Real-Time MPI-Based Software for Processing of XPCS Data

Timothy Madden, *Member, IEEE,* Sufeng Niu, Suresh Narayanan, Alec Sandy, John Weizeorick, *Member, IEEE,* Peter Denes, John Joseph, *Member, IEEE,* Victoria Moeller-Chan, Dionisio Doering, *Member, IEEE,* and Patrick McVittie

***Abstract*—We describe a software library, called MPIFCCD, based on the Message Passing Interface (MPI) for real-time parallel computing on data continuously streamed from the Frame Store Fast Charge-Coupled Device (FSFCCD) Detector located at the Advanced Photon Source (APS) at Argonne National Laboratory. The FSFCCD is used to collect data for X-ray Photon Correlation Spectroscopy (XPCS) experiments at Sector 8-ID at APS. MPIFCCD is integrated into another software package called CINController, developed at APS and Lawrence Berkeley National Laboratory to serve as a QT-based user interface for control and data collection from the FSFCCD. Real-time calculations performed by MPIFCCD include dark image integration and subtraction, noise image integration, image descrambling, and lower-level discrimination. MPIFCCD allows for continuous real-time data collection of FSFCCD data at image rates of 100 frames- per-second (fps) for 1 mega-pixel images and 1000fps for 10 kilo-pixel images. In the future, more complex computations will be implemented in real time with MPIFCCD.**

***Index Terms*—CCD, MPI, real-time, XPCS.**

I. Introduction

X-RAY Photon Correlation Spectroscopy (XPCS) is an important experimental technique at synchrotrons such as the Advanced Photon Source (APS) at Argonne National Laboratory. In this technique, x-rays with energy at around $E = 7keV$ illuminate and scatter off of an experimental sample, creating a speckle pattern of scattered x-rays that is captured with a high speed x-ray sensitive area detector. Typically, the sample is a glassy system exhibiting dynamic material characteristics that must be measured over time. Time auto-correlation of collected speckle patterns reveals the time-varying structural changes in the sample. Because it is desirable to capture rapidly changing material dynamics, the x-ray camera must be run at a high frame rate from 60 frames-per-second (fps) to 1000fps. Furthermore, because long-term dynamics spanning several decades of time must be recorded, the camera must be run for several minutes, potentially creating extremely large data sets [1]. A diagram of the experiment setup is shown in Figure 1.

At Sector 8 at the APS the Fast Charge-Coupled Device Detector Two (FCCD2) is used for collecting speckle patterns at 100fps for $960x960$ pixel images and at 1000fps for $960x90$ pixel images [2] [3]. The FCCD2 operates on the principle of direct x-ray detection, in which x-rays impinge directly upon the detector sensor, a deeply depleted silicon CCD, and create charge that can be detected by the camera electronics. The direct-detection FCCD has a large signal-to noise ratio because each 7keV x-ray photon creates 7000/3.65=1917 electrons, which is much larger than the electronic noise in the detector. The speckle pattern images can be compressed by taking advantage of this large signal-to-noise ratio by using a lower level discriminator (LLD) to determine if a pixel has x-ray photons present. Any pixel devoid of detected x-ray energy can be thrown away, thus compressing the speckle images. Because the speckle patterns tend to be very sparse, an 80% compression ratio can be achieved. The following pages describe the use of MPI-based software for compressing speckle images in real-time. The advantage to real time compression is a great reduction in necessary data storage. For example, while running the FCCD in the 100fps mode for five minutes results in 50GB of data, real-time compression by 80% yields 10GB of data.

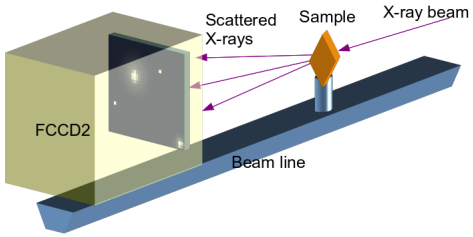

Fig. 1. As x-rays scatter off of sample, high-speed area detector, called FCCD2, captures speckle pattern.

II. FCCD Hardware

The FCCD2 camera head sits on the beam line and collects x-ray speckle patterns. The camera head is wired to a custom electronic circuit board, called the Camera Interface Node (CIN)residing in an Advanced Telecommunication Computing Architecture (ATCA) crate which in turn connects to a back plane running a high speed local Ethernet network. Data is streamed from the CIN through a router residing on the ATCA crate to a resident Linux blade. As the Linux blade processes the camera image data in real time, it streams compressed data to a disk array for storage resident in the ATCA crate.

Timothy Madden, Sifeng Niu, Suresh Narayanan, Alec Sandy are with the X-ray Science Division, Advanced Photon Source, Argonne National Laboratory, Argonne, IL, 60439 USA e-mail: tmadden@anl.gov.

Peter Denes, John Joseph, Victoria Moeller-Chan, Dionisio Doering, Patrick McVittie, and Devis Contararto are with Lawrence Berkeley National Laboratory, Berkeley, CA 94720 USA .

Manuscript received November 15, 2014.

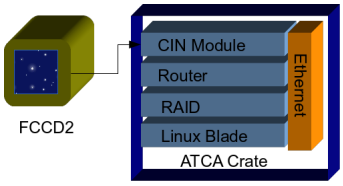

Fig. 2. An example of a cool figure

III. Compression Method

The image data stream from the CIN to the Linux blade in a scrambled pixel order not related to the actual geometry of the CCD sensor.Therefore, the images must be descrambled, that is, rearranged on a pixel-by-pixel basis, to produce images corresponding to the geometry of the sensor. The images are descrambled in real time to produce descrambled images,

$$I_{ds}(x,y) = I_{raw}(f(x,y), g(x,y)), \tag{1}$$

where I_{ds} is the descrambled image, $f(x,y)$ and $g(x,y)$ are maps from raw image space to descrambled space, and I_{raw} is a raw scrambled image as emitted from the CIN.

Because the FCCD2 detector produces a fixed background signal unrelated to captured x-rays, dark subtraction, or the subtraction of an averaged reference image from each streamed data image, must be performed. The reference image is computed from an average of a series of dark images, or images taken in the absence of x-ray light. The dark reference $D(x,y)$ is computed as

$$D(x,y) = \frac{1}{N}\sum_{k=0}^{N-1} I_{ds_k}(x,y). \tag{2}$$

A second reference image $S_q(x,y)$, called the "squared reference image," takes into account the random electronic noise of the FCCD2 on a pixel-by-pixel basis. The squared reference image is calculated as

$$S_q(x,y) = \frac{1}{N}\sum_{k=0}^{N-1} I^2_{ds_k}(x,y). \tag{3}$$

The dark reference $D(x,y)$ and squared reference $S_q(x,y)$ are computed simultaneously by taking a series of dark images before the XPCS experiment begins. Typically, a reference set is acquired before each XPCS data set. Once we have acquired the reference series and computed $D(x,y)$ and $S_q(x,y)$ we can compute the standard deviation image $N(x,y)$ by

$$N(x,y) = \sqrt{S_q(x,y) - D^2(x,y)}. \tag{4}$$

We compute a lower level discriminator or threshold $T(x,y)$ used for image compression as

$$T(x,y) = D(x,y) + P \times N(x,y), \tag{5}$$

where P is some number of standard deviations of electronic noise. Image compression is done as

$$I_{th}(x,y) = \begin{cases} I_{ds}(x,y) - D(x,y) & : I_{ds}(x,y) \geq T(x,y) \\ 0 & : I_{ds}(x,y) < T(x,y) \end{cases} \tag{6}$$

The term $I_{th}(x,y)$ is the image after comparing to the threshold image and $I(x,y)$ is the descrambled image from the camera. When saving the final images $I_{th}(x,y)$ only non-zero pixels are stored. During compression a sparse matrix is generated in which pixel (x,y) position and pixel magnitude are stored. Because of the nature of the XPCS speckle patterns, typically 95% of the pixels are devoid of x-rays and results in a 80% compression rate after storing pixel values and coordinates. The computation load for each raw image acquired in an XPCS experiment is the sum of descrambling, dark subtraction, threshold comparison, and sparse matrix generation. For a pixel of M pixels the number of operations to be performed is around $4M$.

IV. Implementaion of Real-Time Compression for FCCD2

The FCCD2 is preceded by the FCCD1 at APS Sector 8 for XPCS experiments [4]. For compressing XPCS speckle images, the FCCD1 utilized a commercial Camera-Link Dalsa Frame Grabber with a built-in Field-Programmable Gate Array (FPGA). The FPGA on the grabber card was programmed in VHDL to perform descrambling , dark subtraction, lower-level discrimination and sparse matrix generation [5] [6]. Because the ATCA platform used by the FCCD2 supplied no FPGA-equipped circuit board, we were forced to consider alternative technologies for real-time image compression.

A. Real-Time versus Immediate-Time

Before one can decide on which technology to use for real-time data compression, one must be clear on the meaning of "real-time." We use the term "real-time" to mean that the computation keeps up with the incoming data. That is, if a camera generates 1000fps, the computation engine can process 1000fps. In a real-time system, it is not necessary for computations to be complete at any exact clock time. The system may include long data queues that add a varying time delay to the processing. Also, the processing may not run at a uniform rate. But to be real-time, all that is necessary is for the average processing rate to be at least as fast as the incoming data.

We introduce the term "immediate-time" to define a system that must respond to an event at a precise time delay. For example, a trigger system in a particle physics experiment must run in immediate time. The trigger pulses cannot have arbitrary time delays and simply keep up with the experiment, but must occur at a repeatable clock time relative to particle physics events. For image compression, we need a real-time system, but not an immediate-time system.

B. FPGA for Data Compression

An FPGA is the only programmable commercial hardware that can function in immediate time. FPGAs feature high speed processing with predictable and repeatable time delays. As FPGA hardware and development tools improve it has become easier to develop complex FPGA algorithms. Because the FCCD1 project produced extensive FPGA source code,

we were tempted to use an FPGA for real-time compression for the FCCD2. The problem with this plan was that we had no specialized FPGA hardware already resident in the ATCA crate. Therefore, we decided to use a software-based solution.

C. Graphics Processing Unit (GPU) for Data Compression

We investigated the use of a NVidea Tesla GPU for real time processing for the FCCD2 [7]. The GPU can be programmed relatively easily with the CUDA library, and lately can be programmed with more general libraries such as the Message Passing Interface (MPI) [8]. The reason we abandoned the use of a GPU for the FCCD2 is that we could find no GPU to fit into an ATCA crate. We would be required to install a separate Linux computer outside of the ATCA crate to house the GPU.

D. Linux Software Solution

Because of the availability of high-performance Linux blades for the ATCA platform, we chose to create a real-time software library for XPCS data compression. The development time is relatively short when writing in C++, as compared to VHDL for FPGAs. Also, more blades can be added to the ATCA crate if more processing is needed. Because the FCCD2 ATCA crate already houses a Linux blade, we felt it low-risk to develop a data compression software library. If the software solution were to fail, we could always revert to developing specialized FPGA hardware for data compression.

V. Multi-threaded versus Multi-process

In creating a software library for real-time data compression, we were forced to chose a software technology. The most traditional software design is to run a single execution thread to process one image at a time. We quickly found that even a high speed Linux blade cannot process quickly enough to keep up with the streamed FCCD2 data when processing images one-at-a-time. We measured the maximum image rate for single-thread processing to be around 300fps for 960×92 pixel images.

The next technology we evaluated was to run a single process with multiple execution threads, with each thread processing its own image. In this way, if four threads are run, four images are processed at once. Because all the treads are part of one process, each of the threads shares the same memory space. Shared memory gives the benefit that no data need be copied from one execution thread to another. A down side to a single-process, multiple-thread application is the difficulty of developing and maintaining the code. Often the data must be mutex protected, which can slow down the processing. We implemented a simple version of a single-process, multi-threaded library running on a high-performance Linux computer. We found that the software could keep up with the 1000fps data rate of the FCCD2.

Finally, we evaluated the use of the MPICH version MPI to create a multi-process program, with each process running a single execution thread and owning its own memory space, necessitating image copies between processes [8]. The multi-process application was easier to develop than the single-process multi-threaded application because no data was shared, and possibly corrupted, by competing threads. Also, the multi-process program can be run across several Linux blades if more processing power is desired. We found that, like the multi-threaded application, the multi-process application could keep up with the 1000fps data generated by the FCCD2. In fact, the two applications ran at about the same speed. Because of ease of support and expandability, we decided to use MPI-based software to perform real-time data compression for the FCCD2.

VI. Structure of the Software

The FCCD2 software began as QT-based program called CINController with two threads: 1. a thread for receiving the UDP data packets generated by the FCCD2 CIN, 2. a thread for image display, saving, and user interface [9]. The application, originally developed at LBL, runs on a single process and provides the most basic readout system for the FCCD2. The UDP receiver thread is set to a high priority and the UDP packet size is "jumbo" or 9000 bytes to minimize the loss of data from the FCCD2 camera. A diagram of the software is shown in Figure 3.

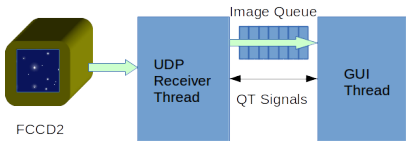

Fig. 3. CINController software structure.

The CINController software was adapted to run in a multi-process mode using the MPICH MPI library. The original CINController software shown in Figure 3 was converted into an MPI program that can run in any number of processes. Each process in the MPI program is called a "rank," and each rank is numbered counting from 0. Rank 0 runs the UDP receiver thread and a second thread for scattering new images to all processes, and running calculations. Rank 1 runs a thread that processes images, then gathers processed images from all ranks and stores them in a queue. A second thread in Rank1 runs the user interface for image display, data saving and camera control. Within any process, threads communicate via QT signals, a mechanism for inter-thread communication provided in the QT library. Between processes, each rank communicates via MPI messages. A diagram of a two-process version of our MPI-based application, called CINControllerMPI is shown in Figure 4. More processes can be added that simply do calculations. Typically we run the CINControllerMPI software with four processes on a single four-core Linux blade during XPCS experiments. We designed the software such that all MPI calls are in a single C++ class called MPIFCCD.

A. Scattering and Gathering of Images

As images are streamed from the CIN to the Linux blade, CINControllerMPI captures the images with the UDP thread of Rank 0. These captured images are stored in a queue for later processing. The MPI thread of Rank 0 takes images off

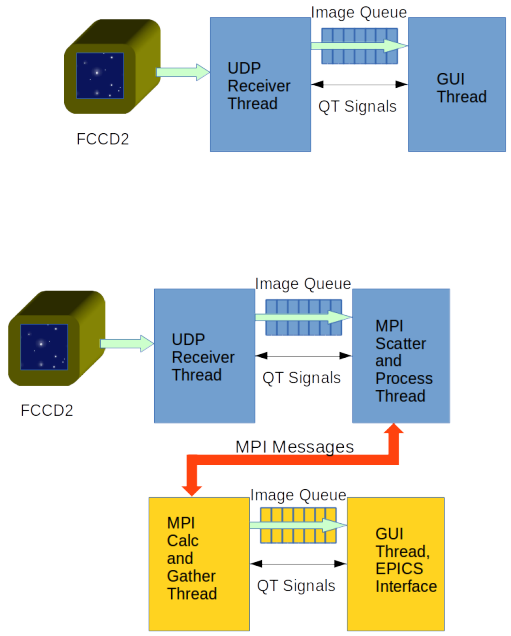

Fig. 4. CINControllerMPI software structure.

the queue and scatters complete images to each rank. That is, all ranks get one complete image to process. Next, all ranks synchronize and process their respective images. Once processing is done, all ranks synchronize, and rank 1 gathers up all processed images from all ranks and stores them on a queue in rank 1. The GUI thread of Rank 1 takes processed images from the queue, displays them, and saves them to the RAID array.

B. Accumulation of Dark Images

Because we must calculate reference images for lower-level discrimination and dark subtraction, book keeping issues arise when the accumulation of images is performed across many processes. In CINControllerMPI, each rank accumulates a subset of the total accumulated images. Each rank must keep track of how many images it has accumulated. Once all images are accumulated all ranks must synchronize to compute the final summed dark and squared reference images as shown in Equations 2 and 3. Once the noise and threshold images of Equations 4 and 5 are computed by rank 1, copies of these reference images must be sent to all ranks. In this way, all ranks can correctly perform dark subtraction and image compression.

C. User Interface Issues

To prevent detector control signals from interferring with data acquisition, he ATCA crate has two separate networks: a 1GB Ethernet network for FCCD2 control, and a 10GB Ethernet network for FCCD2 data. The CINControllerMPI software must communicate with the FCCD2 over both networks without interrupting the MPI real-tome processing. Also, because

TABLE I
MAXIMUM PROCESSING SPEED, 960×92 PIXELS,12K NON-ZERO PIXELS

Cores	Max fps	Max MB/s
1	500	86
2	1000	173
3	1500	259
4	2200	380

APS uses the EPICS control system, CINControllerMPI runs an instance of EPICS as a thread in rank1 [10].

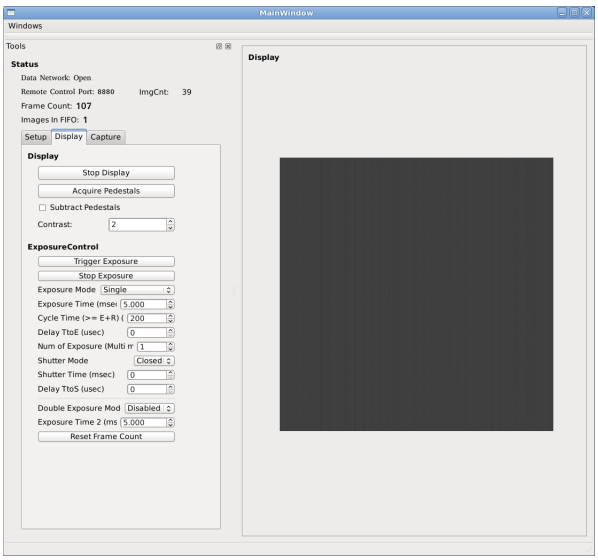

Fig. 5. User Interface developed at LBL

VII. PERFORMANCE DATA AND FUTURE WORK

The FCCD2 system runs Scientific Linux 6 on an Advantec MIC5320 blade utilizing four 2.1GHz cores and residing in a Kontron ATCA crate. The CINControllerMPI software is built with QT version 4.8 and MPICH version 3.0.4 [9], [8], [11]. When running the system at Sector 8 APS we collected software performance data for real-time image compression in the 960×92 and 960×960 modes. For image compression, the system must be able to run indefinitely with the software keeping up with the detector. We tested running CINControllerMPI with one, two, three, and four cores to determine processing speed, and to determine the number of cores necessary for real-time continuous image compression. It was necessary to test of the software could keep up with the detector running at 1000fps in 960×92 mode and at 100fps in 960×960 mode. This was verified by collecting 5minute data sets, necessary for some XPCS experiments. Further, to test the maximum processing rate of the software, the image queues in the software were filled with raw images, and flushed to run the MPI computations as fast as they could run. We tabulated data in maximum frames per second versus number of cores. Tables are shown for both image modes Tables I and II

For acquiring reference images, the system must only run for 10seconds, allowing the software to lag behind the camera by utilizing image queues. By filling up the image queues with raw images and running the reference image calculation as fast

TABLE II
MAXIMUM PROCESSING SPEED, 960×960 PIXELS,115K NON-ZERO PIXELS

Cores	Max fps	Max MB/s
1	50	92
2	100	184
3	140	258
4	160	294

TABLE III
MAXIMUM PROCESSING SPEED, 960×92 PIXELS,REFERENCE IMAGES

Cores	Max fps	Max MB/s
1	300	52
2	550	95
3	780	134
4	960	166

as it could run, we determined the maximum image rate that CINControllerMPI can process. Results are tabulated in Table III for the 960×90 image mode.Because reference image calculation requires more operations than image compression, the maximum data rate is less than the required 1000fps in 960×90 image mode. However, because the reference acquisition needs to be run for a few seconds, the use of image queues to buffer images allows the MPI software to function properly.

Future work for the FCCD2 MPI software includes the running of several Linux blades in the ATCA crate to speed processing and adding more complex computation related to XPCS data analysis.

VIII. CONCLUSION

We have designed MPI-based software running in real-time on a Linux blade to perform image dark subtraction and compression for the FCCD2 XPCS data. This software was design for an application previously performed by a custom FPGA for the FCCD1. It is possible that MPI-based real time software can perform many tasks that are currently performed by FPGAs. In particular, any task that need not run in immediate-time, where the processing must have very short and repeatable time delays relative to the incoming data, but can run in real-time, where the processing must simply keep up with the average incoming data rate with variable instantaneous time-delays, can possibly be run with real-time MPI software as opposed to FPGA firmware. Because of the relative ease of C++ programming compared to FPGA programming, and availability of off-the-shelf computer hardware, many applications can now be performed more simply and cheaply with MPI than with FPGAs.

ACKNOWLEDGMENT

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

REFERENCES

[1] X. Lu, S. G. J. Mochrie, S. Narayanan, A. R. Sandy, and M. Sprung, "How a liquid becomes a glass both on cooling and on heating," *Physical Review Letters*, vol. 100, no. 4, p. 045701, 2008. [Online]. Available: http://link.aps.org/abstract/PRL/v100/e045701

[2] D. Doering, N. Andresen, D. Contarato, P. Denes, J. M. Joseph, P. McVittie, J.-P. Walder, J. Weizeorick, and B. Zheng, "High speed, direct detection 1k frame-store ccd sensor for synchrotron radiation," in *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2011 IEEE*. IEEE, 2011, pp. 1840–1845.

[3] D. Doering, N. Andresen, D. Contarato, P. Denes, J. Joseph, P. McVittie, J.-P. Walder, and J. Weizeorick, "A 1mpixel fast ccd sensor for x-ray imaging," in *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2012 IEEE*. IEEE, 2012, pp. 527–529.

[4] P. Denes, D. Doering, H. A. Padmore, J.-P. Walder, and J. Weizeorick, "A fast, direct x-ray detection charge-coupled device," *Review of Scientific Instruments*, vol. 80, no. 8, p. 083302, 2009. [Online]. Available: http://link.aip.org/link/?RSI/80/083302/1

[5] T. Madden, P. Fernandez, P. Jemian, S. Narayanan, A. Sandy, M. Sikorski, M. Sprung, and J. Weizeorick, "Firmware lower-level discrimination and compression applied to streaming x-ray photon correlation spectroscopy area-detector data," *Review of Scientific Instruments*, vol. 82, no. 7, p. 075109, 2011.

[6] T. Madden, P. Jemian, S. Narayanan, A. Sandy, M. Sikorski, M. Sprung, and J. Weizeorick, "Fpga-based compression of streaming x-ray photon correlation spectroscopy data," in *Nuclear Science Symposium Conference Record (NSS/MIC), 2010 IEEE*. IEEE, 2010, pp. 730–733.

[7] "Nvidea tesla website," 2014. [Online]. Available: http://www.nvidia.com/object/tesla-supercomputing-solutions.html

[8] "Mpich website," 2014. [Online]. Available: www.mpich.org

[9] "Qt project website," 2014. [Online]. Available: www.qt-project.org

[10] M. Rivers, "areaDetector: Epics software for area detectors," 2010. [Online]. Available: http://cars9.uchicago.edu/software/epics/areaDetector.html

[11] "Scientific linux website," 2014. [Online]. Available: https://www.scientificlinux.org/