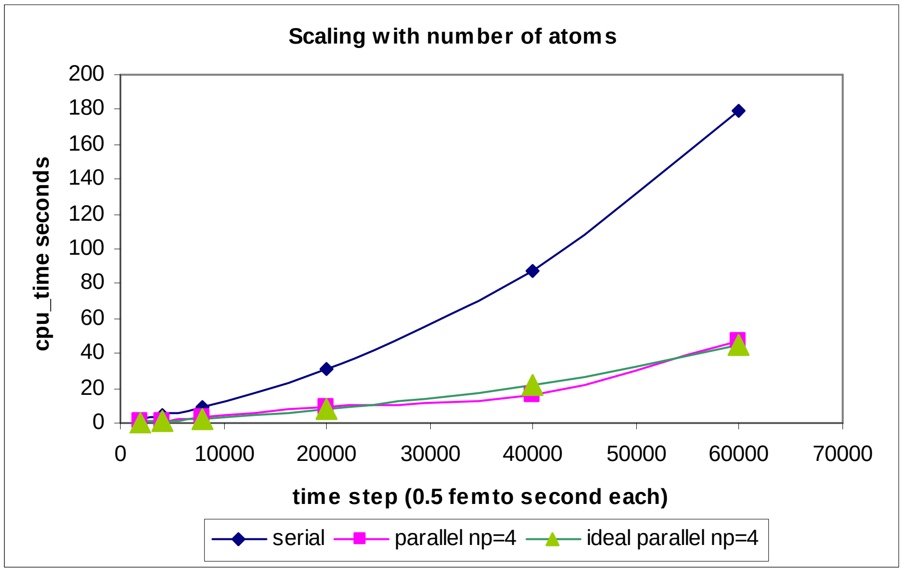


Figure 14 Graph showing scaling of parmd.f with number of atoms

5.3 Scaling with Number of Processes

As the number of processes assigned to a parallel program increase, the execution time of the program decreases down to a point after which the advantage of parallel computing balance out the communication time between processes. For this reason, different parallel algorithms designed for the same physical system, differ in scaling with number of processes, depending on the way communication calls are made. A variety of parallel MD codes, each of which are effective in a particular range of processes, have been invented.

The scaling of parallel program with number of processes was tested on a system with 20,000 water atoms for 10 time-steps. Each data point in the graph shown in Figure 15 represents the mean of the cpu_time per time-step recorded by the program for a number of processes. The tipping point of the curve, beyond which, the communication between processes takes over the advantage of parallel computation of forces, could not be established as the users of Bach were allowed to use a maximum of 15 processes. It was observed that the program scaled well for all the tested cases.

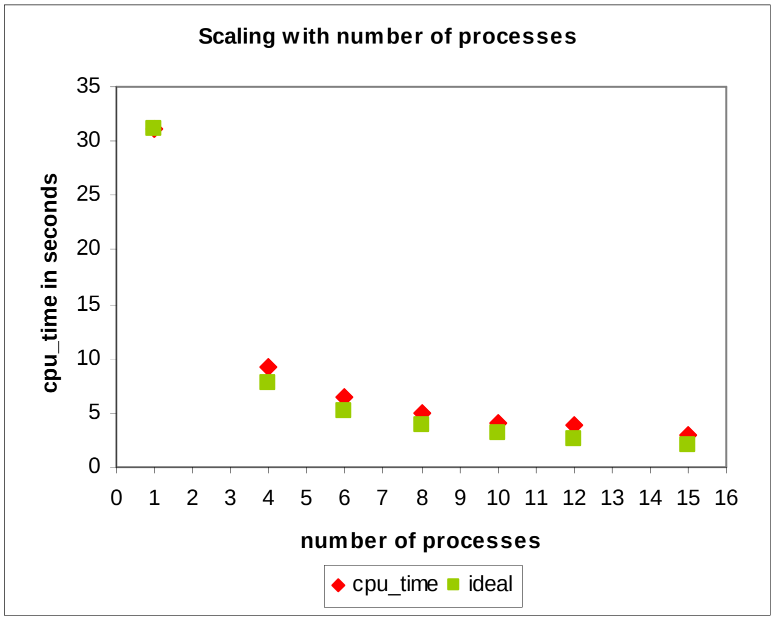


Figure 15 Graph showing the scaling of parmd.f with number of processes

Subtracting the calculated ideal cpu_time from the recorded mean cpu_times by the parallel program parmd.f, generated the graph shown in Figure 16. For a system of 20,000 atoms, Figure 16 shows the best choice of number of processes, which was observed as 15.

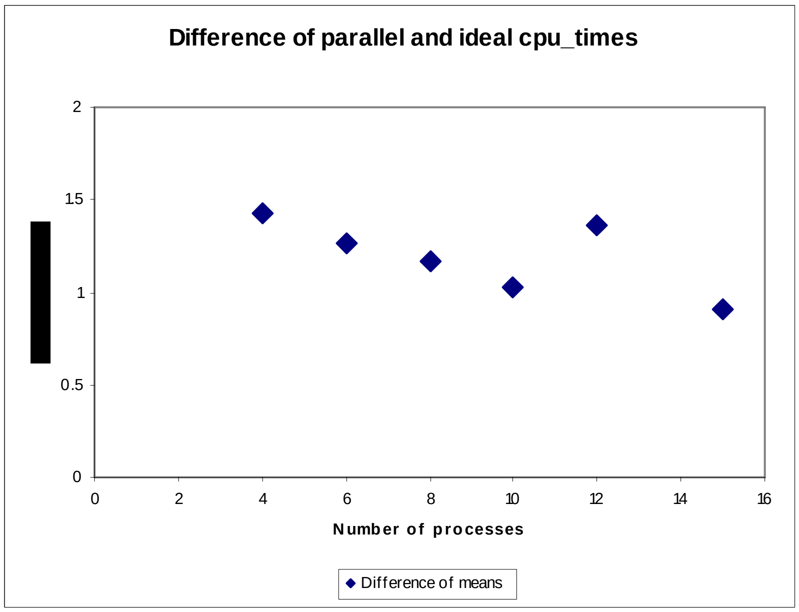


Figure 16 Graph showing the scaling of parmd.f with number of processes

5.4 Load Balance over Processes.

As each process calculated the total force executed by the system on the chunk of atoms allotted, computation-wise, equal load was imposed on all of the processes. This was tested on a system with 20,000 atoms and 15 processes. Figure 17 shows the distribution of number of atoms over the processes. It can be noticed that processes shared the computation on atoms equally, with an extra atom being assigned to processes 1,2 and 3. Figure 18 shows the cpu_time per time step, recorded by each process of the parallel program. Instead of printing the cpu_time to a file, 'print to screen' option was used to record the cpu_time. It can be noticed that the cpu_times recorded by the processes are evenly distributed around 5 seconds/time step.

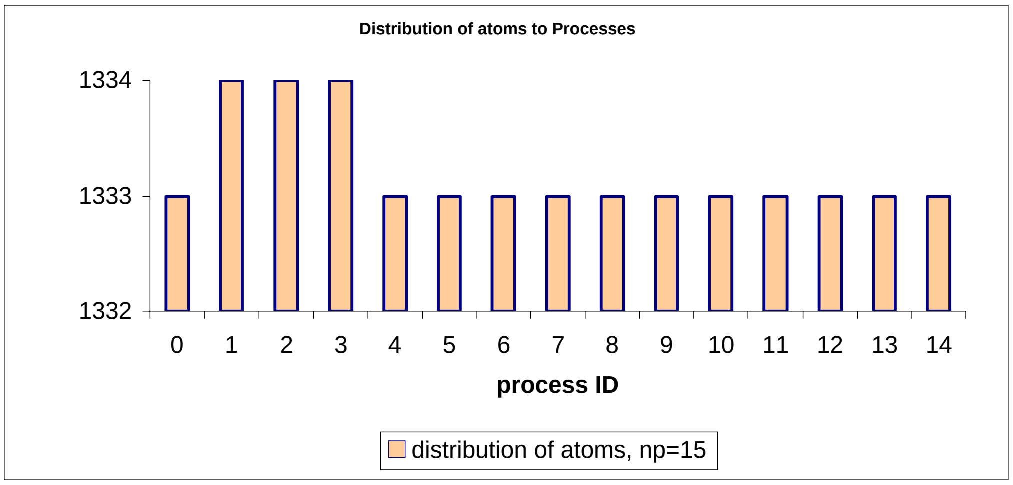


Figure 17 Graph showing the load balance on processes with parmd.f

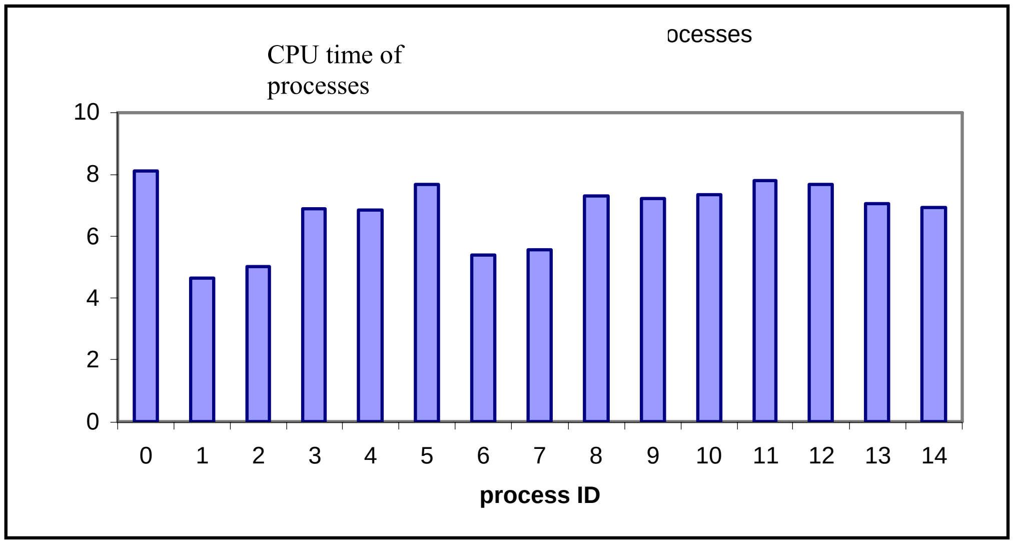


Figure 18 Graph showing load balance on processes with parmd.f

5.5 Application of Newton's Third Law

Applying Newton's Third Law (NTL) in calculating force terms on atoms halved the computation costs but increased the communication costs by a factor of number of atoms involved. To optimize communication costs, application of NTL was skipped in the parallel program. The impact of this choice was tested in 2 cases. (1) A system with 4 processes by varying the number of atoms. (2) A system of 20,000 atoms by varying the number of atoms. It was observed that application of NTL was advantageous for smaller systems with less than 8000 atoms, for which, parallel and serial programs, gave similar results. For larger systems with number of atoms greater than 20,000 and with number of processes greater than 4, parallel program gave better results. Figure 19 shows the scaling of parallel program with 4 processes, serial program without NTL and serial program using NTL, with number atoms. It can be clearly observed that up to 20,000 atoms, and 4 processes, curves representing parallel and serial program using NTL almost overlap. Figure 20 was generated by subtracting the mean of cpu_time recorded by the serial program using NTL from the means of cpu_time recorded by the parallel program and serial program without NTL, for a system with 20,000 water atoms with different number of processes. The graph clearly shows the advantage of using parallel program with number of processes greater than 4.

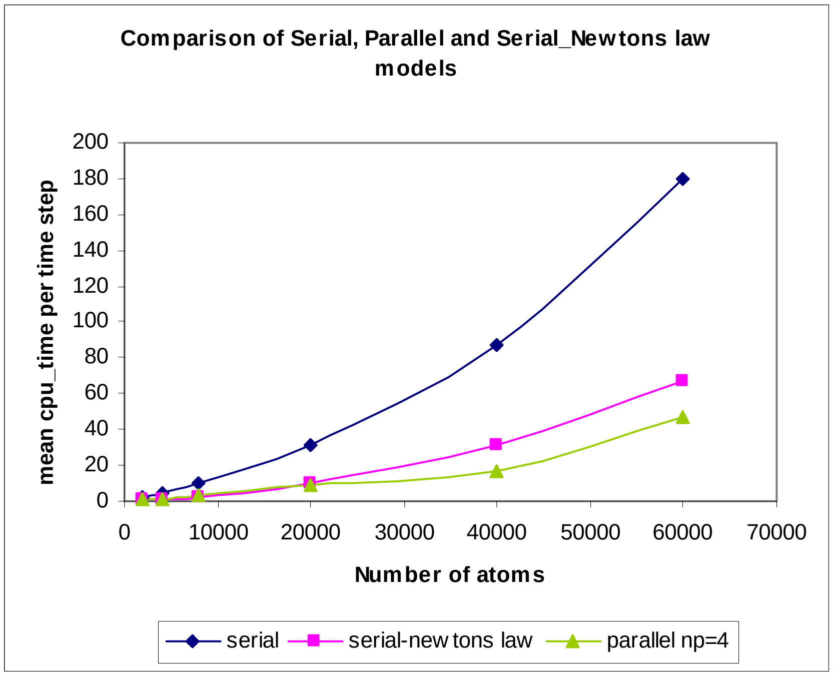


Figure 19 Scaling potch.f, parmd.f and newton.f with number of atoms

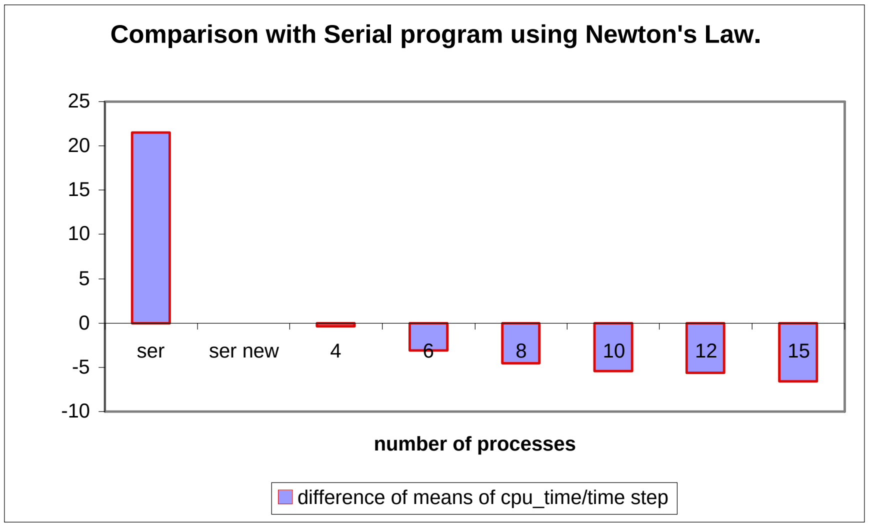


Figure 20 Differential scaling of potch.f, parmd.f with respect to newton.f with number of processes

5.6 Simulation of KcsA

The prepared system of KcsA with 60,000 water atoms and 5200 protein atoms was simulated for 1000 time steps using 8 processes. Positions of the first 100 water atoms were printed out each time step. The positions of water atoms before simulation, after 200 steps and after 500 steps are shown in Figure 22. The total time for simulation was 4999.135 seconds. Figure 21 shows the graph of CPU_time in seconds per time/step. A sharp peak was observed at 635, 637 steps that took around 20 seconds (CPU_time) to complete.

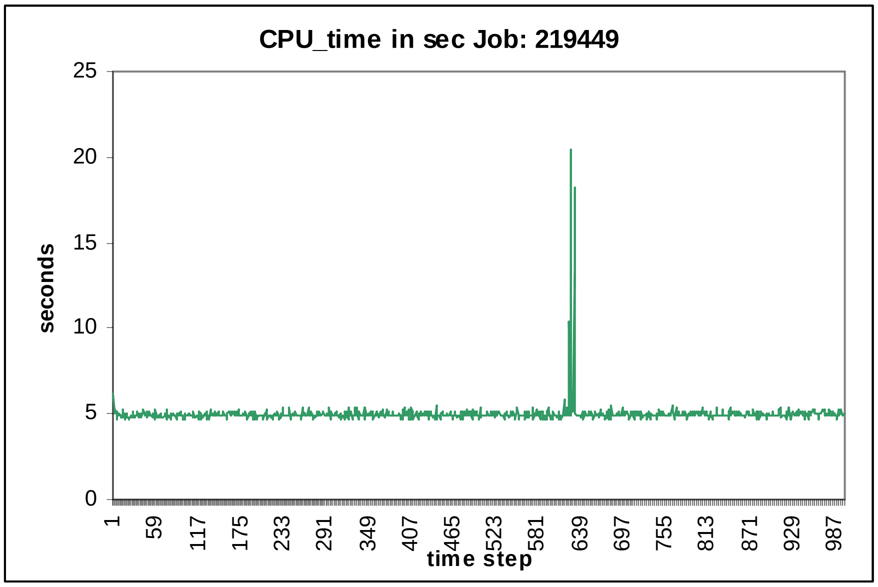


Figure 21 CPU-time/step versus time step for simulation of KcsA

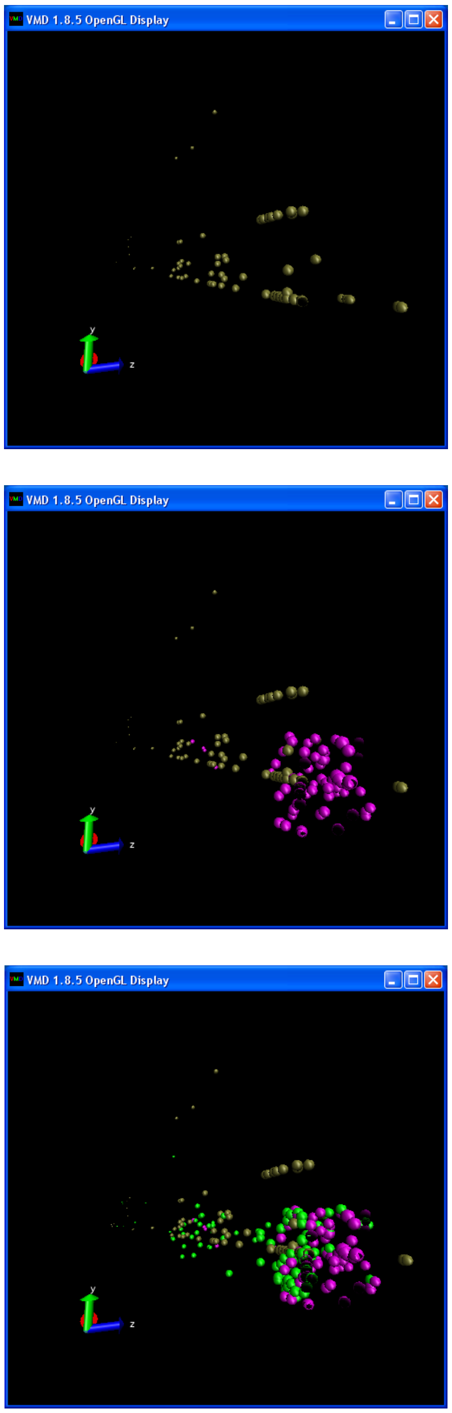


Figure 22 Simulation of KcsA, 100 atoms, after 200, 500 steps.

5.7 Portability

Although the program was developed on a Linux cluster, the code can be used on other platforms, which have either a Fortran 77 or 90 compiler and mpich-1.2.7 library. The path of the mpi.h in the ‘include’ statement, ‘include ‘usr/global/mpich-1.2.7/include/mpif.h’, should be correctly given when using the code on another cluster. Other versions of MPI such as open mpi might require changes in the syntax of MPI functions.

5.8 Interaction with VMD

Output files generated using the parallel program can be saved as text files and opened as excel sheets to change the format to PDB format. Microsoft excel sheets can be reopened as text files in the changed format, which in turn can be opened in VMD editor for viewing the structure of the system.

CHAPTER 6 Future work

A parallel molecular dynamics program was developed for simulation of water atoms inside potassium channel KcsA. The program has verlet integration scheme of MD, SPC (simple point charge) model of treatment of water, reaction field theory for calculation of long range electrostatic forces, scaling of velocities based on system temperature, a hybrid AD algorithm for parallel force computations on water atoms and message driven communication between processes.

The program can be taken to its next level by adopting the following points,

- The intra-molecular force calculations on proteins have to be introduced into the code for introducing flexibility of protein atoms. Flexibility of proteins is particularly important for simulating the gating mechanism of KcsA. An additional loop for calculating the intra-molecular force components on protein atoms, can be included within the loop for calculating force terms on water atoms
- Newton's law could be applied to the force calculations in parallel program to observe the changes in communication it would cause. This is particularly important for larger physical systems (greater than 60,000), after which computations increase exponentially.

- Applets for thermodynamic free energy calculations and data analysis of the updated positions and forces like RMSD plots and Raman plots can be introduced.
- Object oriented programming with C++, would be another interesting change that could be made that might improve the speed of the existing parallel program. The positions, velocities, total force and other properties of a particular atom can be declared as a class. This would significantly decrease the number of communication calls in the parallel MD program.
- The code can be converted to GUI (graphical user interface) software, which would take the simulation parameters and PDB file as input and generate pictorial out put of the dynamics of the system.

REFERENCES

1. Alder, B., T. Wainwright (1957). "Phase transition for a hard sphere system." The Journal of Chemical Physics 27(5): 1208-1209.

2. Aqvist, J., (2001). "K+/Na+ selectivity of the KcsA potassium channel from micrpscopic free energy pertubation calculations." Biochimica et Biophysica Acta (BBA) - Protein Structure and Molecular Enzymology 1548: 2.

3. Brooks, B.R. (1983). "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations" J. Comp. Chem 4(2): 187-217

4. Brown, D., H. Minoux, (1997). "A domain decomposition parallel processing algorithm for molecular dynamics simulations of systems." Computer physics communications 103(2-3): 170-186.

5. Cornell, D. (1995). "A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules", J. Am. Chem. Soc. 117: 5179-5197.

6. Gropp, W., (1994). " Using MPI, Portable parallel programming with message passing

interface", Scientific and Engineering Computation, 1994.

7. Humphrey, W., (1996). "VMD - Visual Molecular Dynamics", J. Molec. Graphics. 14: 33-38.

8. Izaguirre, J., T. Slabach, (2004). "ProtoMol, an object-oriented framework for prototyping novel algorithms for molecular dynamics." ACM transactions on mathematical software 30(3): 237-242.

9. Ponder, T.W., D.A. Case, (2003). "Force fields for protein simulations". J. Adv. Prot. Chem. 66, 27-85.

10. Kale, L., A. Gursoy, (1996). "NAMD: a parallel, Object-Oriented Molecular Dynamics Program." Journal of High performance computing applications 10(4): 251-268.

11. Karplus, M., J. M. Cammon, (2002). "Molecular Dynamics simulations of biomolecules." Nature structural biology 9(9): 646-649.

12. MacKinnon, R. (1998). "The structure of potsassium channel; molecular basis of K+ conduction and selectivity." Science 280(5360): 69-74.

13. Murty, R., D. Okunbor (1999). "Efficient parallel algorithms for molecular dynamics

simulations." Parallel Computing 25(3): 217-230.

14. N Hamid, P., Coddington, (2007). "Averages, distributions and scalability of MPI communication time for Ethernet and Myrinet networks." Proceedings of Parallel and distributed computing and networks 551.

15. Noskov, S., Yu (2007). "Importance of Hydration and Dynamics on the Selectivity of the KcsA and NaK Channels", J. General Physiology. 129: No .2: 135-143.

16. Plimpton, S. (1995). "Fast Parallel algorithms for short range molecular dynamics." Journal of Computational Physics 117(1): 1-7.

17. Refson, K. (2000). "Moldy: a portable molecular dynamics simulation program for serial and parallel computers." Computer physics communications 126(3): 130-135.

18. Wang, W., J. C.Phillips, (2005). "Scalable molecular dynamics with NAMD." Journal of Computational Chemistry 26: 1781-1802.

19. Ying, Z., (2003). "Nonequilibrium, Multiple-Timescale Simulations of Ligand-Receptor Interactions in Structured Protein Systems", J. PROTEINS. 52: 339-348.

20. Zhestkov, Y., B. Fitch, (2003). "Blue Matter, an application framework for molecular

simulation on Blue Gene." Journal of parallel and distributed computing 63(7): 759-764.

APPENDIX A

Submission script.sh

```
#!/bin/bash
######################################################################
######
#  execute script in current directory
#$ -cwd
#  want any .e/.o stuff to show up here too
#$ -e ./
#$ -o ./
#  shell for qsub to use:
#$ -S /bin/bash
#  name for the job; used by qstat
#$ -N laxmi_test
# number of processes is given ni the next line
#$ -pe mpich 8
#$ -V
######################################################################
######

WD="/usr/global/mpich-1.2.7"
export MPIRUN="/usr/global/mpich-1.2.7/bin/mpirun"
#export ssh=rsh
echo "MPIRUN == $MPIRUN..."
echo "----------------------------------------------------------------
--"
echo "sub_test.sh:  My user name is `whoami`..."
echo "sub_test.sh:  I'm on `hostname`.............."
echo "current directory is `pwd`..."
echo "sub_test.sh:  Beginning @ `date`..."
echo "sub_test.sh:  TMPDIR == $TMPDIR..."
echo "P4_RSHCOMMAND == $P4_RSHCOMMAND..."

if [ -e $TMPDIR/machines ]; then
      echo "It's alive...  ALIVE!!!!"
      echo "mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm"
      cat $TMPDIR/machines
      echo "mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm"
      echo "cp-ing machines file to ./"
      cp $TMPDIR/machines .
      echo "./machines == "
      cat ./machines
      echo "............"
else
      echo "OHNOZE..."
fi

echo "Running job with $WD/bin/mpirun:"
echo "Doing:"
echo "    $WD/bin/mpirun \\"
echo "        -np $NSLOTS \\"
echo "        -machinefile ./machines \\"
echo "        /home/mullapudil/RECENT/dorodie/temp3"

# the command mpirun to run the file temp4 in parallel mode
# WD : path of the command
# $NSLOTS (number of processes is taken from pe mpich 8

$WD/bin/mpirun \
 -np $NSLOTS \
 -machinefile ./machines \
 /home/mullapudil/RECENT/doordie/temp4

echo "sub_test.sh:  Ending @ `date`..."
echo "----------------------------------------------------------------------
--"
######################################################################
######
```

APPENDIX B

Communication test newcomtest.f

```
         program communication
c to test the communication of the main MD program
c Forces are calculated in parallel,
c particle positions are clculated along with forces
c individual processes send atom ID's
c root receives them
         integer k,ierr, myid, size,sz,temp3,j,iam
         integer stats(MPI_STATUS_SIZE),m,imax,req(5),i
         real temp2
         real  tempx(100000),tempy(100000)
         real tempz(100000)
         integer temps(100000),temp1(100000),counter
         real temp1x(100000), temp1y(100000),temp1z(10000)
         real x(100000),y(100000),z(100000)
         real fx(100000),fy(100000),fz(100000)
         include '/usr/global/mpich-1.2.7/include/mpif.h'
c--------------------------------------------------------
c mpi functions, init,rank,size are declared
c--------------------------------------------------------
         call mpi_init(ierr)
         call mpi_comm_rank(MPI_COMM_WORLD,myid,ierr)
         call mpi_comm_size(MPI_COMM_WORLD,size,ierr)
         print*,'I am in parallel world'
c--------------------------------------------------------
c parameters m=no 0f particles,x,y,z,co ordinates of particles
c temps carries ID of particles, tempxyz,txyz carry temporary
c co ordinates,k is a counter used
c--------------------------------------------------------
           m=2000
           sz=m
           temps=0.0
           tempx=0.0
           tempy=0.0
           tempz=0.0
           temp1=0.0
           temp1x=0.0
           temp1y=0.0
           temp1z=0.0
         do 100 i=1,m
           x(i)=120.0
           y(i)=120.0
           z(i)=120.0
 100     continue
c
c
          do 1 iclock=1,20
           counter=1
c parallel
           do 2  k=myid,m,size
           x(k)=x(k)/2.0
           y(k)=y(k)/2.0
           z(k)=z(k)/2.0
           temps(counter) = k
           tempx(counter) = x(k)
           tempy(counter) = y(k)
           tempz(counter) = z(k)
           counter=counter+1
           write(7,99)iclock,k,x(k),y(k),z(k)
 99        format(i5,1x,i5,1x,5f8.3,1x,5f8.3,1x,5f8.3)
 2         continue
c-------------------------------------------------------
c all processes send temps to the root
c-------------------------------------------------------
           call MPI_BARRIER(MPI_COMM_WORLD,ierr)
           if(myid.ne.0) then
           call MPI_SEND(temps(1),m,MPI_INTEGER,0,0,
     &      MPI_COMM_WORLD,ierr)
           call MPI_SEND(tempx(1),m,MPI_REAL,0,1,
     &      MPI_COMM_WORLD,ierr)
           call MPI_SEND(tempy(1),m,MPI_REAL,0,2,
     &      MPI_COMM_WORLD,ierr)
           call MPI_SEND(tempz(1),m,MPI_REAL,0,3,
     &      MPI_COMM_WORLD,ierr)
           end if
c-------------------------------------------------------
c root receives temps from processes and stores it in temp1
c---------------------------------------------------------
           if(myid.eq.0) then
           k=1
           do 3 j=1,size-1
           call MPI_RECV(temp1(k),sz,MPI_INTEGER,MPI_ANY_SOURCE,0,
     &      MPI_COMM_WORLD,stats,ierr)
           iam=stats(MPI_SOURCE)
           call MPI_RECV(temp1x(k),sz,MPI_REAL,iam,1,
     &      MPI_COMM_WORLD,stats,ierr)
           call MPI_RECV(temp1y(k),sz,MPI_REAL,iam,2,
     &      MPI_COMM_WORLD,stats,ierr)
           call MPI_RECV(temp1z(k),sz,MPI_REAL,iam,3,
     &      MPI_COMM_WORLD,stats,ierr)
           do 4 i=k,k+sz
           temp3=temp1(i)
           if(temp3.ne.0) then
           x(temp3)=temp1x(i)
           y(temp3)=temp1y(i)
           z(temp3)=temp1z(i)
           write(8,99)iclock,temp1(i),temp1x(i),temp1y(i),temp1z(i)
           end if
 4         continue
            k=k+sz
 3         continue
          end if
          call MPI_Barrier(MPI_COMM_WORLD,ierr)
c broadcast positions
          call MPI_Bcast(x,m,MPI_REAL,0,MPI_COMM_WORLD,ierr)
          call MPI_Bcast(y,m,MPI_REAL,0,MPI_COMM_WORLD,ierr)
          call MPI_Bcast(z,m,MPI_REAL,0,MPI_COMM_WORLD,ierr)
 1         continue
          call mpi_finalize(ierr)
          print*,'OUT OF PARALLEL WORLD WITH',ierr,'ERROR'
          stop
          end
c
```