Acta Cryst. (1995). D**51**, 749–759

Phase Refinement and Extension by Means of Non-crystallographic Symmetry Averaging using Parallel Computers

By Marius A. Cornea-Hasegan,* Zhongyun Zhang, Robert E. Lynch† and Dan C. Marinescu‡
Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907, *USA*

and Andrea Hadfield, Jodi K. Muckelbauer, Sanjeev Munshi,§ Liang Tong¶ and Michael G. Rossmann
Department of Biological Sciences, Purdue University, West Lafayette, Indiana 47907, *USA*

(*Received* 12 *September* 1994; *accepted* 27 *January* 1995)

Abstract

Electron-density averaging, fast Fourier synthesis and fast Fourier analysis programs have been adapted for parallel-computing systems. These have been linked to perform iterative phase improvement and extension utilizing non-crystallographic symmetry and solvent flattening. Various strategies for parallel algorithms have been tested on a variety of computers as a function of the number of computer nodes. Some experimental timing results are discussed.

Introduction

Advances in crystallographic techniques have often been correlated with advances in computational techniques. The last several years have seen major developments in parallel computing (Fox *et al.*, 1987). Such computing requires calculations to be distributed over a set of processors (nodes) running simultaneously. Thus, the calculation time is reduced roughly in proportion to the number of available nodes. However, to achieve such optimal behavior, care must be taken that all nodes complete their tasks at more or less the same time and that their tasks are mostly independent of each other, as communication among nodes or with an external storage device is expensive in time. New algorithms are, therefore, often required to exploit the advantages of a parallel environment.

Phase refinement and extension from low to high resolution in the determination of structures with non-crystallographic redundancy (Rossmann, 1990) is among the most computationally expensive tasks in crystallography. The essential components in this procedure are electron-density averaging and Fourier summations. We have adapted, modified and rewritten programs to perform electron-density averaging, fast Fourier analysis, phase and amplitude combination, and fast Fourier synthesis using parallel computers (Fig. 1). By combining these parallel programs with a new language (Structural Biology Language, SBL), it is possible to run iterative cycles automatically in a variety of program combinations and sequences that can be tailored to each application.

SBL is used to write scripts that can decide whether the next step should be another phase refinement iteration at the current resolution limit, whether a phase extension is warranted (and if so, by how much), whether an improved mask is to be calculated, or whether the non-crystallographic parameters are to be redetermined. In principle, it is now possible to provide a starting phase set at 20 Å resolution, for example, and end up with a high-resolution electron-density map of a 10^7 Da virus one day later without any intervention. While in 1979 a single iteration in the structure determination of southern bean mosaic virus (Abad-Zapatero *et al.*, 1980) required about 6 weeks of frustration, one iteration should now be possible in less than 1 h. Reductions in execution time permit consideration of more challenging problems and switch the emphasis away from the burdens of crystallographic techniques to the investigation of biology and biochemistry.

Structure determination based on the presence of non-crystallographic symmetry has been reviewed fairly extensively (Rossmann, 1972, 1990; Lawrence, 1991; Jones, 1992). The concept is simple (Fig. 1): electron density is improved by averaging among non-crystallographically related positions within the crystallographic asymmetric unit and by solvent flattening outside the molecular envelope where the local symmetry breaks down. The modified map is then inverted with a fast Fourier transformation (FFT) to give what should be an improved set of phases. The observed amplitudes can then be combined with these phases to compute a new electron-density map. Cycling can continue until there is no further improvement in the correlation

* Present address: Supercomputer Systems Division, Intel Corporation, 14924 NW Greenbrier Parkway, CO6-04, Beaverton, OR 97006, USA.
† Also in Department of Mathematics, Purdue University, West Lafayette, Indiana 47907, USA.
‡ Author for correspondence.
§ Present address: NCI–Frederick R & D Center, PO Box B, Frederick, MD 21702-1201, USA.
¶ Present address: Department of Medicinal Chemistry, Boehringer Ingelheim Pharmaceuticals, Inc., 900 Ridgebury Road, Ridgefield, CT 06877, USA.

between observed and calculated structure amplitudes. Once convergence has been reached, phase extension can follow by a small step outwards in reciprocal space.

Although early attempts at phase determination concentrated on reciprocal-space techniques, greater success at phase improvement was subsequently found in real space. Phase extension (Unge *et al.*, 1980), however, remained a hot topic of debate until 1984 when phases were extended from 4.0 to 3.5 Å resolution in the structure determination of hemocyanin (Gaykema *et al.*, 1984), which had sixfold non-crystallographic redundancy. In the structure determination of human rhinovirus 14 (Rossmann *et al.*, 1985), which had 20-fold non-crystallographic redundancy, phases were extended all the way from 6.0 to 3.5 Å resolution for the first time. Since then the procedure has been used for extensions from lower than 20 Å to 3 Å resolution or further in some structure determinations of spherical viruses. Thus, non-crystallographic symmetry, whether within one crystal form or between different forms, is a useful tool in the analyses of viruses, proteins and nucleic acids (Dodson, Gover & Wolf, 1992). The present parallel programs are applicable to any of these problems.

Numerous real-space averaging programs have been described previously (Buehner, Ford, Moras, Olsen & Rossmann, 1974; Bricogne, 1976; Johnson, 1978; Smith, Fraser & Summers, 1983; Hogle, Chow & Filman, 1987; Jones, 1992). The electron-density averaging program (*ENVELOPE*) described by Rossmann *et al.* (Rossmann, McKenna, Tong, Xia, Dai, Wu, Choi & Lynch, 1992; Rossmann, McKenna, Tong, Xia, Dai, Wu, Choi, Marinescu & Lynch, 1992), was chosen for adaptation to parallel environments. The parallel fast Fourier program for analysis of an electron-density map (*FFTINV*) or synthesis of a map (*FFTEXP* and *FFTSYNTH*) were based on programs written by Ten Eyck (Ten Eyck, 1973, 1977). The parallel program (*RECIP*) in which calculated phases are combined with suitably weighted observed amplitudes is based on a program written by Arnold (Arnold *et al.*, 1987).

Parallel computers and appropriate algorithms

Parallel computers like the Thinking Machine CM5, the Intel Paragon and the Cray MPP are ideal for solving problems which require large amounts of computations and a substantial amount of memory. Such systems may consist of tens to thousands of nodes interconnected by a high-speed interconnection network. There are two main classes of parallel systems. In both classes, the memory is physically distributed among the nodes. In the first class, called distributed-memory multiprocessor systems, each node has direct access only to its local memory, as in the case of the CM5 or the Paragon. Each node is an independent computer (or possibly computers) which executes a program stored in its local memory and which has access to data in its local memory and to data on other nodes *via* a network. In the second class, called shared-memory multiprocessor systems, each node has direct access to the entire memory of all the nodes, as in the case of the Cray MPP.

In a multicomputer, the network connecting both compute and input/output nodes has one of several topologies (Fig. 2), such as a two-dimensional mesh (like the Paragon), a three-dimensional torus (like the Cray MPP), a hypercube (like the iPSC/860), or a fat-tree (like the CM5). Each compute node consists of a high-performance microprocessor (Alpha for the Cray MPP, SPARC for CM5 and i860 for the Paragon), a local memory (16 to 128 Mbyte at the present time), and coprocessors (*e.g.* a communication processor). Future generations of multicomputers may have several processors in each node.

The parallel programs discussed in this paper all operate in the same-program multiple-data (SPMD) mode. The same code is loaded and executed by all nodes allo-

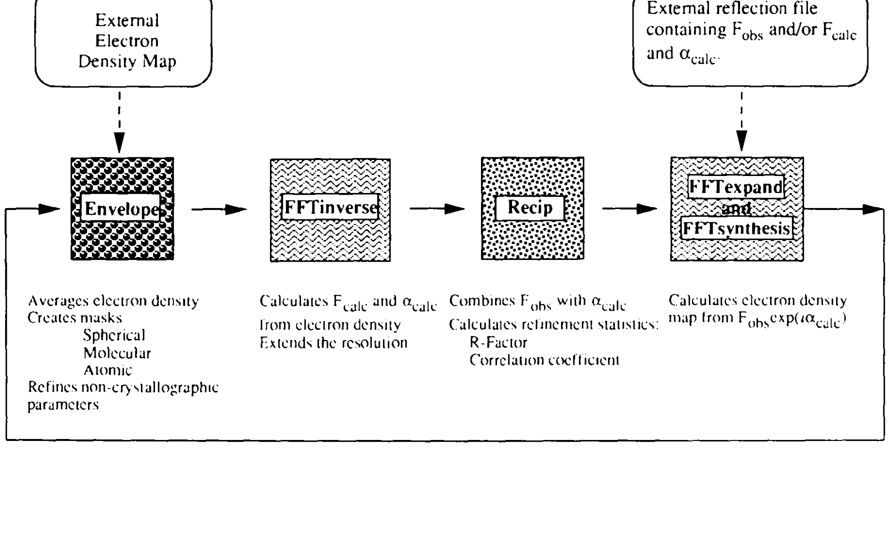
External Electron Density Map
External reflection file containing F_{obs} and/or F_{calc} and α_{calc}.
Envelope
FFTinverse
Recip
FFTexpand and FFTsynthesis
Averages electron density
Creates masks
Spherical
Molecular
Atomic
Refines non-crystallographic parameters
Calculates F_{calc} and α_{calc} from electron density
Extends the resolution
Combines F_{obs} with α_{calc}
Calculates refinement statistics:
R-Factor
Correlation coefficient
Calculates electron density map from $F_{obs}\exp(i\alpha_{calc})$

Fig. 1. A flowchart representing the iterative cycling procedure of molecular replacement real-space electron-density averaging. The boxed programs have been adapted for parallel environments. Two entry points are possible, one takes electron-density map for subsequent averaging, the other takes a set of structure factors for computation of a new map.

cated to the user. Yet the actual sequence of instructions executed by different nodes can be different due to data dependencies, and the identity of the node.

One of the more difficult problems encountered in writing a parallel SPMD mode program is data partitioning, which affects the choice of algorithm and efficiency of the computation. In the case of electron-density averaging, the three-dimensional map, with $nx \times ny \times nz$ grid points, can be partitioned into small three-dimensional volumes called 'bricks'. These bricks are brought into the local memory of a node when needed. Bricks scattered through the entire three-dimensional map might be present in any node memory at a given time (Fig. 3). In contrast the three-dimensional FFT's require planes, or a collection of planes of data, called slabs (see Fig. 4).

Another aspect of a parallel algorithm is related to load balancing. The amount of work allocated to each node should be as even as possible. The efficiency of a parallel computation is reduced by communication among nodes and 'blocking', which occurs when a node cannot continue its computation because it is waiting for data from another node or for the completion of a required calculation in another node.

It is also necessary to make the program efficient for using different numbers of nodes, depending on the size of a problem or availability of resources. The scalability is the property of an algorithm and system which describes the degree to which the execution time is reduced in proportion to the number of available processors. In the applications discussed here, consecutive programs required for phase refinement and extension need different numbers of nodes to run optimally for a problem of a given size. Among all the programs needed for phase refinement and extension, the *ENVELOPE*

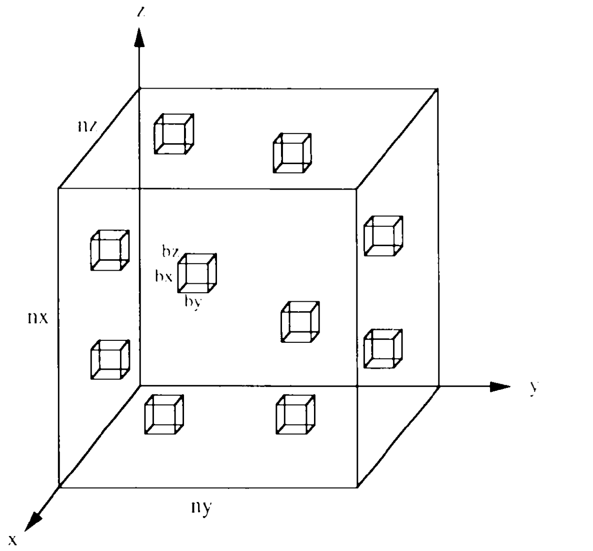

Fig. 3. Data partitioning for electron-density averaging. Given a three-dimensional map with $nx \times ny \times nz$ grid intervals per unit cell edge, a three-dimensional volume called a 'brick' is defined which has $bx \times by \times bz$ grid points. Several bricks are stored in each node.

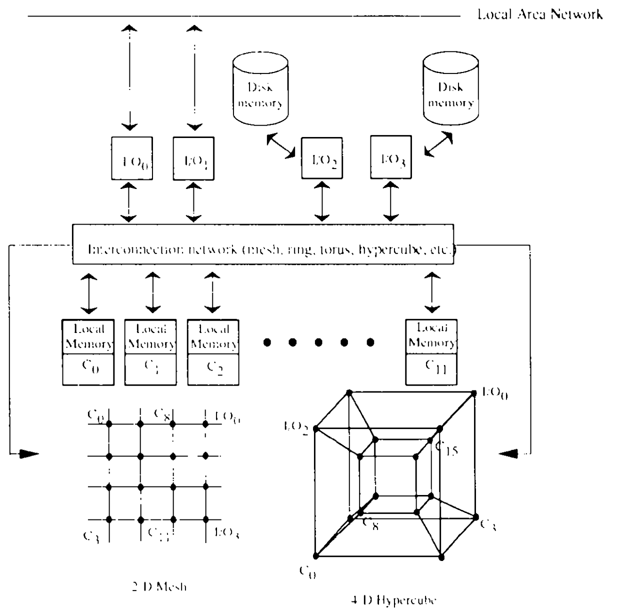

Fig. 2. The parallel environment showing the architecture of a multi-computer with examples of a two-dimensional mesh and four-dimensional hypercube as networks. Each compute node consists of a processor, local memory, coprocessor, *etc*. Dedicated input/output nodes allow for data storage (on disks or tapes) and access to computer networks.

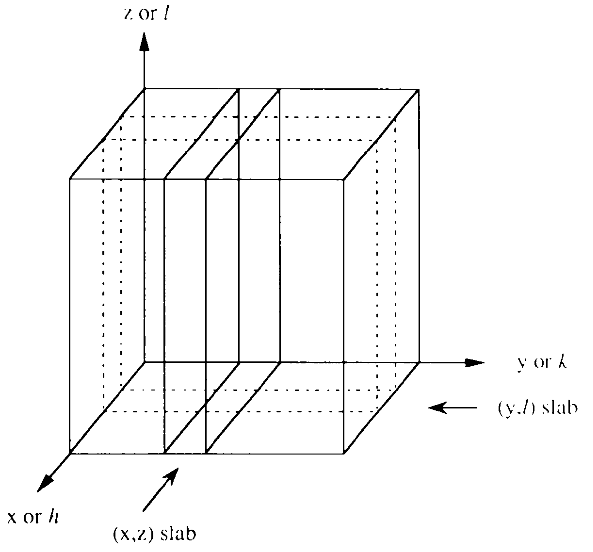

Fig. 4. Data partitioning for the *FFTINV* program. Each node is allocated an (x, z) slab of electron-density data on which the program performs two one-dimensional FFT's transforming x to h, and z to l. A global exchange of data then takes place so that each node has a (y,l) slab. Then a final one-dimensional transformation changes y to k and the result is a set of calculated structure factors at points (h, k, l) in reciprocal space. The *FFTSYNTH* program works in the opposite order, starting with slabs in h and ending up with electron density at all x, y, z grid points.

Table 1. *The three problems used to tune the ENVELOPE program*

Problem	Space group	Cell dimensions (Å, °)	Resolution (Å)	Particles per AU*	Non-crystallographic redundancy	No. of intervals on cell edge	No. of grid points along AU* edges	No. of grid points within envelope per AU*	No. of grid points in solvent or nucleic acid per AU*
A Canine parvovirus	$P4_32_12$	$a = 254.5$, $c = 795.0$	3.0	1/2	30	260 × 260 × 800	131 × 131 × 401	3983888	2897673
B *Nudaurelia* ω virus	$P1$	$a = 413.6$, $\alpha = 59.1$; $b = 410.2$, $\beta = 58.9$; $c = 419.7$, $\gamma = 64.0$	8.0	1	60	190 × 190 × 190	191 × 191 × 191	5087462	1771538
C Coxsackievirus B3	$P2_1$	$a = 574.6$; $b = 302.1$, $\beta = 107.7$; $c = 521.6$	6.0	2	120	308 × 150 × 264	309 × 76 × 265	3980614	2242646

* Asymmetric unit.

program can use the largest number of nodes most effectively.

Another potential difficulty is that when executing a sequence of parallel programs, it may be necessary to convert one data format to another. For example, the *ENVELOPE* program needs data in the brick format while the *FFTINV* program needs planes of data as input.

Electron-density averaging

For structures with large non-crystallographic redundancy, such as those discussed here, the averaging of electron density takes 75–90% of the time needed for one iteration of phase refinement or extension. A considerable effort was, therefore, devoted to the optimization of the *ENVELOPE* program (see *Appendix*) (Marinescu, Rice, Cornea-Hasegan, Lynch & Rossmann, 1993; Cornea-Hasegan, Marinescu & Zhang, 1994). Three problems *A*, *B* and *C*, were used for our measurements (Table 1).

Each grid point of the stored electron-density map is associated with an electron-density value and a mask number. The mask number identifies the molecule within the crystal to which the point belongs and thereby identifies which symmetry operators are required to map the point to all other non-crystallographic equivalent positions. Special values of the mask number identify those grid points which are outside the molecular envelope in the solvent or in the nucleic acid regions (for a virus) where non-crystallographic symmetry breaks down.

Each node is assigned a set of bricks to process and each brick is taken in turn as a 'master brick'. For each grid point within a master brick, 'slave bricks' are identified that contain the electron density at the non-crystallographically related positions. The density at a particular non-crystallographic position is calculated using an eight-point interpolation. This evaluation might require density in more than one slave brick. For moderately high resolution, most crystallographic equivalent points are likely to be in different slave bricks. If there is *N*-fold redundancy, then more than *N* slave bricks might have to be fetched to local memory from storage to calculate the average density at the first grid position of the given master brick. However, because the other grid points to be averaged are adjacent in the master brick, it is probable that they will also generate equivalent positions in the available slave bricks.

The basic parallel algorithm for electron-density averaging is shown in the pseudo-code in Fig. 5. Initially, each node is allocated a subset of master bricks. Grid points located in the solvent or nucleic acid region require a negligible amount of computation, whereas those within the molecular envelope require electron-density averaging and need a significant amount of computation. The program, therefore, allocates master bricks to the nodes in such a way that the total number of grid points to be averaged is about equal for each node, consistent with the requirement of processing an integral number of bricks in each node. This insures roughly simultaneous completion of the calculations in all nodes. The bricks within a node are sequenced to minimize the need for fetching new slave bricks in passing from brick to brick. Every time access is required to a grid point which is not in a slave brick currently in the local memory, execution is interrupted to fetch that slave brick. This is called a 'brick fault'. A 'minimum fault path' is one which guarantees that as many as possible of the slave bricks used for one master brick will be needed for the next master brick processed by that node. Hence, the master bricks are strategically sequenced within a node, by making the least possible spatial alteration when going from master brick to master brick.

Each node has a limited amount of local memory, not sufficient to hold all the bricks (Fig. 6). Three different data-management schemes have been developed for the *ENVELOPE* program to decide where to place the data: (*a*) data from the disk (DD), (*b*) data across nodes (DN) and (*c*) data servers (DS). In scheme (*a*) every time a node needs a brick which is not available in its local memory the brick is fetched from the disk. In (*b*) the data are distributed among nodes at the beginning of the computation and each node has the information about where each brick is stored. In the case of a brick fault a special type of message is sent to the node where the data are located, the computation carried out by that node is interrupted and the brick is sent to the requesting node. In (*c*) a few nodes hold all the data, acting solely as data servers, and carry out no computation, while the other nodes request the data when needed. Table 2 presents a comparison of the three data-management schemes for problem *C* in Table 1, while Table 3 shows the ratio of the execution times with an increasing number of nodes for the three problems. The DN mode is clearly the best and scales well, because the execution time approximately halves when the number of nodes doubles for both a hypercube and mesh architectures. The DS mode compares moderately well with the DN mode as the number of nodes increases, provided that the number of data-server nodes is kept small. The DD mode scales well on the hypercube for 16 and 64 nodes, and for the mesh between 64 to 128 nodes. However, scaling fails as the number of nodes increases further. This is because each node requests data from the disk *via* only a few input/output nodes. As the total number of nodes making such requests increases, bottlenecks occur at the input/output nodes. The Touchstone Delta at Cal Tech uses the same i860 processor as the Intel iPSC/860 hypercube, but has faster communication and a different architecture, producing much smaller execution times (Table 2).

The problems being compared in Table 3 all have very roughly the same number of grid points to be averaged per crystallographic asymmetric unit, but the non-crystallographic redundancy is doubled from *A* to *B* and again from *B* to *C*. This is reflected in the execution times. The greater the amount of computation, the greater are the savings on overheads. As the problem becomes larger the useful number of nodes also increases. A

```
      if (I am the coordinator node) then
          read control input file
          check consistency of input
          count the number of grid points to be averaged in each brick
          send information to other nodes
      else
          receive information from the coordinator node
      endif
      open input and output data files

C     load balancing procedure
      identify my_first_master_brick, and my_last_master_brick

C     process all master bricks allocated to me
      do brick = my_first_master_brick, my_last_master_brick

C     process all grid points in the current master brick
          do grid = 1, number_of_grid_points_in_brick
              if ( mask(grid) is solvent or acid) then
                  flatten
              else

C     in molecular envelope, average:
C     determine all other grid points related to the current one by
C     non-crystallographic symmetry and extract the electron density
                  sum = 0
                  do ncr = 1, non_crystallographic_redundancy
                      compute coordinates of point(grid, ncr)
                      if (point(grid,ncr) not in local memory) then
                          identify slave brick (s ) containing points required for interpolation
                          fetch slave brick (s)
                      endif
                      interpolate electron_density for point (grid, ncr)
                      sum = sum + electron_density
                  enddo
                  new_density = sum / non_crystallographic_redundancy
                  store new electron_density into new_master_brick
              endif
          enddo
          write new_mater_brick on the disk
      enddo

      if (I am not the_coordinator_node) then
          send to the_coordinator_node averaging statistics
      else
          receive statistics
      endif
      close files and terminate computation
```

Fig. 5. Pseudo code showing the parallel algorithm for electron-density averaging.

Table 2. *Execution times of ENVELOPE program using the three basic data-management schemes*

Execution times (s) for problem *C* in Table 1.

System				DS‡			
				No. of data-server nodes			
(network)	No. of nodes	DD*	DN†	2	4	6	8
Gamma (hypercube)	16	3700	3499	3929	4550	5441	—
	32	2049	1782	1960	2047	2182	—
	64	1393	1034	1089	1116	1123	—
Delta§ (mesh)	8 × 8 = 64	973	877	—	—	—	960
	8 × 16 = 128	575	460	—	502	935	—
	16 × 8 = 128	575	460	—	499	—	—
	16 × 16 = 256	509	273	—	303	—	—
	16 × 32 = 512	666	—	—	—	—	—

* DD = data from disk.
† DN = data across nodes.
‡ DS = data servers.
§*n* × *n* describes the aspect ratio of the mesh, altering the communication pattern between nodes.

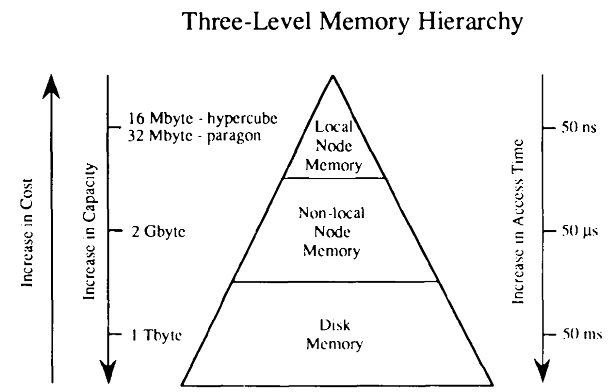

Fig. 6. The storage hierarchy. A data item can be stored either in the local main memory of a node, in the local main memory of another node, or on the shared disk. The data access time increases by approximately three orders of magnitude (from about 50 ns to at least 50 µs and then to about 50 ms). The typical amount of storage space available is about 32 Mbyte node^{-1}, 2 Gbyte for a 64–node system and 1 Tbyte disk space.

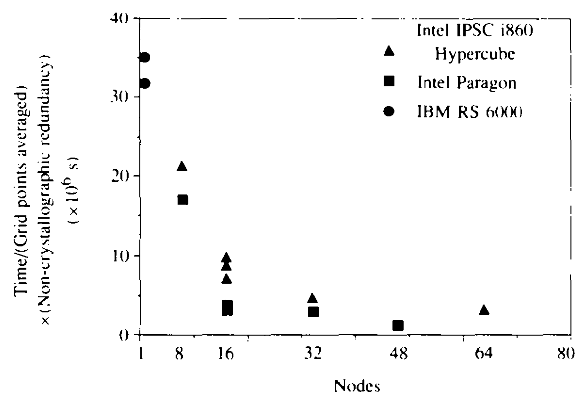

Fig. 7. *ENVELOPE* program execution times using the DN mode *versus* number of nodes for a single processor computer (IBM RS 6000) and two parallel computers (Intel iPSC/860 and Intel Paragon). Data were compiled from the projects listed in Table 4. The graph illustrates how execution time decreases with increasing number of nodes.

Table 3. *The relative speedup in the DN mode of the ENVELOPE program for different size problems*

Type of		Relative speedup*		
system	No. of nodes	*A*	*B*	*C*
iPSC/860	8	1.00	1.00	1.00
(hypercube)	16	2.19	1.94	1.95
	32	3.73	3.38	3.33
	64	5.38	5.46	5.75
Touchstone	8 × 8 = 64	7.31	6.71	6.59
Delta (mesh)	16 × 4 = 64	7.96	6.75	6.57
	8 × 16 = 128	13.60	13.13	
	16 × 8 = 128	13.67	13.46	13.43
	16 × 16 = 256	16.21	13.38	20.98

*Relative speedup = (time on iPSC 860 8 nodes)/(actual time).

sample of execution times for a variety of problems (Table 4) are given in Fig. 7.

Fourier transformations

After the average electron density has been determined in one crystallographic asymmetric unit, fast Fourier analysis (program *FFTINV*) is applied to it to calculate structure factors, $F_{\rm calc}\exp(i\alpha_{\rm calc})$. The calculated phases can then be combined with observed structure-factor amplitudes (program *RECIP*), and an FFT synthesis (program *FFTSYNTH*) can then compute a new electron-density map from the Fourier coefficients, $F_{\rm obs}\exp(i\alpha_{\rm calc})$. These three computing steps take 10 to 25% of the total computation time per cycle of iteration of phase refinement and extension.

The FFT kernels were originally written by Ten Eyck (Ten Eyck, 1973, 1977). These routines are for space group *P*1, thus permitting application to all space groups; the number of intervals along cell edges can be composite numbers with prime factors up to 19.

The Fourier analysis of the electron density is computed by a sequence of one-dimensional FFT's. The values of the input electron density, $\rho(x, y, z)$, are given

Table 4. *Completed and current projects using the parallel programs*

Project*	CCMV	CVB3	HRV3	HRV16	NOV	NωV
Space group	$P2_12_12_1$	$P2_1$	$P2_122_1$	$P22_12_1$	$P2_1$	$P1$
Cell dimensions						
a (Å)	381.3	574.6	400.8	362.6	562.1	413.6
b (Å)	381.3	302.1	344.2	347.1	354.1	410.2
c (Å)	408.6	521.6	303.9	334.9	612.8	419.7
α (°)	90.0	90.0	90.0	90.0	90.0	59.1
β (°)	90.0	107.7	90.0	90.0	110.9	58.9
γ (°)	90.0	90.0	90.0	90.0	90.0	64.0
Non-crystallographic redundancy	60	120	30	30	120	60
Resolution extent of data (Å)	3.2	3.0	2.8	2.8	3.3	2.7
Current extent of phase refinement	3.2	3.5	4.0	3.5	4.2	4.5
No. of grid points in asymmetric unit	12647635	26867217	5485401	10644480	16194821	21952000
No. of protein grid points averaged	8954977	19479677	4334789	5356345	11369066	12334766
No. reflections in an asymmetric unit used to calculate map	631455	1306112	283634	444091	955111	1086127
Protein mask	Spherical	Spherical	Spherical	Molecular	Spherical	Molecular
Inner radius (Å)	80.0	75.0	70.0	85.0	80.0	90.0
Outer radius (Å)	146.0	162.0	165.0	165.0	175.0	220.0

* CCMV, cowpea chlorotic mottle virus; CVB3, coxsackievirus B3; HRV3, human rhinovirus 3; HRV16, human rhinovirus 16; NOV, nodamura virus; NωV, *Nudaurelia ω* virus.

at grid points, $0 \le x \le nx-1$, $0 \le y \le ny-1$, $0 \le z \le nz-1$. Transforming first in the z direction, for each x and y, ny complex-valued coefficients c_1 are obtained from,

$$c_1(x, y, l) = \sum_z \rho(x, y, z)\exp(2\pi ilz).$$

Because ρ is real, $c_1(x, y, -l)$ is equal to the complex conjugate $c_1(x, y, l)$. A second set of one-dimensional transformations produces,

$$c_2(h, y, l) = \sum_x c_1(x, y, l)\exp(2\pi ihx).$$

Finally, the structure factors are obtained in a third set of transformations, where,

$$F(h, k, l) = \sum_y c_2(h, y, l)\exp(2\pi iky).$$

These summations are over points in the unit cell (*e.g.* $x = 0, \ldots, nx - 1$). But only those coefficients within the resolution limits R are saved, where,

$$|h| \le h_{\max} = a/R,\ |k| \le k_{\max} = b/R,\ 0 \le l \le l_{\max} = c/R.$$

Negative values of l are not needed, because ρ is real. The resulting $(2h_{\max} + 1)(2k_{\max} + 1)(l_{\max} + 1)$ structure factors are reduced to those within a standard asymmetric unit of reciprocal space and the given resolution limits.

In general, the number of electron-density values is so large that only a portion of the map can be stored in the local memory of most current computers (either sequential or parallel). Thus, only an '(x, z) slab' (Fig. 4) of density $\rho(x, y, z)$ with $0 \le x \le nx - 1$, $0 \le z \le nz - 1$, and for as many values of y as possible can be stored in a node at one time. The slabs of density are generated from the stored density in one asymmetric unit with knowledge of the pertinent crystallographic symmetry.

The size of a slab and the amount of memory available strongly influence the execution time. Let M denote the size of this storage. For P nodes, if $M \times P$ exceeds the volume $V = nx \times ny \times nz$, then all the data can be in node memory at one time. (At this time about 2×10^6 density values can be stored on each node of the Intel iPSC/860, or about 4×10^6 on the Intel Paragon.) In this case, the data are distributed evenly and slabs with V/P values are stored on each node, otherwise each node must transform several slabs. With a single call to the FFT kernels, the transformation with respect to z is accomplished for each of the z lines in the slab and another call transforms with respect to x. Each of the P nodes can simultaneously perform the transformations in z and x for the allocated slabs of electron density. The coefficients c_2 must then be exchanged among nodes. This is necessary because no node has an entire set of values along any line in the y direction. Data are exchanged among nodes in order to construct (y, l) slabs for a range of h values (Fig. 4). The execution time for *FFTINV* (and similarly for *FFTSYNTH*) depends crucially on how these exchanges are carried out.

Two distinct mechanisms for this global transposition have been implemented: (*a*) the use of an intermediate disk file (scratch file mode) when $M \times P \le V$ and (*b*) an internodal exchange (global exchange mode) when

Table 5. *Analysis of execution times for FFTSYNTH tested on HRV16 (see Table 4) at 3 Å resolution (all times are in s)*

'Start' is the average time per node from start to when structure factors begin to be read. 'Read' is the average time per node to read the structure factor from the disk. 'FFT' is the average time per node to carry out all three (primary, secondary and tertiary) Fourier summations. 'Exchange' is the average time per node to globally exchange the data among nodes. 'Write' is the average time per node for writing the electron density onto the disk. '(Total)' is the average time per node for the execution of the program measuring from the start to the last node to finish.

(*a*) Ratio (time for scratch exchanges)/(time for global exchange)

Nodes	NIH 128 Hypercube	Cal Tech 512 Paragon
8		58.573/5.064 = 11.6
16	24.873/3.357 = 7.41	58.050/3.365 = 17.3
32	44.174/2.407 = 18.35	103.692/3.716 = 27.9
64	69.278/1.857 = 37.31	220.986/3.721 = 59.4

(*b*) Exchange time as a percent of total time

	NIH 128 Hypercube		Cal Tech 512 Paragon	
Nodes	Scratch	Global	Scratch	Global
8			63.5	13.2
16	45.4	10.5	68.3	12.1
32	69.6	11.2	76.7	12.0
64	83.1	10.5	78.0	6.6

(*c*) Breakdown of various components of the *FFTSYNTH* inversion on the Purdue University 140-node Paragon

Nodes	16	32	48	64	96	128
Start	1.855	2.990	4.594	9.292	20.050	24.500
Read	2.939	1.783	2.693	1.803	1.791	1.507
FFT	10.455	5.237	3.493	2.614	1.734	1.295
Exchange	2.619	2.477	2.440	2.639	3.083	4.699
Write	4.091	3.539	3.503	3.284	4.713	4.773
(Total)	22.235	16.252	16.924	19.776	31.489	36.914
Total	30.949	28.158	20.050	30.100	51.046	64.551
% FFT	33.8	18.6	12.0	8.68	3.40	2.01

$M \times P > V$. The first takes much more time than the second, not only because of the time necessary to transfer data between a node and a disk (Fig. 6), but also because of the bottleneck occurring when several nodes need access to the disk at the same time. The program uses (*b*) unless the data do not fit into the combined memory of all nodes allocated to the task.

In the global exchange mode, the exchange of data takes place by a series of messages transmitted among the nodes. For example, assume there are only four nodes, 0, 1, 2 and 3, and suppose $nx = ny = nz = 8$. Node 0 transforms a slab with two (x, z) planes, those with $y = 0$ and 1. Then during the exchange, node 0 keeps the 'stick' of data with $h = 0, 1; y = 0, 1; l = 0, \ldots, 7$. Node 0 must receive the stick with $h = 0, 1; y = 2, 3; l = 0, \ldots, 7$ from node 1 and, likewise, sticks with $h = 0, 1$ from the other nodes. At the end of the exchange, node 0 has the slab of $c_2(h, y, l)$ values with $h = 0, 1; y = 0, \ldots, 7; l = 0, \ldots, 7$. Node 1 collects sticks with $h = 2$ and 3 from nodes 0, 2, 3, and similarly for the other nodes.

This exchange is accomplished in the following three steps.

(1) Nodes 0 and 1 exchange sticks; while also nodes 2 and 3 exchange sticks.

(2) Nodes 0 and 2 exchange sticks; while also nodes 1 and 3 exchange sticks.

(3) Nodes 0 and 3 exchange sticks; while also nodes 1 and 2 exchange sticks.

Because of the architecture and the design of the communication system, each of these three steps takes the same amount of time. Similarly, when P nodes are used, all of the data is exchanged in $P - 1$ steps on a hypercube or P steps on a mesh. The Fourier synthesis is computed in an identical manner to the Fourier inversion, but in the opposite order. Table 5 lists times for these two exchange schemes for *FFTSYNTH* applied to the HRV16 problem (Table 4). The times show that the use of the global exchange method takes an order of magnitude less execution time than the scratch file method. In addition to the exchange time, there is 'overhead': time must be spent to load the program onto the nodes, time is required to read the input data and to write the output data, *etc.* The time actually devoted to the FFT transformation is rather small compared to these other times which are necessary to carry out the computation.

Only the transformation time is scalable. The transformation time (labeled 'FFT' in Table 5) halves when the number of nodes doubles. But because of the 'overhead' times, the total execution time first decreases and then increases as the number of nodes increases. For the HRV16 problem at 3 Å resolution, the minimum time to execute the complete *FFTSYNTH* program was 27 s with 32 nodes of the 128-node hypercube and was 37 s with 16 nodes of the 512-node Paragon. Results of timing experiments for the CVB3 problem (Table 4) are similar (Lynch & Zhang, 1994).

Structure-factor sorting

Calculated phases, obtained from the *FFTINV* routine, are combined with the original observed amplitudes and used in the calculation of statistics useful in the assessment of convergence in phase refinement (*e.g.* correlation coefficients and *R* factors). The *RECIP* program sorts the output from the *FFTINV* program into the same sequence as used for listing the observed amplitudes of one reciprocal-space asymmetric unit. First, the reflections are distributed evenly to *P* nodes. Then each node sorts its allocated set of structure factors in ascending order according to the values of the reflection identifier $h \times 2^{20} + k \times 2^{10} + l$. Next, information about the distribution of the sorted reflections is sent by each node to node 0. Node 0 uses this information to determine the range of reflections for each node, so that each node will merge approximately the same number of reflections. These limits are sent to the nodes and then an exchange of reflections is carried out similar to the exchange used by *FFTINV* and by *FFTSYNTH*. Each node then merges the reflections it has received from the other nodes into a fully ordered set (see Table 6 for execution times).

One of the options of the *RECIP* program is that each calculated structure-factor amplitude which corresponds to an observed amplitude is replaced to obtain the Fourier coefficients, $wF_{\rm obs}\exp(i\alpha_{\rm calc})$, where w is the weight or figure of merit determined from $F_{\rm obs}$ and $F_{\rm calc}$. These reflections are passed to program *FFTEXP*, which uses knowledge of the crystallographic symmetry to expand the reflection data from an asymmetric unit to a hemisphere of reciprocal space. This is necessary as the FFT synthesis program is for space group *P*1. The coefficients are then sorted using the same algorithm as before for presentation to the *FFTSYNTH* program (see Table 6 for execution times).

The Structural Biology Language

A desirable computing environment for solving complex problems on parallel machines would be a high-performance front-end graphics workstation to provide windows and display capability, connected to a variety of remote parallel computers. Parallel machines are inherently more complex and less user friendly than sequential computers. Access to parallel machines is rather restricted, as they are expensive shared resources. To allow a user to focus on the application without being distracted by the problems of different systems, an environment has been created to hide as much as possible of the complexities. It also includes graphical output for the inspection of diffraction patterns, electron-density maps and mask allocation. Part of this environment is a special 'Structural Biology Language' (SBL) (Cornea-Hasegan & Marinescu, 1994).

Table 6. *Execution times* (s) *for RECIP and FFTEXP*

The table refers to the HRV16 project at 3.5 Å resolution (Table 4). All times are affected to some extent by concurrently running programs in other partitions of the systems.

	Intel iPSC/860 at NIH		Intel Delta		Intel Paragon	
Nodes	*RECIP*	*FFTEXP*	*RECIP*	*FFTEXP*	*RECIP*	*FFTEXP*
2					135	95
4	352	151	129	64	75	47
8	313	100	83	36	54	36
16	260	84	50	22	56	27
32	287	76	38	16	58	22
64			37	13		
128			61	14		

The programs currently available in this environment relate to phase refinement and phase extension in the presence of non-crystallographic symmetry, including the special case of merely solvent flattening when the non-crystallographic redundancy is one. It is intended to be augmented to support other types of programs to cover many aspects of structural biology and macromolecular crystallography. For instance, a start has been made to develop parallel algorithms for diffraction data processing (Rossmann, 1979; Rossmann, Leslie, Abdel-Meguid & Tsukihara, 1979).

Two types of programs can be invoked by SBL.

(*a*) Primary programs like *ENVELOPE*, *FFTINV*, *RECIP*, *FFTEXP* and *FFTSYNTH*, rotation-function calculations, rigid-body refinements and structure-factor calculations.

(*b*) Auxiliary programs designed to support the appropriate sequencing of the programs in the first category, to test some of their output, and to transform part of the control input data for these programs.

Currently these programs are available on: (*a*) iPSC/860 systems, at Purdue University and at NIH; (*b*) the Paragon systems at Purdue University and California Institute of Technology and Intel Super Computer System Division. Source code can be obtained from the authors.

SBL permits multiple nested iterations, and two different levels of phase extension. SBL also allows the computation to be restarted after being stopped by the user, by a hardware failure or exhaustion of the allocated time. The computation can be resumed from the end of the last completely executed program in the series of programs requested. Every SBL program is compiled into a UNIX shell script (UNIX command list), which controls the execution of the entire sequence of programs.

Testing for convergence

The major criterion used in tests of convergence during phase refinements is the correlation between $F_{\rm obs}$ and $F_{\rm calc}$. The $F_{\rm calc}$ values are obtained by back Fourier transformation of the modified electron-density map. The correlation coefficient, C, is defined as,

$$C = [\sum(\langle F_{\rm obs}\rangle - F_{\rm obs})(\langle F_{\rm calc}\rangle - F_{\rm calc})] \div \{[\sum(\langle F_{\rm obs}\rangle - F_{\rm obs})^2 \sum(\langle F_{\rm calc}\rangle - F_{\rm calc})^2]^{1/2}\},$$

where $\langle F_{\rm obs}\rangle$ and $\langle F_{\rm calc}\rangle$ are mean values and the sums are over all observed reflections in specified resolution shells. Convergence tests are based on the improvement, Δ, of the overall C in successive iterations, i, where $\Delta = C_i - C_{i-1}$. The first and most important test is that if the overall C improves. However, additional tests can also be made to ensure the following.

(1) That the refinement is uniform over the complete resolution range. If the C's in some resolution shells are substantially poorer than in neighboring shells, then this might imply conflicts between differing phase solutions (Arnold *et al.*, 1987).

(2) That there is no further change in C in all resolution shells.

(3) That the C's have essentially converged for all sizes of F. The larger F's, although fewer in number, have a greater influence on the electron density and will, therefore, converge faster.

Concluding remarks

The parallel algorithms used in the programs described here are designed for distributed and shared-memory MIMD (multiple instruction multiple data) architectures. There are several types of MIMD systems: (*a*) massively parallel processing systems (MPP's), such as discussed in this paper, (*b*) clusters of workstations and (*c*) multiprocessor workstations. The programs for workstation clusters (available by mid-1995 from the authors) use the same algorithms as those described here, but communicate using libraries like MPI (message passing interface) and PVM (parallel virtual machine).

A small problem could be solved using one or a few workstations, while a large problem may require tens or possibly hundreds of workstations or an MPP with tens or hundreds of compute nodes. In contrast, a sequential program can only run on one single processor workstation regardless of the problem size. By harnessing them together or by using many compute nodes in an MPP, the computation will be carried out faster in proportion to the number of computers running parallel. Some of the largest problems we describe here are still outside the usual memory capacity of the largest workstations currently available. Considering the crystallographic difficulties, the coxsackievirus B3 virus structure (Muckelbauer *et al.*, 1995) would have been difficult to solve without the help of a parallel-processing system. Parallel computers permitted testing of a much larger number of procedures (Muckelbauer *et al.*, 1995) in a reasonable time frame eventually leading to a successful solution.

Although faster workstations with larger internal memory are emerging every year, so are larger crystalline biological complexes, as are progressively more computer intensive techniques for the solutions of difficult crystallographic problems. Parallel computers can substantially reduce computing time and permit the study of problems with very large memory requirements. With the ability to collect diffraction data of complex biological crystallized samples at ever increasing rates, it is anticipated that expansion of the use of parallel computers will accompany the exploration of new biological frontiers.

We are grateful for many helpful discussions with Mavis Agbandje, Michael Chapman, Jack Johnson, Robert McKenna, Padmaja Natarajan, Adam Zlotnick and the continuing enthusiasm of John Rice. We are also grateful for help from John Steele, John Jackson and others at the Purdue University Computer Center, as well as the Cal Tech and NIH computing centers. The work was supported by NSF grants to Robert E. Lynch and Michael G. Rossmann, an NIH grant to Michael G. Rossmann, as well as support from the Lucille Markey Foundation for expansion of structural studies at Purdue University.

APPENDIX

The *ENVELOPE* program provides several functions (Rossmann, McKenna, Tong, Xia, Dai, Wu, Choi, Marinescu & Lynch, 1992).

1. *Mask generation*

(*a*) Generates masks based upon intersecting spheres. This is the simplest way of determining a mask, useful in the initial stages of a structure determination.

(*b*) Generates masks based on the current averaged electron density placed into a standard orientation. By first averaging the density into a standard orientation, the surrounding molecules average out, because the local symmetry breaks down outside the molecular envelope. Hence, this new map can be used to determine the limits of the molecular envelope.

(*c*) Generates masks based on atomic positions of a homologous molecule. If a reasonable structure is already available, it might be the best guide for determining an envelope.

(*d*) Modifies a mask (enlarges, shrinks, fills holes). These operations ensure that the mask has the correct non-crystallographic symmetry and touch up other properties.

2. *Averaging operations*

(*a*) Modifies electron density by averaging, solvent flattening, *etc.* This is the central averaging procedure for phase improvement and averaging.

(*b*) Places the averaged electron density into a standard orientation. This operation is referred to in 1(*b*) above. In essence the calculations are similar to those in 2(*a*).

3. *Other facilities*

(*a*) Takes electron density from a standard orientation and places it into the crystal cell taking into account the various molecular envelopes. If a structure was previously defined in a standard orientation, then this procedure is useful in generating a starting density within the crystal of interest.

(*b*) Determines optimal orientation and position of a particle by searching for minimum electron-density scatter or maximum height of a given electron-density feature such as a heavy atom.

(c) Other functions related to data conversion, *e.g.* merging mask and electron-density data, converting data from brick to plane format and back, *etc.*

References

ABAD-ZAPATERO, C., ABDEL-MEGUID, S. S., JOHNSON, J. E., LESLIE, A. G. W., RAYMENT, I., ROSSMANN, M. G., SUCK, D. & TSUKIHARA, T. (1980). *Nature (London)*, **286**, 33–39.

ARNOLD, E., VRIEND, G., LUO, M., GRIFFITH, J. P., KAMER, G., ERICKSON, J. W., JOHNSON, J. E. & ROSSMANN, M. G. (1987). *Acta Cryst.* A**43**, 346–361.

BRICOGNE, G. (1976). *Acta Cryst.* A**32**, 832–847.

BUEHNER, M., FORD, G. C., MORAS, D., OLSEN, K. W. & ROSSMANN, M. G. (1974). *J. Mol. Biol.* **82**, 563–585.

CORNEA-HASEGAN, M. A. & MARINESCU, D. C. (1994). Purdue Univ. Department of Computer Sciences, Tech. Rep., CSD TR-94-008.

CORNEA-HASEGAN, M., MARINESCU, D. C. & ZHANG, Z. (1994). *Concurrency Pract. Exper.* **6**, 205–229.

DODSON, E., GOVER, S. & WOLF, W. (1992). Editors. *Molecular Replacement. Proceedings of the CCP*4 *Study Weekend*, 31 *January*–1 *February* 1992. Daresbury, England: SERC.

FOX, G., JOHNSON, M., LYZENGA, G., OTTO, S. O., SALMON, J. & WALKER, D. (1987). *Solving Problems on Concurrent Processors.* Englewood Cliffs: Prentice-Hall.

GAYKEMA, W. P. J., HOL, W. G. J., VEREIJKEN, J. M., SOETER, N. M., BAK, H. J. & BEINTEMA, J. J. (1984). *Nature (London)*, **309**, 23–29.

HOGLE, J. M., CHOW, M. & FILMAN, D. J. (1987). *Crystallography in Molecular Biology*, edited by D. MORAS, J. DRENTH, B. STRANDBERG, D. SUCK & K. WILSON, pp. 281–292. New York: Plenum Press.

JOHNSON, J. E. (1978). *Acta Cryst.* B**34**, 576–577.

JONES, T. A. (1992). *Molecular Replacement. Proceedings of the CCP*4 *Study Weekend*, 31 *January*–1 *February*, 1992, edited by E. DODSON, S. GOVER & W. WOLF, pp. 91–105. Daresbury, England: SERC.

LAWRENCE, M. C. (1991). *Quart. Rev. Biophys.* **24**, 399–424.

LYNCH, R. E. & ZHANG, Z. (1994). Purdue Univ. Department of Computer Sciences, Tech. Rep. CSD TR-94-054.

MARINESCU, D. C., RICE, J. R., CORNEA-HASEGAN, M. A., LYNCH, R. E. & ROSSMANN, M. G. (1993). *Concurrency Pract. Exper.* **5**, 635–657.

MUCKELBAUER, J. K., KREMER, M., MINOR, I., TONG, L., ZLOTNICK, A., JOHNSON, J. E. & ROSSMANN, M. G. (1995). *Acta Cryst.* D**51**. In the press.

ROSSMANN, M. G. (1972). *The Molecular Replacement Method.* New York: Gordon & Breach.

ROSSMANN, M. G. (1979). *J. Appl. Cryst.* **12**, 225–238.

ROSSMANN, M. G. (1990). *Acta Cryst.* A**46**, 73–82.

ROSSMANN, M. G., ARNOLD, E., ERICKSON, J. W., FRANKENBERGER, E. A., GRIFFITH, J. P., HECHT, H. J., JOHNSON, J. E., KAMER, G., LUO, M., MOSSER, A. G., RUECKERT, R. R., SHERRY, B. & VRIEND, G. (1985). *Nature (London)*, **317**, 145–153.

ROSSMANN, M. G., LESLIE, A. G. W., ABDEL-MEGUID, S. S. & TSUKIHARA, T. (1979). *J. Appl. Cryst.* **12**, 570–581.

ROSSMANN, M. G., MCKENNA, R., TONG, L., XIA, D., DAI, J., WU, H., CHOI, H. K. & LYNCH, R. E. (1992). *J. Appl. Cryst.* **25**, 166–180.

ROSSMANN, M. G., MCKENNA, R., TONG, L., XIA, D., DAI, J., WU, H., CHOI, H. K., MARINESCU, D. & LYNCH, R. E. (1992). *Molecular Replacement. Proceedings of the CCP*4 *Study Weekend*, 31 *January*–1 *February* 1992, edited by E. DODSON, S. GOVER & W. WOLF, pp. 33–48. Daresbury, England: SERC.

SMITH, G. E., FRASER, M. J. & SUMMERS, M. D. (1983). *J. Virol.* **46**, 584–593.

TEN EYCK, L. F. (1973). *Acta Cryst.* A**29**, 183–191.

TEN EYCK, L. F. (1977). *Acta Cryst.* A**33**, 486–492.

UNGE, T., LILJAS, L., STRANDBERG, B., VAARA, I., KANNAN, K. K., FRIDBORG, K., NORDMAN, C. E. & LENTZ, P. J. JR (1980). *Nature (London)*, **285**, 373–377.