Diffraction Integration for Advanced Light Sources: Workshop 1
Diamond Light Source, 13 June 2012

Collaborative Software for Data Reduction

Nicholas K. Sauter, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94720, U.S.A. Email: nksauter@lbl.gov.

1. Abstract. The DIALS-1 Workshop is a watershed for X-ray diffraction methods, offering an opportunity for software developers and high-throughput beamline users to explore and define the expected needs of near-future crystallographers. This article briefly describes the current software efforts at Lawrence Berkeley National Laboratory (LBNL), proposes one possible implementation of software collaboration, and illustrates three prototype applications that exemplify high-performance computing, serial femtosecond crystallography (SFX), and synchrotron data reduction.

2. Current LBNL Projects. Our data reduction efforts are presently funded by two sources. LBNL has initiated a one-year Laboratory Directed Research and Development project for computational methods aimed at elucidating the photosynthetic mechanism of water oxidation by photosystem II. Water splitting is catalyzed by a Mn_4Ca cluster that mediates a four-photon, four-electron redox cycle. In our study, photosytem II crystals will be pumped around this cycle with an optical laser. Intermediate forms will be probed by a combination of X-ray crystallography, to detect any changes in the atomic structure, as well as simultaneous Mn X-ray emission spectroscopy, which will monitor the oxidation state of the catalyst. While this experiment is not possible at synchrotron sources due to the extreme radiation sensitivity of Mn (1), our preliminary data (2) suggest that it is feasible at the Linac Coherent Light Source (LCLS), with the Mn center remaining intact on the 50 fs time scale achievable with X-ray free electron laser (XFEL) pulses. We will therefore reconstruct the diffraction pattern using "diffract and destroy" still shots taken from thousands of microcrystals.

The U.S. National Institutes of Health (NIH) has funded a four-year technology project, *Realizing new horizons in X-ray crystallography data processing*, aimed at synchrotron facilities. With a new generation of pixel-array detectors ensconced at both conventional and XFEL sources (3, 4), it is clear that updated software is needed to correctly model unique detector properties such as the extremely small point spread function, along with neighboring-pixel charge sharing. Very high image framing rates of 10-120 Hz will require the fastest parallel computing approaches, possibly involving graphics processing unit (GPU) hardware. Moreover, it will be advantageous to create improved algorithms to treat marginal data in general, and specifically diffraction patterns exhibiting two or more lattices (5) and various special phenomena like incommensurate modulation (6, 7) as well as lattice (8) and twinning (9) disorders.

3. A Mandate for Collaboration. Recent funding announcements from the NIH convey a strong expectation that software developed under NIH programs be shared with the public. To paraphrase their specific guidance (see http://grants.nih.gov/grants/guide/pa-files/PAR-10-073.html), it is required that software be made freely available to non-profit

institutions, and that licensing should permit the dissemination of derivative works (possibly including a path to commercialization). Upon termination of the project, it is required that the software be transferrable to another party. There must be an ability for others to modify the source code (therefore the source code must be visible) and finally, there must be a plan to manage and disseminate improvements contributed by others.

For the last ten years, we have taken an open-source software development approach that matches the NIH mandate extremely well. The Computational Crystallography Toolbox (*CCTBX*) provides a public repository of algorithms (10) that has served as an effective foundation for end-user applications including the *PHENIX* refinement package (11), the *LABELIT* indexing suite (12), and workflow pipelines such as *XIA2* (13). For this discussion, it is important to distinguish between the end-user applications (*PHENIX* and *LABELIT*) that have been commercially licensed, and the underlying *CCTBX*, which will forever remain a freely available, open source, general-purpose toolbox. Our plan, consistent with the NIH requirements outlined above, is to develop all new data reduction code within the *CCTBX* framework.

4. *CCTBX:* A Quick Tour. As a long-term collaborative effort between loosely associated groups, the *CCTBX* has assumed the form of an assortment of tools, from which the best one can be chosen to solve a given problem. Lower-level implementations impacting almost any project include arrays and linear algebra, unit cells, space group symmetry, and crystallographic structure factors. The `iotbx.detectors` package is of specific interest for data reduction, allowing data to be input from numerous X-ray detector file formats. Object-oriented programming concepts are used throughout the Toolbox, permitting the creation of concise codes that encapsulate the details, allowing the programmer to access underlying services through an application programming interface.

For programmers interested in rapidly testing new algorithms, *CCTBX* thus offers the ability to efficiently express new ideas at a high level of abstraction. Consider the following four-line example, which appears at first to be a very simple segment of code,

```
Image = ImageFactory(filename)
Spots = Image.get_spotfinder()
Tiles = Image.get_tile_manager()
Graphics = Image.get_flex_image()
```

Here, `ImageFactory` is a function that reads any type of detector data regardless of format, and returns an `Image` object that has a uniform interface. The interface includes functions to instantiate a `Spots` object that can find and report the observed Bragg spots, a `Tiles` object that contains geometry information allowing a data reduction program to avoid attempting signal integration on inactive areas of the image, and a `Graphics` object that allows the raw data to be rendered in appropriate form within a graphical user interface. Thus, very complex applications can be written on top of simple commands.

Powerful interfaces like this are enabled by the hybrid-language *boost.python* architecture chosen for *CCTBX* (14). Core functions are written in C++, giving excellent performance to algorithms that are CPU-limited, when processed with modern compilers. Furthermore, the interoperability of C++ and C allows us to link against indispensable libraries authored

by third parties. These include CCP4's MTZ library for structure factor output, Herbert Bernstein's CBFlib for crystallographic binary format (15), and the University of Maryland Approximate Nearest Neighbor Library (16), useful for matching up predicted and observed Bragg spot positions with a fast binary-tree algorithm. Meanwhile, higher-level concepts in the Toolbox are expressed in Python, a scripting language that is ideal for rapid prototyping. New applications are generally prototyped in Python, with numerically intensive sections then ported to C++ as needed.

Python scripting gives the programmer access to numerous outside code libraries, which have proven to be of enormous help in developing code that operates in a variety of experimental contexts. For example, to analyze the Bragg spots with the above-mentioned `Spots` object on Pilatus detector data, it was necessary to employ a 32-CPU Linux processor to handle the rapid image-framing rate of 10 Hz. Good results were achieved by implementing the spotfinder as a Web page served by the Apache Web server, which transparently handles the multiprocessing by delegating each new image to a new Apache child process. The child process is configured to use `mod_python`, a downloadable module that allows us to run *CCTBX* code within the Apache server (17). Also, for analyzing the diffract-and-destroy LCLS data stream at an even higher framing rate (120 Hz), we used the multiprocessing Python-based *Pyana* framework created internally at SLAC, which delivered the images to a *CCTBX*-based function configured for data reduction. Finally, the availability of general programming libraries such as wxPython and matplotlib greatly accelerate, respectively, the development of graphical user interfaces for image viewing (see below) and the plotting of numerical data with very minimal programming effort.

Collaboration on *CCTBX* has been made possible by hosting the source code at the publicly accessible Sourceforge site (http://cctbx.sf.net), with the use of a code versioning system that allows the participants to document the purpose of each code check-in, even at the level of individual lines of code. Nightly bundles are created automatically, allowing users to obtain the latest code in binary-executable form running on Linux, Mac OSX, and Windows platforms. Furthermore the nightly build process includes an extensive set of test scripts, which validate that functions written at one time are not disabled by future code developments. Project members are expected to be diligent in writing these test cases to exercise any important feature, for it is this discipline that has allowed the project to accept contributions across continents for many years. Any failure of a test script shows up on the nightly Web page, along with a traceback identifying the point of failure, allowing immediate corrective action.

To support future data reduction efforts, we recently created a Python-based image viewer for diffraction data. Inspired by other data viewers such as *ADXV*, but amenable to subclassing to support new algorithm development, this graphical user interface (GUI) is now included in the *CCTBX* package. The original publication (18) described the `phenix.image_viewer` as being included in the *PHENIX* package, however it is now also available under the open source *CCTBX* license in either source code or binary form.

Efforts to extend data reduction methods rely heavily on viewing the measured data compared to various models, thus prompting the emphasis on developing a flexible GUI. New code currently being prototyped (`rstbx.image_viewer`) permits easy navigation

through the data using mouse click-and-drag motions to pan the image, and mouse scroll-wheel motions to zoom in and out, similar to the actions of the Web program *Google Maps*. An arbitrary number of colored overlays can be added to the image, *e.g.*, quadrilaterals and dots to represent various physical models of the diffraction, aligned with the data image to subpixel precision. A provision has been made to map the pixel coordinates of the detector onto the laboratory coordinate system, allowing us to represent pixel array detectors made of numerous silicon tiles that may have relative tilt and fractional pixel displacements. This same facility could be readily adapted to cylindrical or spherical detectors. While the mapping from detector to laboratory space consumes extra CPU cycles, the response time is kept to a minimum by rendering only the portion of the data that is currently being viewed at the present zoom level; while caching ahead the neighboring tiles to anticipate mouse-driven pans. Infrastructure for these features was derived from the pySlip project authored by Ross Wilson (http://code.google.com/p/pyslip).

4. Speculation On the Role of GPU Computing. Massively parallel computer architecture could assist data reduction in two ways: either by speeding up the workflow to keep pace with data acquisition, or by allowing the testing of more detailed models. Complexity in the diffraction pattern—beyond simple Bragg spots—was dramatically illustrated by recent low-angle XFEL work on nanocrystallites (19) where numerous fringes are observed between Bragg spots, due to the shape transform of the crystallite. One possible avenue for simulating this pattern is to use an all-atom representation of the crystallite; instead of Fourier transforming the electron density to obtain structure factors, one performs a direct sum to derive the observed amplitude at each fractional Miller index point on the detector,

$$F_{HKL} = \sum_{\substack{\text{unit cells} \\ \Delta U}} \sum_{\substack{\text{rotational symmetries} \\ R}} \sum_{\substack{\text{translational symmetries} \\ T}} \sum_{\substack{\text{atomic sites} \\ x}} f_x \cdot w \cdot e^{2\pi i H \cdot (Rx + T + \Delta U)} \cdot e^{-2\pi^2 u_{iso} d^{*2}}$$

where the compound sum spans all atoms in the crystal, and describes each atom's form factor f_x, occupancy w, position x and Debye-Waller factor. This proposed method would be intractable on a single-process CPU, but can be entertained using GPU hardware such as the Nvidia Tesla C2050, containing 448 hardware threads at fairly low cost (< U.S. $2000).

In a test implementation, we organized the problem such that each F_{HKL} is evaluated by a separate thread. In contrast to the situation with general purpose CPUs, which automatically use the on-die cache to speed up data access, the GPU interface places responsibility for data transfer directly on the programmer. A short account of our structure factor calculation thus serves to illustrate both the power and the limitations of the GPU approach. Data transferred from the CPU host to the GPU device can have two initial destinations. First, there is a small block (64 KB) of *constant memory* that is rapidly readable by the GPU threads, useful in our case for atomic form factor Gaussian coefficients and space group symmetry operators R and T. Secondly, there is ample *global memory* (3 GB) to store all the fractional atomic coordinates x, along with the output list of F_{HKL} prior to its return transfer to the host. The GPU parallelizes its work with a *single-instruction multiple-data* model, in which blocks of 32 hardware threads execute instructions in lockstep. Thread blocks have access to only a tiny amount (48 KB) of on-die *shared memory;* this poses a memory management challenge since the atomic coordinates must

ultimately be transferred on-die for the calculation. We make this efficient by having the 32-thread blocks *coalesce*: each thread reads coordinates for a single atom, thus 32 atomic coordinates are read simultaneously by synchronized threads, and each data element can be used by each of the 32 threads before it is replaced in the next data transfer cycle. Minimizing the number of global-to-shared memory transfers in this way, we are able to simulate a fringe pattern for a 10×12×14 unit cell crystallite of Photosystem I in under two minutes. The calculation is essentially 200-fold faster than the equivalent double precision work on a single-process CPU. The ability to access GPU computation within the Python framework has been added to the *CCTBX* (20).

This short description shows that algorithms must inevitably be refactored to make optimal use of the GPU's hardware resources. Due to this extra effort, it is only beneficial to focus on small sections of the problem (such as the structure factor formula) that are truly rate limiting, while performing the balance of the calculation on the CPU. Furthermore, it is critical to choose the correct programming pattern for parallelizing the algorithm. In our example, each thread was chosen to represent one structure factor, and we were able to use data transfer coalescence to efficiently *gather* all the input atomic coordinate inputs into each thread. The alternate choice, which would be unproductive, is to use threads to represent the contribution of individual atoms. In this pattern, each thread *scatters* its numerical results across all the output channels (the structure factors); however this is extremely inefficient because a global lock must be placed around the output variable each time a thread adds its contribution. While we have not yet used GPUs for routine diffraction data reduction, it is interesting to speculate how these lessons would apply to data modeling. For example, when emulating the ray-tracing approaches that have been recently discussed (21, 22) we may benefit from mapping threads to individual detector pixels, which would gather the ray tracing contributions from an ensemble of input optical rays.

4. Data Reduction for XFEL Experiments. For the photosystem II project, Python scripting within *CCTBX* greatly facilitated computational experiments to probe the many unanswered questions related to serial femtosecond crystallography data processing (23). For example, we do not know the accuracy of the crystal orientation derived from autoindexing, considering that it is calculated from a still diffraction pattern rather than a rotation series. To answer this we collaborated with Ashley Deacon at JCSG, who collected thousands of still shots from a single crystal at a synchrotron, with orientations known from goniometry. This work (in the summer of 2011) gave an early glimpse of the challenges of XFEL data reduction, and provided an opportunity to test new algorithms. A new *CCTBX*-based integration program was developed with rudimentary code to implement simple 2-dimensional pixel summation of Bragg spots. Integration masks were chosen empirically from nearby bright spots, and same-frame background pixels were selected to avoid the signal pixels. No attempt was made to deconvolute overlapping spots; the small percentage of overlapping signals was simply discarded. This new processing suite (dubbed *cctbx.xfel*) also addressed other specialized problems related to XFEL data collection. Detector metrology was a concern, since the Cornell/SLAC pixel array instrument consists of 32 separate silicon sensor chips (3). We were able to calibrate their relative spatial positions with subpixel accuracy using test diffraction data from

thermolysin microcrystals. Also, a result of our photosystem II experiment was that a large percentage of serial femtosecond diffraction patterns were exposed with multiple crystallites in the beam. We were able to autoindex these multi-lattice patterns by indexing one lattice at a time and removing the corresponding Bragg spots for the next round of indexing, as we did recently for synchrotron-based data (5). The SFX-based dark-state photosystem II structure from Sept. and Dec. 2011 data was recently published (2).

5. Application to Synchrotron Datasets. Realizing that the *cctbx.xfel* suite might serve as a starting point for new software development for synchrotron data, my NIH-funded group began collaborating with the BioStruct-X funded group led by Gwyndaf Evans at Diamond. Addition of rocking curve and Lorentz-polarization treatments (24) allowed us to apply the software to a fine-phi sliced synchrotron dataset collected on a pixel array detector. As an initial proof of principle, we demonstrated that integrated structure factor intensities from our program's 2D summation agree extremely well (c.c.=99.5%) with those obtained from 3D profile fitting using an established data reduction program, *XDS* (25). This encourages us to continue the collaborative effort to produce a fast, flexible platform for future methods.

References

1. Yano J, *et al.* (2005) X-ray damage to the Mn4Ca complex in single crystals of photosystem II: a case study for metalloprotein crystallography. *Proceedings of the National Academy of Sciences of the United States of America* 102(34):12047-12052.
2. Kern J, *et al.* (2012) Room temperature femtosecond X-ray diffraction of photosystem II microcrystals. *Proceedings of the National Academy of Sciences of the United States of America* 109(25):9721-9726.
3. Philipp HT, Koerner LJ, Hromalik MS, Tate MW, & Gruner SM (2010) Femtosecond Radiation Experiment Detector for X-ray Free-Electron Laser (XFEL) Coherent X-Ray Imaging. *IEEE Transactions on Nuclear Science* 57(6):3795-3799.
4. Eikenberry EF, *et al.* (2003) PILATUS: a two-dimensional X-ray detector for macromolecular crystallography. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 501(1):260-266.
5. Sauter NK & Poon BK (2010) Autoindexing with outlier rejection and identification of superimposed lattices. *J Appl Crystallogr* 43(Pt 3):611-616.
6. Lovelace JJ, *et al.* (2008) Protein crystals can be incommensurately modulated. *Journal of Applied Crystallography* 41(3):600-605.
7. Porta J, Lovelace JJ, Schreurs AM, Kroon-Batenburg LM, & Borgstahl GE (2011) Processing incommensurately modulated protein diffraction data with Eval15. *Acta crystallographica. Section D, Biological crystallography* 67(Pt 7):628-638.
8. Tsai Y, Sawaya MR, & Yeates TO (2009) Analysis of lattice-translocation disorder in the layered hexagonal structure of carboxysome shell protein CsoS1C. *Acta crystallographica. Section D, Biological crystallography* 65(Pt 9):980-988.
9. Pletnev S, Morozova KS, Verkhusha VV, & Dauter Z (2009) Rotational order-disorder structure of fluorescent protein FP480. *Acta crystallographica. Section D, Biological crystallography* 65(Pt 9):906-912.

10. Grosse-Kunstleve RW, Sauter NK, Moriarty NW, & Adams PD (2002) The *Computational Crystallography Toolbox*: crystallographic algorithms in a reusable software framework. *Journal of Applied Crystallography* 35:126-136.
11. Adams PD*, et al.* (2010) PHENIX: a comprehensive Python-based system for macromolecular structure solution. *Acta crystallographica. Section D, Biological crystallography* 66(Pt 2):213-221.
12. Sauter NK, Grosse-Kunstleve RW, & Adams PD (2004) Robust indexing for automatic data collection. *Journal of Applied Crystallography* 37:399-409.
13. Winter G (2009) xia2: an expert system for macromolecular crystallography data reduction. *Journal of Applied Crystallography* 43(1):186-190.
14. Abrahams D & Grosse-Kunstleve RW (2003) Building Hybrid Systems with Boost.Python. *C/C++ Users Journal* 21(7):29-36.
15. Bernstein HJ & Ellis PJ (2005) *International Tables for Crystallography, Vol. G*, eds Hall SR & McMahon B (Springer, Heidelberg), Vol G.
16. Arya S, Mount DM, Netanyahu NS, Silverman R, & Wu AY (1998) An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *J. Assoc. Comput. Mach.* 45:891-923.
17. Sauter NK (2011) An extremely fast spotfinder for real-time beamline applications. *Computational Crystallography Newsletter* 2:93.
18. Echols N, Hattne J, Gildea RJ, Adams PD, & Sauter NK (2012) Viewing diffraction images in CCTBX. *Computational Crystallography Newsletter* 3:14-17.
19. Chapman HN*, et al.* (2011) Femtosecond X-ray protein nanocrystallography. *Nature* 470(7332):73-77.
20. Poon BK, Echols N, Grosse-Kunstleve RW, Sauter NK, & Zwart PH (2012) The Need for Speed: Integrating CUDA into the CCTBX Framework. *J Appl Cryst* submitted.
21. Diederichs K (2009) Simulation of X-ray frames from macromolecular crystals using a ray-tracing approach. *Acta crystallographica. Section D, Biological crystallography* 65(Pt 6):535-542.
22. Schreurs AMM, Xian X, & Kroon-Batenburg LMJ (2009) EVAL15: a diffraction data integration method based onab initiopredicted profiles. *Journal of Applied Crystallography* 43(1):70-82.
23. Kirian RA*, et al.* (2010) Femtosecond protein nanocrystallgraphy--data analysis methods. *Optics Express* 18(6):5713-5723.
24. Greenhough TJ & Helliwell JR (1982) Oscillation Camera Data Processing: Reflecting Range and Prediction of Partiality. 2. Monochromatized Synchrotron X-radiation from Singly Bent Triangular Monochromator. *Journal of Applied Crystallography* 15:493-508.
25. Kabsch W (2010) Xds. *Acta crystallographica. Section D, Biological crystallography* 66(Pt 2):125-132.