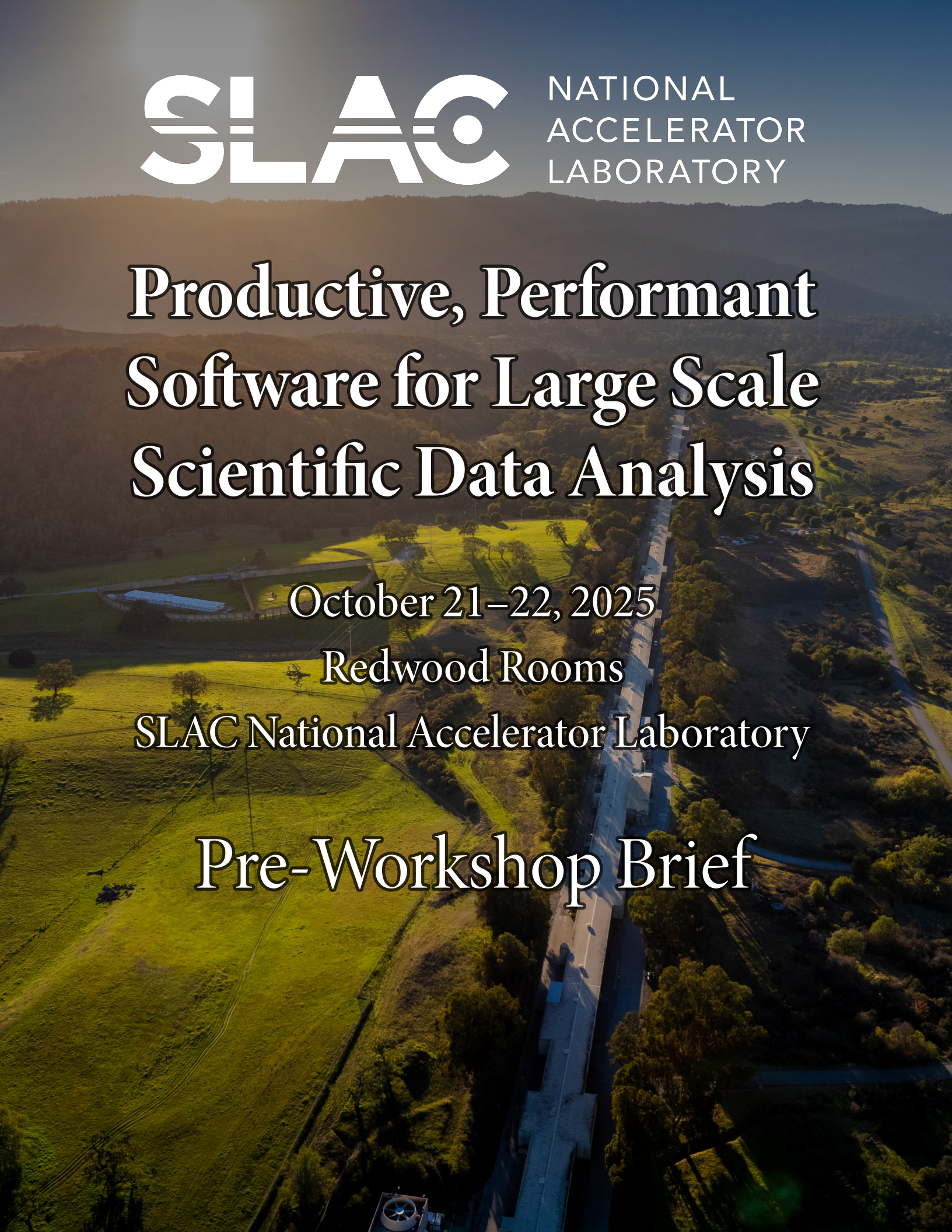


Productive, Performant Software for Large Scale Scientific Data Analysis

October 21–22, 2025

Redwood Rooms

SLAC National Accelerator Laboratory

Pre-Workshop Brief

Productive, Performant Software for Large Scale Scientific Data Analysis

Organizing Committee

- Alex Aiken (Stanford/SLAC)
- Elliott Slaughter (SLAC)
- Johannes Blaschke (NERSC)
- Keita Teranishi (ORNL)
- Patrick McCormick (LANL)
- Roberto Gioiosa (PNNL)

SLAC is operated by Stanford University for the U.S. Department of Energy's Office of Science.

The Office of Science is the single largest supporter of basic research in the physical sciences in the United States and is working to address some of the most pressing challenges of our time.

Contents

Executive Summary

Scientific user facilities (SUFs) at the U.S. Department of Energy (DOE) drive scientific discovery and innovation by delivering world-class experimental capabilities that expand the frontiers of biology, chemistry, physics, and materials science. Over the next 5 years, upgrades at SUFs will generate over an order of magnitude more data, promising to accelerate the pace of scientific innovation if correctly harnessed. However, this flood of data poses challenges for the scientific community, despite continued growth in HPC hardware performance. The current state of the practice and tools optimized for HPC are insufficiently flexible and productive to address the high-stakes, short timelines, and rapidly evolving requirements of highly dynamic scientific user experiments. Additionally, traditional HPC software tools demand expertise that most users of SUFs cannot realistically apply within the pace and pressures of modern experiments, underscoring the need for more accessible, high-productivity approaches. Emerging AI/ML technologies, though promising, do not address these needs, and will not lead to a productive, high-performance software ecosystem without decisive action.

This workshop will explore the research challenges and opportunities in building a highly productive, high-performance software ecosystem for large scale scientific data analysis for users at the SUFs. The goal of the workshop is to identify key research directions that, if addressed, would substantially change the status quo and deliver an order of magnitude increase in productivity and performance for users of SUFs across the DOE complex.

Schedule

Tuesday, October 21, 2025

- **Breakfast**: 8:00–8:30am
- **Session 1**: 8:30–10:00am
  - Introduction: 8:30–8:40am
  - Plenary: Jana Thayer (SLAC): 8:40–9:20am
  - Plenary: Nicholas Sauter and David Mittan-Moreau (LBNL): 9:20–10:00am
- **Break**: 10:00am–10:30am
- **Session 2**: 10:30am–12:00pm
  - Invited Talks: 10:30–11:10am
    - Wah Chiu (SLAC): 10:30–10:50am
    - Martin Pokorny (Caltech): 10:50–11:10am
  - Lightning Talks: 11:10am–12:00pm
    - Xiang Li (SLAC)
    - Oliver Hoidn (SLAC)
    - Sandra Mous (SLAC)
    - Kevin Dalton (SLAC)
    - Elyse Schriber (SLAC)
- **Lunch**: 12:00–1:00pm
- **Session 3**: 1:00pm–2:40pm
  - Plenary: Ben Brown (DOE): 1:00pm–1:40pm

  - Invited Talks: 1:46–2:40pm
    * Ryan Coffee (SLAC): 1:46–2:04pm
    * Hannah Parraga (ANL): 2:04–2:22pm
    * Jean Kossaifi (NVIDIA): 2:22–2:40pm

- **Break**: 2:40–3:10pm
- **Session 4**: 3:10–5:00pm
  - Lightning Talks: 3:10–3:52pm
    * Mike Bauer (NVIDIA)
    * Brad Chamberlain (HPE)
    * Vincent Esposito (SLAC)
    * Valerio Mariani (SLAC)
  - Programming Models Panel: 4:00–5:00pm
    * Arkouda and Chapel, represented by Brad Chamberlain (HPE)
    * cuPyNumeric and Legate, represented by Manolis Papadakis (NVIDIA)
    * DragonHPC, represented by Colin Wahl (HPE)
    * Julia, represented by Johannes Blaschke (NERSC)
    * Lamellar, represented by Ryan Friese (PNNL)
- **Dinner**: 6:00–7:00pm

Wednesday, October 22, 2025

- **Breakfast**: 8:00–8:30am
- **Session 5**: 8:30–9:50am
  - Invited Talk: 8:30–8:50am
    - Kento Sato (RIKEN): 8:30–8:50am
  - Industry Hardware Panel: 8:50–9:50am
    - Austin Ellis (AMD)
    - Larry Kaplin (HPE)
    - Raghu Prabhakar (SambaNova)
    - Skyler Windh (Micron)
- **Break**: 9:50am–10:20am
- **Session 6**: 10:20am–12:00pm
  - Lightning Talks: 10:20–10:54am
    - Dionisio Doering (SLAC)
    - Aashwin Ananda Mishra (SLAC)
    - Joshua Turner (SLAC)
  - Discussion/Response Time: 11:00am–12:00pm
- **Lunch**: 12:00–1:00pm

Plenary Talks

Turning Petabytes into Physics: The Case for Scalable, User-Centric Software for Next-Generation Science at LCLS

Jana Thayer
SLAC National Accelerator Laboratory

The Linac Coherent Light Source (LCLS) enables atomic-scale, time-resolved studies using ultrafast X-ray pulses, but its unprecedented data rates, driven by detector advancements and facility upgrades, demand transformative approaches to data acquisition, processing, and analysis. This talk will outline LCLS' unique experimental challenges, from raw data to scientific insight under tight time constraints. The LCLS Data System architecture meets these needs through scalable infrastructure spanning from edge processing at beamlines to the SLAC Shared Science Data Facility (S3DF) and DOE leadership-class supercomputers, enabling real-time data reduction and adaptive workflows for extreme data rates. Performant, parallel software tools that bridge domain scientists and HPC will democratize access to petascale resources while maintaining performance. Co-designed algorithms and infrastructure prioritize usability without sacrificing scalability, delivering near real-time insights for guiding experiments. This talk will highlight ongoing efforts to develop a collaborative software ecosystem that balances exascale demands with accessibility, unifying detector systems, edge computing, and workflows. Taken together, these advances create a pipeline from detector to discovery, where every photon, pixel, and processor collaborates to push scientific frontiers.

The Grand Bargain of Structural Biology at the kHz Frontier

Nicholas Sauter and David Mittan-Moreau
Lawrence Berkeley National Laboratory

X-ray Free Electron Laser (XFEL) light sources have been used for fifteen years to uncover enzyme mechanisms in extraordinary detail, exploiting a diffract-and-destroy, shoot-and-replace methodology that largely avoids the radiation damage accrued in traditional experiments under continuous illumination. Synchrotron radiation sources are also backporting this "serial crystallography" protocol, which sequentially examines up to 10^5 randomly oriented crystals to give a full dataset. One scientific payoff is that the protein structure is observed under ambient conditions that reflect the operational conformational ensemble. A wide variety of triggering mechanisms have been developed, so enzyme reactions can now be synchronized by either optical pulse, gas incubation, or substrate mixing. Structural changes monitored by diffraction can be correlated with chemical changes detected by optical spectroscopy, and especially for metalloproteins, by X-ray absorption and X-ray emission methods. If the goal is to observe multiple reaction intermediates, time points, and chemical conditions, data collection capabilities will have to increase by one or two orders of magnitude. Current experiments at the Linac Coherent Light Source (LCLS) are clocked at 120 Hz, while the future LCLS-II-HE upgrade will deliver a 1 MHz pulse structure in 2027. Sample delivery methods will struggle to keep pace with this ultrafast beam, while available diffraction imaging detectors will likely put a hard limit on the framing rate of 2 kHz (Jungfrau, 64 GB/s) or 35 kHz (epixUHR, 1 TB/s). These data rates will pose profound challenges for high-performance computing (HPC) systems. The structural biology community is cognizant that some data reduction will be necessary, such as hit finding with edge computing that rejects events containing little information. Some have argued that lossy compression is acceptable, while others maintain that slower data collection without data loss is preferable. This may be a false choice, because a greater volume of data even with loss will increase the overall signal-to-noise of the full dataset. Recent work has focused on data streaming to gain quick feedback to the experiment on the order of minutes. However, this does not relieve the need to write all the data to the hard drive, as we invariably reprocess the data with better parameters at a later time. Indeed, peripheral calculations such as instrument calibration often take considerably more human effort than running the high-performance data pipeline where the automation has largely

been worked out. Furthermore, the high-level organization of the data (which diffraction data corresponds to which experimental sample and condition) is not necessarily provided for by the facility and/or data processing software. Databases turn out to be less-than-ideally portable from one HPC to another, high-level metadata is probably not standardized, and the logs from data processing programs are unlikely to record the full processing history from raw data to final output. All this suggests that a careful balance will have to be struck between raw HPC capacity such as bandwidth, compute and storage; against specialized pipelines developed for particular structural biology experiments.

Invited Talks

Toward a Productive and AI-Ready Software Ecosystem for Scientific User Facilities

Hannah Parraga, Ryan Chard, Ian Foster, and Nicholas Schwarz
Argonne National Laboratory

Scientific user facilities (SUFs) at the U.S. Department of Energy (DOE) like the Advanced Photon Source (APS) are entering an exciting new era, with facility upgrades generating orders of magnitude more data and the Integrated Research Infrastructure (IRI) initiative offering new opportunities for seamless multi-site operations. Pairing advanced computational power across multiple sites with a flexible software ecosystem will enable researchers to accelerate innovation and unlock insights at scale. This talk highlights forward-looking directions that can expand the impact of the APS and similar facilities and empower the next generation of scientific breakthroughs.

In recent years, progress has been made in critical areas that form the foundation for next-generation data ecosystems. Facilities have advanced secure data access and sharing through mechanisms like Globus and facility data management systems[1], while integrated data life cycle management has reduced manual tasks and freed up beamtime for scientific insight[2]. Additionally, advances in data streaming capabilities enable faster analysis without disk storage[3], and collaboration between the APS and HPC facilities connects experimental data with computational resources through dedicated allocations, service accounts, and interfaces like Globus Compute[4]. However, the current ecosystem still faces delays in data movement, job startup, and heterogeneous computing environments across sites that limit real-time experimental decision-making and seamless multi-facility operations. To enable experiment steering and streamline workloads that integrate distributed resources, we must close the loop between measurement and insight. Several opportunities stand out for future progress.

[1]Vescovi, Rafael, Ryan Chard, Nickolaus D. Saint et al. "Linking scientific instruments and computation: Patterns, technologies, and experiences." Patterns 3, no. 10, 2022.

[2]Veseli, Siniša, Nicholas Schwarz, and Collin Schmitz. "APS data management system." Synchrotron Radiation 25, no. 5 (2018): 1574-1580.

[3]Veseli, Siniša, John Hammonds, Steven Henke, Hannah Parraga et al. "PvaPy streaming framework for real-time data processing." Synchrotron Radiation 32, no. 3 (2025).

[4]Prince, Michael, Doğa Gürsoy, Dina Sheyfer, Ryan Chard, Benoit Côté et al. "Demonstrating cross-facility data processing at scale with Laue microdiffraction." In Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, pp. 2133-2139. 2023

First, faster and more flexible access to distributed, on-demand HPC resources across multiple sites is essential. At present, users selecting among three HPC centers lack the information needed to make informed decisions about job placement, such as queue depth. A fully integrated system would provide cross-facility awareness and direct jobs to the machine offering the most efficient access to compute nodes.

Second, data streaming and communication infrastructure must evolve to support secure, wide-area, real-time collaboration. While streaming from detectors has been demonstrated[3], establishing connections with HPC nodes remains cumbersome, and the lack of standard interfaces across facilities creates additional barriers to integration. Opportunities include dynamically provisioning network connections, balancing loads across compute nodes, fault-tolerant transmission, and developing tools that provide consistent access between sites. Further, next-generation visualization tools that ingest detector data streams from HPC centers and deliver live feedback to scientists at the beamline will enable real-time experimental steering based on computational insights.

Finally, uplifting community scientific software is critical. Many widely used x-ray analysis packages were originally designed for single-computer execution at the beamline and cannot process data streams or efficiently use multiple nodes and GPUs. This modernization effort must also include standardized containerized environments that provide consistent software stacks across facilities, improving both cross-site portability and long-term maintainability while simplifying deployment. Investment in modernizing these codebases would ensure that community software fully exploits HPC systems to their full potential and accelerates scientific progress.

Together, these challenges demand new high-performance software ecosystems for the APS and SUFs. Advancing research with real time experiment steering and cross-facility interoperability, we can enable a leap in performance, reliability, and productivity. Addressing these needs will transform research and ensure that DOE facilities remain at the forefront of innovation in the era of data-intensive discovery.

Neural Operators: A Scalable Framework for AI-Driven Scientific Discovery

Jean Kossaifi
NVIDIA

While traditional deep learning has revolutionized learning on fixed dimensional data, from vision to language, it faces fundamental challenges when applied to scientific challenges. Engineering and scientific problems, from weather forecasting to materials science, are governed by partial differential equations (PDEs) on continuous domains, requiring a new class of models that can learn mappings between infinite-dimensional function spaces.

In this talk, I will motivate the need for Neural Operators, a framework designed explicitly for this challenge. Unlike traditional models that are tied to a specific data resolution, neural operators learn the underlying mathematical operators directly from data. This results in a flexible and scalable approach for resolution-independent learning for complex spatiotemporal simulations. We will cover the core principles of the neural operator framework, and discuss the primary challenges and opportunities for its application on concrete engineering problems.

Software Solutions for Large-Scale Data Management in Scientific Research

Kento Sato
RIKEN

Light source facilities produce massive data volumes that strain current storage and I/O systems. We introduce TEZip, an AI-assisted lossy compression framework that leverages deep learning models to capture spatiotemporal correlations in experimental data. TEZip achieves high compression ratios while preserving scientific fidelity, enabling more efficient storage and analysis. We will highlight its design, integration with HPC environments, and applicability to large-scale experimental workflows.

DSA Radio Telescope: Design for High-throughput Data Processing

Martin Pokorny
Caltech

The DSA is proposed to be a world-leading radio survey telescope and multi-messenger discovery engine. The array will consist of 1650 6.155m dishes instantaneously covering the 0.7 – 2 GHz frequency range, spanning an area of 20 km × 16 km in a radio-quiet valley in Nevada. In a five-year initial survey, the DSA will image the entire viewable sky (~31,000 deg^2) repeatedly over sixteen epochs, detecting >1 billion radio sources in a combined sky, increasing the known radio population 100-fold. It will be unprecedented relative to any radio telescope existing or planned and will transform our understanding of the cosmos. It will have near complete sampling of the uv-plane allowing us to replace a traditional correlator digital backend with a "radio camera" that produces images in real-time. Simultaneously, data will be streamed to a second digital backend, called Chronoscope, that will search the skies for millisecond-timescale transients, such as fast radio bursts and pulsars. A third, smaller digital backend will be use for pulsar timing measurements, as a component of the NanoGRAV pulsar timing array project.

The DSA presents many design and data processing challenges, often at scales beyond any yet encountered in radio astronomy. In this talk, I will describe some of these challenges—from signal acquisition through final data storage—together with design elements that will allow the system to meet those challenges. The critical role played by "forward modeling", which implements a sort of digital twin for all system components in the signal path, and informs the design of data processing algorithms needed to produce final data products, will also be described.

Sensors-to-Edge-to-HPC Ecosystem: In Flight Data Analysis for Autonomous Control Systems

Ryan Coffee
SLAC National Accelerator Laboratory

Given the growing ubiquity of AI in commercial domains, we will explore what the complimentary integration into science domains could mean for the National Laboratory ecosystem. This ecosystem could enable an autonomous flow of innovation that draws from all branches of domain science and converges to inject invention directly into the marketplace. To do so, however, the historical dividing line between federally funded research and US private industry would need to blur. The benefits to the US economy of an integrated Sensors-to-Edge-to-HPC ecosystem could lead to the next stage of intelligent systems driving a revolution in industry. Various science domains serve as a development ground for this convergence of technologies for the future.

Computational Demands of Processing Cryogenic Electron Images for Obtaining 3-Dimensional Structures of Biological Samples Across Different Complexities and Resolutions

Wah Chiu and Grace Nye
Division of CryoEM and Bioimaging, SSRL
SLAC National Accelerator Laboratory

The computational requirements for cryogenic electron microscopy (cryo-EM) vary widely depending on the experimental approach, data quality, and the biological complexity of the target. Across techniques such as single-particle analysis, cryo-electron tomography (cryo-ET), and cryo-volume electron microscopy (cryo-vEM), raw data sizes, storage needs, and processing times can range from modest to extremely demanding.

For typical single-particle cryo-electron microscopy (cryo-EM) of modest sized macromolecular complexes (under 1 MDa), raw and processed data can total between 5 to 10 TB. This process typically requires 12 to 24 CPU cores, 2 to 4 GPUs, approximately 128 GB of memory, and can take several days to weeks for processing. In contrast, targets with significant conformational or compositional heterogeneity often necessitate larger datasets, extending processing times from weeks to several months, or even over a year. Larger assemblies, such as viruses (greater than 5 MDa), may demand up to 25 TB of storage, increased memory capacity, and intensified CPU/GPU usage, with similarly prolonged processing durations.

Cryo-electron tomography (cryo-ET) workflows for cellular studies present distinct challenges. A typical dataset consisting of 100 tilt series (~300 GB of raw data) can grow to 10 TB during processing, requiring 48 to 64 CPU cores and up to 512 GB of RAM. Subtomogram averaging of macromolecular assemblies for subnanometer resolution structures within cells can take several weeks to months of GPU resources. In comparison, whole-cell cryo-ET reconstructions may require less GPU time (around one week) but still demand substantial CPU and memory capacity.

Cryo-volume electron microscopy (cryo-vEM) datasets for tissues are relatively lightweight, averaging around 50 GB of raw data, with total processing needs under 500 GB, achievable within approximately 12 hours on 1 to 2 GPUs. However, existing software does not yet support the visualization, segmentation, and annotation of different subcellular features, nor the extensive classification required to derive

meaningful insights.

At our NIH CryoEM and CryoET Centers, between June 2024 and April 2025, we averaged 20,000 images collected daily across all microscopes, generating roughly 15 TB of raw data per day, leading to a total of 1.7 PB over this period. These volumes highlight the urgent need for scalable storage and computing infrastructure. Importantly, the resource requirements of various aforementioned electron images of cryogenic-preserved samples are influenced not only by data volume but also by the biological complexity and heterogeneity of the samples, the quality of the data collected, and the computational demands of refinement strategies, particularly those based on machine learning approaches.

	Raw data size	**Total processing storage (TB)**	**CPU cores**	**Memory**	**GPUs**	**GPU hours**
Single particle <1 MDa	5 TB (10k movies)	~2 TB	~12–24 cores	~128 GB	2–4 GPUs	3 days – years
Virus	5 TB (10k movies)	up to 25 TB	~24–32 cores	~256 GB	2–4 GPUs	5 days – years
Single particle w/ high heterogeneity	5–15 TB (10–30k movies)	up to 10 TB	~12–24 cores	~128 GB	2–4 GPUs	months to years
CryoET (subvolume averaging)	~300 GB (100 tilt series)	up to 10 TB	~48–64 cores	~256–512 GB	2–4 GPUs	months to years
CryoET (cell)	~300 GB (100 tilt series)	~1–5 TB	~48–64 cores	~256–512 GB	2 GPUs	1 week
Cryo-vEM	50 GB	~256–500 GB	~32–64 cores	~256–512 GB	1–2 GPUs	~12 hours

Lightning Talks

Automated Beam Alignment & Stability For Light Sources

Aashwin Mishra, Matt Seaberg, Ryan Roussel, Auralee Edelen, Daniel Ratner, and Apurva Mehta
SLAC National Accelerator Laboratory

The increasing brightness of light sources, for instance, the diffraction-limited upgrade of the Advanced Photon Source (APS) and the high-repetition-rate upgrade of Linac Coherent Light Source (LCLS-II-HE), offer avenues for a deeper understanding of the fundamental nature of essential processes and opportunities to leverage such insights to gain advances in emerging new technologies. However, these deep insights require increasingly complex experiments that require extreme precision to be achieved and maintained over long experimental durations. For instance, experiments at LCLS-II-HE will require the X-ray beam to be within a fraction of a micron in diameter, with pointing stability of a few nano-radians, at the end of a kilometer-long electron accelerator, a hundred-meter-long undulator section, and tens of meters of X-ray optics. This increased complexity and throughput of experiments, enabled by enhanced brightness, will produce data at over 1 Tb/s, rivaling the largest data generators in the world. This requires us to bridge the widening gap between the rate of data collection and that of scientific analysis. Without real-time active feedback control and an optimized pipeline to transform measurements into scientific information in nearly real-time, researchers will struggle to optimally utilize the high-brightness sources, be overwhelmed by a deluge of mostly useless data, and fail to extract the highly sophisticated insights that the upgrades promise. Ultimately, the scientific success of light sources relies on a large and diverse user base. There is an urgent need to render currently heroic experiments routine, even for experienced light source users.

We outline a Machine Learning approach towards alignment and drift correction of complex optics systems. To this end, we use the Hard X-Ray Split and Delay (HXRSND) as the illustrative example. This consists of two branches: the minimally adjustable “channel-cut” (CC) branch, and the “delay” branch with twelve degrees of freedom. This delay range lies between 5 to 500 ps. With the advent of LCLS-II-HE, HXRSND optics will be critical for ultrafast studies of complex materials. Thus, we must ensure that operational inefficiencies and system limitations do not become a bottleneck. The alignment of the HXRSND requires a spatial overlap between the two branches at the sample with very high precision, along with optimized intensity at the output. For instance, in X-ray Photon Correlation Spectroscopy (XPCS), where the HXRSND can provide twin pulses with custom delays, both the branches must be aligned to the same photon energy to within

~ 0.1 eV, with almost perfect overlap and matched intensities from the two branches. Currently, the HXRSND is aligned sequentially, by an experienced system expert, using intermediate sensors, because even a skilled human is unable to optimize in more than 2-3 dimensions. It takes an experienced operator between 1 to 4 hours for alignment, and the final setting is often far from optimal. Additionally, due to thermal and mechanical drift, the system must be manually re-aligned after ~ 15 minutes of operation, further throttling scientific throughput.

Regarding optimization from scratch, we outline a domain-informed Bayesian optimization approach. Single-crystal optics exhibit a peculiar optimum in the scattering plane, characterized by a sharp rise followed by a gently sloping top. Finding and maintaining crystal-optics systems in these sharp, flat-topped optima requires submicron and nano-radian precision in multidimensional manifolds that are thousands of times larger than the optimum along each dimension. This needle-in-a-haystack problem presents a significant challenge not only for experienced beamline scientists but also for traditional Bayesian optimization (BO) approaches. We outline how this problem is intractable for classical single and multiobjective BO, as well as advanced algorithms such as Trust Region Bayesian Optimization (TurBO), etc. Our domain-informed BO approach relies on prior knowledge of the relationship between the objective function topology and the input variables. This inductive bias is incorporated through transformations of the input feature vectors, aligning them with the principal directions of the objective function and leading to faster convergence to a lower-dimensional submanifold of the search space. Additionally, we utilize knowledge of the needle-in-a-haystack nature of the problem to engineer the acquisition function, thereby enabling a progressively higher degree of exploration through reheating. This enables the algorithm to converge to the optimum on this lower-dimensional sub-manifold.

With respect to beam stability and drift correction, traditional approaches have focused on feedback loops that aim to correct deviations from a predetermined setting. We utilize Time Varying Bayesian Optimization (TVBO) for drift correction. Using numerical simulations, we demonstrate the efficacy of TVBO in correcting for linear drift, non-smooth temporal drift, and constrained TVBO for multi-objective control settings, which represent real-life operating conditions. Transitioning from feedback loops to TVBO represents a paradigm shift from reactive stabilization to proactive, continuous performance optimization. While extant drift correction approaches focus on correcting deviations from a predetermined setpoint, TVBO continuously seeks an optimum in the presence of drift. Thus, the beamline would not only be stable but would also strive to operate at its peak achievable performance at all times, even as thermal, mechanical, and electronic conditions

continually change. Furthermore, TVBO can be used in multi-objective settings using different objective functions and constraints, which is not simple using feedback loops. These changes represent a step toward self-driving accelerator facilities, where control transitions from a system of manually tuned, independent feedback loops to an automated agent that can re-optimize over the complex, high-dimensional, and time-varying input space.

Domain-informed Bayesian Optimization for beam alignment and Time-Varying Bayesian Optimization for drift correction represent refinements of current practice and transformative solutions that can increase scientific throughput at next-generation light sources.

Interactive, HPC-scale Exploratory Data Analysis in Arkouda: Past Successes and Future Challenges

Bradford L. Chamberlain, Hewlett Packard Enterprise (blc@hpe.com)

Extended Abstract

Background *Arkouda*[1] is an open-source Python library supporting HPC-scale exploratory data analysis (EDA) at interactive rates. In practice, Arkouda has been run on data sets of dozens to hundreds of Terabytes using dozens to thousands of HPC compute nodes. Crucial data-intensive operations such as 'argsort' or 'groupby' are optimized to use aggregated communication, permitting 256 TiB of 8-byte values to be sorted using 8192 compute nodes of an HPE Cray EX in half a minute[2]. Arkouda supports an extensible framework, permitting new modules and capabilities to be added over time, such as the Arachne framework for graph analytics[3].

Like most high-performance Python libraries, Arkouda achieves its speed and scalability by implementing key capabilities in another language. But where most fast Python libraries are implemented by calling into C++ (say), Arkouda uses a client-server model in which the Python client communicates with a server written in Chapel[4], potentially running on a cluster or supercomputer. Beyond resulting in fast and scalable execution, the approach also enables a Python-based data analyst to call familiar routines from their typical working environment (e.g., a Jupyter notebook or Python shell) and drive a supercomputer interactively. Unlike the typical HPC approach of submitting jobs to Slurm, waiting for results, deciding on a next step, expressing it, and submitting a new job, this permits the analyst to interact with their data within the human thought loop and avoid losing their train of thought.

Though such an HPC server for Python could be written in traditional approaches such as C++, MPI, OpenMP, and/or CUDA, Arkouda's original developers chose Chapel in large part for the productivity, readability, and writability of its code[5], and to lower barriers when its Python users wanted to extend its capabilities without having to learn traditional HPC programming models.

Current Status and Efforts Despite its many virtues—speed, scalability, an open-source implementation, portability, extensibility, productivity—Arkouda has not yet achieved the level of popularity of adjacent tools such as Pandas, Spark, or Dask. In part, this is arguably because it was late to the game relative to those technologies, which had already established a firm foothold in the EDA space. In part, it is likely because, for many mainstream users and data sets, the speed and scalability provided by conventional technologies is sufficient, such that there is no need for something as HPC-oriented as Arkouda. Such users often think they have "big data" when their

---

[1] https://arkouda-www.github.io/

[2] *Chapel 1.31/1.32 Release Notes: SS-11 / IB Performance Status*, Elliot Ronaghan et al., September 28, 2023.

[3] *On the Design of a Framework for Large-Scale Exploratory Graph Analytics*, Oliver Alvarado Rodriguez, Ph.D. thesis, 2025.

[4] https://chapel-lang.org/

[5] *7 Questions for Bill Reus: Interactive Supercomputing with Chapel for Cybersecurity*, Chapel blog, Feb 12, 2025.

volumes may actually be quite modest in size compared to those generated by a DOE Scientific User Facility (SUF) or cyber-defense workload. A third aspect is that the HPC community tends to get caught in negative cycles of (i) hoping to use mainstream solutions in order to leverage their larger communities, only to find that they are insufficient; (ii) failing to invest in productive new software solutions for fear that they are too risky and/or incapable of being sustained by the HPC community (despite pouring much larger amounts of money into speculative hardware designs); and then (iii) continuing to conservatively use and adapt traditional, low-level notations that focus on hardware mechanisms and capabilities rather than user-level abstractions and productivity.

Though Arkouda has not overtaken other EDA technologies, its development proceeds apace, focusing on making it easier to install, deploy, extend, and debug; as well as reducing differences with conventional EDA technologies through API improvements and additions. Beyond that, the Arkouda community is seeking new users and use cases that would benefit from its speed, scalability, interactivity, and flexibility; and to that end is adding new features such as checkpointing, improved file I/O, sparse linear algebra, and multidimensional array operations. With respect to these last points, to our knowledge, Arkouda's use has not been explored by DOE SUFs, and we would be interested in learning about and exploring motivating use cases to understand what more might be required to make Arkouda a viable solution for such workloads.

Future Research Questions and Challenges Beyond the basic "matchmaking" question of Arkouda's applicability to SUF workloads and needs, other research topics that would be valuable to pursue in this space include the following:

- Although Chapel supports targeting GPUs as well as CPUs, Arkouda has not yet made use of GPUs due to its most intensive EDA operations, like 'argsort' and 'groupby', being network-bound. What Scientific Data Analysis (SDA) operations would benefit from GPU acceleration at scale, and what new needs might they have for GPUs and Chapel's support for GPU computing—for example, GPU-initiated communication? Support for GPUs to be first-class citizens in a global namespace?
- As exotic new hardware accelerators become more commonplace, what role will they have in SDA computations, and to what extent will they be treated as general-purpose programmable devices vs. library operations implemented in silicon?
- What developments in IO subsystems and file formats would be beneficial to SDA, and what changes might be needed in system-level software and its user-level interfaces to make good use of those capabilities in a framework like Arkouda?
- Though Arkouda is extensible, what more can be done to reduce barriers to adding new capabilities in a modular way in order to react to rapidly evolving requirements and high-stakes, short-term deadlines?
- How can HPC software innovators help break the HPC community out of its self-defeating mentality with respect to novel programming systems? How can new and innovative HPC software be fostered and sustained once its initial research studies and objectives are met?

High-Rate Detectors: from data generation to data processing in real time

Dionisio Doering[1], Ryan Herbst[1], Lorenzo Rota[1], James Hussel[1], Abhilasha Dave[1], Larry Ruckman[1], Antonino Miceli[2], Ryan Coffee[1], Jana Thayer[1], Conny Hansson[1] and Angelo Dragone[1]

1 - SLAC National Laboratory, 2 - Argonne National Laboratory

Abstract – LCLS-II, a free-electron laser (FEL) X-ray source, began operations at SLAC National Accelerator Laboratory in 2020. This machine is an upgrade to LCLS-I and, upon full commissioning, will produce X-ray pulses at repetition rates up to 1,000,000 Hz. To capitalize on this high repetition rate, detectors and readout systems must operate at matching frequencies and manage the resulting data volume.

The SLAC X-ray detector program's plan for future cameras is outlined in Figure 1. Under this plan, full-frame readout detectors are being designed and deployed for rates up to 100,000 frames per second. At higher rates, experiments are expected to be temporally or spatially sparse, enabling data processing and reduction to begin at the ASIC level. This ranges from simple thresholding and zero suppression to event-driven operation, where empty frames or frames outside user-defined classes are suppressed. In both scenarios, on-detector pre-processing such as gain calibration and dark subtraction can be performed. The ePixUHR$_{35kfps}$ detector already implements these tasks. Beyond pre-processing, more advanced machine-learning–based capabilities are being developed.

SLAC has proposed and is actively developing the SLAC Neural Network Library (SNL), which maps models trained in high-level tools (e.g., Keras, TensorFlow) onto FPGAs for real-time inference. Figure 2 shows the design flow for FPGA-based ML models using SNL. A key capability of SNL is dynamic reconfiguration of weights and biases to support continual learning and model retargeting. This is essential in environments like LCLS-II, where the same detector serves multiple scientific campaigns and algorithms must be updated or retrained to address new scientific questions. Data processed at the edge are then transferred to downstream compute resources (GPUs, CPUs, etc.) for further analysis—work led by the LCLS Data Systems team.

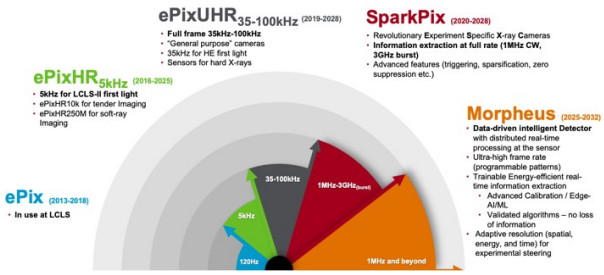


Fig 1. SLAC Detector Family, from the ePix detector operating at 120Hz to 1,000,000 detector designs.

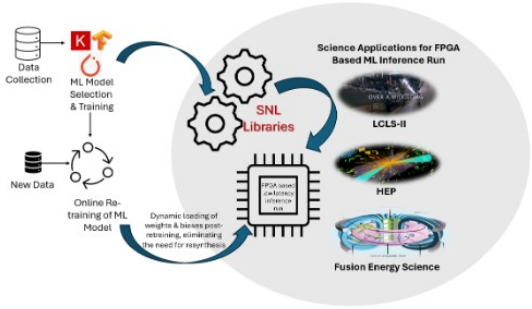


Fig 2. SNL – A Machine Learning Library for real time FPGA inference design flow.

Looking ahead 5–10 years, the SLAC team is researching adaptive, data-driven detectors. These detectors tightly integrate AI cores that monitor incoming data and adapt detector parameters to optimize data quality and maximize on-detector reduction while preserving user-relevant information. The first of its kind is the Morpheus detector, which aims to develop key ASIC-level building blocks (pixel front-end, ADCs, high-speed reconfiguration bus, etc.) and ML models that autonomously inspect data and extract calibration information.

This tight integration of data processing with detectors opens many research avenues, including analog ML; advanced digital algorithms in the ASIC and detector FPGA (e.g., PCA, wavelet transforms, ML inference); and distributed processing across multiple nodes. Ultimately, detectors and edge processing must be proposed and co-designed with HPC algorithms for end-to-end optimization, enabling Scientific User Facilities at the U.S. Department of Energy to drive scientific discovery and innovation.

High-throughput materials discovery via small-molecule serial femtosecond crystallography at LCLS

Elyse Schriber[1], Daniel Rosenberg[1], Daniel Paley[2], Maggie Willson[3], Aaron Brewster[2]

[1]*Linac Coherent Light Source, SLAC National Accelerator Laboratory,* [2]*Molecular Biophysics and Integrated Bioimaging, Lawrence Berkeley National Laboratory, Berkeley, CA* [3]*Department of Chemistry, University of Connecticut, Storrs, CT*

Small-molecule SFX (smSFX) at XFELs has become a reliable tool for determining structures of new, unknown materials from microcrystalline powders[1234] and LCLS is now world-leading in smSFX capabilities available to this new user community. Additionally, we have developed the first-in-world mail-in chemical crystallography program. Our first two mail-in smSFX experiments in Run 22 and Run 23 identified four highly technologically relevant compound classes where smSFX will have the greatest impact. Metal-organic frameworks (MOFs), covalent-organic frameworks (COFs), inorganic covalent solids, and organic molecular crystals. Our automated sampled delivery and data collection methods allow us to collect data in a high-throughput manner and provide mail-in user groups with high-quality, accurate structures. In one mail-in beamtime, we have been able to impact research efforts on a global scale, contributed new science to multiple technologically relevant compound classes and have continued to cultivate the new and growing smSFX user community. smSFX stands to benefit extraordinarily from LCLS-II HE. At our current repetition rate of 120 Hz, we can collect a complete smSFX dataset in 5 minutes, allowing us to potentially measure 200 samples from 50 user groups in five 12-hour shifts. At kHz repetition rates we will be able to rapidly expand the number of users and samples we can accommodate per beamtime. The increase in repetition rate will also rapidly increasing data rates during beamtime. The high-throughput nature of the mail-in chemical crystallography program relies heavily on real-time feedback; therefore, our data processing software and data analysis infrastructure will have to scale accordingly with the high data rates we see during the LCLS-II HE era.

References

1. Schriber, E. A. *et al.* Chemical crystallography by serial femtosecond X-ray diffraction. *Nature* **601**, 360–365 (2022).
2. Aleksich, M. *et al.* XFEL Microcrystallography of Self-Assembling Silver n-Alkanethiolates. *J. Am. Chem. Soc.* **145**, 17042–17055 (2023).
3. Aleksich, M. *et al.* Ligand-Mediated Quantum Yield Enhancement in 1-D Silver Organothiolate Metal–Organic Chalcogenolates. *Advanced Functional Materials* **35**, 2414914 (2025).
4. Kotei, P. A. *et al.* Engineering Supramolecular Hybrid Architectures with Directional Organofluorine Bonds. *Small Science* **4**, 2300110 (2024).

X-ray Photon Fluctuation Spectroscopy Data Handling

Joshua Turner
SLAC National Accelerator Laboratory

The advent of next-generation X-ray free electron lasers, such as the Linac Coherent Light Source II, will soon be capable of delivering X-rays at a repetition rate approaching 1 MHz continuously. This will require the development of data systems and advanced software to handle experiments at these type of facilities. One particular case is in X-ray photon fluctuation spectroscopy, where a large CCD array must be read out and the necessary information extracted on a shot-by-shot basis. Currently, as the LCLS, 5 min of data collection leads to 320Gb of data, but takes 5 times as long to analyze the data. Scaling to how we manage this today, this means at the maximum repetition rate of the future, 5 min of data collection would generate 3.2Pb of data, and would take a couple thousand hours to analyze, presenting a major bottleneck to carry out these important kinds of measurements in the future. We have had some success with demonstrating a framework which implements a machine-learning algorithm to automatically extract the contrast parameter from the collected data, the most critical piece of information, but this represents a major unsolved problem which will be discussed. Targeting this problem will take a major effort and likely involve the incorporation of high-performance computing, advanced ML or AI algorithms, and more a sophisticated software environment.

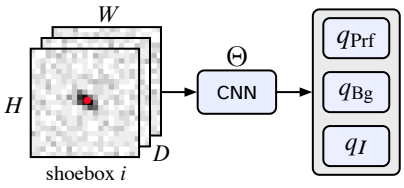
Variational Inference for the Future of Structural Biology

Luis Aldama, Doris Mai, Kevin Dalton

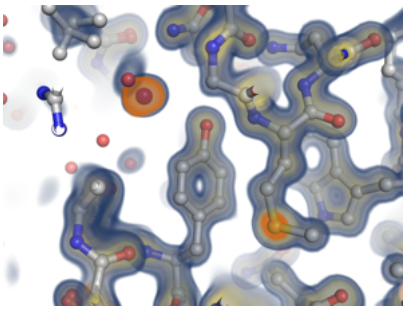
September 11, 2025

Crystallography is a major activity as DOE user facilities including X-ray, electron, and neutron sources. After over a century of development, the analysis of crystallography data is a well established technology for determining the structure of molecules. With the arrival of fourth-generation synchrotrons and X-ray free electron lasers, crystallography is being challenged by new experiments which seek to infer small, time-resolved changes from very heterogeneous samples. This paradigm requires more precise statistical analysis than structure solution. To meet this challenge, new algorithms based on differentiable programming, deep learning, and variational inference are being developed. In this talk, we will present two such algorithms addressing the estimation of diffracted photon flux from diffraction patterns (Figure 1) and the correction of systematic errors in diffraction data (Figure 2). Furthermore, we will speculate on how these two models can be combined to produce an end-to-end differentiable model of diffraction. We hope that this model could form the basis of an inference engine for structural dynamics.

(a) Diagram of integration network



(b) electron density map produced by differentiable integrator

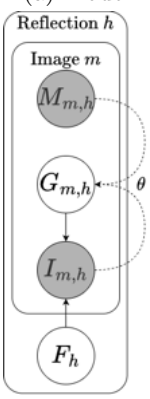


(c) Iodide anomalous peak heights (σ)

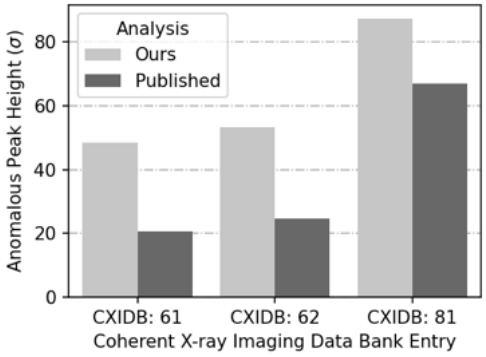
Epoch	204 IOD	205 IOD	206 IOD
epoch 1	30.23	13.19	29.15
epoch 3	31.82	13.90	30.24
epoch 5	33.16	14.58	30.65
epoch 7	33.77	15.10	31.14
epoch 9	32.33	14.42	30.03
epoch 11	**34.09**	**15.26**	**30.88**
epoch 13	32.02	14.02	30.10
Ref. (DIALS)	**32.68**	**14.75**	**29.69**

Figure 1: Preliminary results toward a differentiable model of diffraction by integrating X-ray diffraction data with stochastic variational inference [2, 3, 1]. a) The region of interest around a reflection is called a *shoebox*, which is a $D \times H \times W$ array of pixels centered on a predicted reflection (red dot). Typically, $D > 1$ for rotation-method data, and $D = 1$ for time-resolved serial data. The shoebox is passed through a convolutional neural network (CNN) with learnable parameters Θ, which outputs variational distributions of the reflections profile q_{Prf}, background intensity q_{Bg}, and integrated intensity q_I. The expectation of q_I provides an estimate of the integrated intensity, and the variance of q_I provides the uncertainty of this measurement. b) An example of the electron density which can be reconstructed from this model is shown in blue/yellow. The orange map is the anomalous difference map. This result corresponds to "epoch 11". c) The iodide anomalous peak heights at various stages of model training. The result for a conventional reference integration algorithm (DIALS [4]) is included for comparison.

(a) Model



(b) Precision of inferred structure factors



(c) Example electron density (CXIDB: 81)

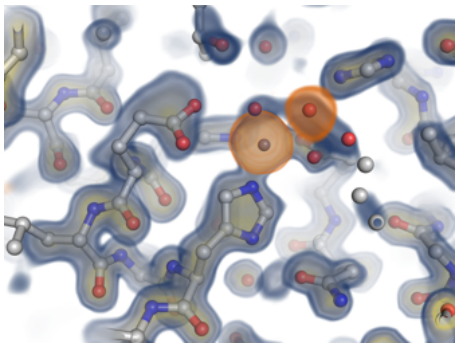


Figure 2: A differentiable model of diffraction by predicting systematic errors in observations with stochastic variational inference [1, 3, 2] a) The graphical model represents the statistical assumptions of the algorithm. The intensity of reflection, $I_{m,h}$ depends on the the structure factor amplitude F_h as well as the scale factor (systematic error), $G_{m,h}$. The structure factor amplitudes are learned as independent, distributions, one for each reflection, h. As indicated by the dashed arrows, the scale factors are estimated by a deep-learning model from the intensities and per-reflection metadata $M_{m,h}$. b) This bar plot shows the performance of this model relative to published values on three publicly available data sets. The measure used for comparison is the signal-to-noise ratio of the largest peak in the anomalous difference electron density map produced by the model, a good proxy for the precision of inference. c) An example of the quality of electron density maps produced by this method. The $2F_O - F_C$ is rendered in blue and yellow. The orange map is the anomalous difference map contoured at 5σ highlighting two Zn^{2+} ions in the thermolysin active site.

References

[1] Blei, David M., Kucukelbir, Alp, and McAuliffe, Jon D. "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112.518 (Apr. 2017), pp. 859–877. arXiv: 1601.00670.

[2] Jordan, Michael I et al. "An introduction to variational methods for graphical models". In: *Machine learning* 37.2 (1999), pp. 183–233.

[3] Kingma, Diederik P. and Welling, Max. *Auto-Encoding Variational Bayes*. Dec. 2013. arXiv: 1312.6114.

[4] Winter, Graeme et al. "*DIALS* : Implementation and Evaluation of a New Integration Package". In: *Acta Crystallographica Section D Structural Biology* 74.2 (Feb. 2018), pp. 85–97.

Reprising Tomasulo's Algorithm for Distributed Machines

Michael Bauer
NVIDIA

One of the most successful abstractions in all of computer science for the past several decades has been Tomasulo's Algorithm: programs appear to execute sequentially, but have implicit (instruction) parallelism dynamically extracted from them even across abstraction boundaries. Despite its success, the idea of leveraging such a powerful idea is wholly absent from modern high performance computing. I'll argue briefly for remedying this state of affairs and the potential opportunities it could confer on a multitude of software ecosystems.

Physics Informed Foundation Models for Coherent Diffraction Imaging

Oliver Hoidn, Aashwin Mishra, Matt Seaberg, and Apurva Mehta
SLAC National Accelerator Laboratory

X-ray imaging is essential at light sources as it directly visualizes the nanoscale structure and dynamics that govern macroscopic material properties. While bulk characterization methods average over a sample, imaging resolves critical spatial heterogeneity—such as defects, domain boundaries, and strain fields—allowing for a direct correlation between local structure and function. Essentially, imaging capabilities transform the light source from a powerful probe into a true multiscale microscope, enabling the discovery of mechanisms in everything from battery degradation to biological function.

Lensed imaging provides a crucial real-time, in situ, and operando context. In contrast, lensless imaging bypasses optics limitations to achieve significantly higher nanometer-scale resolution and thus, quantitatively map dynamic changes. Lensless imaging approaches, such as Coherent Diffraction Imaging (CDI), are crucial because they solve the "lens problem" for X-rays, enabling the imaging of samples at the nanoscale, in 3D, and in their natural state, with a resolution that lensed imaging cannot achieve. However, these improvements are at the cost of imaging speed, as iterative algorithms are utilized to reconstruct the sample from the measured diffraction pattern. A single 3D ptychographic or Bragg CDI reconstruction may take anywhere from minutes to many hours on a powerful workstation or even a small cluster. Iterative reconstruction makes lensless imaging slow, laborious, and highly dependent on user expertise. These shortcomings are exacerbated by next-generation light sources, such as the diffraction-limited upgrade of the Advanced Photon Source (APSU) and the high-repetition-rate upgrade of the Linac Coherent Light Source (LCLS-II-HE). This high rate of data production transforms this from an inconvenience to an existential crisis. For instance, after the LCLS-II-HE upgrade, it will be possible to collect a full 3D dataset in seconds or less, while the CDI reconstruction may still take approximately an hour; thus, data generation will outpace data processing by thousands of orders of magnitude. Within a single 8-hour beamtime shift, one can accumulate years' worth of reconstruction work, rendering the data effectively useless.

Additionally, due to the slow reconstruction, there is no real-time feedback to the scientists. They must collect all their data and hope it was good, only finding out weeks or months later. This converts an interactive experiment into a passive

data collection exercise, resulting in an enormous waste of precious beamtime. Finally, for complex experiments like time-resolved (4D) CDI or multi-modal (5D) CDI, the added complexity renders iterative reconstruction completely infeasible.

The CDI community is exploring various approaches to address this issue. For instance, the massive parallelism of CDI algorithms is a perfect match for hardware acceleration (using GPUs). Similarly, edge and real-time computing improve the delay in real-time feedback. On the algorithmic front, direct inversion using Machine Learning has benefits in reconstruction speed, particularly with U-Net-like architectures. However, this is limited by the resolution of the reconstruction, the generalizability, and the need to generate a vast (and diverse) amount of labeled data for training.

We are developing an innovative approach that combines the reconstruction speed of neural networks with the reliability of physics-based methods. Our key innovation is the embedding of wave propagation physics directly into neural network architectures through differentiable physics simulators, within an uncertainty-aware probabilistic framework. This ensures that reconstructions remain consistent with physical laws even when trained on limited, unlabeled datasets. Our framework explicitly incorporates physical symmetries and invariances that X-ray scattering obeys. This ensures robustness and generalizability of the models beyond the training datasets. Furthermore, we incorporate uncertainty quantification to provide confidence intervals on predicted reconstructions. This provides reliability and trust in the data-driven reconstruction.

We address the physical modeling of noise distribution, physics-based training losses, as well as physics-constrained Generative Adversarial Networks (GANs) for providing highresolution reconstructions. Finally, we utilize attention-based architectures that can capture global relationships in diffraction patterns, as obligated by physics. This would lead to better reconstruction, as well as Foundation Models that possess the data scaling needed to match the extremely high generation rates from LCLS-II-HE. On its final implementation, this system will provide near real-time reconstructions of coherent diffraction patterns that will maintain physical consistency while offering improved resolution. This will enable scientists to fully leverage the new capabilities offered by LCLS-II-HE for studying heterogeneous material systems critical to technologies such as next-generation batteries and microelectronics.

Large Scale Data Analysis for Serial Femtosecond X-ray Crystallography – a Facility Perspective

Sandra Mous
SLAC National Accelerator Laboratory

The advent of X-ray free-electron lasers (XFELs) and exascale supercomputers presents unprecedented opportunities to study biomolecular structure and dynamics on the atomic level[5]. XFELs generate ultrabright X-ray pulses that enable ultrafast conformational changes in biomolecules at nearphysiological temperatures to be probed in Serial Femtosecond X-ray Crystallography (SFX) experiments. The new generation of megahertz repetition rate XFELs allows for improved temporal sampling and detection of rare, short-lived states. Simultaneously, exascale computing provides the computational power to run large-scale molecular dynamics simulations that complement and enhance interpretation of experimental data.

Integrating these technologies requires addressing several data analysis challenges. These include processing and analyzing massive datasets generated by megahertz XFELs, developing algorithms for real-time data processing and experimental steering, improving force fields for molecular simulations, and creating frameworks to integrate experimental and computational results.

Realizing the full potential of coupling XFELs and exascale computing for large-scale analysis of structural biology data will require interdisciplinary collaboration. By addressing the challenges listed above, researchers can leverage the complementary strengths of advanced X-ray sources and supercomputing to significantly advance our understanding of biomolecular structure and dynamics.

[5]Mous, et al. Structural biology in the age of X-ray free-electron lasers and exascale computing. *Curr Opin Struc Biol* 86: 102808 (2024).

The LCLStream: Multi-Institutional Data Analysis in Heterogeneous Computing Environments

Valerio Mariani[1], David Rogers[2], Cong Wang[1], Ryan Coffee[1], Wilko Kroeger[1], Murali Shankar[1], Hans Thorsten Schwander[1], David Mittan-Moreau[3], Aaron Brewster[3], Tom Beck[2], Frédéric Poitevin[1], Jana Thayer[1]

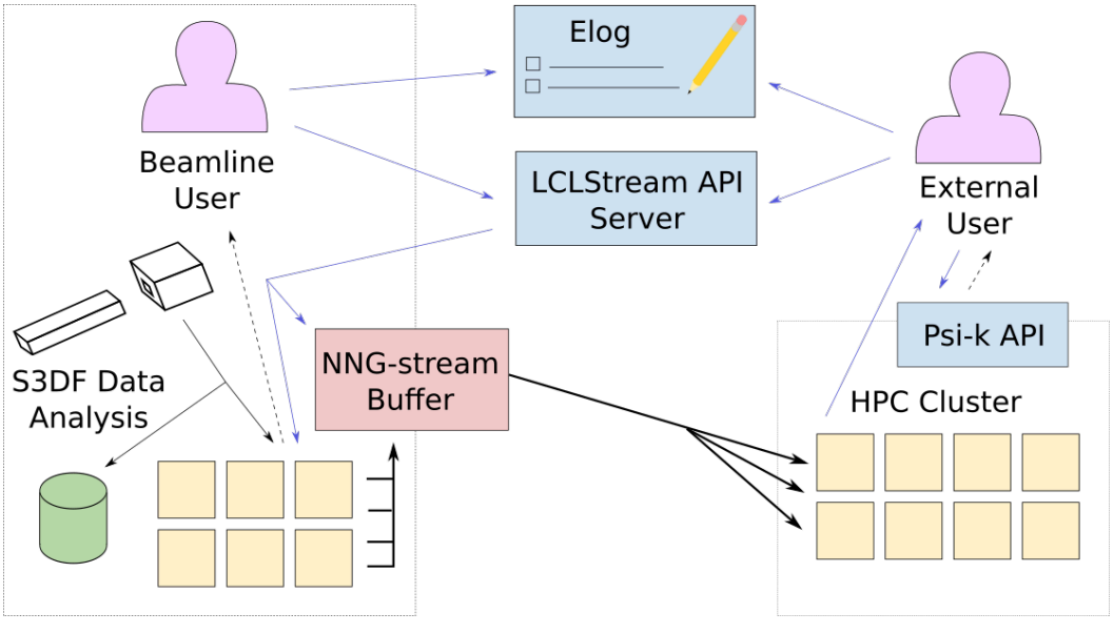
[1] LCLS, SLAC National Accelerator Laboratory , [2] NCCS, Oak Ridge Leadership Computing Facility, [3] LBNL, Lawrence Berkeley National Laboratory
Corresponding author: Valerio Mariani (valmar@slac.stanford.edu)

Recent updates to Free Electron Laser (FEL) light sources, particularly at the Linac Coherent Light Source (LCLS-II and LCLS-II-HE), introduce revolutionary capabilities that will enable novel ground-breaking science and lead to the development of new experimental techniques. However, these sources will generate massive data volumes, up to 1TB/s and millions of diffraction patterns daily, creating significant challenges for analysis, reduction, and storage. Allowing researchers to access the data in a format that is compatible with large-scale data analysis pipelines, tracing a fast path to scientific discoveries and publications, will require the development of new data management tools and techniques. LCLS's local computing resources, located in the SLAC Shared Scientific Data Facility (S3DF), will most likely be insufficient for new computationally-heavy analysis algorithms, especially using modern AI/ML techniques. This will necessitate leveraging external DOE Leadership Computing Facilities. Successfully utilizing these resources, which feature a wide range of hardware and software environments (different CPU/GPU architectures or software stacks), will require exporting data immediately upon collection in formats optimized for downstream analysis workflows.

This presentation will discuss the design and development of a set of integrated services for experimental data collection and analysis that we developed, called CLStream. This new end-to-end experimental data streaming framework is designed from the ground up to support new types of applications – AI training/inference, extremely high-rate X-ray time-of-flight analysis, crystal structure determination with distributed processing, and custom data science applications and visualizers yet to be created. Throughout, we use design choices merging cloud microservices with traditional HPC batch execution models for security and flexibility. By creating a flexible, API-driven data request service, we address a significant need for high-speed data streaming sources for the X-ray science data analysis community. With the combination

of data request API, mutual authentication web security framework, job queue system, high-rate data buffer, and complementary nature to facility infrastructure, the LCLStream framework has prototyped and implemented several new paradigms critical for future generation experiments.

Together, the set of modular services in the LCLStream ecosystem accomplish several objectives that would be impossible with a single, monolithic program: automated data collection, local fast data reduction, data streaming to academic or HPC facilities, long-term data storage for data re-use (e.g. replicating studies or AI model training). Allowing data exploration, cross-facility coordination on co-scheduling of experiments and analysis, web-accessible applications programming interfaces (API-s) and high-bandwidth data transmissions, the LCLStream framework will be crucial in the development distributed high-throughput data analysis pipelines in an heterogeneous computing environment, enabling low latency experimental feedback and even autonomous experiment steering.



Data streaming process diagram. Blue arrows show control paths, and black arrows show data flow. Dotted paths are for returned results. Event assembly and data formatting is performed by the psana framework inside S3DF (left). The LCLStream API can start network buffers and MPI jobs on S3DF to format and send experimental data. External users can pair an LCLStream API call with jobs on other HPC clusters (right).

Scaling "smalldata_tools" for LCLS-II

Vincent Esposito and Silke Nelson
SLAC National Accelerator Laboratory

The upgrade to the Linac Coherent Light Source (LCLS-II) increases the data acquisition rate to up to one million events per second, nearly four orders of magnitude larger than LCLS-I, presenting an immense data processing challenge. While the LCLS Data System team is providing a standardized, real-time Data Reduction Pipeline (DRP), the resulting datasets are still often too large for interactive analysis, requiring any subsequent user-driven processing to be run at scale. We are therefore in the process of formalizing and adapting *smalldata_tools,* a user-centric analysis package, to meet this new imperative. The goal is to provide a flexible framework for custom analysis that hides the complexity of the underlying parallel processing.

The core design principle of smalldata_tools is to act as a high-level abstraction layer over the psana analysis framework, the native interface to the LCLS data format. This insulates the end-users from the need to write low-level parallel code using MPI. Instead, they declaratively define their event-by-event analysis pipelines through a simple configuration file and a library of modular Python functions. These pipelines can perform tasks ranging from basic masking and region-of-interest calculations to sophisticated, custom algorithms.

Once defined, the analysis job is submitted to the SLURM compute cluster via the Automated Run Processing (ARP) on the LCLS experiment portal. Under the hood, smalldata_tools leverages psana to transparently manage the parallel execution across hundreds of cores, handle distributed data access, apply necessary detector calibrations, and aggregate the results into a single, compact HDF5 file.

In addition to this per-event output, the framework features the "cube": a multi-dimensional histogram. Users define a cube by specifying its axes—any combination of experimental variables like motor positions or laser timing—and the data to be accumulated in each bin, such as detector intensity or the output of a custom reduction function. This reduction potentially distills terabytes of raw data into a compact file suited for interactive analysis in Jupyter notebooks.

This talk will describe how the smalldata_tools framework provides a high-level interface for custom analysis, abstracting the complexities of parallel processing for LCLS-II data rates, and highlight the challenges ahead.

The DREAM Data Analysis Framework for Coincidence Momentum Imaging at the LCLS

Xiang Li
SLAC National Accelerator Laboratory

The DREAM endstation is a reaction microscope which can record the coincident momentum vectors of electrons and ions produced from ultrafast atomic and molecular processes. It is one of the new instruments that can take full advantage of the MHz repetition rate of the LCLS-II free-electron laser. This talk will focus on the software development for both online and offline analysis of the DREAM data. It will cover the different measures we have taken to address the data processing challenges presented by the orders-of-magnitude increase in the repetition rate.

Programming Models

Arkouda and Chapel

represented by Brad Chamberlain
HPE

Arkouda

Arkouda is an open-source Python library that was initially developed in 2019 by Mike Merrill and Bill Reus at DoD as a means of permitting analysts to interactively analyze data sets that are larger than can be handled by typical approaches (e.g., Pandas, Dask, Spark). In practice, Arkouda has demonstrated scalability to hundreds of terabytes, thousands of compute nodes, and over a million processor cores. Arkouda's original capabilities focused on a columnar Pandas-like data store supporting a key subset of the NumPy and Pandas APIs for the motivating analyses. Since then, those APIs have been expanded further, and the Arkouda framework itself has been made extensible. New packages that have been developed include support for graph analytics, multidimensional arrays, and sparse matrix-matrix multiplication.

Where most high-performance Python libraries are implemented by calling out to a language like C++, Arkouda achieves its performance and scalability using a client-server model in which most of its computations are performed on a server that may be running elsewhere, such as a supercomputer. Though this server could be written in anything (say, C++ and MPI, or C and SHMEM), Arkouda's developers chose to write it in Chapel for two main reasons: The first is that the original developers were under a deadline and were skeptical that they could complete the work in time using a standard technology; the second is that they wanted the implementation to be something that was approachable for Python users who might want to extend Arkouda's capabilities themselves. Additional characteristics of Chapel that led to its use were its distributed global-view arrays, first-class support for parallelism, performance, portability, and interoperability.

For more information on Arkouda, please see its website (https://arkouda-www.github.io/), this interview with original developer Bill Reus, or the lightning talk and abstract at this workshop, entitled *Interactive, HPC-scale Exploratory Data Analysis in Arkouda: Past Successes and Future Challenges*.

Chapel

Chapel is an open-source language designed to support productive parallel programming from desktops to supercomputers. Chapel supports code that is similarly readable and writable as Python, yet which generates performance that rivals or beats that of conventional HPC technologies. Where HPC typically adopts a new programming notation for each new hardware paradigm (e.g., Fortran/C/C++ for sequential programming, pragmas or libraries for vectorization, OpenMP/Pthreads for multicore, MPI/SHMEM for distributed memory computing, CUDA/HIP/OpenCL/OpenACC/Kokkos/SYCL for GPU programming), Chapel supports all these levels of hardware parallelism using a unified set of features for expressing parallelism and locality/affinity.

Chapel is compiled, portable, and statically-typed. It interoperates with C, Fortran, Python, and other languages using C-based APIs. In practice, Chapel has been used for applications as diverse as astrophysics, hydrological modeling, computational fluid dynamics, quantum many-body physics, satellite image analysis, data analysis/science, branch-and-bound optimizations, and AI. For more information on Chapel, please refer to its website (https://chapel-lang.org) or the Chapel blog (https://chapel-lang.org/blog/).

cuPyNumeric and Legate

represented by Manolis Papadakis
NVIDIA

Motivation

Computational problems today continue to grow both in their complexity as well as the scale of the data that they consume and generate. This is true both in traditional HPC domains as well as enterprise data analytics cases. Consequently, more and more users truly need the power of large clusters of both CPUs and GPUs to address their computational problems. Not everyone has the time or resources required to learn and deploy the advanced programming models and tools needed to target this class of hardware today.

The Legate/cuPyNumeric project aims to bridge this gap, so that any programmer can run code on any scale machine without needing to be an expert in parallel programming and distributed systems, thereby allowing developers to bring the problem-solving power of large machines to bear on more kinds of challenging problems than ever before.

cuPyNumeric

cuPyNumeric is a multi-node, multi-CPU/GPU array computing library that implements the NumPy API.

NumPy is the de facto standard for array computing in Python, providing a simple and easy-to-use programming model with interfaces that correspond closely to the mathematical needs of scientific applications. With cuPyNumeric, you can take your existing NumPy workflows and seamlessly scale them from a single CPU to a single GPU, and up to thousands of GPUs across a multi-node, multi-GPU cluster, without changing your code. This powerful scaling enables you to focus on your research and discovery, not on complex code modifications for different hardware environments.

Legate

Legate is the underlying programming model that supports cuPyNumeric. It provides a common framework for other distributed libraries developed by our group, that all interoperate seamlessly. End users can also build their own libraries

on top of Legate, or simply use Legate primitives to implement small extensions to cuPyNumeric, in cases where the desired algorithm cannot be precisely expressed using only NumPy operations.

The Legate project is built from two foundational principles:

Implicit parallelism: For end users, the programming model must be identical to programming a single sequential CPU on their laptop or desktop. Parallelism, data distribution, and synchronization must be implicit. The cloud or a supercomputer should appear as nothing more than a super-powerful CPU core.

Composability: Software must be compositional and not merely interoperable. Libraries developed in the Legate ecosystem must be able to exchange partitioned and distributed data without requiring "shuffles" or unnecessary blocking synchronization. Computations from different libraries should be able to use arbitrary data and still be reordered across abstraction boundaries to hide communication and synchronization latencies (where the original sequential semantics of the program allow). This is essential to achieve optimal performance on large-scale machines.

The Legate Data Model

Legate's data model is inspired by Apache Arrow. Apache Arrow has significantly improved composability of software libraries by making it possible for different libraries to share in-memory buffers of data without unnecessary copying. However, it falls short when it comes to meeting two of our primary requirements for Legate:

1. Arrow only provides an API for describing a physical representation of data as a single memory allocation. There is no interface for describing cases where data has been partitioned and then capturing the logical relationships of those partitioned subsets of data.

2. Arrow is mute on the subject of synchronization. Accelerators such as GPUs achieve significantly higher performance when computations are performed asynchronously with respect to other components of the system. When data is passed between libraries today, accelerators must be pessimistically synchronized to ensure that data dependencies are satisfied across abstraction boundaries. This might result in tolerable overheads for single GPU systems, but can result in catastrophically poor performance when hundreds of GPUs are involved.

Legate provides an API very similar to Arrow's interface with several important distinctions that provide stronger guarantees about data coherence and synchro-

nization to aid library developers when building Legate libraries. These guarantees are the crux of how libraries in the Legate ecosystem are able to provide excellent composability.

The Legate API imports several important concepts from Arrow such that users that are familiar with Arrow already will find it unsurprising. We use the same type system representation as Arrow so libraries that have already adopted it do not need to learn or adapt to a new type system. We also reuse the concept of an Array from Arrow. The `LegateArray` class supports many of the same methods as the Arrow Array interface (we'll continue to add methods to improve compatibility). The main difference is that instead of obtaining Buffer objects from arrays to describe allocations of data that back the array, the Legate API introduces a new primitive called a `LegateStore` which provides a new interface for reasoning about partitioned and distributed data in asynchronous execution environments.

Any implementation of a `LegateStore` must maintain the following guarantees to clients of the Legate API (i.e. Legate library developers):

1. The coherence of data contained in a `LegateStore` must be implicitly managed by the implementation of the Legate API. This means that no matter where data is requested to perform a computation in a machine, the most recent modifications to that data in program order must be reflected. It should never be the client's responsibility to maintain this coherence.

2. It should be possible to create arbitrary views onto `LegateStore` objects such that library developers can precisely describe the working sets of their computations. Modifications to views must be reflected onto all aliasing views data. This property must be maintained by the Legate API implementation such that it is never the concern of the client.

3. Dependence management between uses of the `LegateStore` objects and their views is the responsibility of the Legate API regardless of what (asynchronous) computations are performed on `LegateStore` objects or their views. This dependence analysis must be sound, but can be imprecise, when the cost of precisely tracking dependencies overshadows the benefit of increased task parallelism. This dependence analysis must also be performed globally in scope. Any use of the `LegateStore` on any processor/node in the system must abide by the original sequential semantics of the program

The Legate Execution Model

A library targeting the Legate API will express its parallel work in terms of "tasks", i.e. specially annotated functions that Legate will execute in parallel on different pieces of a global array.

The library does not have precise control over how the data is partitioned, how many processors are used for a task's execution, or how the computation is distributed across those processors. Rather, the runtime is in control of these decisions, and can use this flexibility to optimize for overall system utilization. To enable this, tasks should be written to work under any degree of parallelism, i.e. they must be "scale-free".

Even though it is undesirable to provide full control over partitioning, it is common that a task will not work correctly under all possible partitionings. For example, an elementwise binary operation will require its input and output arrays to be partitioned in the same way. Legate's solution for this problem is to allow tasks to describe desirable shapes of partitions via partitioning constraints. Thus, a library can guarantee correct execution, while leaving enough flexibility to the runtime.

Partitioning constraints, when combined with access mode information on task arguments (i.e. marking a task argument as input, output or reduction), is a key feature that enables Legate libraries to compose, despite being internally partitioned. A producer task doesn't need to internally support all the partitionings that a downstream task might need. Conversely, a consumer task doesn't need to know how to handle any possible incoming data partitioning. Rather, the runtime keeps track of how the data is partitioned, and decides when it's necessary (or profitable) to repartition, even when moving data between libraries.

DragonHPC

represented by Colin Wahl
HPE

Introduction

DragonHPC (https://dragonhpc.org) accelerates distributed Python applications and workflows at scale. It is a composable, distributed run-time for managing dynamic processes, memory, and data at scale. It scales to 1000s of nodes, is 2–100× faster for data processing than Ray, 5× faster than RedisAI for batch inference, and can run functions, executables, and parallel applications at 10,000+ tasks per second. One of the most compelling demonstrations of Dragon's capabilities can be seen in real-world scientific workflows that demand both scale and speed.

Highlighted Success Story

A drug discovery workflow developed at ALCF is a concrete example of a Python application that Dragon has scaled to 1000s of nodes. The main loop of the workflow loads in terabytes of compound data, uses a model to predict a binding score, runs an expensive simulation on the topk compounds, and then trains the model on those top-k compounds with the simulation data. Prior to the implementation in Dragon, each step in this loop wrote intermediate results out to the filesystem, which greatly limited performance and required re-reading the entire dataset every iteration. Using Python's multiprocessing API, for which Dragon provides a multi-node backend implementation, and the Dragon distributed dictionary, a Python-like dictionary that shards data across nodes, the application loop was able to run continuously and keep all the compound data in-memory—both of which were impossible with the previous implementation.

Multiprocessing API

A core feature of the Dragon run-time is the multi-node implementation of Python's multiprocessing API. Multiprocessing was one of the most heavily utilized Python packages at NERSC when they analyzed user workloads[6]. Multiprocessing is also heavily used by other Python packages and is the de facto module used to add

[6]https://proceedings.scipy.org/articles/majora-1b6fd038-010

parallelism within the Python ecosystem. The Dragon run-time adds a new start method, `dragon`, to the multiprocessing API. Simply switching the start method allows code (including libraries) originally written for a laptop to efficiently scale across multiple nodes of a supercomputer. Dragon has done exactly this for a nuclear reactor data analysis pipeline from a team at Michigan State University. Other packages that Dragon has either enabled to run multi-node or provided an alternative backend for running multi-node include: concurrent.futures, Parsl, Pandaral•lel, Joblib, Dask, and vLLM.

Distributed Dictionary

Another important feature of the Dragon run-time is the distributed dictionary. It provides users with a Python-like dictionary that shards key-value pairs across managers. The number of managers per node or even which nodes the managers are placed on can be specified programmatically, providing an easy path for users to manage data locality. Controlling data locality through the dictionary has been critical for applications doing inference on large datasets or where the inference is in a performance-critical path, e.g. AI + simulation workloads. The dictionary has also been used to hold large datasets in-memory while users are interacting with the data. The Dragon team has worked with developers at the Chan Zuckerberg Biohub to use the distributed dictionary as an in-memory Zarr store that can be used with their existing image analysis tools. Using the Dragon based Zarr store reduces latency compared to the filesystem based Zarr store that was extremely reliant on caching to achieve acceptable performance[7].

Dragon Core

All Dragon run-time objects depend heavily upon a few core objects—the most fundamental being Channels. Although very few users will ever program to Channels, it's useful to understand how Dragon achieves its scalability. The Channels API provides an efficient queuelike interface and an organized means of synchronizing and communicating between processes using shared memory. When Dragon is benchmarked against standard multiprocessing there is very little additional overhead. What overhead does exist is necessary to enable off-node communication.

Off-node communication is handled by a set of separate processes called transport agents. The use of a transport agent rather than a library provides a number

[7] https://bit.ly/dragonhpc

of benefits. Foremost is the ability for processes to come and go efficiently, without any need to initialize a communication library for each new process, since transport agents persist throughout the duration of a job. Another advantage is the ability to aggregate small messages across an entire node, rather than only across a single process. This allows for much greater aggregation of small messages, especially as node sizes grow.

Finally, it is worth mentioning where Dragon can run. The launcher can utilize Slurm, PBSPro, or SSH to bootstrap the run-time initially. Once the run-time has started, Dragon makes no other calls to the workload manager. Dragon also offers a Kubernetes helm chart for deployment of the run-time on Kubernetes clusters. The Dragon team is currently working to harden the proxy API to enable Dragon run-times to communicate, providing hooks for bursting from workstations to clusters to supercomputers[8].

[8] https://zenodo.org/records/10115199

Julia

represented by Johannes Blaschke
NESRC

Computational science and data analysis often start with the rapid exploration of data, algorithms, and ideas; followed by deploying the most promising leads at scale. This approach has long been characterized by a division of labor between different programming languages. Exploration is done using high-level, dynamically-typed languages such as Python and R. Their flexible syntax and interactive nature make them ideal for rapid prototyping, data exploration, and expressing complex ideas quickly. However, the performance of these languages is often insufficient for scaling up computationally intensive tasks involving large datasets or complex numerical simulations. Consequently, performance-critical sections of code are frequently rewritten in low-level, statically-typed languages like C, C++, or Fortran, which provide the necessary execution speed at the cost of increased development time and complexity.

However, not all ideas translate seamlessly from high-level to low-level languages; developers (especially those that are not computing experts) are often hampered by steep learning curves and unfamiliar tool chains. This, often referred to as the "two-language problem," is both inefficient and a barrier to entry for many researchers. It requires scientists to be proficient in two distinct programming paradigms and forces them to spend valuable time on rewriting code rather than on their primary research. The Julia programming language was created to solve this problem. Julia is a high-level, high-performance, dynamically-typed language designed specifically for the needs of technical computing. It aims to provide the performance of C and the usability of Python in a single, cohesive environment.

Performance by Design

Julia's central innovation is its ability to achieve high performance without sacrificing high-level syntax. This is primarily accomplished through its use of a Just-In-Time (JIT) compiler based on the widely-used LLVM compiler infrastructure. When a Julia function is run for the first time, it is compiled into efficient, specialized machine code for the specific argument types it received. Subsequent calls to that function with the same argument types can then execute the fast, pre-compiled code directly. Fundamentally, Julia can be thought of as a templating engine for the LLVM compiler toolchain.

This approach combines the benefits of both static and dynamic languages. Like a dynamic language, the developer can work interactively, writing code without specifying types. The compiler infers the types at runtime. However, like a static language, the resulting code is type-specific and highly optimized, leading to execution speeds comparable to C and Fortran. This design allows researchers to write intuitive code and trust that it will be performant, eliminating the need to offload computational bottlenecks to a different language.

A Paradigm for Technical Computing: Multiple Dispatch

A key feature that distinguishes Julia from mainstream object-oriented languages is its use of multiple dispatch. In traditional object-oriented languages like Python or C++, a function call such as `object.method(argument)` is "owned" by `object`, and the specific implementation of `method` that gets called is determined by the type of that object. This is also called "single dispatch" because execution is dispatched to `method`'s implementation solely based on the type of one input (the type of `object`).

In Julia, functions are dispatched based on the types of *all* of their arguments, not just the first. A function is a generic concept, and its behavior is defined by different implementations, called methods, for different combinations of argument types.

For example, a user can define a new number type, such as one that includes measurement units (e.g., `5 kilograms`) with measurement uncertainties. If the programmer then extends the behavior of all standard arithmetic operations (`+`, `-`, `*`, `/`) to include transformation rules for units and uncertainties, then existing functions for arithmetic, trigonometry, and even complex numerical solvers "just work" with units and uncertainties[9]. Note that Julia already has distinct methods for integers, floating-point numbers, and matrices, implemented for many arithmetic operators, allowing mathematical optimizations to be "backed into" the language.

Moreover, in single dispatch languages, extending methods would require modifying either the original object definitions; requiring alterations to existing code. Inheritance does not save us here, as the additional methods would not be available to any code using the original base classes. There are workarounds of course, yet they generally come at unacceptable costs to performance or usability; and in the example above this is impossible in many languages such as C/C++, or Python as numerical data types are usually not *objects*.

Multiple dispatch is therefore a natural fit for scientific and mathematical programming. Coupled with Julia's type system, multiple dispatch allows pro-

[9] https://www.youtube.com/live/kc9HwsxE1OY

grammers to write generic, reusable algorithms that can be extended by anyone to support new custom data types and find that much of the standard library and community-developed packages simply work with it automatically. This makes code more composable and easier to reason about, as function behavior is not hidden within object definitions.

Pragmatism and Interoperability

Julia is a pragmatic language built for real-world scientific work. It acknowledges the vast ecosystems of existing code and provides seamless interoperability with other languages. It can call C, Fortran, and Python libraries directly, without requiring any "glue" code, wrappers, or code generation. A user can import a Python library like Matplotlib or Scikit-learn and use it alongside Julia code as if it were a native Julia library. A lot of this is driven by the LLVM-based compiler toolchain: Julia programs can make native calls to any C-ABI, and Julia code can be compiled to ELF libraries, which can be used like any other C libraries. This allows research teams to adopt Julia incrementally, leveraging its performance where needed while still relying on the extensive libraries and tools available in other ecosystems.

Furthermore, Julia's syntax is designed to be familiar to users of other technical computing languages. It is clean, expressive, and supports mathematical notations that make the code closely resemble the formulas it implements. This lowers the barrier to entry and makes the code more readable and verifiable for domain experts who may not be programming experts.

Conclusion: HPC with Minimal Effort by Putting Science First

Julia presents a modern and compelling approach to technical computing which has the uncanny ability to express high-performance workloads with minimal effort. By combining JIT compilation, a type system based on multiple dispatch, and seamless interoperability, it is a compelling solution to the two-language problem. It provides an environment where scientific software feels like a first-class citizen so that scientists and data analysts can explore ideas, write intuitive code, and execute computationally intensive analyses in a single, unified language. This allows the focus to remain on the research question at hand, rather than on the limitations of the tools.

Lamellar

represented by Ryan D. Friese
Pacific Northwest National Laboratory

For decades the HPC community has relied heavily on MPI+X programming models using C, C++, or Fortran. While powerful and performant, these languages and programming models often come at the cost of developer productivity and safety. The increasing complexity and scale of scientific data analysis such as AI-enhanced simulations and multi-scale modeling, demand not only high performance but also robust, maintainable, and safe code. This has led to a growing interest in modern programming languages and runtimes that can offer a balance between productivity and performance.

As the lead developer of the Lamellar Runtime, I will be representing Rust and Lamellar as a coherent programming model to address these concerns. Rust, as a systems programming language, offers a unique combination of performance rivaling C and C++ with strong compile-time guarantees for memory safety and concurrency. These guarantees eliminate entire classes of bugs—such as data races, dangling pointers, double frees, and free after use—that have historically plagued HPC applications, often at the cost of extensive debugging or runtime overhead. Rust's ownership model and type system empower developers to write correct, concurrent code from the outset, enhancing productivity without compromising on performance. Moreover, Rust's growing ecosystem of libraries and tools provides a modern foundation for building scientific applications, from numerical computing to data visualization.

Building on Rust's strengths, the Lamellar runtime addresses the specific needs of distributed-memory programming for large-scale scientific data analysis. Lamellar is designed to simplify the development of parallel and distributed applications by providing safe, high-level abstractions for distributed data structures, task-based parallelism via active messages, and efficient overlap of communication via asynchronous APIs. At its core, Lamellar aims to bridge the gap between the low-level control required for performance and the high-level ergonomics needed for productivity. Key features include:

- **Ergonomic Distributed Data Structures:** Lamellar offers abstractions like distributed arrays that present a global view of data spread across many nodes. Instead of manually managing data partitioning, buffer packing, and rank-based communication, developers can interact with a single logical data structure. Crucially, Lamellar leverages Rust's type and ownership system to

make this access safe, preventing common memory related issues at compile time.

- **Task-Based Parallelism with Active Messages:** Lamellar employs a task-based model where developers can launch closures—small, self-contained functions—to execute on remote nodes. This "active message" approach is ideal for the dynamic and irregular workloads common in modern data analysis, such as graph analytics or adaptive mesh refinement. It eliminates the need for rigid, matching send and receive calls, dramatically reducing the risk of deadlocks and simplifying the expression of complex communication patterns.

- **Native Asynchronous Execution for Overlapping Work:** Scientific data analysis often involves irregular communication patterns and large data transfers. By deeply integrating with Rust's native async/await syntax, Lamellar makes it trivial to overlap communication and computation, a critical optimization for hiding network latency. This is particularly valuable for workloads with dynamic dependencies or heterogeneous hardware, where traditional synchronous models can introduce significant bottlenecks.

- **Scalability Across Heterogeneous Architectures**: Lamellar is built with portability in mind, supporting a range of architectures from CPU-based clusters to GPU-accelerated systems (via GPU enabled Rust Crates when appropriate). Its abstractions are designed to map efficiently to underlying hardware, ensuring that performance optimizations do not come at the expense of code maintainability or portability—a critical concern as HPC systems grow increasingly diverse.

- **Integration with Rust's Ecosystem**: By building on Rust, Lamellar composes with widely used crates for scientific computing, such as ndarray or nalgebra for numerical arrays and linear algebra, Serde for serialization, and GPU-oriented crates (e.g., cust/cudarc for CUDA, ocl for OpenCL, wgpu for SPIR-V back ends). This enables holistic development: domain logic, data processing, and distributed execution can be combined without reinventing foundational components.

In essence, Lamellar provides a programming model that allows scientists to express *what* they want to compute on distributed data, while the runtime efficiently and safely manages *how* that computation is orchestrated across the

machine. This combination of Rust's language-level safety and Lamellar's high-level distributed abstractions represents a concrete path forward. It offers a way to break out of the decades-old mold by empowering a new generation of scientists and developers to build performant, scalable, and—most importantly—correct and productive software for the grand challenges of our time.