Acta Crystallographica Section D
Biological Crystallography
ISSN 0907-4449

Ongoing developments in *CCP*4 for high-throughput structure determination

M. D. Winn,* A. W. Ashton, P. J. Briggs, C. C. Ballard and P. Patel

Daresbury Laboratory, Daresbury, Warrington WA4 4AD, England

Correspondence e-mail: m.d.winn@dl.ac.uk

Received 9 May 2002
Accepted 3 September 2002

Collaborative Computational Project Number 4 (CCP4) was established in 1979 to promote collaboration between UK groups writing software for protein crystallography. From these beginnings, CCP4 now distributes a large software suite and is active in developing new software. In this article, an overview is given of recent and ongoing developments in the *CCP*4 software suite, in particular as they pertain to high-throughput studies. Developments in individual programs are discussed first, although these are covered in more detail elsewhere. The bulk of the article focusses on the infrastructure of the software suite which allows the user to move effortlessly between different programs or to create automated schemas. Major changes to the software library at the heart of the *CCP*4 suite, developments in the *CCP*4 graphical user interface, and data management within *CCP*4 are discussed. The latter is crucial to high-throughput studies, where a large number of data are imported, created and finally archived.

1. Introduction

The Collaborative Computational Project Number 4 (CCP4) supports the development and use of software in macromolecular crystallography (Collaborative Computational Project, Number 4, 1994; Dodson *et al.*, 1997). A suite of programs is released periodically to user groups around the world, with new releases typically including several recently developed programs as well as updates to old favourites. Coherent use of the programs is facilitated by standard file formats, common data files and a graphical user interface.

As the producer of a self-contained and portable software suite, CCP4 does not address directly the problem of setting up structure-solution protocols within a particular structural genomics programme. However, CCP4 programs do feature in many such protocols and therefore CCP4 has an important role in providing the correct tools for high-throughput studies. Recent examples of procedures based at least partly around CCP4 programs include the validation script developed by Badger & Hendle (2002) and the *Elves* automated structure-solution package (Holton, 2002).

In this article, we give an overview of CCP4 developments as they pertain to high-throughput initiatives. We consider four broad areas, although the division is somewhat arbitrary. In §2, we highlight a few programs which are under active development. The underlying algorithms of these programs are becoming more sophisticated, with, for example, increasing use of advanced statistical methods. In addition, however, these programs are becoming more self-contained and therefore more suited to automation, although the ability to intervene when problems are encountered is retained.

The individual programs of the *CCP*4 software suite provide the essential functionality for structure solution. Nevertheless, structure solution requires the use of several programs and communication between the programs is essential and is where much of the effort towards automation is expended. This communication depends on the infrastructure of the *CCP*4 suite, in particular the software library which provides a common set of functionality shared by most applications. In §3, we describe the ongoing development of a new software library for CCP4 which will provide the basis for future high-throughput solutions.

The *CCP*4 graphical user interface 'ccp4i' is now well established. While, in principle, user intervention is minimized in high-throughput structure determination, it is likely to remain necessary for some time and easing the task for the user is a valuable aim. Moreover, ccp4i as an intermediate layer between programs and users provides a mechanism for automation. The role of ccp4i is discussed in §4.

Last, but certainly not least, §5 discusses data management within the *CCP*4 suite. High throughput requires the seamless transfer of data from the experiment, through structure solution, to final deposition in the Protein Data Bank (PDB). CCP4's role in the Data Harvesting initiative is described, along with other areas of data management.

It is in the nature of CCP4 that developments involve the participation of many people, from the contributing software developers to the practicing crystallographers who provide suggestions and a testbed for new software. Many of the developments discussed here in the context of CCP4 are important in their own right and the reader is referred where appropriate to the original work.

2. Program developments

The *CCP*4 software suite consists of a large number of individual programs (around 160 binaries at the time of writing). Many of these programs continue to be actively developed and we do not attempt to cover them all. In this section, we highlight briefly a few programs which are or will be important components of high-throughput studies.

The program *MOSFLM* performs data processing for image plate and CCD data, including autoindexing, determination of data-collection strategy and integration of images (Leslie, 1992). The *MOSFLM* program is currently being rewritten with a separate program and graphical interface, linked by a controlling server. The new modular program is in turn being included in an integrated data-collection and data-processing scheme which will handle communication between the different software components (the so-called 'DNA' project). The aim is to reduce the need for human intervention in straightforward cases. This is described in more detail in Leslie *et al.* (2002).

As an independent development, Randy Read and coworkers are developing the *PHASER* program which uses multivariate statistical methods to develop likelihood functions for SIR/MIR, SAD/MAD and molecular replacement within a unified framework. The method includes a treatment for pairwise correlations, for example between the $F(+)$ and $F(-)$ datasets in a SAD experiment. The MR likelihood target has been implemented in the program *BEAST* described in Read (2001). The aim is to have a fully integrated program for experimental phasing and molecular replacement, including heavy-atom location where appropriate. *PHASER* will be included in the *CCP*4 suite and CCP4 is funding the development of a ccp4i interface to the program.

Many *CCP*4 programs are in principle amenable to parallelization, for example where calculations are performed independently on each reflection in a set, and would benefit from the gain in speed on parallel or distributed architectures. Diederichs (2000) has developed parallel versions of *ESSENS* (Kleywegt & Jones, 1997) and *SHELXL* (Schneider & Sheldrick, 1997) using the OpenMP specification for shared memory architectures and has subsequently implemented OpenMP directives for *BEAST*, which are included in the version in *CCP*4 4.2.

ACORN is a flexible and efficient *ab initio* procedure to solve a protein structure when atomic resolution data are available. Initial phases are generated from a fragment, which can be random atoms, heavy atoms (sulfur or heavier) including anomalous scatterers or a feature such as a standard α-helix or a motif from a similar structure. *ACORN* integrates the separate tasks of locating the correct orientation and position of the initial fragment and refinement of the initial phases. In the case of the metalloproteinase deuterolysin, a solution was obtained in a few minutes from 5000 trials of a single atom in a random position (McAuley *et al.*, 2001). The program is included in the *CCP*4 suite from version 4.2. Further details are given in Yao (2002; see also Foadi *et al.*, 2000).

Other *CCP*4 programs which are undergoing active development include the automated molecular-replacement program *MOLREP* (Vagin & Teplyakov, 1997) and the maximum refinement program *REFMAC* (Murshudov *et al.*, 1997). Details of recent developments in *REFMAC* were given in the Study Weekend talk by Garib Murshudov.

3. Library developments

The *CCP*4 software suite is based around a library of routines which cover common tasks such as file opening, parsing keyworded input, reading and writing of standard data formats, applying symmetry operations *etc.* Otherwise independent programs in the *CCP*4 suite, such as those described above, call these shared routines which, as well as saving programmer effort, ensure that the programs in the suite have a similar look and feel.

The current *CCP*4 library is now a mature and extensively tested product and within the context of traditional *CCP*4 software it performs well. However, it was not designed to cater for the demands now being placed on software by automation initiatives. From a conceptual point of view, the library does not reflect the data structures which are being evolved to describe the crystallographic experiment. From a programming point of view, most library functionality is only

available to Fortran programs, whereas the trend nowadays is towards extensive use of scripting languages to glue computational modules together.

Over the past year or so, therefore, there has been a major effort to rewrite much of the *CCP*4 library. The new library is being written in a mixture of C and C++ with the following aims.

(i) To implement a better representation of the underlying data model. For example, Eugene Krissinel's MMDB library acts on a data structure which represents the hierarchical structure of a protein model. The CMTZ library encapsulates the crystal/dataset hierarchy that is increasingly being used by programs.

(ii) To provide support for scripting. It is possible to generate Application Programming Interfaces (APIs) to the Python, Tcl and Perl scripting languages automatically from the core C code. Thus, much of the standard *CCP*4 functionality can be made available to scripts such as those used in ccp4i or the *CCP*4 Molecular Graphics project.

(iii) To maintain support for existing programs. In particular, the existing Fortran APIs will be maintained, although they will now often be only wrappers to functions in the new library. It is intended that many existing programs will be migrated to using the new library directly.

The last point highlights the incremental approach being adopted, in which existing programs can be rewritten to use the new library at the discretion of the individual authors and on a timescale to suit them. This clearly adds a constraint to the new library, but is very important for CCP4 in its role as a collaborative project.

Fig. 1 shows a schematic of the new library. At the centre, the low-level library and the modules MMDB, CMTZ, CMAP and CSYM provide much of the core functionality for crystallographic computing. Built on top of this there are APIs for C, Fortran, Python, Tcl and Perl. The Fortran API is a relatively thick layer, as backwards compatibility with the existing API requires extra manipulations to be performed. For example, Fortran channel numbers are converted to pointers to the appropriate data structures. In contrast, the Python, Tcl and Perl interfaces are generated automatically using *SWIG* (Simplified Wrapper and Interface Generator; http://www.swig.org) and reflect closely the underlying C library.

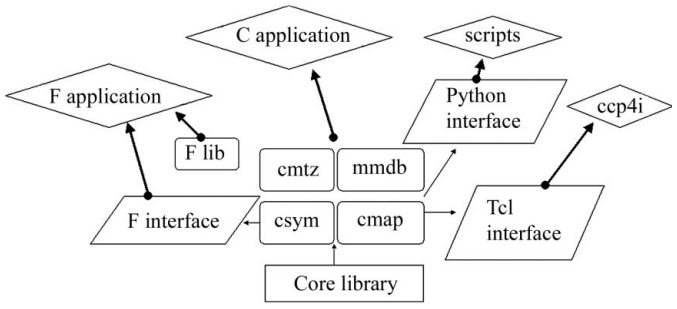

Figure 1
A schematic of the new *CCP*4 software library. Rectangles represent C language modules, which exist as groups of source files. Parallelograms represent interfaces to different programming languages and diamonds represent applications. Further details are given in the text.

In the current incarnation of the new library, there are still some important components which have not been rewritten (indicated by 'F lib' in Fig. 1). The most important example here is the FFT library, but there are also Fortran routines for handling Data Harvesting, for writing HTML log files *etc.* These routines continue to be available to existing Fortran programs, but are not yet available to newer applications.

We now consider some of the components of the new library in more detail. The MMDB library is designed to assist CCP4 developers in working with coordinate data. Coordinates can be held in PDB or mmCIF format files or in an internal binary format portable between different platforms. At the level of the library's interface function there is no difference in handling different formats. The MMDB library provides a number of tools for working with coordinate data, including orthogonal-fractional coordinate transformations, generation of symmetry mates, editing of the molecular structure and finding atomic contacts. More information can be found on the project web pages (http://www.ebi.ac.uk/~keb/cldoc/).

From the point of view of the user, an important development is the specification of an atom-selection syntax which will be standard across *CCP*4. For example, the C^{α} atom of residue 13 of chain *A* is specified as A/13/CA[C]. Omissions are interpreted as suitable defaults, so that A/13 means all atoms in residue 13 of chain *A*. Lists and ranges can be specified, so that A,B/10-50 means all atoms in the residue range 10–50 in both chains *A* and *B*. Finally, provision is made for model numbers, residue types, insertion codes and alternative location indicators. The use of a standard syntax replaces the program-dependent syntax currently in use and will aid communication between programs.

In the longer term, CCP4 is developing a Molecular Graphics viewer (see Potterton, 2001; Potterton, McNicholas *et al.*, 2002). Non-graphical components of the MG viewer, which prepare the model for display and perform calculations on the model, are being written as extensions of the MMDB library. Extensions will include calculation of bond lengths, angles and dihedral angles, calculation of solvent-accessible surface area and assignment of secondary structure. This functionality will also be available to non-graphical *CCP*4 applications.

The CMTZ library is centred on a reinterpretation of the MTZ reflection file. Data are still arranged in columns identified by user-definable column labels, but these columns are now grouped according to the dataset to which they belong. From version 3.5 of the *CCP*4 suite, DATASET records in the MTZ file header list the datasets included in the file and each COLUMN record includes a reference to the associated dataset. Data sets are further grouped into projects, identified by PROJECT records in the MTZ file header. The project and dataset information was first used for Data Harvesting (see §5), but is now being used more generally.

A further level of description is currently being implemented, namely the crystal level included as CRYSTAL

records in the MTZ file header. Fig. 2 illustrates the new hierarchy. Data in an MTZ file is segregated according to the crystal from which it was measured or derived. Each crystal is further divided into datasets, which represent distinct measurements on the crystal, for example at different radiation wavelengths. Finally, each dataset consists of several columns of data. The dataset level corresponds to the dataset introduced previously. The project name does not fit naturally into the new hierarchy and is treated as an attribute of the crystal. For the purposes of this classification, derived data columns are assigned to the most appropriate dataset, although there may sometimes be ambiguity. Thus, for example, a calculated structure factor would belong to the dataset against which the model was refined.

The data hierarchy is encoded by records in the MTZ file header. This is clearly not a true representation of the hierarchy, but ensures compatability with older MTZ files. On loading an MTZ file, the CMTZ library reconstructs the true hierarchy in memory and all further manipulations are performed on this hierarchy. In the current implementation, all reflections are held in memory as column-associated arrays.

The reflection-data model is beginning to be utilized in some *CCP*4 programs and will increasingly become central to the way data is manipulated in *CCP*4. *SCALEIT* and *FHSCAL* are *CCP*4 programs which scale together native and derivative datasets. Scale factors are estimated and applied to the derivative data. The user specifies the MTZ columns which comprise the derivative data *via* the LABIN keyword. However, inconsistencies would result if say only FPH and DPH columns (the mean structure factor and the anomalous difference) were scaled and not $F(+)$ and $F(-)$ columns (the Bijvoet pair). Data set information is used to identify columns of data that should be scaled in a consistent manner. Use of the AUTO keyword in *SCALEIT* and *FHSCAL* activates dataset-based scaling; otherwise a warning is issued if potential inconsistencies are detected. This is an example of using a proper data model to maintain data integrity.

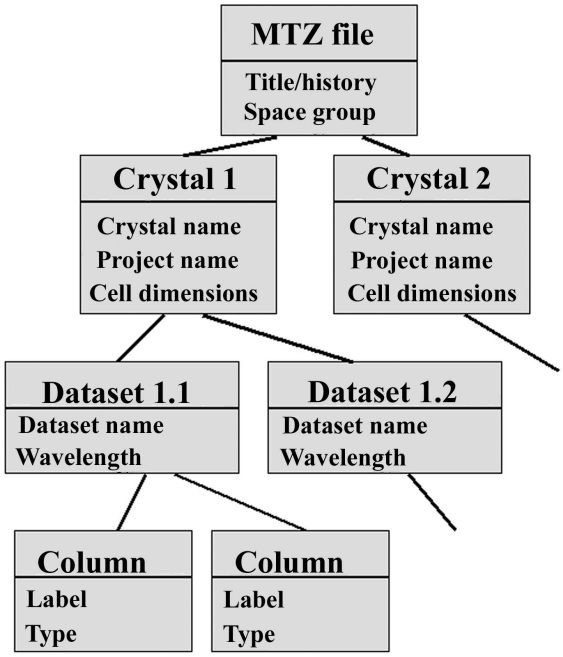

Figure 2
A schematic of the reflection-data hierarchy implemented in the CMTZ software library, as described in the text. Attributes of each level of the hierarchy are listed in the boxes.

Dataset information is also being used in the *CCP*4 program *SCALA*, which scales and merges multiple observations of a reflection. Unmerged data files processed by *SCALA* typically contain a single dataset, but may contain multiple datasets if for instance multiple-wavelength datasets are being scaled together or if a reference set is present. Each batch in the unmerged file (corresponding to a diffraction image) is assigned to a specific dataset. When there is more than one dataset, the dataset classification is used as follows. The scaling algorithm works on sets of batches termed 'runs' and by default a run is assigned for each dataset. A dataset may need to be split into multiple runs, but the default assignment is the minimum division that is required. After scaling and merging, the merged reflections are output to MTZ files and a separate file is created for each dataset. Finally, various analyses are performed between datasets, comparing the anomalous differences and the dispersive differences from a defined base set. Thus, in *SCALA*, datasets are used to control the handling of a large number of images and to automate (in the sense of providing sensible defaults) the running of the program.

CMAP is principally an i/o library for *CCP*4-format map files. There are functions to read and write map file headers, functions to retrieve and set specific values in the header and functions to read and write map sections or parts of sections. In addition, there is a Fortran interface which mimics the current maplib.f.

*CCP*4 software uses several aspects of crystallographic symmetry, for example symmetry operators, reciprocal- and real-space asymmetric units, centric and epsilon zones, Patterson groups *etc.* The CCP4 approach has been to hold some space-group information in a file symop.lib, namely space-group number and name, associated point group and crystal class and a list of symmetry operators, and derive all other information as needed. Although this approach has been made to work for all commonly occurring space groups, there are problems. Firstly, symop.lib is incomplete, with only a handful of alternative space-group settings being included. Secondly, the derivation of some quantities, such as the reciprocal-space asymmetric units, is buried in Fortran routines and is not easily maintainable or extensible.

Grosse-Kunstleve and coworkers (Grosse-Kunstleve, 1999; Grosse-Kunstleve & Adams, 2002) have developed general methods for generating symmetry information for all space groups in all settings. These methods have been implemented in the sgtbx component of the Computational Crystallography Toolbox. Rather than duplicate their work, we have opted to use sgtbx to generate an extended data file, named syminfo.lib, which will be distributed with the *CCP*4 software suite. The information derived from sgtbx is supplemented by *CCP*4-

specific data for backwards compatibility. syminfo.lib contains entries for all space groups in all settings. For each entry, the following items are listed: true space-group number, the *CCP*4 space-group number (for example, 1003 for space group 3 with *c* axis unique), the change of basis for non-standard settings, a variety of symbols for the space group and the associated Laue, Patterson and point groups, the reciprocal-space asymmetric unit, a choice of real-space asymmetric units, the primitive symmetry operators and the centring operators for non-primitive groups.

The CSYM library has been written to read in symmetry information for a particular space group from the syminfo.lib file, as identified by its number, name or symmetry operators. This information is held in memory and is accessible directly to programs. In addition, there is a Fortran API which mimics the current symlib.f.

As an independent development, Kevin Cowtan has developed Clipper, which is a set of object-oriented libraries for the organization of crystallographic data and for crystallographic computation. Clipper provides classes for cell and space-group objects, reflection-data objects, crystallographic and non-crystallographic map objects *etc.* It is likely that some *CCP*4 applications will make use of the Clipper libraries. For more details, see the project web site (http://www.yorvic.york.ac.uk/~cowtan/clipper/clipper.html).

In this section, we have described briefly some of the new libraries being developed by CCP4. These libraries will allow development in languages other than Fortran and will provide advanced functionality so that applications can be written more quickly. From the point of view of high-throughput methodologies, the most important aspect of the new CCP4 library is the availability of crystallographic functions to scripting languages. Scripting allows rapid development and testing of applications and allows easy customization for local projects. Within *CCP*4, the Tcl interface for example will allow direct access to crystallographic functionality from ccp4i.

Interfaces to scripting languages are provided by *SWIG* (Simplified Wrapper and Interface Generator; http://www.swig.org). For C language libraries such as CMTZ and CMAP, the interface can be generated directly from the C header files. Thus, the interfaces automatically keep track of changes in the underlying libraries. For the C++ language libraries such as MMDB, there are additional problems which require manual maintenance of the interface. For example, scripting languages do not support function overloading and explicit renaming of functions is required.

4. Improving the user interface

One of the major developments in the *CCP*4 software suite recently has been the development of a graphical user interface 'ccp4i' (Potterton, Briggs *et al.*, 2002). User interfaces may be considered to be the antithesis of automation and in some ways this is true. However, when a process involves many steps, as is the case with using *CCP*4 programs, the interface can provide a framework in which complex sequences are automated. Thus, ccp4i includes a significant amount of scripting between the user-visible interface and the *CCP*4 programs, which provides the glue between individual programs, automates routine tasks and provides data-management facilities. In this section, we discuss ccp4i in this context, rather than the user interface *per se*.

The principal unit of ccp4i is the task. A task is often based around a single run of a program, for example *ACORN*, *MOLREP* or *REFMAC*. In these cases, additional jiffy programs are often run to perform file-format conversions or post-processing such as generating maps. These additional steps are invisible to the user or are selectable by a single check button. In other cases, the task may perform a more complicated sequence of programs, for example the *AMoRe* task which runs the *AMoRe* program multiple times, or the NCS Phased Refinement task, which cycles through the programs *PDBSET*, *LSQKAB*, *DM* and *REFMAC*.

ccp4i continues to be under active development, with *CCP*4 version 4.2 containing new tasks for *BEAST*, *ACORN*, *OASIS*, *TLSANL*, *ANISOANL* and others. Additional utilities include a built-in map viewer, named *MAPSLICER*, suitable for checking map sections (this effectively replaces the old npo plus plot84driver combination). *MAPSLICER* allows easy traversal through map sections, selection of Harker sections, interactive setting of contouring levels and scale factor, and choice of sectioning axis. A

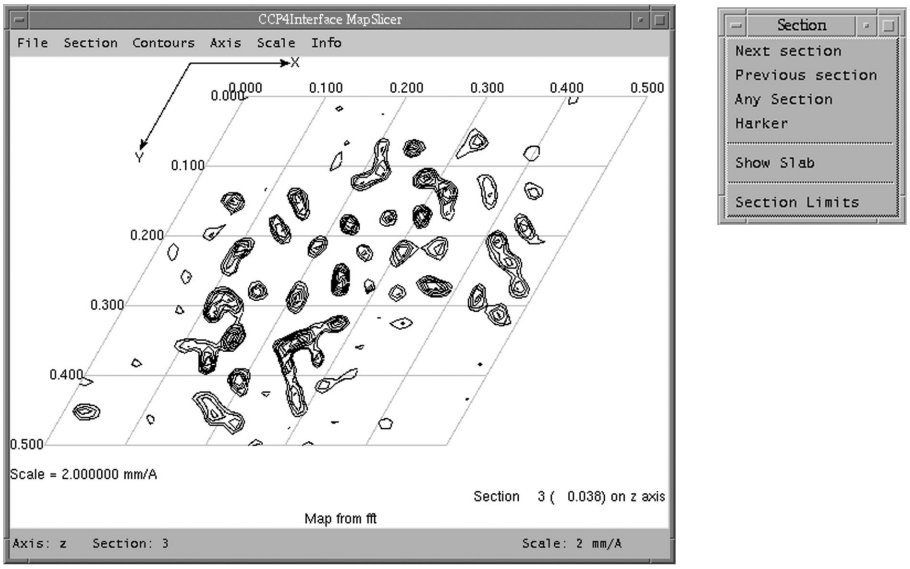

Figure 3
A snapshot of the *MAPSLICER* utility from ccp4i. On the right is a tear-off menu for navigating through map sections.

snapshot of the viewer is shown in Fig. 3.

ccp4i-style interfaces are beginning to be used in other contexts. A number of third-party programs are developing graphical user interfaces using the ccp4i toolkit, for example *ARP*/*wARP* (Perrakis *et al.*, 1999) and *SHELXD* (Sheldrick, 1997). These interfaces remain under the control of the program authors, but can be imported into ccp4i *via* a specialized import task accessible from the main ccp4i window. The CCP4 Molecular Graphics viewer (Potterton, 2001; Potterton, McNicholas *et al.*, 2002) will also incorporate command interfaces based on ccp4i.

The current incarnation of ccp4i eases the burden on the user and thus facilitates fast structure solution, but it is still very much user-driven. The challenge is to allow larger portions of the structure-solution process to be performed without user intervention. This in turn requires ccp4i to make sensible decisions based on the current state and some elementary decision-making is now being used. As mentioned above, the program *SCALA* outputs a separate file for each dataset that is processed. The *SCALA* task in ccp4i detects the number of datasets and runs each output file through *TRUNCATE*, merges the resultant files with *CAD* and finally runs *UNIQUEIFY*. Thus, a significant amount of processing is performed automatically in a data-dependent way.

A recurrent theme of automation is error checking. Error checking through strong data typing, in particular the use of MTZ column types, has long been a part of *CCP*4. The ccp4i layer adds an additional set of types for program parameters which can be used for checking and setting automated actions. For example, cell parameters have types _cell_length and _cell_angle, while program-specific parameters can be defined as _int, _positiveint *etc*. More sophisticated checking is continually being added to the suite. As a recent example, the Merge Datasets task in the Experimental Phasing module includes a single button which activates a checking procedure for consistent indexing between datasets in different MTZ files. If inconsistent indexing is detected, then the relevant datasets are reindexed automatically.

A direct consequence of high-throughput studies is the increasing number of projects each user may be involved with. In addition, projects are not the property of a single user and in a production-line setup will be passed between persons with different expertise. ccp4i provides some simple project-management tools which allow organization of the work within a project. A simple database (implemented as a collection of flat files) keeps a record of all jobs run and for each job it records the parameters used, the name of the log file and the names of the input and output files. This system allows the user to review earlier steps and to rerun a job with the same or similar parameters. In principle, the same facilities could be used by an automated procedure to rerun the same task many times, varying either certain parameters of interest or varying the data supplied.

A ccp4i database is created for each project defined by the user. A user may have access to many projects, but can only work within one project at a time. Several users may have access to one project. Future plans include tools to package and transport projects between sites, to merge projects which were created separately but belong together and to archive completed projects. The ccp4i database is a simple example of data management, which we now discuss in more detail.

5. Data management

Handling large amounts of data is an integral part of high-throughput structure determination. At the most basic level, this means increased storage capacity, faster networking and more processing power. However, correct management of the data is also vital in order to maintain data integrity and completeness. There should be extensive use of metadata to annotate the data which is measured or derived.

Within a particular laboratory or collaborative project, data management can be performed by a Laboratory Information Management System (LIMS; see, for example, Haebel *et al.*, 2001). Such a system can manage data for many projects, keeping track of evolving data in a central database facility. LIMS are best customized to the requirements of a particular site and are therefore not a realistic option for the *CCP*4 software suite. In the context of *CCP*4, we must look for simpler and more generic solutions to data management.

Data management in *CCP*4 is currently based around standard file formats, the most important of which is probably the MTZ file. The latter consists of a reflection-data block together with a header section which holds a description of the data. The data block can hold columns of native data, derivative data, MAD datasets for different wavelengths, experimental phases, calculated structure factors, map coefficients, *etc*.; in short, all measured and derived data for a particular project are kept together (subject to the sole restriction of a common space group). The header section contains metadata such as column labels, column types, cell dimensions, history *etc*., which makes the data file self-documenting to a large extent

The other standard *CCP*4 file formats are the map format for electron-density maps and masks, and a limited PDB file for coordinate data. In addition, *CCP*4 uses mmCIF files in some situations, for example holding restraint information in *REFMAC*5. Central to the mmCIF format is the dictionary of allowed data items, which provides a catalogue of explicitly defined tokens of crystallographic interest. The mmCIF dictionary is designed to be extensible and new data items are added with new versions. Although the mmCIF file format is not suitable for all applications, the dictionary is an important central resource for macromolecular crystallography. The imgCIF/CBF project for two-dimensional image data is also relevant to macromolecular crystallographers.

While the standard file formats are the principle repository of data during structure solution, they do not hold all information that is of interest. Administrative information about a project, in particular details of the jobs run, is held in the ccp4i 'database' and has been described above. In addition, there are various quantities that are required for steps in the structure-solution process which are calculated in earlier steps. Traditionally, these are copied manually from the output of

one job to the input of the later job. A simple example is the estimate of the number of molecules in the asymmetric unit, which is an input parameter to a molecular-replacement search. Manual copying is clearly error prone, as well as explicitly requiring human intervention.

A number of automated procedures (see *e.g.* Badger & Hendle, 2002) parse the log output of programs to capture this additional information. While this approach can be made to work, it is at the mercy of changes in the program output. We therefore intend to adapt critical programs to output the required information into separate XML-formatted files. The use of separate files allows this approach to be implemented in a piecemeal fashion without disrupting established file formats. The output needs to be marked up to enable input to subsequent programs and XML is a convenient choice with many established tools. The XML format is not suitable for large quantities of data, such as reflection data, and there is no question of it replacing the standard *CCP*4 file formats.

The result is a number of XML files each holding pieces of information generated earlier in the process. By default, the XML files will be stored in the ccp4i database of the project to which they refer. In principle, the XML files could be uploaded to a full project relational database, perhaps as part of a local LIMS, but as mentioned above CCP4 is not in a position to impose such a solution on users. On launching a task in ccp4i, the project database is checked for relevant XML files and if any are found then default parameters are set from them. For example, in the *MOLREP* task interface, the number of molecules and the vector for translational NCS can be set in this way. Note that these automatically generated parameters can always be overridden by the user.

In this context, XML is used to communicate between jobs which may be separated in time. XML files may also be used for communication in automated procedures that consist of separate steps which are run together. As described above, there is a task option in ccp4i to check consistent indexing between datasets that are to be merged. An XML file containing the reindexing operator is used to communicate between the program which detects that reindexing is necessary and the program which performs the reindexing. As a further example of XML, the DNA project for integrated data collection and processing uses XML for passing information between components (see Leslie *et al.*, 2002).

Data management is important not only for easing and automating structure solution, but also for allowing all relevant information to be deposited correctly in the Protein Data Bank and related databases. 'Relevant' here means information on how the final model was arrived at, as well as the model itself, and thus covers experimental data (Dodson *et al.*, 1996) and details of the structure-solution process.

One of the main technologies for ensuring complete and accurate deposition is 'Data Harvesting' (Winn, 1999). This protocol means that software used in structure solution outputs to a deposition file details of the method used and results obtained, for example heavy-atom sites used in phasing. By the time the user is ready to deposit the model coordinates, there should be a collection of files holding details of how the model was obtained. These files can be sent directly to the deposition centre, thereby bypassing much of the manual input of data traditionally required. Version 3.1 of *AutoDep* supports uploading of harvest files from *CCP*4 and *CNS* (Brünger *et al.*, 1998). *CCP*4 currently produces harvest files from *MOSFLM*, *SCALA*, *TRUNCATE*, *MLPHARE*, *RESTRAIN* and *REFMAC*.

During the structure-solution process, harvesting files are categorized according to a Project Name and a Dataset Name. The Project Name specifies the structure-solution project and is equivalent to what will become a PDB ID code (or in mmCIF terms the _entry.id). The Dataset Name identifies the particular dataset within the project that is being used (_diffrn.id in mmCIF). The latter may for example be X-ray diffraction structure factors or experimentally determined NMR data. Thus, a particular structure solution may involve several datasets with the same Project Name but distinguished by different Dataset Names (*e.g.* for native and heavy-atom derivatives or for different wavelengths in a MAD experiment). Alternatively, one may have several datasets for an apoprotein and its complexes and these would be distinguished by different Project Names, since they correspond to different structure solutions.

Project Names and Dataset Names are carried in the MTZ file header. They need only be set once and this should be done when an MTZ file is first created, *e.g.* in the programs *MOSFLM*, *COMBAT*, *SCALEPACK2MTZ* or *F2MTZ*. Data-harvesting programs output a file into a subdirectory named after the Project Name. The file itself has the name ⟨Dataset Name⟩.⟨Program Name⟩. In the current implementation, only the latest file from a particular program run on a particular dataset is retained. The NOHARVEST keyword can be used to prevent output from trial runs overwriting earlier harvest files.

In addition to exporting data to databases such as the PDB, usage of the *CCP*4 suite increasingly requires the import of data from databases. A new ccp4i task allows one to import a protein sequence into the suite based on its Swiss-Prot reference code. Some editing of the sequence is allowed to account for Swiss-Prot errors, mutations or domain selection. A record of the original sequence and the editing operations is kept along with the edited sequence. The protein sequence may subsequently be used in molecular replacement, for model building and for structure deposition.

MDW and PJB are supported by the Biotechnology and Biological Sciences Research Council through the CCP4 grant (B10200). Development of *MOSFLM* is partially supported by the EU Autostruct project (QLRI-CT-2000-30398). Autostruct also supports the development of ccp4i-style interfaces to *ARP*/*wARP* and *SHELX*. The EU MAXINF project (EU-HPRI-CT-2000-40021) has provided travel funds for the DNA project. Finally, current work on Data Harvesting is supported by the EU Temblor project (QLRT-2001-00015). CCP4 depends on the contributions of many people including staff, collaborators and users of the software and their role is

gratefully acknowledged. It is impossible to acknowledge all contributors by name, but particular thanks go to Eleanor Dodson, Phil Evans, Liz Potterton, Garib Murshudov, Eugene Krissinel, Kevin Cowtan, Airlie McCoy, Alexei Vagin, Maria Turkenburg, Harry Powell and Kim Henrick.

References

Badger, J. & Hendle, J. (2002). *Acta Cryst.* D**58**, 284–291.
Brünger, A. T., Adams, P. D., Clore, G. M., DeLano, W. L., Gros, P., Grosse-Kunstleve, R. W., Jiang, J.-S., Kuszewski, J., Nilges, N., Pannu, N. S., Read, R. J., Rice, L. M., Simonson, T. & Warren, G. L. (1998). *Acta Cryst.* D**54**, 905–921.
Collaborative Computational Project, Number 4 (1994). *Acta Cryst.* D**50**, 760–763.
Diederichs, K. (2000). *J. Appl. Cryst.* **33**, 1154–1161.
Dodson, E. J., Kleywegt, G. J. & Wilson, K. (1996). *Acta Cryst.* D**52**, 228–234.
Dodson, E. J., Winn, M. D. & Ralph, A. C. (1997). *Methods Enzymol.* **277**, 620–633.
Foadi, J., Woolfson, M. M., Dodson, E. J., Wilson, K. S., Jia-xing, Y. & Chao-de, Z. (2000). *Acta Cryst.* D**56**, 1137–1147.
Grosse-Kunstleve, R. W. (1999). *Acta Cryst.* A**55**, 383–395.
Grosse-Kunstleve, R. W. & Adams, P. D. (2002). *Acta Cryst.* A**58**, 60–65.
Haebel, P. W., Arcus, V. L., Baker, E. N. & Metcalf, P. (2001). *Acta Cryst.* D**57**, 1341–1343.
Holton, J. M. (2002). In preparation.
Kleywegt, G. J. & Jones, T. A. (1997). *Acta Cryst.* D**53**, 179–185.
Leslie, A. G. W. (1992). *Jnt CCP4/ESF–EAMCB Newsl. Protein Crystallogr.*, **26**.
Leslie, A. G. W., Powell, H. R., Winter, G., Svensson, O., Spruce, D., McSweeney, S., Love, D., Kinder, S., Duke, E. & Nave, C. (2002). *Acta Cryst.* D**58**, 1924–1928.
McAuley, K. E., Jia-xing, Y., Dodson, E. J., Lehmbeck, J., Ostergaard, P. R. & Wilson, K. S. (2001). *Acta Cryst.* D**57**, 1571–1578.
Murshudov, G. N., Vagin, A. A. & Dodson, E. J. (1997). *Acta Cryst.* D**53**, 240–253.
Perrakis, A., Morris, R. J. & Lamzin, V. S. (1999). *Nature Struct. Biol.* **6**, 458–463.
Potterton, E. (2001). *CCP4 Newsletter*, Number 39, Article 6.
Potterton, E. A., Briggs, P. J. & Turkenburg, M. G. W. (2002). In preparation.
Potterton, E., McNicholas, S., Krissinel, E., Cowtan, K. & Noble, M. (2002). *Acta Cryst.* D**58**, 1955–1957.
Read, R. J. (2001). *Acta Cryst.* D**57**, 1373–1382.
Schneider, T. R. & Sheldrick, G. M. (1997). *Methods Enzymol.* **277**, 319–343.
Sheldrick, G. M. (1997). *Proceedings of the CCP4 Study Weekend. Recent Advances in Phasing*, edited by K. S. Wilson, G. Davies, A. W. Ashton & S. Bailey, pp. 147–158. Warrington: Daresbury Laboratory.
Vagin, A. & Teplyakov, A. (1997). *J. Appl. Cryst.* **30**, 1022–1025.
Winn, M. D. (1999). *CCP4 Newsletter*, Number 37, Article 13.
Yao, J.-X. (2002). *Acta Cryst.* D**58**, 1943–1949.