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

Scientific Programming 9 (2001) 61–68
ISSN 1058-9244 / $8.00 © 2001, IOS Press. All rights reserved

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

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

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

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

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

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

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

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

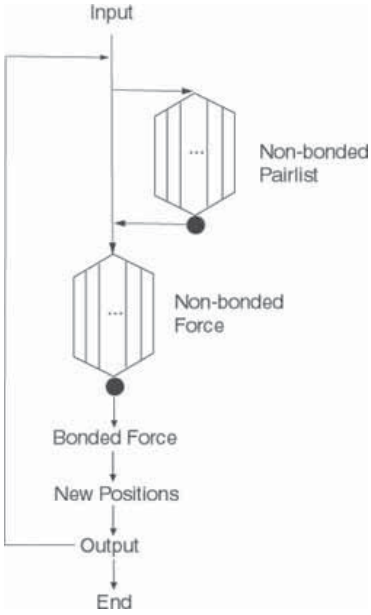

Fig. 1. Parallelization Schematic. One cycle through the flowchart constitutes a single timestep. The parallelized nonbonded pairlist and force calculations are shown as branched regions of the flowchart indicating the process-dependent control flow through that part of the program. The single-line edges 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

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

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

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

for $1 \leq i \leq \text{natoms} * 3$ 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

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

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 $O(N+logP)$ 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-

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.