Scaling and Acceleration of Three-dimensional Structure Determination for Single-Particle Imaging Experiments with SpiniFEL

Hsing-Yin Chang
SLAC National Accelerator Laboratory
Email: iris@slac.stanford.edu

Elliott Slaughter
SLAC National Accelerator Laboratory
Email: eslaught@slac.stanford.edu

Seema Mirchandaney
SLAC National Accelerator Laboratory
Email: seema@slac.stanford.edu

Jeffrey Donatelli
Lawrence Berkeley National Laboratory
Email: jjdonatelli@lbl.gov

Chun Hong Yoon
SLAC National Accelerator Laboratory
Email: yoon82@slac.stanford.edu

Abstract—**The Linac Coherent Light Source (LCLS) is an X-ray free electron laser (XFEL) facility enabling the study of the structure and dynamics of single macromolecules. A major upgrade will bring the repetition rate of the X-ray source from 120 to 1 million pulses per second. Exascale high performance computing (HPC) capabilities will be required to process the corresponding data rates. We present SpiniFEL, an application used for structure determination of proteins from single-particle imaging (SPI) experiments. An emerging technique for imaging individual proteins and other large molecular complexes by outrunning radiation damage, SPI breaks free from the need for crystallization (which is difficult for some proteins) and allows for imaging molecular dynamics at near ambient conditions. SpiniFEL is being developed to run on supercomputers in near real-time while an experiment is taking place, so that the feedback about the data can guide the data collection strategy. We describe here how we reformulated the mathematical framework for parallelizable implementation and accelerated the most compute intensive parts of the application. We also describe the use of Pygion, a Python interface for the Legion task-based programming model and compare to our existing MPI+GPU implementation.**

Index Terms—**Pygion, task-based parallelism, coherent diffractive imaging, image reconstruction, single-particle imaging, free electron lasers, nonuniform fast Fourier transform**

1. Introduction

The high repetition rate and ultrahigh brightness of the Linac Coherent Light Source (LCLS) [1] make it possible to determine the structure of individual molecules, mapping out their natural variation in conformation and flexibility. Structural dynamics and heterogeneities, such as changes in the size and shape of nanoparticles, or conformational flexibility in macromolecules are at the basis of understanding, predicting, and eventually engineering functional properties in the biological, material, and energy sciences. The ability to image the structural dynamics and heterogeneities from noncrystalline proteins is one of the driving forces behind the development of XFEL facilities around the world. Single-particle imaging (SPI) requires femtosecond pulses to outrun radiation damage using a concept called diffraction-before-destruction [2]. When an X-ray pulse interacts with a protein freely rotating in space, a diffraction pattern is measured on the detector before the protein is completely destroyed. A train of identical proteins is injected into the interaction region to replenish the sample, as shown in Figure 1.

Data analysis must be performed quickly to allow users to guide their data collection strategy by providing feedback on whether sufficient data has been collected and whether the data quality is sufficient to achieve the desired resolution. This will require near-real-time analysis of data bursts (within tens of minutes from the completion of a given burst), requiring commensurate bursts of computational power.

As a part of the U.S. Department of Energy's Exascale Computing Project (ECP), the ExaFEL project [3] aims to create an automated analysis pipeline for SPI to transfer diffraction data to supercomputers, where reconstruction is performed, then report results within minutes of data collection. This entails reconstructing a 3D molecular structure from 2D diffraction images by using the multi-tiered iterative phasing (M-TIP) algorithm [4]. At a very high-level, each diffraction pattern samples a 2D slice of a protein's 3D Fourier transform. Orientation of the particle has to be determined so that it can be merged in a 3D diffraction volume. The assembled diffraction volume represents the Fourier amplitude of our protein. By solving the phase problem, we recover the 3D electron density of the molecule. The steps are illustrated in Figure 2.

In Section 2, we present algorithmic improvements to the state of the art SPI data analysis. In Section 3, we describe GPU offloading and implementation in Pygion, a Python interface to the Legion programming model which we demonstrate on a modest number of nodes with reconstruction results. In Section 4, we report on computational performance and compare to the MPI+GPU reference implementation.
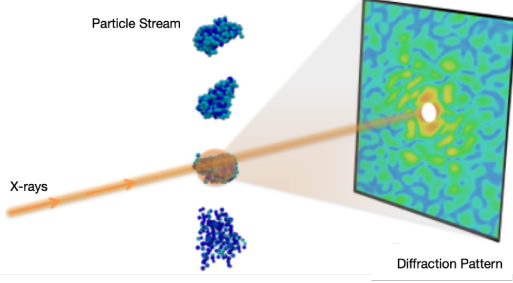
Figure 1: SPI involves recording several thousands to millions of single-shot diffraction patterns of identical proteins before its destruction due to ionization and Coulomb explosion. Protein is replenished via a particle stream.

X-ray diffraction Images

Orientation Determination

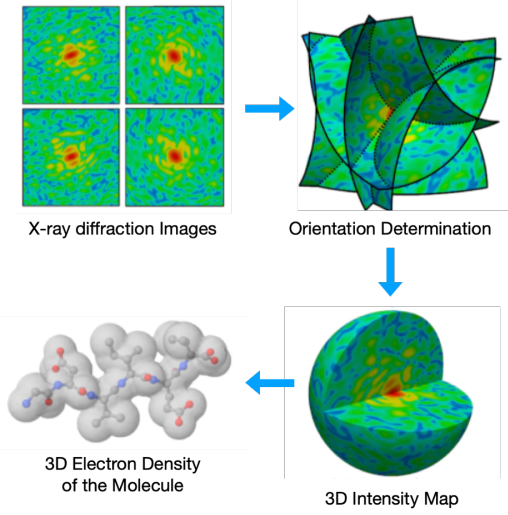3D Electron Density of the Molecule

3D Intensity Map

Figure 2: Reconstruction requires orientation determination, classification into 2D class averages, and 3D merging of intensities in reciprocal space. The Fourier magnitudes are then used to synthesize a real-space image, but only after recovering the phase information missing from raw data.

2. M-TIP Cartesian-NUFFT Framework

The existing M-TIP algorithm [4] relies on a spherical-polar formulation that serially updates the 3D diffraction volume to be consistent with each 2D image, and thus is not scalable. In this section we present a new reformulation of M-TIP based on the nonuniform fast Fourier transform (NUFFT), which allows reconstruction to be parallelized.

A key part of the old algorithm, orientation matching to align 2D images to a 3D model, is based on computing a Wigner-D transform [5], [6] in a spherical basis. Instead, we use a NUFFT of the autocorrelation of the current density model to generate a set of model 2D diffraction images from a fixed orientation grid, and then compare each experimental 2D image to the model image for each orientation.

After alignment, we merge the oriented diffraction patterns into a 3D diffraction volume. Merging is accomplished through the inversion of a NUFFT, which provides

an unbiased approach to modelling the experimental data. Furthermore, the availability of fast libraries for computing NUFFTs [7], [8], coupled with iterative linear algebra solvers, provide a scalable approach to recovering the 3D diffraction volume from a large set of 2D diffraction snapshots.

Scaling the new Cartesian-NUFFT M-TIP framework can be broken down into developing scalable code for four main components: 1) slicing; 2) orientation matching; 3) merging; 4) phasing, as illustrated in Figure 3. The following subsections describe each of these components.
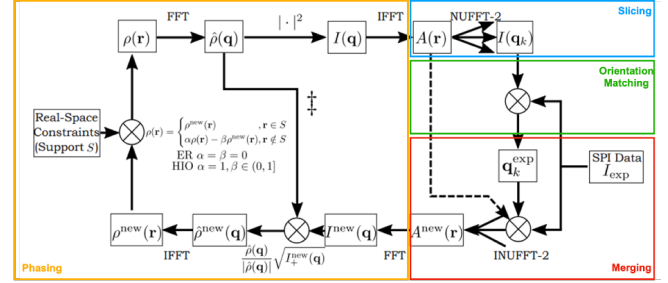

Figure 3: Scalable algorithm for SPI. Phasing (yellow) occurs on a single node, but is independent of data size. Slicing (blue), Orientation Matching (green), and Merging (red) scale with number of images. Notation/nomenclature are in Table 1.

TABLE 1: Notation and nomenclature

Quantities	
$\rho(\mathbf{r})$	Electron Density of Sample
$A(\mathbf{r})$	Autocorrelation of Electron Density
$I(\mathbf{q})$	3D Intensity Function / Diffraction Volume
$I(\mathbf{q}_k)$	Model Diffraction Images / Model Orientations
I_{exp}	Experimental Diffraction Images

Grids	
$\mathbf{r}$	N^3 Cartesian Real-Space Grid ($N = 100 - 500$)
$\mathbf{q}$	N^3 Cartesian Fourier-Space Grid ($N = 100 - 500$)
$\mathbf{q}_k$	Nonuniform Model Data (10^5 points)

2.1. Slicing

In this step, M 2D model images defined on a predetermined set of orientations are computed via the type-2 (uniform to nonuniform, a.k.a. "forward") NUFFT of the autocorrelation of the current electron density estimate.

2.2. Orientation Matching

Each of the N experimental 2D diffraction images are compared to all of the M model images. The orientation of each experimental image is taken to be that of the model image that is closest in the Euclidean distance metric, which scales as $\mathcal{O}(MN)$. Model images are initialized and distributed at startup to minimize communication.

2.3. Merging

Merging can be formulated as inverting a type-2 NUFFT on the oriented experimental data to solve for the autocorre-

lation of the electron density estimate, which is equivalent to solving a linear, discrete ill-posed problem of the form

$$\min_x \|Ax - b\|_2, \qquad (1)$$

where A is the NUFFT operator, x is a vector representing the autocorrelation on a uniform grid, and b is a vector of intensity data on a nonuniform grid.

However, in SPI experiments, the direct beam necessitates a beam stop to prevent damage to experimental equipment, resulting in a missing region in the detector center that corresponds to low frequency information. The incompleteness of the data in combination with the high levels of noise lead to instabilities and nonuniqueness of the solution.

The so-called ill-posed problems can be tackled by regularization, where the experimental data are complemented with external or prior information so that the two sources of information together fully determine a unique solution. Note that when splitting up the problem in M-TIP, since we do not have access to all of the priors from constraints on separate tiers, we penalize the perturbation of the model in each tier. Specifically, we use Tikhonov regularization [9] to approximate the exact solution, and convert the original linear system into another linear system:

$$A^*DAx + \lambda(x - x_0) = A^*Db, \qquad (2)$$

where D is weights, λ is regularization parameter for the penalty term, and x_0 the initial guess of the autocorrelation.

The normal equations for the linear system can be expressed as a convolution on a Cartesian grid with dimensions equal to the number of resolution elements per dimension using the Toeplitz method [10]. Setting this up requires two type-1 NUFFTs to be computed, one on the nonuniform experimental data and one with the nonuniform data replaced with 1's. With this setup, only the type-1 NUFFT computations (computed just once) need to be distributed, since they are a function of the entire experimental dataset. We take advantage of the linearity of the NUFFTs by computing them separately on the experimental data belonging to each node, and use a reduction operation to compute their sum. Once these type-1 NUFFTs are computed, the linear system can then be solved efficiently via conjugate gradient. We repeat this linear solve over nodes, where each rank attempts the solve with a different set of regularization parameters, and use the L-curve method [11] to select the best solution.

2.4. Phasing

The phasing step converts a 3D diffraction volume to a molecular structure. Starting from a random *seed* state, we repeatedly apply real-space constraints on the electron density ρ and reciprocal-space constraints on its Fourier transform $\hat{\rho}$ back and forth through the use of FFT and IFFT. In particular, we apply combinations of error reduction (ER) [12], hybrid input–output (HIO) [13] and shrinkwrap algorithms [14] until convergence is reached.

Phasing only requires computing model values on a Cartesian grid about the same size as the desired density model and can be efficiently solved on just a single node.

3. Performance Optimization

In this section, we discuss the acceleration and scaling strategy of our application.

3.1. Acceleration – GPU Offloading

Merging and slicing operations are offloaded to GPUs using the cuFINUFFT [15] library, and data movement is handled using pyCUDA [16]. We compare performance of a multi-threaded instance using the FINUFFT [8] library with OpenMP to an equivalent CUDA implementation on a NVIDIA V100 and find that the forward function call runs approximately $1.5\times$ faster, and the adjoint function call runs approximately $8\times$ faster for our dataset. A single iteration of merging requires one call to forward, and two calls to adjoint, giving an overall $6\times$ speedup over a similar CPU-only node.

Orientation matching is accelerated using approximately 800 lines of low-level CUDA code. We found this approach performed better than using existing libraries such as Scikit-learn [17] kNN or Facebook AI Similarity Search (FAISS) [18], by a factor of $50\times$ and $4\times$, respectively.

Compared to an equivalent NumPy code [19], offloading phasing to GPU using CuPy [20] gives a $20\times$ speedup.

3.2. Parallelization and Scaling – Pygion

To distribute the application across multiple CPUs and GPUs on multiple nodes, we use Pygion [21], a Python interface to the Legion task-based runtime system. We chose Pygion because the task-based model is a natural fit for the stages of the computation we described in Section 2, and avoids the need for programmers to think about manual data movement or synchronization. Instead, Pygion programs are organized as sets of *tasks* (or functions) that appear to execute sequentially, and *regions*, or data collections passed to tasks. Programmers annotate the *privileges* (read, write, etc.) of the arguments to a task, and the runtime system is responsible for inferring dependencies and issuing data movement as necessary to ensure the program can execute seamlessly on a distributed supercomputer.

Our SpiniFEL implementation in Pygion is structured by turning each major step in the application (as seen in Figure 3) into tasks. The data and partitions in our application are straightforward. For example, for the used images, we simply create a region that holds all the images (even if they could not fit on a single node) and then use Pygion's partitioning operators to decompose them into subregions.

4. Results

We evaluate our approach on Summit [22] using a synthetic dataset. Each Summit node consists of 2 POWER9 CPUs, 6 Nvidia V100 GPUs and 512 GB of memory. Nodes are connected via EDR InfiniBand, and GPUs via NVLink. We consider two implementations, in Legion and MPI, and perform weak scaling experiments up to 32 nodes.

4.1. Dataset

To test our framework, a noise-free synthetic dataset of 500,000 SPI images from a lidless Mm-cpn in the open state from Protein Data Bank (PDB) entry 3IYF is generated, using SPI simulation package skopi [23]. The images provided are 128×128 pixels, with resolution to the edge set to 12 $\mathring{A}$. Figure 4 shows samples of the simulated diffraction patterns.
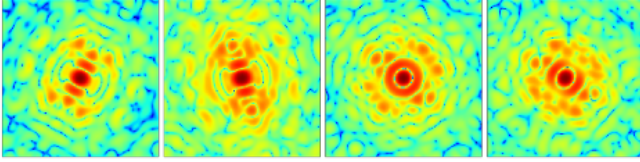
Figure 4: Simulated diffraction patterns of a Mm-cpn at random orientations.

4.2. Performance

Measurements are performed by distributing a different set of 1,000 simulated diffraction images to each node. For weak scaling, we increase the number of nodes from 1 (1,000 images processed) to 32 (for a total of 32,000 images processed) on Summit, with 42 CPUs and 6 GPUs per node.

As shown in Figure 7, both the MPI and Legion versions of SpiniFEL perform well with increasing numbers of nodes as indicated by constant scaling. Compared to an equivalent MPI implementation, Pygion is easier to scale out of the box and leads to speedups of 1.5-fold or more as demonstrated in Figure 5, due to its dynamically managed load-balancing of tasks across cores, shared memory (between distinct Python processes on a node) and high level parallelization constructs. The net result is an alternative HPC programming model capable of providing 3D electron density estimates from large scale datasets in about an hour, as shown in Figure 6.

5. Discussion

Our MPI and Legion implementations both use an offload model based on a combination of Python (CuPy), C++ (cuFINUFFT) and custom CUDA code. The kernel code, i.e., between MPI send/recv calls or inside Legion tasks, is identical between the two implementations. The main difference is that in MPI these kernels are statically assigned to ranks, whereas in Legion this assignment is controlled by the mapper, a performance tuning interface that exists separately from application code, and therefore is more flexible in many ways, including enabling automatic data layout transformations and task placement decisions.

Our analysis indicates that the performance difference, despite identical code, is due to data layout differences. Legion makes it easier to tune data layout for optimal performance, as this can be done separately from the application code.
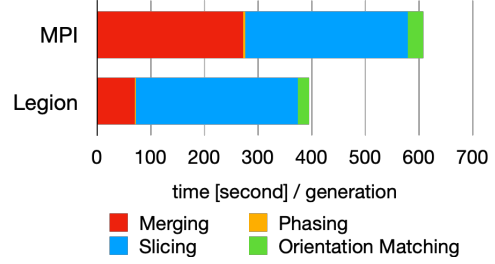

Figure 5: Scaling analysis for SpiniFEL proxy app on one node: breakdown of the time spent in each module using MPI and Legion programming model.
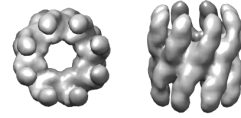

Figure 6: Reconstruction of a lidless Mm-cpn in the open state (PDB ID: 3IYF) from simulated noise-free single-particle diffraction data, displayed as density isosurfaces (top and side views) using UCSF Chimera [24].
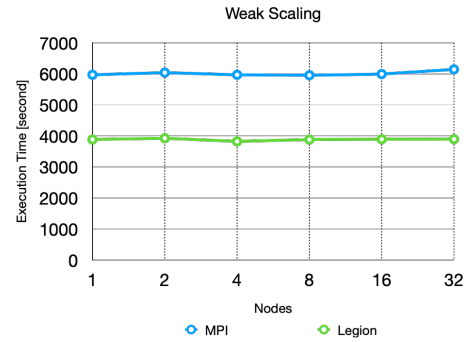

Figure 7: Weak scaling in MPI vs. Legion.

6. Conclusion

LCLS introduces opportunities for novel science in the domain of SPI. We presented improvements to algorithms in SPI that enable potential scaling to large node counts. These algorithms have been implemented in SpiniFEL, an accelerated and distributed implementation that leverages the Pygion programming model.

References

[1] Linac Coherent Light Source. https://lcls.slac.stanford.edu/.

[2] Henry N Chapman, Carl Caleman, and Nicusor Timneanu. Diffraction before destruction. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1647):20130313, 2014.

[3] ExaFEL. https://www.exascaleproject.org/research-project/exafel/.

[4] Jeffrey J Donatelli, James A Sethian, and Peter H Zwart. Reconstruction from limited single-particle diffraction data via simultaneous determination of state, orientation, intensity, and phase. *Proceedings of the National Academy of Sciences*, 114(28):7222–7227, 2017.

[5] Michael G Rossmann et al. Molecular replacement method. 1972.

[6] Peter J Kostelec and Daniel N Rockmore. Ffts on the rotation group. *Journal of Fourier analysis and applications*, 14(2):145–179, 2008.

[7] Leslie Greengard and June-Yub Lee. Accelerating the nonuniform fast fourier transform. *SIAM review*, 46(3):443–454, 2004.

[8] Alexander H Barnett, Jeremy Magland, and Ludvig af Klinteberg. A parallel nonuniform fast fourier transform library based on an "exponential of semicircle" kernel. *SIAM Journal on Scientific Computing*, 41(5):C479–C504, 2019.

[9] Gene Howard Golub and Urs Von Matt. *Tikhonov regularization for large scale problems*. Citeseer, 1997.

[10] Jeffrey A Fessler, Sangwoo Lee, Valur T Olafsson, Hugo R Shi, and Douglas C Noll. Toeplitz-based iterative image reconstruction for mri with correction for magnetic field inhomogeneity. *IEEE Transactions on Signal Processing*, 53(9):3393–3402, 2005.

[11] Per Christian Hansen. The l-curve and its use in the numerical treatment of inverse problems. 1999.

[12] Ralph W Gerchberg. A practical algorithm for the determination of phase from image and diffraction plane pictures. *Optik*, 35:237–246, 1972.

[13] James R Fienup. Reconstruction of an object from the modulus of its fourier transform. *Optics letters*, 3(1):27–29, 1978.

[14] Stefano Marchesini, H He, Henry N Chapman, Stefan P Hau-Riege, Aleksandr Noy, Malcolm R Howells, Uwe Weierstall, and John CH Spence. X-ray image reconstruction from a diffraction pattern alone. *Physical Review B*, 68(14):140101, 2003.

[15] Yu-hsuan Shih, Garrett Wright, Joakim Andén, Johannes Blaschke, and Alex H Barnett. cufinufft: a load-balanced gpu library for general-purpose nonuniform ffts. *arXiv preprint arXiv:2102.08463*, 2021.

[16] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, B. Catanzaro, Paul Ivanov, and Ahmed Fasih. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, 38(3):157–174, 2012.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[19] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[20] ROYUD Nishino and Shohei Hido Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151, 2017.

[21] Elliott Slaughter and Alex Aiken. Pygion: Flexible, scalable task-based parallelism with python. In *2019 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM)*, pages 58–72. IEEE, 2019.

[22] Summit – Oak Ridge Leadership Computing Facility. https://www.olcf.ornl.gov/summit/.

[23] Skopi. https://pypi.org/project/skopi/.

[24] Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Gregory S Couch, Daniel M Greenblatt, Elaine C Meng, and Thomas E Ferrin. Ucsf chimera—a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.