Journal of
Applied
Crystallography

ISSN 0021-8898

Received 2 October 2008
Accepted 24 February 2009

GDASH: a grid-enabled program for structure solution from powder diffraction data

Thomas A. N. Griffin,[a] Kenneth Shankland,[a]* Jacco van de Streek[b]‡ and Jason Cole[b]

[a]STFC Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Didcot OX11 0QX, UK, and [b]Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge CB2 1EZ, UK. Correspondence e-mail: kenneth.shankland@stfc.ac.uk

The simulated annealing approach to structure solution from powder diffraction data, as implemented in the *DASH* program, is easily amenable to parallelization at the individual run level. Very large scale increases in speed of execution can therefore be achieved by distributing individual *DASH* runs over a network of computers. The *GDASH* program achieves this by packaging *DASH* in a form that enables it to run under the Univa UD *Grid MP* system, which harnesses networks of existing computing resources to perform calculations.

1. Introduction

DASH (David *et al.*, 2006) is a computer program for structure solution from powder diffraction data (SDPD) that, since its first release in 1999, has placed computational efficiency as well as effectiveness at the heart of its design. It employs the now widely adopted global optimization approach to SDPD and, in particular, uses simulated annealing (SA) as its primary optimization method. By its nature, no individual SA run is guaranteed to find a global minimum equating to the solved crystal structure in a finite time frame and so multiple SA runs are required when tackling an SDPD problem in order to maximize the chances of locating this global minimum. For relatively small problems of the order of 15–20 degrees of freedom, only a small number of runs (~50) is typically required in order to locate the global minimum several times. For more complex problems, of the order of 30 or more degrees of freedom, the success rate (defined as the number of SA runs reaching the global minimum divided by the total number of SA runs and expressed as a percentage) can fall to only a few percent, and successful structure solution may necessitate several hundred SA runs to be performed. Faced with such a processing load, one can consider invoking fine-grained parallelization of the time-consuming calculations in order to speed up the evaluation of a single SA run, or coarse-grained parallelization, taking advantage of the fact that each SA run is independent of every other and that there is no requirement for these runs to be performed sequentially, *i.e.* this is an 'embarrassingly parallel' problem. The former approach, utilized mostly on symmetric multiprocessing/shared memory based computers, has been exploited in crystallography for a variety of demanding calculations (Diederichs, 2000). The latter approach has also been used to good effect in other crystallographic multisolution-type procedures, such as that employed by *Shake and Bake* (Miller *et al.*, 2007). Both approaches have their advantages, but as a general rule, if a problem falls into the category of embarrassingly parallel then the latter approach is favoured, as it generally does not require modification of the core code and can be scaled to a vast number of processors that can be resident in computers that are geographically quite distinct.

With *DASH*, we have pursued the latter approach, using the Univa UD *Grid MP* system (Univa UD, 2008) which can harness the spare CPU cycles of existing networked computing resources (such as a departmental PC network) in a well defined manner. It thus has the potential to achieve very large scale parallelization without the need to invest in new hardware. Furthermore, *Grid MP* is widely used and has been well tested on other embarrassingly parallel tasks (such as protein–ligand docking), and it has all the features required to enable *DASH* to be easily distributed across a network. *DASH* (Version 3.1 onwards) has thus been modified to allow its SA engine to be driven by command file, in order to allow it to be run under *Grid MP*.

2. *GDASH* overview

GDASH is a command-line driven program that takes, as input, files generated using *DASH* and submits them to a *Grid MP* server (a computer that runs the core *Grid MP* software) for subsequent execution by *DASH* on client PCs (computers that run the *Grid MP MPAgent* software) that are in contact with the *Grid MP* server (Fig. 1). It also has the ability to retrieve the results of these *DASH* calculations from the *Grid MP* server at any time and amalgamate them into a single results file that is readable by *DASH* (Fig. 2).

3. Program description

3.1. Creation of input files for *GDASH*

In setting up a normal SDPD attempt using *DASH*, the user follows a well defined set of steps, generating a series of required files (*e.g.* .sdi, .hcv) along the way, culminating in the execution of a number of sequential SA runs. For a full description of these steps and of the function of the various files generated, the reader is referred to David *et al.* (2006). As of Version 3.1 of *DASH*, the user can specify the generation of an additional batch file for subsequent execution on a grid-type system (or a multiple-core CPU), as opposed to pressing the 'Solve' button to start a standard sequence of runs (Fig. 3, top). The batch (.grd) file generated is simply a text listing of

‡ Present address: Avant-garde Materials Simulation, D-79100 Freiburg, Germany.

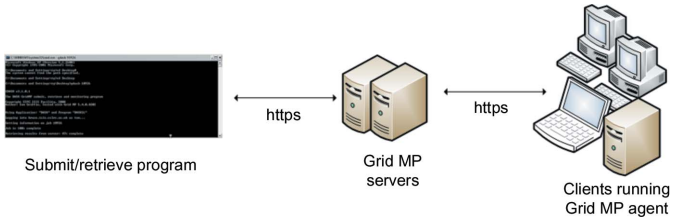

Figure 1
An overview of *GDASH* operation

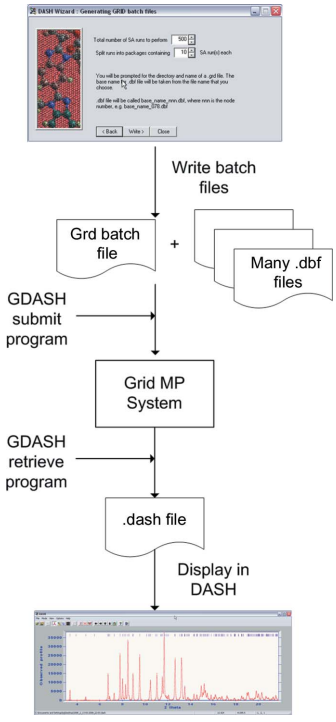

Figure 2
The relationship of *DASH* and *GDASH*. *DASH* produces the batch files that are submitted to the grid system using *GDASH*. *GDASH* is also used to retrieve results from the grid system in a format that can be displayed in *DASH*.

the names of the *DASH* control (.dbf) files that are generated. The number of .dbf files generated depends upon the total number of SA runs requested and the number of SA runs per package (where each package represents a discrete number of SA runs upon which a single instance of *DASH* will operate), with the number of files equal to the total number of SA runs divided by the number of runs per package (Fig. 3, bottom). Typically, each package will contain only a single SA run, and thus a request for 500 SA runs will generate 500 .dbf files. *DASH* gives the user the option of saving the .grd and .dbf files into a new directory, in order to provide a tidy directory structure and allow multiple sets of grid files to be created if desired. Once the .grd and .dbf files have been generated, the user may exit *DASH*.

3.1.1. The .dbf file. The .dbf (*DASH* batch file) file is an ASCII file containing the necessary information that can be read in by *DASH* in order to set up an SA run. Thus, it contains the locations of the .sdi (*DASH* project file, which lists files needed for the correct setup of the SA) and .zmatrix (molecular structure description) files, and a location for the output .dash file into which results are written. The file is fully commented with easy-to-interpret control parameter names, *e.g.* MAXMOVES is the maximum number of SA moves allowed in a single SA run. The .dbf file can be edited by hand if needed, but in practice this should rarely be necessary.

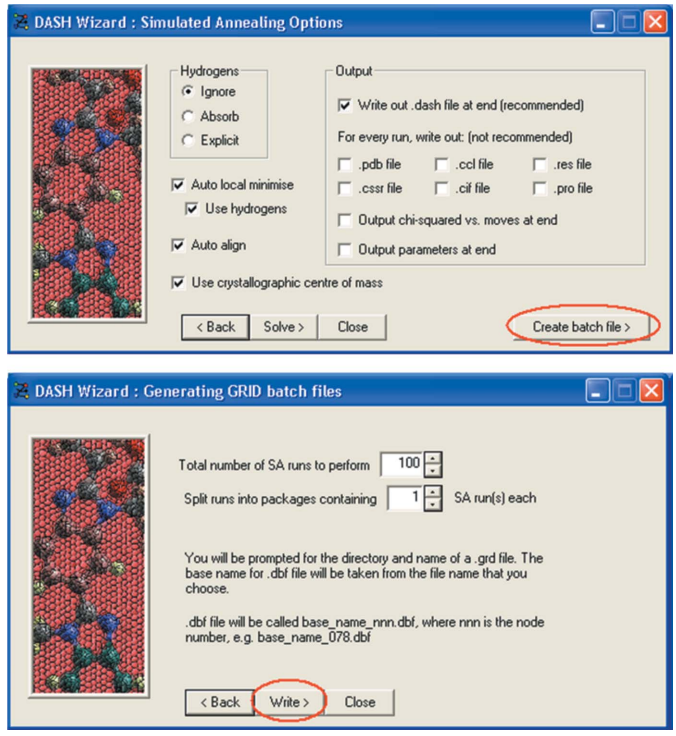

Figure 3
The *DASH* interface allows for the creation of batch files ready for execution on the grid (top), and also allows the user to specify the total number of SA runs to be performed and how they are to be packaged for execution.

3.1.2. *DASH* command line execution. If the *DASH* executable is invoked from the command line with an argument consisting of a .dbf file (*e.g.* `c:\dash.exe mytestfile.dbf`), *DASH* does not display the usual graphical user interface (GUI) but instead begins asynchronous execution of the SA run(s) specified in the .dbf file. Upon completion, the program output is stored in the .dash file specified in the .dbf file. Note that this 'suppressed-GUI' mode of operation is essential to enable *DASH* to operate in a distributed environment where it will be executing in the background at low priority on PCs that are already in use by other people. Note too that running *DASH* in this mode also brings a small performance gain (~10% reduction in execution time) relative to the standard mode of operation, as the program no longer has the overhead of updating the GUI after every SA move.

3.1.3. Submitting jobs using *GDASH*. *GDASH* is invoked from the command line, so the first task is to open a command prompt window in the directory that contains the .grd and .dbf files created using *DASH*. Typing `gdash` at the command prompt returns a brief summary of how to run the program. To submit a job created previously, the user types `gdash filename`, where `filename` is the name of the .grd file. The program displays the progress of the job submission in terms of the percentage of data packages transferred to the grid servers. Once all the necessary files have been uploaded to the server, the job is started and a job summary (including a job_id number) is returned (Fig. 4). Note that at this point, execution of the job is now entirely under the control of the grid servers and the user can close the command window on the machine used for job submission.

3.1.4. Monitoring and retrieving jobs using *GDASH*. In order to monitor the progress of a job, the user invokes *GDASH* from the command line, with the job_id number as an argument. If the job is not yet complete, it returns the '% completeness' and offers the

Table 1
The DASH.mpconfig file.

Parameter	Value
Grid_Username	grid_power_user
Grid_Password	power_user_password
Results_per_WU	1
Max_Concurrent	1
Max_Error	3
Priority	10
WU_Clock_Timeout	36000
WU_CPU_Timeout	36000
AppName	*DASH*
ProgName	*DASH* 3.1
MGSI_FILESVR_URL	https://gridserver.mydomain.com:28443/mgsi/filesvr.fcgi
MGSI_SOAP_URL	https://gridserver.mydomain.com:18443/mgsi/rpc_soap.fcgi

option to download those results already completed (Fig. 5). If it is complete, results files are downloaded from the server to the user's PC and then merged into a single .dash file automatically. Note that the .dash file is assigned a name that incorporates the date and time of submission of the job. The .dash file can then be opened using *DASH* in order to examine the results of the job.

3.1.5. The DASH.mpconfig application configuration file. *GDASH* has the ability to control many of the grid parameters that are relevant to *DASH* jobs and these can all be set in the DASH.mpconfig file. A sample configuration file is shown in Table 1. The only parameters that may need to be adjusted with any frequency are those pertaining to the timeouts that apply to *DASH* program execution. By way of example, the default timeout of 36 000 s sets limits of 10 h on both the CPU time and the elapsed wall clock time, after which a job is deemed to have failed.

4. *DASH* program performance when invoked using *GDASH*

The performance of *DASH* running on the *Grid MP* installation at the ISIS Facility of the STFC Rutherford Appleton Laboratory has been evaluated using the moderately challenging optimization problem of solving the crystal structure of famotidine form B (Shankland *et al.*, 2002; $P2_1/c$, V = 1421 Å^3, Z' = 1, 13 degrees of freedom, 1.64 Å resolution) from synchrotron X-ray powder diffraction data. In order to generate easily measurable execution times, the total number of SA moves per run was set to 1×10^7, about a factor of ten higher than is necessary actually to solve famotidine. Performance tests were carried out on a single PC, a test grid of five PCs and the full production *Grid MP* system, and test results are summarized in Table 2. Fig. 6 shows the progress of the large job submitted to the production grid. During the initial quiet phase, data are transferred from the PC running *GDASH* to the grid servers, where 999 workunits are assembled. About 4 min after job invocation from *GDASH*, ~300 workunits are sent out for execution from the grid servers to grid client machines. Approximately 5 min later the first results are returned, and from then on a steady stream of results is received, with more workunits being sent out to occupy now idle CPUs. The entire job, consisting of 999 SA runs, is complete in just under 40 min.

5. Software and hardware environment

GDASH itself runs under MS Windows XP (SP2) and MS Windows Vista. The *GDASH* installer requires the MS .Net 2.0 Framework (or higher) to be installed; this is present by default in XP SP2 and Vista. *GDASH* only needs to be installed on computers from which grid *DASH* jobs are going to be submitted. The *GDASH* installer will

Table 2
DASH performance on a test SA job under different conditions.

	Single PC	Test grid	Production grid
DASH 3.1 mode	GUI	Batch	Batch
Number of SA runs	64	64	999
Number of PCs used	1 (using 1 core only)	5	163
Core2Quad CPU speed	2.4 GHz	Four 2.4 GHz and one 2.8 GHz	Mixed†
Total elapsed time	9 h	24 min	40 min
Net minutes per single SA run	8.43	0.38	0.04
Relative speed rating	1	22	211

† The production grid consisted of a mixture of Athlon, Duron, P4, Xeon, Core2Duo and Core2Quad CPUs with speeds ranging from 1.2 to 3.6 GHz.

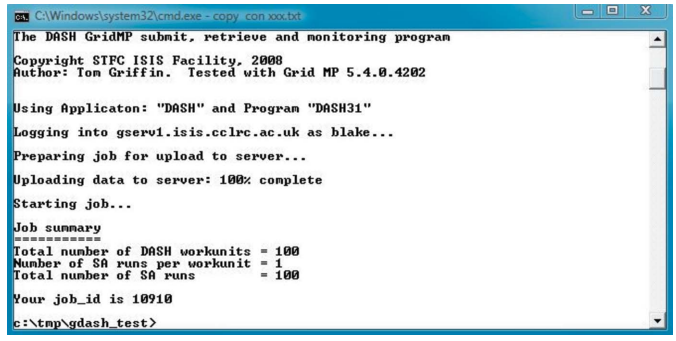

Figure 4
GDASH provides feedback on the progress of a job submission and returns the job_id that is necessary for subsequent job monitoring.

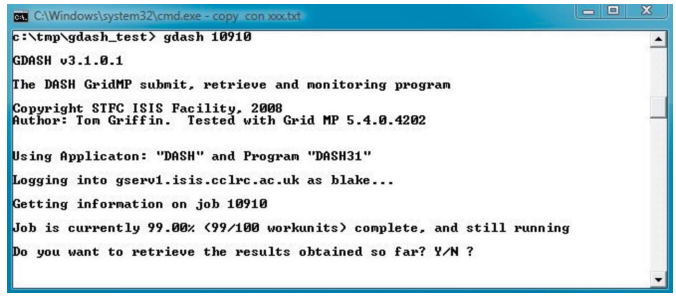

Figure 5
GDASH allows the results of partially completed jobs to be retrieved.

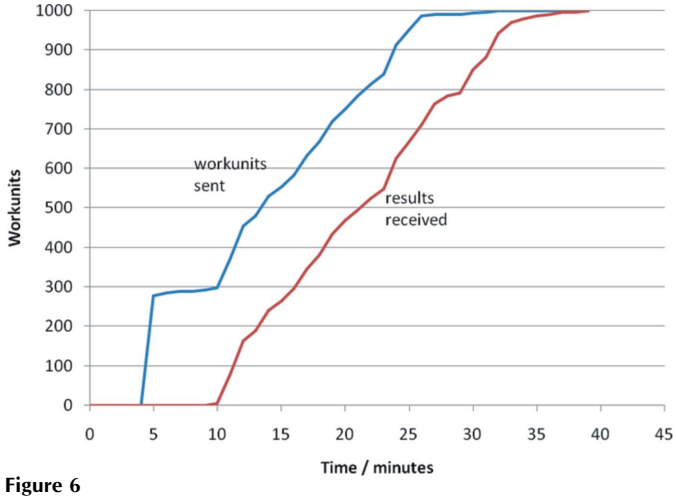

Figure 6
The progress of the 999 workunit test job on the production grid, plotted as a function of time.

perform this installation but will also perform the installation of *DASH* Version 3.1 onto the *Grid MP* servers. Furthermore, the installer sets up all the necessary environment variables that permit *GDASH* to be accessed from the command prompt. Note that *DASH* does not need to be installed onto individual client PCs; the *Grid MP* system sends out a copy of *DASH* (together with a valid licence) to client PCs as part of each workunit.

5.1. Prerequisites

In order to deploy *GDASH* correctly, the user must have a running *Grid MP* system with administrative access rights and access to the *Grid MP SDK*. *DASH* Version 3.1 or higher must be installed on the PC from which the *GDASH* installer is run and the user must have a site licence (or equivalent demonstration licence) for *DASH* in order to permit execution on client machines.

6. Documentation and availability

A *GDASH* executable can be downloaded free of charge from http://www.gdash.info. The download package includes the *GDASH* installer and full user documentation in PDF format. Whilst *GDASH* itself does not require a licence, it does require a licenced copy of *DASH* in order to operate.

We are grateful to Sravish Sridar of Univa UD for his assistance in setting up the original *Grid MP* system at the ISIS Facility. We are also extremely grateful to STFC Facilities Business Unit IT Services for their help and cooperation in deploying the *Grid MP* agent on client PCs throughout the FBU. Thanks are also due to Elna Pidcock and Wei Dong of the CCDC, and to Alastair Florence and Norman Shankland of the University of Strathclyde, for their help in testing and validating *GDASH*.

References

David, W. I. F., Shankland, K., van de Streek, J., Pidcock, E., Motherwell, W. D. S. & Cole, J. C. (2006). *J. Appl. Cryst.* **39**, 910–915.
Diederichs, K. (2000). *J. Appl. Cryst.* **33**, 1154–1161.
Miller, R., Shah, N., Green, M. L., Furey, W. & Weeks, C. M. (2007). *J. Appl. Cryst.* **40**, 938–944.
Shankland, K., McBride, L., David, W. I. F., Shankland, N. & Steele, G. (2002). *J. Appl. Cryst.* **35**, 443–454.
Univa UD (2008). http://www.univaud.com/hpc/products/grid-mp/.