1994

The Parallel Envelope

M.G. Rossmann

M. A. Cornea-Hasegan

D. C. Marinsecu

Z. Zhang

Robert E. Lynch
Purdue University, rel@cs.purdue.edu

See next page for additional authors

Report Number:
94-018

THE PARALLEL ENVELOPE

M.G. Rossmann[1], M.A. Cornea-Hasegan[2], D.C. Marinescu[2],
Z. Zhang[2], R.E. Lynch[2], A. Hadfield[1], J. Muckelbauer[1],
R. McKenna[1], S. Munshi[1], J-B. Dai[1]
Biological Sciences and Computer Sciences Departments
Purdue University
West Lafayette, IN 47907

CSD-TR-94-018
March 1994

[1] Biological Sciences Department
[2] Department of Computer Sciences

THE PARALLEL ENVELOPE

M.G. Rossmann[1], M.A. Cornea-Hasegan[2], D.C. Marinescu[2],
Z. Zhang[2], R.E. Lynch[2], A. Hadfield[1], J. Muckelbauer[1],
R. McKenna[1], S. Munshi[1], J-B. Dai[1]
Biological Sciences and Computer Sciences Departments
Purdue University
West Lafayette, IN 47907

The Envelope program was developed by M.G. Rossmann and others [1] in the late 1980's. This document describes the parallel version of this program written for distributed memory Multiple Instruction Multiple Data (MIMD), systems like the Intel iPSC/860, the Touchstone Delta and the Paragon. It contains:

E1 Envelope User's Guide
E2 Envelope File Format
E3 The Precedence of the Control Information
E4 Envelope Iimplementation Notes

The load balancing and data distribution algorithms for the parallel version of the program are described in [2] and [3].

[1]Biological Sciences Department
[2]Department of Computer Sciences

E1 ENVELOPE USER's GUIDE

Related documents
- E2. Envelope – File Formats
- E3. Envelope – Control Information Precedence
- E4. Envelope – Internals

Last update: dcm, January 11, 1994
ENVELOPE – Real Space Molecular Replacement and Molecular Envelope Finder Program

Authors:

Michael G. Rossmann	(b4p@mace.cc.purdue.edu)
Dan C. Marinescu	(dcm@cs.purdue.edu)
Marius A. Cornea-Hasegan	(cornea@cs.purdue.edu)
Zhongyun Zhang	(zz@cs.purdue.edu)
Robert E. Lynch	(rel@cs.purdue.edu)

plus Liang Tong, Rob McKenna, Hao Wu, and others.

Function: to carry out calculations related to real space averaging.

The input to the program includes crystallographic and non-crystallographic symmetry operators, spatial and cell parameters, and values of electron density at points of a 3-dimensional grid (a 'map'). Output includes a map of the density which has been averaged at all of the non-crystallographic symmetrically related points and 'flatten' (or left unchanged) elsewhere. The program can determine a 'mask', a code which indicates the grid points at which the density is to be averaged. This mask possesses the non-crystallographic symmetry.

Two cells are defined: the p cell and the h cell. The p cell is the actual cell which is being studied and whose density is to be averaged

The h cell is a cell (usually with orthogonal axial directions) with respect to which the non-crystallographic symmetry can be defined. It can also be used to transpose the molecule(s) from the p cell into a standard orientation in the h cell.

A fuller explanation and description is given by Rossmann et al. in J. Appl. Cryst. [1].

Program versions:

The parallel version of the program executes on the Intel iPSC/860, Touchstone Delta and the Paragon. It is a major revision of a program developed by Rossmann and others for execution on the Control Data Cyber 205. A sequential version executes on IBM 6000 RISC workstations.

We describe the January 1994 version of the Envelope program. Its executable files are located at the Purdue University Computing Center file system for its Intel hypercube in

cryst/Prog/i860/Envelope/V1/1.3/ENVELOPE

and in the California Institute of Technology's file system for its Intel Paragon in

cryst/Prog/Paragon/Envelope/V1/1.3/ENVELOPE

These versions of the program for the i860 and Paragon produce identical results.

Input and Output Files:

Most options of the Envelope program use:

Input files:

- a problem description file (control input file) - (UNIX file)
- a data input file - (CFS file for the i860 and PFS file for the Paragon)
- possibly other input data files

and produce:

Output files:

- a summary (a control output) file - (UNIX file)
- a data output file - (CFS or PFS file)
- possibly other output files

CFS stands for Concurrent File System and it is a special file system which strips data for parallel I/O. A file is spread across several I/O devices (disks) and across several I/O nodes. CFS allows fast access to a large file from all the nodes. The CFS is available on the iPSC/860 and the Delta.

PFS stands for Parallel File System. It is a special file system for the Paragon.

CAUTION: There is no automatic backup of the CFS files. The summary output file is the UNIX standard output file, and it can be redirected to a UNIX file. At the end of the job, this output contains all the information generated on node 0 or the other nodes. This output contains the crystallographic output.

In addition each option can control as many as 8 input or output files. The file names are given in the Control Input File along with each option as it is required.

WARNING: If you specify in the control input file a data input file which does not exist then the message given by the operating system is sent to the screen ONLY, it will not appear in your summary output file.

1. The Control Input File

The control input file can be created directly by the user in the format described below or it can be created interactively by using the SBUI (SB-User Interface).

WARNING: the information supplied by the user in a control input file and the information available from the file headers are combined as shown in the document E4.

Line 1 IMODE IGROUP (25x,2i4)

IMODE=0 put the mask/density file onto cfs/pfs (parallel file system).

=1 Distribute the mask/data across the Nodes (the 'DN' mode of operation). This is faster if you have enough memory over all the nodes you use. You must have at least 8 nodes to operate this way.

IGROUP The number of nodes over which the data is to be distributed. This parameter is needed only if IMODE.ne.0. For instance, if IGROUP is 8 and you are using 16 nodes, then the mask and density will be stored twice, once on the first group of nodes and once on the second group. This minimizes the communication required for getting a brick of data into any specific node.

Line 2 IOPTION NFILES (25x,2i4)

IOPTION The option number to be executed. Option 0 terminates the option sequence input.

NFILES The number of files to be specified in using the option

Line 3 IFILE FILENAME (25x,i4,3x,a64)

IFILE The type of file

FILENAME The name of the file (one line per file required)
The type of files permitted for each option is described below. Different ways of running an option may require more or fewer files. Only those files which are actually required need to be given.

Lines 2 and 3 are repeated for each option, in the order that the options are to be executed, until Option 0 occurs.

P CELL INPUT

Line 4 (CELL(2,I),I=1,6) (6f10.2)

cell dimensions a,b,c (in angstroms), and alpha,beta,gamma (in degrees) of the unknown (p) cell

Line 5 NSYM NTYPE IMASK ISTORE IPRNT JPRNT ICELL LXTND IFILL IMPROP (10i3)

NSYM number of crystallographic symmetry operations

NTYPE=1 read polar angles relating h to p cell

=2 read Eulerian angles relating h to p cell

=3 read C matrix relating h to p cell

=4 read P matrix relating h to p cell

=5 read CINV matrix relating h to p cell

=6 read PINV matrix relating h to p cell

where

$$X=[P]Y+d$$

Note that this definition is opposite to that for the HOMO program if you think of the first crystal as the h crystal and the second crystal as the p crystal. It is the same definition as is used in the translation function program.

IMASK=-1 create and write a mask (use one asymmetric unit and label each grid point with a number corresponding to the appropriate particle center, also differentiate between solvent and nucleic acid). The electron density of the points within the envelope is set to the value at the corresponding grid points in the h-cell.
Use IMASK=2 if mask was generated prior to May 92

IMASK=0 create and write a mask (use one asymmetric unit and label each grid point with a number corresponding to the appropriate particle center, also differentiate between solvent and nucleic acid).
Set electron density at points within mask to 100. If you plan to use ENLARGE then use this option.

Use IMASK=3 if mask was generated prior to May 92

ISTORE=0 do automatic rescaling in Option 8 (store p cell density)

=1 do not alter the value of SCALEP

JPRNT=0 do NOT print non-crystallographic symmetry matrices.

=1 print non-crystallographic symmetry matrices.

The next two parameters refer to the creation of a mask in the p cell given density for a single molecule (particle) in the h cell.

ICELL=0 use the standard 'vecplpt' routine

=1 use the special 'vecplpt' routine for Sanjeev

LXTND=0 do not enlarge envelope

>0 Check symmetry and enlarge (or reduce) volume of envelope.
The electron density is set to 100.0 within protein and 0.0 outside. This density is then averaged. Any grid point within the envelope will get an average of 100.0 and outside of 0.0. Points at the edge will have an intermediate density of MAXPROT. Then if a grid point is inside envelope and MAXPROT<LXTND the point is changed to solvent or acid if grid point is outside envelope and MAXPROT>LXTND the point is changed to be inside envelope.
Thus if LXTND=50 then the routine assures that the envelope obeys non-crystallographic symmetry but if LXTND>50 the envelope volume is increased or if LXTND<50 the envelope volume is shrunk.

IFILL<0 do not fill in holes in protein mask

>0 fill in holes in protein mask. If the number of protein points from the same mask that surround a non-protein point is larger or equal to IFILL, then this grid point is changed to be within this mask.

The default is 6. Thus, for the default value, all single grid point holes that are completely within one molecule are filled.

IMPROP the number of particles or molecules in the crystallographic asymmetric unit related by improper non-crystallographic symmetry. For instance if there are two molecules related, by say, kappa=150 degrees rotation and a 30A translation, or if there are two icosahedral virus particles per crystallographic asymmetric unit.
Default=1.

Line 6 Only if IPRNT.ne.0, INCX INCY INCZ (free)

Lines 7 ((CENTER(J,I),I=1,3),J=1,IMPROP) (3f10.4)
centers of particles within first crystallographic asymmetric unit in fractional coordinate for unknown cell (p) with 'y' coordinates. The second and subsequent particles are related by improper non-crystallographic symmetry to the first.

Line 8 RADOUT RADIN CORAD (3f10.4)
outer, inner, and core radii in angstroms.
All density inside the abs(core) radius will be set to nucleic acid.
Between CORAD and RADIN the density will be set to protein if it is greater than CRIT1 and to acid otherwise.
Between RADIN and RADOUT the density will be set to solvent if it is less than CRIT1.
If it is less than CRIT2 it will be set to the mask with the smallest radius.
If it is bigger than CRIT2 it will be set to the mask with the largest density.

Line 9 (((SYM(N,I,J),J=1,4),I=1,3),N=1,NSYM) (12F6.2)
crystallographic real space symmetry operators.

Line 10 NX NY NZ IIX IFX IIY IFY IIZ IFZ SCALEP IIYAV IFYAV (9i3,f8.4,2i3)

NX,NY,NZ are the number of grid intervals in a,b,c (one less than the number of grid steps).
IIX,IIY,IIZ are the first points in a,b,c that defines the asymmetric unit (the first grid point in the array is #1).
IFX,IFY,IFZ are the last grid points that defines the asymmetric unit.
IIYAV,IFYAV are the first and last sections to be averaged in the p cell. Defaults are IIYAV=IIY,IFYAV=IFY.

Note 1. While the initial and final positions may cover any size unit that is equal to or greater than the crystallographic asymmetric unit, it is assumed that at least one asymmetric unit is defined by these values that lies in the range .ge. 0 and .lt. 1.0 in all directions; where the units are in fractions of the unit cell.

Note 2. The header information in the map file (PEDMAP) will overwrite (over ride) the information in Line 6.
SCALEP is scale factor to be applied to the p cell input electron density.

Line 11 ISWPROT ISWSOLV ISWACID CAY SCLSOLV SCLACID (3i2,3f8.4)

ISWPROT=0 leave protein density
=1 leave protein density
(ISWPROT=1 when wishing to put the p cell density into the h cell without averaging)
=2 average protein density using 8 point interpolation
=3 average protein density using 11 point interpolation

ISWSOLV=0 do not change solvent density
=1 set solvent density to zero
=2 set solvent density to its mean value
ISWACID=0 do not change acid density
=1 set acid density to zero
=2 set acid density to its mean value
CAY=1.0 then the input file to Option 6 is a p-cell mask file;
CAY=2.0 the input file to Option 6 is an h-cell mask file (other values of CAY are not accepted by Option 6)
SCLSOLV apply this scale factor to solvent (default=1.0)
SCLACID apply this scale factor to acid (default=1.0)
(Note: as modification into the h cell does not take any notice of the p cell mask, these factors are irrelevant when averaging into the h cell).

H CELL INPUT

Line 12 (CELL(1,I),I=1,6) (6f10.2)
cell dimensions for h cell

Line 13 (CENTERH(I),I=1,3),CRIT1,SCALEH,CRIT2 (6f10.4)
Center of envelope in h cell given in fractional.
CRIT1, CRIT2: Take a grid point in the p cell and find its position in the h cell for each molecule whose center is less than RADOUT from a molecular center in the p cell. Then look up the density in the h cell according to the SMRAD and JSMEAR parameters.

If all these densities are below CRIT1 then this grid point is assigned to be solvent or acid. If one or more of the densities are above CRIT2, then the grid point is assigned to that molecule which has the largest h cell density. If one or more of the possible molecular centers correspond to a density greater than CRIT1, but none attain a value as big as CRIT2 then assign the grid point according to which molecular center is closest. The default for CRIT2 is CRIT1. Thus in the default situation, the mask is assigned only on the basis of h cell density values.

SCALEH is multiplier of h cell density when it is written into storage. The h cell scaled density will be stored in the p cell mask on generating the mask. Limits on the stored density are +/- 512 for the p cell.

Line 14 (NAXIS(I),I=1,3),SMRADSQ,JSMEAR,JSTORE (3i2,f8.2,2i2)

NAXIS defines the directions in the map. Rows in the map correspond to NAXIS(1), columns in the map correspond to NAXIS(2), sections in the map correspond to NAXIS(3), where NAXIS(i)=1,2, or 3 correspond to a,b,c.

SMRADSQ is square of distance used to limit sphere around a grid point (in units of grid points) for smearing density when generating mask in p cell from h cell density. Default=3.0

JSMEAR=0 take maximum density of all those within SMRAD

=1 take mean density of all those within SMRAD when generating mask in p cell from h cell density.

=2 same as 0, but use abs(density)

=3 same as 1, but use abs(density).

JSTORE=0 store h cell density counting from virus center when averaging into h cell (Z=0.5 in h cell);

=1 store h cell density counting from edge of cell when averaging into h cell (Z=0.0 in h cell).

Line 15 MXFULL MYFULL MZFULL IIHX IFHX IIHY IFHY IIHZ IFHZ (9i3)

NXFULL,NYFULL,NZFULL are the number of grid intervals in a,b,c (one less than the number of grid steps)

IIHX,IIHY,IIHZ is the first point in a,b,c that defines the volume for display in the h cell in standard orientation.

IFHX,IFHY,IFHZ is the last grid point that defines the volume. Note: This volume can be either the asymmetric unit of an icosahedron (for instance) or much of the h cell for re-defining the mask.

RELATION OF P CELL MOLECULES TO STANDARD ORIENTATION IN H CELL

Line 16

is repeated once for each particle (molecule) in the p cell crystallographic asymmetric unit related to each other by non-crystallographic symmetry. That means Line 16 is repeated IMPROP times.

if NTYPE=1 or 2 CAPA PSI FI (3f10.2)

polar or Eulerian angles in degrees that relate h to p cell

if NTYPE=3,4,5, or 6 ((A(I,J),J=1,3),I=1,3) (3f10.4)

read C,P,CINV, or PINV matrices

Where X=[RHO]Y

or x=[C]y=[ALPHA(h)][RHO][BETA(p)]y

NON CRYSTALLOGRAPHIC SYMMETRY IN H CELL (standard orientation)

Line 17 NCRYST (i3)

number of non-crystallographic symmetry elements necessary to generate all non-crystallographic asymmetric units (e.g., 4 for an icosahedron)

Line 18 (NCAPA(N),NPSI(N),NPHI(N),N=1,NCRYST) (3f10.4)

polar coordinates of the ncryst symmetry operators referred to the h cell (in

degrees). Thus the non-crystallographic
symmetry operators are defined with respect
to the standard orthogonal coordinate
system.

Line 19 (XPLORE(I),I=1,6),IREF (3f8.2,3f10.5, i2)
used in Option 20 (CLIMB)

Line 20 (DXPLORE(i),I=1,6) (3f8.2,3f10.5)
used in Option 20 (CLIMB)

A sample control input file:

```
# mode,group       (3i4):  1    16
1st op., #files    (2i4): 14     3
filenr, fname(i4,3x,a64):  1    /cfs/dancm/haodata
filenr, fname(i4,3x,a64):  2    /cfs/b4p/scratch
filenr, fname(i4,3x,a64):  3    /cfs/b4p/redbord
2nd op., #files    (2i4):  0
    255.00     255.00     801.00     90.00     90.00     90.00
  8   4   0   0   0   0   0   1  60   6   1
    0.4667     0.4667     0.0000
  1.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  1.00  0.00
 -1.00  0.00  0.00  0.00  0.00 -1.00  0.00  0.00  0.00  0.00  1.00  0.50
  0.00 -1.00  0.00  0.50  1.00  0.00  0.00  0.50  0.00  0.00  1.00  0.25
  0.00  1.00  0.00  0.50 -1.00  0.00  0.00  0.50  0.00  0.00  1.00  0.75
  0.00  1.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00 -1.00  0.00
  0.00 -1.00  0.00  0.00 -1.00  0.00  0.00  0.00  0.00  0.00 -1.00  0.50
 -1.00  0.00  0.00  0.50  0.00  1.00  0.00  0.50  0.00  0.00 -1.00  0.25
  1.00  0.00  0.00  0.50  0.00 -1.00  0.00  0.50  0.00  0.00 -1.00  0.75
260260800   1131   1131   14010.96
 2 1 1  -0.4
    300.00   300.00   300.00   90.00   90.00   90.00
    0.0000   0.0000   0.0000    75.0
 1 3 2   3.00 3 1
300300300-64 19 75144-48 75
0.707107   0.707107   0.
-.689122   0.689122   0.224101
0.158463  -0.158463   0.974566
  3
120.0000   69.0900   90.000
180.0000    0.0000   90.000
 72.0000   31.7200   90.000
```

OPTIONS

A. Mask Generation Options:

Option 1. generates mask based on intersecting spheres.

Option 2. reads h cell electron density (the particle is in a standard orientation in a large h cell) for determining a new mask. If ICELL = 0 the program will read the header of the p cell density otherwise it will use the grid given in Line 6.

Option 3. Enlarge or shrink the mask and check on non-crystallographic symmetry. The program averages the mask generated by Options 1 or 2. The mask is initially all filled with density = 100.0. Thus averaged points that remain at 100 are completely within the mask. Points that average to 0 are completely outside the mask. Points that average to a value larger than LXTND will be made into protein if that is not already the case. Points that average to less than LXTND will be regarded as outside the mask. Hence LXTND = 50 is neutral and the effect is only to impose non-crystallographic symmetry. If LXTND<50 then mask will be shrunk, etc.

Option 4. Fills in holes in mask (routine ENVIRON). See value of IFILL in Line 3.

Option 14. Find the non-crystallographic asymmetric unit for mapping back into this reduced asymmetric unit from outside. When this has been done, then averaging can be accomplished either by going to Option 9, as always, or Options 11 or 15. Using Option 9 implies averaging over the whole crystallographic asymmetric unit. The use of Option 11 involves two steps: First the non-crystallographic asymmetric unit plus a suitable surrounding border is averaged. Then the points outside the non-crystallographic asymmetric unit are folded back into the crystallographic asymmetric unit. This is much faster than Option 9 but may be less accurate due to the double interpolation. The use of Option 11 is a one-step operation where the density inside the non-crystallographic asymmetric unit is averaged and the density outside is folded back into the non-crystallographic asymmetric unit for finding the density of the previous cycle.

B. Options required to average the p cell density and put the result back into the p cell:

Option 7. reads Lines 14 and 15, giving the non-crystallographic symmetry in the cell for generating all the necessary matrices for averaging into the p cell. If this had already been done in another option, then the reading will be skipped, but matrices will continue to be generated. If Option 4 was previously executed, it is not necessary to use this option.

Option 8. stores the p cell electron density along with the mask. If `IPRNT` = 2, then this yet unmodified density, is written on the output file.

Option 9. modify the p cell density within the defined limits (usually the asymmetric unit) and write the results with `WRITE3D` onto `MODFILE`. See Line 8 for the various modifying possibilities.

Option 10. Takes the modified density, reads it back into storage, along with the mask and generates the whole of the unit cell file for back transformation (also brick format).

Option 11. This option averages into the p cell, using the fast fold back procedure described for Option 14. However it does things in two stages (as opposed to one stage in Option 14), as follows:

1. If the non-crystallographic asymmetric unit has not yet been defined, go through `CALL REDAU` (to avoid this go first through Option 14)
2. Average all the points in the non-crystallographic asymmetric unit and write out the results with `WRIT3D`
3. Read back the map into the mask by calling `STORMAP`. Then fold the points outside the non-crystallographic asymmetric unit into the already averaged non-crystallographic asymmetric unit for interpolation.

Note that by using Options 14 and 9, the Step 3 above is always one cycle behind, as the interpolation will be with respect to the old density in the non-crystallographic asymmetric unit.

C. Options required to average the p cell but store results in the standard orientation in the h cell:

These are useful either for obtaining a better mask or for display of one non-crystallographic asymmetric unit in a standard orientation.

Option 12. averages the p cell density (see ISWPROT in Line 8) within a radius RADOUT from CENTERH, and puts it into the h cell. Reads Lines 14 and 15 giving the non-crystallographic symmetry in the cell for generating all the necessary matrices for averaging the p cell into the h cell. If IPRNT = 8 prints modified density in h cell (test phase).

D. Additional options

Option 5. Transforms mask/density file for displaying

Option 6. Transforms a p-cell mask/density file (if input parameter CAY is 1.0), or an h-cell mask/density file (if input parameter CAY is 2.0), from format with 8K header and non-minimal brick padding, to format with 64K header and minimal brick padding.

Option 13. This puts density from the h cell back into the p cell mask. This is useful if the averaged h cell density is to be taken as a starting model in the p cell.

Option 16. Transforms a mask/density file from brick to slab format, in preparation of the FFTINV program. In addition to the control input information read by the INFO routine, the control input file requires a last line with the special options, called debug flags, as described for Line 19 of the control input file for the FFTINV program. (e.g. 'input of debugflags: 1 0 0 0 0 0 0 0 0 0')

Option 20. explores parameters (three positional and three orientational) of one particle to maximize the height (e.g., of heavy atom) IREF = 1 or minimizes the scatter (IREF = 0) XPLORE has THETA1 THETA2 THETA3 CENTER(X) CENTER(Y) CENTER(Z) at start, in degrees and fractional coordinates respectively. DXPLORE contains the increment or decrement of each parameter for exploration

DATA INPUT AND DATA OUTPUT FILES

Option 1: Data Output: p-cell mask

Option 2: Data Input: h-cell density
Data Output: p-cell mask

Option 3: Data Input : p-cell mask
Data Output: p-cell mask

Option 4: Data Input: p-cell mask
Data Output: p-cell mask

Option 5: Data Input: mask/density
Data Output: mask/density, planes

Option 6: Data Input: mask/density, old header
Data Output: mask/density, new header

Option 8: Data Input (1): `FFTSYNTH` output
Data Input (2): p-cell mask
Data Output: p-cell mask/density

Option 9: Data Input: p-cell mask/density
Data Output: p-cell mask/density

Option 10: Data Input: p-cell mask/density
Data Output: p-cell mask/density

Option 12: Data Input: p-cell mask/density
Data Output: h-cell density

Option 13: Data Input (1): h-cell density
Data Input (2): p-cell density
Data Output: p-cell mask/density

Option 16: Data Input: mask/density
Data Output: slabs for `FFTINV`

Option 20: Data Input: p-cell mask/density
Data Output: h-cell density

EXAMPLES of various option combinations:

a.	Create a mask from the h cell density:	2 0
b.	Read a mask and enlarge:	3 0
c.	Create a mask from the h cell density and average within the p cell: Note: if the mask is stored after this then the storage would be of both the mask and the pre-averaged density. See (e) below.	2 8 9 0
d.	Read a mask, read p cell density into mask, average within the p cell, and expand into the whole unit cell for back transformation:	6 8 9 10 0
e.	Perhaps the p cell density was stored previously with the mask. Then to average into the p cell and to expand into the whole cell, all that would be required would be:	6 9 10 0
h.	read the p cell density(PEDFILE) and put the averaged density into the h cell (output onto HOUTFILE):	8 12 0

Note: The grid used in the map read in by Option 8 overwrites whatever is given in Line 6. There is no purpose in defining a mask when averaging from the p cell into the h cell. This sequence is useful when

(i) obtaining a skew averaged map of the non-crystallographic asymmetric unit;

(ii) obtaining an averaged molecule (particle) in the h cell to create a new mask;

(iii) to search for the best particle position by looking for lowest average rms scatter of non-crystallographic equivalent points In this case use only the non-crystallographic asymmetric unit as in (i) above.

2. How to Run the ENVELOPE Program on the iPSC/860

Besides the sample control input file mentioned earlier, two other examples can be found on ipscgate.cc.purdue.edu or sampson.ccsf.caltech.edu, in ~cryst/PROGS/SCRIPT/envelope.inj_9 (for Option 9), and in ~cryst/PROGS/SCRIPT/envelope.inj_89 (for Option 8. followed by Option 9). To use any of these, the /cfs output directories would have to be changed to a place where permission to write files exists.

Steps:

(a) Assuming the (appropriately modified) input file envelope.inj_9 exists in the directory from which the user wants to run e.g., Options 9 on 16 nodes, this can be achieved as follows:

(b) type, on srm.cc.purdue.edu, or on ajax.ccsf.caltech.edu:

```
getcube -c mycube -t16 > myoutfule
load -c mycube -H -cryst/Pprog/i860/Envelope/v1/1.3/ENVELOPE
startcube ; waitcube < envelope.inj_9
```

The output will then go to the file 'myoutfile'. The file 'envelope.inj_9' is the control input file. The '-H' option of 'load' does not let the program start until the 'startcube' command is given. The 'waitcube' command may be given on a separate line, too, but then some output may appear on the screen before the user can type 'waitcube', if it is not redirected in the 'getcube' command. To find out how to use the control input file when running multiple cycles, type:

```
man cycle
```

3. How to Run the ENVELOPE Program on the PARAGON

On the Paragon there is no host (front-end). You login to a service node (a node connected to the network) and you need to specify:

(a) The program you want to execute and the environment for your run

- the program name
- the SIZE of the partition
- the name of the partition (default .compute)
- the priority (optional)
- the option to fix communication buffers in memory ("plk").

(b) The input control file.

(c) The output control file.

Example:

```
   PROGRAM < CONTROL_INPUT > CONTROL_OUTPUT
   /home/sampson6/cryst/Prog/Paragon/Envelope/V1/n6/cryst/Prog/Paragon/Envelope/V1.1/ENVELOPE
-sz 32 -pn .compute -pri 10 -p1 k <
   /home/sampson6/dcm/TEST1.0/opt1/env_opt1.in_paragon >
   /home/sampson6/dcm/TEST1.0/opt1/env_opt1.out_paraton
```

Please report BUGS to:

dcm@cs.purdue.edu
zz@cs.purdue edu

E2 ENVELOPE FILE FORMAT

Related documents
- E1. Envelope – User's Guide
- E3. Envelope – Control information precedence
- E4. Envelope – Internals

Last update: dcm, January 12, 1994

Each file cosists of a 64 K header followed by binary data.

file header (64 K bytes	binary data

There are two types of file produced by the Envelope:

- the brick format
- the section by section format

Different options of the Envelope use the following files:

Option 1 :	Data Output — p-cell mask	(Brick format)
Option 2 :	Data Input — h-cell density	(Brick format)
	Data Output (1) - p-cell mask	(Brick format)
Option 3 :	Data Input — p-cell mask	(Brick format)
	Data Output (1) — p-cell mask	(Brick format)
Option 4 :	Data Input — p-cell mask	(Brick format)
	Data Output : p—cell mask	(Brick format)
Option 5 :	Data Input — mask/density	(Brick format)
	Data Output — mask/density, planes (not yet implemented)	(ASCII format)
Option 6 :	Data Input — mask/density, old header	(Brick format)

	Data Output — mask/density, new header	(Brick format)
Option 8 :	Data Input (1) — FFTsynth output	(Section by Section Format)
	Data Input (2) — p-cell mask	(Brick format)
	Data Output — p-cell mask/density	(Brick format)
Option 9 :	Data Input — p-cell mask/density	(Brick format)
	Data Output (1) — p-cell mask/density	(Brick format)
Option 10 :	Data Input — p-cell mask/density	(Brick format)
	Data Output — p-cell mask/density	(Brick format)
Option 12 :	Data Input — p-cell mask/density	(Brick format)
	Data Output (1) — h-cell density	(Brick format)
Option 13 :	Data Input (1) — h-cell density	(Brick format)
	Data Input (2) — p-cell density	(Brick format)
Option 16 :	Data Input — mask/density	(Brick format)
	Data Output — slabs for FFTINV	(Section by section format)
Option 20 :	Data Input — p-cell mask/density	(Brick format)
	Data Output (1) — h-cell density	(Brick format)

This document describes:

A. The format of the header, the same for both types of files.

B. The brick format.

C. The section by section format.

1. THE HEADER FORMAT

Each file header produced by the Envelope program is 64 Kbytes long. The information written in the file header comes from:

- the control input file
- the information in the file headers of the input data files.

(see E3 for details).

The programs responsible for header processing are:

decodehead.f - it reads an input file header
encodehead.f - it writes out the output file header

1. System information

Data item	Length (bytes)	Offset (bytes)
Date of file creation (for this file)	8	0
Time of file creation (for this file)	8	8
Machine name (e.g. srm.cc.purdue.edu)	64	16
Operating system (e.g. UNIX System V)	32	80
Machine architecture (e.g. iPSC/860)	32	112
Test byte (for little/big endian)	4	144
Next free byte		148

2. Information about the program which created the file and about the file

Data item	Length (bytes)	Offset (bytes)
Program name (ENVELOPE)	16	208
Program version	8	224
Full path name of the executable	64	232
Date when executable was created	8	296
Header length (64K)	4	304
File length (for this file)	4	308
Comment	100	312
Next free byte		412

3. Information about the physical problem

Data item	Length (bytes)	Offset (bytes)
Cell	24	512
Reciprocal cell	24	536
nsym	4	560
sym(i,j,k)	1152	564
ntype	4	1716
improp	4	1720
center(i)	48	1724
centerh(i)	12	1772
radout	4	1784
radin	4	1788
corad	4	1792
Matrix relating p to h cell	144	1796
imask	4	1940
Next free byte		1944

4. Information about the run.

Data item	Length (bytes)	Offset (bytes)
nx	4	2000
ny	4	2004
nz	4	2008
iix	4	2012
ifx	4	2016
iiy	4	2020
ify	4	2024
iiz	4	2028
ifz	4	2032
naxis() (ipa,ipb,ipc)	12	2036
resmin	4	2048
resmax	4	2052
hmax, kmax, lmax	12	2056
ms	4	2068
ls	256	2072
ctinv	3072	2328
tinv	2304	5400
s	3072	7704
Next free byte		10776

Note: The header of an input file created by another program (for example the header of the section by section file produced by FFTsynth and used as input in option 8) differs from the header described above starting at displacement 2048 as follows:

edmin	4	2048
edmax	4	2052
Next free byte		2056

2. The Brick Format

2.A. Overview

The brick format is the standard input and output data format for the Envelope. It is designed to store in a compact format the information about a "dense" 3-D mesh.

The brick file is manipulated by a set of primitives in the program "pbr.c". These primitives allow concurrent access to the brick file from multiple nodes (see E4 for details).

2.B. The 3-D mesh

Given a 3-D mesh of $(na) \times (nb) \times (nc)$ grid points, from iix to ifx in the x-direction, from iiy to ify in the y-direction and from iiz to ifz in the z-direction (see infoprocess.f):

$$\begin{aligned} na &= ifx - iix + 1 \\ nb &= ify - iiy + 1 \\ nc &= ifz - iiz + 1 \end{aligned}$$

For each grid point the information is packed into two bytes:

- 10 bits electron density and
- 6 bits mask.

A brick is a 3-D volume consisting of $(bx) \times (by) \times (bz)$ consecutive grid points. Currently $bx = by = bz = 16$. There are 4096 grid points within a brick and the storage space occupied by a brick is 8K bytes. The grid points within a brick are stored in the $x \rightarrow y \rightarrow z$ order.

We can look at the bricks as a 3-D structure consisting of layers of bricks stored in the $x \rightarrow y \rightarrow z$ order. Call $ibrick$, $jbrick$, $kbrick$ the number of bricks in the x, y and z direction respectively. The values $ibrick$, $jbrick$, $kbrick$ are computed as follows: (see infoprocess.f):

$$\begin{aligned} ibrick &= ((na-1)/bx) + 1 \\ jbrick &= ((nb-1)/by) + 1 \\ kbrick &= ((nc-1)/bz) + 1 \\ ijbrick &= ibrick * jbrick \end{aligned}$$

The "brick reference point" is the brick point closest to the origin of the 3-D brick space. Given the brick with 3-D coordinates $(ibid, jbid, kbid)$ its reference point has the 3-D coordinates $(istart, jstart, kstart)$ given by (see modpmap.f):

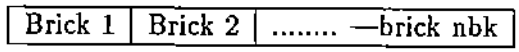
$$\begin{aligned} istart &= bx * (ibid - 1) + iix \\ jstart &= by * (jbid - 1) + iiy \\ kstart &= bz * (kbid - 1) + iiz \end{aligned}$$

The relation between local and global coordinates of a grid point. Assume that you are given (i, j, k) the local coordinates of a point in brick bid with the reference point $(istart, jstart, kstart)$. Then its global coordinates $(iglobal, jglobal, kglobal)$ are given by:

$$\begin{aligned} iglobal &= i + istart - 1 \\ jglobal &= j + jstart - 1 \\ kglobal &= k + kstart - 1 \end{aligned}$$

2.C. The relation between 3-D coordinates of a brick and its displacement in the file, a 1-D structure.

The bricks are stored sequentially in the $x \rightarrow y \rightarrow z$ order.



The brick with 3-D brick coordinates *ibid*, *jbid*, *kbid* is stored sequentially as brick *bid* with:

$$bid = (k-1) * ibrick * kbrick + (j-1) * ibrick + i.$$

The first layer of bricks ($z = 1$) contains *ibrick* $\times$ *jbrick* bricks:

```
(1,       2, ...                                  ibrick),
(ibrick+1, .....                                2*ibrick),

.....................................................

(ibrick*(jbrick-1)+1).............      jbrick*ibrick)
```

Succesive layers of bricks up to *kbrick* layers are The total number of bricks is:

$$nbr = ibrick * jbrick * kbrick$$

The code to determine the 3-D coordinates of brick *bid* follows: (see modpmap.f):

```
C     Find the brick layer kbid  in which bid is
      kbid = bid/ijbrick
      if((bid - kbid*ijbrick).gt.0) kbid = kbid + 1
C     Find position of brick in the layer
      ijbid = bid- (kbid-1)*ijbrick
      jbid = ijbid/ibrick
      if((ijbid - jbid*ibrick).gt.0) jbid = jbid + 1
      ibid = ijbid - (jbid-1)*ibrick
```

Note: The brick format is used to store both the p-cell (which contains either the mask or the electron density and the mask) and the h-cell which contains only the electron density. For the h-cell the computation of the mesh geometry is similar to the one described above for the p-cell.

mx=ifhx-iihx+1
my=ifhy-iihy+1
mz=ifhz-iihz+1

ibrickh = (mx - 1) / bxh + 1
jbrickh = (my - 1) / byh + 1
kbrickh = (mz - 1) / bzh + 1

ijbrickh=ibrickh*jbrickh

nbrh=ibrickh*jbrickh*kbrickh

3. The section by section format

This data format is used to export results produced by Envelope to FFTinv (option 16) or to import from FFTsynth new electron density data (option 8).

3.A. The format produced by option 16 for the FFTinv

The data is written out in y-slabs.

Slab 1	Slab 2	.....	Slab ny

An y-slab is a group of consecutive xz-planes. For example the slab with

slab number slannr=10
slab width slabw = 5

consists of the planes 46, 47, 48, 49 and 50. Each slab is written out as:

slab number (an integer)
slab width (an integer)
first plane (nx*nz floating point numbers)
second plane
.....
last plane (nx*nz floating point numbers)

The total size of a slab in bytes is $4 * (nx * nz + 2)$.

The actual writing of the slabs is done in makeyslb.f.

```
subroutine makeyslb(slabnr, slabw, buff, buffptr)
```

```
      idispl = buffptr
      mbp = ((buff - \%loc (b)) / 4) - 1
      b(mbp+idispl)   = slabnr
      b(mbp+idispl+1) = slabw
      idispl = idispl + 2

      npresnt=1

      do 10 j=1,slabw
      jpt(2)=(slabnr-1)*slabw + j - 1

C     cycle over points in  the section (xz plane)
           do 11 i =  1,nx
             jpt(1) = i-1
             do 12 k =  1, nz
               jpt(3) = k-1
 21            do 13 n=npresnt,nsym
                 do 14 ii=1,3
                    kpt(ii)=ifix(sym(n,ii,4)*float(nxyz(ii)))
                    do 15 jj=1,3
                     kpt(ii)=kpt(ii)+
     >                        ifix(sym(n,ii,jj)*float(jpt(jj)))
 15                 continue
 17                 if(kpt(ii).ge.0) go to 16
                    kpt(ii)=kpt(ii)+nxyz(ii)
                    go to 17
 16                 if(kpt(ii).le.nxyz(ii)) go to 18
                    kpt(ii)=kpt(ii)-nxyz(ii)
                    go to 16
 18                 ipt(ii)=kpt(ii)+1
 14              continue
C
C Test that the point with coordinates ipt(1),ipt(2),ipt(3)
C is within the assymetric unit. If YES go to 19.
C
      if((ipt(1).ge.iix).and.(ipt(1).le.ifx).and.
     >   (ipt(2).ge.iiy).and.(ipt(2).le.ify).and.
     >   (ipt(3).ge.iiz).and.(ipt(3).le.ifz)) then

          go to 19
      endif
```

```
13            continue

              if(npresnt.eq.1) go to 22
              npresnt=1
              go to 21
22            write(6,99) i,j,k
99            format(' point cannot be placed ',
     >               ' into asym. unit ',3i4)
              b(mbp + idispl) = 0.0
              idispl = idispl + 1
20       npresnt=n

12            continue
            b(mbp+idispl) = 0.0
            b(mbp+idispl+1) = 0.0
            idispl = idispl + 2
11         continue
10    continue
```

3.B. The format produced by FFTsynth

The file consists of a sequence of planes. The reading of the planes and filling in of the electron density in bricks is done in stormap.f.

```
C
      secsz   = nx * nz * intiglen
      slabsz  = ibrick * kbrick * bbsize
      maxmem  = 10500000
      if ( (secsz + slabsz).gt.maxmem ) then
        write(0,*)' >> NODE ', iam,
     > ' (stormap): insufficient memory to store a slab of ',
     > ' size: ', slabsz, ' and a plane of size: ', secsz,
     > ' The program will terminate '
        call node_error
        stop
      endif

      nslabs = jbrick
      nsteps = nslabs /nnodes
      if (nslabs .ne. nsteps * nnodes) nsteps = nsteps + 1
      maxoffset = (ny - 1) * secsz + bbsize*nbrhead
```

```
C
C     Read header
C

      call getplane(ioffset, edmin, edsize, 1)
                     ioffset is the displacement in the input file
                            ( ioffset=2048, at displacement 2048 in the
                              header you find edmin. See the header format.)
                     edmin   is the address of the buffer
                     edsize  is the length (8)
      scale1 = 511.0/max(abs(edmax), abs(edmin))
      if (iam .eq. 0) then
      write(6, 88) edmin,edmax,scale1
 88   format(5x,'Information from the Fftsynth '/
     > ,      'Minimum  electron density value  =',1pe19.7/
     > ,      'Maximum  electron density value  =',1pe19.7/
     > ,      'SCALE                            =',1pe19.7)
      endif

C      print *,'Debug-Node:',iam,'edmx,edmn=',hsec(1),hsec(2)
C      print *, ' NODE', iam, ' scale = ', scale1
C      if (iam.eq.0)   print *, ' NODE', iam, ' scale = ', scale1


C
C      Process data in slabs.
C      ----------------------------------------------
C      Bricks are numbered:   1 to nbr
C      Sections are numbered: 1 to ny
C      Slabs are numbered:    1 to nslabs.
C
C

      do 10 istep = 1, nsteps
        myslab = iam + 1 + (istep -1) * nnodes
        if (myslab.gt.nslabs) go to 10
        do 20 iplane = 1, by

          isection = iplane + (myslab-1) * by
          if(isection.gt.ny) isection=isection-ny
          if((isection.lt.iiy).or.(isection.gt.ify)) go to 20
```

```
      ioffset = (isection - 1)*secsz + bbsize*nbrhead
      if (ioffset .gt. maxoffset) then
          write(0,*)'>> NODE (stormap) ',iam,
     >      ' invalid offset',ioffset,maxoffset
          call node_error
          stop
      endif
C     print *,' NODE', iam,
C    >       ' slab: ', myslab, '  section: ', isection,
C    >       '  offset: ',ioffset
      call getplane(ioffset,hsec,secsz,1)


      do 30 kbk = 1,kbrick
C       print *, ' NODE',iam,' *** kbk', kbk
        do 40 ibk = 1,ibrick
C         print *, ' NODE',iam,' *** ibk', ibk
          nnb = (myslab - 1) * ibrick
     >             + (kbk -1)  * ijbrick + ibk
          if (nnb .gt. nbr) then
              write(0,*)'>> NODE (stormap) ',iam,
     >          ' invalid brick number', nnb, nbr
              call node_error
              stop
          endif

C         print *, ' kbrick, ibrick, brick#: ', kbk, ibk, nnb
            do 60 kk = 1,bz

              kglobal = kk + bz * (kbk - 1)
              if(kglobal.gt.nz) kglobal=kglobal-nz
              if((kglobal.lt.iiz).or.(kglobal.gt.ifz)) go to 60
C             print *, ' NODE', iam, ' kglobal:', kglobal

             do 70 ii = 1,bx

              iglobal = ii + bx * (ibk - 1)
              if(iglobal.gt.nx) iglobal=iglobal-nx
              if((iglobal.lt.iix).or.(iglobal.gt.ifx)) go to 70
C             print *, ' NODE', iam, ' iglobal:', iglobal
```

```
                rrho = hsec(iglobal+(kglobal-1)*nx) * scale1
                itest = (iglobal+(kglobal-1)*nx)
                if(rhomax.lt.rrho) rhomax=rrho
                if(rhomin.gt.rrho) rhomin=rrho
                if(rrho.ge.512.0)  then
                     print *,  '>> NODE ',iam,
     x               ' (stormap) WARNING - ',
     x               ' scale out of range in section :',isection,
     x               ' density:', rrho,
     x               ' x, z', iglobal, kglobal
                    rrho=511.0
                endif
                if(rrho.le.-512.0) then
                    print *,  ' NODE ',iam,
     x              ' (stormap) WARNING - ',
     x              ' scale out of range in section :',isection,
     x              ' density:', rrho,
     x              ' x, z', iglobal, kglobal
                    rrho=-511.0
                 endif


          call getentry(ii,iplane,kk,nnb,rhoold,mask,3)

          call putslave(ii,iplane,kk,nnb,rrho,mask)


20     continue
10    continue
```

E3. THE PRECEDENCE OF THE CONTROL INFORMATION

Option	1	2	3	4	8	9	10	12	13	20	5	6
What type of input	–	H	P	P	P	P	P	P	H	P	P or H	P
Data item					F				P			
What type of output	P	P	P	P	P	P	P	H	P	–	–	P
cell(2,i) (p cell)	I/u	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	–	P/nu	P/nu	P/nu	I/u
nsym	I/u	I/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	I/u
ntype	I/u	I/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	I/u
imask (set imask = - 1 for options 8,9,10,13,6)	–	I/u	P/nu	P/nu	P/u	P/u	P/u	–	P/u	–	–	–
istore	–	–	–	–	I/nu	–	–	–	–	–	–	–
iprnt	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	–	–	–
jprnt	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	–
icell	–	–	I/u	I/u	–	I/u	?	I/u	–	I/u	–	–
lxtnd	–	–	I/u	P/nu	P/nu	P/nu	P/nu	–	P/nu	P/nu	P or H/nu	–
ifill	–	–	P/nu	I/u	P/nu	P/nu	P/nu	–	P/nu	P/nu	P or H/nu	–
improp	I/u	I/u	P/u	P/u	P/u	P/u	P/nu	P/u	P/u	I/u	P/nu	I/u
incx, incy, incz (only if iprnt ≠ 0)	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	I/u	–	–	–
center(j,i)	I/u	I/u	P/u	P/u	P/u	P/u	P/nu	P/u	P/u	P/u	P/nu	I/u
radout	I/u	I/u	P/nu	–	–	–	–	I/u	–	I/u	–	
radin	–	I/u	P/nu	–	–	–	–	–	–	–	–	
corad	I/u	I/u	P/nu	–	–	–	–	–	–	–	–	
sym(i,j,k)	I/u	I/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	P/u	P/nu	I/u
nx, ny, nz	I/u	I/u	P/u	P/u	P/u	P/u	I/u,P/u	P/u	P/u	P/u	P/u	P/u
iix,...,ifz	I/u	I/u	P/u	P/u	P/u	P/u	I/u,P/u	P/u	P/u	P/u	P/u	P/u
scalep	–	–	–	–	I/u	–	–	–	–	–	–	
iiyav, ifyav	–	–	–	–	–	I/u	–	–	–	–	–	–
iswprot	–	–	–	–	–	I/u	–	I/u	–	–	–	–
iswsolv	–	–	–	–	–	I/u	–	I/u	–	–	–	–
iswacid	–	–	–	–	–	I/u	–	I/u	–	–	–	–
sclsolv	–	–	–	–	–	I/u	–	I/u	–	–	–	–
sclacid	–	–	–	–	–	I/u	–	I/u	–	–	–	–
cell(1,j)	–	H/u	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	H/u	–	H/nu	–
centerh(i)	–	H/u	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	H/u	–	H/nu	–
crit1, crit2	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	–	P or H/nu	–
scaleh	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	–	P or H/nu	–
naxis(i)	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	–	P or H/nu	–
smradsq	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	–	P or H/nu	–
jsmear	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	–	P or H/nu	–
jstore	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	–	P or H/nu	–
mxfull, myfull, mzfull	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	P/nu	–	P or H/nu	–
iihx,...,ifhz	–	I/u	P/nu	P/nu	P/nu	P/nu	P/nu	I/u	P/nu	–	P or H/nu	–
capa(i), psi(i), fi(i) [Line 16] or a(i,j) [Line 16]	–	I/u	P/u	P/nu	P/nu	P/u	P/nu	I/u	I/u	–	P or H/nu	–

Option	1	2	3	4	8	9	10	12	13	20	5	6
What type of input	–	H	P	P	P	P	P	P	H	P	P or H	P
Data item					F				P			
What type of output	P	P	P	P	P	P	P	H	P	–	–	P
ncryst	–	I/u	P/u	P/nu	P/nu	P/u	P/nu	I/u	I/u	–	P or H/nu	–
ncapa(i), npsi(i), nfi(i) [Line 18]	–	I/u	P/u	P/nu	P/nu	P/u	P/nu	I/u	I/u	–	P or H/nu	–
xplore(i)	–	–	–	–	–	–	–	–	–	I/u	–	–
iref	–	–	–	–	–	–	–	–	–	I/u	–	–
dxplore(i)	–	–	–	–	–	–	–	–	–	I/u	–	–
Option	1	2	3	4	8	9	10	12	13	20	5	6

The control information used by different options of the Envelope program may come from: (a) the control input file. (b) the headers of the data files (p-cell, h-cell, FFTsynth output). This table indicates for every single parameter wether it is used or not by the corresponding option of the Envelope. It also gives the precedence rules when conflicting information is gathered from the different sources described above.

LEGEND

I	:	control input file	–	:	irrelevant
P	:	p-cell header	u	:	used
H	:	h-cell header	nu	:	not used
F	:	map-file header, from FFTsynth			

E4 ENVELOPE IMPLEMENTATION NOTES

Related documents

E1. Envelope — User's Guide

E2. Envelope — file format

E3. Envelope — control information precedence

Last update: dcm, January 14, 1994

This document describes the implementation of the Envelope program for a distributed memory MIMD (Multiple Instruction, Multiple Data) system, DMIMD, like the iPSC/860 and Paragon.

1. Overview

A DMIMD consists of a number of PEs (Processing Elements) each consisting of a processor and local memory, interconnected by a 2-D mesh (the Paragon) or a hypercube (i860). Each PE can only process data in its local memory and may communicate with other nodes by explicitely sending and receiving messages.

The main problem in designing algorithms and programs for DMIMD systems are data partitioning and load balancing. Data partitioning means to decide what data elements are stored in each node and work allocation means to decide on the work carried out by every node. The concepts of "data allocation unit" and "work allocation unit", called "bricks", as well as the algorithms used are outlined in [2] and [3]. The basic idea is to allocate "master bricks" for processing according to a strategy ensuring load balance and then for a master brick to bring in all the "slave bricks" necessary to carry out the computation for all the grid points.

Important observation: most problems on 3-D meshes use some form of geometric partitioning of the mesh. This does not work in case of electron density averaging due to the nature of the computation namely points related by non-crystallographic symmetry are scattered throughout the entire volume. For this reason we have to implement a Shared Virtual Memory.

2. Inter-node Communication

Nodes communicate with one another using messages. There are three types of messages:

- synchronous messages (csend, crecv)
- asynchronous messages (isend, irecv)
- interrupt driven messages (hsend, hrecv)

See Paragon Fortran or C System Call Manuals for the syntax and semantics of communication system calls.

3. The Programming Model

We use a quasi-SPMD (Same Program Multiple Data) programming model. All nodes run the same program with different data, namely the data allocated to them by the data partitioning and load distribution mechanism. Yet one node, the "master" (typically node 0) performs some control functions not performed by other nodes. Examples of control functions performed by the "master".

- reading of the control input data
- reading of the file header
- computation of some initial values of parameter
- the "master" side of the "intial exchange". The master node node sends control information and parameters to all "slave" nodes.
- receiving of partial results at the end of the computation.

A slave node starts by executing the "slave" side of the "initial exchange". It starts by receiving the control information and other parameters necessary for computation from the "master".

The work allocation and the data distribution is done in a distributed fashion. All nodes execute the same work allocation and data distribution algorithm and each decides what data and work is assigned to it.

The "iam=mynode()" system call allows a node to discover its own identity.

4. Data Files

All files are binary files. All binary files are linear strings of bytes. An access routine need only know the "offset", the "length" of the data block to be read or written and the "buffer" where the data block is read from (in case of a "write" operation) or where the data will be read into (in case of a "read" operation).

All data files are accessed through an I/O driver called "pbr.c". This driver allows direct and parallel file access from multiple nodes at the same time. Direct access (as opposed to sequential access) allows access to to an item stored at any given "offset" from the beginning of the file. Parallel access means that several nodes may access the file at the same time.

There are three basic execution modes:

(DFS) The input data space is stored on a Disk File System (CFS for i860 and PFS on the Paragon) and brought in each node as the execution needs dictate.

(DAN) The input data space is distributed accross nodes (the data is cached in the local storage of all nodes). Each node has a copy of the "brick directory" and is able to locate where a certain brick is and then to request it from that node. This mode relays on "interrupt driven communication", (hsend, hrecv messages).

(DS) The input bricks are all stored in a few nodes (Data Servers) which do not carry out any computation and all compute nodes request data from the Data Servers.

The DAN and DS modes are more efficient than the CFS mode.
Note: The DAN mode requires "interrupt driven messages" (hsend/hrecv).

5. Error Conditions

Whenever a node terminates due to an error condition it calls "node_error" and sends an "interrupt_driven_message" to all other nodes to stop the entire computation immediately. Without this feature the failure of a single node would cause the entire partition to hang till an external agent (the user, the operator or the system job scheduler) would stop the partition.

6. I/O Functions performed by the I/O driver pbr.c

FUNCTIONS FOR OPENING AND CLOSING FILES

PBROPEN: opens the file whose name is the (first part of the) string 'name' (up to the first space character); the file descriptor is fd[*fnr-1] if mode *equiv* "RD", the file is opened only for reading if mode *equiv* "RW",the file is open for reading and writing; if the file doesn't exist, node 0 also creates it; the file is open for reading and writing.

```
PBROPEN_(name, fnr, mode)
 char          *name;
 long          *fnr;
 char          *mode;
```

PBRCLOSE closes file fnr → file number (1 to 6 in Fortran, 0 to 5 in C).

```
PBRCLOSE_(fnr)
 long          *fnr;
```

FUNCTIONS FOR BRICK MANIPULATION

GETBRICK: brings the brick with virtual number bvn, from the file with descriptor fd[*fnr-1], into the array master[] ; involves I/O fnr → file number (1 to 6 in Fortran, 0 to 5 in C).

```
GETBRICK_(bvn, fnr)
 int           *bvn;
 long          *fnr;
```

GETDMBRK: for the CFS version : makes the previous master brick DISPosable if the new master brick is not already here, brings it in (synchronous call to cbring) ; if the next master brick (in 'snake' order) is not already here, brings it in (asynchronous call to ibring); sets mbvn & mbrn and also nextmbvn & nextmbrn (mbrn & nextmbrn show where to find the bricks in memory) ; involves

I/O fnr → file number (1 to 6 in Fortran, 0 to 5 in C).
GETDMBRK_(bvn, fakebvn, fnr)
```
int             *bvn, *fakebvn;
long            *fnr;
```

GETMMBRK: for the DAN version : if the new master brick is not already here, brings it in (synchronous call to cbring1); sets mbvn & mbrn (mbrn shows where to find the brick in memory) ; does not involve I/O.
GETMMBRK_(bvn)
```
int             *bvn;
```

PUTMBRICK: for Options 1, 2, 3, 4, 9, 10, 11, 13, 14 and 15 : for both CFS and DAN, write the new master brick from new_mbrick, into the output data file in the cfs; this is a synchronous write — it returns only after the new master brick has been written out; makes the brick DISPosable, if it is not to be stored permanently by the node, if the DAN option is used for Option 12, for both CFS and DAN, write the h-cell master brick from new_mbrickh into the h-cell output data file in the cfs; this is a synchronous write - it returns only after the new master brick has been written out ; no brick has to be made DISPosable, since the h cell master brick has not been brought from an input file fnr → file number (1 to 6 in Fortran, 0 to 5 in C).
PUTMBRICK_(bvn, fnr)
```
int             *bvn;
long            *fnr;
```

WRITEOUTSLAVE: writes into the output data file in the cfs, all the bricks currently in memory fnr → file number (1 to 6 in Fortran, 0 to 5 in C).
WRITEOUTSL_(fnr)
```
long            *fnr;
```

CLEARMBRK: makes DISPosable the brick (frame) which contains the brick with the virtual number bid ; it is assumed that this brick is (already) stored in memory (as for master bricks, for ex.).
CLEARMBRK_(bid)
```
int             *bid;
```

COPYMBRK: Copies master brick master[] to new master brick new_mbrick[].
COPYMBRK_()

FUNCTIONS TO READ AND WRITE PLANES OF DATA

GETPLANE: get a plane

offset	→	offset in the disk file (in bytes)
buff	→	the address of the buffer containg the data
length	→	the length of the data to be read (in bytes)
fnr	→	file number (1 to 6 in Fortran, 0 to 5 in C)

GETPLANE_(offset, buff, length, fnr)

```
int             *offset, *buff, *length;
long            *fnr;
```

PUTPLANE: writes a plane to the file with file descriptor fd[*fnr - 1]

ffset	→	offset in the disk file (in bytes)
buff	→	the address of the buffer containg the data
length	→	the length of the data to be written (in bytes)
fnr	→	file number (1 to 6 in Fortran, 0 to 5 in C)

PUTPLANE_(offset, buff, length, fnr)

```
int             *offset, *buff, *length;
long            *fnr;
```

FUNCTIONS FOR ENTRY MANIPULATION

MGETENTRY: extract the electron density and the mask for the point (i, j, k) in the current master brick.

MGETENTRY_(i, j, k, fden, mask)

```
int             *i, *j, *k;
unsigned        *mask;
float           *fden;
```

GETENTRY: For options other than 8 and 14 : extracts the density and the mask for the point (i, j, k) in the brick with virtual number bvn (brings in the brick first, if it is not already here) and returns them in fden and mask.

For Options 8 and 14 : if the brick vith virtual number bvn is not here, brings it in from the output data file in the cfs ; if the corresponding brick frame was containing a valid brick, writes that one to the output data file in the cfs ;extracts the density and the mask for the point (i, j, k) in the brick that has been brought in, and returns them in fden and mask.

fnr → file number (1 to 6 in Fortran, 0 to 5 in C).

GETENTRY_(i, j, k, bvn, fden, mask, fnr)

```
int             *i, *j, *k, *bvn;
unsigned        *mask;
float           *fden;
long            *fnr;
```

MPUTENTRY: places the density and the mask for the point (i, j, k) in the new master brick.
MPUTENTRY_(i, j, k, fden, mask)

```
int             *i, *j, *k;
unsigned        *mask;
float           *fden;
```

PUTSLAVE: writes the density and the mask from fden and mask respectively, corresponding to point (i, j, k) from the brick with virtual number bvn, into this brick ; the brick should be in memory (error, otherwise).
PUTSLAVE_(i, j, k, bvn, fden, mask)

```
int             *i, *j, *k, *bvn;
unsigned        *mask;
float           *fden;
```

FUNCTIONS FOR MANIPULATING THE FILE HEADER

GETHEAD: this function is called only by node 0 ; reads into array rec1[], the first record from the input data file in the cfs ; this is a synchronous (blocking) call fnr → file number (1 to 6 in Fortran, 0 to 5 in C).
GETHEAD_(rec1, fnr)

```
char            *rec1;
long            *fnr;
```

PUTHEAD: This function is called only by node 0; writes the first record from the input data file (in the cfs), from the array rec1[] into the output data file in the cfs, also, writes the current date and time into the header the file; it is a synchronous (blocking) call fnr → file number (1 to 6 in Fortran, 0 to 5 in C).
PUTHEAD_(rec1, fnr)

FUNCTIONS FOR DETERMINING THE BRICKS TO BE STORED AND/OR PROCESSED BY EACH NODE

COUNTH: function to record the number of points in the h-cell bricks, situated at a distance larger than radout from the center of the h-cell.
COUNTH_(bid, hcnt)

```
int             *bid, *hcnt;
```

BRICKLIMS: for Options 3, 9, 11, 14, 15 , with order ≡ 0 (call preceding pcount) : all nodes compute the limits for the p-cell bricks to be processed by each node (Array range[][]) ; the bricks are evenly distributed to the nodes if imode ≡ 1 (only in the DAN mode) : node 0 computes the limits for the p-cell bricks to be stored by the nodes, (array bound[][]), and the node on which each brick can be found, given by its relative number inside the group of nodes it belongs to ; node 0 then sends these two tables to all the other nodes, which receive them ; all nodes call brickdistr() to bring in the bricks they have to store fnr → file number (1 to 6 in Fortran, 0 to 5 in C)

For Options 3, 9, 11, 14, 15 , with order ≡ 1 : node 0 computes the limits for the bricks to be processed by eachI node (array range[][]), node 0 then sends this table to all the other nodes, which receive them ; bstart and bend get the limits of the range of bricks to be processed **Note:** the first integer in pbrcnt[] represents the total number of protein points to be processed ; the following ones are the numbers of protein points in each brick (the array may not be entirely used)

For Options 2 and 13 (should only be called with order ≡ 1) : all nodes compute the limits for the p-cell bricks to be processed by each node (array range[][]) ; the bricks are evenly distributed to the nodes all nodes compute the limits for the h-cell bricks to be stored by each node (array bound[][]) ; the bricks are evenly distributed to the nodes

For Option 4 (should only be called with order ≡ 1) : all nodes compute the limits for the p-cell bricks to be processed by each node (array range[][]) ; the bricks are evenly distributed to the nodes all nodes compute the limits for the p-cell bricks to be stored by each node (array bound[][]) ; the bricks are evenly distributed to the nodes

For Options 7 and 10 (the value of order does not matter) : all compute the limits for the bricks (p-cell or h-cell in Option 7, only p-cell in Option 10) to be stored by the nodes (array bound[][])

For Option 12 (should only be called with order ≡ 1) : node 0 computes the limits for the h-cell bricks to be processed by each node (array range[][], according to the relationship between radout, and the distances from the points in the h-cell to the center of the h-cell), then it computes the limits for the range of p-cell bricks to be stored by the nodes, (array bound[][]), and the node on which each p-cell brick can be found, given by its relative number inside the group of nodes it belongs to ; bstarth and bendh get the limits of the range of h-cell bricks to be processed ; for the DAN option, all nodes call brickdistr() to bring in the p-cell bricks they have to store fnr → file number (1 to 6 in Fortran, 0 to 5 in C).
BRICKLIMS_(nbr, nbrh, bstart, bend, PBRCNT_F, fnr, order)

```
int     *nbr, *nbrh, *bstart, *bend;
int     *pbrcnt_F;
long    *fnr;
int     *order;
```

FUNCTIONS FOR THE INTERRUPT MECHANISM USED TO TRANSFER BRICKS BETWEEN NODES

INITTRAP: posts hrecv-s for messages of length *len, from all the nodes in the group of igroup nodes.
INITTRAP_(iam, nnodes, len)
int *iam, *nnodes, *len;

ENDTRAP: send the value -1 ('shut' message) to all the nodes in the group of igroup nodes.
ENDTRAP_(iam, nnodes, len)
int *iam, *nnodes, *len;

FUNCTIONS USED TO IMPLEMENT THE 'SNAKE' ORDER

WIND: sets the 'snake' order for the bricks, in array real_index[] void.
WIND_(m, n, p)
int *m, *n, *p;

REALINDEX: returns in re_ind the real index (= virtual number) for the ind-th brick, in the 'snake' order void.
REALINDEX_(ind, re_ind)
int *ind;
int *re_ind;

FUNCTIONS FOR CONVERSION FROM BRICKS TO SLABS

YSLAB: Given a mesh of nx x ny x nz points and the amount of memory available in a node.
MAXMEM / 2 Compute:

plane size (bytes)	psize
slab width (number of planes the node can hold)	slabw
total number of slabs	nslab
slab size (byes)	slabsz
last slab width	lastsw
last slab size	lastssz
number of slabs that can be prepared by node 0	maxstack

PUTSLAB: number fnr

MISCELLANEOUS FUNCTIONS

INITADDRTR: INITADDRTR_()

COPYFILE: this function is called only by node 0; copies (using synchronous cread-s and cwrite-s) the input data file; in the cfs, into the output data file in the cfs.

```
COPYFILE_(fnr1, fnr2)
 long            *fnr1;
 long            *fnr2;
```

UPDATE: sets to 0 the usage counter for all non-master brick frames.

```
UPDATE_()
```

BRICKFLTS: prints the current numbers of brick faults and of master bricks; read/written.

```
BRICKFLTS_()
```

GETFLT: returns in 'total' the current number of brick faults (it is used to transmit this value from the C to the Fortran environment).

```
GETFLT_(total)
 int             *total;
```

SETMARK: transmits the mode and the group size from the Fortran to the C environment.

```
SETMARK_(mod, grp)
 int             *mod, *grp;
```

SETOPT: transmits the option number from the Fortran to the C environment.

```
SETOPT_(opt)
 int             *opt;
```

GETDATE: returns in tyme and date, the current time and date, in the format 23:59:59 10/31/92.

```
GETDATE_(tyme, date)
```

References

1. Rossmann, M. G., R. McKenna, L. Tong, D. Xia, J. Dai, H. Wu, H. K. Choi, and R. E. Lynch, "Molecular replacement real-space averaging," *J. Appl. Crystallogr.*, 25 (1992), 166–180.

2. Marinescu, D.C., J.R. Rice, M.A. Cornea-Hasegan, R.E. Lynch, M.G. Rossmann, "Macromolecular Electron Density Aveaging on Distributed Memory MIMD Systems," *Concurrency: Practice and Experience*, Vol 5(8), 1993, pp. 635–657.

3. Cornea-Hasegan, M., D.C. Marinescu and Z. Zhang, "Data Management for a Class of Iterative Computations on Distributed Memory MIMD Systems," *Concurrency: Practice and Experience*, 1994, (in press).