ADIOS 2.9: A Framework to Enable HPC Tools for Extreme Scale I/O, In Situ Visualization, and Performance Analysis

Presenters: S. Klasky[123], N. Podhorszki[1] , A. Gainaru[1] ,Kevin Huck [5], Sameer Shende [5], Caitlin Ross [4]

[1] Oak Ridge National Laboratory, Computer Science and Mathematics Division

[2] University of Tennessee, Knoxville, Department of Electrical Engineering and Computer Science

[3] Georgia Tech, School of Computer Science

[4] Kitware

[5] University of Oregon

https://tinyurl.com/adios-sc2023

Collaborators: Apps, Workflow, Data Management, Reduction, Viz

- **Scott Klasky**
- **Norbert Podhorszki**
- **Qing Liu**
- **Karsten Schwan**
- **Jay Lofstead**
- **Mark Ainsworth**
- **C.S. Chang**
- **Ana Gainaru**
- Hasan Abbasi
- Rick Archibald
- Chuck Atkins
- Vicente Bolea
- Phillipe Bonnet
- Michael Bussmann
- Jieyang Chen
- Hank Childs
- Jong Choi
- Michael Churchill
- Nathan Cummings
- Shaun de Witt
- Philip Davis
- Ciprian Docan
- Greg Eisenhauer
- Stephane Ethier
- Ian Foster
- Dmitry Ganyushin
- Kai Germaschewski
- Berk Geveci
- William Godoy
- Qian Gong
- Junmin Gu
- Jon. Hollocombe
- Kevin Huck
- Axel Huebl
- Toby James
- Chen Jin
- Mark Kim
- Brad King
- James Kress
- S.H. Ku
- Ralph Kube
- Tahsin Kurc
- Xin Liang
- Zhihong Lin
- Jeremy Logan
- Thomas Maier
- Kshitij Mehta
- Ken Moreland
- Todd Munson
- Manish Parashar
- Franz Pöschel
- Dave Pugmire
- Anand Rangarajan
- Sanjay Ranka
- Stefanie Reuter
- Caitlin Ross
- Nagiza Samatova
- Ari Shoshani
- Eric Suchyta
- Fred Suter
- Keichi Takahashi
- William Tang
- Roselyne Tchoua
- Nick Thompson
- Eric Suchyta
- Seiji Tsutsumi
- Ozan Tugluk
- Lipeng Wan
- Ruonan Wang
- Xinying Wang
- Ben Whitney
- Andreas Wicenec
- Matthew Wolf
- John Wu
- Bing Xie
- Fan Zhang
- Fang Zheng

Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

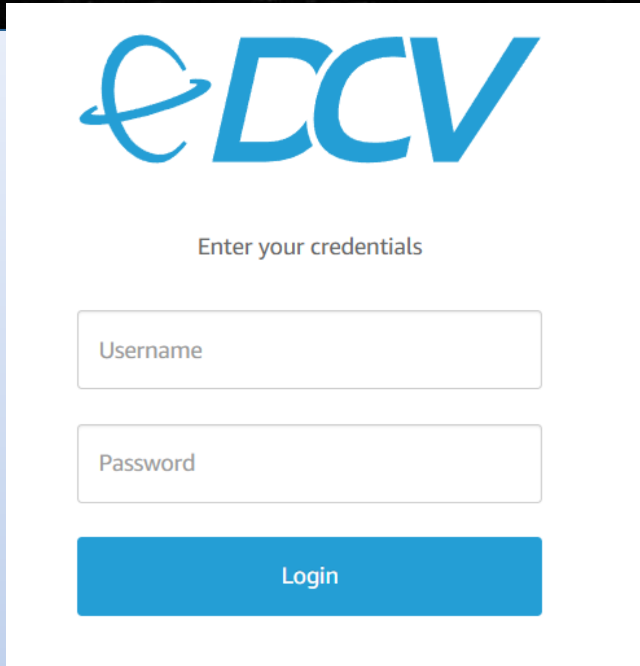
Login to a virtual machine

https://tut###.supercontainers.org:8443/#e4s

is a number assigned for you in this tutorial

Username: tutorial
Password: HPCLinux12!

Launch a couple of gnome-terminals



What should you get out of this tutorial

- Why you should use ADIOS
- What are the advantages of using ADIOS over other parallel I/O and in situ visualization frameworks
- How to use ADIOS
- How to get high performance using ADIOS
- How to use ADIOS for in situ analytics
- How to visualize data with Paraview for in situ and post processing of ADIOS-enabled codes
- How to use TAU with ADIOS and to visualize the TAU-ADIOS files

- What programming language should we focus this tutorial on for you? C++, Python, F90, C
- PLEASE ASK QUESTIONS!

Outline

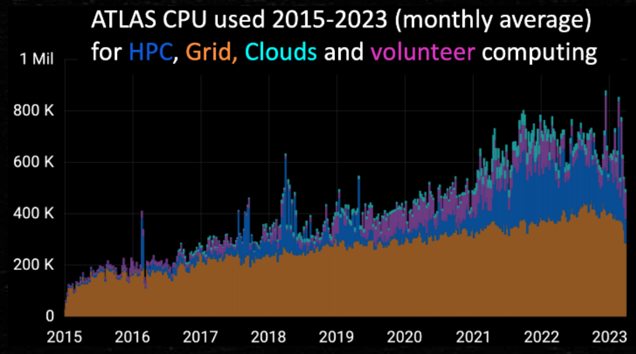
- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

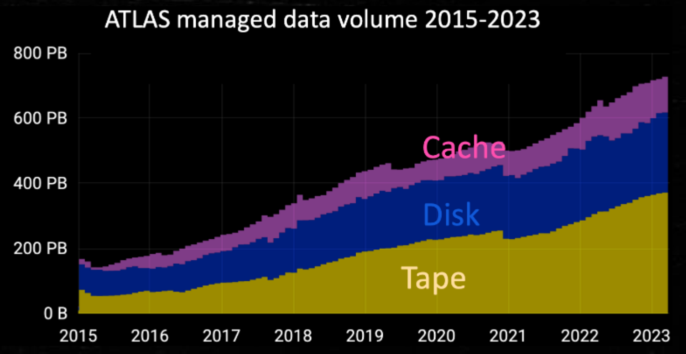
Introduction Outline

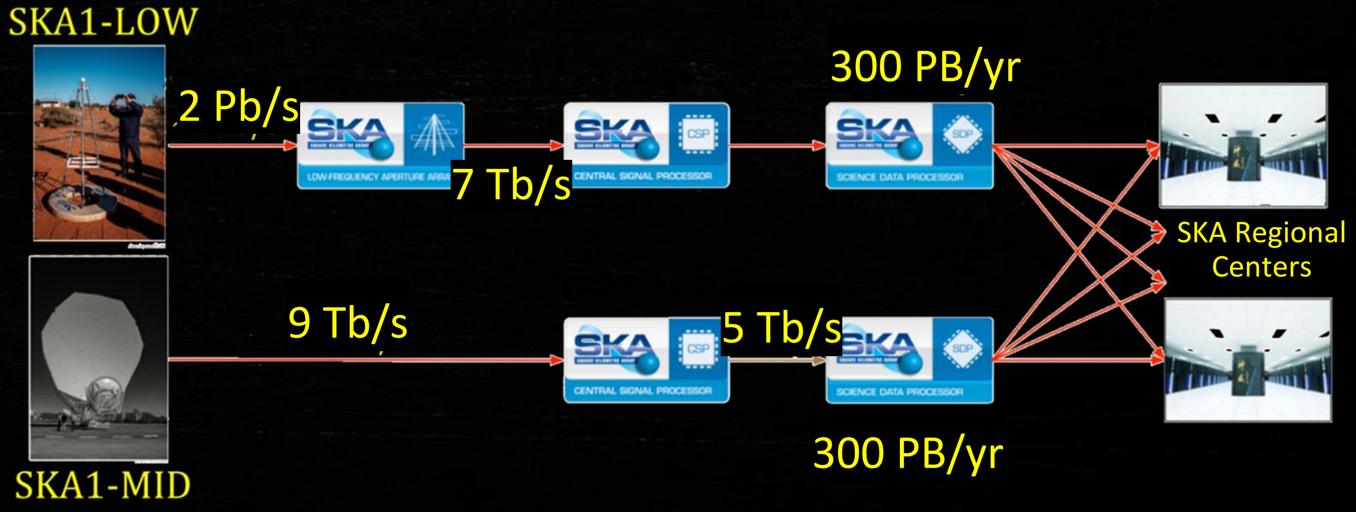
- Vision
  - Next generation workflows to drive our R&D
- Application success stories
- Data Reduction
- Future Steps

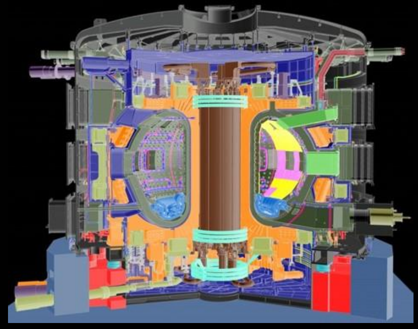
The data deluge: The data V's are growing exponentially

- Volume – High-Luminosity LHC
  - Current storage model doesn't scale
  - Optimized storage model remains too expensive
- Velocity – Square Kilometer Array
  - Unprecedented and sustained throughputs
- Variety – ITER
  - Dozens of diagnostics (cameras, spectrometers, bolometers, sensors and probes) with long pulses
- Value – The cost of these facilities are in the billions of $'s









Supercomputing changes in the last 30 years

1988: Cray Y-MP: 2.7 Gflops: Vector Processors, SSD storage (13.6 GB/s)

1996:Cray T3E: 0.001 PFlops, massively parallel

1998: 0.0025 Pflops,ASCI Blue Mountain: shared memory across all procs

2002: Earth Simulator 40 Tflops, 50MW, vector procs

2009: Cray XT5: 2.5 Pflops: Multi-core, LUSTRE storage system

2013: Cray Titan: 27 Pflops , NVIDIA GPUs

2018: IBM Summit: 200 Pflops , NVIDIA GPUs

2022: Cray Frontier: >1 Eflops , AMD GPUs, Burst Buffer Storage 10TB/s, long term at 4.6 TB/s, 21MW

Observations:

- Ratio of Storage/Flops keeps getting worse
- New complex workflows with AI + HPC
- Experimental/Observational data is outpacing compute & storage

Foster, I., Klasky, S., et al., (2017, August). Computing just what you need: Online data analysis and reduction at extreme scales. In European conference on parallel processing (pp. 3-19). Springer, Cham.

Motivation

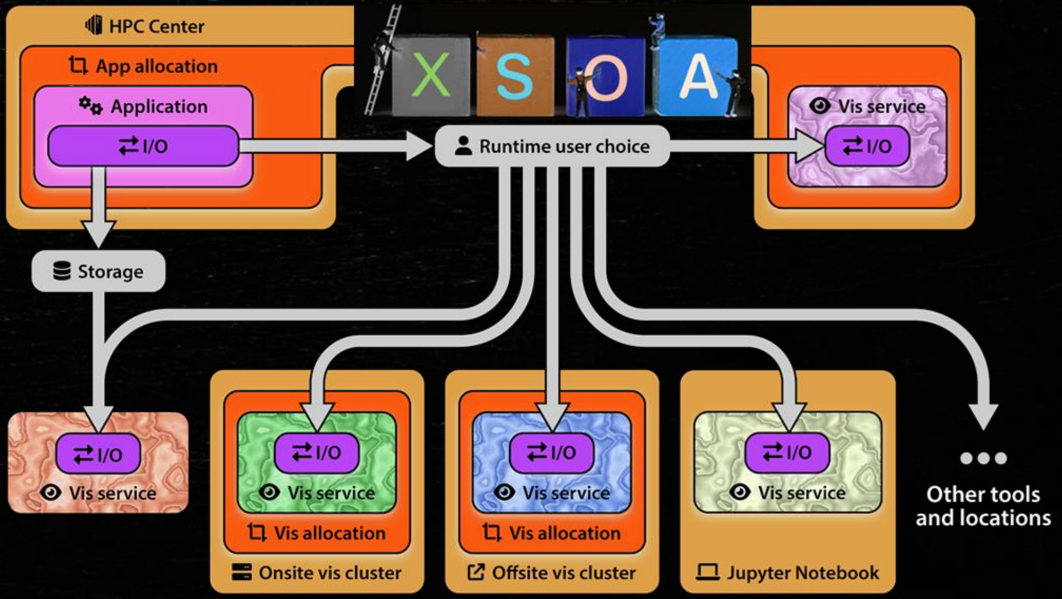
Working with domain scientists who have the following challenges: I need to

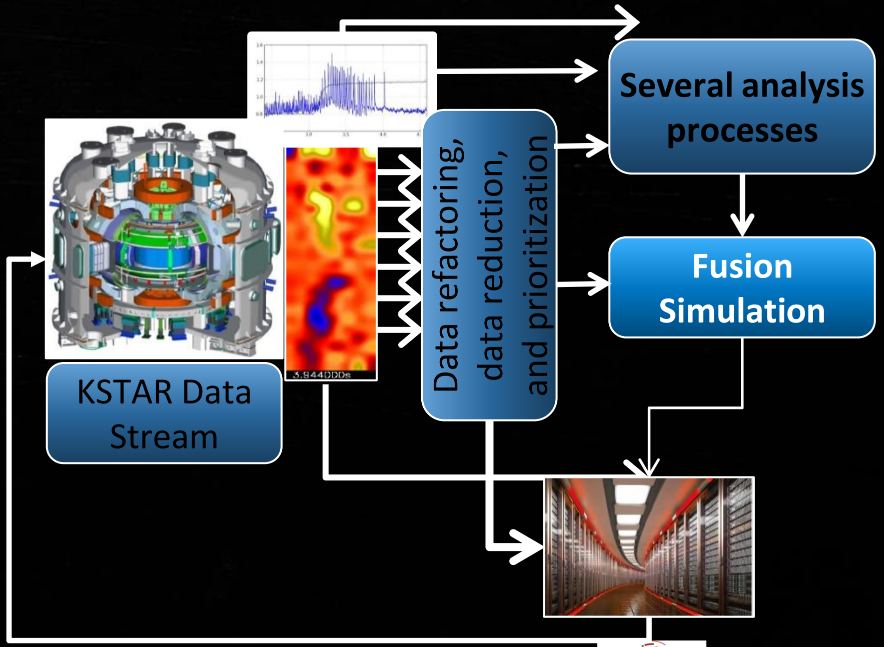
- Write a few times during my simulation, and can you help me write a little more often
  - Time decimation is ok until important physics is lost, but they want ~ 5% overhead for I/O
- Understand/visualize data at a higher time fidelity during the experiment
  - Near Real Time visualization is necessary for many domains
- Couple several codes together and integrate analytics during the experiment
  - Coupling applications with analytics is critical as the science becomes more complex
- Reduce the data sizes which go on tape and move across storage tiers
  - Data Reduction with quantifiable error for the QoIs is critical to reduce storage/network
- Reduce the analysis cost as my datasets grow
  - Remote access to data with different error bounds is critical for working with large data
- Compose and execute the coupled codes on HPC resources

Our vision: creating a pub/sub system for high performance SDM

Our vision is to allows applications to publish and subscribe to data

- With no modifications, any code can tap into the I/O system
- Data can stream in a "refactored" and filtered/queried in a manner allowing the "most important" information to be prioritized
- Data written and read to storage will be highly optimized on HPC resources and queryable in a federated system

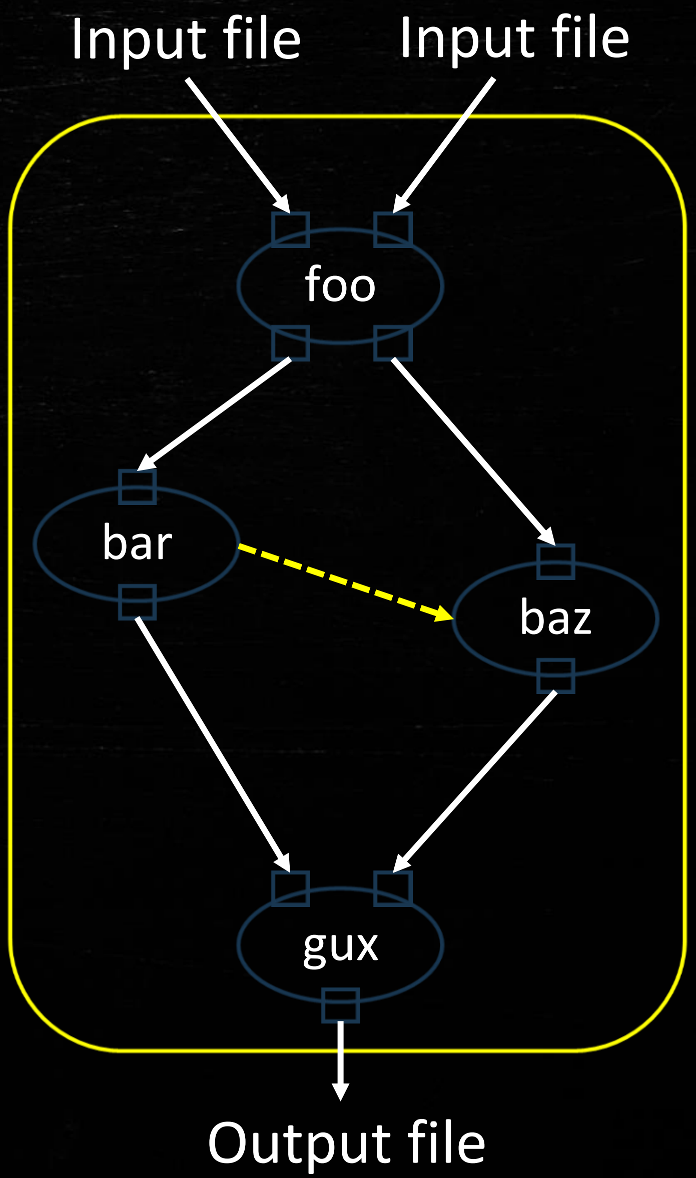




D. Pugmire, J. Huang, S. Klasky, and K. Moreland: The Need for Pervasive In Situ Analysis and Visualization (P-ISAV), WOIV'22.

Traditional scientific workflows

- Direct Acyclic Graphs (DAGs)
  - Tasks: Functions, standalone kernels
  - Data: file-based transfers
  - Dependencies: Flow (→) or control (⇢)
- 1 Workflow = 1 Application
  - Well defined structure
  - Full interoperability between components
  - No intrusion in kernel codes
  - Evolve as a whole

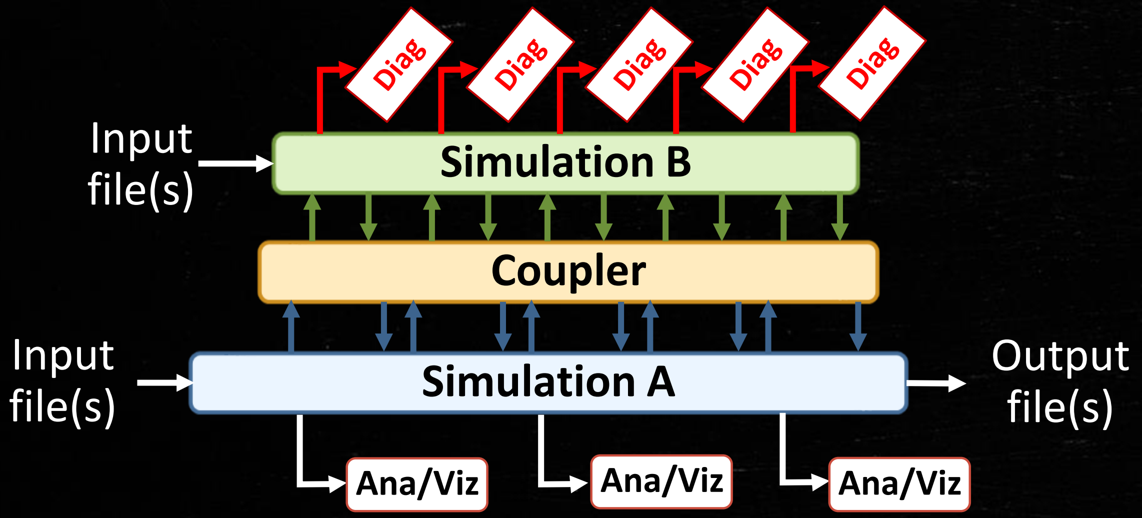


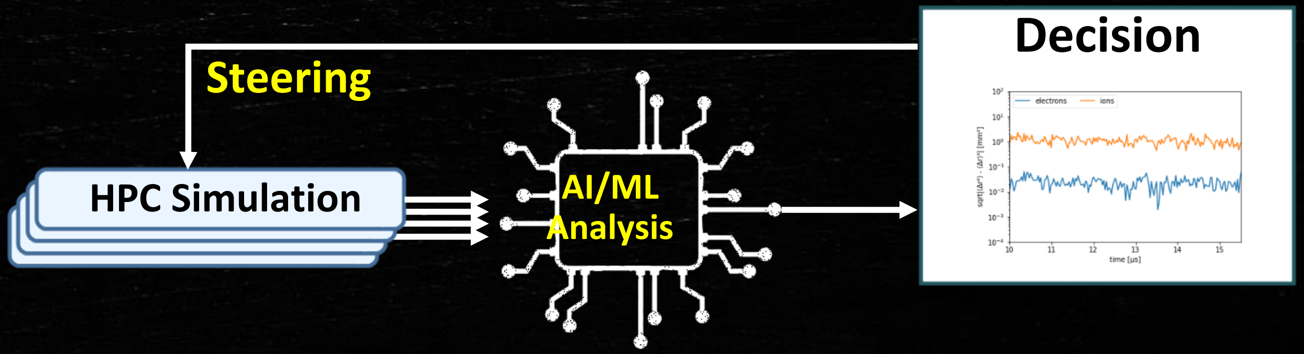
Traditional Workflow Management Systems and limitations

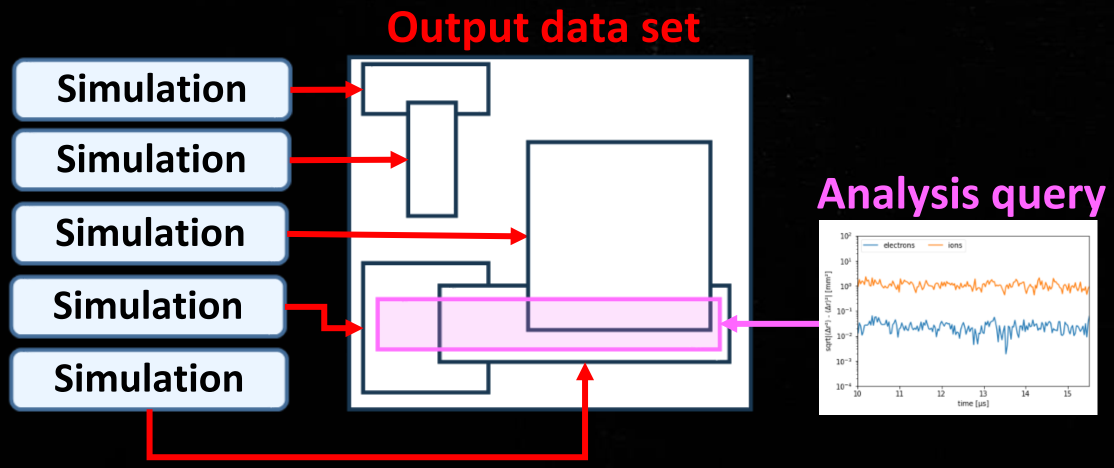
- Composition
  - Provide ways to express the structure of a workflow and describe the components
    - Rigid structure, no cycles, no tight coupling
- Orchestration
  - Plan, deploy, and launch the execution of the components of a workflow
    - Planning is often static with a disconnect between theoretical and practical scheduling and practical implementations
- Data management
  - Manage the storage and transfer of files and handle the intermediate data produced, consumed, and transferred
    - Everything is file-based; read/write/move files
- Code management
  - Deploy/install the code of the workflow components on all systems
    - Burden of compatibility of components is on the WMS

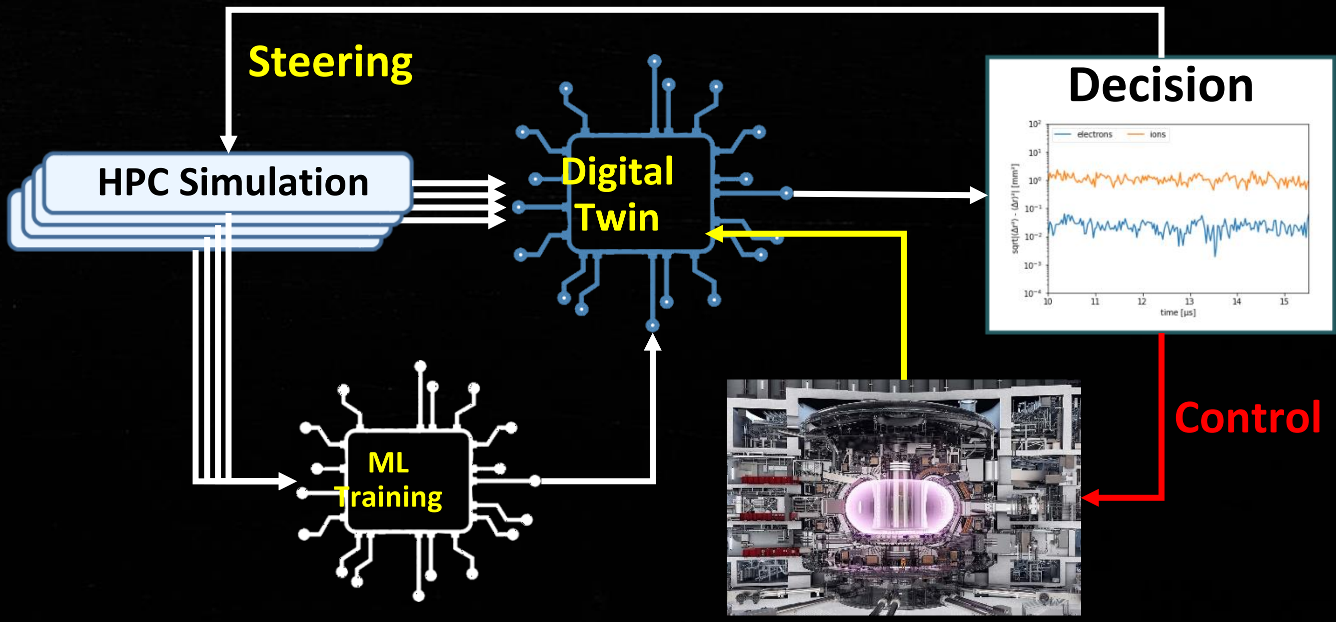
https://s.apache.org/existing-workflow-systems

Moving towards next generation workflows









Next Generation workflows need explicit data movement & command/control

Need an I/O framework that enables the next-generation WMS

Requirements

- Self-Describing data which can be organized in hierarchies
- Fast Parallel and Serial I/O to/from all tiers of storage (LUSTRE, GPFS, AWS, NFS, BGFS, TAPE, …) at all scales
- No changes to work over streams (HPC, WAN)
- Ability to get high performance for older and newer storage technologies
  - Tape, Parallel File System, Laptops, NVMe, Object Storage, next generation NVMe (e.g. Samsung storage)
- Ability to efficiently query the objects
- Can be used as a well-defined interface for a Service Oriented Architecture (files/streams)

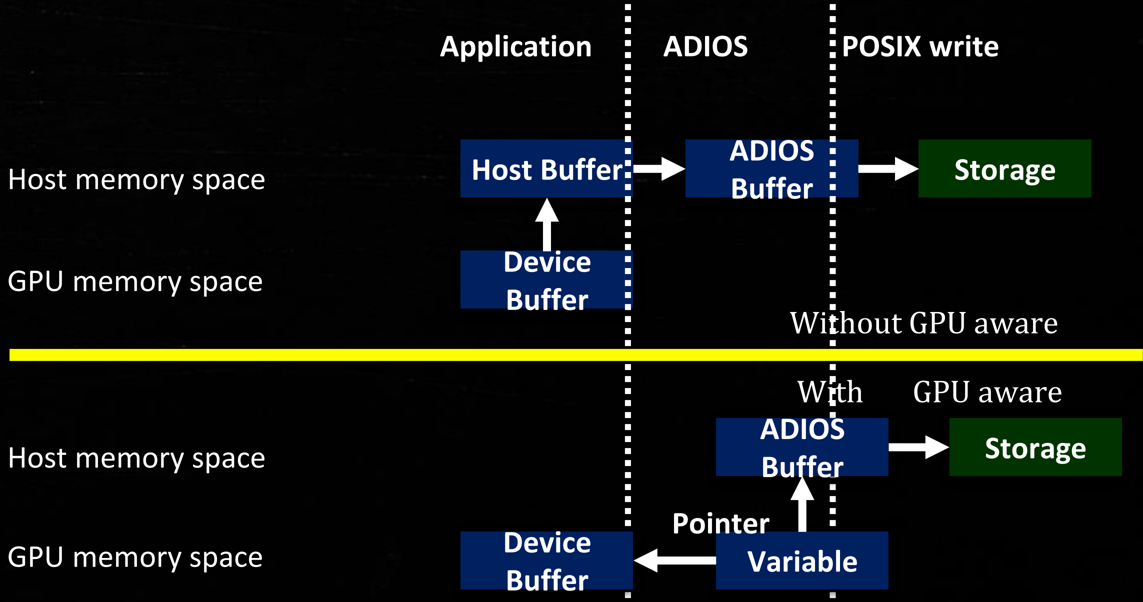
ADIOS: high-performance publisher/subscriber I/O framework:

Vision

- Create an easy-to-use, high performance I/O abstraction to allow for on-line/off-line memory/file data subscription service
- Create a sustainable solution to work with multi-tier storage and memory systems for self-describing data-streams

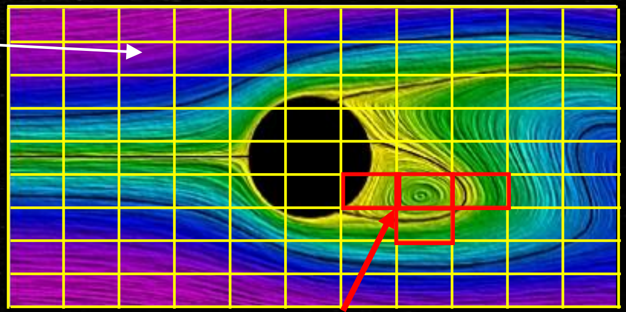
Details

- Declarative, publish/subscribe API separated from the I/O strategy
- Multiple implementations (engines) provide functionality and performance
- Rigorous testing ensures portability
- GPU-aware to reduce data movement
- https://github.com/ornladios/ADIOS2



Godoy, W. F., Klasky, S. ,et al. (2020). ADIOS-2: The adaptable input output system. a framework for high-performance data management. SoftwareX, 12, 100561.

Querying Large Scientific Data Sets



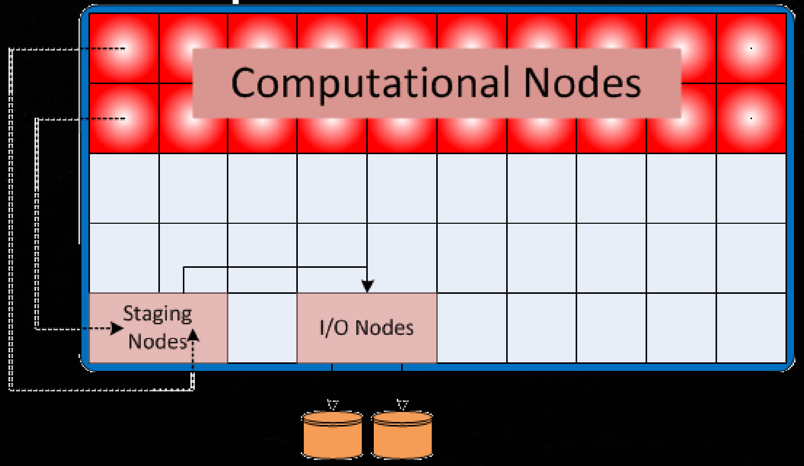
- ADIOS writes metadata for each variable on each "chunk" of data (min/max, + optional variance,…)
- ADIOS supports range queries on each/multiple variables and only reads/moves the chunks which have data in the range of the query
- Next-generation queries can allow "new variables defined and queries when writing/reading", without increasing storage cost
  - E.g. define vecB=vector(BX,BY,BZ); define curlB=curl(vecB);define Bmag = mag(vecB)
  - Get (curlB), get(Bmag, Bmag>5.5)
  - These next-generation queries allow the users to only read data of the derived quantities which satisfy the queries
- Take advantage of next generation storage (e.g. Samsung) includes compute at storage

Queries can reduce the data movement from storage to applications

Gu, J., Klasky, S., Podhorszki, N., Qiang, J., & Wu, K. (2018, March). Querying large scientific data sets with adaptable IO system ADIOS. In Asian Conference on Supercomputing Frontiers (pp. 51-69). Springer, Cham.

Introduction to staging

- Simplistic approach to staging to decouple application performance from storage performance (burst buffer)
- Built on past work with threaded buffered I/O
  - Buffered asynchronous data movement with a single memory copy
  - Application blocks for a very short time to copy data to outbound buffer
- Exploits network hardware for fast data transfer to remote memory
- Today's "modern" solution is to put burst buffers on the HPC resource



Abbasi, H., Klasky, S., et al., (2010). Datastager: scalable data staging services for petascale applications. Cluster Computing, 13(3), 277-290.

Staging Options

Transfer mechanisms

- File based (BP4, BP5)
- Network based on the same resource (SST, SSC)
  - RDMA (libfabric, UCX)
  - MPI (one sided, two sided)
  - TCP/ RUDP
- Memory references
- WAN data transfer (DataMan,SST)
  - Files – GridFTP, scp, …
  - Streams – TCP, RUDP, RoCE

Placement options

- Same core
- Different cores/same node
- Different nodes
- Different resource (LAN)
- Different resource (WAN)
- Hybrid (mixture of options)

Scheduling options

- Fully synchronous
- Fully asynchronous
- Hybrid

Refactoring options

- Prioritize which data gets moved first

Storage options

- ADIOS-BP5
- HDF5

Gainaru, A., Klasky, S., et al., 2022, Understanding the Impact of Data Staging for Coupled Scientific Workflows , IEEE transactions on parallel and distributed systems. 2022.

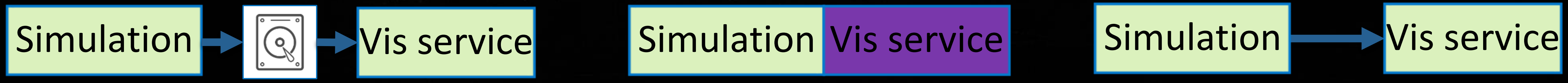
Visualization services with Paraview/Catalyst/FIDES/VTK-M/ADIOS

- Fides is a visualization schema
- Fides maps ADIOS data arrays to VTK-M datasets, enabling viz using shared and distributed-memory parallel algorithms
- Catalyst provides in situ data analysis and visualization capabilities for ParaView

- Interactive visualization in a GUI – post processing
- Batch visualization – post processing on compute nodes
- In situ (inline) interactive visualization in a GUI
- In situ (inline) batch visualization
- In situ (in transit) batch visualization

Using the same application that outputs data using ADIOS

Post hoc **In situ (inline)** **In transit**



BP5, HDF5

ParaViewADIOSInSituEngine

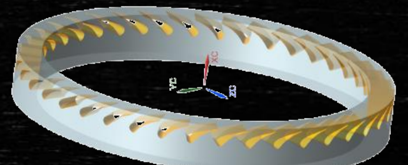
SST, SSC staging and BP5 through file system

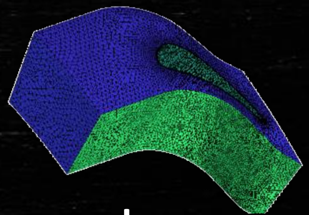
Pugmire, D. , et al. “Fides: a general purpose data model library for streaming data.” ISC 202, Springer International Publishing, 2021.

Outline

- Vision
    - Next generation workflows to drive our R&D
- Application success stories
- Data Reduction
- Future Steps

Wind Turbine optimization: General Electric



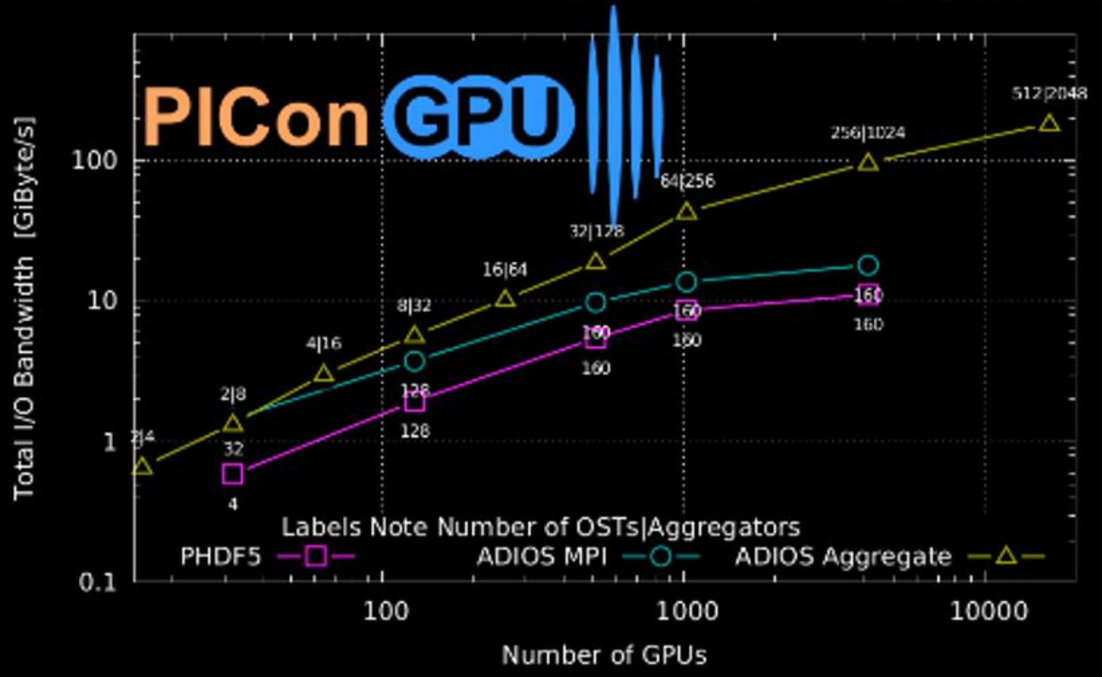


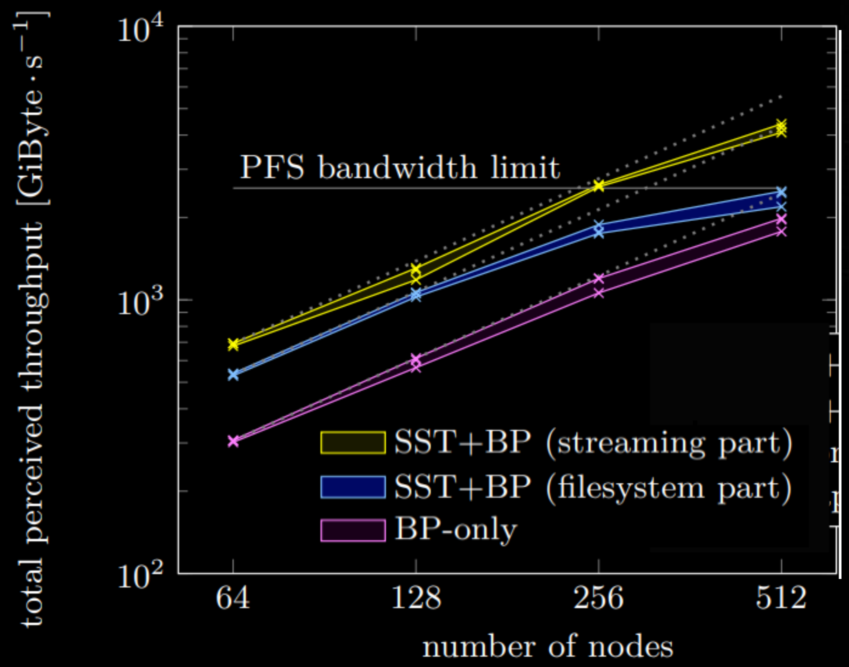
GE is creating next generation simulations to optimize the efficiency of wind turbines and execute these on the OLCF Summit and Frontier resources on over 1K nodes for 20 hours

- They generate 1.4 TB data per output step on 6K GPUs on Summit and want to write data at every 100 steps (300 wallclock seconds)
  - They used CGNS with HDF5 for “efficient” I/O, but because of the size it takes 2700s/step
  - Converting the data to ADIOS-2 BP5 reduced the time to 6s (2% overhead)
- The total output data is now 336 TB, and they need to move it to GE HQ but it’s too much data to store at GE
  - Using MGARD in 1D we can compress the data >10X with 99.999% accuracy for the QoI in the turbulent regions
- New R&D is still necessary to reduce the data to >100X
- New R&D is still necessary to reduce the analysis time with Ensight/Paraview

Accelerator Physics: PIConGPU

- We originally helped PIConGPU to achieve I/O performance on the OLCF Jaguar system
  - Allowed their code to get >10X I/O performance improvement
- Next, we utilized staging to increase the I/O bandwidth on Summit
- Now we are working on more in situ techniques for AI, Digital Twins

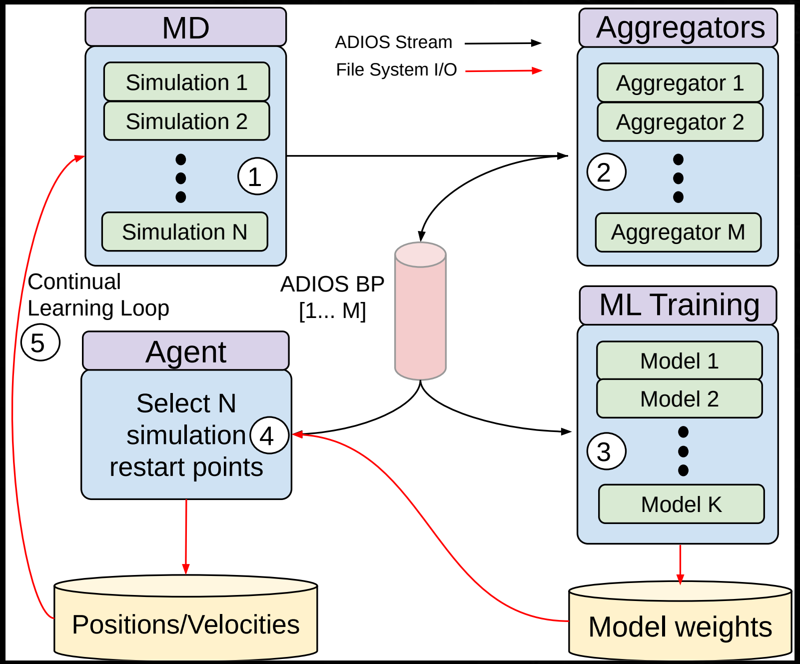


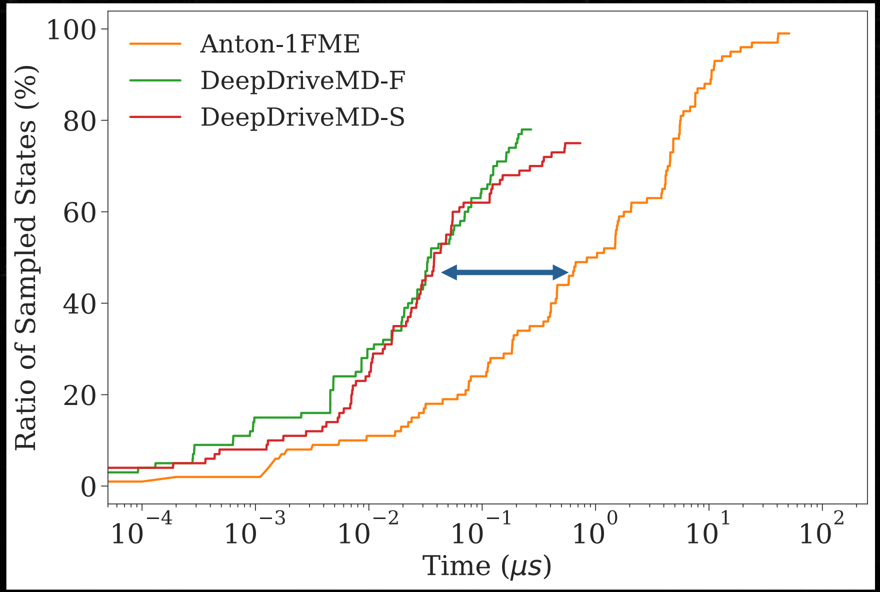


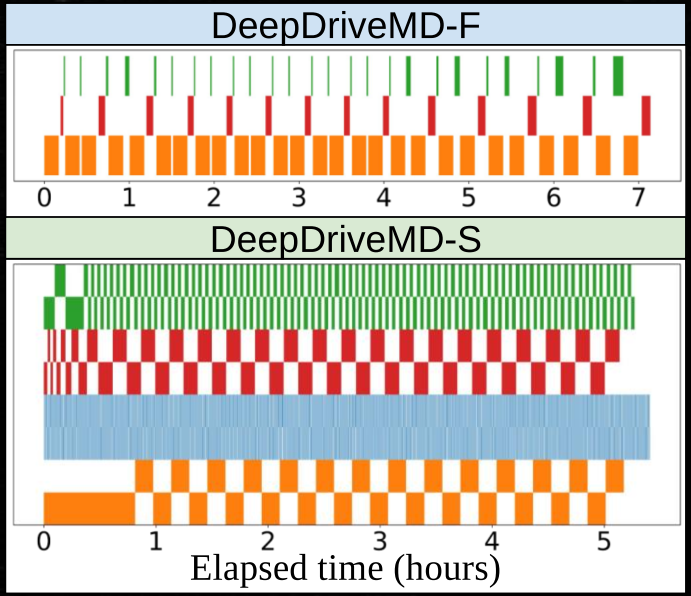
Fusion: GTC

- Our original goal was to reduce the I/O time in parallel I/O (2000) using parallel HDF5
  - 2.3 GB of data on 1024 processors, output every 200 wallclock seconds on NERSC IBM SP2
  - Parallel HDF5 was optimized to have a 37% overhead; 74 seconds/write for 2.3 GB
  - ADIOS 0.0.1 was used; 8 seconds/write for 2.3 GB
- ADIOS 1.0 was used in GTC on OLCF Cray XT4
  - ADIOS was optimized to write >20 GB/s
- ADIOS 2.0 was used in GTC on Summit writing 2.4 TB/step at 2.5 TB/s (< 1 second/step)
- ADIOS 2.X with in situ visualization has been integrated into GTC with EFFIS to allow for in situ analysis and visualization.

DeepDriveMD: enhancing the scalability for streaming AI runs







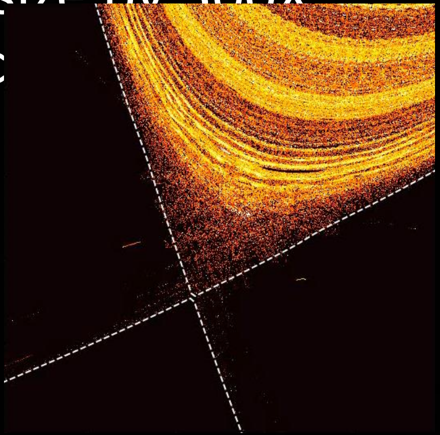
- DeepDriveMD-S: streaming implementation with ADIOS-BP
- Continual learning loop enabled by ADIOS constructs
- Streaming application of ML/AI
- Streaming runs have better resource utilization for protein folding simulations than static file system-based runs

- At least 2 orders of magnitude (100x) acceleration in sampling conformational states related to protein folding
- Faster time-to-solution enabled by streaming runs

Fusion: XGC

The XGC fusion code was created to understand the turbulent structures in the edge of the plasma

- The output data written as a PDF is written every step (35s), but is 1 TB for an ITER run
  - The original output (PHDF5) took over 1000 seconds/step
  - Converting to ADIOS2-BP5 results in <1 second/step
- The total output during a 20hr run is 2 PB which is too large to store for a long time and to move to PPPL
  - New techniques with MGARD2 + Post processing allow us to reduce the size by 300X while maintaining errors with 10^{-8} accuracy on 5 of the QoIs, but new tec remotely access data by accuracy/variable/step/ROI is still critical

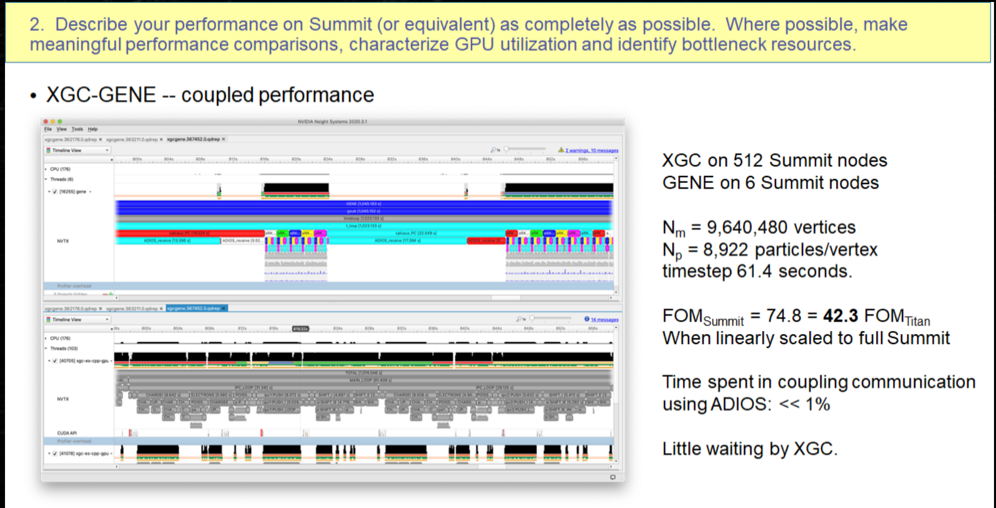


- In situ visualization techniques for Fast Poincare surface plots help understand important physics (homoclinic tangles)
  - GPU optimized VTK-M services for Poincare plots speed up the analysis from 1 hr to 60s using 1 Summit node

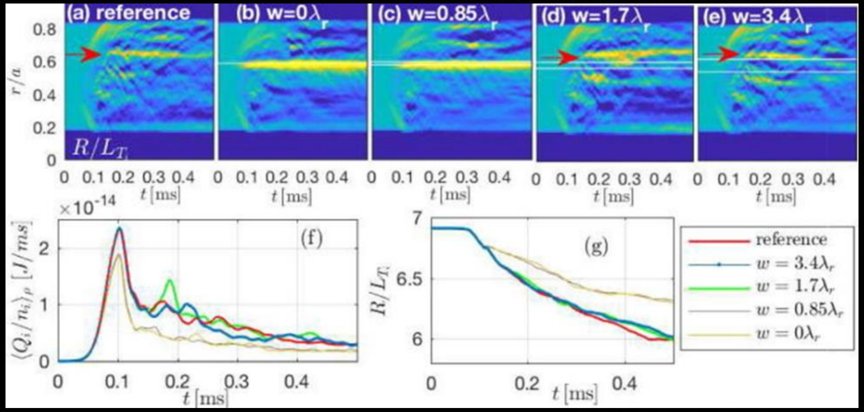
High-Fidelity Whole Device Modeling of Fusion Plasmas

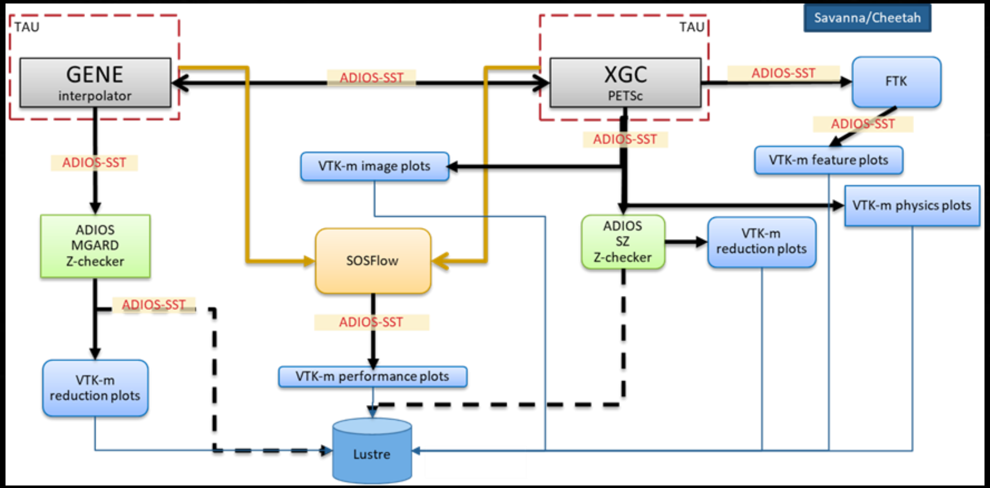
PI: Amitava Bhattacharjee, PPPL,
C. S. Chang, PPPL

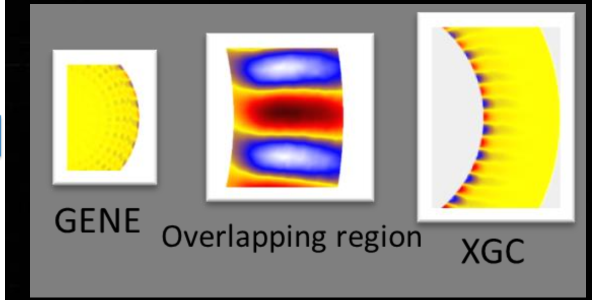
- Different physics solved in different physical regions of detector (spatial coupling)
- Core simulation: **GENE**
  Edge simulation: **XGC**
  Separate teams, **separate codes**
- Recently demonstrated first-ever successful kinetic coupling of this kind
- Data Generated by one coupled simulation is predicted to be > 10 PB/day on Summit



From FY21 WDMApp Review



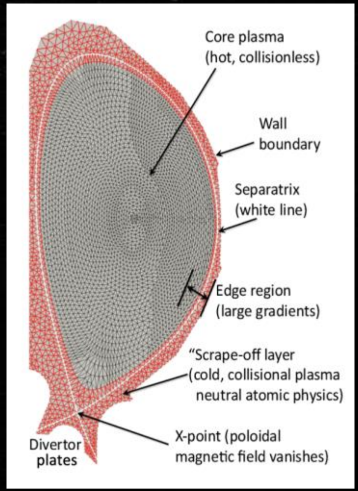


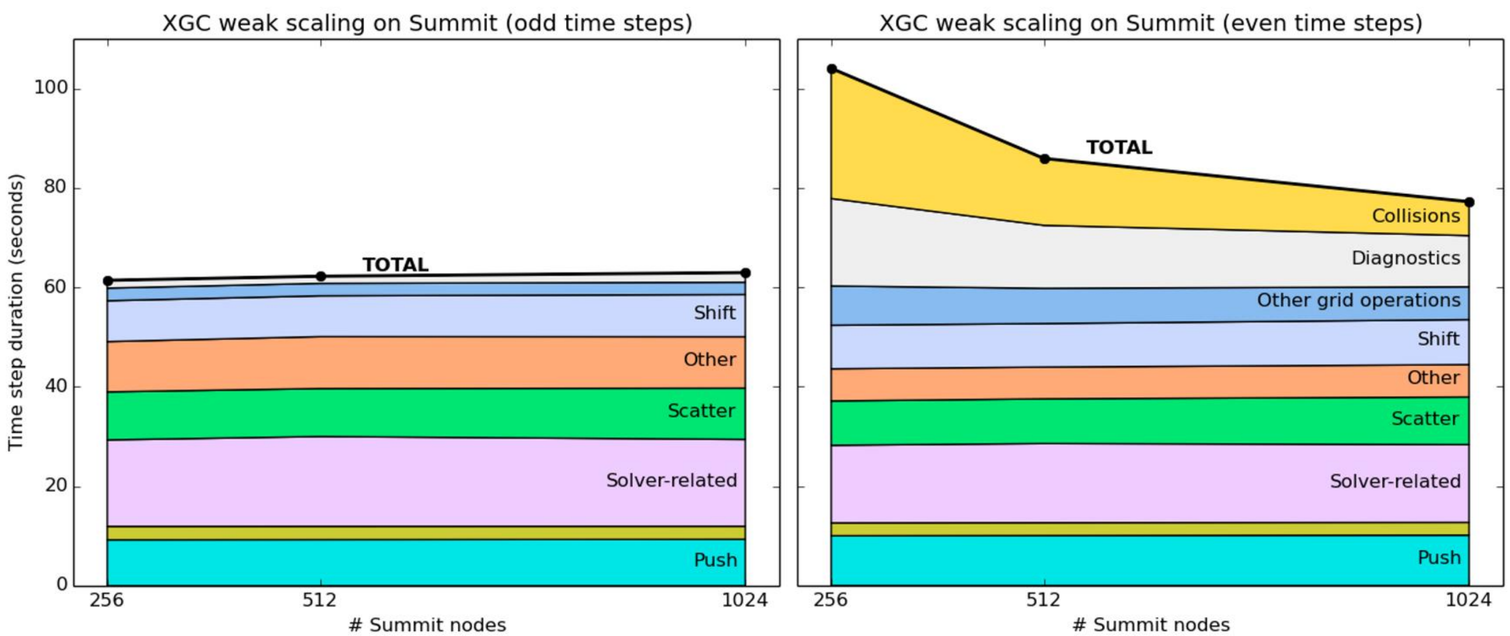


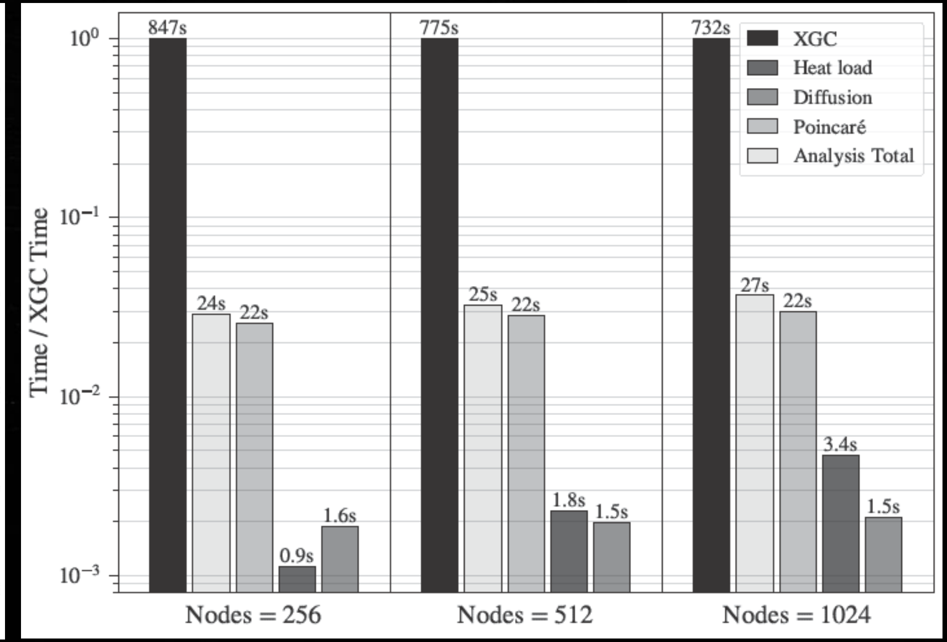
Dominski, J., et al. "Spatial coupling of gyrokinetic simulations, a generalized scheme based on first-principles." Physics of Plasmas 28.2 (2021): 022301.
Merlo, G., et al. "First coupled GENE–XGC microturbulence simulations." Physics of Plasmas 28.1 (2021): 012303.
Cheng, Junyi, et al. "Spatial core-edge coupling of the particle-in-cell gyrokinetic codes GEM and XGC." Physics of Plasmas 27.12 (2020): 122510.

Hybrid Analysis of Fusion Data for Online Understanding of Complex Science on Extreme Scale Computers

- We examine a complex workflow using XGC on Summit, with three in situ analysis for new scientific discovery
- We execute XGC along with three analysis routines (Poincaré surface plot, Head Load calculation, Diffusion Calculation)
- The overhead was 0.1% 1/1024 nodes



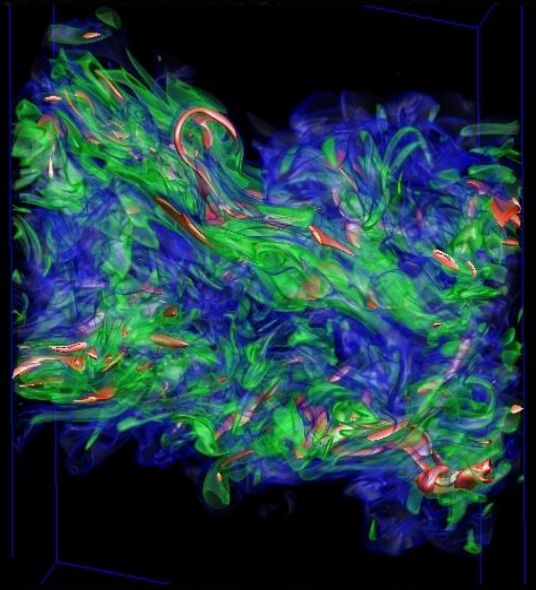




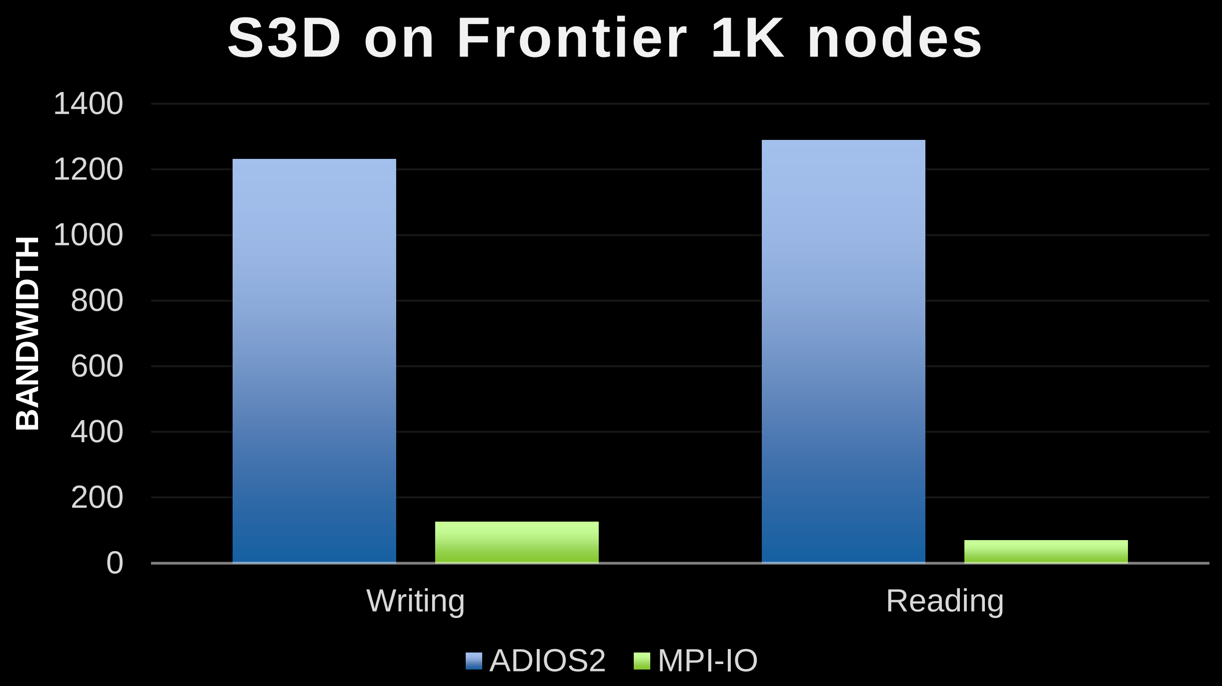
Suchyta, E., Klasky, S. , et al., Hybrid Analysis of Fusion Data for online understanding of complex science on extreme scale computers, Cluster 2023.

Combustion: S3D (most recent progress)

- On Frontier, S3D researchers (Jackie Chen et al. SNL) were bottlenecked from their I/O and wanted help to reduce their data
- Step 1: Reduce the I/O cost
- Step 2: Reduce the datasize but ensure the QoIs from ML feature detection is accurate and place data in community storage for FAIR data

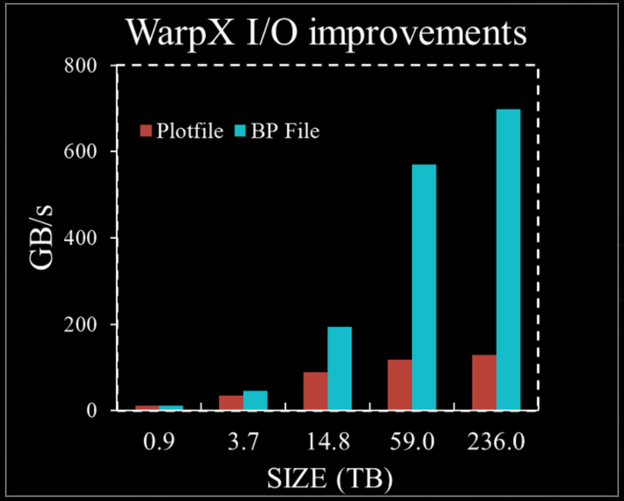


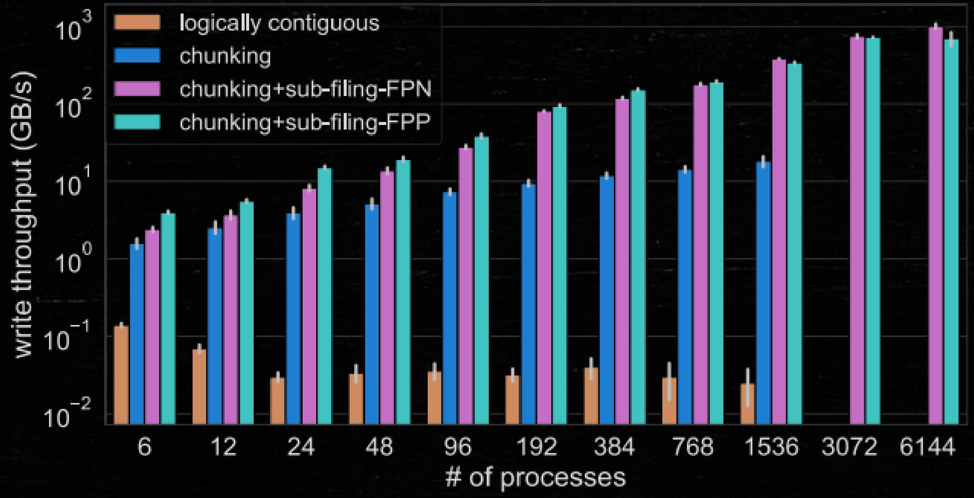
Vorticity fields in DNS of a turbulent jet flame: Visualization: K. L. Ma, H. Akiba



WarpX code

- WarpX is a PIC code with Adaptive Mesh Refinement using AMReX





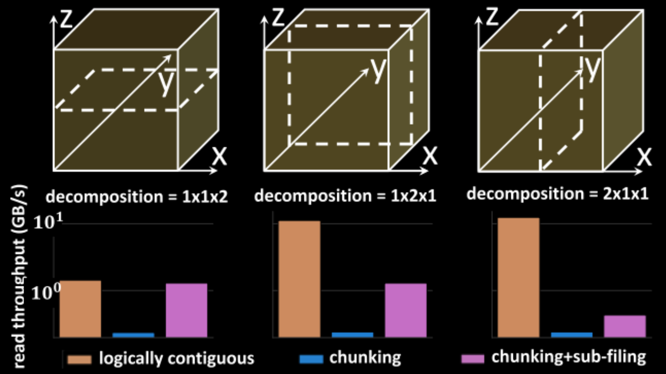
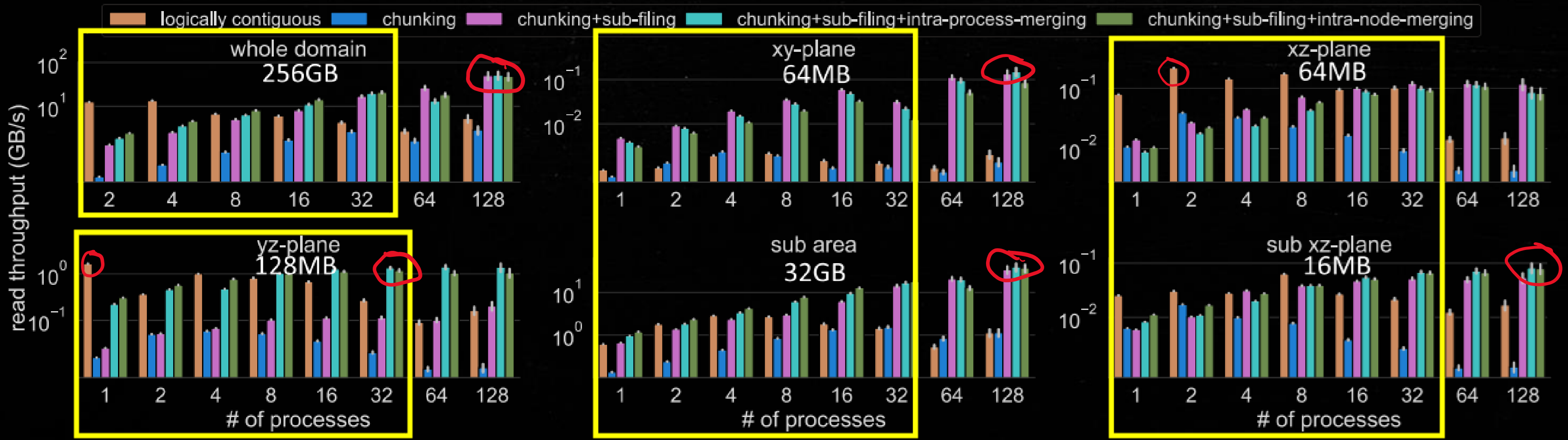


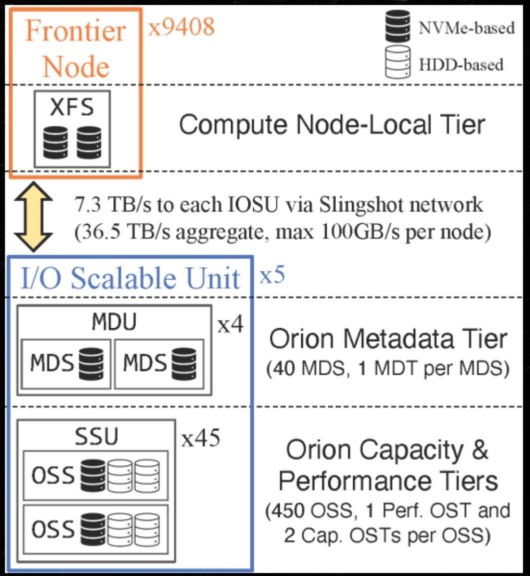
Fig. 5. Impact of decomposition schemes when reading.

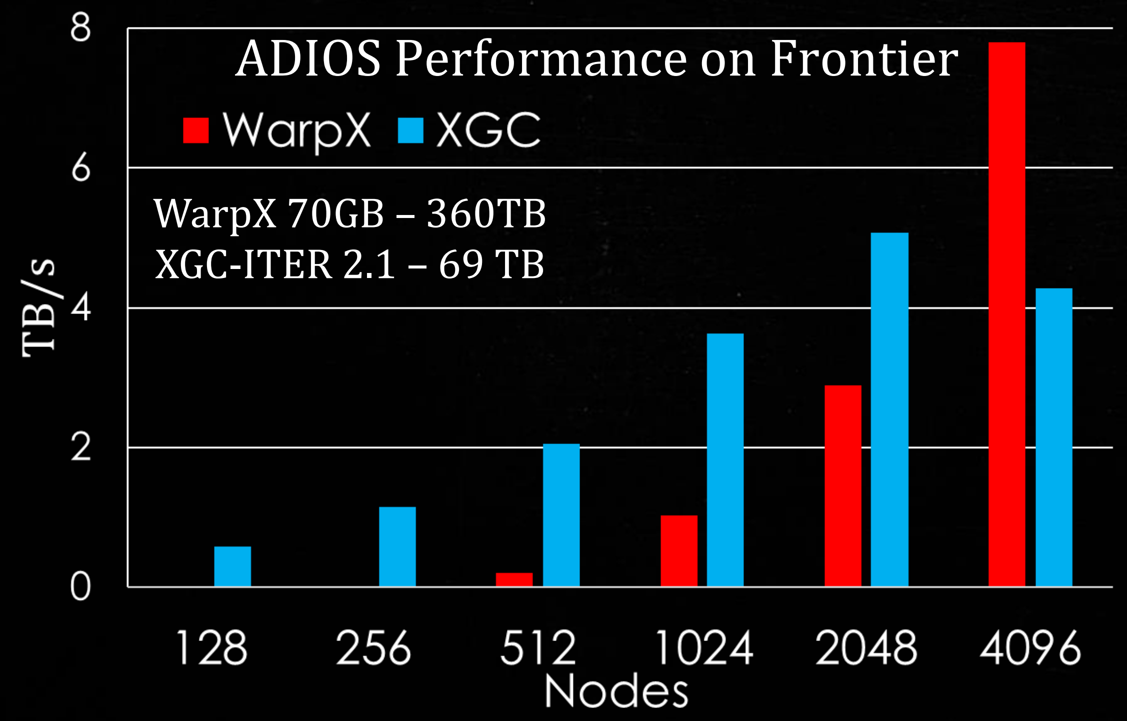


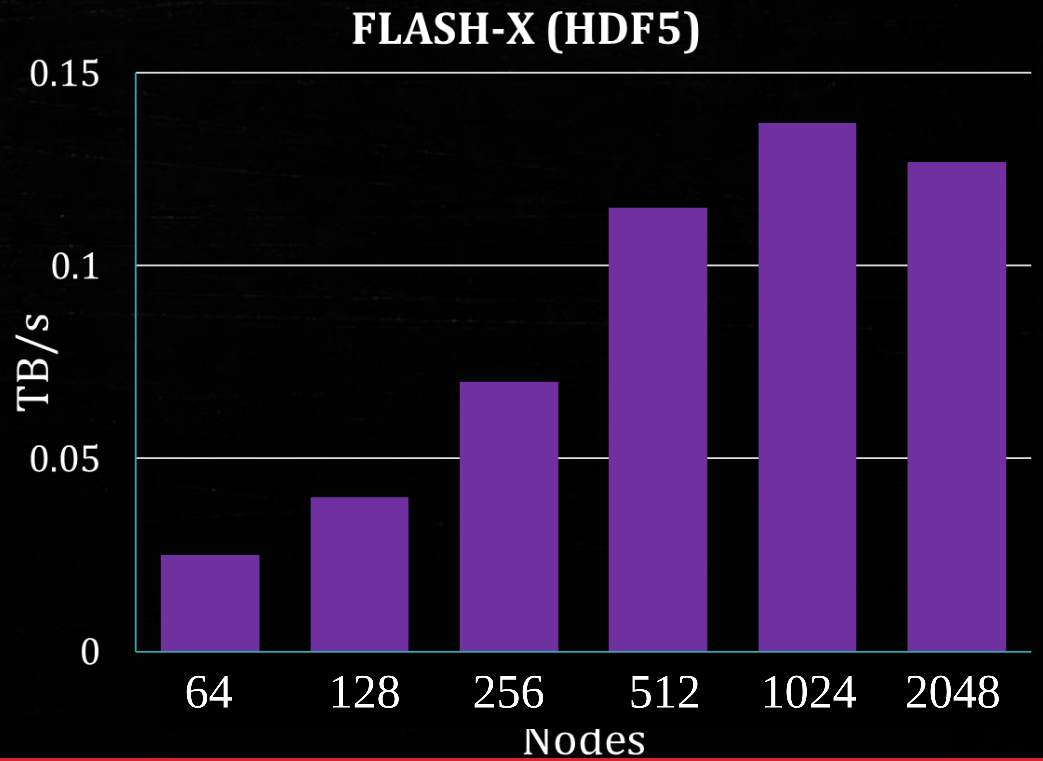
Wan, L., Huebl, A., Gu, J., Poeschel, F., Gainaru, A., Wang, R., ... & Klasky, S. (2021). Improving I/O performance for exascale applications through online data layout reorganization. IEEE Transactions on Parallel and Distributed Systems, 33(4), 878-890.

XGC, WarpX, Flash-X on Frontier

Tier	Capacity (PB)	Read BW (TB/s)	Write BW (TB/s)
Node-Local	33	75	38
Metadata	10	0.8	.5
Performance	11.5	10	10
Capacity	679	5.5	4.6



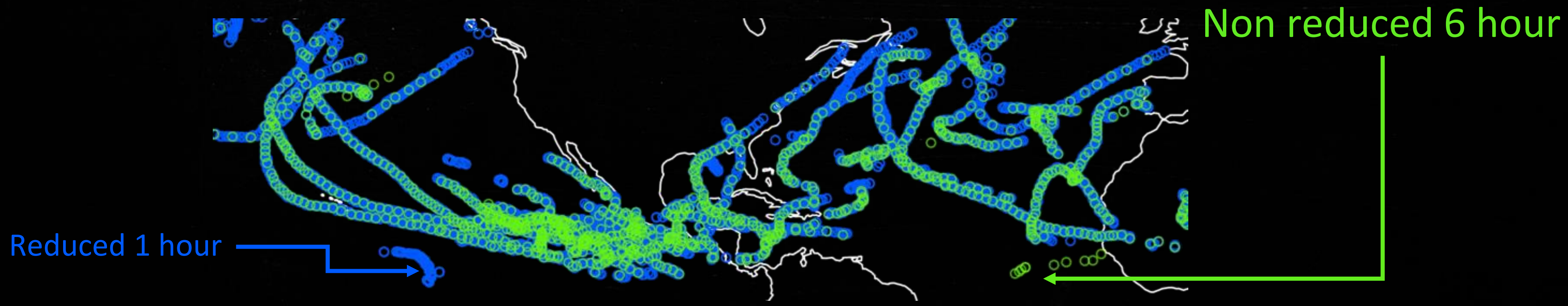




- ADIOS Performance with WarpX and XGC on Frontier > 5TB/s, Flash-X with HDF5 on Frontier <0.2 TB/s

Climate:E3SM

- E3SM researchers asked us at the beginning of ECP to speed up the I/O from 100 MB/s to > 1 GB/s
  - New performance enhancements with their SCORPIO-PNETCDF=1.3 GB/s: write time=123 s
  - Converting the output to ADIOS-2 allows us to achieve >110 GB/s on Frontier: write time = 1.3s
- Output data is still "time averaged" which challenges researchers for accurate prediction of Atmospheric Rivers (output is written every 6 hours), but we wanted to understand if we could output every hour
  - Using MGARD we could reduce the storage footprint by 4X and get 20X more accurate prediction for Atmospheric Rivers and 1.8X for Cyclone tracks



SKA

- Working with the SKA (Perth) team we were asked to speed up the I/O and processing speed to allow more data to be moved/saved for the sky surveys
- The first ever end-to-end workflow for processing the Square Kilometre Array (SKA) data, composed and verified on the Summit Supercomputer
- For the first time, radio astronomy data were generated and processed at 130 PFLOPS peak and 247 GB/s. The results are being used to reveal critical design factors for the next-generation radio telescopes and processing facilities
    - Designed and developed core components of an end-to-end data processing workflow for SKA, including the I/O sub-system using ADIOS achieving 925 GB/s for pure i/O for storing table-based radio astronomy
- New challenges remain to reduce the storage and I/O cost when analyzing the data
    - MGARD with mask encoding can reduce the data by 20X

Seismic Tomography Workflow (PBs of data/run) [2.2 TB/s]

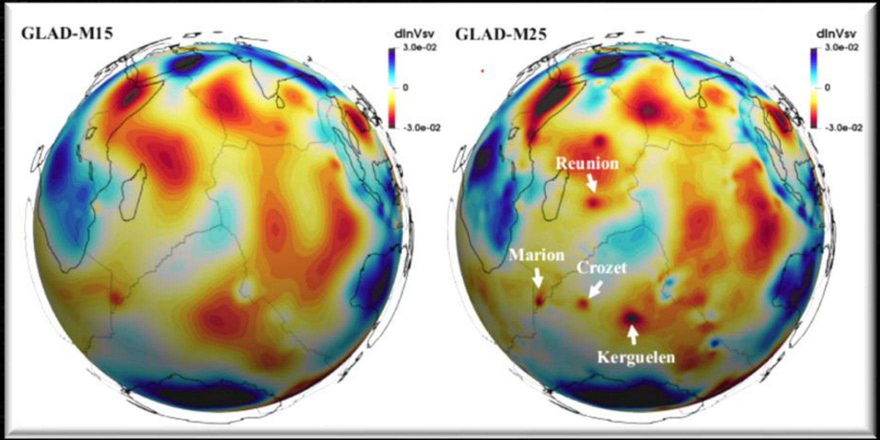
PI: Jeroen Tromp, Princeton

Scientific Achievement

- Most detailed **3-D model of Earth**'s interior showing the entire globe from the surface to the core–mantle boundary, a depth of 1,800 miles

Significance and Impact

- Updated (transversely isotropic) global seismic model GLAD-M25 where no approximations were used to simulate how seismic waves travel through the Earth. The data sizes required for processing are challenging even for leadership computer
- 7.5 PB of data is produced in a single workflow step
  - Which is fully processed later in another step

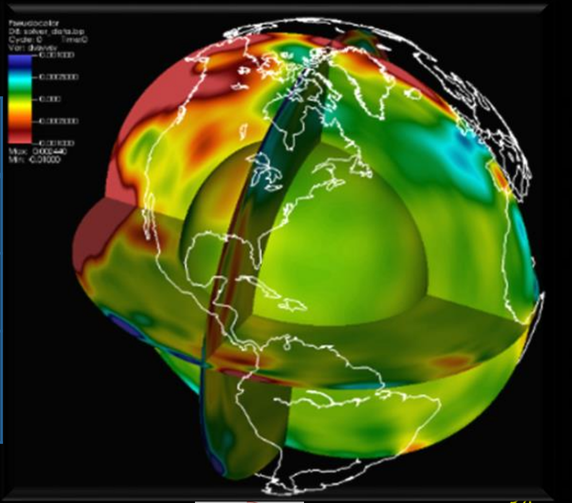


Map views at 250 km depth of vertically polarized shear wave speed perturbations in GLAD-M15 (2017) and GLAD-M25 (2020) in the Indian Ocean. New features have emerged in GLAD-M25, such as the Reunion, Marion, Kerguelen, Maldives, Seychelles, Cocos and Crozet hotspots.

Improvement by appending steps

- 3200 nodes ensemble run, 19200 GPUs
- 50 tasks at once
- 5.2 TB per task in 133 steps
- 260 TB total per 50 tasks
- 7.5 PB per 1500 tasks (total run)

50 tasks, 133 steps, 3200 nodes	Time
No I/O	94s
BP3, one file per step	235s
BP4 one dataset per job 133x reduction in # of files	156s



Lei, Wenjie, et al. "Global adjoint tomography—model GLAD-M25." Geophysical Journal International 223.1 (2020): 1-21.

ECP RAPIDS

E3SM Ultra-high resolution atmospheric simulations

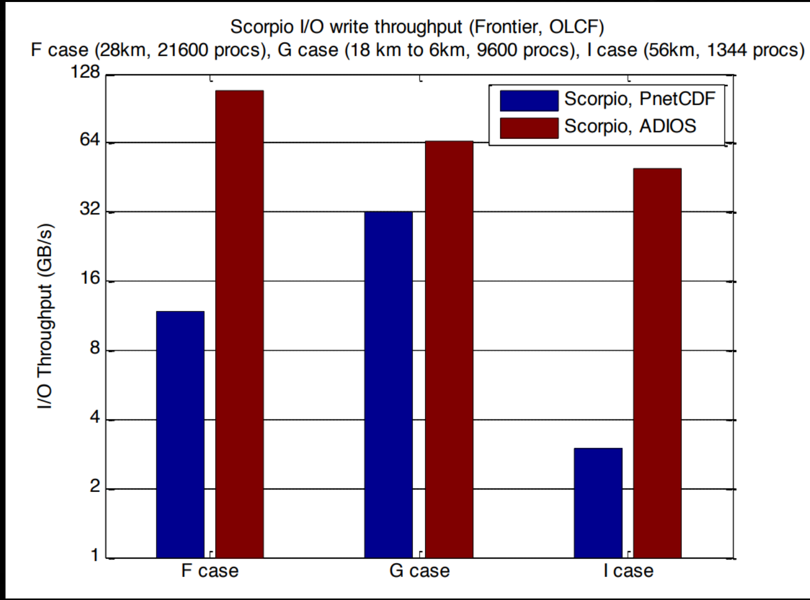
High resolution simulation (E3SM-MMF)

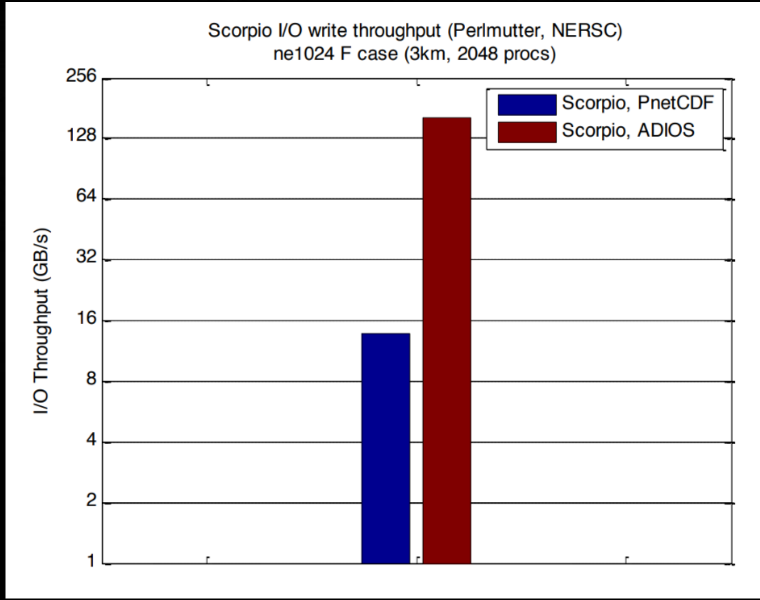
- E3SM F case (Active Atmosphere and Land)
  - ne120 1 day run, no restarts (only history), 21600 procs
  - Data written out: 151.86 GB (1 file, 417 variables)
- E3SM G case (Active Ocean and Sea Ice)
  - 1 day run with no restarts (only history), 9600 procs
  - Data written out: 77.46 GB (1 file, 51 variables)

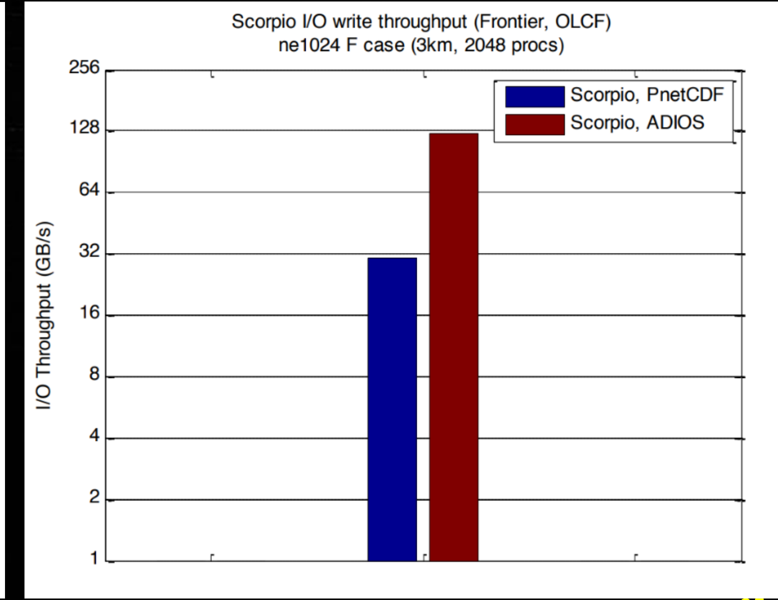
- E3SM I case (Active Land and River)
  - 10 days run with no restarts (only history), 1344 procs
  - Data written out: 360.81 GB (2 files, 1120 variables)

Ultra High resolution simulation (SCREAM) Perlmutter and Frontier

- ne1024 12 hours run with restarts and history, 2048 procs
- Data written out: 3.46 TB (30 files, 1255 variables)





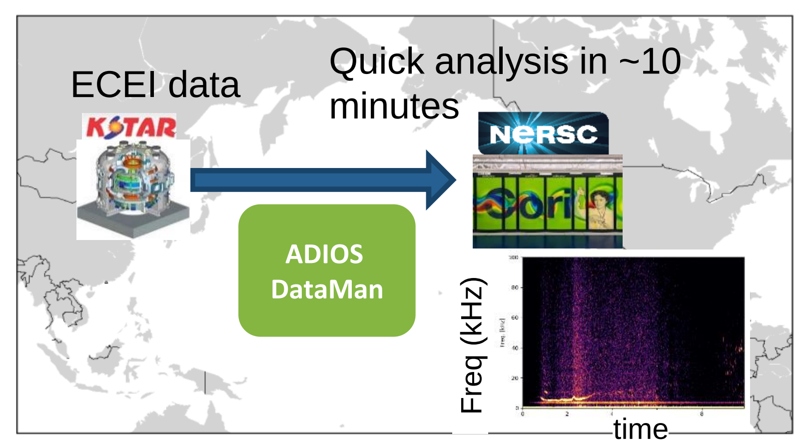


Jayesh Krishna, 2023 E3SM All-hands-meeting poster presentation on SCORPIO

Using staging to establish capability for near-real time networked analysis of fusion experimental data (KSTAR)

Research and develop a streaming workflow framework, to enable near-real-time streaming analysis of KSTAR data on a US HPC

- Allow the framework to adopt ML/AI algorithms to enable adaptive near-real-time analysis on large data streams
- Created a framework to enable US fusion researchers to have broader and faster access to the KSTAR data, enabling
  - Faster analysis of data
  - Faster and autonomous utilization of ML/AI algorithms for incoming data
  - More informed steering of experiment
- Accomplishments
  - Created end-to-end Python framework, streams data using ADIOS over WAN (at rates > 4 Gbps), asynchronously processes on multiple workers with MPImulti-threading
  - Applied to KSTAR streaming data to NERSC Cori.
  - Reduces time for analysis from 12 hours to 10 minutes



Churchill, Klasky, S., et al.(2021). A Framework for International Collaboration on ITER Using Large-Scale Data Transfer to Enable Near-Real-Time Analysis. Fusion Science and Technology, 77(2), 98-108.

Outline

- Vision
  - Next generation workflows to drive our R&D
- Application success stories
- Data Reduction
- Future Steps

MGARD – MultiGrid Adaptive Reduction of Data

Conventional data reduction techniques

- Encoding
- Error unbounded lossy compression

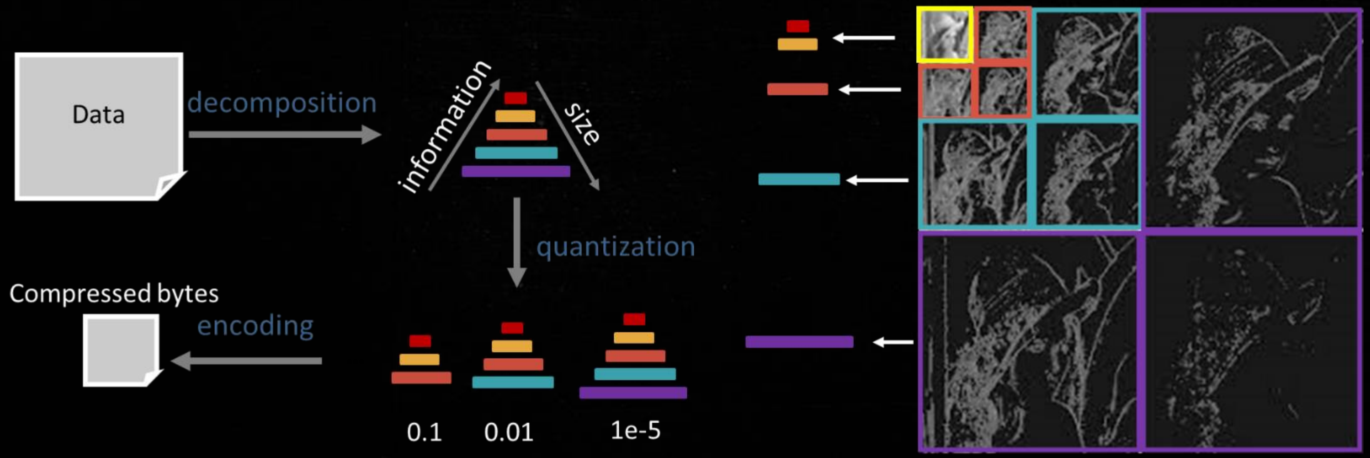
Limited compressibility, unknown errors

Error-controlled lossy data compression

- Data transformation → quantization → encoding
- Mathematically control errors in reconstructed data

Large compression ratio, guaranteed errors

- MGARD is a transform-based compressor (multi-resolution, multi-precision)



Trust – raw & derived quantities

Fast – GPUs, platform portable

Adaptive compression – multi-resolution, localized error control

Progressive retrieval – incremental data recomposition, multi-views

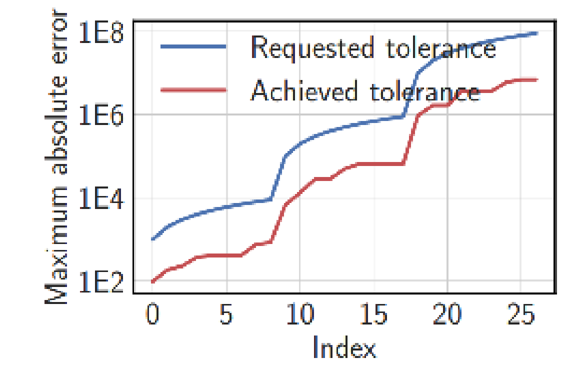
Error Control on Primary and Derived Quantities

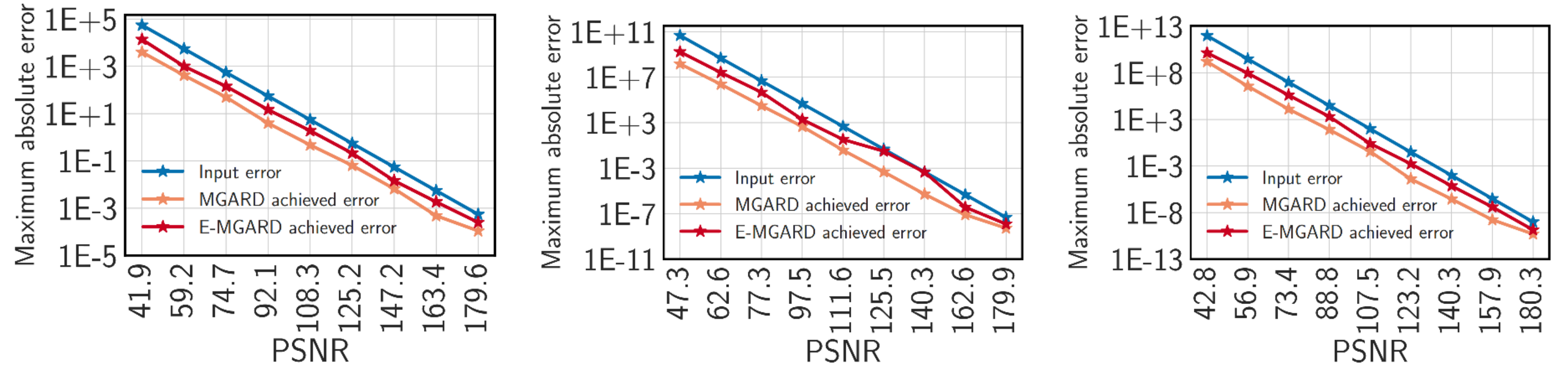
- MGARD decomposes input data into multilevel coefficients u_mc and ensures:

$$R_s(Q)\left(\sum_{l=0}^{L} 2^{2sl} vol(P_l) \sum_{x \in N_l^*} |u\_mc[x] - \tilde{u}\_mc[x]|^2\right) \leq \tau^2$$

where $\tilde{u}\_mc$ is a reduced representation of u_mc, τ^2 is the user-prescribed error bound on either primary or derived quantities, and $R_s(Q)$ is the norm to preserve quantities-of-interest (QoI).

- Machine learning for tighter error bounds
  - Pessimistic error estimation incurs extra storage and I/O cost



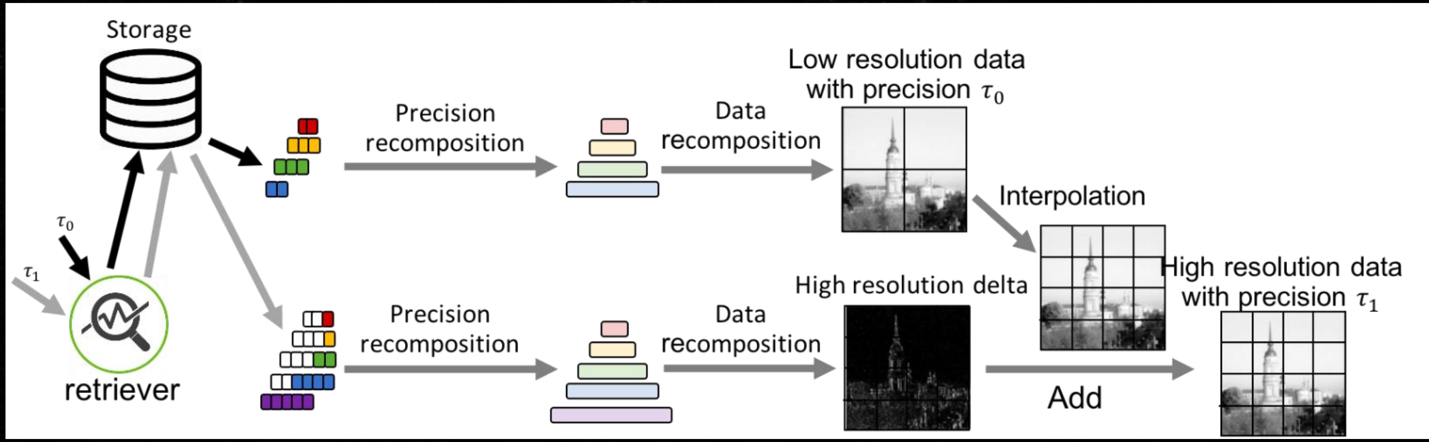


(a) B_x (b) E_x (c) J_x

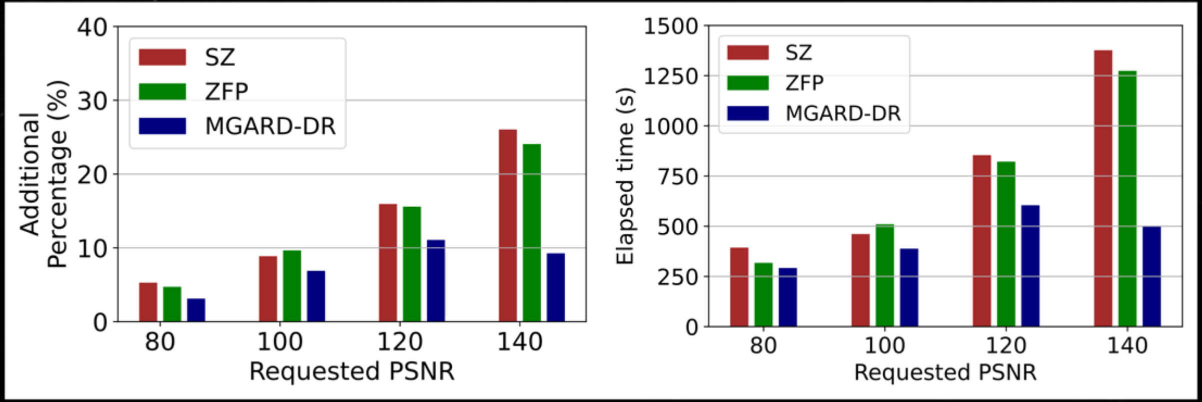
Compression for WarpX data: MGARD vs MGARD+ML prediction

Refactoring and Progressive Retrieval

- Data refactoring through MGARD multilevel decomposition and bitplane encoding
  - Information prioritizing
  - Resource constraints adaptive

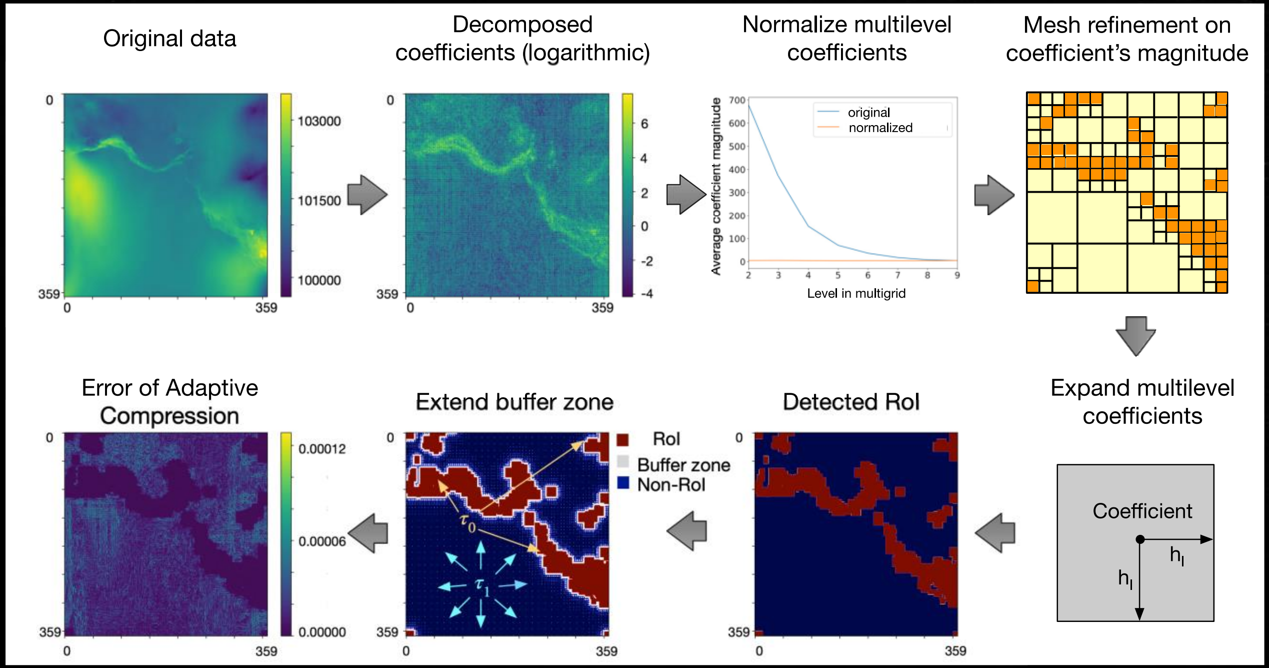


- Progressively retrieve data to desired accuracy
  - For tasks requiring varied data quality (e.g., visualization, statistic analysis)

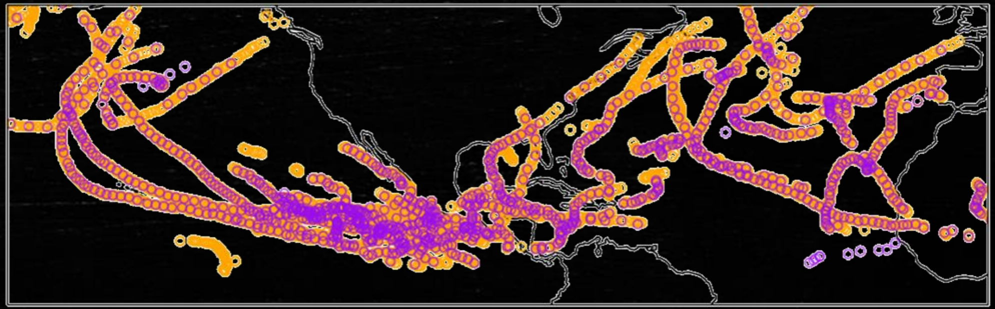


Spatiotemporal Adaptive Compression

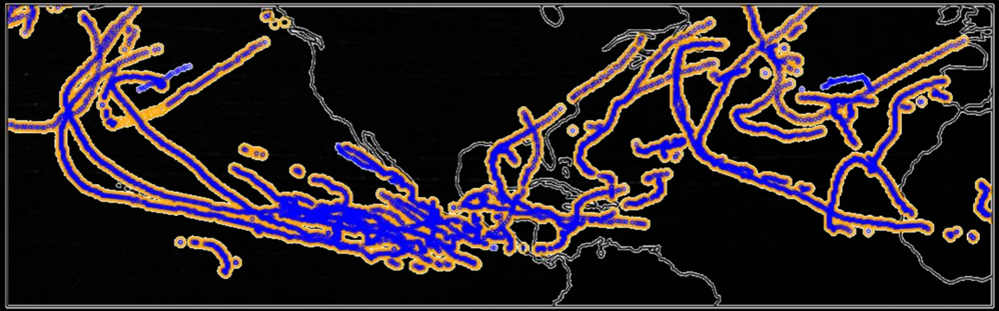
- Adaptively compress spatiotemporal variables and preserve analytical features (e.g., cyclones, atmospheric rivers)



AMR-based critical region detection & localized error control



hourly 6-hourly



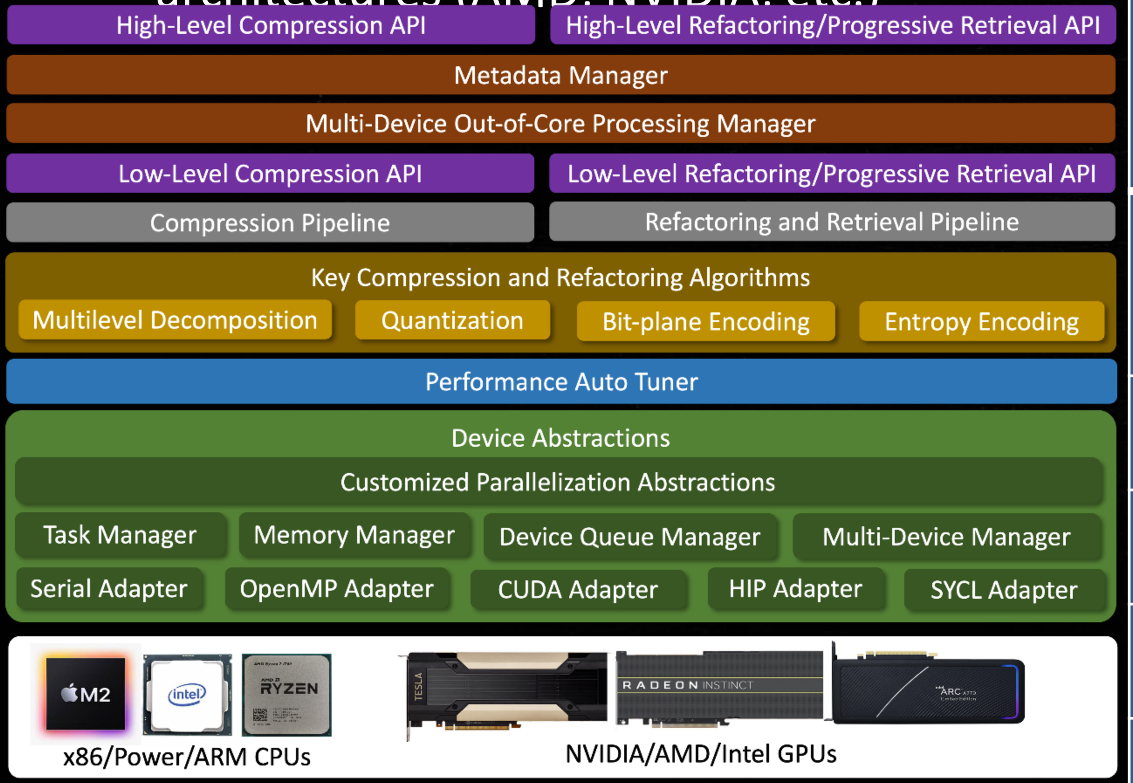
hourly hourly spatiotemporal adaptive (CR 23)

<u>Output data every hour (compared to every 6 hours)</u>

- Output 4x smaller with higher frequency (CR=23)
- 38% more perfect matched and 10X less fully/partially missed cyclone tracks

MGARD Software Architecture & High-performance

- Compression and refactoring APIs
  - Portable across different CPU and GPU architectures (AMD, NVIDIA, etc.)



  - MGARD vs other lossy (cuSZ, ZFP-CUDA) and lossless (NVCOMP) compressors on Summit

Dataset	GPU Compressors											
	MGARD-X			cuSZ			ZFP-CUDA			NVCOMP::LZ4		
	CR	Speed (GB/s) C	Speed (GB/s) D	CR	Speed (GB/s) C	Speed (GB/s) D	CR	Speed (GB/s) C	Speed (GB/s) D	CR	Speed (GB/s) C	Speed (GB/s) D
NYX baryon_density Rel EB: 1e-3	**826**	27	30	32	8	8	3.3	16	14	0.9	5	14
E3SM (PSL) Rel EB: 1e-3	**22**	22	23	Crashed			5.7	13	12	1	6	16
XGC Rel EB: 1e-3	**17**	30	31	Double not supported			5.1	17	14	1.1	6	18
QMCPACK Rel EB: 1e-3	**15**	23	19	26	9	8	2.6	12	13	1.1	5	16
Miranda Rel EB: 1e-3	**51**	29	26	Double not supported			8	17	14	0.9	14	12

Outline

- Vision
    - Next generation workflows to drive our R&D
- Application success stories
- Data Reduction
- Future Steps

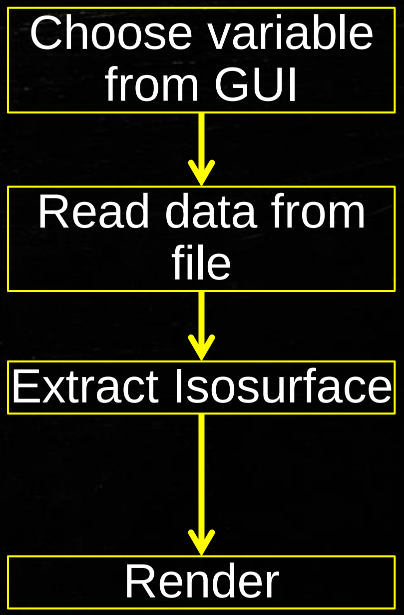
The creation of the Exascale Data Store

- Idea 1: Collect ADIOS metadata file(s) into the ADIOS Campaign File (.acf)
  - Multiple ADIOS files can have their metadata placed into the metadata to describe more about where the "data" is, and the ACF can point to multiple ADIOS files including
    - Raw data: such as particle data file, the 3D mesh file, the 2D output, the diagnostics, ..
    - From multiple restarts
    - For analysis output
    - In multiple formats (BP5, HDF, …) as long as the … came with a schema file
- Idea 2: Allow ADIOS to communicate either through a RESTFUL API (request are translated into pointers into files) or through variables (e.g. SST) over the network (HPC, LAN, WAN)
  - Security will be through SSH tunnels (or other mechanisms)
- Idea 3: Allow ADIOS to perform queries on
  - Primary Data
  - Known QoI
  - Unknown QoI

  With very little storage overheads
- Idea 4: Optimize the return of information over data
  - MGARD can refactor data

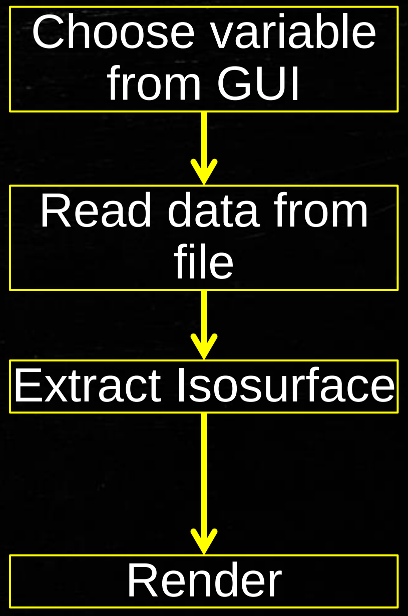
The Viz/Rendering abstraction moving to a SOA approach

- Example: Read in a few variables and the MESH from a HDF5 file
  - GUI picks which variables
  - Variables are read in from the HDF5 file
  - If the data needs the mesh, it will read in the mesh
  - All communication occurs via memory references (copy if necessary)
  - All work is done on the same number of nodes
  - Visualization is done by the render in the same program unless sent in client-server mode to (e.g. Paraview) server



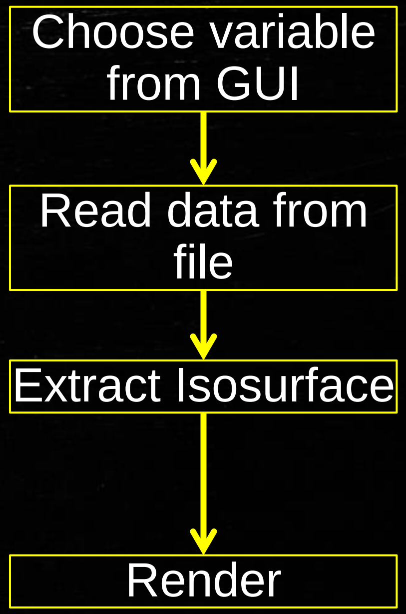
The Viz/Rendering abstraction moving to a SOA approach

- Example: Read in a few variables and the MESH from a BP5 file
  - GUI picks which variables
  - Variables are read in from the BP5 file
  - If the data needs the mesh, it will read in the mesh
  - All communication occurs via memory references (copy if necessary)
  - All work is done on the same number of nodes
  - Visualization is done by the render in the same program unless sent in client-server mode to (e.g. Paraview) server



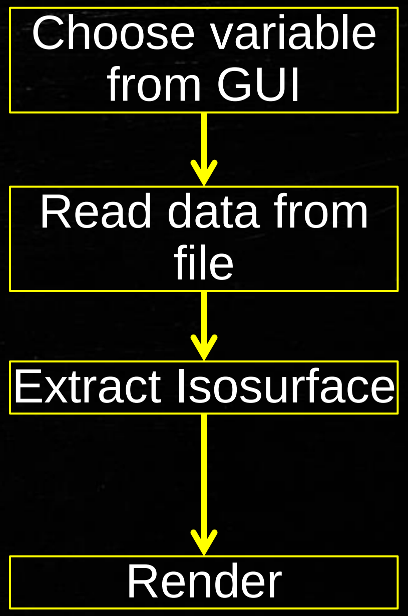
The Viz/Rendering abstraction moving to a SOA approach

- Example: Read in a few variables and the MESH from a BP5 file
  - GUI picks which variables
  - Variables are streamed in from the BP5 file on a/multiple remote servers
  - If the data needs the mesh, it will read in the mesh
  - All communication occurs via memory references (copy if necessary)
  - All work is done on the same number of nodes
  - Visualization is done by the render in the same program unless sent in client-server mode to (e.g. Paraview) server



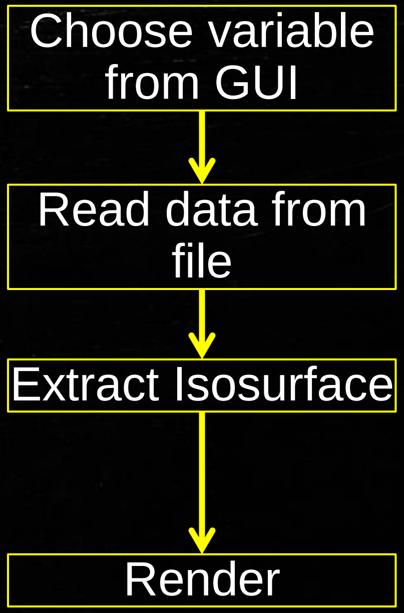
The Viz/Rendering abstraction moving to a SOA approach

- Example: Read in a few variables and the MESH from a BP5 file
  - GUI picks which variables
  - Variables are streamed in from the BP5 file on a/multiple remote servers
  - If the data needs the mesh, it will read in the mesh
  - All communication occurs via memory references (copy if necessary)
  - All work is done on the same number of nodes
  - Visualization is done by the render in a different program which can be on a remote (e.g. Paraview) server or Python Notebook or ..



The Viz/Rendering abstraction moving to a SOA approach

- Example: Read in a few variables and the MESH from a BP5 file
  - GUI picks which variables
  - Variables are streamed in from the BP5 file on a/multiple remote servers
  - If the data needs the mesh, it will read in the mesh
  - All communication occurs via memory references (copy if necessary)
  - All work is done on the same number of nodes
  - Visualization is done by the render in a different program which can be on a remote (e.g. Paraview) server or Python Notebook or ..
  - A dashboard (e.g. esimmon) can subscribe to servers from many producers, allowing some users to see visualizations from 100s of different tools/users



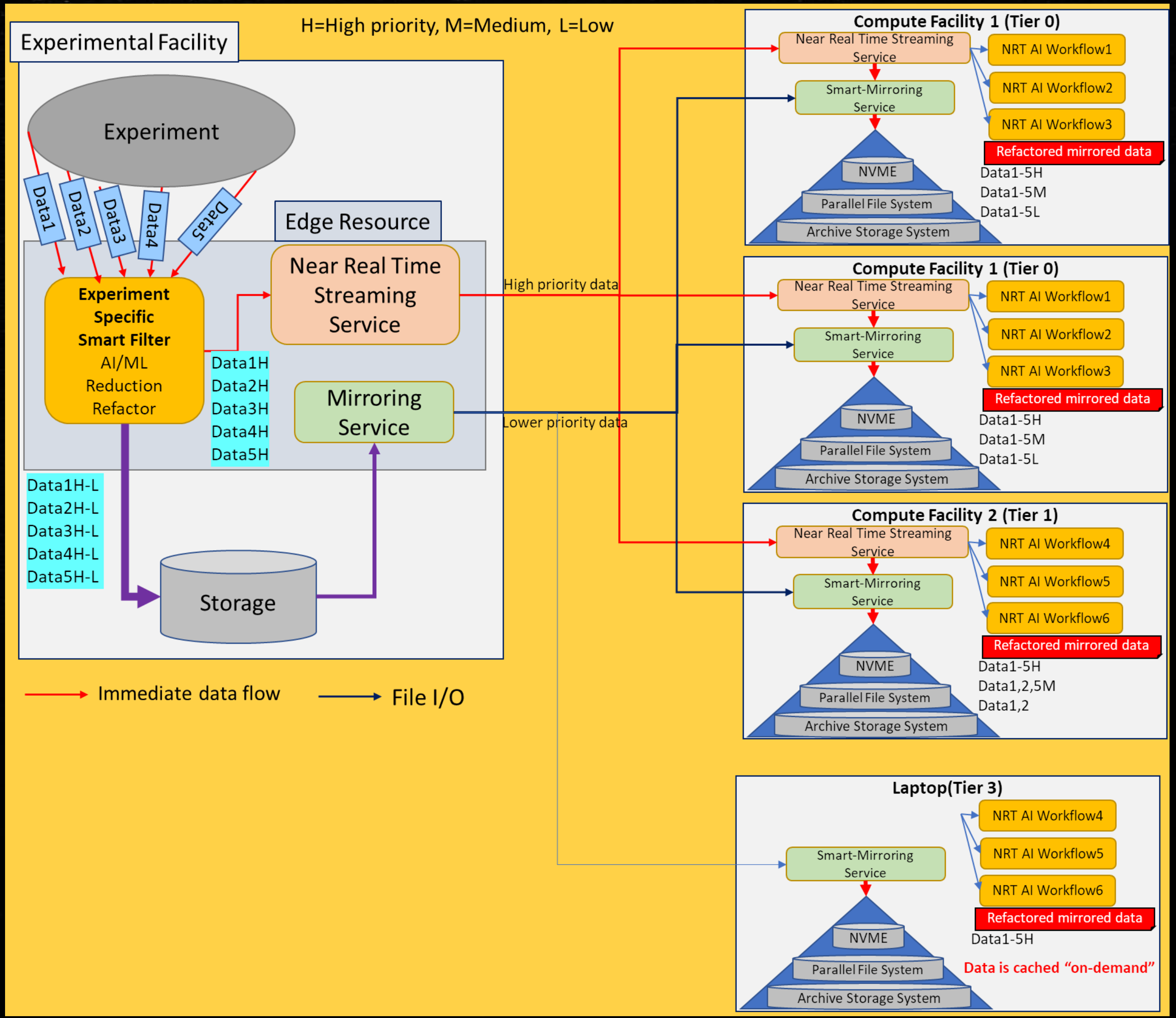
Data reduction for creating a remote data facility

Challenges:

- Storing the "multiple" views of data across distributed facilitates and dynamically accessing "partial" data on demand

Proposed research:

- Designing efficient metadata structure and management algorithm to
  - Access and recompose refactored scientific data across remote sites
  - Create "multiple" views of data per application/analysis requirements
- Integrating with data management tools for
  - Cross-system data prefetching and caching
  - Unified data abstraction and common I/O APIs



Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

ADIOS Concepts and C++ API

ADIOS Useful Information and Common tools

- ADIOS documentation: https://adios2.readthedocs.io/en/latest/index.html
- ADIOS Examples: https://adios2-examples.readthedocs.io/en/latest/
- ADIOS source code: https://github.com/ornladios/ADIOS2
  - Written in C++, wrappers for Fortran, Python, Matlab, C
  - Contains command-line utilities (bpls, adios_reorganize ..)
- This tutorial's code example (Gray-Scott): https://github.com/ornladios/ADIOS2-Examples
- Online help:
  - ADIOS2 GitHub Issues: https://github.com/ornladios/ADIOS2/issues

- Two movies showing the Tutorial for post-processing and on-line processing
- https://users.nccs.gov/~pnorbert/GrayScottPost.mp4
- https://users.nccs.gov/~pnorbert/GrayScottInsitu.mp4

ADIOS Approach: “How”

- I/O calls are of declarative nature in ADIOS
  - which process writes/reads what
    - add a local array into a global space (virtually)
  - EndStep() indicates that the user is done declaring all pieces that go into the particular dataset in that output step or what pieces each process gets
- I/O strategy is separated from the user code
  - aggregation, number of sub-files, target file-system hacks, and final file format not expressed at the code level
- This allows users
  - to choose the best method available on a system without modifying the source code
- This allows developers
  - to create a new method that’s immediately available to applications
  - to push data to other applications, remote systems or cloud storage instead of a local filesystem

ADIOS basic concepts

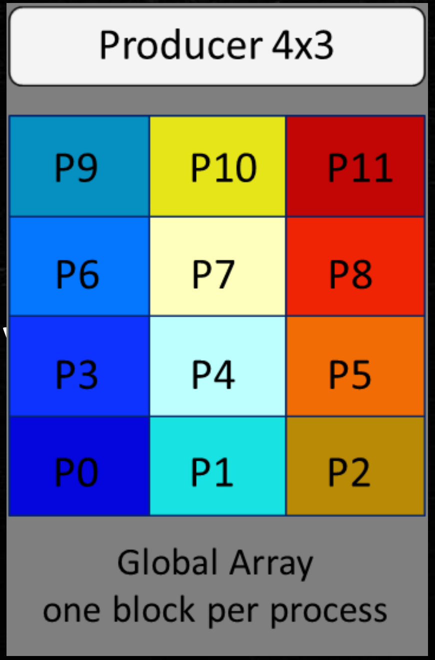
- Self-describing Scientific Data
- Variables
  - multi-dimensional, typed, distributed arrays
  - single values
    - Global: one process, or Local: one value per process
- Attributes
  - static information
    - for humans or machines
  - global, or assigned to a variable

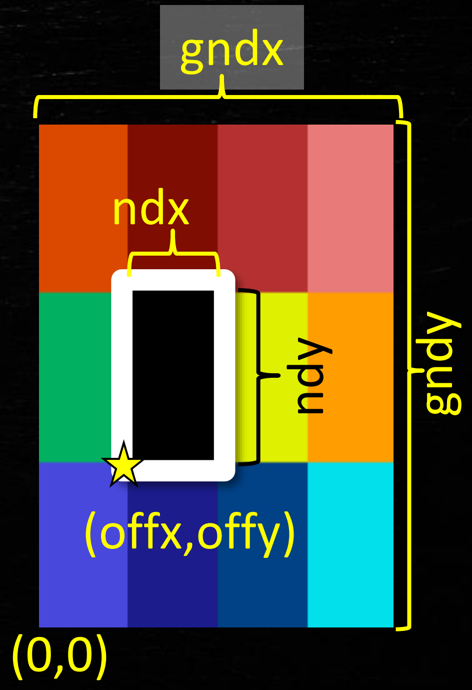
Self-describing Scientific Data

```
real      /fluid_solution/scalars/PREF                    scalar = 0
string    /fluid_solution/domain14/blockName/blockName    scalar = "rotor_flux_1_Main_Blade_skin"
integer   /fluid_solution/domain14/sol1/Rind              {6} = 2 / 2
real      /fluid_solution/domain14/sol1/Density           {8, 22, 52} = 0.610376 / 1.61812
real      /fluid_solution/domain14/sol1/VelocityX         {8, 22, 52} = -135.824 / 135.824
real      /fluid_solution/domain14/sol1/VelocityY         {8, 22, 52} = -277.858 / 309.012
real      /fluid_solution/domain14/sol1/VelocityZ         {8, 22, 52} = -324.609 / 324.609
real      /fluid_solution/domain14/sol1/Pressure          {8, 22, 52} = 1 / 153892
real      /fluid_solution/domain14/sol1/Nut               {8, 22, 52} = -0.00122519 / 1
real      /fluid_solution/domain14/sol1/Temperature       {8, 22, 52} = 1 / 362.899
string    /fluid_solution/domain17/blockName/blockName    scalar = "rotor_flux_1_Main_Blade_shroudga

integer   /fluid_solution/domain17/sol1/Rind              {6} = 2 / 2
real      /fluid_solution/domain17/sol1/Density           {8, 8, 52} = 0.615973
real      /fluid_solution/domain17/sol1/VelocityX         {8, 8, 52} = -135.824
...
```

Global Array: data produced by multiple processes

- N-dimensional array
  - **Shape**
- Has a type (int32, double, etc.)
  - **Type**
- Blocks of data are written into the array
  - **Start** (offset)
  - **Count** (size of block)





Shape = {gndx, gndy}

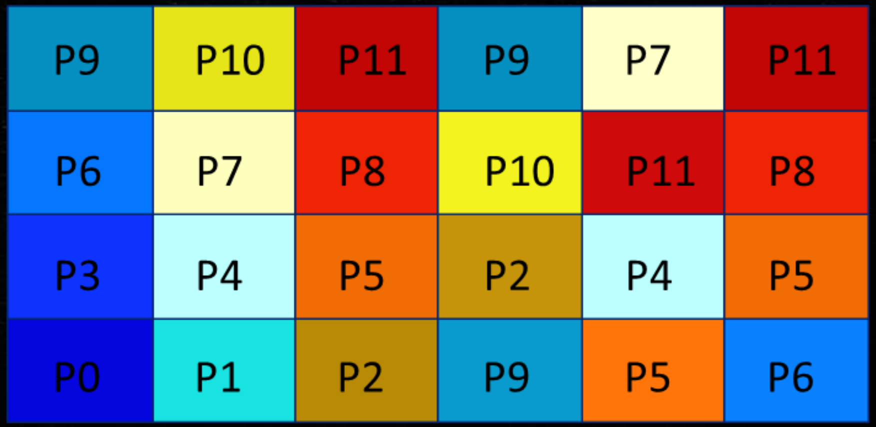
Start = {offx, offy}

Count = {ndx, ndy}

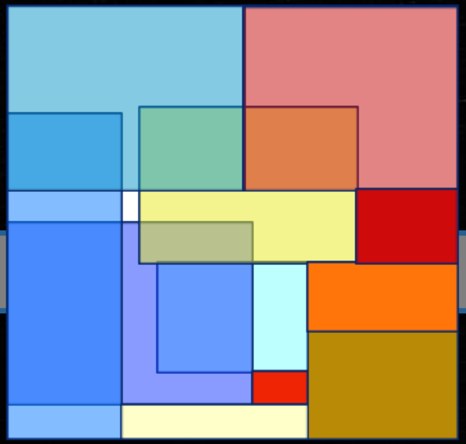
Global Array: data produced by multiple processes

- These are valid global arrays
  - One process can contribute more than one block
  - Some process may not write anything at all
  - Holes can be left in the global array
  - Overlapping of blocks is allowed

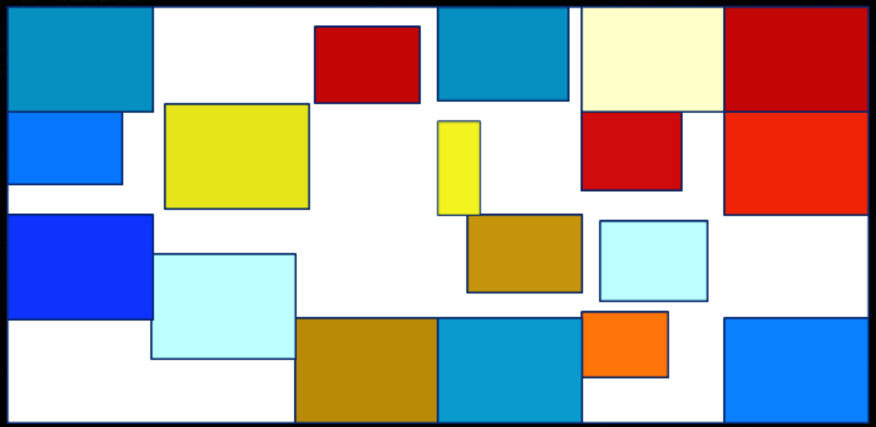
12 Producers
multiple blocks per process



Global Array with overlapping blocks



Global Array sparsely filled out



Read returns “nothing” for those cells.

ADIOS basic concepts

- Step
  - Producer outputs a set of variables and attributes at once
    - This is an **ADIOS Step**
  - Producer iterates over computation and output steps
- Producer outputs multiple steps of data
  - e.g. into multiple, separate files, or into a single file
  - e.g. steps are transferred over network
- Consumer processes step(s) of data
  - e.g. one by one, as they arrive
  - e.g. all at once, reading everything from a file
    - not a scalable approach

Step is a Transaction between producer and its consumers

ADIOS Steps: Rules and constraints

- Step is not necessarily tied to the application timesteps
  - a Step can be constructed over time
- Entire content of a Step is either completely written or not at all
- A new Step can be very different from the previous step
  - may contain a completely different set of variables
  - array sizes can change
  - array decomposition can change
- Consumer is guaranteed to have access to entire content of Step as long as it wants it
- Entire content of a Step must fit into the producer’s memory as a copy

ADIOS coding basics

- Objects
  - ADIOS
  - Variable
  - Attribute
  - IO
    - a group object to hold all variable and attribute definitions that go into the same output/input step
    - settings for the output/input
    - settings may be given before running the application in a configuration file
  - Engine
    - the output/input stream
  - Operator
    - a compression, reduction, data transformation operator for output variables

ADIOS object

- The container object for all other objects
- Gives access to all functionality of ADIOS

```
#include <adios2.h>
adios2::ADIOS  adios(configfile, MPI communicator);
```

- Notes:
  - both arguments are optional
    - `adios(), adios("config.xml"), adios(comm)`
  - Normally use 1 config file and then have different communicators for each I/O target

ADIOS2-Examples/source/cpp/gray-scott/simulation/main.cpp

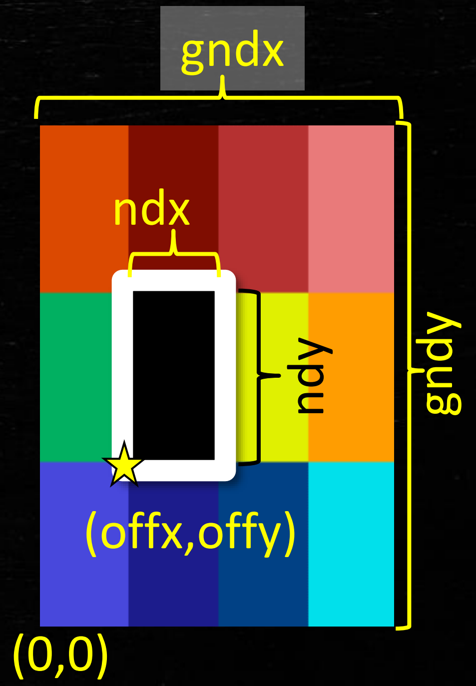
IO object

- Container for all variables and attributes that one wants to output or input at once
- Application settings for IO
- User run-time settings for IO – from configuration file (or input parameters)
  - a **name** is given to the IO object to identify it in the configuration

```
adios2::IO io = adios.DeclareIO("CheckpointRestart");
```

Variable

- N-dimensions
- Type
- Decomposition across many processors
  - global dimensions (Shape), local place (Start, Count)




```
adios2::Variable<double> &varT = io.DefineVariable<double>
(
    "T",                // name in output/input
    {gndx, gndy},       // Global dimensions (2D here)
    {offx, offy},       // starting offsets in global space
    {ndx, ndy}          // local size
);
```

  - C/C++/Python always row-major, Fortran/Matlab/R always column-major

Hint: if it's only checkpoint restart, just use global dimensions {NPROC, N}, local offsets {Rank, 0},

Engine object

- To perform the IO (for a single output step)

```
adios2::Engine writer =
io.Open("checkpoint.bp", adios2::Mode::Write);


writer.Put(varT, T.data());


writer.Close()
```

T is used in here!
Do not invalidate
content of T
before this!

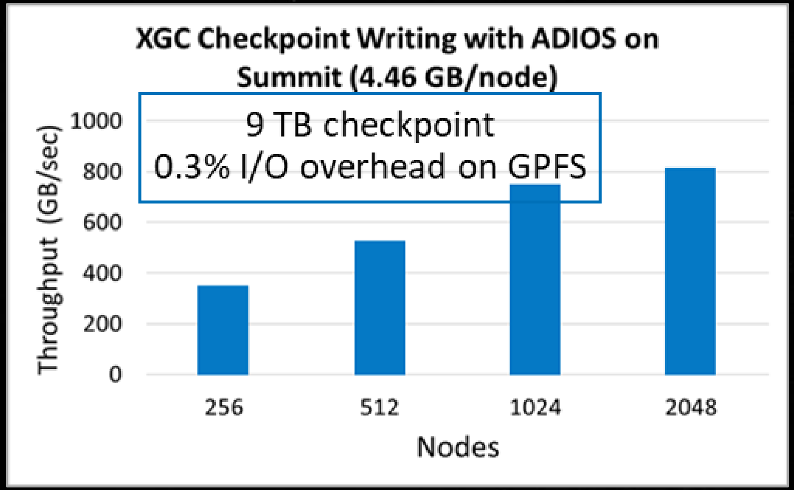
Reading is similar, but we can read from any number of procs

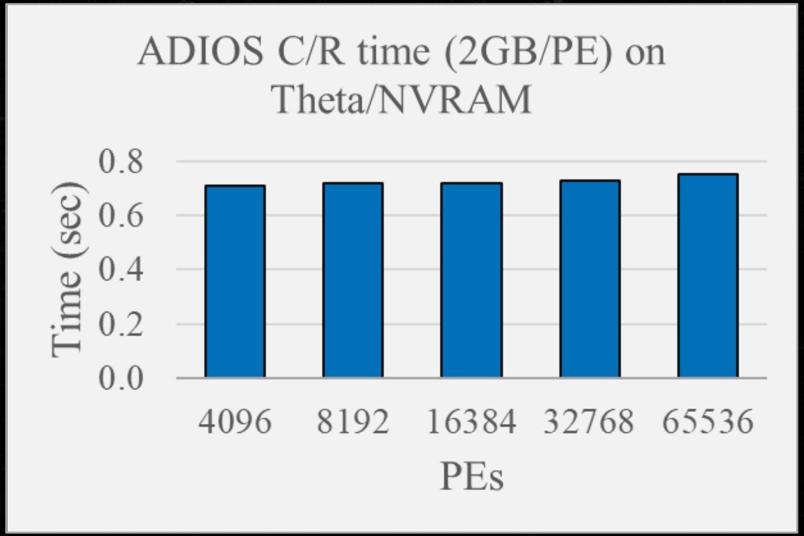
```
adios2::IO io = adios.DeclareIO("CheckpointRestart");

adios2::Engine reader =
     io.Open("checkpoint.bp", adios2::Mode::Read);


adios2::Variable<double> vT =
       io.InquireVariable<double>("T");

if (vT) {
   reader.Get(*vT, T.data());
}

reader.Close()
```

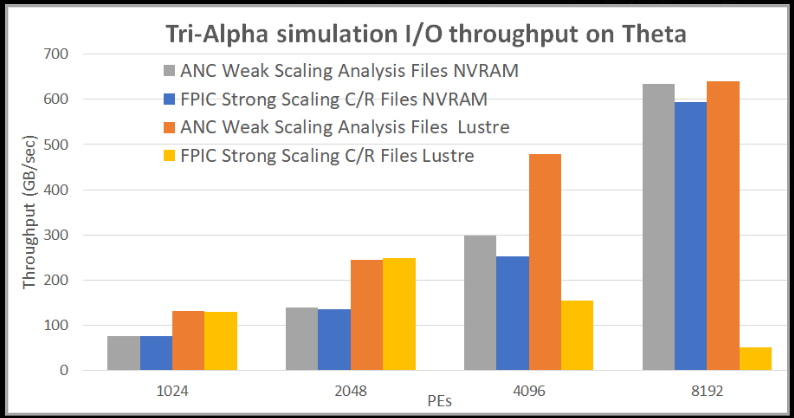
Reserve memory for T before this

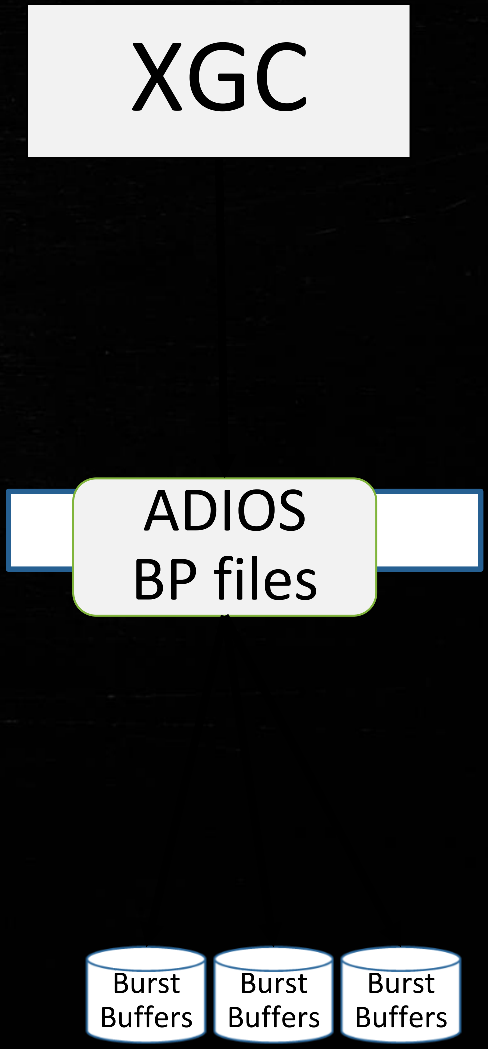
ADIOS APIs for self describing data output for C/R to NVRAM

- No changes to ADIOS APIs to write to Burst Buffers
  - The ADIOS-BP file format required no changes for C/R









Analysis/visualization data

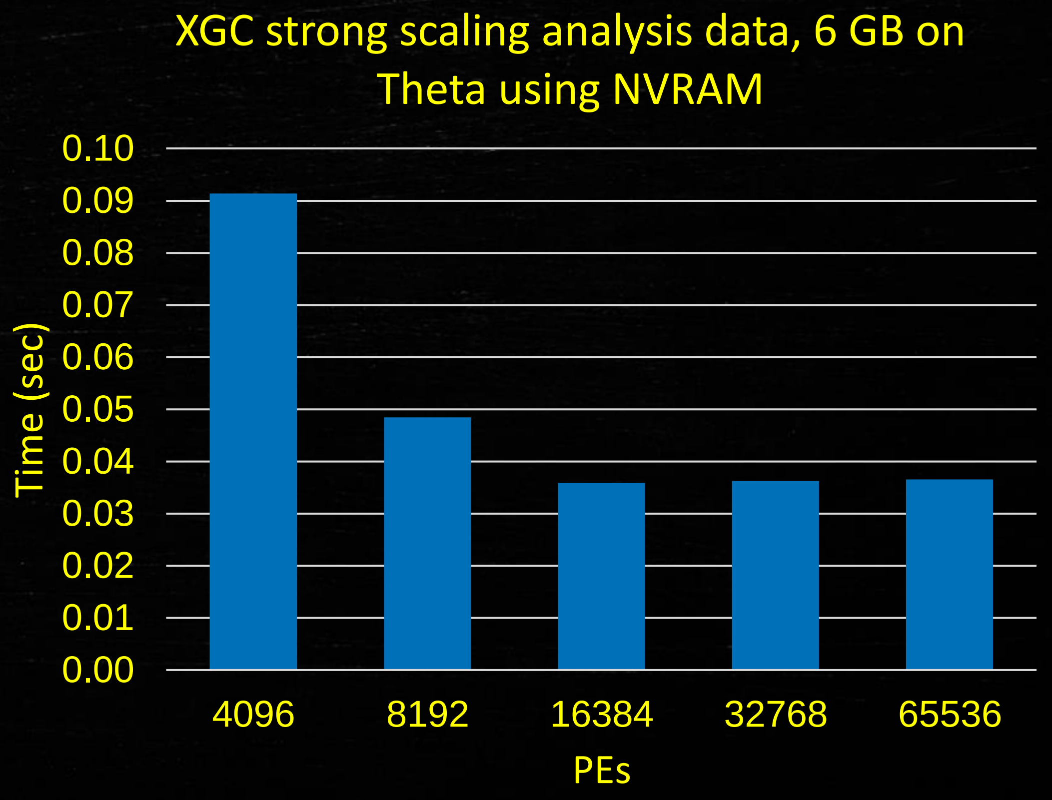
```
adios2::IO io = adios.DeclareIO("Analysis_Data");
if (!io.InConfigFile()) {
    io.SetEngine("FileStream");
}
adios2::Variable<double> varT = io.DefineVariable<double>
(
    "Temperature",        // name in output/input
    {gndx,gndy,gndz},     // Global dimensions (3D here)
    {offx, offy, offz},   // starting offsets in global space
    {nx,ny,nz}            // local size
);
io.DefineAttribute<std::string>("unit", "C", "Temperature");
```

```
double   Temperature         10*{20, 30, 40} = 8.86367e-07 / 200
string   Temperature/unit     attr   = "C"
```

Engine object

- To perform the IO

```
adios2::Engine writer =
io.Open("analysis.bp",
        adios2::Mode::Write);


writer.BeginStep()

    writer.Put(varT, T.data());

writer.EndStep()



writer.Close()
```



Put API explained

`engine.Put(varT, T.data())`

- Equivalent to

`engine.Put(varT, T.data(), adios2::Mode:Deferred)`

- This does NOT do the I/O (to disk, stream, etc.) once put return.
- you can only reuse the data pointer after calling `engine.EndStep()`

`engine.Put(varT, T.data(), adios2::Mode:Sync)`

- This makes sure data is flushed or buffered before put returns
- Get() works the same way
- The default mode is deferred
- BP5 specific:
  - Large Deferred Put() is NOT buffered
  - Use Sync mode only when you need it (when you need to modify the data pointer before EndStep)

ADIOS engines – change from BP5 to HDF5

1. <io name="SimulationOutput">
2. <engine type="**BP5**"/>
3. </io>

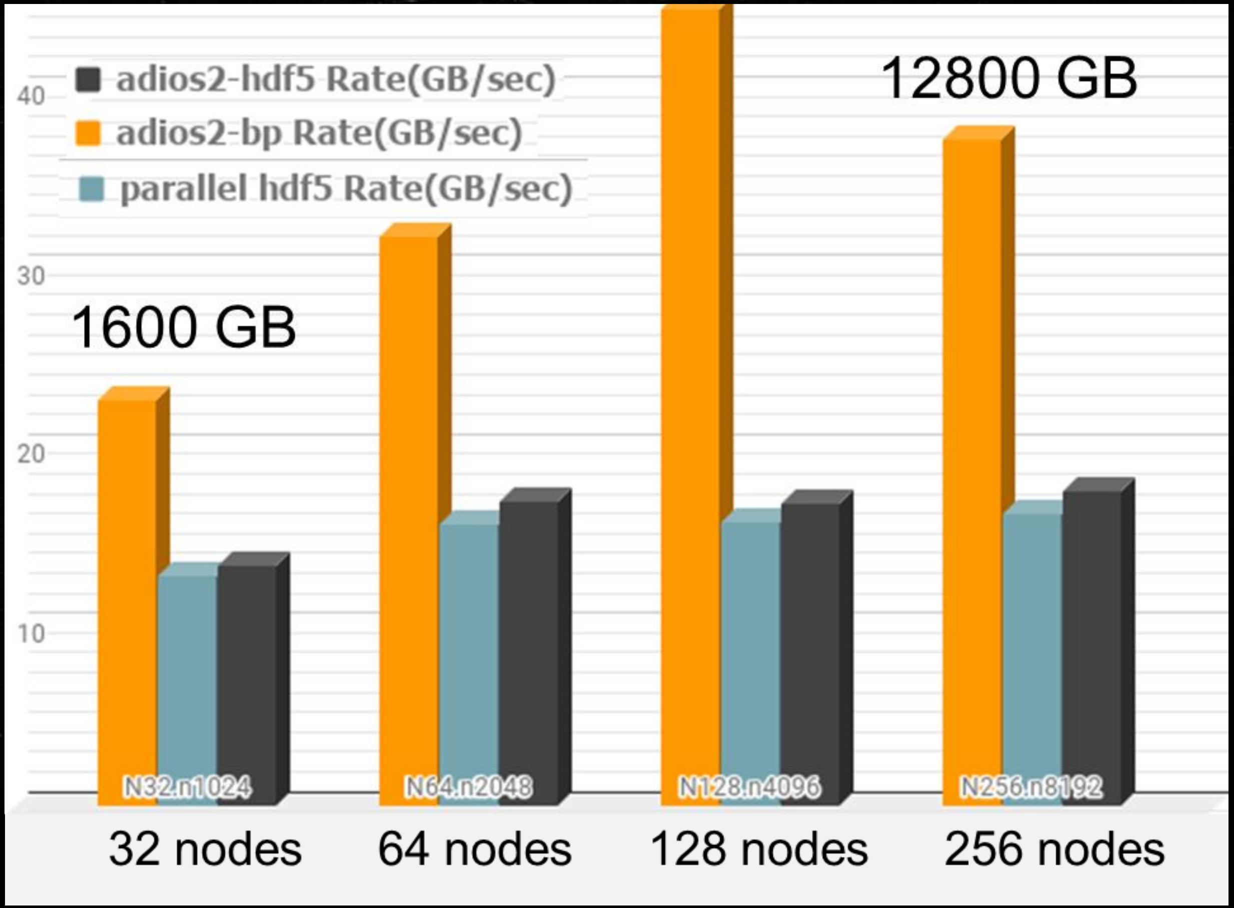
Change Engine name

1. <io name="SimulationOutput">
2. <engine type="**HDF5**"/>
3. </io>

- or in the source code

```
io.SetEngine("HDF5");
```

Cori + Lustre, for the heat equation



ADIOS Python API
Basically, the C++ API in Python

Python common

Serial python:

```
import numpy
import adios2

T = numpy.array(...)
```

Parallel python with MPI:

```
from mpi4py import MPI
import numpy
import adios2

T = numpy.array(...)
```

Python API start: ADIOS, IO and Engine objects

```python
adios = adios2.ADIOS("adios2.xml")
adios = adios2.ADIOS("adios2.xml", comm, True)

io = adios.DeclareIO("SimulationOutput")
fr = io.Open("stream.bp", adios2.Mode.Read)

# adios2.Mode.Read - input stream step-by-step
# adios2.Mode.ReadRandomAccess - to see all steps at once (file)
# adios2.Mode.Write - to create an output stream
# adios2.Mode.Append - to append new steps to existing file

fr.Close()
```

Python API: Step-by-step reading

```
while True:
    status = fr.BeginStep()
    if status == adios2.StepStatus.EndOfStream:
        print("-- no more steps found --")
        break
    elif status == adios2.StepStatus.NotReady:
        sleep(1)
        continue
    elif status == adios2.StepStatus.OtherError:
        print("-- error with stream --")

    cur_step = fr.CurrentStep()
    ...
    fr.EndStep()
```

```
while True:
    status = fr.BeginStep()
    if status != adios2.StepStatus.OK:
        break
    cur_step = fr.CurrentStep()
    ...
    fr.EndStep()
```

Python Read API: List variables

```
vars_info = io.AvailableVariables()

for name, info in vars_info.items():
    print("variable_name: " + name)
    for key, value in info.items():
        print("\t" + key + ": " + value)
    print("\n")


# NOTE: list of variables may change
# from step to step
```

variable_name: **T**
Type: double
AvailableStepsCount: **2**
Max: 200
SingleValue: false
Min: 0
Shape: **10, 16**

variable_name: dT
Type: double
AvailableStepsCount: 2
Max: 1.83797
SingleValue: false
Min: -1.78584
Shape: 10, 16

Variable access, selection and read

- var = io.InquireVariable("U")
- shape = var.Shape()
  - E.g. [64, 64, 64]
- Allocate Numpy array for reading
  - data = numpy.empty([shape[1], shape[2]], dtype=np.float64)
- Selection to read: tuple of start and count (e.g. 2D slice of 3D array):
  - var.SetSelection([[int(shape[0]/2),0,0], [1,shape[1],shape[2]]])
- Read data now
  - fr.Get(var, data, adios2.Mode.Sync)

ADIOS Python API
File reading with ReadRandomAccess

Python API start: Engine objects

```
fr = io.Open("stream.bp", adios2.Mode.ReadRandomAccess)
vars_info = io.AvailableVariables()
ustep = int(vars_info["U"]["AvailableStepsCount"])
vu = io.InquireVariable("U")
vu.SetStepSelection([0, ustep])
print("Number of var 'step' steps after SetStepSelection = 
{0}".format(vstep.Steps()))

# read first 4 elements of a 1D array A for all steps
nelems = 4
data = np.zeros([ustep*nelems],  dtype=np.float64)
vA.SetSelection([ [0], [4] ])
fr.Get(vA, data, adios2.Mode.Sync)
```

ADIOS Python High-level API

Python common

Sequential python script:

```
import numpy
import adios2

T = numpy.array(...)
```

Parallel python with MPI:

```
from mpi4py import MPI
import numpy
import adios2

T = numpy.array(...)
```

Python Read API: Open/close a file/stream

```
adios2.open(path, mode [, configFile, ioName])
adios2.open(path, mode, comm [, configFile, ioName])
fr.close()
```

Examples:

```
fr = adios2.open("data.bp", "r")
fr = adios2.open("data.bp", "r", "adios2.xml", "heat")

fr = adios2.open("data.bp", "r", mpicommunicator)
fr = adios2.open("data.bp", "r", mpicommunicator,
                 "adios2.xml", "heat")

fr.close()
```

Python Read API: List variables

```
vars_info = fr.available_variables()

for name, info in vars_info.items():
    print("variable_name: " + name)
    for key, value in info.items():
        print("\t" + key + ": " + value)
    print("\n")
```

```
variable_name: T
	Type: double
	AvailableStepsCount: 2
	Max: 200
	SingleValue: false
	Min: 0
	Shape: 10, 16

variable_name: dT
	Type: double
	AvailableStepsCount: 2
	Max: 1.83797
	SingleValue: false
	Min: -1.78584
	Shape: 10, 16
```

Python Read API: Read data from **file** -- Random access

```
fr.read(path[, start, count][, stepStart, stepCount])
```

Examples:

```
data = fr.read("T")
>>> data.shape
(10, 16)

data = fr.read("T", [0,0], [10,16])
>>> data.shape
(10, 16)

data = fr.read("T", [0,0], [10,16], 0, 2)
>>> data.shape
(2, 10, 16)
```

variable_name: **T**
- Type: double
- AvailableStepsCount: **2**
- Max: 200
- SingleValue: false
- Min: 0
- Shape: **10, 16**

Python Read API: Read data from **file/stream**

```
fr.read(path[, start, count][, endl=true])
```

Examples:

```
for step_fr in fr:
    data = step_fr.read("T")
    print("Shape: ", data.shape)
...
Shape:  (10, 16)
Shape:  (10, 16)
```

variable_name: **T**
- Type: double
- AvailableStepsCount: **2**
- Max: 200
- SingleValue: false
- Min: 0
- Shape: **10, 16**

ADIOS Fortran API

ADIOS Fortran API

- See documentation at https://adios2.readthedocs.io/en/latest/api_full/api_full.html#fortran-bindings
- See API source code in ADIOS2 source
    - bindings/Fortran/modules
    - https://github.com/ornladios/ADIOS2/tree/master/bindings/Fortran/modules

Writing with ADIOS I/O

Fortran variables

```
use adios2
implicit none
type(adios2_adios)       :: adios
type(adios2_io)          :: io
type(adios2_engine)      :: fh
type(adios2_variable)    :: var_T
type(adios2_attribute)   :: attr_unit, attr_desc
```

Writing with ADIOS I/O

```
call adios2_init (adios, "adios2.xml",  app_comm,
                  adios2_debug_mode_on, ierr)

call adios2_declare_io (io, adios, 'SimulationOutput', ierr )

…

call adios2_open (fh, io, filename, adios2_mode_write, ierr)

call adios2_define_variable (var_T, io, "T", adios2_type_dp,  &
                 2, shape_dims, start_dims, count_dims, &
                 adios2_constant_dims,  adios2_err )


call adios2_put (fh, var_T, T, adios2_err)

call adios_close (fh, adios_err)

…

call adios_finalize (rank, adios_err)
```

Multiple output steps:

```
call adios2_begin_step (fh, &
             adios2_step_mode_append, &
             0.0, istatus, adios2_err)
call adios2_put (fh, var_T, T_temp, adios2_err )
call adios2_end_step (fh, adios2_err)
```

Add attributes to output

```
call adios2_define_attribute(attr_unit, io, "unit", "C", "T", adios2_err)
```

Equivalent code in HDF5

```
call h5screate_simple_f(1, attrdim, aspace_id, err)
call h5tcopy_f(H5T_NATIVE_CHARACTER, atype_id, err)
call h5tset_size_f(atype_id, LEN_TRIM(“C”, err)
call h5acreate_f(dset_id, “unit”, atype_id, aspace_id, att_id, err)
call h5awrite_f(att_id, atype_id, “C, attrdim, err)
call h5aclose_f(att_id, err)
call h5sclose_f(aspace_id, err)
call h5tclose_f(atype_id, err)
```

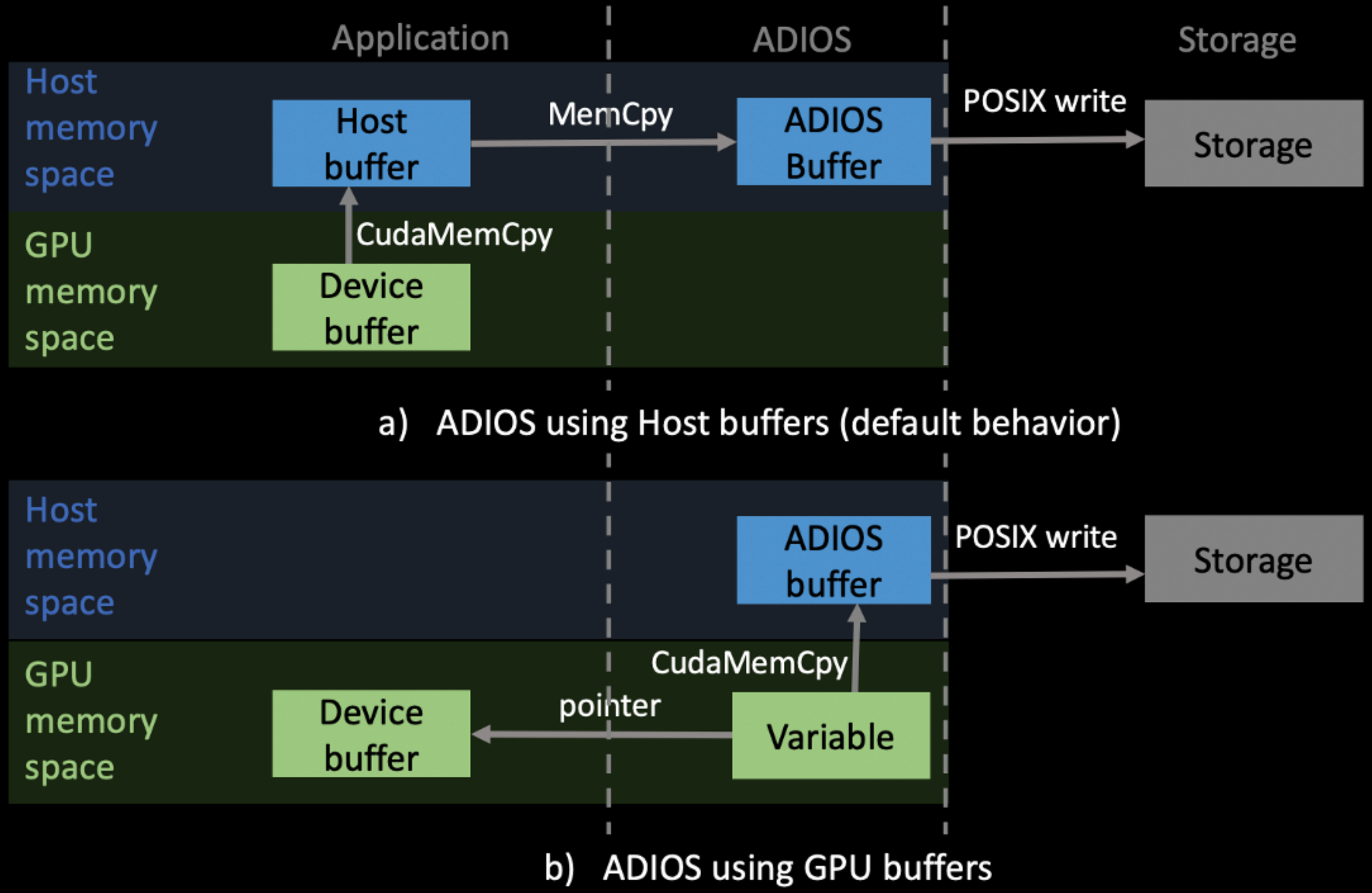
Fortran Read API

```fortran
integer(kind=8) :: adios, io, var, engine
call adios2_init(adios, MPI_COMM_WORLD, adios2_debug_mode_on, ierr)
call adios2_init_config(adios, "config.xml", MPI_COMM_WORLD,
                        adios2_debug_mode_on, ierr)
call adios2_declare_io(io, adios, 'SimulationOutput', ierr)
call adios2_open(engine, io, "data.bp", adios2_mode_read, ierr)
call adios2_inquire_variable(var, io, "T", ierr )
call adios2_variable_shape(var, ndim, dims, ierr)
call adios2_set_selection(var, ndims, sel_start, sel_count, ierr)
call adios2_begin_step(engine, adios2_step_mode_next_available, 0.0, ierr)
call adios2_get(engine, var, T, ierr)
call adios2_end_step(engine, ierr)
call adios2_close(engine, ierr)
call adios2_finalize(adios, ierr)
```

/home/adios/Tutorial/heat2d/fortran/analysis/heatAnalysis_adios2_file.F90

GPU-aware I/O

- Allow applications to give ADIOS GPU buffers
  - Decrease number of copies of the data
  - Transparent performance portability to different GPU architectures
  - Allow ADIOS to use GPU direct to storage, compression on GPU, or other optimizations



API for GPU-aware I/O

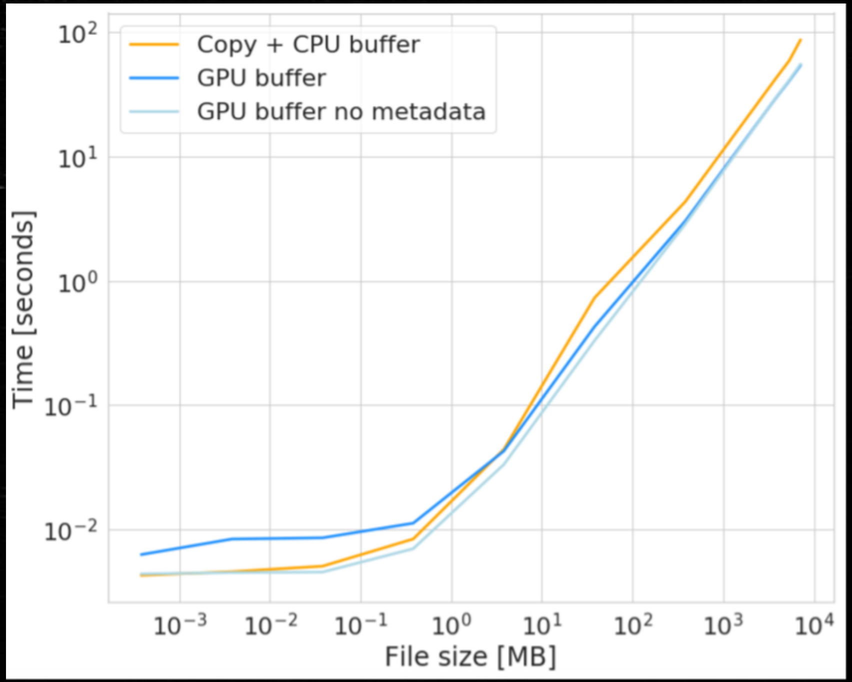
- Build ADIOS2 with CUDA support –D ADIOS2_USE_CUDA=ON
- The user provides a memory space associated with ADIOS2 variables
  - If not set ADIOS2 will detect automatically the memory space

```
adios2::Engine bpWriter;
...
auto data = io.DefineVariable<float>("data", shape, start, count);

bpWriter.Put(data, cpuData);

data.SetMemorySpace(adios2::MemorySpace::GPU);
bpWriter.Put(data, gpuData);
```

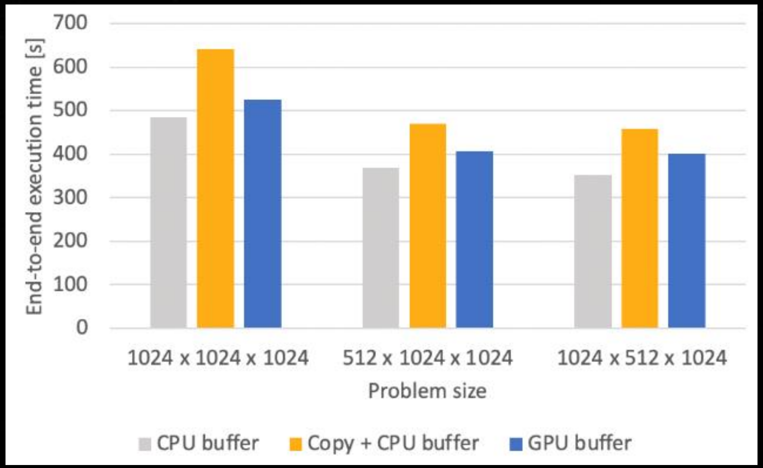
- ADIOS2 saves pointers to data and copies data to internal CPU buffers (in deferred or sync mode)
- Computes metadata for each Get/Put using CUDA kernels

Overhead for detecting where buffers are allocated

CPU STD vector	CUDA CPU buffer	CUDA GPU buffer
5-6 μs	1-2 μs	1-2 μs



Results on I/O kernels and OpenPMD



GPU backends in ADIOS2

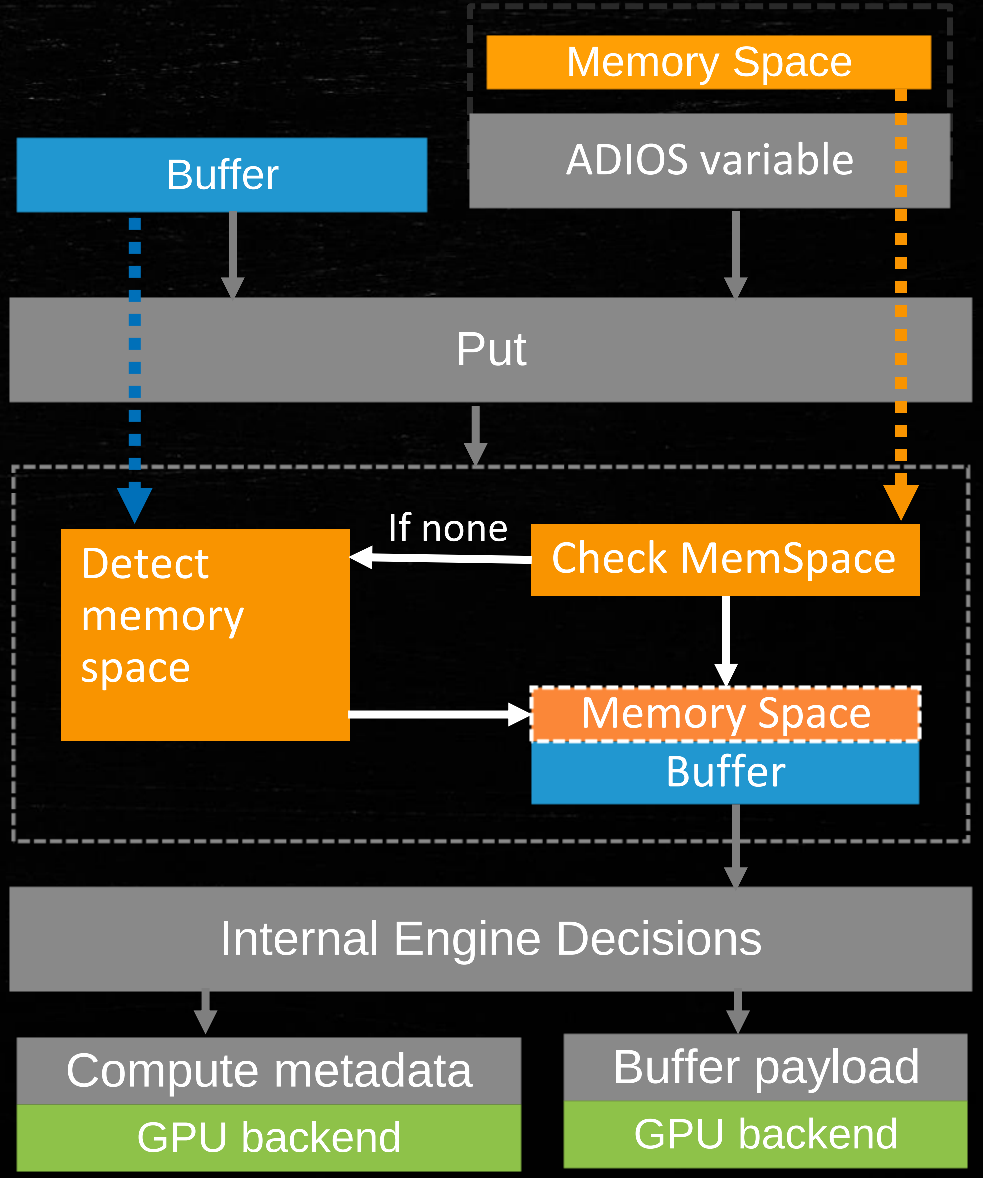
- Backends are chosen during build
  - ADIOS2 can have only one GPU backend active at a given time

```
cmake –D ADIOS2_USE_{BACKEND}=ON
```

  - Current supported backends: CUDA, Kokkos (CUDA), Kokkos (HIP), Kokkos (SYCL)
  - The Kokkos backend is chosen based on the Kokkos build linked to ADOIOS2

```
# build Kokkos
cmake -D Kokkos_ENABLE_CUDA=ON -D Kokkos_ARCH_VOLTA70=ON

# build ADIOS2
cmake -DKokkos_ROOT=${KOKKOS_HOME}/install
-D CMAKE_CUDA_ARCHITECTURES=70 -D ADIOS2_USE_KOKKOS=ON
```

  - The application source code does not change
    - Underneath ADIOS2 is using Kokkos or CUDA functions for computing the metadata and copying the user buffer to ADIOS2 internal buffers



Running an example

- Kokkos backend (same application code, simple write benchmark)
  - Build Kokkos on Summit with CUDA enabled

```
cmake -D CMAKE_CXX_COMPILER=${KOKKOS_HOME}/kokkos/bin/nvcc_wrapper
             -D Kokkos_ENABLE_CUDA=ON
             -D Kokkos_ARCH_VOLTA70=ON
             -D CMAKE_CXX_STANDARD=17
             -D CMAKE_POSITION_INDEPENDENT_CODE=TRUE
```

  - Build ADIOS2 with Kokkos backend

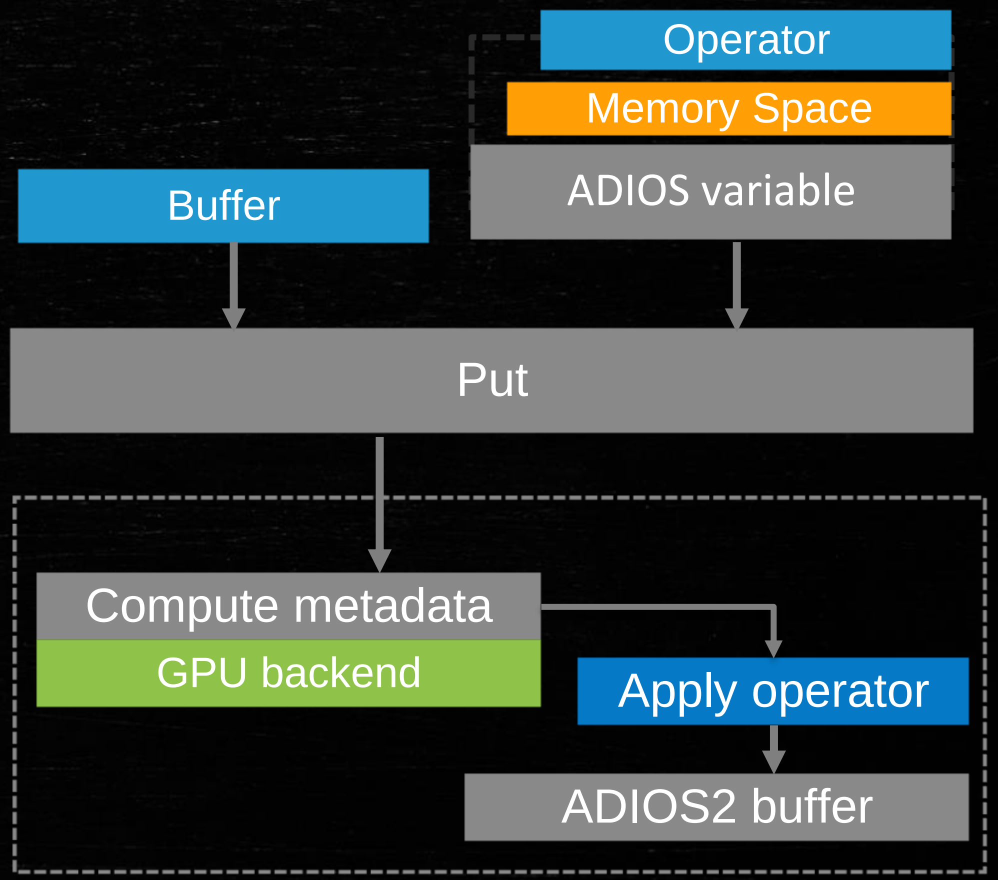
```
cmake -D CMAKE_CXX_STANDARD=17
             -D Kokkos_ROOT=${KOKKOS_HOME}/install
             -D CMAKE_CUDA_ARCHITECTURES=70
             -D CMAKE_CXX_COMPILER=${KOKKOS_HOME}/kokkos/bin/nvcc_wrapper
```

  - The two libraries need to use the same configuration
    - C++ standard 17, VOLTA70 architecture, nvcc_wrapper as the C++ compiler
    - Currently only supported for gcc >= 10

Compression with GPU-aware I/O

- No changes required in the source code
  - Operator attached to a variable
  - Memory space attached to a variable

```
auto var = io.DefineVariable<double>(”test", shape, start, count);

// define an operator
adios2::Operator varOp =
  adios.DefineOperator("mgardCompressor", adios2::ops::LossyMGARD);

//attach operator to variable
var.AddOperation(varOp, parameters);

var.SetMemorySpace(adios2::MemorySpace::GPU); // optional
bpWriter.Put(var, gpuSimData);
```



- Internal logic
  - Metadata is computed using the GPU backend
  - The operator is applied on the GPU buffer and the compressed data is copied directly in the ADIOS buffer

Operators that support GPU buffers:

- MGARD, ZFP
- The operators need to be built with GPU enable

Building applications with ADIOS

Compile ADIOS2 codes

- CMake
  - Use MPI_C and ADIOS2 packages

```
CMakeLists.txt:
project(gray-scott C CXX)
find_package(MPI REQUIRED)
find_package(ADIOS2 REQUIRED)
add_definitions(-DOMPI_SKIP_MPICXX -DMPICH_SKIP_MPICXX)
...
target_link_libraries(gray-scott   adios2::cxx11_mpi   MPI::MPI_C)
```

  - Configure application by adding ADIOS installation to search path

```
cmake -DCMAKE_PREFIX_PATH="/opt/adios2"  <source_dir>
```

  - Available ADIOS2 targets: cxx11 c, fortran, cxx11_mpi, c_mpi, fortran_mpi

Compile ADIOS2 codes

- Makefile
  - Add ADIOS2 library paths to LD_LIBRARY_PATH
  - Use adios2_config tool to get compile and link options

```
ADIOS2_DIR = /opt/adios2/
ADIOS2_FINC=`${ADIOS2_DIR}/bin/adios2-config --fortran-flags`
ADIOS2_FLIB=`${ADIOS2_DIR}/bin/adios2-config --fortran-libs`
```

  - Codes that write and read

```
heatSimulation: heat_vars.F90 heat_transfer.F90 io_adios2.F90
	${FC}   -g -c -o heat_vars.o heat_vars.F90
	${FC}   -g -c -o heatSimulation.o heatSimulation.F90
	${FC}   -g -c -o io_adios2.o ${ADIOS2_FINC} io_adios2.F90
	${FC}   -g -o heatSimulation heatSimulation heat_vars.o io_adios2.o  ${ADIOS2_FLIB}
```

Configure and build the ADIOS2-Examples repository (example)

```
$ cd ~/ADIOS2-Examples/
$ mkdir -p build-cmake
$ cd build-cmake
$ cmake \
  -DCMAKE_INSTALL_PREFIX=/home/adios/Tutorial \
  -DCMAKE_PREFIX_PATH="/opt/adios2;/opt/hdf5-parallel/default;/opt/zfp/default;/opt/sz/default“ \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo \
   ..
$ make -j 4
$ make install
$ cd /home/adios/Tutorial/share/adios2-examples/gray-scott

$ export PATH=$PATH:/home/adios/Tutorial/bin
$ export PYTHONPATH=/opt/adios2/lib/python3/dist-packages
$ export LD_LIBRARY_PATH=/opt/hdf5-1.13.0/parallel/lib::/opt/sz/default/lib:/opt/zfp/default/lib
```

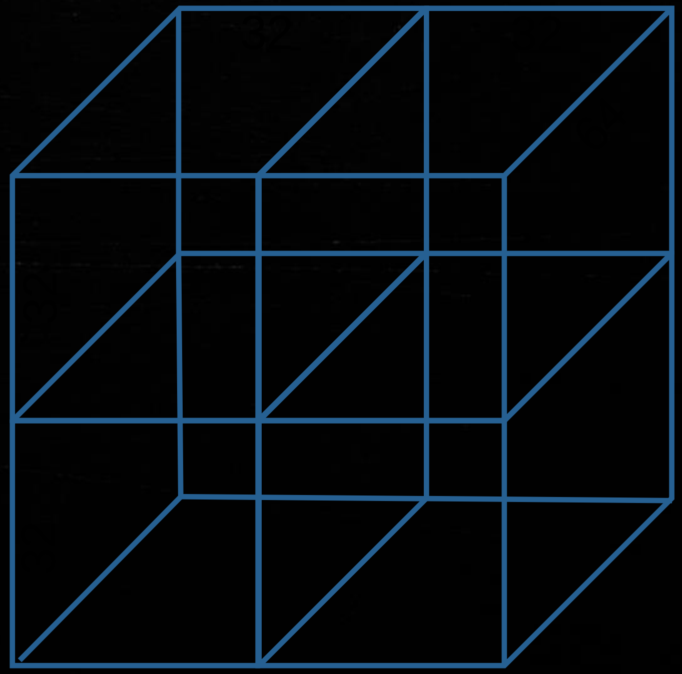
Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

Gray-Scott Example with **ADIOS**: **Write** Part

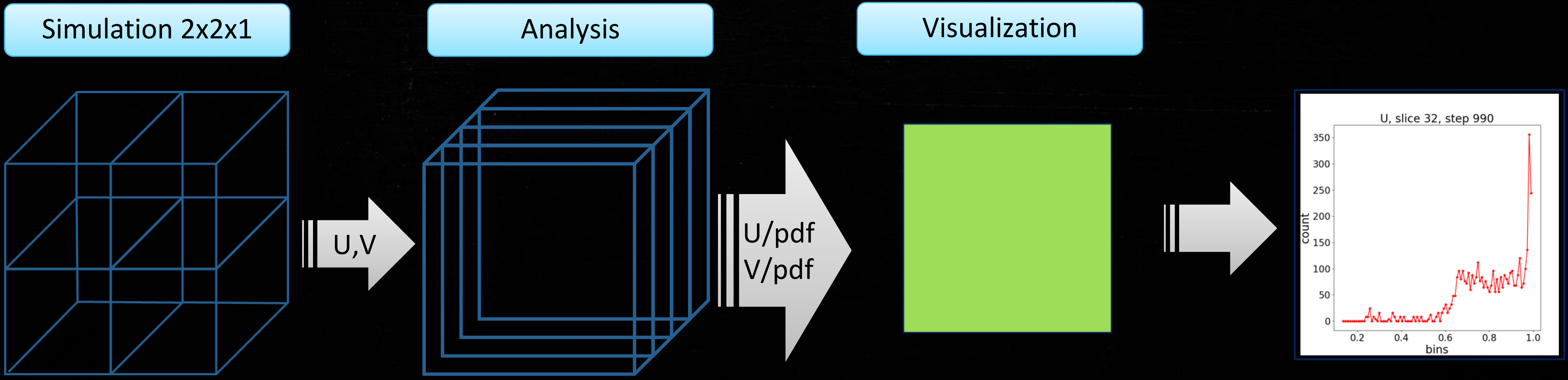
Gray-Scott Example

- In this example we start with a 3D code which writes 3D arrays, with a 3D domain decomposition, as shown in the figure.
  - Gray-Scott Reaction–diffusion system
  - https://en.wikipedia.org/wiki/Reaction%E2%80%93diffusion_system
  - https://github.com/ornladios/ADIOS2-Examples/tree/master/source/cpp/gray-scott
  - We write multiple time-steps, into a single output.
- For simplicity, we work on only 4 cores, arranged in a 2x2x1 arrangement.
- Each processor works on 32x32x64 subsets
- The total size of the output arrays = 4*64*64*64



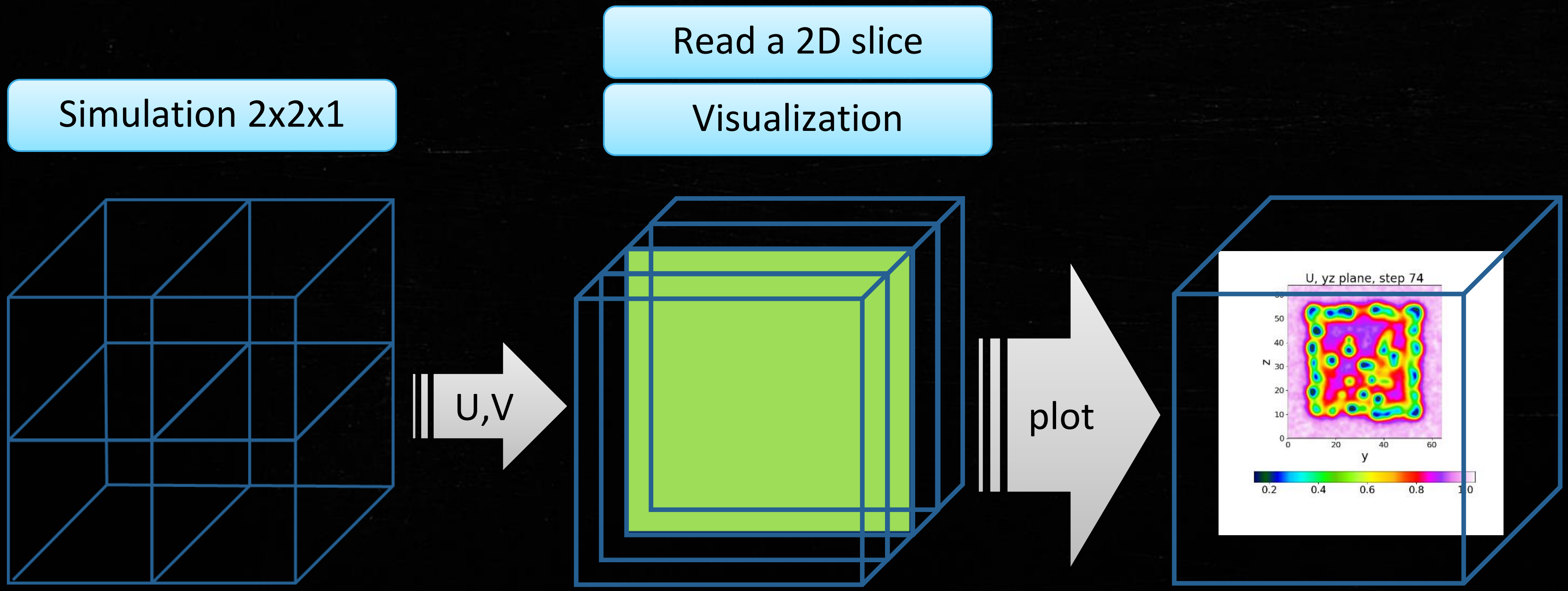
Analysis and visualization

- Read with a different decomposition (1D)
  - Calculate PDFs on a 2D slice of the 3D array
  - Read/Write U,V from M cores, arranged in a M x 1 x 1 arrangement.
- Plot U/pdf
  - image files



Visualization with plot/gsplot.py

- Read a 2D slice and plot it with matplotlib

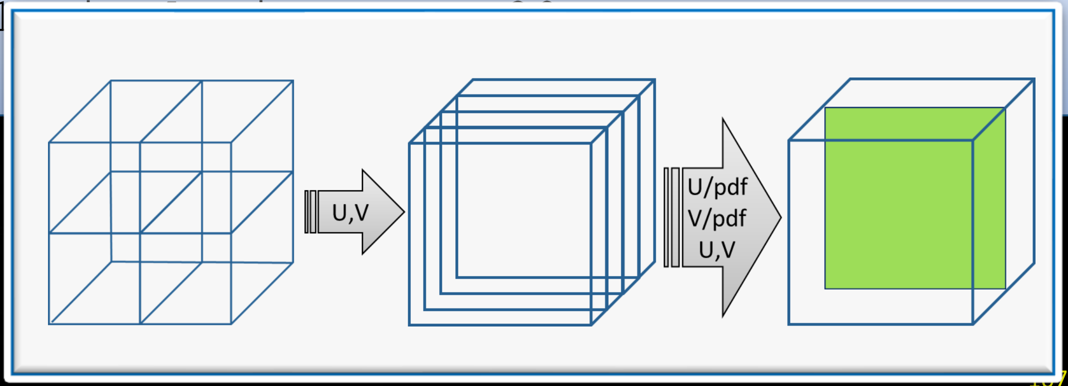


Goal by the end of the day: Running the example in situ

```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
Simulation at step 10 writing output step     1
Simulation at step 20 writing output step     2
Simulation at step 30 writing output step     3
Simulation at step 40 writing output step     4
...
```

```
$ mpirun -n 1 adios2-pdf-calc gs.bp pdf.bp 100
PDF Analysis step 0 processing sim output step 0 sim compute step 10
PDF Analysis step 1 processing sim output step 1 sim compute step 20
PDF Analysis step 2 processing sim output step 2 sim compute step 30
...
```

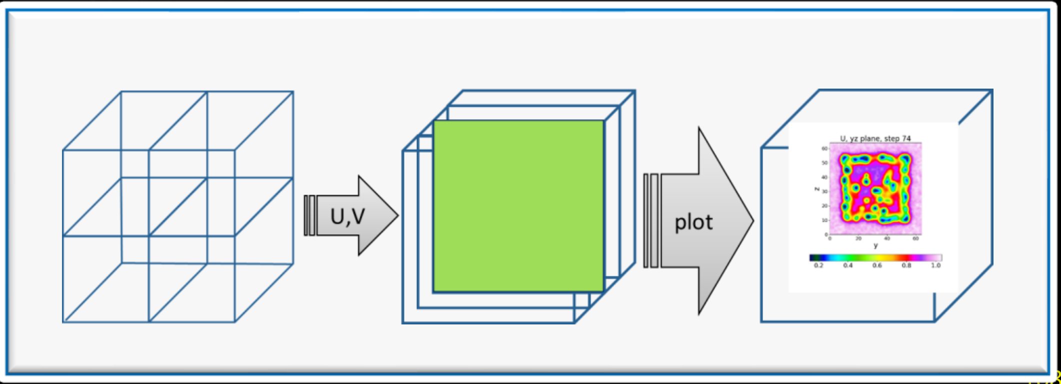
```
$ mpirun -n 1 python3 pdfplot.py -i pdf.bp
PDF Plot step 0 processing analysis step 0 simulation step 10
PDF Plot step 1 processing analysis step 1
...
```



Goal by the end of the exercise: Running the example in situ

```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
Simulation at step 10 writing output step     1
Simulation at step 20 writing output step     2
Simulation at step 30 writing output step     3
Simulation at step 40 writing output step     4
...
```

```
$ mpirun -n 1 python3 gsplot.py -i gs.bp
GS Plot step 0 processing stream step 0 sim step 10 minmax=0.0765537..1.05494
GS Plot step 1 processing stream step 1 sim step 20 minmax=0.111669..1.05908
...
```



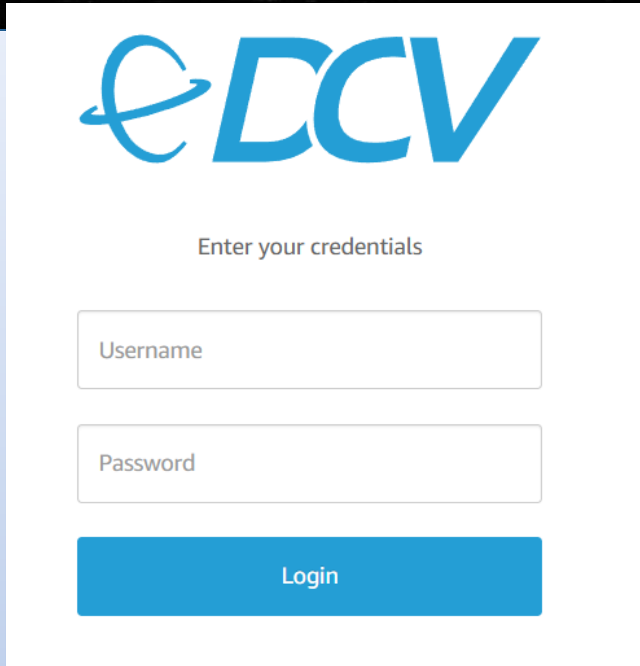
Login to a virtual machine

https://tut###.supercontainers.org:8443/#e4s

is a number assigned for you in this tutorial

Username: tutorial
Password: HPCLinux12!

Launch a couple of gnome-terminals



Configure the environment

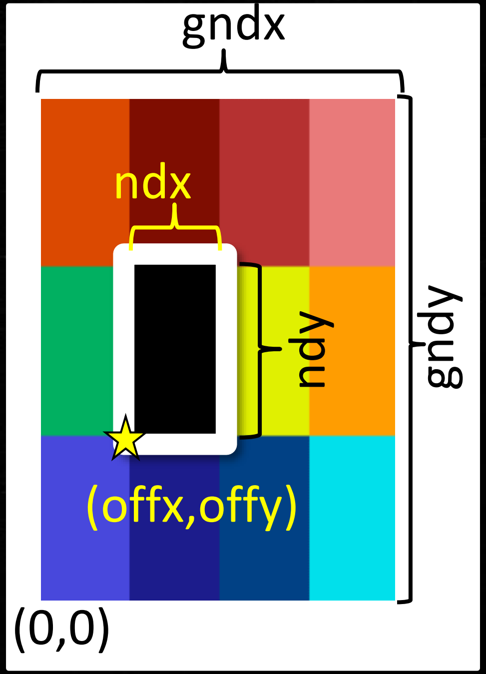
```
$ cd ~/adios2-tutorial-source
$ source ~/adios2-tutorial-source/adios2-tutorial-env.sh
$ cd ~/adios2-tutorial-source/ADIOS2-Examples/build/install/share/adios2-examples/gray-scott
```

Run the code

```
$ source ~/adios2-tutorial-source/adios2-tutorial-env.sh
$ cd ~/adios2-tutorial-source/ADIOS2-Examples/build/install/share/adios2-examples/gray-scott
$ mpirun -n 4 ../../../bin/adios2-gray-scott settings-files.json
Simulation writes data using engine type:              BP5
========================================
grid:             64x64x64
steps:            1000
plotgap:          10
F:                0.01
k:                0.05
dt:               2
Du:               0.2
Dv:               0.1
noise:            1e-07
output:           gs.bp
adios_config:     adios2.xml
process layout:   2x2x1
local grid size:  32x32x64
========================================
Simulation at step 10 writing output step     1
Simulation at step 20 writing output step     2
...
$ du -hs *.bp
401M    gs.bp
```

Gray-Scott Global Array

- N-dimensions
- Type
- Decomposition across many processors
  - global dimensions (Shape), local place (Start, Count)




```
ADIOS2-Examples/source/cpp/gray-scott/simulation
main.cpp:
  adios2::ADIOS adios(settings.adios_config, comm);
  adios2::IO io = adios.DeclareIO("SimulationOutput");

writer.cpp:
  var_u = io.DefineVariable<double>(
    "U",
    {settings.L, settings.L, settings.L},
    {sim.offset_z, sim.offset_y, sim.offset_x},
    {sim.size_z, sim.size_y, sim.size_x});
```

- Fortran/Matlab/R always column-major, C/C++/Python always row-major

bpls

- List content and print data from ADIOS2 output (.bp and .h5 files)
  - dimensions are reported in row-major order

```
$ bpls -la gs.bp
  double    Du        attr     = 0.2
  double    Dv        attr     = 0.1
  double    F         attr     = 0.01
  double    U         100*{64, 64, 64} = 0.0907898 / 1
  double    V         100*{64, 64, 64} = 0 / 0.674844
  double    dt        attr     = 2
  double    k         attr     = 0.05
  double    noise     attr     = 1e-07
  int32_t   step      100*scalar = 10 / 1000
```

bpls (to show the decomposition of the array)

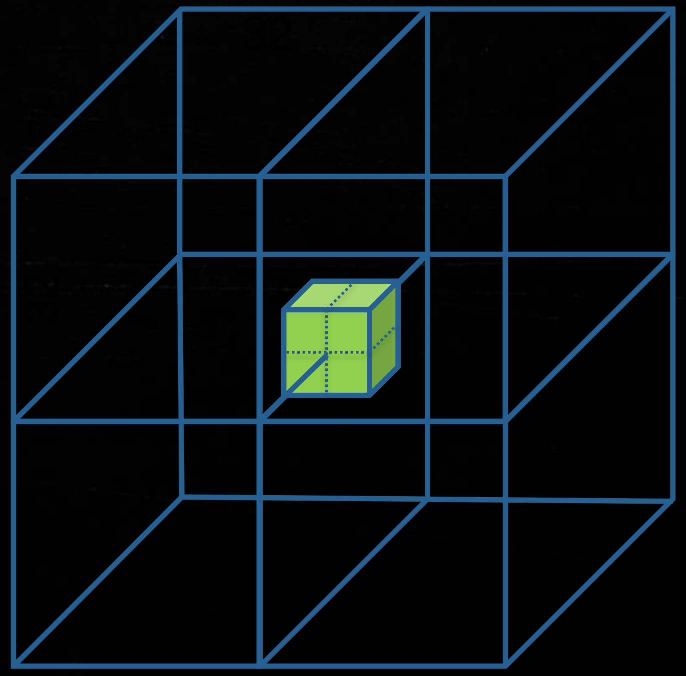
```
$ bpls -l -D gs.bp U
 double    U      100*{64, 64, 64} = 0.0907898 / 1
        step  0:
          block 0: [ 0:63,  0:31,  0:31] = 0.104002 / 1
          block 1: [ 0:63, 32:63,  0:31] = 0.104002 / 1
          block 2: [ 0:63,  0:31, 32:63] = 0.104002 / 1
          block 3: [ 0:63, 32:63, 32:63] = 0.104002 / 1
 ...
        step 99:
          block 0: [ 0:63,  0:31,  0:31] = 0.148308 / 0.998811
          block 1: [ 0:63, 32:63,  0:31] = 0.148302 / 0.998812
          block 2: [ 0:63,  0:31, 32:63] = 0.148335 / 0.998811
          block 3: [ 0:63, 32:63, 32:63] = 0.148302 / 0.998811
```

bpls to dump: 2x2x2 read with bpls

- Use bpls to read in the **center** 3D cube of the **last** output step

```
$ bpls gs.bp -d U -s "-1,31,31,31" -c "1,2,2,2" -n 2
double   U      100*{64, 64, 64}
   slice (99:99, 31:32, 31:32, 31:32)
    (99,31,31,31)    0.916973 0.916972
    (99,31,32,31)    0.916977 0.916975
    (99,32,31,31)    0.916974 0.916973
    (99,32,32,31)    0.916977 0.916976
```

- Note: bpls handles time as an extra dimension
- -s starting offset
  - first offset is the timestep, *-n to* count backwards
- -c size in each dimension
  - first value is how many steps
- -n how many values to print in one line



The ADIOS XML configuration file

- Describe runtime parameters for each IO grouping
  - select the Engine for writing
    - **BP5**, **HDF5**, **SST**
- see ~/Tutorial/share/adios2-examples/gray-scott/adios2.xml
- XML-free: engine can be selected in the source code as well

The runtime config file: adios2.xml

```xml
<?xml version="1.0"?>
<adios-config>

 <!--=======================================
       Configuration for for Gray-Scott and GS Plot
    =========================================-->
  <io name="SimulationOutput">
    <engine type="BP5">
      <parameter key="OpenTimeoutSecs" value="10.0"/>
      <parameter key="NumAggregators" value="2"/>
    </engine>
  </io>
```

Engine types

BP5

BP4

HDF5

FileStream

SST

SSC

DataMan

```xml
   <!--=======================================
       Configuration for PDF calc and PDF Plot
    =======================================-->

   <io name="PDFAnalysisOutput">
     <engine type="BP5">
     </engine>
   </io>

</adios-config>
```

The runtime config file: adios2.xml

adios2.xml

```xml
<?xml version="1.0"?>
<adios-config>

  <!--====================================
      Configuration for for Gray-Scott and GS Plot
    ======================================-->
  <io name="SimulationOutput">
    <engine type="HDF5">
    </engine>
  </io>
```

Engine types
BP5
HDF5
FileStream
SST
SSC
DataMan

settings-files.json

```json
{
    "L": 64,
    "Du": 0.2,
    "Dv": 0.1,
    "F": 0.01,
    "k": 0.05,
    "dt": 2.0,
    "plotgap": 10,
    "steps": 1000,
    "noise": 0.0000001,
    "output": "gs.bp",
    "checkpoint": false,
    "checkpoint_freq": 10,
    "checkpoint_output": "gs_ckpt.bp",
    "adios_config": "adios2.xml",
    "adios_span": false,
    "adios_memory_selection": false,
    "mesh_type": "image"
}
```

gs.h5

Run the code

```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
$ mpirun -n 4 adios2-gray-scott settings-files.json
Simulation writes data using engine type:              HDF5
========================================
grid:             64x64x64
steps:            1000
plotgap:          10
F:                0.01
k:                0.05
dt:               2
Du:               0.2
Dv:               0.1
noise:            1e-07
output:           gs.bp
adios_config:     adios2.xml
process layout:   2x2x1
local grid size:  32x32x64
========================================
Simulation at step 10 writing output step     1
Simulation at step 20 writing output step     2
...
$ du -hs *.h5
401M    gs.h5
```

List the content

```
$ h5ls -r gs.h5
/                        Group
/Step0                   Group
/Step0/U                 Dataset {64, 64, 64}
/Step0/V                 Dataset {64, 64, 64}
/Step0/step              Dataset {SCALAR}
/Step1                   Group
/Step1/U                 Dataset {64, 64, 64}
/Step1/V                 Dataset {64, 64, 64}
/Step1/step              Dataset {SCALAR}
...

$ h5ls -d gs.h5/Step1/U
```

bpls can read HDF5 files as well

```
$ bpls gs.bp -d U -s "-1,31,31,31" -c "1,2,2,2" -n 2
double    U      100*{64, 64, 64}
   slice (99:99, 31:32, 31:32, 31:32)
    (99,31,31,31)     0.916973 0.916972
    (99,31,32,31)     0.916977 0.916975
    (99,32,31,31)     0.916974 0.916973
    (99,32,32,31)     0.916977 0.916976
```

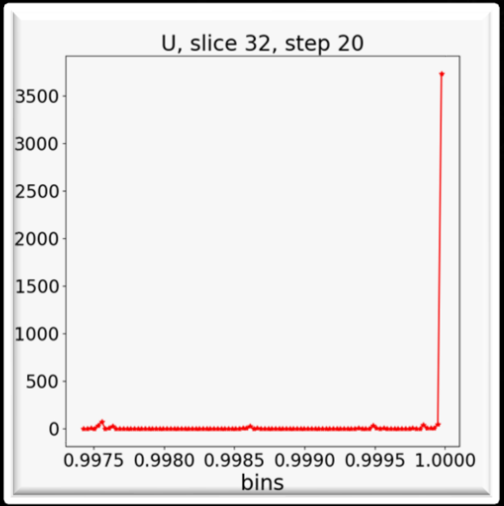
```
$ bpls gs.h5 -d U -s "-1,31,31,31" -c "1,2,2,2" -n 2
double    U      100*{64, 64, 64}
   slice (99:99, 31:32, 31:32, 31:32)
    (99,31,31,31)     0.916973 0.916972
    (99,31,32,31)     0.916977 0.916975
    (99,32,31,31)     0.916974 0.916973
    (99,32,32,31)     0.916977 0.916976
```

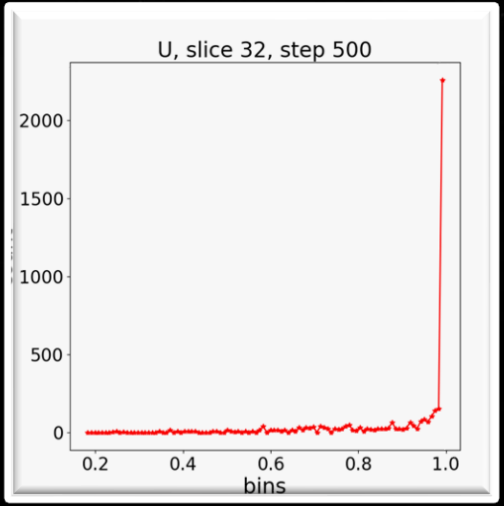
Compile and run the reader

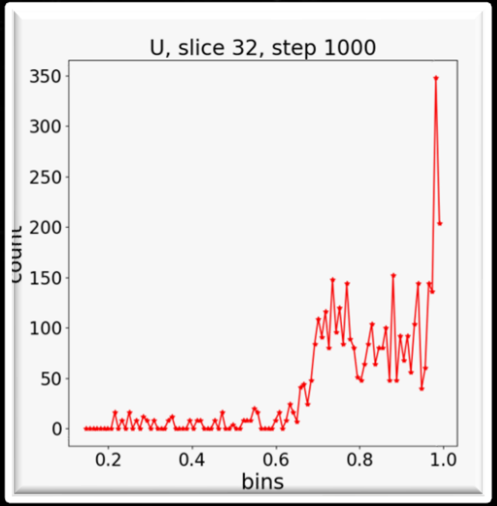
```
# make sure in adios2.xml, SimulationOutput's engine is set to BP5
# make sure in settings-files.json "output" is set to "gs.bp"
$ mpirun -n 3 ../../../bin/adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  BP5
PDF analysis writes using engine type:                 BP5
PDF Analysis step 0 processing sim output step 0 sim compute step 10
PDF Analysis step 1 processing sim output step 1 sim compute step 20
PDF Analysis step 2 processing sim output step 2 sim compute step 30
...
$ bpls -l  pdf.bp
  double    U/bins  100*{100} = 0.0908349 / 1
  double    U/pdf   100*{64, 100} = 0 / 4096
  double    V/bins  100*{100} = 0 / 0.668077
  double    V/pdf   100*{64, 100} = 0 / 4096
  int32_t   step    100*scalar = 10 / 1000
$ bpls -l  pdf.bp -D U/pdf
 double    U/pdf   100*{64, 100} = 0 / 4096
        step  0:
          block 0: [ 0:20,  0:99] = 0 / 894
          block 1: [21:41,  0:99] = 0 / 4096
          block 2: [42:63,  0:99] = 0 / 4096
        step  1:
```

Run the plotting script with file I/O

```
$ python3 pdfplot.py -i pdf.bp -o p
PDF Plot step 0 processing analysis step 0 simulation step 10
PDF Plot step 1 processing analysis step 1 simulation step 20
PDF Plot step 2 processing analysis step 2 simulation step 30
...
$ ls *.png
p00010_32.png  p00100_32.png  p00190_32.png ... p01000_32.png
$ eog . &
```

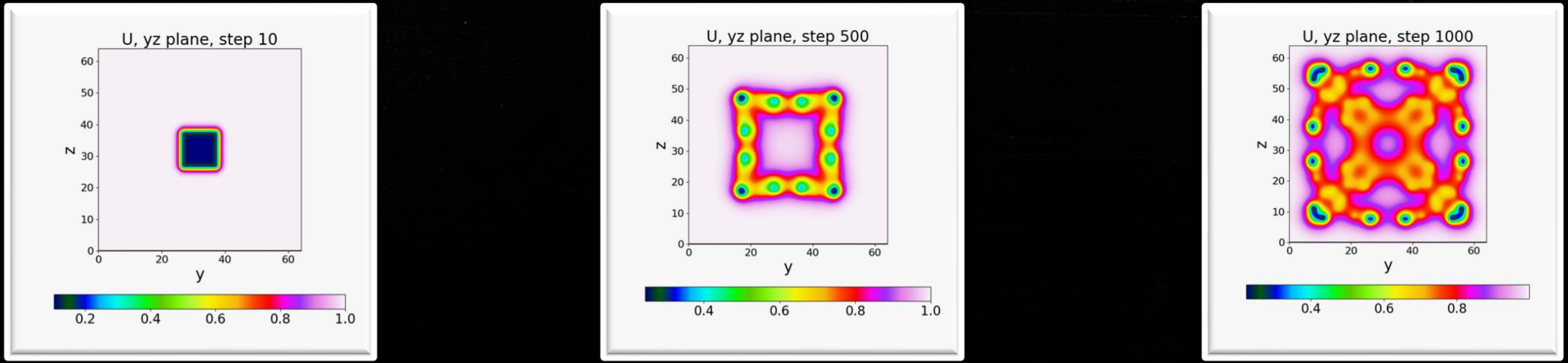






Run the plotting script with file I/O

```
$ python3 gsplot.py -i gs.bp -o g
PDF Plot step 0 processing analysis step 0 simulation step 10
PDF Plot step 1 processing analysis step 1 simulation step 20
PDF Plot step 2 processing analysis step 2 simulation step 30
...
$ ls g*.png
g00010_32.png  g00100_32.png  g00190_32.png ... g01000_32.png
$ eog . &
```



Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

How to scale ADIOS I/O

ADIOS Scaling for large parallel file systems

- There are a **few** options for changing the performance of ADIOS for your code on all HPC systems
  - Aggregation – choosing the number of sub-files for maximizing the filesystem bandwidth
  - Appending multiple steps into a single output for minimizing the filesystem metadata overhead
  - Asynchronous write with BP5 engine
  - Choosing the "best" ADIOS engine/storage system (staging, NVRAM-flush) for minimizing the variability

ADIOS Scaling for large parallel file systems: number of files

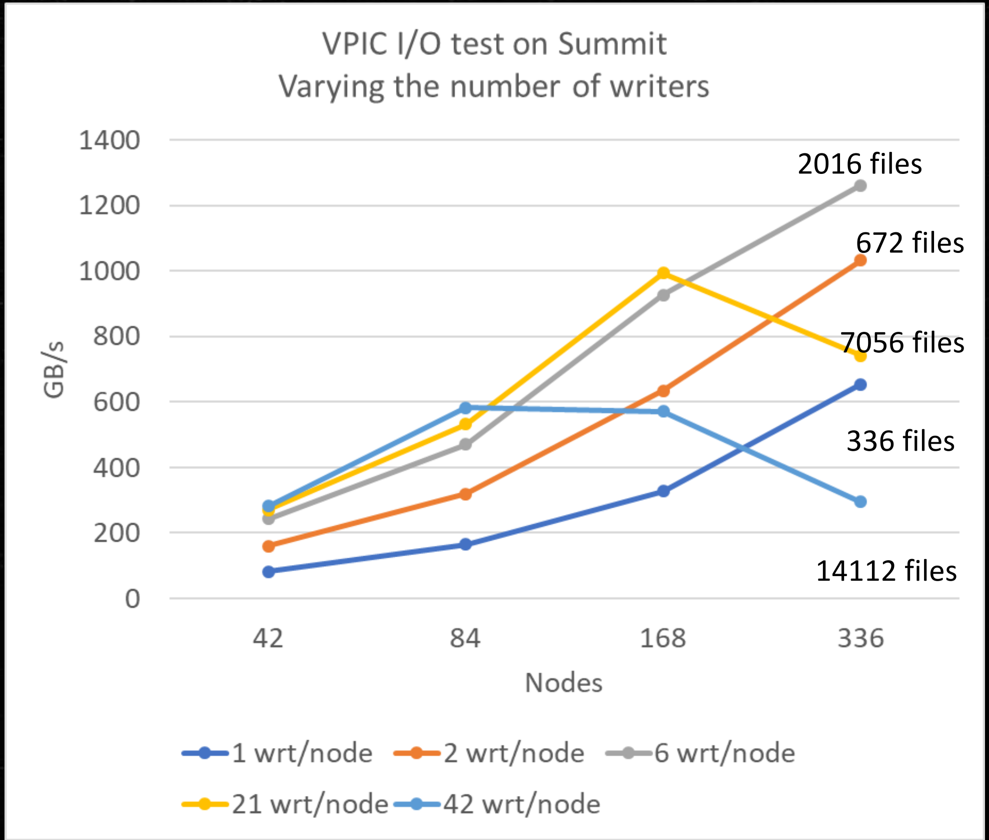
- Not good:
  - Single, global output file from many writers (or for many readers)
    - Bottleneck at file access
  - One file per process
    - Chokes the metadata server of the file system at create
    - Reading from different number of processes is very difficult
- Good:
  - Create K subfiles where K is proportioned to the capability of the file system, not the number of writers/readers
- ADIOS BP engine options
  - **NumAggregators**
  - **AggregatorRatio**
- ADIOS default settings
  - One file per compute node

```xml
<io name="SimulationOutput">
  <engine type="BP5">
    <parameter key="NumAggregators" value="2048"/>
  </engine>
</io>
```

Example: VPIC I/O test on Summit

- A fixed aggregation ratio breaks down as we scale up the nodes
- Best options here:
  - 42 nodes – 42*42=1764 subfiles (1:1)
  - 84 nodes – 84*42=3528 subfiles (1:1)
  - 168 nodes – 168*21=3528 subfiles (1:2)
  - 336 nodes – 336*6=2016 subfiles (1:7)
- Summit general guidance
  - One subfile per GPU (6 per node) is a good start but apps usually have one MPI task/GPU, so keep the total number of subfiles below 4000

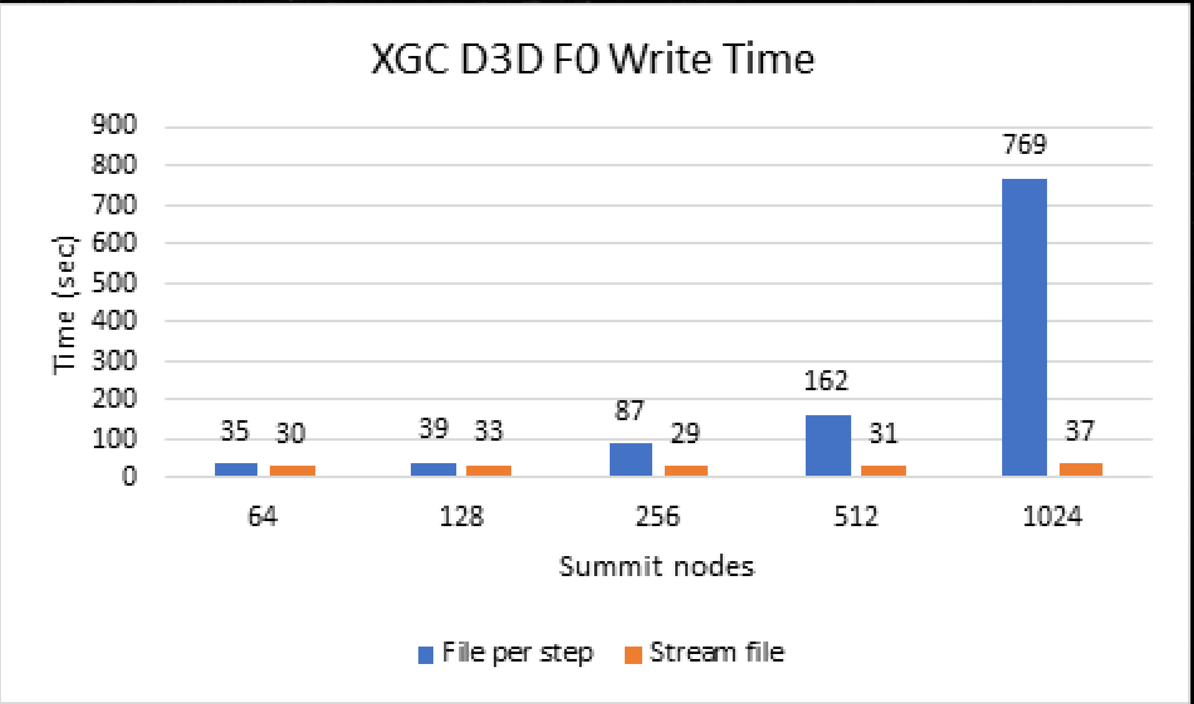
chart courtesy of Junmin Gu, LBNL



Application	Nodes/GPUs, 6 tasks/node	Data Size per step	I/O speed	ADIOS NumAggregators
SPECFEM3D_GLOBE	3200/19200	250 TB	~2 TB/sec	3200 (1:6 aggregation ratio)
GTC	512/3072	2.6 TB	~2 TB/sec	3072 (1:1 aggregation ratio)
XGC	512/3072	64 TB	1.2 TB/sec	1024 (1:3 aggregation ratio)
LAMMPS	512/3072	457 GB	1 TB/sec	512 (1:6 aggregation ratio)

ADIOS Scaling for large parallel file systems: number of steps

- Another aspect of number of files: number of output steps
- New output every step -> many files over the course of simulation
  - If the rate of steps stresses the file system, write performance will drop
  - Actually, the total time of creation of files will add up
- Appending multiple output steps into same file is better



XGC 100 output steps in 1245 second simulation, 40 GB per step (4TB total)

One file per node created in each step

Single output for all steps (one file per node)

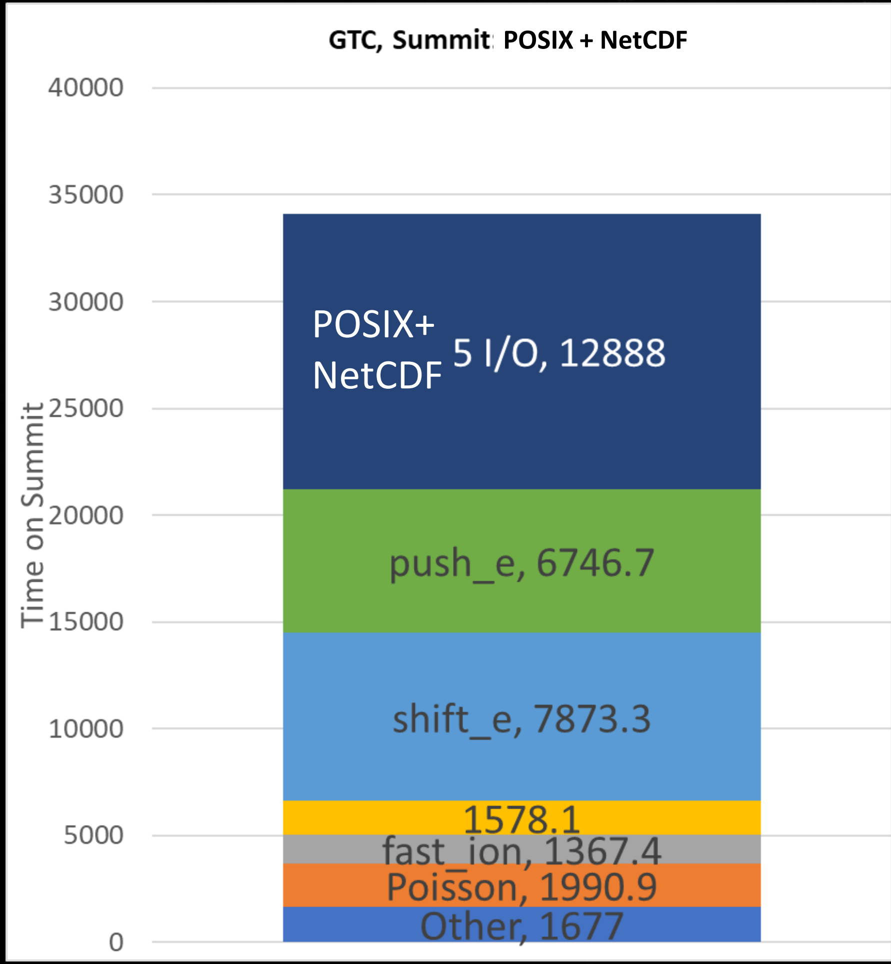
GTC Original I/O vs. ADIOS – Comparing the no. of files created

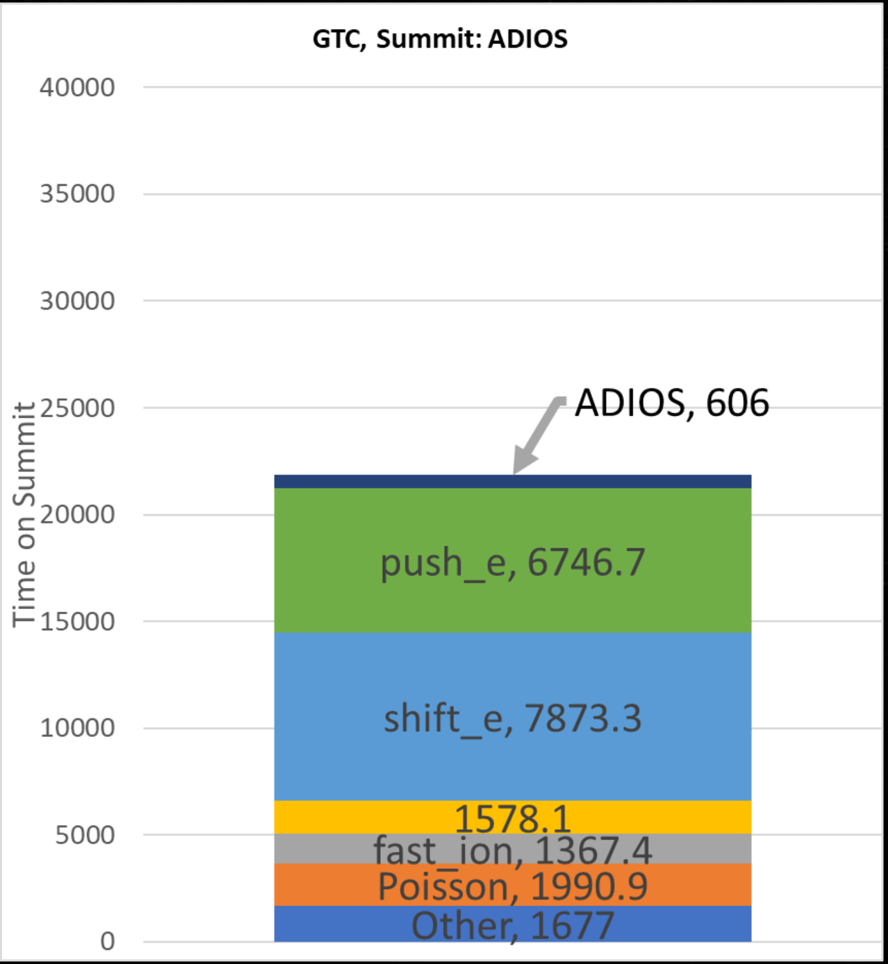
10k steps, 512 nodes (3072 ranks)

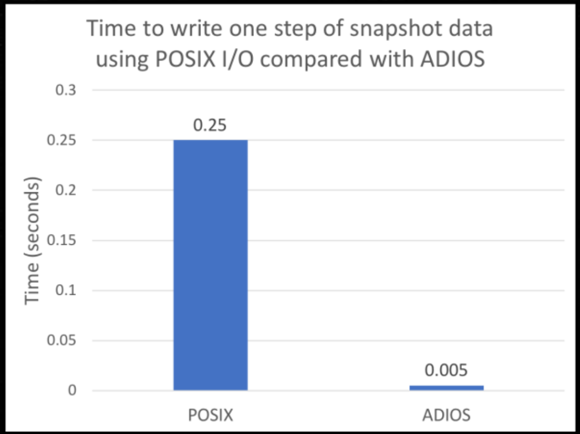
Data Category	Data Subcategory	Number of files in the original code	Number of files in the ADIOS container (1 writer per node)
diagnostics	equilibrium	1	1
diagnostics	data1d	1	1
diagnostics	history	1	1
snapshot	snapshot	**10,000** (1 file per step)	**1**
field data	phi3d	**32,000** (1 file every 10 steps per toroidal root)	**512** (1 file per node)
checkpoint	restart1	**2 * 3072** (1 file per rank)	**2 * 512** (1 file per node)

Optimized GTC, a fusion PIC code, I/O on Summit

PI: Zhihong Lin, UC Irvine







* average cost when measuring 10 000 steps

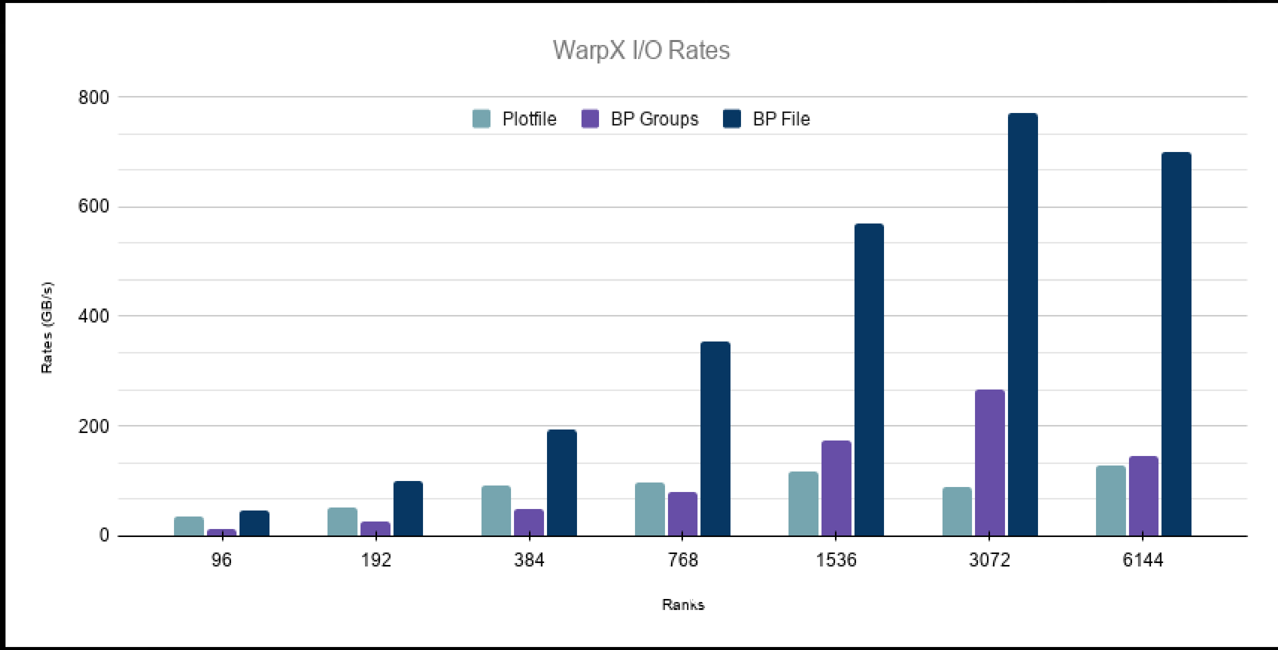
- Change to ADIOS I/O: Total simulation time reduced from 9.5 hours to 6.1 hours on 1024 nodes on Summit

WarpX example: appending + aggregation

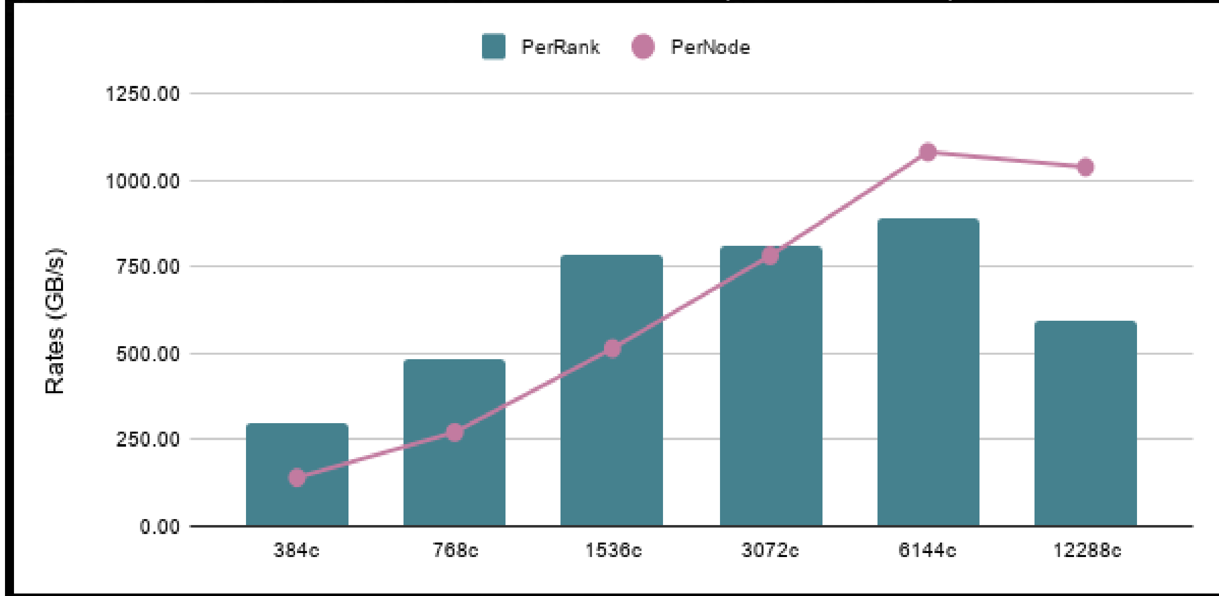
PI: Jean-Luc Vay, LBNL

- One file per compute node (6 processes on Summit)
- Better performance at 512 nodes and over
- One file per rank at smaller jobs

WarpX on Summit, Original vs
ADIOS new output per step vs
ADIOS single output



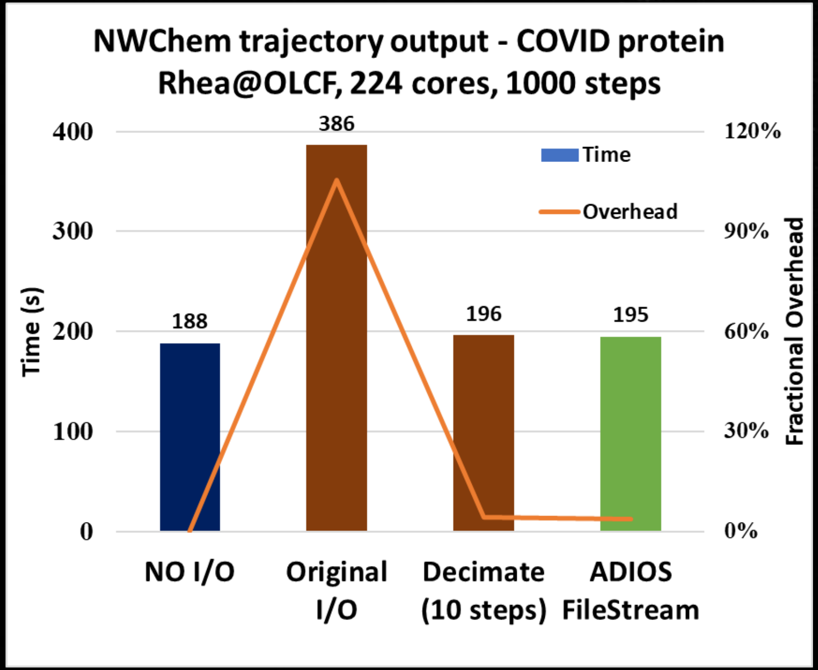
WarpX on Summit, 6 rank per node setup
240 TB on 1024 nodes (6144 cores)



240 TB on 1024 nodes (6144 cores)

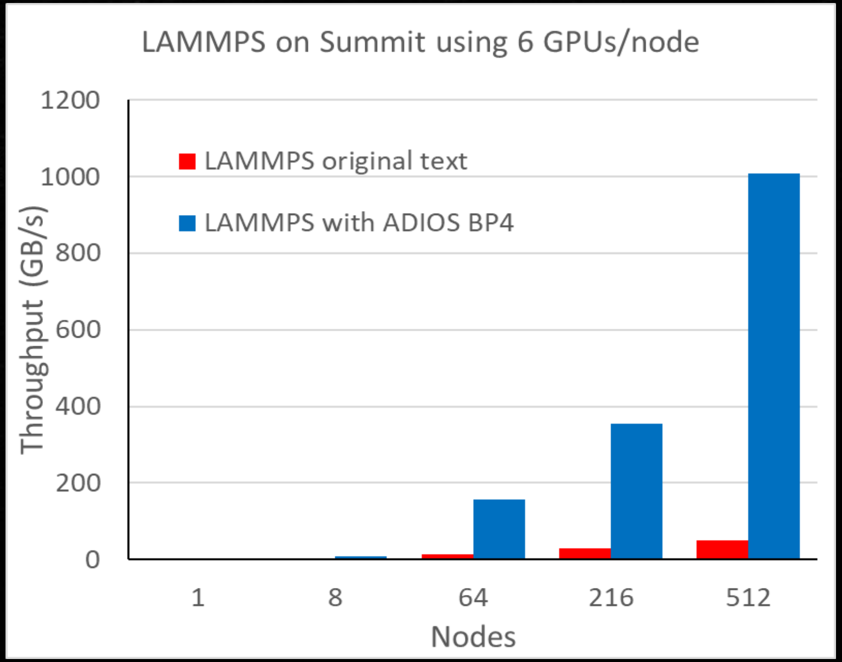
Single process I/O examples: NWChem, LAMMPS

- Moves data between processes as part of preparation for I/O
  - "I am doing POSIX/Fortran I/O on rank 0"
  - While gathering data, no one is writing
- Single file output – not utilizing available bandwidth



ADIOS can write all steps out with little cost
(here every 0.2 seconds)

Summit 512 nodes
12B atoms, 5 TB

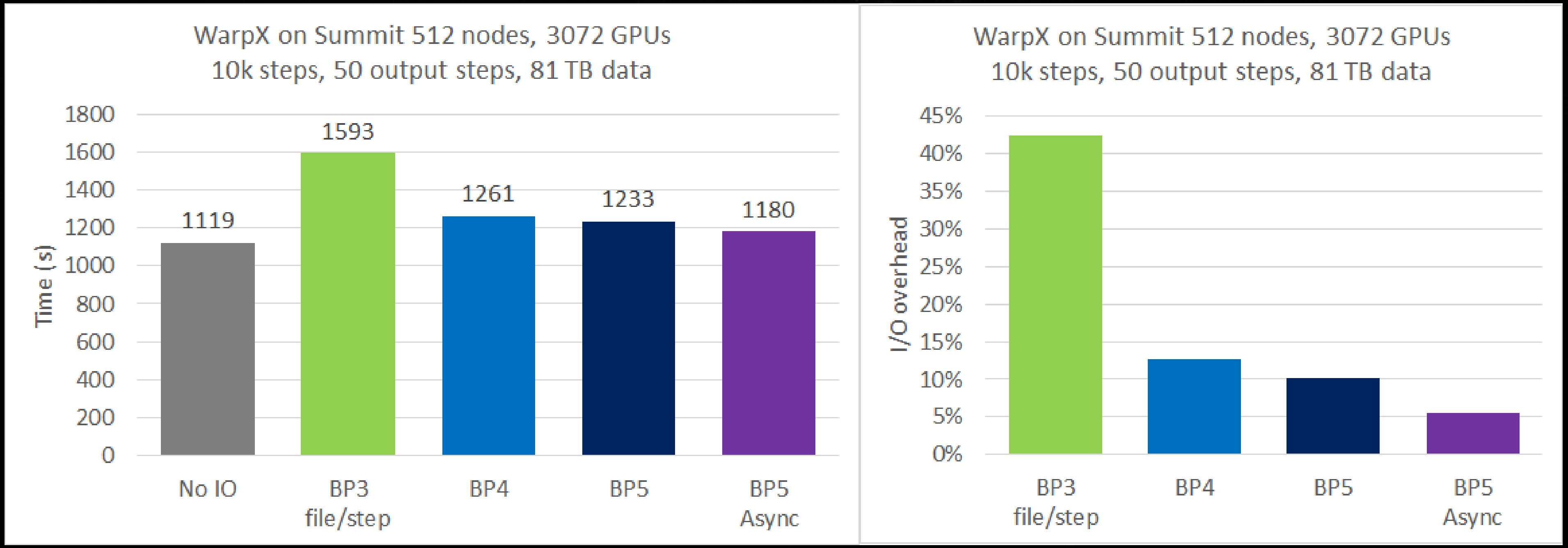


https://github.com/lammps/lammps/tree/master/src/USER-ADIOS

- USER-ADIOS package in LAMMPS for dump commands
  dump atom/adios dump custom/adios
- Output goes into an I/O stream
  BP4 file by default Can use staging engines

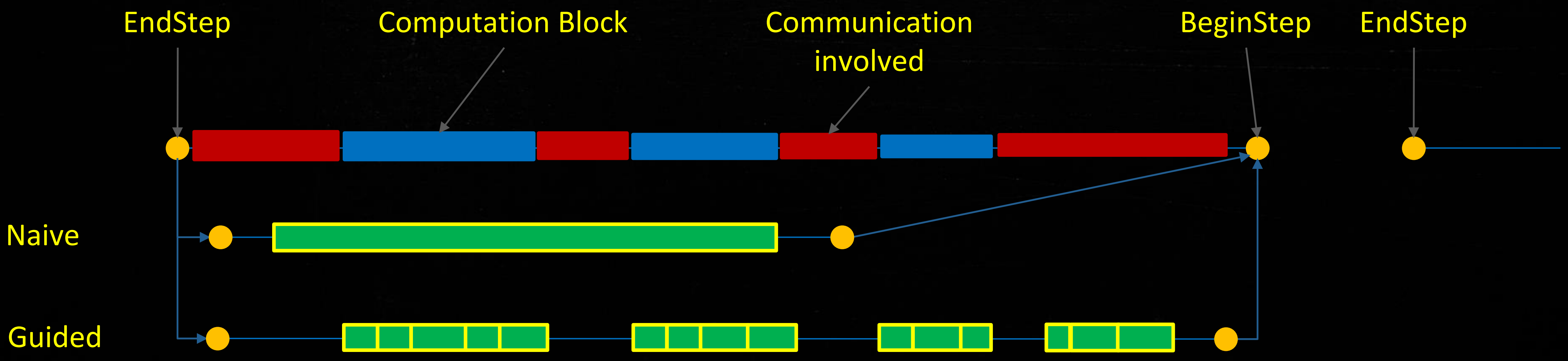
Asynchronous write to storage (BP5 engine, since 2.8.0 release)

- User friendly On/Off option
  - No need to modify the user code
- Only data writing is async, metadata gathering and writing is still sync
- Don't have too much experience with this yet



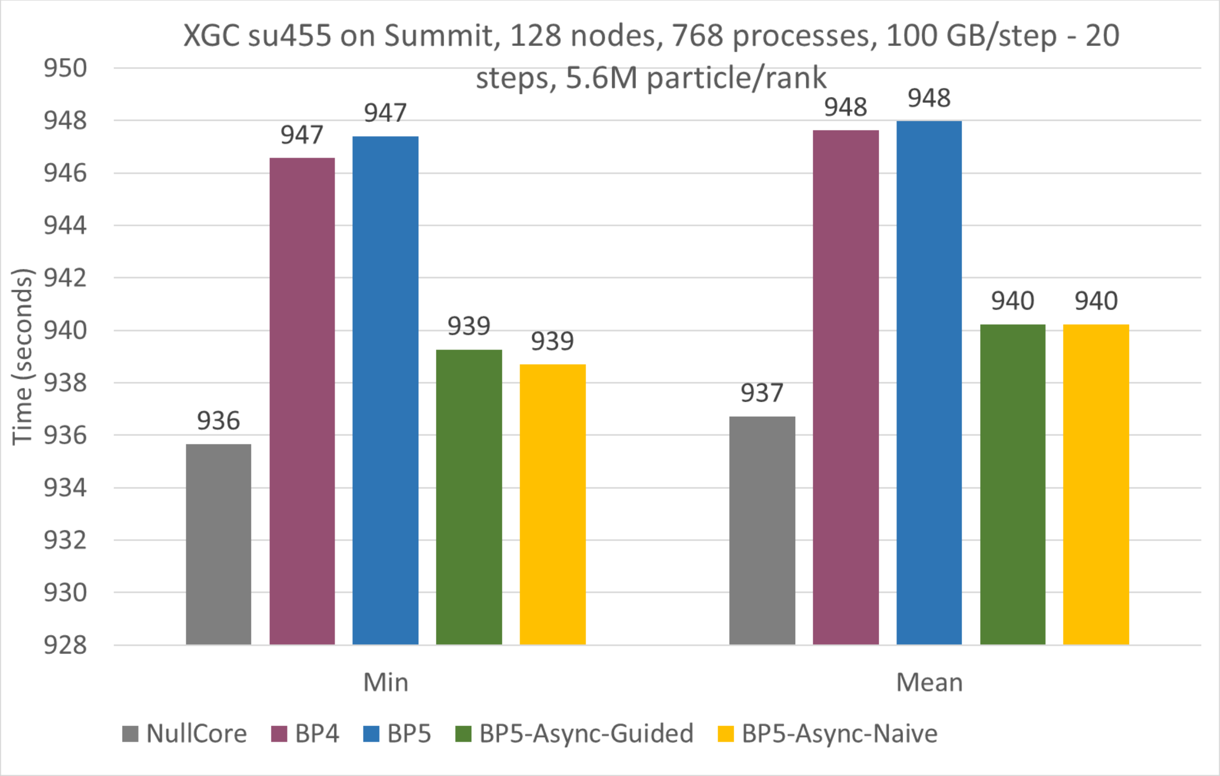
Async strategies

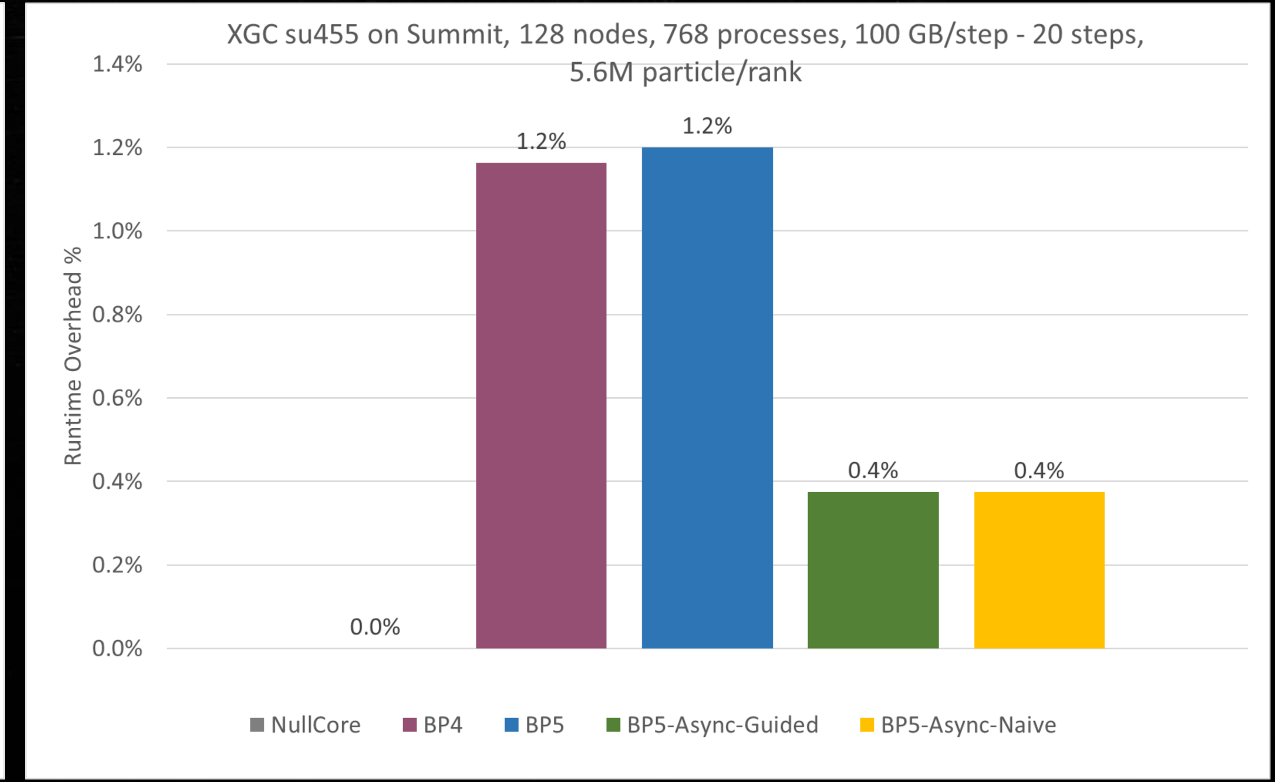
- Naive
  - dump data without thinking
- Guided
  - Application augmented with EnterComputationBlock/ExitComputationBlock pairs
  - Attempt writing during computation blocks
  - Naïve dump when running out of (forecasted) computation blocks



Async IO with XGC only small data

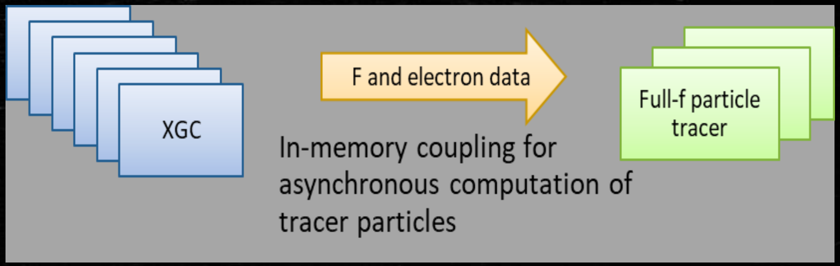
- This is a small D3D run case and quick test



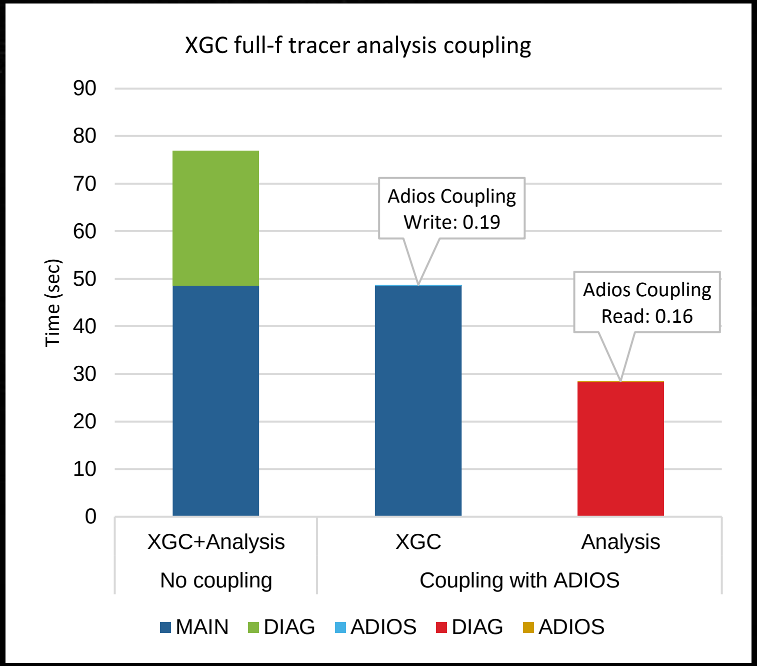


Staging use case: off-load non-scaling code part

- Tracer particle analysis enables understanding of the transport characteristics spanning the pedestal and scrape-off layer
- It is costly to perform and is communication-heavy
- Asynchronously stage data to the tracer particle analysis running on additional nodes
  - Coupled data size: f0(95 GB) + E_rho/pot_rho(1.4GB)
- Reduced 36% of the XGC iteration time by using asynchronous services (only 0.4% time-overhead for coupling data)



Full-f particle analysis coupling



Full-f coupling performance on Summit with ADIOS using 4/1024 extra nodes

Choi, J. Y., Chang, C. S., Dominski, J., Klasky, Churchill, M., S., Merlo, G., Suchyta, E., ... & Wood, C. “Coupling exascale multiphysics applications: Methods and lessons learned”. IEEE e-Science, 2018.

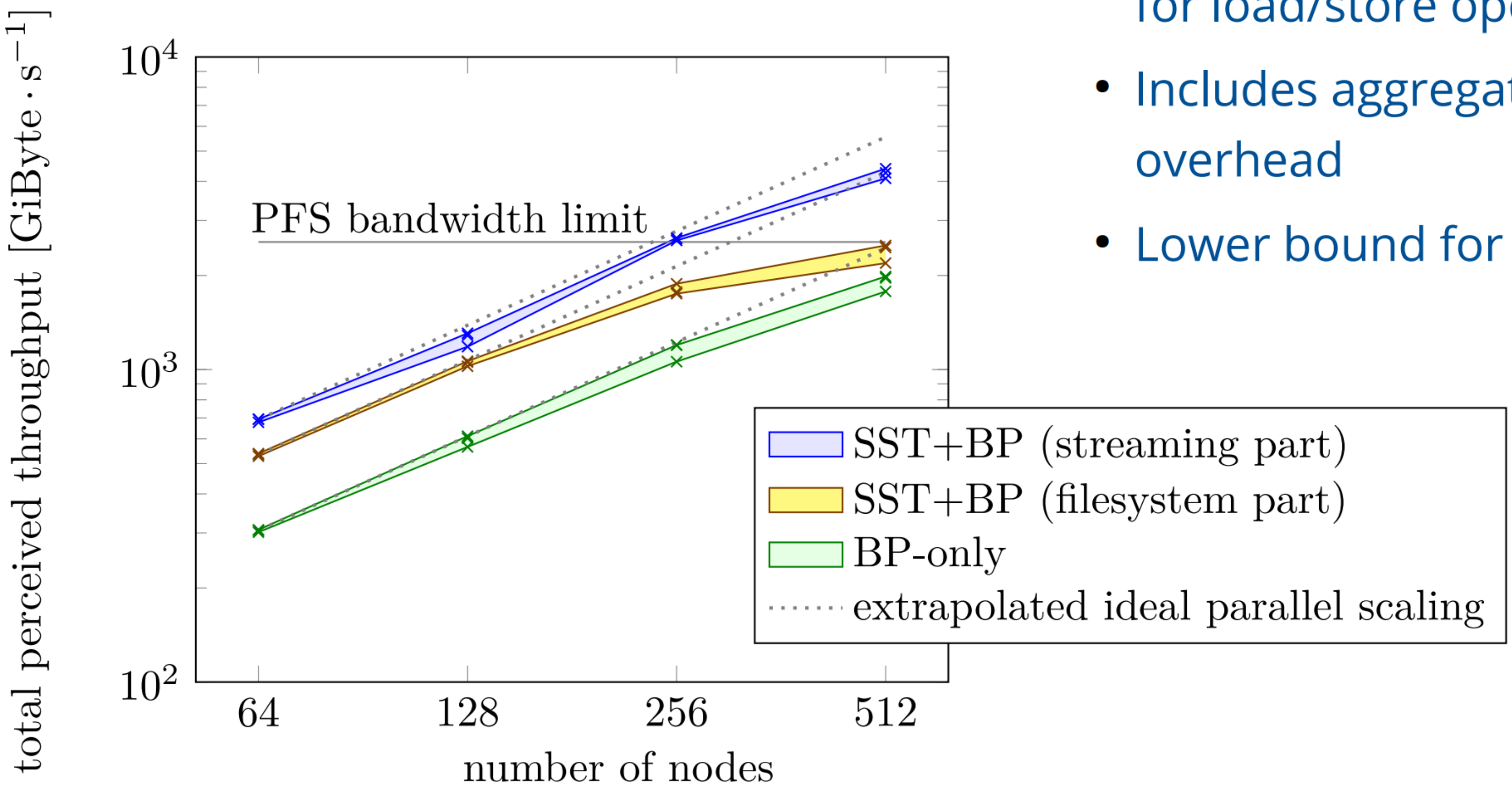
HBPS High-fidelity Boundary Plasma Simulation

How to get beyond the storage bandwidth limit

Staging performance

High throughput of Infiniband Streaming on Summit

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

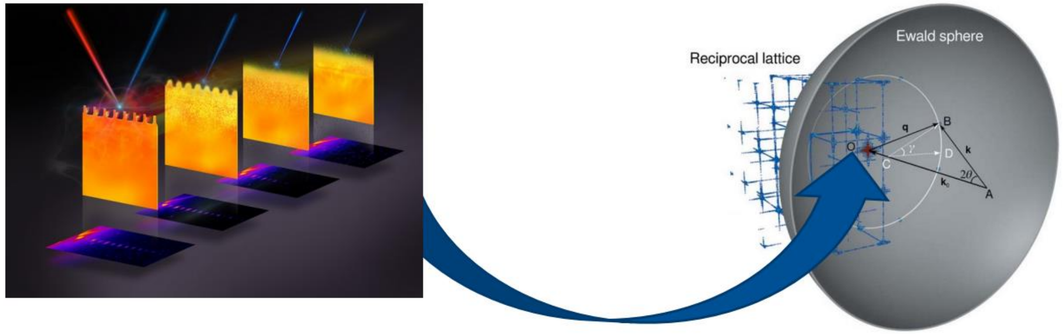


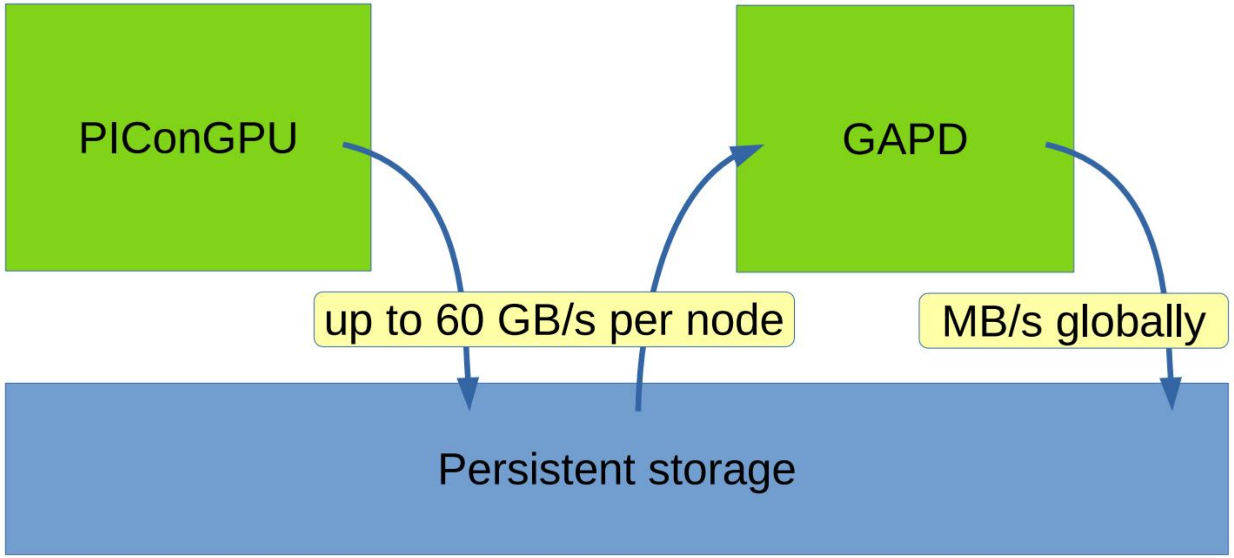
Perceived throughput:

- Based on time measured in application for load/store operation
- Includes aggregation and communication overhead
- Lower bound for precise throughput

Staging performance

Circumvent I/O bottleneck by loose coupling





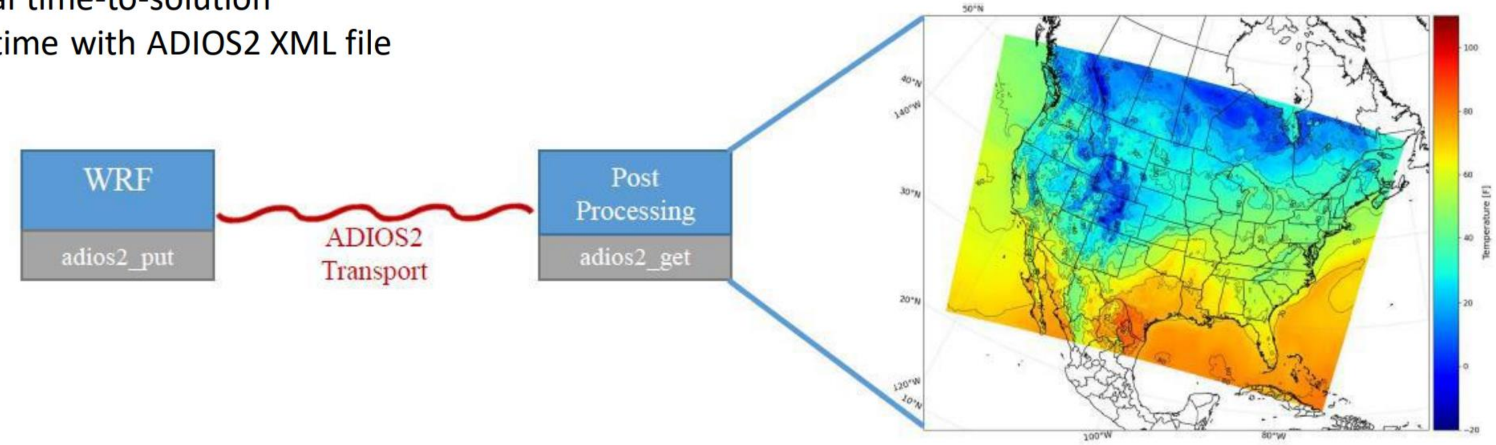
- Simulation pipeline: PIConGPU → GAPD
- Data description in openPMD is **independent of implementation**
- Use legacy, file-IO based implementations, but toggle a **streaming-aware backend**
- **GAPD**:
  Scattering analysis that relies only on particle data
  → Field data need not be sent
- Only **store the final result persistently**

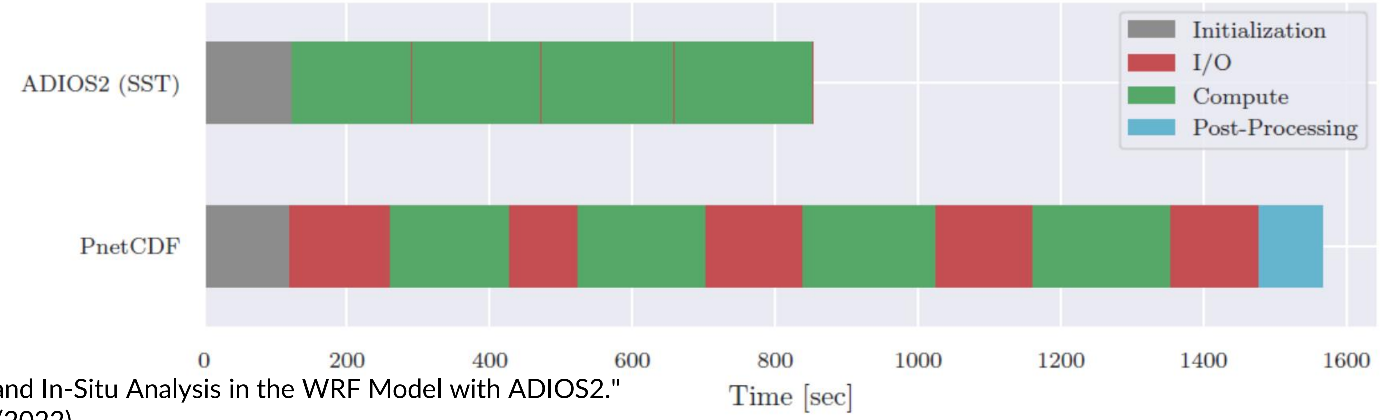
J. C. E, L. Wang, S. Chen, Y. Y. Zhang and S. N. Luo. "GAPD: a GPU- accelerated atom-based polychromatic diffraction simulation code". In: Journal of Synchrotron Radiation 25.2 (Mar. 2018), pp. 604–611.

WRF-ADIOS2 In-Situ Analysis

M. Laufer, E. Fredj (2022)

- Proof-of-concept post processing analysis pipeline built using ADIOS2 high-level Python API
- Data is post-processed as soon as it is available over the network (not filesystem)
- Greatly reduces total time-to-solution
- Configurable at runtime with ADIOS2 XML file





Laufer, Michael, and Erick Fredj.
"High Performance Parallel I/O and In-Situ Analysis in the WRF Model with ADIOS2."
arXiv preprint arXiv:2201.08228 (2022).

Erick Fredj@2022
e-mail: fredj@jct.ac.il

Slide courtesy of:
Prof. Erick Fredj
LEV Academic Center, Israel
Rutgers University, New Jersey

Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

i am hpc.

Paraview

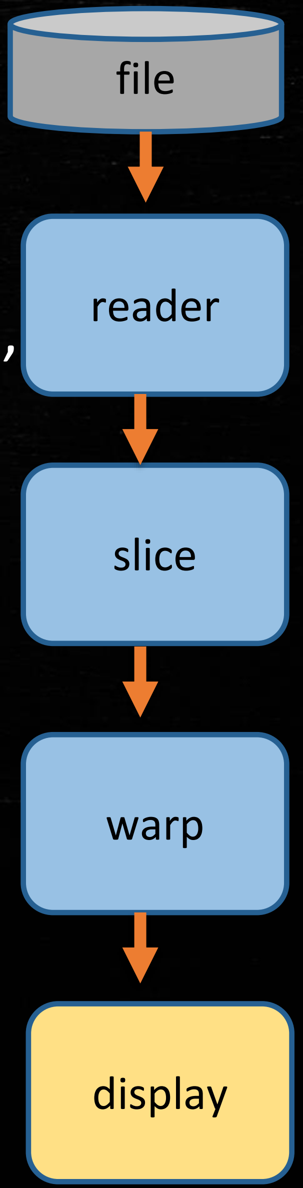
Caitlin Ross, Kitware Inc

ParaView

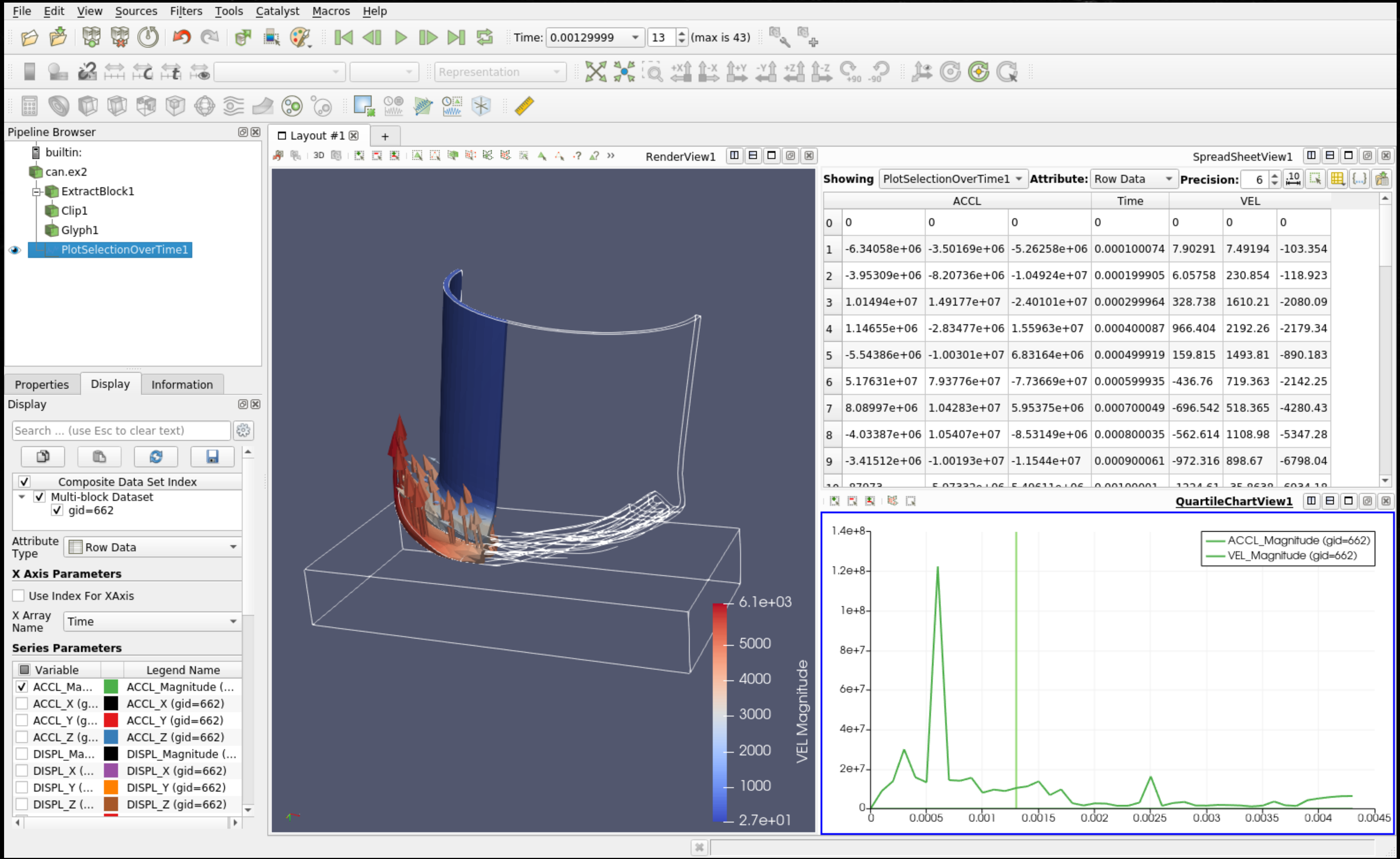
- Open-source software for visualization of large-scale scientific data
  - Supported by an active user and developer community
- Leverages parallel data processing and rendering to enable interactive visualization for extremely large datasets
  - Scales from laptops to supercomputers
  - Client/server: run the ParaView server on a supercomputer and connect the client on your laptop to it
- Can be extended and customized as necessary
- Supports scripting and batch processing using Python

ParaView's Visualization Pipeline

- Read in data
  - Many different file formats supported
  - Handles structured, unstructured, polygonal, graph, time-varying data,
- Build a pipeline to process your data
  - Choose from a number of different filters
  - Tune filter parameters
- Can save results in a variety of formats



ParaView User Interface



Python Scripting with ParaView

- Two Python APIs
  - Python wrapping allows direct access to proxies and their properties
  - paraview.simple module has helper methods that simplifies the API
    - from paraview.simple import *
- Why use Python scripting?
  - Running in batch mode
    - Set up your script on a small representative dataset locally
    - Then use the script on real data on supercomputer

Built-in Python Interpreters

- Shell in GUI
  - Can save trace of actions taken in GUI (e.g., setting up filters)
  - Visual feedback from running shell commands make it easiest place to learn ParaView’s Python API
- pvpython
  - meant for interactive scripts
- pvbatch
  - For batch processing
  - Can be run in parallel with MPI
  - Cannot interact with it
- All three have the python path set automatically

Creating a Simple Visualization

- Create a cone
  - `>>> myCone = Cone()`
- Get docs on properties
  - `>>> help(myCone)`
- Examine a property
  - `>>> myCone.Center`
- Change a property
  - `>>> myCone.Center = [0, 0, 1]`
- Show the Result
  - `>>> Show(myCone)`
  - `>>> Render()`
- Add a filter
  - `>>> clip1 = Clip()`
- List properties
  - `>>> clip1.ListProperties()`
- Change a property
  - `>>> clip1.ClipType.Normal = [0,1,0]`
- Show the filter, hide the cone
  - `>>> Show(clip1)`
  - `>>> Hide(myCone)`
  - `>>> Render()`

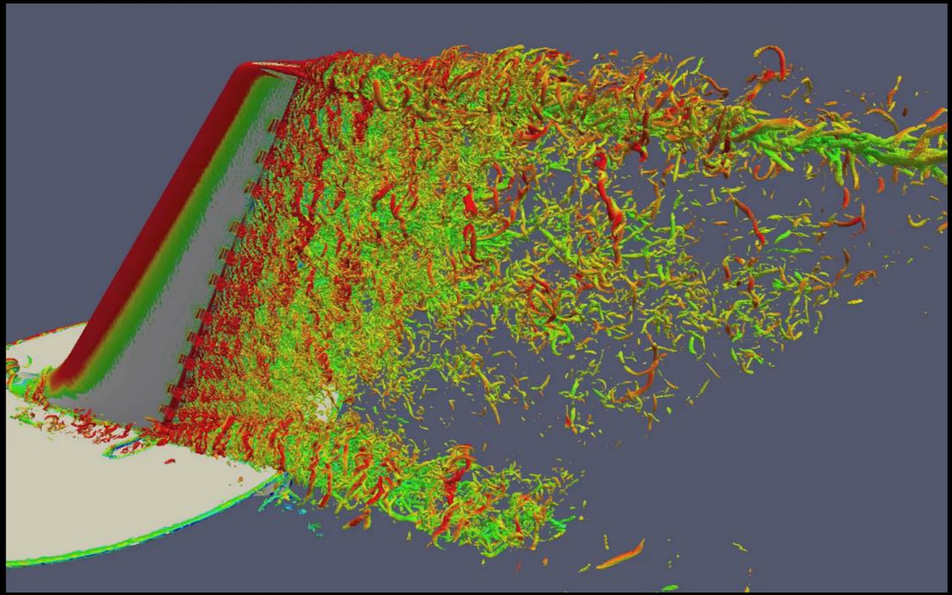
Post-hoc vs In situ analysis

- Post-hoc
  - Configure simulation
  - Run simulation
  - Wait for data
  - Analyze data
- Limitations
  - I/O bottleneck
  - Limited feedback before the end of simulation

- In situ
  - Configure simulation
  - Run simulation
  - Analyze data while results are in memory
  - Visualization/analysis output is smaller than raw data

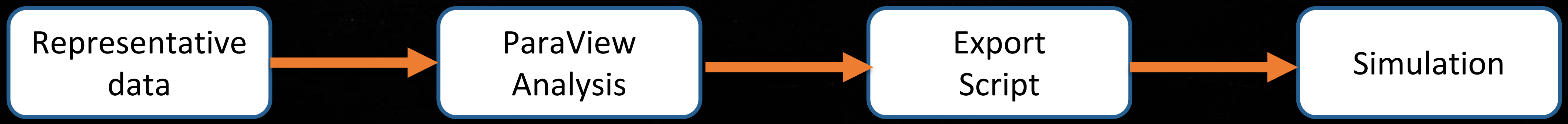
ParaView Catalyst

- Provides in situ data analysis and visualization capabilities for ParaView
- Write a Python script with your pipeline and perform batch processing on each time step
- Can also connect to the simulation with Catalyst Live from the ParaView GUI for interactive in situ visualization
  - Can pause simulation to investigate
  - Then resume simulation



Catalyst Workflow

- Get representative data
  - First timestep or a toy case
- Set up the analysis in ParaView
- Export Python script
- Manually modify script as necessary
- Start Simulation



Catalyst 2.0 Features

- ABI Compatibility
  - Can build your simulation against the Catalyst2 stub library
    - Lightweight library, easy to build
    - At runtime can use the ParaView Catalyst implementation

Reading ADIOS data in ParaView with Fides

- ParaView contains a reader called Fides that can be used to load your ADIOS data
- Fides provides a schema for mapping ADIOS Variables to VTK-m ArrayHandles
- Data model described in JSON
  - Describe mesh and fields
  - No need to write adaptors
  - Don't need to modify how ADIOS writes your data
- User guide: https://fides.readthedocs.io/

```json
"VTK_Structured_Grid": {
  "data_sources": [{
    "name": "source",
    "filename_mode": "input" }],
  "coordinate_system" : {
    "array" : {
      "array_type" : "basic",
      "data_source": "source",
      "variable" : "points",
      "is_vector" : "true",
      "static" : true }},
  "cell_set": {
    "cell_set_type" : "structured",
      "dimensions" : {
        "source" : "variable_dimensions",
        "data_source": "source",
        "variable" : "density" }},
  "fields": [{
      "name": "density",
      "association": "points",
      "array" : {
        "array_type" : "basic",
        "data_source": "source",
        "variable" : "density" }}]}
```

Fides Schema

```
"data_sources": [{
  "name": "source",
  "filename_mode": "input" }],
```

- Specify one or more **data_sources**
  - Each data source is an ADIOS file/stream
  - **name** lets you refer to it elsewhere in the data model
  - **filename_mode** should be set to “input” in most cases
- Supports multiple sources may be used
  - e.g., the mesh may be written to one file and fields written to another

Fides Schema

- **coordinate_system** and **cell_set** define the mesh
  - Can specify the mesh arrays are **static**, so Fides will cache the mesh details and not read it on every time step
  - **data_source** points back to a defined data source
  - **variable** is the name of the ADIOS variable
- **coordinate_system**
  - Can be **basic**, **uniform_point_coordinates**, or **cartesian_product**
- **cell_set**
  - Can be **structured**, **explicit**, or **single_type**

```
"coordinate_system" : {
  "array" : {
    "array_type" : "basic",
    "data_source": "source",
    "variable" : "points",
    "is_vector" : "true",
    "static" : true }},
"cell_set": {
  "cell_set_type" : "structured",
    "dimensions" : {
      "source" : "variable_dimensions",
      "data_source": "source",
      "variable" : "density" }},
```

Fides Schema

```
"fields": [{
      "name": "density",
      "association": "points",
      "array" : {
         "array_type" : "basic",
         "data_source": "source",
         "variable" : "density" }}]}
```

- **fields**
  - specify any ADIOS variables you would like read as field data
  - **association** should be either "points" or "cells"
  - array_type will usually be "basic", but there are some other options to support special cases such as the XGC simulation
  - **variable** is the name of the ADIOS Variable, while name can be a different **name** if you prefer

Fides Schema

- It's also possible to add some ADIOS Attributes with metadata and Fides will generate the data model for you
- For instance, if you have a uniform grid, you can specify the following attributes
  - **Fides_Data_Model**: "uniform"
  - **Fides_Origin**: [x, y, z]
  - **Fides_Spacing**: [x, y, z]
- For fields, it's also possible to use wildcard fields, so you don't have to specify every field
  - **Fides_Variable_List**: vector of variable names
  - **Fides_Variable_Associations**: vector of variable associations

```
"fields": [
    {
        "variable_list_attribute_name": "Fides_Variable_List",
        "variable_association_attribute_name": "Fides_Variable_Associations",
        "array": {
            "array_type": "basic",
            "data_source": "source",
            "variable": ""
        }
    }
]
```

Fides and ParaView Catalyst

- As of ParaView 5.11, the Fides reader has been integrated into ParaView Catalyst
- To use this feature, use the ADIOS engine plugin ParaViewADIOSInSituEngine
  - To get the engine plugin, build ADIOS with the Catalyst stub library: https://gitlab.kitware.com/paraview/catalyst
  - This engine plugin is used like other ADIOS engines
  - No need to instrument your code with calls to the Catalyst API – this engine handles that for you

Resources

- ParaView
  - User Guide: https://docs.paraview.org/en/latest/
  - Self-directed tutorial: https://docs.paraview.org/en/latest/Tutorials/index.html
  - Forum: https://discourse.paraview.org/
- Fides
  - User Guide: https://fides.readthedocs.io/en/latest/
  - Using Fides with ParaView: https://fides.readthedocs.io/en/latest/paraview/paraview.html

Configure the environment

```
$ cd ~/adios2-tutorial-source
$ source ~/adios2-tutorial-source/adios2-tutorial-env.sh
$ cd ~/adios2-tutorial-source/ADIOS2-Examples/build/install/share/adios2-examples/gray-scott
```

- ParaView is in PATH, so you can open by just typing paraview

Hands-on Demo: Visualizing Gray Scott with ParaView (Post-hoc)

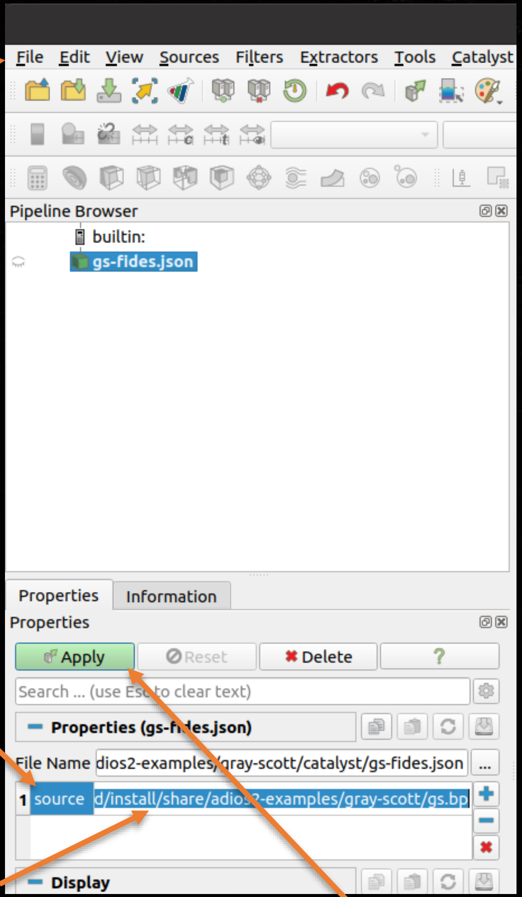
- Opening BP file with a Fides JSON data model file

1. Open JSON file

2. Add data source name from JSON file

```
"VTK-Cartesian-grid": {
  "data_sources": [
    {
      "name": "source",
      "filename_mode": "input"
    }
  ],
```

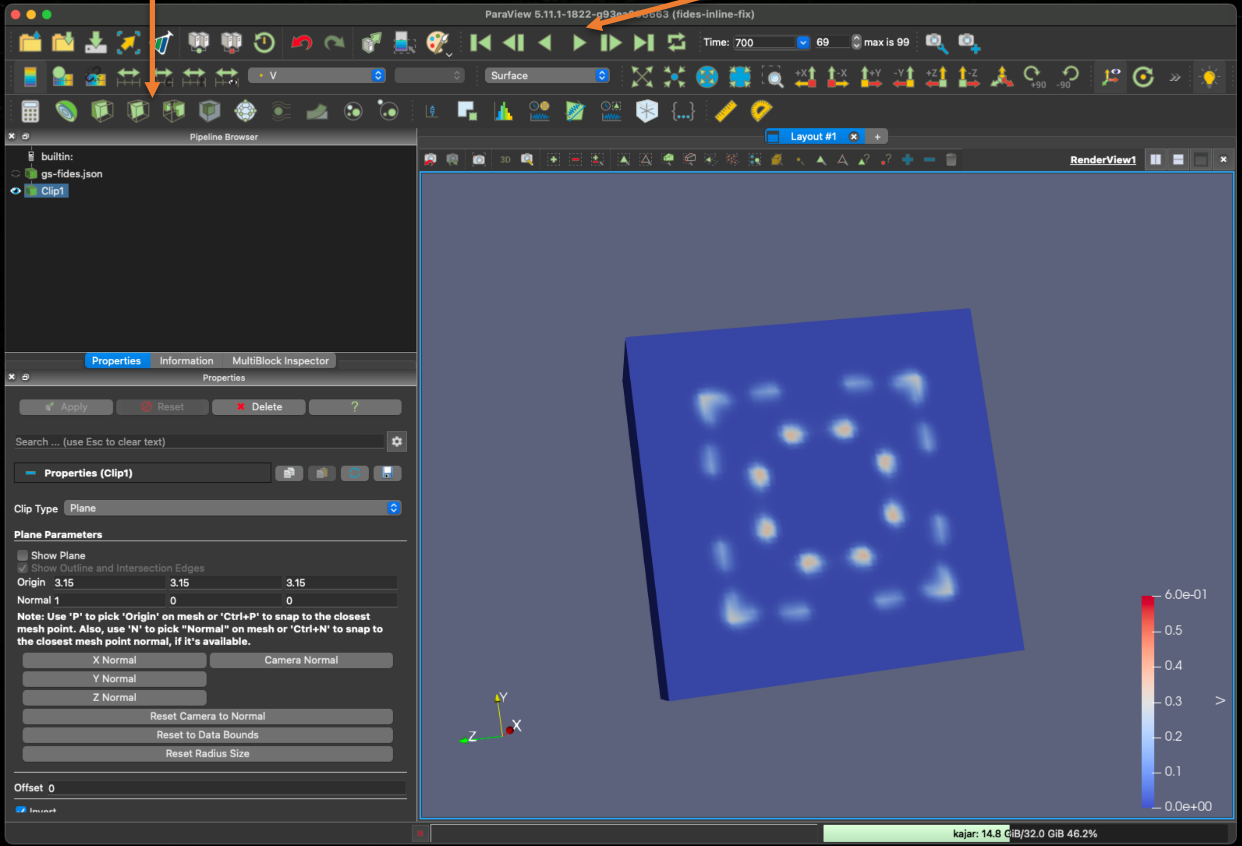
3. Add path to bp file



4. Click "Apply"

5. Add desired filters

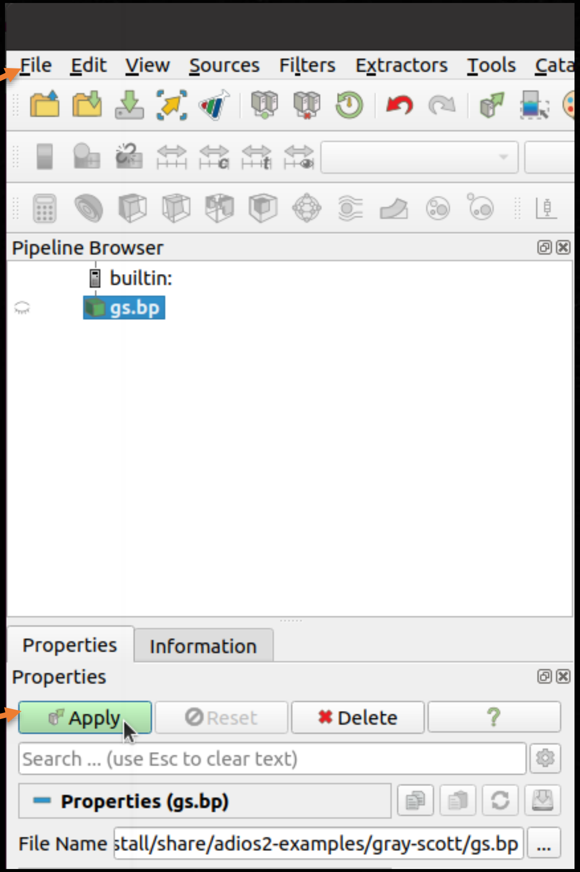
6. Play animation



Hands-on Demo: Visualizing Gray Scott with ParaView (Post-hoc)

- Easier approach: Opening BP file directly

1. Open BP file



2. Click "Apply"

3. Can now add filters, play animation, etc

- Requires writing Fides metadata in ADIOS attributes
- From simulation/writer.cpp constructor:

```cpp
// add attributes for Fides
io.DefineAttribute<std::string>("Fides_Data_Model", "uniform");
double origin[3] = {0.0, 0.0, 0.0};
io.DefineAttribute<double>("Fides_Origin", &origin[0], 3);
double spacing[3] = {0.1, 0.1, 0.1};
io.DefineAttribute<double>("Fides_Spacing", &spacing[0], 3);
io.DefineAttribute<std::string>("Fides_Dimension_Variable", "U");

std::vector<std::string> varList = {"U", "V"};
std::vector<std::string> assocList = {"points", "points"};
io.DefineAttribute<std::string>("Fides_Variable_List", varList.data(), varList.size());
io.DefineAttribute<std::string>("Fides_Variable_Associations", assocList.data(), assocList.size());
```

```
[tutorial@ip-172-31-37-244 gray-scott]$ bpls gs.bp/ -Ad
  double   Du                          attr   = 0.2
  double   Dv                          attr   = 0.1
  double   F                           attr   = 0.01
  string   Fides_Data_Model            attr   = "uniform"
  string   Fides_Dimension_Variable    attr   = "U"
  double   Fides_Origin                attr   = {0, 0, 0}
  double   Fides_Spacing               attr   = {0.1, 0.1, 0.1}
  string   Fides_Variable_Associations attr   = {"points", "points"}
  string   Fides_Variable_List         attr   = {"U", "V"}
```

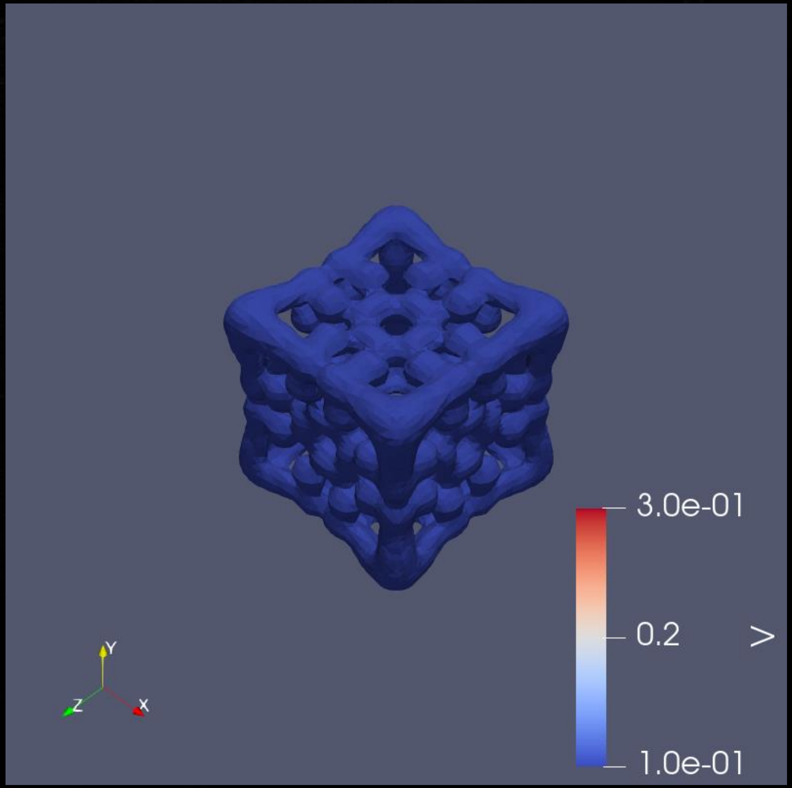
Hands-on Demo: Visualizing Gray Scott with ParaView (Post-hoc)

- Batch visualization with a Python script (not using the GUI)
- Run:

$ mpirun -n 2 pvbatch --force-offscreen-rendering catalyst/gs-pipeline.py \
-j catalyst/gs-fides.json -b gs.bp

- You'll see output-XXXXX.png files in the directory

```
at 11:27:45 with caitlin.ross in share/adios2-examples/gray-scott took 22s
➜  ls
adios2-fides-staging.xml  output-00014.png  output-00038.png  output-00062.png  output-00086.png
adios2-inline-plugin.xml  output-00015.png  output-00039.png  output-00063.png  output-00087.png
adios2.xml                output-00016.png  output-00040.png  output-00064.png  output-00088.png
catalyst                  output-00017.png  output-00041.png  output-00065.png  output-00089.png
ckpt.bp                   output-00018.png  output-00042.png  output-00066.png  output-00090.png
cleanup.sh                output-00019.png  output-00043.png  output-00067.png  output-00091.png
datasets                  output-00020.png  output-00044.png  output-00068.png  output-00092.png
decomp.py                 output-00021.png  output-00045.png  output-00069.png  output-00093.png
gs.bp                     output-00022.png  output-00046.png  output-00070.png  output-00094.png
gsplot.py                 output-00023.png  output-00047.png  output-00071.png  output-00095.png
output-00000.png          output-00024.png  output-00048.png  output-00072.png  output-00096.png
output-00001.png          output-00025.png  output-00049.png  output-00073.png  output-00097.png
output-00002.png          output-00026.png  output-00050.png  output-00074.png  output-00098.png
output-00003.png          output-00027.png  output-00051.png  output-00075.png  output-00099.png
output-00004.png          output-00028.png  output-00052.png  output-00076.png  pdfplot.py
output-00005.png          output-00029.png  output-00053.png  output-00077.png  README.md
output-00006.png          output-00030.png  output-00054.png  output-00078.png  settings-files.json
output-00007.png          output-00031.png  output-00055.png  output-00079.png  settings-inline.json
output-00008.png          output-00032.png  output-00056.png  output-00080.png  settings-staging.json
output-00009.png          output-00033.png  output-00057.png  output-00081.png  visit-bp4.session
output-00010.png          output-00034.png  output-00058.png  output-00082.png  visit-bp4.session.gui
output-00011.png          output-00035.png  output-00059.png  output-00083.png  visit-sst.session
output-00012.png          output-00036.png  output-00060.png  output-00084.png  visit-sst.session.gui
output-00013.png          output-00037.png  output-00061.png  output-00085.png
```



Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

The TAU Performance System

Kevin Huck, Sameer Shende, Allen Malony

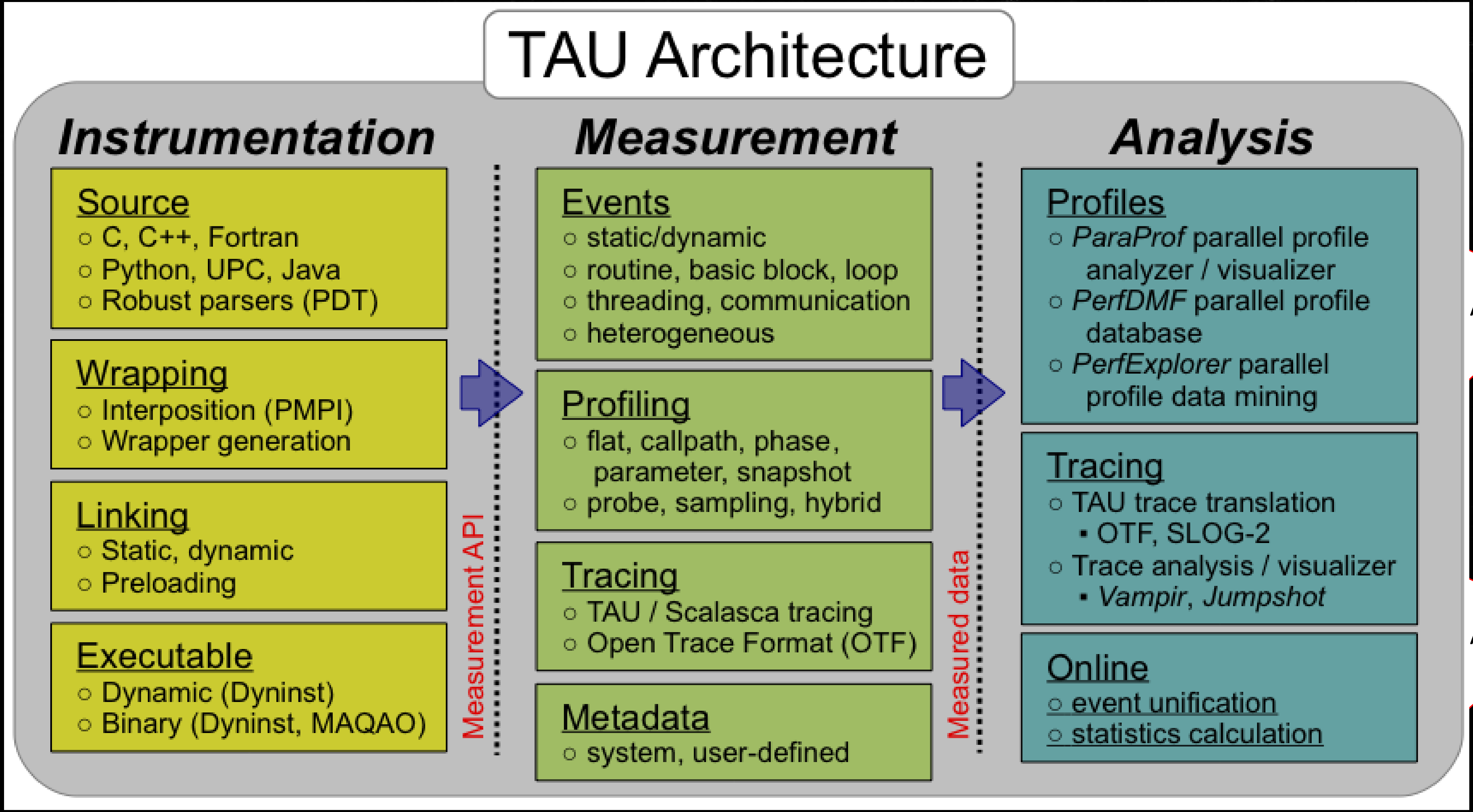
khuck@cs.uoregon.edu

http://tau.uoregon.edu

TAU : brief overview

- Tuning and Analysis Utilities (28+ year project)
- Integrated performance toolkit:
  - Multi-level performance instrumentation
  - Highly configurable
  - Widely ported performance profiling / tracing system
  - Portable (java, python) visualization / exploration / analysis tools
- Supports all major HPC programming models
  - MPI/SHMEM, OpenMP/ACC, CUDA, HIP, OneAPI, Kokkos...

TAU : brief overview

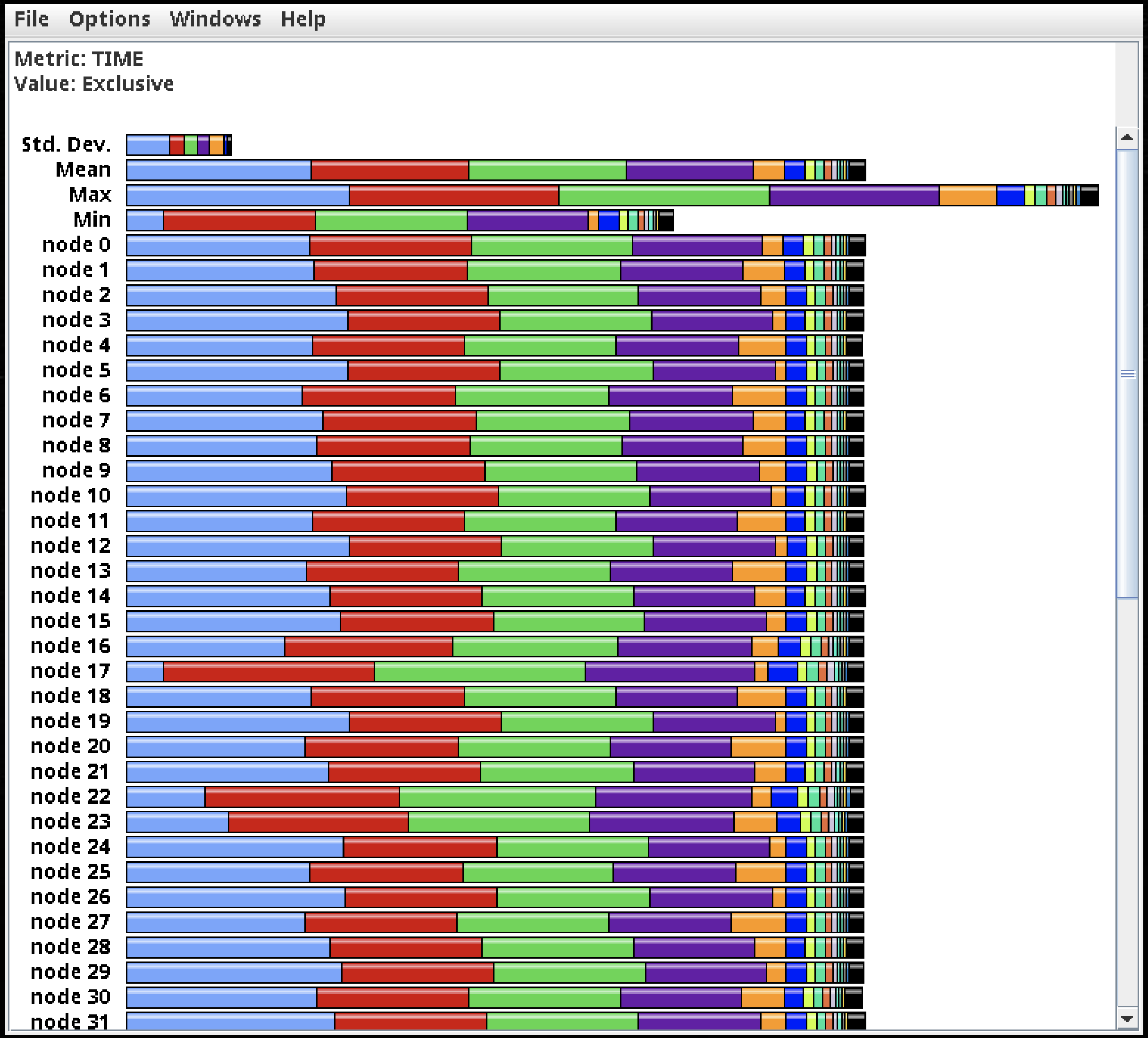


New!
ADIOS2
BP4

New!
ADIOS2
SST

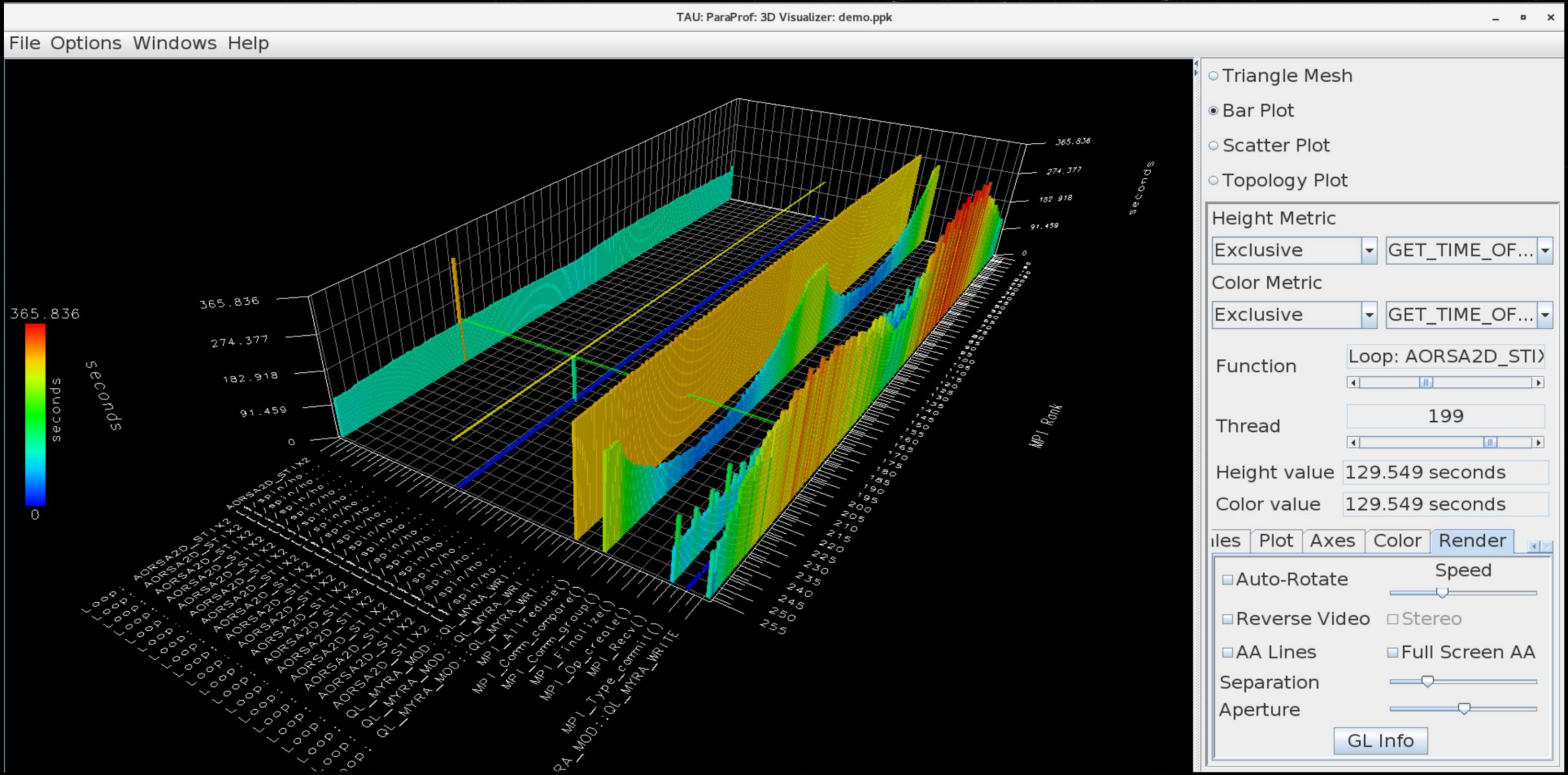
ParaProf Profile Browser

```
% paraprof
```

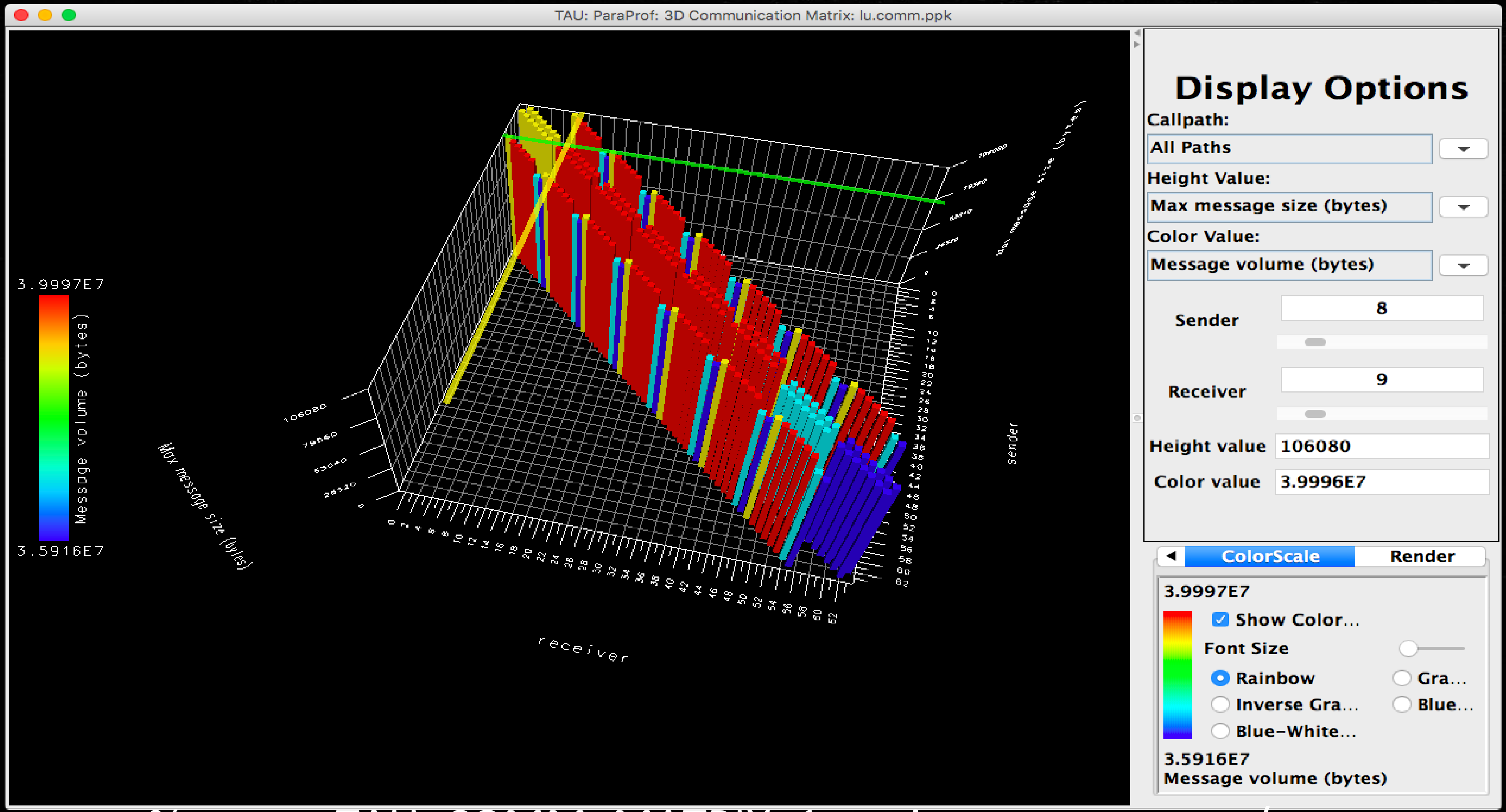


Each line is a different process/thread of execution, each color is a different function

ParaProf 3D Profile Browser



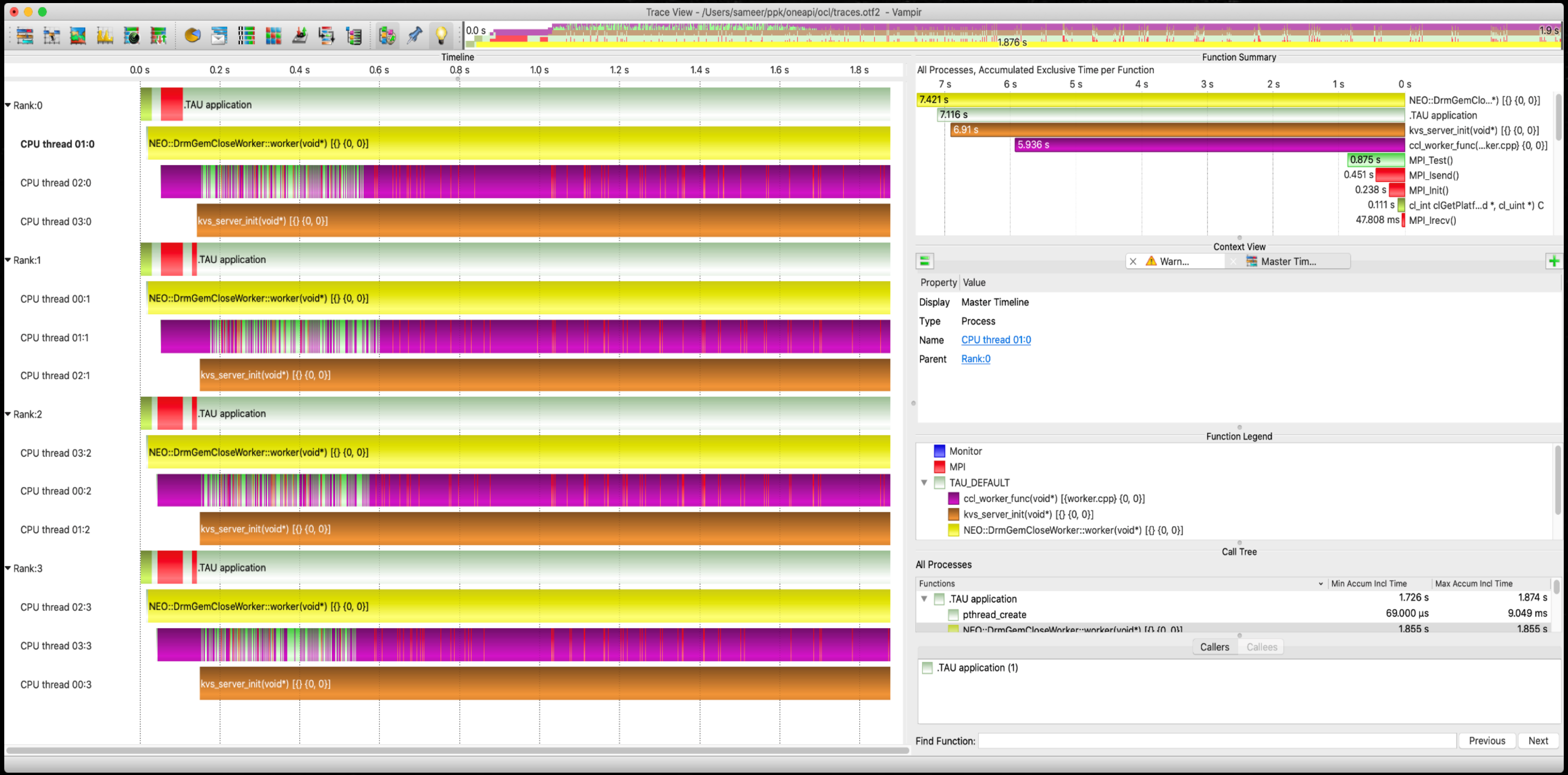
TAU – 3D Communication Window



% export TAU_COMM_MATRIX=1; mpirun … tau_exec ./a.out

% paraprof ; Windows -> 3D Communication Matrix

TAU and Vampir [TU Dresden]: Intel oneAPI OpenCL



% export TAU_TRACE=1; export TAU_TRACE_FORMAT=otf2

% mpirun –np 4 tau_exec –T level_zero –opencl ./a.out

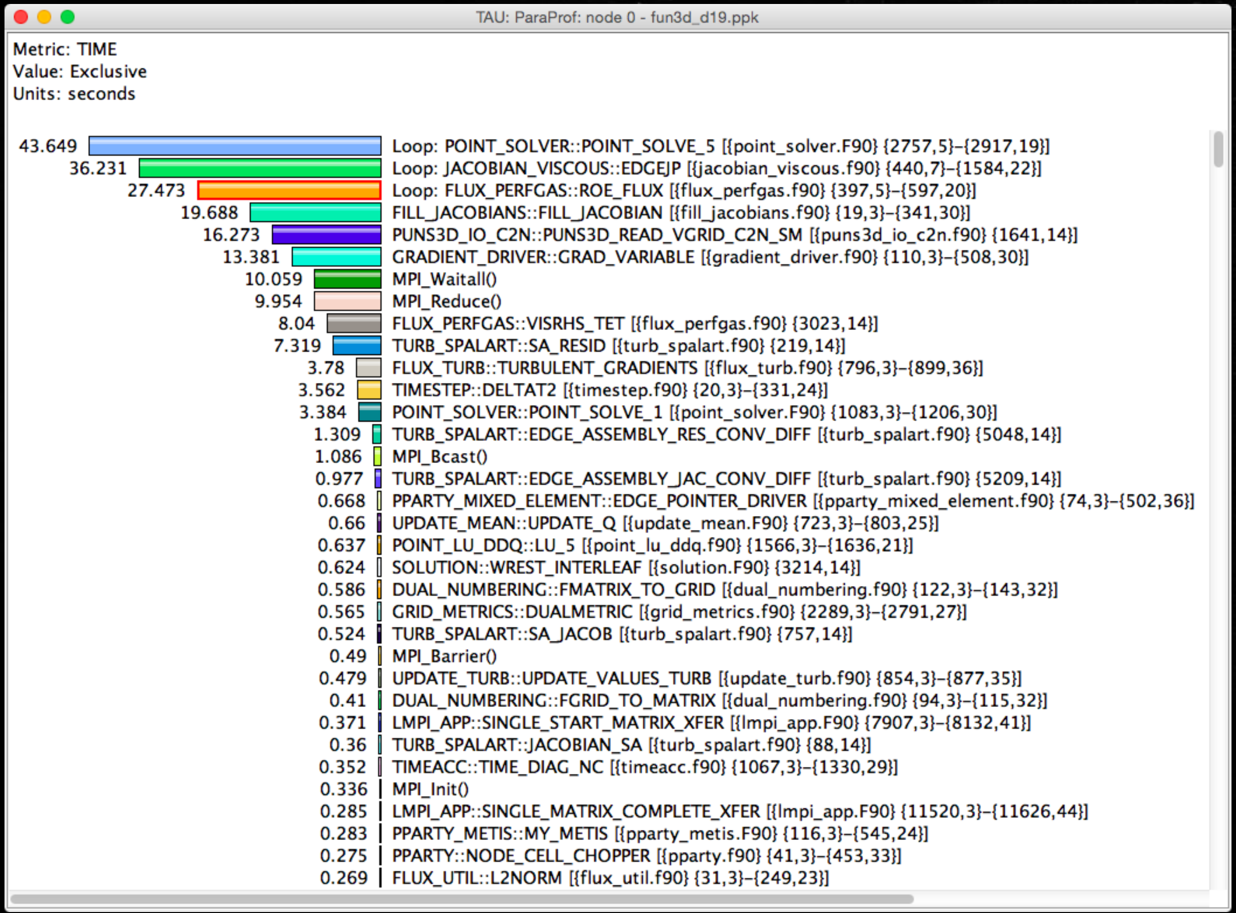
Performance Measurement

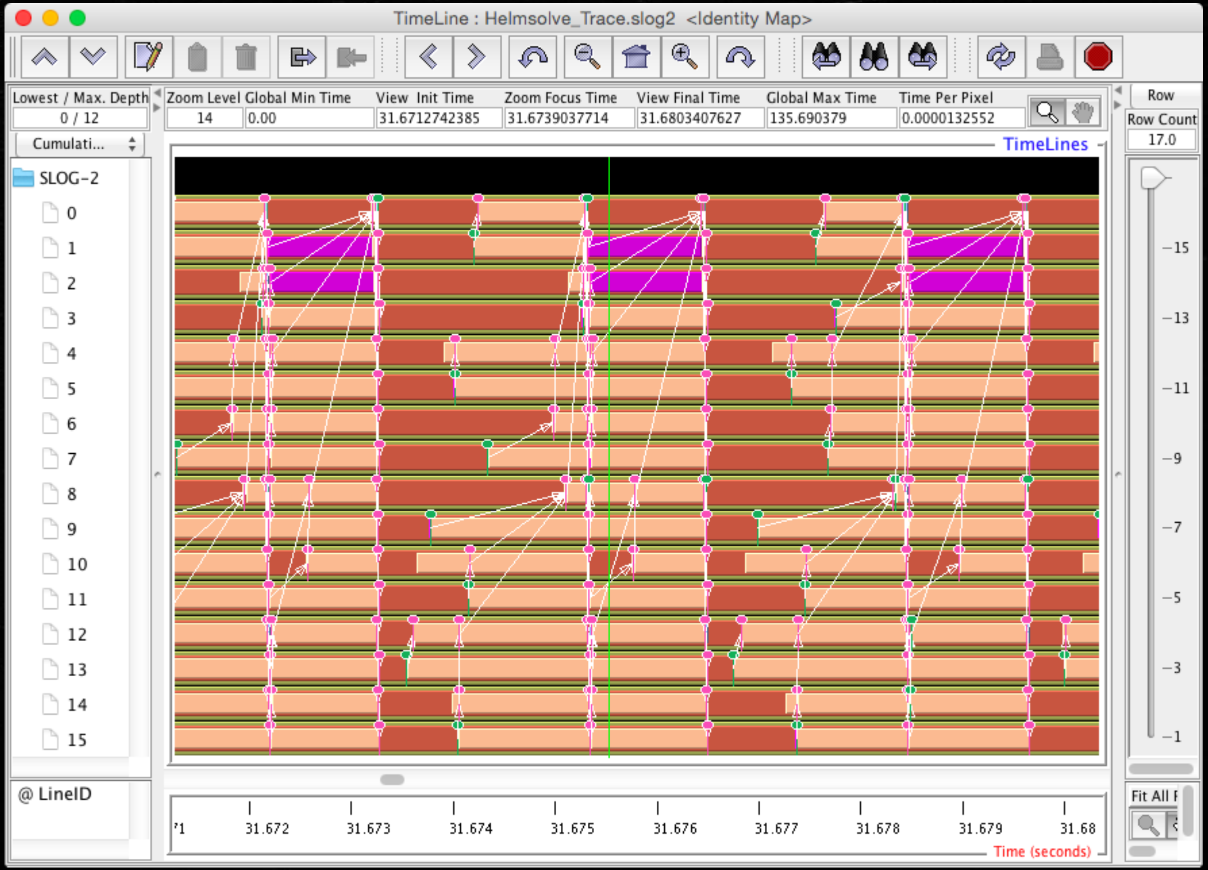
- **Timers**
  - Requires instrumentation of some kind
    - Manual, automated
    - Source, compiler provided, binary
    - Library callbacks, API wrappers, weak symbol replacement
  - Simple to implement
- **Sampling**
  - Requires specialized system libraries / support
    - Periodic signals, signal handler
  - No modification to executable/library needed
  - Potential to interfere with system support (signal handlers)

Profiling and Tracing

- **Profiling**: how much time was spent in each measured function on each thread in each process?
  - Collapses the time axis
  - No ordering or causal event information
  - Small summary per thread/process, regardless of execution time – only grows with number of timers & threads/processes
- **Tracing**: record all function entry & exit events on a timeline
  - Detailed view of what happened
  - The longer the program runs, the bigger the trace

Profiling and Tracing





Profiling shows you **how much** (total) time was spent in each routine

Tracing shows you **when** the events take place on a timeline

Integrating TAU

Compile Time

- Compile with TAU compiler wrappers (see next slides)
- Link with TAU libraries

Runtime

- Execute with `tau_exec`
- Preloads the TAU shared object library and instantiates measurement support for different models
- More later…

Instrumentation Approaches

- Manual
  - Add TAU API calls to the code by hand: https://www.cs.uoregon.edu/research/tau/docs/newguide/bk05rn01.html
- Automated:
  - PDT – optional TAU configuration
  - Compiler based instrumentation – comes with TAU
  - LLVM plugin – comes with TAU
  - Binary instrumentation - using Dyninst, PIN, or MAQAO
    - Optional TAU configuration, not covered in this tutorial
- PerfStubs API: https://github.com/UO-OACISS/perfstubs

Used by ADIOS2

Sampling

- Run the application with `tau_exec -ebs`
  - Preloads the TAU library, instantiates a signal handler and periodic interrupt to process the signal
  - The signal handler will record the current instruction pointer, all requested metrics, and optionally unwind the callstack
  - At the end of execution, all addresses are resolved to symbols in the application using binutils/libdwarf
- Some things that help:
  - For best support, build application with debug (-g) - all other optimizations are fine

Using TAU's Runtime Preloading Tool: tau_exec

- <mpirun> `tau_exec -T` <config> <options> <executable>
- `tau-config --list-matching` <mpi/serial> will show available configs
- Preload a wrapper that intercepts the runtime calls and substitutes with another (using dlsym() or weak symbol replacement)
  - MPI
  - OpenMP
  - POSIX I/O
  - Memory allocation/deallocation routines
  - Wrapper library for an external package
- No modification to the binary executable
- Enable other TAU options (communication matrix, OTF2, event-based sampling)

Sampled Measurement

Instrumentation example:

```
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0           21        1,538           1           1    1538547 .TAU application
 98.6        0.078        1,517           1           9    1517236 int main(int, char **) C [{simple.c} {49,1}-{63,1}]
 97.1        1,494        1,494           1           0    1494336 void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]
  1.3           20           20           3           0       6813 double **allocateMatrix(int, int) C [{simple.c} {10,1}-{17,1}]
  0.1            2            2           2           0       1129 void init(double **) C [{simple.c} {28,1}-{35,1}]
  0.0        0.125        0.125           3           0         42 void freeMatrix(double **, int) C [{simple.c} {19,1}-{25,1}]
```

Sampling example:

```
khuck@instinct:~/src/tau2/examples/simple$ make
gcc   -g -O2  -no-pie -c simple.c -o simple.o
gcc   -g -O2  -no-pie  simple.o -o simple
khuck@instinct:~/src/tau2/examples/simple$ tau_exec -T serial -ebs ./simple
c[9][9] = 367967744.000000
khuck@instinct:~/src/tau2/examples/simple$ pprof -a | grep -v CONTEXT
Reading Profile files in profile.*

---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        1,730        1,730           1           0    1730983 .TAU application
 84.9        1,469        1,469          49           0      30000 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {43}]
 13.9          240          240           8           0      30003 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {42}]
```

Both *more* and *less* information at the same time…

Sampled Measurement

```
khuck@instinct:~/src/tau2/examples/simple$ make
gcc   -g -O2  -no-pie -c simple.c -o simple.o
gcc   -g -O2  -no-pie  simple.o -o simple
khuck@instinct:~/src/tau2/examples/simple$ tau_exec -T serial -ebs ./simple
c[9][9] = 367967744.000000
khuck@instinct:~/src/tau2/examples/simple$ pprof -a | grep -v CONTEXT
Reading Profile files in profile.*

---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        1,730        1,730           1           0    1730983 .TAU application
 84.9        1,469        1,469          49           0      30000 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {43}]
 13.9          240          240           8           0      30003 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {42}]
```

```
37 /* Perform matrix multiply */
38 void compute(double **a, double **b, double **c) {
39     int i,j,k;
40     for (i=0; i < SIZE; i++) {
41         for(j=0; j < SIZE; j++) {
42             for (k=0; k < SIZE; k++) {
43                 c[i][j] += a[i][k] * b[k][j];
44             }
45         }
46     }
47 }
```

14% spent here

85% spent here

Simple Transformation – loop inversion

```
37 /* Perform matrix multiply */
38 void compute(double **a, double **b, double **c) {
39     int i,j,k;
40     for (i=0; i < SIZE; i++) {
41         for(j=0; j < SIZE; j++) {
42             for (k=0; k < SIZE; k++) {
43                 c[i][j] += a[i][k] * b[k][j];
44             }
45         }
46     }
47 }
```

```
37 /* Perform matrix multiply */
38 void compute(double **a, double **b, double **c) {
39     int i,j,k;
40     for (i=0; i < SIZE; i++) {
41         for (k=0; k < SIZE; k++) {
42             for(j=0; j < SIZE; j++) {
43                 c[i][j] += a[i][k] * b[k][j];
44             }
45         }
46     }
47 }
```

Reduced from 1.73 seconds

```
khuck@instinct:~/src/tau2/examples/simple$ make
gcc   -g -O2  -no-pie -c simple.c -o simple.o
gcc   -g -O2  -no-pie  simple.o -o simple
khuck@instinct:~/src/tau2/examples/simple$ tau_exec -T serial -ebs ./simple
c[9][9] = 367967744.000000
khuck@instinct:~/src/tau2/examples/simple$ pprof -a | grep -v CONTEXT
Reading Profile files in profile.*

---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0          976          976           1           0     976578 .TAU application
 70.7          690          690          23           0      30001 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {43}]
 27.6          269          269           9           0      29997 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {42}]
```

Both together!

Timers

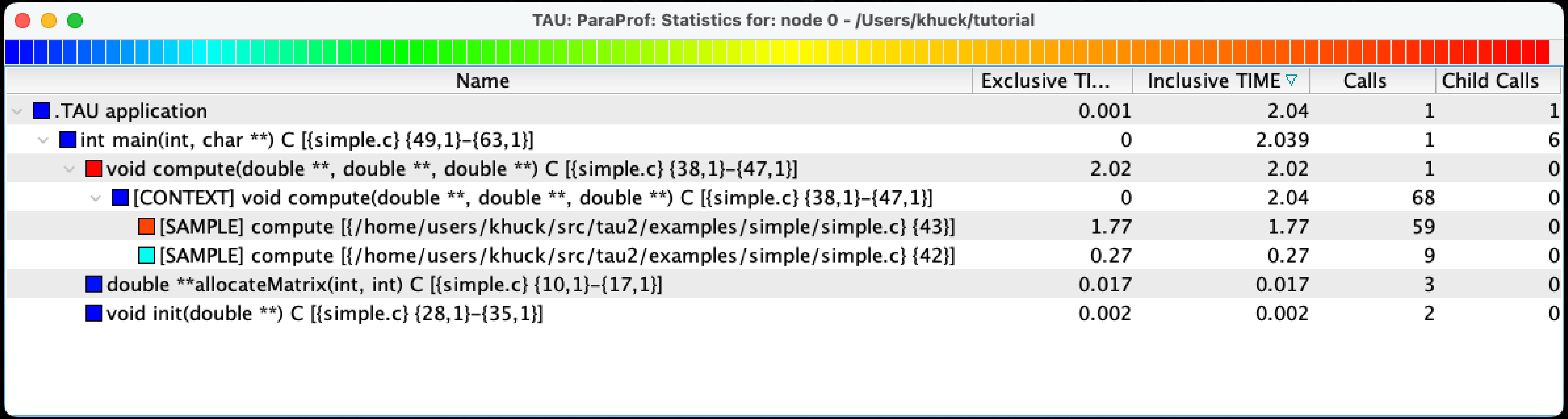
```
khuck@instinct:~/src/tau2/examples/simple$ tau_exec -T serial -ebs ./simple
c[9][9] = 367967744.000000
khuck@instinct:~/src/tau2/examples/simple$ pprof -a | grep -v CONTEXT
Reading Profile files in profile.*

---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0           18        1,611           1           1    1611909 .TAU application
 98.9        0.116        1,593           1           6    1593593 int main(int, char **) C [{simple.c} {49,1}-{63,1}]
 97.6        1,572        1,572           1           0    1572560 void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]
 67.0        1,079        1,079          36           0      30000 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {43}]
 29.8          480          480          16           0      30000 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {42}]
  1.9           30           30           1           0      30018 [SAMPLE] UNRESOLVED /usr/lib/x86_64-linux-gnu/libc-2.31.so
  1.2           18           18           3           0       6198 double **allocateMatrix(int, int) C [{simple.c} {10,1}-{17,1}]
  0.1            2            2           2           0       1161 void init(double **) C [{simple.c} {28,1}-{35,1}]
```

Samples

...with callpath profiling

```
khuck@instinct:~/src/tau2/examples/simple$ TAU_CALLPATH=1 TAU_CALLPATH_DEPTH=100 tau_exec -T serial -ebs ./simple
c[9][9] = 367967744.000000
khuck@instinct:~/src/tau2/examples/simple$ pprof -a
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            1        2,040           1           1    2040179 .TAU application
100.0            0        2,040          68           0      30000 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}] => void compute(double **
, double **, double **) C [{simple.c} {38,1}-{47,1}] => [CONTEXT] void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]
100.0            0        2,040          68           0      30000 [CONTEXT] void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]
 99.9         0.15        2,038           1           6    2038709 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}]
 99.9         0.15        2,038           1           6    2038709 int main(int, char **) C [{simple.c} {49,1}-{63,1}]
 99.0        2,019        2,019           1           0    2019510 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}] => void compute(double **
, double **, double **) C [{simple.c} {38,1}-{47,1}]
 99.0        2,019        2,019           1           0    2019510 void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]
 86.8        1,769        1,769          59           0      30000 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}] => void compute(double **
, double **, double **) C [{simple.c} {38,1}-{47,1}] => [CONTEXT] void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}] => [SAMPLE] compute [{/
home/users/khuck/src/tau2/examples/simple/simple.c} {43}]
 86.8        1,769        1,769          59           0      30000 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {43}]
 13.2          270          270           9           0      30001 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}] => void compute(double **
, double **, double **) C [{simple.c} {38,1}-{47,1}] => [CONTEXT] void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}] => [SAMPLE] compute [{/
home/users/khuck/src/tau2/examples/simple/simple.c} {42}]
 13.2          270          270           9           0      30001 [SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {42}]
  0.8           16           16           3           0       5562 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}] => double **allocateMatri
x(int, int) C [{simple.c} {10,1}-{17,1}]
  0.8           16           16           3           0       5562 double **allocateMatrix(int, int) C [{simple.c} {10,1}-{17,1}]
  0.1            2            2           2           0       1182 .TAU application => int main(int, char **) C [{simple.c} {49,1}-{63,1}] => void init(double **) C
 [{simple.c} {28,1}-{35,1}]
  0.1            2            2           2           0       1182 void init(double **) C [{simple.c} {28,1}-{35,1}]
```

...easier to view in ParaProf



TAU: ParaProf: Statistics for: node 0 - /Users/khuck/tutorial

Name	Exclusive TI...	Inclusive TIME ▽	Calls	Child Calls
.TAU application	0.001	2.04	1	1
int main(int, char **) C [{simple.c} {49,1}-{63,1}]	0	2.039	1	6
void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]	2.02	2.02	1	0
[CONTEXT] void compute(double **, double **, double **) C [{simple.c} {38,1}-{47,1}]	0	2.04	68	0
[SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {43}]	1.77	1.77	59	0
[SAMPLE] compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {42}]	0.27	0.27	9	0
double **allocateMatrix(int, int) C [{simple.c} {10,1}-{17,1}]	0.017	0.017	3	0
void init(double **) C [{simple.c} {28,1}-{35,1}]	0.002	0.002	2	0

Other measurement support

- Many programming models provide "hooks" for tools
- Often, instrumentation isn't necessary!
  - **MPI**, SHMEM, Charm++
  - Pthreads, **OpenMP**, Kokkos
  - CUDA, **HIP/ROCm**, OneAPI, OpenACC, OpenCL, OpenMP offload
  - Python
  - Wrappers: POSIX, Chapel, UPC, memory, ARMCI, GASNet…
  - Java

Other TAU features

- Binary instrumentation
    - Dyninst, MAQAO, PIN
- Hardware counter support
    - PAPI, LIKWID
- Tracing support (native or converters)
    - Vampir (OTF2), Perfetto (JSON), Jumpshot (SLOG2), …
- Plugins
    - OS/HW monitoring, ADIOS2, SOS, Mochi, SQLite3, …

NEW!

OpenMP

- https://www.openmp.org
- Pragma-based language extension to facilitate threading
- OpenMP 5.0 standard includes OpenMP Tools (OMPT/OMPD) specification for providing callbacks from the runtime to performance/debugging tools
- Provided by Intel, LLVM, IBM compilers
- GCC can use drop-in replacement (LLVM 8.0 runtime)
- TAU provides OPARI legacy support (when using PDT)

Adding OpenMP

```
/* Perform matrix multiply */
void compute(double **a, double **b, double **c) {
        int i,j,k;
#pragma omp parallel for
        for (i=0; i < SIZE; i++) {
                for(j=0; j < SIZE; j++) {
                        for (k=0; k < SIZE; k++) {
                                c[i][j] += a[i][k] * b[k][j];
                        }
                }
        }
}
```

If OMP_NUM_THREADS=4, SIZE=1024, then iteration space will be split into 4 of chunk size 256 each – 4x speedup

Compiling, Running, Reporting

```
khuck@instinct:~/src/tau2/examples/simple$ make
TAU_MAKEFILE=/storage/users/khuck/src/tau2/x86_64/lib/Makefile.tau-ompt-pdt-openmp tau_cc.sh -optShared -optQuiet -g -O2 -fopenmp  -no-pie -c simple.c -o simple.o
TAU_MAKEFILE=/storage/users/khuck/src/tau2/x86_64/lib/Makefile.tau-ompt-pdt-openmp tau_cc.sh -optShared -optQuiet -g -O2 -fopenmp  -no-pie  simple.o -o simple
khuck@instinct:~/src/tau2/examples/simple$ export OMP_NUM_THREADS=2
khuck@instinct:~/src/tau2/examples/simple$ ./simple
c[9][9] = 367967744.000000
khuck@instinct:~/src/tau2/examples/simple$ pprof -s -a
Reading Profile files in profile.*

FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0           18        1,787           2           2     893950 .TAU application
 97.2        1,738        1,738           2           1     869314 OpenMP_Implicit_Task
 50.3        0.078          899           1           9     899006 int main(int, char **) C [{simple.c} {50,1}-{64,1}]
 49.0            6          875           1           1     875235 void compute(double **, double **, double **) C [{simple.c} {38,1}-{48,1}]
 48.7        0.012          870           1           1     870224 OpenMP_Thread_Type_ompt_thread_worker
 48.6        0.131          868           1           1     868546 OpenMP_Parallel_Region compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {51, 0}]
  1.2           21           21           3           0       7012 double **allocateMatrix(int, int) C [{simple.c} {10,1}-{17,1}]
  0.1            2            2           2           0       1071 void init(double **) C [{simple.c} {28,1}-{35,1}]
  0.0        0.516        0.516           3           0        172 void freeMatrix(double **, int) C [{simple.c} {19,1}-{25,1}]
  0.0        0.012        0.012           1           0         12 OpenMP_Sync_Region_Barrier compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {51, 0}]

FUNCTION SUMMARY (mean):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            9          893           1           1     893950 .TAU application
 97.2          869          869           1         0.5     869314 OpenMP_Implicit_Task
 50.3        0.039          449         0.5         4.5     899006 int main(int, char **) C [{simple.c} {50,1}-{64,1}]
 49.0            3          437         0.5         0.5     875235 void compute(double **, double **, double **) C [{simple.c} {38,1}-{48,1}]
 48.7        0.006          435         0.5         0.5     870224 OpenMP_Thread_Type_ompt_thread_worker
 48.6       0.0655          434         0.5         0.5     868546 OpenMP_Parallel_Region compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {51, 0}]
  1.2           10           10         1.5           0       7012 double **allocateMatrix(int, int) C [{simple.c} {10,1}-{17,1}]
  0.1            1            1           1           0       1071 void init(double **) C [{simple.c} {28,1}-{35,1}]
  0.0        0.258        0.258         1.5           0        172 void freeMatrix(double **, int) C [{simple.c} {19,1}-{25,1}]
  0.0        0.006        0.006         0.5           0         12 OpenMP_Sync_Region_Barrier compute [{/home/users/khuck/src/tau2/examples/simple/simple.c} {51, 0}]
```

Compiler flag to Enable OpenMP

Thread lifetime

Worker lifetime

Region

Synchronization

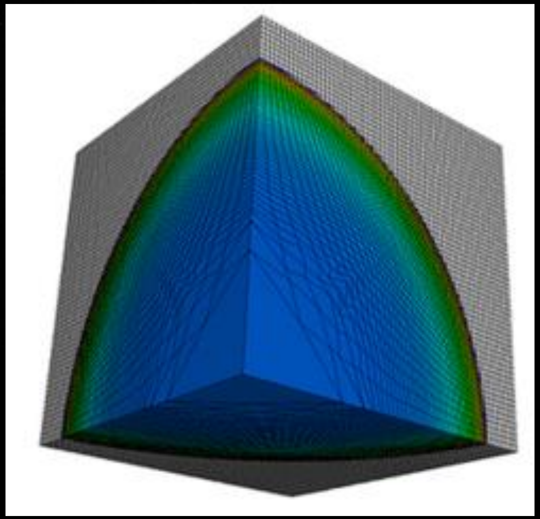
MPI Support

- MPI standard includes tool support
  - MPI_* functions are thin, weak wrappers around PMPI_* API
  - Tools create their own wrappers to replace them and intercept MPI calls
  - Tool library is preloaded or linked ahead of MPI library(ies)
  - Example:

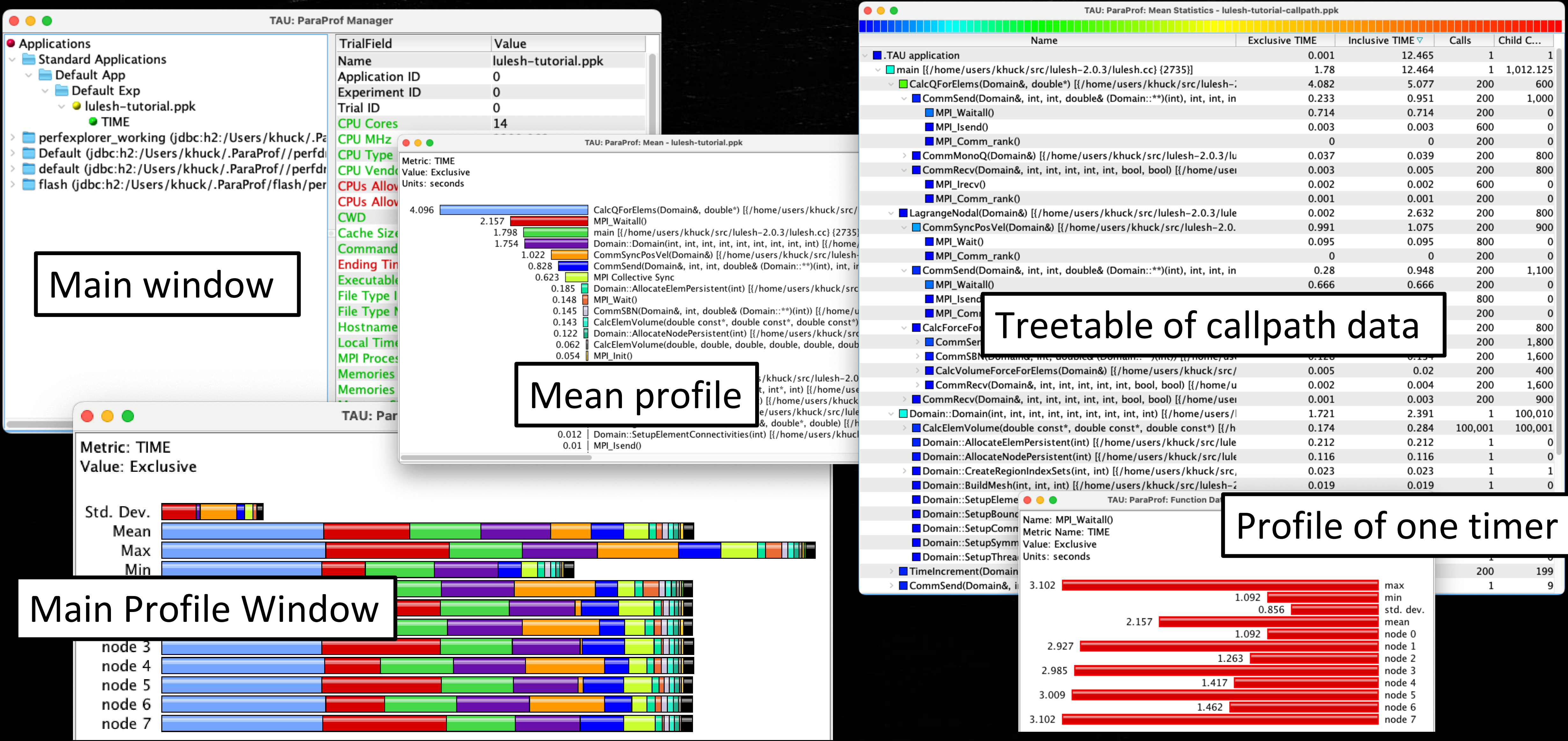
```
int MPI_Barrier(MPI_Comm comm) {
    int returnVal;

    TAU_PROFILE_TIMER(tautimer, "MPI_Barrier()",  " ", TAU_MESSAGE);
    TAU_PROFILE_START(tautimer);

    returnVal = PMPI_Barrier( comm );

    TAU_PROFILE_STOP(tautimer);
    return returnVal;
}
```

MPI example – Lulesh

- Lulesh 2.0.3 https://asc.llnl.gov/codes/proxy-apps/lulesh
- “The Shock Hydrodynamics Challenge Problem was originally defined and implemented by LLNL as one of five challenge problems in the DARPA UHPC program and has since become a widely studied proxy application in DOE co-design efforts for exascale.”
- C++, Serial, OpenMP, MPI
- CUDA, OpenACC, OpenCL, other models



Lulesh Profile - ParaProf



Lulesh Trace – Vampir

Measuring HIP kernel performance

- Hip-stream – small program with 4+ kernel

```
* Copyright 2015: Tom Deakin, Simon McIntosh-Smith, University of Bristol HPC
* Based on John D. McCalpin's original STREAM benchmark for CPUs
```

Program output

```
[khuck@login2.crusher add4]$ ./gpu-stream-hip
GPU-STREAM
Version: 1.0
Implementation: HIP
GridSize: 26214400 work-items
GroupSize: 1024 work-items
Operations/Work-item: 1
Precision: double

Running kernels 10 times
Array size: 200.0 MB (=0.2 GB) 0 bytes padding
Total size: 1000.0 MB (=1.0 GB)
Using HIP device  (compute_units=110)
Driver: 50013601
d_a=0x7f0cb0000000
d_b=0x7f0ca0000000
d_c=0x7f0c90000000
d_d=0x7f0c80000000
d_e=0x7f0c70000000
Function    MBytes/sec  Min (sec)   Max         Average
Copy        1331503.944 0.00032     0.00032     0.00032
Mul         1332392.192 0.00031     0.00032     0.00032
Add4        1196944.446 0.00088     0.00089     0.00088
Triad       1256501.941 0.00050     0.00051     0.00050
GEOMEAN     1278064.946
```

HIP kernels

```
template <typename T> __global__ void copy(const T * a, T * c) {
    const int i = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    c[i] = a[i];
}

template <typename T> __global__ void mul(T * b, const T * c) {
    const T scalar = 3.0;
    const int i = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    b[i] = scalar * c[i];
}

template <typename T> __global__ void
add(const T * a, const T * b, const T *d, const T *e, T * c) {
    const int i = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    c[i] = a[i] + b[i] + d[i] + e[i];
}

template <typename T> __global__ void
triad(T * a, const T * b, const T * c) {
    const T scalar = 3.0;
    const int i = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    a[i] = b[i] + scalar * c[i];
}
```

Measuring HIP kernel performance

- Just add **`tau_exec`** and arguments to the command (between srun/mpirun and application when applicable)
- **`tau-config`** shows available configs

"use serial,rocprofiler configuration with HIP/ROCm support enabled"

```
[khuck@login2.crusher add4]$ tau_exec -T serial,rocprofiler -rocm ./gpu-stream-hip
GPU-STREAM
Version: 1.0
Implementation: HIP
GridSize: 26214400 work-items
GroupSize: 1024 work-items
Operations/Work-item: 1
Precision: double

Running kernels 10 times
Array size: 200.0 MB (=0.2 GB) 0 bytes padding
Total size: 1000.0 MB (=1.0 GB)
Using HIP device  (compute_units=110)
Driver: 50013601
d_a=0x7f48e0000000
d_b=0x7f48d0000000
d_c=0x7f47b0000000
d_d=0x7f47a0000000
d_e=0x7f4790000000
Function      MBytes/sec   Min (sec)    Max          Average
Copy          1320624.685  0.00032      0.00032      0.00032
Mul           1321623.393  0.00032      0.00032      0.00032
Add4          1217965.813  0.00086      0.00088      0.00087
Triad         1254042.504  0.00050      0.00051      0.00051
GEOMEAN       1277787.457
```

Pprof output, timers

```
[khuck@login2.crusher add4]$ pprof
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        1,031        1,031           1           1    1031765 .TAU application
  0.0         0.03         0.03           1           0         30 pthread_create

NODE 0;CONTEXT 0;THREAD 1:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            1          479           1           1     479989 .TAU application
 99.6          478          478           1           0     478248 [PTHREAD] _ZN4rocr2os16ThreadTrampolineEPv

NODE 0;CONTEXT 0;THREAD 2:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.638           20           1          40      20298 .TAU application
 42.4            8            8          10           0        861 void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
 24.2            4            4          10           0        492 void triad<double>(double*, double const*, double const*) [clone .kd]
 15.1            3            3          10           0        307 void copy<double>(double const*, double*) [clone .kd]
 15.1            3            3          10           0        306 void mul<double>(double*, double const*) [clone .kd]
---------------------------------------------------------------------------------------
```

Main Thread

Rocprofiler Thread

Device activity

Pprof output, counters

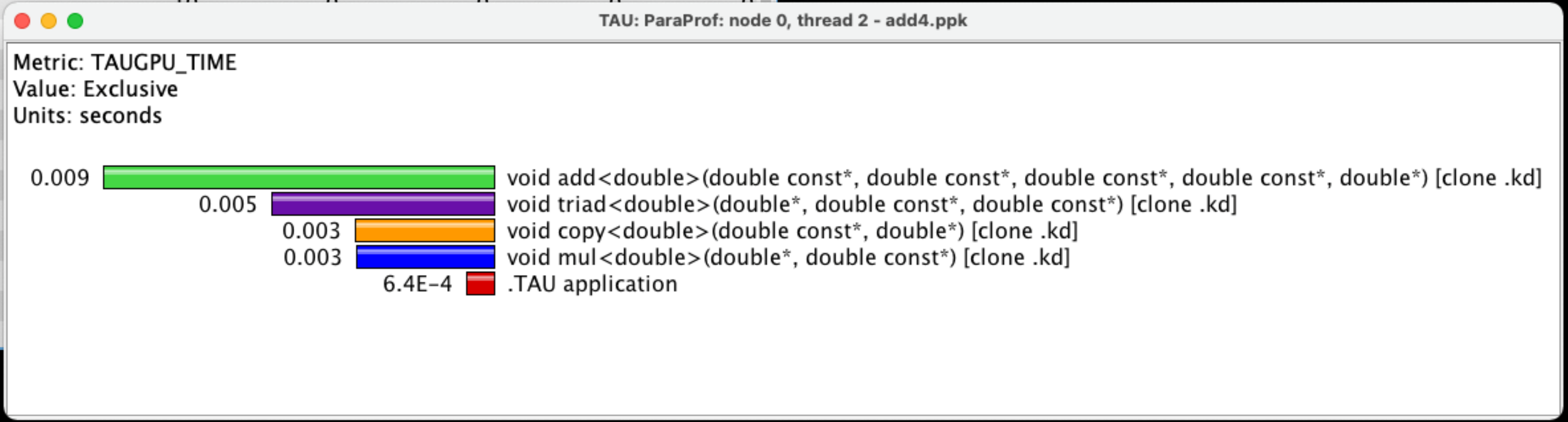
Counters for measuring register pressure and occupancy

```
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 2
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        10  2.621E+07  2.621E+07  2.621E+07          0  Grid Size : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10  2.621E+07  2.621E+07  2.621E+07          0  Grid Size : void copy<double>(double const*, double*) [clone .kd]
        10  2.621E+07  2.621E+07  2.621E+07          0  Grid Size : void mul<double>(double*, double const*) [clone .kd]
        10  2.621E+07  2.621E+07  2.621E+07          0  Grid Size : void triad<double>(double*, double const*, double const*) [clone .kd]
        10          0          0          0          0  LDS Memory Size : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10          0          0          0          0  LDS Memory Size : void copy<double>(double const*, double*) [clone .kd]
        10          0          0          0          0  LDS Memory Size : void mul<double>(double*, double const*) [clone .kd]
        10          0          0          0          0  LDS Memory Size : void triad<double>(double*, double const*, double const*) [clone .kd]
        10         32         32         32          0  Scalar Register Size (SGPR) : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10         24         24         24          0  Scalar Register Size (SGPR) : void copy<double>(double const*, double*) [clone .kd]
        10         24         24         24          0  Scalar Register Size (SGPR) : void mul<double>(double*, double const*) [clone .kd]
        10         24         24         24          0  Scalar Register Size (SGPR) : void triad<double>(double*, double const*, double const*) [clone .kd]
        10          0          0          0          0  Scratch Memory Size : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10          0          0          0          0  Scratch Memory Size : void copy<double>(double const*, double*) [clone .kd]
        10          0          0          0          0  Scratch Memory Size : void mul<double>(double*, double const*) [clone .kd]
        10          0          0          0          0  Scratch Memory Size : void triad<double>(double*, double const*, double const*) [clone .kd]
        10          8          8          8          0  Vector Register Size (VGPR) : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10          4          4          4          0  Vector Register Size (VGPR) : void copy<double>(double const*, double*) [clone .kd]
        10          4          4          4          0  Vector Register Size (VGPR) : void mul<double>(double*, double const*) [clone .kd]
        10          4          4          4          0  Vector Register Size (VGPR) : void triad<double>(double*, double const*, double const*) [clone .kd]
        10       1024       1024       1024          0  Work Group Size : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10       1024       1024       1024          0  Work Group Size : void copy<double>(double const*, double*) [clone .kd]
        10       1024       1024       1024          0  Work Group Size : void mul<double>(double*, double const*) [clone .kd]
        10       1024       1024       1024          0  Work Group Size : void triad<double>(double*, double const*, double const*) [clone .kd]
        10       5952       5952       5952          0  fbarrier count : void add<double>(double const*, double const*, double const*, double const*, double*) [clone .kd]
        10       3392       3392       3392          0  fbarrier count : void copy<double>(double const*, double*) [clone .kd]
        10       4672       4672       4672          0  fbarrier count : void mul<double>(double*, double const*) [clone .kd]
        10          0          0          0          0  fbarrier count : void triad<double>(double*, double const*, double const*) [clone .kd]
---------------------------------------------------------------------------------------
```

ParaProf view of same data

TAU: ParaProf: Context Events for: node 0, thread 2 - add4.ppk

Name	NumSamp...	MaxValue	MinValue	MeanVal...	Std. Dev.
void add<double>(double const*, double const*, double const*, double cor					
Grid Size	10	26,214,400	26,214,400	26,214,400	0
LDS Memory Size	10	0	0	0	0
Scalar Register Size (SGPR)	10	32	32	32	0
Scratch Memory Size	10	0	0	0	0
Vector Register Size (VGPR)	10	8	8	8	0
Work Group Size	10	1,024	1,024	1,024	0
fbarrier count	10	5,952	5,952	5,952	0
void copy<double>(double const*, double*) [clone .kd]					
Grid Size	10	26,214,400	26,214,400	26,214,400	0
LDS Memory Size	10	0	0	0	0
Scalar Register Size (SGPR)	10	24	24	24	0
Scratch Memory Size	10	0	0	0	0
Vector Register Size (VGPR)	10	4	4	4	0
Work Group Size	10	1,024	1,024	1,024	0
fbarrier count	10	3,392	3,392	3,392	0
void mul<double>(double*, double const*) [clone .kd]					
Grid Size	10	26,214,400	26,214,400	26,214,400	0
LDS Memory Size					
Scalar Register Size (SGPR)					
Scratch Memory Size					
Vector Register Size (VGPR)					
Work Group Size					
fbarrier count					
void triad<double>(double*, double const*, double const*) [clone .kd]					
Grid Size					
LDS Memory Size					
Scalar Register Size (SGPR)					
Scratch Memory Size					
Vector Register Size (VGPR)					



VERY helpful for understanding **register pressure** and **occupancy**

Tracing support uses Roctracer

$ TAU_TRACE=1 TAU_TRACE_FORMAT=otf2 tau_exec -T serial,roctracer ./gpu-stream-hip

Each device has 2-3 virtual threads:
1) kernels,
2) memory transfers
3) synchronization
(prevents overlapping timers)

tau_exec command reference

- Uninstrumented execution
  - % mpirun -np 256 ./a.out
- Track GPU operations
  - % mpirun –np 256 tau_exec –l0 ./a.out
  - % mpirun –np 256 tau_exec –opencl ./a.out
  - % mpirun –np 256 tau_exec –openacc ./a.out
  - % mpirun –np 256 tau_exec –cupti ./a.out
  - % mpirun –np 256 tau_exec –rocm ./a.out
- Track MPI performance
  - % mpirun -np 256 tau_exec ./a.out
- Track I/O, and MPI performance (MPI enabled by default)
  - % mpirun -np 256 tau_exec -io ./a.out
- Track OpenMP and MPI execution (using OMPT for Intel v19+ or Clang 8+)
  - % export TAU_OMPT_SUPPORT_LEVEL=full;
  - % mpirun –np 256 tau_exec –T ompt,mpi -ompt ./a.out
- Track memory operations
  - % export TAU_TRACK_MEMORY_LEAKS=1
  - % mpirun –np 256 tau_exec –memory_debug ./a.out (bounds check)
- Use event based sampling (compile with –g)
  - % mpirun –np 256 tau_exec –ebs ./a.out
  - Also export TAU_METRICS=TIME,PAPI_L1_DCM… -ebs_resolution=<file | function | line>
- Non-MPI execution: use –T serial
  - % tau_exec –T serial,level_zero –l0 –ebs ./a.out

TAU Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_SAMPLING	1	Setting to 1 enables event-based sampling.
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Throttles instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_CALLSITE	0	Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.
TAU_PROFILE_FORMAT	Profile	Setting to "merged" generates a single file. "snapshot" generates xml format
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

TAU Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	Default	Setting to "otf2" turns on TAU's native OTF2 trace generation (configure with –otf=download)
TAU_EBS_UNWIND	0	Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec –ebs or TAU_SAMPLING=1)
TAU_EBS_RESOLUTION	line	Setting to "function" or "file" changes the sampling resolution to function or file level respectively.
TAU_TRACK_LOAD	0	Setting to 1 tracks system load on the node
TAU_SELECT_FILE	Default	Setting to a file name, enables selective instrumentation based on exclude/include lists specified in the file.
TAU_OMPT_SUPPORT_LEVEL	basic	Setting to "full" improves resolution of OMPT TR6 regions on threads 1.. N-1. Also, "lowoverhead" option is available.
TAU_OMPT_RESOLVE_ADDRESS_EAGERLY	1	Setting to 1 is necessary for event based sampling to resolve addresses with OMPT. Setting to 0 allows the user to do offline address translation.

For more info…

- https://tau.uoregon.edu
- https://github.com/UO-OACISS/tau2
- https://github.com/UO-OACISS/tau2/wiki
- https://github.com/UO-OACISS/tau2/wiki/Frequently-Asked-Questions-%28FAQ%29
- Email tau-bugs@cs.uoregon.edu

ADIOS2 Output

- As discussed, **`tau_exec`** is the execution wrapper to attach ”inject” or ”attach” TAU to the application
- TAU comes with a plugin API (see Malony, Allen D., et al. "A plugin architecture for the TAU performance system”, *ICPP*, 2019.)
  - Code is executed at TAU events (timer start/stop, profile dump, program start/stop, etc.), or periodically
  - TAU comes with an ADIOS2 plugin that will write the profile data to ADIOS2 instead of TAU profile files

ADIOS2 plugin events

- PostInit – after MPI_Init, initialize ADIOS2 for writing
- Dump – when the "Tau_dump()" function is called, update all available profile statistics, write to ADIOS2
- PreEndOfExecution – before MPI_Finalize is executed, perform any final unification and write and close
- EndOfExecution – cleanup

Using the ADIOS2 plugin

- Just add the **`-adios2`** flag to tau_exec
- Useful environment variables
  - **`TAU_ADIOS2_PERIODIC`**: yes/true/on/1 – tells the plugin to write output periodically and asynchronously, (defaults to **`false`**)
  - **`TAU_ADIOS2_PERIOD`**: number in microseconds (default 2,000,000)
  - **`TAU_ADIOS2_ONE_FILE`**: write one ADIOS2 file, or each MPI rank writes their own (defaults to **`true`**)
  - **`TAU_ADIOS2_FILENAME`**: defaults to **`tauprofile-<executable name>.bp`** (can specify path with TAU **`$PROFILEDIR`** env var)
  - **`TAU_ADIOS2_ENGINE`**: (defaults to **`BPFile`**)
  - **`TAU_ADIOS2_CONFIG_FILE`**: for specifying all ADIOS2 settings (default: **`./adios2.xml`**)

What data is stored?

- TAU Metadata stored as ADIOS2 attributes (strings)
- TAU Counters
  - Bytes sent, GPU registers used, etc.
- TAU Timers
  - Time in foo(), bar(), etc.

Metadata examples (bpls -A -l)

```
khuck@Kevins-Air xgc % $HOME/src/ADIOS2/install/bin/bpls -A -d tauprofile-xgc-es-cpp-gpu-checkpoint.bp | head -n 40
  string   TAU:0:0:MetaData:CPU Cores                          attr   = "64"
  string   TAU:0:0:MetaData:CPU MHz                            attr   = "1894.426"
  string   TAU:0:0:MetaData:CPU Type                           attr   = "AMD EPYC 7A53 64-Core Processor"
  string   TAU:0:0:MetaData:CPU Vendor                         attr   = "AuthenticAMD"
  string   TAU:0:0:MetaData:CPUs Allowed                       attr   = "00000000,00000000,00000000,000000ff"
  string   TAU:0:0:MetaData:CPUs Allowed List                  attr   = "0-7"
  string   TAU:0:0:MetaData:CRAY_CORE_ID                       attr   = "3"
  string   TAU:0:0:MetaData:CWD                                attr   = "/gpfs/alpine/fus123/proj-shared/khuck/FOM_
run/WeakScalingESFrontier/rundir_2_planes_check_tau_adios2"
  string   TAU:0:0:MetaData:Cache Size                         attr   = "512 KB"
  string   TAU:0:0:MetaData:Command Line                       attr   = "/ccs/home/khuck/ECP-WDM/ecp-wdm-coe/crushe
r_rocm5.2.0/build/xgc.princeton/bin/xgc-es-cpp-gpu"
  string   TAU:0:0:MetaData:Cpus_allowed_list                  attr   = "0-7"
  string   TAU:0:0:MetaData:Executable                         attr   = "/autofs/nccs-svm1_home1/khuck/ECP-WDM/ecp-
wdm-coe/crusher_rocm5.2.0/build/xgc.princeton/bin/xgc-es-cpp-gpu"
  string   TAU:0:0:MetaData:Hostname                           attr   = "crusher047"
  string   TAU:0:0:MetaData:Local Time                         attr   = "2022-10-05T19:26:33-04:00"
  string   TAU:0:0:MetaData:MPI Comm World Rank                attr   = "0"
  string   TAU:0:0:MetaData:MPI Comm World Size                attr   = "256"
  string   TAU:0:0:MetaData:MPI Host Name                      attr   = "crusher047"
  string   TAU:0:0:MetaData:MPI Processor Name                 attr   = "crusher047"
  string   TAU:0:0:MetaData:MPI Unique Hosts                   attr   = "1"
  string   TAU:0:0:MetaData:Memories Allowed                   attr   = "00000000,0000000f"
  string   TAU:0:0:MetaData:Memories Allowed List              attr   = "0-3"
  string   TAU:0:0:MetaData:Memory Size                        attr   = "526858456 kB"
  string   TAU:0:0:MetaData:Mems_allowed_list                  attr   = "0-3"
  string   TAU:0:0:MetaData:Node Name                          attr   = "crusher047"
  string   TAU:0:0:MetaData:OS Machine                         attr   = "x86_64"
  string   TAU:0:0:MetaData:OS Name                            attr   = "Linux"
  string   TAU:0:0:MetaData:OS Release                         attr   = "5.3.18-150300.59.68_11.0.76-cray_shasta_c"
  string   TAU:0:0:MetaData:OS Version                         attr   = "#1 SMP Sun May 29 15:23:06 UTC 2022 (2104b
1c)"
  string   TAU:0:0:MetaData:Starting Timestamp                 attr   = "1665012393419330"
```

Timer Examples (bpls -l)

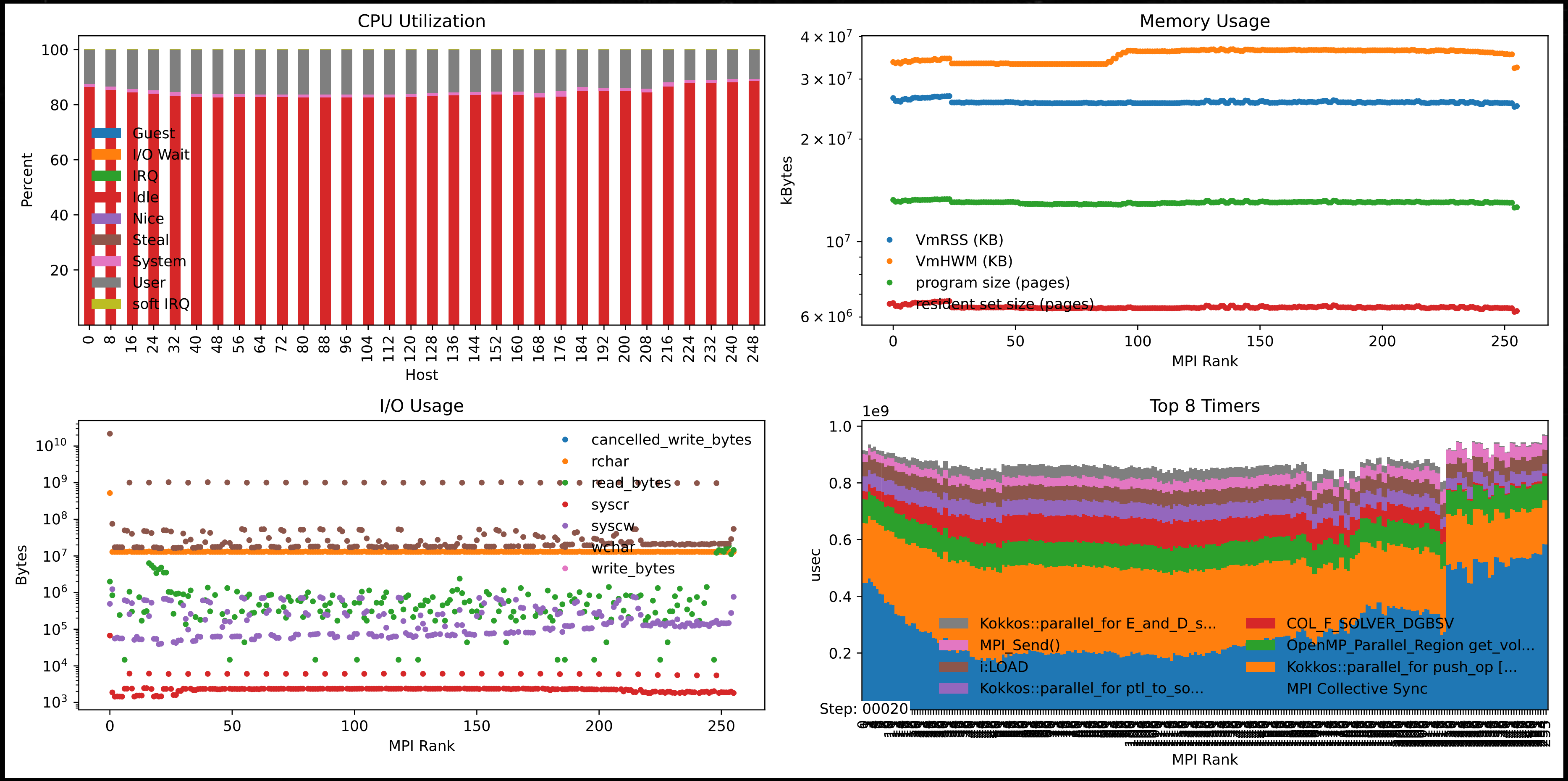
```
double   .TAU application / Calls
double   .TAU application / Exclusive TIME
double   .TAU application / Inclusive TIME
double   ADD_F0_ANALYTIC / Calls
double   ADD_F0_ANALYTIC / Exclusive TIME
double   ADD_F0_ANALYTIC / Inclusive TIME
double   ADIOS_WRITE_RESTART / Calls
double   ADIOS_WRITE_RESTART / Exclusive TIME
double   ADIOS_WRITE_RESTART / Inclusive TIME
double   ADIOS_WRITE_RESTARTF0 / Calls
double   ADIOS_WRITE_RESTARTF0 / Exclusive TIME
double   ADIOS_WRITE_RESTARTF0 / Inclusive TIME
double   ASSIGN_TO_TMP / Calls
double   ASSIGN_TO_TMP / Exclusive TIME
double   ASSIGN_TO_TMP / Inclusive TIME
double   BP4Writer::AggregateWriteData / Calls
double   BP4Writer::AggregateWriteData / Exclusive TIME
double   BP4Writer::AggregateWriteData / Inclusive TIME
double   BP4Writer::BeginStep / Calls
double   BP4Writer::BeginStep / Exclusive TIME
double   BP4Writer::BeginStep / Inclusive TIME
double   BP4Writer::Close / Calls
double   BP4Writer::Close / Exclusive TIME
double   BP4Writer::Close / Inclusive TIME
double   BP4Writer::EndStep / Calls
double   BP4Writer::EndStep / Exclusive TIME
double   BP4Writer::EndStep / Inclusive TIME
double   BP4Writer::Flush / Calls
double   BP4Writer::Flush / Exclusive TIME
double   BP4Writer::Flush / Inclusive TIME
double   BP4Writer::Open / Calls
double   BP4Writer::Open / Exclusive TIME
double   BP4Writer::Open / Inclusive TIME
double   BP4Writer::PerformPuts / Calls
double   BP4Writer::PerformPuts / Exclusive TIME
double   BP4Writer::PerformPuts / Inclusive TIME
double   BP4Writer::PopulateMetadataIndexFileContent / Calls
double   BP4Writer::PopulateMetadataIndexFileContent / Exclusive TIME
double   BP4Writer::PopulateMetadataIndexFileContent / Inclusive TIME
```

Counter Examples (bpls -l)

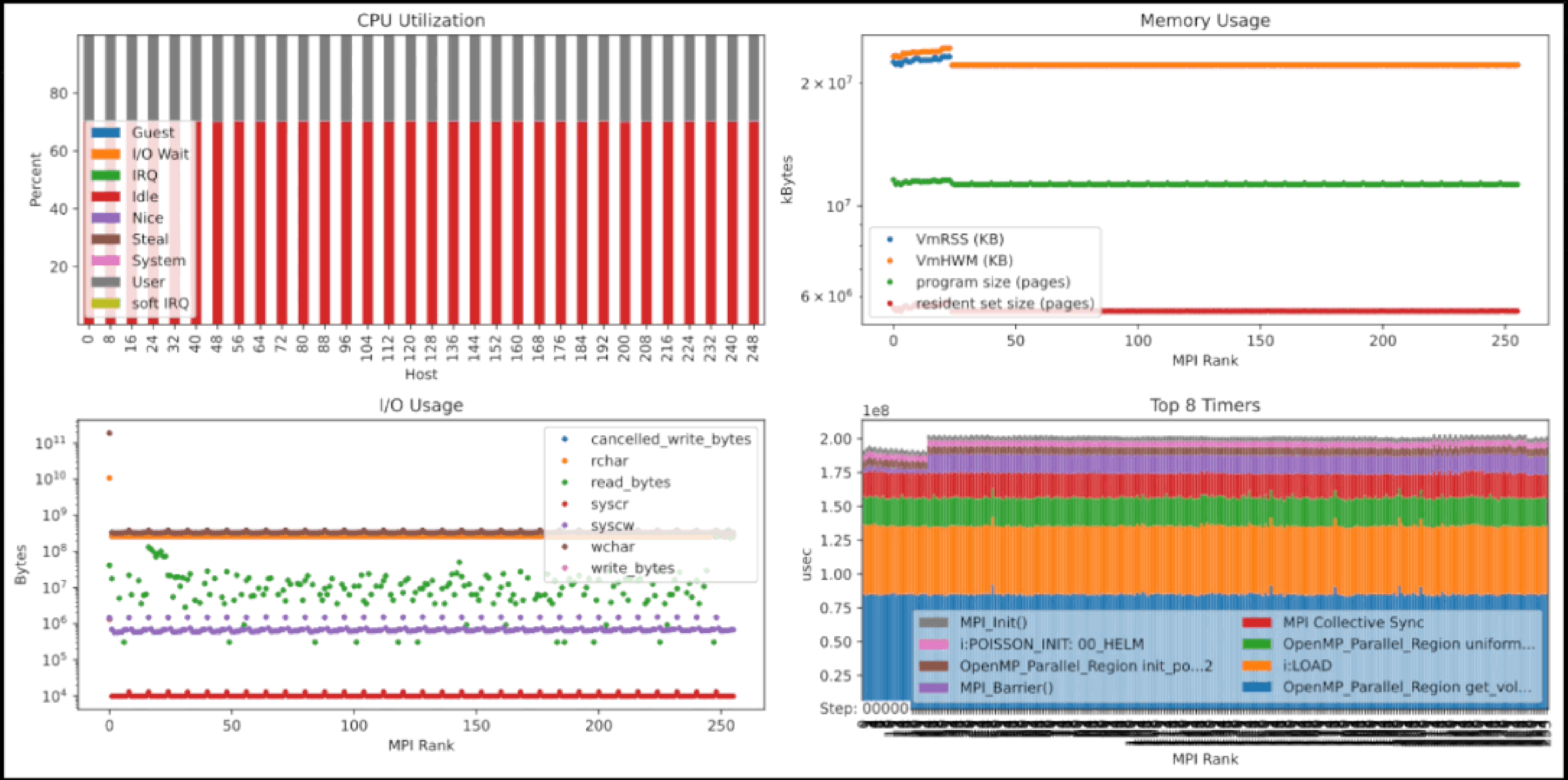
```
double    Memory Footprint (VmRSS) (KB) / Max
double    Memory Footprint (VmRSS) (KB) / Mean
double    Memory Footprint (VmRSS) (KB) / Min
double    Memory Footprint (VmRSS) (KB) / Num Events
double    Memory Footprint (VmRSS) (KB) / Sum Squares
double    Message size for all-gather / Max
double    Message size for all-gather / Mean
double    Message size for all-gather / Min
double    Message size for all-gather / Num Events
double    Message size for all-gather / Sum Squares
double    Message size for all-reduce / Max
double    Message size for all-reduce / Mean
double    Message size for all-reduce / Min
double    Message size for all-reduce / Num Events
double    Message size for all-reduce / Sum Squares
double    Message size for all-to-all / Max
double    Message size for all-to-all / Mean
double    Message size for all-to-all / Min
double    Message size for all-to-all / Num Events
double    Message size for all-to-all / Sum Squares
double    Message size for broadcast / Max
double    Message size for broadcast / Mean
double    Message size for broadcast / Min
double    Message size for broadcast / Num Events
double    Message size for broadcast / Sum Squares
double    Message size for gather / Max
double    Message size for gather / Mean
double    Message size for gather / Min
double    Message size for gather / Num Events
double    Message size for gather / Sum Squares
double    Message size for reduce / Max
double    Message size for reduce / Mean
double    Message size for reduce / Min
double    Message size for reduce / Num Events
double    Message size for reduce / Sum Squares
double    Message size for scan / Max
double    Message size for scan / Mean
double    Message size for scan / Min
double    Message size for scan / Num Events
```

Visualizing the data

- Using ADIOS2 Python API, we can read the data and visualize it



Monitoring of application data

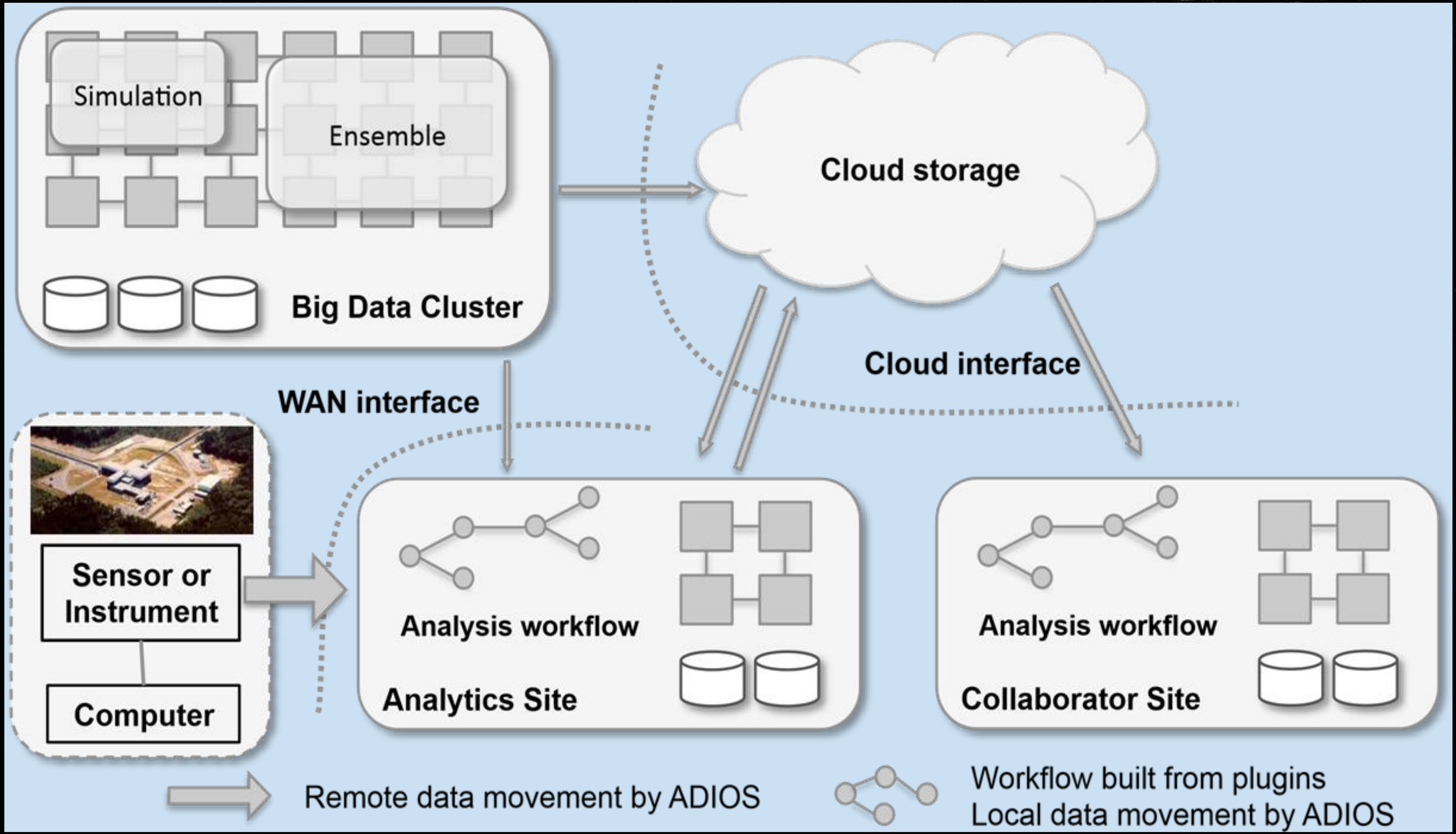


Current/Previous Acknowledgements

Outline

- 8:30 Introduction to data management at extreme scale – Scott Klasky
- 9:15 ADIOS 2 concepts and API (C++, Python, F90) – Norbert Podhorszki
- 10:00 BREAK
- 10:30 ADIOS 2 API – Norbert Podhorszki
- 11:00 Hands on with ADIOS 2 – Files – Ana Gainaru
- 12:00 Lunch
- 1:30 ADIOS @ scale – Norbert Podhorszki
- 2:00 Introduction to Paraview + ADIOS + hands on Caitlin Ross
- 3:00 BREAK
- 3:30 Introduction to TAU and TAU + ADIOS – Kevin Huck
- 4:00 Hands on with TAU + ADIOS
- 4:15 Staging with ADIOS – hands 0n – Ana Gainaru
- 5:00 END

Vision: building scientific collaborative applications

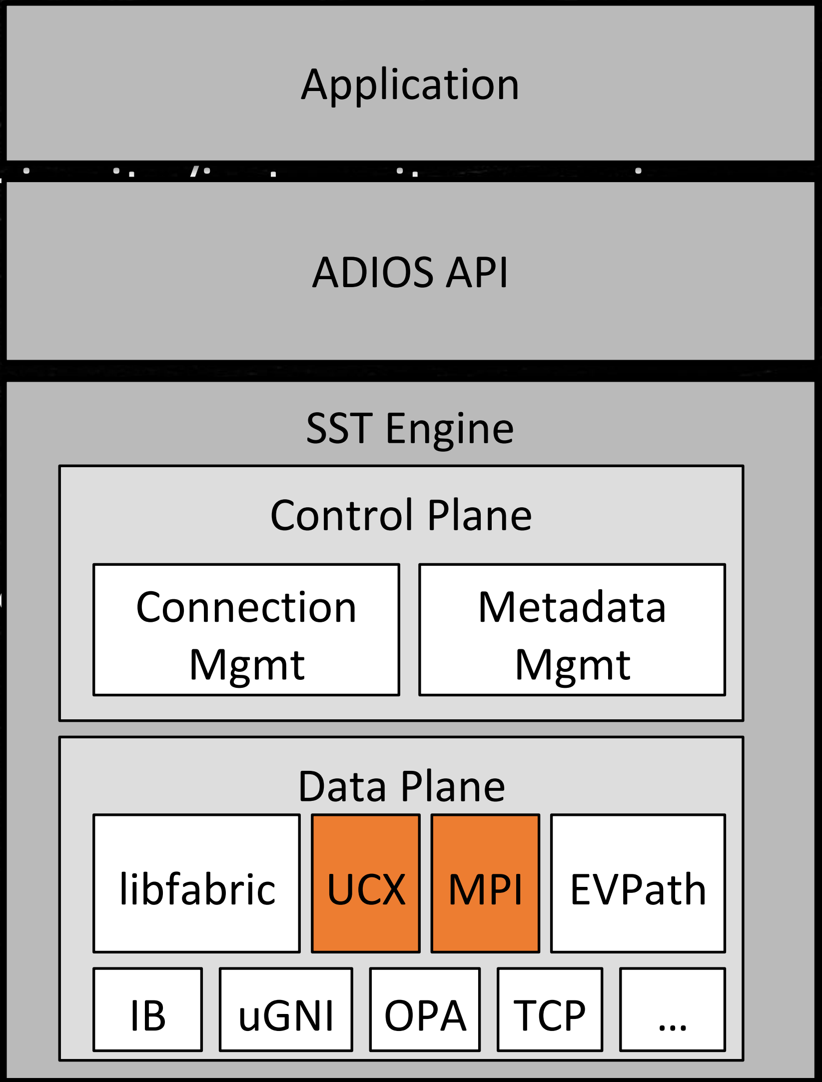


Data Staging in ADIOS

- Sustainable Staging Transport (SST)
  - In-situ infrastructure for staging in a streaming-like fashion using RDMA, MPI, SOCKETS
- DataMan
  - WAN transfers using sockets and ZeroMQ
- Staging for Strong Coupling (SSC)
  - One-sided and asynchronous MPI for strong coupling of codes
- Inline
  - Traditional in-situ execution of consumer code inside the producer code

Sustainable Staging Transport (SST)

- Direct connection between data producers and consumers for
- Designed for portability and reliability.
- Control Plane
  - Manages meta-data and control using a message-oriented protocol
  - Uses EVPath library from Georgia Tech
  - Allows for dynamic connections, multiple readers and complex flow contr
- Data Plane
  - Exchange data using RDMA, or sockets, or MPI
  - Responsible for resource management for data transfer
  - Uses libfabric/ucx for portable RDMA support or MPI
  - Threaded to overlap communication with computation and for asynchronous progress monitoring



Selecting the data plane for SST

Enable threaded MPI in the applications

```
int provided;
MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
```

XML:

```
<io name="SimulationOutput">
  <engine type="SST">
    <parameter key="RendezvousReaderCount" value="0"/>
    <parameter key="QueueLimit" value="1"/>
    <parameter key="QueueFullPolicy" value="Discard"/>
    <parameter key="OpenTimeoutSecs" value="60.0"/>
    <parameter key="DataTransport" value="MPI"/>
  </engine>
</io>
```

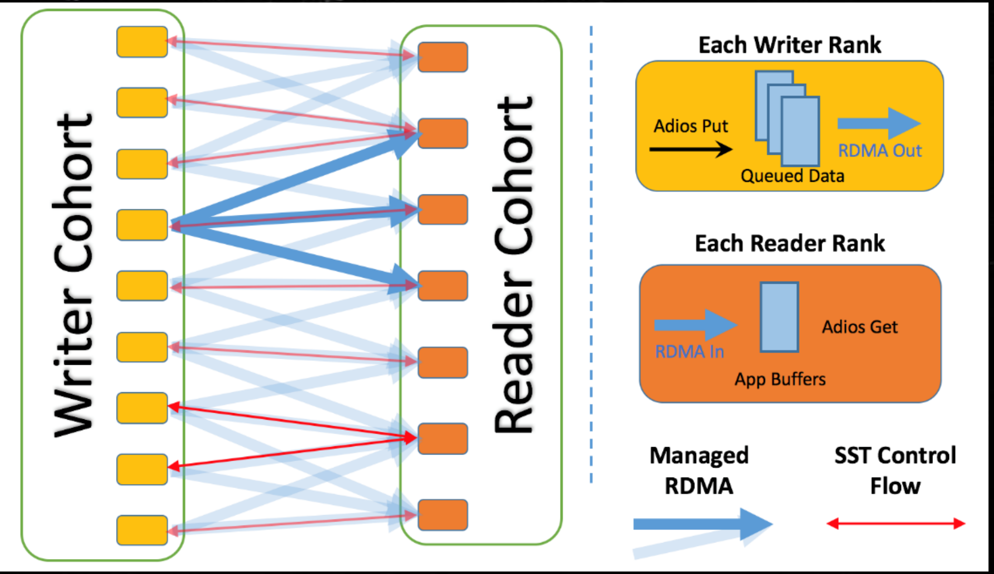
Options: MPI, UCX, WAN, fabric

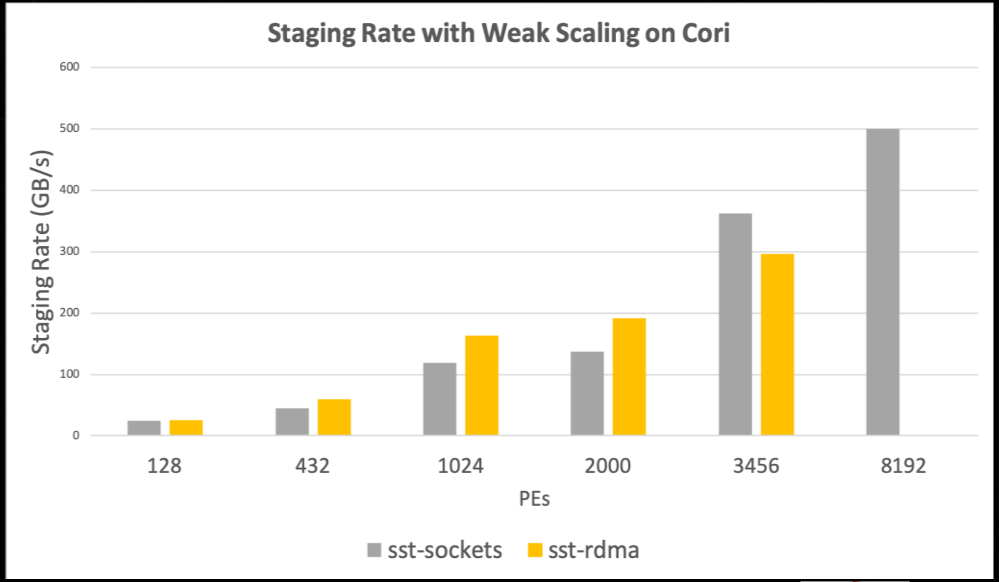
Source code:

```
adios2::ADIOS ad = adios2::ADIOS(configfile, comm);
adios2::IO io = ad.DeclareIO("SimulationOutput");
io.SetParameter("DataTransport", "MPI");
```

Sustainable Staging Transport (SST)

- Data is staged in writer ranks' memory.
  - Metadata is propagated to subscribed readers, reader ranks perform indexing locally.
  - Metadata structure is optimized for single writer, N reader workflows.
  - Supports late joining/early leaving readers.
- Modular design
  - Well-encapsulated interface between DP and CP.
  - Multiple inter-changeable DP implementations.
- RDMA for asynchronous transfer
  - Currently tested and performant for GNI, IB (verbs), OPA (verbs/psm2).





SSC (Staging for Strong Coupling)

- An ADIOS 2 engine using MPI for portability and performance
- Features
  - Use a combination of one-sided MPI and two-sided MPI methods for flexibility and performance.
  - Use threads and non-blocking MPI methods to hide communication time.
  - Optimized for fixed I/O pattern – push data to consumer
- Target Applications
  - XGC, Gene, or Gem (Edge-core strong coupling)
  - Other ECP applications requiring strong coupling or always-on in-situ analysis/visualization

DataMan

- An ADIOS 2 engine subroutine designed for wide area network data transfer, data staging, near-real-time data reduction and remote in-situ processing
- Designed with following principles:
  - Flexibility: allowing transport layer switching (ADIOS BP file, ZeroMQ, google-rpc etc.)
  - Versatility: supporting various workflow modes (p2p, query, pub/sub, etc.)
  - Adaptability: allowing adaptive data compression based on near-real-time decision making
  - Extendibility: taking advantage of all ADIOS transports and operators, and other potential third-party libraries. For example, DataMan can use ZFP, Bzip, SZ that have been built into ADIOS, as well as any compression techniques that will be built into ADIOS in future.
- Features
  - Step aggregation to improve data rate, by sacrificing latency.
  - Lossy compression to reduce data size to be transferred, by sacrificing latency and precision.
  - Fast mode to improve latency and data rate, by sacrificing reliability.
  - Combinations of features above to achieve a certain set of performance requirements.
- Target Applications:
  - ECP applications requiring wide area network data transfer and adaptive data reduction
  - Square Kilometer Array observational data
  - KSTAR fusion experimental data

Inline engine: in-process in situ visualization

- An ADIOS 2 engine providing in-process communication between the writer and reader

- Features
  - Consumer has zero-copy access to the local data block of the producer's data in memory
  - Synchronous execution of consumer code, during producer's EndStep() call

- Target Applications
  - Traditional in situ visualization routines that scale well with the producer's size

- Caveat: Writer and reader must be created in the same IO object
  - Can't easily swap in at runtime as with other ADIOS engines

ParaView In Situ Engine plugin

- ADIOS plugin that uses the inline engine internally, along with the Catalyst API to enable in situ visualization with ParaView

- Features
  - This plugin sets up the inline writer and reader on the same IO object for you
  - No code changes are now necessary to use the inline engine
  - The engine is instrumented with the Catalyst API calls – no need to add this to your application directly

Fides and ParaView Catalyst

- Need to set 3 environment variables:
  - ADIOS2_PLUGIN_PATH: set this to point to the directory containing libParaViewADIOSInSituEngine.so
  - CATALYST_IMPLEMENTATION_NAME=paraview
  - CATALYST_IMPLEMENTATION_PATHS: set this to point to the directory containing the ParaView Catalyst build. Most likely something like: /path-to-paraview-build/lib/catalyst
- Setting the Catalyst environment variables will swap in the ParaView Catalyst implementation at runtime so you can use the full

Configuring the in situ engine plugin

- Set engine type to plugin
- PluginName is a name given to it for ADIOS to keep track of it
- PluginLibrary is the name of the shared lib for the plugin
- DataModel is a JSON file describing the mesh/fields
- Script is a ParaView Catalyst script

```xml
<io name="SimulationOutput">
    <engine type="plugin">
        <!-- general plugin engine parameters -->
        <parameter key="PluginName" value="fides"/>
        <parameter key="PluginLibrary" value="ParaViewADIOSInSituEngine"/>
        <!-- ParaViewFides engine parameters -->
        <parameter key="DataModel" value="catalyst/gs-fides.json"/>
        <parameter key="Script" value="catalyst/gs-pipeline.py"/>
    </engine>
</io>
```

Application: Which staging method to use?

- System considerations
  - Staging between machines/over a WAN? **DataMan**
  - Co-located execution? **SST**
  - Availability of RDMA high-speed network? **SST** optimized for this case.
- Coupling considerations
  - Highly synchronized writer/reader? **SSC**
  - 1 writer, many readers streaming? **SST** optimized for this use case.
  - 1 reader, many small producers (e.g. AI training)? **SST**
- Performance considerations
  - Often application-specific, and not always obvious.
  - Here's where it helps to have a flexible I/O framework…

Staging I/O

- Processing data on the fly by

  1. Reading from file concurrently, or

  2. Moving data without using the file system

Design choices for reading API

- One output step at a time
  - **One step is seen at once after writer completes a whole output step**
  - streaming is not byte streaming here
  - reader has access to all data in one output step
  - as long as the reader does not release the step, it can read it
    - potentially blocking the writer
- Advancing in the stream means
  - get access to another output step of the writer,
  - while losing the access to the current step forever.

ADIOS concepts for the read API (get)

- Step
  - A dataset written within one adios_begin_step/…/adios_end_step
- Stream
  - A file containing of series of steps of the same dataset
- Open for reading as a stream
  - for step-by-step reading (both staged data and files)

```
adios2::Engine reader = io.Open(filename, adios2::Mode::Read, comm);
```

- Close once at the very end of streaming

```
reader.Close();
```

Advancing a stream

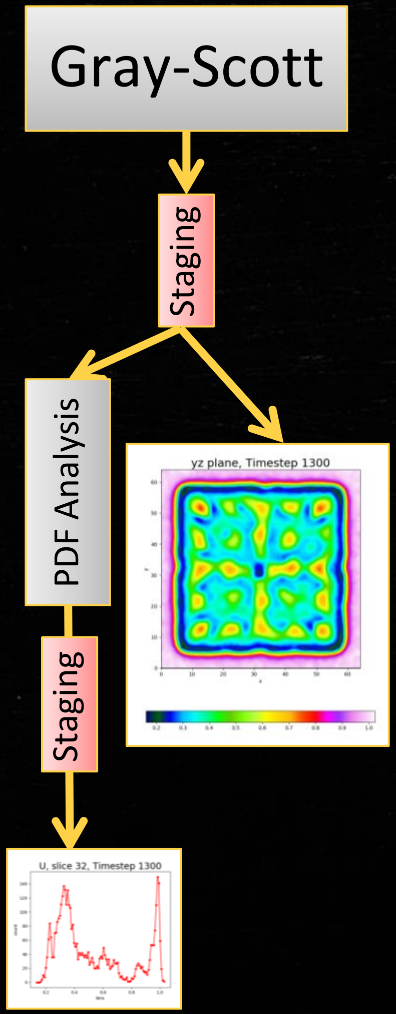
- One step is accessible in streams, advancing is only forward

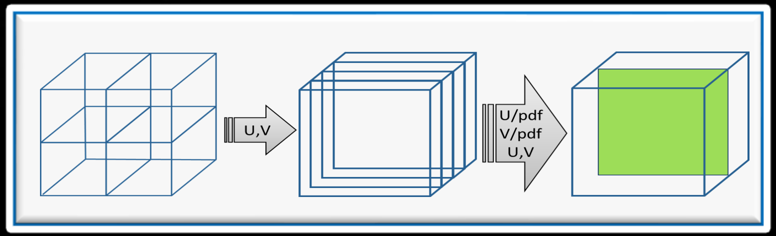
```
adios2::StepStatus read_status =
            reader.BeginStep(adios2::StepMode::Read, timeout);
if (read_status != adios2::StepStatus::OK) {
    break;
}
```

  - float timeout: wait for this long for a new step to arrive
- Release a step if not needed anymore
  - optimization to allow the staging method to deliver new steps if available

```
reader.EndStep();
```

Gray-Scott example with staging





Run the code

```
$ source ~/adios2-tutorial-source/adios2-tutorial-env.sh
$ cd ~/adios2-tutorial-source/ADIOS2-Examples/build/install/share/adios2-examples/gray-scott
$ ./cleanup.sh data
$ ls -l *.bp
ls: cannot access *.bp: No such file or directory
```

The runtime config file: adios2.xml

adios2.xml

```xml
<?xml version="1.0"?>
<adios-config>

  <!--=========================================
      Configuration for for Gray-Scott and GS Plot
    ===========================================-->
  <io name="SimulationOutput">
    <engine type=“SST">

    </engine>
  </io>

  <io name=“PDFAnalysisOutput">
    <engine type=“SST">
    </engine>
  </io>
```

Engine types
BP5
HDF5
FileStream
SST
SSC
DataMan

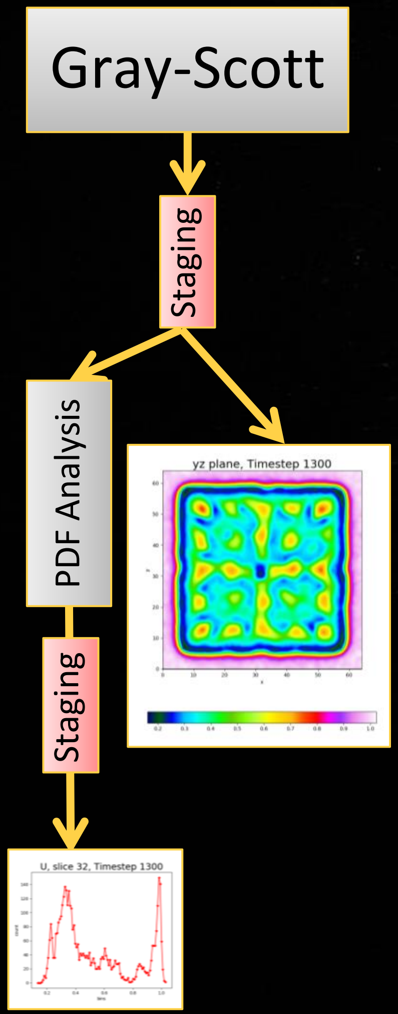
SST runtime parameters

RendezvousReaderCount = 0 [1, 2, etc]
Block Producer to wait for N Consumers at Open()
QueueLimit = 1 [2, 3, etc]
Buffer K steps in Producer’s memory for slow consumers
QueueFullPolicy = Disard or Block
What to do when Producer is faster than Consumer(s)
DataTransport = MPI, WAN, RDMA
Data is moved by MPI or TCP or libfabric (RDMA)
TCP is always available, MPI for MPICH 4.x, libfabric on systems that support it

Run the code


```
$ mpirun -n 4 ../../../bin/adios2-gray-scott settings-files.json : -n 1 ../../../bin/adios2-pdf-calc gs.bp pdf.bp 100 : -n 1 python3 pdfplot.py -i pdf.bp
PDF analysis reads from Simulation using engine type:   SST
PDF analysis writes using engine type:                  SST
Simulation writes data using engine type:               SST
…
Simulation at step 10 writing output step     1
Simulation at step 20 writing output step     2
Simulation at step 30 writing output step     3
Simulation at step 40 writing output step     4
Simulation at step 50 writing output step     5
PDF Analysis step 0 processing sim output step 3 sim compute step 40
PDF Plot step 0 processing analysis step 0 simulation step 40
Simulation at step 60 writing output step     6
PDF Analysis step 1 processing sim output step 5 sim compute step 60
Simulation at step 70 writing output step     7
PDF Analysis step 2 processing sim output step 6 sim compute step 70
Simulation at step 80 writing output step     8
…
PDF Plot step 1 processing analysis step 1 simulation step 60
```

Run the code



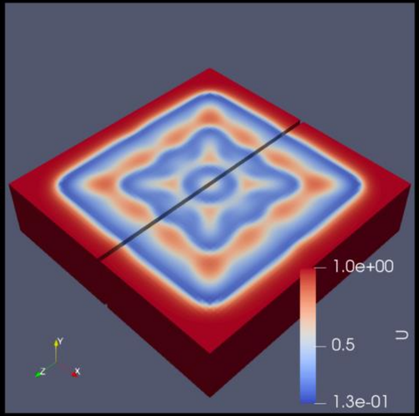

```
$ mpirun -n 4 ../../../bin/adios2-gray-scott settings-files.json : -n 1
../../../bin/adios2-pdf-calc gs.bp pdf.bp 100 : -n 1 python3 pdfplot.py -i pdf.bp :
-n 1 python3 gsplot.py -i gs.bp
```

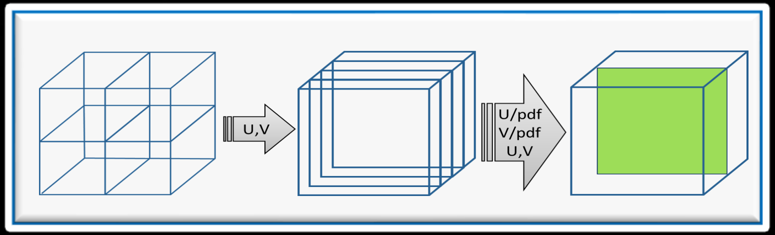
Gray-Scott example

with inline in situ

Gray-Scott

Inline in situ

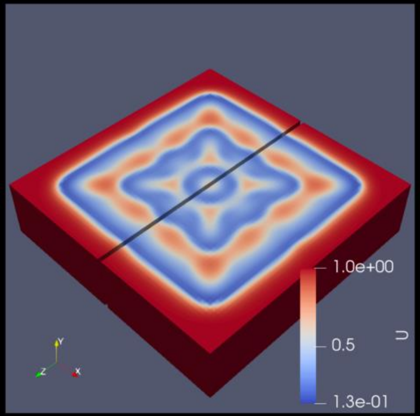




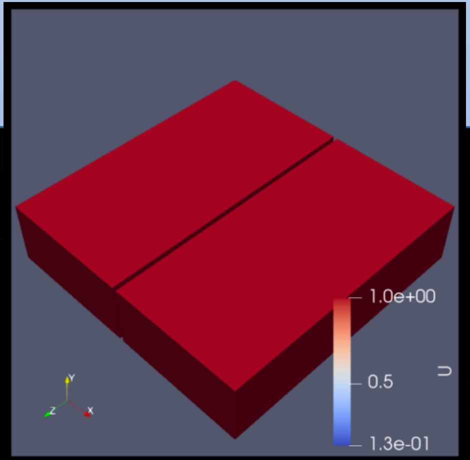
Run the code

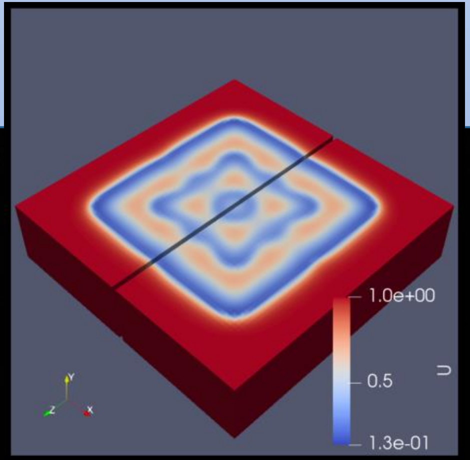
Gray-Scott

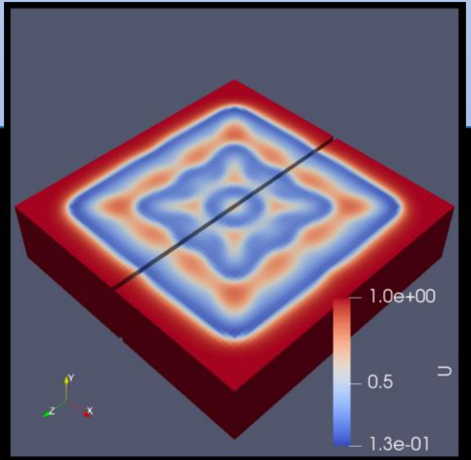
Inline in situ




```
$ ./cleanup.sh data
$ mpirun -n 4 ../../../bin/adios2-gray-scott settings-inline.json
        Catalyst Library Version: 2.0
        Catalyst ABI Version: 2
        Implementation: paraview
Simulation writes data using engine type:              plugin
…
Simulation at step 100 writing output step     1
(   1.781s) [pvbatch.0       ]          gs-pipeline.py:10    INFO| begin 
'gs-pipeline'
(   2.418s) [pvbatch.0       ]          gs-pipeline.py:65    INFO| end 
'gs-pipeline'
(   2.418s) [pvbatch.0       ]          gs-pipeline.py:52    INFO| in 
'gs-pipeline::catalyst_execute'
$ eog . &
```

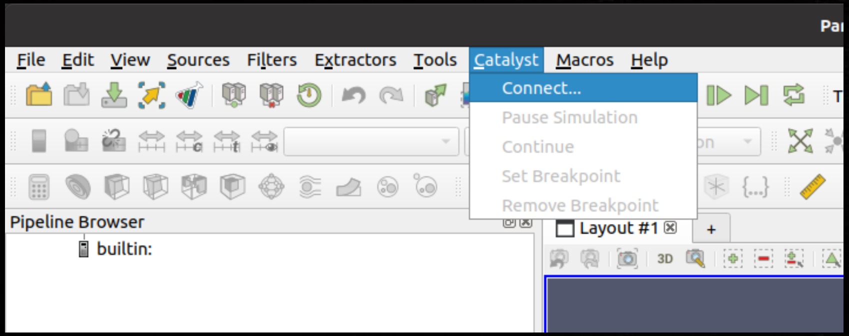




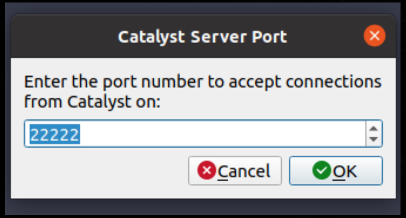


Setting up Catalyst Live in ParaView

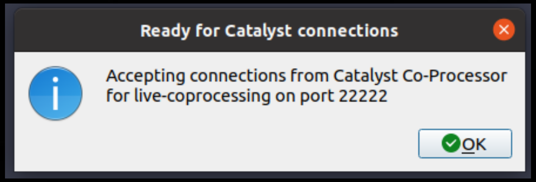
1. Connect to Catalyst



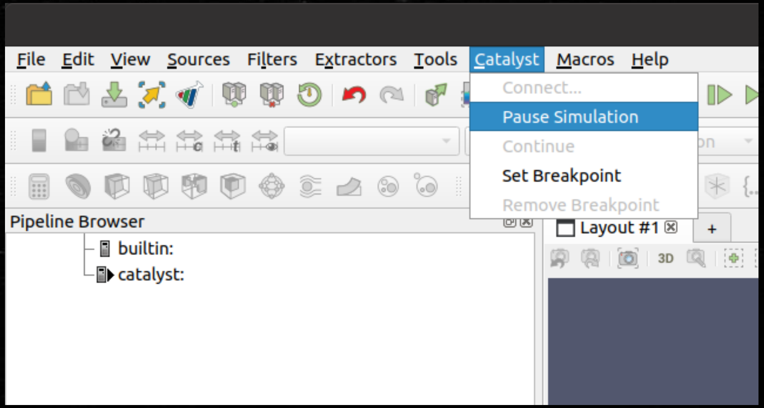
2. Enter the Port number.
We'll stick with the default 22222



3. Hit Okay. Catalyst Live is ready to connect to the simulation now.



4. Pause the simulation. This will ensure the simulation stops on the first step, so we can adjust the visualization if necessary

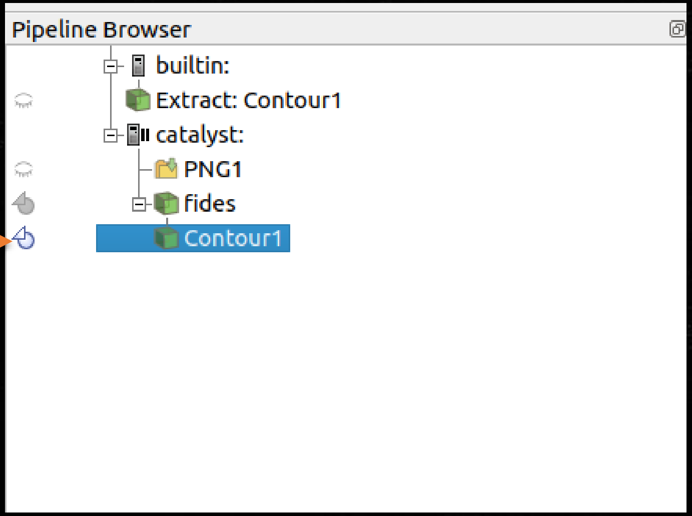


5. Run the simulation as before:

$ mpirun -n 4 ../../../bin/adios2-gray-scott settings-**inline**.json

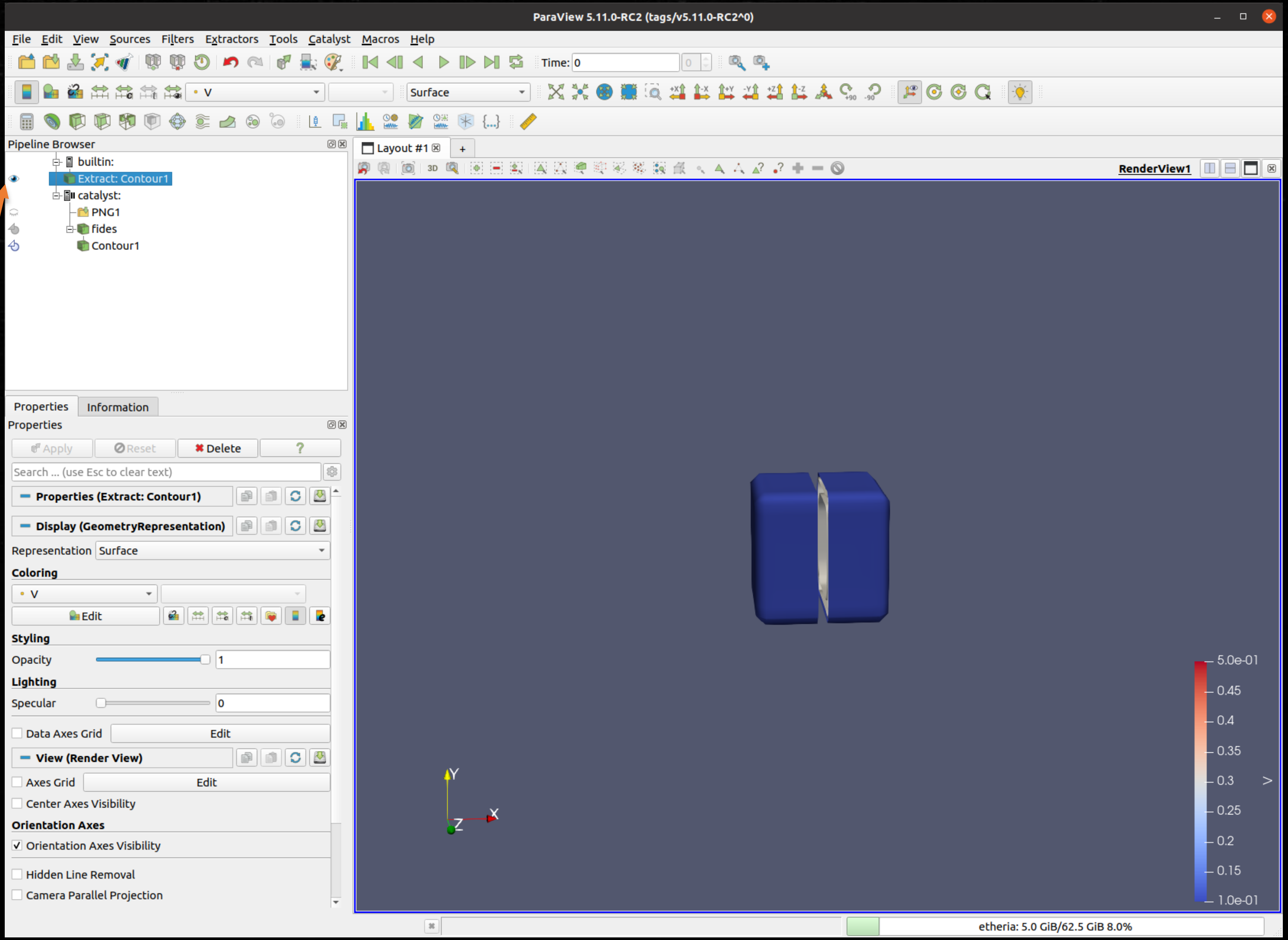
Catalyst Live

6. Select an Extract



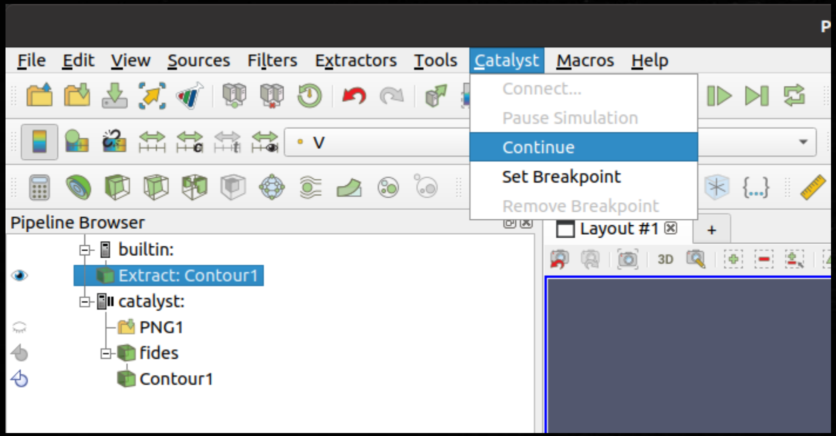
Click

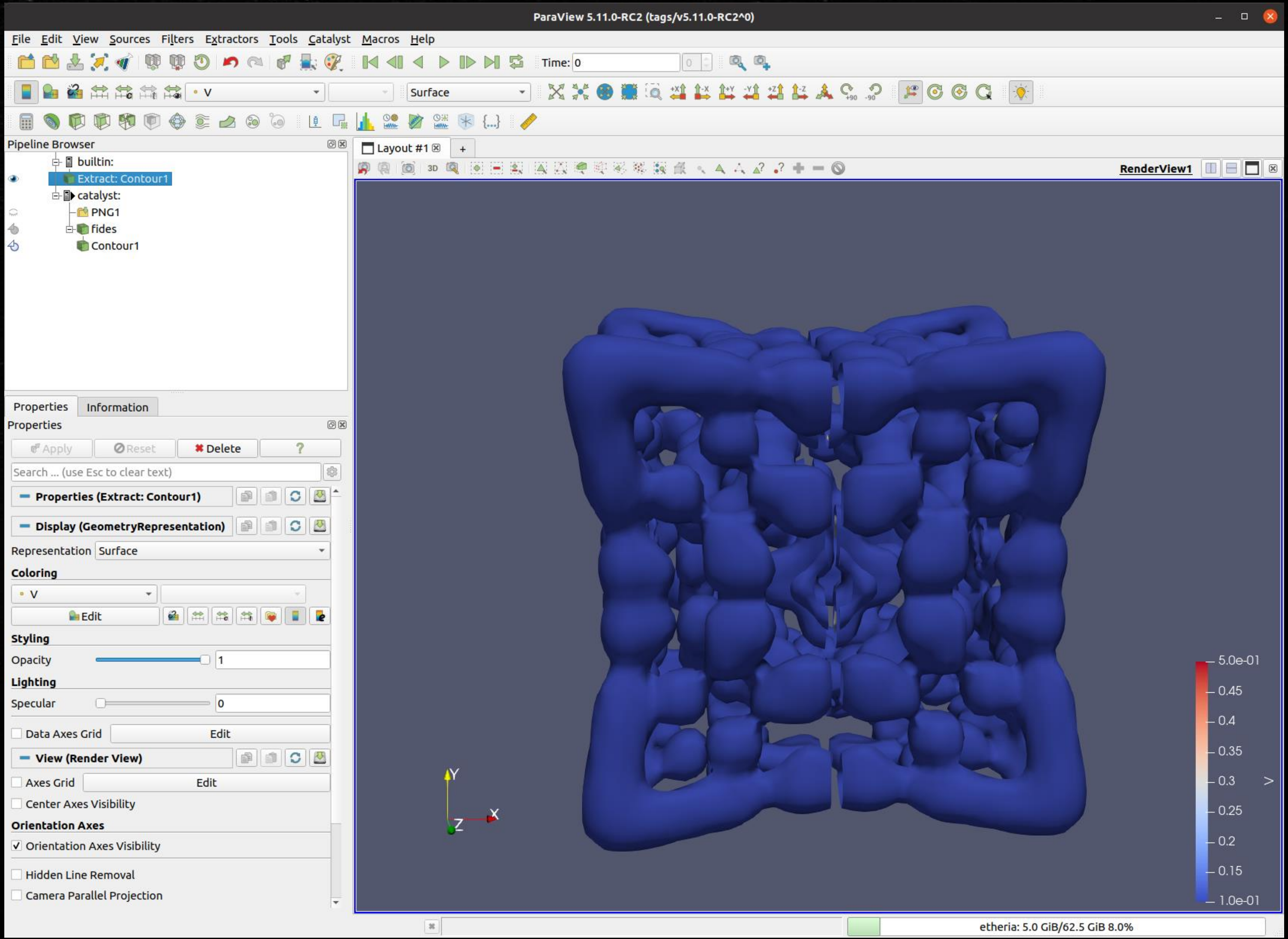
7. Toggle the visibility of the extract (”eye” button)



Catalyst Live

8. Continue the simulation from the Catalyst menu

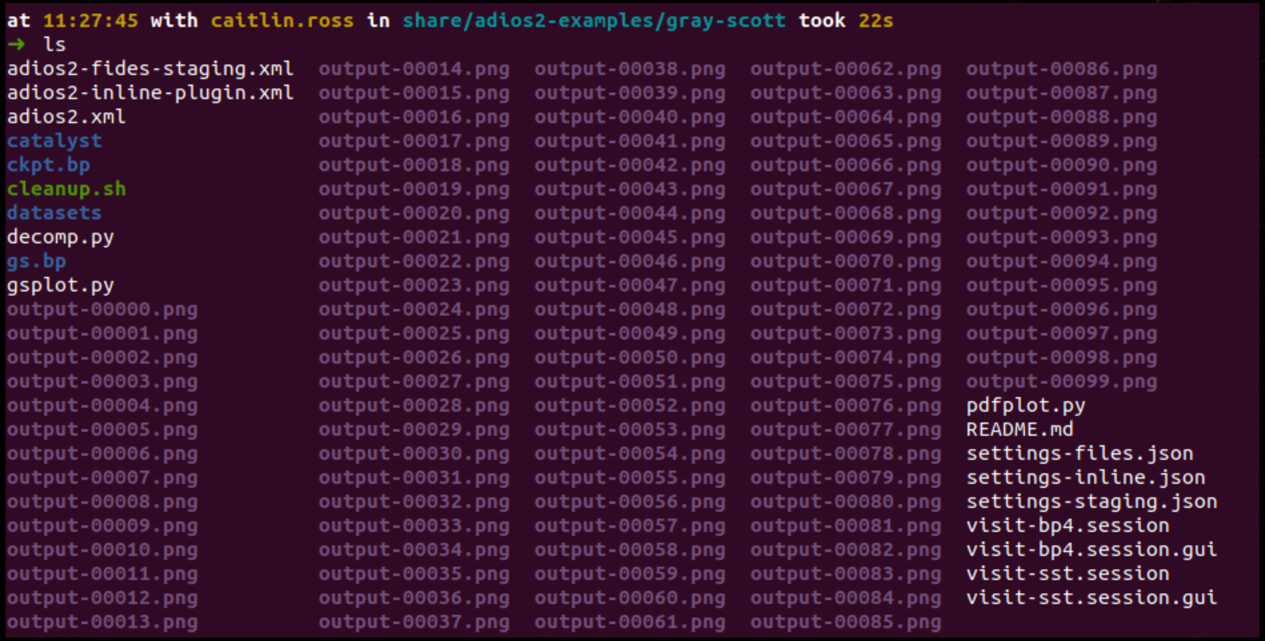


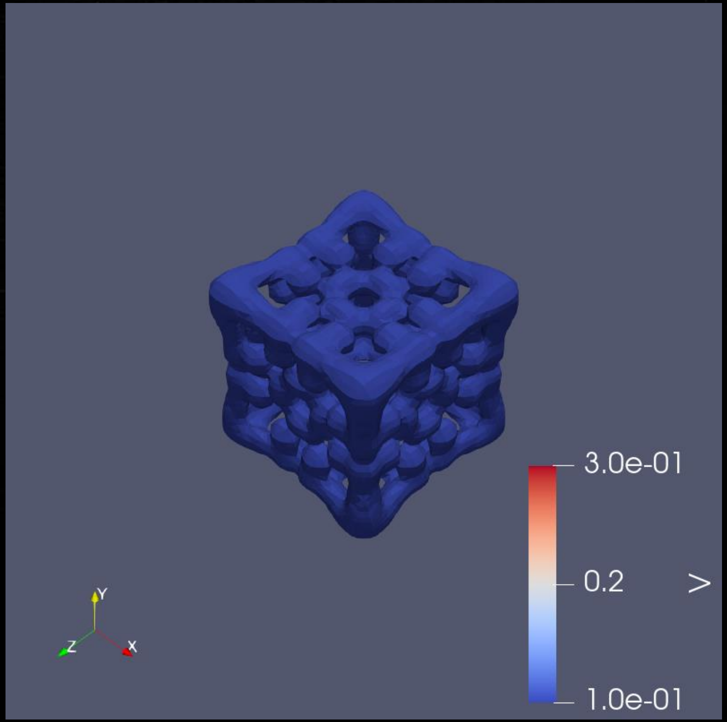


- Visualization will update as the simulation progresses.
- Can Pause/Continue as desired.
- Can add more filters to your pipeline

Hands-on Demo: Batch Visualization with ParaView and SST engine

- Batch visualization with a Python script (not using the GUI)
- Update settings-staging.json to use the adios2-fides-staging.xml
- Run from two terminals:
  $ mpirun -n 4 ../../../bin/adios2-gray-scott settings-staging.json
  $ pvbatch --force-offscreen-rendering catalyst/gs-pipeline.py -j catalyst/gs-fides.json -b gs.bp --staging





Summary

ADIOS brings a programming interface and a framework of many solutions to the generic problem of producing and consuming data

- The interface frees scientists from the limited scope of file-based data processing
  - Being fully applicable to file-based data processing
- Offering a bridge from their scientific workflows that work now to the future, where they will extend their workflows with
  - More efficient data processing
  - Interactive visualization
  - Code coupling
  - On-the-fly AI training
  - Combining experimental data with simulation data
- ADIOS + TAU gives scientists a way to monitor ADIOS performance along with writing ADIOS files and streams for on-line, off-line analytics of performance data
- ADIOS + Paraview gives scientists more advanced ways to investigate their data for on-line, off-line analytics