Demonstrating Cross-Facility Data Processing at Scale with Laue Microdiffraction

Michael Prince
Argonne National Lab
Lemont, Illinois, USA
mprince@anl.gov

Doğa Gürsoy
Argonne National Lab
Lemont, Illinois, USA
dgursoy@anl.gov

Dina Sheyfer
Argonne National Lab
Lemont, Illinois, USA
dsheyfer@anl.gov

Ryan Chard
Argonne National Lab
Lemont, Illinois, USA
rchard@anl.gov

Benoit Côté
Argonne National Lab
Lemont, Illinois, USA
bcote@anl.gov

Hannah Parraga
Argonne National Lab
Lemont, Illinois, USA
hparraga@anl.gov

Barbara Frosik
Argonne National Lab
Lemont, Illinois, USA
bfrosik@anl.gov

Jon Tischler
Argonne National Lab
Lemont, Illinois, USA
tischler@anl.gov

Nicholas Schwarz
Argonne National Lab
Lemont, Illinois, USA
nschwarz@anl.gov

ABSTRACT

In February and April 2023 live, at-scale data processing demonstrations were conducted between the Advanced Photon Source (APS), a synchrotron light source, and the Argonne Leadership Computing Facility (ALCF). These tests were run as part of a novel beamline technique: coded aperture laue micro-diffraction. This technique requires a significant amount of compute to decode appeture patterns embedded in the detector stream. An autonomous system was able to send data to ALCF during an experiment, utilize 50 nodes of the Polaris supercomputer to process 6-12 hour scans, and return the data back to the APS within 12-15 minutes behind the detector. With scan points arriving every 72 seconds, the system kept up with the beamline, potentially enabling in-experiment analysis. The data processing system utilizes Globus infrastructure and an on-demand queue to dynamically acquire nodes on Polaris. The underlying reconstruction algorithms were parallelized via MPI and accelerated with custom CUDA kernels.

CCS CONCEPTS

• **Applied computing** → **Physics**; • **Computing methodologies** → *Distributed algorithms*; *Parallel algorithms*; • **Information systems** → *Information systems applications.*

KEYWORDS

experimental facilities, data processing, distributed and parallel computing, GPU acceleration

ACM Reference Format:
Michael Prince, Doğa Gürsoy, Dina Sheyfer, Ryan Chard, Benoit Côté, Hannah Parraga, Barbara Frosik, Jon Tischler, and Nicholas Schwarz. 2023. Demonstrating Cross-Facility Data Processing at Scale with Laue Microdiffraction. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3624062.3624613

XLOOP '23, Nov 12-–17, 2023, Denver, CO

ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3624613

1 INTRODUCTION

The Advanced Photon Source (APS) is synchrotron light source which produces high intensity, focused x-rays in order to conduct a wide array of experiments from material science to microbiology. The APS is undergoing a substantial upgrade [2] expected to increase its brilliance by up to 500x. Slated for completion in 2024, this upgrade will enable new types of experiments and speed up data collection rates of existing experiments.

Post-processing and analysis are required for many of the experiments conducted at the facility. Normally this work is done on local workstations. Post upgrade, the compute demands are expected to increase dramatically, to the point where HPC is required, even for in-experiment analysis. To address this, the APS has partnered with the Argonne Leadership Computing Facility (ALCF). ALCF's supercomputers provide centralized, interconnected compute resources which can readily handle the requirements of multiple experiments simultaneously.

In February and April 2023, shortly before upgrade began, two experiments were conducted with a novel data collection technique. The technique, Coded Aperture Laue Microdiffraction [6, 7], requires a significant amount of compute, comparable to the requirements of some expected, post-upgrade techniques. Processing a single 12-hour scan on a CPU workstation computer would have taken over 250 days to complete. This experiment provided a unique opportunity to demonstrate the capability of off-site processing at ALCF and exercise the infrastructure that has been constructed to enable this type of cross-facility analysis.

2 EXPERIMENT OVERVIEW

Laue microdiffraction is a technique used at synchrotron facilities in order to analyze materials with crystalline structures. As a focused

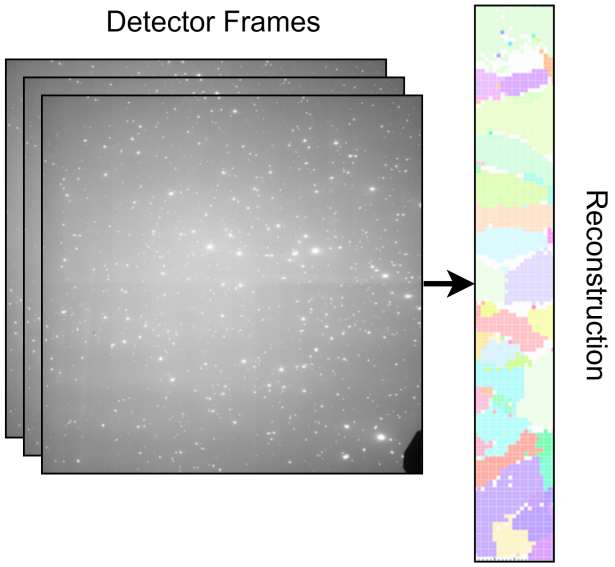

Figure 1: An example of an Aluminum dataset and the corresponding reconstruction. This dataset contains 20 points and a total of 8020 frames. Each point is processed into a vertical slice of the 2D reconstruction displayed on the right. The reconstruction segment has a 1 micron resolution and the colors display the orientation of the crystalline grains within the sample.

x-ray beam passes through a material, individual crystallites along the path of the beam diffracts along different angles depending on their orientation. A pixel array detector positioned above the sample collects the position of these diffracted beams.

In order to obtain a full, 3D map of the structure, the angle and position of each diffracted beam must be resolved. To do this, typically, a wire is swept between the sample and detector in order to occlude the rays. By observing the position where the wire occludes a given ray, the depth of the ray's origin can be calculated.

In practice, this is a very time-intensive process. Scans can take anywhere from hours to a day to complete. Coded aperture laue microdiffraction, aims to reduce the time it takes to scan a sample. In some experimental setups, this technique could lead to a 5x reduction in scan times. This is done by paralleling the collection of Laue diffraction spots. By employing a coded aperture, a unique sequence encoded into the detector stream depending on the position, depth, and angle of the diffracted beam. As an added step in post-processing, each beam's sequence can be decoded to resolve the aforementioned parameters. The signal that's received is very sensitive to real-world factors. As such, the decoding process is extremely computationally intensive as it requires a significant amount of signal analysis and search algorithms to find optimal fits for the pattern.

2.1 Reconstruction Algorithm

A separate work [6] describes the reconstruction algorithm in detail. At a high level, for each pixel of a scan point, the reconstruction algorithm performs 3 distinct steps to determine the depth of the diffraction beam's origin. The first step is to take the mask pattern and correct it relative to the physical pixel position and the mask's orientation in space. The parameters for this part of the algorithm are collected via physical measurements and a calibration scan to determine the orientation of all elements of the experiment to within a few tenths of a degree.

The corrected mask is then compared against the measured data to find the position on the mask the beam traveled through. First the incoming data is normalized and bias-corrected in order to maximize the the contrast in the comparison. When an encoded beam is measured by a pixel in the detector, the encoding should align with a sub-section of the mask. The mask is encoded with a de Bruijn sequence so that each sub-section of the mask is unique. An exhaustive search is performed between the measured and expected pattern to determine the section of the mask where they best align.

Next this subsection is fed into a finer analysis in order to determine the bounds of the incoming signal. Since the problem may be underdetermined, the calculation is done with a non-negative least-squares for additional robustness.

Finally, with the results of both the position and signal reconstruction, the ray of the signal can be traced back into the sample via the orientation parameters used in the mask correction. Using the reconstructed depths of every ray of origin, downstream analysis can be done to reconstruct 3d models of the crystalline structure and extract various material properties from the sample.

2.2 Data Requirements

A full scan of a material consists of an arrangement of scan locations, referred to as "points", are typically spaced 1 micron apart. The scanning grid of the samples conducted in the tests were approximately 400 points each. Each point contains a stack of 300-400, 2048x2048 images. Each image is taken at a motor step of the coded aperture.

Data sizes and rates can vary depending on the experiment parameters. The total data size of a point is typically between 3-5 GB per point. A data point arrives every 60-90 seconds. The output is tied to the motor and detector timings and is therefore consistent, with sub-second precision. The output size and format depends on the parameters of the reconstruction. Generally they can range from 1/2 to 2x the original input size and are also stacks of 2MP images. Therefore the transfer rates are trivial at a maximum of around 1.4 MBps. However, the storage requirements are fairly large, with a full reconstruction, including input and output data can produce 4-8 TB of data.

3 DATA PROCESSING SYSTEM

The data processing system was constructed primarily with existing tools and infrastructure hosted at ANL. Most sensor and motor data at the APS is transferred internally via the Experimental Physics and Industrial Control System (EPICS) [9]. Monitoring and control software at the beamline automatically reads the data from the detector and writes to HDF5 [13] files onto a NAS on the beamline network.

The APS has developed a software package, the Data Management System (DM) [14]. DM is designed to automatically assign

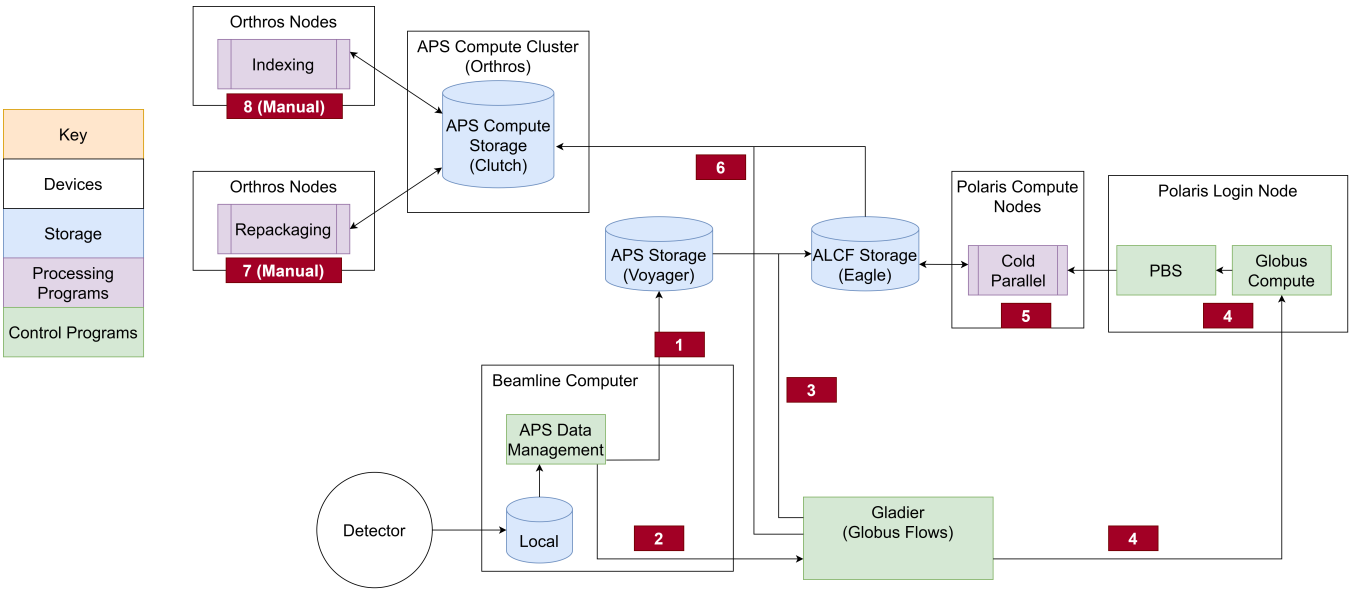

Figure 2: An overview of the systems and elements involved in the data processing system. Data is produced by the detector and is automatically picked up by the APS Data Management system which uploads it to APS central storage (1). Then, a Globus flow is spawned (2), which transfers the data to ALCF's Eagle storage (3). The flow triggers a job on Polaris (4) which reconstructs the scan (5). Then the flow transfers results back to the APS (6) for manual downstream processing and viewing (7, 8).

user permissions, move data around facility systems, and contains a workflow engine to automate data prepossessing. The latter two components were used as part of the data processing system.

Globus [1] handles much of the cross-facility data management. Both APS and ALCF host Globus collections. Globus collections serve a shared file system with APIs and GUIs to manage data across multiple collections. Both the APS and ALCF have dedicated hardware to host the collections with high-speed interconnects between the two facilities. Much of the orchestration is constructed with the Gladier toolkit [5], a python API over Globus Flows [5] that allows developers to create automations that span Globus infrastructure. While the workflows themselves are static once launched, the flows can be parameterized when invoked. Globus also provides the Globus Compute [4] service, to manage and launch compute tasks over remote HPC systems. A Globus Compute endpoint was deployed on Polaris to enable the workflow to provision resources on the machine.

In order to balance needs of typical ALCF users and experiment use, the ALCF has allocated 10%, 56 nodes, of Polaris to two special queues: a preemptable and on-demand queue. Implemented with the Portable Batch System (PBS) [11], the preemptable queue functions similar to the standard production queue, except that preemptable jobs can be terminated without notice if not enough resources are available to start on-demand jobs. The on-demand queue is a FIFO queue which, if necessary, can preempt jobs from the other queue until all 56 nodes are utilized. It is designed for time-sensitive jobs, like this demonstration, where scientists may want intermediate results while a physical experiment is still running. These demonstrations were the fist instance where the demand queue on Polaris was exercised at scale.

An overview of this system is described in Figure 2. The DM Data Acquisition Service (DAQ) is set up to monitor the folder where data is written from the EPICS infrastructure. The DAQ performs two tasks. First, for each point that arrives, the DAQ uploads data to the APS Voyager central storage. Voyager is a 10 PB GridScaler GS14KX file storage appliance with a GPFS high performance filesystem. The Voyager storage system hosts a Globus Connect Server 5 (GCS5) endpoint. Once the data is uploaded to central storage, the DM Processing Service launches a custom Globus flow designed to process the Laue data. The DM workflow invokes a python program and feeds in information about the file to be processed via command line arguments to the Gladier python client, which then configures and launches a Globus flow.

From this point, the Globus flow handles all downstream processing and data transfer. Depending on the experiment name defined in DM and the incoming data from the detector, file locations for the experiment and individual point are determined and embedded into the workflow. First data is transferred from Voyager to ALCF's file store, Eagle. Globus collections on both ends manage the data transfer. Once the data is on Eagle, the workflow queries the Globus Compute endpoint running on Polaris. The endpoint then interacts with the PBS queuing system directly to allocate compute resources. Each workflow query sub-processes a blocking PBS call that doesn't release until the job has finished. This is done to hold the workflow until data has been processed.

Once the data has been processed, the data is repackaged into a format more convenient for downstream processing. Depending on the number of nodes used to process a flow, this could be done as part of the job, or later as a downstream task. After the job finishes, the workflow resumes and transfers data to the APS

Clutch endpoint, a data store attached to the APS's local compute resources. From there, the scientists on the experiment can begin manual analysis of the experimental data. Initial prototypes of the system contained hooks to automatically query downstream processing from the local compute resources at the APS via a hook in the DM workflow. However, this automation was dropped due to experiment requirements.

4 PARALLELISM AND DISTRIBUTION

The incoming data stream can be paralleled along multiple points, between every pixel, and within some of the searches performed by the reconstruction algorithm. The system parallelizes along each of these dimensions. Each point that arrives is queued as a separate job. Depending on the number of nodes allocated and the rate of incoming jobs, Polaris's queuing system aumatically handles parallelization of multiple points.

Within each image, each pixel can be processed independently. Multiprocessing, synchronized with MPI across multiple nodes handles this layer of parallelism. Since the high level control of the program is performed via python and is subject to the global interpreter lock, we choose to spawn one python process per rank. Polaris nodes contain a single AMD EPYC 7543P processor with 32 cores. Due to this, we assign 32 ranks to each node with each rank bound to a core. Since the processors have 64 threads, tests were conducted with 64 ranks per node. However, no noticeable speedup came from this configuration. We suspect this is the case due to the CPUs being highly utilized and underlying python libraries like NumPy using available threads.

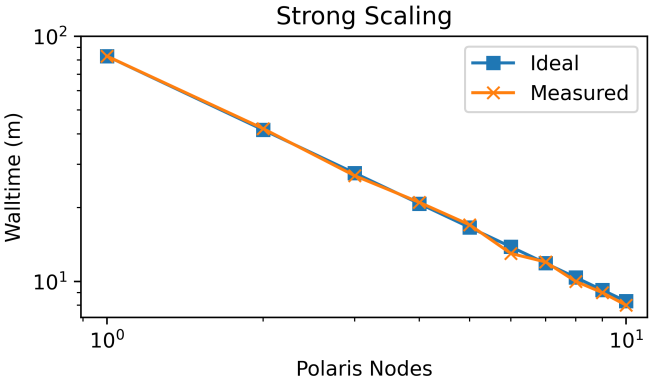

Figure 3: Strong scaling test of the initial program before the February live demo. Since no communication is required between different processes, scaling is near-ideal, relative to the initial run time at 1 node.

Within a single point, the pixels are distributed before processing begins. The software analyzes the number of nodes and ranks from environment and MPI variables and automatically distributes the appropriate number of pixels to each rank. Past the initial distribution, each pixel is processed independently with no synchronization or communication required. Since read and write times are minimal compared to to the total run time, the program scales near-ideally with the number of nodes in the system. This scaling, shown in Figure 3 was examined in a strong scaling test between 1 and 10 nodes.

GPU acceleration was required in order to reduce the runtime and node usage to the point where it could be reasonably be run during an experiment scan's time-frame. Most of this optimization took place in the reconstruction algorithm as the surrounding IO and infrastructure was minimal in comparison.

Each Polaris node contains 4 NVIDIA A100 GPUs. The initial implementation of the program assigned 1 process for each GPU, leaving 28 CPU only processes. More pixels are distributed to the GPU processes to ensure that all processes finished at around the same time. In order to perform the GPU distribution, the CUDA environment variable is set before the python process was launched for each rank. However, in stark contrast to the tests where cores were oversubscribed, we found significant performance gains from distributing 8 ranks per GPU and letting the OS handle resource contention. The demonstrations shown in the following sections were performed with this configuration.

4.1 GPU Acceleration

Of the components of the algorithm, the position reconstruction was the most time-intensive portion of the program. In the initial NumPy CPU implementation, this component would take over 95% of processing time. Fortunately, within a single pixel, the process of checking the signal's position is highly parallelizable. To effectively utilize GPU hardware, the original program was refactored to batch pixels together. The batch size parameter, controlling the number of pixels processed together will be referred to as b.

Table 1: Position Reconstruction GPU Performance

Batch Size	Pixels per Second
CPU Baseline	7.95
1	443.30
8	1015.12
32	1199.04
128	1263.11
1024	1286.67
2048	1287.33
4096	1282.54

In a process similar to a convolution, the signal is swept across the corrected mask. At each position, the absolute difference of each discrete element is taken between the signal and the mask segment. When averaged together, this difference represents how closely the signal matches the segment of the mask. The segment with the lowest difference is the segment that is selected for the finer downstream calculation.

The size of the mask array will be referred to as m and the size of the detector array, d. Both are floating point arrays. Typically m would be around 4000-5000 elements in size and d would be 200-400 depending on the experiment and processing parameters.

Initial implementation of the algorithm in NumPy and CuPy [12] would allocate a new array with the subtracted values and perform the reduction on this new array. This kind of operation more naturally fits into NumPy's interface where loops are discouraged

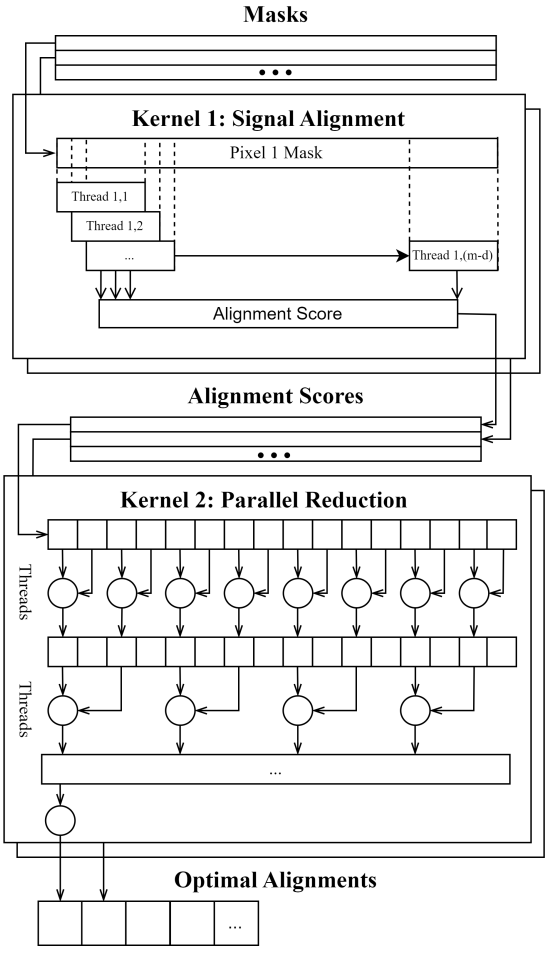

Figure 4: A visual overview of the two CUDA kernels handling position reconstruction. Both kernels are 2D with data along one dimension and batch along the other to better utilize the hardware. The alignment kenrel initializes a 2d set of threads of size $(m-d)\times b$. Each thread operates independently of each other and requires no block-level synchronization. The alignment scores are then fed into a parallel reduction kernel to find the optimal alignment among all the scores. The optimal alignments array of size b is then returned to the CPU for downstream processing.

and separate operations, such as the subtraction and reduction are performed sequentially. The new array before the reduction was prohibitively large at a size of $(m - d) \times d \times b$. Even with small batch sizes, with 32 processes running on the machine, the nodes would rapidly run out of GPU, and occasionally, system memory. A jit-compiled JAX [3] implementation was able to achieve high performance with a low memory overhead. However, d could vary slightly in size due to scaling depending on the physical location of the pixel on the detector. Due to Jax's static shape and control flow requirements, the necessary logic to handle these edge cases couldn't be JIT compiled.

In order to work around the aforementioned memory and edge case constraints two custom CUDA kernels were developed to perform this calculation. Figure 4 describes their operation. These kernels were implemented with Numba's [10] python CUDA wrapper. First the mask and signal data, sizes $m\times b$ and $d\times b$ respectively are batched and sent to GPU memory. An additional array of size b containing the size of the signal data is also passed into the kernel to handle the variation in signal size. The first kernel performs the difference at each point in the mask. Rather than storing each value in an intermediate array, a loop iterates over the mask slice and accumulates the difference at each point into a thread variable. A conditional checks for the edge case and sets the loop to stop early if needed. The output is an array of size $(m - d) \times b$. The array is kept in GPU memory and then fed into a second kernel to find the index of the minimum element. A parallel reduction kernel, designed similarly to Harris's summation kernel [8] was implemented to find the index of the minimum element in each pixel. The resulting array of size b is then transferred back to CPU memory for signal reconstruction.

The position reconstruction performance gain from these kernels was measured across several different batch sizes and compared to the CPU baseline in Table 1. These benchmarks were conducted on the AL dataset using single rank run on a Polaris node. Even at small batch sizes, the performance is significantly better than processing on CPU. By increasing the batch size, performance increases by up to 3x by better utilizing the GPU hardware.

5 DEMONSTRATIONS

Several demonstrations were conducted using the networks, equipment, and infrastructure that would be used in production runs of this system. In order to simulate the detector, an existing dataset is copied, file by file, in the order and at the same data rate as it was collected. For DM, this functions identically as the data arriving directly from the detector. A summary of all the tests conducted is shown in Table 2.

Table 2: Demonstration Resource Statistics

Dataset	Points	Nodes	Node H.	NH/Job (SD)
AL (2/9)	40	11	54.11	1.71 (0.22)
AL (2/10)	40	10	60.80	1.52 (0.01)
Ni2(1) (2/25)	422	10	598.68	1.42 (0.01)
Ni2(1) (2/28)	422	10	600.79	1.42 (0.72)
Ni2(2)* (4/1)	345	5	228.52	0.66 (0.04)

*Processing conducted on a later software version

5.1 Initial Scaling

Initial scaling was performed with previously collected data sets which had already been processed and verified to have the correct analysis parameters. This dataset consisted of an aluminum sample 20 points. In order to increase the run time of these tests, the dataset was iterated over twice for a total of 40 points per test. Since 55 nodes were allocated to the job, initially 11 nodes were assigned per job. However a node-specific issue was discovered and the node was

removed from the pool. So, for the following test was conducted at only 10 nodes per job with only a minor decrease in performance.

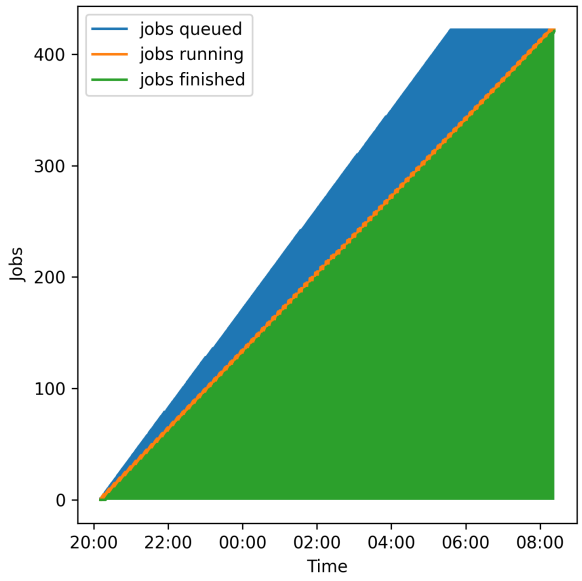

Figure 5: Data recorded as part of the Feb. 28 run. This full-scale run lasted for 9 hours. However as shown in the graph, the reconstruction was being processed for 3 hours after the data collection had finished.

5.2 Experiment-Scale Demonstrations

The first experiment-scale test was conducted on Feb 25 and a second one shortly afterwards on Feb 28. In this tests, a Ni2 sample of 422 points was reconstructed twice using different parameters. The experiment was configured to read a point every 72s. At this rate, with the software at the time, 50 nodes was not enough to keep up with the compute demand of the incoming nodes. As shown in Figure 5, a buffer built up in the Globus Flows system and the nodes continued to process for a few hours afterwards.

Another experiment scale test was conducted on April 1 as part of one of the last experiments before the APS-U shutdown. In this test, analysis was performed on a second Ni2 sample. Further optimizations had been made to the reconstruction program at this time leading to an 2-2.5x speedup. Shown in Figure 6 only 40-45 of the 56 available nodes were needed at a time to meet the demand of the beamline.

6 FUTURE WORK

Much of the future work for this technique centers around ease-of-use and continued optimization of the reconstruction algorithms. No live tests are to be conducted till the APS comes back online in early 2024. However, several datasets were collected during both experiments that haven't been processed yet. Work is being done to provide a user interface to allow scientists to conveniently launch batch-processing jobs onto Polaris. Much of the underlying infrastructure built for live experiment processing can be reused for

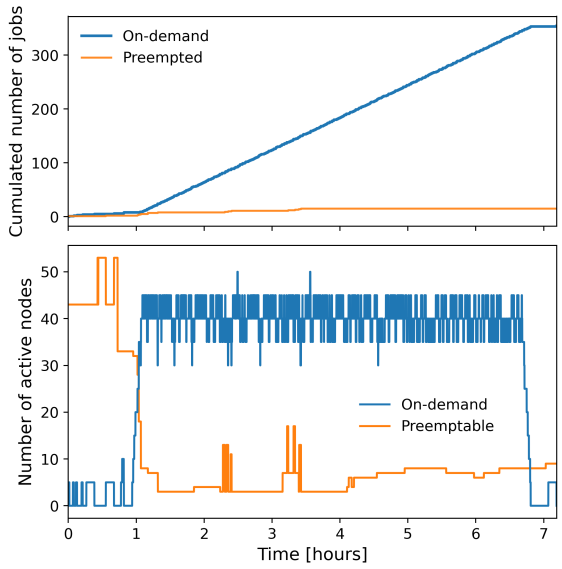

Figure 6: Data recorded by the queuing system on Polaris as part of the April 1 experiment. As jobs start to enter the demand queue, jobs are preempted to make space for the demonstration's jobs.

this batch processing. Additionally some parameters of the experiment can be automatically retrieved from beamline sensors and integrated into the data processing system.

While the most computationally intensive component of the program is executed on GPU, a significant amount of vectored computation is still being executed on CPU. Further speedups may be realized with custom CUDA implementations to better utilize the available hardware. Additionally, the calculations are done sequentially. A future version of the program could utilize asynchronous dispatch to prevent the GPUs from idling while the CPU processes data.

An additional energy calculation has been added to the reconstruction. This calculation is search-based and the CPU compute time is about as intensive as the original position reconstruction calculation. Since the search is parallelizable, a GPU based implementation may provide significant speedups.

7 CONCLUSIONS

Both the APS and ALCF have been investing significant resources into developing hardware and software infrastructure to make ALCF's compute resources available for experiments at the APS. This series of demonstrations was the first application of much of this infrastructure at a large scale on live data. While there's still much work to be done in usability and optimization, the underlying data processing system was capable of consistently reconstructing 6-12 hour scans, allocating resources to meet demand, and keeping up with the data rate generated by the beamline. After the APS upgrade, it is expected that other APS techniques may utilize significant compute resources, to the point where HPC-level processing is a requirement. Since much of the system is built on common

infrastructure between the two facilities, this work and the lessons learned may serve as a model to help develop and scale compute infrastructure for other techniques.

For the technique itself, the parallelization, optimization, and deployment onto Polaris has enabled full-scale analysis of the microdiffraction data. The infrastructure developed for these demonstrations is currently being used to analyze a backlog of data collected at these and previous experiments. A separate work, more focused on the scientific analysis will cover results from both these experiments and the data backlog.

ACKNOWLEDGMENTS

This research used resources of the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Basic Energy Sciences, under Contract No. DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

The authors would like to thank all colleagues at the APS and the ALCF for their fruitful discussions and dedication.

REFERENCES

[1] Rachana Ananthakrishnan, Kyle Chard, Ian Foster, and Steven Tuecke. 2015. Globus platform-as-a-service for collaborative science applications. *Concurrency and Computation: Practice and Experience* 27, 2 (2015), 290–305.

[2] Michael Borland and Alexei Blednykh. 2018. *The upgrade of the advanced photon source.* Technical Report. Brookhaven National Lab.(BNL), Upton, NY (United States).

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/google/jax

[4] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. 2020. Funcx: A federated function serving fabric for science. In *Proceedings of the 29th International symposium on high-performance parallel and distributed computing*. 65–76.

[5] Ryan Chard, Jim Pruyne, Kurt McKee, Josh Bryan, Brigitte Raumann, Rachana Ananthakrishnan, Kyle Chard, and Ian T Foster. 2023. Globus automation services: Research process automation across the space–time continuum. *Future Generation Computer Systems* 142 (2023), 393–409.

[6] Doğa Gürsoy, Dina Sheyfer, Michael Wojcik, Wenjun Liu, and Jonathan Z Tischler. 2022. Depth-resolved Laue microdiffraction with coded apertures. *Journal of Applied Crystallography* 55, 5 (2022).

[7] Doğa Gürsoy, Dina Sheyfer, Michael Wojcik, Wenjun Liu, and Jonathan Z Tischler. 2023. Digital autofocusing of a coded-aperture Laue diffraction microscope. *Review of Scientific Instruments* 94, 1 (2023).

[8] Mark Harris et al. 2007. Optimizing parallel reduction in CUDA. *Nvidia developer technology* 2, 4 (2007), 70.

[9] M Knott, D Gurd, S Lewis, and M Thuot. 1993. *EPICS: A control system software co-development success story.* Technical Report. Argonne National Lab., IL (United States).

[10] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 1–6.

[11] Bill Nitzberg, Jennifer M Schopf, and James Patton Jones. 2004. PBS Pro: Grid computing and scheduling attributes. In *Grid resource management: state of the art and future trends*. Springer, 183–190.

[12] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. 2017. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. http://learningsys.org/nips17/assets/papers/paper_16.pdf

[13] The HDF Group. 2000-2010. *Hierarchical data format version 5.* http://www.hdfgroup.org/HDF5

[14] Siniša Veseli, Nicholas Schwarz, and Collin Schmitz. 2018. APS data management system. *Journal of Synchrotron Radiation* 25, 5 (2018), 1574–1580.

Received 18 August 2023