Computing Platforms for Parallel Molecular Dynamics

A dissertation submitted to the SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH for the degree of Doctor of Technical Sciences presented by Thomas Peter Gössi Dipl. El.-Ing. ETH born 1st January 1971 citizen of Küsnacht am Rigi SZ accepted by recommendation of Prof. Dr. Gerhard Tröster, examiner Prof. Dr. Lothar Thiele, co-examiner 2000
Acknowledgments

The work presented in this thesis was carried out within the ETH-Polyproject: Parallel Computing in Molecular Dynamics Simulation, which was a joint venture of the Laboratory of Physical Chemistry, the Institute of Scientific Computing, the Computer Engineering and Networks Laboratory and the Electronics Laboratory, all at ETH Zurich.

First I would like to express my gratitude to my research advisor, Prof. Dr. Gerhard Tröster, head of the Electronics Laboratory for his encouragement and the assistance, he gave me throughout this project and who allowed me to perform this thesis at his laboratory with its excellent infrastructure.

I also want to thank my co-examiner, Prof. Dr. Lothar Thiele for carefully reading the final draft of my dissertation and for his suggestions to improve my work.

My special thanks go to Markus Thaler, who accompanied me in the project for a while, for his constructive proposals and for many fruitful discussions.

Further I’m indebted to the colleagues in the project group for their contributions and the discussions we had, that led me to new ideas. Especially I would mention my colleague Martin Gerber, who programmed most of the software, for his support in my investigations, his useful suggestions and for the pleasurable collaboration. In particular, I’m very grateful to Heiko Schäfer, who helped me in comprehending the chemical matters, for his endurance in answering lots of my questions and finally for the meticulous reviewing of the chemical topics in my thesis.

Special thanks to all people, who enriched my life at IFE, especially to be mentioned Rolf Enzler, Didier Cottet, Etienne Hirt and Geert Bernaerts staging many amusing times.

Finally I wish to express many thanks to Marianne Rogger for the careful proof-reading of this text and the correction of my English.

In acknowledgment of my sincere gratitude I devote this work to my parents and my sister for their continuous benefit and love during my studies at ETH.

Zurich, in September 2000

Thomas Peter Gössi
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kurzfassung</td>
<td>vii</td>
</tr>
<tr>
<td>Abstract</td>
<td>ix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Fundamentals of Atomic Many-Body Systems</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Molecular Dynamics</td>
<td>8</td>
</tr>
<tr>
<td>1.2.1 Introduction</td>
<td>8</td>
</tr>
<tr>
<td>1.2.2 Bonded and Non-bonded Interactions</td>
<td>10</td>
</tr>
<tr>
<td>1.2.2.1 Bonded Forces and Energies</td>
<td>10</td>
</tr>
<tr>
<td>1.2.2.2 Non-bonded Forces and Energies</td>
<td>15</td>
</tr>
<tr>
<td>1.2.3 Long Range Interactions and Cutoff</td>
<td>16</td>
</tr>
<tr>
<td>1.2.3.1 Twin Range Method</td>
<td>17</td>
</tr>
<tr>
<td>1.2.3.2 P³M Method</td>
<td>18</td>
</tr>
<tr>
<td>1.2.3.3 Ewald Method</td>
<td>19</td>
</tr>
<tr>
<td>1.2.4 Treatment of Boundaries</td>
<td>20</td>
</tr>
<tr>
<td>1.2.4.1 Vacuum Boundary Condition</td>
<td>20</td>
</tr>
<tr>
<td>1.2.4.2 Periodic Boundary Conditions</td>
<td>21</td>
</tr>
<tr>
<td>1.2.4.3 Extended Wall Region Boundary Condition</td>
<td>23</td>
</tr>
<tr>
<td>1.2.5 Simulation Methods</td>
<td>24</td>
</tr>
<tr>
<td>1.2.5.1 Systematic Search methods</td>
<td>25</td>
</tr>
<tr>
<td>1.2.5.2 Monte Carlo Simulation</td>
<td>25</td>
</tr>
</tbody>
</table>
2 Methodology

2.1 Computer Simulation of Molecular Systems ........................................ 31
  2.1.1 Modelling ............................................. 31
  2.1.2 Neighbor List Techniques ............................................. 33
    2.1.2.1 Brute Force Algorithm ............................................. 34
    2.1.2.2 Cell Index Method ............................................. 34
    2.1.2.3 Grid Cell Method ............................................. 37
  2.1.3 Parallelising ............................................. 39
    2.1.3.1 Decomposition Methods ............................................. 39
    2.1.3.2 Data Distribution ............................................. 40
    2.1.3.3 Techniques for Parallelising MD Algorithms ............................................. 41
  2.2 The GROMOS Package ............................................. 42
    2.2.1 Introduction ............................................. 42
    2.2.2 Simulation Methods in GROMOS ............................................. 43
    2.2.3 Algorithms in GROMOS ............................................. 45
      2.2.3.1 The Distance Algorithm ............................................. 46
      2.2.3.2 The Pairlist Algorithm ............................................. 48
      2.2.3.3 The Non-bonded Solvent-Solvent Interaction Algorithm ............................................. 51
  2.2.4 Parallel Versions of GROMOS ............................................. 52
    2.2.4.1 UHGROMOS ............................................. 52
    2.2.4.2 EULERGROMOS ............................................. 53
    2.2.4.3 GROMOS96P ............................................. 53
  2.3 GROMOS Analysis ............................................. 57
    2.3.1 Benchmark Topology ............................................. 57
    2.3.2 Profiling ............................................. 59
    2.3.3 Function Modelling ............................................. 62
      2.3.3.1 Distance Calculation ............................................. 62
      2.3.3.2 Pairlist Generation ............................................. 64
      2.3.3.3 Non-bonded Solvent-Solvent Interaction Calculation ............................................. 65
      2.3.3.4 Floating-point Operations for the Thrombin Simulation ............................................. 66
  2.4 Concept for Enhancing the Performance of GROMOS ............................................. 70
  2.5 Conclusion ............................................. 74
3 Hardware Approaches

3.1 General Overview of Hardware Solutions .......................... 77
3.2 Existing MD Hardware .............................................. 78
  3.2.1 The GRAPE Family ........................................... 79
    3.2.1.1 Introduction ......................................... 79
    3.2.1.2 GRAPE-1 and GRAPE-1A .............................. 82
    3.2.1.3 GRAPE-3 and GRAPE-3A .............................. 83
    3.2.1.4 GRAPE-5 ............................................. 85
    3.2.1.5 GRAPE-2 and GRAPE-2A .............................. 90
    3.2.1.6 GRAPE-4 and GRAPE-4A .............................. 92
    3.2.1.7 GRAPE-6 ............................................. 99
    3.2.1.8 MD-GRAPE ........................................... 101
    3.2.1.9 PROGRAPE-1 ........................................ 104
  3.2.2 GROMACS .................................................... 106
  3.2.3 MD-Engine .................................................. 107
  3.2.4 Conclusions ................................................. 110
3.3 MD on Massively Parallel Computers ................................. 112
3.4 MD on Dedicated Hardware ........................................... 115
3.5 Digital Signal Processors for MD .................................. 121
  3.5.1 ADSP-2106x SHARC ........................................ 122
  3.5.2 Benchmarks on the ADSP-2106x DSP .......................... 124
    3.5.2.1 Method ............................................. 124
    3.5.2.2 Distance Routine ................................... 126
    3.5.2.3 Force Routine ...................................... 131
    3.5.2.4 Conclusions ....................................... 135
3.6 Microprocessor Units for MD ....................................... 139
3.7 Summary ......................................................... 143

4 Communication .......................................................... 145

4.1 Parallelism ........................................................... 147
4.2 Topologies for the Parallel Processor System ....................... 148
  4.2.1 Bus Topology .............................................. 150
  4.2.2 2D-Net ..................................................... 151
  4.2.3 Hyper Cube ............................................... 155
  4.2.4 Recursive Topology ...................................... 157
4.2.5 Ring .................................................. 160
4.2.6 Other Topologies ................................. 162
4.2.7 Comparison ........................................ 162
4.3 Network Choice .................................... 166
  4.3.1 Ethernet ......................................... 166
  4.3.2 ATM Network .................................. 168
  4.3.3 Myrinet ......................................... 169
  4.3.4 ServerNet ....................................... 170
  4.3.5 Memory Channel ............................... 171
  4.3.6 Backplane ...................................... 173
  4.3.7 Conclusions .................................... 173

5 Cluster Architecture ................................ 175
  5.1 Hardware Overview ............................... 177
  5.2 Communication Network .......................... 180

6 Results ................................................. 185
  6.1 Sequential and Parallel GROMOS96 Version .... 187
  6.2 Influence of the Network ......................... 189
  6.3 Synchronization Methods .......................... 194
    6.3.1 Homogeneous Cluster ......................... 194
      6.3.1.1 Communication over Ethernet ............ 194
      6.3.1.2 Communication over Myrinet using GM .... 197
    6.3.2 Heterogeneous Cluster ....................... 199
      6.3.2.1 Communication over Myrinet using TCP/IP . 200
    6.3.3 Summary ........................................ 202
  6.4 Performance Measurements ......................... 202
    6.4.1 Homogeneous Cluster .......................... 202
    6.4.2 Heterogeneous Cluster ....................... 204
    6.4.3 Summary ........................................ 206
  6.5 Load Balancing ...................................... 206
    6.5.1 Homogeneous Cluster ......................... 207
    6.5.2 Heterogeneous Cluster ....................... 209
    6.5.3 Summary ........................................ 210
Kurzfassung


Molekulardynamiksimulationen sind jedoch sehr rechenintensiv und trotz Verfügbarkeit schnellerer Workstations müssen ziemlich lange Rechenzeiten in Kauf genommen werden. Aus diesem Grund wurde ein interdisziplinäres ETH-Polyprojekt gestartet, mit dem Ziel, den bestehenden MD Algorithmus zu verbessern und eine Hardware zu suchen, welche die Geschwindigkeit der Simulation mindestens um den Faktor 10 erhöht.

Die vorliegende Arbeit beschäftigt sich mit der Evaluierung einer geeigneten Hardware, um die Rechengeschwindigkeit von MD Simulationen zu erhöhen. Dabei wird eine neue Methodik angewendet, mit der die Rechenleistung verfügbareer Hardwarelösungen abgeschätzt wird. Die Untersuchungen basieren auf dem GROMOS™ Softwarepaket, welches unter der Leitung von Prof. W.F. van Gunsteren entwickelt wurde und das zur Simulation von Biomolekularsystemen dient.

Wie eine Analyse zeigt, bestehen die zeitintensivsten Teile einer MD Simulation aus einfachen Algorithmen, welche in kurzen überblickbaren Codesegmenten programmiert sind. Die effektivste Möglichkeit, die Performance einer MD Berechnung zu erhöhen, bietet deshalb eine Portierung dieser Teile auf einen geeigneten

Molecular systems are characterized as systems of thousands of particles, that interact with each other. The methods of Molecular Dynamics (MD) treats the characteristics of these particles in chemical processes, where the properties of molecules, liquids and solids are investigated. As chemical processes are complex, computer simulations are often used to solve problems in Molecular Dynamics. They offer a better interpretation of the experimental results, semi quantitative estimations and the capability to extrapolate experimental data into regions, that are difficult to access in the laboratory. Thus computational chemistry has become a field of increasing importance in the last decades.

However, Molecular Dynamics simulations are very computational intensive and their computing time is still very high, although workstations with a higher performance become currently available. Therefore an interdisciplinary ETH-Poly-project was started to improve the MD algorithm and to find a hardware solution for accelerating the simulation at least for a factor 10.

The subject of this thesis is an evaluation of a dedicated hardware for enhancing the performance of MD simulations. For this purpose a new method is used to estimate the computing power of available hardware solutions. The investigations are based on the GROMOS™ software package, used for the simulation of biomolecular systems, which was developed by Prof. W.F. van Gunsterens research group.

An analysis of an MD simulation shows, that the most time consuming parts are simple algorithms, programmed in short code segments. Outsourcing only these parts to a dedicated coprocessor hardware, which is attached to a standard workstation, offers the most effective way to enhance the performance for computing an MD problem. The following approaches of different hardware solutions pointed out, that a workstation cluster offers not only a fast and efficient solution, but convinces also of its flexibility, the low cost and it is future-proof. Performance meas-
urements on the workstation cluster with different networks show the first time, that the speed of the network has only a low impact on the runtimes of parallel MD simulations. Therefore the share of the communication time lies at 2% on a cluster with 4 processors. On an Alpha workstation cluster with 4 processors a speedup of 3.5 can be achieved. Due to the technological advances during the three years of the project, the computing performance has been enhanced by 4.5 times. Consequently we achieved a total performance gain of a factor 15.

Comparisons of the results with that on networked workstations in an office environment and that on a massively parallel processor system show, that a workstation cluster with 8 - 16 processors achieves the same performance as a parallel processor system. A cluster of this size offers therefore a good alternative for a parallel computer. Further the comparisons indicate, that workstations, connected to a network in a business environment, should achieve the computing power of a parallel system in the future.
This chapter gives an overview over the basic idea of chemical N-body problems and molecular dynamics calculations. In the first part the fundamentals of atomic many-body systems are explained and how their states can be determined by using classical and quantum mechanical methods. The further parts treat the various aspects of molecular dynamics in biomolecular systems, which describes the force field and the handling of long range interactions. A description of the treatment of different boundary conditions follows and the chapter finishes with an overview of different simulation methods for molecular dynamics.
1.1 Fundamentals of Atomic Many-Body Systems

A many-body system, also called N-body system, consists of N particles, from which the positions and the velocities have to be determined. The states of such systems can be either obtained by using classical physics or with the help of the quantum mechanics.

In classical physics, the particles are treated as point masses, which interact with each other through the gravitational force [1]. The motions of point masses can be evaluated by solving the Newton's equation of motion

$$ F_i(t) = m_i \cdot \frac{d^2}{dt^2} \mathbf{r}_i(t) = -\frac{\partial}{\partial \mathbf{r}_i} V(\mathbf{r}), $$

(1.1)

and Newton’s gravitational law

$$ F_i = -Gm_i \sum_{j}^{N} \frac{m_j \mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^3}. $$

(1.2)

Equations 1.1 and 1.2 are mainly used for macroscopic N-body systems, such as planetary- and astro-structures, in which the current status of each particle can exactly be determined. In microscopic and atomic N-body systems, it is not possible to treat the state exactly by methods of the classical physics, because the uncertainty principle of W. Heisenberg (1901-1976) says, that certain pairs of physical terms cannot be measured exactly at the same time [8]. Thus we have the problem, that the motions of the particles can only be determined with statistical probability densities and can be described more exactly with quantum mechanical methods.

The quantum mechanics operates with a probability density of identical elementary particles. The status of the particles is defined by its statistical entirety with characteristics, that are taken from the classical physics, such as coordinates, impulse, momentum, energies etc.. These characteristics are defined as operators in the quantum mechanics, which acts on complex, time- and locus-dependent wave functions $\Psi$. The wave functions $\Psi$ describe the statistical totality of a system. Thus the probability $w$, that a particle resides in a certain volume $dV$ can be written as
\[ w = |\Psi(r, t)|^2 dV. \tag{1.3} \]

The definition of a quantity in the classical physics is also valid in quantum mechanics if it has no derivations, e.g. \( \hat{r} \) is the same as \( r \) [8]. In equations the sign "\(^\wedge\)" is used for quantum mechanical operators.

The following example shows, how the quantum momentum of the impulse is deduced. In the classical physics the impulse of a mass point \( m \) moving with the velocity \( v \) is defined as \( p = mv \). The corresponding quantum momentum is given by

\[ \hat{p} = \frac{i\hbar}{2\pi} \cdot \nabla(r) = \frac{i\hbar}{2\pi} \left( \frac{\partial}{\partial x} \frac{\partial}{\partial y} \frac{\partial}{\partial z} \right). \tag{1.4} \]

To obtain all the information about a particle in a N-body system, which is consistent with the postulates of quantum mechanics, the time dependent Schrödinger equation

\[ \hat{H} \cdot \Psi(r_i, t) = i \cdot \frac{\hbar}{2\pi} \cdot \frac{\partial}{\partial t} \Psi(r_i, t) \tag{1.5} \]

has to be solved. Here \( \hat{H} \) is the quantum mechanical Hamiltonian-Operator, \( \Psi \) is the wave function, \( \hbar \) the Planck's constant and \( r_i = (x_i, y_i, z_i) \) is the position vector of particle \( i \), for \( i = (1, ..., N) \). The Hamiltonian-Operator \( \hat{H} \) represents the total energy of the system and is defined as

\[ \hat{H} = H(\hat{q}_1, \hat{q}_2, \hat{q}_3, ..., \hat{q}_N; \hat{p}_1, \hat{p}_2, \hat{p}_3, ..., \hat{p}_N, t) \tag{1.6} \]

where \( \hat{q}_i \) are the positions- and \( \hat{p}_i \) the impulse-coordinates of the N-body system.

Unfortunately equation 1.5 cannot be solved directly for most of the N-body systems because of the many degrees of freedom. Thus there are different approximations, which may be used to obtain a description of a particle [2]. Other approaches treat only a small subsystem with few degrees of freedom in a quantum dynamical sense, so that the Schrödinger equation can be solved directly, while the rest of the system is performed in a classical dynamical sense [3].

In an atomic system, the electrons are much lighter than the nuclei. Thus the electronic and the nuclear motion can be treated separately in most of the molecular structures. Electrons move much faster than nuclei and underlie an essential static potential due to the nuclei. On the other side, nuclei experience an average potential due to the electrons. This is the principle of the Born-Oppenheimer separation. The Schrödinger equation 1.5, which describes the motions of the electrons and nuclei
1.1 Fundamentals of Atomic Many-Body Systems

together, is separated into individual equations for the electrons and nuclei [4].
Using the Born-Oppenheimer separation, equation 1.5 can be written as

\[
[\hat{K}_n(r_n) + \hat{K}_e(r_e) + \hat{V}_{nn}(r_n) + \hat{V}_{ne}(r_n, r_e) + \hat{V}_{ee}(r_e)]\Psi(r_n, r_e) = E_{tot}\Psi(r_n, r_e),
\]

where \(\hat{K}_n(r_n)\) and \(\hat{K}_e(r_e)\) are the kinetic-energy operators for the nuclei at position \(r_n\) and for the electrons at position \(r_e\), which is relative to the centre of mass. The potential-energy terms \(\hat{V}_{nn}(r_n)\), \(\hat{V}_{ne}(r_n, r_e)\) and \(\hat{V}_{ee}(r_e)\) are the electrostatic potential energies of the nucleus-nucleus, nucleus-electron and electron-electron interactions.

If the motions of the nuclei and the electrons are separable, the total wave function \(\Psi(r_n, r_e)\) can be written as

\[
\Psi(r_n, r_e) = \Phi(r_n, r_e)\chi(r_n).
\]

The electronic wave function \(\Phi(r_n, r_e)\) depends parametrically on the nuclear configuration \(r_n\) and is the solution of the electronic Schrödinger equation

\[
[\hat{K}_e(r_n) + \hat{V}_{ne}(r_n, r_e) + \hat{V}_{ee}(r_e)]\Phi(r_n, r_e) = E(r_n)\Phi(r_n, r_e).
\]

The nuclear wave function \(\chi(r_n)\) is the solution of the nuclear Schrödinger equation

\[
[\hat{K}_n(r_n) + \hat{V}_{nn}(r_n) + E(r_n)]\chi(r_n) = E_{tot}\chi(r_n).
\]

In equation 1.10 the nuclear potential-energy \(\hat{V}_{nn}(r_n)\) and the electronic energy \(E(r_n)\), which is obtained by solving equation 1.9, can be combined to

\[
\hat{V}(r_n) = \hat{V}_{nn}(r_n) + E(r_n)
\]

which is the potential-energy function or potential-energy surface.

The separability of the total wave function into the product of an electronic wave function and a nuclear wave function is not exact but depends on the neglect of terms coupling electronic and nuclear motion. It can be shown, that in the most cases these terms are negligible.

In addition to the approximation, in which the motion of the electrons have been averaged out, we can make the additional approximation, that a classical description is adequate. Using Schrödingers equation 1.5 in combination with the equations 1.10 and 1.11, the Hamiltonian-Operator \(\hat{H}\) can be written as
\( \hat{H}(\hat{q}, \hat{p}) = \hat{K}_n(\hat{p}_1, \ldots, \hat{p}_N) + \hat{V}(\hat{q}_1, \ldots, \hat{q}_N), \) (1.12)

where \( \hat{q}_i \) are the coordinates and \( \hat{p}_i \) the momenta of each molecule \( i \). Often molecules are treated as rigid bodies. In this case \( \hat{q}_i \) are the coordinates of each molecular centre of mass together with a set of variables \( \Omega_i \), which specify the molecular orientation.

The kinetic energy \( \hat{K}_n \) is defined as

\[
\hat{K}_n = \sum_{i=1}^{N} \sum_{\alpha} \frac{p_{i\alpha}^2}{2m_i},
\]

(1.13)

where \( m_i \) is the molecular mass and the index \( \alpha \) runs over the different \( (x, y, z) \) components of the momentum \( p \) of the molecule \( i \).

The potential energy \( \hat{V} \) contains the information regarding intermolecular interactions. If the potential \( \hat{V} \) is fairly sensibly behaved, an equation of motion can be constructed from \( \hat{H} \), which determines the entire time evolution of the system and all its mechanical properties. As the potential \( \hat{V} \) is only dependent on the position vectors, it is not necessarily a quantum mechanical operator and is \( \hat{V} = V \). This allows to use a classical potential energy function and to solve atomic many body problems with the methods of classical physics. The result is a classical atomic model with moving point masses, where the trajectories of the particles can be determined using the potential function and Newton's equation of motion (eq. 1.1).

The classical potential energy function can be divided into terms depending on the coordinates of individual atoms, pairs, triplets etc.:

\[
V(r) = \sum_i v_1(r_i) + \sum_{i < j} v_2(r_i, r_j) + \sum_{i < j < k} v_3(r_i, r_j, r_k) + \ldots
\]

(1.14)

The first term \( v_1(r_i) \) represents the effect of an external field on the system and the remaining terms represent the particle interactions. The second term \( v_2(r_i, r_j) \) is the pair potential of the particles \( i \) and \( j \). If the potential is spherically symmetric, it depends only on the magnitude of the pair separation \( r_{ij} = |r_j - r_i| \) and can also be written as \( v_2(r_{ij}) \). The notation of the pair potential summation in equation 1.14 indicates that only all distinct particles \( i \) and \( j \) are considered without counting any pair twice. The third term \( v_3(r_i, r_j, r_k) \) involves triplets of molecules and is significant at liquid densities. Unfortunately the calculation of any quantity involving a sum over triplets of molecules is very time consuming on a computer. Thus an approximation by pairs is made by defining an effective pair potential \( v_2^{\text{eff}} \), which
includes the average three body effects and gives a good description of the liquid properties. The fourth and higher order terms are expected to be small in comparison with \( u_2 \) and \( u_3 \), and have practically no influence on the result of equation 1.14. Thus they can be omitted in computer simulations [5].

With the definition of an effective pair potential \( u_2^{\text{eff}} \) and the neglecting of the fourth and higher order terms, equation 1.14 can be rewritten as

\[
V(r) = \sum_i u_1(r_i) + \sum_{i<j} u_2^{\text{eff}}(r_{ij}).
\]

The effective pair potential \( u_2^{\text{eff}} \), which is needed to reproduce the experimental data, depends on the density, temperature etc., whereas the true two-body potential does not. The *Lennard-Jones 12-6 potential* is often used as an approximation to the effective pair potential:

\[
u_2^{\text{eff}} = u^{\text{LJ}}(r_{ij}) = 4\varepsilon \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right]
\]  

(1.16)

The parameters \( \varepsilon \) and \( \sigma \) are determined by experimental results. Values of \( \varepsilon/k_B \approx 120K \), where \( k_B \) is the Boltzmann's constant and \( \sigma \approx 0.34\text{nm} \) provide a reasonable agreement with the experimental properties of liquid argon. The potential has a long-range attractive tail of the form \(-1/r^6\), a negative well of depth \( \varepsilon \) and a steeply rising repulsive wall at distances less than \( r - \sigma \) [5]. It is a simple, more idealized pair potential, which is commonly used in computer simulations.

To illustrate the accuracy of the approximations, which we have described above, figure 1.1 shows the measured pair potential (solid line) and the effective pair potential (dashed line) of two Argon atoms [5]. The measured pair potential was derived by considering a large quantity of experimental data and the effective pair potential was obtained by using the Lennard-Jones potential in a computer simulation of liquid Argon with \( N \approx 2 \).

The potential is characteristic for intermolecular interactions. In short distances of 0.3 - 0.35nm, the steeply rising part indicates a strong repulsion to compression because of the non-bonded overlap between the electron clouds. The negative well is responsible for cohesion in condensed phases. There is an attractive tail at large separations due to correlation between the electron clouds surrounding the atoms. Here we have the van der Waals or London dispersion, which is proportional to \(-r^{-6}\). For charged particles, *Coulomb terms* would be present, which depend on a potential with \(-r^{-1}\) even at longer distances up to 2nm and more. Liquid argon is uncharged.
Figure 1.1: Normalized Argon pair potential: measured pair potential (solid line) and effective pair potential (dashed line) of two Argon atoms.

1.2 Molecular Dynamics

1.2.1 Introduction

The molecules in a Molecular Dynamics (MD) system have different atomic structures and are not all indistinguishable. In addition to intermolecular interactions, covalent (intramolecular) interactions have to be treated. Molecules are typically represented by atoms with orientation-dependent forces. Thus we have interactions between the atoms within a molecule, which must not be neglected. These interactions are the bonded forces, because they occur only within the molecule, where all atoms are bonded. The other interactions are the non-bonded forces, because they occur also between such atoms, which have no bonds. The molecules interact with each other only over the non-bonded interactions, which are van der Waals and Coulombic electrostatic interactions [6].
In biomolecular systems, the molecules are typically separated into two groups: the *solute-molecules*, which are often proteins and the *solvent-molecules*, mostly water or other liquids. A biomolecular system consists of many more solvent-molecules than solute-molecules. The structure of the solvent molecules is elementary in contrast to that of the solute-molecules. Thus the solvent molecules can be treated as rigid bodies. The non-bonded interactions occur between the atoms of all molecules of each group and we can distinguish between *solute-solute*, *solute-solvent* and *solvent-solvent* interactions [7].

**Figure 1.2: Simplified illustration of the interactions in a biomolecular simulation**

The range of the electrostatic interaction can be considerably reduced if atoms are packed together into *charge groups*, which have a zero net charge. Thus the sum of the \( 1/r \) monopole contributions of the various atom pairs to the group-group interaction will be zero and the electrostatic interaction between two groups of atoms is of dipolar character and proportional to \( 1/r^3 \). For groups of atoms with a total charge of \(+e\) or \(-e\), the partial atomic charges of the charge group may add up to a multiple of \( \pm e \). Using charge groups, the electrostatic interaction of the atoms with atoms of the other groups is either calculated for all atoms of the charge group or for none. [7].

The position of a charge group is defined differently for solute and solvent molecules. For a solute charge group, it is the centre of geometry and is determined with

\[ e = 1.6022 \cdot 10^{-19} \text{ Coulomb} \]
\[ r_{cg} = \frac{1}{N_{cg}} \sum_{i=1}^{N_{cg}} r_i, \]  

(1.17)

where \( N_{cg} \) is the number of the solute atoms, which belong to the charge group. A solvent molecule has mostly no atomic charges and can be defined as only one charge group. In this case, the position of the solvent charge group is taken by the position of the first, which is mostly the largest atom of a solvent molecule.

### 1.2.2 Bonded and Non-bonded Interactions

Equation 1.15 is used to calculate the interactions. As the different potentials are additive, it is preferable to treat the intermolecular and the covalent interactions as separate terms. In this way the potential is composed of bonded and non-bonded interactions

\[ V(r) = V_{bon}(r) + V_{nonb}(r). \]  

(1.18)

#### 1.2.2.1 Bonded Forces and Energies

The bonded interactions represent the forces and the energies between covalently bonded atoms within a molecule [7]. It is the sum of four different interaction terms and can be written as

\[ V_{bon}(r) = V_{bond}(r) + V_{angle}(r) + V_{trig}(r) + V_{har}(r). \]  

(1.19)

The four terms are the bond-stretching \( V_{bond} \), the bond-angle bending \( V_{angle} \), the harmonic improper dihedral-angle torsion \( V_{trig} \) and the trigonometric dihedral-angle torsion \( V_{har} \).

The covalent bond forces are the forces between two directly bonded atoms [6].

![Figure 1.3: Covalent bond forces](image-url)
The bond-stretching interaction is dependent of the bond length $b_n$ between the atoms $i$ and $j$ and can be calculated as

$$V_{bond}^b(r) = \sum_{n=1}^{N_b} \frac{1}{4} K_{b_n} \cdot \left( \frac{b_n^2}{b_0^2} \right)^2.$$

(1.20)

The summation runs over all $n = 1, ..., N_b$ covalent bonds in the molecular system, $K_{b_n}$ is the force constant and $b_0$ is the bond-length parameter of the bond with sequence number $n$. The forces on the atoms $i$ and $j$ are

$$F_i = -\frac{\partial V_{bond}^b}{\partial b_n^2} \frac{\partial b_n}{\partial r_i} = -K_{b_n} \cdot \left( \frac{b_n^2}{b_0^2} \right)^2 r_{ij}$$

(1.21)

and

$$F_j = -F_i$$

(1.22)

The covalent bond-angle forces are the forces caused by the bending of the bond angles between the atoms [4].

$$F_{angle}(r) = \sum_{n=1}^{N_\theta} \frac{1}{2} K_{\theta_n} \cdot \left( \cos \theta_n - \cos \theta_{0_n} \right)^2.$$

(1.23)
The summation runs over all \( n = 1, \ldots, N \) bond angles in the molecular system, \( K_{\theta_n} \) and \( \theta_{\theta_n} \) are the bond-angle type parameters of bond-angle sequence number \( n \). The forces on the atoms \( i, j \) and \( k \) can be calculated as

\[
F_i = -\frac{\partial V_{\text{angle}}}{\partial \cos \theta_n} \frac{\partial \cos \theta_n}{\partial r_i} = -K_{\theta_n} \cdot (\cos \theta_n - \cos \theta_{\theta_n}) \left( \frac{r_{kj}}{r_{kj}} - \frac{r_{ij}}{r_{ij}} \cos \theta_n \right) \frac{1}{r_{ij}},
\]

\[
F_k = -\frac{\partial V_{\text{angle}}}{\partial \cos \theta_n} \frac{\partial \cos \theta_n}{\partial r_k} = -K_{\theta_n} \cdot (\cos \theta_n - \cos \theta_{\theta_n}) \left( \frac{r_{ij}}{r_{ij}} - \frac{r_{kj}}{r_{kj}} \cos \theta_n \right) \frac{1}{r_{kj}},
\]

and

\[
F_j = -F_i - F_k.
\]

The dihedral-angle torsion forces are the forces caused by the angle between two bonds, which are stretched over a torsional bond [4].

![Figure 1.5: Dihedral-angle torsion forces](image)

The trigonometric dihedral angle torsion interaction arises due to the angle \( \varphi_n \) between the planes of the bonds \( i-j \) and \( k-l \), in which cross at the bond \( j-k \). It acts on the atoms \( i, j, k \) and \( l \) and is calculated as

\[
V^{\text{trig}}(r) = \sum_{n=1}^{N_{\varphi}} K_{\varphi_n} \cdot [1 + \cos(\delta_n) \cos(m_n \varphi_n)]
\]

(1.27)
The summation runs over a set of dihedral angles \( n = 1, ..., N \phi \) and the parameters \( K_{\phi_n}, m_n \) and \( \delta_n \) are the dihedral type code of the dihedral sequence number \( n \). The actual torsion dihedral angle is denoted by \( \phi_n \). The forces on the atoms \( i, j, k \) and \( l \) are calculated with

\[
F_i = -\frac{\partial V_{\text{trig}}}{\partial \cos \phi_n} \frac{\partial \cos \phi_n}{\partial \cos \phi_n} \frac{\partial \cos \phi_n}{\partial r_i} \\
= -K_{\phi_n} \cos \delta_n \frac{\partial \cos (m_n \phi_n)}{\partial \cos \phi_n} \left( \frac{r_{ln'}}{r_{im'}} - \frac{r_{ln'}}{r_{im'}} \cos \phi_n \right) \frac{1}{r_{im'}} ,
\]

\[
F_l = -\frac{\partial V_{\text{trig}}}{\partial \cos \phi_n} \frac{\partial \cos \phi_n}{\partial \cos \phi_n} \frac{\partial \cos \phi_n}{\partial r_l} \\
= -K_{\phi_n} \cos \delta_n \frac{\partial \cos (m_n \phi_n)}{\partial \cos \phi_n} \left( \frac{r_{im'}}{r_{ln'}} - \frac{r_{im'}}{r_{ln'}} \cos \phi_n \right) \frac{1}{r_{ln'}} ,
\]

\[
F_j = -\frac{\partial V_{\text{trig}}}{\partial \cos \phi_n} \frac{\partial \cos \phi_n}{\partial \cos \phi_n} \frac{\partial \cos \phi_n}{\partial r_j} = \left( \frac{(r_{ij} \cdot r_{kj})}{r_{kj}^2} - 1 \right) F_i - \frac{(r_{kl} \cdot r_{kj})}{r_{kj}^2} F_l
\]

\[
F_k = -F_i - F_j - F_l .
\]

In equations 1.28 and 1.29 we have the terms \( r_{ln'} \) and \( r_{im'} \) defined as

\[
r_{ln'} = -r_{kl} + \frac{(r_{kl} \cdot r_{kj})}{r_{kj}^2} r_{kj} ,
\]

\[
r_{im'} = r_{ij} - \frac{(r_{ij} \cdot r_{kj})}{r_{kj}^2} r_{kj} .
\]

The last term are the improper dihedral-angle forces. These forces keep the bonds and atoms into planar or tetrahedral geometry [7].
Chapter 1: Introduction

Figure 1.6: Improper dihedral-angle forces

The harmonic improper dihedral-angle bending interaction acts on all atoms $i$, $j$, $k$ and $l$ of a planar or tetrahedral atomic structure and is dependent of the angle $\xi_n$ between the two vectors $r_{mj}$ and $r_{nk}$, which are the normals to the planes, defined by the bonds $i$-$k$, $j$-$k$ and the bonds $j$-$k$, $k$-$l$ respectively. It can be written as

$$V^{har}(r) = \sum_{n=1}^{N_\xi} \frac{1}{2} K_{\xi_n} \cdot (\xi_n - \xi_{0,n})^2. \quad (1.33)$$

The summation runs over a set of dihedral angles $n = 1, \ldots , N_\xi$, which are selected to keep groups of atoms near a specified spatial configuration. $K_{\xi_n}$ and $\xi_{0,n}$ are parameters of the improper dihedral type code of the improper dihedral sequence number $n$. The forces on the atoms $i$, $j$, $k$ and $l$ are

$$F_i = -\frac{\partial V_n^{har}}{\partial \xi_n} \frac{\partial \xi_n}{\partial r_i} = -K_{\xi_n} \cdot (\xi_n - \xi_{0,n}) \frac{r_{kj}}{2} r_{mj}. \quad (1.34)$$

$$F_l = -\frac{\partial V_n^{har}}{\partial \xi_n} \frac{\partial \xi_n}{\partial r_l} = +K_{\xi_n} \cdot (\xi_n - \xi_{0,n}) \frac{r_{kj}}{2} r_{nk}. \quad (1.35)$$

$$F_j = -\frac{\partial V_n^{har}}{\partial \xi_n} \frac{\partial \xi_n}{\partial r_j} = \left(\frac{(r_{ij} \cdot r_{kj})}{r_{kj}^2} - 1\right) F_i - \frac{(r_{kl} \cdot r_{kj})}{r_{kj}^2} F_l \quad (1.36)$$

and

$$F_k = -F_i - F_j - F_l. \quad (1.37)$$
In equations 1.34 and 1.35 the terms $r_{mj}$ and $r_{nk}$ are defined as
\[ r_{mj} = r_{ij} \times r_{kj}, \]
\[ r_{nk} = r_{kj} \times r_{kl}. \]  

### 1.2.2.2 Non-bonded Forces and Energies

The non-bonded interactions are the forces and interaction energies between all atoms in a molecular system [5].

![Non-bonded interactions of two diatomic molecules](image)

**Figure 1.7: Non-bonded interactions of two diatomic molecules**

The van der Waals term is calculated with the Lennard Jones approximation
\[ V^{lj}(r) = \sum_{\text{nonbonded pairs}(i,j)} \left( \frac{C_{12}(i,j)}{r_{ij}^6} - \frac{C_6(i,j)}{r_{ij}^6} \right) \cdot \frac{1}{r_{ij}^6}, \]  

and the Coulomb term can be written as
\[ V^{crf}(r) = \sum_{\text{nonbonded pairs}(i,j)} \frac{q_i q_j}{4 \pi \varepsilon_0 \varepsilon_r r_{ij}}. \]
Equation 1.40 is obtained from equation 1.16 by defining the van der Waals parameters $C_6(i, j) = 4\varepsilon\sigma^6$ and $C_{12}(i, j) = 4\varepsilon\sigma^{12}$. They define the type of the van der Waals interaction and are dependent on the atom type of the atoms $i$ and $j$. The van der Waals interactions diminish rapidly with larger distances and can be neglected for long distances $r_{ij}$.

The Coulomb interactions are the electrostatic interactions between the atoms with the charges $q_i$ and $q_j$. They are inversely proportional to the distance $r_{ij}$ between the atoms $i$ and $j$. Neglecting the Coulomb interactions for long distances, would result in a large error and therefore all atom pairs have to be considered for the calculation, which would dramatically increase the simulation time. Therefore charge groups (see section 1.2.1 on page 9) are defined to make the electrostatic interactions as dipolar interactions, which have a $1/r_{ij}^3$ distance dependence. The relative dielectric permittivity $\varepsilon_r$ has a value ranging from 1 to 8 and the value $(4\pi\varepsilon_0)^{-1}$ is specified as 138.9354 kJmol$^{-1}$e$^{-2}$nm.

The summation runs in principle over all atom pairs $i, j$ in a molecular system and is proportional to $N^2$. Excluded are all first and second neighbours and all atoms, whose distance exceeds a range, where the interactions are so small, that they can be neglected [7]. First neighbours are all covalently bounded atoms and second neighbours are those atoms, which are covalently bounded to one common neighbour atom. The non-bonded interactions between such atoms are very large and would result in large forces on the bonds if they would be calculated with the equations 1.39, 1.40 and 1.41.

Due to the terms in equations 1.40 and 1.41, the non-bonded forces on atoms $i$ and $j$ can be written as

$$ F_i = \left( \frac{2C_{12}(i, j)}{r_{ij}^6} - C_6(i, j) \right) \cdot \frac{6r_{ij}}{r_{ij}^8} + \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_r} \cdot \frac{r_{ij}}{r_{ij}^3} \tag{1.42} $$

and

$$ F_j = -F_i \tag{1.43} $$

### 1.2.3 Long Range Interactions and Cutoff

As we have seen in section 1.2.2.2 the summation covering the non-bonded forces runs over all atom pairs of a molecular system and is therefore proportional to $N^2$, whereas the calculations of the bonded forces are proportional to $N$. The non-bonded van der Waals interactions are proportional to $1/r^6$ and that of the Coulomb
forces are proportional to $1/r^3$ if charge groups are used. Thus the non-bonded 
interactions can be neglected for larger distances and have not to be calculated [6].

Therefore a cutoff criterion can be applied for the non-bonded interactions and a 
list of neighbour atoms lying within the cutoff can be used. The cutoff radius $r_c$ has 
usually a value between 6Å and 9Å. However, the neighbor list is generated for 
each atom in a molecular system, but the cutoff criterion, which decides if an atom 
is enlisted into the neighbor list, depends on the distances between the charge 
groups, because the Coulomb forces have to be calculated either for all atoms of 
two charge groups or for none (see section 1.2.1). As the positions of the atoms are 
shifted only little in one simulation step, the neighbor list should only be updated 
every 5th-20th simulation step.

The contribution of far-away atoms, which are in the long range, is relatively 
small but not zero. To improve the accuracy of a MD simulation, the cutoff radius 
should be enlarged. But then, the effort for the calculation of the non-bonded inter¬
actions raises dramatically. Therefore other methods are used to treat the long range 
interactions.

1.2.3.1 Twin Range Method

In the twin range method two cutoff radii $r_{c1}$ and $r_{c2}$ are defined. The range 
between the two radii is defined as the twin range [6].

![Figure 1.8: Twin range method](image)
The atoms \( j \) lying within a distance \( r_{cj} \) from atom \( i \), are stored in the neighbor list of atom \( i \). The interactions between those atom pairs \((i, j)\) are calculated at every time step. The interactions of the atoms \( j \) for which the distance is \( r_{c1} < r_{ij} < r_{c2} \) to atom \( i \), are the long range interactions. The contribution of these interactions to the result is small and their values are relatively constant. Thus the long range interactions are updated simultaneously with the neighbor list every 5th-20th time step.

The twin range method is based on the assumption that the high frequency components of the long range force may be safely neglected. For example, the mean and low frequency field of the correlated peptide dipoles of the long \( \alpha \)-helix are accurately calculated and only the fast vibrations (\( \leq 0.2 \) ps) are neglected [6].

As not all atom pairs must be considered to calculate the interactions, the simulation time can be reduced by using a cutoff radius. The additional effort, which is required for generating the neighbor list, is smaller than calculating the interactions of all atom pairs, due to the neighbor list is not generated every time step and only the charge groups have to be incorporated into the neighbor list generation.

### 1.2.3.2 P\(^3\)M Method

The P\(^3\)M (particle-particle, particle-mesh) method combines the advantages of the PP and the PM methods.

In the PP (particle-particle) method, the interactions between the particles are calculated directly, based on the distances between the particle positions. It can only be used for small systems with long-range forces or for large systems, where the forces of the interactions are non-zero for only few interparticle distances.

The PM (particle-mesh) method approximates the charge density by assigning charges to a mesh of which the potential is calculated at each mesh-point. From the mesh-defined potential, the forces at the particle positions are interpolated. The method is computationally fast, but generally less accurate. It can only handle smoothly varying forces [1].

In the P\(^3\)M method, the force is divided into a rapidly varying short-range (PP) part and a slowly varying long-range (PM) force, which is periodic and can be calculated in the wavenumber space using the fast Fourier transform (FFT).

The PP force from a particle located at \( r_j \) can be calculated with

\[
F(r_i) = \frac{m(r_j - r_i)}{|r_j - r_i|^3} g_{PM}^3(R),
\]

(1.44)
where \( m \) is the mass of the particle and \( r_i \) is the position at which the force is calculated. The term \( g_{P^3M} \) is the cutoff function and expressed as

\[
g_{P^3M}(R) = \begin{cases} 
1 - \frac{1}{140} (224R^3 - 224R^5 + 70R^6 + 48R^7 - 21R^8) & \text{for } 0 \leq R < 1 \\
1 - \frac{1}{140} (12 - 224R^2 + 896R^3 - 840R^4 + 224R^5 + 70R^6 - 48R^7 + 7R^8) & \text{for } 1 \leq R < 2 \\
0 & \text{for } R \geq 2
\end{cases}
\]

where \( R = |r_j - r_i|/\eta \), and \( \eta \) is a scale length.

For the long-range part, the \( P^3M \) method uses the fast Fourier transform (FFT) to solve the Poisson equation

\[
\nabla^2 \psi(r) = -\varepsilon_0^{-1} \rho(r)
\]

for the potential \( \psi(r) \) and the charge density \( \rho(r) \). This second-order partial differential equation can be written in the wavenumber space as

\[
\tilde{\psi}(k) = (\varepsilon_0 k^2)^{-1} \tilde{\rho}(k)
\]

with position \( k \) in wavenumber space.

Due to the mesh assignment and the use of the fast Fourier transform, the forces are not so exact and include errors [10].

### 1.2.3.3 Ewald Method

The Ewald method is used in simulations of ionic systems. Similar to the \( P^3M \) method the forces are divided into a short-range and a long-range part. Thus the total Ewald-sum can be written as

\[
E = E^S + E^G.
\]

The Ewald-sum for the short-range part is calculated in real space.
and that for the long-range part, which is a lattice sum minus a self term, is calculated in wavenumber space

\[ E^G = (4\pi\varepsilon_0)^{-1} 2\pi L^{-3} \sum_{k \neq 0} \frac{-k^{-2}}{4\alpha^2} \sum_{j=1}^{N} q_j e^{-i k \cdot r_j} \left| q_j^2 - (4\pi\varepsilon_0)^{-1} \frac{1}{\pi} \frac{1}{\alpha} \sum_{j=1}^{N} q_j^2 \right|, \quad (1.50) \]

where \( k = 2\pi L^{-1} (l_x, l_y, l_z) \), \( L \) are the lattice points in the wavenumber space with the coordinates \( l_x, l_y, l_z \) as integers. The term \( \alpha \) is a parameter for optimising the convergence properties of both sums in equations 1.49 and 1.50.

The Ewald-sum calculates the force with high accuracy and uses the analytical Fourier transform to calculate the long-range force [11]. It is not suitable for non-crystalline systems, such as liquids or solutions. Because of the use of the Fourier transform, periodical artefacts are introduced to the non-periodical structure.

### 1.2.4 Treatment of Boundaries

When simulating a system of finite size, some thought must be given to the way the boundary of the system will be treated. The simplest choice is the vacuum boundary condition. For the simulation of a liquid, a solution or a solid, which is more usual than simulating a molecule in the gas phase, periodic boundary conditions are used. Another way to reduce edge or wall effects is to treat a part of the system as an extended wall region in which the motion of the atoms is partially restricted.

#### 1.2.4.1 Vacuum Boundary Condition

A simulation of a molecular system in vacuum has no wall or boundary and corresponds to the gas phase at zero pressure. If the vacuum boundary is used for a solid or a molecule in a solution, the properties of the atoms near or at the surface of the system will be distorted. The vacuum boundary condition may also distort the shape of a (non-spherical) molecule, because it tends to minimize the surface area. The best results in vacuum are obtained for large globular molecular systems [6].
1.2.4.2 Periodic Boundary Conditions

Periodic boundary conditions are used to avoid edge and wall effects on a molecular system [5]. The atoms of the system are put into any periodically space-filling box, which is treated as if it is surrounded by identical translated images of itself. When an infinite sum of the atomic interactions is performed, the interactions of an atom in the central computational box with all its periodic images are computed. In this case, a crystal would be simulated, which is not desirable in the most cases [6]. To treat non-crystalline structures, an atom should not simultaneously interact with another atom and a periodic image of that. Thus only the interactions with the nearest neighbours are taken into account. As shown in figure 1.9 the interactions of a solute-atom are calculated. The solvent-atom in the central box is farther away from the solute-atom than its image in the periodical box on the left. Therefore only the interaction with the image of that atom is calculated.

Using a neighbor list, only the interactions between the atoms within the cutoff sphere with radius $r_c$ are calculated as shown in figure 1.9. To consider only the interactions between the molecules of the nearest images, the cutoff radius $r_c$ must not exceed the half of the box size.

![Figure 1.9: Periodic boundary conditions](image)
The periodic boundary conditions affect not only the computation of the interactions, but also the positions of the atoms. It is common practice to keep the atoms in the central box. When an atom leaves the central box on one side, it enters with identical velocity on the opposite side at the translated image position.

As shown in figure 1.10 different boundary boxes can be used.

**Figure 1.10: Common used periodic boundary boxes**
The cubic or rectangular box with the side lengths $l_{box}$ is surrounded by 26 boxes over the distances $\pm l_{box}$ in the x-, y-, z-directions. The next layer of the neighbour images of the central computational box contains 98 boxes and so on. Using a cutoff, it must meet the following condition

$$r_c < \frac{1}{2} \text{MIN} (l_{box_x}, l_{box_y}, l_{box_z}).$$  \hspace{1cm} (1.51)

When simulating a spherical solute, the truncated octahedron as a more spherically shaped space-filling periodic box is used. Thus the number of solvent-molecules can be reduced, which is otherwise needed to fill the empty space in the cubic box. The truncated octahedron is obtained by cutting off the corners of a cube in a symmetrical way, that the distance between the opposite hexagonal planes is $\sqrt{3}/2$ times the distance between the opposite square planes. The volume of the truncated octahedron is half the size of the cube and yields in a sizeable reduction of the system to be simulated. The cutoff condition for the truncated octahedron is written as

$$r_c < \frac{1}{4} \sqrt{3} \cdot l_{box},$$  \hspace{1cm} (1.52)

where $l_{box}$ is the distance between the squared planes of the octahedron.

A third possible boundary box is the monoclinic box. It is mainly used for the simulation of crystals. The angle of the crystal cell determines the angle of the monoclinic box. Thus the periodic boundary can be used to model the regular structure of the crystal. The monoclinic box is not suitable for liquids and proteins. If only the nearest images of a molecule should be considered and a cutoff is used, its condition is

$$r_c < \frac{1}{2} \text{MIN} \left\{ l_{box_x}, \left( l_{box_x}^2 + l_{box_z}^2 \pm l_{box_x} \cdot l_{box_z} \cdot \cos \beta \right)^{1/2} \right\},$$  \hspace{1cm} (1.53)

where $l_{box_x}$, $l_{box_y}$ and $l_{box_z}$ are the lengths of the edges and $\beta$ is the angle of the monoclinic box.

1.2.4.3 Extended Wall Region Boundary Condition

An alternative to the periodic boundary condition is the extended wall region boundary condition. In this case the distorting effect of the vacuum outside the molecular system may be reduced by using a layer of atoms as an extended wall region to avoid the deforming influence of the nearby vacuum.
The extended wall region forms a buffer between the fully unrestrained part of the system and the vacuum. The motion of the atoms in this region can be kept fixed or harmonically restrained to stationary positions. In any case the type of the forces, which is exerted to the atoms in the extended wall region, should be chosen in a way, that the motion of the atoms in the finite system is similar to that of the infinite system.

The extended wall region technique may be used in the simulation of solids, liquids and proteins [6].

1.2.5 Simulation Methods

This section gives a short summary over the most common simulation methods for molecular dynamics. The various existing simulation methods can be classified as:

- Systematic search methods, which scan the complete configuration space of a MD system.
- Methods, which generate a representative set of configurations. It can be subdivided in:
  1. Non-step methods, which generate an uncorrelated series of random configurations.
  2. Step methods, which generate a new configuration from the previous one.
1.2.5.1 Systematic Search methods

If the molecular system contains only a small number of degrees of freedom (coordinates) and if the force field $V(r)$ does not have too many minima upon the variation of the degrees of freedom, it is possible to systematically scan the complete configuration space of the system. The computing effort, which is required by a systematic search of the degrees of freedom of a system grows exponentially with their number. Thus only very small molecular systems can be treated by systematic search methods. The complexity of the function $V(r)$ determines the degrees of freedom, that still can be handled within a reasonable computing time [6].

A possibility to speed up the calculation is to split the selection of low $V(r)$ configuration space into different stages, where in the first stage a low computing cost energy function $V_{\text{simple}}(r)$ is used and in the second stage the complete $V(r)$ is only evaluated for the remaining space. The simplified function $V_{\text{simple}}(r)$ has to be a correct projection of the complete function $V(r)$. This method has been used to predict the loop structure in proteins and to predict the stable conformation of small peptides.

1.2.5.2 Monte Carlo Simulation

The Monte Carlo simulation is one of the random search methods. From a starting configuration $r_s$, a new configuration $r_{s+1} = r_s + \Delta r$ is generated by a random displacement of one or more atoms. The random displacements $\Delta r$ should be in a way, that the available cartesian space of all atoms is uniformly sampled in the limit of a large number of successive displacements. The newly generated configuration $r_{s+1}$ is accepted if the energy criterion involving the change of the potential energy $\Delta E = V(r_{s+1}) - V(r_s)$ is $\Delta E \leq 0$ or $\Delta E > 0$ with $e^{-\Delta E/k_BT} > R$, where $R$ is a random number taken from a uniform distribution over the interval [0,1].

In order to obtain high computational efficiency, one would like to combine a large step size with a high acceptance ratio. This is possible, when applying the Monte Carlo technique to simulate simple atomic or molecular liquids. For complex systems with many covalently bounded atoms, very small step sizes should be used, because a random displacement will generate a very high bond energy of the bonds of the displaced atom. Therefore the Monte Carlo simulation is inefficient for macromolecular systems [6].
1.2.5.3 Distance Geometry Methods

The distance geometry method converts a set of bounds on distances between atoms into a configuration space of these atoms, which is consistent with these bounds. A molecular structure is described in terms of the set of all interatomic distances by pairs, which are written in the form of a distance matrix. The maximum distances of the atom-pairs are entered in the upper right-hand triangle whereas the minimum distances are entered in the lower left-hand triangle. This matrix describes the complete configuration space within the specific bounds. For the calculation a set of random configurations is generated by choosing atom-atom distances at random within the specific bounds. The resulting distance matrices are then converted into a structure in three-dimensional cartesian space [6].

It is not possible to apply an energy function $V(r)$ as e.g. equation 1.19 in a distance geometry calculation. The energy function has to be converted into a function of atom-atom distances and must be simplified to a set of bounds on these distances. Thus most of the information of $V(r)$ is lost. The distance geometry method cannot properly handle solvent structures, because a limited distance description of a liquid gives no possibility to have a proper statistical weight for many structures. Another problem is the characterization of the distribution of the generated three-dimensional configurations, because the conversion from the distance space into the three-dimensional space is non-linear.

1.2.5.4 Molecular Dynamics Simulation

In the molecular dynamics method a trajectory of the molecular system is generated by the integration of Newton’s equation of motion (eq. 1.1) for all atoms in the system [4]. The force on atom $i$ is denoted by $F_i$ and is the derivative of the potential energy $V(r)$, which must be a differentiable function of the atomic coordinates $r_i$ and whose gradient must be calculated in the MD simulation.

The simulation is performed in small time steps, where the interaction forces between the particles are evaluated, which determine the velocities and at last the new positions of the particles. The real time of a time step is typically 1-10 fs for molecular systems and a simulation runs over some nanoseconds.

The trajectory must have a sufficient length to form a representative ensemble of the state of the system. By averaging over the trajectory, static equilibrium quantities can be obtained. In addition, dynamic information can be extracted.

The molecular dynamics method is used for systems with 10'000-100'000 atoms with typically one or several proteins dissolved in liquid. A simulation of complex
systems with many covalent and intermolecular interactions is very time consuming [6]. E.g. a molecular system with 36'000 atoms requires about 40 seconds for one time step on a SUN Ultra 1/170.

1.2.5.5 Stochastic Dynamics Simulation

The method of stochastic dynamics is an extension of the molecular dynamics method. A trajectory of the molecular system is generated by integration of the stochastic Langevin’s equation of motion

\[
\frac{d^2 r_i(t)}{dt^2} = \frac{F_i(t)}{m_i} + \frac{R_i(t)}{m_i} - \gamma_i \frac{d}{dt} r_i(t).
\]

(1.54)

The term \( R_i(t) \) is a stochastic force and \( \gamma_i \) is the coefficient of a frictional force. The stochastic force introduces energy and the frictional force removes kinetic energy from the system. Here a condition for zero energy loss can be given as

\[
\langle R_i^2(t) \rangle = 6m_i \gamma_i k_B T_{ref},
\]

(1.55)

where \( k_B \) is the Boltzmann factor and \( T_{ref} \) the reference temperature of the system.

The stochastic dynamics method can be used to simulate a solvent effect or the coupling of the individual atom motion to a heat bath [6].
Methodology

In this chapter the methodology of computer simulations for molecular dynamics is discussed and the GROMOS computer program package is described and analysed. The first part shows general aspects of modelling chemical systems followed by some techniques of generating neighbor lists and a short description of different decomposition methods for parallelising a molecular dynamics simulation. In the next part, the GROMOS program is discussed and an overview of its most important algorithms is given. It finishes with a description of some parallel versions of GROMOS. The last part is an analysis of GROMOS. A molecular simulation topology is taken for benchmarking to find the most time consuming routines. Then a new technique is presented to estimate the performance of GROMOS on computer hardware by evaluating the number of floating-point operations. The chapter finishes by discussing aspects and methods for outsourcing the various routines onto a coprocessor for accelerating GROMOS.
2 Methodology

2.1 Computer Simulation of Molecular Systems

2.1.1 Modelling

Using computational methods for solving chemical problems is a growing discipline beyond experimental chemists. In computer simulations of molecular systems the macroscopic behaviour is computed from microscopic interactions. This offers a better understanding and interpretation of experimental results, semi quantitative estimates of experimental results and the capability to interpolate or extrapolate experimental data into regions, that are difficult to access in the laboratory [6].

In a computer simulation a model of the real world is constructed by computing measurable and immeasurable properties of a system. The former is compared with experimental results, which determines whether the used model is valid or invalid. A valid model may be used to study relationships between model parameters or to predict unknown and immeasurable quantities.

A computer simulation is used to describe or predict
- the structure and stability of a molecular system
- the (free) energy of different states of a molecular system
- reaction processes within a molecular system
in terms of interactions at the atomic level.

Chemical systems are generally too inhomogeneous and complex to be treated by analytical theoretical methods [6]. The treatment of molecular systems in a crystalline solid state or the gas phase can be simplified by reducing the many-particle system to a few-particle system and using a classical statistical mechanics approximation instead of quantum mechanical methods. No simplifications can be made treating liquid state macromolecules. A full treatment of the many-particle system is required to adequately describe the properties of molecular systems (see figure 2.1).
The simulation of molecular systems requires the generation of a statistically representative set of configurations, a so-called ensemble. The chemical properties of a system are defined as ensemble averages or integrals over the configuration space (or more generally phase space). These averaging can only be carried out over a part of the configuration space because a many-particle system has many degrees of freedom. A smaller configuration space gives a better approximation of the ensemble averaging, but affects a proper representation of the interesting properties. Therefore the size of the configuration space should be chosen in a way, that all those degrees of freedom are included, which are essential for a proper evaluation of the interesting property of a system.

The accuracy of the predicted quantities in a simulation depends on the quality of the assumptions and approximations of the applied molecular model and the interatomic force field. There exists a variety of models and force fields, which differ in the accuracy by which different physical quantities are modelled. The choice of a force field should depend on the system properties and the level of accuracy, which are interesting. To save computing power, a balance between the refinement of different parts of a molecular model and the level of accuracy has to be found.

In computer simulations of fluid-like molecular systems the best compromise between the size of the configuration space, the accuracy of the molecular model and the available computing power has to be found (see figure 2.2).
The proceeding to find an adequate molecular model between these three factors are:

1. The size of the configuration space is determined
2. The accuracy of the properties should be specified
3. The simulation time should be estimated

There is a trade off between the computing power for searching the configuration space and the accuracy of the force field. In many practical cases it is not possible to simulate the system of interest, because the available computing power does not allow for a sufficiently accurate simulation [6].

### 2.1.2 Neighbor List Techniques

For systems consisting of a small amount of particles $N$, it is convenient to treat the interactions, such as forces and energies, between all particle pairs. In this case, the calculation effort is proportional to $N^2$. Since the interactions between particles decrease with the distance between them, interactions between particles exceeding a cutoff radius $r_c$ have not to be calculated (see section 1.2.3). With the generation of a neighbor list, which contains only the particle pairs, whose distance is within the cutoff radius $r_c$, the calculation time of the interactions can be drastically reduced, mainly for large molecular systems.

The neighbor list should only be updated every 5th-20th simulation step (see section 1.2.3.1) and the calculation of the non-bonded interactions remains proportional to $N$. 

---

![Diagram](image.png)

**Figure 2.2: Choice of the molecular model, force field and sample size**

- Property or Quantity of Interest
- Molecular Model
- Force Field
- Sampling
- Required Accuracy
- Available Computing Power

The size of the configuration space is determined. The accuracy of the properties should be specified. The simulation time should be estimated. There is a trade off between the computing power for searching the configuration space and the accuracy of the force field. In many practical cases it is not possible to simulate the system of interest, because the available computing power does not allow for a sufficiently accurate simulation [6].

---

### 2.1 Computer Simulation of Molecular Systems

33
2.1.2.1 Brute Force Algorithm

The simplest way to find the neighbours of a particle, which are within the cutoff radius, is to scan all possible particles of the system. This makes the computing time for generating a neighbor list proportional to \( N^2 \), because \( N(N - 1)/2 \) particle pairs must be scanned for a system of \( N \) particles.

Although the neighbor list has not to be generated at each time step, the calculation effort is relatively high. The brute force algorithm is only efficient for systems with \( N < 10^4 \), because larger systems use too much time for updating the neighbor list and require considerable storage capacity [12].

2.1.2.2 Cell Index Method

Hockney and Eastwood have proposed a neighbor searching method, which requires computer time proportional to \( N \) [12]. In that method the simulation box is subdivided into a grid of cubic cells, whose side lengths \( h_{cell} \) are chosen that

\[
\hat{h}_{cell} > r_c
\]

and if periodic boundary conditions are applied

\[
n \cdot h_{cell} = l_{box} \quad n = \text{integer}, \geq 2 ,
\]

where \( l_{box} \) is the side length of the cubic periodic box. In rectangular simulation spaces, equations 2.1 and 2.2 apply separately for each edge length in direction \( x, y, z \).

Due to the condition in equation 2.1, the cutoff radius does not exceed the size of a cell and therefore an interacting particle can only be located in the same cell or in one of the 26 neighbouring cells [13]. Using Newton's Third Axiom [8] of forces by pairs, the neighbours of a particle must only be searched in one direction, what means, that only the same cell and 13 neighbouring cells have to be searched for neighbours.

Figure 2.3 shows a two dimensional profile of a slice of the simulation box with nine cells. For generating a neighbor list, only the greyed cells must be searched for neighbours. In \( z \)-direction additionally the nine cells located in the rows behind the cells have to be considered for neighbour searching (see also figure 2.4).
The cells are numbered in turn to the $x$-, $y$- and $z$-directions. The number $n_c$ of the cell is determined as

$$n_c = (c_y \cdot nb_x + c_x) + c_z \cdot nb_x \cdot nb_y,$$

(2.3)

where $nb_x$ and $nb_y$ are the number of rows of cells in $x$- and $y$-direction and $c_x, c_y, c_z$ represents the cell coordinates. For searching the interaction partners of a particle, which resides in a cell at the position $x, y, z$, the coordinates of the cells to be searched for neighbours can be obtained with the following rules:

$$c_x = [x, x+1] \mod nb_x, \quad c_y = y, \quad c_z = z$$

$$c_x = [x-1 \ldots x+1] \mod nb_x, \quad c_y = (y+1) \mod nb_y, \quad c_z = z$$

$$c_x = x-1 \ldots x+1 \mod nb_x, \quad c_y = [y-1 \ldots y+1] \mod nb_y, \quad c_z = (z+1) \mod nb_z$$

(2.4)

The cell number $n_c$ of the concerning cells can then be obtained using equation 2.3.

Figure 2.4 shows an example of the cell numbering with a simulation box divided up in $4 \times 3 \times 5$ cell rows. Additionally it illustrates, which cells must be searched for neighbouring particles using periodic boundary conditions.
For cell 6, with the cell coordinates (2,1,0), the periodic boundary conditions does not apply. The neighbour searching considers the primary cell 6, cell 7 (3,1,0), which is on the right, cells 9 - 11 on the rows \( c_x = 1 - 2 - 3 \) and \( c_y = 2 \), on the row \( c_z = 0 \) as well as cells 13 - 15, 17 - 19 and 21 - 23 on the rows \( c_x = 1 - 2 - 3 \), \( c_y = 0 - 1 - 2 \) and \( c_z = 1 \).

Cell 11 (3,2,0) has neighbouring cells, which belong to periodically repeated images of the simulation box. These are the cells 0, 2, 3, 8, 12, 14, 15, 16 and 20, which are on the next higher \( x \)- and \( y \)-rows of cell 11.

The particles in the system are identified with a unique sequence number starting from zero to \( N-1 \) and are spatially sorted into their appropriate cell. The assignment of the particles to their respective cell is made with the linked list method, which uses a head and a chain array for addressing [5].

The index of the head array is the cell number and its value is the sequence number of the first particle of the respective cell. The sequence number of the particle is used as index of the chain array, where the appropriate value delivers the sequence number of the next particle in the cell and so on. The sequence number of the last particle is identified with a value of -1 in the chain array.

Figure 2.5 shows an example of the linked list method of cells 0 and 1 with five and six particles.
2.1 Computer Simulation of Molecular Systems

The drawback of this neighbor searching method is that the cells cover a cubic volume, whereas the cutoff radius covers a sphere. This may introduce an artificial cubic symmetry into the system, which is called the cube corner effect [12]. It can be avoided by introducing a distance check for every pair, which has been found.

The cell index method is efficient for systems with large simulation boxes and short cutoffs. To be more efficient than the brute force method, at least three rows of cells are required in each direction, that the cell searching method comes in effect.

In practice, MD simulations, especially for biomolecules, are defined to have a minimum system size. Thus the cutoff is relatively large, compared with the size of the simulation box dimensions and therefore too few cells can be obtained from the box. This makes the method no more efficient for such simulations.

2.1.2.3 Grid Cell Method

For a better approximation towards the cutoff sphere and to avoid the drawbacks of the cell index method, a finer grid as in the previous method can be used. The grid cell method subdivides the box into a grid of cubic cells with such a small size, that not more than one particle occurs in one cell [12]. This means, that the distance over the diagonal of the volume of a cell is shorter than the shortest distance between two particles. The side lengths $h_{cell}$ of a cell follow the condition in equation 2.2.
Figure 2.6: Two dimensional view of the grid cell method

The coordinates $c_x$, $c_y$, $c_z$ of the cells, which have been searched for neighbours of the primary cell at position $x$, $y$, $z$, can be found using

$$
\begin{align*}
  c_x &= [x \ldots x + nc_x], & c_y &= y, & c_z &= [z - nc_z \ldots z + nc_z] \\
  c_x &= [x - nc_x \ldots x + nc_x], & c_y &= [y + 1 \ldots y + nc_y], & c_z &= [z - nc_z \ldots z + nc_z]
\end{align*}
$$

(2.5)

where $nc_x$, $nc_y$ and $nc_z$ are defined as

$$
n_{c_{x,y,z}} = \left\lceil \frac{r_c}{h_{cell_{x,y,z}}} \right\rceil,
$$

(2.6)

which represents the rounded up fraction to the next higher integer. To comply the periodic boundary condition, the coordinates may be adjusted with the modulo function or the grid can be extended by a few surface layers into the adjacent periodic images of the simulation cell [14].

The cells on the corners of the searching cube are farther away from the primary cell than the cutoff length and fall out of the cutoff sphere. For a better approximation to the cutoff sphere, this cells may be eliminated if their minimum distance to the primary cell is bigger than the cutoff radius. The minimum distance $r_{min}$ is defined as
\[ r_{\text{min}} = \sqrt{ (h_{\text{cell}_x} \cdot \max(|d_x| - 1, 0))^2 + (h_{\text{cell}_y} \cdot \max(|d_y| - 1, 0))^2 + (h_{\text{cell}_z} \cdot \max(|d_z| - 1, 0))^2 }, \tag{2.7} \]

where the terms \( d_x, d_y \) and \( d_z \) are the coordinates of the grid distance between two cells.

The particles are sorted spatially into the cells. As each cell contains only one particle or none, the linked list method is obsolete and for a representation of the cell grid of the simulation box, a three dimensional array may be used, which contains either the particle number or a 'L' for an empty cell.

This method has the drawback, that in some simulations many cells are empty, which results in an inefficient neighbour searching. To avoid many empty cells, a coarser grid may be used, where particles of multiply occupied cells are reassigned to empty neighbour cells and the search space is extended to reach this relocated particles [14]. With this the grid cell method becomes even in such simulations more efficient.

### 2.1.3 Parallelising

#### 2.1.3.1 Decomposition Methods

For parallelising a molecular dynamics problem, there must be determined, which part of the problem is calculated by a particular processor of a number of \( P \) processors. In principle, there are the two following ways for decomposing a MD problem [15]:

- logical decomposition
- geometric decomposition

In a logical decomposition, each processor performs the calculation on a subset of \( N/P \) particles. At the end of a time step, each processor has to broadcast the new positions of its appropriate particles to the other processors. With the updated positions the calculation of the next step occurs.

In a geometric decomposition the simulation box is decomposed and each processor performs the MD step on the particles within a partition of the simulation box. When a particle moves from one partition to another, its position is transferred from one processor to another. As particles within a cutoff distance of a bordering partition must be known to the processor, which is responsible for the neighbouring partition, the positions of these particles must be exchanged between neighbouring
processors after each time step. A problem introduced by this decomposition is the load balancing, because the number of particles in a partition can vary. This problem can be reduced by adjusting the size of the partitions at runtime [16].

2.1.3.2 Data Distribution

The principle of the storage of the particle data is dependent of the used memory architecture on the parallel processor hardware. On a shared memory architecture, all processors access the same memory and the positions of the particles are all stored in this memory [57]. No communication between processors is needed, because all work with the same data-set. In distributed memory systems, each processor has its own memory, where the data for its own calculation are written [57]. As each processor calculates only a part of the problem, there are partial results in each memory, which have to be adjusted over all processor memories at least after each time step. The processor and its memory in a distributed memory system is called processing element (PE) or processor node.

Today's parallel computers use mostly a distributed memory architecture, because such systems are easy scalable and each processor manages its own memory. In a distributed memory system, the data can be stored in the following two methods:

- replicated data
- distributed data

Using the replicated data method, each PE stores the whole set of all positions. This method requires more memory, but no communication is required during a time step.

In the distributed data method, the positions of the particles are distributed over all PEs in such a way, that each of them stores the positions of the particles, for which it has to calculate the interactions. As only data of \( N/P \) particles are stored on each PE, the size of the memory per processor can be the smaller, the more processors are used. In this method the PEs have also to communicate during a time step, because a particle has basically to be tied together with all other particles for the evaluation of its interaction. Also if a cutoff is used, positions of such particles are required, which are stored on another PEs.

To have a good compromise between memory size and communication effort a mixed form between the two methods can be used. Especially in geometric decomposition the position of the particles in border regions are stored on the two PEs, which handle the neighbouring partitions of this region [17].
2.1.3.3 Techniques for Parallelising MD Algorithms

Already different techniques have been used for parallelising MD problems. The techniques use different combinations of the methods, which are specified above and are described in the following:

Atom Decomposition: This technique uses logical decomposition. Each PE receives $N/P$ particles. Whereas in the past mostly distributed data was used, replicated data is used with today's larger memories [18].

Force Decomposition: The technique uses logical decomposition due to all possible interactions. Each PE receives only the corresponding atom pairs of the interactions, which have to be calculated on them [19]. The data storage is a mixed form between distributed and replicated data, because particles may be stored on more than one PE. If a neighbor list is used, the interactions are decomposed according to the entries in this list [20].

Interaction Decomposition: Commonly the same as force decomposition [21].

Domain Decomposition: This is the common technique of dividing the domain associated with the problem into parts or grains, where one grain is assigned to each PE [22]. In MD problems domain decomposition means a geometric decomposition, where communication occurs usually at the boundaries of the local domains and is mostly restricted to PEs, which are responsible for neighbouring domains. Data may be assigned using distributed data, replicated data or a mixed form, where the particles in boundary regions are sent to adjacent PEs [23].

Spatial Decomposition: This is the technical term for domain decomposition in MD using distributed data. The size of the local domains are so large, that each PE must only communicate with the PEs owning adjacent domains [24].
Systolic Loops: This technique uses atom decomposition with distributed data and defines fundamentally the manner of the data communication. The data packets move around a loop in a synchronised and circulatory flow, such that each PE simultaneously transmits a packet to the next neighbour and receives a packet from the previous neighbour [25]. Each PE receives a data packet with a subset of all $N$ particles, which can be communicated in four different methods: A copy of each data packet is sent over the half of the loop (systolic loop double method), the half of the respective data packets are sent over the whole loop (systolic loop half method), two equal data packets are sent in two different directions around a bidirectional loop (systolic loop bidirection method) or each half of the respective data packets are sent in two different directions around a bidirectional loop switching its directions after passing the half of the loops (systolic loop single) [26].

2.2 The GROMOS Package

2.2.1 Introduction

GROMOS is an acronym of the GROningen MOlecular Simulation computer program package and has been developed since 1978 by W. van Gunsteren and his research group [27]. The package is made for the dynamic modelling of biomolecules and has the following capabilities:

- Simulation of proteins or arbitrary molecules using the molecular dynamics (see section 1.2.5.4), stochastic dynamics (see section 1.2.5.5) or path-integral method.
- Energy minimisation of these molecules
- Analysis of molecular conformations obtained by experiment, by model building or by computer simulation.

The GROMOS™ package has originally been written for the simulation of protein molecules and was developed further into a general purpose molecular simulation
package. It is meant for use in a scientific field with always changing users, which have different applications of simulation methods [7]. Therefore GROMOS has the following characteristics:

- Transparency of code for easy modification
- Modular architecture for using parts of it and introducing own subroutines
- Independence of the code of the used forced field
- Independence of the code of the available computer hardware

GROMOS has special optimised routines for modern vector- or parallel shared-memory computers and contains programs for the analysis of MD trajectories and structures [27]. The whole package has the following six types of programs:

1. Programs for building the molecular topology
2. Programs, which transform non-GROMOS data into the format required by GROMOS
3. Programs for generating the atom coordinates
4. Programs for simulations
5. Programs for analysing the structures
6. Programs, which merge or reduce coordinate files or transform the atom coordinates to a special format

The user has a large flexibility to choose the combination of the program modules, which are suitable for solving a particular problem. Often a new combination of program modules can perform a task, which was not considered as GROMOS was developed [7].

New and improved simulation methodologies are continuously developed by van Gunsterens research group and are implemented in GROMOS. The latest version of GROMOS is GROMOS96, which consists of about 73’000 lines of standard Fortran (Fortran77) code.

The GROMOS™ software is currently used by more than 400 academic and industrial research groups all over the world.

2.2.2 Simulation Methods in GROMOS

GROMOS calculates the trajectories of the particles by solving Newton’s equation of motion (eq. 1.1) using the molecular dynamics method (see section 1.2.5.4) or by solving Langevin’s equation of motion (eq. 1.54) using the stochastic dynamics method (see section 1.2.5.5). Both methods use the Leapfrog algorithm [9] for the integration of the equations of motion. Generally a simulation may include several constraints like periodic boundary conditions (see section 1.2.4.2), cutoff and dis-
tance rules (see section 1.2.3), virial calculation and more. These constraints are handled with the *SHAKE algorithm* [28].

In biomolecular simulations and therefore also in GROMOS two different types of molecules are treated. Due to the different structure of the solute- and solvent-molecules, the force calculation subdivides, based on the different interaction types, into the *solute-solute*, *solute-solvent* and *solvent-solvent* interactions. As described in section 1.2.1 the bonded forces may only be calculated for the solute-solute interactions and just the non-bonded forces should be calculated for all interaction types. Basically each atom of each molecule should be combined with each other. The amount of the calculations of the different interaction types for a molecular system with $N$ atoms can be illustrated by using a $N \times N$-matrix, whose row and column numbers correspond to the respective atom numbers (see figure 2.7). The elements $a_{ij}$ of the matrix are then the results of the interactions between the atoms $i$ and $j$.

![Figure 2.7: Interaction calculation matrix for illustrating the amount of calculations of the different interaction types](image)

Due to Newton’s Third Axiom [8], it is $a_{ji} = -a_{ij}$, what means, only the half of the elements of the matrix have to be calculated. These are the grey shaded parts of the matrix in figure 2.7. The other elements, illustrated as hatched parts, can be obtained by inverting the corresponding value of the other half of the matrix.

The number of solute-atoms is generally much smaller than that of the solvent-atoms. Thus there are few solute-solute interactions and therefore also few bonded interactions to calculate. The calculation of these forces is more complicated and is described with the corresponding equations in section 1.2.2.1. The number of the non-bonded interactions is relatively high, but also easier to calculate. GROMOS
uses the Lennard Jones approximation for the calculation of these forces, which is described in section 1.2.2.2.

To reduce the number of solvent atoms, periodic boundary conditions (see section 1.2.4.2) can be used. GROMOS can treat all the periodic boundary boxes, which are illustrated in figure 1.10, but also simulations in vacuum (see section 1.2.4.1) are possible. Additionally to the simulation methods with one cut-off, GROMOS can also use the twin range method (see section 1.2.3.1).

### 2.2.3 Algorithms in GROMOS

In simulations using step methods, the Verlet algorithm is generally used [9]. From the positions of the particles, the forces and the energies are calculated. The forces determine the velocities, with those the new positions for the next simulation step can be evaluated. The basic GROMOS algorithm is shown in figure 2.8.

![Figure 2.8: GROMOS algorithm](image)
After the initialisation, where the parameters and the initial positions of the particles are read from an input file and written to the corresponding variables, the program enters to the main loop. At the beginning of each simulation step it is decided whether a neighbor list or a pairlist respectively has to be generated or updated. If yes, the pairlist is generated and the long range forces are calculated. After this, the program calculates the interactions between the atoms, which are the bonded forces (see section 1.2.2.1) and the non-bonded forces (see section 1.2.2.2), the energies and if required the virial, which is a term to evaluate the pressure of the molecular system. In the integration step all partial forces for each atom are added together, so that in the next step the velocities can be calculated and the new positions of the particles can be evaluated. At last it is decided whether a further step has to be calculated. Then the program returns back to the pairlist generation, otherwise it ends.

In the following, the most important algorithms of GROMOS are described in more detail. They belong to the most time consuming parts of the simulation program, whose structure is short and clear.

2.2.3.1 The Distance Algorithm

The distance algorithm calculates the distance vector and the square of the distance. It is used in two different parts of GROMOS: for the pairlist generation and for the force calculation.

The algorithm handles four different spaces: vacuum, cubic periodic, truncated octahedron periodic and monoclinic periodic. As GROMOS may simulate molecules in a space with four dimensions, three- or four-dimensional distances can be calculated with this routine. Due to its physical definition and chemical requirements, the distances in the octahedral space and the space with the monoclinic boxes have only to be calculated in three dimensions.

There are many conditions in the distance algorithm program. First the distance vector in vacuum is calculated. Then the distance is calculated for atoms, which are in the vacuum, otherwise the distance vector is adjusted to the boundary box and the distance is calculated. If we have monoclinic boxes or truncated octahedrons, the distance vector is adjusted to this structure.
The adjustment of the distance vector runs as follows. Every component of the distance vector is first compared with the positive value of the half length of the boundary box. If the component is higher, the side length of the box is subtracted, else the component is compared with the negative value of the half length of the box. If it is lower, the side length of the box is added. In a space with periodic truncated octahedrons the same adjustments as for monoclinic periodic boxes are made. After then, the distance vector is adjusted for the octahedral space unless both atoms are in the middle of the octahedron and no adjustment is required.

If the algorithm is used for the pairlist generation, no distance vector is calculated and the square of the distance is only calculated in three dimensions for all simulation spaces.

Figure 2.9: Distance algorithm in GROMOS
2.2.3.2 The Pairlist Algorithm

The pairlist algorithm generates the neighbor list, called pairlist in GROMOS. This algorithm may only be executed every 5th-20th simulation step (see section 1.2.3.1). For the pairlist calculation the distance between all charge groups has to be evaluated and compared with the cutoff length. The position of the solute charge groups is their centre of geometry, whereas the position of the solvent charge groups is the position of their largest atom (see section 1.2.1 on page 10).

Figure 2.10: Pairlist algorithm in GROMOS
The pairlist generation uses the distance algorithm to evaluate the distances between the charge groups \( i, j \). If the distance is smaller than the cutoff, the charge group \( j \) is entered into the list of charge group \( i \). Using the twin range method, the algorithm evaluates immediately the interactions between the charge groups \( i, j \) if the distance lies between the two cutoffs. If no twin range method is used, the two cutoffs must have the same value.

The pairlist algorithm consists mainly of decisions. First it has to be determined if a pairlist has to be generated in the current time step. If no, the program goes further to the interaction algorithm, else the centre of geometry of all solute charge groups is determined. After that, the program enters into the pairlist generation loop. The first step in this loop is the calculation of the distance between the two charge groups \( i \) and \( j \), using the distance algorithm (see section 2.2.3.1). The calculated distance is compared with the short cutoff and if it is smaller, charge group \( j \) is entered into the pairlist at the position of charge group \( i \). If not, the distance is compared with the long cutoff and if it is smaller, the long range forces between the charge groups are calculated, else nothing further is done. After that, the program jumps back to the distance calculation to handle the next charge group pair or finishes after considering all charge group pairs.

For the pairlist generation each charge group has to be combined with each other. The algorithm uses two nested loops, which run over the charge group numbers \( i \) and \( j \). To avoid a double calculation of the distances, the loops follow the condition \( i < j \). Having a number \( CG \) of charge groups, the loops are programmed as:

\[
\text{for } i = 1 \text{ to } CG-1 \\
\quad \text{for } j = i+1 \text{ to } CG \\
\quad \quad \text{pairlist generation} \\
\text{end (for } j) \\
\text{end (for } i) \\
\]

\text{Figure 2.11: Nested loops in the pairlist algorithm of GROMOS}

The pairlist generation is a typical \( CG^2 \) problem because the effort for the calculation raises quadratically with the number \( CG \) of charge groups. Regarding that no charge group pair is calculated twice, the distances of

\[
n = \frac{CG \cdot (CG - 1)}{2}
\]

pairs have to be calculated.
The size of the pairlist determines the calculation time of the non-bonded forces, because these interactions are only evaluated between the atoms, which are enlisted in the pairlist. For an estimation of the simulation time, therefore the size of the pairlist has to be known.

The number of the elements in the pairlist depends on the density of charge groups in the simulation space. Since a neighbor list of a charge group contains only that charge groups, which are within the cutoff radius, it contains the average number of charge groups, which are in a sphere with the radius of the cutoff length. However the density of the solute charge groups is higher than that of the solvent charge groups and there are density fluctuations, but the deviation between the so estimated number of neighbours and that of the real simulation is relatively slow and can be neglected.

Because there are different forces between solute-solute, solute-solvent and solvent-solvent calculations, other programming routines for the force calculations are used. Thus the neighbor list has to be divided up into three lists, one for solute-solute (subscript ‘ss’), one for solute-solvent (subscript ‘sv’) and one for solvent-solvent (subscript ‘vv’) interactions. The size of the neighbor list is

\[ NN = NN_{ss} + NN_{sv} + NN_{vv}, \]  

(2.9)

where

\[ NN_{ss} = \frac{2\pi}{3} \cdot \frac{r_c^3}{V_{box}} \cdot CG_{su}^2 \]  

(2.10)

\[ NN_{sv} = \frac{2\pi}{3} \cdot \frac{r_c^3}{V_{box}} \cdot CG_{sv}^2 \]  

(2.11)

and

\[ NN_{sv} = \frac{2\pi}{3} \cdot \frac{r_c^3}{V_{box}} \cdot CG_{su} \cdot CG_{sv}. \]  

(2.12)

A comparison of the estimated size of the pairlist using equations 2.9 - 2.12 with the rectangular Thrombin simulation (see section 2.3.1) results in a deviation of \( NN_{vv} \) with approximately -5% and the deviation of \( (NN_{ss} + NN_{sv}) \) lies between +18% and +50% varying the cutoff from 2.0nm to 0.6nm. Thus the estimation delivers too many solute interactions at the cost of too few solvent pairs.
2.2.3.3 The Non-bonded Solvent-Solvent Interaction Algorithm

This algorithm evaluates the forces, the energies and the virial of the non-bonded solvent atoms, whose charge groups are in the pairlist. It uses the distance and the distance vector, which are obtained from the distance routine. The force and the virial are described as vectors, the energies are described as scalars. In each run, the partial energies, the virial- and the force vector from atom $j$ exerted on atom $i$ are calculated and added to the already calculated sums of the energies, the virial- and the force vector of atom $i$.

![Diagram](https://via.placeholder.com/150)

*Figure 2.12: Non-bonded solvent-solvent interaction algorithm in GROMOS*
The force vector can be calculated in three or four dimensions. However there is no handling of the simulation spaces in the force routine because the influences of the simulation space are already included in the distance and the distance vector. The vector of the virial is ever calculated in three dimensions and can also be skipped for simulations, which do not need them.

The non-bonded interaction algorithm for the solvent atoms is a sequential algorithm. First, the van der Waals parameters \( C_6 \) and \( C_12 \) (see section 1.2.2.2), which depend on the atom types, are evaluated. Using these parameters and the distance, the energies are calculated with equations 1.40 and 1.41. In the next step the force vector is evaluated with equations 1.42 and 1.43 by using the distance vector. If wished, the vector for the virial is also calculated at last.

The other interaction algorithms, that for the bonded and the non-bonded interactions between the solute atoms have a similar structure as in figure 2.11, which is however more complicated. In addition to equations 1.40 - 1.43, these program parts compute equations 1.20 - 1.38.

2.2.4 Parallel Versions of GROMOS

2.2.4.1 UHGROMOS

UHGROMOS is a parallel implementation of the GROMOS molecular dynamics simulation software and was developed at the University of Houston and the Texas Center for Advanced Molecular Computation. The parallel implementation uses atom decomposition with replicated data (see section 2.1.3.3). Only the most computational-intensive parts of the MD algorithm, the non-bonded interaction calculation and the pairlist generation, which uses more than 80% of execution time in a serial implementation, have been parallelised. All other parts are executed sequentially.

As each processor has the whole data structure, communication is only necessary to accumulate the forces and additional information about the actual workload is exchanged to accommodate dynamic load balancing. The non-parallelised parts of the program are executed redundantly on each processor.

The efficiency of UHGROMOS is low for large molecular systems as well as a high number of processors. The software can be run on workstation clusters and on supercomputer-class machines [29].
2.2.4.2 **EULERGROMOS**

EULERGROMOS is another parallel implementation of GROMOS and was developed at the Rice University and the University of Houston. It uses spatial decomposition (see section 2.1.3.3), but does not limit communication to the nearest neighbours. Thus the lower bound of the partition size and therefore the number of processors are not limited. All major phases of the time step loop of the MD program have been parallelised, including numerical integration with constraints, non-bonded interaction calculation and pairlist generation.

Each processor receives the particles of its partition of the simulation space. Particles in border regions of adjacent partitions are distributed on all neighbouring processors. EULERGROMOS communicates data in one of two possible ways. In Point-to-Point communication, messages are sent directly between processor pairs, that share data; in Shift communication, each processor communicates exclusively with its six immediate logical neighbours. This results in a coordinated use of the interconnection network and allows high performance also with a large number of processors. Another characteristic of the program is the use of dynamic load balancing, where individual partition sizes are adjusted to accommodate inhomogeneous atom densities.

EULERGROMOS achieves a high performance for a large number of processors. If few processors are used the performance is lower than that of UHGROMOS. Therefore the program should be run on supercomputer machines with a high number of processors [30].

2.2.4.3 **GROMOS96P**

GROMOS96P is a parallelised version of the GROMOS96 code and was developed at the Institute of Computer Engineering and Networks Laboratory, at the ETH Zurich, Switzerland during this interdisciplinary project. The parallel algorithm uses the spatial decomposition method (see section 2.1.3.3) with replicated data. In GROMOS96P, only the pairlist generation and the non-bonded interaction calculation have been parallelised. All other parts are calculated sequentially on one processor.

To improve the performance of the basic algorithm and for a better parallelisation, a new pairlist algorithm was implemented into GROMOS96P. Whereas GROMOS96 uses the brute force algorithm (see section 2.1.2.1), which is very slow, GROMOS96P uses an algorithm, which is based on the grid cell method (see section 2.1.2.3) with a coarser grid similar to [14], but with another criterion of the cell size.

The size of the cells are chosen in such a way, that they approximate a cube and
that their total number approaches the total number of charge groups. To simplify the search for neighbouring cells, the number of cells per cutoff is the same in all spatial directions. In contrast to [14], the cells are allowed to be empty or to contain more than one charge group. The algorithm searches all neighbouring cells as delivered by equation 2.5, but does not test minimal cell distances with equation 2.7. This means, that the cells with the cross in figure 2.6 are also checked. Instead of the minimal cell distances, the distances of the charge groups located in the neighbouring cells are evaluated to decide if they are enlisted to the pairlist. Experiments have shown, that the efficiency of both methods is almost the same [31].

The parallel GROMOS96P algorithm supports rectangular, truncated octahedron and monoclinic periodic and non-periodic boxes (see section 1.2.4.2) as used in GROMOS. For the parallelisation, the whole simulation space is divided into $P$ regions, which are allocated to the single processors. The regions are further subdivided into slices of cells in accordance with the pairlist algorithm, whereas the size of the cells must fit into the regions as well as into the simulation box. The regions have all the same size for rectangular and monoclinic simulation boxes. Using truncated octahedrons, the widths of the border regions on the edges are wider than that of the centre regions to achieve a better load balancing.

Each of the processors generates the pairlist of the charge groups in its own slice and calculates the non-bonded interactions of the concerned molecules.

Figure 2.13: Spatial decomposition in GROMOS96P
Figure 2.13 shows an example of the spatial decomposition of a rectangular box for 4 processors used in GROMOS96P. The cell size is chosen, that there are 2 cells per cutoff and a total of 20 cells along the z-axis. Due to the cubic cell size and the rectangular box, there are only 10 cells along the x-axis and 15 cells along the y-axis. 4 regions, each 5 slides wide, divide the simulation space up for the processors. The searching domain of each processor overlaps the region of its neighbour processor by 2 slices, because charge groups, which are on the right border of a region, may have pairs within the cutoff distance, which lie in the left two slices of the next region on the right side. Thus, each processor has a searching domain, which reaches over 7 slices, whereas its primary cells are all within the 5 slices of its own region. The interaction calculation of each processor considers only the molecules in the 5 slices within its region.

GROMOS96P does not yet support the twin range method (see section 1.2.3.1). Using two cutoffs, the short cutoff is increased to the long cutoff and twin range interactions are handled as short range interactions.

If distributed data would be used, the coordinates of the charge groups and the atoms may be sent to the particular processors, which handle the particles of the neighbouring regions to the left. After the pairlist and interaction calculation, the partial forces may be sent back to the processors, which are responsible for the regions to the right. With a large cutoff and a large number of processors, the regions would be so narrow, that the searching domain of each processor reaches over multiple regions and the coordinates of each region might be sent to more than one processor. Especially in biomolecular simulations with liquids, the cutoff is rather long compared with the simulation box and the use of distributed data would lead to a large communication overhead during the simulation. Therefore GROMOS96P uses the replicated data method for storing the simulation data. With this, the algorithm may use an efficient broadcast algorithm for sending the data to the processors.

The use of replicated data gives another gain for GROMOS96P. The handling of the data remains still the same by using parallel algorithms. At the beginning of the simulation, the first processor, which is called host processor in GROMOS96P, sorts the charge groups in their appropriate cells and sends all data to the processors, where the pairlist and the interactions of the according region are calculated. The results are sent back to the host, which integrates the partial forces and evaluates the new positions and sends the new data to the processors again and so on.

The major advantage of the replicated data method is that only few changes of the existing code are required [17]. Thus the sequential parts of GROMOS could be reused in GROMOS96P and only the parts for parallel execution had been newly developed. Thus the most sequential parts are calculated redundantly on each paral-
lei processor and not only on the host. With this method many communication cycles can be saved. Figure 2.14 illustrates the control and data flow graph of GROMOS96P.

![Control and data flow graph of GROMOS96P](image)

*Figure 2.14: Control and data flow graph of GROMOS96P*
Communication is only done at the start of the simulation and at the beginning and the end of the iteration. The most routines within the iteration are executed on all machines either in parallel or redundantly. In GROMOS96P also the routines for evaluating the centres of mass of the molecules, which are used for the virial calculation, have been parallelised. These are separate routines in GROMOS and not part of the pairlist or the interaction calculation.

Three synchronisation methods can be used in GROMOS96P: synchronous, barrier synchronous and asynchronous. In the synchronous method, all processors are synchronized after processing each routine. The barrier synchronous method uses synchronization points in the code, which are shown in figure 2.14. In the asynchronous method, the calculation and the communication run asynchronous during a time step and the processors are only synchronized at the general synchronisation point before the data are sent back to the host. The synchronous method uses the blocking communication, where the sender has to wait until the communication is finished. In the barrier synchronous and the asynchronous methods, the communication between the nodes is non-blocking, what means, that the sender writes the data into a data buffer and processes the next part of the program without waiting, till the receiver has fetched the data.

The load balancing is coupled to the spatial decomposition and is not dynamic. In case of a rectangular and a monoclinic simulation box, where the regions have equal size, a good load balancing can be achieved, but in truncated octahedrons, the regions on the border contain fewer particles and a good load balancing is difficult to find. Additionally the load balancing of the parallel pairlist algorithm becomes worse the more processors are used [31].

GROMOS96P uses MPI software interface routines [97] for the communication. The code is in standard Fortran and C. This makes the code independent of the computer platform and the communication network. The only requirements are a UNIX system with C and Fortran compilers and an MPI interface on the communication hardware [31].

2.3 GROMOS Analysis

2.3.1 Benchmark Topology

For analysing the performance of the GROMOS™ software, a molecular simulation topology is necessary, which covers a most possible wide range of simulations. The main problem, which has to be often simulated with GROMOS, is a solute, typically
a protein, surrounded by solvent in periodic boundary conditions [32]. Thus a molecular topology had to be found, which is similar to the main problem and can be used for doing timing measurements of GROMOS. The results can be used to give an estimation how much time is taken by simulations of the main problem.

The Thrombin molecular topology fulfills the requirements of the main problem quite well. It has 5 - 10 times more solvent atoms than solute atoms. We use two simulations with rectangular and truncated octahedron periodic boxes. A simulation in vacuum as well as monoclinic boxes is disclaimed, because vacuum is hardly used and monoclinic boxes are mainly used for crystalline simulations (see section 1.2.4.2).

<table>
<thead>
<tr>
<th>Thrombin Molecular Topology</th>
<th>Rectangular Periodic Box</th>
<th>Truncated Octahedron Periodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of solute atoms</td>
<td>3078</td>
<td>3078</td>
</tr>
<tr>
<td>Number of solvent atoms</td>
<td>32883</td>
<td>16281</td>
</tr>
<tr>
<td>Total number of atoms</td>
<td>35961</td>
<td>19359</td>
</tr>
<tr>
<td>Number of solute charge groups</td>
<td>1285</td>
<td>1285</td>
</tr>
<tr>
<td>Number of solvent charge groups</td>
<td>10961</td>
<td>5427</td>
</tr>
<tr>
<td>Total number of charge groups</td>
<td>12246</td>
<td>6712</td>
</tr>
<tr>
<td>Box dimensions</td>
<td>6.9 x 7.2 x 7.42 nm</td>
<td>7.4 x 7.4 x 7.4 nm</td>
</tr>
<tr>
<td>Cutoff range</td>
<td>1.4 nm</td>
<td>1.4 nm</td>
</tr>
<tr>
<td>Pairlist update</td>
<td>every 5th step</td>
<td>every 5th step</td>
</tr>
<tr>
<td>Total time steps</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

*Table 2.1: Thrombin molecular topology parameters for benchmarking*

All simulations run over 100 time steps, which gives a good average, how much time is used per single step. The calculations are done in 3 dimensions. Only one cutoff and no twin range is used. The Thrombin protein consists of the same number of atoms for both simulation boxes, whereas only the half of solvent atoms are used for the truncated octahedron periodic box, because of its smaller volume. Although the box dimension parameters are nearly the same for both, they describe only the profile length of the octahedral simulation box. Therefore the volume of the truncated octahedron is about half of that of the rectangular box because of its cut corners.

The Thrombin molecular simulation is further used for all benchmarks and profilings of the different GROMOS versions on the various tested hardware. This allows a continuity of our investigations and a good comparison of the gathered results.
2.3 GROMOS Analysis

2.3.2 Profiling

We made first a profiling of the GROMOS96 program using gprof to find the most time consuming programming routines. Gprof is an automatic profiler for C and Fortran programs and is enabled with a compiler option. In this case the compiler builds structures into the binary code of which the profiler extracts the runtime of the different programming routines.

We ran the Thrombin molecular simulation in a truncated octahedron on a SUN SPARC Station 10 with a 85 MHz CPU and 64 MB RAM under Solaris 2.5.1. The gprof output is shown in table 2.2:

| Granularity: each sample hit covers 2 byte(s) for 0.00% of 5591.13 seconds |
| % time | cumulative seconds | self seconds | calls | self s/call | total s/call | name |
| 79.73 | 4457.92 | 4457.92 | 100 | 44.58 | 44.89 | nonbml_ |
| 15.02 | 5297.74 | 839.82 | 20 | 41.99 | 41.99 | nbnone_ |
| 0.81 | 5424.02 | 45.19 | 200 | 0.23 | 0.23 | shake_ |
| 0.16 | 5515.04 | 8.82 | 542901 | 0.00 | 0.00 | cenmas_ |
| 0.10 | 5527.96 | 5.49 | 500 | 0.01 | 0.01 | dihang_ |
| 0.10 | 5533.33 | 9.37 | 100 | 0.05 | 0.51 | mdleap_ |
| 0.06 | 5544.87 | 3.23 | 200 | 0.02 | 0.02 | angle_ |
| 0.03 | 5560.43 | 1.85 | 100 | 0.02 | 0.12 | prpvir_ |
| 0.02 | 5567.96 | 1.06 | 1 | 1.06 | 5427.38 | runmd_ |
| 0.02 | 5568.92 | 0.96 | 100 | 0.01 | 0.01 | shiag_ |
| 0.01 | 5578.31 | 0.53 | 100 | 0.01 | 0.01 | conat_ |
| 0.01 | 5579.29 | 0.48 | 100 | 0.00 | 53.39 | force_ |
| 0.01 | 5582.14 | 0.33 | 31752 | 0.00 | 0.00 | chkmy_ |
| 0.00 | 5584.79 | 0.23 | 12 | 0.02 | 1.88 | dmpred_ |
| 0.00 | 5585.37 | 0.19 | 317700 | 0.00 | 0.00 | getdmp_ |
| 0.00 | 5586.72 | 0.15 | 20 | 0.01 | 0.01 | clggeo_ |
| 0.00 | 5587.75 | 0.12 | 37923 | 0.00 | 0.00 | gimme_ |
| 0.00 | 5587.87 | 0.12 | 19359 | 0.00 | 0.00 | chpnr_ |
| 0.00 | 5588.53 | 0.10 | 20 | 0.01 | 42.00 | nbpm_ |
| 0.00 | 5588.80 | 0.09 | 100 | 0.00 | 0.01 | dihmon_ |
| 0.00 | 5589.04 | 0.08 | 1 | 0.08 | 5.26 | gtoor_ |
| 0.00 | 5589.79 | 0.06 | 25515 | 0.00 | 0.00 | lismty_ |
| 0.00 | 5590.05 | 0.05 | 2 | 0.03 | 3.41 | dmparr_ |
| 0.00 | 5590.41 | 0.04 | 2 | 0.02 | 0.37 | gbnd_ |
| 0.00 | 5590.45 | 0.04 | 1 | 0.04 | 2.50 | gsluat_ |
| 0.00 | 5590.54 | 0.03 | 77828 | 0.00 | 0.00 | chpint_ |
| 0.00 | 5590.62 | 0.02 | 10199 | 0.00 | 0.00 | chpvel_ |
| 0.00 | 5590.68 | 0.02 | 1 | 0.02 | 4.47 | getarr_ |
| 0.00 | 5590.83 | 0.01 | 19359 | 0.00 | 0.00 | skpchr_ |
| 0.00 | 5590.94 | 0.01 | 200 | 0.00 | 0.01 | dihdis_ |

Table 2.2: GROMOS96 profiling with Gprof on a SUN SPARC Station 10 using the octahedral Thrombin molecular topology
Almost 95% are used for the subroutines nonbml_, which calculates the non-bonded interactions, and for nbnone_, which generates the pairlist without long-range interactions. The shake_ routine handles constraints, cenmas_ calculates the centre of mass and the kinetic energies, dihang_ calculates the dihedral angles, energies and forces and mdleap_ is for the integration of the partial forces. All other tasks do not really matter.

The profiler has a major drawback. It introduces timing routines for measuring the time for all tasks. These routines require time itself and do slightly distort the real simulation time. A further drawback is, that the profiler is not available or uses other methods on other computer platforms, which makes it difficult to compare the results of the simulations on different hardware.

Therefore we introduced our own timing measurement routines into the source code of GROMOS96, which profile only the most substantial and time consuming parts, namely the pairlist, the interaction and the integration routine. Our timing routines give more information about the runtimes of the regarding parts and is widely independent of the hardware.

A further simulation of the Thrombin molecular topology using our timing routines in GROMOS96 on the same SPARC Station 10, delivered the following results:

<table>
<thead>
<tr>
<th>Programming Routine</th>
<th>Rectangular Simulation [s]</th>
<th>Octahedral Simulation [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairlist</td>
<td>2171.72</td>
<td>824.63</td>
</tr>
<tr>
<td>Bonded Interactions (solute-solute)</td>
<td>14.49</td>
<td>15.54</td>
</tr>
<tr>
<td>Non-bonded Interactions (solute-solute, solute-solvent)</td>
<td>3069.71</td>
<td>1065.10</td>
</tr>
<tr>
<td>Non-bonded Interactions (solvent-solvent)</td>
<td>6842.20</td>
<td>3380.92</td>
</tr>
<tr>
<td>Non-bonded Interactions (all)</td>
<td>9911.91</td>
<td>4446.02</td>
</tr>
<tr>
<td>Integration</td>
<td>106.58</td>
<td>54.78</td>
</tr>
<tr>
<td>Other</td>
<td>74.31</td>
<td>38.01</td>
</tr>
<tr>
<td>Total time</td>
<td>12279.01</td>
<td>5378.98</td>
</tr>
</tbody>
</table>

*Table 2.3: GROMOS96 profiling with timing measurement routines on a SUN SPARC Station 10 using the Thrombin molecular topologies*
The measured times in table 2.3 are shorter than that, which we have made in the profiling with Gprof (see table 2.2). This verifies our statement, that the Gprof timing measurement routines increase the simulation time.

The rectangular simulation requires twice the time of the octahedral simulation. Due to the number of solvent atoms, which is about the half in the octahedral simulation, the number of solvent-solvent interactions and therefore also the time of its calculation is only half. This affects also the times of the pairlist generation and the non-bonded interaction calculation of the solute atoms. Their times amount to one third, because of its quadratic dependence of the total number of atoms, which differ about a factor of 1.8 in both simulations. Also the time of the integration is dependent on the number of atoms and halves in the octahedral simulation. Only the time of the bonded interaction calculation remains the same because bonded interactions are only between solute atoms, whose number is the same for both simulations.

We are interested, how much percent of the total simulation time is used by each part of GROMOS96 and accordingly, which part predominates the simulation time. Figure 2.15 shows the percentage of the same routines used in table 2.3

![Figure 2.15: Percentage of the Thrombin simulation times by GROMOS96 routines](image)

The calculation of the non-bonded interactions uses about 80% of the time in both simulations, where the non-bonded solvent-solvent interactions (token 'vv') uses 2 - 3 times more than the non-bonded interactions with solute (tokens 'ss', 'sv'). The other routine, which is also time dominant, is the pairlist generation with a percentage of 15 - 18%. All other routines, the integration and especially the calculation of the bonded interactions uses so little time, that they do not really affect the simulation time. Therefore the pairlist generation and the non-
bonded interaction calculation cover up to 98% of the total simulation time, whereof the shortest and most trivial routines, which are the pairlist generation and the non-bonded solvent-solvent interaction calculation, comprise still 75 - 80%.

### 2.3.3 Function Modelling

The number of floating-point and fixed-point operations in dependence of the molecular topology parameters gives an information about the function complexity if the number of these operations are relatively high in comparison with other commands, such as memory accesses or branches. With the number of arithmetic operations an estimation of the simulation time can be made for computer hardware, whose performance specifications are available. Otherwise the performance of a hardware for MD simulations can be given if the respective execution time is known. This allows to compare the performance of different hardware solutions for MD simulations.

In the following, we make an analysis and a function model of the pairlist and the non-bonded solvent-solvent interaction routines. Both routines use practically only floating-point operations and even few memory accesses and branches. As they are the most time consuming routines in GROMOS96, the number of floating-point operations of these routines should give a good information about the function complexity of GROMOS for estimating the performance of different hardware solutions.

#### 2.3.3.1 Distance Calculation

Both, the pairlist routine and the non-bonded solvent-solvent force routine, use the distance calculation between two atoms or charge groups respectively. The distance calculation uses only floating-point operations and also some conditional branches, whose comparisons use also floating-point operations, which are listed by the equal sign in the following tables. Table 2.4 shows the number of floating-point operations for the distance calculation between two particles for the different periodic boundary conditions.
### 2.3 GROMOS Analysis

<table>
<thead>
<tr>
<th>Op.</th>
<th>Vacuum</th>
<th>Rectangular</th>
<th>Octahedral</th>
<th>Monoclinic</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/-</td>
<td>$2 \cdot NDIM + NDIM \cdot P_{rbc} + 1 \cdot 4D$</td>
<td>$2 \cdot NDIM + NDIM \cdot P_{rbc} + 1 \cdot 4D$</td>
<td>$3 \cdot NDIM + NDIM \cdot P_{rbc} + (NDIM + 1) \cdot P_{obc} + 1 \cdot FC \cdot P_{obc} + 1 \cdot 4D$</td>
<td>$2 \cdot NDIM + NDIM \cdot P_{rbc} + 1 \cdot 4D + 1$</td>
</tr>
<tr>
<td>*</td>
<td>$NDIM + 1 \cdot 4D$</td>
<td>$NDIM + 1 \cdot 4D$</td>
<td>$NDIM + 1 \cdot P_{obc} + 1 \cdot FC \cdot P_{obc} + 1 \cdot 4D$</td>
<td>$NDIM + 1 \cdot 4D + 2$</td>
</tr>
<tr>
<td>/</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>=</td>
<td>0</td>
<td>$NDIM$</td>
<td>$NDIM + 1$</td>
<td>$NDIM$</td>
</tr>
</tbody>
</table>

| Table 2.4: Number of floating-point operations to calculate the distance of two particles |

The different parameters in table 2.4 are used as follows:

- **NDIM**: The number of dimensions to calculate in, which is three or four
- **4D**: =1 in four-dimensional and =0 in three-dimensional simulations
- **FC**: =1 if distance calculation is for the force algorithm else =0
- **$P_{rbc}$**: The probability for rectangular periodic boundary correction if one of the coordinates is not in the periodic box
- **$P_{obc}$**: The probability for additional octahedral boundary correction

The parameter **NDIM** is ever three and **4D** is equals zero in the pairlist generation also for four-dimensional simulations. The parameters for the probability of the periodic boundary corrections $P_{rbc}$ and $P_{obc}$ are difficult to derive. They are dependent of the number of particles on the borders of the periodic box and may be determined most exactly by running a simulation.
2.3.3.2 Pairlist Generation

The pairlist generation can be separated into two parts. First the geometric centre of the solute charge groups is calculated and then the distances between all charge groups are evaluated to generate the pairlist. Note, that for the solvent molecules, which consist each of only one charge group, the first atom is assumed to be the largest and therefore the position of this atom is taken for the pairlist generation. Thus the subroutine for the geometric centre has also a lot of copy and fixed point operations, but the number of floating-point operations remains still dominant.

The number of floating-point operations for the calculation of the geometric centre of one charge group are summarized in table 2.5.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Geometric Centre</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/-</td>
<td>( \frac{NAT_{su}}{CG_{su}} \cdot NDIM \cdot P_{acg} )</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>/</td>
<td>NDIM \cdot P_{acg}</td>
</tr>
</tbody>
</table>

*Table 2.5: Number of floating-point operations to calculate the geometric centre of a CG*

The parameters in table 2.5 are used as:

- \( NAT_{su} \): Number of solute atoms
- \( CG_{su} \): Number of solute charge groups
- \( P_{acg} \): The probability of a charge group with more than one atom

The equations in table 2.5 deliver the average number of floating-point operations for calculating the geometric centre of any solute charge group. For determining the exact number of floating-point operations for one certain charge group, the term \( \frac{NAT_{su}}{CG_{su}} \), which delivers the average number of atoms in a solute charge group for the whole simulation, should be replaced by the exact number of atoms of the respective charge group. In this case, the parameter \( P_{acg} \) is equals one if the charge group consists of more than one atom else \( P_{acg} \) is equals zero because the position of this one atom is taken as charge group position and the subroutine uses no floating-point operations.
The number of floating-point operations for the distance calculation can be taken from table 2.4. Both tables deliver the number of floating-point operations for only one charge group or charge group pair respectively. For the whole pairlist generation, the results of tables 2.4 and 2.5 must be multiplied with the number of loops over the appropriate subroutines, which is dependent of the number of particles in the simulation. Thus the total number of floating-point operations for the whole pairlist generation is

\[ OP_{pg} = CG_{su} \cdot OP_{cg} + n \cdot OP_{dcp}, \]  

(2.13)

where \( CG_{su} \) is the number of solute charge groups and \( n \) is the number of distance calculations of all charge group pairs, which is taken from equation 2.8 on page 49. The parameter \( OP_{cg} \) is the result from table 2.5 and \( OP_{dcp} \) is from table 2.4.

### 2.3.3.3 Non-bonded Solvent-Solvent Interaction Calculation

The non-bonded solvent-solvent interaction routine consists of the distance calculation and the calculation of the forces, energies and the virial. The whole routine has practically only floating-point operations. Conditional branches, which require floating-point operations, are used in the distance calculation. Additionally some few memory operations are used.

Table 2.6 shows the number of floating-point operations for the calculation of one interaction between two solvent atoms, which is only different if virial or no virial is calculated. It is independent of the structure of the simulation box.

<table>
<thead>
<tr>
<th>Op.</th>
<th>No Virial</th>
<th>With Virial</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/-</td>
<td>2 \cdot NDIM +8</td>
<td>2 \cdot NDIM +17</td>
</tr>
<tr>
<td>*</td>
<td>NDIM +13</td>
<td>NDIM +19</td>
</tr>
<tr>
<td>/</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \sqrt{\cdot} )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 2.6: Number of floating-point operations to calculate the interactions between two atoms*
The results of table 2.6 are only dependent of the number of dimensions \( \text{NDIM} \) of the simulation and deliver the number of floating-point operations from the calculation of the forces, energies and the virial, but without considering the distance calculation, whose numbers of floating-point operations are already assumed in table 2.4.

To obtain the total number of floating-point operations of the non-bonded solvent-solvent interaction routine for one time step, the number of floating-point operations of the distance calculation (see table 2.4) and the force, energies and virial calculation (see table 2.6) have to be added up and then multiplied with the number of atom-atom interactions between all charge group pairs, which are listed in the pairlist. This results in

\[
\text{OP}_{nnv} = \text{NN}_{vv} \cdot n_a \cdot (\text{OP}_{dci} + \text{OP}_{ivv}),
\]

where \( \text{NN}_{vv} \) is the number of charge group pairs in the pairlist and can be obtained with equation 2.11 on page 50 or by running a simulation. The parameter \( n_a \) represents the number of atom-atom interactions of one solvent charge group pair and can be obtained with \( n_a = (\text{NAT}_{sv} / \text{CG}_{sv})^2 \), where \( \text{NAT}_{sv} \) is the number of solvent atoms and \( \text{CG}_{sv} \) the number of solvent charge groups. The parameters \( \text{OP}_{dci} \) and \( \text{OP}_{ivv} \) are the results of the tables 2.4 and 2.6 respectively.

### 2.3.3.4 Floating-point Operations for the Thrombin Simulation

The function model, which we have made in the previous chapters, allows to estimate the function complexity for any GROMOS simulation. Based on the parameters of the simulation, the number of floating-point operations, which are processed in the pairlist and in the non-bonded solvent-solvent interaction routine, can be determined. This allows, together with the runtimes of these routines, for an estimation of the performance of a computer for the simulation.

The performance in floating-point operations per seconds \( \text{OPS} \) is defined as

\[
\text{OPS} = \frac{\text{OP}}{t}
\]

where \( \text{OP} \) can be obtained from equations 2.13 and 2.14 and \( t \) is the measured runtime of the routines.

In the following, we determine the number of floating-point operations for the Thrombin simulation by using tables 2.4 - 2.6 and equations 2.13 and 2.14. We can use it to derive the performance of different computer hardware with equation 2.15 to compare its computing power in our hardware evaluations.
The required parameters for evaluating the number of floating-point operations in the Thrombin simulations are defined in table 2.7:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAT&lt;sub&gt;su&lt;/sub&gt;</td>
<td>3078</td>
<td>3078</td>
</tr>
<tr>
<td>NAT&lt;sub&gt;sv&lt;/sub&gt;</td>
<td>32883</td>
<td>16281</td>
</tr>
<tr>
<td>CG&lt;sub&gt;su&lt;/sub&gt;</td>
<td>1285</td>
<td>1285</td>
</tr>
<tr>
<td>CG&lt;sub&gt;sv&lt;/sub&gt;</td>
<td>10961</td>
<td>5427</td>
</tr>
<tr>
<td>CG</td>
<td>12246</td>
<td>6712</td>
</tr>
<tr>
<td>V&lt;sub&gt;box&lt;/sub&gt;</td>
<td>368.63</td>
<td>202.61</td>
</tr>
<tr>
<td>r&lt;sub&gt;c&lt;/sub&gt;</td>
<td>1.4</td>
<td>1.4</td>
</tr>
<tr>
<td>N&lt;sub&gt;DIM&lt;/sub&gt;</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4D</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P&lt;sub&gt;rbc&lt;/sub&gt;</td>
<td>0.761</td>
<td>0.353</td>
</tr>
<tr>
<td>P&lt;sub&gt;obc&lt;/sub&gt;</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>P&lt;sub&gt;acg&lt;/sub&gt;</td>
<td>0.927</td>
<td>0.927</td>
</tr>
</tbody>
</table>

Table 2.7: Definition of the parameters for evaluating the number of floating-point operations during the Thrombin simulation

We have determined the values P<sub>rbc</sub>, P<sub>obc</sub> and P<sub>acg</sub> by running the appropriate Thrombin simulation by introducing variables in the code for counting the regarding paths of the program. We see, that for about 75% of the distance calculations a periodic boundary correction is required. In the octahedral simulation, the corrections are equally divided to rectangular and octahedral corrections.

Table 2.8 shows the average number of floating-point operations for one run of the programming routines. For determining the computer performance, the total number of floating-point operations of one time step or the whole simulation is necessary, which can be derived of equations 2.13 and 2.14.
Table 2.8: Average number of floating-point operations for one run of the programming routines for the Thrombin simulation.

For the pairlist generation, we obtained the required parameter $n$ from equation 2.6 as $n = 74976135$ for the rectangular and $n = 22522116$ for the octahedral Thrombin simulation. The parameter $NN_{yv}$ can be either derived from equation 2.11 or by running a simulation. Although equation 2.11 delivers a quite sufficient result (see section 2.2.3.2 on page 50), we used a simulation for determining $NN_{yv}$ to have a more exact result for our investigations. Thus we obtained $NN_{yv} = 1981463$ (1873063) for the rectangular and $NN_{yv} = 916439$ (835414) for the octahedral simulation. The values in brackets are derived from equation 2.11. As the solvent charge groups consist of each 3 atoms, the number of atom-atom interactions per charge group pair amounts to $n_{a} = 9$.

Table 2.9 shows the average number of floating-point operations for one time step of the Thrombin simulation.
Table 2.9: Average number of floating-point operations for one time step of the Thrombin simulation

Below the total number of operations the ratio of the different floating-point operations is given in percent-quota as (addition, subtraction/multiplication/division). We have taken the comparison operations for the conditional branches as additions/subtractions and we have counted the square root to the divisions.

The ratio helps to estimate the simulation time on a computer, whose floating-point performance is given. As a processor does not need the same number of clock cycles for the different floating-point operations, its performance values vary if it is tested only with additions or only with multiplications. Therefore a mix of the different operations: addition/subtraction, multiplication and division is commonly used for testing. Considering the quota of each floating-point operation is not the same in the various applications, but often additions and subtractions are the most used operations, followed by multiplications and at last divisions, a ratio is defined for testing the performance of a computer, which covers the ratio of the most applications. Combining the ratio of an application with the ratio of the performance test on a computer, its performance for that application can be estimated.
Table 2.10 shows the floating-point performance of the SUN SPARC Station 10 for the pairlist generation and the non-bonded solvent-solvent interaction calculation. We evaluated the results with equation 2.15 by using the runtimes of these routines from the profiling of the Thrombin simulation (see table 2.3) and the number of floating-point operations (see table 2.9), which had been multiplicated with the total number of time steps, 100 in our case.

<table>
<thead>
<tr>
<th>Programming Routine</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairlist generation</td>
<td>9.86 MFlops</td>
<td>10.41 MFlops</td>
</tr>
<tr>
<td>Non-bonded solvent-solvent interaction calculation</td>
<td>12.06 MFlops</td>
<td>16.31 MFlops</td>
</tr>
</tbody>
</table>

*Table 2.10: Performance of a SUN SPARC Station 10 for the Thrombin simulation*

The SPARC 10 reaches the higher performance for the non-bonded interaction calculation than for the pairlist generation. The reason is, that in the interaction calculation more operations have to be done for fewer particles, which causes fewer memory operations. The reached performance of the octahedral simulation is a little higher than that of the rectangular, due to the higher calculation effort and the fewer particles, used in the octahedral simulation, which requires fewer memory accesses.

### 2.4 Concept for Enhancing the Performance of GROMOS

The profiling in section 2.3.2 has shown, that the most time intensive parts of GROMOS are short and trivial programming routines, which cover up to 98% of the simulation time. This offers the use of a dedicated hardware, which treats only the time intensive programming routines of GROMOS. All other parts can be processed on a host computer, which may be a standard workstation, because these program parts are not time critical. In this case the dedicated hardware is used as coprocessor, which is attached to a standard workstation. As only the short and trivial programming routines have to be implemented on the coprocessor hardware and no changes are required for the most of the GROMOS routines, since they can be run on a standard workstation, this solution offers the most effective way for accelerating the MD simulation.

The programming routines, which are sourced out on the coprocessor are taken
out of the program flow. This requires an exchange of data between the host and the coprocessor, because the input and output data of the outsourced programming routines, which had been processed internally on the workstation so far, are now used by the coprocessor. The principle of the data handling and outsourcing of the program routine is shown in figure 2.16:

The handling of the data is dependent on the manner of the use of the data in the MD simulations. There can be distinguished between constants, static parameters and dynamic data. Constants are data, which are the same for all MD simulations, e.g. the van der Waals parameters $C_6$ and $C_{12}$. The static parameters are those, which are defined before the simulation starts and remain the same during the simulation, e.g. the number of molecules, size of the simulation box and cutoff length. These two types of data have to be sent to the coprocessor only once per simulation. However the dynamic data are those, which change at each call of the coprocessor routines, e.g. the coordinates of the atoms or the pairlist. They have to be sent to the coprocessor each time before these routines are called. The coprocessor must send back the results, which are the calculated forces, the energies and the virial, after processing the routines.

The computational accuracy on the coprocessor must be at least the same as that on the host, which is a standard workstation. This means, that the coprocessor must process the calculations in 32-bit or wider floating-point format. Although a 32-bit fixed-point format is more accurate, a 32-bit floating-point format is preferred for the coprocessor, because the number range in 32-bit fixed-point format is too small.
Figure 2.17: Data flow and dependency of the GROMOS programming routines
for the calculations in GROMOS and a conversion of the data format between host and coprocessor can be avoided.

There are different possibilities, which and how many of the GROMOS routines may be sourced out on a coprocessor. The more programming routines are sourced out, the higher performance can be achieved, but this requires also the higher implementation effort, whereas it is dependent of the used coprocessor hardware. The outsourcing determines also the amount of communication data, which has been exchanged between the host and the coprocessor in each time step. In choosing the program routines for the coprocessor, this must be taken into account, that the communication effort does not become too high.

Figure 2.17 shows the dependency of the programming routines in GROMOS and the data packets, which are exchanged between them. The gray shaded frames illustrate the possible routines, which may be sourced out on the coprocessor. The data packets, which lie on the edges of a gray shaded frame, have to be communicated between the host and the coprocessor.

There are substantially four variants for outsourcing programming routines of GROMOS on the coprocessor. Table 2.11 lists these variants with the information of the percentage of the simulation time, which can be accelerated and an estimation of the communication data between the host and the coprocessor for one time step of the rectangular Thrombin simulation.

<table>
<thead>
<tr>
<th>Shaded Frame</th>
<th>Outsourced Programming Routines</th>
<th>Percentage of Acceleration</th>
<th>Communication Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Non-bonded interaction calculation (vv)</td>
<td>55 - 60%</td>
<td>770.69 KB +7316.65 KB</td>
</tr>
<tr>
<td></td>
<td>• Pairlist generation (vv) • Non-bonded interaction calculation (vv)</td>
<td>70%</td>
<td>770.69 KB</td>
</tr>
<tr>
<td></td>
<td>• Pairlist generation (ss,sv,vv) • Non-bonded interaction calculation (vv)</td>
<td>75 - 80%</td>
<td>806.77 KB +3263.34 KB</td>
</tr>
<tr>
<td></td>
<td>• Pairlist generation (ss,sv,vv) • Non-bonded interaction calculation (ss,sv,vv)</td>
<td>98%</td>
<td>842.84 KB</td>
</tr>
</tbody>
</table>

*Table 2.11: Outsourcing of programming routines with percentage of acceleration and amount of communication data between host and coprocessor*
The amount of communication data has been obtained with the parameters in table 2.1. To estimate the size of the pairlist data, we used equations 2.9 - 2.12 on page 50. For all data, we assumed a 32 bit data format.

The values in italics are the data of the pairlist, which add to the total communication data, when the pairlist is sent between host and coprocessor. If the pairlist is stored on the machines, where the non-bonded interactions are calculated, communication time could be saved, because the pairlist data should only be sent every 5th-20th time step, when the pairlist is generated.

The implementation effort raises from the first to the last variant, whereas the percentage of the simulation time, which can be accelerated, grows. Concerning the amount of communication data, it is recommendable to source out those parts of the pairlist generation, which treat the particles of the outsourced interaction calculation routine, to avoid a transmission of the pairlist from the host to the coprocessor, which is quite large.

The first variant, which would require the lowest implementation effort, has a large amount of data to be communicated and a low percentage of acceleration. A better solution is the second variant, where the pairlist generation and the non-bonded interaction calculation, both for solvent, are sourced out. In this case, fewer data have to be communicated and a higher percentage of acceleration is reached for a higher implementation effort. An outsourcing of the whole pairlist generation and the non-bonded interaction calculation for all atoms requires the highest implementation effort with an acceptable communication, but it may reach the highest yield in accelerating 98% of the simulation time.

2.5 Conclusion

We have seen, that the most effective way in accelerating GROMOS is the use of a coprocessor, which processes only the most time consuming parts of the program. As these programming routines are short and trivial, the implementation effort on the coprocessor would not be too high.

The choice of the programming routines for outsourcing is dependent of the coprocessor hard- or software language and the resulting communication. Implementing more routines on the coprocessor would result in a higher acceleration of the simulation. If the coprocessor could be programmed with a standard language as Fortran or C, all time consuming parts may be sourced out with a low effort, as only few changes would be required for the program code.
Hardware Approaches

As there exist many approaches to realize a coprocessor for accelerating a software algorithm, an evaluation of different hardware solutions was made to find the most adequate one for outsourcing GROMOS. First a general overview of the characteristics of different hardware solutions is given, followed by an evaluation and a test of their suitability for the GROMOS routines, which are planned to be sourced out to the coprocessor. For the existing MD hardware and parallel computers, an estimation based on already implemented MD algorithms is made. An evaluation model was used for FPGAs and ASICs. The performance on DSPs has been evaluated by implementing the most time consuming GROMOS routines and the MPUs were tested by compiling and running GROMOS96. The chapter closes with a comparison of the evaluated hardware solutions.
3 Hardware Approaches

3.1 General Overview of Hardware Solutions

Possible hardware solutions for implementing a software algorithm reach from ASICs to MPUs. To find the suitable solution, we have to consider the three characteristics of the hardware: speed, flexibility and implementation effort. A higher speed leads to a better performance of the software, more flexibility allows a simpler adaptation of the hardware for changes and new features of the software, and a lower effort reduces the time for implementing and adapting the software to the hardware.

Figure 3.1 shows the general characteristics of speed and flexibility concerning the different hardware solutions.

![Speed and flexibility of different hardware solutions](image)

Speed

- fast
- slow

ASIC FPGA DSP MPU

Flexibility

- low
- high

Figure 3.1: Speed and flexibility of different hardware solutions

Speed and flexibility are opposite to each other. Where an ASIC offers the highest speed for executing an algorithm, it is quite inflexible for software updates. On the other side an MPU is very flexible for software changes, because only a new compilation is necessary, but the MPU is the slowest of all hardware solutions.

For the implementation effort no general characteristic exists. It depends on the programming language of the software source code. In table 3.1 is a list of the programming languages and development tools for each hardware solution:
Concerning GROMOS, which is written in Fortran code, the effort has the same characteristic as the flexibility. For the MPU a standard Fortran compiler can be used, and for the DSP a C compiler can be used, whereby the transfer from Fortran to C is relatively simple. The effort is much higher for an implementation on a FPGA or an ASIC, because the Fortran code has to be transferred to a hardware programming language.

Due to the opposite characteristics of speed in comparison with flexibility and effort, we have to find the best compromise between these three characteristics.

### 3.2 Existing MD Hardware

We investigated some hardware solutions for the molecular dynamics problem to find out if one of them may be suitable for the GROMOS coprocessor. The most of them are systems developed under the GRAPE\(^\dagger\) project. The systems reach from special-purpose processors on single boards up to massively parallel special-purpose computers. Others are GROMACS, a transputer-ring and MD-Engine, a special-purpose parallel machine. All of them are described in more detail in the following sections.

\(^\dagger\) GRAPE for GRavity PipE
3.2 Existing MD Hardware

3.2.1 The GRAPE Family

3.2.1.1 Introduction

The GRAPE family is a series of special-purpose computers for the simulation of N-body systems, developed by the Department of Earth Science and Astronomy together with the Department of Information Science and Graphics of the College of Arts and Sciences at the University of Tokyo.

GRAPE is a coprocessor attached to a general purpose computer and is specialized to calculate the forces between the particles of a N-body system. All other computations, such as time integration of orbits etc. are performed on the host computer [33]. In the simplest case, the host computer sends positions and masses of all particles to GRAPE. Then GRAPE calculates the forces between particles and sends them back to the host computer, which updates the positions, velocities, etc. of the particles using the received forces (see figure 3.2).

![Diagram](#)

**Figure 3.2: Basic structure of the GRAPE systems**

The first GRAPE systems were developed for the acceleration of gravitational N-body simulations of astrophysics, where the forces and motions of particles and point masses are calculated by Newton’s equation of motion (eq. 1.1) and Newton’s gravitational force (eq. 1.2).

GRAPE has a pipeline, which embodies the data flow for the force calculation [34]. It reaches a rate of one interaction per cycle. Multiple pipelines can be attached to one memory unit so that they can calculate the force on different particles (see figure 3.3).
The parameters $r_i$ and $r_j$ are the coordinates for the particles $i,j$, $m$ their masses and $\varepsilon$ is a softening parameter to avoid any divergence of the force for small distances.

There exist already eleven different GRAPE machines, which are numbered from 1 to 6. They are divided into low-accuracy and high-accuracy types. The machines with the odd numbers are among the low-accuracy types and had been designed for simulations of collisionless systems, such as galaxies, in which only the mean potential plays a role. The machines with the even numbers are among the high accuracy types and had been designed for collisional systems, such as globular clusters and proto-planetary systems, in which the forces have to be calculated accurately. Some machines of the high-accuracy type contain user specified functions and had been designed for MD simulations.
We can see in figure 3.4, that all machines have its origin from GRAPE-1, the first version of the GRAPE family. Each machine is a further development of its previous machine, where new features and functions were added. The machines are described in more detail in the following sections.
3.2.1.2 GRAPE-1 and GRAPE-1A

GRAPE-1 was the first machine, which was built in 1989. It calculates the gravitational force (eq. 1.2) with equivalent masses \( m_i = m_j \) (see figure 3.5):

![GRAPE-1 system diagram]

The coordinate \( r_i \), at which the force is calculated is stored in registers \( x_i, y_i \) and \( z_i \). The coordinates of the particles \( r_j \) are stored in memory units, denoted as \( x_j, y_j \) and \( z_j \). During the calculation a counter supplies the particle index \( j \) to the memory unit, which outputs the coordinates of the particle to the force calculation pipeline unit. This pipeline has six stages, where one force is accumulated at each clock period. Thus the total force for particle \( i \) from all other particles is calculated in \( N+6 \) clock periods.

In GRAPE-1 the subtraction of the positions are performed in 16-bit fixed-point format and the accumulation of the force is performed in 48-bit fixed-point format. All other calculations are performed in an 8-bit logarithmic format by using a table-lookup procedure.
3.2 Existing MD Hardware

GRAPE-1A is an improved version of GRAPE-1 and was developed in 1990. It can handle particles with different masses and has circuits, which can search for neighbouring particles. A VME bus interface was used as the interface between the host computer and GRAPE-1A instead of the GPIB interface of GRAPE-1. Both machines work with a clock frequency of 8 MHz and the peak speed is 240 MFlops [33].

3.2.1.3 GRAPE-3 and GRAPE-3A

GRAPE-3 was developed in 1991. It is the improved version of GRAPE-1 with 24 force calculation pipelines (see figure 3.6).

![Figure 3.6: GRAPE-3 processor board](image)

The pipelines share one common memory unit, which broadcasts the positions and masses of the particles to all pipelines, that exert the gravitational force. Different pipelines calculate the force on different particles [40]. Each of the GRAPE-3 board has a VME interface. Multiple boards can be attached to a single VME bus.

For GRAPE-3 a custom LSI chip (see figure 3.7) was developed in order to construct a highly parallel system [37].
The structure and the force calculation pipeline of the GRAPE chip are adopted from GRAPE-1A. Additionally the potential of the particles can be calculated and a neighbor flag is generated if the distance $r_{ij}$ is smaller than a given distance $h_i$. The GRAPE chip can calculate one gravitational interaction during each clock cycle. The subtraction of the forces is performed in 20-bit fixed-point format and the accumulation of the force and potential is performed in 56-bit fixed-point format. All other calculations are performed according to a 12-bit logarithmic format.

The memory unit stores the positions $r_j$ and masses $m_j$ of total 32768 particles. For larger simulations, the particles have to be divided into groups, each not exceeding 32768 particles, and the interactions of each group have to be calculated sequentially.

The neighbor list unit stores the numbers of particles, whose distance $r_{ij}$ is less than a given value $h_i$. The value $h_i$ is sent to each GRAPE chip before the force calculation is started. When any of the 24 GRAPE chips asserts the neighbor flag $NB$, the chip number corresponding to $i$ and the address of the memory corresponding to $j$ are written to a FIFO memory, which is read from the host computer after the force calculation of particle $i$ is finished.

One GRAPE chip running at 10 MHz has a computing speed of 300 MFlops. The GRAPE-3 system was built with two boards and has therefore 48 force calculation pipelines. It achieves a computing speed of about 14.4 GFlops.
3.2 Existing MD Hardware

GRAPe-3A is a version of GRAPe-3 with only 8 pipelines. It was built in order to test the GRAPe chip at a higher clock rate. At a clock speed of 20 MHz it achieves a computing speed of 4.8 GFlops [37].

3.2.1.4 GRAPe-5

GRAPe-5 is the successor of GRAPe-3 and was built in 1998. In addition to the $O(N^2)$ direct summation algorithm of the gravitational force (eq. 1.2) it can be applied to the Barnes-Hut tree algorithm and the $P^3M$ (particle-particle particle-mesh) or the Ewald method (see sections 1.2.3.2 and 1.2.3.3). The GRAPe-5 processor board is shown in figure 3.8.

![Figure 3.8: GRAPe-5 processor board](image)

The $O(N^2)$ direct summation is the simplest algorithm and efficient for small-N systems. For large N-body systems the force calculation becomes expensive, even with GRAPe.

The Barnes-Hut tree algorithm reduce calculation cost from $N^2$ to $N \log N$ by replacing forces from distant particles by that from particle at their centre of mass. A vectorized version of the tree algorithm based on Barnes’ modified algorithm [38] is used for GRAPe, that GRAPe can be used more efficiently than with the original algorithm.

The mostly used algorithms for the force calculation under periodic boundary conditions are the $P^3M$ method and the Ewald method. The PP force is not a pure $1/r^2$ force but with cutoff. On GRAPe-3 it was implemented with a linear combi-
nation of $1/r^2$ forces based on several approximations as this had been done by Brieu [39]. This results in a rather large loss of accuracy and performance. The GRAPE-5 hardware can handle force laws other than $1/r^2$ and calculate it in a single call using a programmable cutoff table. The PM force is calculated on the host computer.

The GRAPE-5 board (see figure 3.8) consists of 8 G5 chips, a memory unit, a particle index unit, a neighbor list unit and an interface unit. It is connected via Hlink to a PCI Host Interface Board, which is itself attached to the PCI bus of the host computer.

The G5 chip calculates the force $f_{ij}$ exerted on particle $i$ at position $r_i$ with equations 1.44 and 1.45 (see pages 18 and 19) as well as the potential $\phi_{ij}$. It consists of two pipeline units and one I/O unit. The pipeline unit calculates the gravitational interaction whereas the I/O handles the data transfer between the pipeline unit and the external I/O port. Figure 3.9 shows the block diagram of the pipeline unit:

---

Figure 3.9: Pipeline unit of the G5 chip
The position vector $r_i$, the softening parameter $\varepsilon$ and the neighbor radius $h_i$ are stored in the on-chip register before the calculation is started. During the calculation, the position vector $r_j$ and the mass $m_j$ are supplied from the memory unit (see figure 3.8). The pipeline unit calculates one interaction per clock cycle and accumulates it to on-chip registers. The function generator calculates $1/g(r_s, ij/\eta)$ and $1/g_0(r_s, ij/\eta)$ from $r_s^2, ij$ and $\eta^2$ using lookup tables, which are implemented as on-chip RAM blocks.

The G5 chip has the virtual multiple pipeline architecture to reduce the bandwidth of data transfer between the memory unit and the pipeline units. In this architecture one real pipeline acts as multiple pipelines using multiple on-chip registers. The G5 chip has 6 virtual pipelines per real pipeline unit. This means, that each real pipeline unit stores the positions of six position vectors $r_i$ and their parameters in their corresponding on-chip registers, and has therefore six accumulation registers for the components of the forces and the potentials. Thus the real pipeline calculates the force exerted on 6 different particles $i$, whereas the data for particle $j$ is used for 6 clock cycles. The partial forces $f_{ij}$ are evaluated in the following order:

$$f_{11}, f_{21}, f_{31}, f_{41}, f_{51}, f_{61}, f_{12}, f_{22}, f_{32}, f_{42}, f_{52}, f_{62}, \ldots$$

The memory unit operates at a clock frequency $1/6$ of that of the G5 chip, which simplifies the board design.

Each component of the position vectors $r_i$ and $r_j$ is stored in 32-bit fixed-point format. The subtraction of the positions is also performed in 32-bit fixed-point format and the accumulation of the force and potential is performed in 64-bit and 50-bit fixed-point format. For all other calculations a 17-bit logarithmic format is used. This format was preferred over the fixed-point format because of its larger dynamic range for the same word length and it was preferred over the usual floating-point format because operations as multiplication and square root, are easier to implement in logarithmic format. Format converters transform the values between the fixed-point and the logarithmic format after the subtraction of the positions and before the accumulation of the force and potential.
The particle index unit supplies the particle indices to the memory unit. It is optimised for the cell-index method (see section 2.1.2.2).

The particle index unit contains the cell-index memory and two counters: a cell counter, which is a 7-bit counter and a particle index counter, which is a 17-bit counter. The order of the particles is rearranged so that the particles of the same cell are stored in consecutive locations in the memory unit. The cell index memory supplies addresses and number of particles per cell. Initially, cell number $c$ is 0 and $j$ is set to $start[c]$, which points to the start address of the present cell. Then the data of the memory unit on which $j$ points are sent to the G5 chips and $j$ is incremented. These steps are repeated until $j$ reaches the address $start[c]+np[c]$. At this point $c$ is incremented by one and $j$ is set to $start[c]$, which points to the first address of the next cell. This procedure is repeated until $c$ reaches the maximum number of cells. In this way correct coordinates of particles are automatically sent to the G5 chips.

The memory unit supplies the position vectors $\mathbf{r}_j$ and the masses $m_j$ of the particles to the G5 chips according to the indices supplied by the particle index unit. It has four 4 Mbit (128 Kword x 32 bit) synchronous SRAM chips. Three of them are for storing the position vectors and one is for the masses. An amount of 131072 particles can be stored.

**Figure 3.10: Particle index unit**
The neighbor list unit stores the list of the nearest neighbours for particle \( i \). Particle \( j \) is the neighbour of particle \( i \) if \( r_{ij} < h_j \).

![Neighbor List Unit Diagram]

**Figure 3.11: Neighbor list unit**

There are two neighbor list units on the GRAPE-5 processor board. Each of them handles the neighbours of particles calculated on four G5 chips where the neighbor flags of 48 particles are set at every clock cycle of the board. These neighbor flags are stored with the corresponding particle index \( j \), that is given from the particle index unit, in the neighbor list memory if any of them is asserted. The host computer reads the data from the neighbor list memory after the force calculation is finished. The neighbor list memory is composed of two 1 Mbit (32 Kword x 32 bit) synchronous SRAMs and can hold up to 32768 neighbours for 48 particles. The 48 flag-bits and the lower 16 bits of the particle index \( j \) are stored in this memory and the MSB of the particle index \( j \), which changes only once during calculation, is stored in the MSB storage logic. If the MSB changes, the value of the internal address counter is stored in a register. The host can recover the MSB of the particle index using this register.

The particle index unit, the logic for the neighbor list units and the interface logic unit, which controls the communication between the host and the GRAPE-5 system, are each implemented in an Altera FPGA chip.

The G5 chip operates at 80 MHz and calculates \( 1.6 \times 10^8 \) interactions per second. If we count the calculation of the gravitational force as 30 floating-point operations, the peak performance of one G5 chip is equivalent to 4.8 GFlops. Thus the GRAPE-5 board with eight G5 chips achieves a computing speed of 38.4 GFlops [49].
3.2.1.5 GRAPE-2 and GRAPE-2A

GRAPE-2 was the first machine of the high accuracy type and was developed in 1990. It calculates the gravitational force (eq. 1.2) as GRAPE-1 does. In contrast to GRAPE-1, where three sub-pipelines, one for each of the components \( x, y, z \), were implemented, GRAPE-2 has only one sub-pipeline in order to reduce wiring complexity.

\[
\begin{align*}
\Delta x &= x_i - x_j \\
\Delta y &= y_i - y_j \\
\Delta z &= z_i - z_j
\end{align*}
\]

Counter

Particle
Memory

\( x_i, y_i, z_i \)

Converter

64 \rightarrow 32

Pipeline
Register

32 \times 32

\( f_{ijx}, f_{ijy}, f_{ijz} \)

\Sigma f_{ijx}, \Sigma f_{ijy}, \Sigma f_{ijz}

Mass
Memory

\( m_j \)

\( r_{ij}^2 = \Delta x^2 + \Delta y^2 + \Delta z^2 + \epsilon \)

\( m_j r_{ij}^3 \)

Figure 3.12: GRAPE-2 system

The positions of particle \( i \) are stored in a register and the positions of the particles \( j \) are stored in the particle memory in the order \( x_1, y_1, z_1, x_2, y_2, z_2, ..., x_n, y_n, z_n \). One value of positional data is sent out at each clock cycle. The mass of the corresponding particle \( j \) is stored in the mass memory and is sent out at each three clock cycles. Therefore one force \( f_{ij} \) between particle \( i \) and particle \( j \) is calculated in three clock cycles and is accumulated at the end of the pipeline.

The subtraction between the coordinates and the accumulation of the forces are performed in 64-bit floating-point format, while all other calculations are performed in 32-bit floating-point format. The size of the particle memory is \( 16K \times 64 \) bit and that of the mass memory is \( 16K \times 32 \) bit. A VME bus interface was used as the interface between the host computer and GRAPE-2.
GRAPE-2A is the second machine of the high accuracy type and was built in 1991. In addition to N-body simulations, it was also developed for MD simulations of proteins and crystals. GRAPE-2A is the improved version of GRAPE-2 regarding the following four points:

1. It has three sub-pipelines like GRAPE-1 had, but with 64-bit and 32-bit floating-point data format, which is equally organized as in GRAPE-2. Additionally the clock rate of GRAPE-2A was increased.
2. It is able to calculate the potential of particle $i$ with an additional function block.
3. It can make a list of all neighbours of a particle. This neighbor list is useful in various N-body simulations and can be used by the host for dealing with short-range forces.
4. It can calculate a force with an arbitrary functional form using an interpolation table. The contents of this RAM table is sent by the host before the calculation is started. With this, simulations with periodic boundary conditions can be calculated using the Ewald method (see section 1.2.3.3). GRAPE-2A can calculate the PP force with a Gaussian cutoff. The interpolation table makes it also possible to calculate the van der Waals and the Coulomb forces, which are used in MD simulations.

The block diagram of GRAPE-2A is similar as that of GRAPE-2 (see figure 3.12) with some changes. The register for the accumulation of the squares of the differences of the components $\Delta x$, $\Delta y$ and $\Delta z$, as well as its adder and the multiplicator for the squares of the differences have been replaced with a squared distance unit, which calculates the square of the distance $r_{ij}$ in parallel. The logic block, which converts $r_{ij}^2$ to $r_{ij}^{-3}$ and the succeeded multiplicator for the output and the mass have been replaced with the function evaluator unit, which is an interpolation table for calculating the different forces. At the output of the squared distance unit, additionally a neighbor list unit, which stores the indices $j$ of the particles, whose squared distances $r_{ij}^2$ are less than a given value $h_i^2$, and the potential accumulator unit for the potential of particle $i$, were connected.

GRAPE-2A is made of commercial floating-point chips. The coordinates of $r_j$ are stored in a 16K x 192 bit memory (six 16K x 32 bit SRAM modules). GRAPE-2A uses also a VME bus interface as the interface to the host computer.

GRAPE-2 works with a clock frequency of 4 MHz and the pipeline executes about 10 floating-point operations per clock cycle. Its peak speed is 40 MFlops [35]. GRAPE-2A is 4.5 times faster than GRAPE-2. The clock frequency is 6 MHz and one interaction is calculated at each clock cycle. The peak speed of GRAPE-2A is 180 MFlops [36].
3.2.1.6 GRAPE-4 and GRAPE-4A

GRAPE-4 has been developed between 1992-1995. It is a massively parallel computer of the high accuracy type and the successor of GRAPE-2. The system consists of 1692 processor chips, that each integrates about 15 floating-point arithmetic units and one function evaluator [42]. The system consists of four GRAPE-4 clusters. One cluster consists of a host-interface board, a control board and nine processor boards, thus the total number of processor boards is 36.

![Diagram of GRAPE-4 system]

In the community of astrophysics, the Aarseth scheme [46] has been widely used. It is a linear-multistep scheme, in which the change in the position and velocity is calculated from the accelerations of several previous time steps. The implementation of the Aarseth scheme is rather complicated, because the accelerations at previous time steps are not available, when the time integration is started and a special procedure to start up the integration is required [43].

Therefore in GRAPE-4 the Hermite scheme was implemented, which offers a similar accuracy as the Aarseth scheme [44]. It uses the Hermite interpolation formula to construct the predictor and the corrector. The corrector requires information of the present and the old time steps only. The Hermite scheme is a one-step, self starting scheme. It uses only the first time derivative of the acceleration explicitly calculated in order to construct the predictor and the corrector [47].
The predictor is expressed as

\[
\begin{align*}
\mathbf{r}_p &= \frac{\Delta t^3}{6} \mathbf{a}_0 + \frac{\Delta t^2}{2} \mathbf{a}_0 + \Delta t \mathbf{v}_0 + \mathbf{r}_0 \\
\mathbf{v}_p &= \frac{\Delta t^2}{2} \mathbf{a}_0 + \Delta t \mathbf{a}_0 + \mathbf{v}_0,
\end{align*}
\]  

(3.1) (3.2)

where \( \mathbf{r}_p \) and \( \mathbf{v}_p \) are the predicted positions and velocity, \( \mathbf{r}_0, \mathbf{v}_0, \mathbf{a}_0 \) and \( \hat{\mathbf{a}}_0 \) are the position, velocity, acceleration and its time derivative at time \( t_0 \), and \( \Delta t \) is the time step.

The acceleration \( \mathbf{a} \) and its time derivative \( \hat{\mathbf{a}} \) are calculated as

\[
\begin{align*}
\mathbf{a}_i &= \sum_j G m_j \frac{\mathbf{r}_{ij}}{(r_{ij}^2 + \varepsilon^2)^{3/2}} \\
\hat{\mathbf{a}}_i &= \sum_j G m_j \left( \frac{\mathbf{v}_{ij}}{(r_{ij}^2 + \varepsilon^2)^{3/2}} - \frac{3(\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}) \mathbf{r}_{ij}}{(r_{ij}^2 + \varepsilon^2)^{5/2}} \right),
\end{align*}
\]

(3.3) (3.4)

where \( \mathbf{r}_{ij} \) are the distance and \( \mathbf{v}_{ij} \) the velocity vectors, and \( \varepsilon \) is the softening parameter.

The corrector is given by the following formulas:

\[
\begin{align*}
\mathbf{r}_c &= \mathbf{r}_0 + \frac{\Delta t}{2} (\mathbf{v}_c + \mathbf{v}_0) - \frac{\Delta t^2}{12} (\mathbf{a}_1 - \mathbf{a}_0) \\
\mathbf{v}_c &= \mathbf{v}_0 + \frac{\Delta t}{2} (\mathbf{a}_1 + \mathbf{a}_0) - \frac{\Delta t^2}{12} (\hat{\mathbf{a}}_1 - \hat{\mathbf{a}}_0)
\end{align*}
\]

(3.5) (3.6)

The fourth-order Hermite scheme allows a time step, that is roughly twice as long as that of the Aarseth scheme of the same order for the same accuracy and is close to optimal for a wide range of required accuracy. The algorithm is simpler and more efficient as that of the Aarseth scheme [48].
In figure 3.14 the block diagram of the processor board is shown:

![Figure 3.14: GRAPE-4 processor board](image)

The data of the particles are stored in the particle data memory. They are used by the PROMETHEUS and the HARP\textsuperscript{\textregistered} chips, which are custom LSI chips. THE PROMETHEUS chip is used to predict the position \( r_p \) (eq. 3.1) and velocity \( v_p \) (eq. 3.2) of the particles at a specified time. A block diagram is shown in figure 3.15.

Each processor board has 48 HARP chips, which form multiple pipelines. All 48 chips share the particle data memory unit by one 32-bit data bus. The HARP chips calculate the distance \( r \), the velocity \( v \), the acceleration \( a \) (eq. 3.3) and its time derivative \( \ddot{a} \) (eq. 3.4). The block diagram is shown in figure 3.16.

Processor boards and control boards are connected with a 96-bit backplane bus, which is called the HARP bus. This bus is divided into three 32-bit sub-buses, where each of them is shared by 16 HARP chips. A synchronous pipelined protocol with fixed latency is used on this bus. The control logic unit controls the calculation on the board and generates all necessary signals to control the HARP bus.

\[\text{HARP: Hermite AcceleRator Pipeline}\]
3.2 Existing MD Hardware

Figure 3.15: PROMETHEUS chip

The PROMETHEUS chip processes the x, y and z components of all vector quantities sequentially in order to reduce the gate count. The calculation of $\Delta t$ and the addition of $r_j$ and the higher order term are performed in 64-bit format. All other calculations are performed in 32-bit format. The clock of this chip is the same as the system clock of the processor board.

Figure 3.16: HARP chip

The HARP chip handles the x, y and z components sequential in the same way as the PROMETHEUS chips does. The hardware is fully pipelined and calculates one interaction every three clock cycles. The HARP chip has the virtual multiple pipeline architecture, which is described on page 87. Two virtual pipelines calculate the interactions between two different particles at alternate clock cycles. The clock fre-
The frequency of the HARP chip is twice as quick as that of the board. From the outside, the chip works as it would contain two pipelines.

The subtraction of the position vectors and the accumulation of the calculated accelerations are performed in 64-bit floating-point format. Other calculations to obtain the acceleration are performed in 32-bit format. The subtraction of the velocity vectors and the accumulation of the time derivatives are performed in 38-bit format. Other calculations to obtain time derivative are done in 29-bit format, where the length of the mantissa is 20 bits.

Eight HARP chips are packaged in a MCM. As 16 HARP chips share the same HARP bus, all eight HARP chips in a MCM shares the same input data and I/O bus. One board has 6 MCM packages [45].

There are two different ways to use multiple GRAPE-4 boards for the calculation of the interactions:

- The first is to let all chips calculate the interactions on different particles from the same set of particles. In this case, the content of the memory of all processor boards would be identical. This is not practical in the case of GRAPE-4 with more than 1000 pipelines, because much memory space is required for storing the coordinates of all particles.

- The second is to let each processor board calculate the interactions on the same set of particles, but from a different set of particles. In this case, each processor board calculates the partial velocities, accelerations, forces and potentials. After the calculation, the partial results of all processor boards have to be added together.

The problem of the second approach is, that if the host would sum up the partial results of all processor boards, the amount of communication would be proportional to the number of processor boards because all their results should be sent back to the host.

Therefore, the control board was designed to reduce communication bandwidth between the host computer and GRAPE-4. It distributes the data received from the host computer to the processor boards, accumulates the partial velocities, accelerations, forces and potentials calculated on the processor boards and transfers them back to the host. The block diagram is shown in figure 3.17.
The control boards consist of the control logic unit, three accumulator/buffer units and several transceivers and buffers. The control logic controls the distribution and summation of the accumulator/buffer units and the data transmission over the data buses.

Each accumulator/buffer unit is connected with 16 HARP chips on each of the nine processor boards. Thus 144 HARP chips are connected with one accumulator/buffer unit over a separate 32-bit sub-bus of the HARP bus. The accumulator/buffer unit receives the coordinates of the particles from the host interface board and distributes it to the processor boards. After the calculation is finished, it adds the partial results being sent by the HARP chips and transmits the sum to the host interface board. The accumulator/buffer unit contains a 64-bit floating-point ALU for the accumulation.

The host interface board extends the I/O bus of the host. It converts the data transfer protocol of the host I/O bus to the protocol used on the links between the host interface boards and the control boards. The protocol on the link is designed in such a way, that it is independent on the protocol of the host I/O bus. In this way, GRAPE-4 can be connected to different host computers. The block diagram of the host interface board is shown in figure 3.18.
The data transceivers (trcv) exchange data with the host, the control board, the FIFO, which is 2048 x 32-bit words (8 KByte) and the control logic. The data transfer rate between the host interface and the control board is slower than that between the host and the host interface. Using the FIFO buffer, the host can send data at peak transfer speed. On the other side, the FIFO buffer allows the control board to transfer data without checking the status of the host bus. Therefore it works independent of the host bus.

The communication between the host interface board and the host computer relies on DMA transfers. An interface library for UNIX has been developed, to allow the application to invoke DMA transfer to/from its memory space directly.

The internal structure of the cluster is not visible to the application program. A cluster looks like a board with a single memory unit and multiple pipeline chips. Using more clusters the host can treat them as one cluster with more processor boards. If we have for example 4 clusters, the host sends \( N/4 \) particles to each cluster, where \( N \) is the total number of particles. In this case each processor board takes care of the contributions from \( N/36 \) particles. The control boards adds the partial results calculated on processor boards and the host computer adds finally the partial results calculated on the clusters.

GRAPE-4A has the same structure and chips as GRAPE-4 has. The only difference between GRAPE-4 and GRAPE-4A is the interface to the host computer. GRAPE-4 uses a TURBO channel interface, whereas GRAPE-4A uses a PCI interface.

GRAPE-4 has about 35'000 floating-point arithmetic units. It works with a system clock frequency of 15 MHz. The theoretical peak performance of a chip is 640 MFlops and that of the total machine is 1.08 TFlops. The achieved speed is roughly proportional to the number of particles for \( N^3 < N < N^5 \). For a gravitational N-body simulation with \( 1/r \) potential and \( N = 5 \cdot 10^4 \), the system achieves about 90 GFlops [44].
3.2.1.7 GRAPE-6

GRAPE-6 is the successor of GRAPE-4 and is planned for the year 2000. Its architecture is very similar to that of GRAPE-4 but with more pipelines and a faster speed. With GRAPE-6 larger simulations with a higher number \( N \) of particles can be simulated.

Whereas the GRAPE systems were used for the acceleration of the force calculation in astrophysical N-body problems, it is more and more used for other N-body simulations, such as cosmological smoothed particle hydrodynamics (SPH) simulations, evaluation and shifting of spherical harmonics in the fast multipole method, and MD simulations [50]. Such simulations use specific computation parts, e.g. the SPH interactions between neighbours, which cannot be calculated on the GRAPE systems and must therefore be calculated on the host computer consuming much CPU time.

Thus the initial basic structure of the GRAPE system (see figure 3.2) has been extended by using reconfigurable computing to achieve a more flexible pipeline architecture than the hard wired GRAPE pipeline (see figure 3.19).

![Extended structure of the GRAPE system](image)

*Figure 3.19: Extended structure of the GRAPE system*

However, reconfigurable computing is not as flexible as general-purpose computers, but faster for small and specific problems. It is not so efficient as GRAPE,
but the calculation cost of the specific computation parts are only proportional to \( N \) and therefore not as large as that of the gravity which is proportional to \( N^2 \). Thus reconfigurable computing offers an ideal solution for the specific computation parts, which demand flexibility, but are relatively time consuming [50].

The GRAPE-6 system is organized into 16 clusters, each with 8 processor boards. One processor board carries 32 G6 chips as processor chips. Thus the total system consists of 4096 G6 chips.

At the level of the cluster, different chips calculate the forces on different particles from the same set of particles. Therefore all clusters have the same image of particles \( j \) but different particles \( i \). This implies that particles \( j \) have to be broadcasted to all clusters, whereas particle \( i \) have to be distributed over the clusters.

Thus the 16 clusters are connected to the host computer through two separate networks. The one is the broadcast link (B-net), which connects host and the clusters in a one-dimensional, one-directional communication path. The other is a star-type network (S-net), which directly connects each cluster to the host computer. The B-net is used to broadcast the data to be written to the memories of the clusters and the S-net is used to read/write the registers in the G6 chips.

Within one cluster, different G6 chips calculate the forces on the same set of particles, but from different particles. Thus the data for particles \( i \) must be broadcasted,
whereas particles $j$ are distributed over the G6 chips. The partial forces, which are calculated on the G6 chips are summed up through a reduction network before they are sent back to the host.

The G6 chip is the integration of the PROMETHEUS chip and multiple pipelines of the HARP chip of GRAPE-4 (see section 3.2.1.6 on page 95). It contains 6 pipelines, which calculate one interaction each at one clock cycle.

The increased number of pipelines and the larger number $N$ of particles demand a higher memory bandwidth. Thus an MCM-module has been developed as processor module to keep the memory and G6 chips physically close. The MCM contains two G6 chips and four SRAMs. One G6 chip is connected to two SRAM chips over a 32-bit data line. The two G6 chips share a common 64-bit I/O port, through which they are connected to the communication path of the board.

One G6 chip runs on a clock frequency of 125 MHz and has a peak performance of 50 GFlops. The total machine has 200 TFlops [51].

### 3.2.1.8 MD-GRAPE

MD-GRAPE is the improved version of GRAPE-2A and was built in 1995. The system consists four MD chips, a particle index unit, a memory unit and an interface unit and is assembled on a single board (see figure 3.21).
MD-GRAPE can calculate the gravitational force using the $O(N^2)$ direct summation algorithm, the $P^3M$ method or the Ewald method (see sections 1.2.3.2 and 1.2.3.3).

In the $O(N^2)$ direct summation, MD-GRAPE calculates the forces and sends them back to the host computer, which calculates the new positions. In this calculation, the position vectors $r_i$ are distributed over the MD chips whereas $r_j$ are sent to all chips.

In the $P^3M$ method, MD-GRAPE calculates the real space ($PP$) force, while the wavenumber space ($PM$) force is calculated on the host computer. After the force calculation, the new positions are calculated on the host computer.

In the Ewald method, both forces can be calculated by MD-GRAPE.

On MD-GRAPE, an arbitrary central force is calculated with the following equations

$$ F_i = \sum_{j} F_{ij} = \sum_{j} a_j g(b_j r_s^2) r_{ij} $$

where $\varepsilon$ is a softening parameter, $a_j$ and $b_j$ are coefficients, and $g(\zeta)$ expresses an arbitrary function. Index $i$ is used for the particle at which the force is calculated and index $j$ for particles which exert the forces on particle $i$.

The potential is given by

$$ \phi_i = \sum_{j} \phi_{ij} = \sum_{j} a_j g(b_j r_s^2) $$

and using the DFT/IDFT, the summation is calculated by

$$ f_i = \sum_{j} f_{ij} = \sum_{j} a_j g(k_i \cdot r_j). $$

The MD chip integrates the GRAPE-2A pipeline and can be used in three modes: in force mode, in potential mode and in DFT/IDFT mode. In force mode equations 3.7 and 3.8 are directly calculated by the MD chip. In potential mode equation 3.9 is calculated and in DFT/IDFT mode equation 3.10 is calculated directly. The block diagram of the MD chip is shown in figure 3.22.
The coordinates $x, y, z$ of the position vectors $\mathbf{r}_i$ and $\mathbf{r}_j$ are given through the $r$-port and the coefficients $a_j$ and $b_j$ are given through the $c$-port of the chip. The softening parameter $\varepsilon$, which is used in force and potential mode, is hardwired. At the beginning of a sum calculation the position vector and the constant $k$ in DFT/IDFT mode of particle $i$ are given through the ports. During the calculation the position vectors and coefficients of particles $j$ are given through the ports. Additionally the MD chip can generate a neighbor flag if the distance $r_{ij}$ of the particles is less than the given neighbor radius $h_i$, which is stored in a register on the MD chip.

The MD chip has the virtual multiple pipeline architecture, which is described on page 87. It has 6 virtual pipelines and evaluates one interaction in each clock cycle. Therefore the MD chip calculates the forces on six different particles $\mathbf{r}_i$ using the same position vector $\mathbf{r}_j$ and coefficient vectors $a_j$ and $b_j$, which are supplied to the chip during six clock cycles.

The MD chip has 8 multipliers, 9 adders and 1 function evaluator. The function evaluator contains the arbitrary function $g(\zeta)$. Positions are expressed in 40-bit fixed-point format. The internal calculations are performed in 32-bit floating-point format, and the accumulation of the force is done in 80-bit fixed-point format. The function evaluator is a look-up table (LUT) with 1024 values.
The particle index unit, which is used for the cell-index method (see section 2.1.2.2), supplies the particle indices to the memory unit. It has the same structure as that of GRAPE-5 and is described on page 88. The only differences are the size of the cell counter, which is 14 bit wide and the size of the memory, which is a 1 Mbit (16 K x 32 bit) SRAM module.

The memory unit supplies the data of positions $r_j$ and coefficients $a_j$ and $b_j$ to the MD chip according to the particle indices supplied by the particle index unit. The data $r_j$, $a_j$ and $b_j$ are shared by all MD chips. The memory unit is composed of nine 1 Mbit (128 K x 8 bit) SRAMs. Five are for the position vectors and four are for the coefficients. It can hold up 43\,690 particles in three dimensions.

The neighbor list unit stores the numbers of particles, whose distance $r_{ij}$ is less than a given value $h_j$. When any of the MD chips asserts the neighbor flag, the particle number $i$ and the address of the memory corresponding to particle $j$ are written to a FIFO memory.

The first versions of MD-GRAPE had a VME-bus ‘slave’ interface, which transmits and receives data according to the request of the host computer. The interface interacts with the host by handshake, writes the received data, such as $r_j$, to the location specified by the address and supplies data, as the calculated force, with a read cycle. This makes the communication speed to be rather slow and therefore a PCI interface has been used for the further versions of MD-GRAPE [41].

On the MD-GRAPE board a N-body simulation with the P$^3$M method would take $240 \cdot \left(\frac{N}{10^6}\right)$ seconds per step and the Ewald method would take $600 \cdot \left(\frac{N}{10^6}\right)^{3/2}$ seconds per step. Each MD chip calculates $3.5 \cdot 10^7$ interactions at a clock frequency of 35 MHz. If we count the square root and division operations as 13 floating-point operations, the calculation of one gravitational force corresponds to 30 floating-point operations. Thus the MD-GRAPE board has a peak performance of 4.2 GFlops [52].

### 3.2.1.9 PROGRAPE-1

PROGRAPE-1 is a special-purpose computer for many-body simulations with pipelines specialized for computations of interactions between two particles. It was developed in 1998. In contrast to the GRAPE systems, which use custom LSI chips, it uses FPGAs for the pipelines. A block diagram is shown in figure 3.23.
PROGRAPE-1 calculates the summation

\[ f_i = \sum_{j} G(a_i, a_j), \quad (3.11) \]

where \( a_i \) and \( a_j \) are the physical values of the \( i \)-th and \( j \)-th particle, such as positions and velocities and \( G \) is a user-specified function.

It consists of two pipeline FPGAs, a memory unit and an interface unit. The memory unit stores the coordinates of \( a_j \) and supplies them to the interface unit. It uses four SRAM chips and is organized as 64 Kword x 32 bit.

The pipeline FPGAs calculate the summation of equation 3.11 and consists of two Altera EPF10k100 FPGAs [74]. They are connected to the memory over a 128-bit wide bus, where \( a_j \) is supplied, and to the interface unit over a 32-bit bidirectional bus, where \( a_i \) is received and the result \( f_i \) is sent back. On one pipeline FPGA a pipeline for Gravity/Coulomb force is implemented.

The interface generates the addresses for the pipeline FPGAs and the memory unit and manages the data transfer between PROGRAPE-1 and the host computer. It uses the same H-link as used for GRAPE-4 for connecting the PROGRAPE-1 system to a PCI interface. There was an Altera EPF10k20 [74] used for the interface unit.

The Gravity/Coulomb pipeline requires 30 floating-point operations in the FPGA and the clock frequency is 40 MHz. Thus the system achieves a peak performance of 2.4 GFlops [53].
3.2.2 GROMACS

The GROMACS hardware and system software were designed in a joint project with the University of Groningen and Chess Engineering in Harlem, Netherlands. It had been developed on the basis of the GROMOS algorithm in 1994 [54].

The architecture consists of a ring of 32 processor nodes. Each processor node consists of a 40 MHz Intel i860 processor, 8 MByte RAM, two 8-bit wide parallel ports to connect the board to the ring, four transputer links and an 8-bit PC-bus connector.

The ring communication with a bandwidth of 3 MByte/s is used for the actual simulation. The PC-bus is used to load programs and initial data and to store results. The transputer links are not used (see figure 3.24).

GROMACS uses a neighbor list for calculating the interactions, which is generated by using the grid-cell method (see section 2.1.2.3). Because the set of non-bonded forces changes every few time steps, the position of every particle has to be distributed over half of the ring. It is impossible to allocate particles to processors in such a way, that the communication remains minimal.

A distribution of the particles over half of the ring is insufficient for three and four particle bonded force calculations because in this way not every position triple or quadruple is present on at least one processor. During a simulation the set of bonded force calculations does not change and the communication characteristics
remain the same for a given allocation. Thus an allocation, which is generated during a preprocessing phase, remains valid during the whole simulation. Particles in triplet and quadruple interactions get close numbers, that they are allocated on close processors.

The GROMACS software consists of the preprocessing software, which runs on the host and the MD simulation software, which runs on the ring. The functionality includes Lennard-Jones, Coulomb and harmonic potentials, many types of bond-angle and dihedral interactions, different types of neighbour searching, notions like charge groups, exclusions, position and distance restraints, coupling to temperature and pressure baths and free energy calculation. During the actual simulation process an MD simulation can be monitored, interrupted, modified, resumed and stopped.

The software is designed to work on a ring of any size, also if only a single i860 board is used. Scaling from 1 to 8 processors (tested on 2000 water molecules) gives negligible scaling overhead and scaling from 8 to 32 processors yields only an efficiency of 75%. The communication overhead is in the region of 10 to 15% for 32 processors. Based on these numbers one may expect 20 to 30% overhead for 64 processors and 40 to 60% for 128 processors.

GROMACS does \((3.5\text{ to }7) \cdot 10^6\) non-bonded force calculations per second on a typical problem including overhead of neighbour searching and bonded force calculations. It is 5 - 6 times faster than a CRAY and 3 - 4 times faster than the NEC SX-3 vector supercomputer [55].

### 3.2.3 MD-Engine

MD-Engine is a special purpose parallel machine for molecular dynamics calculations made by Fuji Xerox in 1997. In order to simplify the hardware, it was developed to calculate only the non-bonded forces.

An MD-Engine system consists of a host computer and up to 4 MD-Engine system boxes. Each box can house up to 20 MD-Engine cards, of which each contains 4 special purpose processors called MODEL\# chip. The host computer sends the coordinates of the particles or reciprocal lattice vectors to the MD Engine, then the MD-Engine calculates and sends back the forces, the virials or a coefficient of the reciprocal lattice vector (see figure 3.25).

\# MODEL: MOlecular Dynamics Processing ELement
It uses the van der Waals (eq. 1.40) and Coulomb force (eq. 1.41) for the non-bonded force calculation. For periodically charged particles, the Ewald method (see section 1.2.3.3) can be used on the MD-Engine.

MD-Engine is a single-bus multi-port local memory multi-processor system. The SBus and VME bus interface card translates an SBus access request to a VME bus access request. All MODEL chips are connected to the VME bus in parallel (see figure 3.26).

Three types of local memories are connected to the individual memory ports of the MODEL chips. The coordinate memory stores the coordinates, the electrical charges and the index numbers of the particles. The function memory stores the
4 sets of three coefficients for quadratic interpolation. The parameter memory stores 2 or 8 sets of the force field parameters. These local memories are mapped to the internal register addresses of the MODEL chip.

The host computer can simply access the internal registers of the MODEL chip and the local memories as SBus memory mapped devices. While executing an MD simulation, often the same data are written to the registers of the MODEL chips. The host computer can write the MODEL chip registers at once using a global chip address. Before starting the force calculation, the host computer broadcasts coordinates of all particles, force field parameters and coefficients of interpolation to the appropriate local memories of the MODEL chips. Only the coordinate memories have to be updated at each time step. The MODEL chip carries out all particles $j$ to obtain the force acting on the $i$-th particles. The force calculations of the $i$ particles are equally distributed to all MODEL chips. The coordinate memories of every MODEL chip contains the coordinates of all particles.

The VME bus is not used during the force calculation. Furthermore, a MODEL chip is able to access its three local memories simultaneously, which allows for parallel operations.

The MODEL chip is responsible for all the required arithmetic operations to calculate the van der Waals and the Coulomb forces directly or with the Ewald method. In addition the potential energy of the system can be evaluated to assume that the simulation is running in the right way. The block diagram of the MODEL chip is shown in figure 3.27.

![Figure 3.27: MODEL chip](image)
The format of the position vector is defined in 40-bit floating-point format with 31-bits mantissa. A 64-bit double floating-point format with 52-bits wide mantissa is used to accumulate forces and virials by pairs. There are four 40-bit floating-point adders, three 40-bit floating-point multipliers and two 64-bit floating-point adders in a MODEL chip.

When calculating the Coulomb force, all position vectors and electrical charges are fetched from the coordinate memory. The Coulomb force and the virial are calculated simultaneously and the reciprocal lattice vector unit and the parameter memory are not used. When calculating the van der Waals force, a neighbor list scheme can be used, which may be generated by the MODEL chip itself during calculating the Coulomb force or by the host computer. In this mode, the reciprocal lattice vector unit is not used as well. In order to calculate the Ewald sum, the MODEL chip is used in five steps and the parameter memory and neighbor list are not used.

There are three library classes delivered with the MD-Engine. The low level and interface libraries and the application program. The functions in the low level library control the hardware of the MD-Engine and support broadcasting of the data to all MODEL chips or only to one. The SBus devices are mapped into the user virtual addresses. The functions in the low level library are called from the functions in the interface library, which are called themselves from the application program. In this way the system architecture is hidden from the application programmer.

A MODEL chip calculates a force by pairs within 400 nanoseconds. If few MODEL chips are used, the calculation time is proportional to $N^2$. If the number of the MODEL chips is so high, that the calculation of the non-bonded forces is shorter than the calculation time spent by the host, the simulation time is proportional to $N$, because it is limited by the computation time of the host computer and the communication time between the host computer and the MODEL chips.

An experiment with $N=11940$ and 24 MODEL chips takes 3.6 seconds to advance one time step. With this the MD-Engine accelerates an MD calculation by a factor of 98 compared to a SUN SPARC Station 10. The experiment shows that with a 6-card MD-Engine system, the accelerator is quite well optimised for the simulation of molecular systems with ~12'000 particles [56].
3.2 Existing MD Hardware

3.2.4 Conclusions

The drawback of the existing MD hardware is, that they are optimised for other MD algorithms and not for GROMOS. The most systems lack functions, that are used in GROMOS. These functions might be processed on the host computer, which would result in a performance loss.

The GRAPE systems of the low-accuracy type, GRAPE-1, GRAPE-1A, GRAPE-3, GRAPE-3A and GRAPE-5, have been designed for astro-simulations and collisionless systems and do not offer a direct calculation of the van der Waals and Coulomb forces. In addition the number range of the 32-bit fixed-point is not as high as used for GROMOS, where the coordinates are stored in 32-bit floating-point format.

The high-accuracy types offer the 32-bit floating-point formats. In contrast to GRAPE-2, it is possible on GRAPE-2A to calculate the van der Waals and Coulomb forces, but it is obsolete as machines with higher speeds are available. GRAPE-4 is a huge system for astro-simulations. The required forces may be calculated by programming the hardware, but it lacks a function for the energy and virial calculations. In addition the GRAPE-4 system is very expensive. The same problem has the GRAPE-6 system, which is quite expensive and not available yet.

MD-GRAPE calculates forces and virial directly. For energies two runs are necessary. It could be used in force- or potential mode. The DFT/IDFT mode may not be needed. Unfortunately MD-GRAPE uses fixed-point format, which is not suitable for GROMOS. In addition, there is a problem for the pairlist generation, because MD-GRAPE has no neighbor list unit and the distances are stored internally in the MDchip and cannot be obtained externally.

The PROGRAPE-1 system is a smart system for a good price, but it allows only to calculate the forces and not energies or virial.

All GRAPE systems lack a function to handle periodic boundary conditions. Further the required energies and virial, which are used in GROMOS, are only supported on MD-GRAPE, which is not suitable for a coprocessor, because of its fixed-point format. On the other machines, the energy and virial calculations are not implemented and should be calculated on the host. We conclude, that none of the GRAPE systems is suitable for a GROMOS coprocessor.

The GROMACS system is built similar as our coprocessor system should be. As it is based on the GROMOS algorithm, it supports all relevant functions as the calculation of the non-bonded forces, the energies and the virial as well as the use of periodic boundary conditions. Particles data distribution and communication over the ring is a good idea to solve the N-body problem. But the performance of the processing elements is quite low, faster processors would be necessary.
MD-Engine is also a large and very expensive system. The most interesting thing about MD-Engine is the MODEL chip, on which we could calculate the forces, the energies and the virial of the atoms. But the performance of this chip is low and a large amount of them would be necessary to achieve a high speed. Thus MD-Engine has a bad price/performance ratio and is not suitable for our coprocessor.

We conclude, that already existing MD hardware is not suitable for the GROMOS coprocessor, because the most solutions support other algorithms and have not implemented all functions, which are used in GROMOS. These should be calculated on the host and the performance gain by using an existing MD hardware would be small.

### 3.3 MD on Massively Parallel Computers

Instead of an existing hardware, which is specialized for MD calculations, a massively parallel computer system may be used as coprocessor. There exist many of such systems with different structures as shared-memory systems, vector supercomputers, array processors and so on [58].

It was early known, that MD simulations are time intensive and require a high performance. Therefore efforts had been made to implement MD algorithms on parallel computer machines, for reducing the simulation time [59].

Parallel computer systems may offer a high performance for MD calculations as shown in an implementation on the Connection Machine 5, where 50 Gflops is reached for an MD simulation with 131'072'000 particles [60]. The problem of the use of parallel machines is, that their different and special hardware architectures require each an individual adaptation of the algorithm to achieve a high performance. This causes a high implementation effort and inhibits a simple portation of a parallel program from one to another computer system. As an evaluation of parallel machines for MD is quite expensive, PARALLACS, a benchmark for parallel MD has been developed. But this benchmark program is not available for all parallel machines and supports not the full functionality of MD simulations [61]. In addition, commercial available parallel computer systems are large and very expensive machines.

A good alternative offers the MUSIC system, which is a DSP array processor, developed at the Electronics Laboratory at the ETH Zurich, Switzerland in 1992. Its network supports the intelligent communication method [62].
The MUSIC system consists of processor boards, which are connected over a communication network. Each processor board has three PEs and a board manager, which controls the data flow between the boards and the PEs. One PE has a Motorola DSP96002, 2 - 8 MB dual-ported VRAM, 256 KB - 1 MB SRAM and a communication controller, which manages the intelligent communication. In addition, I/O boards can be connected to the communication network of the MUSIC system, which allow to incorporate fast external data sources and sinks directly into the communication network for e.g. graphical applications.

The communication network is a pipelined bus, which allows to introduce a various number of processor and I/O boards. It works at 5 MHz clock rate and is 40 bit wide, which are organized as 32 data-bits and 8 token-bits for the identification of the transmitting PE. On each clock cycle, the PEs shift a data value to their right and receive a new one from their left neighbours. The communication interface decides if a data value is stored in the PEs memory and if the received value or a value from the own memory must be shifted to the next PE.

The DSPs work at a clock frequency of 40 MHz and have a peak performance of 60 MFlops. Using 20 boards with an amount of 60 DSPs a performance of
3.6 GFlops is achieved [63].

To test the performance of the MUSIC system for MD simulations, a GROMOS based MD algorithm was implemented by the W. van Gunsterens research group. The algorithm focuses only on simulations of relatively simple systems of atomic fluids with a small number of 1000 atoms, which require long simulation times. As the gain of using a pairlist is not relevant for small systems, the algorithm does not generate a pairlist and calculates the forces on the atoms directly. For the parallelisation a logical decomposition method with replicated data (see section 2.1.3) has been used. The MD algorithm is programmed in Assembler to achieve a better performance on the DSPs [64].

A simulation of a molecular system with 1000 atoms and 100 time steps was tested by the research group on MUSIC systems with a different number of processors. On a MUSIC-10 system (10 boards = 30 DSPs) the simulation requires a time of 3.8 seconds, which results in a performance of 295 MFlops [65]. Comparisons with any implementations of the same MD algorithm in Fortran on NEC SX-3, CRAY-YMP, SUN-4 (IPX) and IBM 6000/350 show, that the MUSIC achieves very good results.

We have estimated the performances of the different parallel machines for the MD simulation above by using the runtimes, that had been evaluated by the research group [15]. The results are shown in table 3.2: MUSIC-1 is a 1 board system with 3 DSPs, MUSIC-10 has 10 boards with 30 DSPs and MUSIC-20 contains 20 boards with 60 DSPs. The performances of NEC and CRAY are for 1 processor. For the MUSIC, the MD algorithm has been programmed in Assembler. The algorithms for the other parallel machines are implemented in Fortran.

<table>
<thead>
<tr>
<th>Parallel Computer</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUSIC-1 (1 board=3DSPs)</td>
<td>32 MFlops</td>
</tr>
<tr>
<td>MUSIC-10 (10 boards=30 DSPs)</td>
<td>295 MFlops</td>
</tr>
<tr>
<td>MUSIC-20 (20 boards=60 DSPs)</td>
<td>555 MFlops</td>
</tr>
<tr>
<td>NEC SX-3 (1 Processor)</td>
<td>255 MFlops</td>
</tr>
<tr>
<td>CRAY-YMP (1 Processor)</td>
<td>92 MFlops</td>
</tr>
<tr>
<td>SUN-4 (IPX)</td>
<td>1.7 MFlops</td>
</tr>
<tr>
<td>IBM 6000/350</td>
<td>3.4 MFlops</td>
</tr>
</tbody>
</table>

Table 3.2: Performance of parallel computer systems for MD algorithms
The performance estimations validate, that parallel computer systems may offer a high performance if we would use them for the GROMOS coprocessor. However, the algorithm, which had been tested on these machines, does not have the full functionality of GROMOS, but it may be expected, that the performance does not drop significantly for an implementation of the most time consuming GROMOS routines.

The highest performance is achieved on the NEC SX-3 followed by the CRAY machine, but the machines are large and very expensive. The MUSIC system may offer a good price/performance ratio. The problem is, that the DSPs must be programmed in Assembler, which requires a large effort for implementing. Additionally the MUSIC system is over eight years old and has become obsolete. There is no successor available.

We conclude, that parallel computer machines are not suitable for our GROMOS coprocessor, because a smart system with an acceptable price, that requires only few implementation effort is not available.

### 3.4 MD on Dedicated Hardware

The most time intensive parts of GROMOS, which are the distance algorithm for the pairlist generation and the non-bonded solvent-solvent interaction calculation, may be implemented on a dedicated hardware for accelerating. In this case the algorithms would be hardwired on an ASIC or an FPGA.

We investigated the suitability of hard wiring these algorithms by designing block diagrams for it and estimating the speed and size of such circuits by implementing arithmetic units on FPGAs.

The distance algorithm is used by the pairlist generation and the non-bonded interaction calculation (see section 2.2.3.1). A block diagram is shown in figure 3.29:
An implementation of this block diagram allows to calculate the distances between two particles in all simulation spaces. If the distances are calculated for the pairlist, the input parameters $v_i$ and $v_j$ of the fourth dimension are not used and only the output $r^2$ must be calculated. In this case the block diagram would be simpler.

For the distance algorithm only adders, subtractors, multipliers, squares, multiplexers and circuits for building the norms and the minima are necessary. If all functions could be executed in parallel, a pipeline with 8 stages may be built, such that one distance per step is calculated. The input parameters may be stored in registers or fetched from a memory. The results, which are used for the pairlist and force calculation, may be stored in registers or sent directly to their corresponding arithmetic blocks.
Figure 3.30 shows the block diagram for the calculation of the non-bonded solvent-solvent forces and the energies.

The Poisson-Boltzmann energies $ERF$, $ERC$ and the electrostatic energies $EEL$ are calculated from the charges $q_i$ and $q_j$ of the atoms, their three and four dimensional distances $r_{3D}^2$ and $r^2$, and the energy parameters $RFE$, $RFC$. The van der Waals energies $ELJ$ are calculated from the distances $r^2$ of the atoms and the van der Waals parameters $C_6$ and $C_{12}$, which are obtained over a parameter index.
table by the atom type codes \( AC_i \) and \( AC_j \). The distances \( r^2 \), the van der Waals parameters \( C_6 \) and \( C_{12} \) and the charges \( q_i \) and \( q_j \) together with the parameter \( RFF \) are used to calculate the non-bonded force factor \( DF_3 \), which is an interme¬diate result. It is used for the virial calculation and multiplied with the coordinates of the distance vector to evaluate the partial force vector, which is summed up to the force vectors of the atoms \( i \) and \( j \). The calculation of the non-bonded force uses equation 1.42 on page 16 and is described in section 1.2.2.2.

In addition to the adders, subtractors and multipliers, there is one square root, a reciprocal and a exponential function, which are quite complex to implement with logic blocks and would require many clock cycles for executing even if fast versions are used [66]. Such implementations are inappropriate for an FPGA, since too many cells would be necessary for them. Therefore an adequate and fast solution may be the use of lookup tables for realizing such functions.

Realizing all arithmetic functions in parallel, a pipeline with 12 stages could be implemented. The input values may be obtained directly from the distance calculation. The parameters, mainly constants, are stored in registers, as well as the results, which are added up.

The virial is easier to calculate. The block diagram is shown in figure 3.31.

![Block diagram](image)

*Figure 3.31: Calculation of virial on an ASIC or FPGA*

The virial is calculated from the coordinates of the distance vector \( r_x, r_y, r_z \), the centres of mass \( cm_x, cm_y, cm_z \) of the molecules and the force factor \( DF_3 \). Distance vector and force factor are obtained from the distance and the non-bonded
force calculation. The centres of mass may be calculated on the host and are sent to
the coprocessor.

The virial calculation requires only adders, subtractors, multipliers and sign
switchers. A pipeline with 6 stages could be realised by implementing these func-
tions in parallel.

If we would implement the three algorithms, distance calculation, non-bonded
solvent-solvent force calculation and virial calculation in the manner of the block
diagrams as shown in the figures 3.29, 3.30 and 3.31, a pipeline with a total of 23
stages may be realized as some parts of the force and virial calculation could be
overlapped. In this case, all arithmetic blocks should be available in parallel, what
means, that 31 adders, 22 subtractors, 25 multipliers, 4 norms, 4 squares, 1 square
root, 1 reciprocal value, 1 exponential function with radix 3, 3 sign switchers, 4
minima and 8 multiplexers are necessary.

There exist a large amount of implementations of arithmetic functions [67]. The
simplest are the adders, subtractors, multipliers and squares. An implementation of
divisions, reciprocals, square roots and exponential functions is more difficult.
Often, they are evaluated with approximation algorithms using standard functions
as addition, subtraction and multiplication [68]. Thus the approximation algorithm
may be hard wired with simple arithmetic blocks or executed by software, e.g. on
processors, where an implementation of divisions and square root lacks. A further
possibility is the use of lookup tables [69]. The norms, sign switchers and minima
functions may be implemented easily by using single gates and multiplexers.

Having all arithmetic functions in parallel on an FPGA, would result in a lack of
space. Due to the required accuracy of the GROMOS simulation, 32-bit floating-
point arithmetic functions are necessary. Floating-point arithmetic is more compli-
cated than fixed-point arithmetic. A floating-point multiplier must be constructed of
a multiplier for the base and an adder for the exponent. A floating-point adder con-
ists of an adder and shifter for the base, an incrementor/decrementor for the expo-
nent and a decision logic for keeping the base and the exponent in the correct range.
The 32-bit word width requires very large arithmetic blocks, because their size
grows quadratic by doubling the number of bits. Faster implementations of arith-
metic blocks require in general more space than slower ones, because on fast imple-
mentations all bits may be processed in parallel, whereas slow implementations
process the bits sequentially by using shift-registers.

We tried to implement a parallel 32-bit floating-point adder and multiplier on a
FPGA to estimate the required resources for implementing the distance, the non-
bonded forces and the virial calculation algorithms.

For the tests, we used a FLEX 8000 and 10K of the Altera FPGA family [74].
Our investigations have shown, that the implementation of parallel 32-bit floating-
point arithmetic blocks is impossible. The required logic and routing resources are too high even for the actual largest available Altera FPGA. A redimension of the arithmetic to 16-bit showed, that an EPF10K20 with a maximum of 63'000 gates is needed. In the largest Altera FPGA, the EPF10K50 with 116'000 gates, only two floating-point arithmetic blocks can be fitted, which means, that about 50 Altera FPGAs might be necessary to implement all the required arithmetic blocks of the three algorithms in parallel.

Instead of parallel 32-bit floating-point arithmetic, a bit serial approach can be used, where a clock frequency of up to 33 Mhz is reached. But this technique requires still much resources on an FPGA and has high latencies with 9 - 15 clock cycles [70].

The use of fixed-point arithmetic requires fewer resources on an FPGA, but the word width should be limited to 8 bits or less to implement algorithms with several arithmetic units. This shows the implementation of a Golomb Ruler algorithm [71]. It uses 1 adder, 3 subtracters, 1 barrel-shifter and 5 decoders. The word width had to be limited to 6 bits to fit the whole circuit into an XC6216 Xilinx FPGA [72]. An implementation of an 8-bit fixed-point processor on the same FPGA showed, that practically all cells of the XC6216 are used. The processor has logical functions as NOT, AND, OR, XOR, shift and rotate functions, and arithmetic functions as adder, subtractor, increment, decrement and negation. There has been no multiplier implemented on the processor, because it would be too large for the XC6216 [73]. However, the XC6216 is a small reconfigurable FPGA with an array of $64 \times 64 = 4096$ configurable logic blocks. The fine-grain architecture does not offer many routing resources, which may result in problems for fitting arithmetic blocks [75]. But there exists larger FPGAs, which allow an implementation of any of the three algorithms using 8-bit fixed-point arithmetic.

For estimating the speed of an FPGA, we implemented an 8-bit version of the distance algorithm on a XC4025-5 Xilinx FPGA, which has 25'000 gates [76]. But the XC4025 was to small for implementing the whole algorithm. Therefore we left out the implementation of octahedral and monoclinic simulation boxes as well as the calculation of the distance vector and the calculation in the fourth dimension. The implemented algorithm calculates only the distance of two particles in vacuum or periodic rectangular boxes in three dimensions and can only be used for the pair-list generation.

The implemented circuit uses 5 adders, 6 subtractors, 3 squares and 3 minimizers. We used VHDL for the specification of the circuit, which was synthesized with the Synopsys compiler by optimising the size, pipelining and speed.

The highest clock frequency we achieved on the XC4025 is 6.4 MHz for the distance calculation and a valid result is given on the outputs every 2 clock cycles.
Using the XC4025 for generating the pairlist of the Thrombin simulation, 74976135 distances have to be calculated for the rectangular simulation and a time of 23.4 seconds is needed. This results in a performance of 54.4 MOPS.

The performance of an FPGA is quite high for calculating the distance with 8-bit fixed-point format, but it does not offer enough accuracy for GROMOS. The use of larger word widths would require multiple FPGAs, to which the arithmetic blocks should be distributed. This requires interconnections between the FPGAs, which have long delays, decrease the clock frequency and therefore the performance. In addition, there is no way to implement the algorithms with 32-bit floating-point arithmetic units for enough accuracy.

We conclude, that an FPGA is not suitable for a GROMOS coprocessor. Single FPGAs are too small for implementing the required algorithms with enough accuracy and the use of multiple FPGAs would result in a high performance loss.

However, an ASIC instead of an FPGA could be used. There is a better yield of the area on a full custom chip for implementing arithmetic units and an ASIC may work at higher clock frequencies. An implementation of the distance, force and virial calculation pipeline on an ASIC, which works with a frequency of 100 MHz, may reach a performance of about 9 GFlops. But the development of an ASIC is relatively expensive and it is very inflexible for modifications of the algorithm. As the GROMOS algorithm is permanently improved (see section 2.2.1), an ASIC is not suitable for an implementation, because a further version would require a development of a new ASIC with much effort and high costs.

### 3.5 Digital Signal Processors for MD

DSPs offer a high performance at low costs. Mostly they are used for real-time applications. Thus the architecture is built with hardwired logic, which is faster than micro code as quite often used in microprocessors.

In digital signal processing applications a data stream is going through the DSP. In the most applications and particular for filtering and echoing, each sample has to be multiplied with one or more parameters and the results have to be accumulated to obtain the new processed sample, which is written into the memory or sent out of the DSP. Therefore DSPs have a fast implementation of the MAC\(^\text{\textsuperscript{\(\text{\textregistered}\)}}\)-operation or are equipped with multiple accumulators. For the scaling of sample data, it has additionally barrel-shifters.

\[\text{MAC: Multiply And Accumulate}\]
DSPs have built-in memories for look-up tables or filter parameters. The memories are also used for implementing ring-buffers. Thus DSPs offer separate address generators. They implement the Harvard architecture with separate command- and data-buses [77].

Most of the DSP specific functions may be used for the interaction calculation of GROMOS. The data of the particles $j$ could be stored in the on-chip memory, whereas the data for particle $i$ could be stored in the registers. The accumulators could be used to sum up the forces and the energies. With the address generator the address for the next particle $j$, which is in the pairlist, could be calculated.

We tested the performance and the suitability of DSPs for GROMOS by implementing the distance and force calculation on an ADSP-21060 SHARC by Analog Devices. In the following paragraph there is a short description of the ADSP-2106x processors.

### 3.5.1 ADSP-2106x SHARC

The ADSP-2106x SHARC by Analog Devices is a high-performance 32-bit digital signal processor with a clock-frequency of 40 MHz, which results in a sustained performance of 40 MIPS and a peak performance of 120 MFlops. It came out in 1995. The Block diagram is shown in figure 3.32.

![Figure 3.32: ADSP-2106x SHARC DSP](image)
3.5 Digital Signal Processors for MD

The ADSP-2106x core processor consists of 3 independent computation units: an ALU, a multiplier with a fixed-point accumulator for the MAC operation and a shifter. The computation units process data in three formats: 32-bit fixed-point, 32-bit and 40-bit IEEE floating-point format. The computation units perform single-cycle operations without being pipelined. There can be a multifunctional computation, where the ALU and multiplier perform independent, simultaneous operations.

The data register file contains two sets of sixteen 40-bit registers to allow fast context switching. It is used for transferring data between the computation units and the data buses and for storing intermediate results.

The program sequencer supplies instruction addresses to the program memory. It controls loop iterations and evaluates conditional instructions. The ADSP-2106x achieves its fast execution rate by means of pipelined fetch, decode and execute cycles. With its instruction cache, it can simultaneously fetch an instruction from the cache and access two data operands from the memory.

The data address generators (DAGs) provide memory addresses when data is transferred between the memory and the registers and perform automatic modulo addressing for ring-buffers. With an internal loop counter and loop stack, the ADSP-2106x executes loop code with zero overhead.

The processor core has 4 buses: the program memory (PM) address bus, the data memory (DM) address bus, the program memory data bus, and the data memory data bus. This allows for dual data fetches, when an instruction is supplied by the instruction cache.

The ADSP-21060 contains 4 Mbit and the ADSP-21062 contains 2 Mbit of on-chip SRAM, each organized as two blocks. Each memory block is dual-ported for single-cycle independent access by the core processor and the I/O processor or the DMA controller.

The external port provides the processors interface to off-chip memory and peripherals. The 4-gigaword off-chip address space is included in the ADSP-2106x’s unified address space.

The host interface allows easy connection to standard microprocessor buses with little additional hardware required. It is accessed through the external port and is memory mapped into the unified address space. The host can directly read and write the internal memory of the ADSP-2106x.

The ADSP-2106x has two synchronous serial ports that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices and it has six 4-bit link ports, especially useful for point-to-point interprocessor communication in multiprocessing systems.

The DMA controller allows zero-overhead data transfers without processor intervention. It operates independently and invisibly to the processor core. DMA
transfers can occur between the ADSP-2106x's internal memory and external memory, external peripherals, a host processor, or its serial port or link ports.

Up to 6 ADSP-2106xs and a host processor can directly be connected together. Distributed bus arbitration logic is included on chip. Bus arbitration is selectable as either fixed or rotating priority. The unified address space allows direct interprocessor access of each ADSP-2106x's internal memory. Broadcast writes allow simultaneous transmission of data to all ADSP-2106xs [78].

3.5.2 Benchmarks on the ADSP-2106x DSP

The ADSP-2106x SHARC can either be programmed in C or in Assembler. The programming source of the GROMOS96 software is in Fortran. First we translated the Fortran routines to C, have then optimised the code and at last we have translated it into Assembler to use the special structures of the SHARC processor to improve speed. We used the ADSP-2106x SHARC DSP processor simulator for testing and benchmarking both programming routines.

3.5.2.1 Method

The distance algorithm and the force algorithm for the non-bonded solvent-solvent interactions are implemented as two separate programming routines. To migrate the two algorithms from GROMOS96 to the ADSP-2106x, we took the routines in Fortran source code and translated it directly into C-code. Then we tested its correctness with the SHARC DSP simulator and made the first benchmark.

The directly translated C-routines offer potential for different improvements of the code to enhance speed. Thus we made the following three manual optimisations of the C-code:

1. We simplified the if-statements to make the code more efficient. We adapted the if-conditions to the internal structure of the SHARC processor. Additionally we moved some if-statements to another place in the routine, so that fewer if-statements are executed in certain runs.

2. We replaced the loops with sequential code. Although the SHARC has a program sequencer, which controls loops more efficiently than in a normal processor, it is faster to code the loops sequentially in the routines, because they are executed not many times.

3. We removed all temporary variables. This caused an interesting effect as we used the automatic optimizer.
The C-compiler has an automatic optimizer for the code. All C-program versions with their manual optimisation steps were benchmarked with and without the automatic optimisations. Both values are listed in the tables, the first with no optimizer, the second with optimizer.

The best performance on a DSP is reached with Assembler program coding. Therefore we used this program language to make further speed optimisations. By investigating the compiled Assembler-code of the best C program version, we saw that there are mainly three properties of the SHARC to speed up the program, which were not used by the C-code and must therefore be introduced manually.

We optimised the Assembler-code in four steps and made the following optimisations:

1. The compiler uses registers of the DAG to fetch values from the memory and to store the values of the register file into the memory, but it does not use it optimally. For normal variables in C, a direct addressing of the memory requires fewer clock cycles and for pointer variables the DAG-registers can be used more effective by using commands for fetching data and incrementing the registers simultaneously. Thus we changed the program to achieve faster memory accesses.

2. In the C-code, many intermediate results are stored in the internal memory, although there are registers in the processors register file, which are not used. This requires more time. Therefore we optimised the code for better using the register file.

3. The SHARC has an instruction pipeline with three steps: fetch - decode - execute. Therefore it can execute one instruction per clock cycle. If there is a branch, the program pointer is set to the branch destination not before the execute stage. Thus the pipeline is filled with the two instructions, which are placed immediately after the branch in the program code. These are then in the fetch and decode stages of the pipeline and the pipeline must be refilled with the new instructions after the branch is executed. Thus the program flow idles for two clock cycles.

There is a special instruction, called delayed branch, to avoid this. In the delayed branch the two instructions in the program code, which are placed after the branch instruction are then executed too. In this way the two instructions before the branch instruction can be placed after the branch instruction, which can then be changed into a delayed branch.

4. On the SHARC many arithmetic functions and memory transfer instructions can be parallelized. Nothing of them is done if C-code is used. In the last optimisation step we parallelized the instructions as much as possible and achieved a high speedup in comparison with the last optimisation step once again.
We optimised both programming routines step by step and made benchmarks for each optimisation step. The SHARC-simulator produces the number of clock cycles, during the time a program is being simulated. For the evaluation of the calculation time, we took the number of clock cycles for one run of each programming routine, which is the calculation of the distance and the force of one pair of atoms. The so produced number of clock cycles, the known number of molecules and the also known number of elements in the pairlist allow us to estimate the simulation time of the Thrombin molecular dynamics topology on a SHARC DSP.

3.5.2.2 Distance Routine

The distance routine calculates the distance vector and the square of the distance between two particles. It is used in GROMOS for the pairlist generation and the interaction calculation. The algorithm is described in section 2.2.3.1.

The distance calculation is implemented as a single programming routine, where two atoms are sent to and which gives back the distance vector and the square of the distance. Many different paths through the programming routine are possible and we have all of them examined. In the following the different paths are listed:

- Atoms in the vacuum can be calculated in three and four dimensions.
- If distances in a rectangular periodic box are calculated, the three following paths are possible: Both atoms are in the middle of the boundary box and no adjustment of the distance vector is needed or the atoms are at the border of the box and a positive or a negative adjustment has to be done. All of them can be calculated in three or four dimensions.
- The same paths are possible for the monoclinic box, but only in three dimensions.
- In a space with periodic truncated octahedrons the same adjustments as for monoclinic boxes are made. After then, the distance vector is adjusted for the octagonal space unless both atoms are in the middle of the octahedron and no adjustment is required.

The components of the distance vector are calculated sequentially. Therefore the programming routine contains many loops, where the components of the distance vector are calculated in three or four dimensions. As a lot of decisions about the dimensions and simulation spaces have to be made in the algorithm, many if-statements are used in the programming routine.

In table 3.3 the number of clock cycles of the C-code versions for all possible paths of the distance calculation between two atoms are listed. The values are from that distance routine, which is used for the interaction calculation. The distance
routine for the pairlist requires 2 - 10 clock cycles less than that for the interaction calculation. Again, the two values in each column are the measurement results of the non automatically/automatically optimised C-code.

<table>
<thead>
<tr>
<th>Boundaries</th>
<th>Dim.</th>
<th>Adjustment</th>
<th>Optimisation Step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Origin</td>
</tr>
<tr>
<td>Vacuum</td>
<td>3D</td>
<td>none</td>
<td>219/159</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>none</td>
<td>287/211</td>
</tr>
<tr>
<td>Rectangular</td>
<td>3D</td>
<td>none</td>
<td>309/225</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>303/216</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>330/237</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>none</td>
<td>404/296</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>396/284</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>432/312</td>
</tr>
<tr>
<td>Octahedral</td>
<td>3D</td>
<td>none</td>
<td>333/247</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>514/399</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>541/420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>octahedron only</td>
<td>520/408</td>
</tr>
<tr>
<td>Monoclinic</td>
<td>3D</td>
<td>none</td>
<td>330/241</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>324/232</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>351/253</td>
</tr>
</tbody>
</table>

| Table 3.3: Clock cycles of the distance routine in C on the SHARC ADSP-2106x |

In Table 3.3 the values with no automatic optimisation in the 3rd column are lower than that in the 2nd column, but the values with automatic optimisation in the 3rd column are higher than that in the 2nd column.

The reason is, we removed all temporary variables in the 3rd optimisation step. This causes a more efficient code if we do not use the automatic optimizer, because the values are not copied into temporary variables, which are stored then in memory. If we use the automatic optimizer, we obtain a better result, compared to using temporary variables, because the automatic optimizer recognizes temporary varia-
bles and uses registers in the register file for it. In this case the calculation is done by using more registers, what makes the calculation more efficient. If we have no temporary variables defined, the optimizer uses as few registers as possible and all temporary values are stored in memory, which requires more time.

In table 3.4 we have listed the clock cycles of the Assembler-code versions of the distance algorithm. They again are separated in all possible paths of the program. Additionally the values of the fastest C-code are listed for comparison too.

<table>
<thead>
<tr>
<th>Boundaries</th>
<th>Dim.</th>
<th>Adjustment</th>
<th>Optimisation Step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Best C</td>
</tr>
<tr>
<td>Vacuum</td>
<td>3D</td>
<td>none</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>none</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rectangular</td>
<td>3D</td>
<td>none</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>none</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>152</td>
</tr>
<tr>
<td>Octahedral</td>
<td>3D</td>
<td>none</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>193</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td></td>
<td>octahedron only</td>
<td>202</td>
</tr>
<tr>
<td>Monoclinic</td>
<td>3D</td>
<td>none</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>positive</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td></td>
<td>negative</td>
<td>136</td>
</tr>
</tbody>
</table>

*Table 3.4: Clock cycles of the distance routine in Assembler on the SHARC ADSP-2106x*
Many clock cycles can be saved by using the register file instead of the memory for storing intermediate results, which was done at the optimisation step 2. A lot more clock cycles can be saved if we parallelise the instructions to use the arithmetic units and memory accesses at the same time.

In figure 3.33 the calculation times of each optimisation step for all different simulation boxes, in which the atoms can be calculated, are compared. We took the worst case of the clock cycles for that boxes, which have more possible paths. The calculation times are evaluated for an ADSP-2106x with a clock frequency of 40 MHz.

![Diagram](image.png)

**Figure 3.33: Runtimes of the distance routine on the SHARC ADSP-2106x**

On C we considered only the runtimes of the original program with and without automatic optimisation, the runtimes of the best manual only optimised code and the runtimes of the best C-code, which is manually and automatically optimised. On Assembler, we considered the runtimes of these codes, on which the optimisation had the highest effect. From the different paths of the program routine we took always the worst case.

Table 3.5 shows the runtimes of the C-code, which was directly translated from Fortran and the runtimes of the fastest C- and Assembler-codes.
We have reached a quite high speedup by coding and optimising the distance algorithm in Assembler. Figure 3.34 shows the speedups of all program versions with the same optimisation steps as used in figure 3.33, which are in comparison to the not optimised C-code.

Table 3.5: Runtimes of the distance routine on the SHARC ADSP-2106x

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3D</td>
<td>4D</td>
<td>3D</td>
<td>4D</td>
</tr>
<tr>
<td>C (directly from Fortran)</td>
<td>5.48</td>
<td>7.18</td>
<td>8.25</td>
<td>10.8</td>
</tr>
<tr>
<td>C (fastest code)</td>
<td>1.38</td>
<td>1.68</td>
<td>3.05</td>
<td>3.8</td>
</tr>
<tr>
<td>Assembler (fastest code)</td>
<td>0.58</td>
<td>0.6</td>
<td>1.25</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Figure 3.34: Speedups of the distance routine on the SHARC ADSP-2106x

The automatic optimizer is able to make a better optimisation with the manual optimised C-code. A further acceleration can be reached with register optimising in Assembler and at last the fastest speedup is reached by parallelising the instructions of the Assembler-code.
To compare the achieved speedups with the effort of the manual optimisation of the code, table 3.6 lists the speedups of the fastest C- and Assembler-code in contrast to the not optimised C-code and additionally the speedups between the fastest C- and Assembler-code.

<table>
<thead>
<tr>
<th>Programming Language Version</th>
<th>Vacuum</th>
<th>Rectangular</th>
<th>Octahedral</th>
<th>Monoclinic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3D</td>
<td>4D</td>
<td>3D</td>
<td>4D</td>
</tr>
<tr>
<td>C (fastest code)</td>
<td>3.98</td>
<td>4.28</td>
<td>2.7</td>
<td>2.84</td>
</tr>
<tr>
<td>Assembler (fastest code)</td>
<td>9.52</td>
<td>11.96</td>
<td>6.6</td>
<td>9.0</td>
</tr>
<tr>
<td>Fastest C-code vs. Assembler-code</td>
<td>2.39</td>
<td>2.79</td>
<td>2.44</td>
<td>3.17</td>
</tr>
</tbody>
</table>

Table 3.6: Speedups of the distance routine on the SHARC ADSP-2106x

If we compare the fastest Assembler-code with the directly translated C-code, the average speedup of 8.65 is very high. The optimising of the C-code, which was relatively fast and simple, results already in an average speedup of 3.18. The effort for the Assembler-coding and -optimising was three times higher than that of the C-code, but we reached an additional speedup of only 2.73.

These results show, that the C-code can be optimised with little effort to achieve more speed. Using Assembler-coding the speed of a program can be raised about 2.5 times, but this is relatively complicated and time-intensive and makes the program more inflexible. Thus the effort for Assembler-coding is only worthwhile if the algorithm does not change often.

### 3.5.2.3 Force Routine

The force routine evaluates the forces, the energies and the virial of the non-bonded atoms. The algorithm is described in section 2.2.3.3, whereas the routine on the SHARC contains only the parts of the interaction calculation, without the distance calculation.

The calculation of the forces, energies and virial is implemented in the same way as the distance algorithm, with a single programming routine. The distance and the distance vector, which are obtained from the distance routine, are sent to the pro-
gramming routine, which gives back the energies, the forces and if desired the virial.

The components of the force vector are calculated sequentially. Here no loop was used in the original Fortran program. An if-statement decides if the component of the fourth dimension has also to be calculated.

Since the forces can be calculated in three or four dimensions and the virial can be evaluated or not, four different paths are possible.

In table 3.7 the number of clock cycles of the C-code versions for these possible paths are listed. Because the original Fortran program uses no loops and we have only two if-statements, which cannot be optimised, we used only optimisation step 3 (see section 3.5.2.1 on page 124) to improve the C-code. Again, the two values in each column are the measurement results of the non automatically/automatically optimised C-code.

<table>
<thead>
<tr>
<th>Virial</th>
<th>Dim.</th>
<th>Optimisation Step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Origin</td>
</tr>
<tr>
<td>no</td>
<td>3D</td>
<td>373/232</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>417/244</td>
</tr>
<tr>
<td>yes</td>
<td>3D</td>
<td>469/293</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>513/305</td>
</tr>
</tbody>
</table>

Table 3.7: Clock cycles of the force routine in C on the SHARC ADSP-2106x

The best values can be achieved by using automatic optimisation only. More clock cycles are used by the C-code, which was automatically and manually optimised. The reason is the same as in the distance routine, where the compiler stores temporary values in the memory if no temporary variables are defined.

Table 3.8 lists the clock cycles of the Assembler-code versions of the force routine. They are separated in the four possible paths of the program again. Since we have no branch-instructions in the programming routine, we have omitted optimisation step 3 (see section 3.5.2.1 on page 125). For a comparison, the values of the fastest C-code are also listed in the table.
3.5 Digital Signal Processors for MD

<table>
<thead>
<tr>
<th>Virial</th>
<th>Dim.</th>
<th>Optimisation Step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best C</td>
</tr>
<tr>
<td>no</td>
<td>3D</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>244</td>
</tr>
<tr>
<td>yes</td>
<td>3D</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>4D</td>
<td>305</td>
</tr>
</tbody>
</table>

Table 3.8: Clock cycles of the force routine in Assembler on the SHARC ADSP-2106x

Many clock cycles can be saved by a better use of the registers of the DAG due to the pointers, which are used for accessing the parameters of the force field of the $C_6$ and $C_{12}$ atoms. More clock cycles can be saved by using the internal registers for the temporary variables and additional clock cycles can be saved with the parallelizing. Compared to the distance routine much more clock cycles can be saved with the register optimisations, because more variables are used in the force routine.

For a comparison, the calculation times of each optimisation step of the force routine on a 40 MHz ADSP-2106x are shown in figure 3.35.

![Figure 3.35: Runtimes of the force routine on the SHARC ADSP-2106x](image-url)
The automatic optimizer is able to produce faster code with no manual optimisation. Thus the manual optimisation of the C-code does not make sense for the force routine. In table 3.9 the runtimes of the C-code, which was directly translated from Fortran, that of its automatically optimised version and that of the fastest Assembler-code are listed.

<table>
<thead>
<tr>
<th>Programming Language Version</th>
<th>no Virial [\mu s]</th>
<th>with Virial [\mu s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3D</td>
<td>4D</td>
</tr>
<tr>
<td>C (directly from Fortran)</td>
<td>9.33</td>
<td>10.43</td>
</tr>
<tr>
<td>C (fastest code)</td>
<td>5.8</td>
<td>6.1</td>
</tr>
<tr>
<td>Assembler (fastest code)</td>
<td>2.3</td>
<td>2.4</td>
</tr>
</tbody>
</table>

*Table 3.9: Runtimes of the force routine on the SHARC ADSP-2106x*

The speedup, which we have achieved with the code optimising, is not so high as that of the distance routine. Figure 3.36 shows the speedups of all optimisation steps of the force routine.

*Figure 3.36: Speedups of the force routine on the SHARC ADSP-2106x*
3.5 Digital Signal Processors for MD

We cannot reach a faster program version with the manual optimisation of the C-code. The automatic optimisation is better without a manual optimisation. A higher speedup is reached by using Assembler-code. By parallelising the instructions an additional speedup can be achieved.

Table 3.10 lists the speedups of the fastest C- and Assembler-code in contrast to the not optimised C-code and additionally the speedups between the fastest C- and Assembler-code.

<table>
<thead>
<tr>
<th>Programming Language Version</th>
<th>no Virial</th>
<th></th>
<th>with Virial</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3D</td>
<td>4D</td>
<td>3D</td>
<td>4D</td>
</tr>
<tr>
<td>C (fastest code)</td>
<td>1.61</td>
<td>1.71</td>
<td>1.6</td>
<td>1.68</td>
</tr>
<tr>
<td>Assembler (fastest code)</td>
<td>4.05</td>
<td>4.34</td>
<td>4.23</td>
<td>4.46</td>
</tr>
<tr>
<td>Fastest C-code vs. Assembler-code</td>
<td>2.52</td>
<td>2.54</td>
<td>2.64</td>
<td>2.65</td>
</tr>
</tbody>
</table>

*Table 3.10: Speedups of the force routine on the SHARC ADSP-2106x*

The force programming routine cannot be as well optimised as the distance routine, as there are no loops and no branch-instructions. Also the if-statements could not be optimised. We achieve a total average speedup of 4.27 with all optimisations of C- and Assembler codes. For the C-code optimisation we reach only an average speedup of 1.65. Using Assembler-code we achieve a further average speedup of 2.59.

Thus we have to implement the force routine in Assembler on the SHARC DSP to obtain a faster program version, which requires a high effort. But this makes the program also more inflexible.

3.5.2.4 Conclusions

Using the results, which we have discussed in sections 3.5.2.2 and 3.5.2.3, we can make an estimation of how much time the pairlist and the non-bonded solvent-solvent interaction calculation of the Thrombin molecular simulation on a SHARC ADSP-2106x would take. The parameters of the Thrombin molecular simulation are in table 2.1
To obtain a realistic estimation, we have to consider the possible paths of the distance routine (see section 3.5.2.2). This can be done by using the probabilities of periodic boundary corrections $P_{rcb}$ and $P_{ocb}$ (see section 2.3.3.1), which determine the share of each possible path. A multiplication of the corresponding probability with the respective number of clock cycles of the corresponding path, delivers the share of clock cycles of that path for calculating one distance. Considering all possible paths, the average of the number of clock cycles for the distance calculation of one particle pair on the ADSP-2106x can be obtained with

$$c_{\text{dist}} = (1 - P_{rcb} - P_{ocb}) \cdot c_{\text{none}} + P_{rcb} \cdot c_{\text{pos/neg}} + P_{ocb} \cdot c_{\text{octo}},$$

(3.12)

where $c_{\text{none}}$, $c_{\text{pos/neg}}$ and $c_{\text{octo}}$ are the clock cycles of tables 3.3 and 3.4. For the parameter $c_{\text{pos/neg}}$ we took the worst case. $P_{rcb}$ and $P_{ocb}$ are defined in table 2.7 for the Thrombin simulation.

For the pairlist calculation the distance between all charge groups has to be evaluated and compared with the cutoff length. Thus we have to multiply the number of clock cycles $c_{\text{dist}}$ of one distance calculation with the number $n$ of charge group pairs, which is obtained from equation 2.8.

The runtime for the pairlist calculation can be obtained with

$$t_{\text{pairlist}} = n \cdot \frac{c_{\text{dist}}}{f_{\text{SHARC}}},$$

(3.13)

The ADSP-2106x requires 226 seconds in C and 94 seconds in Assembler for calculating the pairlist in the rectangular Thrombin simulation. For the octahedral simulation, it requires 104 seconds in C and 34 seconds in Assembler for the pairlist.

The number of clock cycles for the non-bonded interaction algorithm between two solvent atoms is the sum of that of the distance and the force routine. For one time step, it must be multiplied with the number of charge group interactions, which is determined by the number of elements $N N_{vv}$ in the pairlist and the number of atom-atom interactions $n_{a}$ of one solvent charge group pair.

Thus the runtime for the interaction calculation is obtained with

$$t_{\text{interaction}} = \frac{N N_{vv} \cdot n_{a} \cdot (c_{\text{dist}} + c_{\text{force}})}{f_{\text{SHARC}}},$$

(3.14)

where $c_{\text{dist}}$ is the number of clock cycles of the distance routine and obtained with equation 3.12 and $c_{\text{force}}$ is that of the force routine, which is taken from tables 3.7 and 3.8. The parameter $N N_{vv}$ was derived by running the Thrombin simulation.
and amounts to $NN_{vv} = 1981463$ for the rectangular and $NN_{vv} = 916439$ for the octahedral simulation. As the solvent molecules are water molecules with 3 atoms each, there are $3 \cdot 3 = 9$ atom interaction and we have $n_a = 9$.

The interaction calculation in one time step requires 158 seconds in C and 64 seconds in Assembler for the rectangular and 86 seconds in C and 32 seconds in Assembler for the octahedral Thrombin simulation on an ADSP-2106x.

For a comparison of the SHARC with the other hardware solutions, we estimated the time for the Thrombin molecular dynamics simulation with 100 time steps. In the simulation, the SHARC would generate the pairlist for all charge groups and it would calculate the non-bonded solvent-solvent interactions.

Figure 3.37 shows the times of the different C and Assembler programming routines of the Thrombin simulation on the SHARC.

![Figure 3.37: Runtimes of the Thrombin molecular dynamics simulations with 100 time steps on the SHARC ADSP-2106x](image)

The force routine requires the most of the calculation time. The times of the calculation of the distances for the force routine is practically the same as that of the pairlist calculation. Thus the pairlist calculation takes relatively much time if we realize, that it is generated only every 5th time step.

The runtimes of the programming routines of the Thrombin molecular dynamic simulations are listed in table 3.11.
Programming Routine | Rectangular Simulation | Octahedral Simulation |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[minutes:seconds]</td>
<td>[minutes:seconds]</td>
</tr>
<tr>
<td>C</td>
<td>Assembler</td>
<td>C</td>
</tr>
<tr>
<td>Pairlist</td>
<td>75:20</td>
<td>34:33</td>
</tr>
<tr>
<td>Distance</td>
<td>89:36</td>
<td>63:16</td>
</tr>
<tr>
<td>Force</td>
<td>172:24</td>
<td>79:44</td>
</tr>
<tr>
<td>TOTAL</td>
<td>337:19</td>
<td>177:33</td>
</tr>
</tbody>
</table>

Table 3.11: Runtimes of the Thrombin molecular dynamics simulation with 100 time steps on the SHARC ADSP-2106x

If we compare the runtimes of the pairlist calculation of the two simulations, we see, that the effort raises quadratic with the number of charge groups used in the simulation. The cubic simulation has double the charge groups as the octahedral simulation, but the time for the pairlist calculation is four times longer. If we use Assembler-code we can save much time in both simulations.

The Thrombin molecular dynamics simulation takes a long time on a SHARC DSP processor. As described above, the ADSP-2106x reaches a theoretical performance of 40 MFlops. However, there has been a successor of the ADSP-2106x, which is the ADSP-21160, developed by Analog Devices in 1998. It has two sets of the three independent computation units: ALU, multiplier and fixed-point accumulator/shifter and its clock frequency has been increased to 100 MHz [79]. Using that processor for the GROMOS coprocessor, two distances and interactions between particle pairs could be calculated simultaneously. Thus the ADSP-21160 may be up to 5 times faster than the ADSP-2106x and reaches a theoretical peak performance of 200 MFlops.

We evaluated the performance of both, the ADSP-2106x and the ADSP-21160, for the pairlist and non-bonded solvent-solvent interaction routine by using the function model in section 2.3.3.4. As the processor calculates only the distances in the pairlist routine, the number of floating-point operations for the geometric centre are not considered.

The performance of the SHARC DSPs is listed in table 3.12.
3.6 Microprocessor Units for MD

<table>
<thead>
<tr>
<th>Programming Routine</th>
<th>Processor</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairlist generation</td>
<td>ADSP-2106x</td>
<td>11.42 MFlops</td>
<td>12.63 MFlops</td>
</tr>
<tr>
<td></td>
<td>ADSP-21160</td>
<td>57.10 MFlops</td>
<td>63.13 MFlops</td>
</tr>
<tr>
<td>Non-bonded solvent-solvent interaction calculation</td>
<td>ADSP-2106x</td>
<td>13.03 MFlops</td>
<td>17.55 MFlops</td>
</tr>
<tr>
<td></td>
<td>ADSP-21160</td>
<td>65.17 MFlops</td>
<td>87.76 MFlops</td>
</tr>
</tbody>
</table>

Table 3.12: Performance of the SHARC DSPs for the Thrombin simulation

The performance of the ADSP-2106x is a little better than that of a SUN SPARC Station 10 (see table 2.10). For a GROMOS coprocessor we should use the ADSP-21160, which is the newer version. But we must consider, that we require a high effort for implementing the pairlist and the non-bonded solvent-solvent interaction calculation, because Assembler should be used to reach such a high performance. As the SPARC 10 is relatively old, we may achieve the same performance with fewer effort on an actual available MPU by using a standard Fortran compiler.

3.6 Microprocessor Units for MD

Another way for implementing the GROMOS coprocessor is to use standard microprocessors, which are built in workstations. The major advance is, that there are operating systems and compilers available, which require only minor adaptations of the GROMOS™ software. Thus the effort for the implementation of GROMOS may be very low and new versions could be ported to the coprocessor in a short time. In addition, MPUs are available for a good price, because of their wide spread.

We tested the performance of MPUs for GROMOS by running GROMOS96 on different workstations and measuring the runtimes of the most substantial routines. As GROMOS96 is written for UNIX systems, we took workstations, where UNIX systems or their derivatives LINUX, IRIX or OSF1 had been installed. With this, no changes of the code were necessary and we had only to compile the source code of GROMOS with the appropriate Fortran77 compiler on the respective workstation.

For the performance tests, we ran GROMOS96 on the following workstations:
S-S10: SUN SPARC Station 10/85 with 85 MHz SuperSPARC CPU, 192 KB Cache, 64 MB RAM
S-U1: SUN Ultra 1/170 with 167 MHz UltraSPARC CPU, 256 KB Cache, 192 MB RAM
S-U30: SUN Ultra 30/300 with 296 MHz UltraSPARC CPU, 256 KB Cache, 256 MB RAM
K6: AMD PC with 200 MHz AMD K6 CPU, 512 KB Cache, 128 MB RAM
PPRO: Dell GXPro PC with 200 MHz Intel Pentium Pro CPU, 256 KB Cache, 64 MB RAM
PII233: Vobis Highscreen PC with 233 MHz Intel Pentium II CPU, 512 KB Cache, 64 MB RAM
PII350: Dell GX1 PC with 350 MHz Intel Pentium II CPU, 512 KB Cache, 64 MB RAM
PII400: Dell Precision 410 PC with 400 MHz Intel Pentium II CPU, 512 KB Cache, 128 MB RAM
A-064: DEC Alpha Station 400 4/233 with 233 MHz Alpha 21064 CPU, 512 KB Cache, 96 MB RAM
A-164: DEC Personal Workstation 500 with 500 MHz Alpha 21164 CPU, 2 MB Cache, 128 MB RAM
SGI: Silicon Graphics Octane with 195 MHz MIPS R10000 CPU Chip Rev. 2.7, 1 MB Cache, 128 MB RAM

We used the following UNIX systems and Fortran77 Compilers:

SUNs: Solaris 2.5.1, SUN Fortran77 4.2
PCs: SuSe Linux 5.2 (2.0.35), Gnu Fortran77 0.5.19.1
ALPHAs: DEC Unix 4.0D, DEC Fortran77 4.0
SGI: Irix 4.6, MIPSpro Fortran77 7.2.1.2m

Each compiler has various compiler options for the optimisation of the code. Because the MPUs have different structures, they respond in various manners to the options. To achieve the highest performance on each MPU, we tested the different options to find the right ones and compiled GROMOS96 with these options for the tests.
The optimal compiler options for each MPU are the following:

**SUN SPARC:** -O4 -unroll=4 -depend -native  
**AMDK6:** -O2 -ffast-math -unroll=4  
**Intel Pentium Pro:** -O5 -ffast-math -fcse-follow-jumps  
**Intel Pentium II:** -O5 -ffast-math -frerun-cse-after-loop  
**ALPHA 21064:** -O5 -fast -unroll 4  
**ALPHA 21164:** -O4 -fast -unroll 4 -u  
**MIPS R10000:** -O3 -LNO:ou_max=4

The best performances have been achieved by using the automatic loop unrolling (-unroll=4) for a maximum of fourfold loops, the use of faster mathematical libraries, special conditional branch predictions and the overall optimising option (-Ox).

We tested the workstations with the rectangular and the octahedral Thrombin simulations. The results are shown in figures 3.38 and 3.39:

*Figure 3.38: Runtimes of the rectangular Thrombin simulation on different workstations*
Figure 3.39: Runtimes of the octahedral Thrombin simulation on different workstations

The results show that the Alpha 21164 achieves the highest speed for GROMOS96 for both simulation topologies. It is 5-6 times faster than the slowest SUN SPARC Station 10/85. The SGI is only slightly slower than the Alpha.

The routines of the simulations have overall the same percentages on all workstations. The only discrepancy shows the SPARC 10 in the rectangular simulation, where the non-bonded interaction calculation of solute-solute and solute-solvent has a percentage of 25%, whereas the other have an average of 10%. The same routine has discrepancies on the Alpha 21064 in the octahedral simulation.

If we compare the results of the SGI and the Alpha 21164, we see, that the latter has the higher percentage in the pairlist routine, where also the calculation takes a bit longer than that of the SGI, whereas the Alpha requires fewer time for the non-bonded interaction routines. As these routines are sourced out to the coprocessor, we estimated the floating-point performance of the Alpha and the SGI for the pairlist and non-bonded solvent-solvent interaction calculation by using the function model, which we have made in section 2.3.3.4. The performance of the processors is listed in table 3.13.

Although the SGI has the higher performance for the pairlist generation, we prefer the Alpha for the GROMOS coprocessor, because it is faster for the non-bonded interaction calculation, which has the higher percentage in the simulation. The pairlist is only generated every 5th-20th time step and considers only charge groups,
whereas the non-bonded interactions have to be calculated in each time step for every single atom. In addition, the Alpha processor has the lower cost than the MIPS R10000 of the SGI.

<table>
<thead>
<tr>
<th>Programming Routine</th>
<th>Workstation</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairlist generation</td>
<td>A-164</td>
<td>44.10 MFlops</td>
<td>47.63 MFlops</td>
</tr>
<tr>
<td></td>
<td>SGI</td>
<td>55.28 MFlops</td>
<td>56.32 MFlops</td>
</tr>
<tr>
<td>Non-bonded solvent-solvent interaction calculation</td>
<td>A-164</td>
<td>61.78 MFlops</td>
<td>85.34 MFlops</td>
</tr>
<tr>
<td></td>
<td>SGI</td>
<td>56.66 MFlops</td>
<td>79.05 MFlops</td>
</tr>
</tbody>
</table>

Table 3.13: Performance of the SGI and Alpha for the Thrombin simulation

The major advance for using an MPU is, that practically no hardware must be developed, due to the already existing hardware solutions with MPUs. A further gain is the development of faster MPUs and processor boards, which allows a renewal of the coprocessor by an easy change of components, due to the introduced standard.

### 3.7 Summary

In this chapter we evaluated different hardware solutions for a GROMOS coprocessor hardware. Their characteristics and performance are summarised in table 3.14.

<table>
<thead>
<tr>
<th>Hardware Solution</th>
<th>Performance</th>
<th>Flexibility</th>
<th>Effort</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing MD Hardware</td>
<td>1 Gflops - 200 Tflops</td>
<td>middle</td>
<td>middle</td>
<td>high</td>
</tr>
<tr>
<td>Parallel Computers</td>
<td>1.7 - 555 MFlops</td>
<td>middle</td>
<td>high</td>
<td>very high</td>
</tr>
<tr>
<td>ASIC</td>
<td>~ 9 GFlops</td>
<td>low</td>
<td>middle</td>
<td>high</td>
</tr>
<tr>
<td>FPGA</td>
<td>54.4 MOPS</td>
<td>high</td>
<td>middle</td>
<td>middle</td>
</tr>
<tr>
<td>DSP</td>
<td>57.1 MFlops</td>
<td>middle</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>MPU</td>
<td>44.1 MFlops</td>
<td>high</td>
<td>low</td>
<td>low</td>
</tr>
</tbody>
</table>

Table 3.14: Characteristics and performance of different hardware solutions
The performances of the DSP and MPU were taken from the non-bonded solvent interaction calculation of the rectangular Thrombin simulation and those of the existing MD hardware and parallel computers are the ones of other MD simulations.

Specialized MD hardware and parallel computers offer the highest performance. The problem of the MD hardware is, that not all required functions of GROMOS are implemented and the lacking ones should be calculated on the host computer, which results in a performance loss. Parallel computers are huge and very expensive machines, which require a high implementation effort.

The use of an ASIC would offer a high performance but is inflexible for changes of the GROMOS algorithm. In addition the production of an ASIC is quite expensive. However, an FPGA would offer more flexibility than an ASIC but today's available FPGAs are too small for implementing the most intensive parts of the GROMOS algorithm.

As shown in section 3.5.2, a good performance can be reached by using a SHARC ADSP-21160 processor, but if we compare its speed with that of an MPU, there is not much difference. As the implementation effort for a DSP is higher due to the requirement of Assembler programming, we must prefer an MPU for our coprocessor.

The investigations in section 3.6 show, that an Alpha 21164 processor offers the highest performance of all tested MPUs. However, an Alpha 21164 workstation with 500 MHz is only 5 times faster than our reference machine, the SUN SPARC 10. In addition, there are already faster workstations available. To offer a performance gain with the use of our GROMOS coprocessor, we shall use multiple Alpha processors in parallel.

The use of Alpha processors does not require a self-development of hardware even if more of them are used in parallel. There are CPU-boards with one up to 4 Alphas available, which include a standard memory bus or PCI interface. Our coprocessor could be built by assembling several of these boards together to a parallel machine. Another possibility is the use of standard Alpha motherboards with one or more processors. Multiple of them could be clustered over a fast bus-system or a high-performance network.

We have now found the most adequate hardware solution for our coprocessor by connecting multiple Alpha 21164 processor boards to a parallel computer cluster. But the interconnection between the Alpha processors is not defined yet.

In the next chapter we evaluate the communication requirements and define the parallel processor structure for our coprocessor.
In a multi-processor system, the communication between the processor units plays an important role. As the best solution for our GROMOS coprocessor is a parallel system of standard microprocessor units, the communication in a parallelised GROMOS algorithm has to be analysed to find the right communication network. The chapter begins with a discussion of different communication aspects in a parallel processing system. In the following, various network topologies and communication methods are investigated by using a simple model of the N-body calculation problem. The last part gives an overview of available off-the-shelf standard networks to choose the right one for our parallel coprocessor system.
4 Communication

4.1 Parallelism

Using a computer hardware in parallel is a common way for enhancing speed of an application. If the algorithm can be parallelised and distributed over more platforms, which is possible with GROMOS, the speed of the chosen hardware solution becomes less important and the characteristics of flexibility and implementation effort take the more important role.

We have seen in chapter 3, that the use of general purpose microprocessors offer the best compromise between speed, flexibility and implementation effort. However, this is not the best hardware solution concerning speed, but using parallelism, a higher speed can be achieved and we obtain a fast coprocessor for accelerating GROMOS, which offers a high flexibility and requires low effort for the implementation of the software.

In each parallel system communication is required for exchanging data and results between the processor units, which plays an important role in multi-processor systems. The influence of the communication on the performance of a parallel algorithm grows in general with the number of processor units [80]. This limits in principle the maximum possible number of processors, which can be used for the algorithm. If the efficiency of the communication is bad, the communication overhead is increased and leads to a bad speedup of the parallel algorithm. Thus we have to make the communication as efficient as possible to achieve a high speedup of the GROMOS algorithm on our parallel computer cluster.

The efficiency of the communication in a parallel system is determined by three factors [81]:
1. The data distribution and communication method in the parallel algorithm
2. The structure of the parallel processor system
3. The performance of the network

The first two points are related to each other. On the one hand the structure of the parallel processor system should be chosen by the manner of the data communication in the parallel algorithm. On the other hand the communication method should be adapted to the structure of the parallel processor system.

The performance of the network is dependent of the latency and the throughput of the data. The throughput is determined by the width of the data lines and the network frequency whereas the latency is determined by the frequency of the collisions on the network and the network topology.
As the network performance is determined by the topology of the multi-processor, which is related to the data communication method, we have to find first the suitable structure of our parallel coprocessor system. Because the structure is mainly given by the construction of the Alpha processor boards, our work is now to determine the adequate network topology for our cluster.

4.2 Topologies for the Parallel Processor System

There exist a large variety of network topologies for parallel processor systems [82]. To find the right one for our cluster, we estimated the communication overhead and compared the speedup of parallel systems with different topologies in respect of the number of processors. For the investigations we modelled the Alpha computers as simple PEs and used the N-body calculation problem, which is often used in MD simulations and also for the pairlist calculation in GROMOS.

Each PE consists of the processor, a memory and a communication node. The processor can either access its own memory or the memory of any other PE through its communication node. We assume, that the communication node can directly access data from the memory without interrupting the processor and it can send data to a PE and simultaneously request data from another PE.

Our model uses distributed data, which means, that the coordinates of the particles are only stored in the memory of one single PE. Locally not available coordinates are requested through the communication network from the corresponding PE and are immediately used by the requesting processor for the computation. The results are stored in the local memory of the PE, where they have been calculated and the coordinates of the requested particles are discarded.

The performance of the PEs are modelled in such a way, that one processor is able to calculate the interaction between two particles in one communication step, which means, that the processor requires the data of one particle in each cycle.

The number of communication cycles is determined by the communication latency, which is measured in clock cycles. We assume the data width in such a way, that all coordinates of a particle can be sent within one cycle.

In the N-body pairlist calculation problem, the interactions between all pairs of particles are calculated. Due to Newton's third axiom [8] of pairwise forces, nested loops (see figure 2.11) are used to avoid a double calculation of the interactions. This means, that the first particle has to be tied with all others, the second must be tied with all others except the first, the third must be tied with all others except the first and the second and so on (see figure 4.1).
4.2 Topologies for the Parallel Processor System

In parallelising the N-body pairlist calculation problem, we have to distribute the particles evenly to all PEs to ensure an even load balancing for the calculation. As the particles must be exchanged between all processors, we have to find a communication method for an efficient use of the underlying network topology, which must meet the following requirements:

- There should be preferably used direct connections between the PEs, because sending data from one PE to another over several nodes requires more communication time.
- The communication should occur in a way, that collisions are avoided.
- Connections should be shared by many processors to keep the network simple.

For each network topology, we took the most capable communication method of the parallel N-body problem to estimate the communication overhead. This allows, to compare the efficiency of the network topologies, to find the most suitable one for the N-body problem.

We compare our estimations with an ideal speedup, where the communication time is 0 and the load is evenly balanced on all PEs. The computation time on an ideal parallel processor is

\[
T = \frac{1}{P} \left( \sum_{i=1}^{N-1} i \right) = \frac{1}{2P} (N - 1) \cdot N, \tag{4.1}
\]

where P is the number of PEs and N is the number of particles.

The investigations of different network topologies and communication methods are described in the following sections.
4.2.1 Bus Topology

![Bus Topology Diagram](image)

The bus is the simplest architecture, but implies also the biggest communication overhead if the particles are equally distributed to the PEs. For this case it is not possible to have computation and communication in parallel. Therefore we have not further analysed this case.

The best method for the bus is to store all particles on all PEs. With this, the particles must be distributed over all PEs at the beginning of the calculation and the bus is not used during computation. This model allows for almost linear speedup with the number of processors but at the cost of memory.

Because there is no communication during computation the communication time is 0 as for the ideal speedup. The computation time for a bus system is:

\[
T = \left[ \frac{1}{P} \sum_{i=1}^{N-1} i \right] = \left[ \frac{1}{2P} \cdot N \cdot (N-1) \right] \tag{4.2}
\]
4.2.2 2D-Net

With a 2D-net including diagonal connections every processor can simultaneous read data from any other neighbouring PEs memory (see figure 4.3). All communication links are assumed to be full duplex.

If we investigate the N-body pairlist calculation problem, we see, that every processor must first compare all particles in its local memory. Then all particles from its local memory must be compared with all particles from the other PEs. To avoid double calculations, every processor must only compare half of the particles in its local memory with all particles on the other PEs to avoid double calculating. For a better understanding, we split the particles in a PEs memory into two sets A and B. Each set of every PE must be compared with all other sets of the same and the other PEs.

With this a computation scheme can be found, that implies no communication conflicts. Table 4.1 shows, how the calculation is done and from which memory each processor takes the particles:
The available communication bandwidth is not fully utilized in table 4.1. With the 6 full duplex links 12 data sets can be exchanged simultaneously among PEs. In the above computation scheme only 4 connections are used simultaneously. If we only allow half duplex links, the following computation scheme can be found.

<table>
<thead>
<tr>
<th>Comp. Step</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0A - 0A</td>
<td>1A - 1A</td>
<td>2A - 2A</td>
<td>3A - 3A</td>
</tr>
<tr>
<td>2</td>
<td>0B - 0B</td>
<td>1B - 1B</td>
<td>2B - 2B</td>
<td>3B - 3B</td>
</tr>
<tr>
<td>3</td>
<td>0A - 0B</td>
<td>1A - 1B</td>
<td>2A - 2B</td>
<td>3A - 3B</td>
</tr>
<tr>
<td>4</td>
<td>0A - 1A</td>
<td>1B - 1B</td>
<td>2A - 3B</td>
<td>3B - 2B</td>
</tr>
<tr>
<td>5</td>
<td>0A - 1B</td>
<td>1A - 0B</td>
<td>2A - 3B</td>
<td>3A - 2B</td>
</tr>
<tr>
<td>6</td>
<td>0A - 2A</td>
<td>1A - 0B</td>
<td>2B - 3B</td>
<td>3B - 2B</td>
</tr>
<tr>
<td>7</td>
<td>0A - 2B</td>
<td>1A - 3B</td>
<td>2B - 3B</td>
<td>3A - 1B</td>
</tr>
<tr>
<td>8</td>
<td>0A - 3A</td>
<td>1A - 2B</td>
<td>2B - 1B</td>
<td>3B - 0B</td>
</tr>
<tr>
<td>9</td>
<td>0A - 3B</td>
<td>1A - 2B</td>
<td>2A - 1B</td>
<td>3A - 0B</td>
</tr>
</tbody>
</table>

Table 4.1: Computation scheme with 4 PEs in a 2D-net with 6 full duplex links

In table 4.2 also 4 connections are used simultaneously. The results show, that no connections are utilized in the first 3 computation steps. During the 9 computation
steps only 24 connections are set up in total. Without diagonal communication links, two adjacent communication links must be used to support one diagonal transaction. Since there are 8 diagonal transactions, we require $24 + 8 = 32$ connections for the whole calculation.

![2D-net with 4 communication links](image)

*Figure 4.4: 2D-net with 4 communication links*

Table 4.3 shows the computation scheme if only 4 halfduplex links are available.

<table>
<thead>
<tr>
<th>Comp. Step</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0A - 0A</td>
<td>1A - 1A</td>
<td>2A - 2A</td>
<td>3A - 3A</td>
</tr>
<tr>
<td>2</td>
<td>0A - 0B</td>
<td>1A - 2A</td>
<td>2B - 1B</td>
<td>3A - 3B</td>
</tr>
<tr>
<td>3</td>
<td>0B - 0B</td>
<td>1A - 2B</td>
<td>2A - 1B</td>
<td>3B - 3B</td>
</tr>
<tr>
<td>4</td>
<td>0A - 3A</td>
<td>1B - 1B</td>
<td>2B - 2B</td>
<td>3B - 0B</td>
</tr>
<tr>
<td>5</td>
<td>0A - 3B</td>
<td>1A - 1B</td>
<td>2A - 2B</td>
<td>3A - 0B</td>
</tr>
<tr>
<td>6</td>
<td>0A - 1A</td>
<td>1B - 3B</td>
<td>2B - 0B</td>
<td>3A - 2A</td>
</tr>
<tr>
<td>7</td>
<td>0A - 1B</td>
<td>1A - 3B</td>
<td>2A - 0B</td>
<td>3A - 2B</td>
</tr>
<tr>
<td>8</td>
<td>0A - 2A</td>
<td>1B - 0B</td>
<td>2B - 3B</td>
<td>3A - 1A</td>
</tr>
<tr>
<td>9</td>
<td>0A - 2B</td>
<td>1A - 0B</td>
<td>2A - 3B</td>
<td>3A - 1B</td>
</tr>
</tbody>
</table>

*Table 4.3: Computation scheme with 4 PEs in a 2D-net with 4 half duplex links*
With the 2D-net and 6 communication links, the load balancing is regular in each computation step. If we only have 4 links, the load balancing is uneven. In table 4.3, at step 3 processors 0 and 3 calculate particles both within the second half of their local memory, where processors 1 and 2 calculate particles from their local and from other memories. Therefore processors 1 and 2 have to calculate more interactions than processors 0 and 3 in step 2. In step 4 it is the same but vice versa.

However, the communication links are only used, when a new particle from another PE has to be fetched. After that, the network is not used. Therefore the processors must not be halted, if load balancing is uneven at a computation step. They can already begin to calculate the interactions for the next step. In this way, the load balancing over the whole computation becomes even, because all processors have to calculate the same number of interactions during the whole computation.

The calculation time for a 2D-net can be computed as follows:

$$ T_{cal} = \frac{N - 1}{P} \sum_{i=1}^{N} i + \frac{1}{2} \cdot N \cdot \left( \frac{1}{P} \right) N $$ \hspace{1cm} (4.3)

For the communication time only the number of particles must be considered, which are transferred through the network. This number has to be multiplied with the communication latency $l$. The communication time in a 2D-net results in:

$$ T_{com} = l \cdot \left( 1 - \frac{1}{P} \right) N $$ \hspace{1cm} (4.4)

The whole computation time for the 2D-net is:

$$ T = T_{cal} + T_{com} = \frac{1}{2} \left( \frac{N}{P} - 1 \right) + \left( \frac{N}{2P} + i \right) \left( 1 - \frac{1}{P} \right) N $$ \hspace{1cm} (4.5)

A 2D-net supports up to 4 PEs. If more are required, the network will become a bottleneck. In the next section we investigate network topologies, which support more processors.
4.2.3 Hyper Cube

In a hyper cube only the PEs on the edges at one side are directly connected. Transfers of data from across the diagonal have to be routed through another PEs. The communication links are assumed to be half duplex as in the 2D-net.

Let us consider a system with 8 PEs. The particles are divided into two sets again as in section 4.2.2. One set on a processor has to be compared with all other sets on the other PEs, which takes $7 \cdot 2 = 14$ computation steps. In addition each processor has to compare all particles within its local memory. This takes 3 computation steps since each of the two sets has to be compared with itself and with each other. Therefore it takes 17 computation steps to calculate all interactions.

We can now calculate the required connections. 17 computation steps and 12 communication links are available. This results in totally 204 available connections for the entire computation. It takes two communication links at a time for fetching data from a diagonally placed PE on the same plane and three communication links for fetching data from a PE, which is placed diagonally across the cube. Determin-
ing the required links, we get $8 \cdot (2 \cdot 3 \cdot 1 + 2 \cdot 3 \cdot 2 + 2 \cdot 1 \cdot 3) = 192$ connections for the entire computation. It should theoretically be possible to develop a computation method without communication collisions. But the problem is, that $192/17 = 11.3$, which means, that all communication links should be used in every computation step. However, a computation method, which utilizes all 12 communication links simultaneously, cannot be found.

Therefore we have added 4 diagonal communication links as shown in figure 4.5. This reduces the required number of connections to $8 \cdot (2 \cdot 3 \cdot 1 + 2 \cdot 3 \cdot 2 + 2 \cdot 1 \cdot 1) = 160$ for the entire computation. This means, that $160/17 = 9.4$, which allows to find a computation scheme with the now available 16 communication links.

Table 4.4 shows the computation scheme:

<table>
<thead>
<tr>
<th>Comp. Step</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0A - 0A</td>
<td>1A - 1A</td>
<td>2A - 2A</td>
<td>3A - 3A</td>
<td>4A - 4A</td>
<td>5A - 5A</td>
<td>6A - 6A</td>
<td>7A - 7A</td>
</tr>
<tr>
<td>2</td>
<td>0A - 0B</td>
<td>1A - 4A</td>
<td>2A - 2B</td>
<td>3A - 6A</td>
<td>4B - 1B</td>
<td>5A - 5B</td>
<td>6B - 3B</td>
<td>7A - 7B</td>
</tr>
<tr>
<td>3</td>
<td>0B - 0B</td>
<td>1A - 4B</td>
<td>2B - 2B</td>
<td>3A - 6B</td>
<td>4A - 1B</td>
<td>5B - 5B</td>
<td>6A - 3B</td>
<td>7B - 7B</td>
</tr>
<tr>
<td>4</td>
<td>0A - 5A</td>
<td>1B - 1B</td>
<td>2A - 7A</td>
<td>3B - 3B</td>
<td>4B - 4B</td>
<td>5B - 0B</td>
<td>6B - 6B</td>
<td>7B - 2B</td>
</tr>
<tr>
<td>5</td>
<td>0A - 5B</td>
<td>1A - 1B</td>
<td>2A - 7B</td>
<td>3A - 3B</td>
<td>4A - 4B</td>
<td>5A - 0B</td>
<td>6A - 6B</td>
<td>7A - 2B</td>
</tr>
<tr>
<td>6</td>
<td>0B - 2A</td>
<td>1B - 0B</td>
<td>2B - 3B</td>
<td>3A - 1A</td>
<td>4B - 5B</td>
<td>5A - 7A</td>
<td>6A - 4A</td>
<td>7B - 6B</td>
</tr>
<tr>
<td>7</td>
<td>0A - 2B</td>
<td>1A - 0B</td>
<td>2A - 3B</td>
<td>3A - 1B</td>
<td>4A - 5B</td>
<td>5A - 7B</td>
<td>6A - 4B</td>
<td>7A - 6B</td>
</tr>
<tr>
<td>8</td>
<td>0A - 4A</td>
<td>1B - 3B</td>
<td>2B - 0B</td>
<td>3A - 7A</td>
<td>4B - 6B</td>
<td>5A - 1A</td>
<td>6A - 2A</td>
<td>7B - 5B</td>
</tr>
<tr>
<td>9</td>
<td>0A - 4B</td>
<td>1A - 3B</td>
<td>2A - 0B</td>
<td>3A - 7B</td>
<td>4A - 6B</td>
<td>5A - 1B</td>
<td>6A - 2B</td>
<td>7A - 5B</td>
</tr>
<tr>
<td>10</td>
<td>0A - 1A</td>
<td>1B - 6B</td>
<td>2A - 4A</td>
<td>3A - 5A</td>
<td>4B - 2B</td>
<td>3B - 5B</td>
<td>6A - 7A</td>
<td>7B - 0B</td>
</tr>
<tr>
<td>11</td>
<td>0A - 1B</td>
<td>1A - 6B</td>
<td>2A - 4B</td>
<td>3A - 5B</td>
<td>4A - 2B</td>
<td>3B - 5A</td>
<td>6A - 7B</td>
<td>7A - 0B</td>
</tr>
<tr>
<td>12</td>
<td>0A - 3A</td>
<td>1B - 5B</td>
<td>2B - 6B</td>
<td>3B - 0B</td>
<td>4A - 7A</td>
<td>5A - 2A</td>
<td>6A - 1A</td>
<td>7B - 4B</td>
</tr>
<tr>
<td>13</td>
<td>0A - 3B</td>
<td>1A - 5B</td>
<td>2A - 6B</td>
<td>3A - 0B</td>
<td>4A - 7B</td>
<td>5A - 2B</td>
<td>6A - 1B</td>
<td>7A - 4B</td>
</tr>
<tr>
<td>14</td>
<td>0A - 6A</td>
<td>1A - 7A</td>
<td>2B - 5B</td>
<td>3A - 2A</td>
<td>4B - 3B</td>
<td>5A - 4A</td>
<td>6B - 0B</td>
<td>7B - 1B</td>
</tr>
<tr>
<td>15</td>
<td>0A - 6B</td>
<td>1A - 7B</td>
<td>2A - 5B</td>
<td>3A - 2B</td>
<td>4A - 3B</td>
<td>5A - 4B</td>
<td>6A - 0B</td>
<td>7A - 1B</td>
</tr>
<tr>
<td>16</td>
<td>0A - 7A</td>
<td>1A - 2A</td>
<td>2B - 1B</td>
<td>3A - 4A</td>
<td>4B - 0B</td>
<td>5A - 6A</td>
<td>6B - 5B</td>
<td>7B - 3B</td>
</tr>
<tr>
<td>17</td>
<td>0A - 7B</td>
<td>1A - 2B</td>
<td>2A - 1B</td>
<td>3A - 4B</td>
<td>4A - 0B</td>
<td>5A - 6B</td>
<td>6A - 5B</td>
<td>7A - 3B</td>
</tr>
</tbody>
</table>

Table 4.4: Computation scheme with 8 PEs in a hyper cube
As in table 4.3, the load balancing is not equal in all computation steps, but the processors have not to be halted after each step for the same reason as discussed in section 4.2.2. This makes the load balancing even over the whole computation.

Due to the same number of particles are transferred through the network as with the 2D-net, the computation time of the hyper cube can also be calculated with equations 4.3, 4.4, and 4.5.

An extension of a hyper cube to more processors is difficult. 16 processors would result in a 4D-net, 32 processors require a 5D-net and so on. It is expensive to realize such a system physically for a high number of processors.

### 4.2.4 Recursive Topology

![Recursive Topology Diagram]

*Figure 4.6: Recursive topology*
A multiple processor architecture could e.g. recursively be built from a two dimensional network topology as shown in figure 4.6. In this approach one PE is a cluster with 4 PEs. The computation method on the PEs in a cluster is the same as in table 4.3 of the 2D-net. Additionally each communication node is connected to a global node, which is itself connected to the global communication net, on which again the same computation method is used as on the processor clusters.

The following table shows a possible computation scheme. It is a recursive extension of table 4.3 of the 2D-net and is simplified to improve readability. Only the number of the processor memory from which the second particle is fetched, is listed, because the first particle is always fetched from the own processor memory. One should remember, that each of the processor considers only half of the particles from its local memory. This is not listed in table 4.5 to leave it clearly arranged.

<table>
<thead>
<tr>
<th>Comp. Step</th>
<th>P 0</th>
<th>P 1</th>
<th>P 2</th>
<th>P 3</th>
<th>P 4</th>
<th>P 5</th>
<th>P 6</th>
<th>P 7</th>
<th>P 8</th>
<th>P 9</th>
<th>P 10</th>
<th>P 11</th>
<th>P 12</th>
<th>P 13</th>
<th>P 14</th>
<th>P 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>11</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>9</td>
<td>11</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>14</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4.5: Computation scheme with 16 PEs in a recursive topology
4.2 Topologies for the Parallel Processor System

With the recursive topology the calculation time remains the same as with the other topologies:

\[ T_{cal} = \left( \frac{N}{p} - 1 \right) \sum_{i=1}^{p} i + \frac{N}{2p} \cdot \left( 1 - \frac{1}{p} \right)^N \]  (4.6)

Concerning the communication time we have now to consider two different latencies. One is the local latency \( l_i \) for fetching data in the same processor cluster over the local network and the second is the global latency \( l_g \) for fetching data from another processor cluster over the global nodes and global network, which takes more time.

For the communication time in a recursive topology follows

\[ T_{com} = l_i \cdot \left( \frac{1}{Q} - \frac{1}{P} \right)^N + l_g \cdot \left( 1 - \frac{1}{Q} \right)^N \]  (4.7)

where \( P \) stands for the number of processors and \( Q \) is the number of processor clusters.

The total computation time for a recursive topology is

\[ T = T_{cal} + T_{com} = \frac{1}{2} \left( \frac{N}{P} - 1 \right) \left( \frac{N}{P} \right) + \frac{N}{2p} \left( 1 - \frac{1}{P} \right) + l_i \left( \frac{1}{Q} - \frac{1}{P} \right) + l_g \left( 1 - \frac{1}{Q} \right) \left( \frac{N}{P} \right) \]  (4.8)

The advantage of this network topology is, that it can simply be extended up to a system with 64 processors. In this case, the global topology would be a hyper cube and every processor cluster would be again a hyper cube itself. A system with 32 processors could either be a 2D-net with hyper cube processor clusters or a hyper cube with 2D-net processor clusters. Concerning the communication time, the former is better, since only 3/4 of the communication is done over the global network, compared to 7/8 in the latter.

The recursive topology can be refined into more recursive levels such as you have for example in a 2D-net global network with 4 processor systems, where every processor system itself being a hyper cube with 8 processors clusters and every processor cluster is a 2D-net again with 4 PEs. In this case, the communication latency would increase to an unacceptable level.
4.2.5 Ring

Because on a ring it is not possible to transfer data directly from one PE to a not adjacent PE without intermediate communication steps, the method of the systolic loops (see section 2.1.3.3) has to be taken for the computation method. If we look at the computation schemes of the other topologies, we see, that e.g. the second part of the particles of PE 0 is sent to all other PEs. As the particles are temporarily stored on the PEs, on which they are calculated, they have not to be sent directly from PE 0 to one of the other PEs at the beginning of each computation step, they can directly be forwarded from the PE, on which they are calculated, to the next PE.

Table 4.6 shows the computation method. All particles are sent directly from the preceding PE, where they had been processed in the previous computation step. The scheme is made with the topology of figure 4.7.

Figure 4.7: Ring topology
The particles of a PE must only be sent over half of the ring. In table 4.6, the memory is still split into two parts and every part is sent separately over the ring. A better method is to send all particles together over half of the ring. If the number of processors is even, only half of the particles have to be compared in the last computation step because the two processors, which are opposite over the ring have the particles from each other. E.g. in figure 4.7, these are processors 0 and 3. If the number of processors is odd, the particles of the processors have only to be sent over half of the ring minus one. In this case, all particles are compared in the last computation step.

The calculation time is the same as in equation 4.3 and the communication time is the same as in equation 4.4. With this, the computation time is the same as for the 2D-net (eq. 4.5).

The cost of the circuit are much lower for a ring compared to the other topologies, because data on the ring must be sent only in one direction and therefore all links are unidirectional. In this case, the communication time is shorter than in the other topologies. Another advantage of this topology is the simple expansion possibility. The ring can easily be extended with more processors.
4.2.6 Other Topologies

The N-body pairlist calculation problem requires an exchange of data between all processors and therefore a topology, where all processors have a similar distance. Thus linear arrays, meshes and trees are basically not suitable for our processor cluster, because the outer located processors respectively the nodes on the bottom of the tree have the larger distances than the others.

Further a star is not capable for the N-body problem, because the PE, which is in the middle, indicates a bottleneck, when data are exchanged between the outer PEs.

However, a completely connected topology, where each processor is connected with each other, would deliver enough performance, but the utilization of the connections may be bad in the N-body problem and this network complicates drastically for a higher number of processors, which is not practical.

Therefore we have not considered these topologies for our investigations.

4.2.7 Comparison

As shown above the calculation time is the same for all topologies. Thus we only have to compare the efficiency of the network topologies with respect to its communication time. Most of the computation time is used by the calculation itself and not for the communication. Therefore the communication time is less important, when more particles have to be calculated. In our investigations, we took a N-body simulation with 1000 and 10’000 particles. We compared parallel processor systems with up to 64 processors.

For the estimation of the communication efficiency, we defined the latencies of the different network topologies as follows:
4.2 Topologies for the Parallel Processor System

<table>
<thead>
<tr>
<th>Network Topology</th>
<th>Network Latency</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>$l=1$</td>
<td>The simplest architecture. The network is not used during computation.</td>
</tr>
<tr>
<td>2D-Net</td>
<td>$l=3$</td>
<td>Communication relatively simple. Every node has to handle two communication links and memory access.</td>
</tr>
<tr>
<td>Hyper Cube</td>
<td>$l=3$</td>
<td>This is the 3D-expansion of the 2D-net. The distances between the PEs are not larger than that on the 2D-net, which results in the same latency.</td>
</tr>
<tr>
<td>Recursive Structure</td>
<td>$l=3$, $l_g=5$</td>
<td>The local communication is the same as in the 2D-net, because we have 4 clusters with 4 processors. The global communication is the sum of the local communication and that of the 2D-net minus 1, then the global communication nodes can access directly via the local communication nodes.</td>
</tr>
<tr>
<td>Ring</td>
<td>$l=2$</td>
<td>Communication is very simple, because you have only unidirectional communication links.</td>
</tr>
</tbody>
</table>

*Table 4.7: Communication latencies of the different network topologies*

The speedups, which may be achieved on parallel processor systems with the different network topologies, gives a good information about the efficiency of the communication. A better communication efficiency causes fewer performance loss, which results in a higher speedup.

We obtained the speedups by evaluating the computation times with equations 4.2, 4.5 and 4.8 and the parameters of table 4.7. The results for a N-body simulation with 1000 and 10'000 particles are shown in figures 4.8 and 4.9:
Figure 4.8: Speedup on parallel processor systems with the different network topologies for a N-body simulation with 1000 particles.

Figure 4.9: Speedup on parallel processor systems with the different network topologies for a N-body simulation with 10’000 particles.
The communication loss is the smallest on a ring architecture. It is larger on a 2D-net and a hyper cube and is the largest on a recursive structure. Additionally we learn that the communication loss is lower for simulations with more particles.

The advantages and disadvantages of the different network topologies are shown in table 4.8:

<table>
<thead>
<tr>
<th>Property</th>
<th>Bus</th>
<th>2D-Net</th>
<th>Hyper Cube</th>
<th>Recursive Structure</th>
<th>Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation Distribution</td>
<td>complex</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
</tr>
<tr>
<td>Data Distribution / Memory Use</td>
<td>bad</td>
<td>good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>Computation Scheme</td>
<td>not used</td>
<td>complex</td>
<td>very complex</td>
<td>complex</td>
<td>simple</td>
</tr>
<tr>
<td>Speedup</td>
<td>ideal</td>
<td>medium</td>
<td>medium</td>
<td>bad</td>
<td>good</td>
</tr>
<tr>
<td>Expansion Possibility</td>
<td>simple</td>
<td>difficult</td>
<td>difficult</td>
<td>medium</td>
<td>simple</td>
</tr>
</tbody>
</table>

Table 4.8: Characteristics of the different network topologies

The best speedup can be reached by using the simple bus topology, but only if the data are replicated on all PEs. Otherwise, the communication overhead results in a performance loss.

In this case, the ring offers the best compromise for the N-body calculation problem. Its topology is simple, few memory is used and an expansion of the network is easily feasible. Further the ring has the fewest communication loss.

Our investigations did not cover the data distribution at the beginning of an iteration as well as the data recollection at the end of it. But this is not a problem for both, the bus and the ring topology. Using a ring, the data could be pipelined for sending it over the ring. For the recollection, the same principle may be used. On a bus, where each PE stores the data of all particles, broadcasting may be used for sending. For the recollection, the PEs send back their results one after the other.

We conclude, that a ring should only be used for the network topology if the memories on the PEs are too small for storing all particles and the data must be distributed. If we have enough memory on each PE for storing all particles, the bus topology should be used, because we have no communication overhead during the calculation and the data distribution is much simpler than on a ring.
4.3 Network Choice

The previous chapter has shown, that the bus topology offers the fastest as well as the simplest solution for our network. But we must use replicated data for the parallelisation of GROMOS, which requires more memory. Therefore we have to estimate the amount of data, which has to be stored on each PE to decide if replicated data and a bus topology can be used for our Alpha cluster.

In the case of the Thrombin molecular topology, we have 35961 atoms (see table 2.1). For each of the atom three coordinates with four bytes each must be stored. This results in 421.5 KB of data. If we expect, that we have one force per atom as results, the same amount of data must be added. Thus a total of 850 KB data must be stored, which is not a problem for todays memories.

Also for very large simulations with 100'000 atoms, the memory should not be a problem, because this amounts in 3 MB of data if a four dimensional simulation is made. If we consider, that the results are calculated in parallel and stored only on the respective PE, there is anyway fewer memory used than in the estimations above and we have enough memory on each PE to use replicated data.

Therefore we use a bus topology for the network of our processor cluster. In the replicated data method the communication occurs only at the begin and at the end of a time step, whereby the same amount of data has to be communicated, that we have estimated for the storage. Consequently the communication is very small in our parallel application, so that the performance of a standard network should be sufficient.

This allows to use an off-the-shelf network for our cluster, which offers concurrently the simplest solution. The network should have the following requirements:

- bus topology
- support broadcasting
- low latency
- high throughput
- good scalability
- easy handling
- low cost

In the following we evaluated different off-the-shelf standard networks.

4.3.1 Ethernet

The Ethernet is the most widely used local area network. It offers an easy installation and handling and is very flexible for low cost. The topology of the Ethernet is
in principle a simple bus, where all workstations are directly connected to. For the access to the network, the CSMA/CD\(^\dagger\)-mechanism is used, which means, that each workstation tests if the network is free before it begins to send data. When a collision occurs between the communication of two workstations, they stop sending data, wait for a specified time and resend the data. The advantage of the CSMA/CD-mechanism is, that all workstations have the same priority, but the possible collisions may result in a large latency [84].

The first version of the Ethernet uses a twisted pair coax-cable. Each workstation is connected directly over a T-cable-tap to the network cable, such that all workstations were chained to the network. For larger distances a repeater has to be included into the line (see figure 4.10a). This version has a throughput of 10 Mbit/s.

Todays most used Ethernet is the Fast Ethernet. Due to its higher throughput of 100 Mbit/s, shielded UTP-cables are used and all workstations are connected to a hub, which allows the higher data rates and longer distances between the workstations [85]. The hub has the function of a repeater. Basically the network lines are connected together, so that collisions are still possible. Better hubs manage the collisions in a way, that one workstation has not to stop the communication. For connecting more Ethernet buses, switches are used, which allow more connections at the same time and support full duplex transmittance between other switches, servers and workstations, with high communication traffic (see figure 4.10b) [86].

\[\text{Figure 4.10: Ethernet network: a) standard Ethernet, b) Fast Ethernet}\]

The newest Ethernet version is the Gigabit Ethernet, which offers a throughput of 1 Gbit/s. Its structure is mainly the same as that of the Fast Ethernet.

The Ethernet supports broadcasting, is good scalable and offers a quite good throughput for low cost. The only disadvantage is its high latency, because of its network access method.

\[\dagger\text{CSMA/CD: Carrier Sense Multiple Access with Collision Detection}\]
4.3.2 ATM Network

ATM stands for *Asynchronous Transfer Mode*. The asynchronous packet transfer allows to share the available bandwidth in a more flexible and efficient way to achieve a higher data bandwidth [87]. Figure 4.11 shows the difference between the synchronous and asynchronous transfer modes.

![Figure 4.11: Difference between STM and ATM](image)

The central unit in ATM networks is the ATM switch, where all workstations, servers etc. are directly connected to. Before a packet transfer is started, a virtual connection between the sender and receiver is made and the network resources are set up. This guarantees fewer packet loss and the handling of the addressing is simpler. The ATM switch is a massively parallel architecture, which allows to route and transfer several packets simultaneously through the network. Additional buffers or a recirculation mechanisms avoid collisions if two packets have the same receiver [88].

The ATM works with a fixed packet size of 53 bytes. This allows a better allocation of the network components, because their busy time for a packet transfer is known.

![Figure 4.12: ATM network](image)
4.3 Network Choice

The asynchronous packet transfer allows to allocate various bandwidths to each different workstation. The predefined bandwidth for a workstation can be guaranteed by the ATM network independently of the number of stations, which are connected to it.

Over an ATM network throughputs of up to 100 Gbit/s can be reached and the structure of the ATM switch allows low latencies [89]. ATM networks are scalable from LANs up to WANs, but do not support broadcasting. Another disadvantage is the high costs of ATM network components.

4.3.3 Myrinet

Myrinet is a new type of networks, based on the technology, which is used for packet communication and switching within massively parallel processors. The idea of Myrinet is a message-passing network, that can span campus dimension, but not in the sense of wide area networks [90].

The network is constructed by interconnecting the workstations over elementary routing switches. This allows a good scalability, because the capacity grows with the number of switches, so that many data packets may be in transit concurrently along different paths. Due to the regular structure of the network, the routing is simple and deadlocks, which might occur in case of cyclic dependencies, are avoided.

An example of a Myrinet network is shown in figure 4.13:

![Figure 4.13: Myrinet network](image)
The Myrinet network is composed of point-to-point full-duplex links, that connect the switches and the PE over a Myrinet interface. A Myrinet link has a throughput of 1.28 Gbit/s, which is composed of a pair of 640 Mbit/s full-duplex channels. The packet size in Myrinet is 8 KByte.

The multi-port switches may be connected to other switches and to the single-port interfaces in any topology. There are 4-, 8-, 16- and 32-port switches available.

The Myrinet interface includes a LANai processor and a SRAM as buffer with a maximum size of 2 MB. The LANai processor handles the data transfer between processor memory and buffer on the card, and manages the message handling and other network runtime communication processing tasks on the Myrinet [91].

The network has a very low error rate, which simplifies the packet protocols and error handling and leads to appropriate lower latencies.

Myrinet components are available as SAN (System Area Network) as well as LAN (Local Area Network). The SAN-components require shorter connections, but offer the higher performance.

However, the switching technology is not a bus topology and does not support a direct broadcasting mechanism. This is handled by the LANai, which broadcasts a packet in sending it several times over the network with a high performance.

The Myrinet is a good scalable network and the handling is easy. It offers a very high bandwidth and a low latency for a good price.

4.3.4 ServerNet

ServerNet is a wormhole-routed, packet switched, point-to-point network for a system area environment. It provides a high bandwidth and low overhead message passing and I/O connectivity, which allows to build clusters of PEs and I/O devices with an excellent performance [92].

The ServerNet uses multiple high-speed, low-cost routers to rapidly switch data directly between multiple data sources and destinations. The routers have a six-port crossbar switch on a single chip, which supports an intelligent switching, that the requirement for a processor in every data path is eliminated. The crossbar switch guarantees, that several packets can be routed through the network simultaneously, which leads to a low latency. Each router supports connections to any ServerNet interface and to other routers, which allows a high scalability.

All PEs and I/O devices are connected over interfaces to the network. The processor interface supports dual ports for fault tolerant connection to two ServerNet networks and the peripheral device interface is an open external interface, which allows connections to standard buses as VME, PCI and SCSI [93].
An example of a ServerNet network is in figure 4.14:

Figure 4.14: ServerNet system area network

The first generation of the ServerNet reaches a throughput of 50 MByte/s with a maximum packet size of 64 bytes and the successor, ServerNet II, reaches up to 125 MByte/s, supporting a larger packet size of 512 bytes. The components of both network versions are compatible and allows to extend a ServerNet I network with faster ServerNet II components [94].

ServerNet is a high performance network with a good scalability, but supports no broadcasting. The price of the network lies in the middle range.

4.3.5 Memory Channel

Memory Channel is system area network for connecting standard workstations to a cluster. The network should enhance the poor communication performance of a cluster if a standard network is used and increase the parallel performance.

The components of the Memory Channel are standard PCI adapters and an optional Memory Channel hub, which is necessary if more than two workstations are connected together. Memory Channel supports clusters with a maximum of 8 workstations [95].

There exist two versions of Memory Channel. The first generation has half duplex channels, which support a point-to-point bandwidth of 66 MByte/s using a maximum packet size of 32 bytes. The hub contains a shared bus, where the chan-
Channels to the workstations are directly connected. Memory Channel 2 supports full duplex channels and achieves a point-to-point bandwidth of 100 MByte/s. The maximum packet size is 256 bytes. The hub is an 8 x 8 crossbar switch, which supports higher throughputs and a simultaneous routing of several packets [96]. The network topology of Memory Channel 2 is shown in figure 4.15a.

The result of a high latency is mostly the impact of the operating system and not of the network itself. The Memory Channel eliminates this latency by supporting direct process-to-process communication, which bypasses the operating system. This is made by implementing an extension of the virtual memory space, which supports a direct, but protected, memory access to other workstations. If the operating system accesses an address in the Memory Channel address space, which is mapped to another workstation, the Memory Channel interface manages the fetching of the data from the other workstation and writes the data to the provided address in the memory of the workstation, which has requested the data. Figure 4.15b shows the principle of the data handling over Memory Channel.

**Figure 4.15: Memory Channel network: a) architecture, b) addressing principle**

Memory Channel seems to be the ideal solution for smaller workstation clusters. It is a low latency network, with a high throughput. Broadcasting is supported by mapping an address range of the virtual memory space to the memories of all workstations and the crossbar switch may connect all lines together to a bus.

The disadvantages of Memory Channel are the limit of the maximum of 8 workstations and its relatively high cost.
4.3 Network Choice

4.3.6 Backplane

Another solution for building our parallel coprocessor system is to use a backplane with a PCI or a system memory bus. In the case of a PCI bus, the PEs should be on PCI cards, which are plugged into the slots of the backplane. The other possibility is to use a server board, which houses up to 8 Alpha processors. Such boards use connection structures with a proprietary protocol.

Using a backplane would guarantee high throughputs with low latencies, but they have a significant disadvantage. It is very inflexible, the scalability is limited and the choice of the PE is restricted, which means, that an exchange of the processor units with newer and faster versions may require a new backplane board. Mostly the broadcasting mechanism is not supported. In addition, backplanes with processor modules are relatively expensive.

4.3.7 Conclusions

The following table gives an overview of the specified performances of the networks and the properties, which are important for our selection.

<table>
<thead>
<tr>
<th>Network</th>
<th>Latency</th>
<th>Throughput</th>
<th>Scalability</th>
<th>Broadcasting</th>
<th>Handling</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>120 µs</td>
<td>100 Mbit/s</td>
<td>unlimited</td>
<td>supported</td>
<td>easy</td>
<td>low</td>
</tr>
<tr>
<td>ATM</td>
<td>100 µs</td>
<td>622 Mbit/s</td>
<td>unlimited</td>
<td>not supported</td>
<td>easy</td>
<td>high</td>
</tr>
<tr>
<td>Myrinet</td>
<td>5 µs</td>
<td>550 Mbit/s</td>
<td>unlimited</td>
<td>supported by LANay</td>
<td>easy</td>
<td>low</td>
</tr>
<tr>
<td>ServerNet</td>
<td>3 µs</td>
<td>125 MByte/s</td>
<td>unlimited</td>
<td>not supported</td>
<td>middle</td>
<td>middle</td>
</tr>
<tr>
<td>Memory Channel</td>
<td>2.5 µs</td>
<td>100 MByte/s</td>
<td>8 PE(s)</td>
<td>supported by addressing</td>
<td>middle</td>
<td>high</td>
</tr>
<tr>
<td>Backplane</td>
<td>~1 µs</td>
<td>132 MByte/s</td>
<td>4 - 8 PE(s)</td>
<td>not supported</td>
<td>middle</td>
<td>high</td>
</tr>
</tbody>
</table>

*Table 4.9: Characteristics and specified performances of the different off-the-shelf networks*

The Myrinet and the ServerNet networks offer the best compromise between performance, scalability, handling and cost.
However, the backplane is the fastest solution, but is very inflexible and its costs are quite high. The Memory Channel, which offers a similar performance, has practically the same limits as the backplane. Its scalability is limited to eight PEs and the network components are expensive. A faster throughput than Myrinet and ServerNet is given by the ATM network, but the latency is quite high. Additionally it does not support broadcasting and the cost of the components are in a similar range as that of the backplane and the Memory Channel. The Ethernet is an easy handling low cost network, but its performance is too low and the latency is too high for our application.

We have now to choose between Myrinet and ServerNet for our Alpha cluster. The ServerNet has the better performance than Myrinet, but the handling is more difficult. A further handicap of ServerNet is, that it supports only point-to-point connections and no broadcasting. Additionally ServerNet is more expensive than Myrinet.

Myrinet offers an easy handling and reliable network for low cost. As the performance is not so important in our application, due to the use of replicated data, we use Myrinet for connecting the Alpha processors to a computer cluster.
Cluster Architecture

The most adequate solution for a GROMOS coprocessor is a cluster of Alpha workstations, which are connected over a Myrinet network. An additional installed Fast Ethernet network in the cluster allows the first time to analyse the impact of different networks to the MD simulation on a workstation cluster.

This chapter specifies the technical data of the workstation cluster and the communication networks. A description of the MPI software interface, which uses different ways for accessing the networks, follows. The chapter finishes with a performance analysis of the networks and an estimation of the communication time in the GROMOS simulations.
5.1 Hardware Overview

The hardware evaluations in chapter 3 have shown, that the use of multiple Alpha 21164 processor boards in parallel offers the best compromise between performance, implementation effort, flexibility and hardware cost for the GROMOS coprocessor. In chapter 4, we have pointed out, that the network requirements are not so high due to the few communication we have, and a Myrinet can be used for the interconnections of the processor boards. As the Myrinet network cards have a PCI interface, the simplest way is the use of standard Alpha motherboards, which are employed in commercial workstations.

Based on the results of our investigations, we can conclude:

**Our GROMOS coprocessor is realized on an Alpha workstation cluster, which uses a Myrinet as interconnection network.**

Our goal is to be at least 10 times faster than our reference machine, the SUN SPARC Station 10/85. As the Alpha 21164/500 workstation is 5 - 6 times faster than this machine (see section 3.6), a cluster of 2 workstations should be enough. But as we want to achieve a higher performance towards the current workstations and to analyse the performance of parallel GROMOS on a workstation cluster, we use a cluster of 4 Alpha 21164 workstations.

The Alpha workstations had been delivered with a Fast Ethernet network. This network is used for servicing the workstations and for analysing and comparing the impacts of two different networks to the MD simulation time.

The chemists often use Silicon Graphics workstations for their work and run also simulations on them. Therefore we use an SGI Octane as host computer, which is connected over both networks to the Alpha cluster.

We assembled the SGI as host and the Alpha cluster as coprocessor to a test system, where we can make performance measurements of the GROMOS simulations. A picture of the machines is shown in figure 5.1. As parallel GROMOS version, we use GROMOS96P, which was developed at the Institute of Computer Engineering and Networks Laboratory at the ETH Zurich, Switzerland during this project. It is described in section 2.2.4.3 and in [31].
The SGI and the Alpha workstations run under a UNIX operating system. An MPI interface is installed to support the communication functions of GROMOS96P. The interface is described in more detail in section 5.2. The source code of GROMOS96P can be compiled directly on the appropriate machines of the cluster. Simulations can be run on any desired number of processors and they can be run just on the Alpha workstations, used as \textit{homogeneous cluster}, or on the SGI and the Alphas, used as \textit{heterogeneous cluster}.

Figure 5.2 shows the block diagram of our parallel cluster, followed by the technical data of the host and coprocessor machines and the networks.
5.1 Hardware Overview

![Block diagram of the GROMOS workstation cluster and the networks](image)

**Figure 5.2: Block diagram of the GROMOS workstation cluster and the networks**

- **SGI Workstation**: CPU: 195 MHz MIPS R10000, Chip Rev.2.7
  1st level cache: 32 KB instruction and 32 KB data internal
  2nd level cache: 1 MB external SRAM
  main memory: 128 MB
  operating system: Irix 6.4
  compilers: MIPSpro C and Fortran77 7.2.1.2m

- **Alpha Workstation**: CPU: 500 MHz Alpha 21164
  1st level cache: 8 KB instruction and 8 KB data internal
  2nd level cache: 96 KB instruction and 96 KB data internal
  3rd level cache: 2 MB external SRAM
  main memory: 128 MB SDRAM
  operating system: DEC Unix 4.0D
  compilers: DEC C 5.6-071 and Fortran77 4.0

- **Myrinet**: host interface: 32-bit PCI Myrinet-SAN card,
  LANai RISC processor, 1 MB memory
  switch: dual 8-port Myrinet-SAN switch with two
  8-input/output full-crossbar switches

- **Fast-Ethernet**: host interface: built-in standard 100Base-T interface
  hub: 12-port Fast-Ethernet 100Base-T hub
5.2 Communication Network

As described above, an MPI interface is installed on the workstation cluster to support the communication functions of GROMOS96P. MPI is a message passing interface standard for parallel machines, especially those with distributed memory. It offers library routines for processing the communication between the processes on the nodes of a parallel system, which makes it possible to write software, which is independent of the hardware [98]. Thus the MPI interface allows to run GROMOS96P on other parallel machines, that support MPI. On the other hand our workstation cluster is able to execute any other parallel MPI program.

Multiple implementations of MPI have been developed. We use MPICH on our cluster as a MPI version, which is free available and combines portability with high performance [99].

The MPI on our workstation cluster uses different ways for accessing the two communication networks, which are illustrated in figure 5.3. For a better understanding of the protocols and the driver hierarchy, we used the OSI reference model, which is an abstraction standard of the layered communication concept in computer networks [83].

![Figure 5.3: MPI and driver structure on the communication networks](image-url)
The MPI uses TCP/IP for accessing the Ethernet network. For the Myrinet, two methods are possible. MPI may use TCP/IP, such as for accessing the Ethernet or it may use GM, a message passing interface, which has been especially developed for using the Myrinet network.

The GM interface allows the MPI a faster access to the hardware, because the communication does not use the slow TCP/IP protocol and the standard Myrinet driver. It covers three layers of the OSI Reference Model and manages the data transformation between the MPI interface and the network transmission, which has to be treated by the TCP/IP module and the Myrinet driver in the other case.

For the GM another MPI libraries are required than for TCP/IP. Thus we have to recompile the source code if we want to use GM. This may cause problems if we want to compare the three communication ways because using other libraries could affect the optimizer and result in a slower or a faster code. For TCP/IP we can use the same program code, which should allow a good comparison of the networks.

The present version of GM does only support homogeneous clusters. We cannot use it in simulations, which run on the SGI as well as on the Alphas yet.

We evaluated the performance of the networks with a benchmark tool, that uses MPI interface routines for the profiling. It sends data packets of different size over the three communication ways and measures the time, which is used by the communication. Figure 5.4 shows the throughputs in dependence of the data packet size in logarithmic scale:

![Graph showing network throughputs](image)

Figure 5.4: Measured throughput of the networks using the different communication ways
The communication over GM reaches overall the highest performance. The throughput grows with the size of the data packets up to 37 MB/s. For data packets larger than 64K, the limit of the Myrinet network bandwidth is reached and the throughput does not further increase. The overhead for the communication over TCP/IP is larger than that over GM, which results in a lower performance. TCP/IP works better over Ethernet than over Myrinet for smaller data packets, but reaches the limit of the Ethernet bandwidth for packets larger than 64K, where the higher throughput is achieved by TCP/IP over Myrinet.

As in GROMOS96P mainly large data packets with a size between 256 - 512 KB are transferred over the network, only the throughputs for this packet size are interesting. In addition, the latency is also an important factor. Table 5.1 shows the latencies and the throughputs for data packets with 512 KB of the communication ways, which are important for an analysis of the communication in GROMOS.

<table>
<thead>
<tr>
<th>Communication Way</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet over TCP/IP</td>
<td>170 µs</td>
<td>10.9 MB/s</td>
</tr>
<tr>
<td>Myrinet over TCP/IP</td>
<td>280 µs</td>
<td>21.7 MB/s</td>
</tr>
<tr>
<td>Myrinet with GM</td>
<td>36 µs</td>
<td>37.0 MB/s</td>
</tr>
</tbody>
</table>

*Table 5.1: Performance of the different communication ways for data packets with 512 KB*

The performance analysis shows, that the communication ways have different performances. This means, that in addition to the hardware also the software interface for accessing the hardware and the communication protocol plays an important role for the communication. Myrinet over GM has the lower latency and delivers the higher throughput than Myrinet over TCP/IP.

In comparison with the specified network performances in table 4.9, the data in table 5.1 show overall the worse performance. The reason is, that the specified values in table 4.9 are the *theoretical* peak performance, which may be achieved by the network hardware, whereas the data in table 5.1 have been determined by measurements, which include also the overhead of the network protocols and the data handling by the software interface. The values in table 5.1 are therefore more realistic, as they show the *effective* achievable network performance.
The measurements of the network performance allow us to make an estimation of the communication time for the Thrombin simulation in GROMOS96P. For this, we must know the number of data, which is communicated between the host and the parallel processors during the simulation. We obtained the amount of communication data by introducing structures into the programming code, which delivers the number of bytes being sent and received by the host. The number of data, which is communicated per parallel processor is listed in Table 5.2:

<table>
<thead>
<tr>
<th>Communication</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters sent to parallel processors</td>
<td>331.60 KB</td>
<td>266.75 KB</td>
</tr>
<tr>
<td>Data distributed to parallel processors per time step</td>
<td>421.47 KB</td>
<td>226.91 KB</td>
</tr>
<tr>
<td>Results sent back to the host per time step</td>
<td>421.67 KB</td>
<td>227.15 KB</td>
</tr>
<tr>
<td>Total communication data for 100 time steps</td>
<td>82.66 MB</td>
<td>44.60 MB</td>
</tr>
</tbody>
</table>

*Table 5.2: Amount of communication data during the Thrombin simulation*

The number of data, which is distributed to the parallel processors at each time step, consists only of the coordinates of the atoms. The concerning values in Table 5.2 can be obtained as \(3 \cdot 4 \cdot \text{number of atoms}\), where the 4 stands for the 32-bit data word, which is 4 bytes long and we have 3 coordinates for each atom. The number of data of the results, which are sent back to the host, is a little bit higher than the data, which is sent to each parallel processor. It consist of one force component per atom, which results in the same size of data package as the coordinates, and additionally the energies and the virial.

Using the data in tables 5.1 and 5.2, we can now estimate the communication time for any desired number of parallel processors. As no broadcasting is used in GROMOS96P, the amount of communication data raises proportionally with the number \(P\) of processors. If the host is included to the number \(P\) of processors, the values in Table 5.2 must be multiplied with the factor \((P-1)\) to obtain the amount of communication data.

We estimated the communication time for both Thrombin molecular topology simulations on our homogeneous cluster with 4 processors and listed the results in Table 5.3.
<table>
<thead>
<tr>
<th>Communication Way</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet over TCP/IP</td>
<td>22.75 s</td>
<td>12.28 s</td>
</tr>
<tr>
<td>Myrinet over TCP/IP</td>
<td>11.43 s</td>
<td>6.17 s</td>
</tr>
<tr>
<td>Myrinet over GM</td>
<td>6.70 s</td>
<td>3.62 s</td>
</tr>
</tbody>
</table>

*Table 5.3: Estimated communication time for the Thrombin simulation on 4 processors*

In comparison with the total simulation time on a workstation (see table 2.3 and figures 3.38, 3.39), the communication time in GROMOS96P is quite short. This verifies our presumptions, which we have made in section 4.3, that the communication is very small if replicated data is used for the parallelisation.

In the following, we ran different simulations for analysing the performance of GROMOS96P on our workstation cluster. Further we investigated the impact of the communication ways and the synchronisation methods on the total runtime of the simulations. The results of our measurements are discussed in the following chapter.
Results

As the adequate coprocessor hardware has been found for accelerating GROMOS, different measurements were made to investigate its performance. The chapter begins with a comparison of the sequential and the parallel version of GROMOS96 to show the performance gain of the new pairlist routine and to compare the efficiency of the codes. Then the impact of the network is investigated the first time by running simulations using different communication ways and changing the appropriate simulation parameters. An analysis of the synchronisation methods follows to find out if synchronous or asynchronous communication is better. In the next part, a performance analysis is made by measuring the runtimes of GROMOS96P on a different number of processors. An investigation of the load balancing shows the scalability of the parallel code, followed by an evaluation of the speedup, which achieves quite good results. The chapter closes with a prediction of the performance of GROMOS96P for a higher number of processors in the cluster, followed by a comparison with its performance on other clusters.
6 Results

We tested the performance of GROMOS96P on our cluster by using the rectangular and the octahedral Thrombin molecular simulation. As the UNIX on our cluster is a multitasking system, the task switches may cause unexpected events during the time running a simulation. This leads to timing measurement fluctuations in a benchmark, which are the higher, the longer the benchmark runs [100]. Therefore we have run each simulation minimally three times and compared the results. If the deviations were fewer than 1%, we took the median, which is the result of the simulation, whose value lies in the middle of that of the other simulations, else we ran more simulations till the required accuracy had been reached.

During the simulations we made sure, that nobody was logged in to the machines and that only the most necessary system processes were running on the system.

6.1 Sequential and Parallel GROMOS96 Version

Due to the parallelising of GROMOS96, some changes of the code had been made, which influence the performance of the simulation. The differences between GROMOS96 and GROMOS96P are the following:

- GROMOS96P uses a new and more efficient pairlist algorithm
- The original code is written in Fortran, whereas the parallel GROMOS version uses C for the newly developed and parallelised routines.
- Due to the parallelisation of the code, some routines were splitted into more subroutines.
- Communication structures have been introduced to GROMOS96P.

The use of a new pairlist algorithm leads to a speedup between the sequential and the parallel version of GROMOS96. To measure this speedup, we ran the Thrombin simulation with the sequential and the parallel version on one Alpha machine. The results are shown in figure 6.1.
Figure 6.1: Sequential vs. parallel version of GROMOS96 on one Alpha machine

- The new pairlist algorithm is up to 7 times faster for the rectangular and only about 2.5 times faster for the octahedral simulation. The reason is, that the cell based searching method is more efficient if the whole simulation box has the same structure as the cells. Due to the only use of rectangular cells, there is the problem of the cut off corners in the truncated octahedron, where cells are diagonally cut away. Additionally periodicity is diagonal in octahedral simulation boxes. Therefore more conditions have to be checked during neighbour searching in a truncated octahedron.

- Although the gain of the new pairlist algorithm is not so high for octahedral simulations, the percentage of the routine on the total simulation time decreases and amounts therefore to 3 - 6% yet.

- All other programming routines are a little slower in the parallel version of GROMOS96 due to the use of more subroutines and the communication structures, which require also processing time if no communication occurs. In addition the Fortran compiler generates more efficient code than the C compiler. Although the most routines of the parallel GROMOS96 version are slower, a shorter time can be achieved for the rectangular simulation, because of the more efficient pairlist algorithm. In the octahedral simulation, the time gain for the pairlist generation is not so high to compensate the slower speed of the other routines and the simulation takes a bit longer.
6.2 Influence of the Network

As discussed in section 5.2 the communication can be done over three ways. This allows to investigate the influence of the speed of the communication network on the performance of the parallel GROMOS96P simulation.

To make a statement about the importance of the network performance, we should know the share of the communication time to the whole simulation time. Our estimations in table 5.3 have shown, that the communication time is very low. This indicates a small influence of the network. The reason is, that GROMOS96P uses replicated data, where communication occurs only at the start of the simulation and at the beginning and the end of each time step. There is no communication during the calculation.

The communication time is practically only dependent of the number of atoms and the number of processors. As the calculation time and the communication time are both proportional to the number of atoms in the simulation, the share of the communication remains also the same if the size of the MD system is changed. This means, that the impact of the communication is independent of the MD system size. We can see this by comparing the rectangular and the octahedral Trombin simulation, where in the latter almost only half of the number of atoms are used. In the octahedral simulation the communication time (see table 5.3) and the simulation time (see figure 6.1) are rather half of that of the rectangular simulation.

For investigating the influence of the network, we should change the simulation parameters, which affect only the calculation time, such as cutoff radius, box dimension or interval of the pairlist update. This allows to shorten the calculation time, as the influence of the network can be better seen, due to the fewer timing fluctuations in the measurements. However, the pairlist update interval has not a large impact on the calculation time, because the pairlist generation in GROMOS96P requires only 3 - 5% of the total simulation time, and a modification of the box dimensions would also affect the communication time, because the conditions of density in chemical simulations require an adaptation of the number of atoms.

Therefore we use the cutoff radius for investigating the impact of the network. It determines the size of the spherical range of the neighbour searching space. An alteration of the cutoff sphere changes the numbers of atoms, which are considered for the interaction calculation in a cubical way. Thus the calculation time changes exponentionally. This can be used to show, that the impact of the network performance gets more important for shorter cutoffs and a higher number of processors.

We have run the rectangular and the octahedral Trombin simulations with varied cutoffs between 0.8 - 1.8nm on 4 processors using the three different communication ways.
Figure 6.2: Rectangular Thrombin simulation with different cutoffs on 4 Alpha processors using the three communication ways.

Figure 6.3: Octahedral Thrombin simulation with different cutoffs on 4 Alpha processors using the three communication ways.
The curves in figures 6.2 and 6.3 lie practically on top of each other, which means that the influence of the network performance is very low.

We can see, that the curves of the simulations using Ethernet and Myrinet over TCP/IP run parallel in both figures and that the time differences are overall the same for all cutoffs. These are just the differences of the communication times, because both simulations have exactly the same compiled program code as discussed in section 5.2, which leads to the same calculation time. This means that the communication is independent of the cutoff length.

An advance of the faster GM interface can only be seen in the simulations with a cutoff smaller than 1.2. For cutoffs higher than 1.4, the simulations using GM require more time than the others using Ethernet and Myrinet over TCP/IP. In the rectangular simulation (see figure 6.2), the time for the GM simulation grows faster than that of the other simulations by increasing the cutoff. In the octahedral simulation (see figure 6.3) more fluctuations in the GM simulations can be seen, but it takes overall longer than the other simulations. This is the problem of the different compiled program versions as discussed in section 5.2. Obviously the code for the GM is a little bit slower than that for TCP/IP.

As the communication way using Myrinet over TCP/IP is twice as fast as Ethernet over TCP/IP for GROMOS96P, we can presume, that the difference of the simulation times has in principle the same size as the communication time of the simulation using Myrinet over TCP/IP. This allows to make a comparison with the estimated communication times in table 5.3.

The average difference between the times of the simulations using Ethernet and Myrinet over TCP/IP amounts to 10.7 seconds for the rectangular and 5.2 seconds for the octahedral simulations. Our estimations of the communication times in table 5.3, which result in 11.43 seconds for the rectangular and 6.17 seconds for the octahedral simulation, are therefore quite exact.

For an analysis of the communication time in the GM simulations, we should only consider the simulation with a cutoff of 0.8, where the calculation time is relatively short, that the slower code has the lowest impact on the simulation time and the influence of the network performance can be best seen.

To obtain the communication time in the GM simulation, the difference between the times of the simulations using Myrinet over GM and TCP/IP must be subtracted from the communication time of the simulation using Myrinet over TCP/IP. This results in 5.7 seconds for the rectangular and 2.1 seconds for the octahedral simulation. A comparison with the estimated values in table 5.3 of 4.53 seconds for the rectangular and 2.51 seconds for the octahedral simulation shows, that our estimations are quite good.
We are interested how the speedup of GROMOS96P is affected by the different communication ways in dependence of various cutoffs. Figure 6.4 shows the speedups of the rectangular Thrombin simulation.

![Figure 6.4: Impact of the communication ways and networks on the speedup of the rectangular Thrombin simulations with cutoff radii 1.0, 1.4 and 1.8](image)

- There are only small differences between the speedups of the simulations using different communication ways but the same cutoff. This means, that the performance of the network has only a low impact on the speedup. The differences grow a little with the number of processors because the share of the communication time and with this the influence of the network increases.
- Varying the cutoff has the larger impact on the speedup, because the calculation time grows exponentially with the cutoff whereas the communication time remains the same. Thus the influence of the network gets smaller if the higher cutoff is used.
- We can further see, that the simulations using Myrinet over GM, whose code is a little slower, has only the better speedup for short cutoffs, where the faster communication takes more influence. Using longer cutoffs, the simulation time is more determined by the calculation time, which is higher in comparison with the other simulations. This leads to a lower speedup.

The results in figures 6.2 and 6.3 show, that the simulations using Myrinet over TCP/IP are the fastest, which result also in the highest speedup, as illustrated in figure 6.4. The code of the version, which uses Myrinet over TCP/IP has the faster code than that using Myrinet over GM and uses the faster communication way than that using Ethernet over TCP/IP.
To show the influence of the network performance in a quantitative way, we calculated the percentage of the communication time to the total simulation time of the Thrombin simulations using Myrinet over TCP/IP. As communication time we took the differences between the times of the simulations using Ethernet and Myrinet over TCP/IP, which are in principle the same as discussed above.

As in figure 6.4 we can see, that the influence of the communication is lower for larger cutoffs. Additionally the network performance has also the lower impact in octahedral simulations because it is more complicated and uses the longer calculation time.

The influence of the network performance grows four times with the number of processors because of two reasons: first the parallel version of GROMOS96 does not use broadcasting and therefore the communication time raises proportionally with the number of processors; second the calculation time divides on the number of processors due to its parallel execution.

We can conclude, that the influence of the network performance is very low for a small number of processors but may drastically increase for a higher number of processors. Whereas the percentage of the communication time for the rectangular Thrombin simulation with a cutoff of 1.4 is 1.75\% for 4 processors, it may result already in 7\% for 8 processors and grows up to 28\% for 16 processors.

The influence of the communication for a high number of processors could be decreased with the following solutions:
1. Distributing the data to all processors with broadcasting. Thus the communication time raises less with the number of processors.

2. Using overlapping communication and to shift the load balancing in such a way, that the host processor, which should distribute the data, has to make fewer calculations. In this way, the host sends the data to each parallel processor one after another, whereby each processor can already begin to calculate as soon as it has received the first data even while the host is still sending data to it. Thus the parallel processors finish their calculation in succession and can send back the data without hindering the others. As the communication occurs synchronous with calculation and no processor has to wait for the others, no calculation time is lost for the communication.

GROMOS96P has implemented neither the broadcasting nor the shifted load balancing. Only the overlapping communication is implemented in the barrier synchronous and the asynchronous versions, whose performance is investigated in the next section.

6.3 Synchronization Methods

In GROMOS96P the full synchronous (‘Fs’), barrier synchronous (‘Bs’) or asynchronous (‘As’) method can be used for the communication (see section 2.2.4.3 on page 57).

We compared these communication methods by running the rectangular and octahedral Thrombin simulations on our homogeneous and heterogeneous cluster configuration with different number of processors using Ethernet and Myrinet over TCP/IP and Myrinet over GM.

6.3.1 Homogeneous Cluster

Using only the Alpha workstations in our cluster without the SGI, the simulations show the same behaviour for Ethernet and Myrinet over TCP/IP, but performs in another way if Myrinet over GM is used.

6.3.1.1 Communication over Ethernet

The results of the simulations using Ethernet are illustrated in figures 6.6 and 6.7
6.3 Synchronization Methods

Figure 6.6: Runtimes of the rectangular Thrombin simulation over Ethernet using different synchronisation methods

Figure 6.7: Runtimes of the octahedral Thrombin simulation over Ethernet using different synchronisation methods
The full synchronous version delivers overall the fastest simulations.

The simulations, which has been run only on one processor, show that the code with the full synchronous method is the fastest. Obviously the barrier synchronous and asynchronous versions cannot be as good optimised by the compiler as the synchronous version. In addition the synchronization structures in the non-blocking versions require additional time for processing also if no communication occurs. Although the sending processors have not to wait until the data has been fetched by the receiver and can continue the processing, the time loss effected by the slower code and the synchronization structures cannot be compensated by using more processors.

The asynchronous version is faster than the barrier synchronous version for a higher number of processors. The reason for that is, that the barrier synchronous version has five synchronization points per time step, whereas the asynchronous has only one. Thus the barrier synchronous requires overall more time for the synchronisation.

In the octahedral simulation, the synchronisation time in the barrier synchronous and the asynchronous version grows with the number of processors. This is caused by the uneven load balancing, where the processors have to wait for each other at the general synchronisation point. The synchronous version has no general synchronisation point and the synchronisation time is not measured.

The non-blocking communication mechanism has no advantage on the homogeneous cluster using TCP/IP as communication interface. This may be caused by the following two reasons:

1. All processors have the same performance, which means when the host sends data to a parallel processor and can continue its process, it must wait for this processor to get back the data, because both have to do the same work and require therefore the same time.

2. There is practically no gain in the communication performance if it runs over TCP/IP, because most of the communication process is treated by the operating system and not by the hardware itself. Whenever a communication occurs the operating system starts a new process, which handles the TCP/IP protocol and prepares the data for sending it over the network. This process must be done by the Alpha processor. Using non-blocking communication, the processor has to treat the calculation and the communication processes in parallel and only the time for the effective communication on the network hardware can be saved, which is very short on our high performance networks. Thus the performance of non-blocking communication is practically the same as that of blocking communication.
6.3 Synchronization Methods

6.3.1.2 Communication over Myrinet using GM

As mentioned above, the simulations show another behaviour if the communication runs over Myrinet with GM. The results are shown in figures 6.8 and 6.9.

![Figure 6.8: Runtimes of the rectangular Thrombin simulation over GM using different synchronisation methods](image)

- In the simulations using GM on one processor, the synchronous version is slower than the barrier synchronous and asynchronous version. It seems, that in contrast to the simulations using TCP/IP, a better code optimisation can be done for the barrier synchronous and asynchronous version although the introduced synchronisation structures require additional time.
- The simulations with the barrier synchronous method are the fastest of all three versions for a higher number of processors. This is the consequence of the better code optimisation.
• Comparing the barrier synchronous version of the octahedral Thrombin simulation using Myrinet over GM (see figure 6.9) with the corresponding simulation, that uses Ethernet over TCP/IP (see figure 6.7) an interesting effect can be seen. Whereas the latter is faster than the former using 1 processor, it is slower on 4 processors. The reason is, that the non-blocking communication over GM is more efficient. In contrast to the communication using TCP/IP, where the Alpha processor has to handle the communication process, it is handled in GM by the LANai of the Myrinet board. Therefore the Alpha processor is practically not used for the communication in GM and can devote its full performance power to the calculation.

• The asynchronous version shows quite bad values. In the rectangular simulation (see figure 6.8) the time for the synchronisation on the host raises dramatically with the number of processors and amounts practically to the same time as used for the calculation. It seems, that the communication subsystem of GM does not start a non-blocking communication, until the host reaches its next synchronisation point.

The phenomenon in the last point above requires an investigation of the communication process. If we presume, that the communication is started only at the synchronisation points, the host should wait at the first synchronisation point until the
6.3 Synchronization Methods

parallel processors have received the data and processed the first routine. In the barrier synchronous version, this routine is very short, which means, that the communication would conform with blocking communication as used in the synchronous version. But as the communication in the barrier synchronous version is more efficient than that of the synchronous version, our thesis cannot be true. The data must be communicated before the host reaches the first synchronisation point. Thus the synchronisation problem in the asynchronous version may have another cause.

An analysis of the communication in the parallel program showed, that after the communication is started by the host, the data are sent immediately to the parallel processors. In the case of non-blocking communication, the processors should receive an acknowledge, when the communication is finished and the data are ready in the input buffer. But this acknowledge is delayed for an undetermined time if the MPI interface over GM is used and is generated no later than the next communication is started by the sender. This leads to these quite long synchronisation times in the asynchronous version, because the parallel processors have to wait for the acknowledge till the host reaches the general synchronisation point. In the octahedral simulation (see figure 6.9) obviously the acknowledge is generated before the synchronisation point is reached, because the synchronisation times are not as long as in the rectangular simulation (see figure 6.8).

The communication problem using GM has practically no impact on the barrier synchronous version, because the communication is made during the time the host executes the first routine and the parallel processors receive the acknowledge immediately after the host has reached the first synchronisation point. As they have already received the data, they can directly begin to execute the first routine, which is quite short and the host has not to wait long for the synchronisation.

Comparing the Thrombin simulations on our homogeneous cluster, we see, that the synchronous version using Myrinet over TCP/IP is the fastest. Although the communication of the barrier synchronous version using Myrinet over GM is more efficient, its slower code leads to longer calculation times, which cannot be compensated by the faster communication, because of the few communication data.

6.3.2 Heterogeneous Cluster

We investigated the performance of the three synchronisation methods on the Alpha cluster with the SGI as host. As described in section 5.2, Myrinet over GM cannot be used if different types of machines are used in the cluster. Thus we compared only the simulations using Ethernet and Myrinet over TCP/IP, which show the same behaviour for both communication ways.
6.3.2.1 Communication over Myrinet using TCP/IP

Figures 6.10 and 6.11 show the results of the simulations using Myrinet over TCP/IP using the SGI with a various number of Alpha workstations. Due to the different performance of the SGI and the Alpha, a simulation using the SGI with only one Alpha workstation is not executable and we left out this configuration for the simulations.

![Figure 6.10: Runtimes of the rectangular Thrombin simulation over Myrinet and TCP/IP using different synchronisation methods](image)

- In the rectangular simulations, the asynchronous version is the best for all numbers of processors. This is the inverse behaviour in comparison with the results on the homogeneous cluster, where the synchronous versions are overall the fastest.
6.3 Synchronization Methods

Figure 6.11: Runtimes of the octahedral Thrombin simulation over Myrinet and TCP/IP using different synchronisation methods

- In the octahedral simulation, which runs only on the SGI, the synchronous version is the fastest. But for a higher number of processors, the asynchronous version has the better performance gain and achieves the fastest speed of all simulations using the SGI and 4 Alphas. Obviously the more efficient communication method leads to a higher speedup in the asynchronous version.

- There is a higher synchronisation time in the asynchronous version of the octahedral simulation on the SGI with 2 and 4 Alphas. The reason is the uneven load balancing in the octahedral simulation for an odd number of processors.

On the heterogeneous cluster, the asynchronous version using Myrinet over TCP/IP is the fastest. This is not only due to the faster code, but also because of the non-blocking communication, where the slower SGI has not to wait until the communication is finished and can immediately continue its process. Although it finishes the calculation earlier than using blocking communication, it has not to wait for the faster Alphas, because they still finish the calculation before the SGI does. This leads to shorter calculation times in each time step, due to the communication time, which can be saved using overlapping communication in the asynchronous simulation.
6.3.3 Summary

The investigations with the different synchronisation methods show, that the performance of the code and the parallelisation of the program is much more important for GROMOS96P than the efficiency of the communication, due to the few communication data. In addition, the non-blocking respectively overlapping communication has only an advance on heterogeneous clusters and if the host is slower than the parallel processors.

In all simulations, the effects of the faster code can well be seen, but hardly that of the more efficient communication. The three synchronisation methods show the same behaviour on multiple processors as on a single processor except in the octahedral simulations on the heterogeneous cluster.

On both cluster configurations, the simulations using Myrinet over TCP/IP have the highest performance. Whereas the synchronous version is the fastest on the homogenous cluster, the asynchronous version is the fastest on the heterogeneous cluster. For our further investigations, we used only these two versions of GROMOS96P.

6.4 Performance Measurements

We measured the runtimes of the fastest parallel GROMOS96P versions using different numbers of processors on our homogeneous and heterogeneous cluster to investigate the performance of the most important programming routines and the whole simulation. For our measurements, we used the rectangular and the octahedral Thrombin molecular topology.

6.4.1 Homogeneous Cluster

The previous chapter showed, that the synchronous version of GROMOS96P, which uses Myrinet over TCP/IP as communication way, is the fastest on the homogenous cluster. The results of the measurements are illustrated in figures 6.12 and 6.13.
6.4 Performance Measurements

Figure 6.12: Runtimes of the rectangular Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP

Figure 6.13: Runtimes of the octahedral Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP
• The parallelised routines, which are the pairlist generation and the calculation of the non-bonded forces, require the fewer time the more processors are used.
• The integration routine has the same time for all numbers of processors. It is not parallelised and is calculated on the host only.
• The time of others increases, because it contains the preparing of the data for parallel processing, which increases with the number of processors. Additionally, the uneven load balancing causes longer synchronisation times on the host, which are included in the time of others. A further reason is the lack of the broadcasting mechanism for distributing the data in GROMOS96P. Because the first processor has to send the data separately to every processor, the communication grows with their number.
• As the time of the total simulation decreases in comparison with that of the serial parts, which remains the same respectively grows with the number of processors, the percentage of these routines increases. This leads to a slight decrease of the percentage of the parallel routines, whose times reduce proportionally with the total simulation time.
• The percentage of the non-bonded force calculation for solute-solute, solute-solvent in the rectangular simulation reduces drastically with the number of processors. This effect is not only caused by the increasing percentage of the serial routines but also due to the uneven load balancing of the programming routines.

6.4.2 Heterogeneous Cluster

The fastest GROMOS96P on the heterogeneous cluster is the asynchronous version, which uses Myrinet over TCP/IP as well. Figures 6.14 and 6.15 show the results.
6.4 Performance Measurements

Figure 6.14: Runtimes of the rectangular Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP

Figure 6.15: Runtimes of the octahedral Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP
• The programming routines of the simulations on the heterogeneous cluster behave overall in the same manner as on the homogeneous cluster. Whereas the time of the parallel routines decreases, that of the serial remains respectively increases with the number of processors. Obviously there is no impact on the behaviour of the parallelism if the lightly slower SGI as host is used.

• The runtimes of simulations are a little higher on the heterogeneous cluster in comparison with that on the homogeneous cluster for the same number of processors. This is due to the slower host in the heterogeneous cluster.

6.4.3 Summary

The parallel Thrombin simulation achieves quite good results on our workstation clusters, which are assembled on a maximum of 4 respectively 5 machines. As mentioned above, GROMOS96P achieves the better performance on the homogeneous cluster. The reason is the static load balancing in the spatial decomposition (see section 2.2.4.3), which leads to an uneven time spread if the workstations have not all the same speed. A more detailed analysis of the load balancing is given in the next section.

6.5 Load Balancing

The execution times of the parallel routines should be preferably the same on every parallel processor to avoid idle times and to utilize the available computing power as efficient as possible. Therefore the workload should be evenly distributed to the parallel processors to achieve a better load balancing, which causes shorter simulation times and a higher speedup.

We analysed the load balancing of GROMOS96P on our homogeneous and heterogeneous cluster by measuring the time of the parallel routines on each processor in the rectangular and octahedral Thrombin simulation.
6.5 Load Balancing

6.5.1 Homogeneous Cluster

The parallelised routines of GROMOS96P scale in a different way for the rectangular than for the octahedral simulation.

Figure 6.16 shows the time of the parallelised routines on each parallel processor for the rectangular simulation on different numbers of processors.

- The load balancing of the pairlist routine is almost ideal for all numbers of processors.
- The routines for the non-bonded interaction calculation have an uneven load balancing for more than two processors, which is caused by the static spatial decomposition method in GROMOS96P (see figure 2.13). As the solute molecules are located in the centre of the simulation box, the processors, which are responsible for the middle regions, have to calculate more solute-solute and solute-solvent interactions, whereas that for the border regions have to calculate more solvent-solvent interactions. Thus the uneven distribution of the solute and solvent interaction calculation levels mainly out over the whole non-bonded interaction calculation. Due to the lack of communication and synchronisation points between the solute-solute, solute-solvent and the solvent-solvent interaction routines, the processors have not to wait for each other and finish the calculation almost simultaneously at the end of the time step.
The load balancing of the parallel routines in the octahedral simulation is illustrated in figure 6.17.

Figure 6.17: Load balancing of the parallel routines in the octahedral Thrombin simulation on the homogeneous cluster

- The load balancing behaves in a similar way for 2 and 4 processors as in the rectangular simulation.
- In the octahedral simulation on 3 processors, the load balancing is bad. The reason is, that due to the octahedral structure of the simulation box, GROMOS96P uses wider border regions and smaller centre regions to distribute the work evenly across the processors. This works fine for 2 and 4 processors, but the border regions are too large for 3 processors. In that case processor 1 and 3 have to consider more molecules than processor 2, which results in the uneven load balancing.
6.5.2 Heterogeneous Cluster

On the heterogeneous cluster we made our analysis only on the SGI as host with 2 or more Alpha workstations, because a simulation on the SGI with only one Alpha is not executable.

The load balancing of the rectangular simulation is shown in figure 6.18.

![Load Balancing Chart](image)

Figure 6.18: Load balancing of the parallel routines in the rectangular Thrombin simulation on the heterogeneous cluster

- The characteristics of the load balancing is nearly the same as on the homogeneous cluster. The processors in the middle have to consider more solute molecules, whereas the one on the border must treat more solvent molecules.
- The slower host requires more time for the routines than the parallel processors for all numbers of processors. This can well be seen in the pairlist generation, which has a good load balancing on the parallel processors.

Figure 6.19 shows the load balancing of the octahedral simulation on the heterogeneous cluster.
Figure 6.19: Load balancing of the parallel routines in the octahedral Thrombin simulation on the heterogeneous cluster

- In comparison with the homogeneous cluster, the load balancing is more uneven.
- The better load balancing is reached for an even number of processors than for an uneven number of processors.
- The simulations on 3 and 4 processors behave in another way on the heterogeneous cluster. Obviously the processors, which are responsible for the middle regions have to do more calculations than the one for the border regions.
- Using 5 processors, the smaller centre regions cause the processor for this region to treat fewer molecules and to finish its work earlier than the others.

6.5.3 Summary

The high drop of the percentage of the non-bonded solute-solute, solute-solvent interaction calculation in section 6.4 can now be explained, that the results in the figures 6.12, 6.13, 6.14 and 6.15 have been measured on the host processor, which is responsible for the border region in the spatial decomposition. As mentioned above, this region contains fewer solute molecules, which are further reduced due to the narrower region, used for a higher number of processors. This causes a high decrease of the solute-solute, solute-solvent interaction calculation time.
The best load balancing is reached in the rectangular simulation on the homogeneous cluster. An even number of processors is required in the octahedral simulation to obtain a good load balancing. On the heterogeneous cluster the influence of the slower host processor can be seen, which requires overall more time for the calculation. Thus we can conclude, that the static spatial decomposition method in GROMOS96P is more suitable for the homogeneous cluster due to the regular density of the molecular structure in the simulation box and the regions of the same size.

### 6.6 Speedup

The speedup shows the quality of the parallelisation of a program. It is defined as the ratio of the execution time of the fastest serial version on 1 processor $t_1$ and the execution time of the parallel version on $P$ processors $t_p$

$$Speedup_p = \frac{t_1}{t_p}. \quad (6.1)$$

If the ratio is exactly equal to the number of processors, the speedup would be ideal. But this remains unattainable in real environments, because each parallel program contains parts, which can not be parallelised and are only executable on one processor. Additionally, parallel overheads, such as load imbalance and communication between the processors, reduce the speedup relative to the ideal.

It is an important task in parallel applications to keep the share of the serial part as small as possible, to reach a good load balancing and to make the communication efficient to obtain a high speedup.

We have evaluated the speedups for the parallel routines and the total simulation of GROMOS96P, which we have achieved on our clusters.
6.6.1 Homogeneous Cluster

Figures 6.20 and 6.21 show the speedups of the Thrombin simulation of the synchronous GROMOS96P version, which uses Myrinet over TCP/IP as communication way.

![Graph 6.20](image1)

*Figure 6.20: Speedup of the rectangular Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP*

![Graph 6.21](image2)

*Figure 6.21: Speedup of the octahedral Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP*
• The speedup is quite high for the rectangular and the octahedral Thrombin
  simulation.
• For 2 and 3 processors, the speedup for the rectangular simulation is a little
  better than that for the octahedral simulation, where the influence of the bad
  load balancing causes a light drop for 3 processors in the speedup curve.
  Using 4 processors, the speedup is practically the same for both simulations.
• There are overall no differences in the speedups for only 2 processors, because
  all routines have a good load balancing. Differences can only be seen for more
  than 2 processors.
• The pairlist routine reaches the best speedup of all routines, due to its load bal¬
  anger, which is very good.
• The speedup of the non-bonded solute-solute, solute-solvent interaction rou¬
  tine is quite bad for the rectangular simulation, above all for 3 processors. The
  reason is, that the load balancing of this routine is extremely uneven, which
  causes large differences in the execution times on the parallel processors (see
  figure 6.16). In the octahedral simulation the speedup is quite good only for 3
  processors due to the different widths of the regions, where the number of sol¬
  ute molecules is well distributed, but it drops again for 4 processors.
• The non-bonded solvent-solvent interaction routine has practically the same
  speedup as the total simulation, because it is the most time intensive and deter¬
  mines largely the total simulation time.

6.6.2 Heterogeneous Cluster

In contrast to the homogeneous cluster, the speedups of the rectangular and the
octahedral simulation behave in a different manner. They are shown in figures 6.22
and 6.23. We took the reference time each from the fastest serial version, which is
the asynchronous version in the rectangular (see figure 6.10), and the synchronous
version in the octahedral simulation (see figure 6.11).

As mentioned above, simulations on the heterogeneous cluster using the host
with only one parallel processor are not possible. Thus we estimated the speedups
for this configuration. The respective values in the graphs are illustrated in italic.
The speedup behaves in a similar way and reaches practically the same values as on the homogeneous cluster.

- The non-bonded solute-solute, solute-solvent interaction routine has the lowest speedup. It drops for an odd number of processors.

Figure 6.22: Speedup of the rectangular Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP

Figure 6.23: Speedup of the octahedral Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP
6.7 Prediction

- There is quite a high speedup reached in the octahedral simulation for up to 4 processors, which drops for 5 processors. Obviously the simulation scales quite good on the heterogeneous cluster and the asynchronous communication causes the fewer time loss.
- The programming routines have values of the speedup, which are higher than the number of processors. The reason is, that the time of the serial version on one processor has been measured on the slower host. As the parallel processors are faster, a higher time gain can be reached if the host has to do less work.

6.6.3 Summary

The speedups on the homogeneous and the heterogeneous cluster show the same results except that of the octahedral simulation on the heterogeneous cluster. In that simulation, the high speedup is confusing if we remark, that the simulation times on the homogeneous cluster have the better values. It seems, that the reference time of the serial version on the slower host enhances the speedup on the heterogeneous cluster to such high values. Another reason may be fluctuations and inaccuracies in the measurements.

We have achieved quite high speedups on our clusters for up to 4 processors. The speedups on the heterogeneous cluster are a little higher than that on the homogeneous cluster due to the reference time on the slower host. Thus we can conclude, that the homogeneous cluster is more suitable for GROMOS96P mainly if we consider the better simulation times for the same number of processors.

6.7 Prediction

The GROMOS96P simulation could be further accelerated by adding more workstations to our cluster. As the parallel overhead grows with the number of processors, the speedup is reduced relative to the ideal and reaches a maximum for a certain number of processors. At this point, no more performance gain can be achieved by adding more processors, in fact the performance shrinks in this case.

We are interested in the performance gain, which could be achieved for GROMOS96P on a larger cluster. In addition, we want to find the maximum achievable speedup, to determine the highest possible number of processors.
Our cluster has only 4 processors and we cannot test the performance of GROMOS96P for a higher number of processors. Therefore we make an estimation of the speedup of GROMOS96P to illustrate its performance on a larger cluster. As the speedups are practically the same for the homogeneous and the heterogeneous cluster, we can make a single estimation for both in common.

For estimating the speedup of a parallel program, its execution time for a various number of processors is required. The execution time of a parallel program is not only the time for processing. An additional time is required for the overhead due to the parallel processing, which arises of the communication between the processors and the synchronisation caused by an uneven load balancing. The time of the overhead must be added to the processing time to obtain the total execution time [101].

As not all parts of a program can be parallelised, each parallel program has a serial part in addition to the parallel parts, which can not be distributed to more processors and its execution time remains constant independently of the number of processors. Thus the processing time may be further separated into the serial time and the parallel time.

The overhead in our program consists mainly of the communication. We have practically no synchronisation overhead, which remains mainly constant in our simulations. Therefore we can treat the synchronisation time with the serial time in our estimations and the overhead time consists only of the communication time.

As our parallel program consists in principle of a serial part, a parallel part and a communication part, which perform each in a different manner in dependence of the number of processors, but are normally independent of each other, their execution times may be treated separately. Thus the execution time of the parallel program can be written as

$$t_p = t_{\text{processing}} + t_{\text{overhead}} = t_{\text{serial}} + t_{\text{parallel}} + t_{\text{com}}.$$  \hspace{1cm} (6.2)

As the speedup shows only the relative performance, the execution time of the parallel program on only one processor may be normalized to 1 and the shares of the program parts can be written as fractions to that execution time.

Under Amdahl's law, the time of the serial part is expressed as a fraction of $s$, where $0 \leq s \leq 1$. All remaining parts with a fraction of $(1-s)$ are supposed to be executed completely in parallel [103]. Because no communication occurs if only one processor is used, the communication time is zero and the fraction $(1-s)$ refers only to the time of the parallel part, which is divided through the number $P$ of the parallel processors.

In fact, Amdahl's Law handles basically only the processing time and not the communication time [104]. Thus the fractions $s$ and $(1-s)$ refer in principle to the
processing time on one processor. As the communication time is zero in this case, the processing time is equal to the total execution time.

Thus the processing time on \( P \) processors can be written as

\[
 t_{\text{processing}} = t_{\text{serial}} + t_{\text{parallel}} = \left( s + \frac{1-s}{P} \right) \cdot t_1. \tag{6.3}
\]

The communication time can be separately expressed with a fraction \( c \) of the total execution time. It grows linearly with the number of processors and is zero for one processor [105]. The communication time can be written as

\[
 t_{\text{com}} = (P-1) \cdot c \cdot t_1. \tag{6.4}
\]

The speedup is defined as the ratio between the time on one processor and the time on \( P \) processors (eq. 6.1). Using equations 6.2, 6.3 and 6.4, we can write

\[
 \text{Speedup}_P = \frac{t_1}{t_P} = \frac{1}{s + \frac{1-s}{P} + (P-1) \cdot c} . \tag{6.5}
\]

Our work is now to find the parameters \( s \) and \( c \). Then we can evaluate the speedup with equation 6.5 for any desired number of processors.

In our case the speedups up to 4 processors as well as the communication time are known. Transforming equation 6.4, we can obtain the communication factor \( c \) with

\[
 c = \frac{t_{\text{com}}}{(P-1) \cdot t_1}. \tag{6.6}
\]

The factor \( s \) can be evaluated by transforming equation 6.5 into

\[
 s = \frac{P \cdot \left( \text{Speedup}_P^{-1} - (P-1) \cdot c - \frac{1}{P} \right)}{P-1}. \tag{6.7}
\]

Because of the fluctuations in the measurements, we get slightly different results for \( s \) in equation 6.7 for the same simulation if we use the speedups of the different numbers of processors. The results for \( s \) lie in the range between 0.027 - 0.061, whereby the mean deviation amounts to 0.007 for both of the Thrombin simulations. To define the parameter \( s \) for our estimations, we took the average of these
results, because it yields in the smallest deviation of 1 - 2% between the measured and the estimated speedups for 2 - 4 processors.

In the octahedral simulation, we have a good load balancing for 2 and 4 processors, whereas that for 3 processors is quite bad, caused by the octahedral structure of the simulation box (see section 6.5.1). As this structure has less influence on the load balancing for a higher number of processors, these speedups have rather the tendency of that of 2 and 4 processors than that of 3 processors. Therefore we used only the speedups of 2 and 4 processors to define the parameter $s$ for the predictions of the octahedral simulation.

The parameter $c$ will be the most exact if we evaluate it from the measurements of the simulation on the highest available number of processors. In this case, the communication time is the longest, the calculation time the shortest and measurement fluctuations have the lowest impact. Thus we evaluated the parameter $c$ from the communication time of the simulation with 4 processors.

Table 6.1 shows the evaluated constants for $s$ and $c$ for the rectangular and the octahedral simulation.

<table>
<thead>
<tr>
<th>Thrombin Molecular Topology</th>
<th>$s$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular Simulation</td>
<td>0.0381</td>
<td>0.0018</td>
</tr>
<tr>
<td>Octahedral Simulation</td>
<td>0.0542</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

Table 6.1: Parameters for predicting the speedup of the Thrombin simulations

Due to the slowest parameter $s$ in the rectangular simulation, we see, that it has the smaller serial part and the best load balancing in comparison with the octahedral simulation.

The parameter $c$ shows the share of the communication and we see, that it is higher in the rectangular simulation. The reason is, that the calculation is more complicated in the octahedral simulation than in the rectangular, but we have fewer data to communicate, which makes the communication part smaller in the octahedral simulation.

With the evaluated data, we give now a prediction of the speedup for our cluster with more than 4 processors. The result is shown in figure 6.24:
As mentioned above, the deviation between the measured and the estimated speedups for 2-4 processors amounts to 1-2%, except that for 3 processors of the octahedral simulation, which amounts to 8%.

The rectangular simulation has the better performance than the octahedral simulation till up to 27 processors. For more processors, the octahedral simulation shows the better speedup. The reason is, that the share of the sequential part (parameter $s$) plays the more important role for a small number of processors, whereas the share of the communication (parameter $c$) takes the higher influence on the speedup for more processors, because the communication grows with the number of processors. Thus the lower factor $s$ and the higher factor $c$ of the rectangular simulation lead to a higher speedup for few processors, which gets lower than the octahedral simulation for more processors.

The speedup curve of the rectangular simulation rises steeply up to 12 processors, then it flattens down and reaches the maxima at 23 processors, where the highest speedup is reached with 8.33. The average workload of each processor amounts there only to 36.2%. The octahedral simulation has a lower increase in the speedup curve, which reaches the maxima at 28 processors. The speedup amounts there to
8.23 and the average workload per processor is 29.4%. Both curves drop continuously for more than 23 respectively 28 processors.

As a conclusion of this prediction, we recommend a number between 8 - 16 processors to use for both simulations, whereby the workload of each processor is about 70% for a number of 8 and is still 48% for 16 processors.

6.8 Performance on Other Clusters

We have run GROMOS96P on other parallel processor clusters for a comparison of their speedups with the results on our workstation cluster. The following hardware environments have been tested:

Alpha: 1-4 single processor 21164 Alpha (500 MHz, 128 MB RAM, 2 MB Cache) workstation cluster, Myrinet

SUN: 1-4 single processor SUN Ultra 30 (300 MHz, 256 MB RAM) workstation cluster, standard Ethernet (LAN)

SP2: IBM RS/6000 SP2 (1-64 160 MHz thin nodes with POWER2 Super Chip Uniprocessors, 256 MB / 4.5 GB per node)

We made our tests with the rectangular and the octahedral Thrombin molecular topology. Table 6.2 shows the speedups, which we have reached on the different hardware for a different number of processors. The values in italics are estimated.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Cluster</th>
<th>1P</th>
<th>2P</th>
<th>3P</th>
<th>4P</th>
<th>16P</th>
<th>20P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>Alpha</td>
<td>1.00</td>
<td>1.94</td>
<td>2.75</td>
<td>3.44</td>
<td>7.96</td>
<td>8.27</td>
</tr>
<tr>
<td></td>
<td>SUN</td>
<td>1.00</td>
<td>1.38</td>
<td>2.15</td>
<td>2.64</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>SP2</td>
<td>1.00</td>
<td>1.93</td>
<td>2.83</td>
<td>3.68</td>
<td>11.36</td>
<td>13.6</td>
</tr>
<tr>
<td>Octahedral</td>
<td>Alpha</td>
<td>1.00</td>
<td>1.88</td>
<td>2.49</td>
<td>3.46</td>
<td>7.58</td>
<td>7.99</td>
</tr>
<tr>
<td></td>
<td>SUN</td>
<td>1.00</td>
<td>1.79</td>
<td>2.19</td>
<td>2.53</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>SP2</td>
<td>1.00</td>
<td>1.93</td>
<td>2.32</td>
<td>3.44</td>
<td>7.79</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 6.2: Speedup of GROMOS96P on different computer clusters*

- The rectangular simulation achieves also on the other clusters higher speedups than the octahedral. This corroborates, that the spatial decomposition method in GROMOS96P works better for the rectangular simulation box.
• The speedups of the SUNs are for both simulations very bad, for different reasons. The first is, that the percentages of the routines are not the same as on the other machines. The pairlist routine, which has a good scalability, has the lower percentage, whereas that of the force calculation, where the load balancing is not so good and that of the sequential parts is higher. Thus the longer sequential part and the higher impact of the uneven load balancing results to a worse performance. The second reason is, that the standard Ethernet is relatively slow. If it is used in a LAN, its latency can be relatively high, because of the share and the caused collisions with other users. Sometimes, the configuration of the workstations, which are used in a business environment, differs a little, which causes uneven calculation speeds on the workstations. This leads to further scalability problems and is the third reason for the bad performance on the SUNs.

• The SP2 reaches the highest speedups of the investigated systems, due to its fast communication subsystem, which achieves a throughput of 86.3 MB/s \[102\]. This is four times faster than the Myrinet over TCP/IP, which we have used in the simulations on the Alpha cluster. Using the communication time of the rectangular simulation on the Alpha cluster with 4 processors, which is 10.7 seconds, that on the SP2 can be estimated to 2.69 seconds.

• The Alpha cluster achieves the shorter simulation times for less than 16 processors, although the SP2 has the higher speedups. The reason is, that the Alpha processor has the better performance than one single node of the SP2. Whereas the rectangular Thrombin simulation requires 1951 seconds on one single Alpha workstation, it has 2971 seconds on one SP2 node [31]. Using 16 processors, we have 245.2 seconds on the Alpha cluster and 261.5 seconds on the SP2.

• The speedups of the SP2 in table 6.2 allow to make a validation of our prediction model, which we have made in section 6.7. Using the times of the rectangular simulation, we obtain the parameters \(s=0.027\) and \(c=0.0003\), which predict a speedup of 10.83 for 16 and 12.29 for 20 processors. The validation shows that the predicted speedups in our model are 5 - 10% too low for 16 - 20 processors. But if we consider, that the barrier synchronous version has been used on the SP2, which means, that some of the communication may occur overlapped with processing, we can explain the better performed speedups than the predicted ones.

The comparisons show, that a workstation cluster is a good alternative to massively parallel computers for a small number of processors because the same or even the higher performance can be reached for a better price. A major advantage is the possibility for exchanging the workstations with faster ones to enhance the
Chapter 6: Results

speed. If thereby workstations can be found, whose computing power is higher than that of one node of the parallel processor system, the better performance can be achieved on the workstation cluster, as we have seen in our case. However, for a higher number of processors, a parallel processor system is better qualified, because it has the better speedups than a workstation cluster.

6.9 Conclusions

We achieve a quite high performance for GROMOS96P on our workstation cluster, but the maximum number of processors should be limited to be effective. Regarding the measurements in this chapter, we can make the following conclusions:

- There should be a maximum of 16 workstations used in the cluster to achieve a good workload and to get a good price/performance ratio.
- As we use replicated data, the communication ratio is very low in the simulations and has only an impact for a higher number of processors.
- Consequently the synchronisation method (synchronous, barrier synchronous and asynchronous) plays an inferior role.
- The load balancing is quite good for static spatial decomposition, but could be improved by using dynamic decomposition.
- The comparison with the other clusters shows, that at the present time an office environment is still not suitable for parallel MD simulations. The workstation cluster should be stand-alone.
- Although the SP2 has the better speedups, the Alpha workstation cluster achieves the better simulation times than the SP2 for fewer than 16 processors. This is caused by the higher performance of the Alpha processor.

A workstation cluster is a quite good solution for accelerating an MD algorithm by parallelising it for a number of 8 - 16 processors. It offers a small parallel system for a good price.
Conclusions

In this chapter, the essential points concerning this thesis are discussed and some ideas for future aspects are given. The first section concludes the fundamental results of our investigations and makes a statement of the performance of our coprocessor hardware. In the second part, some proposals for improving the efficiency of MD simulations on workstation clusters are given, followed by an outlook to further trends in the field of parallel computing.
7 Conclusions

7.1 Fundamental Results

Our goal was to find a hardware for accelerating an MD simulation to a minimal factor of 10 times. An analysis of the GROMOS algorithm pointed out, that the most time intensive parts are the simplest. Therefore the most adequate hardware structure is a coprocessor, attached to a host computer, where only these intensive parts are sourced out, and the other complex parts are computed on the host.

An evaluation of different hardware solutions showed, that standard microprocessor units offer the best compromise in speed, flexibility and implementation effort. In addition they are future-proof and have a good price/performance ratio. As the market offers already off-the-shelf components comprehending MPUs, a development of the hardware around the processor is not necessary and standard workstations can be used instead.

The desired acceleration of the MD algorithm can only be achieved by clustering multiple workstations to a parallel machine. Thus the most time intensive routines of the GROMOS96™ software had been parallelised for distributing the calculation to the workstations in the cluster. The use of standard C and Fortran code as well as the MPI software interface routines for the communication make the parallel GROMOS96 code independent of the hardware and permit a portation of the program to newer available computer clusters with faster workstations.

The appliance of replicated data allows to use a simple bus for the interconnection network between the workstations in the cluster. As there are only few data to communicate, a Myrinet offers enough performance for the network.

We assembled 4 Alpha 21164/500 workstations to a cluster with a Myrinet as interconnection network. The host is an SGI Octane with a 195 MHz MIPS R10000 CPU.

To give a statement about the acceleration, which we have achieved by using a workstation cluster as coprocessor, the runtimes of the Thrombin simulation on our reference machine SUN SPARC Station 10 and that on our cluster configuration are shown in table 7.1. However, the SPARC 10 was an actual machine as our project had been started three years ago. Today it is outdated and we must take an actual workstation as reference machine to give a realistic view of the performance gain using the GROMOS coprocessor. Thus we listed also the runtimes of the Thrombin simulation on the SGI Octane in table 7.1.
Hardware | Rectangular Simulation [s] | Octahedral Simulation [s]
--- | --- | ---
**Start of the Project:**
Reference machine SUN SPARC Station 10 | 12279.0 | 5379.0

**Today:**
Reference machine SGI Octane | 2192.6 | 1179.2

**Host and Coprocessor:**
SGI and 4 Alpha workstations with Myrinet | 529.2 | 345.1

Table 7.1: Comparison of the runtimes of the Thrombin simulations on the reference machines and on the host with GROMOS coprocessor

The runtimes of the simulations had been reduced only by the use of a faster workstation, which has become available due to the technological advance in the past three years. Using the workstation cluster, a further reduction of the runtimes is possible.

Table 7.2 shows the speed factors, which we have achieved in accelerating the GROMOS simulation with our coprocessor hardware.

<table>
<thead>
<tr>
<th>Speed Factor</th>
<th>Rectangular Simulation</th>
<th>Octahedral Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since starting the project</td>
<td>23.2</td>
<td>15.6</td>
</tr>
<tr>
<td>Achieved with todays reference machine</td>
<td>4.1</td>
<td>3.4</td>
</tr>
<tr>
<td>Technological advance</td>
<td>5.6</td>
<td>4.6</td>
</tr>
</tbody>
</table>

Table 7.2: Achieved speed factors, based on past and todays reference machines, using the GROMOS coprocessor

The achieved speed factors lie above the desired requirement of 10 since the start of the project. This means in principle, that we have accomplished the demands of the project. But if we consider, that about 1/4 of the speed factors is made by the technological advance in the past three years, we must moderate our result. Regarding, that the MD simulation is made on a todays actual and fast workstation, the simulation can only be accelerated by a factor of 4 if our Alpha workstation cluster
is used as coprocessor. To offer a solution, which enhances the speed of GROMOS by a factor of 10 on the basis of today's actual workstation, the number of processors in the cluster should be increased. But our estimations in section 6.7 showed, that a speed factor of 10 can not be achieved on the Alpha cluster with the presently used GROMOS96P version, because only a maximum factor of 8.3 by using 23 processors can be obtained. As the workload is very bad with 36.2% in this case, it is strongly recommended to use a maximum of 16 processors, where a speed factor of 8 is reached.

The desired speed factor of 10 can only be achieved on the SP2 with the actual GROMOS96P version. Using 20 nodes on the SP2, the rectangular Thrombin simulation takes 218.5 s, which is 10 times faster than on the SGI Octane. However, a speed factor of 10 should be in fact possible with the Alpha cluster as well. If broadcasting would be used in GROMOS96P, the communication time would not increase with the number of processors. For this case our prediction model prognosticates a speedup of 10 on the workstation cluster with 16 Alpha processors.

We have shown in this thesis, that a workstation cluster offers a smart parallel system for accelerating a MD simulation. It is a flexible solution for a good price and the availability of newer and faster workstations makes this solution future-proof.

7.2 Future Perspectives

We have shown in this thesis, that a workstation cluster offers the better performance than a parallel processor system, but only for a small number of processors. Therefore it is desirable to increase the speedup of GROMOS96P on the cluster to achieve the same performance as on parallel processor systems also for a higher number of processors. This would enable higher speed factors by using larger clusters.

The major factors, which decrease the speedup on parallel systems, are the communication and scalability of a parallel program. To enhance the speedup of GROMOS96P on our cluster, it should therefore be improved in the following way:

- For distributing the data, broadcasting may be used. This would lower the impact of the communication for a higher number of processors.
- To decrease the communication time further, the program should be changed, so that each parallel processor sends only the results of its region back to the host.
• The load balancing could be improved by implementing dynamic spatial decomposition, that the simulation box is divided in each time step newly into regions, according to the calculation time of each parallel processor in the last time step. This would also allow a cluster with various processors, which have a different performance.

The improvements in the communication may cause, that the amount of data, which is sent through the network, remains constant. The communication time would not increase with the number of processors and a higher speedup could be reached mainly on larger workstation clusters.

The use of dynamic load balancing may allow to run a parallel MD program in an office environment, during the time people are working on their machines. As the load would be balanced by measuring the calculation time on the workstations, the parallel program could dynamically adapt to the changing environment, which is caused by the work of the users.

Today's computer components offer enough performance for assembling a workstation cluster with a similar computing power as a parallel system, but the cluster should be stand-alone. However, the progress in the computer and network technology will allow to have a parallel computer in a business environment. In principle, this is already presently possible for parallel applications with few communication, but it must provided for a Gigabit Ethernet or an ATM network. Hereafter, it might be possible to use each office environment with a standard LAN as a high performance workstation cluster.
Glossary

ASIC: Application Specified Integrated Circuit. A custom microchip designed for a specific application.

CG: Charge Group. Set of atoms joined together to have a zero net charge for reducing the range of electrostatic interactions.

CPU: Central Processing Unit. The main processor of a computer, which performs all arithmetic and logic operations and all basic instructions.

DMA: Direct Memory Access. A method, which allows coprocessors or other system devices and peripherals to read or write the memory directly, without having to interrupt the CPU.

DSP: Digital Signal Processor. A special-purpose microprocessor designed for high-speed signal processing, used in audio and image manipulation or in communication applications.

FIFO: First In First Out. A memory for buffering data in a way, that the first value stored in the buffer is the first value subsequently read.

FPGA: Field Programmable Gate Array. A high density programmable logic device comprised of an array of configurable logic blocks, that can be reconfigured to perform various applications.

GROMOS: GROningen MOlecular Dynamics Simulation Program Package. Simulation software for the study of biomolecular systems.
HDL: **Hardware Description Language.** A particular programming language to describe the circuits on a semiconductor in a textual code.

LAN: **Local Area Network.** A data network intended to serve a small area of only few square kilometers or less.

LSB: **Least Significant Bit.** The bit with the lowest place value in a binary number.

LSI: **Large Scale Integrated.** Semiconductor fabrication technology, that can create a density of between 100 and 1000 transistors on a microchip.

MCM: **Multi Chip Module.** A chip package containing multiple bare microchips directly mounted and interconnected on a substrate, base material or laminate.

MD: **Molecular Dynamics.** Calculations for simulating the motion of each atom in a molecular system at a fixed energy, fixed temperature, or with controlled temperature changes.

MPI: **Message Passing Interface.** A communication library establishing a practical, efficient and flexible standard for writing portable message passing programs.

MPU: **Microprocessor Unit.** A CPU on a single chip for performing computations and operations of general-purpose applications.

MSB: **Most Significant Bit.** The bit with the highest place value in a binary number.

MUSIC: **Multi Signalprocessor System with Intelligent Communication.** Parallel supercomputer with DSPs, developed at the Electronics Laboratory at the Swiss Federal Institute of Technology.

OSI: **Open Systems Interconnection.** Describes a set of abstraction standards and recommendations for the layered communication concept in computer networks to allow intercommunication between different computer systems.
**PE:** Processing Element. Part of a computer or parallel processing system containing CPU, memory and a communication interface.

**SAN:** System Area Network. A data network, that establishes a high-speed connection between the PEs in computer clusters and parallel processing systems.

**ss:** Solute-Solute. Abbreviation, used in this document for interactions between that type of molecules.

**sv:** Solute-Solvent. Abbreviation, used in this document for interactions between that types of molecules.

**VHDL:** VHSIC Hardware Description Language. An IEEE-standard hardware description language.

**VLSI:** Very Large Scale Integrated. Semiconductor fabrication technology, that can create a density of between 1000 and 1’000’000 transistors on a microchip.

**vv:** Solvent-Solvent. Abbreviation, used in this document for interactions between that type of molecules.

**WAN:** Wide Area Network. A network that spans a wide geographic area.
Seite Leer / Blank leaf
# List of Figures

1.1 Normalized Argon pair potential: measured pair potential (solid line) and effective pair potential (dashed line) of two Argon atoms. ..... 8
1.2 Simplified illustration of the interactions in a biomolecular simulation. ..... 9
1.3 Covalent bond forces ..... 10
1.4 Covalent bond-angle forces ..... 11
1.5 Dihedral-angle torsion forces ..... 12
1.6 Improper dihedral-angle forces ..... 14
1.7 Non-bonded interactions of two diatomic molecules ..... 15
1.8 Twin range method ..... 17
1.9 Periodic boundary conditions ..... 21
1.10 Common used periodic boundary boxes ..... 22
1.11 Extended wall regions ..... 24

2.1 Treatment of molecular systems by analytical theoretical methods and computer simulations ..... 32
2.2 Choice of the molecular model, force field and sample size ..... 33
2.3 Two dimensional view of the simulation box using the cell index method ..... 35
2.4 Cell numbering scheme and neighbour searching in cell index method ..... 36
2.5 Linked list method ..... 37
2.6 Two dimensional view of the grid cell method ..... 38
2.7 Interaction calculation matrix for illustrating the amount of calculations of the different interaction types ..... 44
2.8 GROMOS algorithm ..... 45
2.9 Distance algorithm in GROMOS ..... 47
2.10 Pairlist algorithm in GROMOS ..... 48
2.11 Nested loops in the pairlist algorithm of GROMOS ..... 49
2.12 Non-bonded solvent-solvent interaction algorithm in GROMOS ..... 51
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.13</td>
<td>Spatial decomposition in GROMOS96P</td>
<td>54</td>
</tr>
<tr>
<td>2.14</td>
<td>Control and data flow graph of GROMOS96P</td>
<td>56</td>
</tr>
<tr>
<td>2.15</td>
<td>Percentage of the Thrombin simulation times by GROMOS96 routines</td>
<td>61</td>
</tr>
<tr>
<td>2.16</td>
<td>Principle of data handling and outsourcing of the GROMOS programming routines to a dedicated coprocessor</td>
<td>71</td>
</tr>
<tr>
<td>2.17</td>
<td>Data flow and dependency of the GROMOS programming routines</td>
<td>73</td>
</tr>
<tr>
<td>3.1</td>
<td>Speed and flexibility of different hardware solutions</td>
<td>77</td>
</tr>
<tr>
<td>3.2</td>
<td>Basic structure of the GRAPE systems</td>
<td>79</td>
</tr>
<tr>
<td>3.3</td>
<td>Block diagram of the GRAPE force calculation pipeline</td>
<td>80</td>
</tr>
<tr>
<td>3.4</td>
<td>Family tree of the GRAPE systems</td>
<td>81</td>
</tr>
<tr>
<td>3.5</td>
<td>GRAPE-1 system</td>
<td>82</td>
</tr>
<tr>
<td>3.6</td>
<td>GRAPE-3 processor board</td>
<td>83</td>
</tr>
<tr>
<td>3.7</td>
<td>GRAPE chip</td>
<td>84</td>
</tr>
<tr>
<td>3.8</td>
<td>GRAPE-5 processor board</td>
<td>85</td>
</tr>
<tr>
<td>3.9</td>
<td>Pipeline unit of the G5 chip</td>
<td>86</td>
</tr>
<tr>
<td>3.10</td>
<td>Particle index unit</td>
<td>88</td>
</tr>
<tr>
<td>3.11</td>
<td>Neighbor list unit</td>
<td>89</td>
</tr>
<tr>
<td>3.12</td>
<td>GRAPE-2 system</td>
<td>90</td>
</tr>
<tr>
<td>3.13</td>
<td>GRAPE-4 system</td>
<td>92</td>
</tr>
<tr>
<td>3.14</td>
<td>GRAPE-4 processor board</td>
<td>94</td>
</tr>
<tr>
<td>3.15</td>
<td>PROMETHEUS chip</td>
<td>95</td>
</tr>
<tr>
<td>3.16</td>
<td>HARP chip</td>
<td>95</td>
</tr>
<tr>
<td>3.17</td>
<td>Control board of GRAPE-4</td>
<td>97</td>
</tr>
<tr>
<td>3.18</td>
<td>Host interface board of GRAPE-4</td>
<td>98</td>
</tr>
<tr>
<td>3.19</td>
<td>Extended structure of the GRAPE system</td>
<td>99</td>
</tr>
<tr>
<td>3.20</td>
<td>GRAPE-6 system</td>
<td>100</td>
</tr>
<tr>
<td>3.21</td>
<td>MD-GRAPE system</td>
<td>101</td>
</tr>
<tr>
<td>3.22</td>
<td>MD chip</td>
<td>103</td>
</tr>
<tr>
<td>3.23</td>
<td>PROGRAPE-1 system</td>
<td>105</td>
</tr>
<tr>
<td>3.24</td>
<td>GROMACS system</td>
<td>106</td>
</tr>
<tr>
<td>3.25</td>
<td>MD-Engine system and cards</td>
<td>108</td>
</tr>
<tr>
<td>3.26</td>
<td>MD-Engine multi-processor system</td>
<td>108</td>
</tr>
<tr>
<td>3.27</td>
<td>MODEL chip</td>
<td>109</td>
</tr>
<tr>
<td>3.28</td>
<td>MUSIC system</td>
<td>113</td>
</tr>
<tr>
<td>3.29</td>
<td>Calculation of distance and distance vector on an ASIC or FPGA</td>
<td>116</td>
</tr>
<tr>
<td>3.30</td>
<td>Non-bonded solvent-solvent force calculation on an ASIC or FPGA</td>
<td>117</td>
</tr>
<tr>
<td>3.31</td>
<td>Calculation of virial on an ASIC or FPGA</td>
<td>118</td>
</tr>
<tr>
<td>3.32</td>
<td>ADSP-2106x SHARC DSP</td>
<td>122</td>
</tr>
<tr>
<td>3.33</td>
<td>Runtimes of the distance routine on the SHARC ADSP-2106x</td>
<td>129</td>
</tr>
</tbody>
</table>
3.34 Speedups of the distance routine on the SHARC ADSP-2106x .... 130
3.35 Runtimes of the force routine on the SHARC ADSP-2106x .... 133
3.36 Speedups of the fore routine on the SHARC ADSP-2106x .... 134
3.37 Runtimes of the Thrombin molecular dynamics simulations with 100 time steps on the SHARC ADSP-2106x .... 137
3.38 Runtimes of the rectangular Thrombin simulation on different workstations .... 141
3.39 Runtimes of the octahedral Thrombin simulation on different workstations .... 142

4.1 Tie of the particles in the N-body problem .... 149
4.2 Bus topology .... 150
4.3 2D-net with 6 communication links .... 151
4.4 2D-net with 4 communication links .... 153
4.5 Hyper cube (3D-net) .... 155
4.6 Recursive topology .... 157
4.7 Ring topology .... 160
4.8 Speedup on parallel processor systems with the different network topologies for a N-body simulation with 1000 particles .... 164
4.9 Speedup on parallel processor systems with the different network topologies for a N-body simulation with 10'000 particles .... 164
4.10 Ethernet network .... 167
4.11 Difference between STM and ATM .... 168
4.12 ATM network .... 168
4.13 Myrinet network .... 169
4.14 ServerNet system area network .... 171
4.15 Memory Channel network .... 172

5.1 Picture of the host and the GROMOS coprocessor, which is realized as an Alpha workstation cluster with 4 processors .... 178
5.2 Block diagram of the GROMOS workstation cluster and the networks .... 179
5.3 MPI and driver structure on the communication networks .... 180
5.4 Measured throughput of the networks using the different communication ways .... 181

6.1 Sequential vs. parallel version of GROMOS96 on one Alpha machine .... 188
6.2 Rectangular Thrombin simulation with different cutoffs on 4 Alpha processors using the three communication ways .... 190
6.3 Octahedral Thrombin simulation with different cutoffs on 4 Alpha processors using the three communication ways .... 190
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>Impact of the communication ways and networks on the speedup of the rectangular Thrombin simulations with cutoff radii 1.0, 1.4 and 1.8.</td>
</tr>
<tr>
<td>6.5</td>
<td>Percentage of the communication time on the Thrombin simulations using Myrinet over TCP/IP with cutoff radii 1.0, 1.4 and 1.8.</td>
</tr>
<tr>
<td>6.6</td>
<td>Runtimes of the rectangular Thrombin simulation over Ethernet using different synchronisation methods.</td>
</tr>
<tr>
<td>6.7</td>
<td>Runtimes of the octahedral Thrombin simulation over Ethernet using different synchronisation methods.</td>
</tr>
<tr>
<td>6.8</td>
<td>Runtimes of the rectangular Thrombin simulation over GM using different synchronisation methods.</td>
</tr>
<tr>
<td>6.9</td>
<td>Runtimes of the octahedral Thrombin simulation over GM using different synchronisation methods.</td>
</tr>
<tr>
<td>6.10</td>
<td>Runtimes of the rectangular Thrombin simulation over Myrinet and TCP/IP using different synchronisation methods.</td>
</tr>
<tr>
<td>6.11</td>
<td>Runtimes of the octahedral Thrombin simulation over Myrinet and TCP/IP using different synchronisation methods.</td>
</tr>
<tr>
<td>6.12</td>
<td>Runtimes of the rectangular Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.13</td>
<td>Runtimes of the octahedral Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.14</td>
<td>Runtimes of the rectangular Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.15</td>
<td>Runtimes of the octahedral Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.16</td>
<td>Load balancing of the parallel routines in the rectangular Thrombin simulation on the homogeneous cluster.</td>
</tr>
<tr>
<td>6.17</td>
<td>Load balancing of the parallel routines in the octahedral Thrombin simulation on the homogeneous cluster.</td>
</tr>
<tr>
<td>6.18</td>
<td>Load balancing of the parallel routines in the rectangular Thrombin simulation on the heterogeneous cluster.</td>
</tr>
<tr>
<td>6.19</td>
<td>Load balancing of the parallel routines in the octahedral Thrombin simulation on the heterogeneous cluster.</td>
</tr>
<tr>
<td>6.20</td>
<td>Speedup of the rectangular Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.21</td>
<td>Speedup of the octahedral Thrombin simulation on the homogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.22</td>
<td>Speedup of the rectangular Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.23</td>
<td>Speedup of the octahedral Thrombin simulation on the heterogeneous cluster using Myrinet over TCP/IP.</td>
</tr>
<tr>
<td>6.24</td>
<td>Predicted speedup of the Thrombin simulations on the Alpha cluster.</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 Thrombin molecular topology parameters for benchmarking ........................................ 58  
2.2 GROMOS96 profiling with Gprof on a SUN SPARC Station 10 using the octahedral Thrombin molecular topology ................................................................. 59  
2.3 GROMOS96 profiling with timing measurement routines on a SUN SPARC Station 10 using the Thrombin molecular topologies .......................................................... 60  
2.4 Number of floating-point operations to calculate the distance of two particles ................................................................. 63  
2.5 Number of floating-point operations to calculate the geometric centre of a CG ................................................................. 64  
2.6 Number of floating-point operations to calculate the interactions between two atoms ................................................................. 65  
2.7 Definition of the parameters for evaluating the number of floating-point operations during the Thrombin simulation ................................................................. 67  
2.8 Average number of floating-point operations for one run of the programming routines for the Thrombin simulation ................................................................. 68  
2.9 Average number of floating-point operations for one time step of the Thrombin simulation ................................................................. 69  
2.10 Performance of a SUN SPARC Station 10 for the Thrombin simulation ............................ 70  
2.11 Outsourcing of programming routines with percentage of acceleration and amount of communication data between host and coprocessor ................................................................. 72  

3.1 Programming languages and development tools for different hardware solutions ................................................................. 78  
3.2 Performance of parallel computer systems for MD algorithms ................................................................. 114  
3.3 Clock cycles of the distance routine in C on the SHARC ADSP-2106x ................................................................. 127  
3.4 Clock cycles of the distance routine in Assembler on the SHARC ADSP-2106x ................................................................. 128
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>Runtimes of the distance routine on the SHARC ADSP-2106x</td>
<td>130</td>
</tr>
<tr>
<td>3.6</td>
<td>Speedups of the distance routine on the SHARC ADSP-2106x</td>
<td>131</td>
</tr>
<tr>
<td>3.7</td>
<td>Clock cycles of the force routine in C on the SHARC ADSP-2106x</td>
<td>132</td>
</tr>
<tr>
<td>3.8</td>
<td>Clock cycles of the force routine in Assembler on the SHARC ADSP-2106x</td>
<td>133</td>
</tr>
<tr>
<td>3.9</td>
<td>Runtimes of the force routine on the SHARC ADSP-2106x</td>
<td>134</td>
</tr>
<tr>
<td>3.10</td>
<td>Speedups of the force routine on the SHARC ADSP-2106x</td>
<td>135</td>
</tr>
<tr>
<td>3.11</td>
<td>Runtimes of the Thrombin molecular dynamics simulation with 100 time steps on the SHARC ADSP-2106x</td>
<td>138</td>
</tr>
<tr>
<td>3.12</td>
<td>Performance of the SHARC DSPs for the Thrombin simulation</td>
<td>139</td>
</tr>
<tr>
<td>3.13</td>
<td>Performance of the SGI and Alpha for the Thrombin simulation</td>
<td>143</td>
</tr>
<tr>
<td>3.14</td>
<td>Characteristics and performance of different hardware solutions</td>
<td>143</td>
</tr>
<tr>
<td>4.1</td>
<td>Computation scheme with 4 PEs in a 2D-net with 6 full duplex links</td>
<td>152</td>
</tr>
<tr>
<td>4.2</td>
<td>Computation scheme with 4 PEs in a 2D-net with 6 half duplex links</td>
<td>152</td>
</tr>
<tr>
<td>4.3</td>
<td>Computation scheme with 4 PEs in a 2D-net with 4 half duplex links</td>
<td>153</td>
</tr>
<tr>
<td>4.4</td>
<td>Computation scheme with 8 PEs in a hyper cube</td>
<td>156</td>
</tr>
<tr>
<td>4.5</td>
<td>Computation scheme with 16 PEs in a recursive topology</td>
<td>158</td>
</tr>
<tr>
<td>4.6</td>
<td>Computation scheme with 6 PEs in a ring topology</td>
<td>161</td>
</tr>
<tr>
<td>4.7</td>
<td>Communication latencies of the different network topologies</td>
<td>163</td>
</tr>
<tr>
<td>4.8</td>
<td>Characteristics of the different network topologies</td>
<td>165</td>
</tr>
<tr>
<td>4.9</td>
<td>Characteristics and specified performances of the different off-the-shelf networks</td>
<td>173</td>
</tr>
<tr>
<td>5.1</td>
<td>Performance of the different communication ways for data packets with 512 KB</td>
<td>182</td>
</tr>
<tr>
<td>5.2</td>
<td>Amount of communication data during the Thrombin simulation</td>
<td>183</td>
</tr>
<tr>
<td>5.3</td>
<td>Estimated communication time for the Thrombin simulation on 4 processors</td>
<td>184</td>
</tr>
<tr>
<td>6.1</td>
<td>Parameters for predicting the speedup of the Thrombin simulations</td>
<td>218</td>
</tr>
<tr>
<td>6.2</td>
<td>Speedup of GROMOS96P on different computer clusters</td>
<td>220</td>
</tr>
<tr>
<td>7.1</td>
<td>Comparison of the runtimes of the Thrombin simulations on the reference machines and on the host with GROMOS coprocessor</td>
<td>226</td>
</tr>
<tr>
<td>7.2</td>
<td>Achieved speed factors, based on past and todays reference machines, using the GROMOS coprocessor</td>
<td>226</td>
</tr>
</tbody>
</table>
Bibliography


[75] Xilinx Inc.: XC6200 Field Programmable Gate Arrays: Data Book. 1997

[76] Xilinx Inc.: The Programmable Logic Data Book. 1994


Curriculum Vitae

Name: Thomas Peter Gössi
Date of Birth: 1st January 1971 in Lucerne, Switzerland
Nationality: Swiss, citizen of Küsnacht am Rigi, SZ

Education:
4.1978-6.1986 Primary and secondary school in Küsnacht am Rigi
8.1986-6.1991 Gymnasium in Immensee, Switzerland
6.1991 Matura Typus B at Gymnasium Immensee
3.1996 Diploma in Electrical Engineering (Dipl. El.-Ing. ETH) at the Swiss Federal Institute of Technology, Zurich.
Diploma thesis: *PCI Interface for Line Scan Camera*
4.1996-7.2000 Ph.D student at the Electronics Laboratory, ETH Zurich

Professional Experiences:
3.1992-4.1992 Workshop Internship at Siemens, Landis & Staefa Division, Zug, Switzerland
10.1993-2.1994, Student Assistant at the Computer Engineering and Networks Laboratory, ETH Zurich
10.1994-2.1995 Internship at the Swiss Federal Aircraft Factory, Emmen, Switzerland
7.1994-9.1994 Teaching and Research assistant at the Electronics Laboratory, ETH Zurich