ELSEVIER

Computer Physics Communications 91 (1995) 43–56

Computer Physics Communications

GROMACS: A message-passing parallel molecular dynamics implementation

H.J.C. Berendsen, D. van der Spoel, R. van Drunen

Bioson Research Institute and Laboratory of Biophysical Chemistry, The University of Groningen, Nijenborgh 4, 9747 AG Groningen, The Netherlands

Received 2 December 1994

Abstract

A parallel message-passing implementation of a molecular dynamics (MD) program that is useful for bio(macro)molecules in aqueous environment is described. The software has been developed for a custom-designed 32-processor ring GROMACS (GROningen MAchine for Chemical Simulation) with communication to and from left and right neighbours, but can run on any parallel system onto which a a ring of processors can be mapped and which supports PVM-like block send and receive calls. The GROMACS software consists of a preprocessor, a parallel MD and energy minimization program that can use an arbitrary number of processors (including one), an optional monitor, and several analysis tools. The programs are written in ANSI C and available by ftp (information: gromacs@chem.rug.nl). The functionality is based on the GROMOS (GROningen MOlecular Simulation) package (van Gunsteren and Berendsen, 1987; BIOMOS B.V., Nijenborgh 4, 9747 AG Groningen). Conversion programs between GROMOS and GROMACS formats are included.

The MD program can handle rectangular periodic boundary conditions with temperature and pressure scaling. The interactions that can be handled without modification are variable non-bonded pair interactions with Coulomb and Lennard-Jones or Buckingham potentials, using a twin-range cut-off based on charge groups, and fixed bonded interactions of either harmonic or constraint type for bonds and bond angles and either periodic or cosine power series interactions for dihedral angles. Special forces can be added to groups of particles (for non-equilibrium dynamics or for position restraining) or between particles (for distance restraints). The parallelism is based on particle decomposition. Interprocessor communication is largely limited to position and force distribution over the ring once per time step.

Keywords: Molecular dynamics; Parallel computing

PROGRAM SUMMARY

Title of program: GROMACS version 1.0

Program obtainable from: University of Groningen by ftp, information by email to `gromacs@chem.rug.nl`. Manual obtainable in postscript format, from
`ftp://ftp.chem.rug.nl/dist/gromacs/manual.ps`
or from
`http://rugmd0.chem.rug.nl/~gmx/gmx.cgi`
For those without internet access a tape distribution will be available.

Hardware obtainable from: CHESS Engineering B.V., Nieuwegracht 39, 2011 NC Haarlem, The Netherlands, fax: +31-23-323302.

Licensing provisions: The GROMACS source code is available for a nominal administration fee for **non-commercial** use. For commercial use a licensing fee is charged. To get access to the software a license agreement must be signed. To receive a license agreement and all other information needed to obtain GROMACS source code, as well as information on the licensing fee, please send an e-mail to `gmx-license@chem.rug.nl` with a message body of SEND LICENSE and an information sheet and license agreement form will be automatically sent to the `from:` address of the message.

Computer for which the program is designed and others on which it has been tested: GROMACS processor ring; Convex C220, HP 735, Sun, Silicon Graphics single processors; SGI Power Challenge, Convex Exemplar, CRAY C90 and IBM CP-2 multiprocessor systems.

Operating system under which the program has been tested: UNIX

Programming language used: C

Memory required to execute with typical data: Depends on number and type of atoms and cut-off range; typically a system of 10000 atoms will run on a 16 Mbyte single processor work station.

No. of bits in a word: 32 (64 optional); some variables use 64 bit

No. of processors used: up to 32; software allows $n \geq 1$ processors; communication to neighbours in a ring architecture

No. of lines in distributed program, including test data, etc.: approx. 80000

Keywords: molecular dynamics, GROMACS, parallel computation, particle decomposition

Nature of physical problem
Analysis and prediction of dynamic behaviour of (biological) macromolecules in solution; study of non-equilibrium processes under external driving forces; structural refinement using experimental constraints; optimization and regularisation of proposed structures.

Method of solution
Molecular dynamics simulation of (bio)macromolecules in a solvent, using classical equations of motion and force fields based on variable non-bonded interactions, and fixed bonded interactions. The system is coupled to an external bath of constant temperature and/or pressure. Rectangular periodic conditions are allowed. Bond lengths (and angles) can be constrained. External forces and force field terms related to experimental constraints can be added.

Restrictions on the complexity of the problem
The following restrictions apply to this version of GROMACS: *system size* limited by memory and number of processors and dependent on complexity of interactions; *long-range electrostatic interaction* is not explicitly treated and will restrict validity of ionic systems; *classical* dynamics limits validity if degrees of freedom are included with essential quantum character; *non-polarisable* force fields are used.

Typical running time
A typical small biomolecule (a peptide of 20 residues) in water (8600 atoms total) runs 100 time steps (0.2 ps) in one minute on a 32-i860 processor machine. This means 12 ps per hour or 1 ns in 3 days.

Unusual features of the program
The program consists of a serial preprocessor that takes care of system decomposition and molecular topology definition, the parallel MD program kernel, and a number of post-processing analysis programs. The pre- and postprocessors are ANSI C programs that can run on any workstation. The parallel MD program runs on any single-processor machine in its `nproc=1` option and runs on parallel machines, on which a ring of processors can be mapped and which support PVM-like send and receive calls, in its `nproc` > 1 option.

LONG WRITE-UP

1. Molecular dynamics of large molecular systems

Classical molecular dynamics simulations of molecular systems involves the solution of Newton's equations of motion in small time steps, based on cartesian coordinates of the particles and using a conservative force field. To avoid adverse boundary effects, periodic (but not necessarily cubic) boundary conditions are employed. The methods are well-known and will not be described here [1,2]. The GROMACS program is essentially based on the sequential GROMOS package [3], which was developed for the purpose of simulating bio(macro)molecules in solution. Descriptions of our prototype machine and of special software features incorporated in GROMACS have been published [4–6]. The design choices that were made for GROMACS include:

(i) There are three types of forces: *bonded forces* based on fixed lists and including up to four-body interactions, *non-bonded forces* based on dynamic lists of particle pairs, and *external forces* allowing non-equilibrium driving forces. The force-field terms used are summarized in Section 3.

(ii) The non-bonded force calculation is based on a *pair list* which is updated every n steps. Particles are grouped in *charge groups* containing one or a few particles. The criterium for inclusion in the list is whether the centers of the charge groups are within a given cut-off radius. This procedure avoids the 'creation' of charges from neutral groups by applying a cut-off criterium to individual pairs of partially charged atoms [7]. The parallel pair list generation is decribed in Section 4.

(iii) Optionally a *twin-range cut-off* can be used: while making up the pair list with charge-group cut-off radius R_{short}, the Coulomb forces between particles of charge groups at a distance between R_{short} and R_{long} are computed, accumulated per atom and stored. These *long-range shell* forces are kept constant over n time steps and added to the short-range forces. This represents the simplest form of the multiple time-step method of Street et al. [8].

(iv) The *leap-frog* algorithm [9], which is equivalent to the Verlet [10] algorithm, is used to solve the equations of motion. This involves positions at integer times (measured in time steps) and velocities at half-integer times. The system is coupled by a first-order response to a 'bath' of constant temperature and pressure [11] with adjustable time constants. This coupling is realized by scaling the velocities, resp. coordinates, based on temperature, resp. pressure, at the previous time step. The update algorithm is decribed in Section 2.

(v) Covalent bond lengths and angles can be optionally constrained; the resulting constrained equations of motions are solved by the SHAKE algorithm. [12] SHAKE changes an unconstrained configuration $\{\boldsymbol{r}'\}$ into a constrained configuration $\{\boldsymbol{r}\}$ with displacement vectors in a direction given by a reference configuration $\{\boldsymbol{r}_{\text{ref}}\}$; this is denoted by

$$\text{SHAKE}(\boldsymbol{r}' \rightarrow \boldsymbol{r}\,;\,\boldsymbol{r}_{\text{ref}})$$

SHAKE uses the coordinates of the previous step as reference. The correction $\boldsymbol{r}' - \boldsymbol{r}$ represents the constraint forces acting on the particles, multiplied by $(\Delta t)^2/m_i$, and can be used to derive the contributions of the constraint forces to the pressure.

2. The update algorithm

The following algorithm for the update of velocities and coordinates is used [11]:

Given at time t: velocities $\boldsymbol{v} = \boldsymbol{v}_i(t - \frac{1}{2}\Delta t)$, coordinates $\boldsymbol{r} = \boldsymbol{r}_i(t)$, box vectors $\boldsymbol{b}$, velocity and coordinate scaling factors λ and μ.

The scaling factors are given by

$$\lambda = \left[1 + \frac{\Delta t}{\tau_t}\left\{\frac{T_0}{T(t - \frac{1}{2}\Delta t)} - 1\right\}\right]^{1/2}, \qquad \mu = \left[1 + \frac{\Delta t}{\tau_p}\beta\{P(t) - P_0\}\right]^{1/3}.$$

Here T_0 and P_0 are the reference temperature and pressure and β is an estimate for the compressibility. The program allows different temperature scaling for different groups of atoms. The scaling factor μ is a diagonal *tensor* in case an anisotropic scaling is used and the pressure is also expressed as a (diagonal) tensor. The coupling time constant τ_p may also be chosen to be anisotropic, thus even allowing pressure scaling in one dimension and constant size simulation in another.

(i) Compute accelerations: $\boldsymbol{a} = \boldsymbol{F}_i(t)/m_i$, where $\boldsymbol{F}$ is the force disregarding constraints.

(ii) Update and scale velocities: $\boldsymbol{v}' = \lambda(\boldsymbol{v} + \boldsymbol{a}\Delta t)$.
(iii) Compute new unconstrained coordinates: $\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{v}'\Delta t$.
(iv) Apply SHAKE to coordinates: SHAKE($\boldsymbol{r}' \to \boldsymbol{r}''$; $\boldsymbol{r}$).
(v) Correct velocities for constraints: $\boldsymbol{v} = (\boldsymbol{r}'' - \boldsymbol{r})/\Delta t$.
(vi) Scale coordinates and box: $\boldsymbol{r} = \mu \boldsymbol{r}''$; $\boldsymbol{b} = \mu \boldsymbol{b}$.
The particle position may drift outside the box. They are reset in the box before a new pair list is made up. There are provisions to "freeze" (prevent motion of) selected particles.

3. Force field

There are several choices for the three types of forces. We list the equations for the potential energy; the forces are simply derivatives with respect to particle coordinates. The choice of parameters is made by including appropriate files in the preprocessing stage; at present GROMOS and OPLS [13] force fields files are available. In this version of GROMACS analytical rather than numerical derivatives are implemented. Note that non-bonded forces are only considered for pairs of atoms that belong to charge groups within a given cut-off radius. This means that, although forces are derivatives of potentials, the potentials V are not equal to the integrals of the forces V':

$$V(r) \neq V'(r) = \int_{-\infty}^{r} F(r')dr'.$$

We define: $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$ and $r = |\boldsymbol{r}|$.

3.1. Non-bonded forces

Based on a pair list that also contains information on the image shift that must be made to obtain the nearest image of the considered pair, central force components are computed:

$$\boldsymbol{F}_i = -\boldsymbol{F}_j = -\frac{\partial V(r_{ij})}{\partial r_{ij}} \frac{\boldsymbol{r}_{ij}}{r_{ij}}.$$

The possible interactions are
- *short-range repulsion:* $V(r) = C_{12}/r^{12}$ or $V(r) = A\exp(-Br)$,
- *dispersion interaction:* $V(r) = -C_6/r^6$,
- *Coulomb interaction* fq_iq_i/r,

The parameters C_{12}, C_6, A, B are dependent on the *atomtypes* of both atoms of a pair. The charges q_i are *attributes* of each atom (not of atomtypes, thus allowing more flexibility in force field adjustments). The constant f is an electric factor equal to $1/4\pi\varepsilon_0$; for the units used in GROMACS: (mass: atomic mass units, length: nm, time: ps, charge: elementary charge e, and therefore energy: kJ/mol) $f = 138.935\,485(\pm 9)$.

3.2. Bonded forces

Bonded forces are based on a fixed list of atoms per interaction. The following types are incorporated:
Covalent bonds: This is a two-body harmonic potential $V(r) = \frac{1}{2}k_b(r - b)^2$, which can be replaced by a constraint.
Covalent bond angles: This is a three-body harmonic potential

$$V(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3) = \tfrac{1}{2}k_\theta(\theta - \theta_0)^2,$$

where θ is the angle between r_{21} and r_{23}. The bond angle potential can also be replaced by a constraint of the distance 2–3 if the two bonds 1–2 and 2–3 are constrained as well.
Dihedral angles: This is a four-body potential

$$V(r_1,r_2,r_3,r_4) = \tfrac{1}{2}V_0[1+\cos(n\phi-\phi_0)]\,,$$

where ϕ is the angle between the planes defined by $\{r_{12},r_{31}\}$ and $\{r_{32},r_{43}\}$. In conjunction with this periodic potential a special pair interaction of the form $C_{12}/r^{12} - C_6/r^6$ can be added between particles 1 and 4. An allowed alternative form is the Ryckaert-Bellemans potential [14]

$$V(\phi) = \sum_{n=0}^{5} a_n \cos^n \phi\,.$$

Improper dihedrals: These are four-body harmonic potentials used to keep groups planar or to prevent dihedral angles excursions that could lead to unwanted mirror images.
Distance-restraining potentials: These are meant for the incorporation of experimental data on the distance between given atom pairs, such as can be derived from Nuclear Overhauser Effect measurements in nuclear magnetic resonance (NMR). They are based on a special fixed pair list and have the form of a continuous function with continuous derivative, with $V = 0$ for $r < r_1$, increasing quadratically between r_1 and r_2 and linearly beyond r_2.

3.3. External forces

Position-restraining forces: Forces that represent an external potential

$$V = \tfrac{1}{2}\sum_i k_i(r_i - r_i^0)^2\,,$$

where r_i^0 is a given reference structure, can be added on the basis of a separate fixed list. Its purpose is to restrain a structure close to a given reference, e.g. in an outer shell for reduced systems, or during an equilibration to avoid unrealistic structural changes.
External accelerations: There are provisions to add external forces in the form of additional accelerations that are applied to specified *groups* of atoms. They can be used in NEMD (Non-equilibrium molecular dynamics).

4. Parallel concepts

There are two basically different methods to parallelize an algorithm: *data parallel* and *message passing* methods. The former method allows the user to define arrays on which to operate in parallel; programming in this way is much like vectorising. Its advantage is that it is easier for the user; the parallelization is taken care of by the compiler. Its disadvantages are that standard data-parallel programming languages are still under development and that the level of parallelization can not always be optimized. With message-passing methods all parallelism is explicitly programmed by the user. The disadvantage of the latter is that it takes extra code and effort, but the advantage is that the programmer keeps full control over the data flow and can do optimisations that a compiler would never generate. For better efficiency, portability and for historical reasons we chose message-passing as the programming method. Our program was initially designed for a special-purpose machine with a ring architecture and without tools for data-parallel programming.

The approach we took to parallellism was a minimalist one: use as few non-standard elements as possible and use the simplest processor topology that does the job. We therefore decided to use a standard language (ANSI-C) with as few non-standard routines as possible: Only five non-standard communication routines are used. The simplest communication architecture, a ring of processors, provides all functionality that is needed; a ring can be mapped onto almost all commercial parallel computers (including hypercube and tree architectures) and a ring can be realized with a minimum of special hardware requirements.

When using a message-passing scheme one has to divide particles over processors, which can be done in two ways, either by *space* (or *domain*) decomposition or by *particle* decomposition. Space decomposition divides real space over the processors, in the case of a ring architecture and homogeneous density of particles most logically in slabs of equal width. It has the advantage that neighbour search can be done locally (involving a limited number of neighbouring processors) rather than globally, and that coordinates and partial forces have to be communicated only locally to compute forces. But this advantage only works if the cut-off radius is small compared to system size. A disadvantage is that particles move between processors, so much bookkeeping is required to keep track of all complex data related to fixed interaction lists such as occur in macromolecules with 3- and 4-body interactions and with SHAKE. Particle decomposition means that particles are allocated to processors (thus each particle has its 'home processor') and each processor computes interactions for its 'home particles', which do not exchange between processors. Programming with particle decomposition is more straightforward at the expense of more communication. The latter is not severe in our cases of interest where cut-off radii are not really small compared to system size.

We have chosen for a simple particle decomposition and for a communication scheme that sends all coordinates and partial forces once per time step around *half* the ring. Thus each processor keeps a copy of half of all coordinates. This seems a wasteful use of memory because only a fraction (maybe 20%) of these coordinates are really needed, but memory requirements are overwhelmingly determined by the storage of pair lists. Of the latter only the local part is made up and stored by each processor.

The need to communicate only over half the ring is related to the use of action = reaction (Newton's third law). For every pair i,j the force $\boldsymbol{F}_{ij}$ needs to be calculated once; this means that the pair coordinates need to be present in only one processor. Communicating over half the ring implies that for every pair i,j either $\boldsymbol{r}_i$ is transmitted to the home processor of particle j or $\boldsymbol{r}_j$ is transmitted to the home processor of particle i. After the forces have been computed and partial sums (over each processor) have been made up, these partial sums are transmitted over half the ring (in a direction opposite the coordinate communication), after which the forces can be globally summed.

The pairs should be evenly distributed over the processors for proper load balancing. The communication scheme mentioned above allows to assign an equal number of non-bonded neighbours to each particle in its home processor. The pairs (i,j) considered by the home processor of i do not simply consist of all $j : i < j \leq n$ (a shorter list for larger i) but rather of $j = k \bmod n$ for all $k : i < k \leq (i + n/2)$ (a list of constant length).

A special method has been implemented to avoid determination of the nearest image each time forces are evaluated. While the pair list is made up, the nearest image is determined by a box displacement vector for each pair. The displacement vector is one of 27 possible vectors and is completely identified by an index 1...27. This index is kept in the pair list and remains valid until the pair list is updated. The use of the image identifier also enables an efficient summing of terms for the *virial* calculation, which is required to compute the pressure. The virial Ξ from non-bonded pair interactions in a periodic system is given by

$$\Xi = -\tfrac{1}{2} \sum_{\text{pairs } i,j} \boldsymbol{r}_{ij} \cdot \boldsymbol{F}_{ij} \, .$$

The virial summation is normally carried out in the inner loop of the double (i,j) loop, but this can be avoided and reformulated as a single sum. A full description of this method is given in Ref. [16].

5. The GROMACS algorithm

After loading the program and data, the MD program performs a preset number of time steps. After each specified number of steps output may be written. Every MD time step consists of the following phases:

(i) Starting from its home processor, the position of each particle i is distributed anticlockwise over half the ring.

(ii) If required (e.g., once every 20 time steps), a neighbour list is constructed and 'shell forces' from particles between R_{short} and R_{long} are computed.

(iii) All forces are computed and partial sums of forces are accumulated in each processor.

(iv) Partial sums of forces on every particle i are communicated, going in clockwise direction over half the ring. The forces are summed to net forces in each home processor.

(v) Using the net forces, velocities and positions are updated for every particle on its home processor. If required, SHAKE is performed.

The last point needs clarification. In the present version of GROMACS, SHAKE has not been parallelized yet. SHAKE is an iterative procedure, extending over a molecule, the parallelization of which needs special care. A parallelized version has been proposed by DeBolt and Kollman [15]. There is no need for a parallel version if SHAKE is only performed on molecules that are entirely situated on one home processor (as can be set in GROMACS), or if SHAKE is only applied to bonds or bond angles involving hydrogen.

6. GROMACS preprocessor

The gromacs software employs a *preprocessor* which has to be run before the simulations or minimization can be started. The preprocessor is not parallellized. The reasons to separate the preprocessing phase are the following:

(i) Flexibility. A user can prepare GROMACS runs on a local system without occupying a parallel system. Parsing input files and processing the input takes time, proportional to the number of atoms. For a large system (e.g. 10000 atoms) this is in the order of a few minutes on a workstation. There is also a considerable amount of disk I/O necessary which can remain local: only one file communicates with the simulation engine. The required disk space is often prohibitive on our special purpose MD computers.

(ii) Memory. As we use special purpose MD engines with a limited amount of RAM, preprocessing is not possible there on a single processor for large systems.

(iii) Reusability. The binary file that is produced by the preprocessing software is also used as an input file in many analysis tools that require information about the molecular topology (e.g. the trajectory viewer)

(iv) Clarity of the code. Because of the broad functionality of the preprocessor, the amount of code is large ($>$ 10000 lines), separating this code from the MD code implies that bugs in the preprocessing software do not invalidate the MD software.

The gromacs preprocessor (`grompp`) reads a molecular topology file, checks the validity of the file, and expands the topology from a molecular description to an atomic description. The topology file contains information about molecule types and the number of molecules, the preprocessor copies each molecule as needed. There is no limitation on the number of molecule types. Bonds and bond-angles can be converted into constraints, separately for hydrogens and heavy atoms. Then a coordinate file is read and velocities can be generated from a Maxwellian distribution if requested. `grompp` also reads parameters for the mdrun (eg. number of MD steps, time step, cut-off), and others such as NEMD (Non-Equilibrium Molecular Dynamics) parameters, such as external accelerations, which are corrected so that the net acceleration is zero. Eventually a binary file is produced that can serve as the sole input file for the MD program (see Fig. 1)

Another important program is pdb2gmx, the pdb to gromacs converter. This program converts a Brookhaven Protein Data Bank file to a molecular topology, generates hydrogen positions and disulphide bridges. It uses a

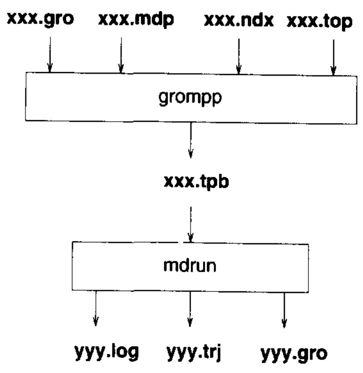

Fig. 1. Data flow scheme from preprocessor to mdrun.

Table 1
File formats: A stands for ascii, B for binary

Name	Ext.	B/A	Opt.	Description
grompp	.mdp	A	-f	grompp input file with MD parameters
traj	.trj	B		Trajectory file
conf	.gro	A	-c	Coordinate file in Gromos-87 format
run	.log	A	-l	Log file from MD run
index	.ndx	A	-n	GROMACS indexfile
topol	.top	A	-p	GROMACS topology file
topol	.tpb	B	-s	Binary GROMACS topology

database of *residues* to build up lists of bonds. From these bonds, angles and dihedrals are generated, dihedrals are compared to a list of improper dihedrals from the residue database, and converted when necessary. From every possible description of a torsion angle one is chosen, such that the number of hydrogen atoms describing the torsion is minimal. The force constants in the Gromos forcefield are such that a torsion is modelled by a single potential. The program is not restricted to proteins, but can handle any residue type that is in the database.

A number of other programs and tools that we will not describe in detail here are available; among these are conversion utilities for Gromos and OPLS forcefield files.

The files listed in Table 1 are in use.

7. Communication primitives

The communication scheme used in the GROMACS software is based on a ring architecture, so every processor only communicates with its nearest two neighbours (i.e., the processor on the left and the processor on the right). To implement the interprocessor communication structure we have designed a communication interface consisting of six different routines. This communication interface is designed in such a way that the calls are easily mappable onto other message passing paradigms. The choice of designing this propiarity communication interface instead of using an existing paradigm is made with the background of our own ring-based hardware. Implementing a so-called message passing 'standard' on this hardware would be too much effort for the simple ring-communication protocol the GROMACS software is using.

7.1. GROMACS implementation

As stated before the message passing interface designed for GROMACS consists of 6 different routines. The implementation of these communication primitives is located in one single file, in order to minimize the effort needed for porting the software to another message passing paradigm. A brief overview of the primitives used is given here.

`void network_tx(int chan,void *buf,int bufsize)`
This routine asynchronously sends `bufsize` bytes from the buffer pointed to by the pointer `buf` over the communication channel. The channel is identified by `chan`.
`void network_tx_wait(int chan)`
This routine implements a wait function until the asynchronous send operation associated with the communications channel designated by `chan` has ended.
`void network_txs(int chan,void *buf,int bufsize)`
This routine synchronously sends `bufsize` bytes from the buffer pointed to by the pointer `buf` to the processor or the process identified by `chan`.
`void network_rx(int chan,void *buf,int bufsize)`
This routine asynchronously receives bufsize bytes in the buffer pointed to by te pointer `buf` from an communication channel identified by `chan`.
`void network_rx_wait(int chan)`
This routine implements a wait function until the asynchronous receive operation, associated with the communications channel designated by `chan`.
`void network_rxs(int chan,void *buf,int bufsize)`
This routine asynchronously receives `bufsize` bytes from the buffer pointed to by the pointer `buf` over the communication channel identified by `chan`.
`void network_init(int pid,int nprocs)`
This call initialises the network. On various machines one needs to for example define the network communications and open the channels for use. This is being done in this routine. It sets the variable `pid` to a unique id for the processor it is executed on, and sets `nprocs` to the number of processors allocated for this particular run.
`void get_left_right(int nprocs,int pid,int *left,int *right)`
This routine maps the internal representation of the variables `right` and `left` to the processor numbers of the processors that are virtually located on the 'left' and on the 'right' of the calling processor. The GROMACS program assumes a **ring** communication topology. Therefore it will only call the communication routines with a `chan` value of either `left` or `right`.

The implementation of the **synchronous** send and receive calls can be done by calling the corresponding **asynchronous** call followed by a either `rx` or `tx_wait` call. This reduces the number of message-passing paradigm specific calls to six. The message passing calls presented here are implemented directly onto the GROMACS hardware.

7.2. PVM implementation

To make the GROMACS program run on a machine that supports the PVM (Parallel Virtual Machine) [17] library as a message-passing paradigm one need to make an implementation of the C-routines described in ref [17] that makes use of the calls supplied by the PVM library. The C source code of such an implementation is given below for the network_tx,network_txs, and network_tx_wait as an example. The complete implementation of all communication functions can be found in the file pvmio.c in the program code.

```
#define network_tx   pvmio_tx
#define network_txs  pvmio_txs
#define network_tx_wait pvmio_tx_wait
#define DEFAULT_STRIDE 1

void pvmio_tx(int pid,void *buf,int bufsize)
{
  int bufid;
  bufid= pvm_initsend(PvmDataRaw);
  info = pvm_pkbyte( buf, bufsize, DEFAULT_STRIDE);
  info=pvm_send(pid,TAG);
}

void pvmio_tx_wait(int pid)
{
#ifdef PROFILING
    idle_send++
#endif
      ;
}

void pvmio_txs(int pid,void *buf,int bufsize)
{
  pvmio_tx(pid,buf,bufsize);
  pvmio_tx_wait(pid);
}
```

7.3. TCGMSG implementation

GROMACS will support the tcgmsg message passing layer as well. Some vendors have special optimized versions of this library for their machines in addition to other libraries. GROMACS has a special routine tcgio.c that can be used on machines that support this library. The implementation of the GROMACS message passing functions using the tcgmsg calls is straightforward.

7.4. MPI implementation

Although in the current version GROMACS does not yet support the MPI (Message Passing Initiative) [18] as a communication paradigm it is as easy to implement as the PVM version of the program. MPI supports a number of extra features that will make the communication within the program more efficient, but when using these features one has to change the MD-kernel code as well as the message passing interface of GROMACS. This will result in less compatibility over the different soft- and hardware platforms with different (non-MPI) message passing layers.

8. Performance tests

When comparing benchmark results for various machines running GROMACS a number of different quantities should be compared. At first we can compare the performance of GROMACS to another MD program running a simulation on the same molecular system with the same simulation parameters on the same computer. For the functionally similar GROMACS and GROMOS programs such comparisons have been made on a Convex C-220 single processor. GROMACS is roughly twice as fast as GROMOS, presumably due to optimizations in neighbour search and pair list structure.

Secondly we can compare the performance of GROMACS running the same simulation on different computers, and thus comparing the computer performance. Several such tests have been made (on Cray C-90, IBM SP-2, Convex Exemplar, Silicon Graphics Power Challenge). We do not list the results here since a fair comparison is complicated by the effects of different individual optimizations.

A comparison of our 32-processor prototype machine with CRAY Y/MP and NEC SX-3 (both single processor) on simulations of water, using optimized programs in all cases, has been published earlier [4]. Our 32-processor system exceeds the Y/MP performance by a factor of 6 and the SX-3 by a factor between 3 and 6.

A third test concerns multi-processor runs with a varying number of processors on the same computer. We then can observe the scaling factor. The scaling factor depends highly on the time ratio between computation phases and the communication phases in a single simulation.

Here we present a number of timings on a multi-processor i860 machine for different simulations, as well as different runs of the same system performed on an increasing number of processors.

8.1. Test systems

Three different test systems have been selected in order to form the GROMACS benchmark suite.

- Water, 216 molecules; 1728 molecules. Update of the neighbourlist every 10 time steps, single cut-off .85 nm.
- Box of Water with small peptide (20 residues), 8652 particles, Update of the neighbourlist every 10 time steps, single cut-off 1.0 nm, timings of 100 simulation time steps

The results of the benchmarks can be found in Table 2. Scaling tests are performed on the same i860 multi-processor system, running the peptide simulation for 100 steps on 6 - 32 processors. The results of this scaling experiment are displayed in Fig. 2

8.2. Results

Here we can see that the scaling performance is not optimal. The reason for this is partly that the 'peptide-in-water' problem is not very well balanced: the peptide is completely kept on one processor in order to allow SHAKE to be used, while the water molecules are distributed. Further optimisation will certainly be possible.

9. Input and output conventions

One of the key design initiatives of the GROMACS software was user-friendlyness. This is particularly important for the data formats used. The input files must be clear, readable ASCII text and comments must be possible. Also an interface to the standard file formats used in the field of biophysical chemistry (eg. The Brookhaven Protein Data Bank (PDB) format and eg. GROMOS) must be available.

The output files have to be generic, to allow the user to write his own analysis programs in addition to the

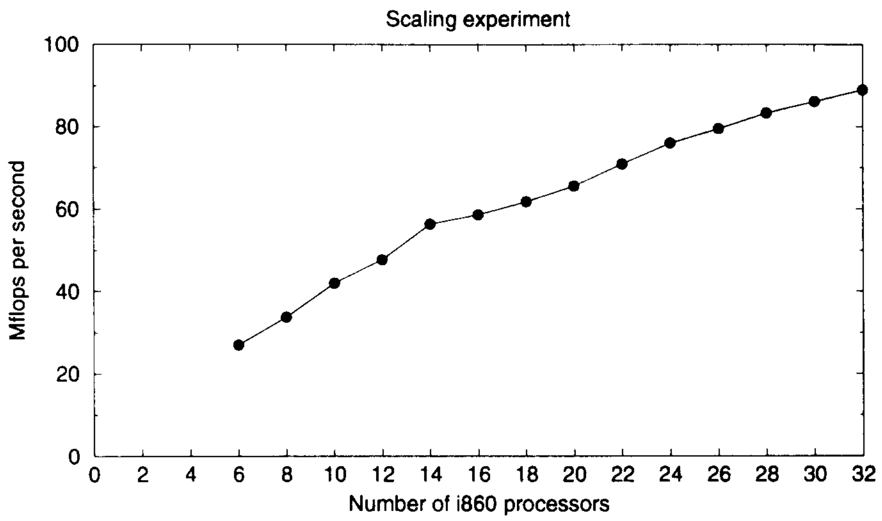

Fig. 2. Scaling performance for peptide in water.

Table 2
Benchmark results with GROMACS on a multi-processor i860 machine; Mflops have been 'counted' in the program; non-floating point operations are not included (* : not enough memory)

System	Particles	Steps	CPU	Mflop	CPU-time (s)	Mflop/s
water216	648	1000	4	3098	154	20.1
water216	648	1000	28	3098	77	40.2
water1728	5148	1000	4	25033	1294	19.3
water1728	5148	1000	28	25033	299	83.7
peptide	8652	100	4	5246	*	
peptide	8652	100	6	5246	194	27.0
peptide	8652	100	28	5246	59	88.9

analysis tools provided with the GROMACS software. A library of C-functions to read the binary output files of the MD kernel, as well as a program to convert these files into a (human-readable) ASCII format is available.

10. Hardware

For running GROMACS at our laboratory, a special piece of hardware has been constructed by CHESS Engineering B.V., located in Haarlem, the Netherlands. This multi-processor machine has a **ring** communication structure to fit the requirements of the program without imposing routing overhead when using another topology. The GROMACS machine consists of a number of custom designed, VME-sized CPU-boards, each holding two Intel i860 CPU's running at 40 MHz. These CPUs are particularly suited for scientific calculations because of their dual vector-oriented floating point units. The theoretical peak performance of the i860 CPU is 80 Mflop per second.

Each VME board holds 8 Mbytes of dynamic memory per CPU (this memory is expandable to 64 Mbytes per CPU). The CPUs on the board are totally independent of each other. Per CPU a custom designed DMA (Direct Memory Access) unit is used to drive the two 8-bit communication channels. The interprocessor communication

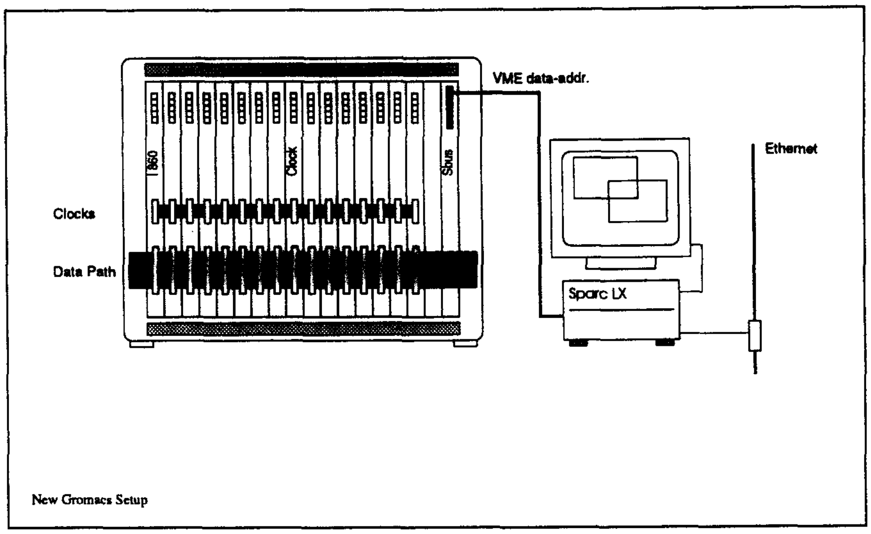

Fig. 3. Gromacs setup.

speed over these DMA channels is 10 Mbytes per second when communicating sufficiently large blocks of data (typically 100 Kbytes).

Currently the machine in our laboratory holds 16 CPU boards, making a total of 32 CPUs on a single VME backplane. The machine is connected by means of a VME-to-sbus coupler to a SUN-LX type workstation that is used as a host. Fig. 3 shows a the setup. The SUN machine provides a easily network-accessible programming environment (UNIX) and the necessary disk storage.

11. Future improvements

The following improvements to the GROMACS software are intended:

(i) Oblique (triclinic) box sizes and tensorial pressure coupling, thus allowing all kinds of primitive cells.
(ii) Long-range interactions by augmenting a modified short-range interaction with a Poisson solver. The need for charge groups and twin-range interactions will then become obsolete.
(iii) Parallel SHAKE.
(iv) Automatic load balancing by redistribution of particles over processors.
(v) More flexible force field handling.
(vi) Polarisable shell model [19].

References

[1] M.P. Allen and D.J. Tildesley, Computer Simulations of Liquids (Clarendon Press, Oxford, 1987).
[2] W.F. van Gunsteren and H.J.C. Berendsen, Angew. Chem., Int. Ed. Engl. 29 (1990) 992.
[3] W.F. van Gunsteren and H.J.C. Berendsen, GROMOS (GROningen MOlecular Simulation package), BIOMOS B.V., Nijenborgh 4, 9747 AG Groningen, The Netherlands.
[4] H. Bekker, H.J.C. Berendsen, E.J. Dijkstra, S. Achterop, R. van Drunen, D. van der Spoel, A. Sijbers, H. Keegstra, B. Reitsma and M.K.R. Renardus, in: Physics Computing '92, R.A. de Groot and J. Nadrachal, Eds. (World Scientific, Singapore, 1993) p. 252.
[5] *Ibid.* p. 257.
[6] H. Bekker, E.J. Dijkstra and H.J.C. Berendsen, Supercomputer 54 (1993) 4.

[7] H.J.C. Berendsen, in: Computer Simulation of Biomolecular Systems, W.F. van Gunsteren, P.K. Weiner and A.J. Wilkinson, Eds. (Escom, Leiden, 1993) p. 161.
[8] W.B. Street, D.J. Tildesley and G. Saville, Mol. Phys. 35 (1978) 639.
[9] R.W. Hockney and J.W. Eastwood, Computer Simulation Using Particles (McGraw-Hill, New York, 1981).
[10] L. Verlet, Phys. Rev. 165 (1968) 201.
[11] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola and J.R. Haak, J. Chem. Phys. 81 (1984) 3684.
[12] J.P. Ryckaert, G. Ciccotti and H.J.C. Berendsen, J. Comp. Phys. 25 (1977)327
[13] W.L. Jorgenson and J. Tirado-Reeves, J. Am. Chem. Soc. 110 (1988) 1657.
[14] J.P. Ryckaert and A. Bellemans, Faraday Disc. Chem. Soc. 66 (1978) 95; Chem. Phys. Lett. 30 (1975) 123.
[15] S.E. DeBolt and P.A. Kollman, J. Comput. Chem. 14 (1993) 312.
[16] H. Bekker, E.J. Dijkstra, M.K.R. Renardus and H.J.C. Berendsen, Mol. Simul. 14 (1995) 137.
[17] V.S. Sunderam, G.A. Geist, J. Dongarra and R. Manchek, Parallel Comput. 20 (1994) 531.
[18] R. Hempel, Lect. Notes Comput. Sci. 797 (1994) 247.
[19] P.C. Jordan, P.J. van Maaren, J. Mavri, D. van der Spoel and H.J.C. Berendsen, J. Chem. Phys. 103 (1995) 2272.