Program Suites

Harry Powell MRC-LMB

1 general introduction to suites

2 a quick look the differences between a small molecule/powder suite and one for proteins

3 a more detailed look at CCP4 and and example of using its built in tools

A definition of a program suite:

A set of complementary programs designed for use within a common domain

The programs may have a common "look and feel" or might be tied together through some external binding (e.g. a GUI or a similar command-line syntax).

Why contribute to a suite rather than develop your own packages?

- infrastructure pre-exists & is already tested
- someone else has already looked at porting
- often a large and established user base
- plenty of help in using underlying structure
- licensing should have been worked out
- someone else can take care of distribution (and, to some extent, users & their problems)

Why *not* contribute to a suite and develop your own packages instead?

- you may prefer another licence (GPL, LGPL, etc)
- it can be more restrictive & harder to develop in your own style
- your software may pre-exist and would need major changes to fit in with the suite
- you may want to maintain independence
- you may want to compete with the status quo

Macromolecular Crystallography
- CCP4
- Phenix

Chemical Crystallography
- Crystals
- Platon/System S
- WinGX
- CCP14

Big differences:
- MX suites tend to be looked after by multiple developers, CX by a single person (or a tiny group).
- MX suites tend to be better developed for Unix systems, CX for MS-Windows.

CCP4 model vs CCP14 model

Both are principally funded by grants from UK Research Councils.

CCP4 is a single set of programs which use a common input/output & data storage model; the software is ported and curated by a funded group of people, and licences are sold to commercial companies. Core funding currently by 5-year grant from BBSRC.

CCP14 is a collection of software which uses many file formats for i/o; no attempt is made to bring the various component programs into a common model. Funding currently by a 5-year grant by EPSRC.

(1) Chemical Crystallography

CCP14 - proposed in 1994, funded since 1995 - gathers together several complete small molecule and powder crystallography tools *e.g.*

Crystals, WinGX, ORTEX, Platon/System S, GSAS

and loads of software for individual tasks *e.g.*

SHELX*, SIR*, *etc, etc*

These are all developed independently and made available by their authors. Essentially there is a single member of staff who does not develop the software.

(2) Protein Crystallography

CCP4 then and now

1979 - various groups in PX in the UK set up a collaboration to produce a common framework for the essential software for protein structure solution (SERC funded).

2005 - 17 funded developers (supported by BBSRC + income from sales) + 7 other main contributors/developers. Executive committee of 7 people. Contributions from other groups.

Software distribution then and now

1979 - Relatively easy to share software produced in an academic environment (employers were happier to allow it with few restrictions).

2005 - Can be much harder to release software to the wider world; even public sector employers put stringent requirements on licensing & release of software, patent offices seem happy to ignore prior art in issuing patents. This can lead to problems for software developers (google for “erroneous software patents”) both in protecting their genuine innovations and in infringing spuriously granted patents.

CCP4 contains

- ~175 programs (mostly Fortran) which cover most aspects of macromolecular crystallography
  - diffraction image processing
  - reflection dataset analysis
  - structure solution (MIR, MR, Direct Methods...)
  - model building
  - structure refinement
  - analysis of results
  - validation
  - etc...

CCP4 contains

- comprehensive software libraries for use by applications -
  - Crystallographic functions, e.g.
    - symmetry
    - unit cell manipulations
    - FFT calculations (© Lynn ten Eyck)
  - general operations, e.g.
    - smart file opening/closing for many OS’s
    - reading/writing reflection files
    - reading/writing coordinate files
    - keyword parsing

CCP4 runs on all commonly used platforms (Linux, Mac OS X, Irix, Tru64, Solaris, MS-Windows...)

- the hard work of porting has already been done
- CCP4 staff can give help in porting your application
- contact *via* `ccp4@dl.ac.uk`

Q If there are already 175 applications how can I contribute?

A(i) Some functions aren't covered (recent additions include molecular graphics (ccp4mg) and model building (COOT), but there is still room for more).

A(ii) Who said they're perfect? Some programs are decades old, and your ideas may be an improvement

A(iii) Contribute to *part* of the project but not to the core, e.g. SHELXS, ARP/wARP & Phaser have ccp4i interfaces but are not part of CCP4.

In CCP4, reflections are stored in an MTZ file which has a binary (*i.e.* not ASCII) format. In practice, this does not present problems in reading from or writing to the file - you just need to use the appropriate CCP4 tools (or you could try Ralf Grosse-Kunstleve's alternatives).

"The MTZ reflection file format ... [was] renamed from LCF for three of its progenitors, Sandra McLaughlin, Howard Terry, and Jan Zelinka"

An MTZ file is more than just a reflection file; it is a container for a data structure with experimental information, *e.g.*

- unit cell information (dimensions & symmetry)
- reflection data (h,k,l,F,σ(F),I,σ(I), coordinates on image, etc)
- for multiple crystals

Can be viewed and manipulated using standard CCP4 programs, *e.g.*

- `mtzdump` - for viewing the contents
- `mtz2various` - for writing to other programs' formats (e.g. SHELX, TNT, X-PLOR/CNS/CIF...)
- `mtzutils` - to change the information contained

Examples of code to manipulate MTZ files;

`http://www.ccp4.ac.uk/dev/templates/templates.php`

```
* Title:
.
* Base dataset:
        0 HKL_base
          HKL_base
          HKL_base
* Number of Datasets = 1
* Dataset ID, project/crystal/dataset names, cell dimensions, wavelength:
        2 sorted
          hg
          camillo
             58.4479   58.4479  156.0206   90.0000   90.0000  120.0000
             0.99970
* Number of Columns = 9
* Number of Reflections = 4925
* Missing value set to NaN in input mtz file
* HISTORY for current MTZ file :
 From SCALA: run at 12:04:08 on  3/ 7/05
 From SORTMTZ  3/ 7/2005 11:25:14  using keys: H K L M/ISYM BATCH
 From MOSFLM run on  3/ 7/05
* Column Labels :
 H K L IMEAN SIGIMEAN I(+) SIGI(+) I(-) SIGI(-)
* Column Types :
 H H H J Q K M K M
* Associated datasets :
 0 0 0 2 2 2 2 2 2
```

```
* Cell Dimensions : (obsolete - use crystal cells)
   58.4479   58.4479  156.0206   90.0000   90.0000  120.0000
 *  Resolution Range :
    0.00117    0.19363     (     29.223 -      2.273 A )
 * Sort Order :
      0     0     0     0     0
 * Space group = 'H32' (number     155)
 OVERALL FILE STATISTICS for resolution range   0.001 -   0.194
 ======================
 Col Sort  Min  Max  Num    %   Mean   Mean Resolution Type Column
 num order              Missing complete          abs.   Low    High
    labe
l

 1 ASC    0    22     0 100.00   10.6   10.6  29.22 2.27 H  H
 2 NONE   0    12     0 100.00    3.6    3.6  29.22 2.27 H  K
 3 NONE -68    68     0 100.00    1.2   25.9  29.22 2.27 H  L
 4 NONE -78.8 36533.5 0 100.00  820.07 820.28 29.22 2.27 J  IMEAN
 5 NONE   8.5 2066.7  0 100.00   62.64  62.64 29.22 2.27 Q  SIGIMEAN
 6 NONE -98.0 36399.4 0 100.00  775.22 776.09 29.22 2.27 K  I(+)
 7 NONE   0.0 3171.7  0 100.00   65.21  65.21 29.22 2.27 M  SIGI(+)
 8 NONE -85.1 36632.4 0 100.00  782.03 782.57 29.22 2.27 K  I(-)
 9 NONE   0.0 2724.5  0 100.00   65.29  65.29 29.22 2.27 M  SIGI(-)


 No. of reflections used in FILE STATISTICS     4925
```

```
 LIST OF REFLECTIONS
 ===================

   0   0  51    5487.92   424.86     0.00     0.00   5487.92    424.86

   0   0  54     146.24    38.18     0.00     0.00    146.24     38.18

   0   0  57    2737.31   209.20     0.00     0.00   2737.31    209.20

   0   0  60     542.89    66.86     0.00     0.00    542.89     66.86

   0   0  63    1148.36   129.10     0.00     0.00   1148.36    129.10

   0   0  66      67.84    53.68     0.00     0.00     67.84     53.68

   1   0 -68    1095.55    64.61  1095.55    64.61      0.00      0.00

   1   0 -65      51.30    27.21    51.30    27.21      0.00      0.00

   1   0 -62     489.80    46.13   489.80    46.13      0.00      0.00

   1   0 -59     101.72    33.04   101.72    33.04      0.00      0.00
  h   k   l    IMEAN    SIGIMEAN    I(+)    SIGI(+)    I(-)     SIGI(+)
```

Access the information with low level tools (written in C) for reading from MTZ structure with FORTRAN

e.g.

```
LROPEN - opens mtz file & does some checks
LRINFO - numbers of reflections, data columns, ranges
LRTITL - title from file
LRCELL - cell dimensions of 1st crystal in the data
   structure
LRRSOL - overall resolution limits
LRSYMI - symmetry information (names & numbers, point &
   space groups, etc)
LRCLOS - close MTZ file
```

etc., etc...

http://www.ccp4.ac.uk/dist/html/ccplib.html

contains details on low-level ccp4 routines which deal with (among many other things) potentially machine dependent behaviour, *e.g.*

CCPFYP - sets up environment & parses command line arguments

CCPDPN - opens files, forces program to address error issues

CCPONL - tests to see if program is being run interactively

LITEND - determined endedness of current architecture

CCP4 example - FORTRAN opening an MTZ reflection file & printing the header

```
      PROGRAM MTZOPEN
      IMPLICIT NONE
      INTEGER MTZIN,MTZPRT,MTZERR
C CCP4 initialisations
      CALL CCPFYP
C MTZ-specific initialisations
      CALL MTZINI
      MTZIN = 1
      CALL LROPEN(MTZIN,'HKLIN',MTZPRT,MTZERR)
      IF (MTZERR.EQ.-1) THEN
         CALL CCPERR(1,' LROPEN no such file for HKLIN')
         STOP
      ENDIF
      CALL LRCLOS(MTZIN)
      END
```

CCP4 example - C opening an MTZ reflection file & printing the header

```
#include "/Users/harry/ccp4-5.0.2/include/ccp4/cmtzlib.h"
int main(int argc, char **argv) {
  int *iprint;
  MTZ *file1=NULL;
  if (argc != 2) {
    puts("Usage: mtzopen <file>");
    exit(1);
  }
  file1 = MtzGet(argv[1],1); /* opens MTZ file and reads header */
  if (!file1) {
    printf("FATAL: failed to read file \"%s\"\n",argv[1]);
    exit(1);
  }
  MtzAssignHKLtoBase(file1); /* assigns base dataset */
  if (*iprint > 0) ccp4_lhprt(file1, *iprint); /* prints header */
  MtzFree(file1); /* closes MTZ file */
  printf("Done:   closed file \"%s\"\n",argv[1]);
}
```

Build Fortran example:

```
f77 -o fmtzopen mtzopen.f $CLIB/libccp4f.a $CLIB/libccp4c.a
```

run:

```
cmtzopen HKLIN <filename>
```

Build C example:

```
cc -o cmtzopen mtzopen.c $CLIB/libccp4c.a
```

run:

```
cmtzopen <filename>
```

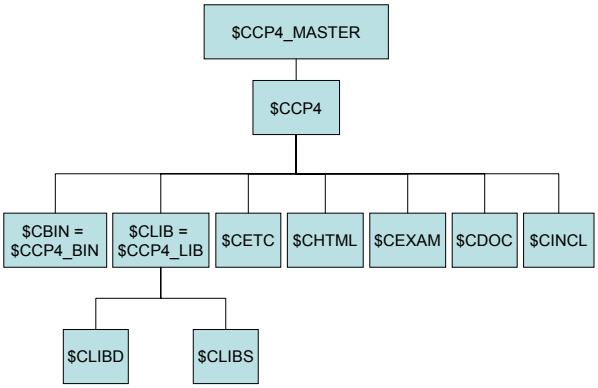
CCP4 development environment (UNIX)

1. download from www.ccp4.ac.uk
2. install by non-root user -

- set up local environment by editing “ccp4.setup-inst” (for csh/tcsh) or “ccp4.setup-bash” and source-ing it to create a set of environment variables (logicals) defining the basic CCP4 directory structure, containing (among others);

  $CCP4 - top-level directory
  $CLIB - directory containing link libraries
  $CPROG - executables
  $CINCL - run-time include files, e.g. for symmetry, extra logicals, ccp4.setup-#### files

Basic CCP4 directory hierarchy



Further online help -

`http://www.ccp4.ac.uk/dev/templates/templates.php`

basics for MTZ, maps, pdb files and symmetry lookup using CCP4.

`http://www.ccp4.ac.uk/dist/html/symlib.html`

general CCP4 programming tips:

`http://www.ccp4.ac.uk/dist/html/ccplib.html`

To summarize:

There are definite advantages to developing software within the context of a coherent suite such as CCP4 or PHENIX, since many of the issues regarding *e.g.* basic crystallography or file handling will have already been implemented as ready-to-use tools. This is at the cost of some personal programming flexibility, since someone else will have decided the rules!