electronic reprint

Journal of
Applied Crystallography
ISSN 0021-8898

Parallel processing for Rietveld refinement

Todd R. Zeitler and Brian H. Toby

Journal of
Applied Crystallography
ISSN 0021-8898

Received 1 October 2001
Accepted 2 January 2002

Parallel processing for Rietveld refinement

Todd R. Zeitler and Brian H. Toby*

NIST Center for Neutron Research, National Institute of Standards and Technology, Gaithersburg, Maryland 20899-8562, USA. Correspondence e-mail: brian.toby@nist.gov

A method for speeding Rietveld refinements using parallel computing is presented. The method can be applied to most, if not all, Rietveld refinement programs. An example implementation for the Los Alamos *General Structure Analysis System* (*GSAS*) package is described. Using a cluster with seven processors, least-squares refinements are completed 2.5 to 5 times faster than on an equivalent single-processor computer.

1. Introduction

The Rietveld technique has revolutionized the analysis of powder diffraction data and has allowed increasingly complex crystallographic problems to be solved (Rietveld, 1969). Increasingly sophisticated instrumentation has also expanded the scope of powder diffraction. While in the past, crystallographic computations were performed as 'batch' runs, which might require researchers to wait for hours or days between runs, at present most scientists perform Rietveld analyses interactively, watching a run complete before performing their next action. Thus, productivity is in part determined by the speed of refinements. For many studies, refinements are virtually instantaneous, but for complex systems where many data points are used and where many reflections are present, refinements can be quite time-consuming. The improved resolution and increased $\sin\theta/\lambda$ range of the latest synchrotron and neutron instruments, along with the more common use of multiple diffraction measurements to provide information that is unobtainable from a single measurement, will increase computing demands for Rietveld studies.

Personal computing has drastically decreased the cost of computing; prices continue to drop with concomitant increases in speed. The processors in the most recently introduced personal computers are among the fastest available. This means that spending more money on a more expensive computer does not necessarily provide a faster processor. One must instead rewrite programs to utilize multiple processors simultaneously, a concept known as parallel processing. In this article, we show how the speed of Rietveld refinements can be increased dramatically through the use of parallel processing. We demonstrate this *via* a parallel-processing implementation for the *General Structure Analysis System* (*GSAS*) package (Larson & Von Dreele, 2000). By making minor changes to the least-squares minimization program, we were able to obtain 2.5- to 5-fold increases in speed. These factors will be maintained with future increases in processor speed. The methods used here should be applicable to most, if not all, least-squares minimization programs, and in particular to most Rietveld implementations.

2. Parallel processing

The conventional method of parallel processing requires that a master program generates many independent subtasks that can then be run simultaneously on slave processors (for example, see Pillai *et al.*, 1988). The slave processors receive the input needed to perform the subtask and return a result. The results from the subtasks are then incorporated into the overall computation by the controller program, decreasing the computation time. If many processes are competing for computing resources, this results in very efficient use of the processors, since idle processors can be used for other tasks. Alas, it can be very difficult to adapt older Fortran programs to this paradigm without very extensive revisions, since all the parameters needed as input and produced as output for the subtask must be identified and code must be modified so that these parameters are passed between processors.

We recently became aware of another approach to parallel processing that appears better suited to older programs. In this paradigm, a single version of a program is run simultaneously on multiple processors. The computation proceeds identically on each processor except for the most computationally intensive sections, which are then parallelized by sharing results between processors (Sims *et al.*, 2000). In this way, only small program sections must be modified to achieve a considerable amount of parallelization. This method is less efficient with regard to computer utilization, since many computations are duplicated. However, since these tasks run concurrently, there is no impact on the elapsed time, which is what matters from the user's perspective. The loss of processing efficiency as a result of duplication occurs in the least computationally demanding sections of the program, and further is usually of little concern, since the wasted computer time is of little value.

3. The least-squares method

The Rietveld technique uses least-squares optimization to find the best crystallographic and experimental parameters that fit a computed powder diffraction pattern to observations. *GSAS* allows simultaneous fitting of single-crystal, powder and

restraint (soft-constraint) information, so in the general case one must minimize

$$D = \sum_{S=1}^{N_S} w_h (F_{ho}^2 - SF_{hc}^2)^2 + \sum_{p=1}^{N_p} w_p (I_{po} - I_{pc})^2 + f_r \sum_{r=1}^{N_r} w_r (d_{ro} - d_{rc})^2, \quad (1)$$

where D describes the total discrepancies between the observations and corresponding values computed from a model. The first term in equation (1) fits to the N_S single-crystal observations, where w_h are the weights for these observations, S is a scaling factor, and F_{ho} and F_{hc} are observed and computed single-crystal structure factors, respectively. The second term is the powder contribution, where there are N_p observations and where I_{po} and I_{pc} are the observed and calculated powder intensities, respectively. The weight values, w_p, are usually determined from counting statistics, such that $w_p = 1/[\sigma(I_{po})]^2 = 1/I_{po}$. The third term defines the restraint contribution, where d_{ro} is the 'target' restraint value (for example, an expected bond distance), d_{rc} is the value computed from the model, and weights, w_r, are selected by the researcher (Larson & Von Dreele, 2000).

In linear least squares, D is minimized by solving the N equations generated by setting the derivative of equation (1) with respect to each parameter, v_i, to zero. Crystallographic refinements use non-linear relationships. For non-linear least-squares minimization, the derivatives of equation (1) are approximated using only the first-order terms in a Taylor series expansion around a set of starting values for each parameter. The minimization yields better values for each parameter, $v_i + \Delta v_i$ (Stout & Jensen, 1968). In matrix notation, vector $\mathbf{x}$ corresponds to the shifts to be determined from $\mathbf{Ax} = \mathbf{y}$, where $x_i = \Delta v_i$ (Prince, 1994). The gradient vector, $\mathbf{y}$, and the Hessian matrix, $\mathbf{A}$, are defined by

$$y_i = \sum_{p=1}^{N_p} w_p (I_{po} - I_{pc}) \frac{\partial I_{pc}}{\partial v_i} + f_r \sum_{r=1}^{N_r} w_r (d_{ro} - d_{rc}) \frac{\partial d_{rc}}{\partial v_i} + \ldots \quad (2)$$

and

$$A_{ij} = \sum_{p=1}^{N_p} w_p \frac{\partial I_{pc}}{\partial v_i} \frac{\partial I_{pc}}{\partial v_j} + f_r \sum_{r=1}^{N_r} w_r \frac{\partial d_{rc}}{\partial v_i} \frac{\partial d_{rc}}{\partial v_j} + \cdots, \quad (3)$$

where ellipses indicate the omitted single-crystal terms. Unlike the linear case, good starting values for v_i are needed and since an approximation is employed, multiple iterations are needed until the magnitude of the shifts becomes insignificant.

In a single-processor computation, the time needed for a least-squares cycle will be approximately $t_S + t_r + t_p + t_{\rm cyc}$, where subscripts S, r and p refer to the computation of contributions to y_i and A_{ij} from the single-crystal, restraints and powder data, respectively. The subscript cyc refers to the time needed for solving for x, applying these shifts, R-factor computation and other computations performed in each cycle.

In our experience, computation of the contributions to y_i and A_{ij} from powder data takes the vast majority of computing time in a Rietveld refinement, even when single-crystal data and restraints are used; thus, $t_p \gg t_S + t_r + t_{\rm cyc}$. Further, most of this time is spent on computing Ipc and $\partial I_{pc}/\partial v_i$ (which in turn require F_{hkl} for the contributing reflections) and evaluation of the appropriate profile functions and profile derivatives. Since only the sums of the contributions, y_i and A_{ij}, are needed, this computation is ripe for parallelization.

4. Implementation

We have found that the *GSAS* least-squares refinement program, *GENLES*, can be accelerated many-fold by simply parallelizing the computation of y_i and A_{ij} and duplicating virtually all other calculations on all processors. From profiling *GENLES*, we found that in a typical refinement over 99% of the processing time is spent in *PDCALC*, the subroutine used for evaluating contributions to y_i and A_{ij} from powder data, and in its children. So, while the basic sequence of the least-squares optimization calculations that are performed in *GENLES* remains intact, we have introduced some code to coordinate the multiple processors. Further, *PDCALC* has been adapted to run on subsets of the pattern. These were virtually the only code modifications needed.

4.1. Task distribution

We have found it easy to distribute the tasks over m processors by using the first processor for points $1, m+1, 2m+1, \ldots$ and the second processor for points $2, m+2, 2m+2, \ldots$, and finally, the mth processor for points $m, 2m, 3m, \ldots$ Since adjacent data points are distributed over all processors, every structure factor must be evaluated on every processor. This duplication usually has a very minor impact on efficiency, as the time used for structure-factor computations is typically very small. Thus, the powder pattern computation time, t_p, drops to approximately t_p/m with parallelization and the net speedup in the refinement is almost m-fold.

As will be shown later, the speedup is less effective when a very large number of reflections must be computed and where the reflection computation becomes slow, because there are a large number of atoms in the asymmetric unit. In this case, the structure-factor calculations consume a large fraction of t_p, so an alternate method for distributing the computation must be used to achieve effective parallelization. Each processor must be given a block of points that can be processed in serial, so that structure-factor computations are distributed rather than duplicated. This approach is difficult to implement as the blocks must be divided in such a way that each process requires similar computational time; otherwise speedup suffers. We have not attempted this approach yet.

No attempt has been made to parallelize the single-crystal or restraint y_i and A_{ij} computations, since the effort would produce minor gains at best. This means that if many processors are being used, then $t_S + t_r$ could become comparable with t_p/m. If this occurs, then the overall processing speed can still be improved by dedicating one processor to computing only the non-powder terms and using the remaining processors to perform the powder diffraction

computation. We call this 'asymmetric processing'. This will likely be of interest where many processors are available and the non-powder computations are extensive. In other cases, it is faster to use a 'symmetric processor distribution', where the powder diffraction computation is run on all processors. Very minor differences in coding are needed to select between symmetric and asymmetric processing; thus, this can be selected as a run-time option.

4.2. Interprocess communication

Parallel processing requires passing of data between processors. The processor we designate the controller must obtain the contributions to the y_i and A_{ij} values that are computed on each slave. Each slave must obtain the total y_i and A_{ij} values from the controller so that the slaves apply the same least-squares shifts as the controller. The controller could instead provide the x_i values to the slaves, as well as any intermediate values used later in the refinement, but this is potentially more difficult to implement and does not enhance speedup. In some Rietveld implementations, R factors are computed simultaneously with Hessian contributions; in this case, the controller must obtain the contributions to the R-factor terms from each slave in order to determine the overall R factors.

In addition to passing of data, timing information must also be passed between processors. The controller must wait for each slave to produce the y_i and A_{ij} contributions and the slaves must then wait for the total y_i and A_{ij} values.

4.3. *GSAS*-specific implementation aspects

GSAS reads all input and control parameters from a single experiment (*expnam*.EXP) file. Since this file is updated as each run progresses, to avoid potential interference, each processor must access a separate version of this file. This may be true for some of the intermediate files as well. Thus, it is most convenient to provide a separate directory for each processor. We have chosen to pass results and timing signals between slaves and the controller using files. This is simple to implement and is portable, but does require that the controller and slaves be able to access files written by each other and thus all processors must share a common file system.

We use a Unix shell script to initiate runs of *GENLES* on a series of processors. The script also creates copies of the *GSAS* input files needed by the slaves and provides an input file for each processor that specifies a processor number, the total number of processors, a flag for asymmetric processing, and the directories used by each processor. The directories are needed so that the controller can read files written by slaves and the slaves can read files written by the controller.

Fig. 1 outlines the flow of timing signals and results between processors in our parallel version of *GENLES*. In the initial stages, the same computations are duplicated on all processors. When the least-squares cycling begins, the computation of y_i and A_{ij} is distributed among the processors. The controller processor performs the restraint and single-crystal computations (and in the 'symmetric' case, a section of the powder diffraction computation). Simultaneously, the slaves begin their sections of the powder diffraction computation. The slaves do no restraint or single-crystal computations. When the contributions to y_i and A_{ij} are completed, the slaves write these results, as well as R-factor intermediates, to a disk file. Each slave then creates a signal file to indicate that this step is complete. The controller sums these contributions after waiting for each slave's signal file to be created. The controller then deletes each slave's signal file. When the y_i and A_{ij} sums are complete, these values are written to a disk file and the controller creates a signal file. Each slave waits for its signal file to be deleted and the controller's signal file to be created before attempting to read the y_i and A_{ij} sums. This use of the two different signal files prevents a slave from reading results from a previous cycle. Once the slaves have read the y_i and A_{ij} sums, all processors have an identical Hessian matrix. The refinement cycle is completed by determination of x_i values from the y_i and A_{ij} sums, application of shifts and other subsequent computations that are duplicated on all processors.

When all requested cycles have been completed, or *GENLES* terminates (for example due to convergence), one extra step is performed by the controller processor only. The controller reads and collates the computed profile points from

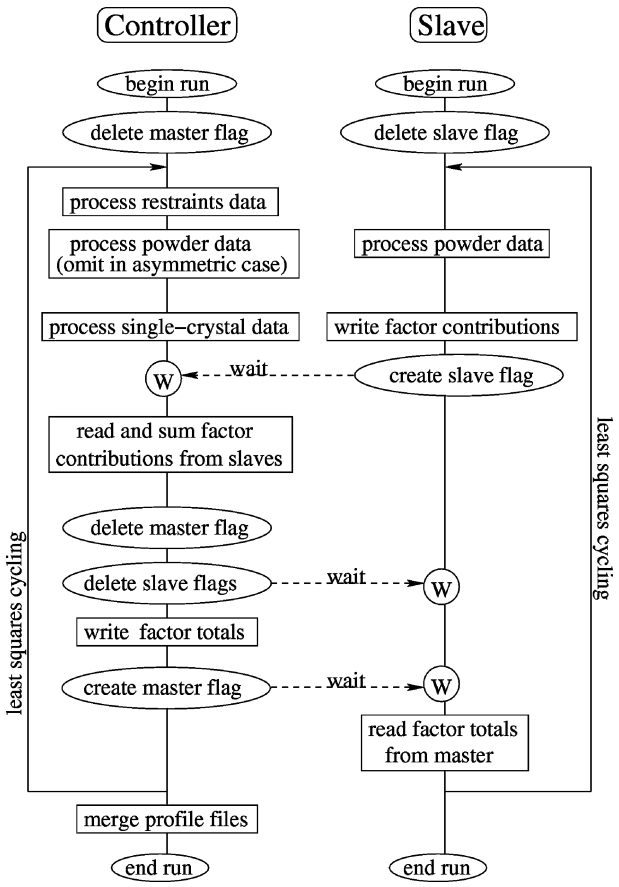

Figure 1
Steps used for parallel Rietveld refinements using a modified version of the *GSAS* program, *GENLES*. Note that there will typically be more than one slave processor.

the 'histogram' (*expnam*.P*xx*) files from each slave to produce a complete computed profile as an output file. No attempt is made to collate results in the reflection file, as this is used only occasionally. If needed, *GENLES* can be run quickly on a single processor with zero refinement cycles to produce this file.

5. Results

Our parallel-processing version of *GENLES* was compared with the original single-processor version using a Beowulf system of eight 400 MHz Pentium II processors, each with 256 Mbyte of RAM. In all cases, refinement results were equivalent, within typical round-off errors. Results from three different test cases are reported here. Test case *A* was performed with an orthorhombic unit cell (space group *Pba*2) and 37 atoms in the asymmetric unit. Nearly 2000 reflections were calculated for approximately 3000 data points in a single histogram and a total of 137 parameters were refined. Case *A* was selected as an example of a typical computationally demanding Rietveld refinement. Case *B* involved a pseudo-monoclinic unit cell with 324 atoms in the asymmetric unit (space group *C*1), approximately 8000 data points in two histograms and required the calculation of approximately 30 000 reflections. While refining only 12 parameters, this run also processed 1400 restraints. Despite the large number of restraints, asymmetric processing did not result in significant speedup. Case *B* is an example of an extremely large and perhaps pathological problem, where parallelization would be least effective. Case *C* tested the protein capabilities and restraints of *GSAS* using a monoclinic structure with 16 atoms in its asymmetric unit (space group $P2_1$), calculating 400 reflections and processing 14 000 data points and 64 restraints in the refinement of 77 parameters.

Fig. 2 shows the decrease in *GENLES* elapsed processing time with an increased number of processors. Each run time was normalized with respect to the time for a single-processor refinement. Lines show a fit of the form $t = b/m + (1 - b)$, where t is the normalized elapsed processing time and m is the number of processors used in a refinement, so that b is the parallelized fraction of processing time (Diederichs, 2000). Values of b for the three refinements are 0.93, 0.74 and 0.86 for cases *A*, *B* and *C*, respectively, showing that substantial parallelization is obtained in all cases. Note that in case *A*, *GENLES* runs about 5 times faster with 7 processors than on a single processor. For comparison, the dashed line in Fig. 2 shows ideal speedup behavior ($b = 1$), where m processors allow the computation to be performed m times faster than on a single processor.

The parallel version of *GENLES* does not compute all results obtained in the single-processor version. Since the powder diffraction data are not processed in serial fashion, the Durbin–Watson statistic is not computed. Since the reflection intensities are not tabulated, the Bragg *R* factor (R_{F^2}) is also not computed. To obtain these values, *GENLES* can be run quickly on a single processor with zero refinement cycles.

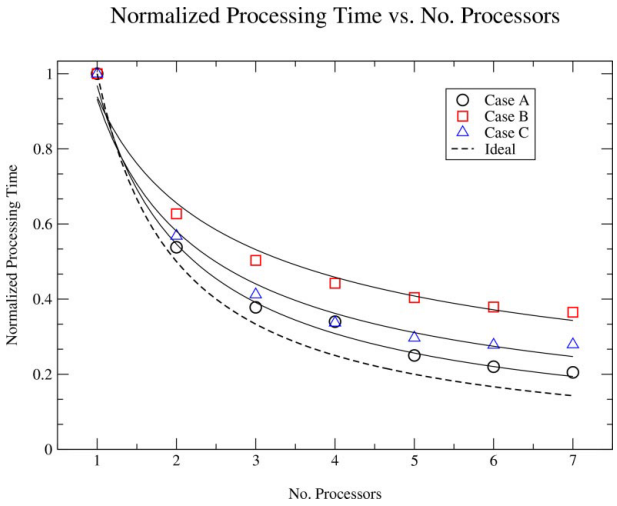

Figure 2
The decrease in computing time as a function of the number of processors for three different test cases. Points show the normalized processing time and the solid lines are fits to these points (see text). The dashed line shows the idealized case of a fully parallelizable task.

6. Future work

Currently, a run of the parallelized version of *GENLES* is initiated by running a simple Unix shell script. This script will require minor customization specific to the local computing architecture. While this has not been attempted, a similar script could allow the parallelized version of *GENLES* to be run on any set of compatible processors sharing a file system, not only a Beowulf system. Since the script is the only system-specific coding, we hope to implement this parallel version of *GENLES* for multiprocessor Windows NT/2000 systems. We also plan to investigate the possible advantages of MPI, a message-passing system designed for parallel processing on clusters, as a replacement for passing timing and data information *via* files (Gropp *et al.*, 1999). An MPI implementation may allow for effective load balancing even when each processor is given a block of adjacent points, so that efficient parallelization can be obtained even for very large problems.

Our parallel version of *GENLES*, along with the source code changed to implement this version, can be downloaded from a NIST Web site (http://www.ncnr.nist.gov/xtal/software/parallel) or from one of the CCP14 mirror sites (Cockcroft, 2001). In the future, we hope to see this program integrated into the *GSAS* distribution. It may also be possible to simplify the customization needed to prepare the submission script by inclusion of this feature into the *EXPGUI* program (Toby, 2001).

7. Conclusions

A method to parallelize Rietveld refinements has been developed. This method is relatively straightforward and applicable to most, if not all, Rietveld refinement programs, as well as other types of least-squares minimizations. While a few intermediate results are not available with this method, the parallelized version of *GENLES* is five times faster than the

original single-processor version of *GENLES*. Even in the worst case, a factor of nearly 2.5 improvement was seen. Relatively minor programming changes were needed. These changes are detailed in a companion Web page (http://www.ncnr.nist.gov/xtal/software/parallel).

The authors wish to thank Drs James Sims and Charles E. Bouldin for demonstrating the parallel computation approach implemented here and for valuable discussions. We are also indebted to Mr Bud Dickerson and Dr Przemek Klosowski for creating the Beowulf cluster from surplus computers and Dr Howard Hung for his help in profiling *GENLES*.

References

Cockcroft, J. K. (2001). Chairman, Collaborative Computational Project, Number 14 (CCP14), http://www.ccp14.ac.uk/.

Diederichs, K. (2000). *J. Appl. Cryst.* **33**, 1154–1161.

Gropp, W., Lusk, E. & Skjellum, A. (1999). *Using MPI*. Cambridge, MA: MIT Press.

Larson, A. C. & Von Dreele, R. B. (2000). *General Structure Analysis System (GSAS)*, Los Alamos National Laboratory, Report LAUR 86-748.

Pillai, K. N., Suter, B. W. & Carson, M. (1988). *J. Appl. Cryst.* **21**, 512–515.

Prince, E. (1994). *Mathematical Techniques in Crystallography and Materials Science*. New York: Springer Verlag.

Rietveld, H. M. (1969). *J. Appl. Cryst.* **2**, 65–71.

Sims, J. S., Hagedorn, J. G., Ketcham, P. M., Satterfield, S. G., Griffin, T. J., George, W. L., Fowler, H. A., am Ende, B. A., Hung, H. K., Bohn, R. B., Koontz, J. E., Martys, N. W., Bouldin, C. E., Warren, J. A., Feder, D. L., Clark, C. W., Filla, B. J. & Devaney, J. E. (2000). *J. Res. Natl Inst. Stand. Technol.* **105**, 875–894.

Stout, G. H. & Jensen, L. H. (1968). *X-ray Structure Determination; a Practical Guide*. New York: Macmillan.

Toby, B. H. (2001). *J. Appl. Cryst.* **34**, 210–213.