SCALABLE, FULL-MATRIX LEAST-SQUARES REFINEMENT OF SMALL MOLECULAR STRUCTURES ON A DISTRIBUTED MEMORY MULTIPROCESSING SYSTEM

Technical Report 94-03

Michael Kenneth Davoli

Department of Computer Science and Engineering
Auburn University
Auburn University, Alabama 36849-5347

February 3, 1994

THESIS ABSTRACT

SCALABLE, FULL-MATRIX LEAST-SQUARES REFINEMENT OF SMALL MOLECULAR STRUCTURES ON A DISTRIBUTED MEMORY MULTIPROCESSING SYSTEM

Michael Kenneth Davoli

Master of Science, March 18, 1994
(B.S., James Madison University, 1984)

84 Typed Pages

Directed by Stephen Seidman

The design and implementation of a scalable, parallel processing solution to full-matrix, least-squares refinement is presented in this thesis. In addition, all mathematical equations related to molecular refinement and all design considerations related to processor configuration are discussed. A solution describing a family of processor configurations is shown and the refinement times of four configurations are compared.

SCALABLE, FULL-MATRIX LEAST-SQUARES REFINEMENT OF SMALL MOLECULAR STRUCTURES ON A DISTRIBUTED MEMORY MULTIPROCESSING SYSTEM

Michael Kenneth Davoli

Certificate of Approval:

William B. Day Associate Professor Computer Science & Engineering	Stephen B. Seidman Chairman Professor Computer Science & Engineering
Mrinal Thakur Professor Materials Engineering	Thomas R. Webb Associate Professor Chemistry

Norman J. Doorenbos, Dean
Graduate School

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Determination of the structure of a molecule often involves X-ray diffraction analysis of crystals of the molecule. This method of structure determination can be described as a three step process. First, high-quality crystals must be grown and X-ray diffraction data collected. Next, the phase problem [15] must be overcome and the structure determined. Finally, the structure is refined by statistical comparison to calculated models of the molecule.

Refinement of a molecular structure is computationally intensive - it can require as much as 5 minutes of CPU time on a supercomputer [11]. This is because there are many atoms per molecule, and each atom can have as many as ten parameters which must be refined. For macromolecules, diffraction data is often incomplete because of an inability to make enough observations for all the atomic parameters. Macromolecular structure refinements normally include physical restraints such as bond lengths and bond angles which are treated as additional observations [10]. For smaller molecules the diffraction data are more complete (the structure is over-determined) and the refinement straightforward. The research treats the small molecule case.

There are two predominant techniques used in molecular refinement computer programs: Fourier refinement and least-

squares refinement. Fourier methods require more user interaction and the mathematical expressions are generally more complicated. Least-squares methods are easily automated and allow the inclusion of a variety of parameters which would require special analysis for treatment by the Fourier method [13]. The least-squares refinement technique was adopted for this research.

In the past, the main approach to reducing refinement times was to run a program on larger and faster computers [6]. However, access to supercomputers is sometimes limited and owning a supercomputer is prohibitively expensive. The advent of inexpensive multiprocessor computers made it possible for researchers to acquire a system that will allow refinement times to be reduced at an affordable price. A 16 processor system, based on the INMOS T800 transputer chip, costs only $22,000. Each chip is capable of 2 million floating-point operations per second [9]. The research targets transputer-based multiprocessor systems.

Because of the regular nature of least-squares refinement, it is well-suited for implementation on parallel processing systems. A variety of configurations is possible on transputer systems; likewise, a variety of message-passing strategies can be associated with each configuration. The first goal of this research was to determine the fastest combination of configuration and strategy for least-squares refinement of molecular structures.

Because the number of processors that make up a transputer system can vary, if a refinement program is to be useful, it must be

able to adapt to the number of processors that are available. The optimal number of processors is likely to depend on the size of the problem. If fewer processors are available, the program solution should be able to cope with the deficiency gracefully. If more processors are available, the program should utilize as many as are needed to assure optimal performance. The second goal of this research was to build a scalable and modular program which can run on different sized transputer systems.

I. LITERATURE REVIEW

According to Buerger [2], least-squares refinement was first introduced in 1941, but because of the computational effort required it was not widely used until the availability of high-speed computers in the 1950's. According to [6], before 1976 most protein structure refinements were performed using difference Fourier methods, while the least-squares method with geometrical restraints, such as bond angles and bond lengths, came later. Today, two of the more popular crystallography software packages, XTAL [14] and SHELX [12], offer least-squares refinement with a plethora of refinement options.

The popularity of the least-squares method is undoubtedly due in part to the ease with which the process is automated. However, as the ability to collect higher resolution data has progressed, the equations describing the underlying physics have become more detailed. This, in turn, has increased the number of parameters to be refined which has subsequently increased the amount of time and computer memory required to perform a refinement. In 1976, Konnert [10], speaking about approximate atomic models for large structures such as biological macromolecules, stated "The cost of conventional least-squares refinement of these trial structures, which may contain from

several hundred to thousands of atoms, is in many cases prohibitive." In 1985, Watkin [16] estimated that 76% of the CPU time is used in structure determination performing least-squares refinement. He based his estimate on detailed statistics derived from several months of computer runs. In 1988, Watkin [17] further emphasized the need for faster least-squares refinement. He stated that "...about 80% of CPU time used by crystallographers is spent in least squares, so that is the only real area where great efficiency must be achieved in the code."

Because of the great computational effort required, a number of techniques are used to reduce refinement times, occasionally at the price of accuracy. A block-diagonal matrix is sometimes used in place of a full least-squares matrix. In this matrix, the off-diagonal elements involving correlations between scale and thermal parameters and among parameters of the same atom are retained, while all others are set to zero. The storage requirements and computational effort are much less for this approximation than for a full matrix [13],[15]. In the case of a rigid molecule, the number of parameters can be reduced to 3 positional parameters (x,y,z) at the molecular origin, and 3 parameters (ϕ_1,ϕ_2,ϕ_3) which specify the molecular orientation [15]. The atoms on a rigid molecule are restricted in their rotational and vibrational motion due to the character of their chemical bonds. The bond lengths and bond angles are effectively static, so the molecule can be refined as one object. This greatly reduces the number of parameters and

therefore the refinement times. However, while most molecules are not rigid, many contain groups of atoms which can be treated as rigid bodies. In this approximation, a molecule is refined as a group of rigid bodies, thus greatly reducing the number of parameters. This strategy is often used in the refinement of macromolecules [18]. In yet another approximation, refinement times are reduced by combining bond distance restraints with the conjugate gradient method for solving linear systems [10].

As the computational effort required to perform least-squares (LS) refinement has increased, authors of these refinement programs have ported their code to the bigger and faster vector machines [6]. Indeed, both XTAL and SHELX can run on a variety of computers from an IBM PC to a Cray supercomputer. With the development of parallel computers, parallelizing compilers, and parallel languages, it is only natural to assume that LS refinement programs would soon be written for parallel machines in order to further reduce refinement times. Indeed, in [12] it is suggested that parallel processing be used in the LS matrix building stage, as that is the slowest step.

In [11], Laskowski et al. implemented restrained least-squares refinement of protein structures on a distributed memory multiprocessor system based on the INMOS transputer. Their approach was to parallelize an existing sequential program, RESTRAIN, so that parts of it could run on two or more transputers, concurrently. They identified the most CPU-intensive part of the

program as the calculation of the structure factors and their derivatives. They therefore implemented the program as a master/slave arrangement with the slaves performing the structure factor calculations.

Because the original program was written in FORTRAN, and because the transputer has a FORTRAN compiler, much of the original code was left unmodified. The control and communications parts of the parallel program were written in occam, a parallel language. Because of this, much of RESTRAIN on the transputer is implemented as library routines containing the FORTRAN code. The software was implemented on a Meiko Computing Surface hosted by a microVAX II, and containing a T414 host processor, an 8 Mbyte T800 Mass Store processor, and a board with four 4 Mbyte T800 processors.

Laskowski et al. use a 3.9 Mbyte scratch file on the microVAX to store the least squares matrix. Because file access via the microVAX host is slow, and because RESTRAIN must read the scratch file 7 times during the solution of the least-squares matrix, I/O became a rate-determining factor in their implementation. To reduce the I/O cost, the authors installed a Meikos internal disk so the transputer could read directly from the disk rather than through the host. This solution reduced the I/O cost; however, one transputer was dedicated to just controlling the disk and another to running the requisite software. The overall execution time was reduced by about 20 minutes, from 74.1 to 55.5 minutes. To reduce

the I/O cost even further, the authors modified RESTRAIN so that it stores as much of the scratch file in memory as will fit. The parallel parts of the code ran at the same rate, but the serial contribution was reduced from 17.8 to 2.8 minutes. Laskowski et. al. conclude that given enough processors, the refinement time can be reduced to the 2.8 minute serial limit. In addition, they mention that a new optimized version of the FORTRAN compiler for the transputers should reduce that time even further. They do not, however, explain any changes that might need to be made to the code in order that more processors can be accommodated.

A T800 transputer is a single-chip computer manufactured by INMOS [9]. Each T800 contains a 32-bit processor, a 64-bit floating point processor, 1 Megabyte of RAM, and 4 bidirectional serial links. Each link can be used to connect one transputer to another or to a host computer. All 4 links are capable of concurrent communication at 20 megabits per second. In addition, the 64-bit floating point processor can run concurrently with the 32-bit processor. The links can be used to connect transputers to build a distributed memory multiprocessor system. Each transputer runs independently of the others, so when information is to be shared, it must be passed over the serial links.

The programming language occam (now occam2) [5] was designed to implement C.A.R. Hoare's concurrent specification language *Communicating Sequential Processes* (CSP) [7] on an array

of transputers. Interprocess communication is an essential part of the occam2 programming language.

Distributed memory multi-processor systems share information by communicating via links connecting the processors. The communication rate between processors is often significantly lower than the rate at which the processors can execute commands. To maximize overall program execution speed on this type of system, a program must be distributed with consideration for the difference between processor and communication speeds. This type of optimization is known as load balancing.

According to [1], with the recent availability of parallel computers much attention has been given to algorithms from linear algebra, and in particular, to the Gaussian elimination and LU decomposition algorithms. In [3] Chu and George implemented Gaussian elimination with partial pivoting on an Intel hypercube multiprocessor. They employed a row-oriented distribution scheme where the rows of the matrix are distributed to each node processor and the computations performed row by row. They note that as the algorithm progresses, a disparity in work load among processors can occur due to the choice of pivot row. They further note that in the worst case, this disparity in work load can lead to a 50% increase in execution time. Their solution to this problem was to dynamically balance the load by explicitly performing the row interchanges so that each processor node retains approximately the same number of uneliminated rows. They conclude that this extra communication

effort results in only a modest increase in execution time. It should be noted that, in a scheme which distributes entire rows to each node, there is an upper limit to the size of a matrix that can be accommodated. The upper limit is proportional to the memory of a node.

In [1] Bisselling and van de Vorst implemented LU decomposition with partial pivoting on a 2-dimensional mesh of transputers. They compared the efficiency of a variety of Cartesian mapping schemes for the coefficient matrix. They found that block clustering, where the matrix is effectively overlaid on the mesh, degrades the performance of the algorithm due to poor load balancing. They found that the cyclic or interleaved assignment of matrix rows (or columns) to processors results in good load balancing. In a cyclic mapping scheme, matrix row i is mapped to processor row $i \bmod p$, where p is the number of processors in a mesh row. In addition, they found that a grid or scattered mapping scheme, where the matrix elements are mapped cyclically in both directions, has an even better load distribution. They conclude by noting that the method scales well, that is, as the size of the mesh increases, the efficiency drops very slowly.

II. THE MATHEMATICS OF MOLECULAR REFINEMENT

Crystallographic Equations

In X-ray crystallography, a crystal is irradiated with a coherent beam of X-rays, and because of the regular nature of crystals, discrete spots or reflections of the incident X-ray beam are diffracted by the crystal. Each reflection represents the intensity of the reflected X-rays from a particular set of crystal lattice planes. These lattice planes are related to the size and shape of the unit cell. A unit cell can be thought of as the basic repeating unit from which a crystal is built. It is defined by six parameters: three axial lengths (designated a, b, and c) and three interaxial angles (α, β, and γ). A unit cell contains one or more unique volumes called the asymmetric unit. Each unit is identical to any others in the cell in terms of size and atomic content. Because of this, only the contents of one asymmetric unit and the symmetrically equivalent positions for the space group are needed to describe the contents of the unit cell. Each atom's location in the unit cell is defined by its x, y, and z coordinates from the cell origin. In addition, the shape of each atom's electron cloud can be described roughly as a sphere or more precisely, as an ellipsoid. One isotropic temperature parameter, Uiso, describes a sphere. Six anisotropic temperature parameters (designated U_{11}, U_{22}, U_{33}, U_{12}, U_{13}, and U_{23}) describe an ellipsoid. It

is an atom's electron cloud that reflects the X-rays, and each type of atom reflects X-rays differently, so that an atomic scattering factor (f) must also be used to describe the observed scattering.

The structure factor equation (2.1) is an attempt to describe the observed scattering and therefore includes all the above information. In the crystallography literature [15], "The structure factor is the resultant of N waves scattered in the direction of the reflection hkl by the N atoms in the unit cell." The structure factor for one reflection is normally written in the form:

$$|F_{calc}| = \sqrt{A^2 + B^2} \qquad (2.1)$$

where

$$A = \sum_{j=1}^{n} \sum_{symm} f_j T_j \cos(2\pi(hx_j + ky_j + lz_j)),$$

$$B = \sum_{j=1}^{n} \sum_{symm} f_j T_j \sin(2\pi(hx_j + ky_j + lz_j)),$$

where: n is the number of atoms in the unit cell.
f_j is the atomic scattering factor for the j^{th} atom
x_j, y_j, z_j are the positional parameters of the j^{th} atom
T_j is the temperature factor for the j^{th} atom
h,k,l are the Miller indices for this particular reflection,

and the summation over *symm* is a summation over all the symmetrically equivalent positions.

When a structure is refined using just one isotropic temperature parameter, the temperature factor, T_j, is defined by the following equation

$$T_j = \exp\left(-8\pi^2 U_{iso_j} \left(\frac{\sin\theta}{\lambda}\right)^2\right),$$

where $Uiso_j$ is the isotropic temperature parameter for the j^{th} atom, and the expression defining $\sin\theta/\lambda$ is shown in Figure 2.1.

$$\frac{\sin\theta}{\lambda} = 0.5\sqrt{h^2a^{*2} + k^2b^{*2} + l^2c^{*2} + 2(hka^*b^*\cos\gamma^* + hla^*c^*\cos\beta^* + klb^*c^*\cos\alpha^*)}$$

Figure 2.1 Calculation of $\sin\theta/\lambda$ for reflection hkl

where: a^*, b^*, c^* are the sides of the reciprocal unit cell
$\alpha^*, \beta^*, \gamma^*$ are the angles of the reciprocal unit cell
h,k,l are the Miller indices for this particular reflection

When a structure is refined using six anisotropic temperature parameters, the expression defining T_j is

$$T_j = \exp\left(-2\pi^2\left(\begin{array}{l} U_{11,j}\,h^2\,a^{*2} + U_{22,j}\,k^2\,b^{*2} + U_{33,j}\,l^2\,c^{*2} + 2\,U_{12,j}\,h\,k\,a^*\,b^*\cos\gamma^* \\ \quad + 2\,U_{13,j}\,h\,l\,a^*\,c^*\cos\beta^* + 2\,U_{23,j}\,k\,l\,b^*\,c^*\cos\alpha^* \end{array}\right)\right),$$

where: h,k,l are the Miller indices for this particular reflection
a^*, b^*, c^* are the sides of the reciprocal unit cell
$\alpha^*, \beta^*, \gamma^*$ are the angles of the reciprocal unit cell
$U_{11}, U_{22}, U_{33}, U_{12}, U_{13}, U_{23}$ are the anisotropic temperature parameters for the j^{th} atom

The atomic scattering factor, *f*, is calculated for each atom type for each reflection. Figure 2.2 shows pseudocode for an algorithm that calculates *f* for atom type *j*, where *cm* are the Cromer-Mann coefficients [8]. The value $(sin\Theta/\lambda)$ is different for each reflection (see Figure 2.1).

```
fj  = cm[j][8]
For i = 0 to 6 in steps of 2
  {
  fj  = fj  + cm[j][i]  * exp(-min(75,cm[j][i+1]*(sinΘ/λ)^2 ))
  }
```

Figure 2.2 Calculation of the atomic scattering factor for atom type j

Least Squares

Least squares is a statistical method used to find the best fit of a set of calculated parameters to a larger set of observed values. Given a linear function, f, of n variables, $x_1, x_2, ..., x_n$, and n independent parameters, $p_1, p_2, ..., p_n$

$$f = p_1x_1 + p_2x_2 + p_3x_3 + ... + p_nx_n$$

if the values of the function are measured at m different points (where $m > n$), the best values for the parameters are defined to be those that minimize the weighted sum of the squares of the differences between the observed and calculated values of the function for all the data points. In other words, the goal is to minimize

$$D = \sum_{r=1}^{m} w_r (f_{obs,r} - f_{calc,r})^2$$

where: w_r is the weight to be assigned an observation
$f_{obs,r}$ is one of the m observed values of the function
$f_{calc,r}$ is the corresponding calculated value
m is the number of points measured

To obtain the best fit, the parameters $p_1, p_2, ..., p_n$ are treated as variables that can be adjusted to minimize D. The minimization is performed by setting the partial derivative of the equation with respect to each parameter to zero.

$$\sum_{r=1}^{m} w_r (f_{obs,r} - f_{calc,r}) \frac{\partial f_{calc,r}}{\partial p_j} = 0$$

where j = 1, 2, 3, ..., n.

This creates the following set of *normal equations*:

$$\begin{aligned}
&\sum_{r=1}^{m} w_r x_{1,r}^2 p_{1,r} + \sum_{r=1}^{m} w_r x_{1,r} x_{2,r} p_{2,r} + ... + \sum_{r=1}^{m} w_r x_{1,r} x_{n,r} p_{n,r} = \sum_{r=1}^{m} w_r x_{1,r} f_{obs,r} \\
&\sum_{r=1}^{m} w_r x_{2,r} x_{1,r} p_{1,r} + \sum_{r=1}^{m} w_r x_{2,r}^2 p_{2,r} + ... + \sum_{r=1}^{m} w_r x_{2,r} x_{n,r} p_{n,r} = \sum_{r=1}^{m} w_r x_{2,r} f_{obs,r} \\
&\qquad\vdots \qquad\qquad\qquad\vdots \qquad\qquad\qquad\vdots \qquad\qquad\qquad\vdots \\
&\sum_{r=1}^{m} w_r x_{n,r} x_{1,r} p_{1,r} + \sum_{r=1}^{m} w_r x_{n,r} x_{2,r} p_{2,r} + ... + \sum_{r=1}^{m} w_r x_{n,r}^2 p_{n,r} = \sum_{r=1}^{m} w_r x_{n,r} f_{obs,r}
\end{aligned}$$

The solution of this system of linear equations gives the best values for the parameters.

In least-squares molecular refinement, the function to be minimized is

$$D = \sum_{r=1}^{m} w_r (|F_{obs,r}| - s|F_{calc,r}|)^2 \qquad (2.2)$$

where: $|F_{obs,r}|$ is the intensity of the r^{th} reflection (the observed value)

$|F_{calc,r}|$ is the structure factor of the intensity of the r^{th} reflection (the calculated value)

m is the number of measured reflections

w_r is the weight assigned to the r^{th} reflection.
s is the scale factor (this is normally designated "k" in crystallographic texts; however, it is designated "s" here for clarity).

Because the observed values are not necessarily on the same scale as the calculated values, a scale factor, *s*, is defined as

$$s = \frac{\sum_{r=1}^{m} |F_{obs}|}{\sum_{r=1}^{m} |F_{calc}|}$$

where *m* is the number of reflections in the data set.
This factor is used to keep the two values on the same scale. The scale factor is calculated only once, at the beginning of the refinement; thereafter the scale factor is an independent parameter which is refined along with the others.

To find the minimum, set the derivative of (2.2) with respect to each of the experimental parameters equal to 0. We obtain the equation

$$\sum_{r=1}^{m} w_r (|F_{obs,r}| - s|F_{calc,r}|)\left(\frac{\partial(|F_{obs,r}| - s|F_{calc,r}|)}{\partial p}\right) = 0 \qquad (2.3)$$

In (2.3), *p* is an element of the set **P**. In the case of isotropic temperature factors, **P** contains 3 positional and 1 thermal parameter for each of the *n* atoms in the asymmetric unit, plus the scale factor:

$$\mathbf{P} \equiv \{x_j, y_j, z_j, Uiso_j \mid j=1,...n\} \cup \{s\}$$

In the case of anisotropic temperature factors, **P** contains 3 positional and 6 thermal parameters for each of the n atoms in the asymmetric unit, plus the scale factor:

$$\mathbf{P} \equiv \{x_j, y_j, z_j, U11_j, U22_j, U33_j, U12_j, U12_j, U23_j \mid j=1,...n\} \cup \{s\}$$

The method of least squares fits a linear function to a data set. Since $|F_{calc}|$ is a transcendental function, it is replaced with its first order Taylor series expansion about the experimental parameters (the set **P**). Because the scale factor, once established, is an independent parameter, the function to be linearized is not just $|F_{calc}|$, but $s|F_{calc}|$. The Taylor series for the isotropic case is given below:

$$s\,|F_{calc}(a)| = s\,|F_{calc}(a_0)| + \frac{\partial(s\,|F_{calc}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n}\left(\frac{\partial(s\,|F_{calc}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s\,|F_{calc}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s\,|F_{calc}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s\,|F_{calc}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j\right) \tag{2.4}$$

If the approximation in (2.4) is substituted into (2.3), equation (2.5) is obtained for $p \in \mathbf{P}$ (see Figure 2.3). Taking the derivative with respect to each of the parameters in the set yields the set of least-squares equations. For example, (2.6) shows how the equation looks when $p = x_j$ (see Figure 2.3). Equations (2.7), (2.8), and (2.9) show how (2.6) is expanded and rearranged. Defining $\Delta F = |F_{obs,r}| - s|F_{calc,r}(a_0)|$, (2.9) becomes (2.10) which in turn becomes (2.11), which is one of the least-squares equations (see Figures 2.3 and 2.4). The solution to the system of least-squares equations is the

amount each parameter should be shifted to move toward minimum deviation. The entire system of least-squares equations for refinement with isotropic temperature parameters is shown in Figure 2.5.

$$\sum_{r=1}^{m} w_r \left(|F_{obs,r}| - \left(s|F_{cald}(a_0)| + \frac{\partial(s|F_{cald}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{cald}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{cald}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{cald}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{cald}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right) \right) \left(\frac{\partial \left(s|F_{cald}(a_0)| + \frac{\partial(s|F_{cald}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{cald}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{cald}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{cald}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{cald}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right)}{\partial p} \right) = 0$$

$$\sum_{r=1}^{m} w_r \left(|F_{obs,r}| - \left(s|F_{calc,r}(a_0)| + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right) \right) \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} = 0$$

$$\sum_{r=1}^{m} w_r \left(|F_{obs,r}| - s|F_{calc,r}(a_0)| - \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right) \right) \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} = 0$$

$$\sum_{r=1}^{m} w_r \left(|F_{obs,r}| - s|F_{calc,r}(a_0)| \right) \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} \right) - \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right) \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} = 0$$

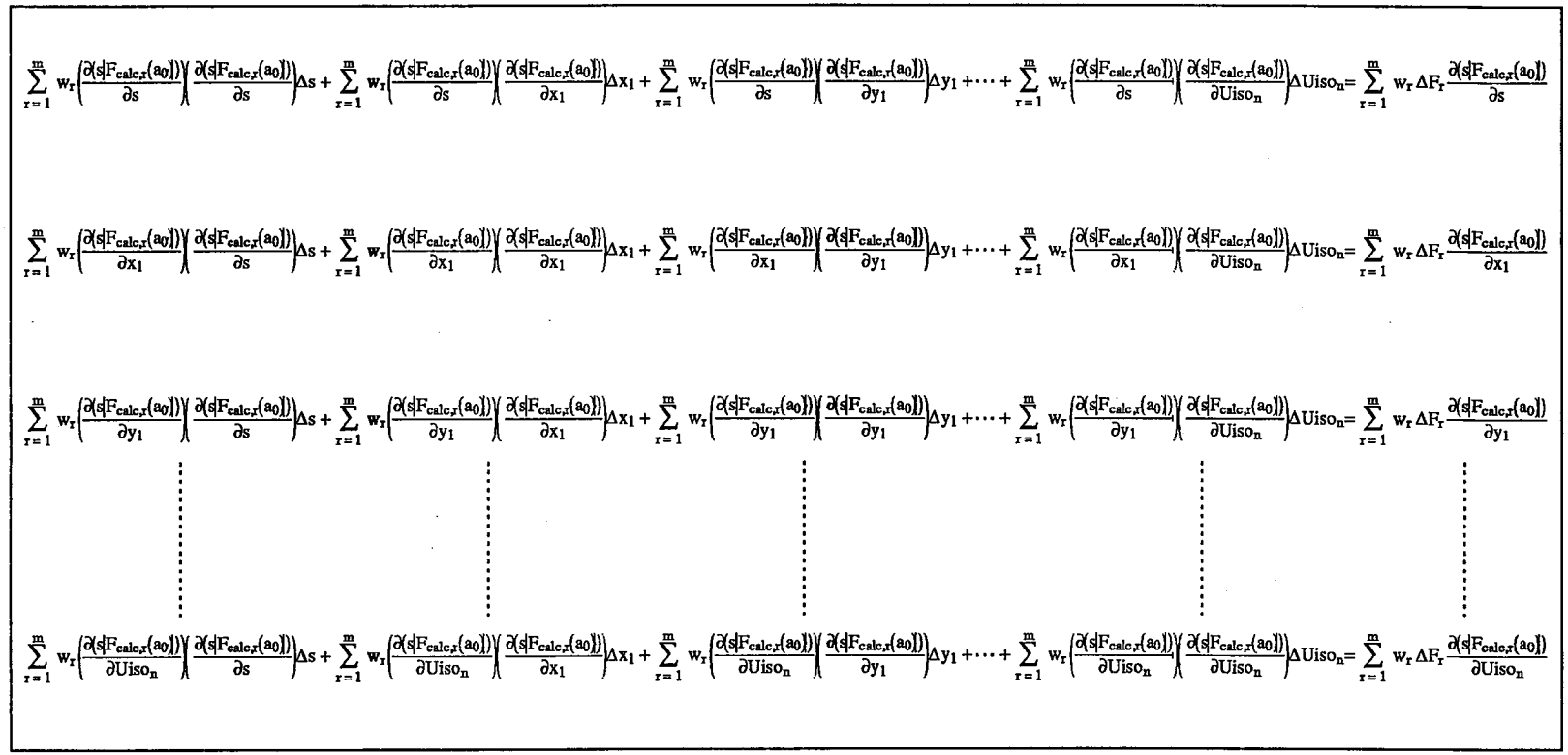
Figure 2.3 From top to bottom, equations (2.5), (2.6), (2.7), and (2.8), respectively

$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right) \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} = \sum_{r=1}^{m} w_r (|F_{obs,r}| - s|F_{calc,r}(a_0)|) \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}$$

$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial s}\Delta s + \sum_{j=1}^{n} \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}\Delta x_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial y_j}\Delta y_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial z_j}\Delta z_j + \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial Uiso_j}\Delta Uiso_j \right) \right) \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} = \sum_{r=1}^{m} w_r \Delta F_r \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}$$

$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} \right) \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial s} \right) \Delta s + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} \right) \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_1} \right) \Delta x_1 + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} \right) \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial y_1} \right) \Delta y_1 + \cdots + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j} \right) \left(\frac{\partial(s|F_{calc,r}(a_0)|)}{\partial Uiso_n} \right) \Delta Uiso_n = \sum_{r=1}^{m} w_r \Delta F_r \frac{\partial(s|F_{calc,r}(a_0)|)}{\partial x_j}$$

Figure 2.4 From top to bottom, equations (2.9), (2.10), and (2.11), respectively



$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\Delta s + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\Delta x_1 + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\Delta y_1 + \cdots + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\Delta Uiso_n = \sum_{r=1}^{m} w_r \Delta F_r \frac{\partial(s|F_{calc,r}(a_0])}{\partial s}$$

$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\Delta s + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\Delta x_1 + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\Delta y_1 + \cdots + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\Delta Uiso_n = \sum_{r=1}^{m} w_r \Delta F_r \frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}$$

$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\Delta s + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\Delta x_1 + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\Delta y_1 + \cdots + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\Delta Uiso_n = \sum_{r=1}^{m} w_r \Delta F_r \frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$\sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial s}\right)\Delta s + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial x_1}\right)\Delta x_1 + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial y_1}\right)\Delta y_1 + \cdots + \sum_{r=1}^{m} w_r \left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\left(\frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}\right)\Delta Uiso_n = \sum_{r=1}^{m} w_r \Delta F_r \frac{\partial(s|F_{calc,r}(a_0])}{\partial Uiso_n}$$

Figure 2.5 System of least-squares equations for refinement with isotropic temperature parameters

The solution of the system of linear equations shown in Figure 2.5 is the set of shifts, $\{\Delta \mathbf{P}\}$, corresponding to the set of parameters, $\mathbf{P}$. Because the function giving rise to the least squares matrix is a truncated Taylor series, the building of the matrix and solving of the system must be repeated many times. Each Δp is added to its corresponding p, and the least squares matrix is rebuilt with the new values for the parameters. Ideally, this process is repeated until a value called the *R-factor* falls below a desired value. The R-factor, also known as the *reliability index*, is defined as:

$$\text{R-factor} = \frac{\sum_{r=1}^{m} \left| |F_{obs,r}| - |F_{calc,r}| \right|}{\sum_{r=1}^{m} |F_{obs,r}|}$$

To build the LS matrix, the partial derivative with respect to each parameter is required. As an example, the partial derivative with respect to x_j, where x_j is the x-coordinate of the *jth* atom, is

$$\frac{\partial |F_{calc}|}{\partial x_j} = s\left(\frac{A\frac{\partial A}{\partial x_j} + B\frac{\partial B}{\partial x_j}}{\sqrt{A^2 + B^2}}\right) = s\left(\frac{A\frac{\partial A}{\partial x_j} + B\frac{\partial B}{\partial x_j}}{|F_{calc}|}\right),$$

where

$$\frac{\partial A}{\partial x_j} = \frac{\partial\left(\sum_{j=1}^{n} \sum_{symm} f_j T_j \cos\left(2\pi(hx_j + ky_j + lz_j)\right)\right)}{\partial x_j}$$

and

$$\frac{\partial B}{\partial x_j} = \frac{\partial\left(\sum_{j=1}^{n}\sum_{symm} f_j\, T_j \sin\left(2\pi(hx_j + ky_j + lz_j)\right)\right)}{\partial x_j}$$

When taking the partial derivative with respect to the positional parameters, atoms in symmetrically equivalent positions may possibly add terms that will cancel or double. For example, for the symmetry relations X,Y,Z and -X,-Y,-Z,

$$\frac{\partial A}{\partial x_j} = \frac{\partial\left(\sum_{j=1}^{n} f_j\, T_j \cos\left(2\pi(hx_j + ky_j + lz_j)\right) + \sum_{j=1}^{n} f_j\, T_j \cos\left(2\pi(h(-x)_j + k(-y)_j + l(-z)_j)\right)\right)}{\partial x_j}$$

$$= (-4)\,\pi\, h\, f_j\, T_j \sin\left(2\pi\left(hx_j + ky_j + lz_j\right)\right)$$

and

$$\frac{\partial B}{\partial x_j} = \frac{\partial\left(\sum_{j=1}^{n} f_j\, T_j \sin\left(2\pi(hx_j + ky_j + lz_j)\right) + \sum_{j=1}^{n} f_j\, T_j \sin\left(2\pi(h(-x)_j + k(-y)_j + l(-z)_j)\right)\right)}{\partial x_j} = 0$$

The partial derivatives of $s|F_{calc}|$ with respect to each of the parameters of the j^{th} atom are given in Figure 2.6.

positional parameters

$$\frac{\partial s|F_{calc}|}{\partial x_j} = \frac{-2\pi s h f_j T_j (A\sin(2\pi(hx_j + ky_j + lz_j)) - B\cos(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial y_j} = \frac{-2\pi s k f_j T_j (A\sin(2\pi(hx_j + ky_j + lz_j)) - B\cos(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial z_j} = \frac{-2\pi s l f_j T_j (A\sin(2\pi(hx_j + ky_j + lz_j)) - B\cos(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

isotropic thermal parameter

$$\frac{\partial s|F_{calc}|}{\partial Uiso_j} = \frac{-8\pi^2 s\left(\frac{\sin\theta}{\lambda}\right)^2 f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

anisotropic thermal parameters

$$\frac{\partial s|F_{calc}|}{\partial U_{11_j}} = \frac{-2\pi^2 s h^2 a^{*2} f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial U_{22_j}} = \frac{-2\pi^2 s k^2 b^{*2} f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial U_{33_j}} = \frac{-2\pi^2 s l^2 c^{*2} f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial U_{12_j}} = \frac{-4\pi^2 s h k a^* b^* (\cos\gamma^*) f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial U_{13_j}} = \frac{-4\pi^2 s h l a^* c^* (\cos\beta^*) f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

$$\frac{\partial s|F_{calc}|}{\partial U_{23_j}} = \frac{-4\pi^2 s k l b^* c^* (\cos\alpha^*) f_j T_j (A\cos(2\pi(hx_j + ky_j + lz_j)) + B\sin(2\pi(hx_j + ky_j + lz_j)))}{|F_{calc}|}$$

scale factor

$$\frac{\partial s|F_{calc}|}{\partial s} = |F_{calc}|$$

Figure 2.6 Partial derivatives of $s|F_{calc}|$ with respect to each of the parameters of the j^{th} atom and with respect to the scale factor

III. AN ALGORITHM FOR LEAST-SQUARES MOLECULAR REFINEMENT

The Algorithm

The algorithm to perform the LS molecular refinement is described below. It is described from a serial standpoint so that it can be better understood and so the need for parallelization is highlighted.

First, the initial value of the scale factor is determined. This is done by setting the scale factor to 1.0, calculating the structure factor for each reflection, and summing all the structure factors. At the same time the reflection intensities (observed values) are also summed. The initial value of the scale factor is set to be the sum of the observed values divided by the sum of the calculated values. This is done only once for the refinement; thereafter, the scale factor is a parameter which will be modified during the refinement.

Next, the structure factor and the partial derivatives are determined for each reflection. Once they are determined, the matrix for that reflection is built. The least-squares matrix is the sum of all the matrices for each reflection, so as each reflection's matrix is built, it is added to the running total. After all reflections have been processed the least-squares matrix has been built and can be solved.

The solution to the least-squares matrix is the set of shifts to be applied to each parameter that is being refined. After the shifts have been applied, the R-factor is determined and the refinement is performed again. This cycle of building and solving the least-squares matrix is repeated until either the parameter shifts are acceptably small (refinement converges) or the maximum number of iterations has been performed.

A data set normally includes the 3 positional parameters and 1 isotropic temperature parameter per atom. In the early cycles of refinement the isotropic temperature factor is used in the calculation of the structure factor and derivatives, so only one temperature parameter is refined per cycle. The size of the least-squares matrix is therefore $(4n + 1)^2$, where n is the number of atoms.

As the refinement progresses and the positions of the atoms vary less with each refinement cycle, the isotropic temperature parameter is replaced by six anisotropic temperature parameters. This increases the precision of the refinement, but also increases the time to calculate the structure factors and derivatives. In addition, the size of the least-squares matrix increases to $(9n + 1)^2$. Again, this refinement continues until either the R-factor is below the desired level, or until the specified maximum number of iterations has been performed.

Complexity

Least-squares refinement consists of three major computational tasks: calculating the structure factors and partial derivatives for each reflection, building the LS matrix, solving the LS matrix.

The calculation of the structure factors and derivatives takes O(n) time, where n is the number of atoms in the asymmetric unit. Figure 3.1 contains pseudocode describing the crystallographic equations. In the pseudocode, *SQRT* represents the square root function, and left and right curly braces, *{ }*, represent the beginning and end of a group of statements. In addition, the pseudocode is indented to show the nesting of statements more clearly.

```
For each reflection
  {
  For each symmetrically equivalent position
    {
    For each atom, j
      {
      A := A + Aj
      B := B + Bj
      For each atomic parameter
        {
        calculate part1 of partial derivatives
        }
      }
    }
    SF := SQRT(A^2 + B^2)
    For each parameter (the atoms plus the scale factor)
      {
      calculate part2 of partial derivatives
      }
    }
  }
```

Figure 3.1 Calculating structure factors and derivatives

The partial derivatives require 2 steps to build. The first step requires both A_j and B_j for each atom j, while the second step requires the structure factor.

Building the LS matrix is a $O(n^2)$ operation. Figure 3.2 contains pseudocode describing the building of the LS matrix. In the pseudocode, *parms* represents the number of parameters to be refined; pd_i represents the partial derivative with respect to the i^{th} parameter; *mat[i][j]* is the matrix element from row i and column j; and left and right curly braces, *{ }*, represent the beginning and end of a group of statements. The pseudocode is indented to show the nesting of statements more clearly.

```
For each reflection
  {
  For i = 0 to parms-1
    {
    For j = 0 to parms-1
      {
      mat[i][j] := mat[i][j] + (pdi *pdj )
      }
    mat[i][parms] := mat[i][parms] + (ΔF*pdi )
    }
  }
```

Figure 3.2 Building the LS matrix

Because of the great number of reflections and thus the great number of times the matrix is built, the time to build the entire LS matrix is normally much greater than the time to solve it.

Solving the LS matrix using Gaussian elimination with partial pivoting is an $O(n^3)$ operation. Figure 3.3 contains pseudocode

describing Gaussian elimination with partial pivoting. In the pseudocode, *n* is the number of rows in the LS matrix; *mat[i][j]* is the matrix element from row *i* and column *j*; *pivot* is the pivot element; *prow* is the row containing the pivot element; and left and right curly braces, *{ }*, represent the beginning and end of a group of statements.

```
For i = 1 to n
  {
  pivot := mat[i][i]
  prow := i
  For j = i+1 to n
    if (ABS(mat[i][j]) > ABS(pivot))
      {
      pivot := mat[i][j]
      prow := j
      }
  For j = 1 to n+1
    {
    mat[prow][j] := mat[prow][j]/pivot
    }
  switch pivot row with swap row
  For j = i+1 to n
    {
    For k = 1 to n
      {
      eliminate row
      }
    }
  }
```

Figure 3.3 Gaussian elimination with partial pivoting

IV. DESIGN OF A PARALLEL ALGORITHM FOR LEAST-SQUARES MOLECULAR REFINEMENT

Building the LS matrix

There are many considerations that affect the design of a scalable, distributed solution to the crystallographic problem. Most significant is the connectivity of the processor elements because this will limit the type and number of possible configurations. Next in importance are the data dependencies and the concurrency which are inherent to the calculations. These will further limit the choice of configuration. In addition, the design should include a scalable message-passing scheme combined with modular code as these will facilitate the adaptation of a solution to a variety of configurations. Finally, because of the large amount of data required to perform a molecular refinement, the I/O should be buffered to reduce the cost.

At the highest level, LS molecular refinement can be described as building a matrix and solving the corresponding linear system many times. The first consideration in designing a scalable, distributed solution is whether the LS matrix will reside entirely on one processor or will be distributed among two or more processors. Limiting the LS matrix to one processor restricts the possible matrix size to the amount of memory on that processor. Consequently, the maximum problem size is fixed; this would remove the motivation

for a scalable solution. In addition, solving the LS matrix would become an expensive serial operation. A distributed LS matrix was chosen to facilitate scalability and to speed up the solving of the linear system.

The LS matrix is the sum of all the matrices for each reflection; and, as noted by Laskowski et. al. [11], "...the calculation for each reflection can be performed completely independently of every other." Because of this obvious concurrency, a processor farm (or master/slave) paradigm was chosen to calculate the structure factors and the partial derivatives. A unique set of reflections is processed by each node in the processor farm. A modular piece of code designated SF was written to perform these calculations.

To build the LS matrix, the matrices from all the reflections must be summed. This summation can be done in a number of ways. One strategy is to calculate and sum the matrices on each node in the processor farm, and to then merge the sums of all the nodes yielding the LS matrix. This solution is also limited by processor memory.

An alternative strategy is to calculate on each node the matrix for one reflection, one row at a time. The rows from all nodes are then merged until all rows from all matrices of all reflections have been summed. This second strategy indeed overcomes the storage limitation, but it creates a large number of communications, greatly increasing the processing time.

However, because the matrix for each reflection is diagonally symmetric, only the partial derivatives, the ΔF, and that reflection's weight are needed to build the matrix. In light of this, a strategy was chosen that builds a message containing the above data, collects this message from each node, and broadcasts the messages to each of the processors which will contain part of the LS matrix. Each of these processors then builds only those partial columns and partial rows that are located on its section of the LS matrix. The communications cost for this strategy is clearly significantly less than that of the second strategy. A modular piece of code designated BUILD was written to perform these calculations.

A piece of code designated MAIN was written to perform the buffered I/O and to distribute the data to the nodes in the processor farm. The processor containing MAIN is connected to the host computer via one of its four serial links. The disk drives containing the data files for the refinement are connected to the host computer. Binary input and output files are used to reduce the cost of data conversion.

The processor farm paradigm features a many-branched tree with a single node at the root. Data flows from the root to the leaf nodes where much of the work is performed. The transputer, however, has only four serial links. Because one of the root node's links is connected to the host computer, only three links are available to connect the root node to the leaf nodes. In order to increase the possible number of leaf nodes, a modular piece of code

designated FORK was written (see Figure 4.1). FORK is a one-place buffer that receives data from a single predecessor and distributes the data to 2 or 3 successors. FORK can be used to build a processor farm containing many leaf nodes.

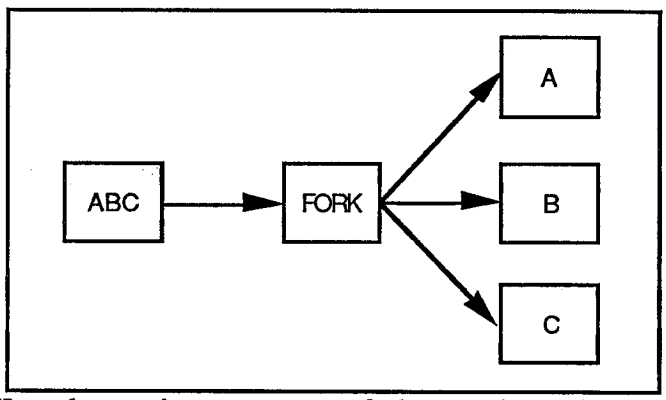


Figure 4.1 FORK node receives a group of three unique data packets, A, B, and C, and sends each packet to a different successor

In order to collect the SF results from a large number of leaf nodes, a modular piece of code designated MERGE was written (see Figure 4.2). MERGE implements a one-place buffer that collects data from 2 or 3 predecessors and sends the data to a single successor. MERGE can be used to collect data from a processor farm containing many leaf nodes.

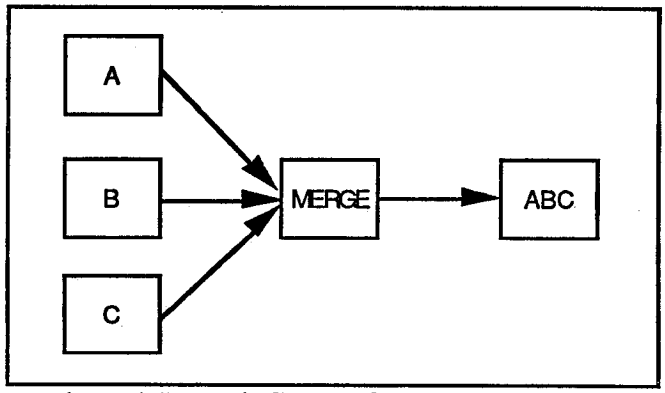


Figure 4.2 Data packets A,B, and C sent from predecessors to a MERGE node, collected into one message, and sent to a successor

Based on the results from [1], a square mesh of processors was chosen to store the LS matrix. Matrix row i is mapped to mesh row i mod p, where p is the number of processors in a mesh column. This distribution was shown to enhance processor efficiency. More complicated distribution strategies were avoided because, although they would lead to a faster solution, they were far more difficult to code. Also, the chosen mapping strategy resulted in code that was fast enough that the strategy was never a rate-determining step in the refinement. In addition, it allowed the design and implementation of a simple, fast backsubstitution step.

Because all of the data collected from the leaf nodes must be sent to each processor in the mesh containing the LS matrix, and because the mesh, like the processor farm, can contain many processors, a modular piece of code designated BROADCAST was written to distribute the data to the mesh (see Figure 4.3). BROADCAST is a one-place buffer that receives a collection of

messages from a single predecessor and broadcasts a copy of the collection to each of its successors.

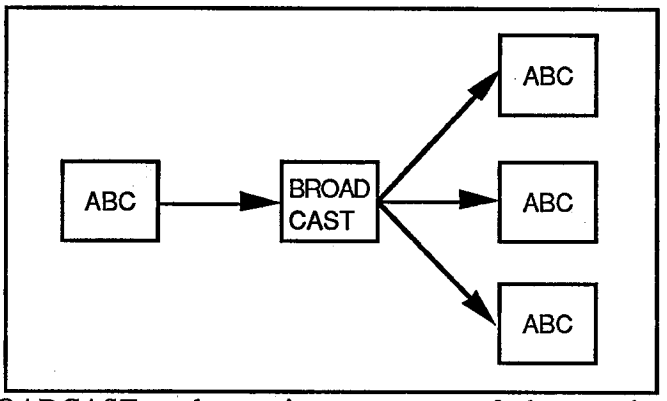


Figure 4.3 BROADCAST node receives a group of three unique data packets, A,B, and C, and sends a copy of the group to each successor

For simplicity, the MERGE nodes form a tree that is a mirror image of the tree formed by the FORK node (see Figure 4.11). A single MERGE node resides at the root of this tree and sends its data to a single BROADCAST node. Like FORK, BROADCAST is used to build a tree. The leaf nodes of the BROADCAST tree are the processors on the left side of the mesh of processors containing the LS matrix. Once the data reaches the mesh, it is passed from the left to the right across the mesh to the right edge. After each mesh processor has passed along the data, it calls BUILD to build its section of the LS matrix.

Solving the LS system of equations

A square mesh of processors, with three or more processors on a side, contains nine unique positions. Consecutive columns of the LS matrix are distributed from left to right. Row *i* of the LS

matrix is mapped to processor row i mod p, where p is the number of processors in a mesh row (see Figure 4.4). Modular code was written for each of the nine unique positions. NW is the code for the processor in the top left corner of the mesh; SW, the bottom left corner; NE, the top right corner; SE, the bottom right corner; NORTH, a processor on the top side of the mesh; SOUTH, the bottom side; WEST, the left side; EAST, the right side; and MIDDLE, a processor that is contained entirely within the mesh (see Figure 4.4). Because the code is modular and the message passing scheme scalable, there is no limit to the size of the mesh on which it can run. In addition, the code will run on any rectangular mesh.

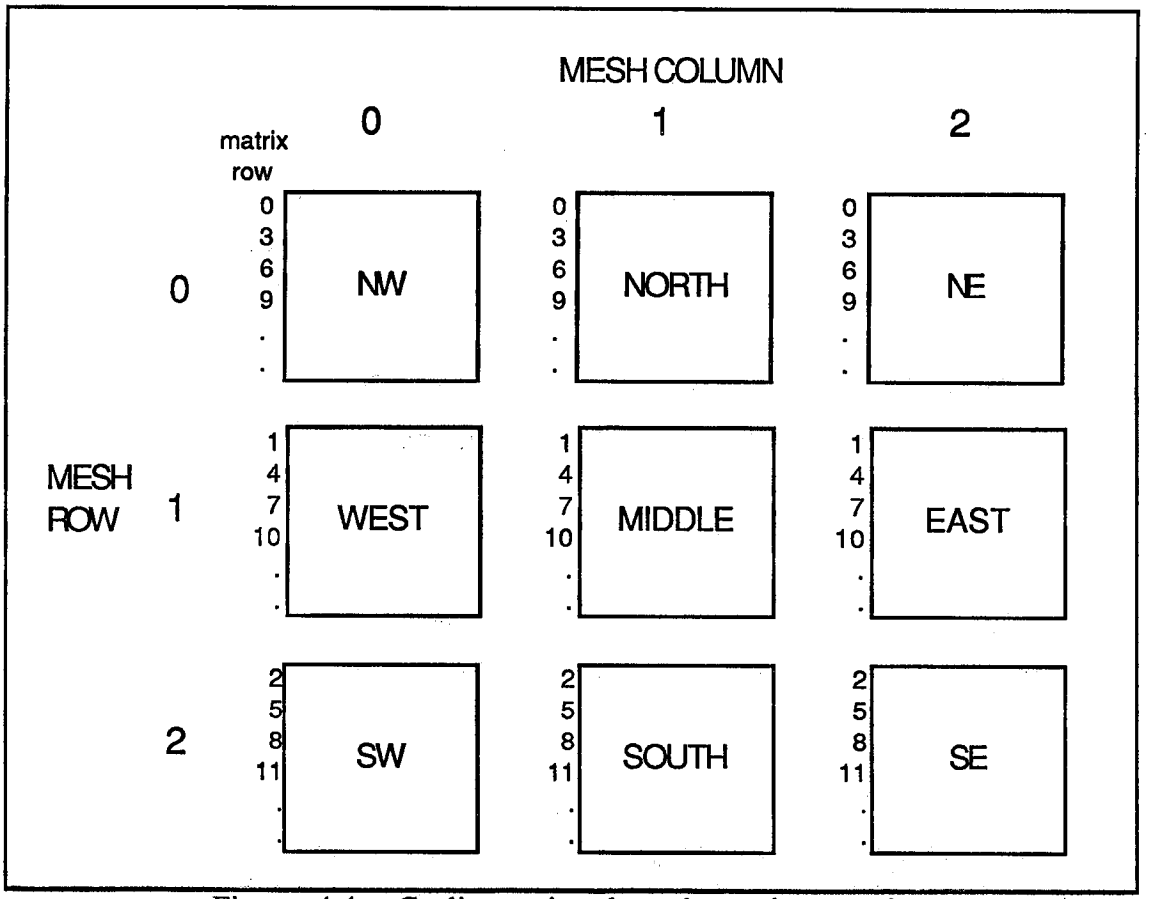


Figure 4.4 Cyclic or interleaved matrix mapping

The solution code performs Gaussian elimination with partial pivoting and backsubstitution on a square (rectangular) mesh, with the matrix stored as described above. The code terminates gracefully on completion, and also recognizes singular matrices.

Each mesh node stores all information describing that node's part of the LS matrix. During the Gaussian elimination computation, the location of the pivot row and hence the direction to send candidate pivot rows can be determined for each mesh node. In addition, each mesh node is synchronized with the others during the matrix building, Gaussian elimination, and backsubstitution stages

of computation. The pseudocode for Gaussian elimination with partial pivoting for a mesh node is given in Figure 4.5. It is described below with illustrations.

```
For each row in the LS matrix
  {
  If (this processor contains the pivot column)
    {
    find pivot and local pivot row index (prndx)
    send pivot and prndx horizontally
    }
  else
    {
    receive pivot and prndx horizontally
    send pivot and prndx horizontally
    }
  If (pivot row maps to this processor)
    {
    receive candidate pivots and pivot rows vertically
    determine pivot and pivot row
    divide pivot row by pivot
    If (pivot row is not from local list)
      {
      swap pivot row and local row
      send swapped row to source of pivot row
      }
    broadcast pivot row vertically
    }
  else
    {
    receive candidate pivot and pivot row vertically
    determine larger absolute value pivot
    send larger pivot and corresponding row vertically toward
         the processor to which the pivot row maps
    }
  If (this processor contains the pivot column)
    {
    For each uneliminated row on this processor
      {
      send element in pivot column horizontally
      eliminate row
      }
    }
  else
    {
    For each uneliminated row on this processor
      {
      receive element in pivot column horizontally
      send element in pivot column horizontally
      eliminate row
      }
    }
  }
```

Figure 4.5 Pseudocode for Gaussian elimination with partial pivoting for one mesh node

Once the LS matrix is built, a message is sent to all mesh nodes and the Gaussian elimination computation begins. For each node in the pivot column, a pivot is chosen from that node's list of rows remaining to be eliminated and the pivot row number and pivot are passed horizontally to the other nodes in the mesh row (see Figure 4.6).

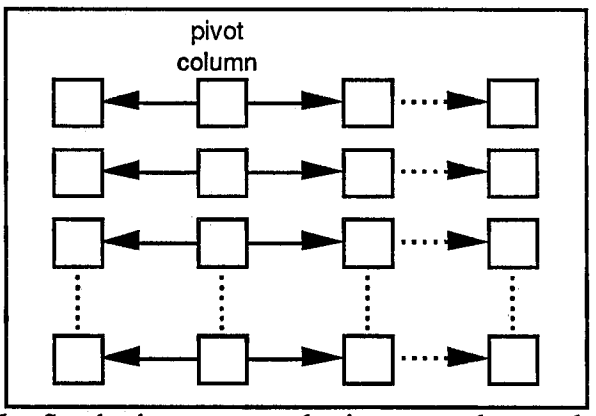


Figure 4.6 Send pivot row and pivot to other nodes in row

Next, a candidate pivot and its corresponding partial row are received by each node (except those on the bottom or top edges, depending on the location of the pivot row). The received candidate pivot is compared to the node's own pivot. The pivot with the largest absolute value, its corresponding partial row, and its mesh row number are sent toward the node containing the pivot row (see Figure 4.7). The pivot row is located on node (i mod p), where i is the current pivot column and p is the number of nodes on a side of the square mesh.

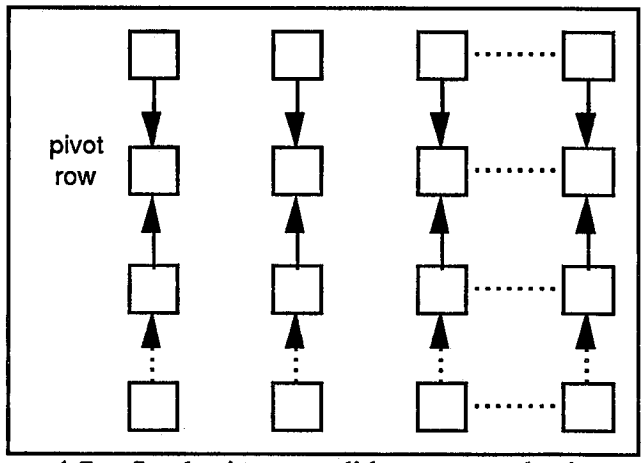


Figure 4.7 Send pivot candidates toward pivot row

When all candidates have been received and a pivot chosen, the row is swapped and divided by the pivot (again, each node in the grid row has the pivot). If the pivot row came from another mesh row, then the swapped row is first sent back to that mesh row. Next, the pivot row is sent to all other mesh rows (see Figure 4.8).

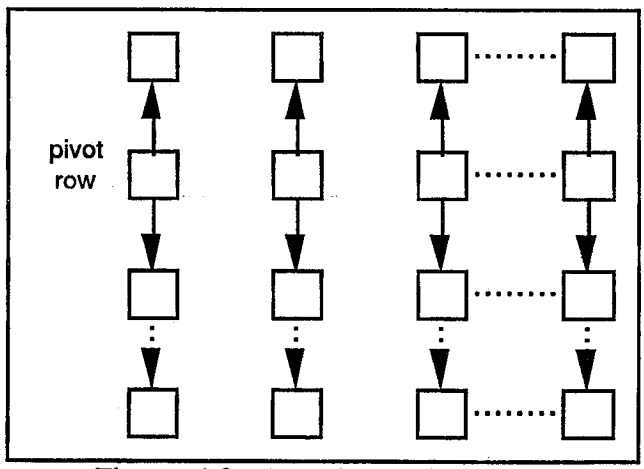


Figure 4.8 Broadcast pivot row

The elimination begins with the nodes in the pivot column. As each row is eliminated, the value of the element in that row's pivot column is sent to the other nodes in the mesh row (see Figure 4.9).

In this way elimination can be performed on those nodes which are not in the pivot column. This process is continued until either the pivot is zero (singular matrix) or until the reduced-row-echelon matrix is built.

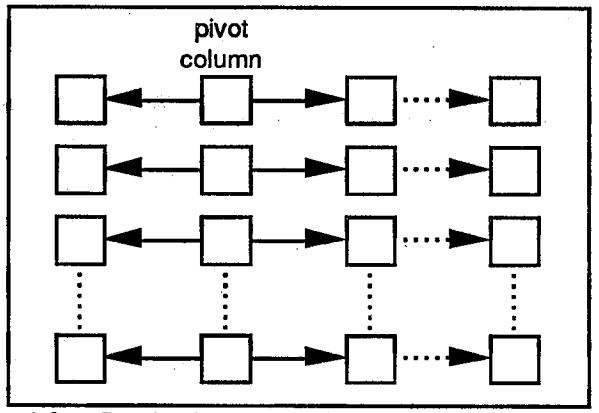


Figure 4.9 Send element value to each node in row

Backsubstitution begins on the node containing row n, where n is the number of rows in the LS matrix. The value for x_n is $mat[n][n+1]$, where x_n is an element of the vector **X** that is the solution of the matrix equation $\mathbf{AX} = \mathbf{B}$. Due to the interleaved matrix mapping (see Figure 4.4), no two consecutive rows of the matrix lie on the same mesh row. Because of this, each node in the mesh has a copy of **X**, and as each element of **X** is determined, its value is passed vertically to all other nodes in the mesh column. After each node receives a solution vector element, that element is multiplied by each element in that solution vector element's corresponding coefficient matrix column. Each of these products is subtracted from its corresponding **B** vector element, and each element in the coefficient matrix column is set to zero. Due to the

matrix mapping, the backsubstitution occurs on only one mesh column at a time. Backsubstitution begins on the east mesh column and, when that mesh column's coefficients have been exhausted, continues on the next column to the west. When the calculation shifts from mesh column to mesh column, both the solution vector **X** and the vector **B** are passed horizontally from the nodes to the east to the nodes to the west. The backsubstitution can then continue as before. Due to this message passing scheme, at the completion of backsubstitution each node on the west edge of the mesh contains a complete copy of the solution vector, **X**.

Multiprocessor Configuration

The code described above can be implemented on a family of configurations made up of T800 transputers. All configurations, however, share a number of topological and functional characteristics. First, all include a tree whose root node is connected to the host computer and whose leaf nodes execute the SF code. These configurations also include a second tree which is identical to the first and which shares the first tree's leaf nodes. Next, all contain a third tree whose root node is connected to (or is the same as) the root node of the second tree, and whose leaf nodes form the left side of a square mesh. Finally, all contain a square mesh where the LS matrix is built, stored, and solved (see Figure 4.10).

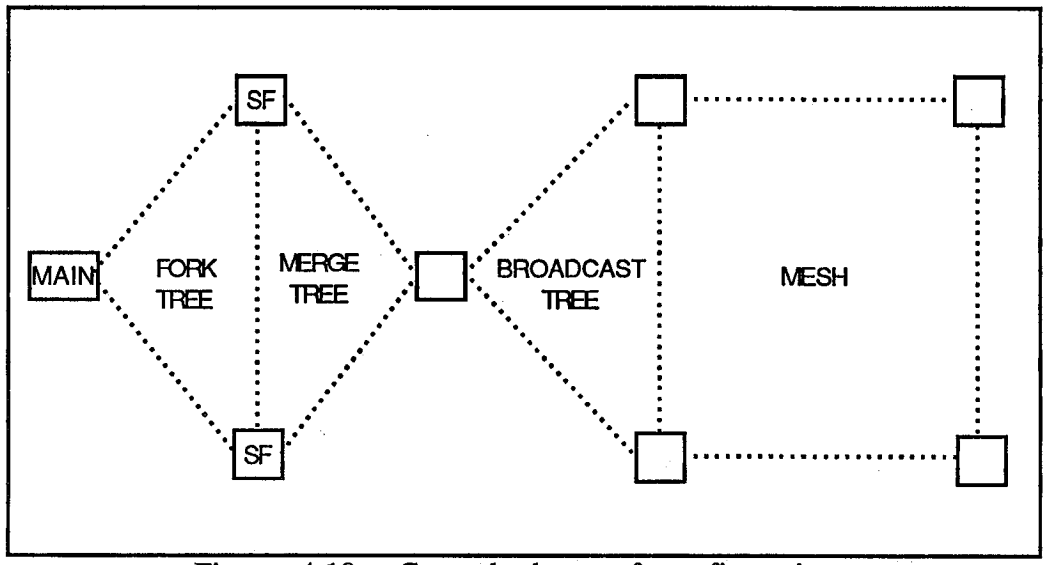


Figure 4.10 General shape of configurations

A scalable message-passing scheme was designed for the family of configurations described above. When passing data or control information from the MAIN root node to the mesh nodes, all intermediate nodes are traversed. However, when passing data or control information back to the MAIN root node from the mesh nodes, the message passing scheme is different. From the mesh nodes to the BROADCAST root node, only a single path is required to transmit the data. From the MERGE root node to the SF nodes, all intermediate nodes are traversed in order to give each SF node a copy of the shifts. And from the SF nodes to the MAIN root node, only a single path is required. Because of the semantics of discriminated parallel input, a single path is required to avoid the possibility of deadlock.

A numbering system was designed for the nodes in the families of configurations described by Figure 4.10 so as to ensure

the scalability of the message-passing scheme. Each SF, FORK, MERGE, and BROADCAST node is assigned two numbers describing its position in the topology (see Figure 4.11). These two numbers are designated the group id and the node id. For the SF nodes, group 0 is the group of nodes connected to the topmost FORK and MERGE nodes. Group 1 is the next group down, and so on. For the FORK nodes, group 0 is the group of nodes connected to the topmost previous FORK node (or to the MAIN node if there is no previous FORK node). Similarly, for the BROADCAST nodes, group 0 is the group of nodes connected to the topmost previous BROADCAST node (or to the root MERGE node if there is no previous BROADCAST node). For the MERGE nodes, group 0 is the group of nodes connected to the topmost following MERGE node (or to the root BROADCAST node if there is no following MERGE node). Within each group, the topmost node is node 0, the node below it is node 1, and so on.

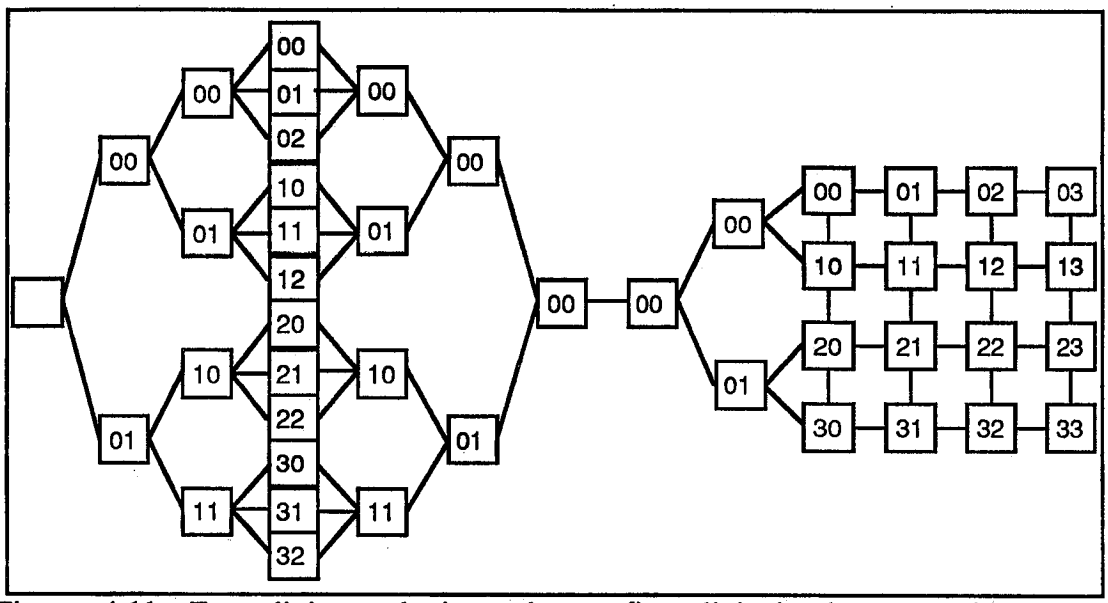


Figure 4.11 Two digit numbering scheme: first digit is the group id, second digit is the node id

The mesh has a numbering scheme which is independent of the rest of the configuration (see Figure 4.11). Each mesh node is assigned row and column numbers. Columns are numbered from left to right beginning with 0, while rows are similarly numbered from top to bottom.

The scalable message-passing scheme uses the group and node id numbers to avoid the possibility of deadlock. While information flows from the MAIN root node to the mesh nodes, control messages travel only through those MERGE nodes whose node id is 0. Data messages flow through all nodes from the MAIN root node to the mesh nodes. Similarly, while information flows from the mesh nodes to the MAIN root node, control messages travel through only those BROADCAST and FORK nodes whose node

id is 0. Data messages travel through only those nodes whose group and node id are both 0.

V. DISCUSSION

Configurations

The hardware used in the research is a Transtech MCP1000 board containing sixteen 1Mbyte T800 transputers connected by 32-way crossbar switches (Figure 5.1). The board is connected to a Sun SPARCServer 330 host computer.

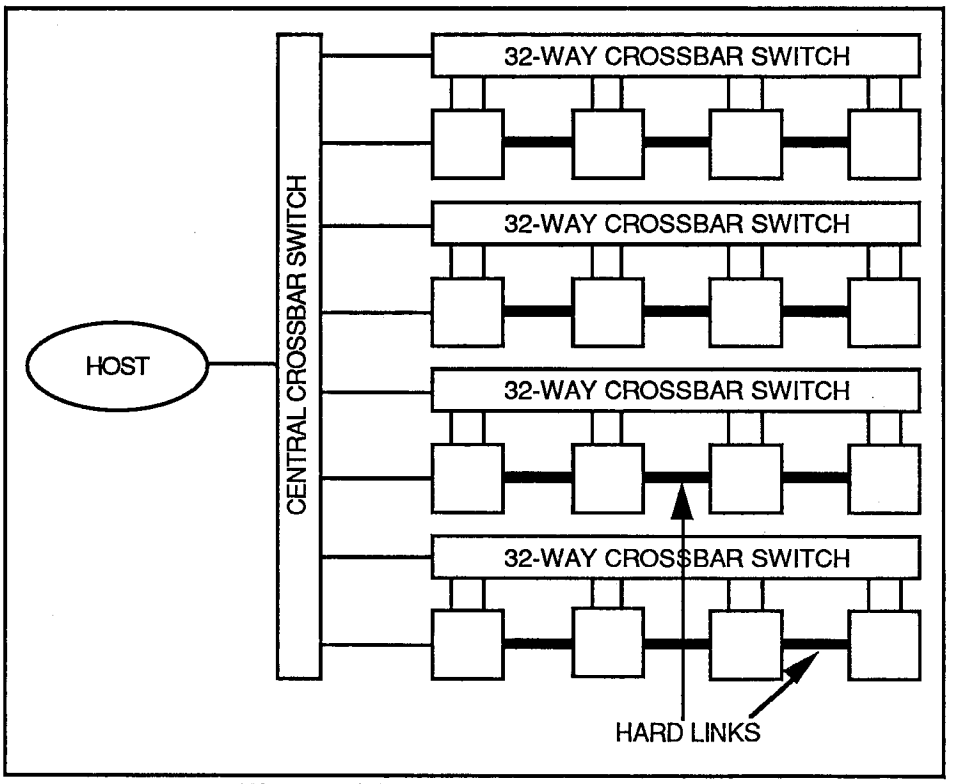


Figure 5.1 Transtech hardware

There are four banks of four processors. Each processor has one or two of its links hardwired to another processor (hard links).

The other links are programmable (soft links) and can be connected to any other processor via the crossbar switches. In addition, the processors on the right and left edges have only three links (1 hard link, 2 soft links). The processors on the left edge have their fourth links reserved for connection to the host computer. The fourth links on the processors on the right edge are not usable. Because of this limited connectivity, only four different configurations were used in the research. The configurations used in the research are shown in Figure 5.2.

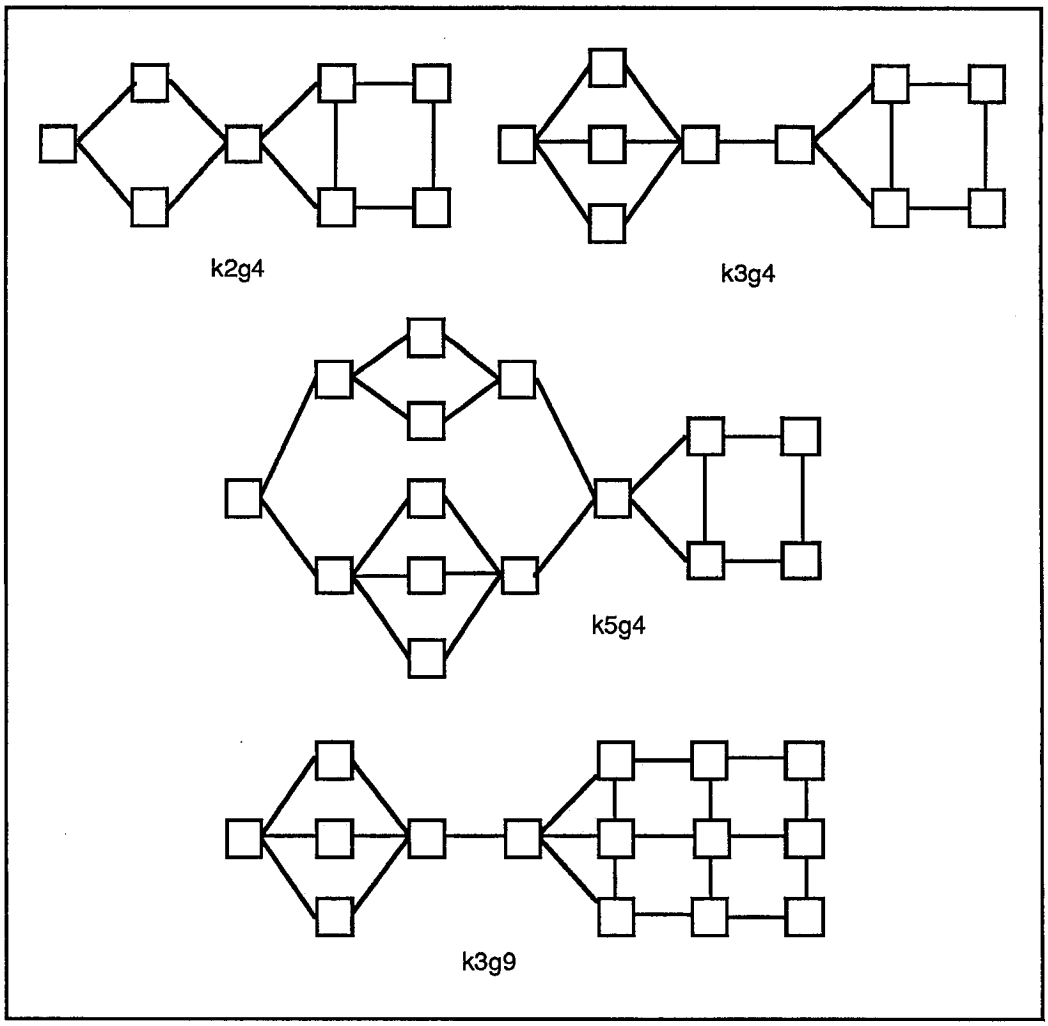


Figure 5.2 Experimental processor configurations

As discussed previously, each of the configurations in Figure 5.2 consists of three trees and a square mesh.

Timing

Each T800 transputer has an internal clock which can be used for timing and scheduling processes. The clock ticks 15,625 times per second, so one tick equals 0.000064 seconds. The occam2 programming language contains a special channel type, TIMER,

which is used to read the time from this clock, in ticks. The clock was used to collect timing data for each of the different topologies. For each of the configurations, the time was measured for one cycle of isotropic refinement, one cycle of anisotropic refinement, and for the solution of the LS system. In addition, a serial version of the program was run on a single transputer as a timing standard. The time for the solution of the LS system was measured from the moment the LS matrix was built until the moment the solutions were determined. The time was measured on the NW processor in the mesh. The time for one cycle of refinement was measured on the root node from the moment the reflection file is opened until the moment the shifts are written to the output file. In addition, to measure the effect of I/O, the refinement timing tests were performed in the presence and absence of concurrent file output.

While timing the various configurations, it became clear that, due to the communications cost, sending more than one reflection at a time to each leaf node results in a faster solution. This is due to a delay caused when the leaf node has sent data toward the grid and must then input data from the root, and also to the fact that sending a group of reflections instead of a single reflection reduces the number of communications. After testing the different configurations, it was determined that sending reflections to the leaf nodes in groups of twelve was optimal. While this number was selected partially due to the type of structure being refined, it was

nevertheless used as the standard number of reflections sent for each configuration.

The Data

The test data was obtained from Dr. James Stewart, one of the authors of the XTAL system [14]. It is a standard test file used to check the accuracy of the XTAL system on installation. The test data contains 19 atoms in the asymmetric unit, and has the symmetry relations X,Y,Z, -X, -Y, -Z. This is known as P1 bar (or P bar 1) to crystallographers. The asymmetric unit consists of seven nitrogen atoms, seven hydrogen atoms, four oxygen atoms, and one carbon atom. During isotropic refinement, 77 parameters are refined: 4 per atom, plus the scale factor. In molecular refinement, hydrogen atoms are refined using only isotropic temperature parameters. Therefore, during anisotropic refinement, 137 parameters are refined: 9 per non-hydrogen atom, 4 per hydrogen atom, plus the scale factor.

The Program

The program reads data from two input files and writes data to one output file. To reduce the data conversion cost, all input and output files are binary. The first data file, *b1.bin*, contains the cell constants and atomic parameters. The second file, *b2.bin*, contains the reflection data.

All input data files are assumed to contain isotropic temperature parameters only; there is no provision for reading data files containing anisotropic temperature parameters. Associated with each atom in the data file is a binary code that indicates which parameters are to be refined and which are to be held constant during the refinement. During refinement, some atoms' positional parameters must not be modified when those atoms occupy a special position in the unit cell. Occasionally associated with atoms in special positions is an occupancy factor which, due to symmetry relations, is used to properly calculate those atoms' contributions to the overall scattering. The program, however, does not use the occupancy factor, so it can be used to refine only those structures not requiring occupancy factors for refinement.

Data Files

The file *b1.bin* contains the reciprocal unit cell constants a^*, b^*, c^*, $\cos(\alpha^*)$, $\cos(\beta^*)$, and $\cos(\gamma^*)$. For each atom in the structure, *b1.bin* contains the X, Y, and Z positional parameters, an isotropic temperature parameter, a binary refinement code, and an atom type. The atom type is the atom's index in the table of Cromer-Mann coefficients. Also included in *b1.bin* is a list of atom types found in the structure and a list of symmetry operations. Last, *b1.bin* contains the number of atoms in the asymmetric unit, the number of atom types, the number of symmetrically equivalent positions, and the number of reflections in the reflection file, *b2.bin*.

For each reflection, the file *b2.bin* contains the hkl Miller indices, the value of $(\sin\theta/\lambda)^2$, the intensity (I), the sigma of intensity (σI, used in weighting schemes), and an atomic scattering factor for each atom type in the molecular structure.

Limitations

The program uses a list of 45 Cromer-Mann coefficients [17] which includes ions ranging from neutral hydrogen up to chromium 3+. Atoms with higher atomic numbers cannot be handled by the current program. As stated above, the occupancy factor is not used, so the program is limited to refining those structures not requiring occupancy factors. The current program uses no weighting scheme; σI was included in *b2.bin* so that weighting could be added in the future. The current program is limited to 20 different atom types per structure. However, the limit is a constant and can easily be increased if necessary.

Table 5.1 shows how the storage requirements for the major data structures grow as the number of atoms is increased. The calculations are based on 10 parameters per atom and 4 bytes per number.

Number of atoms	Number of bytes: Augmented LS matrix	Atomic parameters	TOTAL BYTES
10	132,860	400	133,260
20	524,900	800	525,700
30	1,176,140	1,200	1,177,340
40	2,086,580	1,600	2,088,180
50	3,256,220	2,000	3,258,220
60	4,685,060	2,400	4,687,460
70	6,373,100	2,800	6,375,900
80	8,320,340	3,200	8,323,540
90	10,526,780	3,600	10,530,380
100	12,992,420	4,000	12,996,420
200	51,904,820	8,000	51,912,820
300	116,737,220	12,000	116,749,220
400	207,489,620	16,000	207,505,620
500	324,162,020	20,000	324,182,020
600	466,754,420	24,000	466,778,420
700	635,266,820	28,000	635,294,820
800	829,699,220	32,000	829,731,220
900	1,050,051,620	36,000	1,050,087,620
1,000	1,296,324,020	40,000	1,296,364,020

Table 5.1 Storage requirements for major data structures

Because of the size of the LS matrix, the configurations containing a 4-processor mesh can accommodate only those data sets which contain fewer than about 55 atoms. The k3g9 configuration can accommodate data sets containing fewer than about 85 atoms.

The theoretical upper limit of atoms that the current method can accommodate is between 20,000 and 30,000 atoms, depending on the size of the code segment. This is due to the requirement that each SF node possess a copy of all the atomic parameters. This

approximation is based on the assumption that each atom will have 10 parameters and each parameter will require 4 bytes for storage.

VI. RESULTS

The speedup of a parallel solution is defined as the single processor execution time divided by the multiprocessor execution time [4]. Optimal speedup (linear speedup) occurs when the speedup is equal to the number of processors in the multiprocessor solution. All parallel programs incur some communications overhead, and most also contain code that cannot be executed in parallel. As a consequence, linear speedup is almost never realized. Processor efficiency is reported as a percentage of linear speedup [4]. It is calculated by dividing the speedup by the linear speedup and multiplying by 100. Table 6.1 contains the results of timing studies for one isotropic refinement cycle and one anisotropic refinement cycle.

	single processor	k2g4	k3g4	k5g4	k3g9
isotropic					
time(sec)	103.3	18.11	18.24	18.58	9.07
speedup	1	5.70	5.66	5.56	11.4
efficiency	100%	71.3%	56.6%	37.1%	76.0%
anisotropic					
time(sec)	264.7	55.92	56.03	56.56	26.53
speedup	1	4.73	4.72	4.68	9.98
efficiency	100%	59.1%	47.2%	31.2%	66.5%

Table 6.1 Timing results for refinement cycles

As shown in Table 6.1, the 15-processor k3g9 configuration attained the greatest speedup. This result was expected, because the $O(n^2)$ BUILD process was distributed among nine processors in this solution and among only four in the other solutions. From the timing studies it can be concluded that the BUILD process is the rate-determining step in those topologies containing a 4-processor mesh. This can be seen by noting that the k2g4, k3g4, and k5g4 configurations had similar runtimes, though each has a different number of leaf nodes performing the SF process. Also, if the BUILD process were not the rate-determining step, the k3g9 solution would not have run any faster than the k3g4 solution.

Each refinement cycle is a pipelined process in which reflection data is continuously sent from the MAIN root node through the trees and into the mesh. The facts that the k2g4 configuration ran slightly faster than the k3g4, and the k3g4 slightly faster than the k5g4 are due to the different startup costs of filling the pipeline.

	single processor	4 processor mesh	9 processor mesh
isotropic			
time(sec)	1.352	.4489	.2888
speedup	1	3.01	4.68
efficiency	100%	75.3%	52.0%
anisotropic			
time(sec)	7.281	2.207	1.325
speedup	1	3.30	5.50
efficiency	100%	82.5%	61.1%

Table 6.2 Timing results for LS solution

Table 6.2 contains the timing results for the solution of the LS system. The results clearly show the great disparity in computational effort between building the LS matrix and solving the LS matrix for small molecules. The efficiency which is reported in Table 6.2 was calculated by dividing the speedup by the number of processors in the mesh only. This illustrates the claim in [1] that the solution method reported in that paper is more efficient when the number of matrix rows is much greater than the number of processors. Because all configurations containing a 4-processor mesh had virtually identical matrix solution times, only one time is reported for a 4-processor configuration.

	single processor	k2g4	k3g4	k5g4	k3g9
isotropic					
time(sec)	103.0	18.05	18.08	18.43	8.88
speedup	1	5.71	5.70	5.59	11.6
efficiency	100%	71.4%	57.0%	37.3%	77.3%
anisotropic					
time(sec)	264.2	55.80	55.84	56.38	26.35
speedup	1	4.73	4.73	4.69	10.0
efficiency	100%	59.1%	47.3%	31.3%	66.7%

Table 6.3 Timing results for refinement cycles with no output

Finally, Table 6.3 contains the timing results for refinement cycles during which no data was output. These measurements were done to show more clearly the division of labor between building and solving the LS matrix. They were also done to show the need to reduce the impact of I/O, perhaps by buffering.

VII. CONCLUSIONS

As noted in [11], building the LS matrix is often a far greater computational task than solving the LS matrix. Using the test data, the timing studies confirm this point dramatically. In the case of isotropic refinement on a single processor, 98.7% of a cycle was dedicated to building the LS matrix. The cost dropped to 97.2% in the anisotropic case. Clearly, as the number of atoms, n, increases, the $O(n^3)$ operation of solving the matrix should become the rate-determining step. It is clear that it is desirable to distribute the LS matrix for both small molecule and macromolecular refinement; this distribution reduces both the construction and solution time for the LS system.

This research has shown that significant reduction in refinement times can be realized using distributed memory multiprocessor computers. The refinement times were reduced by a factor of 10 in the fastest configuration built. The research also highlights the importance of scalable, modular code. Programs running on four different topologies were written using essentially the same code. In addition, larger processor networks could have been built had more processors been available. The research has also shown that careful load balancing is essential for good performance of a parallel processing solution. Though both the

k5g4 and the k3g9 configurations contained 15 processors, the code ran significantly faster on k3g9 due to better load balancing. Finally, the research has shown one of the weaknesses of multiprocessor computers - limited processor connectivity. T800 transputers have only 4 links per processor. Paradigms such as the processor farm must use some of the processors solely as one-step buffers to distribute or collect data. Greater processor connectivity would increase the amount of work being done by each processor and increase the efficiency of the overall system.

In retrospect, an even faster and simpler solution can be created by connecting the processors as shown in Figure 7.1. Currently the parameter shifts are passed back through the network from the mesh to the main root node. This requires more communication and more code to handle the reverse flow. Because the NW node uses only three links, its fourth link can be connected directly to the main root node. The shifts then travel across only one link, thereby reducing the communications cost. This also removes the need for the tree numbering system while simplifying and shortening the code.

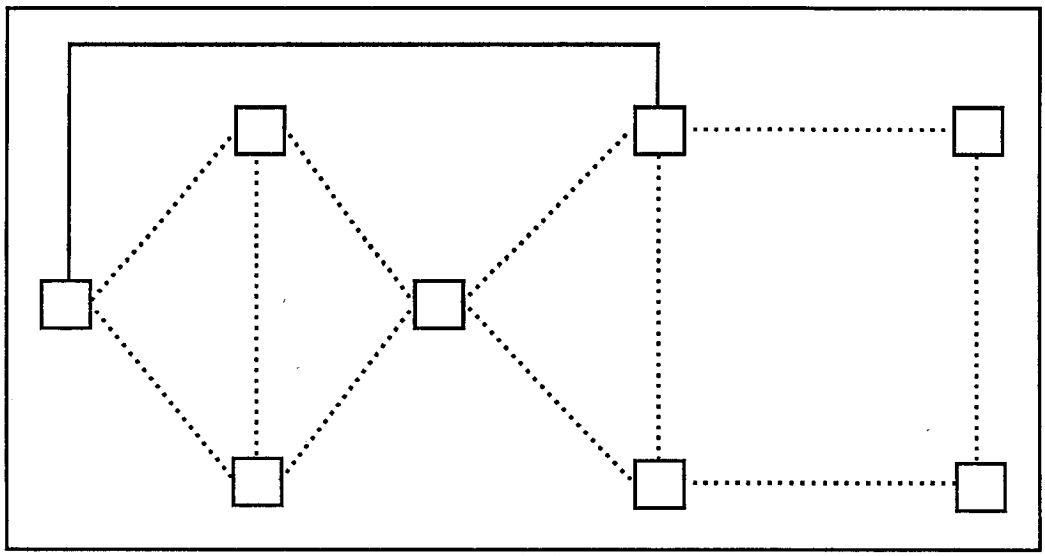


Figure 7.1 Proposed design improvement

Another design decision that will decrease refinement times is to add a processor between the host computer and the MAIN root node, and to load the entire reflection file onto that processor. Because the entire reflection file is read once per refinement cycle, the disk input cost would be significantly reduced. The added processor can also be used to perform output while simultaneously supplying the MAIN root node with reflection data. Clearly, this will be practical only if the added processor's memory is large enough to accommodate the file.

In conclusion, macromolecular refinements can be performed on distributed memory multiprocessor computers by applying the design strategy used in the research. The memory requirements are much greater for the macromolecular case, so the 1 Mbyte transputers used in the research would not be adequate. However,

transputers possessing as much as 16 Mbyte are manufactured, and these should suffice for this purpose.

BIBLIOGRAPHY

1. Bisselling, R.H., and van de Vorst, J.G.G. Parallel LU Decomposition on a Transputer Network. In *Proceedings of Parallel Computing 1988*, (June 1988) Springer-Verlag, New York, NY, pp. 61 - 77.

2. Buerger, M.J. *Contemporary Crystallography*. McGraw-Hill, New York, NY,1970.

3. Chu, E., and George, A. Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor. *Parallel Computing*. **5** (1987) 65.

4. Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., and Walker, D.W. *Solving Problems on Concurrent Processors (Vol. 1)* Prentice-Hall, Englewood Cliffs, NJ, 1988.

5. Galletly, J. *Occam 2*. Pitman Press, London,1990.

6. Haneef, I., Moss, D.S., Stanford, M.J., and Borkakoti, N. Restrained Structure-Factor Least-Squares Refinement of Protein Structures Using a Vector Processing Computer. *Acta Crystallographica* . A**41** (1985) 426.

7. Hoare, C.A.R. *Communicating Sequential Processes*. Prentice/Hall International, Englewood Cliffs, NJ, 1985.

8. Ibers, J.A., and Hamilton, W.C., editors. *International Tables for Crystallography, volume IV*. The Kynoch Press, Birmingham, England, 1974.

9. INMOS Limited. *Transputer Reference Manual*. Prentice Hall, New York, NY, 1988.

10. Konnert, J.H. A Restrained-Parameter Structure-Factor Least-Squares Refinement Procedure for Large Asymmetric Units. *Acta Crystallographica.* A32 (1976) 614.

11. Laskowski, R.A., Driessen, H.P.C., and Moss, D.S. Protein structure refinement using transputers. In *Proceedings of Applications of Transputers 1*, (August 23-25, Liverpool, UK). IOS Press, Amsterdam, 1990, pp. 223 - 229.

12. Robinson, W., and Sheldrick, G.M. SHELX. In *Proceedings of Crystallographic Computing 4*, (August 22-29, Adelaide, Australia). Oxford U. Press, New York, NY,1988, pp. 366-380.

13. Rollett, J.S. Basic Processes Involved in Least Squares and Fourier Refinement. In *Proceedings of Computational Crystallography*, (August 7-15, Ottawa, Canada). Clarendon Press, Oxford <Oxfordshire>, 1982, pp 338-353.

14. Skelton, B., Hall, S., and Stewart, J. The XTAL system: an application primer. In *Proceedings of Crystallographic Computing 4*, (August 22-29, Adelaide, Australia). Oxford U. Press, New York, NY,1988, , pp 325-358.

15. Stout, G.H., and Jensen, L.H. *X-ray Structure Determination: A Practical Guide.* John Wiley & Sons, New York, NY, 1990.

16. Watkin, D. Least Squares Refinement. In *Proceedings of Crystallographic Computing 3: Data Collection, Structure Determination, Proteins, and Databases*, (July 30 - August 8, Milheim an der Ruhr, FRG). Clarendon Press, Oxford, 1985, pp. 245-254.

17. Watkin, D. CRYSTALS, for teaching and research. In *Proceedings of Crystallographic Computing 4*, (August 22-29, Adelaide, Australia). Oxford U. Press, New York, NY,1988, pp 354-365.

18. Wyckoff, H.W., Hirs, C.H.W., and N. Timasheff, S.N. *Diffraction Methods for Biological Macromolecules. Part A.* Academic Press, Orlando, 1985.

APPENDICES

APPENDIX A

Sample Data File (truncated)

```
Stewart
*lambda      a        b        c      alpha   beta   gamma
 1.54178   7.097    7.368    8.251    105.34  94.59  115.12
************* SYMMETRY RELATIONS ***************
SYMM       X,       Y,       Z
SYMM      -X,      -Y,      -Z
**** list of atom types
SFAC O  N  C  H
* type    X         Y         Z    occupancy  Uiso     refinement
N       0.39397   0.23601   0.16233   1.0    0.05132   1111111111
N       0.19985   0.23432   0.16906   1.0    0.05405   1111111111
O       0.14591   0.20634   0.30355   1.0    0.07116   1111111111
O       0.09662   0.26290   0.06570   1.0    0.08136   1111111111
N       0.45833   0.24393   0.01254   1.0    0.05027   1111111111
O       0.35384   0.23129  -0.11680   1.0    0.07574   1111111111
O       0.64126   0.26260   0.02087   1.0    0.05471   1111111111
C       0.69280   0.25947   0.55464   1.0    0.04994   1111111111
N       0.76471   0.25330   0.41207   1.0    0.05492   1111111111
H       0.88595   0.25281   0.41031   1.0    0.05381   1111000000
H       0.67619   0.24631   0.30998   1.0    0.08047   1111000000
N       0.50848   0.26302   0.56139   1.0    0.06545   1111111111
H       0.41856   0.26076   0.45611   1.0    0.11101   1111000000
H       0.46564   0.25465   0.63396   1.0    0.02920   1111000000
N       0.80535   0.26348   0.69229   1.0    0.06464   1111111111
H       0.76254   0.27221   0.76265   1.0    0.04321   1111000000
N       1.00252   0.26616   0.68627   1.0    0.05469   1111111111
H       1.00705   0.16546   0.71928   1.0    0.10296   1111000000
H       1.10681   0.38207   0.76626   1.0    0.12719   1111000000
**  H     K     L  Intensity   sigI
    0     0     1    12.633   0.00000
    0     0     2    12.503   0.00000
    0     0     3    16.346   0.00000
    0     0     4     0.861   0.00000
    .     .     .       .        .
    .     .     .       .        .
```

APPENDIX B

Sample Isotropic Refinement

CYCLE 0

SCALE FACTOR is 1.00000

X	Y	Z	Uiso
0.39397	0.23601	0.16233	0.05132
0.19985	0.23432	0.16906	0.05405
0.14591	0.20634	0.30355	0.07116
0.09662	0.26290	0.06570	0.08136
0.45833	0.24393	0.01254	0.05027
0.35384	0.23129	-0.11680	0.07574
0.64126	0.26260	0.02087	0.05471
0.69280	0.25947	0.55464	0.04994
0.76471	0.25330	0.41207	0.05492
0.88595	0.25281	0.41031	0.05381
0.67619	0.24631	0.30998	0.08047
0.50848	0.26302	0.56139	0.06545
0.41856	0.26076	0.45611	0.11101
0.46564	0.25465	0.63396	0.02920
0.80535	0.26348	0.69229	0.06464
0.76254	0.27221	0.76265	0.04321
1.00252	0.26616	0.68627	0.05469
1.00705	0.16546	0.71928	0.10296
1.10681	0.38207	0.76626	0.12719

CYCLE 1

SCALE FACTOR is 0.99711

X	Y	Z	Uiso
0.39397	0.23601	0.16233	0.05132
0.19985	0.23432	0.16906	0.05405
0.14591	0.20634	0.30355	0.07116
0.09662	0.26290	0.06570	0.08136
0.45833	0.24393	0.01254	0.05027
0.35384	0.23129	-0.11680	0.07574
0.64126	0.26260	0.02087	0.05471
0.69280	0.25947	0.55464	0.04994
0.76471	0.25330	0.41207	0.05492
0.88595	0.25281	0.41031	0.05381
0.67619	0.24631	0.30998	0.08047
0.50848	0.26302	0.56139	0.06545
0.41856	0.26076	0.45611	0.11101
0.46564	0.25465	0.63396	0.02920
0.80535	0.26348	0.69229	0.06464
0.76254	0.27221	0.76265	0.04321
1.00252	0.26616	0.68627	0.05469
1.00705	0.16546	0.71928	0.10296
1.10681	0.38207	0.76626	0.12719

R-FACTOR IS 0.2915

CHANGE IN PARAMETERS

Scale Factor -0.00462

X	Y	Z	Uiso
-0.00009	0.00014	-0.00006	0.00022
-0.00000	-0.00009	-0.00028	-0.00004
-0.00064	-0.00040	-0.00059	-0.00001
-0.00025	-0.00003	0.00037	-0.00032
0.00019	0.00034	-0.00034	-0.00012
-0.00032	-0.00024	0.00002	-0.00047
0.00024	0.00007	-0.00009	-0.00012
-0.00025	-0.00013	-0.00028	-0.00031
0.00029	-0.00004	-0.00015	-0.00006
-0.00191	-0.00051	-0.00472	0.00156
0.00362	0.00761	0.01183	0.01057
0.00017	-0.00013	-0.00031	-0.00041
0.00918	-0.00214	0.01021	0.00245
-0.00269	-0.00022	-0.00232	-0.00273
0.00006	0.00032	0.00004	0.00025
-0.00028	-0.00121	-0.00122	0.00391
0.00009	0.00018	0.00049	-0.00018
0.01321	0.01400	0.00264	-0.00627
0.01771	0.02090	-0.00255	0.00291

CYCLE 2

SCALE FACTOR is 0.99249

X	Y	Z	Uiso
0.39388	0.23615	0.16227	0.05154
0.19985	0.23423	0.16878	0.05401
0.14527	0.20594	0.30296	0.07115
0.09637	0.26287	0.06607	0.08104
0.45852	0.24427	0.01220	0.05015
0.35352	0.23105	-0.11678	0.07527
0.64150	0.26267	0.02078	0.05459
0.69255	0.25934	0.55436	0.04963
0.76500	0.25326	0.41192	0.05486
0.88404	0.25230	0.40559	0.05537
0.67981	0.25392	0.32181	0.09104
0.50865	0.26289	0.56108	0.06504
0.42774	0.25862	0.46632	0.11346
0.46295	0.25443	0.63164	0.02647
0.80541	0.26380	0.69233	0.06489
0.76226	0.27100	0.76143	0.04712
1.00261	0.26634	0.68676	0.05451
1.02026	0.17946	0.72192	0.09669
1.12452	0.40297	0.76371	0.13010

R-FACTOR IS 0.3076

CHANGE IN PARAMETERS

Scale Factor 0.00032

X	Y	Z	Uiso
-0.00004	-0.00003	-0.00001	0.00003
-0.00004	-0.00003	-0.00001	0.00002
-0.00003	-0.00001	-0.00001	-0.00000
-0.00001	-0.00003	-0.00001	-0.00000
-0.00002	-0.00001	-0.00000	0.00004
-0.00003	-0.00005	-0.00002	0.00003
0.00002	0.00003	0.00001	0.00002
0.00002	0.00004	0.00001	0.00002
0.00001	-0.00000	-0.00005	0.00003
0.00024	0.00024	0.00028	-0.00023
-0.00034	0.00074	-0.00002	-0.00005
-0.00000	0.00003	0.00003	0.00011
-0.00014	-0.00084	0.00005	-0.00004
-0.00008	-0.00003	0.00020	0.00068
0.00005	0.00003	0.00002	0.00003
0.00018	0.00031	0.00005	0.00027
-0.00000	0.00002	0.00003	0.00003
0.00032	-0.00137	-0.00075	0.00155
0.00233	-0.00018	-0.00173	0.00076

CYCLE 3

SCALE FACTOR is 0.99281

X	Y	Z	Uiso
0.39385	0.23612	0.16227	0.05157
0.19981	0.23420	0.16877	0.05403
0.14524	0.20593	0.30296	0.07115
0.09635	0.26283	0.06606	0.08104
0.45849	0.24426	0.01220	0.05019
0.35348	0.23101	-0.11680	0.07529
0.64152	0.26270	0.02079	0.05461
0.69257	0.25937	0.55437	0.04965
0.76501	0.25325	0.41188	0.05489
0.88428	0.25254	0.40587	0.05515
0.67947	0.25466	0.32180	0.09099
0.50864	0.26291	0.56111	0.06515
0.42760	0.25777	0.46637	0.11342
0.46287	0.25440	0.63184	0.02715
0.80546	0.26383	0.69235	0.06492
0.76244	0.27131	0.76148	0.04739
1.00260	0.26636	0.68679	0.05453
1.02058	0.17810	0.72117	0.09824
1.12686	0.40279	0.76198	0.13086

R-FACTOR IS 0.3079

CHANGE IN PARAMETERS

Scale Factor 0.00004

X	Y	Z	Uiso
-0.00000	-0.00000	-0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000
-0.00000	0.00000	-0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
0.00002	0.00003	-0.00000	-0.00001
0.00002	0.00004	0.00000	-0.00008
-0.00000	0.00000	0.00000	0.00000
0.00008	0.00001	-0.00004	0.00018
-0.00002	-0.00001	-0.00000	0.00002
0.00000	-0.00000	0.00000	0.00000
0.00001	0.00002	0.00000	0.00002
0.00001	0.00002	0.00001	0.00000
0.00024	0.00040	0.00010	-0.00036
0.00074	0.00073	0.00009	0.00117

CYCLE 4

SCALE FACTOR is 0.99284

X	Y	Z	Uiso
0.39385	0.23612	0.16227	0.05157
0.19981	0.23420	0.16877	0.05404
0.14524	0.20592	0.30296	0.07115
0.09635	0.26283	0.06606	0.08104
0.45849	0.24426	0.01220	0.05019
0.35348	0.23101	-0.11680	0.07530
0.64152	0.26270	0.02079	0.05461
0.69257	0.25937	0.55437	0.04965
0.76501	0.25325	0.41188	0.05489
0.88430	0.25257	0.40586	0.05513
0.67949	0.25470	0.32180	0.09092
0.50864	0.26291	0.56111	0.06516
0.42768	0.25778	0.46632	0.11360
0.46285	0.25439	0.63184	0.02717
0.80546	0.26383	0.69235	0.06492
0.76245	0.27133	0.76148	0.04741
1.00261	0.26637	0.68680	0.05453
1.02082	0.17850	0.72127	0.09789
1.12760	0.40351	0.76207	0.13203

R-FACTOR IS 0.3080

CHANGE IN PARAMETERS

Scale Factor 0.00001

X	Y	Z	Uiso
0.00000	-0.00000	0.00000	0.00000
0.00000	0.00000	-0.00000	0.00000
-0.00000	0.00000	0.00000	0.00000
-0.00000	0.00000	-0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000
0.00000	-0.00000	-0.00000	0.00000
0.00000	-0.00000	-0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
-0.00000	-0.00000	-0.00000	0.00000
-0.00000	0.00000	0.00000	0.00000
-0.00000	-0.00002	-0.00001	0.00001
-0.00000	-0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
-0.00004	-0.00003	-0.00002	0.00011
0.00017	0.00006	-0.00009	0.00025

CYCLE 5

SCALE FACTOR is 0.99285

X	Y	Z	Uiso
0.39385	0.23612	0.16227	0.05157
0.19981	0.23420	0.16877	0.05404
0.14524	0.20592	0.30296	0.07115
0.09635	0.26283	0.06606	0.08104
0.45849	0.24426	0.01220	0.05019
0.35348	0.23101	-0.11680	0.07530
0.64152	0.26270	0.02079	0.05461
0.69257	0.25937	0.55437	0.04966
0.76501	0.25325	0.41188	0.05489
0.88431	0.25258	0.40587	0.05514
0.67949	0.25470	0.32179	0.09092
0.50864	0.26291	0.56111	0.06516
0.42768	0.25776	0.46632	0.11361
0.46285	0.25439	0.63184	0.02717
0.80546	0.26383	0.69235	0.06492
0.76245	0.27133	0.76148	0.04742
1.00261	0.26638	0.68680	0.05453
1.02078	0.17846	0.72125	0.09800
1.12777	0.40357	0.76198	0.13228

R-FACTOR IS 0.3080

CHANGE IN PARAMETERS

Scale Factor 0.00000

X	Y	Z	Uiso
0.00000	-0.00000	0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
-0.00000	0.00000	-0.00000	0.00000
-0.00000	0.00000	-0.00000	0.00000
-0.00000	0.00000	-0.00000	0.00000
0.00000	-0.00000	-0.00000	0.00000
0.00000	-0.00000	-0.00000	0.00000
0.00000	-0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
-0.00000	-0.00000	0.00000	-0.00000
-0.00000	0.00000	0.00000	0.00000
0.00000	-0.00000	-0.00000	0.00000
-0.00000	-0.00000	0.00000	0.00000
0.00000	-0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000
0.00001	0.00002	0.00000	-0.00002
0.00006	0.00005	-0.00000	0.00009