Journal of
Applied
Crystallography
ISSN 0021-8898

Received 1 March 2002
Accepted 28 March 2002

SnB version 2.2: an example of crystallographic multiprocessing

Jason Rappleye,[a] Martins Innus,[a,b] Charles M. Weeks[c]* and Russ Miller[a,b,c]

[a]Center for Computational Research, Norton Hall Rm 9, SUNY at Buffalo, Buffalo, NY 14260-1800, USA, [b]Department of Computer Science and Engineering, Bell Hall, SUNY at Buffalo, Buffalo, NY 14260-1800, USA, and [c]Hauptman-Woodward Medical Research Institute and Dept. of Structural Biology, SUNY at Buffalo, 73 High Street, Buffalo, NY 14203-1196, USA. Correspondence e-mail: weeks@hwi.buffalo.edu

The computer program *SnB* implements a direct-methods algorithm, known as *Shake-and-Bake*, which optimizes trial structures consisting of randomly positioned atoms. Although large *Shake-and-Bake* applications require significant amounts of computing time, the algorithm can be easily implemented in parallel in order to decrease the real time required to achieve a solution. By using a master–worker model, *SnB* version 2.2 is amenable to all of the prevalent modern parallel-computing platforms, including (i) shared-memory multiprocessor machines, such as the SGI Origin2000, (ii) distributed-memory multiprocessor machines, such as the IBM SP, and (iii) collections of workstations, including Beowulf clusters. A linear speedup in the processing of a fixed number of trial structures can be obtained on each of these platforms.

1. Introduction

Shake-and-Bake is a direct-methods procedure that makes possible the *ab initio* phasing of crystal structures containing as many as 2000 independent non-H atoms provided that accurate diffraction data have been measured to a resolution of 1.2 Å or better (Frazão *et al.*, 1999). It has also been used to determine the anomalously scattering substructures of selenomethionyl-substituted proteins containing as many as 160 selenium sites using 3–4 Å data (Frank von Delft, personal communication). *Shake-and-Bake* belongs to the class of phasing methods known as 'multisolution' procedures (Germain & Woolfson, 1968) in which multiple sets of trial phases are generated in the hope that one or more of the resultant combinations will lead to a solution. Solutions, if they occur, are identified on the basis of a suitable figure of merit.

The distinctive feature of *Shake-and-Bake* is the repeated and unconditional alternation of reciprocal-space phase refinement (*Shaking*) with a complementary real-space process (*Baking*) that seeks to improve phases by applying constraints (Weeks *et al.*, 1994). This automated recycling has proven to be considerably more powerful than previous direct methods. Phases are refined either by the tangent formula (Karle & Hauptman, 1956) or by constrained minimization of the so-called minimal function (DeTitta *et al.*, 1994) using the parameter-shift algorithm (Bhuiya & Stanley, 1963). The minimal function also serves as an effective figure of merit. Peak picking is used to impose the atomicity constraint in real space. A random-number generator is used to assign initial coordinates to the atoms comprising trial structures, and structure-factor calculations are performed to generate sets of starting phases. The complete *Shake-and-Bake* algorithm has been described in detail in recent reviews (Weeks *et al.*, 2001; Sheldrick *et al.*, 2001).

SnB is a user-friendly computer program that implements the *Shake-and-Bake* procedure. *SnB* version 1.0 (Miller *et al.*, 1994) provided a simple ASCII menu for users to input structure-specific information and, if necessary, make changes to the default values provided for the operating parameters (*e.g.* the numbers of phases, refinement cycles, and peaks to be selected). *SnB* version 2.0 (Weeks & Miller, 1999) introduced a graphical user interface (GUI), written in Java, not only for the main *Shake-and-Bake* phasing algorithm, but also for computing the necessary normalized structure-factor magnitudes using the *DREAR* program suite (Blessing & Smith, 1999, and references therein). The GUI allows for input of parameters, selection of data files, and the submission of jobs to perform the actual computations using the back-end Fortran executables. The repetitive shuttling of trial structures between real and reciprocal space gives *SnB* its power, but the need to perform two Fourier transformations in each cycle yields a computationally intensive procedure. Fortunately, each of the trial structures can be refined independently. Therefore, the algorithm readily lends itself to a simple coarse-grained parallel approach in which trial structures are distributed among numerous processors. The *SnB* interface provided in the most recent version, *SnB* version 2.2, makes it easy for users to take full advantage of the opportunities they have for coarse-grained parallel processing. The features of the *SnB* interface that expedite parallel operation are described in detail in the following sections.

2. The master–worker model

The master–worker paradigm is suitable for a wide variety of algorithms, including *SnB*, that can be readily adapted to a parallel-processing environment. A single master process is responsible for allocating tasks (*i.e.* the refinement of trial structures) to be completed by multiple worker processes. As the workers finish their tasks, the master gathers the results and assigns new tasks to available workers until the whole job has been completed. The fact that each task is independent of all the others allows for very little processing overhead.

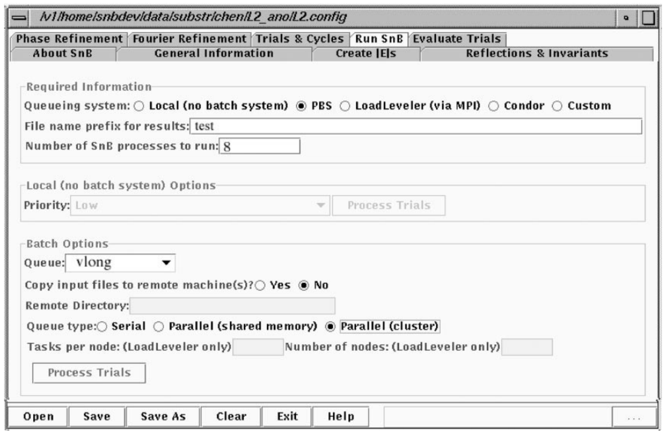
2.1. Multiprocessor machines

There are two different ways in which *SnB* has been equipped to increase throughput on machines having multiple processors. First, the version of the *SnB* executable available for IBM SP systems utilizes the message-passing interface MPI (Snir *et al.*, 1998) to perform explicit communication between processes during execution. In a program utilizing MPI, all processes are started simultaneously, and each is assigned a unique rank. In the case of *SnB*, the process with rank zero is designated as the master, and all remaining processes are workers. The master reads in the reflection data file and a control file produced by the GUI, and it transfers their contents to the workers. The master then enters a loop in which it assigns a fixed number of trials to each of the workers. If, for some reason, one node processes trials much more slowly than the other nodes, it will not degrade the overall throughput by a large amount. As each worker completes its batch of trials, the results are returned to the master, which responds by assigning that worker a new batch of trials. This procedure is repeated until the number of trials requested by the user has been completed. The master produces a single set of output files that combine the results of all the worker processes. Currently, one disadvantage of running an MPI job is the requirement that all processors be free at the outset, and this may result in a delay in initiating the jobs.

The second way in which *SnB* exploits the master–worker model is by simultaneously running multiple serial jobs, each of which runs exactly the same number of trial structures. Multiple control files are written, and the number of the trial structure at which processing is to begin is varied in these files in an appropriate manner. Each job runs independently of every other job, and there are no dependencies between jobs. Consequently, individual jobs can be started at any time, and no cycles are wasted waiting for a certain number of processors to become free. Each job produces its own set of output files, and no attempt is made to combine them into a single set of output files. However, as described below (§3), the GUI presents a composite view of the output from the multiple jobs so this is not normally a concern.

2.2. Networks

Condor (Litzkow *et al.*, 1988) is a system for utilizing available unused computing cycles on a network of workstations (typically referred to as a *Condor* ‘flock’). The primary user of a workstation, hereafter called the ‘owner’, often utilizes the machine only during the time he or she is at work. Valuable computing resources are wasted because workstations frequently sit idle during evenings, weekends and vacations. *Condor* allows these unused resources to be allocated so that other people can take advantage of them to run jobs.



Required Information

Queueing system: Local (no batch system) [✓ PBS] LoadLeveler (via MPI) Condor Custom

File name prefix for results: test

Number of SnB processes to run: 8

Local (no batch system) Options

Priority: Low

Batch Options

Queue: vlong

Copy input files to remote machine(s)? Yes [✓ No]

Remote Directory:

Queue type: Serial Parallel (shared memory) [✓ Parallel (cluster)]

Tasks per node: (LoadLeveler only) Number of nodes: (LoadLeveler only)

Figure 1
The ‘Run SnB’ screen of the *SnB* interface helps the user to take advantage easily of multiprocessing options. In this example, an eight-processor job (‘test’) is being submitted to a Linux cluster *via* a *PBS* queue (‘vlong’). See the text for further explanation.

Condor is a batch queuing system that queues user-submitted jobs and runs them when the appropriate resources are available. Traditional batch queuing systems, such as *PBS*, *LSF* and the *Loadleveler* system of an IBM SP, utilize a pool of dedicated machines to provide compute cycles. *Condor* differs from these traditional systems in that its pool of resources consists of a network of workstations that are sometimes reserved for use by their owners, whether it be for interactive purposes (*e.g.* surfing the web, reading e-mail, editing code) or for running their own codes locally. *Condor* is able to detect when machines in the pool it manages are idle, in which case it utilizes them to run jobs submitted *via* the *Condor* queue. If a workstation claimed by *Condor* becomes unavailable due to owner activity, *Condor* suspends the job so that it will not interfere with the machine’s normal operation. If the workstation remains unavailable to *Condor* for a certain specified period of time (configurable for each machine in the *Condor* flock), the job is ‘checkpointed’ and removed from the machine. The checkpointing process saves the state of the job so that it may be restarted on the same machine, or even on a different machine. The workstations in a *Condor* flock do not need to share a file system. *Condor* intercepts system calls made by an application and executes them on the machine that submitted the job. For example, when an ‘open’ system call is performed, *Condor* executes the call on the machine that submitted the job and passes the results to the caller. This facility makes the migration of checkpointed processes much easier, reducing the dependence on local resources.

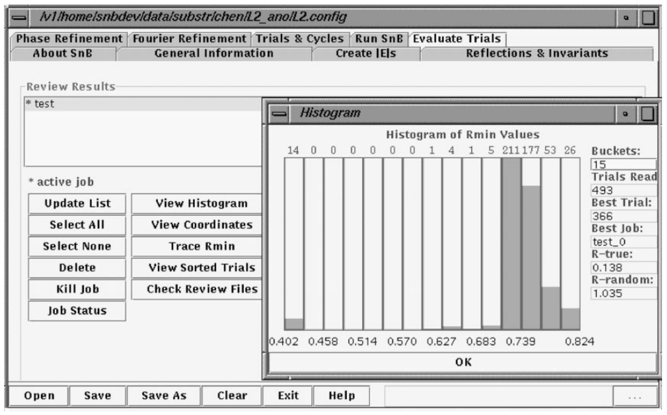
SnB has been enhanced to run on a *Condor* flock. For *n SnB* jobs submitted through the *Condor* interface in *SnB*, the GUI submits *n* serial jobs to *Condor*, dividing the trial structures equally among the jobs. Each job runs as the resources required become available. The end result, from the *SnB* user’s perspective, is the same as if the multiple serial jobs had been run on a single machine with multiple processors. For more information regarding *Condor*, the reader is referred to http://www.cs.wisc.edu/condor/.

3. Using the multiprocessing features

When the *SnB* graphical interface is invoked, the user is presented with several screens containing fields in which to enter data as well as buttons to perform pertinent tasks (Weeks & Miller, 1999). For example, the ‘General Information’ screen allows the user to supply structure-specific information (such as space group, cell constants, *etc.*). The ‘Create Es’ screen controls the data normalization process, and several other screens permit the user to examine and change the parameters that control the *Shake-and-Bake* phasing procedure.

The ‘Run SnB’ screen (Fig. 1) facilitates multiprocessing applications. It allows the user to specify how, using the available computing resources, the phasing job is to be carried out. First, the user must indicate if the job is to be run on the local computer or submitted to a batch processing system such as *PBS*, *LoadLeveler* or *Condor*. In all cases, the name of the job (to be used to identify output files) must be given, and the desired number of processes (subjobs) must be indicated. If multiprocessing capabilities are not available, the number of processes should be specified as ‘1’. One or more jobs can then be submitted to the local computer (or to a network of computers *via* *Condor*) simply by clicking the ‘Process Trials’ button. The ‘Custom’ queuing option permits the user to create control (`dat`) files for multiple jobs that can be submitted later *via* a queuing system that is not supported directly by the *SnB* interface.

J. Appl. Cryst. (2002). **35**, 374–376 Jason Rappleye *et al.* • Multiprocessing in *SnB* **375**



Buckets: 15
Trials Read 493
Best Trial: 366
Best Job: test_0
R-true: 0.138
R-random: 1.035

Figure 2
The 'Evaluate Trials' screen of the *SnB* interface facilitates analysis of a *Shake-and-Bake* phasing job. By clicking buttons such as 'View Histogram', the user will see the composite results for the test job.

In particular, if the *PBS* or *LoadLeveler* batch queuing systems are chosen, then the user must enter the information requested in the fields, including the queue name, queue type, the directory for staging files, and additional batch-queuing-specific parameters. For example, suppose a user chooses to run eight *SnB* processes. Then, if the user chooses a serial queue type, *SnB* will submit eight single-processor jobs to the queue, while if the user chooses one of the available parallel queues, *SnB* will submit a single eight-processor job.

The time spent waiting in a queue is dependent on the site's scheduling policy. However, note that it is typically the case that a serial job will start as soon as a single processor becomes free, while a parallel job that requires n processors will typically have to wait until n processors are free. Therefore, having both the serial and parallel options available is useful in terms of optimizing utilization based on available computing platforms, scheduling systems and scheduling policies.

The 'Evaluate Trials' screen (Fig. 2) provides a consistent and convenient interface for analyzing the output of *SnB* jobs, regardless of the manner in which they were run. An *SnB* job name appears only once in the 'Review Results' box regardless of whether the job involved one or many processes. The output of each *SnB* process consists of several files containing information regarding the set of trial structures examined by that process. The names of these output files are prefixed with the user-selected identifier followed by an underscore and a number indicating the rank of the respective process. For example, the 'peak' files containing the fractional coordinates for the best trial structures for an eight-processor job where the prefix is 'test' would be named `test_0.SnB_peak`, `test_1.SnB_peak`, ..., and `test_7.SnB_peak`. For a single processor job, the GUI simply presents the data from each of the output files as requested. For multiprocessor jobs, the GUI examines each set of output files and presents the composite information to the user. The on-line documentation provided within *SnB* provides detailed help for analyzing the results of *SnB* jobs.

4. Conclusion

The direct-methods program *SnB* can be run conveniently in a variety of parallel-processing environments. Since each trial structure can be processed independently, throughput increases linearly with the number of processors being used. *SnB* version 2.2 is available for computers running IRIX, OSF/1, AIX, Solaris, Alpha Linux, and Intel Linux. Further, it provides support for many of the popular queuing systems, including *LoadLeveler* and *PBS*. Executable copies of *SnB* can be obtained by clicking the 'Download' button on the *SnB* home page at http://www.hwi.buffalo.edu/SnB/. Users requiring assistance utilizing the multiprocessing features described here can submit inquiries to snb-help@hwi.buffalo.edu.

The development of *SnB* is supported by grants GM-46733 (NIH) and ACI-9721373 (NSF), a DOE Grand Challenge grant, and the Center for Computational Research at SUNY-Buffalo.

References

Bhuiya, A. K. & Stanley, E. (1963). *Acta Cryst.* **16**, 981–984.
Blessing, R. H. & Smith, G. D. (1999). *J. Appl. Cryst.* **32**, 664–670.
DeTitta, G. T., Weeks, C. M., Thuman, P., Miller, R. & Hauptman, H. A. (1994). *Acta Cryst.* A**50**, 203–210.
Frazão, C., Sieker, L., Sheldrick, G. M., Lamzin, V., LeGall, J. & Carrondo, M. A. (1999). *J. Biol. Inorg. Chem.* **4**, 162–165.
Germain, G. & Woolfson, M. M. (1968). *Acta Cryst.* B**24**, 91–96.
Karle, J. & Hauptman, H. (1956). *Acta Cryst.* **9**, 635–651.
Litzkow, M., Livny, M. & Mutka, M. (1988). *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111. Los Alamitos, CA: IEEE Computer Society.
Miller, R., Gallo, S. M., Khalak, H. G. & Weeks, C. M. (1994). *J. Appl. Cryst.* **27**, 613–621.
Sheldrick, G. M., Hauptman, H. A., Weeks, C. M., Miller, R. & Usón, I. (2001). *International Tables for Crystallography*, Vol. F, edited by M. G. Rossmann & E. Arnold, pp. 333–345. Dordrecht: Kluwer Academic Publishers.
Snir, M., Otto, S., Huss-Lederman, S., Walker, D. & Dongarra, T. (1998). *MPI: The Complete Reference*, Vol I, *The MPI Core*. Cambridge, MA: MIT Press.
Weeks, C. M., DeTitta, G. T., Hauptman, H. A., Thuman, P. & Miller, R. (1994). *Acta Cryst.* A**50**, 210–220.
Weeks, C. M. & Miller, R. (1999). *J. Appl. Cryst.* **32**, 120–124.
Weeks, C. M., Sheldrick, G. M., Miller, R., Usón, I. & Hauptman, H. A. (2001). *Advances in Structure Analysis*, edited by R. Kuzel & J. Hasek, pp. 37–64. Praha: Czech and Slovak Crystallographic Association.