PARALLEL MOLECULAR DYNAMICS

Terry W. Clark
Department of Computer Science
University of Houston

J. Andrew McCammon
Department of Chemistry
University of Houston

L. Ridgway Scott
Department of Mathematics
University of Houston

appears in: Proc. Fifth SIAM Conf. on Parallel Proc. for Sci. Comp., J. Dongarra *et al.* ed's, Philadelphia: SIAM, 1991, 338–344.

Abstract. We describe the results of a preliminary port of a large molecular dynamics (MD) code, **gromos** [15], to a distributed–memory parallel computer. The objectives of this study were three fold. First, we wanted to assess the suitability of our software tools [14] for porting existing Fortran codes (dusty decks). This involved developing Fortran versions of previously developed parallel extensions to the C programming language. Secondly, we wanted to be able to quantify various components of an MD simulation with respect to communication and computation costs. Here the objective was to have timing data that would aid in the design of a scalable code that could execute efficiently on a large number of processors. Finally, we wanted to see if execution speeds comparable to a Cray could be achieved easily with a modest number of current–generation nodes. Such performance seemed feasible based on a rough assessment of current processor performance and communication speed. One of our principal conclusions is that we were able to achieve performance on a standard benchmark problem comparable to reported performance of a Cray–2 using between 8 and 16 processors on an iPSC/860. Moreover, this was done with a minimal amount of coding and debugging, due primarily to the use of our software tools.

1. Description of MD algorithms. The basic model of molecular dynamics [1, 5, 12] involves only classical mechanics, although the forces, F, between atoms are more complicated than the gravitational or electrostatic forces encountered in basic mechanics. The model reduces to a system of ordinary differential equations for the positions, x, of the atoms which express essentially that

$$F = m\frac{d^2x}{dt^2}.$$

The most computationally intensive part of the simulation is the evaluation of the forces between atoms (on the face of it, a computation involving $\mathcal{O}(N^2)$ operations for N atoms). This consists of two parts, the *bonded* forces among atoms linked by covalent bonds, and the *non–bonded* forces among atoms not so linked. The latter involve the greater amount of work, since the bonded forces require only $\mathcal{O}(N)$ operations for N atoms.

Efficient approximation algorithms for evaluating the non–bonded forces among atoms have been introduced recently using diverse techniques. An approximation algorithm for evaluating the Coulombic forces has been considered by Greengard and Rokhlin [9] using multipole expansions. A related algorithm for computing simultaneously the Coulombic and short–range forces (e.g. Lennard–Jones potentials) has been introduced by Draghicescu in her thesis [8]. Both of these techniques (as well as techniques that involve solving a Laplace equation for the electrostatic potential [7]) yield a close approximation to the actual interaction among all particles. At the moment, techniques of this type are not in wide–spread use in existing molecular dynamics codes. However, we intend to examine such algorithms for possible use in the next version of our port of `gromos`.

Existing codes employ, typically, some sort of *cut–off radius* outside of which a very simple approximation to the forces among atoms is employed. Interactions between an atom and its neighbors inside that atom's cut–off radius are computed exactly. The key to efficiency in this approximation is the way lists of neighboring atoms are handled. The approach taken by `gromos` will be described in detail in the next section. Another effective implementation divides space into boxes [13] and keeps track of particles inside each box via a *linked list*. The spatial subdivision simplifies updating the neighbor lists. (Atoms may move from one box to another during the computation.) Only neighboring boxes can have interactions if the cut-off radius is smaller than the dimension of the boxes. Boris *et al.* ([3], [11], [4]) use a more specialized data structures geared toward vector-processing architectures for this problem.

2. The `gromos` *code and its parallelization.* The `gromos` suite [15] consists of 39 separate programs in 127 files, with over 74K lines of Fortran. The MD module comprises 27 of these files with over 14K lines of Fortran, of which the longest subroutine is over 1,000 lines! Our efforts at parallelization were focused only on the MD module. Our work began with a version of the code that had been developed previously for shared–memory MIMD machines [6].

A generic flow chart for MD simulations using a cut–off radius, augmented to include algorithmic features of our parallel version, is shown in Figure 1. The asymptotic time complexity of each part as a function of the number, N, of atoms, the number, P, of processors and the cut–off radius, R_c, is indicated in the figure. The main parts of the computation are updating the pair list, computing the non–bonded forces and the time-stepping. In the `gromos` code, the pair list is treated as a pair of arrays, `INB` and `JNB`. For each atom, `I`, `INB(I)` is the number of neighboring atoms, `J`>`I`, within the cut–off radius, and `JNB(Start(I)+1:Start(I)+INB(I))` is the corresponding set of atom indexes (`Start(I)`$= \sum_{k=1}^{I-1}$ `INB`(`k`)).

To compute these arrays, `gromos` loops over pairs of atoms, involving $\mathcal{O}(N^2)$ work. This, however, is done only every few (ten, in the benchmark runs) time–steps, so that for typical problems, the amount of work is minimal compared with the computation of the non–bonded forces. But, we wish to improve this in later modifications to `gromos`.

Data parallelism can be exploited simply by subdividing both the `INB` and `JNB` arrays appropriately. Firstly, the arrays themselves can be computed in parallel. The amount of work to be done for each `I` is proportional to `N-I` (compute the distance from atom `I` to the remaining atoms). Thus the work load can be viewed as a triangle that must be divided into trapezoids of roughly equal area. For P processors, one can laboriously verify that the balanced workload for the j-th processor is to compute `INB(I)`, and the corresponding part of `JNB`, for `I` from `L(j)+1` to `L(j+1)` where

$$L(j) = N - \left(\frac{(P-j)N^2 + j}{P} \right)^{\frac{1}{2}}.$$

Once `INB` and `JNB` are computed, the work load to compute the non–bonded forces is likely not well balanced using the indices just described in the scheme for pairlist generation. Thus it is essential to distribute the load in such a way that the cumulative lengths of the parts `JNB` assigned to each processor are roughly balanced. First of all, a global accumulation of `INB` is done (in $\log P$ steps on a hypercube) which provides each processor a complete picture of the total work load. The balanced work load can be visualized as an atom sequence `JNB` that needs to be divided into groups of whole subunits (`JNB` segments) consisting of the neighbors of blocks of atoms, with the lengths of the subunits approximately the same for each processor. In the benchmark described subsequently, each atom has typically at most on the order of 100 neighbors. However, it is hard to predict how imbalanced the load could be before balancing is done (in contrast to the balancing for the pairlist generation).

Once the load is balanced, the non–bonded forces are computed in parallel and independently accumulated at each processor. After calculating the non–bonded forces for their assigned pair interactions, the processors participate in a global–sum operation resulting in an identical copy of the force array at each node. The time–step, which involves solving a constrained ordinary differential equation, is then executed redundantly at each processor. This rather crude approach will be refined dramatically in the next version of the code.

3. Programming in Pfortran. The key concept in the family of languages we are developing [14, 2] is a global name space for separate codes executing on different processors. They utilize expressions such as `x@m = y@n` to effect data communication among processors. This expression means that the value of `y` at processor `n` will be transferred to processor `m` and stored in the variable `x` at that processor. The load–balancing step in the previous section can be achieved quite elegantly in our parallel extension to Fortran, *Pfortran*, as follows:

```
jnb(jta:jta+jnbc-1)@(i-1) = jn(js:js+jnbc-1)@(pseg-1)
```

where the Fortran 90 array sub-section syntax (`:`) has been used. This line of Pfortran sends the appropriate segment of `JN`, the original part computed by a particular processor, to its appropriate destinations. Only the processors participating in the communication are involved, and this executes in parallel to the extent possible given the data. Thus the load balancing is implemented in an optimal way with a minimum of coding effort.

Pfortran also includes standard reduction [10] operators. These are used in three places in our parallel version of `gromos` to accumulate the non–bonded forces. Together with the distribution of the `INB` array, this yields a total of only five nonstandard lines of code in the entire MD module. Other lines of standard Fortran code had to be added as well, but the total additional lines was less than 200.

4. Results. We used as a test problem a standard benchmark for the `gromos` code, *CPA8* [15]. The problem involves ten time steps with 2382 atoms, with only one pairlist generation (and corresponding load–balancing). The results of the computation on the iPSC/860 are summarized in Figure 2, which indicates the times of the various components of the simulation as well as the total time as a function of the number of processors used. For reference, a horizontal line is drawn to indicate the time for the benchmark problem that has been reported for the Cray–2. Note that the performance of a sixteen–processor iPSC/860 exceeds the Cray–2 on this benchmark.

As Figure 2 shows, we have successfully parallelized the most computationally intensive part of gromos, computing the nonbonded forces. The plot indicates nearly perfect linear speed–up for this part of the code. Similarly, the $\mathcal{O}(N^2)$ part of the computation, computing the pair list, has similarly been parallelized nearly perfectly. No other parts of the code were parallelized, and these clearly limit efficiency using more than, say, 32 processors on this benchmark problem. Calculations were also done with the same atomic data, but with a cut–off radius of twice the size. This increases by a factor of eight the amount of computation in the nonbonded–force calculation, and the code remains efficient for a much larger number of processors. If no cut–off radius were used, this code would have a very high scaled speed–up [10] as a function of the number of atoms.

Results for the nCUBE were similar, except for effects due to the faster communication speed and lower processor performance, as shown in Figure 3.

Acknowledgements. The work of Scott was supported in part by the National Science Foundation through award number DMS-8903548. The work of Clark and McCammon was supported in part by the National Science Foundation, the Robert A. Welch Foundation, and the G. H. Hitchings Award from the Burroughs Wellcome Fund. Additional support was provided by Intel. Computer facilities for this work were provided by Oak Ridge National Laboratory and Sandia National Laboratory.

References

[1] M. P. Allen and D. J. Tildesley, **Computer Simulation of Liquids**, Oxford Univ. Press, 1987.

[2] B. Bagheri, B. Raghavachari and L. R. Scott, *Sharing variables in parallel programs*, Research Report UH/MD–79, Dept. Math., Univ. Houston, 1990.

[3] J. P. Boris, *A vectorized "near neighbors" algorithm of order N using a monotonic logical grid*, **J. Comp. Phys. 66** (1986), 1–20.

[4] J. P. Boris, *New directions in computational fluid dynamics*, **Ann. Rev. Fluid Dynamics 21** (1989), 345–385.

[5] C. L. Brooks III, M. Karplus and B. M. Pettitt, **Proteins: A Theoretical Perspective of Dynamics, Structure and Thermodynamics**, New York: Wiley, 1988.

[6] T. W. Clark and J. A. McCammon, *Parallelization of a molecular dynamics non–bonded force algorithm for mimd architecture*, **Computers & Chem. 14, No. 3,** 1990, 219–224.

[7] M. E. Davis and J. A. McCammon, *Electrostatics in biomolecular structure and dynamics*, **Chem. Rev. 90** 1990, 509-521.

[8] C. I. Draghicescu, *Efficient Algorithms for Particle Methods*, Ph. D. Thesis, Pennsylvania State University, 1991.

[9] L. Greengard and V. Rokhlin, *On the evaluation of electrostatic interactions in molecular modeling*, **Chemica Scripta 29A** (1989), 139-144.

[10] J. L. Gustafson, G. R. Montry and R. E. Benner, *Development of parallel methods for a 1024-processor hypercube*, **SIAM J. Scientific and Stat. Comput. 9, No. 4**, 1988, 609-638.

[11] S. G. Lambrakos, J. P. Boris, *et al.*, *Molecular dynamics simulation of* $(N_2)_2$ *formation using the monotonic Lagrangian grid*, **J. Chem. Phys. 90** (1989), 4473–4481.

[12] J. A. McCammon and S. Harvey, **Dynamics of Proteins and Nucleic Acids**, Cambridge Univ. Press, 1987.

[13] M. R. S. Pinches and D. J. Tildesley, *Large scale molecular dynamics on parallel computer using the link–cell algorithm*, **Mol. Simulation 6** (1991), 51–87.

[14] L. R. Scott, J. M. Boyle and B. Bagheri, *Distributed data structures for scientific computation*, **Hypercube Multiprocessors 1987**, M. T. Heath, ed., Philadelphia: SIAM, 1987, 55-66.

[15] W. F. van Gunsteren and H. J. C. Berendsen, GROMOS: GROningen MOlecular Simulation Software, Laboratory of Physical Chemistry, University of Groningen, Nijenborgh, The Netherlands.

Figure attached

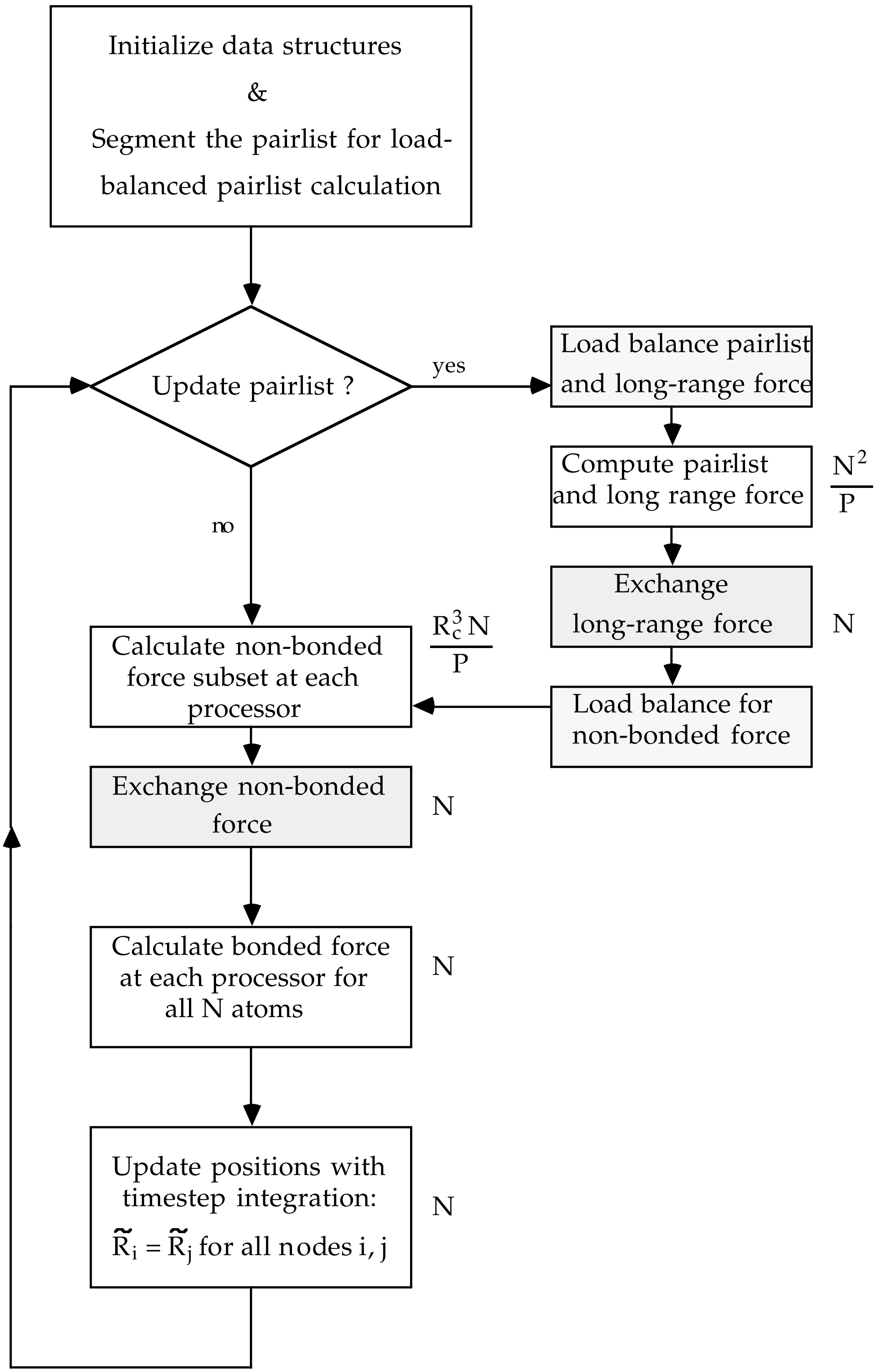
Figure 1. Flow chart for molecular dynamics showing order of magnitude time complexities for principal phases. Phases involving communication are shaded. N = number of atoms, P = number of processors and R_c = cut–off radius.

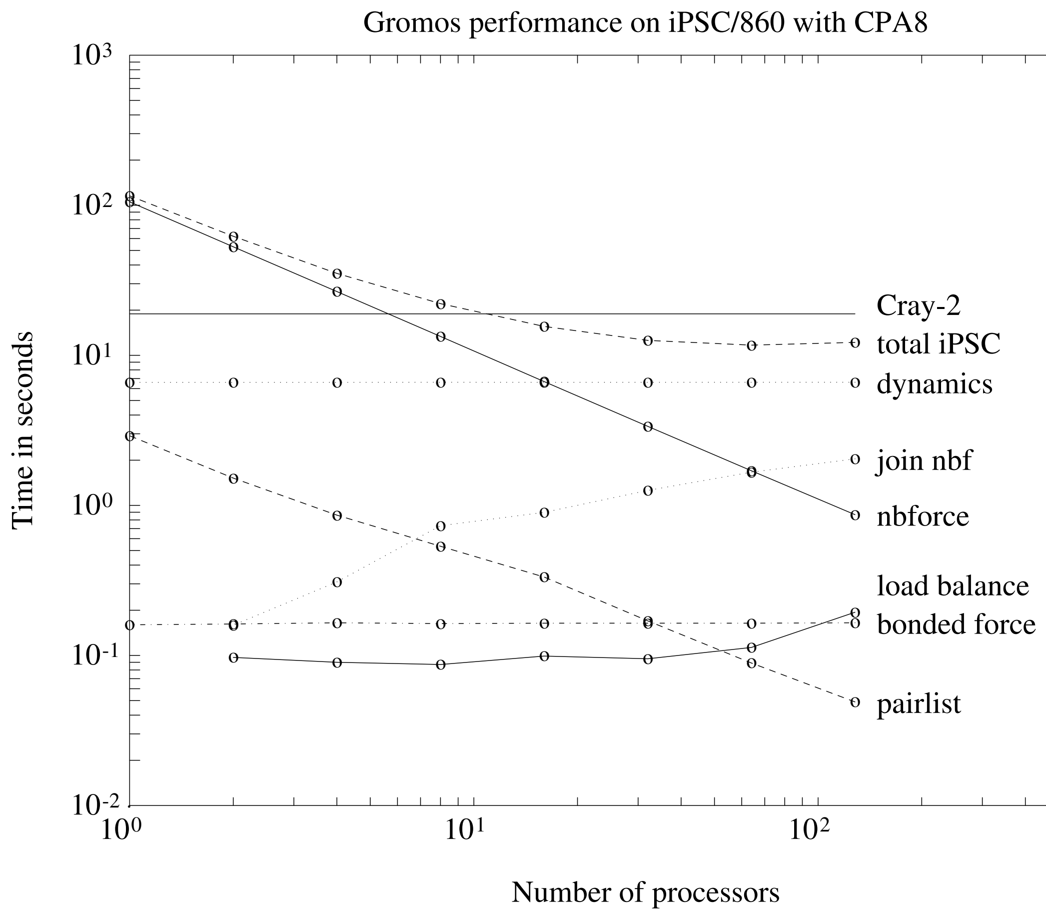
Figure attached

Figure 2. Performance of `gromos` on the Intel iPSC/860 for CPA8

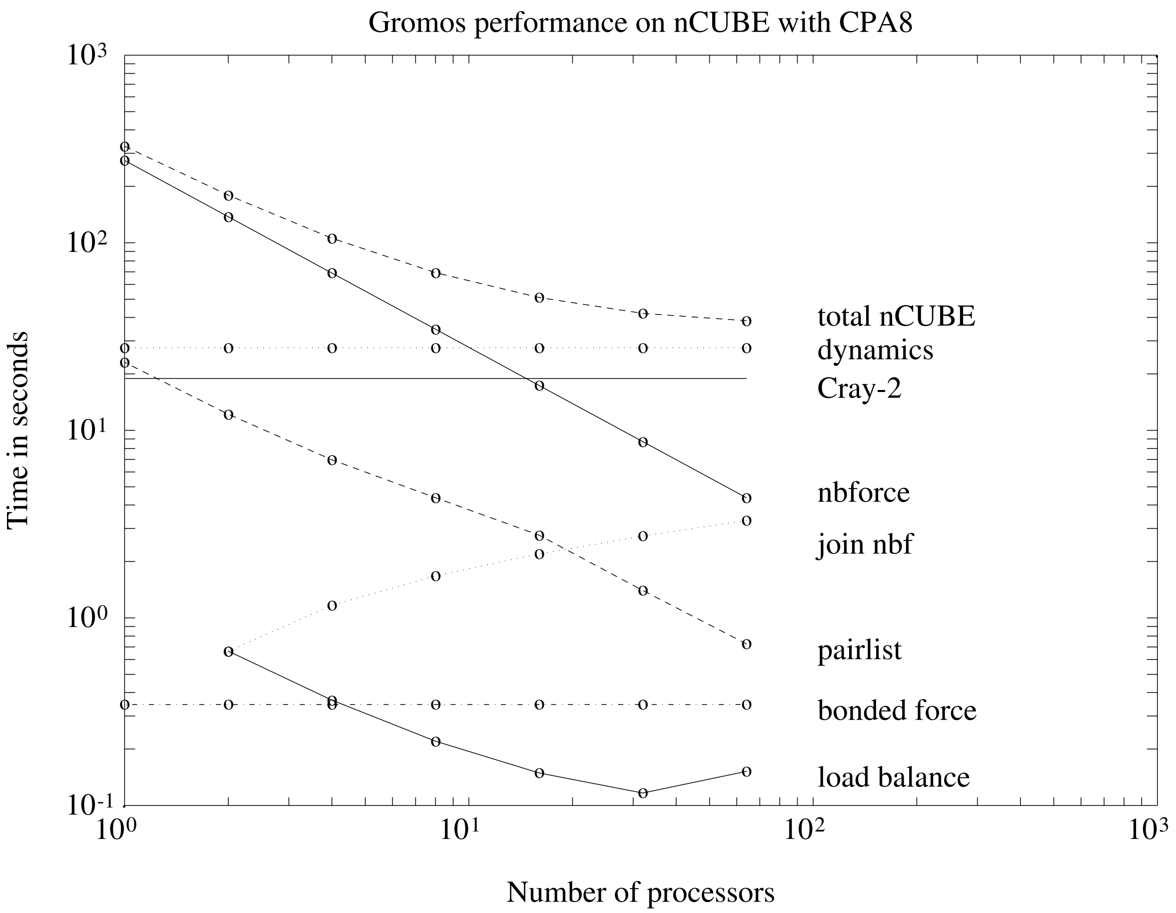
Figure attached

Figure 3. Performance of `gromos` on the nCUBE/2 for CPA8


Initialize data structures
&
Segment the pairlist for load-balanced pairlist calculation
Update pairlist ?
yes
no
Load balance pairlist and long-range force
Compute pairlist and long range force
$\frac{N^2}{P}$
Exchange long-range force
N
Load balance for non-bonded force
Calculate non-bonded force subset at each processor
$\frac{R_c^3 N}{P}$
Exchange non-bonded force
N
Calculate bonded force at each processor for all N atoms
N
Update positions with timestep integration:
$\widetilde{R}_i = \widetilde{R}_j$ for all nodes i, j
N



Gromos performance on iPSC/860 with CPA8
Time in seconds
Number of processors
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{0}
10^{1}
10^{2}
10^{3}
Cray-2
total iPSC
dynamics
join nbf
nbforce
load balance
bonded force
pairlist


Gromos performance on nCUBE with CPA8
Time in seconds
Number of processors
10^3
10^2
10^1
10^0
10^-1
10^0
10^1
10^2
10^3
total nCUBE
dynamics
Cray-2
nbforce
join nbf
pairlist
bonded force
load balance