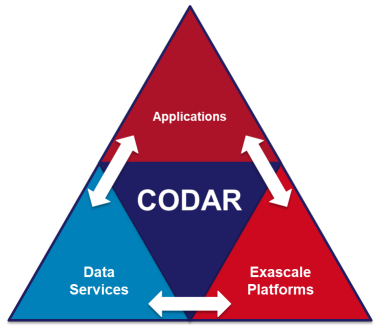
ADIOS: Storage and in situ I/O

ECP AHM – online – Apr 13, 2021

Presenters: S. Klasky[123], A. Gainaru[1], N. Podhorszki[1]

With contributions from: M. Wolf[1], C. Atkins[8], William Godoy[1], Matthew Wolf[1] , Ruonan Wang[1], Chuck Atkins[8], Greg Eisenhauer[3], Junmin Gu[7], Philip Davis[5], W. Godoy[1], Jong Choi[1], Kai Germaschewski [10], Kevin Huck [11], Axel Huebl[7], Mark Kim[1], James Kress[1], Tahsin Kurc[1], Jeremy Logan[1], Kshitij Mehta[1], Franz Poeschel[8], Eric Suchyta[1], Keichi Takahashi, Lipeng Wan[1], Pradeep Subedi, Mark Ainsworth[19], Berk Geveci [8], Ben Whitney[1]

[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]New Jersey Institute of Technology
[5]Rutgers University
[6]National Science Foundation, OAC
[7] Lawrence Berkeley National Laboratory
[8] Kitware
[9] Brown University
[10] University of New Hampshire
[11] University of Oregon



WDM, HBPS, ISEP, Sirius

Software used in this tutorial

- ADIOS 2.7.1
  - https://github.com/ornladios/ADIOS2/releases/tag/v2.7.1
- SZ 2.0.2.1
  - https://github.com/disheng222/SZ/releases
- ZFP 0.5.5
  - https://github.com/LLNL/zfp/releases
- MGARD 0.0.0.2
  - https://github.com/CODARcode/MGARD/releases/tag/0.0.0.2
- VisIt 3.1.0
  - https://wci.llnl.gov/simulation/computer-codes/visit/downloads
- ParaView 5.8.0-RC1
  - https://www.paraview.org/download/
  - Need the MPI builds for Windows to have ADIOS support

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/tree/master/source/cpp/gray-scott
- Online help:
  - ADIOS2 GitHub Issues: https://github.com/ornladios/ADIOS2/issues

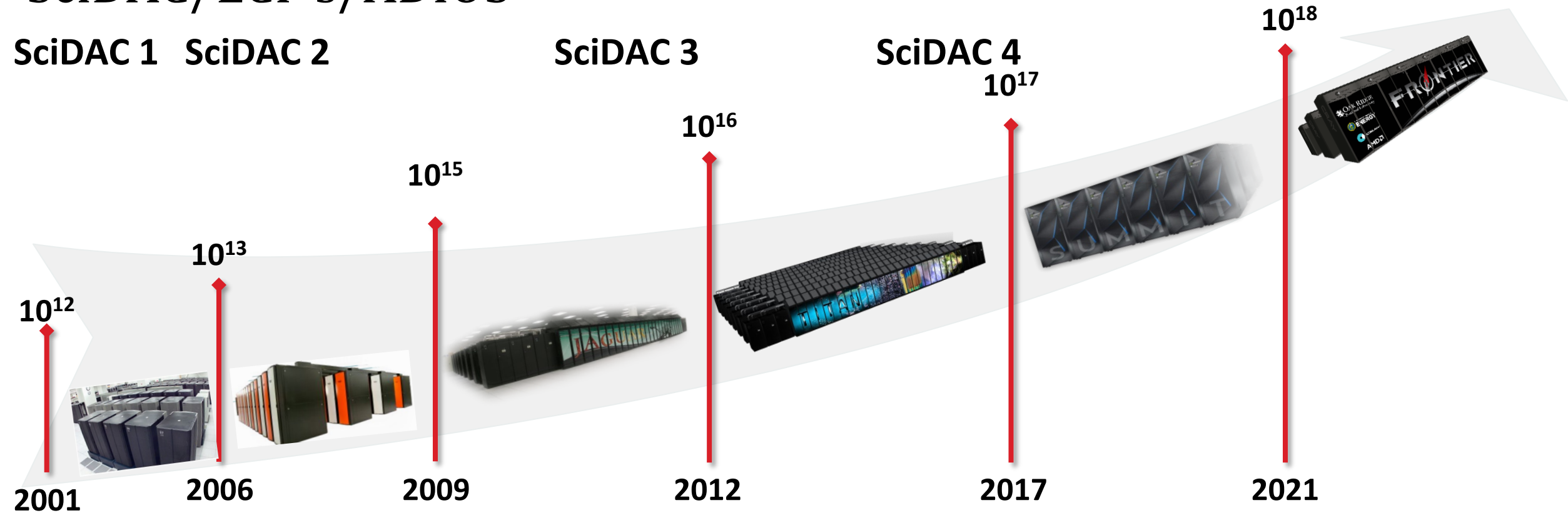
- Two movies showing the Tutorial for post-processing and on-line processing
- It's on the VM in /home/adios/Videos
  - GNOME Mplayer can play it on the VM
- https://users.nccs.gov/~pnorbert/GrayScottPost.mp4
- https://users.nccs.gov/~pnorbert/GrayScottInsitu.mp4

The Data Problem

- **Push** from Storage and Network technology, not keeping pace with the growing data demand
  - Current Storage technologies for HPC
  - New storage technologies are giving new opportunities for Storage and I/O
  - Growth of new storage tiers
  - New types of User-Defined Storage for new user-defined tiers
  - **Common Parallel File Systems**
    - Lustre
    - GPFS
    - Burst Buffer File Systems
- **Pull from** Applications
  - HPC Simulations – traditional
  - HPC Simulations – new I/O patterns
  - Experiments – streaming data
  - Observations
- **The need for self-describing data**
  - On line processing
  - Off line processing
  - Data Life-cycle

SciDAC/LCF's/ADIOS

SciDAC 1 **SciDAC 2** **SciDAC 3** **SciDAC 4**



Seaborg: 20 TF **Jaguar:** 25 TF **Jaguar 2:** 2.3 PF **Titan:** 27 PF **Summit:** 200 PF **Frontier 1500 PF**

ADIOS timeline

BP self-describing file format ADIOS 0.01

Research writing performance

Research reading performance

Data Staging ADIOS 0.1

1.2 Data Staging

ADIOS 1.0 released

1.4 weak code coupling

1.6 index, compression

1.8 Query /Indexing

1.10 Recovery/Ease of use

1.11 compression

1.12 Burst buffer

2.2 New framework in C++

2.3 Summit optimizations

SciDAC funding in purple boxes

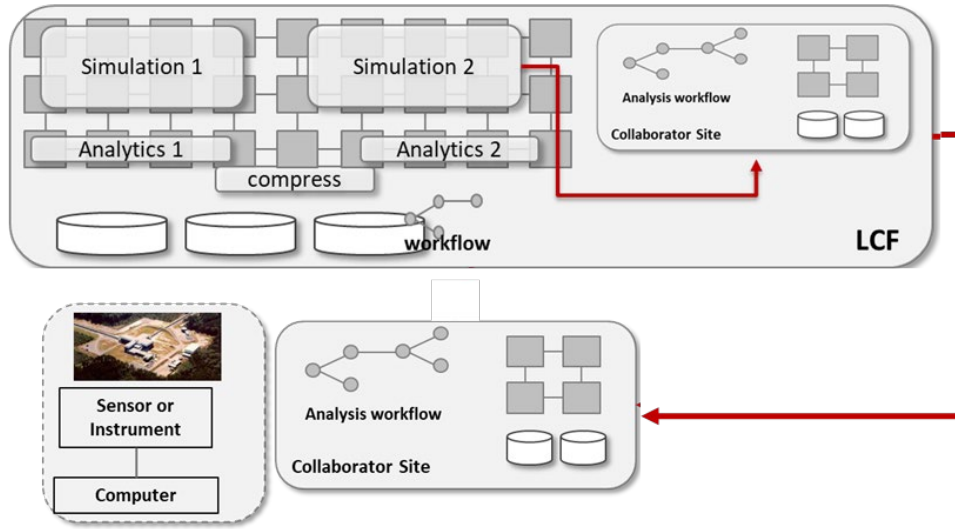
ECP funding

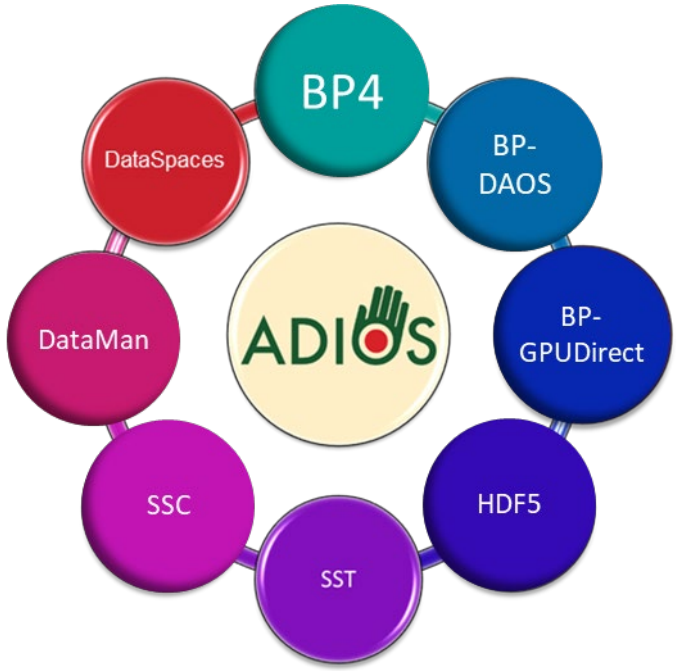
ASCR Research funding

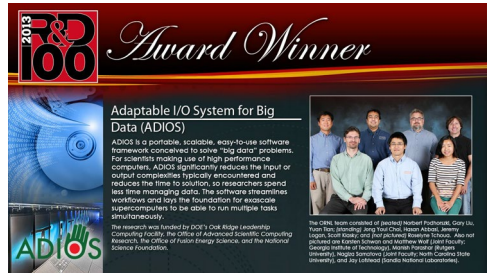
OLCF funding

ADIOS: High-Performance Publisher/Subscriber I/O framework

- An abstraction to allow for high-performance I/O to/from storage and for in situ processing
- Utilizes a publish/subscribe mechanism with self-describing data
- Optimized I/O engines for C/R, strong/loose in situ coupling WAN data streaming, and in-memory object storage
  - Fast Writing/Reading: BP4
  - DAOS optimizations: BP-DAOS
  - Write from GPU: BP-GPUDirect
  - Compatibility with HDF5: HDF5
  - Weak Code Coupling: SST
  - Tight Code Coupling: SSC
  - WAN streaming: DataMan
  - In memory object store: DataSpaces
  - Works with ECP reduction libs: MGARD, SZ, ZFP
- Typical for applications to achieve > 1 TB/s on Summit

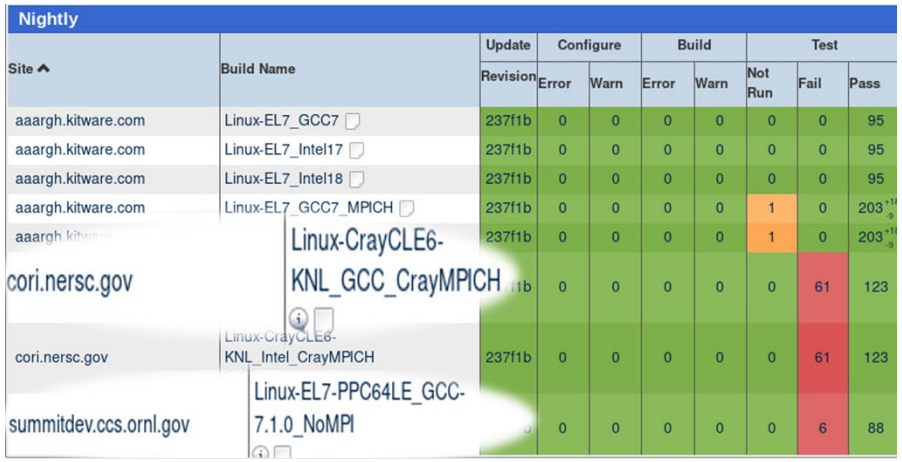




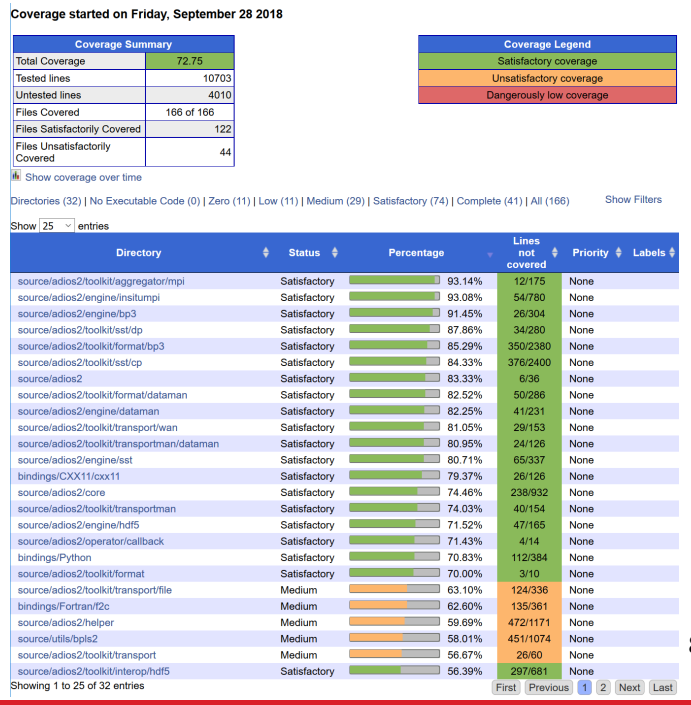


Sustainability: is a primary goal of the ADIOS project

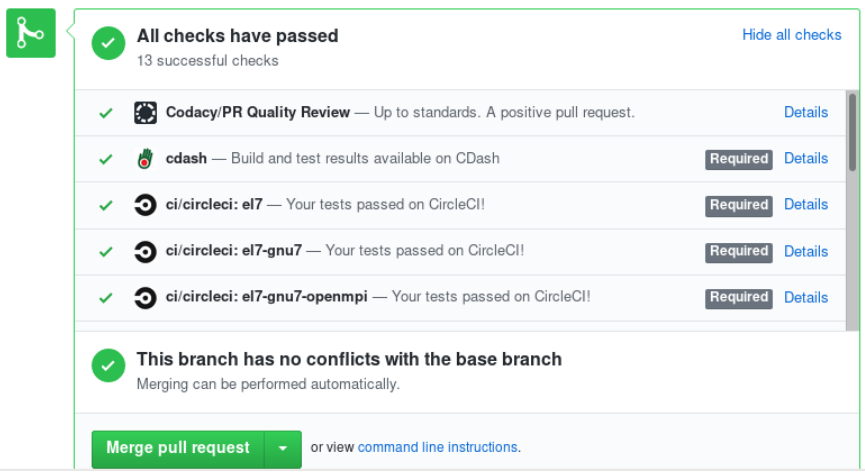
- Nightly testing
  - Testing on many different platforms
- Continuous Integration
  - Only allow tested code to be merged
  - Almost 2,000 tests for each commit
- Static and dynamic analysis reports
  - Compile-time and run-time analysis
- Code coverage
  - Level of testing
- External testing
  - Allow feedback from user projects



Code coverage report



Continuous Integration



ADIOS Approach: "How"

- I/O calls are of declarative nature in ADIOS
  - which process writes what: add a local array into a global space (virtually)
  - adios_close() indicates that the user is done declaring all pieces that go into the particular dataset in that timestep
- 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

W. Godoy, N. Podhorszki, R. Wang, et al, ADIOS 2: The Adaptable Input Output System. A framework for High-Performance Data Management, SoftwareX, 2020.

Creating I/O abstractions to accelerate I/O to storage

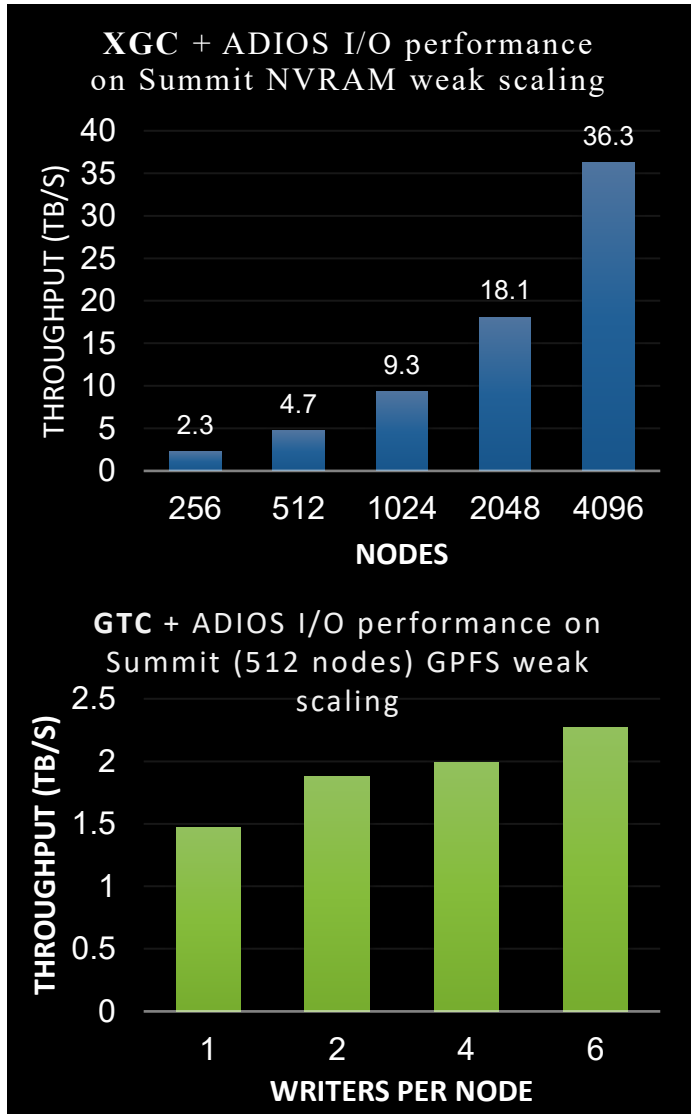
- One change in the code or input file, to specify the engine

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

writer.BeginStep()

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

writer.EndStep()

writer.Close()



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

reader.BeginStep()

adios2::Variable<double> T =
reader.InquireVariable("Temperature");

std:vector<double> t;

reader.Get(varT, t);

reader.EndStep()

reader.Close()
```

The APIs are identical for code coupling

36.3 TB/s to NVRAM

2.3 TB/s to GPFS



ADIOS performance results (measured by the app teams/not us)

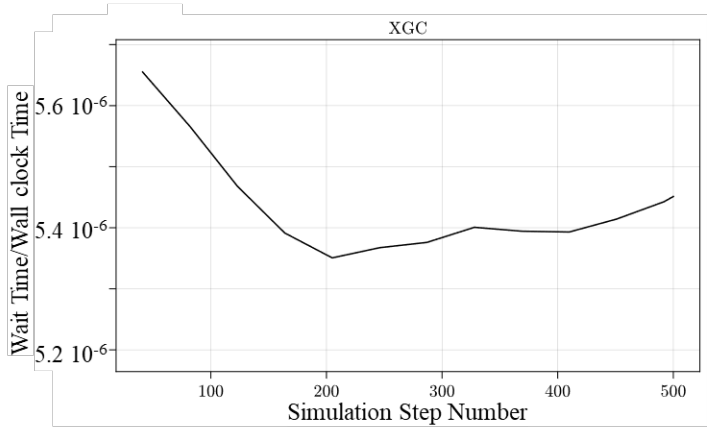
WDMApp

https://github.com/PrincetonUniversity/XGC-Devel

From the WDMApp annual ECP review

XGC on 512 Summit nodes
GENE on 6 Summit nodes
N_m = 9,640,480 vertices
N_p = 8,922 particles/vertex
Timestep = 61.4 seconds.

XGC wait time during charge coupling



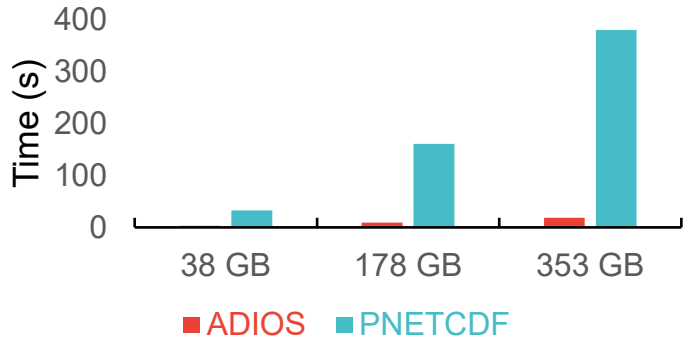
Contact: Amitava Bhattacharjee (PPPL)

E3SM-MMF

https://github.com/E3SM-Project/scorpio/tree/master

ADIOS 2.x Port is integrated into master SCORPIO

- SCREAM project evaluated it on their own and found 4-5x improvement in IO using ADIOS for TBs of data
- New I-Case stresses IO

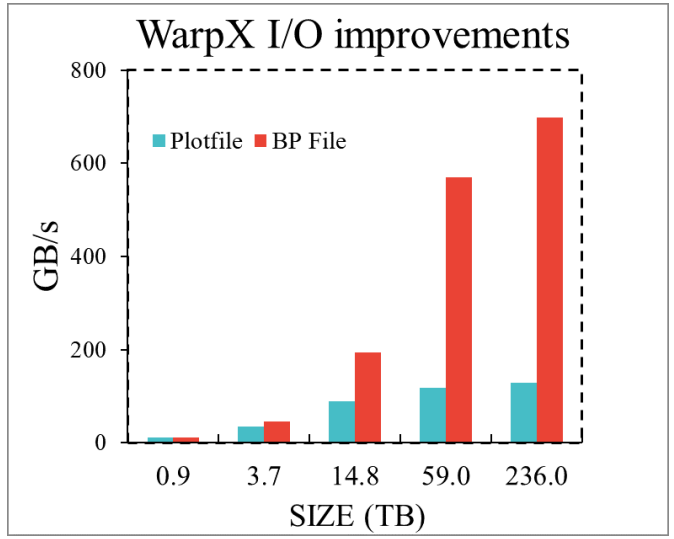


- Simulating 1 day, 5 days and 10 days
- Writing data every simulated hour
- Run on Summit, 1344 MPI processes

Contact: Mark Taylor (SNL), I-Case Peter Thornton (ORNL), SCREAM Peter Caldwell (LLNL)

WarpX

- BP4 improved append performance for ADIOS and now applications can see the benefits of that
- WarpX and in general, OpenPMD users can get high throughput



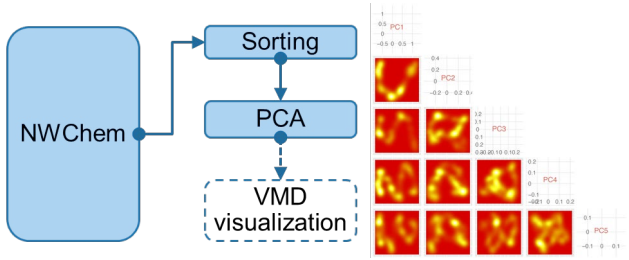
- WarpX on Summit, weak scaling
- 6 GPUs, up to 256 nodes
- ADIOS vs original AMReX Plot files

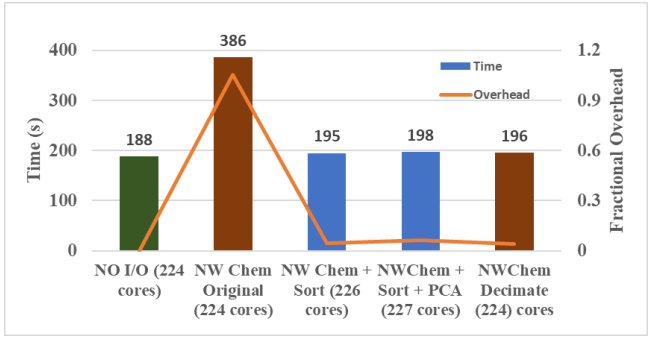
Contact: Jan-Luc Vay, Axel Huebl (LBNL)

ADIOS performance results (measured by the app teams/not us)

NWChem

- In situ sorting of atom trajectories can save 50% of runtime
- Motion correction with PCA analysis (pbdR script) in situ



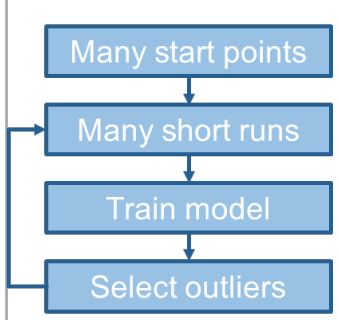


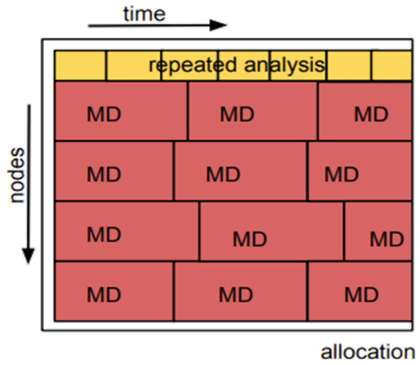
Overhead
(I/O + extra computation + extra resources)

Contact: Tjerk Straatsma (ORNL)

CANDLE/DeepDriveMD

- CODAR collaboration for accelerating sampling of macromolecule potential energy surface via online coupling
- Many concurrent MD runs + online training + inference (outlier search)
- ADIOS for async collection of MD results to training allows for continuous simulation running and training
  - Python scripts



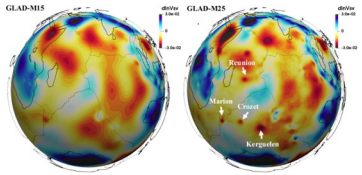


Concurrent solution is 17.5x more efficient on Summit than a sequential workflow implementation

Contact: Arvind Ramanathan, Igor Yakushin (ANL)

Specfem3D_globe

- The Adaptable Seismic Data Format (ASDF) was developed that leverages the Adaptable I/O System (ADIOS) parallel library.
- It allows for recording, reproducing, and analyzing data on large-scale supercomputers
- 1.5 PB of data is produced in every workflow step, which is fully processed later in adjoint simulation
- https://www.olcf.ornl.gov/2019/07/05/tromp-titan/
  - Python scripts



Global adjoint tomography—model GLAD-M25, Geophysical Journal International, Volume 223, Issue 1, October 2020, Pages 1–21, https://doi.org/10.1093/gji/ggaa253

Contact: Jeroen Tromp, Princeton University

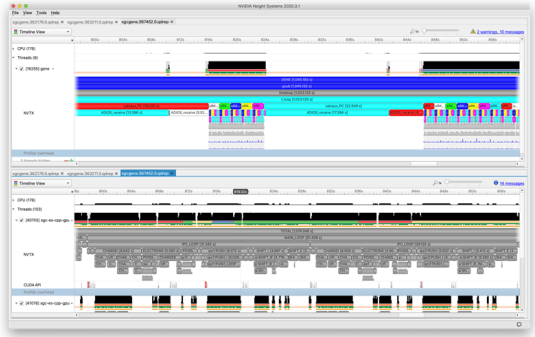
2.2.2.05 ADSE12-WDMApp: High-Fidelity Whole Device Modeling of Magnetically Confined Fusion Plasmas

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

2. Describe your performance on Summit (or equivalent) as completely as possible. Where possible, make meaningful performance comparisons, characterize GPU utilization and identify bottleneck resources.

- XGC-GENE -- coupled performance



XGC on 512 Summit nodes
GENE on 6 Summit nodes

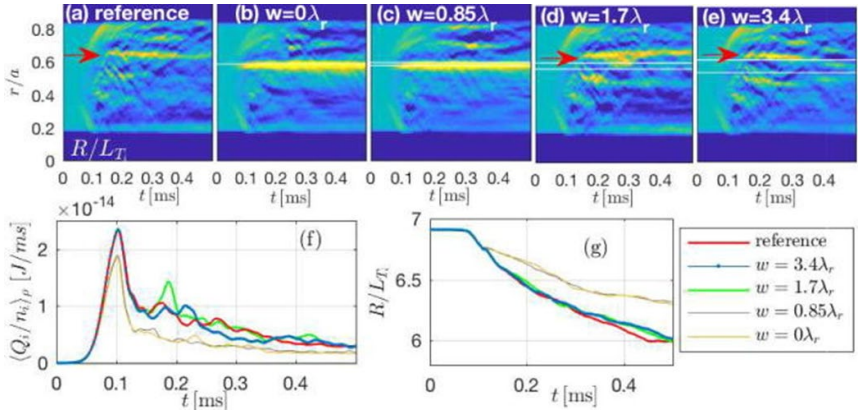
N_m = 9,640,480 vertices
N_p = 8,922 particles/vertex
timestep 61.4 seconds.

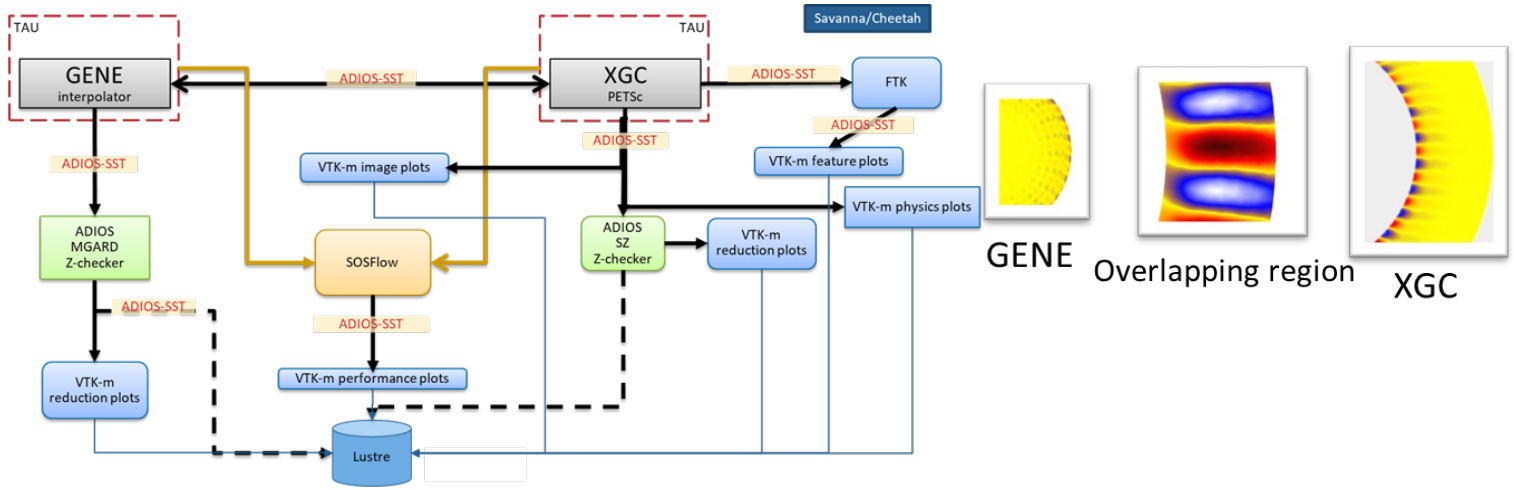
FOM_{Summit} = 74.8 = **42.3** FOM_{Titan}
When linearly scaled to full Summit

Time spent in coupling communication using ADIOS: << 1%

Little waiting by XGC.

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.

Results: Seismic Tomography Workflow (PBs of data/run)

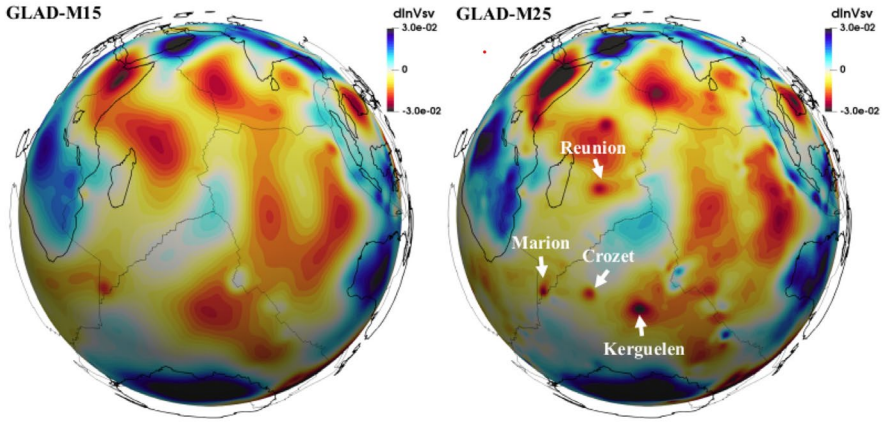
PI: Jeroen Tromp, Princeton

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.

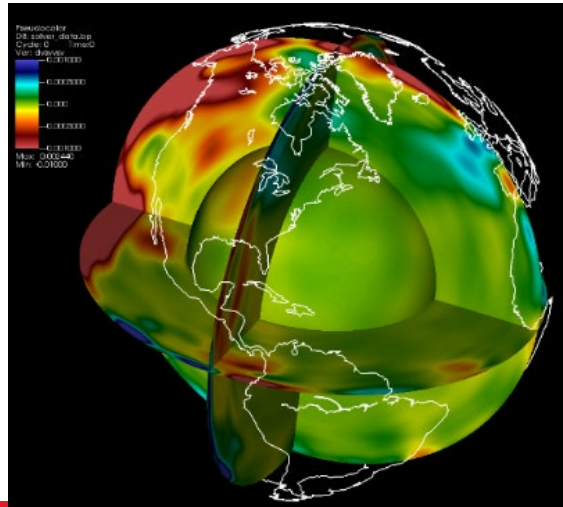


Map views at 250 km depth of vertically polarized shear wave speed perturbations in GLAD-M15 (2017) and GLAD-M25 (2020) in the Indian Ocean. New features have emerged in GLAD-M25, such as the Reunion, Marion, Kerguelen, Maldives, Seychelles, Cocos and Crozet hotspots.

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 dataset per step	235s
BP4 one dataset per job 133x reduction in # of files	156s



Wenjie Lei, Youyi Ruan, Ebru Bozdağ, Daniel Peter, Matthieu Lefebvre, Dimitri Komatitsch, Jeroen Tromp, Judith Hill, Norbert Podhorszki, David Pugmire **Global adjoint tomography—model GLAD-M25**, Geophysical Journal International, Volume 223, Issue 1, October 2020, Pages 1–21, https://doi.org/10.1093/gji/ggaa253

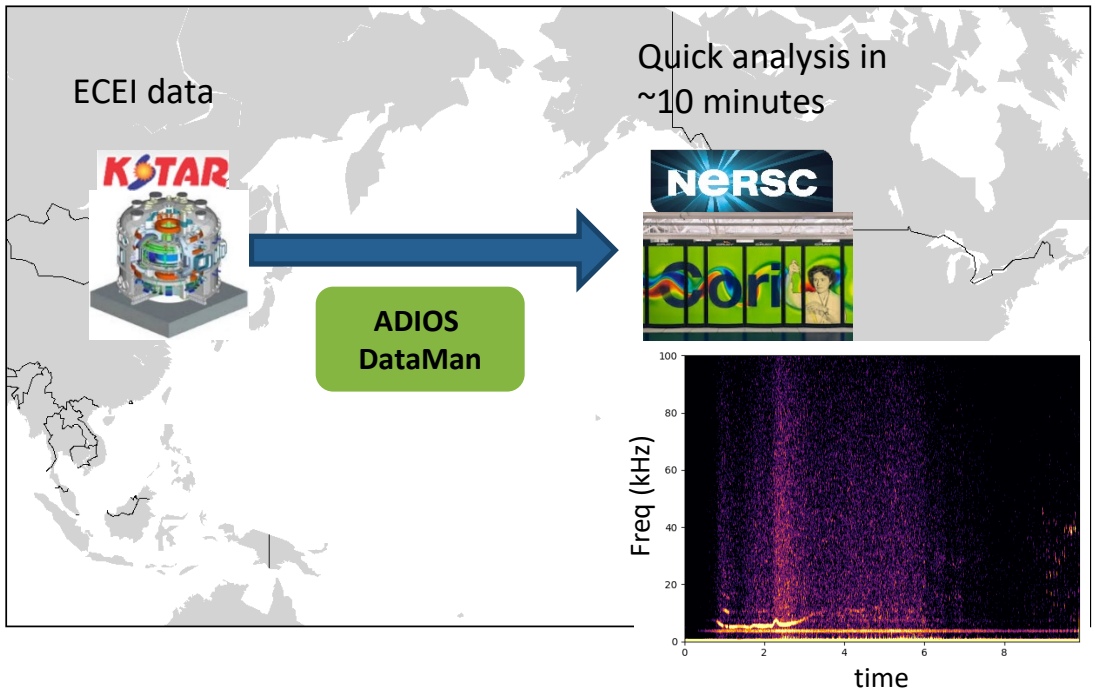
FES Highlight: Established capability for near-real time networked analysis of big KSTAR data at NERSC (PPPL, ORNL, ESnet, NERSC, KSTAR, KISTI)

Objectives

- 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

Impact

- 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
  - Quicker utilization of US HPC for KSTAR collaboration



Accomplishments

- Created end-to-end Python framework DELTA, streams data using ADIOS DataMan over WAN (at rates > 4 Gbps), asynchronously processes on multiple workers with MPI multi-threading
- Applied to KSTAR streaming data to NERSC Cori. Reduces time for an ECEi analysis from 12 hours on single-process to 10 minutes on 6 Cori nodes.
- Implemented deep convolutional neural networks for working with multi-scale fusion data, e.g. ECEi, for recognizing events of interest.[2]
- On-going: improve “adaptive” nature of data stream: adaptive compression at KSTAR source

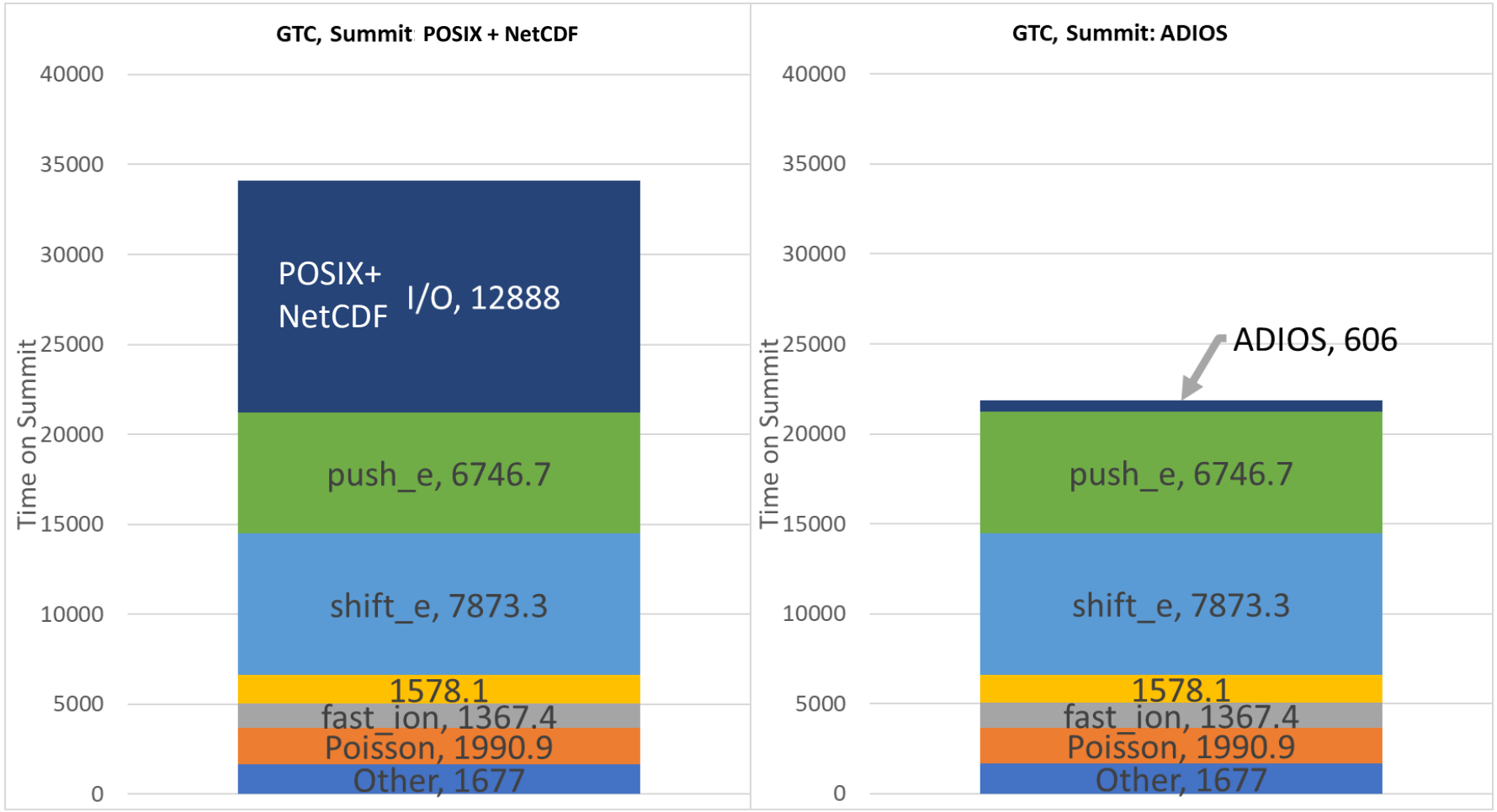
Churchill RM, Klasky et al. A Framework for International Collaboration on ITER Using Large-Scale Data Transfer to Enable Near-Real-Time Analysis. Fusion Science and Technology. 2021 Feb 17;77(2):98-108

[2]R.M. Churchill, NeurIPS 2019

GTC

PI: Zhihong Lin, UC Irvine

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



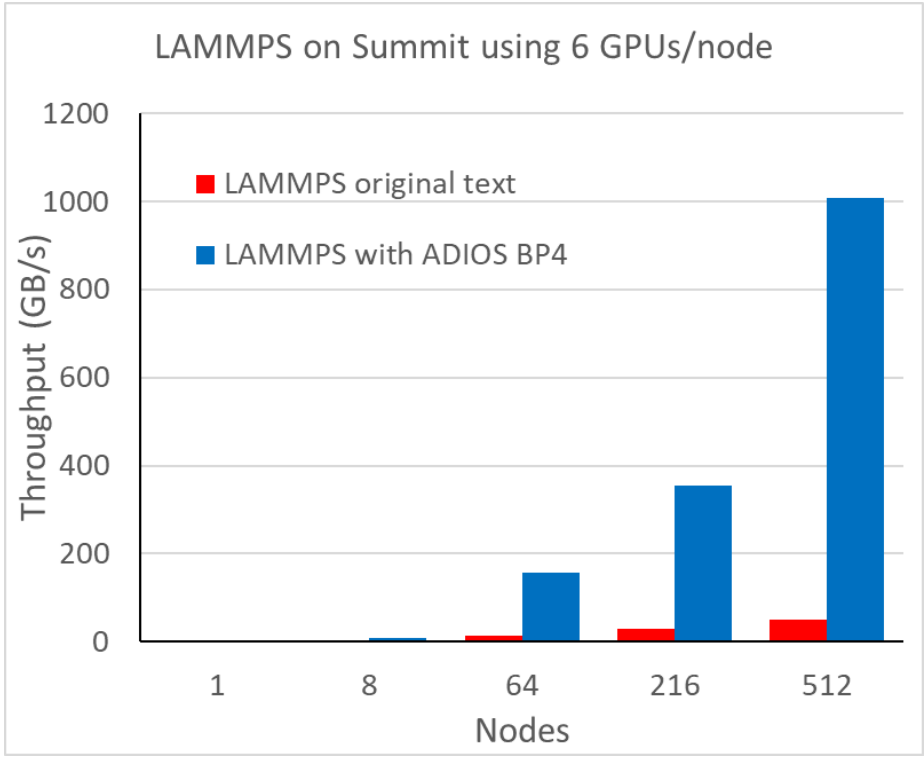
Results: LAMMPS

PI: Steve Plimpton, Sandia

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
- Concurrent reading is enabled

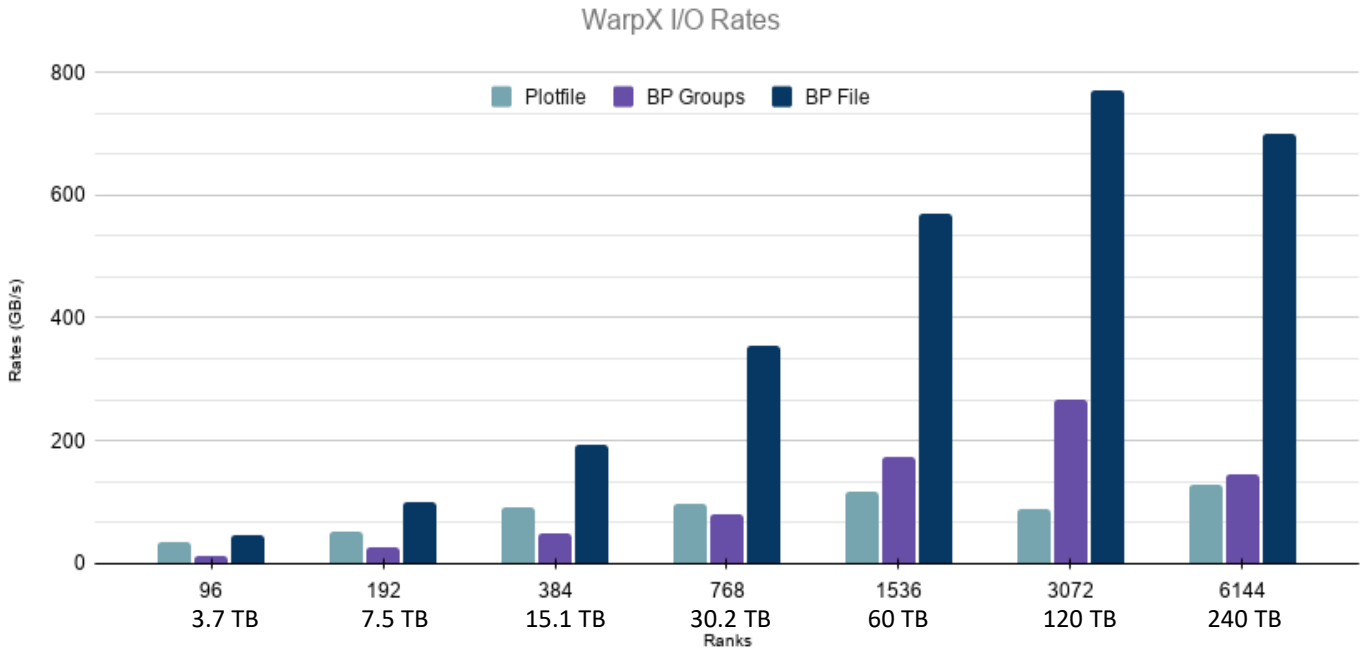
Results from ECP EXAALT Q4/FY19 milestone report (for 2.2.1.04 EXAALT ADSE04-54)
Summit 512 nodes
12B atoms, 5 TB



Results: WarpX

PI: Jan-Luc Vay, LBNL

- BPFile:
  - BP4: Use one file for all outputs.
- BPGroups:
  - BP3: Use one file / timestep
- Plot:
  - The AMReX plot file.
- One way to improve the I/O performance, is to use one ADIOS file for all time steps



Summit, 6 GPUs, 6 cores per node, up to 1024 nodes

E3SM

PI: Mark Taylor, SNL

E3SM Energy Exascale Earth System Model

Search | Select Language | GitHub | ESGF

ABOUT | RESEARCH | MODEL | DATA | PUBLICATIONS | RESOURCES

ABOUT

- **Vision and Mission**
  - Long Term Roadmap
  - Science Drivers
- **Organization**
  - The Leadership Team
  - NGD Sub-Projects
    - NGD Atmospheric Physics
    - NGD Land and Energy
    - NGD Nonhydrostatic Atmosphere
    - NGD Software and Algorithms
    - NGD BISICLES
    - NGD Coastal Waves
- **Events**
  - E3SM Conferences
    - 2019 E3SM Spring Meeting
  - E3SM Tutorials
  - All-Hands Presentations
- **Collaboration**
  - Collaboration Request
  - Ecosystem Projects
  - Closely Related Projects
- **News**

Home > About > News > PIO2 + ADIOS = Performance Improvement

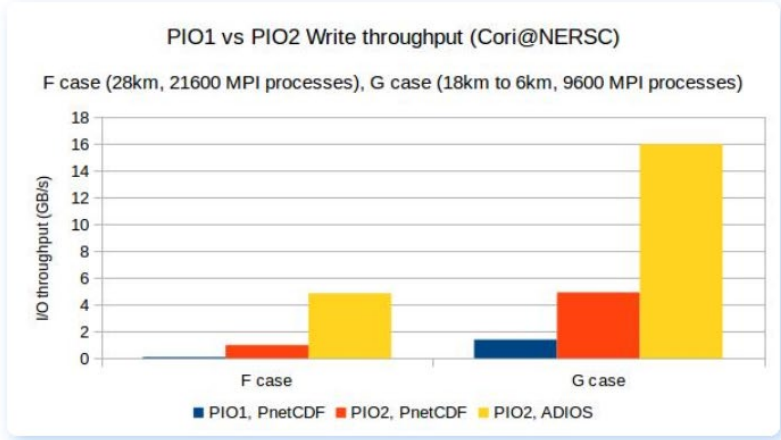
PIO2 + ADIOS = PERFORMANCE IMPROVEMENT

August 8, 2019

Releases

The **P**arallel **I**nput/**O**utput (I/O) library (PIO) is used by all the model components in E3SM for reading input and writing model output. The library supports reading and writing data using low-level I/O libraries like PnetCDF and NetCDF. The user data in E3SM is not typically decomposed across the compute processes in an "I/O friendly" way and this requires some data rearrangement, supported by PIO, before using these low-level I/O libraries to write the model data. PIO2 is the latest version of the PIO library that includes a complete rewrite of the original Fortran PIO (PIO1) library into C/C++. PIO2 also supports advanced caching and data rearrangement algorithms and includes support for more low-level I/O libraries.

In recent months developers added support in PIO to read and write data using the The Adaptable I/O System (ADIOS) library. The ADIOS library provides a flexible way to describe scientific data that may be read, written or processed outside a simulation. The library supports MPI individual I/O, MPI collective I/O, POSIX I/O, asynchronous I/O and a visualization engine to process scientific data. The library also supports a NULL output option to disable all model output. The data is written out in the ADIOS file format (which uses the .bp extension) and can be converted to the NetCDF format using a post processing tool included with PIO. Since the user data is decomposed across multiple compute processes it typically requires some data rearrangement in PIO or the low level I/O libraries to write the data efficiently in contiguous chunks, as required by the NetCDF format. Since ADIOS writes data out in multiple files and does not require data to be written out in contiguous chunks, it saves time by partially rearranging data and reducing contention in the file system.



As shown in the adjacent figure, the performance of PIO2 was measured using two E3SM simulation configurations on Cori: a configuration with high resolution atmosphere and a configuration with high resolution ocean. The high resolution atmosphere case, F case, runs active atmosphere and land models at 1/4 degree (28km) resolution, the sea ice model on a regionally refined grid with resolutions ranging from 18km to 6km and the runoff model at 1/8 degree resolution. The atmosphere component is the only component that writes output data. In this configuration all restart output is disabled and the component only writes history data, which has been historically shown to have poor I/O performance. A one-day run of this configuration generates two output files with a total size of approximately 20 GB. The high resolution ocean case, G case, runs ocean and sea ice models on a regionally refined grid with resolutions ranging from 18km to 6km. All output from the sea ice component is disabled in this configuration. A one-day run of this configuration generates 80 GB of model output from the ocean model.

The I/O write throughput for the F case was < 100 MB/s with PIO1 on Cori. PIO2 with its improved caching and data rearrangement algorithms provides a 10x improvement in the write throughput. Using ADIOS as the I/O library provides about 4x improvement over the PnetCDF library and results in a 40x improvement (ignoring post processing to convert ADIOS files to NetCDF) over PIO1. The I/O write throughput for the G case was about 1.4 GB/s with PIO1. PIO2 provides a 4x improvement in performance compared to PIO1 and using ADIOS with PIO2 provides a further 4x improvement in the write performance, resulting in a 16x improvement in write performance on Cori.

The high resolution G case was also run with PIO2 on Summit, using PnetCDF as the low-level I/O library, and the measured I/O write throughput was around 22 GB/s. Using ADIOS as the I/O library and leveraging the asynchronous I/O feature in ADIOS provided a 5x performance improvement in the write throughput compared to PnetCDF. Increasing the model output (higher output frequency) can further increase the ADIOS I/O throughput to about 7x compared to PnetCDF. This is a work in progress and the developers will continue to measure and tune performance of PIO2 on Summit.

https://e3sm.org/pio2-adios-performance-improvement/

SKA

https://www.icrar.org/summit/

WORLD'S FASTEST SUPERCOMPUTER PROCESSES HUGE DATA RATES IN PREPARATION FOR MEGA-TELESCOPE PROJECT

ANDREAS WICENEC, ICRAR, PI

Wang, Ruonan, et al. "Processing full-scale square kilometre array data on the summit supercomputer." *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2020.

TAGS

BIG DATA

OAK RIDGE NATIONAL LABORATORY

SHANGHAI ASTRONOMICAL OBSERVATORY

SHAO

SKA-LOW

SQUARE KILOMETRE ARRAY

SUMMIT

SUPERCOMPUTING

Scientists have processed 400 gigabytes of data a second as they tested data pipelines for the Square Kilometre Array (SKA) telescope.

Researchers from ICRAR in Perth, Oak Ridge National Laboratory in the US and Shanghai Astronomical Observatory in China used the world's most powerful supercomputer—Summit—to process simulated observations of the early Universe ahead of the radio telescope being built in Western Australia and South Africa.

Summit — Oak Ridge National Laboratory's 200 petaflop supercomputer. Credit: Oak Ridge National Laboratory.

The data rate achieved was the equivalent of more than 1600 hours of standard definition YouTube videos every second.

Professor Andreas Wicenec, the director of Data Intensive Astronomy at the International Centre for Radio Astronomy Research (ICRAR), said it was the first time radio astronomy data has been processed on this scale.

"Until now, we had no idea if we could take an algorithm designed for processing data coming from today's radio telescopes and apply it to something a thousand times bigger," he said.

2020 Gordon Bell Finalist

https://www.icrar.org/summit/

Computer generated image of what the SKA-low antennas will look like in Western Australia. Credit: SKA Project Office.

The billion-dollar SKA is one of the world's largest science projects, with the low frequency part of the telescope set to have more than 130,000 antennas in the project's initial phase, generating around 550 gigabytes of data every second.

Summit is located at the US Department of Energy's Oak Ridge National Laboratory in Tennessee.

It is the world's most powerful scientific supercomputer, with a peak performance of 200,000 trillion calculations per second.

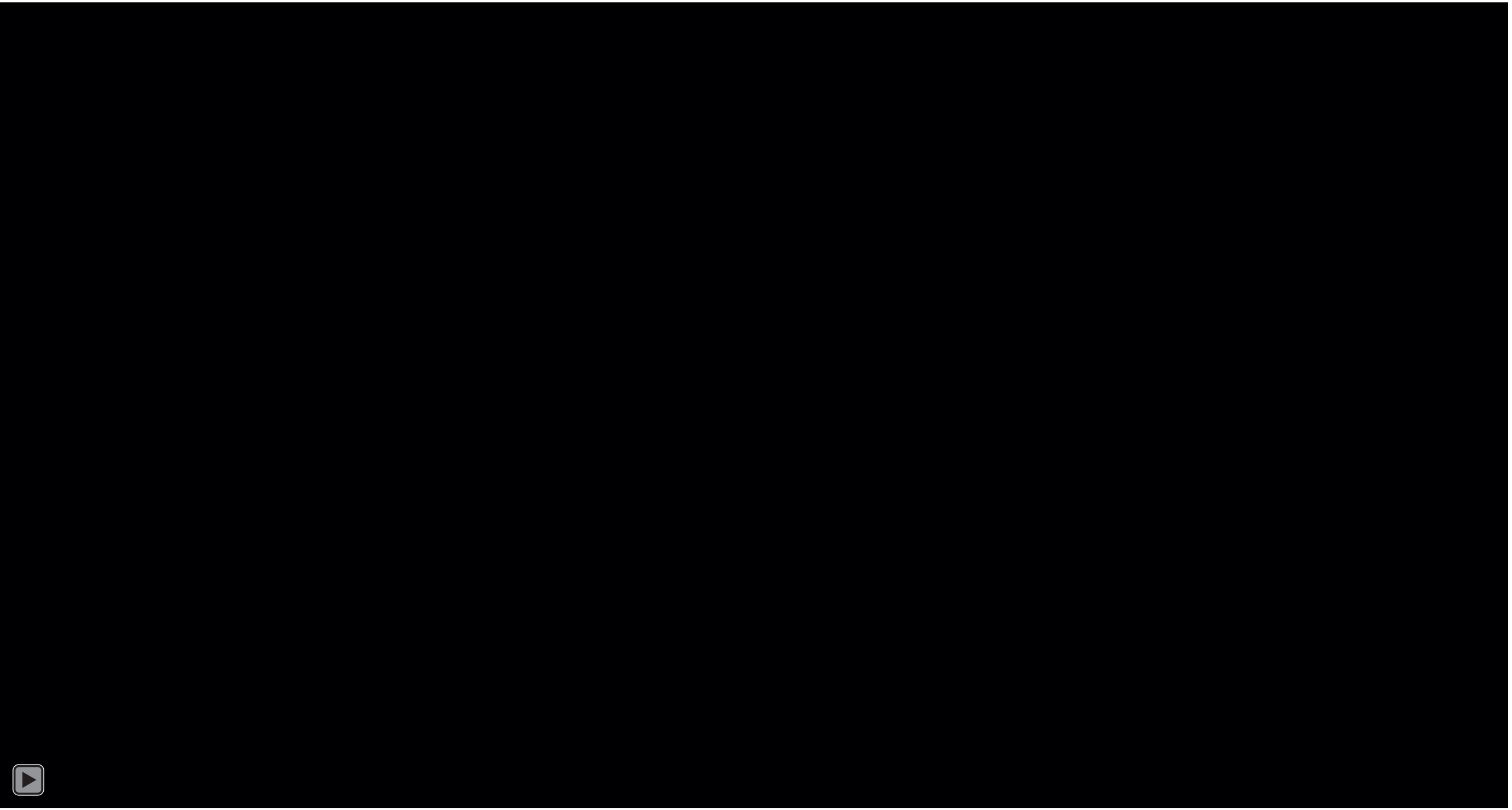
Oak Ridge National Laboratory software engineer and researcher Dr Ruonan Wang, a former ICRAR PhD student, said the huge volume of data used for the SKA test run meant the data had to be generated on the machine itself.

"We used a sophisticated software simulator written by scientists at the University of Oxford, and gave it a cosmological model and the array configuration of the telescope so it could generate data as it would come from the telescope observing the sky," he said.

"Usually this simulator runs on just a single computer, generating only a small fraction of what the SKA would produce.

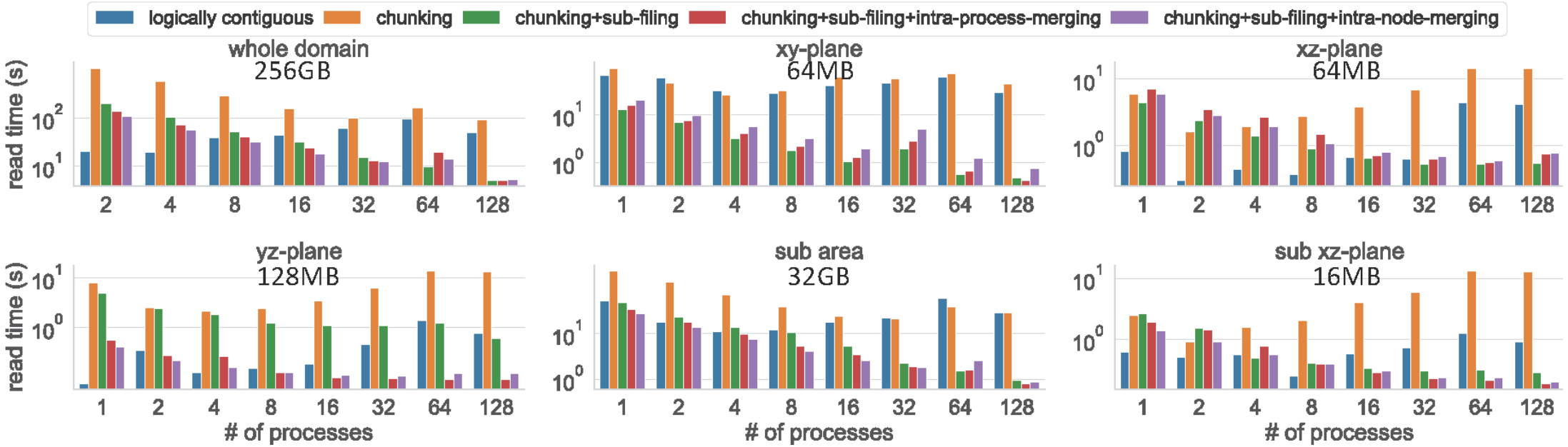
"So we used another piece of software written by ICRAR, called the Data Activated Flow Graph Engine (DALiuGE), to distribute one of these simulators to each of the 27,648 graphics processing units that make up Summit.

"We also used the Adaptable IO System (ADIOS), developed at the Oak Ridge National Laboratory, to resolve a bottleneck caused by trying to process so much data at the same time."



Writing performance is great but what about reading?

- Codes such as the WarpX code, which uses AMReX can take advantage of ADIOS-BP4 for “fast” writing
- The challenge is reading
- Development of a clustering algorithm for WarpX/AMReX data for fast writing/reading performance



Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

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/tree/master/source/cpp/gray-scott
- Online help:
  - ADIOS2 GitHub Issues: https://github.com/ornladios/ADIOS2/issues

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

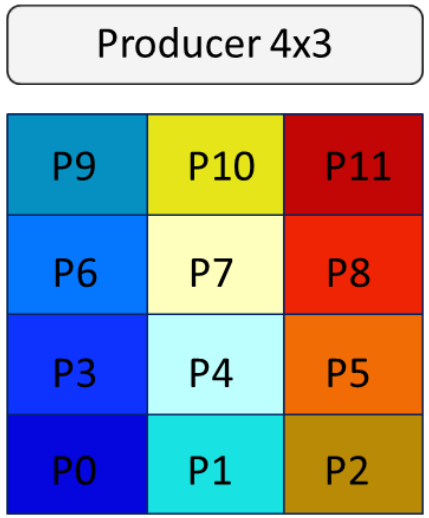
- Self-describing Scientific Data
- Variables
  - multi-dimensional, typed, distributed arrays
  - single values
    - Global: one process, or Local: one value per process
- Attributes
  - static information
    - for humans or machines
  - global, or assigned to a variable

Self-describing Scientific Data

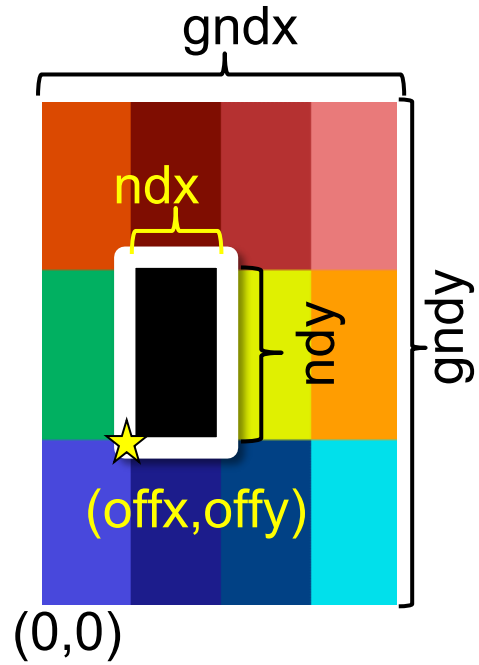
```
real     /fluid_solution/scalars/PREF                        scalar = 0
string   /fluid_solution/domain14/blockName/blockName        scalar = "rotor_flux_1_Main_Blade_skin"
integer  /fluid_solution/domain14/sol1/Rind                  {6} = 2 / 2
real     /fluid_solution/domain14/sol1/Density               {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)



Global Array
one block per process



Shape = {gndx, gndy}

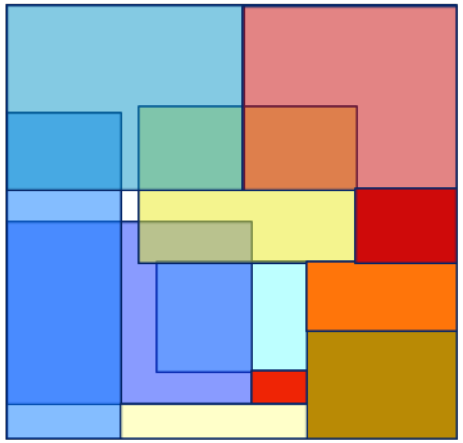
Start = {offx, offy}

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

Global Array with overlapping blocks

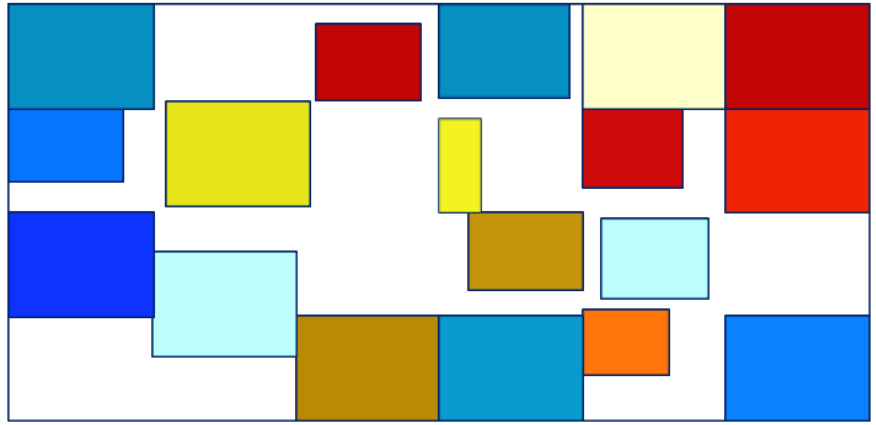


An overlapped cell has "one of the" values

12 Producers
multiple blocks per process

P9	P10	P11	P9	P7	P11
P6	P7	P8	P10	P11	P8
P3	P4	P5	P2	P4	P5
P0	P1	P2	P9	P5	P6

Global Array sparsely filled out



"White" space is not stored in ADIOS2.
Read returns 0 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

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);
```

NOTE: Normally use only 1 config file and then have different communicators for each I/O target

/home/adios/Tutorial/gray-scott/simulation/main.cpp

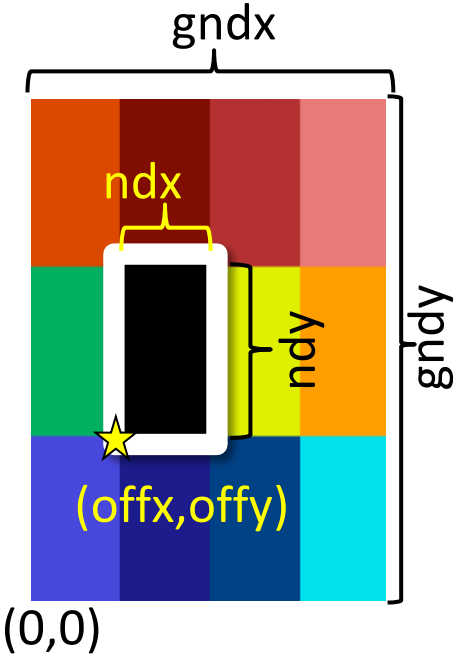
IO object

- 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!

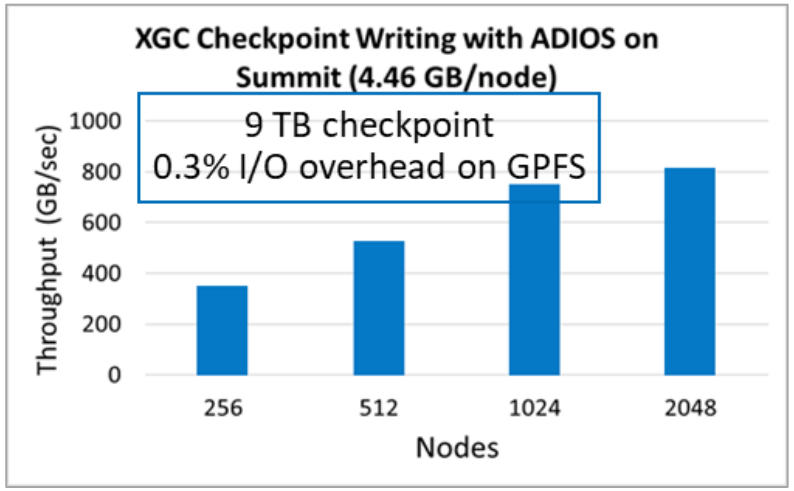
Reading is similar, but we can read from any number of procs

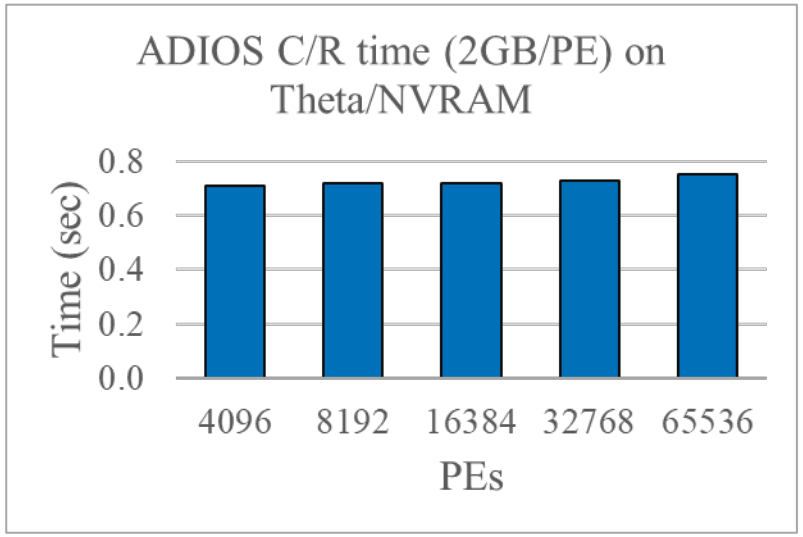
```
adios2::IO io = adios.DeclareIO("CheckpointRestart");

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()
```

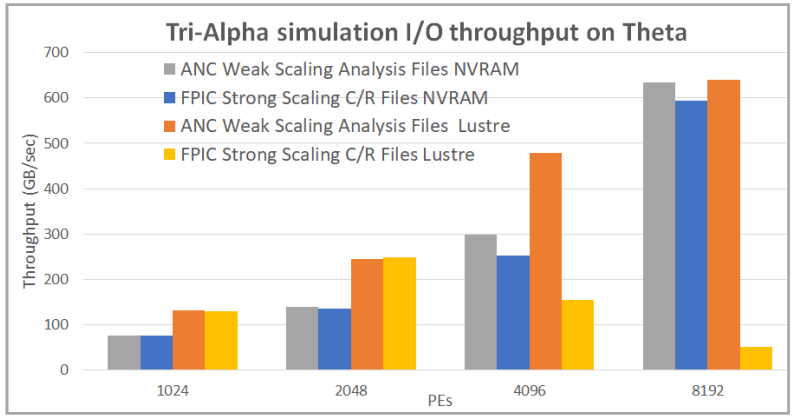
Reserve memory for T before this

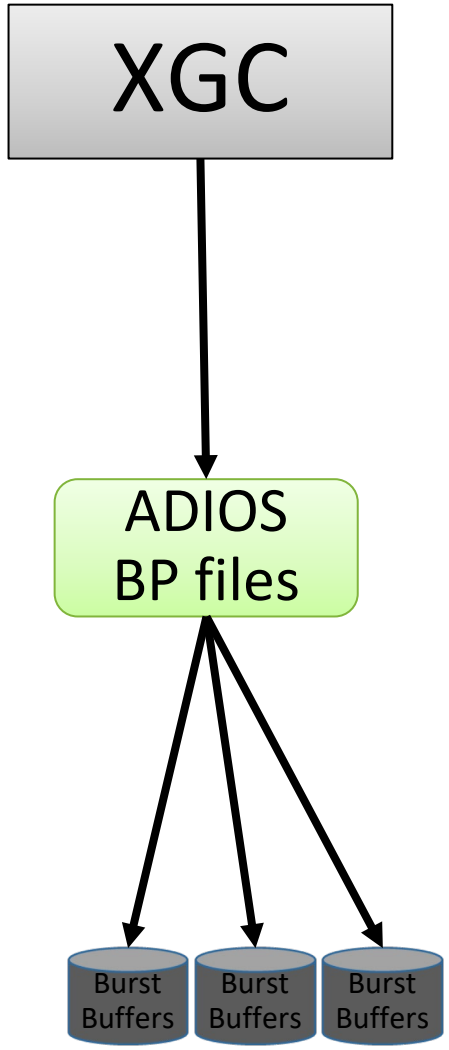
ADIOS APIs for self describing data output for C/R to NVRAM

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









Analysis/visualization data

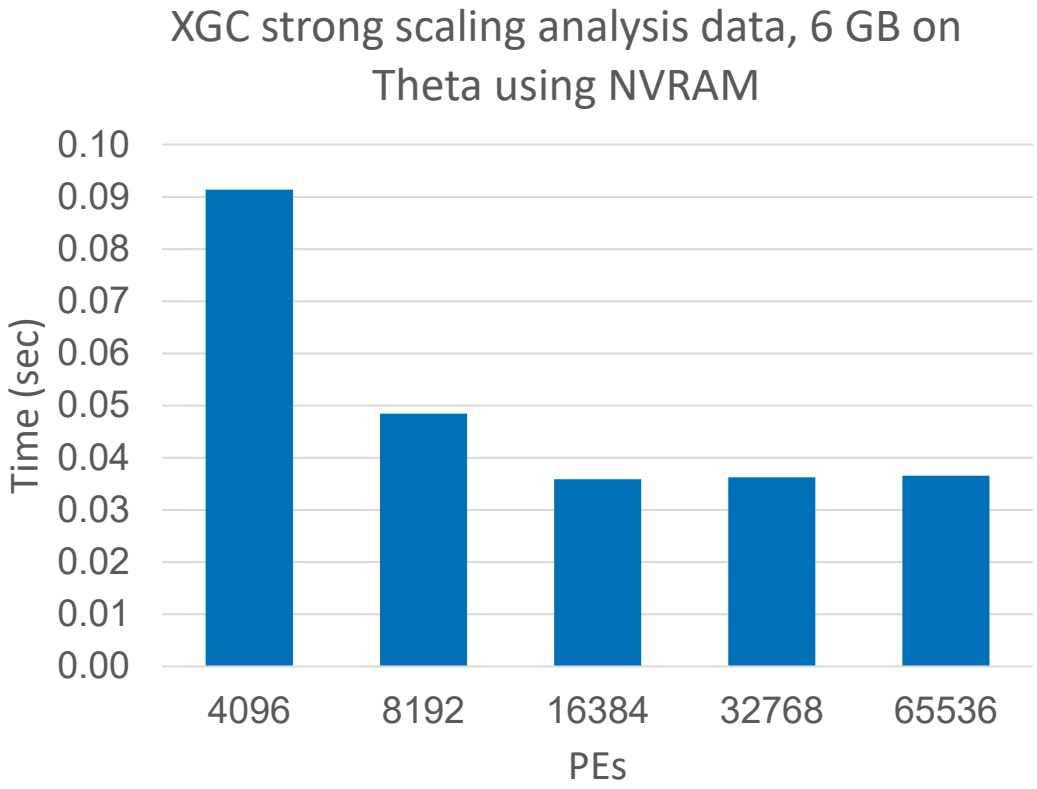
```
adios2::IO io = adios.DeclareIO("Analysis_Data");

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
- Disk I/O:
    - Put only flushes to disk if the buffer is full, otherwise flushed in EndStep()
    - No difference in performance between using sync and deferred Put

ADIOS engines – change from BP4 to HDF5

1. <io name="SimulationOutput">
2. <engine type="**BP4**"/>
3. </io>

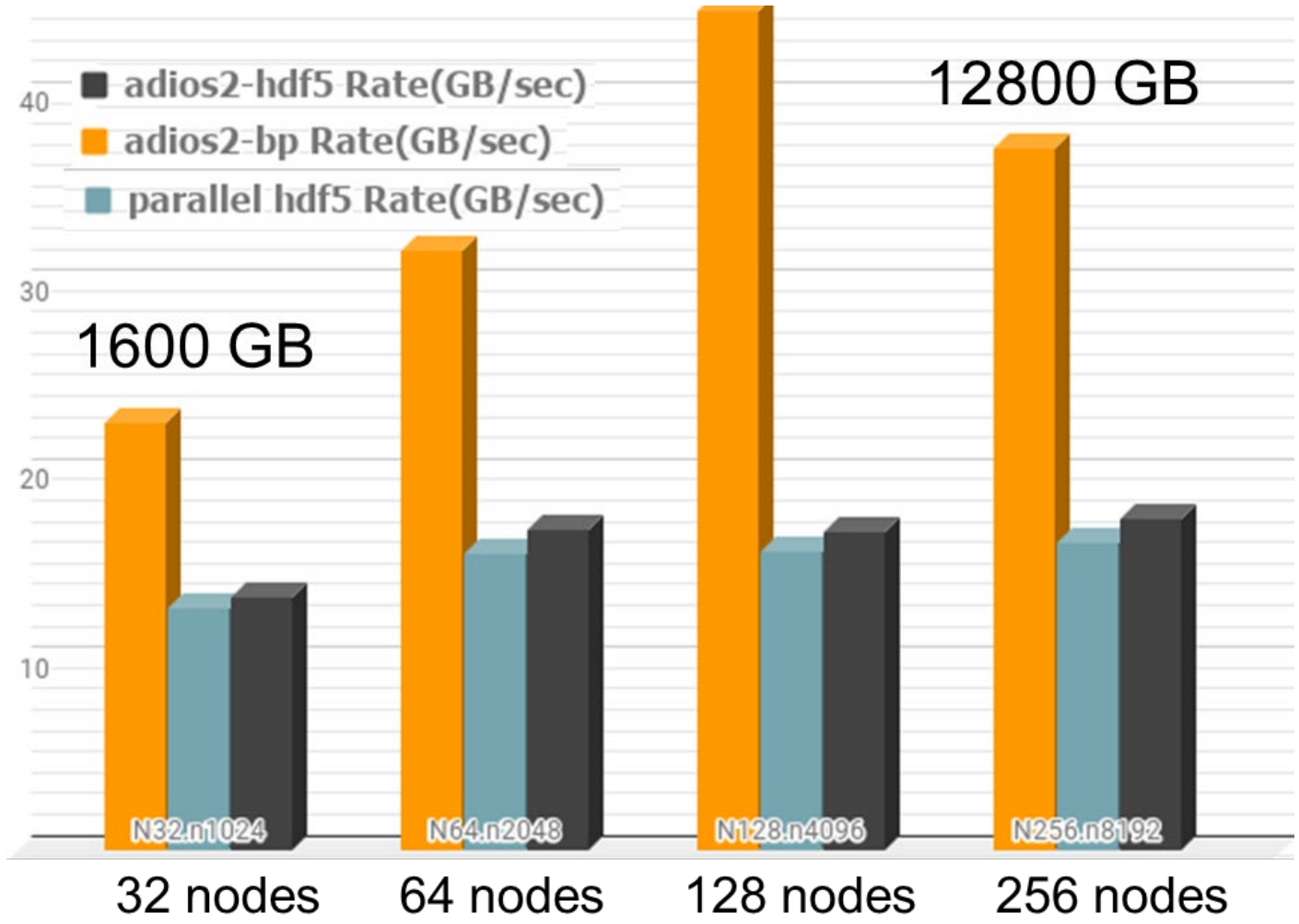
Change Engine name

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 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

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

```
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

Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

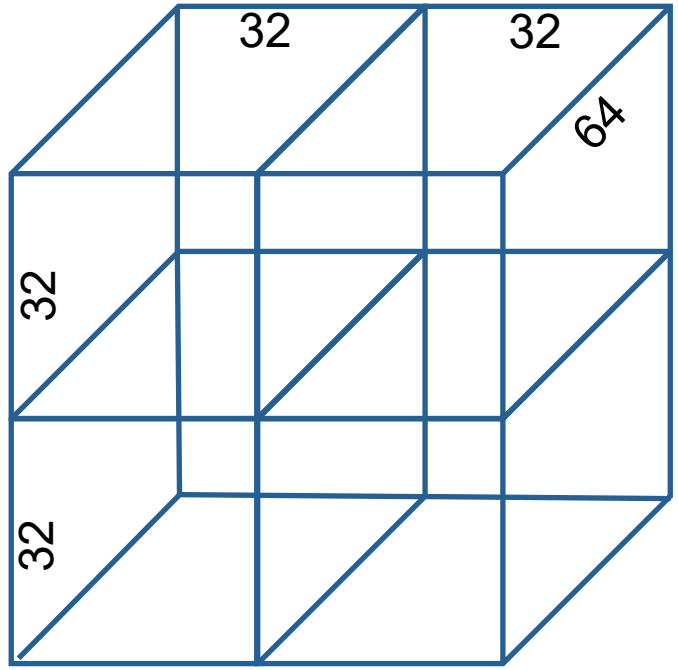
- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

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

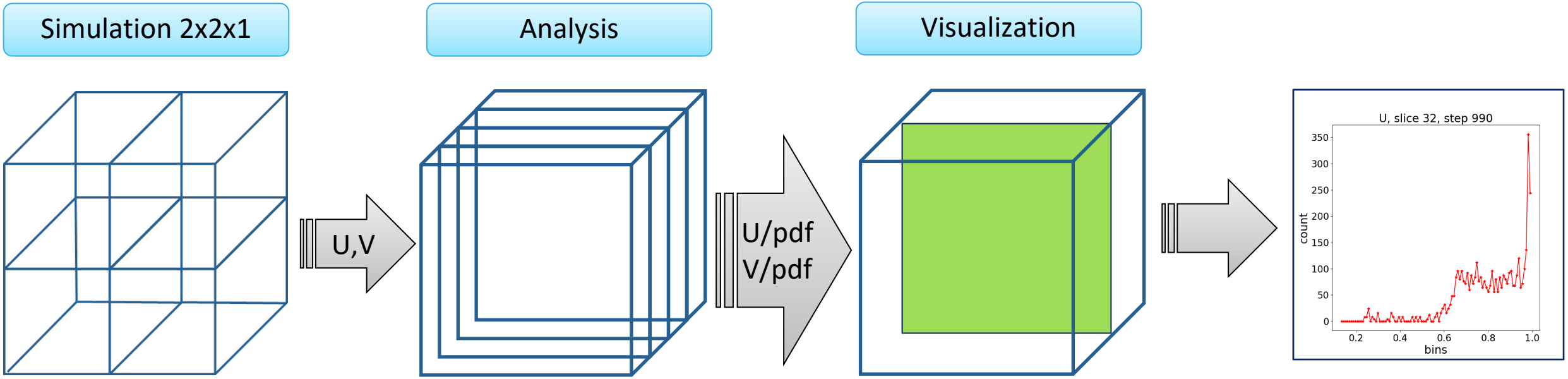
Gray-Scott Example

- 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

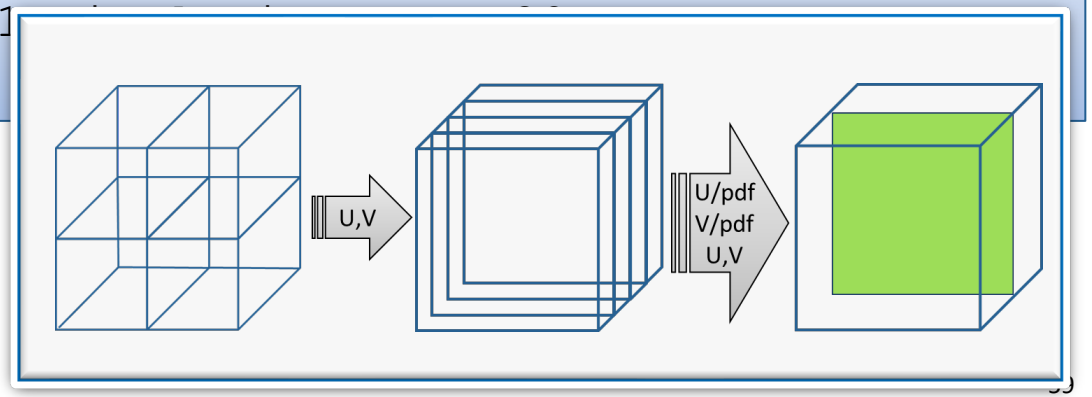


Goal by the end of the day: Running the example in situ

```
$ 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
...
```



Compile ADIOS codes

- CMake
  - Use MPI_C and ADIOS 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::adios2   MPI::MPI_C)
```

  - Configure application by adding ADIOS installation to search path

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

Compile ADIOS codes

- Makefile
  - Add ADIOS and HDF5 library paths to LD_LIBRARY_PATH
  - Use adios2_config tool to get compile and link options

```
ADIOS_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
	${FC}   -g -c -o io_adios2.o -I${ADIOS_DIR} io_adios2.F90
	${FC}   -g -o heatSimulation heatSimulation heat_vars.o io_adios2.o  ${ADIOS_FLIB}
```

Configure and build the code

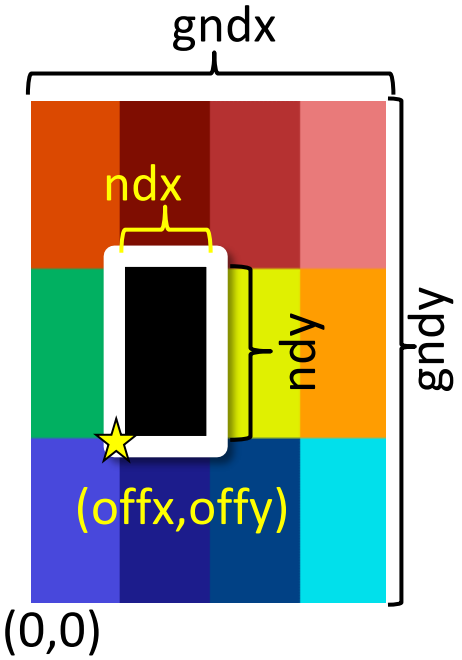
```
$ cd ~/ADIOS2-Examples/
$ mkdir -o build-cmake
$ cd build-cmake
$ cmake \
  -DCMAKE_PREFIX_PATH=/home/adios/Tutorial \
  -DADIOS2_DIR=/opt/adios2   \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo   \
  ..
$ make -j 4
$ make install
$ export PATH=$PATH:/home/adios/Tutorial/bin
$ cd /home/adios/Tutorial/share/adios2-examples/gray-scott
```

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:              BP4
=========================================
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/003_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)

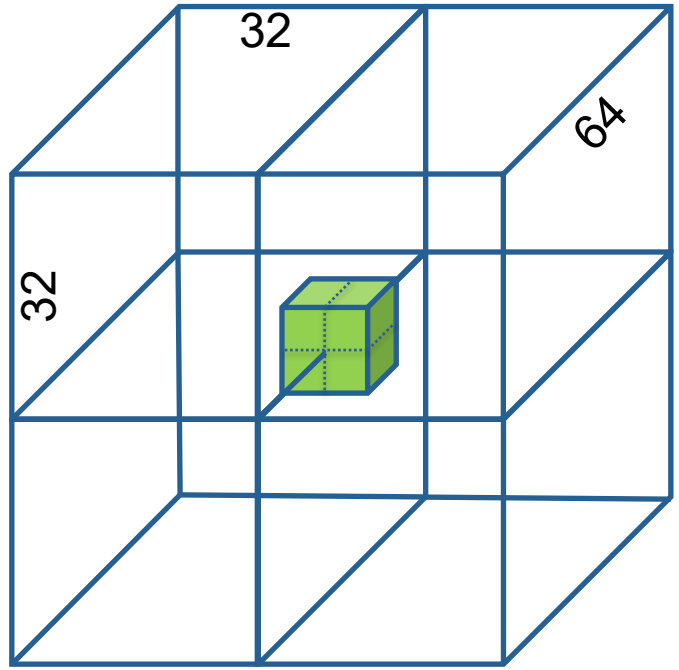
```
$ bpls -D gs.bp U
 double   U      100*{64, 64, 64} = 0.0907898 / 1
        step  0:
          block 0: [ 0:63,  0:31,  0:31] = 0.104002 / 1
          block 1: [ 0:63, 32:63,  0:31] = 0.104002 / 1
          block 2: [ 0:63,  0:31, 32:63] = 0.104002 / 1
          block 3: [ 0:63, 32:63, 32:63] = 0.104002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



Pretty print with bpls

```
$ bpls gs.bp -d U -n 64 -f "%5.2f" -s "-1,0,0,0" | less -S
 double   U      100*{64, 64, 64}
    slice (99:99, 0:63, 0:63, 0:63)
    (99, 0, 0, 0)     0.99  0.99  0.99  0.98  0.97  0.97  0.97  0.97  0.97  0.97
    (99, 0, 1, 0)     0.99  0.99  0.98  0.97  0.96  0.96  0.95  0.95  0.95  0.96
    (99, 0, 2, 0)     0.99  0.98  0.97  0.95  0.94  0.93  0.92  0.92  0.93  0.94
    (99, 0, 3, 0)     0.98  0.97  0.95  0.93  0.90  0.89  0.88  0.89  0.90  0.91
    (99, 0, 4, 0)     0.97  0.96  0.94  0.90  0.87  0.85  0.85  0.85  0.87  0.88
    (99, 0, 5, 0)     0.97  0.96  0.93  0.89  0.85  0.83  0.83  0.84  0.86  0.87
    (99, 0, 6, 0)     0.97  0.95  0.92  0.88  0.85  0.83  0.83  0.84  0.86  0.88
    (99, 0, 7, 0)     0.97  0.95  0.92  0.89  0.85  0.84  0.84  0.86  0.87  0.89
    (99, 0, 8, 0)     0.97  0.95  0.93  0.90  0.87  0.86  0.86  0.87  0.89  0.91
    (99, 0, 9, 0)     0.97  0.96  0.94  0.91  0.88  0.87  0.88  0.89  0.91  0.92
    (99, 0,10, 0)     0.97  0.96  0.95  0.92  0.90  0.89  0.89  0.90  0.92  0.93
    (99, 0,11, 0)     0.98  0.97  0.95  0.93  0.91  0.90  0.90  0.91  0.93  0.94
    (99, 0,12, 0)     0.98  0.97  0.96  0.94  0.93  0.91  0.91  0.92  0.93  0.94
    (99, 0,13, 0)     0.98  0.98  0.97  0.95  0.94  0.92  0.92  0.93  0.94  0.95
    (99, 0,14, 0)     0.99  0.98  0.97  0.96  0.94  0.93  0.93  0.93  0.94  0.95
    (99, 0,15, 0)     0.99  0.98  0.98  0.97  0.95  0.94  0.94  0.94  0.94  0.95.
⋮
```

The ADIOS XML configuration file

- Describe runtime parameters for each IO grouping
  - select the Engine for writing
    - **BP4**, **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 version="1.0"?>
<adios-config>

 <!--=====================================
       Configuration for for Gray-Scott and GS Plot
     ======================================-->
  <io name="SimulationOutput">
    <engine type="BP4">
      <parameter key="OpenTimeoutSecs" value="10.0"/>
      <parameter key="NumAggregators" value="2"/>
    </engine>
  </io>
```

Engine types

BP4

HDF5

FileStream

SST

SSC

DataMan

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

  <io name="PDFAnalysisOutput">
    <engine type="BP4">
    </engine>
  </io>

</adios-config>
```

Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

HDF5 engine to read/write HDF5 files

- Let's write output in HDF5 format without changing the source code
- Edit adios2.xml and
- set the SimulationOutput engine to **HDF5**
- run the same example again

The runtime config file: adios2.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
BP4
HDF5
FileStream
SST
SSC
DataMan

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
```

Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

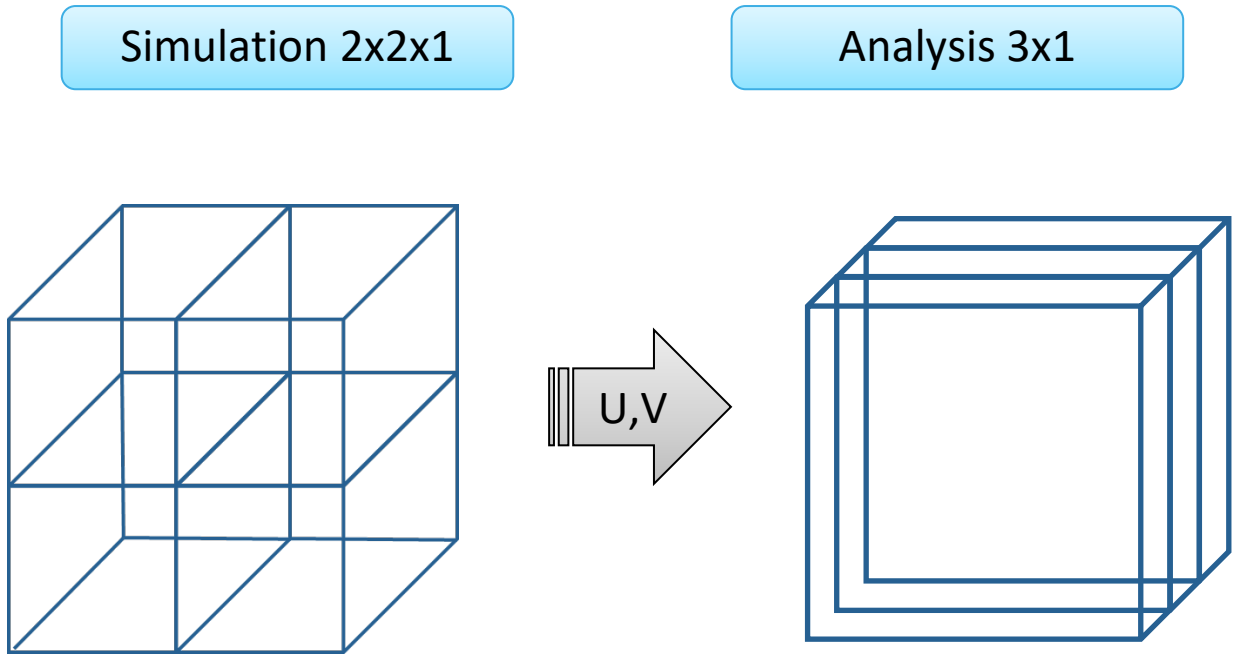
- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

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

Analysis

- Read with a different decomposition (1D)
  - Read/Write from 3 cores, arranged in a 3 x 1 arrangement.



Compile and run the reader

```
# make sure in adios2.xml, SimulationOutput's engine is set to BP4
$ mpirun -n 3 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  BP4
PDF analysis writes using engine type:                 BP4
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:
```

HDF5 engine to read/write HDF5 files using ADIOS

- Let's read back from the ADIOS/HDF5 output without changing the source code
  - Edit adios2.xml and
  - set the SimulationOutput engine to **HDF5**
  - also set PDFAnalysisOutput engine to **HDF5**
  - run the same example again

Compile and run the reader

make sure in adios2.xml, both SimulationOutput and PDFAnalysis engines are set to HDF5

$ mpirun -n 3 adios2-pdf-calc gs.**h5** pdf.**h5** 100

```
PDF analysis reads from Simulation using engine type:  HDF5
PDF analysis writes using engine type:                 HDF5
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.**h5**

```
  double   U/bins  100*{100} = 0 / 0
  double   U/pdf   100*{64, 100} = 0 / 0
  double   V/bins  100*{100} = 0 / 0
  double   V/pdf   100*{64, 100} = 0 / 0
  int32_t  step    100*scalar = 0 / 0
```

HDF5 format does not have Min/Max info

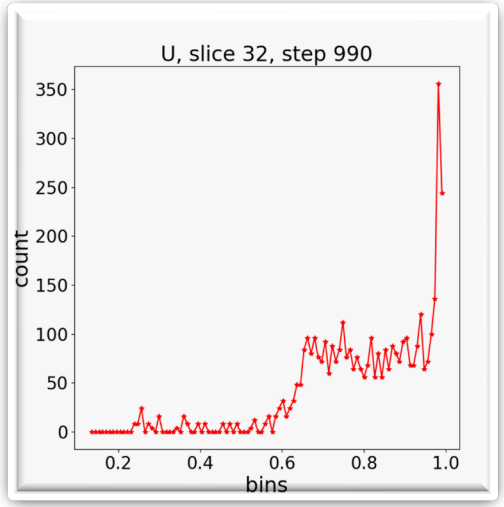
$ bpls -l pdf.**h5** -D U/pdf

```
  double   U/pdf   100*{64, 100} = 0 / 0
        step  0:
          block 0: [ 0:63,  0:99] = 0 / 0
        step  1:
          block 0: [ 0:63,  0:99] = 0 / 0
```

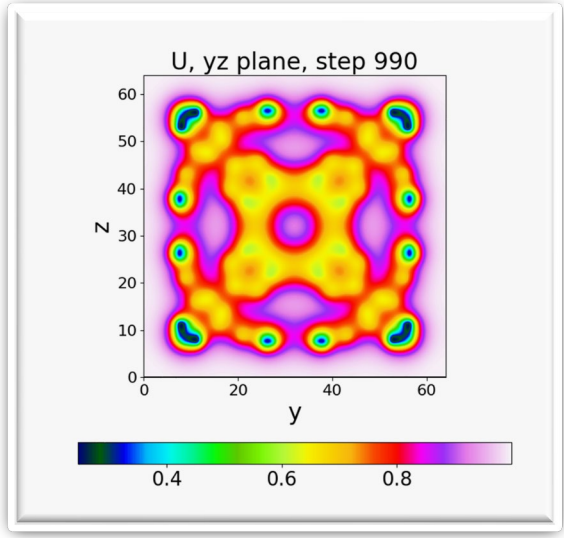
HDF5 format: array seen as one big block

Python scripts to read in the data and plot with Matplotlib

- pdfplot.py
  - Read the pdf calculation output
  - Plot the middle slice PDF



- gsplot.py
  - Read the ADIOS2 pdf calculation output
  - Plot a 2D slice of variable U
    - or -v <varname>



The runtime config file: adios2.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
BP4
HDF5
FileStream
SST
SSC
DataMan

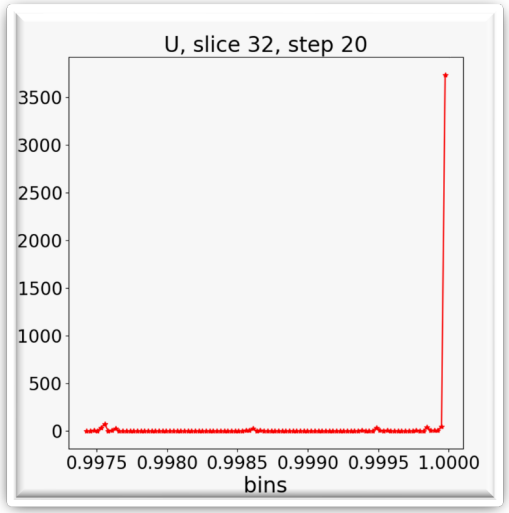
```
  <!--================================
       Configuration for PDF calc and PDF Plot
     ================================-->

  <io name="PDFAnalysisOutput">
    <engine type="HDF5">
    </engine>
  </io>

</adios-config>
```

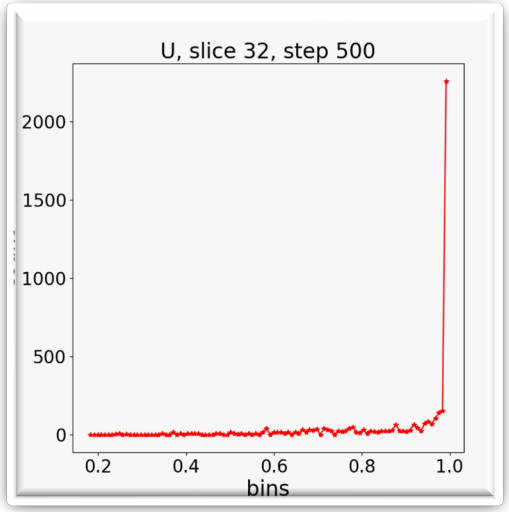
Run the code again if you removed gs.h5

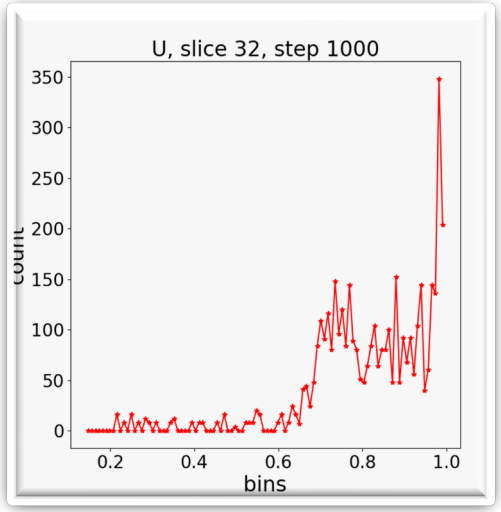
```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
$ mpirun -n 4 adios2-gray-scott settings-files.json
Simulation writes data using engine type:              HDF5
=========================================
grid:               64x64x64
steps:              1000
plotgap:            10
F:                  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
```

Run the plotting script with file I/O

```
$ python3 pdfplot.py -i pdf.h5 -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
$ gpicview &
```

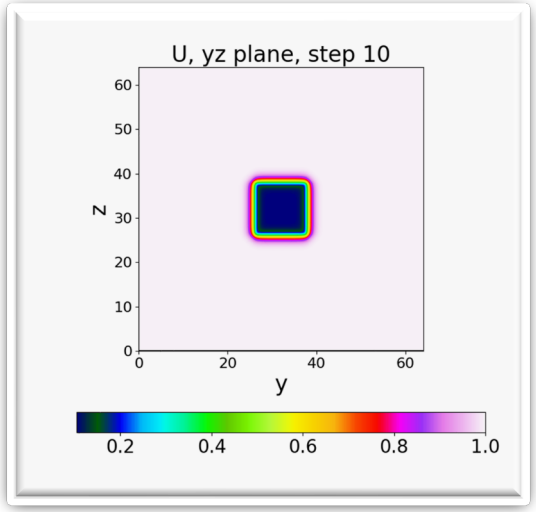


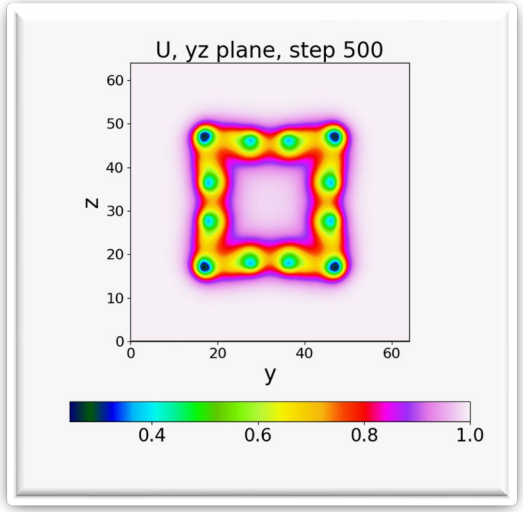


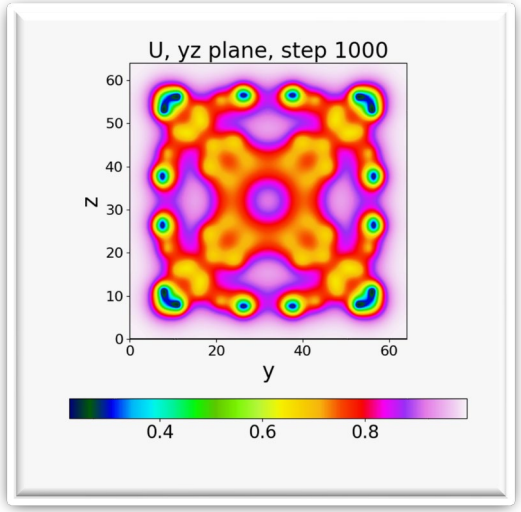


Run the plotting script with file I/O

```
$ python3 gsplot.py -i gs.h5 -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
$ gpicview &
```







Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

ADIOS Scaling for large parallel file systems

- There are **only** two 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
  - 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

```
<io name="SimulationOutput">
  <engine type="BP4">
    <parameter key="NumAggregators" value="2048"/>
  </engine>
</io>
```

Latest 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 starting point as apps usually have one MPI task per GPU
  - However, try to keep the total number of subfiles below 4000

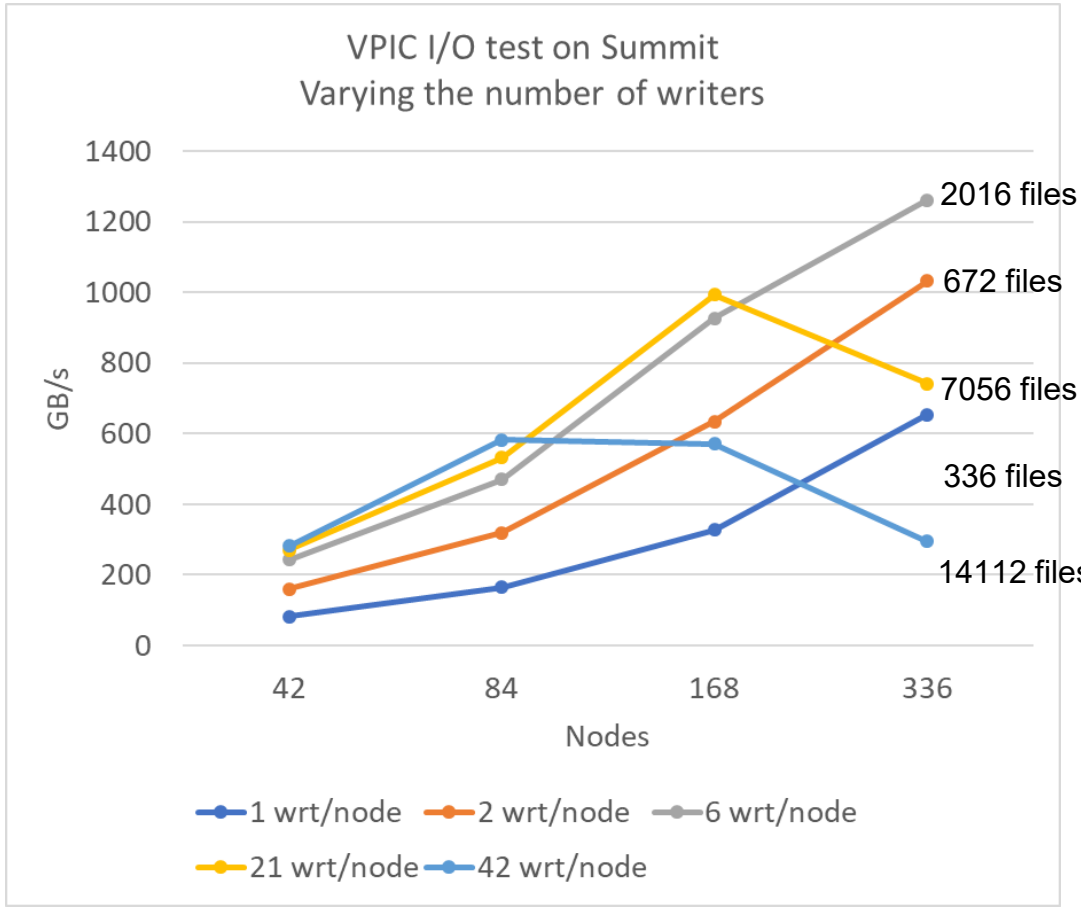
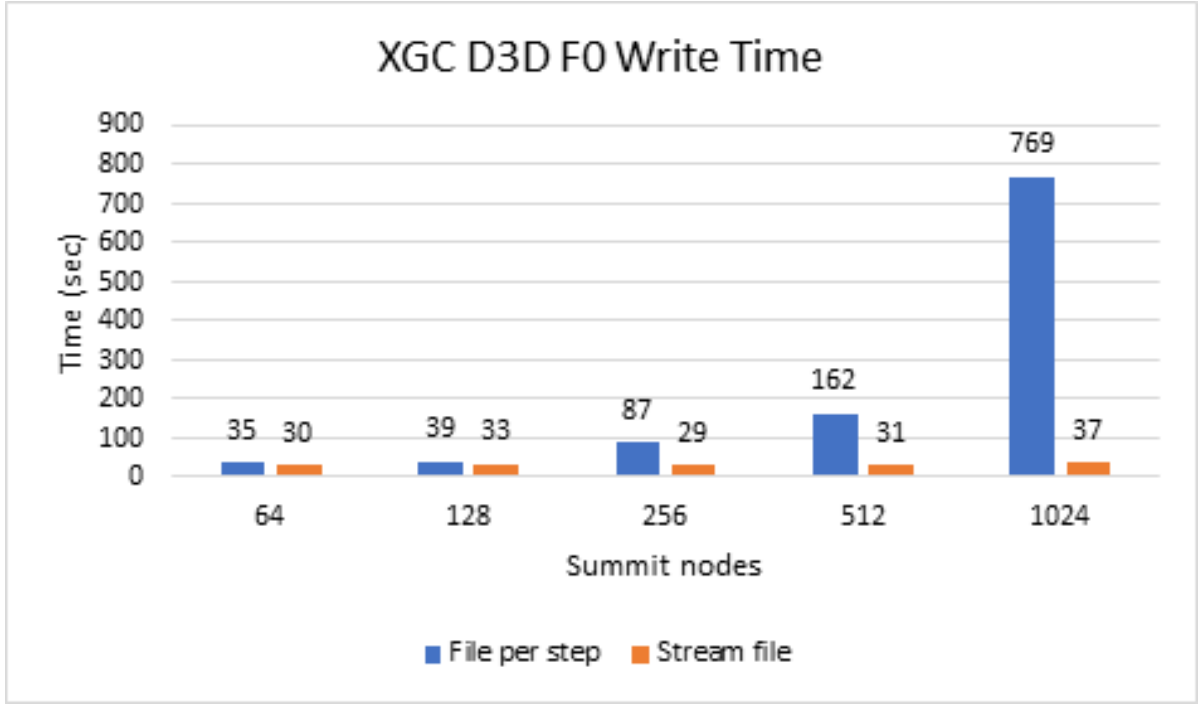


chart courtesy of Junmin Gu, LBNL

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

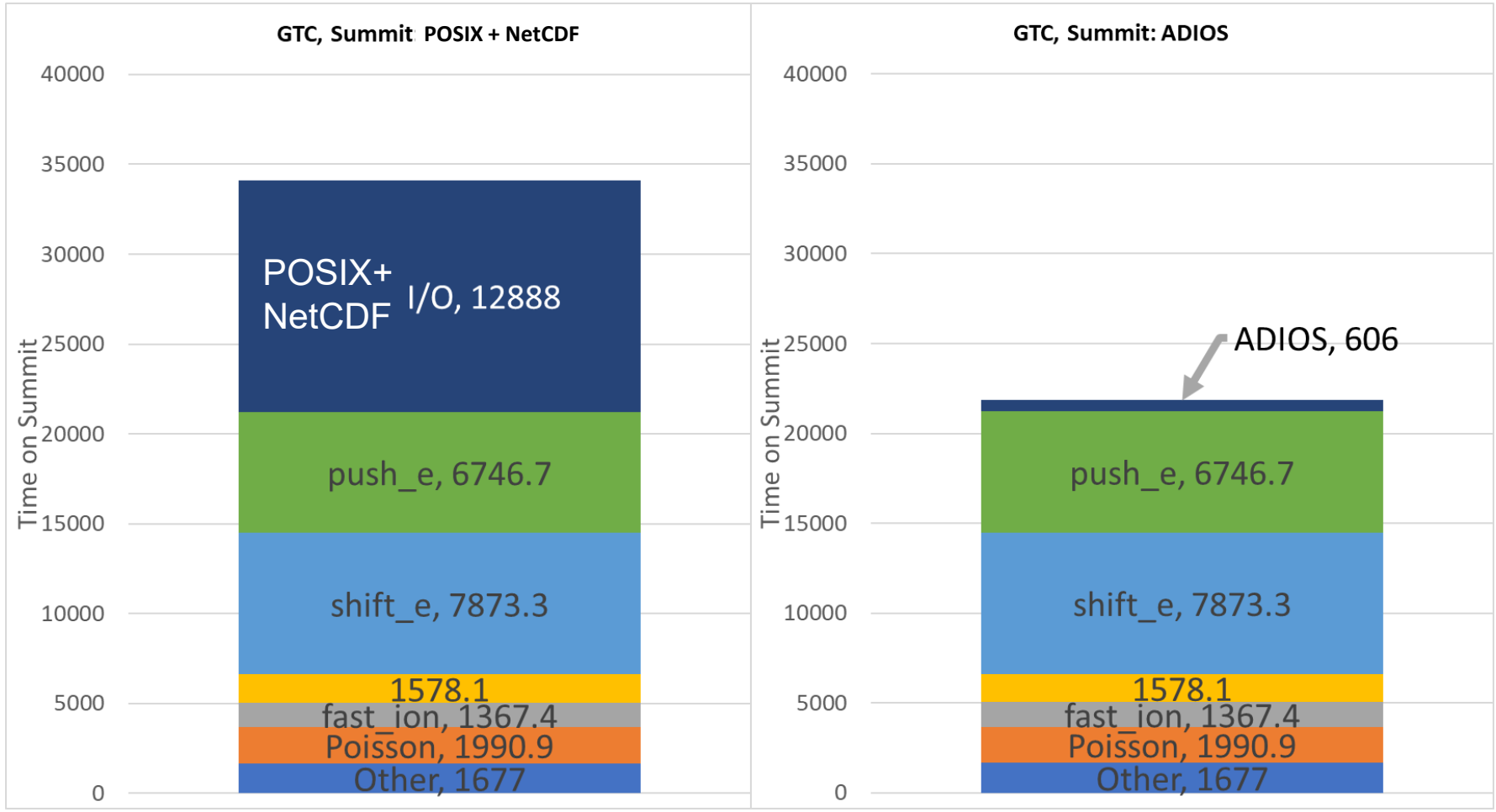
10k steps, 512 nodes (3072 ranks)

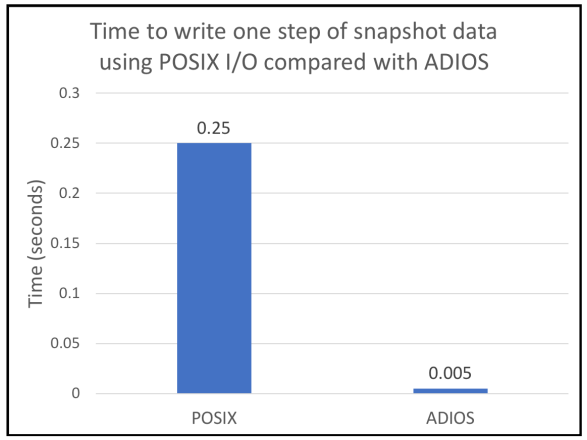
Data Category	Data Subcategory	Number of files in the original code	Number of files in the ADIOS container (1 writer per node)
diagnostics	equilibrium	1	1
diagnostics	data1d	1	1
diagnostics	history	1	1
snapshot	snapshot	**10,000** (1 file per step)	**1**
field data	phi3d	**32,000** (1 file every 10 steps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

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





* average cost when measuring 10 000 steps

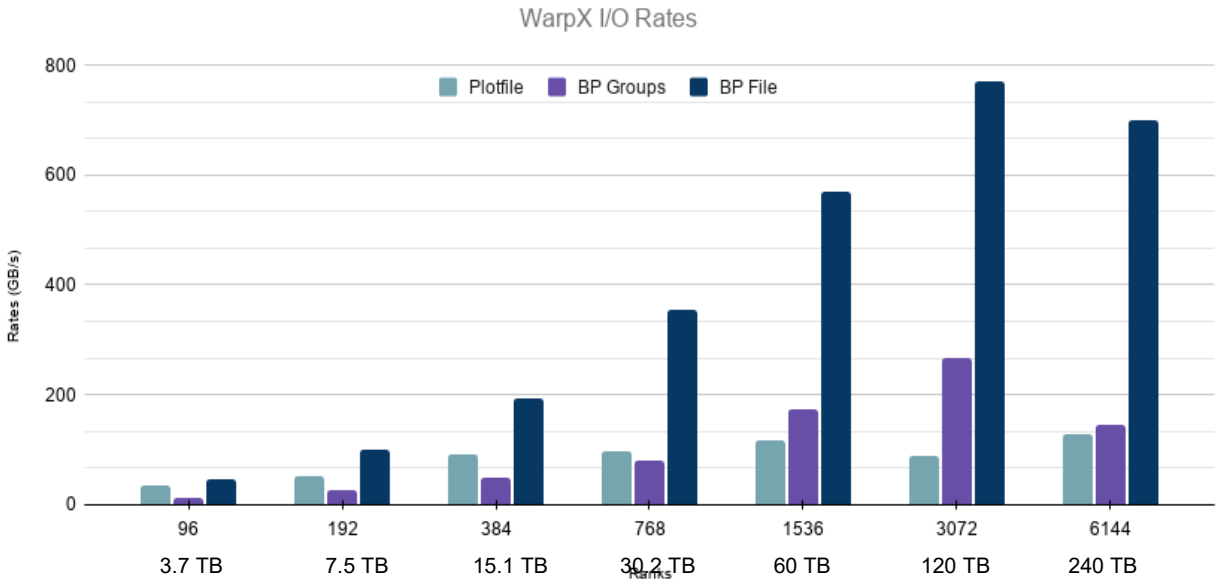
slide courtesy of Kshitij Mehta, ORNL

WarpX example: appending + aggregation

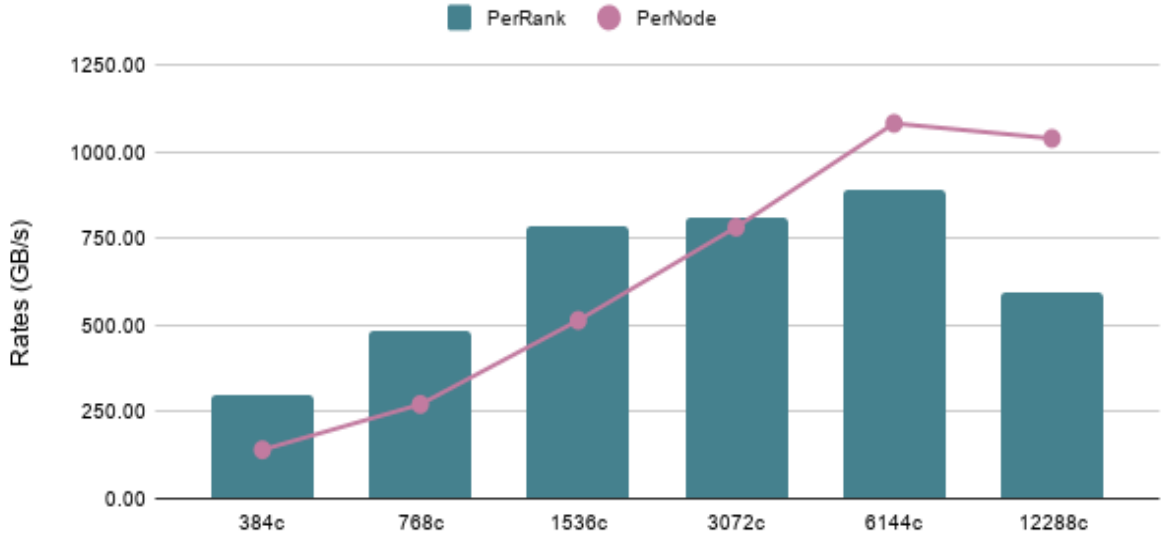
PI: Jean-Luc Vay, LBNL

- 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)

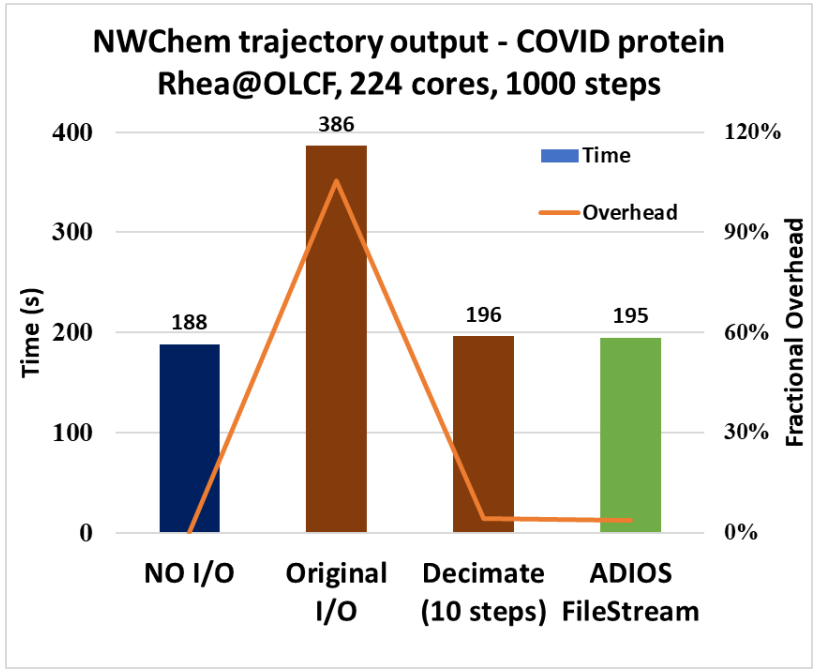


240 TB on 1024 nodes (6144 cores)

charts courtesy of Junmin Gu, LBNL

Single process I/O examples: NWChem, LAMMPS

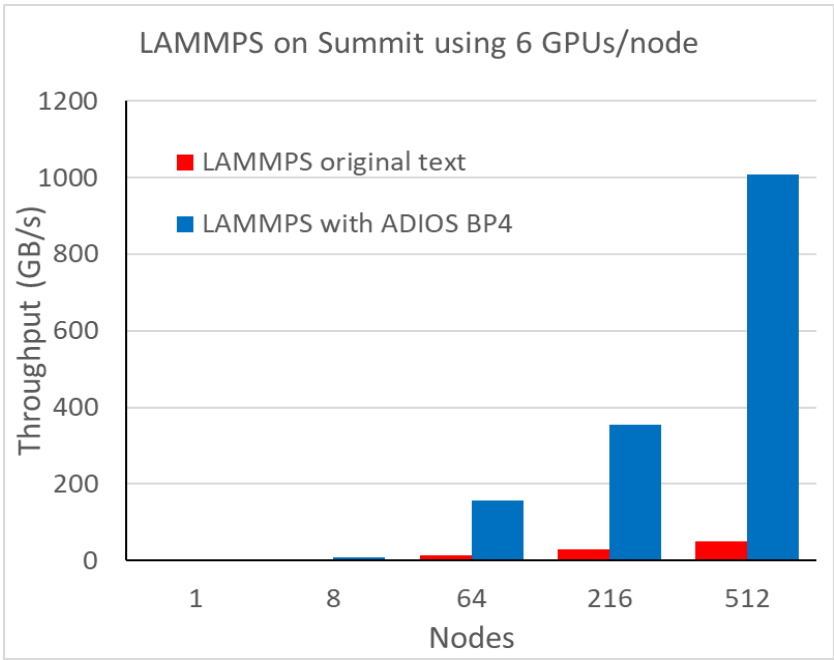
- Moves data between processes as part of preparation for I/O
  - "I am doing POSIX/Fortran I/O on rank 0"
  - While gathering data, no one is writing
- Single file output – not utilizing available bandwidth



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

chart courtesy of Norbert Podhorszki, ORNL

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

chart courtesy of Lipeng Wan, ORNL

Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

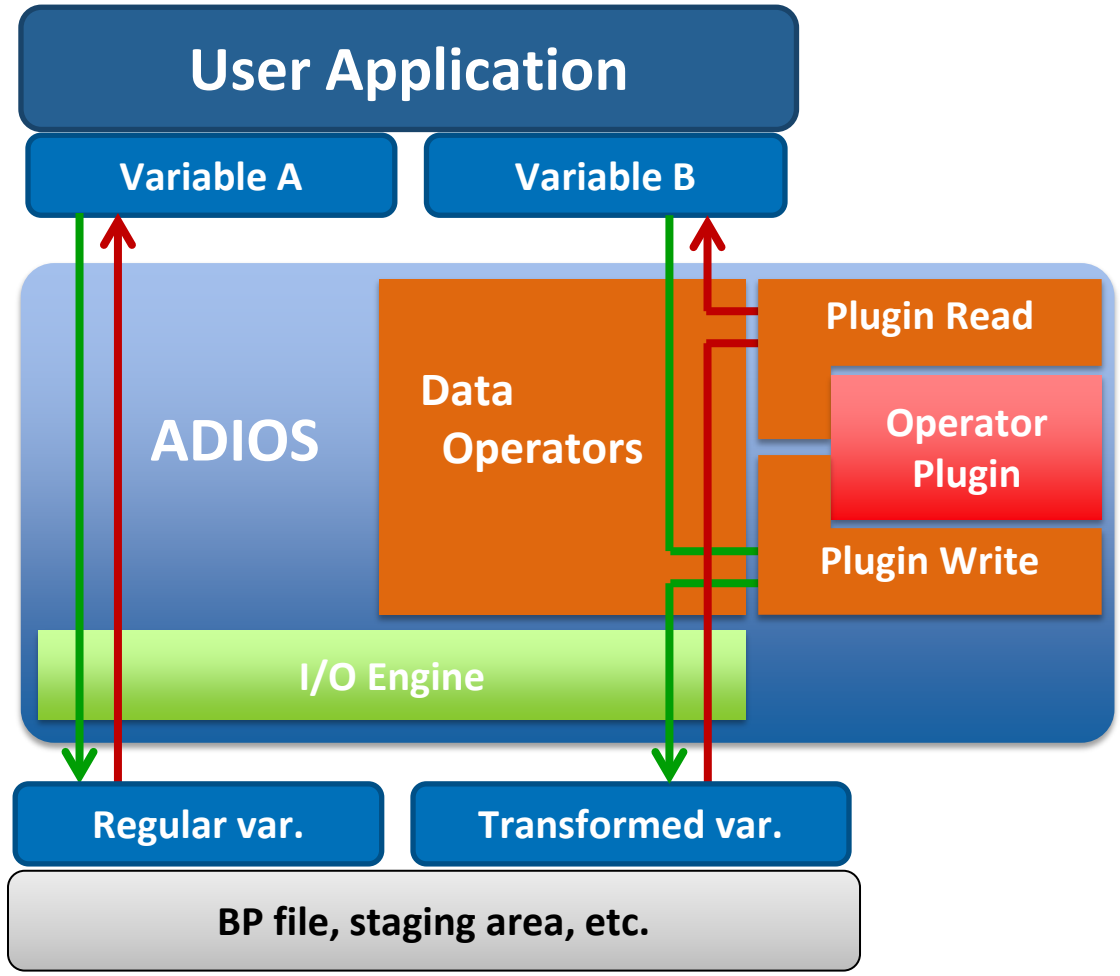
Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

ADIOS Operators

- ADIOS allows users to transparently apply operators to data, using code that looks like its still using the original untransformed data
- Can swap operators in/out at runtime (vs. compile time)
- Plugin based, enabling easy expansion
- Focus on compression today



Operator in ADIOS

- An entity that works on Variable data of a writer process to transform the data before/during output
  - Lossy compression: **ZFP**, **SZ**, **MGARD**
  - Lossless compression: BLOSC, BZIP2
  - Type conversion: e.g. double to float decimation
- One can apply different Operators to the Variables in an IO group

Operators in source code

- Operator ADIOS::DefineOperator(name, type)
- Variable::AddOperation(Operator, {Parameter[,…]})

```
adios2::IO io = adios.DeclareIO("TestIO");
auto varF = io.DefineVariable<float>("r32", shape, start, count, adios2::ConstantDims);
auto varD= io.DefineVariable<double>("r64", shape, start, count, adios2::ConstantDims);

adios2::Operator zfpOp = adios.DefineOperator("zfpCompressor", "zfp");

varF.AddOperation(zfpOp, {{"accuracy", "0.01"}});
varD.AddOperation(zfpOp, {{"accuracy", std::to_string(10 * accuracy)}});

adios2::Engine engine = io.Open(fname, adios2::Mode::Write);
```

Operators in the ADIOS XML configuration file

- Describe **runtime** parameters for each IO grouping
  - Select the Engine for writing
    - BP4, SST, InSituMPI, DataMan, SSC engines support compression
    - HDF5 engine does not support compression
  - **Define an Operator**
  - **Select an Operation for Variables (operator + parameters)**
    - "zfp", "mgard", "sz" – all support "accuracy" parameter
- See ~/Tutorial/share/adios2-examples/gray-scott/adios2.xml

Simple operator definition in XML: Operation only

```
<io name="PDFAnalysisOutput">
    <engine type="BP4">
    </engine>
     <variable name="U">
        <operation type="sz">
                <parameter key="accuracy" value="0.01"/>
        </operation>
    </variable>
</io>
```

See Tutorial/gray-scott/adios2.xml

Task

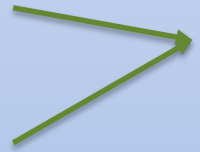
- Using the BP4 file output engine
- Run the pdf-calc example
  - with dumping the input data
  - with compressing U and V
- With SZ
  - with different accuracy levels (0.01, 0.0001, 0.000001)
- Compare the size of gs.bp and pdf.bp
- Run
  `python3 gsplot.py -i <file> -o <picname>`
  to plot data and `gpicview` to look at them

Rerun the gray-scott simulation again if you removed gs.bp

```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
$ mpirun -n 4 adios2-gray-scott settings-files.json
Simulation writes data using engine type:              BP4
=========================================
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
```

Run the PDF calc with extra parameter to save data

```
$ mpirun -n 3 adios2-pdf-calc gs.bp pdf-sz-0.0001.bp 100 YES
PDF analysis reads from Simulation using engine type:  BP4
PDF analysis writes using engine type:                 BP4
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
...
$ du -sh  *.bp
401M    gs.bp
25M     pdf-sz-0.001.bp
$ bpls -l  gs.bp
  double   U     100*{64, 64, 64} = 0.0907832 / 1
  double   V     100*{64, 64, 64} = 0 / 0.674825
  int32_t  step  100*scalar = 10 / 1000
$ bpls -l  pdf-sz-0.0001.bp
  double   U       100*{64, 64, 64} = 0.0907832 / 1
  double   U/bins  100*{100} = 0.0908349 / 1
  double   U/pdf   100*{64, 100} = 0 / 4096
  double   V       100*{64, 64, 64} = 0 / 0.674825
  double   V/bins  100*{100} = 0 / 0.668077
  double   V/pdf   100*{64, 100} = 0 / 4096
  int32_t  step    100*scalar = 10 / 1000
```



U and V from gray-scott are included in pdf-sz.bp but they are compressed

Dump the data

```
$ bpls -l gs.bp -d U -s "99,20,20,20" -c "1,4,2,3" -n 6 -f "%12.9f"
double   U     100*{64, 64, 64} = 0.0907832 / 1
    slice (99:99, 20:23, 20:21, 20:22)
    (99,20,20,20)     0.799132816  0.794047216  0.774524644  0.794044013  0.809334311  0.805353502
    (99,21,20,20)     0.794048720  0.809339614  0.805357399  0.809337339  0.830566649  0.831157641
    (99,22,20,20)     0.774524443  0.805355761  0.811208162  0.805354275  0.831156412  0.835480782
    (99,23,20,20)     0.762227773  0.795309063  0.801802280  0.795309102  0.821057779  0.826293178
```

```
$ bpls -l pdf-sz-0.0001.bp -d U -s "99,20,20,20" -c "1,4,2,3" -n 6 -f "%12.9f"
double   U       100*{64, 64, 64} = 0.0907832 / 1
    slice (99:99, 20:23, 20:21, 20:22)
    (99,20,20,20)     0.799861830  0.793861830  0.773547600  0.793861830  0.809861830  0.805547600
    (99,21,20,20)     0.794870268  0.808870268  0.804870268  0.808870268  0.830870268  0.830870268
    (99,22,20,20)     0.775469240  0.805469240  0.811469240  0.805469240  0.831469240  0.835469240
    (99,23,20,20)     0.761874701  0.795874701  0.801874701  0.795874701  0.821874701  0.825874701
```

Outline

Part I: Introduction to Parallel I/O and HPC file systems (0.5 hours) (Beginner/Intermediate)

- **Lecture:** Parallel I/O
- **Lecture:** HPC Storage Systems – GPFS, Lustre, Burst Buffers

Part II: Self-describing I/O using ADIOS (1 hour) (Beginner/Intermediate)

- **Lecture:** ADIOS framework, I/O abstraction, file format
- **Hands-on**: use a parallel MiniApp to write self-describing data
  - Use ADIOS write API to write data in parallel
  - Write HDF5 files using the ADIOS API
- **Hands-on**: Parallel data reading
  - ADIOS read API in Fortran90, C++, and Python
  - Read HDF5 files using the ADIOS API
- **Lecture**: How to scale ADIOS I/O

BREAK

Part III: Data Compression (0.5 hour) (Intermediate)

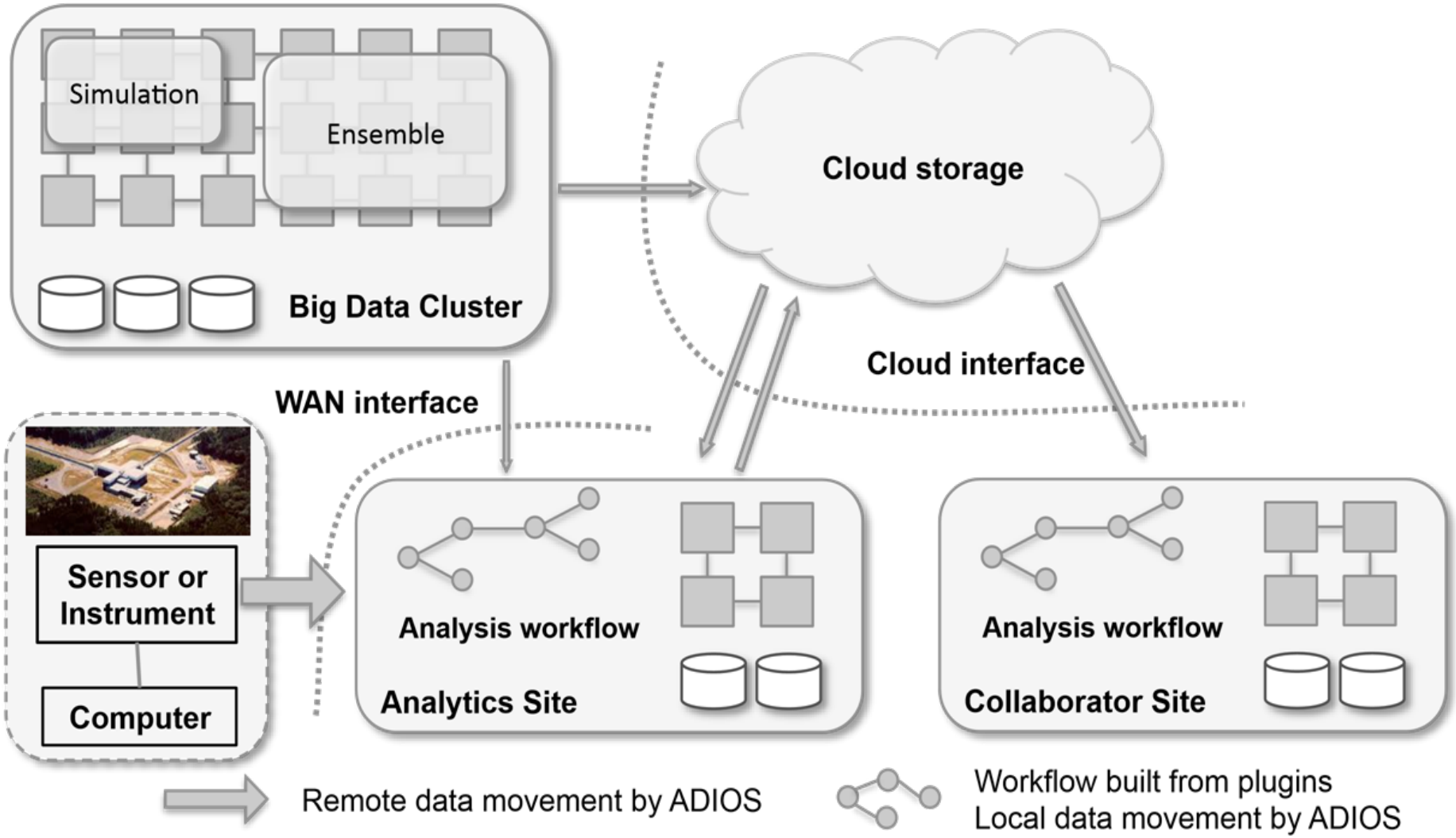
- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization with Visit and Paraview
- **Hands-on**: Staging and converting with adios_reorganize tool

Wrap-up

Vision: building scientific collaborative applications

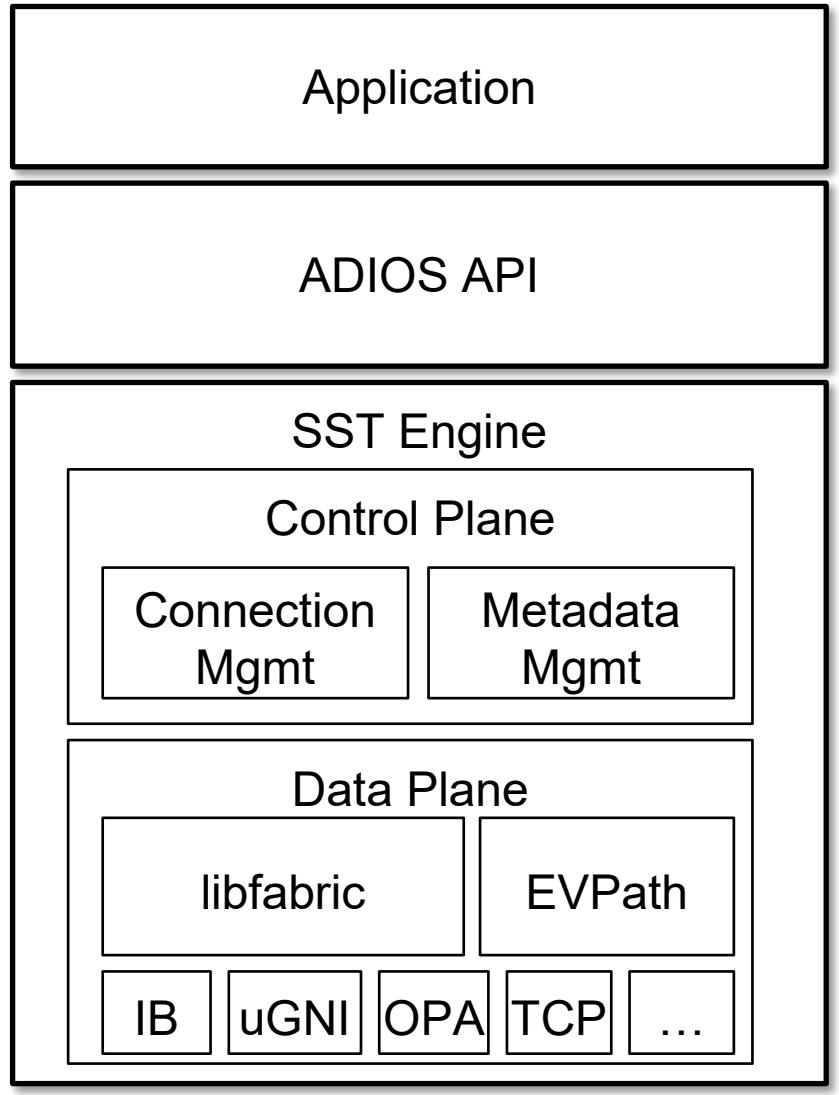


Data Staging in ADIOS

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

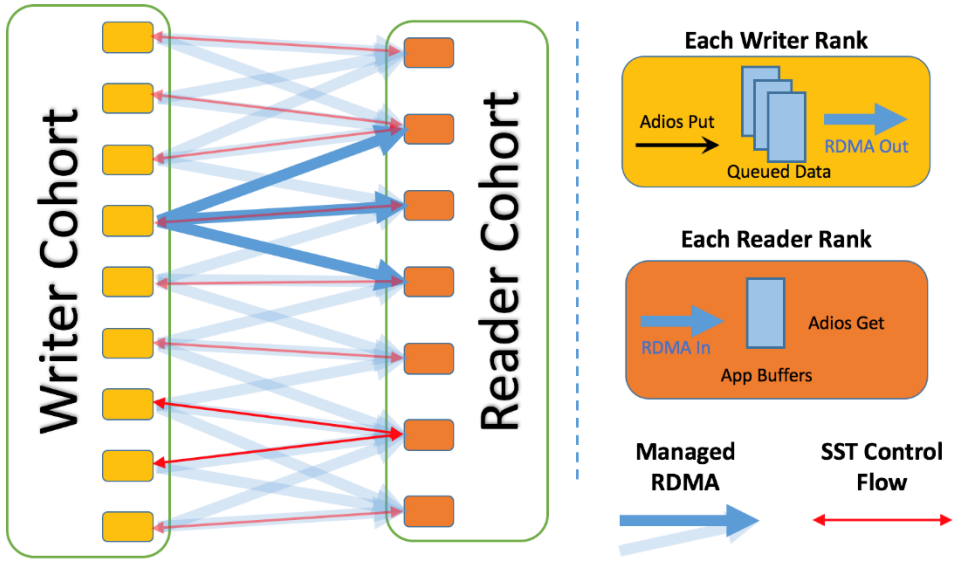
Sustainable Staging Transport (SST)

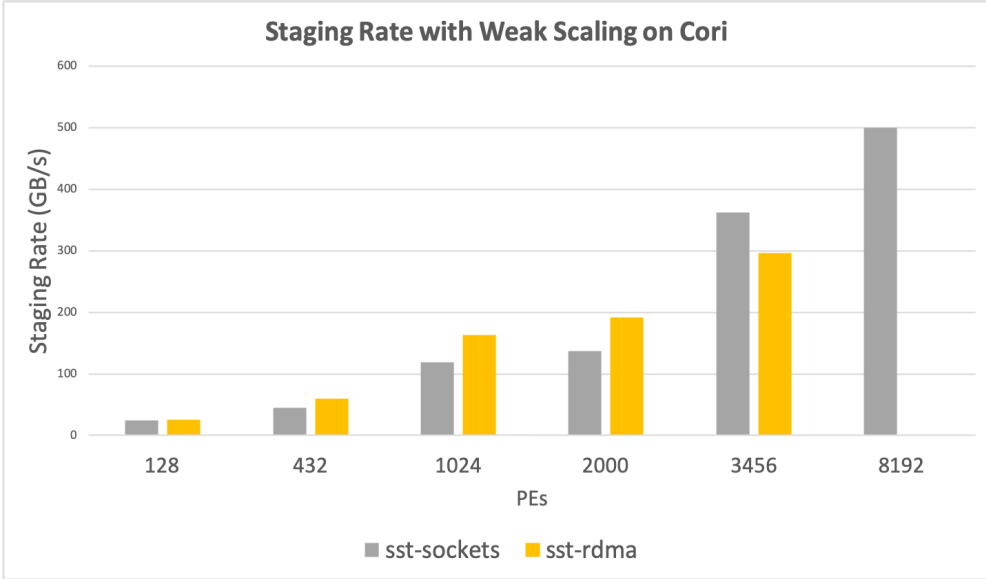
- Direct coupling between data producers and consumers for in-situ/in-transit processing
- Designed for portability and reliability.
- Control Plane
  - Manages meta-data and control using a message-oriented protocol
  - Inherits concepts from Flexpath, DIMES, uses EVPath
  - Allows for dynamic connections, multiple readers and complex flow control management
- Data Plane
  - Exchange data using RDMA
  - Responsible for resource management for data transfer
  - Uses libfabric for portable RDMA support
  - Threaded to overlap communication with computation and for asynchronous progress monitoring
  - Modular interface with the control plane supports alternative data plane implementations



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, and Gem three-way coupling.
  - Other ECP applications requiring strong coupling or always-on in-situ analysis/visualization

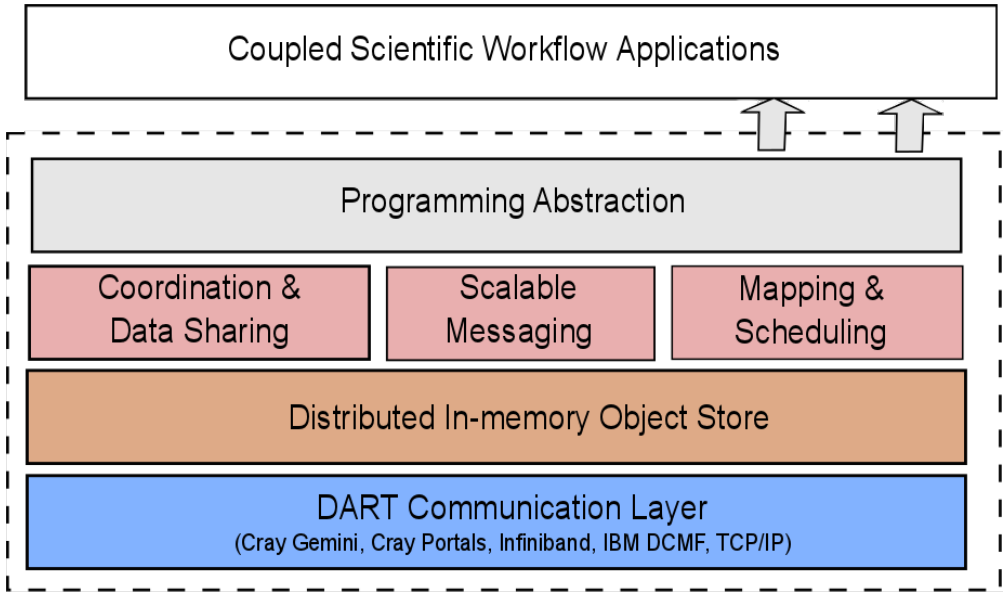
DataMan

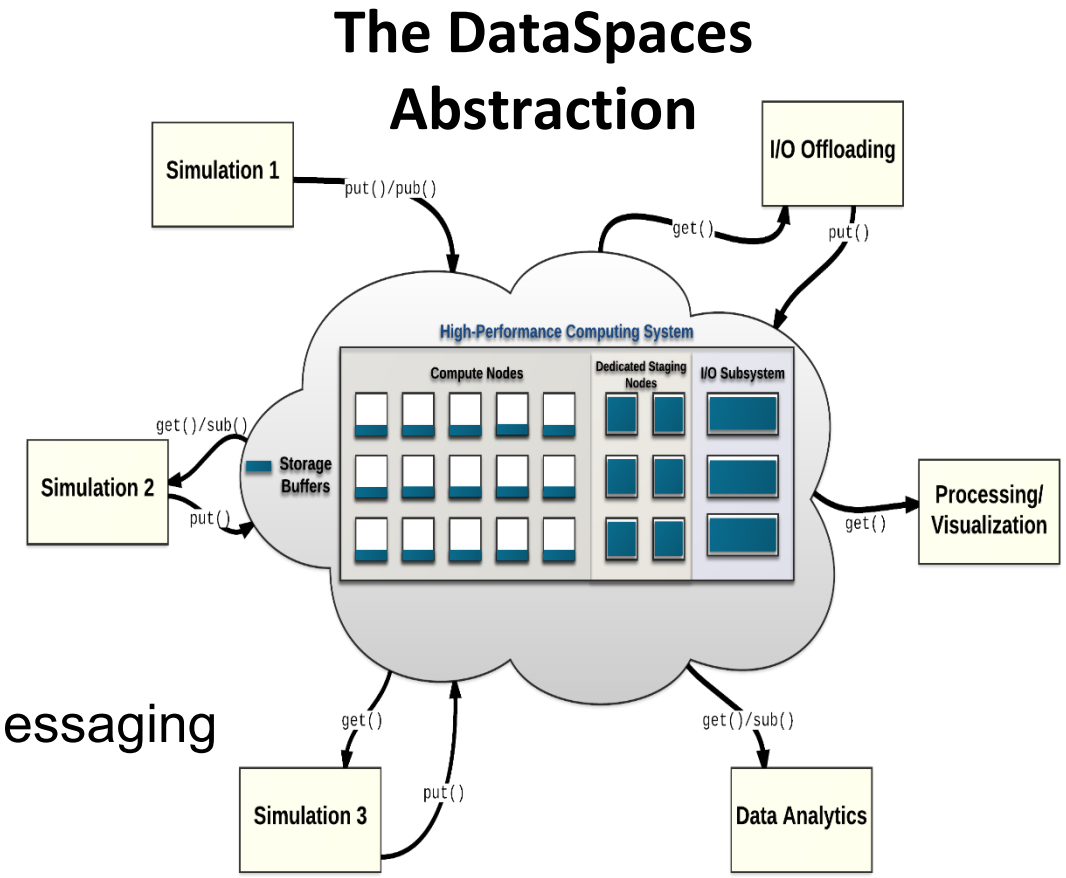
- An ADIOS 2 engine subroutine designed for wide area network data transfer, data staging, near-real-time data reduction and remote in-situ processing
- Designed with following principles:
  - Flexibility: allowing transport layer switching (ADIOS BP file, ZeroMQ, google-rpc etc.)
  - Versatility: supporting 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

DataMan: Supports three workflow modes

- Push
  - Sender driven workflow
  - Senders are configured to send data to pre-defined receivers.
- Query
  - Receiver driven workflow
  - Receivers are configured to query particular data (particular subsets of particular variables) from senders.
- Subscribe
  - Ad-hoc workflow
  - Free combination of senders (publishers) and receivers (subscribers).
  - Senders and receivers can be launched in any order, and they can be attached and detached freely without affecting other senders or receivers.

DataSpaces: Extreme Scale Data Staging Service





- Virtual shared-space programming abstraction
  - Simple API for coordination, interaction and messaging
- Distributed, associative, in-memory object store
  - Online data indexing, flexible querying
- Adaptive cross-layer runtime management
  - Hybrid in-situ/in-transit execution
- Efficient, high-throughput/low-latency asynchronous data transport

www.dataspaces.org

Inline engine: in-process in situ visualization

- An ADIOS 2 engine executing consumer inside producer's EndStep()

- 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

Application: Which staging method to use?

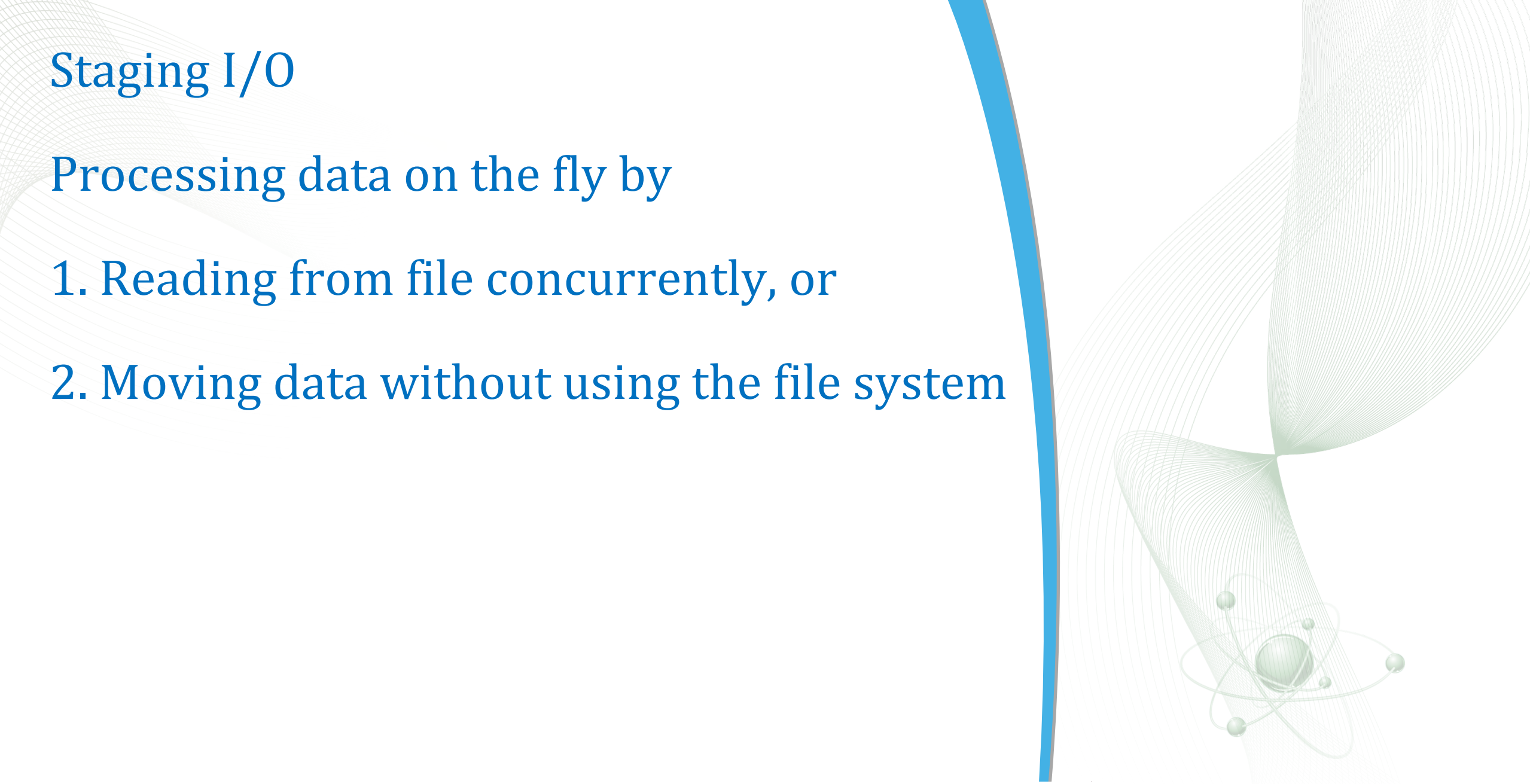
- System considerations
  - Staging between machines/over a WAN? **DataMan**
  - Co-located execution? **SST**
  - Availability of RDMA high-speed network? **SST** and **DataSpaces** optimized for this case.
- Coupling considerations
  - Highly synchronized writer/reader? **SSC**
  - Ensemble workflow / multiple independent writers / data lives independently of any given writer component? **DataSpaces**
  - 1 writer, many readers streaming? **SST** optimized for this use case.
  - 1 reader, many small producers (e.g. AI training)? **SST** and **DataSpaces**
- 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

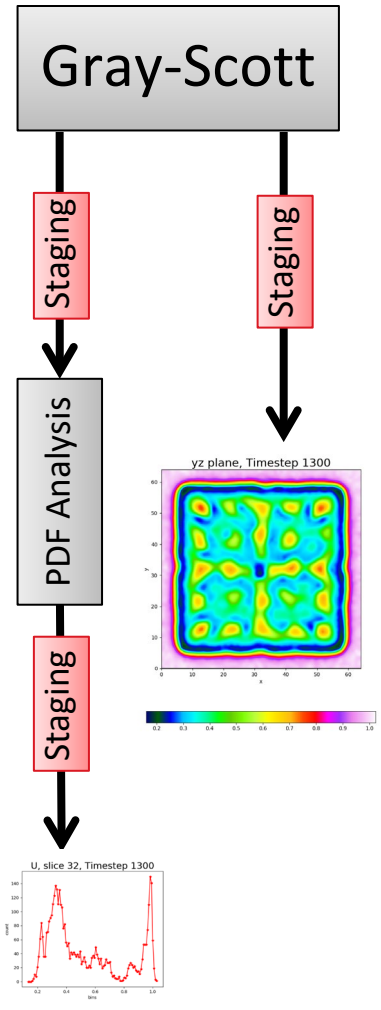
```
adios2::StepStatus read_status =
        reader.BeginStep(adios2::StepMode::Read, timeout);
if (read_status != adios2::StepStatus::OK) {
    break;
}
```

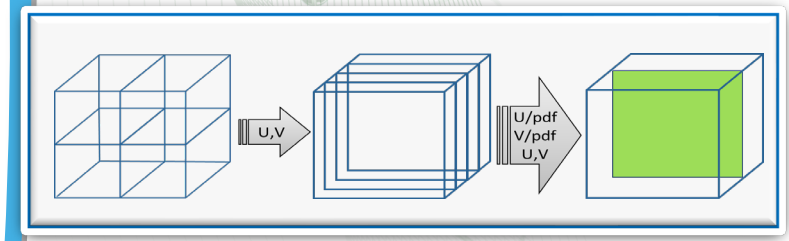
  - float timeout: wait for this long for a new step to arrive
- Release a step if not needed anymore
  - optimization to allow the staging method to deliver new steps if available

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

Gray-Scott example

with staging





File-based in situ processing

The runtime config file: adios2.xml

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

 <!--====================================
       Configuration for for Gray-Scott and GS Plot
     ====================================-->
  <io name="SimulationOutput">
    <engine type="BP4">
      <parameter key="OpenTimeoutSecs" value="10.0"/>
    </engine>
  </io>
```

Engine types

BP4

HDF5

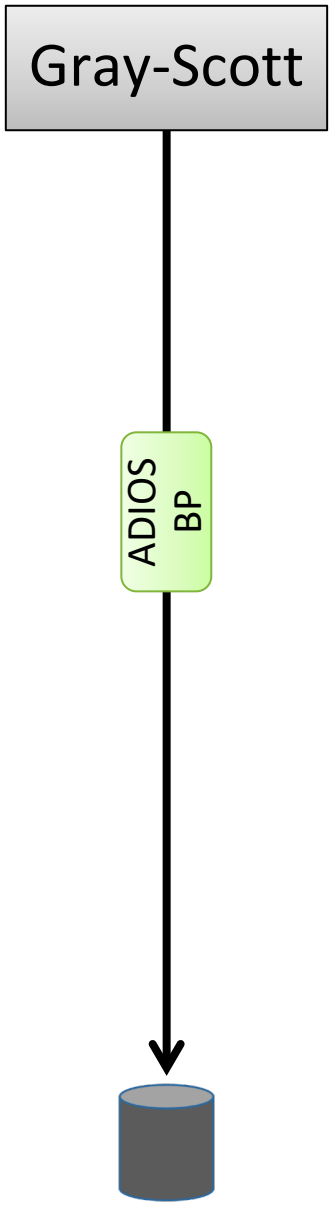
SST

InSituMPI

DataMan

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

  <io name="PDFAnalysisOutput">
    <engine type="BP4">
      <parameter key="OpenTimeoutSecs" value="10.0"/>
    </engine>
  </io>

</adios-config>
```

In Situ Visualization with ADIOS

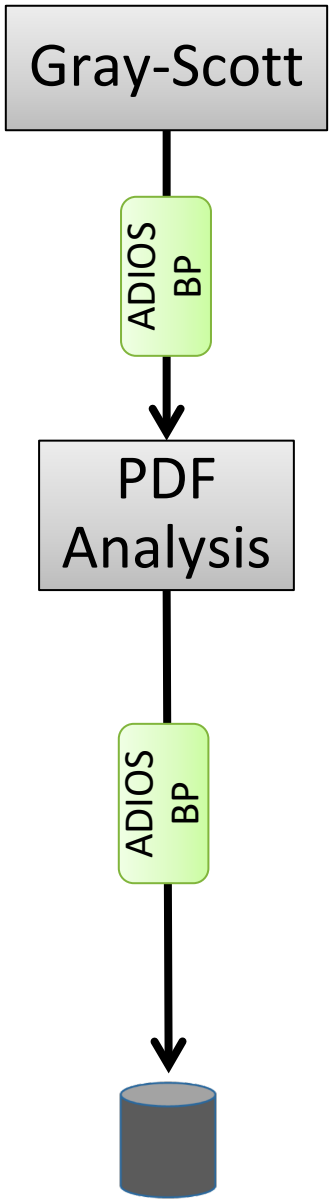


edit **adios2.xml: SimulationOutput, PDFAnalysisOutput** uses **BP4** engine

```
$ ./cleanup.sh data
```

```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
Simulation writes data using engine type:              BP4
=========================================
grid:               64x64x64
steps:              60000
plotgap:            100
F:                  0.02
k:                  0.048
dt:                 1
Du:                 0.2
Dv:                 0.1
noise:              0.01
output:             gs.bp
adios_config:       adios2.xml
decomposition:      2x2x1
grid per process: 32x32x64
=========================================
Simulation at step 0 writing output step     0
Simulation at step 100 writing output step     1
```

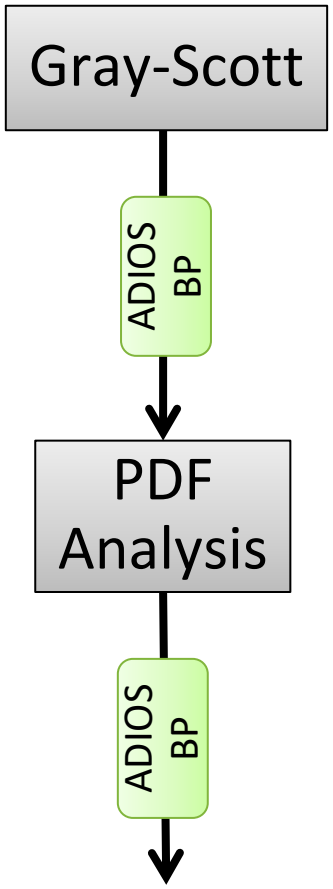
In Situ Visualization with ADIOS




```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:              BP4
=========================================
grid:               64x64x64
steps:              60000
 …
```

```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  BP4
PDF analysis writes using engine type:                 BP4
PDF Analysis step 0 processing sim output step 0 sim compute step 0
PDF Analysis step 1 processing sim output step 1 sim compute step 100
PDF Analysis step 2 processing sim output step 2 sim compute step 200
PDF Analysis step 3 processing sim output step 3 sim compute step 300
PDF Analysis step 4 processing sim output step 4 sim compute step 400
PDF Analysis step 5 processing sim output step 5 sim compute step 500
PDF Analysis step 6 processing sim output step 6 sim compute step 600
...
```

In Situ Visualization with ADIOS

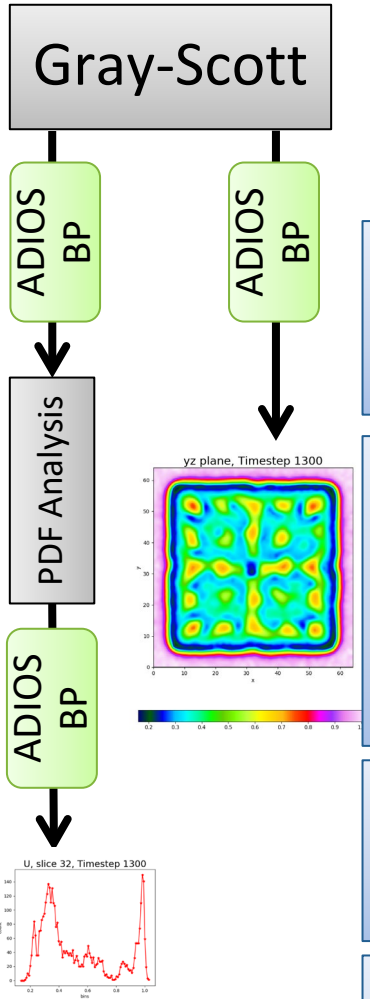



```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:                BP4
=========================================
grid:                 64x64x64
steps:                60000
 …
```

```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  BP4
PDF analysis writes using engine type:                 BP4
PDF Analysis step 0 processing sim output step 0 sim compute step 0
...
```

```
$ python3 pdfplot.py -i pdf.bp
PDF Plot step 0 processing analysis step 0 simulation step 0
PDF Plot step 1 processing analysis step 1 simulation step 100
PDF Plot step 2 processing analysis step 2 simulation step 200
PDF Plot step 3 processing analysis step 3 simulation step 300
...
```

In Situ Visualization with ADIOS




```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:                BP4
=========================================
```

```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  BP4
PDF analysis writes using engine type:                 BP4
PDF Analysis step 0 processing sim output step 0 sim compute step 0
...
```

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

```
$ python3 gsplot.py -i gs.bp
GS Plot step 0 processing simulation output step 0 or computation step 0
GS Plot step 1 processing simulation output step 1 or computation step 100
GS Plot step 2 processing simulation output step 2 or computation step 200
...
```

Streaming in situ processing with SST engine

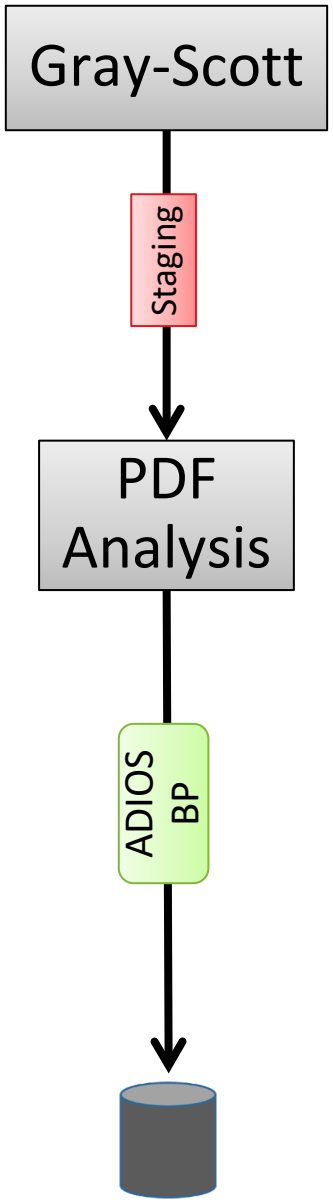
The runtime config file: adios2.xml

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

   <!--================================
         Configuration for the Simulation Output
       ================================-->
   <io name="SimulationOutput">
      <engine type="SST">
      </engine>
   </io>
```

Engine types
BP4
HDF5
SST
InSituMPI
DataMan

```
   <!--================================
         Configuration for the Analysis Output
       ================================-->

   <io name="AnalysisOutput">
      <engine type="SST">
      </engine>
   </io>

   <!--================================
         Configuration for the Visualization Input
       ================================-->
   <io name="VizInput">
      <engine type="SST">
      </engine>
   </io>

</adios-config>
```

In Situ Visualization with ADIOS

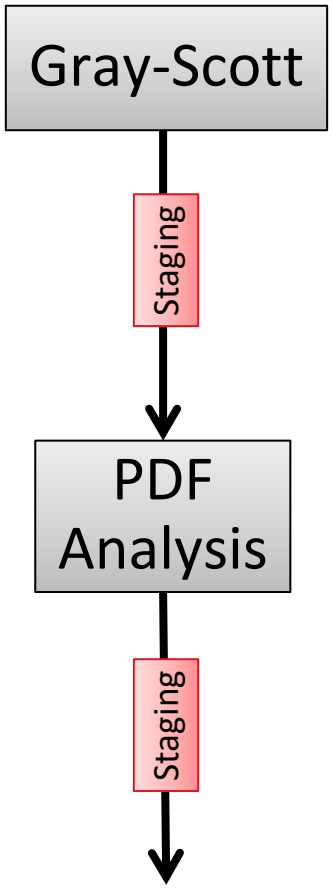


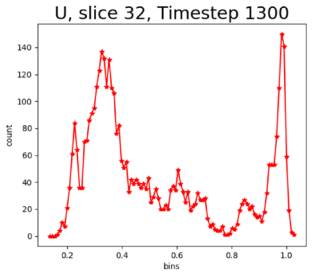

```
edit adios2.xml and change SimulationOutput to SST
                           PDFAnalysisOutput   to BP4
$ ./cleanup.sh data
```

```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:              SST
=========================================
grid:               64x64x64
steps:              60000
 …
```

```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  SST
PDF analysis writes using engine type:                 BP4
PDF Analysis step 0 processing sim output step 0 sim compute step 0
PDF Analysis step 1 processing sim output step 1 sim compute step 100
PDF Analysis step 2 processing sim output step 2 sim compute step 200
PDF Analysis step 3 processing sim output step 3 sim compute step 300
PDF Analysis step 4 processing sim output step 4 sim compute step 400
PDF Analysis step 5 processing sim output step 5 sim compute step 500
PDF Analysis step 6 processing sim output step 6 sim compute step 600
...
```

In Situ Visualization with ADIOS





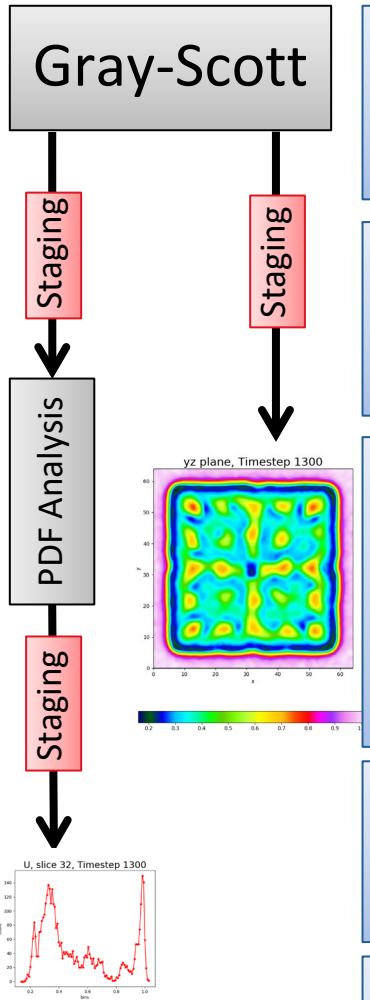

```
edit adios2.xml and change PDFAnalysisOutput to SST as well

$ ./cleanup.sh data
```

```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:                SST
=========================================
grid:                 64x64x64
steps:                60000
 …
```

```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  SST
PDF analysis writes using engine type:                 SST
PDF Analysis step 0 processing sim output step 0 sim compute step 0
...
```

```
$ python3 pdfplot.py -i pdf.bp
PDF Plot step 0 processing analysis step 0 simulation step 0
PDF Plot step 1 processing analysis step 1 simulation step 100
PDF Plot step 2 processing analysis step 2 simulation step 200
PDF Plot step 3 processing analysis step 3 simulation step 300
...
```

In Situ Visualization with ADIOS




```
edit adios2.xml and change AnalysisOutput to SST
                           VizInput       to SST
$ ./cleanup.sh data
```

```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:              SST
=========================================
```

```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  SST
PDF analysis writes using engine type:                 SST
PDF Analysis step 0 processing sim output step 0 sim compute step 0
...
```

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

```
$ python3 gsplot.py -i gs.bp
GS Plot step 0 processing simulation output step 0 or computation step 0
GS Plot step 1 processing simulation output step 1 or computation step 100
GS Plot step 2 processing simulation output step 2 or computation step 200
...
```

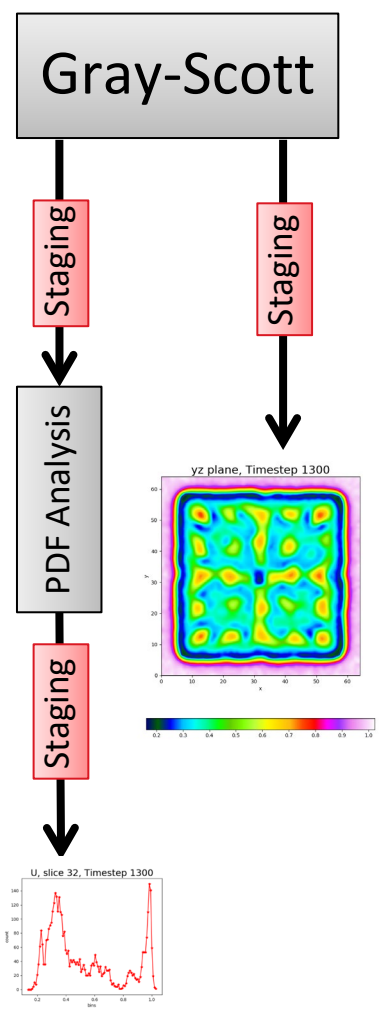
Changing the pipeline

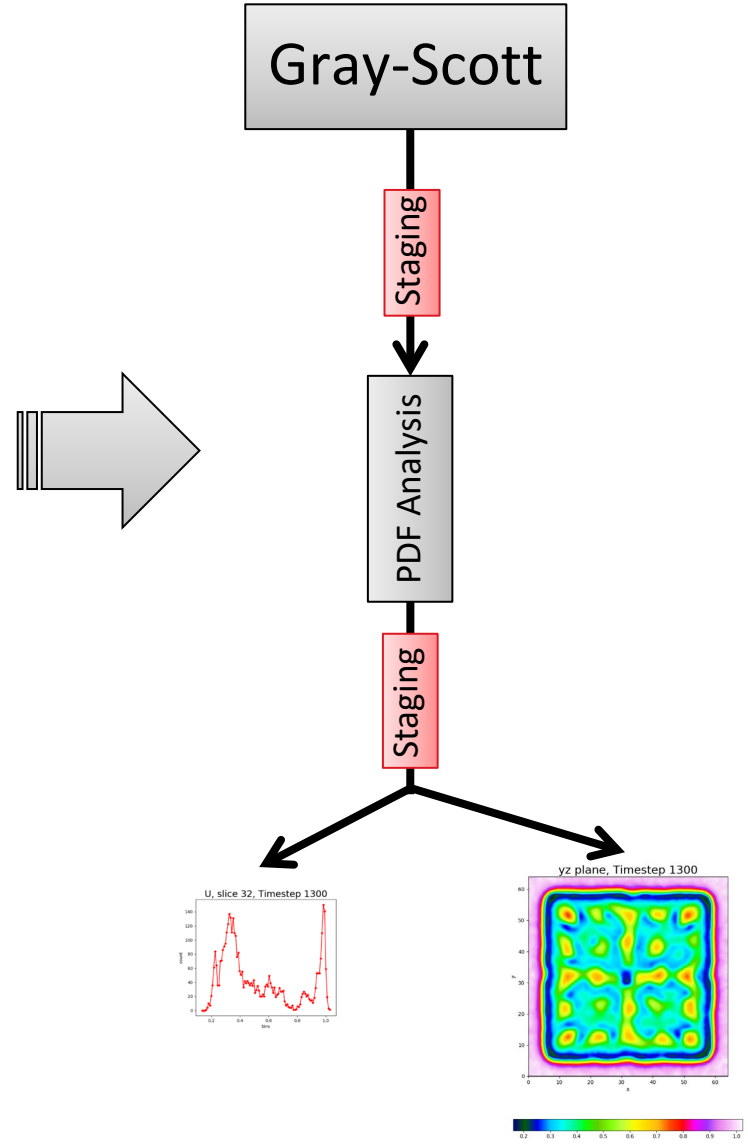
- Run both Plotting from PDFAnalysis output

Hints

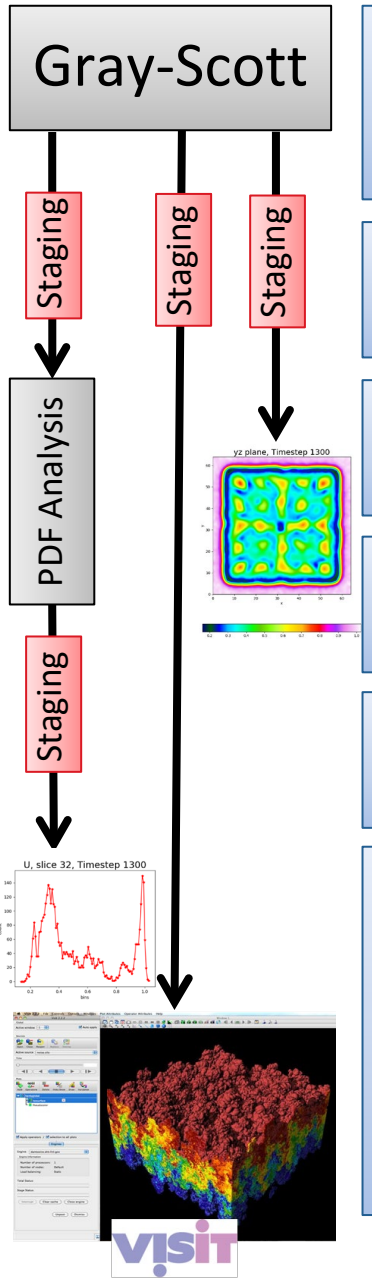
- pdf_calc: Add extra command-line argument to : YES
  to write U, V variables to pdf.bp

- gsplot.py: Plot "pdf.bp"
  instead of "gs.bp"
  - Change on command line





In Situ Visualization with ADIOS




```
edit adios2.xml and change AnalysisOutput to SST
                           VizInput       to SST
$ ./cleanup.sh data
```

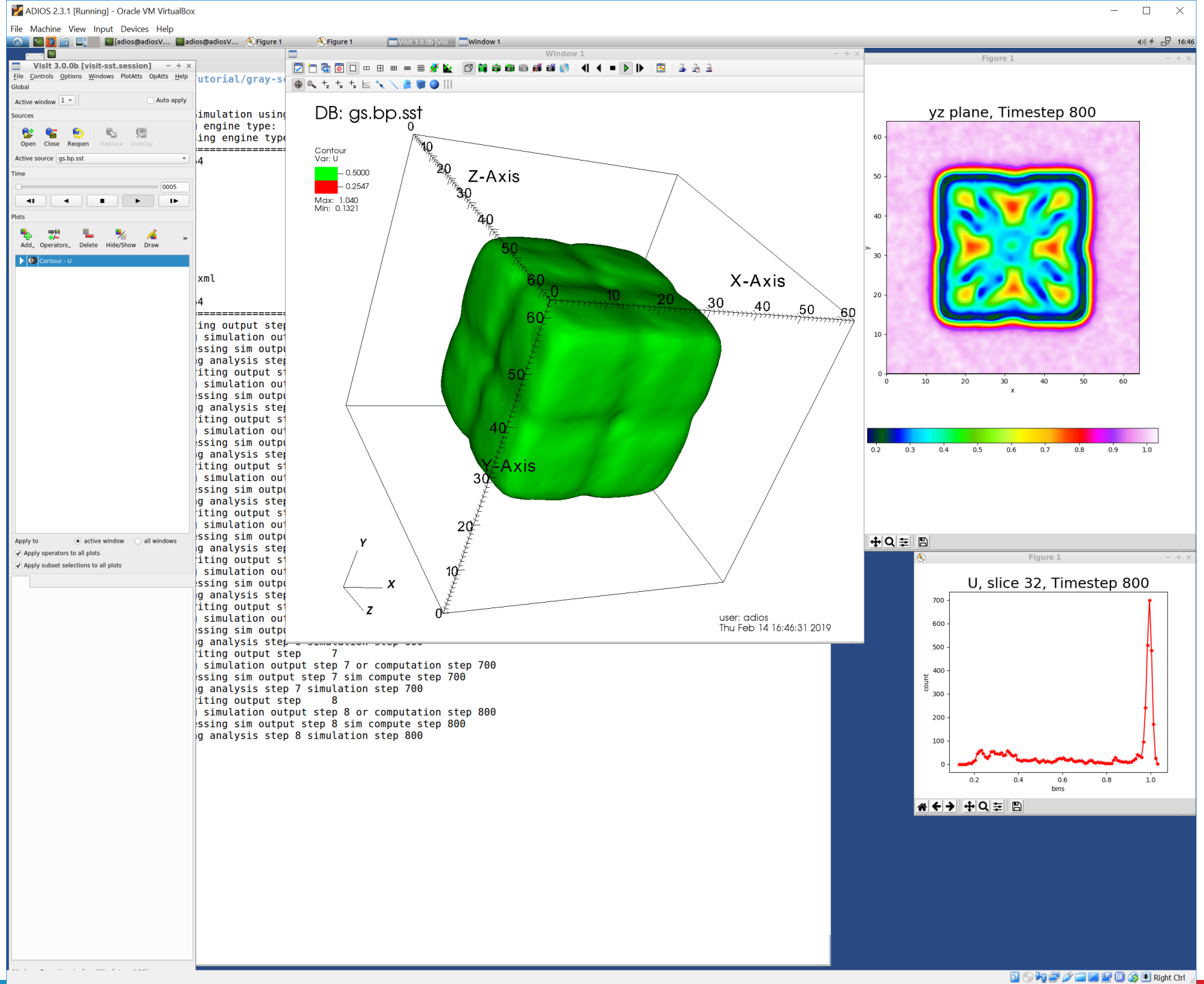
```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:                SST
```

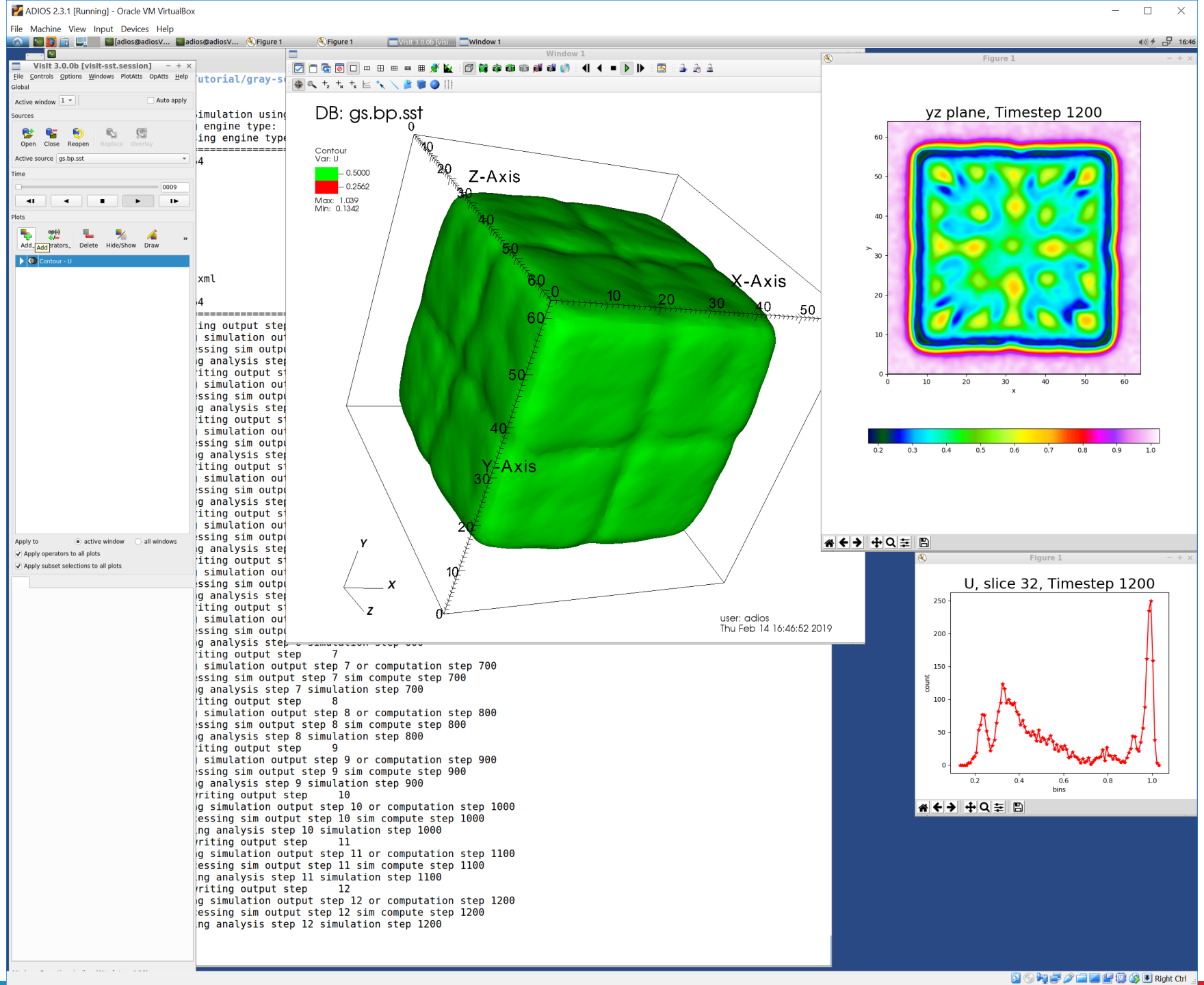
```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  SST
```

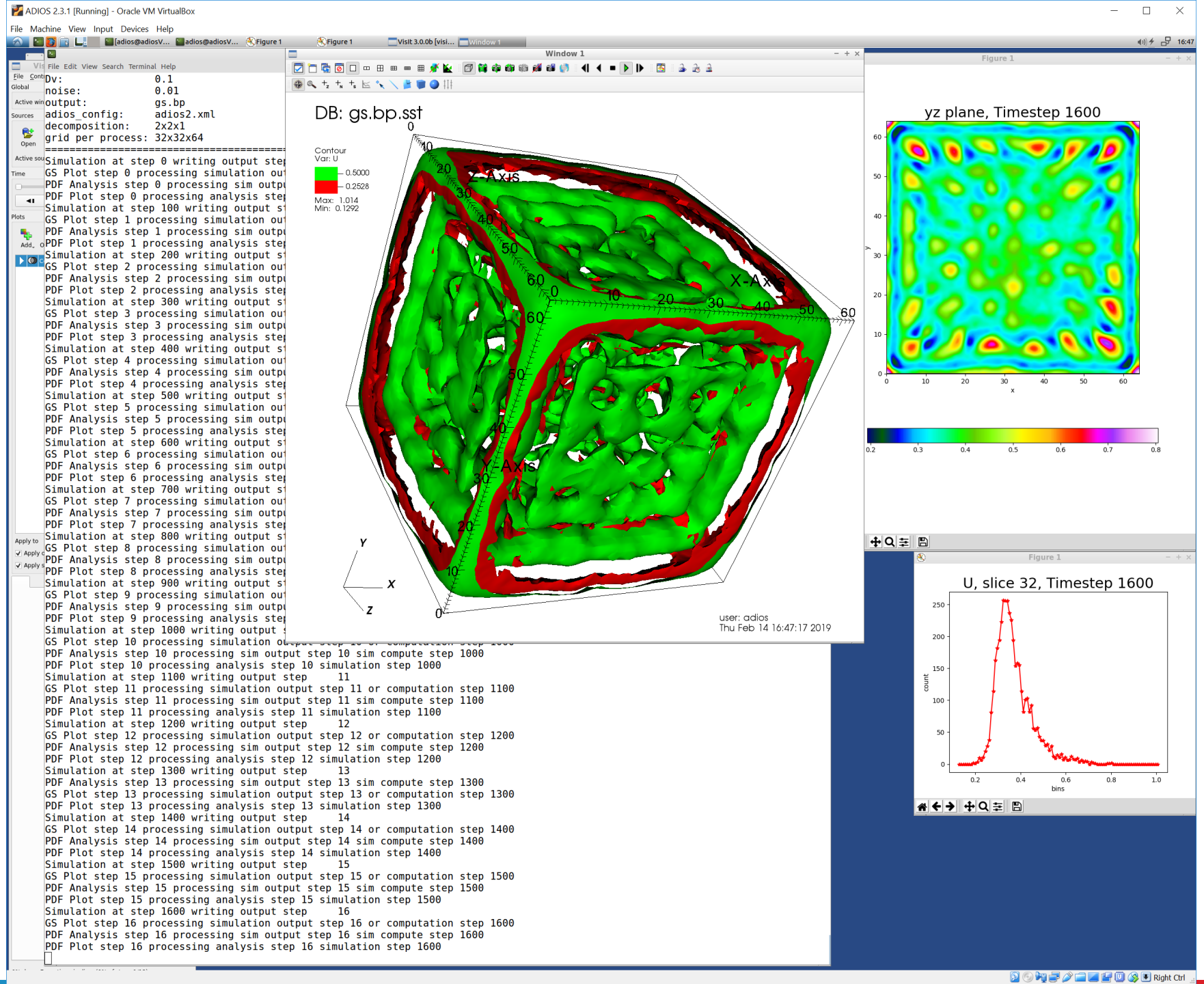
```
$ python3 pdfplot.py -i pdf.bp
PDF Plot step 0 processing analysis step 0 simulation step 0
```

```
$ python3 gsplot.py -i gs.bp
GS Plot step 0 processing simulation output step 0 or computation step 0
```

```
$ visit -sessionfile visit-sst.session
Running: gui3.0.0b -sessionfile visit-sst.session
Running: viewer3.0.0b -forceversion 3.0.0b -geometry 2182x1967+354+26 -
borders 26,4,1,1 -shift 1,26 -preshift 0,0 -defer -host 127.0.0.1 -port
5600
Running: mdserver3.0.0b -forceversion 3.0.0b -host 127.0.0.1 -port 5601
```


ADIOS 2.3.1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
VisIt 3.0.0b [visit-sst.session]
Window 1
DB: gs.bp.sst
Contour
Var: U
0.5000
0.2547
Max: 1.040
Min: 0.1321
Z-Axis
X-Axis
Y-Axis
user: adios
Thu Feb 14 16:46:31 2019
Figure 1
yz plane, Timestep 800
Figure 1
U, slice 32, Timestep 800
count
bins
simulation output step 7 or computation step 700
sim output step 7 sim compute step 700
analysis step 7 simulation step 700
output step 8
simulation output step 8 or computation step 800
sim output step 8 sim compute step 800
analysis step 8 simulation step 800


ADIOS 2.3.1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
VisIt 3.0.0b [visit-sst.session]
DB: gs.bp.sst
Contour
Var: U
0.5000
0.2562
Max: 1.039
Min: 0.1342
Z-Axis
X-Axis
Y-Axis
user: adios
Thu Feb 14 16:46:52 2019
yz plane, Timestep 1200
U, slice 32, Timestep 1200
count
bins
writing output step 7
simulation output step 7 or computation step 700
ssing sim output step 7 sim compute step 700
g analysis step 7 simulation step 700
writing output step 8
simulation output step 8 or computation step 800
ssing sim output step 8 sim compute step 800
g analysis step 8 simulation step 800
writing output step 9
simulation output step 9 or computation step 900
ssing sim output step 9 sim compute step 900
g analysis step 9 simulation step 900
writing output step 10
g simulation output step 10 or computation step 1000
essing sim output step 10 sim compute step 1000
ng analysis step 10 simulation step 1000
writing output step 11
g simulation output step 11 or computation step 1100
essing sim output step 11 sim compute step 1100
ng analysis step 11 simulation step 1100
writing output step 12
g simulation output step 12 or computation step 1200
essing sim output step 12 sim compute step 1200
ng analysis step 12 simulation step 1200


ADIOS 2.3.1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Dv: 0.1
noise: 0.01
output: gs.bp
adios_config: adios2.xml
decomposition: 2x2x1
grid per process: 32x32x64
Simulation at step 0 writing output step
GS Plot step 0 processing simulation output step 0 or computation step 0
PDF Analysis step 0 processing sim output step 0 sim compute step 0
PDF Plot step 0 processing analysis step 0 simulation step 0
Simulation at step 100 writing output step 1
GS Plot step 1 processing simulation output step 1 or computation step 100
PDF Analysis step 1 processing sim output step 1 sim compute step 100
PDF Plot step 1 processing analysis step 1 simulation step 100
Simulation at step 200 writing output step 2
GS Plot step 2 processing simulation output step 2 or computation step 200
PDF Analysis step 2 processing sim output step 2 sim compute step 200
PDF Plot step 2 processing analysis step 2 simulation step 200
Simulation at step 300 writing output step 3
GS Plot step 3 processing simulation output step 3 or computation step 300
PDF Analysis step 3 processing sim output step 3 sim compute step 300
PDF Plot step 3 processing analysis step 3 simulation step 300
Simulation at step 400 writing output step 4
GS Plot step 4 processing simulation output step 4 or computation step 400
PDF Analysis step 4 processing sim output step 4 sim compute step 400
PDF Plot step 4 processing analysis step 4 simulation step 400
Simulation at step 500 writing output step 5
GS Plot step 5 processing simulation output step 5 or computation step 500
PDF Analysis step 5 processing sim output step 5 sim compute step 500
PDF Plot step 5 processing analysis step 5 simulation step 500
Simulation at step 600 writing output step 6
GS Plot step 6 processing simulation output step 6 or computation step 600
PDF Analysis step 6 processing sim output step 6 sim compute step 600
PDF Plot step 6 processing analysis step 6 simulation step 600
Simulation at step 700 writing output step 7
GS Plot step 7 processing simulation output step 7 or computation step 700
PDF Analysis step 7 processing sim output step 7 sim compute step 700
PDF Plot step 7 processing analysis step 7 simulation step 700
Simulation at step 800 writing output step 8
GS Plot step 8 processing simulation output step 8 or computation step 800
PDF Analysis step 8 processing sim output step 8 sim compute step 800
PDF Plot step 8 processing analysis step 8 simulation step 800
Simulation at step 900 writing output step 9
GS Plot step 9 processing simulation output step 9 or computation step 900
PDF Analysis step 9 processing sim output step 9 sim compute step 900
PDF Plot step 9 processing analysis step 9 simulation step 900
Simulation at step 1000 writing output step 10
GS Plot step 10 processing simulation output step 10 or computation step 1000
PDF Analysis step 10 processing sim output step 10 sim compute step 1000
PDF Plot step 10 processing analysis step 10 simulation step 1000
Simulation at step 1100 writing output step 11
GS Plot step 11 processing simulation output step 11 or computation step 1100
PDF Analysis step 11 processing sim output step 11 sim compute step 1100
PDF Plot step 11 processing analysis step 11 simulation step 1100
Simulation at step 1200 writing output step 12
GS Plot step 12 processing simulation output step 12 or computation step 1200
PDF Analysis step 12 processing sim output step 12 sim compute step 1200
PDF Plot step 12 processing analysis step 12 simulation step 1200
Simulation at step 1300 writing output step 13
PDF Analysis step 13 processing sim output step 13 sim compute step 1300
GS Plot step 13 processing simulation output step 13 or computation step 1300
PDF Plot step 13 processing analysis step 13 simulation step 1300
Simulation at step 1400 writing output step 14
GS Plot step 14 processing simulation output step 14 or computation step 1400
PDF Analysis step 14 processing sim output step 14 sim compute step 1400
PDF Plot step 14 processing analysis step 14 simulation step 1400
Simulation at step 1500 writing output step 15
GS Plot step 15 processing simulation output step 15 or computation step 1500
PDF Analysis step 15 processing sim output step 15 sim compute step 1500
PDF Plot step 15 processing analysis step 15 simulation step 1500
Simulation at step 1600 writing output step 16
GS Plot step 16 processing simulation output step 16 or computation step 1600
PDF Analysis step 16 processing sim output step 16 sim compute step 1600
PDF Plot step 16 processing analysis step 16 simulation step 1600
Window 1
DB: gs.bp.sst
Contour
Var: U
0.5000
0.2528
Max: 1.014
Min: 0.1292
Z-Axis
X-Axis
Y-Axis
user: adios
Thu Feb 14 16:47:17 2019
Figure 1
yz plane, Timestep 1600
Figure 1
U, slice 32, Timestep 1600
count
bins
Right Ctrl

Staging I/O

adios_reorganize tool

heat transfer example with *adios_reorganize*

- Staged reading code
  - **`/opt/adios2/bin/adios_reorganize`**
- This code
  - reads an ADIOS dataset using an ADIOS read method, step-by-step
  - writes out each step using an ADIOS write method
- Use cases
  - Staged write
    - asynchronous I/O using extra compute nodes, a.k.a burst buffer
  - Reorganize data from N process output to M process output
    - many subfiles to less, bigger subfiles, or one big file

Gray-scott example with staged I/O

```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
edit adios2.xml  (vi, gedit)
set engine to SST with blocking policy and waiting for the reader at start
   <io name="SimulationOutput">
      <engine type="SST">
         <parameter key="RendezvousReaderCount" value="1"/>
         <parameter key="QueueLimit" value="5"/>
         <parameter key="QueueFullPolicy" value="Block"/>
      </engine>
   </io>
$ mpirun -n 4 adios2-gray-scott settings-files.json
```

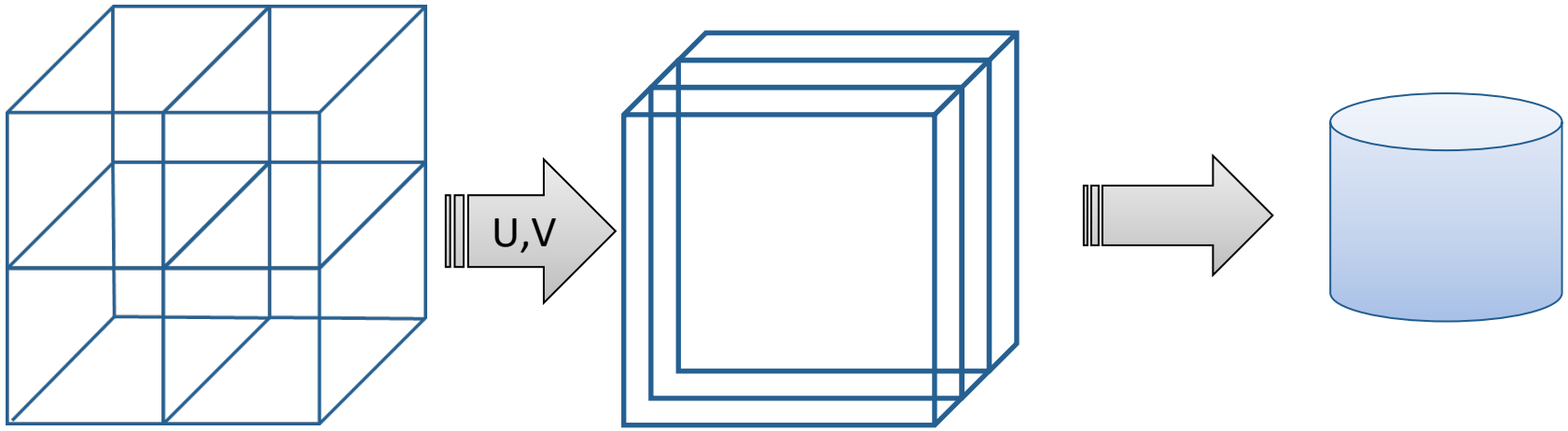
In another terminal

```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
$ mpirun -n 2   /opt/adios2/bin/adios_reorganize   gs.bp   staged.bp   SST   ""   BP4   ""   2
Input stream              = gs.bp
Output stream             = staged.bp
Read method               = SST
Read method parameters    =
Write method              = BP4
Write method parameters   =
Waiting to open stream gs.bp...
$ bpls -l staged.bp
```

```
$ bpls -D   staged.bp   U
   double   U      100*{64, 64, 64}
        step  0:
          block 0: [ 0:31,   0:63,   0:63]
          block 1: [32:63,   0:63,   0:63]
        step  1:
```

N to M reorganization with adios2_reorganize

- Gray-scott + adios_reorganize running together
  - Write out 6 time-steps.
  - Write from 4 cores, arranged in a 1x2x2 arrangement.
  - Read from 3 cores, arranged as 3x1x1



N to M reorganization with adios_reorganize

```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
edit adios2.xml  (vi, gedit)
set engine to BP4
 <io name="SimulationOutput">
     <engine type="BP4">
     </engine>
  </io>
$ mpirun -n 4 adios2-gray-scott settings-files.json
$ bpls -D  gs.bp   U
 double   U     100*{64, 64, 64}
        step  0:
          block 0: [ 0:63,  0:31,  0:31]
          block 1: [ 0:63, 32:63,  0:31]
          block 2: [ 0:63,  0:31, 32:63]
          block 3: [ 0:63, 32:63, 32:63]
$ mpirun -n 3   /opt/adios2/bin/adios_reorganize   gs.bp   g_3.bp   BP4 ""   BP4  ""  3
$ bpls -D  g_3.bp   U
   double   U     100*{64, 64, 64}
        step  0:
          block 0: [ 0:20,  0:63,  0:63]
          block 1: [21:41,  0:63,  0:63]
          block 2: [42:63,  0:63,  0:63]
```

Summary

- Scientific data is growing faster in size compared to the increase of computing power
- The use of accelerators and NVRAM are vital for the growth of our area but we should let high level libraries (ADIOS, HDF5) be the preferred way to match application needs and the network, and storage infrastructure
- Both ADIOS and HDF5 are powerful middleware packages for high performant parallel I/O
- ADIOS provides an abstraction for data-in-motion (streams, pipelines)
  - New engines in ADIOS allow for powerful techniques necessary for 21st century science
- Data compression, and Data querying are necessary items in any high-performance library which should be heavily integrated into the high level middleware (ADIOS, HDF5)