Application of Pfortran and Co-Array Fortran in the parallelization of the GROMOS96 molecular dynamics module

Piotr Bała[a], Terry Clark[b] and L. Ridgway Scott[b]
[a]*Faculty of Mathematics and Computer Science, N. Copernicus University, Chopina 12/18, 87-100 Toruń, Poland*
Tel.: +48 56 611 3468; Fax: +48 56 622 8979;
E-mail: bala@mat.uni.torun.pl
[b]*Department of Computer Science, University of Chicago and Computation Institute, 1100 E. 58th Street, Chicago, IL 60637, USA*
E-mail: {ridg,twclark}@cs.uchicago.edu

After at least a decade of parallel tool development, parallelization of scientific applications remains a significant undertaking. Typically parallelization is a specialized activity supported only partially by the programming tool set, with the programmer involved with parallel issues in addition to sequential ones. The details of concern range from algorithm design down to low-level data movement details. The aim of parallel programming tools is to automate the latter without sacrificing performance and portability, allowing the programmer to focus on algorithm specification and development. We present our use of two similar parallelization tools, Pfortran and Cray's Co-Array Fortran, in the parallelization of the GROMOS96 molecular dynamics module. Our parallelization started from the GROMOS96 distribution's shared-memory implementation of the replicated algorithm, but used little of that existing parallel structure. Consequently, our parallelization was close to starting with the sequential version. We found the intuitive extensions to Pfortran and Co-Array Fortran helpful in the rapid parallelization of the project. We present performance figures for both the Pfortran and Co-Array Fortran parallelizations showing linear speedup within the range expected by these parallelization methods.

1. Introduction

Molecular dynamics (MD) is widely used to investigate function of biomolecular systems with large size and long time scales. Biomolecular complexes consisting of components such as proteins, lipids, DNA and RNA, and solvent are typically large in simulation terms. The explosive growth in interest in investigating inherently complex biomolecular systems such as solvated protein complexes leads to molecular systems with tens to hundreds of thousands of atoms, as for example in [39]. Parallel algorithms are critical to the application and progress of MD in order to 1) improve the accuracy of simulation models, 2) extend the length of simulations, and 3) simulate large, complex systems. Numerous MD parallelizations have been described in the literature, ranging from the easy to implement replicated algorithm [6,20] to the more difficult to implement spatial decomposition [9,30], which is generally more scalable. The force decomposition algorithm is an intermediate approach in that it is generally more efficient than the replicated algorithm and easier to implement than the spatial decomposition [27].

The ease of implementation of an MD algorithm is important given the need for multiple algorithms to address the variability encountered in mapping molecular dynamics algorithms onto parallel architectures [8, 9]. In addition, experimenting with MD algorithms on novel parallel architectures is facilitated by tools aiding the parallelization process. Various tools have been applied to molecular dynamics simulations with varying success. Data parallel approaches have been found to be problematic due to the irregularity inherent to molecular dynamics [38], which is compounded by unstructured legacy applications [7]. Low-level tools such as MPI have been successful for performance [27], but do compromise readability and consequently maintenance after the development period. Many good tools have been developed for problems structured similarly to molecular dynamics, but often target regularly structured applications, for example [12].

There remains a long way to go in expediting *the development* of robust molecular dynamics algorithms. At the moment, there are tools which we have found to fill somewhat the void. We used the tool Pfortran to implement the replicated algorithm for the GROMOS96 MD module [36], followed by a parallelization using Co-Array Fortran. Our Pfortran parallelization was completed after an aggregate of about 60 hours for a team of two over a period of one week. The effort started with an SGI parallelization based on SGI directives. The parallelization is machine independent and performs robustly.

We briefly review the MD model; the interested reader is referred to [1,21,18] for detailed treatments. The MD method provides a numerical solution of classical (Newtonian) equations of motion

$$m_i \frac{d^2 r_i}{dt^2} = F_i(r_1, r_2, \ldots, r_N) \quad (1)$$

where the force $F_i(r_1, r_2, \ldots, r_N)$ acting on particle i is defined by the interaction potential $U_i(r_1, r_2, \ldots, r_N)$. The general functional form of the potential is

$$\begin{aligned} &U(r_1, r_2, \ldots, r_N) \\ &= \sum_{\text{bonds}} K_b (r_{ij} - r_{ij}^0)^2 + \sum_{\text{angles}} K_a (\psi_{ij} - \psi_{ij}^0)^2 \\ &\quad + \sum_{\text{torsions}} K_t \left(1 + C_t cos(m_t \phi_t - \phi_t^0)\right) \quad (2) \\ &\quad + \sum_{i<j} \left(\frac{A_{ij}}{r_{ij}^6} + \frac{B_{ij}}{r_{ij}^{12}} \right) + \sum_{i<j} \left(\frac{q_i q_j}{r_{ij}} \right) \\ &\quad + U_{\text{special}} \end{aligned}$$

where r_{ij} is the distance between atoms i and j, and other constants define force field parameters for different chemical atom types. Well known algorithms such as leap-frog [34] and Verlet [33,22] are used to calculate new positions and velocities.

2. Related work

The parallelization of molecular dynamics has been explored widely in the literature [4,6,8,9,11,15,19,23, 27,29–31]. Fortunately, molecular dynamics simulations of biomolecular systems are well suited for parallel computation since the forces acting on each atom can be calculated independently with a small amount of boundary information consisting of a neighborhood of atomic coordinates and in some cases, velocities. The leading computational component of the MD calculation involves the nonbonded forces, a calculation generally quadratic in the number of atoms that can be reduced to close to a linear dependence with the cutoff radius approximation coupled with strategic use of a pairlist [16,37]. Other algorithms used to reduce the cost of evaluation of nonbonded interactions include reaction field methods [32] and multipole expansions of coulombic interactions [1,5,10,28].

The shortcomings of parallel paradigm support for molecular dynamics stems from the difficulties posed by the irregularity of the calculation [12,14,40], and a general shortage of integrated tools for parallelization. Popular parallelization libraries such as PVM [13] and MPI [24], while suitable for irregular applications, offer little abstraction, requiring the programmer to manage low-level details in the communication mechanism such as message identifiers. Higher-level methods such as HPF [17] encounter difficulties in dealing with irregular problems and legacy code [7].

3. Pfortran and Co-Array Fortran

A Fortran implementation of the Planguages, the *Pfortran* compiler extends Fortran with the Planguage operators which are designed for specifying off-process access [2,3]. In a sequential program the assignment statement specifies a move of a value at the memory location represented by j to the memory location represented by i. Planguages allow the same type of assignment, however, the memory need not be local, as in the following example in a two-process system

$$i@0 = j@1$$

stating the intention to move the value at the memory location represented by j *at process 1* to the memory location represented by i *at process 0*.

With the aid of the @ operator one can efficiently specify broadcast of the value at memory location a for logical *process 0* to the memory location a on all processes:

$$a = a@0$$

The other Pfortran operator consists of a pair of curly braces with a leading function, $f\{\}$. This operator represents in one fell swoop the common case of a reduction operation where the function f is applied to data across all processes. For example, to sum an array distributed across $nProc$ processes, with one element per process, one can write

$$sum = +\{a\}$$

Although *a* is a scalar at each process, it is logically an array across *nProc* processes. With @ and $\{\}$, a variety of operations involving off-process data can be concisely formulated.

In the Planguage model, processes interact through the same statement. Programmers have access to the local process identifier called *myProc*. With *myProc*, the programmer distributes data and computational workload. The Planguage translators transform user-supplied expressions into algorithms with generic calls to a system-dependent library using MPI, PVM, shared memory libraries or other system-specific libraries.

Cray Co-Array Fortran is the other parallelization tool considered in this study [25,26]. Co-Array Fortran introduces an additional array dimension for arrays distributed across processes. Co-Array Fortran generally requires more changes in the legacy code than does Pfortran, however, Co-Array Fortran provides automatic distribution of user-defined arrays. Co-Array Fortran does not supply intrinsic reduction-operation syntax; these algorithms must be built on point-to-point exchanges by the programmer.

4. Parallelization strategy

We noted in the introduction that the molecular dynamics parallelization methods of *domain decomposition* and the *replicated algorithm* are at the extremes in implementation difficulty, domain decomposition being the more difficult. In terms of minimizing communication, domain decomposition can be shown to be optimal for various communication topologies and switches. With the replicated model, on the other hand, the accumulation of forces is a global operation. Both algorithms scale with respect to increasing problem size while maintaining a suitable workload per process [8, 9], however, the replicated algorithm reaches a scalability limit as a function of the number of processes as a result of the global force accumulation [8]. The replicated algorithm, which was implemented in this study, performs more robustly than the spatial decomposition for a range of processor and problem configurations [9], making it the preferred method under some common conditions. In addition, the replicated algorithm is straightforward to implement.

4.1. Pairlist parallelization

Our parallelization of the replicated algorithm modified three parts of the program: 1) force calculation, 2) pairlist calculation, and 3) I/O. We consider the principal components of the overall strategy shown in

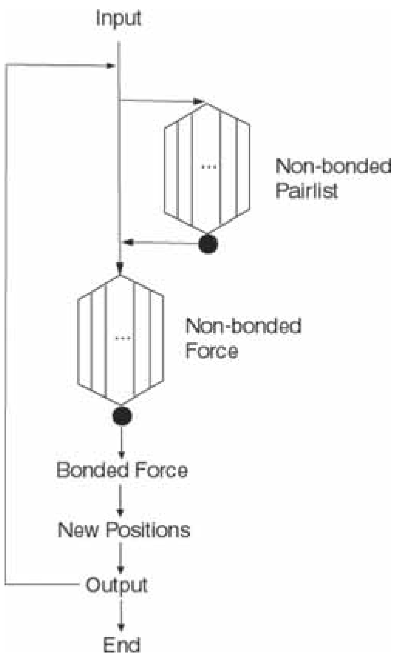


Fig. 1. Parallelization Schematic. One cycle through the flowchart constitutes a single timestep. The parallelized nonbonded pairlist and force calculations are shown as branched regions of the flowchart indicating the process-dependent control flow through that part of the program. The single-line edges between components represent portions of the program executed redundantly at each process. The pairlist is calculated at intervals (roughly once every 10 timesteps); inter-process data movement (black circle) follows to accumulate energies and to exchange pairlist data used to retain an indexing scheme compatible with the sequential code. The force calculation is performed in parallel and at every timestep; immediately following, the forces are accumulated with a reduction operation (second black circle) as described in the text. The remainder of the timestep is performed redundantly by all processes, with each process holding the same system state.

Fig. 1. In the pairlist calculation all atom pairs are scanned, and for each atom a list of atoms located within the cut-off radius is tabulated:

```
do i = 1, n
  jl = 0
  do j = i+1, n
    if ( | X(i) - X(j) | < R ) then
      jl = jl + 1
      JNBL(jl,i) = j
    endif
  enddo
  JNB(i) = jl
enddo
```

GROMOS96 interactions models non-bonded interactions collectively between charge groups, rather than atoms. This modification does not change the algorithm presented above, however its practical implementation is more complicated (see [7] for details).

Based on the pair list, the forces can be evaluated efficiently as given in this pseudo code:

```
do i = 1, n
  F(i) = 0
  do j = start(i), end(i)
    tmpforce = force (X(i), X(JNB(j)))
    F(i) = F(i) + tmpforce
    F(JNB(j)) = F(JNB(j)) - tmpforce
  enddo
enddo
```

Both the force and pairlist portions parallelization are based on a modulo strategy implementing a cyclic distribution of pairs for the nonbonded-force routines [4], where in pseudo code

```
if (MOD(chargeGroup-1,nProc).EQ.
myProc)
  perform calculations
endif.
```

Our parallelization began with the GROMOS96 distribution's SGI-specific shared-memory code where each process calculates neighbors and forces for the portion of the pairlist assigned to it by the cyclic distribution. In a distributed memory implementation is a good idea to distribute (rather than replicate) the pairlist array since it is the largest data structure in the program. Note that the cyclic distribution approximately, but effectively, distributes the load in the calculation.

4.2. *Force calculation parallelization*

At the end of the force-calculation loop the replicated algorithm leaves processes with incomplete nonbonded forces, making it necessary to accumulate the values with a global summation. This step requires significant communication and becomes the barrier to scalability with the replicated algorithm. However, the algorithm is effective over a wide range of process and problem configurations where the computation costs dominates the communication cost.

The partial results calculated at each process are stored in a local copy of the force array (`F` in Pfortran pseudo code and `F_dist` where Co-Array Fortran is used). Upon completion of the force calculation in each timestep, the partial forces are summed into the the force array at each process.

The global accumulation of the force array is expressed concisely by the Pfortran reduction operator as

```
F(1:natoms*3) = +{F(1:natoms*3)}.
```

The notation specifies that the summation operator be applied to each instance of F in the process group with the mathematical meaning

$$F_i = F_i^{(0)} + F_i^{(1)} + \cdots + F_i^{(P-1)} \qquad (3)$$

for $1 \leq i \leq$ natoms $*$ 3 and *P* processes.

Without reduction operators, the Co-Array Fortran implementation of the force accumulation can be performed through explicit point-to-point exchanges as

```
F(1:natoms) = 0.0
call sync_images()
do iproc = 0, nProc-1
  F(1:natoms*3) = F(1:natoms*3) +
  F_dist(1:natoms*3)[iproc]
enddo.
```

The co-array `F_dist` is distributed across images, the Co-Array Fortran equivalent to processes, with the image specified by the index within the square brackets. The array `F` is a usual sequential array, local to each process and therefore considered replicated. So, in the code segment above, each process accesses the co-array portion of each other process to perform the sum in Eq. 3. `sync_images` is a familiar shared–memory construct required to insure the one-sided accesses of non-local memory are consistent with the point of access in the program. In the Pfortran implementation, the consistency determination is left to the implementation of the compiler. In the current Pfortran, synchronization is achieved through message buffering.

The force calculation for covalent bonds, angles, dihedrals and torsions may be performed independently and in parallel for each component. In the present implementation this part of the program was not parallelized due to its meager contribution to the total execution time.

4.3. *I/O*

A typical I/O strategy for SPMD codes is to use a designated process to open and read data, then to communicate the data obtained from files to all other processes over a network. Similarly, non-replicated data is sent to, and then output from, a designated process. In that way, sequential semantics are retained for file I/O. In the Pfortran model, I/O from the sequential program must be modified to retain the sequential semantics. With the Cray Co-Array Fortran, however, the compiler allows for synchronous file operations; that is, the disk operations are performed by all nodes and data is read by all images. Thus with Co-Array Fortran, I/O modifications are not required in general.

A typical read operation is written using Pfortran as follows

```
if ( myProc.eq.0 ) then
  read(unit,*) temperature
endif
temperature = temperature@0
```

with the designated process broadcasting the value read. For the typical write operation, the designated process outputs the values. In the following example, partial `energy` terms are summed to the total energy for the system, and output by the designated process:

```
energy = +{energy}
if ( myProc.eq.0 ) then
  write(unit,*) energy
endif
```

More complicated "gathers" of data to the designated process may be required, however for the replicated algorithm, the resulting replication of state simplifies this step. Performed manually, the I/O modifications were the most tedious aspect of our replicated algorithm implementation using Pfortran.

4.4. *Parallelization details*

GROMOS96 is written in FORTRAN77 for which Co-Array Fortran and Pfortran are supersets. The Pfortran implementation can run on systems where Pfortran is ported, independent of the underlying communication paradigm; at present, MPI, PVM and a parallel simulator are targeted by Pfortran.[1] A port to a new communication library requires only changes in the Pfortran communication library. The Co-Array Fortran version is dependent on Cray systems, thus limiting the portability.

The roughly 40,000 lines of GROMOS molecular dynamics code required the introduction of 65 declarations of co-arrays, about 300 lines containing co-array syntax, and almost the same number of calls to the Co-Array Fortran image-synchronization procedure. With Pfortran, various reduction operations were required 33 times, with another 290 off-process data-access operations. In both the Co-Array Fortran and Pfortran parallelizations, most of modifications were associated with the I/O subroutines, a feature of the replicated algorithm and I/O in general. The source code modifications to the code were reduced with the abstractions provided by Co-Array Fortran and Pfortran, compared to an implementation using standard communication libraries such as MPI or PVM. Co-Array Fortran and especially Pfortran requires just one additional line for each point-to-point communication compared to at least several lines of code using MPI or PVM libraries. Moreover, the co-array syntax and Pfortran operators provide an intuitive notation aiding the reasoning about the program in a way not dissimilar to the "$+$" operator in sequential languages.

[1]Other Pfortran ports exist, but for machines that are no longer marketed.

5. Program performance

The performance of the parallelized GROMOS MD codes was measured using HIV-1 protease in water. The total system of 18,700 atoms consists of 1,970 protein atoms, 14 ions and 5,572 water molecules. Periodic boundary conditions were used and a nonbonded-interaction cut-off radius of 8 Å. The principal features of the three multiprocessor systems used in this study are summarized in Table 1.

We found close to linear speedup for the systems tested (Fig. 2). On the Cray T3E the program scales up to 32 processors (Fig. 3). With more processors we expect the communication costs to dominate (Figs 2 and 3) for the HIV-1 system and parameters. Note that the one-time cost of data inputting was not removed from the total time. In practice this cost will be amortized by runs longer than our short, 100-step run with the I/O costs effectively going to zero and improving prospects for scalability.

The communication costs are dominated by the reduction of the force array during each timestep. This cost depends on the algorithm and the underlying communication layer. From Fig. 4, as expected, the $O(N + log P)$ algorithm underlying Pfortran reductions outperforms the Co-Array Fortran reduction algorithm we implemented in this study, a collection of point-to-point exchanges (see Fig. 3).

6. Concluding remarks

We completed an adaptation of the replicated algorithm implemented in the GROMOS96 MD module (as an SGI-specific implementation for shared-memory) to a portable distributed-memory version in about 60 pro-

Table 1
Features of the parallel computer systems and environments

	Nodes	Processor	Network	OS
CRAY T3E	36	300MHz	4 Gbit/sec 3D torus	Unicos mk 2.0.5 MPT 1.3
SGI Power Challenge R10K	8	194 MHz	8 Gbit/sec custom interconnect	IRIX 6.2
Pentium cluster	8	300–600 MHz	0.1 Gbit/sec Ethernet	Linux 6.2 MPICH 1.1.2

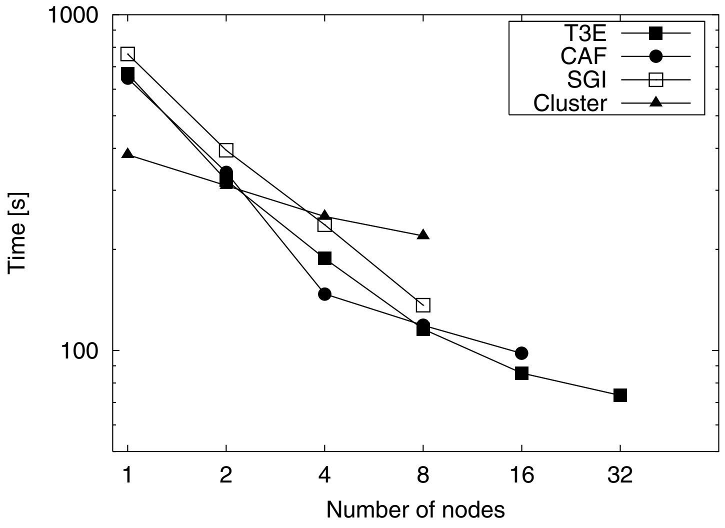


Fig. 2. Total execution time for several processor configurations on SGI, Cray T3E and cluster of workstations. “CAF” denotes results for Co-Array Fortran on the Cray T3E, whereas the Pfortran implementation using MPI is labeled “T3E”.

grammer hours. The parallel version of the code retains full functionality of the numerous options in the GROMOS96 MD module. The adaptation of the SGI-specific code was tantamount to parallelizing from the sequential version given the few changes made to the sequential coded to arrive at the SGI-specific shared-memory. The abstractions provided by the Planguages allowed us to focus our attention on details of logic unhampered by message-passing details, such as providing a message-passing API with message sizes and tags, proper communicator and so on. What is more, the maintainability of the Pfortran code after parallel development is improved over an MPI version and the source code is portable to systems where the Pfortran compiler has been ported.

We next adapted the Pfortran code to Co-Array Fortran, finding both implementations to scale with number of processors as expected using the replicated algorithm on a constant-size problem. Pfortran has performance comparable to Co-Array Fortran, without the portability limitations. Pfortran also provides built in reduction operations and the facility for user-defined ones. On the T3E, the $O(N + logP)$ algorithm underlying Pfortran reductions outperforms the Co-Array Fortran reduction algorithm that we implemented as a collection of point-to-point exchanges. Development of efficient Co-Array Fortran reductions is left to the programmer. Having a library of such routines would be very useful.

The small number of extensions and intuitive application of Pfortran and Co-Array Fortran contribute to making them effective tools for developing explicitly-parallel scientific applications. Key advantages in using both approaches are 1) code with performance close to if not at the underlying libraries, 2) machine independence, 3) compiler optimizations, and 4) improved code readability. The abstractions allowing the programmer to manipulate off-process data accesses fa-

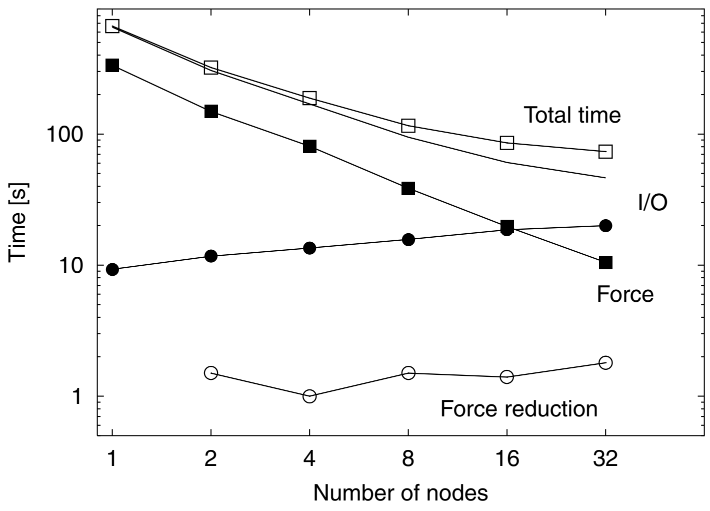


Fig. 3. Execution time for different sections of the code for as a function of the number of Cray T3E computing nodes.

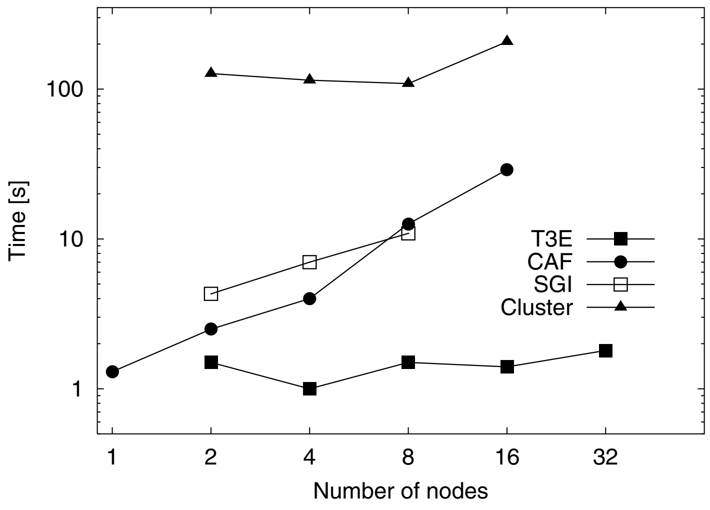


Fig. 4. Communication time for the reduction of the long-range force array for different numbers of computing nodes. CAF denotes results for Co-Array fortran on Cray T3E.

cilitate one's ability to reason about a code. That the off-process data access syntax are part of the language also permits compiler optimizations, which would be difficult to perform with an API such as that provided by MPI (unpublished work in progress). Our use of Co-Array Fortran does not take advantage of its shared-memory paradigm, but it does highlight the advantage of the abstractions provided by Co-Array Fortran and Pfortran. The parallel programming models provided by the two paradigms have orthogonal and potentially complementary aspects. We are exploring the use of the paradigms together in a programming language.

Acknowledgements

Piotr Bala was supported by the Polish State Committee for Scientific Research. Terry Clark and L. Ridgway Scott were supported by NSF with the National Partnership for Advanced Computational Infrastructure, NPACI. The computations were performed at the ICM, Warsaw University and the High Performance Computing Laboratory at the University of Chicago.

References

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

[2] B. Bagheri, *Parallel Programming with Guarded Objects*, Pennsylvania State University, Department of Computer Science, PhD dissertation, 1994.

[3] B. Bagheri, T. Clark and L.R. Scott, Pfortran: a parallel dialect of Fortran, *ACM Fortran Forum* **11** (1992), 20–31.

[4] B.R. Brooks and M. Hodoscek, Parallelization of CHARMM for MIMD machines. *Chemical Design Automation News* **7** (1992), 16–22.

[5] T.C. Bishop, R.D. Skeel and K. Schulten, Difficulties with multiple time stepping and fast multipole algorithm in molecular dynamics. *Journal of Computational Chemistry* **18**(14) (1997), 1785–1791.

[6] T. Clark and J.A. McCammon, Parallelization of a molecular dynamics non-bonded force algorithm for MIMD architectures, *Computers & Chemistry* **14** (1990), 219–224.

[7] T. Clark, R. von Hanxleden and K. Kennedy, Experiences in data-parallel programming. *Journal of Scientific Programming* **6** (1997), 153–138.

[8] T. Clark, R. von Hanxleden, K. Kennedy, C. Koelbel and L.R. Scott, Evaluating parallel languages for molecular dynamics computations. in: *Scalable High Performance Computing Conference*, IEEE Comput. Soc. Press, 1992, pp. 98–105.

[9] T. Clark, R. von Hanxleden, J.A. McCammon and L.R. Scott, Parallelizing molecular dynamics using spatial decomposition. in: *Scalable High Performance Computing Conference*, IEEE, Knoxville, 1994, pp. 95–102.

[10] H.-Q. Ding, N. Karasawa and W.A. Goddard III, Atomic level simulations on a million particles: The cell multipole method for Coulomb and London nonbond interactions. *Journal of Computational Physics* **97** (1992), 4309–4315.

[11] D. Fincham, Parallel computers and molecular simulation. *Molecular Simulation* **1** (1987), 1–45.

[12] S.J.Fink, S.R. Kohn and S.B. Baden, Efficient run-time support for irregular block-structured applications, *J. Parallel and Distributed Computing* **50** (1998), 61–82.

[13] G.A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, *PVM: Parallel Virtual Machine*, The MIT Press, 1994.

[14] R. von Hanxleden, K. Kennedy, C. Koelbel, R. Das and J. Saltz, Compiler analysis for irregular problems in Fortran D, in: *Proceedings of the Fifth Workshop on Languages and Compilers for Parallel Computing*, New Haven, 1992.

[15] H. Heller, H. Grubmuller and K. Schulten, Molecular dynamics simulation on a parallel computer. *Molecular Simulation* **5** (1990), 133–1650.

[16] J.W. Eastwood R.W. Hockney, *Computer Simulation Using Particles*, Cambridge University Press, Cambridge, 1987.

[17] *Proceedings of the High Performance Fortran Forum*, Houston, 1992.

[18] R.W. Hockney and J.W. Eastwood, *Computer simulation using particles*, Institute of Physics Publishing, Bristol and Philadelphia, 1994.

[19] L.V. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan and K. Schulten. NAMD2: Greater scalability for parallel molecular dynamics, *Journal of Computational Physics* **151** (1999), 283–312.

[20] S.L. Lin, J. Mellor-Crummey, B.M. Pettitt and G.N. Jr. Phillips, Molecular dynamics on a distributed-memory multiprocessor, *Journal of Computational Chemistry* **13**(8) (1992), 1022–1035.

[21] J.A. McCammon and S.C. Harvey, *Dynamics of proteins and nucleic acids*, Cambridge University Press, Cambridge, 1987.

[22] J.A. McCammon, B.M. Pettitt and L.R. Scott, Ordinary differential equations of molecular dynamics. *Computers Math. Applications* **28**(10–12) (1994), 319–326.

[23] F. Müller-Plathe and D. Brown, Multi-colour algorithms in molecular simulation: Vectorisation and parallelisation of internal forces and constraints, *Computer Physics Communications* **64** (1991), 7–14.

[24] Message Passing Interface Forum, MPI: a message-passing interface standard. Technical report, 1994.

[25] R.W. Numrich, F^{--}: a parallel extension to Cray Fortran, *Scientific Programming* **6** (1997), 275–284.

[26] R.W. Numrich, J. Reid and K. Kim, Writing a multigrid solver using co-array Fortran, in: *Recent Advances in Applied Parallel Computing, Lecture Notes in Computer Science 1541*, B. Kågström, J. Dongarra, E. Elmroth and J. Waśniewski, eds, Springer-Verlag Berlin, 1998, pp. 390–399.

[27] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics* **117** (1995), 1–19.

[28] J. Singer, The Parallel Fast Multipole Method in Molecular Dynamics. PhD thesis, University of Houston, August 1995.

[29] W. Smith, Molecular dynamics on hypercube parallel computers. *Computer Physics Communications* **62** (1991), 229–248.

[30] T.P. Straatsma, M. Philippopoulos and J.A. McCammon, NWChem: Exploiting parallelism in molecular simulations. *Computer Physics Communications* **128** (2000), 377–385.

[31] V.E. Taylor, R.L. Stevens and K.E. Arnold, Parallel molecular dynamics: Implications for massively parallel machines. *Journal of Parallel and Distributed Computing* **45** (1997), 166–175.

[32] I.G. Tironi, R. Sperb, P.E. Smith and W.F. Gunsteren, A generalized reaction field method for molecular dynamics simulations, *Journal of Chemical Physics* **102** (1995), 5451–5459.

[33] W.F. van Gunsteren and H.J.C. Berendsen, Algorithms for Brownian dynamics, *Molecular Physics* **45** (1982), 637–647.

[34] W.F. van Gunsteren and H.J.C. Berendsen, A leap-frog algorithm for stochastic dynamics. *Molecular Simulation* **1** (1988), 173–182.

[35] W.F. van Gunsteren, H.J.C. Berendsen, F. Colonna, D. Perahia, J.P. Hollenberg and D. Lellouch, On searching neighbors in computer simulations of macromolecular systems, *Journal of Computational Chemistry* **5** (1984), 272–279.

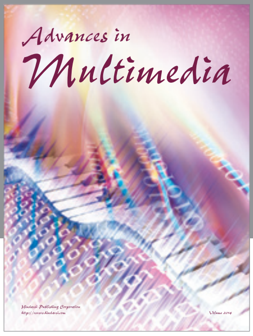
[36] W.F. van Gunsteren, S.R. Billeter, A.A. Eising, P.H. Hunenberger, P. Kruger, A.E. Mark, W.R.P. Scott and I.G. Tironi, *Biomolecular Simulation: The GROMOS96 Manual and User Guide*, ETH Zurich and BIOMOS b.v., Zurich, Groningen, 1996.

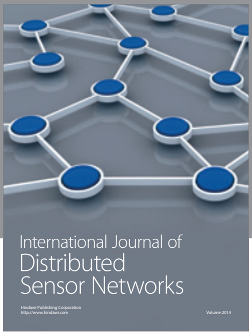
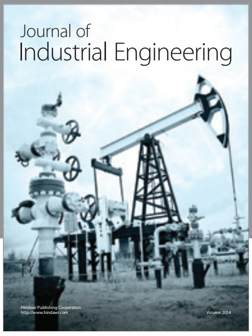
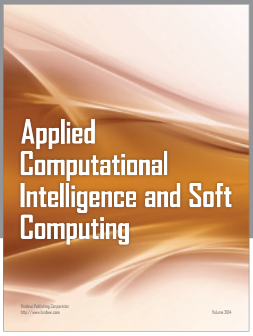
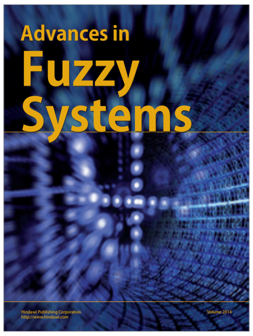
[37] L. Verlet, Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, *Physical Review* **159** (1967), 98.

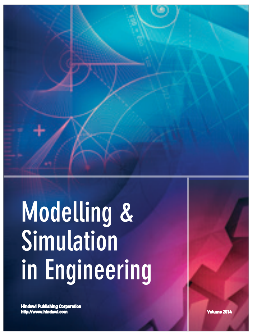
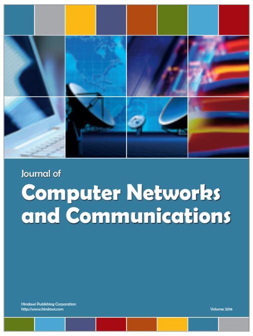
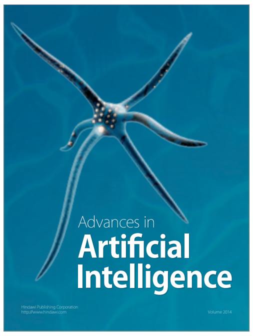
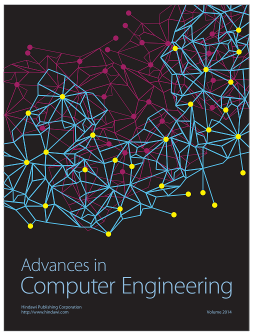
[38] R. von Hanxleden, *Compiler Support for Machine-Independent Parallelization of Irregular Problems*, PhD thesis, Rice University, 1994.

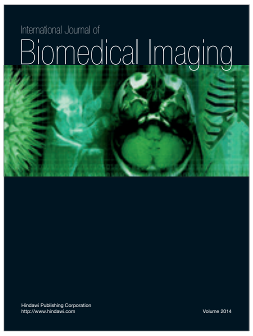
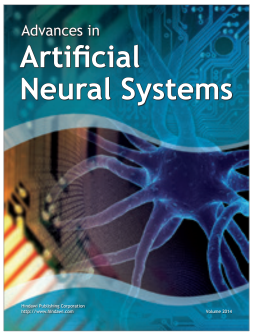
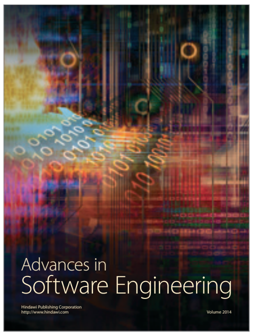
[39] S.T. Wlodek, T. Clark, L.R. Scott and J.A. McCammon, Molecular dynamics of acetylcholinesterase dimer complexed with tacrine, *Journal of the American Chemical Society* **119** (1997), 9513–9522.

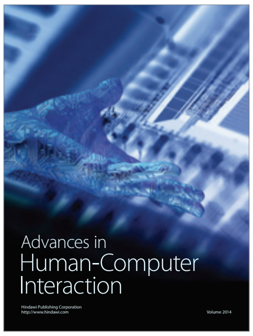
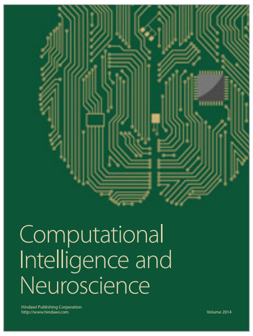
[40] Z. Zhang and J. Torrellas, Speeding up irregular applications in shared-memory multiprocessors: Memory binding and group prefetching, *Comp. Arch. News* **23** (1995), 188–199.


Advances in
Multimedia

The Scientific World Journal

International Journal of
Distributed Sensor Networks

Journal of
Industrial Engineering

Applied Computational Intelligence and Soft Computing

Advances in
Fuzzy Systems


Modelling & Simulation in Engineering

Journal of
Computer Networks and Communications

Advances in
Artificial Intelligence

Advances in
Computer Engineering

International Journal of
Computer Games Technology

International Journal of
Biomedical Imaging

Advances in
Artificial Neural Systems

Advances in
Software Engineering

Journal of
Robotics

Advances in
Human-Computer Interaction

Computational Intelligence and Neuroscience

International Journal of
Reconfigurable Computing

Journal of
Electrical and Computer Engineering