1996

Exploiting Symmetry in Parallel Computations for Structural Biology

Ioana Maria Martin

Dan C. Marinescu

Report Number:
96-017

EXPLOITING SYMMETRY IN PARALLEL COMPUTATIONS FOR STRUCTURAL BIOLOGY

Ioana M. Boier Martin
Dan C. Marinescu

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR-96-017
February 1996

Exploiting Symmetry in Parallel Computations for Structural Biology

Ioana M. Boier Martin and Dan C. Marinescu
(boier,dcm)@cs.purdue.edu
Computer Sciences Department
Purdue University
West Lafayette, IN 47907

February 12, 1996

Abstract

In this paper we outline a technique to carry out parallel computations for objects which exhibit a high degree of symmetry. We describe an algorithm used for the computation of the envelope of a spherical virus with icosahedral symmetry and discuss its performance. The algorithm is quite general and is based on a very simple idea: carry out the required computations only for the unit repeating itself throughout the data space, then propagate the results by folding back the points throughout the space to the unit that has been updated. The algorithm is advantageous if folding back is computationally less intensive than the original computation.

Contents

1 Introduction

The 3-D atomic structure determination of viruses is a computationally intensive area of research which requires the fusion of biology and high performance computing. Parallel computers are needed for the data acquisition, data analysis, and the modeling phase of structure determination of large macromolecules like viruses.

The process of structure determination begins with a data collection phase, when suitably prepared biological specimens are either irradiated with X-rays and investigated using crystallographic methods, or studied using electron microscopy techniques. X-ray crystallography is able to provide high resolution information to about 2-3Å, while electron microscopy currently provides structural information at lower resolution, 20Å or so. X-ray crystallography and electron microscopy are increasingly taking advantage of on-line data acquisition techniques using CCD detectors. Typically a CCD detector produces one frame every few seconds, each frame consists of 1–16 Mpixels with 16–24 bits per pixel. We envision the use of parallel computers for processing of images containing structural information.

An analysis phase follows the data acquisition in case of X-ray crystallography and electron microscopy. From X-ray diffraction images, one obtains information in the reciprocal space, the structure factors defined as the Fourier transform of the electron density. As a first step in structure determination, the amplitudes of the structure factors and the Miller indices are calculated from the X-ray diffraction images. This process is called indexing. Several methods are used to determine the phase of the structure factors, lost in the process of recording X-ray images. One of these methods, called Molecular Replacement [11], is discussed in some depth in the next section. In case of electron microscopy, one obtains 3-D models of the virus through a particle reconstruction process. The phase refinement and extension used in X-ray crystallography and the image reconstruction process used in electron microscopy currently run on parallel computers [6], [7].

While the structure determination of proteins which contain tens to hundreds of residues is common today, the determination of the structure of viruses is still very challenging. Viruses are fairly large proteins, typically they contain tens of thousands of residues and have millions of atoms. Spherical viruses are of special interest. They consist of a core containing the nucleic acid genome, surrounded by a protein shell called capsid, and some viruses have also a lipid bilayer membrane, an envelope enclosing the capsid. The nucleic acid genome cannot provide the blueprint for such a big protein molecule so the protein shell of viruses is built from many identical copies of one or a few polypeptid chains. This reflects the principle of genetic economy [1] which states that the capsid is built from many copies of a few kinds of sub-units.

One can make a sphere by arranging identical objects symmetrically on its surface, either building an icosahedron or a dodecahedron. The icosahedron is built from 20 identical equilateral triangles and has 12 vertices, each with a fivefold axis, 20 faces each with a threefold axis, and 30 edges each with a twofold axis, Figure 1.

An icosahedal virus can be divided into a number of identical units related by symmetry. Each unit is an asymmetric object consisting of a polypeptide chain. A symmetry axis cannot pass through an asymmetric object. The minimum number of identical sub-units that can form a virus with icosahedral symmetry is equal to the number of tiles, times the number of objects to fill in a tile. We have 20 triangular tiles, each one with threefold symmetry, therefore three objects are required to form a tile and the minimum number of asymmetric units is 60. If a virus has 60 identical sub-units, only one gene is necessary to code the entire protein shell.

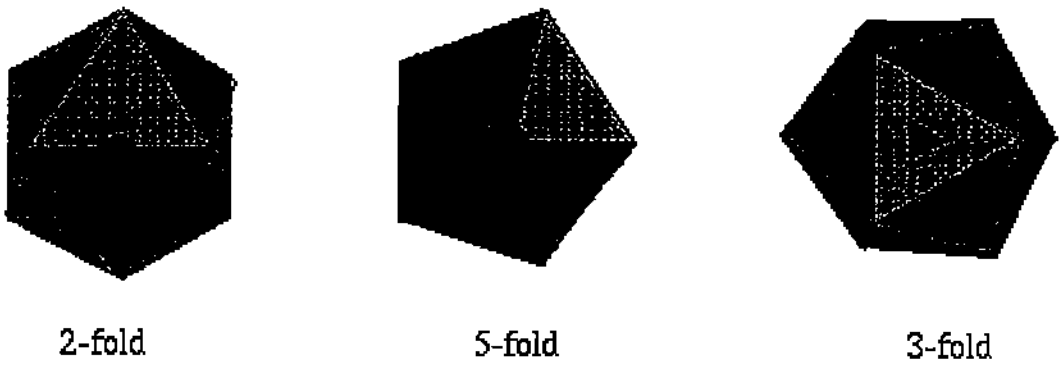

Figure 1: The 2,3 and 5-fold symmetry axis of an icosahedran.

In general, a virus may have more than one type of sub-unit. Caspar & Klug [4] have shown that the number of identical sub-units, T is given by $T = h^2 + hl + l^2$ with h and l integers, so valid values for T are $1, 3, 4, 7, 9, 12, 13, ...$ The symmetry of an icosahedron extends throughout its volume, therefore an asymmetric unit is a wedge from the surface to the center of the icosahedron.

In this paper we investigate means to exploit this symmetry to reduce the amount of computations required for the structure determination of spherical viruses. We discuss parallel algorithms used in structure determination and examine an idea of Michael Rossmann [12] to exploit the symmetry of spherical viruses for the computations needed for phase refinement and extension.

2 Electron Density Averaging for Computing the Molecular Envelope of a Virus

In 1963 Michael Rossmann and David Blow suggested the Molecular Replacement Method for determining the phase of the structure factors in X-ray crystallography [11]. One starts with (a) a low resolution model of a virus, e.g., hollow spheres and (b) the measured amplitudes of the structure factors. From the low resolution model of the virus one can compute the amplitudes and phases of the structure factors. Then the measured structure factors and the calculated phases are used to obtain a better electron density map. The symmetry of the virus is used to replace the electron density at every point with an averaged one among all points related by non-crystallographic symmetry.

Recall that once the amplitudes and the phases of the structure factors are determined, a back Fourier transform produces the electron density. If the structure factors are determined at a sufficiently high resolution, then the electron density map allows building the atomic model of the virus.

The phase refinement and extension is an iterative process consisting of electron density averaging, transformation to the reciprocal space, replacement of the amplitudes of the calculated structure factors with the measured ones, back-transformation to the real space followed by averaging. The process stops when there is no longer an improvement of the phases of the structure

factors. The stopping criteria is given by the correlation coefficient of phases obtained in two successive iterations of the phase refinement method. Therefore, the total time required by the phase refinement process at a given resolution is $t_R = I \times t_I$ with I the number of iterations and t_I the time per iteration. The time per iteration t_I is dominated by the time required for averaging which represents 85–95% of t_I. Therefore to reduce t_R, one can either attempt to reduce the number of iterations I or to reduce the time for the averaging of the electron density points in the protein shell, and thus t_I.

The algorithms for envelope computation are described in detail in [11] and [13]. Given a 3-D lattice with $N = n_x \times n_y \times n$ grid points and assuming that (a) a fraction η of all the grid points are within the protein shell, (b) the virus consists of $C = T \times 60$ sub-units, and (c) one uses an 8-point interpolation to compute the electron density of a point with non-integer grid coordinates, than the total number of operations needed is $\alpha \times \eta \times N \times C$ with α a constant. Here N can be as large as 10^9, η is typically close to 0.5 and C can be 60, 180, 240, etc.

The parallel version of this algorithm suitable for MIMD supercomputers is described in [6] and [7]. The basic idea of the algorithm is to divide the entire data space into Data Allocation Units, DAUs [5], 3-dimensional volumes each containing B grid points, and to allocate them to the PEs to ensure load balancing and to minimize the number of DAU faults. A DAU fault occurs whenever a DAU needed for computation by PE_i is not available locally and has to be fetched from the shared virtual memory backing storage. The results reported in [5] show that when the shared virtual memory is implemented using either fat nodes as data servers, or distributing the DAUs over the static memory of the PEs, then the algorithm is scalable and we obtain very good speedups, up to 512 nodes. The experiments were carried out on a Paragon system.

In this paper we discuss an algorithm for averaging suggested by Michael Rossmann [12]. The basic idea of the algorithm is straightforward and can be applied to other problems exhibiting a high degree of symmetry: carry out the required computations only for the unit repeating itself throughout the data space, then propagate the results to the entire data space by folding back the points throughout the space to the unit that has been updated. This algorithm is advantageous if folding back requires fewer operations than the original computation.

In case of electron density averaging, the original computation for every grid point requires $C = T \times 60$ linear transformations, each one of them followed by an 8-point interpolation and averaging of the C numbers. Folding back requires a linear transformation followed by an 8-point interpolation. Clearly folding back is C times less expensive than averaging.

If we call the new algorithm, the double interpolation, its execution time t_d is given by

$$\begin{aligned} t_d &= k\frac{N \times \eta}{C}(C) + k'\frac{N \times \eta}{C}(C-1) = \\ &= \frac{N \times \eta}{C}(C(k+k') - k') \simeq N \times \eta(k+k') \end{aligned}$$

The speedup of the double interpolation compared with the original algorithm, the single interpolation is

$$S = \frac{t_s}{t_d} = \frac{k \times N \times \eta \times C}{N \times \eta(k+k')} = \frac{k}{k+k'}C.$$

When $k' \simeq k$, $S \simeq \frac{C}{2}$. Yet what we would like to minimize is the total time for all the iterations at a given resolution. The double interpolation is better iff:

$$t_d \times I_d < t_s \times I_s.$$

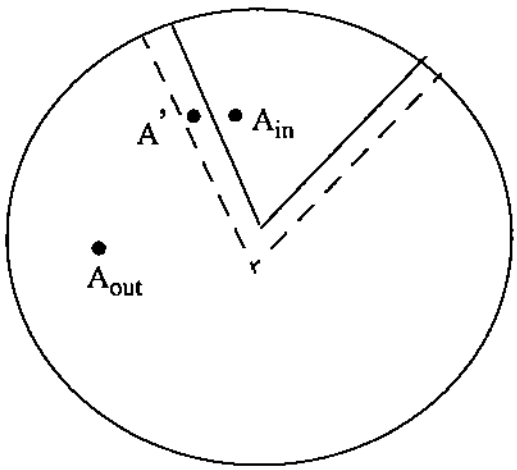

Figure 2: The need to carry out the averaging process for the border region outside the asymmetric unit.

This happens when

$$\frac{I_d}{I_s} < S.$$

We do expect that the double interpolation algorithm converges slower than the single interpolation, $I_d > I_s$ with I_d, I_s the number of iterations for the double and respectively single interpolation. For a sequential algorithm, the double interpolation is better than the single interpolation, but this is not necessarily true for a parallel implementation of the algorithm.

3 Implementation and Results

We report here the results obtained for a parallel implementation of the double interpolation algorithm described in the previous section. First, we observe that one needs to carry out averaging for a region slightly larger than the asymmetric unit. Figure 2 shows that folding back a grid point A_{out} outside the asymmetric unit may map into a point A_{in} inside the asymmetric unit. To determine the electron density for A_{in} may require the knowledge of the electron density of a grid point A' outside the asymmetric unit, but inside the border region.

With this observation the algorithm for the double interpolation consists of three phases:

Phase 1. Identify all protein points inside the asymmetric unit and the border region.
Phase 2. Carry out the averaging for all the grid points marked as being inside.
Phase 3. Fold back all grid points marked as being outside.

The double interpolation algorithm outlined above is expected to take slightly longer than the analysis carried out in the previous section, because of the algorithmic overhead for identifying all protein within the asymmetric unit and the border region. However, this is an iterative computation and Phase 1 of the algorithm only be carried out once, during the first iteration. The information about the membership of every grid point may be stored in a suitable data structure and reused for subsequent iterations at the same resolution.

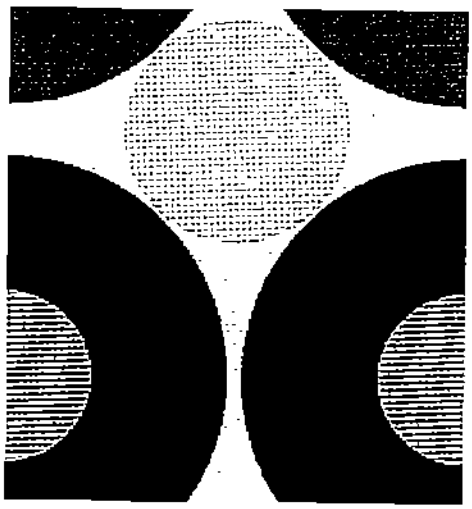

Figure 3: The two particles in one asymmetric unit of HRV16.

To parallelize this algorithm, we use the same strategy of dividing the entire data space into DAUs and carry out the computation required at each phase for all the DAUs assigned to a given Processing Element, PE. Unfortunately, there are two significant sources of inefficiencies for the parallel double interpolation algorithm. First, each phase of the algorithm has to be preceded by a load balancing computation and data has to be redistributed accordingly. At the beginning of the first phase, the DAUs are distributed with the objective that the total number of protein points assigned to any given PE, is about $\frac{N \times \eta}{P}$ where P is the number of PEs. In the second phase the objective of the load distribution is to ensure that every PE gets about $\frac{N \times \eta}{C \times P}$ grid points in the asymmetric unit and, in addition, the total number of DAU faults during averaging is minimized. To minimize the number of DAU faults, a certain PE must process the DAUs in the order which maximizes the intersection of their working sets. For the third phase, each PE is assigned DAUs such that the number of grid points outside the asymmetric unit and the border region is approximately the same and equal to $\frac{N \times \eta}{CP}(C-1)$.

The second source of inefficiency is the global synchronization required before the beginning of phases 2 and 3. Recall that the single interpolation algorithm is embarrassingly parallel, each node is assigned only once the workload and no synchronization is necessary.

To compare the single and double interpolation procedures, we used data for the HRV16, the Human Rhinovirus 16. The unit cell for HRV16 has the following dimensions: $a = 362.49$ Å, $b = 347.11$ Å $c = 334.75$ Å, $\alpha = \beta = \gamma = 90°$. The number of grid points are $nx = 360$, $ny = 352$ and $nz = 336$. Figure 3 shows the masks identifying the two particles in a unit cell, and indicates that the averaging process is done at a low resolution and the viral particles are based upon a hollow shell model. We see the capsid, the protein surrounding the nucleic acid, and surrounded in turn by the solvent. One of the viral particles is split among neighboring unit cells. Figure 4 shows the electron density map for the input data. We present contour levels for the electron density values 100, 150 and 200.

Figure 4: Electron density contour maps at level 100, 150 and 200 for the HRV16 virus.

The execution time for several cycles of averaging for the single and the double interpolation method are summarized in Table 1. The results obtained on a 64 nodes partition of a Paragon system are quite disappointing and indicate the single interpolation takes less time than the double interpolation.

As suggested earlier, the double interpolation is expected to converge slower than the single interpolation and this is confirmed by the results in Figure 5.

	Double Interpolation				Single Interpolation
Cycle	Phase 1	Phase 2	Phase 3	Total	
1	78	37	131	248	132
2	74	39	119	232	127
3	72	39	116	227	127
4	109	41	136	286	130
5	86	40	130	256	131

Table 1. The execution time (seconds) of several cycles of electron density averaging for the single and the double interpolation algorithms.

At the present time, we are implementing an optimized version of the double interpolation algorithm. In this version the first phase of the algorithm and the load balancing procedures for phase two and three are carried out only during the first iteration and their results are reused during subsequent iterations. Even in this case it will be very difficult for the double interpolation to compete with the single interpolation simply because of its considerably slower convergence rate. In addition to the global synchronization required at the end of phase 2, an inherent problem of the parallel double interpolation algorithm is that the size of the problem to be solved in phase 2 is considerably smaller than that of phase 3. In other words, the degree of parallelism and consequently, the optimal number of PEs for phase 2 and 3 are different $P_2 << P_3$.

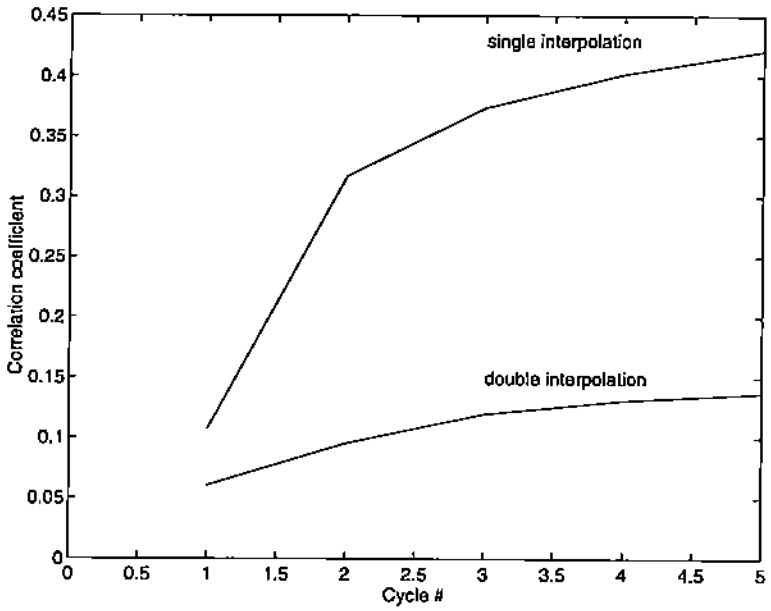

Figure 5: The convergence rate of the single and the double interpolation method. The convergence rate is determined by the correlation coefficient of the phases of the structure factors calculated during two consecutive iterations.

4 Applications to Molecular Dynamics Computations

The algorithm described in the previous section is quite general, and the more intensive the computations carried out during phase 2 are, the more interesting such a folding algorithm is. In this section we discuss the application of the folding technique to molecular dynamics computations [10]. Such calculations allow biochemists to model large molecular assemblies at atomic level.

Molecular dynamics packages like CHARMM [2], [3], [9] solve N-body problems by computing the energy and the forces affecting every atom in the assembly. The atomic interactions are summed up for all pairs of atoms within a given distance. Figure 6 illustrates the pseudo-code for a folding algorithm for molecular dynamics calculations for an assembly with icosahedral symmetry.

To parallelize the algorithm, each PE is assigned a number of atoms inside the asymmetric unit and the border region, but ensuring load balance is a rather delicate matter [9]. As before, global synchronization is required before propagating the results to the entire structure. To minimize communication among nodes we duplicate the computations involving the (i, j) pair whenever atoms i and j are originally assigned to different PEs.

```
identify atoms inside the asymmetric unit and the border

for every atom inside identify all atoms at distance at most d
and construct list of pairs

     do until convergence
          for timestep = first, last
             for every atom inside
                for every pair including the atom
                   compute all forces
                endfor
                sum up all forces
                update the coordinates of the atom
             endfor
          endfor
          propagate the results to all points outside
          update list of pairs
     enddo
endfor
```

Figure 6: Pseudocode for a folding molecular dynamics algorithm for a spherical virus.

5 Conclusions

To carry out computations for objects which exhibit a high degree of symmetry one can first transform the unit cell which repeats itself throughout the object and then propagate the results by symmetry. An algorithm based upon these ideas requires (a) to identify the unit cell, (b) to carry out the transformations within the unit cell, and (c) to map every point in the data space back into the unit cell. The more complex the transformations required in step (b) compared with folding back required by step (c) are, the more advantageous the algorithm is. Unfortunately, parallel algorithms based upon the techniques outlined above require synchronization and load balancing at the beginning of steps (b) and (c) above. Also, the size of the problem solved in step (b) is considerably smaller than that for phase (c).

We do expect that applications of this technique to molecular dynamics calculation will show its advantages.

Acknowledgments

The authors express their gratitude to Michael Rossmann who has provided insight into structural biology computations. This research is supported in part by National Science Foundation grants BIR-9301210 and MCR-9527131, by the Scalable I/O Initiative and by a grant from the Intel Corporation.

6 Literature

[1] C. Branden and J. Toose, Introduction to Protein Structures, *Garland Publishing*, 1991.

[2] B.R. Brooks, R.E. Cruccoleri, B.D. Olafson, K.J. States, S. Swaminathan and M. Karplus, *J. Comp. Chem.* vol 4, p 187, 1983.

[3] B.R. Brooks and M. Hodoscek, Parallelization of CHARMM for MIMD Machines, *Chemical Design and Automation News* vol 7, p. 16, 1992.

[4] Caspar, D.L.D. and Klug, A., Physical Principles in the Construction of Regular Viruses, *Cold Spring Harbor Symp. Quant. Biol.* 27, pp. 1–24, 1962.

[5] M.C. Cornea-Hasegan, D.C. Marinescu and Z. Zhang, Data Management for a Class of Iterative Computations on Distributed-Memory MIMD Systems, *Concurrency: Practice and Experience* vol **6(3)**, pp. 205–229, May 1994.

[6] M.C. Cornea-Hasegan, Z. Zhang, R.E. Lynch, D.C. Marinescu, A. Hadfield, J.K. Muckelbauer, S. Munshi, L. Tong and M.G. Rossmann, Phase Refinement and Extension by Means of Non-crystallographic Symmetry Averaging using Parallel Computers, *Acta Cryst.*, **D51**, pp. 749–759, 1995.

[7] D.C. Marinescu, J.R. Rice, M.C. Cornea-Hasegan, R.E. Lynch and M.G. Rossmann, Macromolecular Electron Density Averaging on Distributed Memory MIMD Systems, *Concurrency: Practice and Experience* vol **5(8)**, pp. 635–657, December 1993.

[8] I.M. Martin and D.C. Marinescu, Graphics for Macromolecular Crystallography and Electron Microscopy, CSD-TR-96-009, (submitted) 1996.

[9] T.G. Mattson, Porting Applications to the MP-Program Supercomputer: The CHARMM Molecular Dynamics Program, *Intel Supercomputer Users Group Meeting*, 1996.

[10] C. Post – Private Communication, 1996.

[11] Rossmann, M.G., The Molecular Replacement Method, *Gordon and Breach*, 1972.

[12] Rossmann, M.G., Private Communication, 1995.

[13] Rossmann M.G., McKenna R., Tong L., Xia D., Dai J., Wu H., Choi H.K., Marinescu D.C and Lynch R.E., Molecular Replacement, *Proceedings of the CCP4 Study Weekend*, 31, January – February 1992, edited by E. Dodson, S. Gover and W. Wolf, pp. 33–48, Daresbury, England: SERC.