Accelerated Synchrotron X-ray Diffraction Data Analysis on a Heterogeneous High Performance Computing System

J. Qin and M. A. Bauer

Computer Science Department, The University of Western Ontario, London, ON. N6A 5B7 Canada

Email: qin.jinhui@gmail.com, bauer@uwo.ca

Abstract. The analysis of synchrotron X-ray Diffraction (XRD) data has been used by scientists and engineers to understand and predict properties of materials. However, the large volume of XRD image data and the intensive computations involved in the data analysis makes it hard for researchers to quickly reach any conclusions about the images from an experiment when using conventional XRD data analysis software. Synchrotron time is valuable and delays in XRD data analysis can impact decisions about subsequent experiments or about materials that they are investigating. In order to improve the data analysis performance, ideally to achieve near real time data analysis during an XRD experiment, we designed and implemented software for accelerated XRD data analysis. The software has been developed for a heterogeneous high performance computing (HPC) system, comprised of IBM PowerXCell 8i processors and Intel quad-core Xeon processors. This paper describes the software and reports on the improved performance. The results indicate that it is possible for XRD data to be analyzed at the rate it is being produced.

1. Overview of XRD data analysis

In order to understand and predict properties of materials, detailed information about the structure of the material at the atomic-scale level is needed. Synchrotrons can be used to generate X-rays in order to probe materials at the atomic level. One approach is to use X-ray diffraction (XRD) to do this. The Laue X-ray diffraction (XRD) technique has been used to determine the atomic-scale structures of materials based on the Bragg’s Law [1]. When a material that consists of a periodic atomic structure, such as a crystal, is irradiated with X-rays, it produces a diffraction pattern showing numerous sharp spots, called Bragg diffraction peaks. By measuring and analyzing the positions and intensities of these peaks, one can determine the arrangement of atoms in the crystalline material. Not only can Laue diffraction provide geometrical information about the crystalline structure, it can also be extended to measure the intensity of X-ray diffraction to determine the strain tensor in sample materials [2]. The strain tensor information will then help scientists/engineers to determine possible flaws in materials.

During an XRD experiment, a material sample is scanned by X-rays at a number of points over an area of the sample. A CCD detector captures the diffractions and produces a 2D image for each scan point (see Figure 1 for an example of an XRD image). Image sizes are dependent on the CCD detector. For an area XRD scan, a typical step size for moving the scan point by point is a couple of microns. The data generated in XRD experiments can consist of a large number of images. A number of existing software packages for XRD data analysis have been developed to analyze these images. These include the 3D X-ray Micro-diffraction Analysis Software Package [3] at the Advanced Photon Source (APS), which was developed by scientists at the Oak Ridge National Laboratory (ORNL), and
the X-ray Micro-diffraction Analysis Software (XMAS) [4] at the Advanced Light Source (ALS) in Lawrence Berkeley National Laboratory (LBNL). The common feature of these two packages is that they both are Windows-based software, implemented in the Interactive Data Language (IDL) [5]. Both can process a large amount of XRD data sequentially.

Analysis of XRD data analysis typically consists of applying three computational methods on each of the XRD images sequentially. The three procedures are peak searching, indexing and strain analysis. The peak searching process attempts to extract useful information about intensity points (peaks) from an image to be used as input for the next two procedures. The indexing procedure takes the output from the peak searching procedure and generates the structure information about the sample material, e.g. the orientation of a crystalline lattice plane from which a diffraction spot (i.e. a peak that appeared in the XRD image) is generated. Based on the indexing results and peak information, the strain analysis procedure then produces strain tensor information of the sample. Based on the indexing results and strain tensor information, an orientation map and a strain map can be generated for the entire scanned area from which all XRD data were collected. Those maps become the most interesting part for scientists and engineers.

The main purpose of peak searching procedure is to identify the locations and intensities of those peaks captured on an XRD image. The accuracy of peak search results become critical to the quality of indexing and strain analysis. However, images generated from X-ray diffraction are often noisy, and peaks shown within an image are often “blobs” of different sizes and shapes. A number of factors, often referred to as the Lorentz factors [6], affect the quality of images from X-ray diffraction, including scattering, angular-velocity factor, polarization, etc. These make the peak searching procedure complicated.

Figure 1 An example of an XRD image
In order to produce more precise peak information for indexing and strain analysis, the peak searching procedure can be split into three subtasks: a threshold finding process, which is used to remove background noise; a “blob” searching process, which searches for all the blobs with intensities above the threshold; and a blob fitting process, which applies some 2D curve fitting functions on each blob area to determine a reasonable location and intensity of a peak.

Basically, the source data from an XRD experiment is a sequence of digital image files. The XRD data analysis software actually deals with a large amount of data, typically, digital images in the hundreds or even thousands. Existing analysis software processes these images individually and sequentially. This is a very time consuming process, normally requiring days to complete the processing of an entire set of data from one XRD experiment. As noted, synchrotrons are typically used to generate X-rays to probe the structure of materials. Synchrotron time is valuable and it is often difficult to get a scheduled beam time. Data analysis using existing software means that researchers are typically doing the analysis after their time on the synchrotron. More timely analysis could help researchers make decisions on subsequent experiments and gain significant insight into the materials that they are studying. Performance improvement in the analysis software could make the analysis more useful.

In this paper, we report on the development of parallel XRD data analysis software which has been developed on a heterogeneous HPC system comprised of Intel and IBM Cell processors. The software developed and the system it runs on makes it possible for XRD data to be processed in “near-real time”, that is, nearly as fast as it is being produced. A description of the target HPC system is provided in Section 2. The development approach is presented in Section 3, including the challenges and an overview of the design and implementation of the software. Performance evaluations are reported in Section 4, and Section 5 provides some conclusions and future work. This work was done as part of the Science Studio project [7].

2. Target HPC system

Our target system, called prickly, is one of the clusters in SHARCNET [9]. Prickly is located at the University of Western Ontario (UWO). It is a heterogeneous HPC system consisting of one head node for hosting user logins and a chassis with 12 Linux cluster blades providing total 160 computing cores. Among the 12 blades, four blades are Intel blades and the other eight are IBM QS22 Cell blades. On each of the Intel blades, there are two quad-core Xeon E5420 processors running at 2.5GHz with 8GB of memory. Each of the IBM Cell blades contains two PowerXCell 8i processors, so called Cell processors, running at 3.2GHz with 16GB of memory. Gigabit Ethernet provides the inter-connection between any two blades on prickly, including Cell blades and Intel blades.

Unlike traditional multi-core processors which are homogenous, such as those on Intel blades, the Cell processor itself has heterogenous multi-cores [10]. It employs two types of cores optimized for different kind of tasks. Each Cell processor has nine cores, i.e. one PowerPC Processor Element (PPE) and eight Synergistic Processor Elements (SPEs). The PPE is just a traditional 64-bit Power processor with 512K cache and acts as a large-scale processor core to run the operating system and performs control-intensive tasks. In contrast, the SPEs are much simpler, but devote more resources to perform computationally intensive tasks. Since each Cell blade has two Cell processors, in total, there are sixteen SPEs on each Cell blade. The sixteen SPEs are independent, 128-bit vector processors, which can execute Single Instruction Multiple Data (SIMD) instructions. Each SPE has its own local storage (256KB) for instructions and data. The local storage can be considered as the cache for each SPE, except that users’ programs need to decide and manage what should be taken from main memory and stored locally during the execution. The SPE access to the memory is achieved through its Direct Memory Access Controller (DMAC). The DMAC can work concurrently with SPE executions, which hides the latency caused by memory accesses. However, user programs need to decide how to overlap

---

1 The Science Studio project has been funded by CANARIE [8].
the data movement between the main memory and a local store in order to make the computations done on an SPE as efficient as possible.

The Element Interconnect Bus (EIB) provides four 128-bit data transmission channels for the inter-communication among PPE, SPEs, main memory and I/O. The EIB can support up to 307GB/s bandwidth between any two bus units. Therefore, with EIB, each SPE can not only work alone, but also be chained together to perform data processing with an intensive workload, such as stream processing.

The PPE is intended to run the operating system and coordinate computations. A key performance advantage comes from Cell’s sixteen de-coupled SPE SIMD engines with dedicated resources for computations, including large register files and DMA channels. By incorporating advanced multiprocessing technologies, the Cell processor is especially suitable for high performance workloads. While the Cell’s special architecture offers many advantages for high performance computations, the architecture also makes programming on Cell a non-trivial task.

3. Application development approach
We had two main objectives in porting the XRD data analysis application onto the target HPC system, i.e. prickly. First, we wanted to create an implementation of the three major analysis procedures where the processing tasks were pipelined in order to accelerate the processing of an XRD image. Second, we also wanted an implementation so that multiple XRD images could also be processed in parallel.

3.1. Challenges
There were two major challenges involved in this porting process. First, the original software was written in IDL [5]. IDL is a proprietary scripting language provided with extensive subroutine libraries for processing/visualizing images in the fields of physics, astronomy and medical imaging. Our target system ran Linux and could only compile and execute programs in C/C++/Fortran. The software needed to be rewritten in C in order to be able to adapt it for execution on Cell processors. The source code of the XRD analysis software was originally developed at ORNL and made use of many of the library functions within the IDL package. With neither the source code of the library functions nor detailed computation algorithms for XRD data analysis procedures, rewriting the software in C became a challenging task.

Another challenge was programming on Cell itself. Because of the Cell’s heterogeneous architecture, the PPE and SPE execute different instruction sets. Programs written for the PPE and SPEs need to be compiled by different compilers. The PPE manages SPE processes as POSIX threads and so threaded programming skill is required. As each SPE has its own local store for holding instructions and input/output data, the data needs to be moved back and forth between the local store and the main memory with explicit DMA commands. Moreover, in order to hide the memory access latency in order to make efficient programs, programmers must handle the local store management, such as data locality, overlapping data transformation and inter-unit communication within SPE’s computations. Because of the limited space (256K) for a local store on an SPE, only tasks that can fit within that local store are good candidates to be considered for a SPE, otherwise, advanced overlay management needs to be considered [10].

3.2. Redesign of the software
In order to attain an accelerated XRD data analysis on the target HPC system, the software needed to be redesigned to be parallel and take an advantage of the Cell’s multi-core processing. Considering the challenges of programming on Cell, it was more efficient to first identify the performance “bottleneck” of the entire process and then target an implementation around that “critical” part. As a result, instead of simply porting all three procedures onto Cell, we focused on the critical part first.

As introduced in Section 1, there are three major procedures involved in XRD data analysis. Our measurements indicated that more than 80% of the processing time was spent in the peak searching
procedure; therefore, it was initially targeted as the “critical” computation to be considered for porting to the Cell.

The peak searching procedure involves finding a threshold, blob searching, and curve fittings on each of the blobs. Among all three subtasks in peak searching, curve fitting is the most intensive one. During curve fitting on each blob, it applies two 1-D fittings (i.e., one for X direction and one for Y direction) and one 2-D fitting for a box area around each blob. Figure 2 illustrates blobs identified in an XRD image with a certain intensity threshold. Each fitting process actually entails solving a multi-variable, non-linear least square minimization problem. It involves iterations to update the state of corresponding variables continually until a certain criteria are met. Specifically, the 1-D fitting involves solving four variables; these results become the initial states for the 2-D fitting. In turn, the 2-D fitting involves solving for six variables. The existing software carries out the curve fittings sequentially for each of the blobs in an image, which is very time consuming and becomes the bottleneck of the entire XRD data analysis.

Figure 2. An XRD image with identified blobs that need curve fittings

Considering the computational power of a Cell’s SPE, with a limited local store, it works well for a process with relatively small size but needs to run many times. Fortunately, the curve fitting is applied to each blob, which is in a relatively smaller area than the entire image area. The computation of the fitting process is also relatively intense and needs to be applied to every blob in an image. Therefore, the curve fitting process was selected as the processing task for the Cell’s SPEs. Then, once a collection of blobs has been identified, the fitting process can be done on the Cell’s multiple SPEs in parallel, i.e. multiple blobs can be fitted by multiple SPEs simultaneously.

Even though the size of the data accessed during each curve fitting process is relatively small, it still took some effort to implement such a task on Cell’s SPE because of its limited local store. As has been mentioned, the curve fitting process involves actually solving a non-linear, multi-directional least square fitting problem. The GNU Scientific Library (GSL) [11] provides a minimization package in C
for solving such problems. However, the size of GSL, or even only the size of the minimization package, is too large to fit into a SPE’s local store. In order to fit the desired code on the SPE, we manually extracted each of functions needed for the minimization computation and then included them as part of the source code to be compiled on Cell’s SPEs. Combined with the two other sub-tasks i.e., threshold discovery and blob identification, on Cell’s PPE, after the curve fitting for blobs had been successfully ported onto the Cell’s SPEs, the peak searching procedure could then be carried out entirely on a Cell processor.

To analyze a large set of XRD images, and in order to achieve a better throughput, there are two key challenges that must be overcome. First, we need to find out what needs to be run only once and which result can be shared by all the other images from the same set of experiments, i.e., to identify those computations that do not need to be applied to every image. For example, the background information of images from the same set of experiments can be very similar because they all have the same experimental settings. Therefore, the threshold finding process only needs to be applied to one of the images, and the threshold can be passed to and shared by all the other images.

Second, for a parallel application on such a heterogeneous HPC system, it is a multi-threaded and multi-process program. Therefore, we need to consider the overhead in generating such two kinds of parallel computational elements. For example, in order to have $n$ images processed in parallel requires $n$ parallel processes. In processing each of the images, doing curve fitting to $m$ blobs within each image in parallel requires $m$ parallel threads. For a given set of computation resources, the execution time for processing a large set of images varies when the configurations are different, i.e. the settings of $m$ and $n$. Efficient processing depends on the computational workload and the difference in overhead between multi-thread and multi-process. We explore this difference in settings in the experiments presented in Section 4.

3.3. Parallel XRD data analysis
The three major procedures involved in XRD data analysis have been designed to execute on different types of nodes on prickly, a blade center with two different types of blades. The peak searching procedure has been ported to Cell nodes, and the indexing and strain analysis procedures have been ported to the Intel Xeon nodes.

In order to achieve high performance in processing a large number of XRD images, we consider a parallel computational approach involving processing multiple images in parallel and fitting multiple blobs in parallel for each image. Since curve fitting is the most intensive task it has been assigned to the Cell’s SPEs and threshold finding and blob searching are kept as tasks for Cell’s PPEs. It is assumed that the intensity levels or background information are similar among images collected from the same XRD experiment and therefore the threshold finding process only needed to be performed once on one of the images, e.g. normally the first image. The rest of the images can then share this same threshold during the rest of the analysis. The design of such a parallel XRD data analysis program on our target system is illustrated in Figure 3.

Using Figure 3 as a guide, the processing proceeds as follows. Initially, $n$ images are loaded to be processed in parallel. Each of these images is initially processed on one of the PPEs in order to identify blobs; a list of blobs is produced. Curve fitting is then done on each list of blobs on an SPE. The processing of the blobs from an image results in a list of possible peaks (PeakList in the Figure 3). The list of peaks is then passed to the indexing computation which results in a grain orientation map and the index data. The index data is then used in the strain computation which produces a strain map.

Because of relatively less computational workload when compared with the peak searching procedure, the indexing procedure and strain analysis procedure are currently kept as sequential processes for each image and done on Intel blades. The size of each of these processes is too large to easily fit into the limited local storage on Cell’s SPE as they stand. Moreover, the Cell’s PPE is structured specifically for handling control intensive tasks instead of computational tasks [10], therefore, we decide to allocate the process of indexing and strain analysis to the Intel blades instead of the Cell blades on prickly.
The communication and data transformation between any two nodes on *prickly* is achieved through MPI [12]. MPI is a language independent protocol. It is commonly used in computer clusters and supercomputers to support point-to-point and collective communications. For the computational elements allocated to the Cells’ SPEs, management is achieved through Linux POSIX Threads (Pthread). All the nodes share the same file system.

In summary, the XRD data analysis developed is a mixed multi-threaded and multi-process application which can achieve an accelerated processing performance as we shall describe in the next section.

![Diagram](image)

**Figure 3.** The configuration of high performance XRD data analysis on Prickly

### 4. Performance evaluation

The design in Figure 3 has been implemented and deployed on *prickly*. In order to evaluate the processing performance and study the impact of various parallel settings on the processing, we have carried out a number of experiments considering images of different sizes and considering different numbers of processing elements on *prickly*. Since XRD images could have different sizes depending on the CCD detector that had been used for collecting the images, we considered two different sizes of XRD images. The XRD images were collected from experiments from the Advanced Photon Source (APS) [13] and others were from experiments done at the Canadian Light Source (CLS) [14]. An XRD image from APS has 1042x1042 pixels and is about 2MB/image; while an image from CLS has 2084x2084 pixels and is about 8MB/image. That means the workload for processing these two types of images is different.
4.1. XRD analysis on a desktop machine

We measured the time needed to process a set of images sequentially with the original IDL software on a desktop machine, i.e., each processing step was done sequentially and one image after the other. We compared the performance of using the original IDL software and using the non-parallelized C version of the software on the same desktop machine. The desktop machine used was an Intel dual core 2.4GHz processor with 3GB memory. The average processing speed was measured in seconds/image and is presented in Table 1. As expected, the software in C is faster than the original IDL version. The speedup is relatively modest for both the APS images and the CLS images, respectively, 1.47 and 1.25.

| Image size (pixels/image) | Total number of images | Total processing time (sec.) | Average speed (sec./image) | Speedup (C vs. IDL) |
|---------------------------|------------------------|-----------------------------|---------------------------|---------------------|
| 1042X1042 (APS)           | 64                     | 276.0                       | 4.31                      | 1.47                |
| 2084X2084 (CLS)           | 64                     | 919.0                       | 14.36                     | 1.25                |

4.2. XRD analysis on prickly

The C version of the software was then ported to prickly, as illustrated in Figure 3, where the three procedures have been chained together to work in a pipeline. Multiple images can then be processed in parallel simultaneously, which means that multiple pipelines can be created for XRD image analysis. Since the system has total of 4 Intel blades and 8 Cell blades, we examined the performance of XRD data analysis on one pair of Cell-Intel blades and on multiple pairs of Cell-Intel blades.

In using one of the Cell-Intel nodes on prickly, i.e. 1 Cell blade and 1 Intel blade, as described in Figure 3, peak searching is done on the Cell blade; while indexing and strain analysis are done on the Intel blade. Since each Cell blade has total 16 SPEs, if \( n \) images are processed in parallel, i.e. \( n \) pipelines, and \( m \) SPEs are allocated for each image or each pipeline, \( m \) and \( n \) are constrained to be values such that \( m \times n = 16 \). Different settings for \( m \) and \( n \) were tested. For the workload on the Intel blade, if \( n \) pipelines are initiated for processing \( n \) images in parallel, \( n \) processes are created on the Intel blade, and each of these \( n \) processes works on the indexing and strain analysis on one of \( n \) images. The operating system takes care of workload distribution among the eight cores on the Intel blade. We measured the average processing speed of various settings. The speedup of each test case is compared to the desktop speed using IDL and C software. Tables 2 and 3 present the measured results on one pair of computation nodes on prickly. Table 2 reports the results for the APS images and Table 3 reports the results for the CLS images.

| Total number of images | Number of images in parallel (number of pipelines) | Number of SPEs for each image | Total processing time (seconds) | Average speed (sec./image) | Speedup vs. IDL (speed 4.31 sec./image) | Speedup vs. C (speed 2.94 sec./image) |
|------------------------|----------------------------------------------------|-------------------------------|---------------------------------|----------------------------|------------------------------------------|---------------------------------------|
| 64                     | 1                                                  | 16                            | 40.54                           | 0.63                       | 6.84                                    | 4.67                                  |
| 64                     | 2                                                  | 8                             | 27.42                           | 0.43                       | 10.02                                   | 6.84                                  |
| 64                     | 4                                                  | 4                             | 22.64                           | 0.35                       | 12.31                                   | 8.4                                   |
| 64                     | 8                                                  | 2                             | 16.41                           | 0.26                       | 16.58                                   | 11.31                                 |
| 64                     | 16                                                 | 1                             | 14.05                           | 0.22                       | 19.59                                   | 13.36                                 |
The results presented in Table 2 illustrate that for processing images of the size of the APS images, the more images that are processed in parallel, the better throughput, i.e. processing 16 images resulted in speedups of 19.59 compared to IDL on the desktop and 13.36 compared to C on the desktop.

In processing the larger size images, i.e. those from CLS, the results of Table 3 suggest that processing 4 images in parallel and with 4 SPEs allocated for each image can produce the best throughput. The processing of multiple images in parallel was achieved through multi-process programming, while the parallel blob fitting on the Cell was achieved through multi-threaded programming. In general, process creation cost is much larger than thread creation cost. The overall performance gain of a parallel application is dependent on balancing the computational workload and the trade-off in setting up the parallel processing elements. The result of reducing the number of SPEs allocated for each image, i.e., to 2 or 1 in Table 3, in lieu of having more parallel pipelines to process more images in parallel is not sufficient to overcome the blob processing done on each of the larger images within the SPEs. Consequently, using 4 pipelines and 4 SPEs results in the best performance for images of this size.

| Total number of images | Number of images in parallel (number of pipelines) | Number of SPEs for each image | Total processing time (seconds) | Average speed (sec./image) | Speedup vs. IDL (speed 14.36 sec./image) | Speedup vs. C (speed 11.49 sec./image) |
|------------------------|--------------------------------------------------|-----------------------------|-------------------------------|---------------------------|--------------------------------------|--------------------------------------|
| 64                     | 1                                                | 16                          | 182.07                        | 2.84                      | 5.06                                 | 4.05                                 |
| 64                     | 2                                                | 8                           | 115.95                        | 1.81                      | 7.93                                 | 6.35                                 |
| 64                     | 4                                                | 4                           | 98.27                         | 1.55                      | 9.26                                 | 7.41                                 |
| 64                     | 8                                                | 2                           | 107.12                        | 1.67                      | 8.60                                 | 6.88                                 |
| 64                     | 16                                               | 1                           | 107.50                        | 1.68                      | 8.55                                 | 6.84                                 |

It is also possible to use multiple Intel and Cell blades to process a sequence of images. To examine the performance implications of this, we also ran a number of experiments of XRD data analysis using multiple pairs of Cell-Intel nodes. Based on the results presented in Tables 2 and 3, 1 SPE per APS image and 4 SPEs per CLS image were used. Tables 4 and 5 present the results.

| Total number of images | Pairs of Cell-Intel node(s) | Number of images in parallel (number of pipelines) | Total processing time (sec.) | Average speed (sec./image) | Speedup vs. IDL (speed 4.31 sec./image) | Speedup vs. C (speed 2.94 sec./image) |
|-----------------------|-----------------------------|--------------------------------------------------|-------------------------------|---------------------------|--------------------------------------|--------------------------------------|
| 64                    | 1                           | 16                                               | 14.05                         | 0.22                      | 19.59                                | 13.36                                |
| 64                    | 2                           | 32                                               | 9.20                          | 0.14                      | 30.78                                | 21.00                                |
| 128                   | 3                           | 48                                               | 9.58                          | 0.07                      | 61.57                                | 42.00                                |
| 128                   | 4                           | 64                                               | 9.28                          | 0.07                      | 61.57                                | 42.00                                |

The results presented in Tables 4 and 5 illustrate that the performance of XRD data analysis has been boosted significantly when more computational resources are used. For the smaller sized XRD images collected at APS, as presented in Table 4, when 48 processing pipelines were setup on three pairs of Cell-Intel blades, the average processing speed can reach as high as 0.07 sec./image, which is 61.57 times speedup compared with 4.31 sec./image of using existing IDL software on a desktop.
machine. For larger size XRD images collected at CLS, when 16 processing pipelines were setup on four pairs of Cell-Intel blades, the average processing speed (see Table 5) can reach as high as 0.59 sec./image, which is 24.34 times speedup compared with 14.36 seconds/image when using the IDL software on a desktop machine.

Notably, the measured speedup from both sets of experiments presented in Tables 4 and 5 do not result in a linear improvement as more nodes are added. One of the factors affecting the speedup is the increased overhead of MPI in setting up more pipelines exchanging information across the two types of nodes on prickly. Another factor that affects achieving a linear speedup is the data transfer and communication cost of the Gigabit Ethernet; as more processes are added there is an increase in communication. A more thorough investigation of exactly which factors play the most prominent role in limiting the speedup when processors are added could be one of the research tasks in future.

Table 5. Results of processing CLS images on multiple pairs of Cell-Intel nodes on prickly

| Total number of images | Pairs of Cell-Intel node(s) | Number of images in parallel (number of pipelines) | Total processing time (sec.) | Average speed (sec./image) | Speedup vs. IDL (speed 14.36 sec./image) | Speedup vs. C (speed 11.49 sec./image) |
|------------------------|-----------------------------|---------------------------------------------------|-----------------------------|---------------------------|-----------------------------------------|----------------------------------------|
| 64                     | 1                           | 4                                                 | 98.27                       | 1.55                      | 9.26                                    | 7.41                                   |
| 64                     | 2                           | 8                                                 | 61.45                       | 0.96                      | 14.96                                   | 11.97                                  |
| 64                     | 3                           | 12                                                | 44.71                       | 0.70                      | 20.51                                   | 16.41                                  |
| 64                     | 4                           | 16                                                | 37.80                       | 0.59                      | 24.34                                   | 19.47                                  |

5. Conclusion and future work
In this paper we reported on the development of accelerated XRD data analysis software running on a heterogeneous HPC system, i.e. the cluster prickly on SHARCNET. Using the computational power of prickly, especially the Cell processors, the accelerated XRD data analysis can achieve a speedup of up to 60 times faster than a desktop performance of the original IDL software package, depending on the size of images and the number of computation nodes used on prickly. The design and implementation has not yet been optimized to fully use the computing resources provided by prickly and so there is likely still room to further improve the computational performance if there is such a need in the future. Other designs using the computing resources of prickly could also be examined as alternative schemes for computation. For example, it would be interesting to compare and examine the performance of a parallel implementation without using any of the Cell blades and only using the quad-core Xeon processors on the Intel blades.

The current implementation has achieved the functionality for a high performance data analysis of XRD data. Our next step is to implement the process of data transmission followed by our analysis in near real time, that is, as it is collected during a synchrotron experiment. This would result in a near real time data collection and analysis process. This is currently underway using the VESPERS beamline [15] at the Canadian Light Source (CLS), combined with CANARIE [8]’s dedicated lightpath for data transmission from CLS to UWO. The expected performance will allow researchers to examine initial results from beamline experiments, make decisions regarding the experiment (e.g. stop it and reposition the beam if the scan seems unproductive), and to make decisions regarding subsequent experiments. Such a service model could also be adapted to other synchrotron and HPC facilities.

References
[1]. M. M. Woolfson, An Introduction to X-Ray crystallography, 2nd edition, Cambridge University Press, 1997.
[2]. J. Chuang and G. Ice, “Automated indexing for texture and strain measurement with broad-band pass x-ray micro-beams”, *Journal of Applied Physics*, No. 9, Vol. 86, 1999

[3]. XRD data analysis at APS: [http://www.aps.anl.gov/Sectors/33_34/microdiff/downloads/](http://www.aps.anl.gov/Sectors/33_34/microdiff/downloads/)

[4]. XRD data analysis at ALS: [http://xraysweb.lbl.gov/microdif/user_resources.htm](http://xraysweb.lbl.gov/microdif/user_resources.htm)

[5]. K. P. Bowman, *An Introduction to Programming with IDL: Interactive Data Language*, Elsevier Inc. 2006

[6]. A.J.C. Wilson, *Elements of X-Ray Crystallography*, Addison-Wesley Pub. Co. ,1970

[7]. Science Studio website: [http://sciencestudioproject.com/](http://sciencestudioproject.com/)

[8]. CANARIE website: [http://www.canarie.ca/](http://www.canarie.ca/)

[9]. SHARCNET website: [https://www.sharcnet.ca/](https://www.sharcnet.ca/)

[10]. M. Scarpino, *Programming the Cell Processor: For Games, Graphics, and Computation*, Printice Hall, 2008

[11]. M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth and F. Rossi, *GNU Scientific Library Reference Manual - Third Edition (v1.12)*, Network Theory Ltd. 2009

[12]. P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, Inc. 1997

[13]. APS website: [http://www.aps.anl.gov/About/ALD/](http://www.aps.anl.gov/About/ALD/)

[14]. CLS website: [http://www.lightsource.ca/index.php](http://www.lightsource.ca/index.php)

[15]. VESPER website: [http://vespersbeamline.com/index.php](http://vespersbeamline.com/index.php)