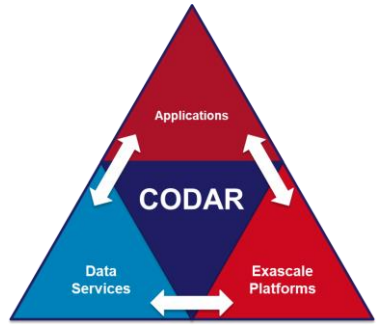
High-performance I/O Frameworks 101

ECP AHM – Houston, TX - Feb 5, 2020

Presenters: S. Klasky[123], N. Podhorszki[1], W. Godoy[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], 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, Department 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 2.5.0
  - https://github.com/ornladios/ADIOS2/releases/tag/v2.5.0
- HDF5 1.10.4
  - https://www.hdfgroup.org/downloads/hdf5/source-code/
- 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

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

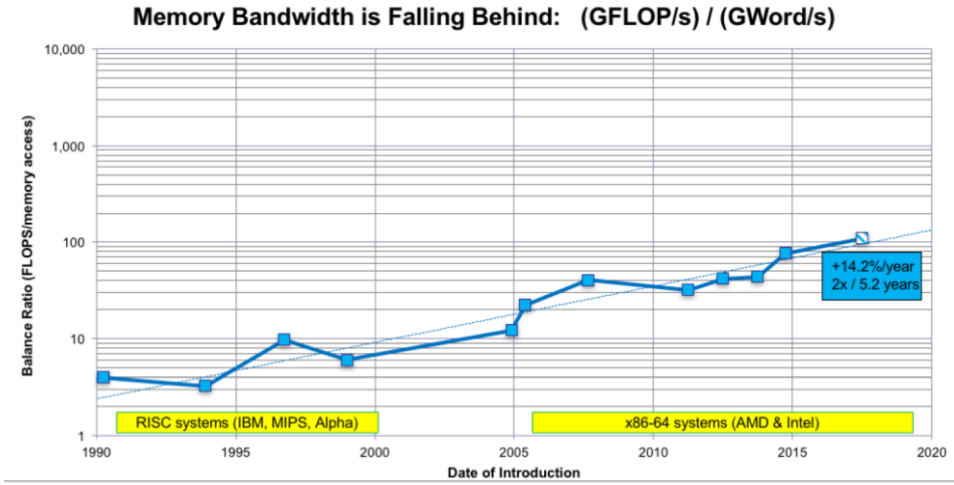
- Two movies showing the Tutorial for post-processing and on-line processing
- It's on the VM in /home/adios/Videos
  - GNOME Mplayer can play it on the VM
- https://users.nccs.gov/~pnorbert/GrayScottPost.mp4
- https://users.nccs.gov/~pnorbert/GrayScottInsitu.mp4

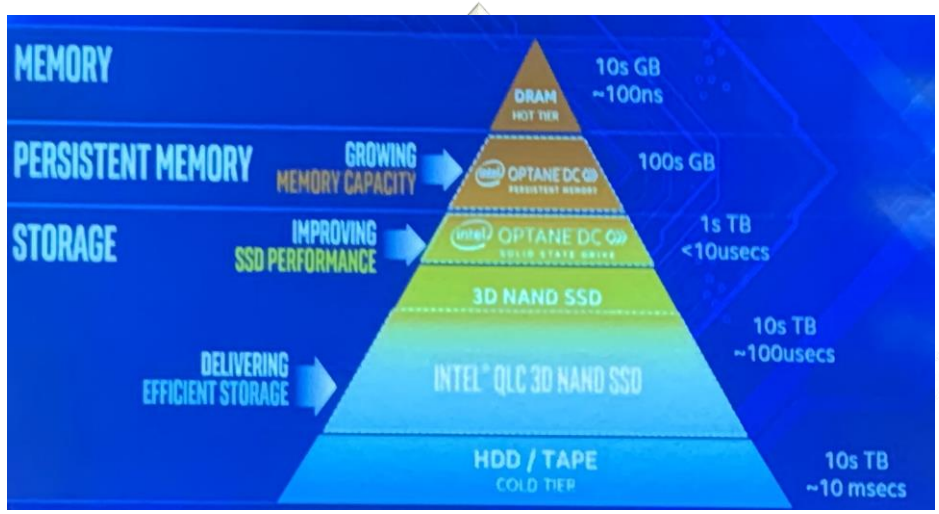
The Data Problem

- **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

I/O on HPC machines is challenging

- Problem
  - File system/network bandwidth does not keep up with computing power
  - Too much data which is written to the storage system is either purged or never read back for post-processing
- Approach
  - Create an I/O abstraction layer for data movement
    - Data at Rest
    - Data in motion
  - Create a self describing I/O package for extreme scale data
  - Refactor the data into different levels of *importance* according to the information





Push: The widening compute-data gap on multi-tier storage

- Filesystem/network bandwidth falls behind CPU/memory: Fewer bytes/operation -Need efficient I/O
- Filesystem has an additional layer (Burst Buffer) which is different on all of the LCFs/NERSC- Need new functionality to write/read to all storage layers

Feature	Titan	Summit
Peak Flops	**27 PF**	**200 PF**
Application Performance	Baseline	5-10x Titan
Number of Nodes	18,688	~4,600
Node performance	1.4 TF	> 40 TF
Memory per Node	32 GB DDR3 + 6 GB GDDR5	512 GB DDR4 + 96 GB HBM
NV memory per Node	**0**	**1600 GB**
Total System Memory	**710 TB** (600 TB DDR3 + 110 TB GDDR5)) **14X**	**10 PB** (2.3 PB DDR4 + 0.4 PB HBM + 7.4 PB NVRAM)
System Interconnect (node injection bandwidth)	Gemini (6.4 GB/s)	Dual Rail EDR-IB (23 GB/s)
Interconnect Topology	3D Torus	Non-blocking Fat Tree
Processors per node	1 AMD Opteron™ 1 NVIDIA Kepler™	2 IBM POWER9™ 6 NVIDIA Volta™ **2.5X**
File System	**32 PB, 1 TB/s, Lustre®**	**250 PB, 2.5 TB/s, GPFS™**

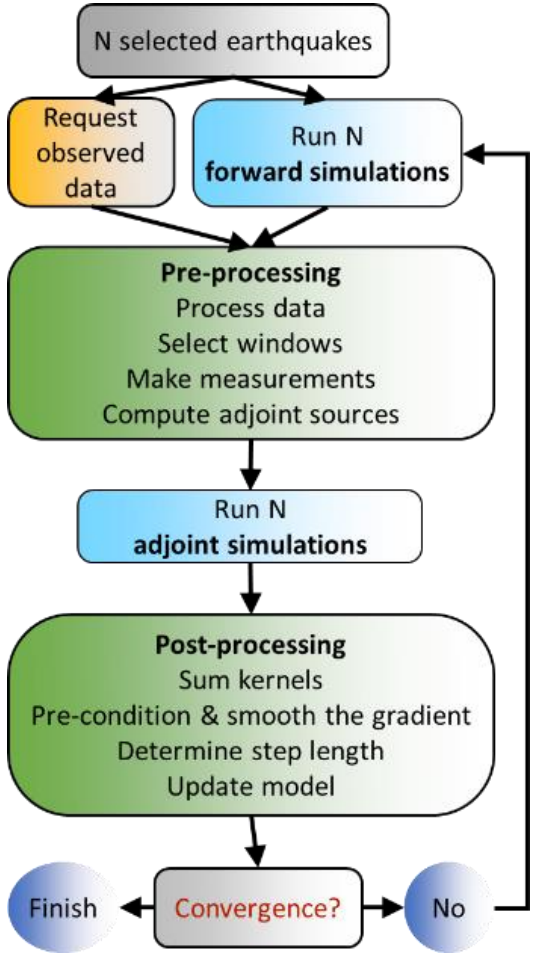
We need a Parallel File System

- High Performance Computing & Big Data requirements have outgrown the capabilities of any single host
  - (data set sizes) > (drive capacities)
  - Single server bandwidth is not sufficient to support access to all data from thousands of clients
- Need a parallel file system that can:
  - Scale capacity/bandwidth
  - Support large numbers of clients
- Lustre and GPFS are popular choices to meet these needs

Applications Pull

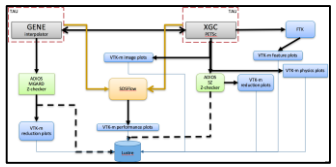
- Experimental science
  - Data Streams from experiments/observation
  - Detectors are generating data faster than we can move and store
  - Data has many variables/events, but each one may be small compared to simulations
  - Data often has “missing” values
- AI/ML applications
  - **Unstructured data**
  - Random access
- In situ data processing
  - **High Volume/Velocity communication between services**
  - Data reproducibility becomes an issue
  - Near Real Time processing
- Reproducibility
  - **Greater reliance on provenance information**

Codes such as SPECFEM3D_GLOBE produce over 10 PB of data, which need to be written efficiently and read back, during a 2.5 hours run on Summit



Pull: Applications are moving to online coupling

I. Foster: ECP CODAR review



End-to-end Application workflow

Can couple tasks via file system?

No: Too much data to output, store, or analyze offline. Must couple tasks online.

The motif decomposed

Which tasks?

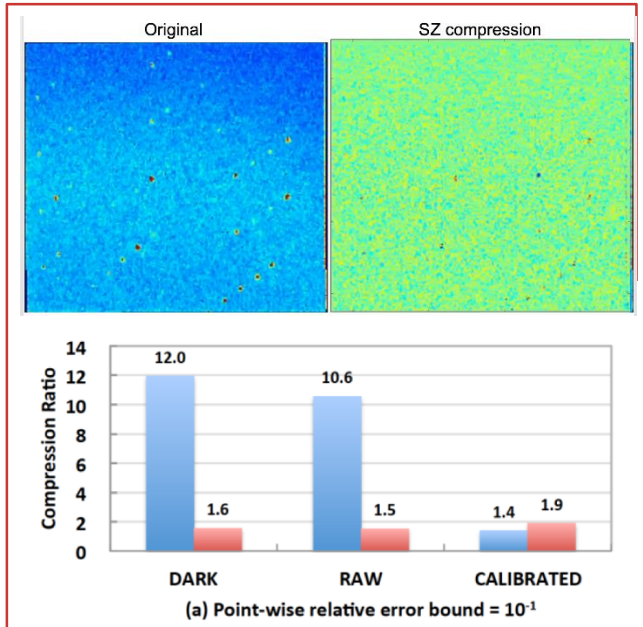
Application + Reduction

Application + Analysis

Application + Application

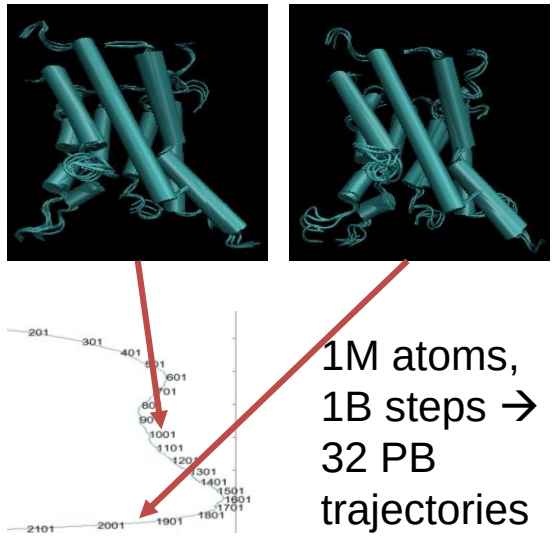
Many Applications

Online reduction



ExaFEL: X-ray laser imaging

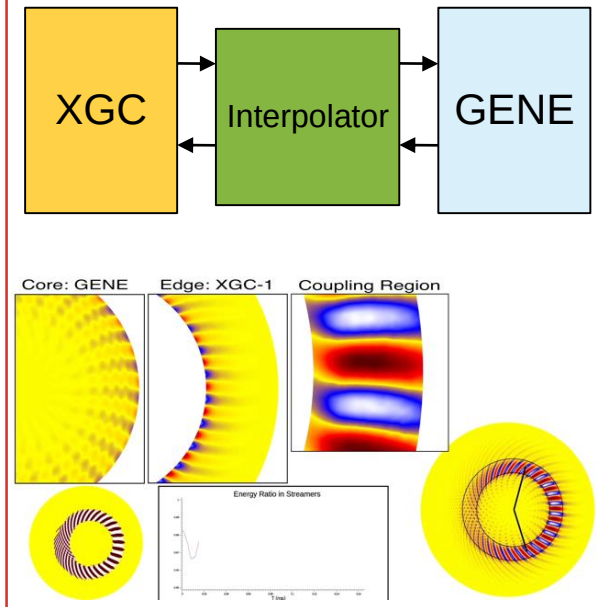
Online analysis



1M atoms, 1B steps → 32 PB trajectories

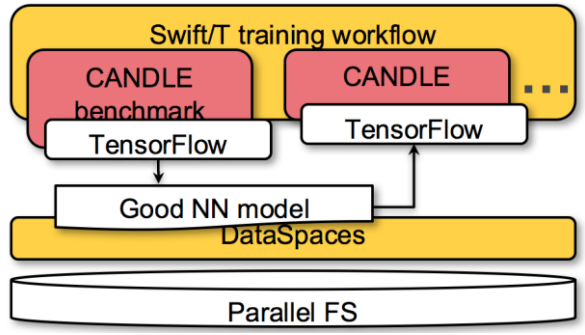
NWChemEx: Molecular dynamics

Online coupling



WDMApp: Fusion whole device model

Online aggregation



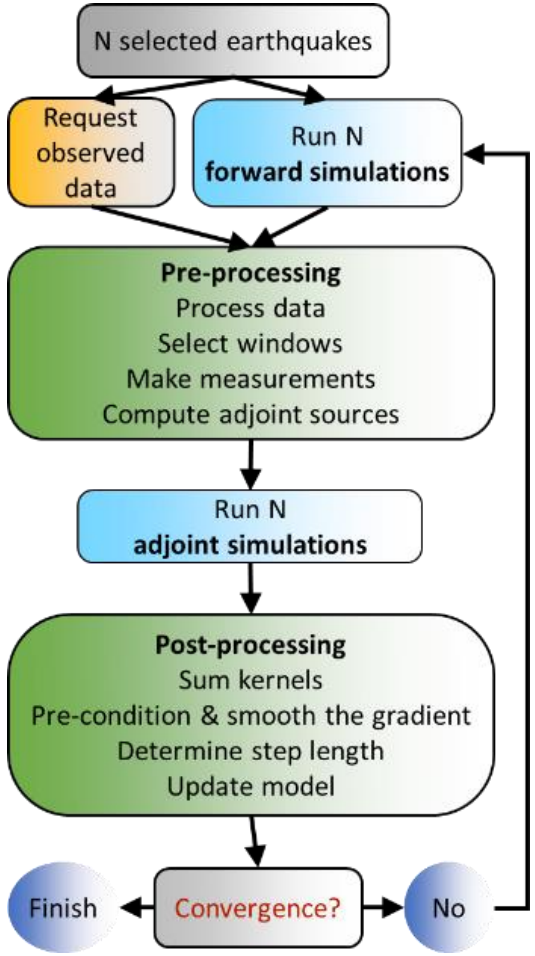
Hyperparam. optimization: 10^3–10^6 training runs, each fitting many parameters

CANDLE: Cancer deep learning

Technologies Push

- New Memory technologies
  - NVMe, Optane
- Data Movement across accelerators
  - GPU Direct, NIC directly attached to GPU or CPU
- Deep Storage Tiers
  - Number of storage tiers continues to grow
  - I/O solutions must efficiently move data to/from storage tiers
  - New solutions to “intelligently” move data across tiers becomes essential.

Codes such as SPECFEM3D_GLOBE produce over 10 PB of data, which need to be written efficiently and read back, during a 2.5 hours run on Summit



Does performance matter? (look at Summit)

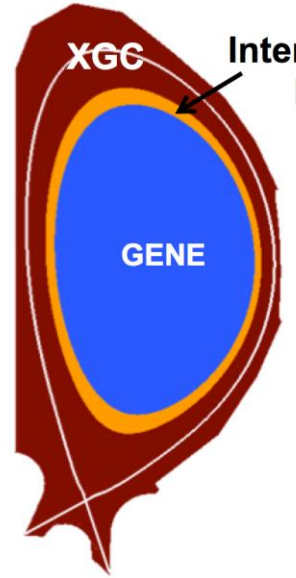
- Let's look at the difference for XGC1 on Summit
  - We need to output for 5D distribution function at every timestep, from every processor
  - Velocity= 1 TB/20 s, Volume = 42 PB, Total runtime = 10 days = 3.5M node hours
  - Speed with (other) ECP I/O technology =.060 TB/s=720,000s, I/O overhead = 83%
  - Speed with ADIOS = 1.2 TB/s = 36,000 s, I/O overhead = 4%
  - **Difference = 2.8 M node hours (cost savings) using ADIOS to the "other" ECP technology**
- Look at SPECFEM3D_GLOBE
  - Velocity = 5.2 TB/event
  - 50 events on 3,200 nodes, 260 TB in 94 (s) with no I/O
  - ADIOS speed = 125 s, = 2.1 TB/s = 133% I/O overhead
  - "other ECP technology" = 4,333 s = 60 GB/s = 4600% I/O overhead
- Experimental data from KSTAR (1,500 signals)
  - "other ECP technology" = 300 s
  - ADIOS = .2 s

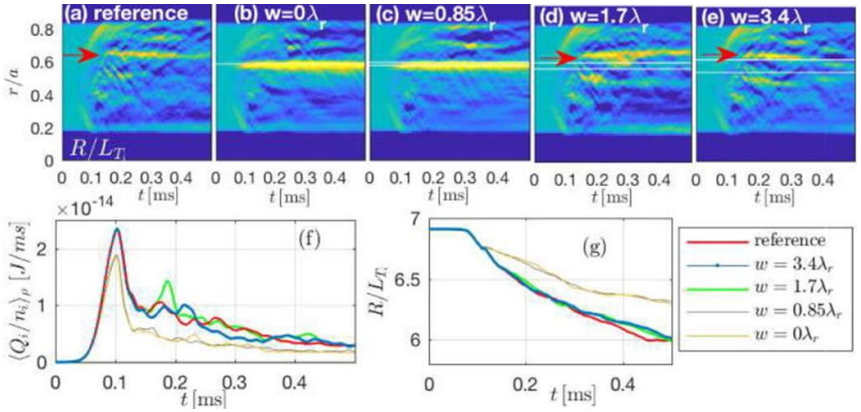
2.2.2.05 ADSE12-WDMApp: High-Fidelity Whole Device Modeling of Magnetically Confined Fusion Plasmas

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

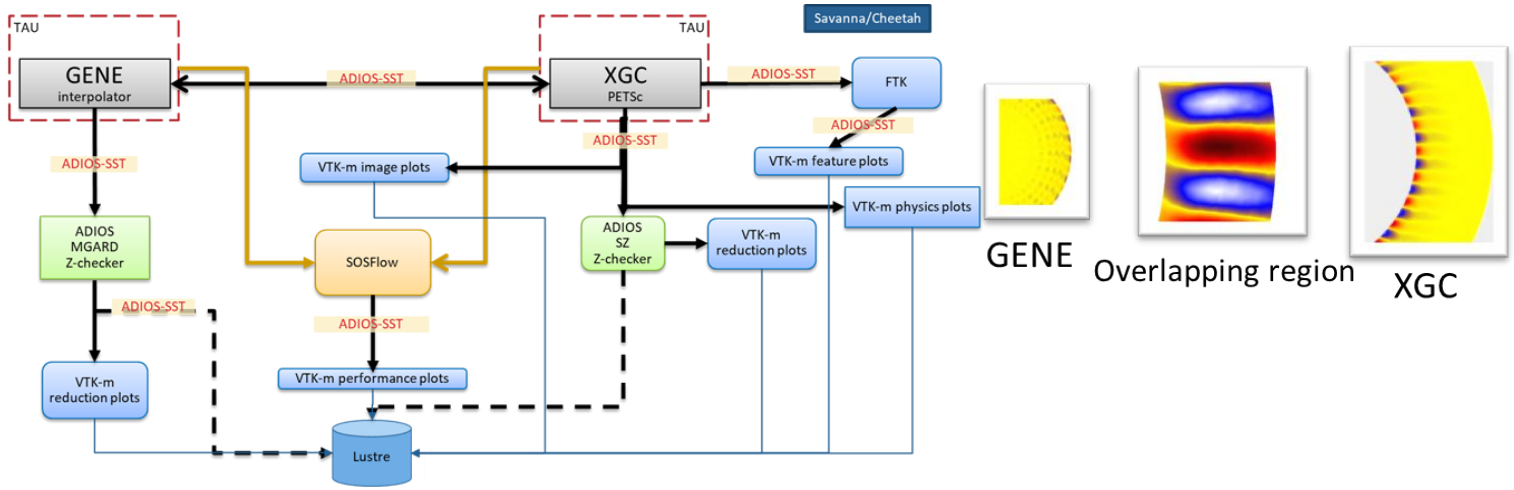
- Different physics solved in different physical regions of detector (spatial coupling)
- Core simulation: **GENE**
  Edge simulation: **XGC**
  Separate teams, **separate codes**
- Recently demonstrated first-ever successful kinetic coupling of this kind
- Data Generated by one coupled simulation is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

Core-edge coupling





J. Dominski; S. Ku; C.-S. Chang; J. Choi; E. Suchyta; S. Parker; S. Klasky; A. Bhattacharjee; *Physics of Plasmas* **2018,** 25, DOI: 10.1063/1.5044707



New I/O Pattern: emerging from running complex in situ workflows

I/O in Seismic Tomography Workflow (PBs of data/run)

Scientific Achievement

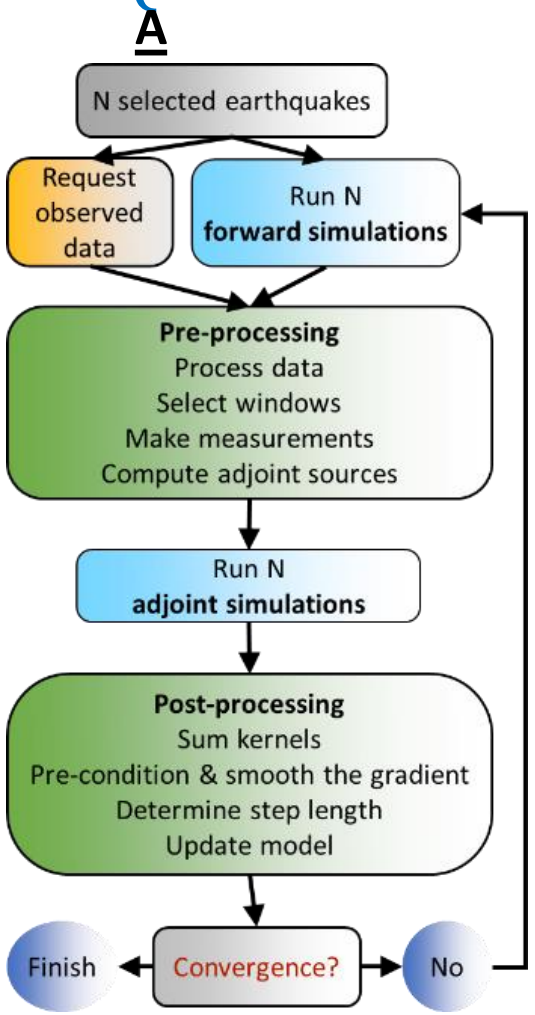
Most detailed 3-D model of Earth's interior showing the entire globe from the surface to the core–mantle boundary, a depth of 1,800 miles

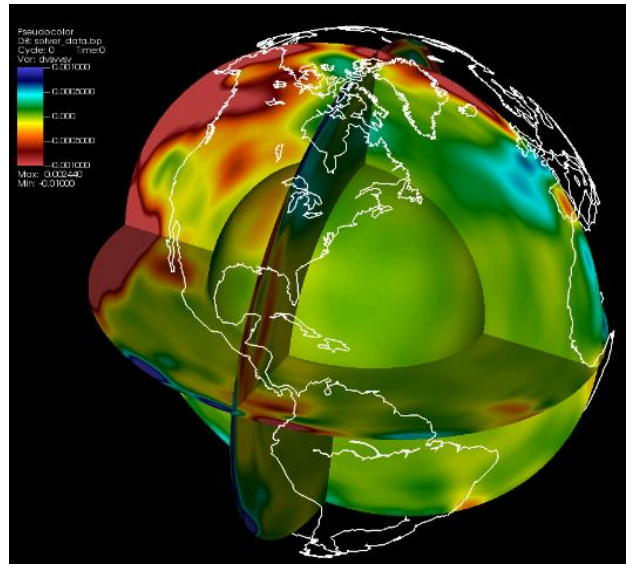
Significance and Impact

First global seismic model where no approximations were used to simulate how seismic waves travel through the Earth. Over 1 PB of data was generated in a 6 hour simulation (Titan@OLCF)

Research Details

- To improve data movement and flexibility, the Adaptable Seismic Data Format (ASDF) was developed that leverages the Adaptable I/O System (ADIOS) parallel library
- ASDF allows for recording, reproducing, and analyzing data on large-scale supercomputers
- 1PB of data is produced in a single workflow step, which is fully processed later in another step
- https://www.olcf.ornl.gov/2017/03/28/a-seismic-mapping-milestone

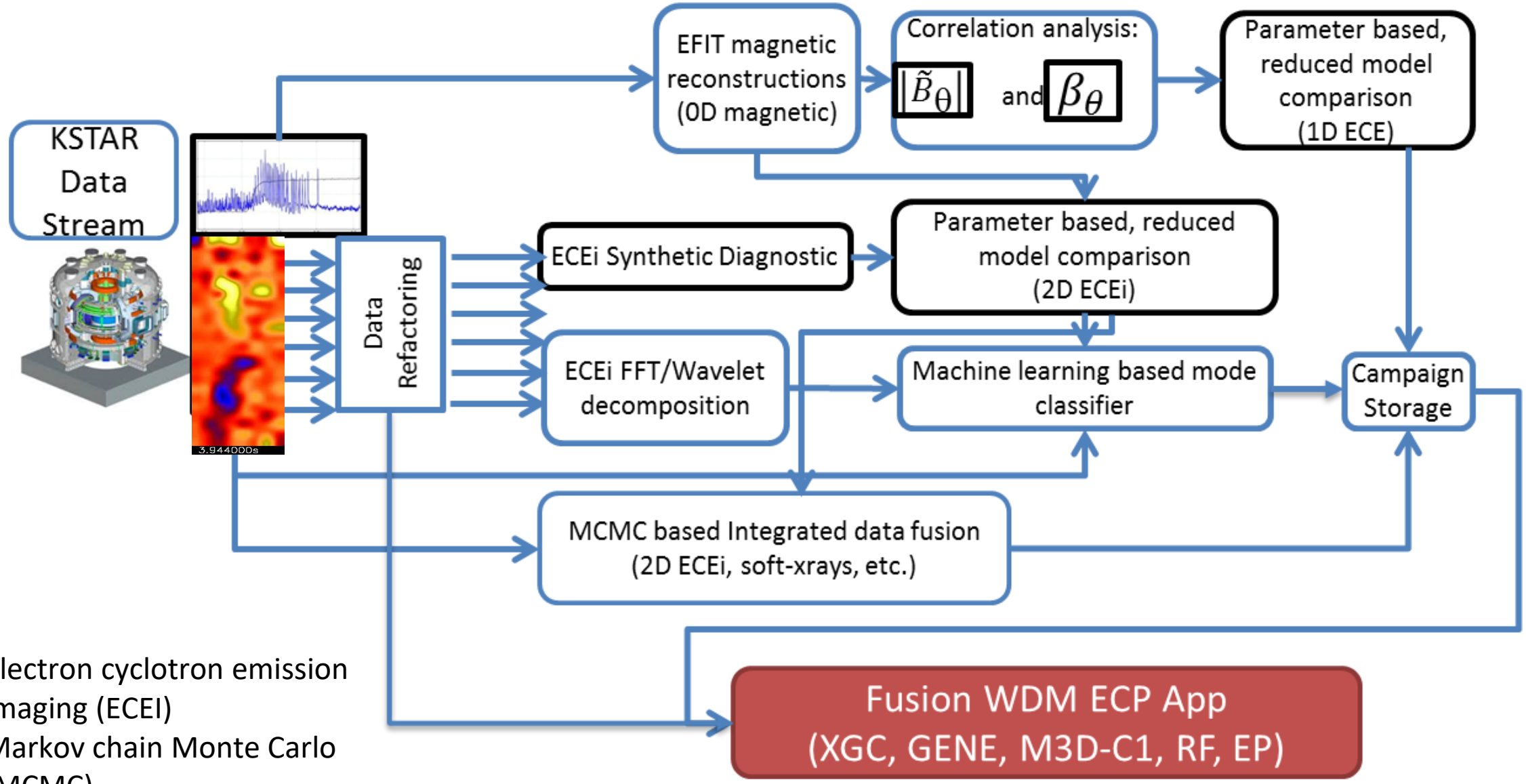




E. Bozdag; D. Peter; M. Lefebvre; D. Komatitsch; J. Tromp; J. Hill; N. Podhorszki; D. Pugmire.
Global adjoint tomography: first-generation model.
Geophysical Journal International 2016 207 (3): 1739-1766
https://doi.org/10.1093/gji/ggw356

New I/O Pattern: Creating PBs/6 hours and processing the data in a single step

Near Real Time Analysis of Experimental Data



Electron cyclotron emission imaging (ECEI)

Markov chain Monte Carlo (MCMC)

New I/O Pattern: Streaming, reducing data from experiments to HPC with ML

The climate I/O tale on Cori@NERSC (in progress)

- 2017: dream about writing 100GB historical data from an E3SM atmospheric simulation
- High-resolution F-case test defined
  - 20 GB data
  - I/O performance on Cori at the time: < 100MB/sec
  - 2019 status of original PIO/pnetcdf: 1 GB/sec
  - 2019 status of PIO/ADIOS: ~5GB/sec
  - ADIOS I/O time is determined and limited by file creation time.
- Lead to dreaming bigger and to definition of Ultra-high resolution case
  - 3 TB data
- 2020 January status of Ultra-high resolution case – *preliminary results*
  - Scorpio/pnetcdf: 2.68 GB/sec
  - Scorpio/ADIOS: 82 GB/sec

https://confluence.exascaleproject.org/pages/viewpage.action?pageId=61905118

Recent results on the E3SM webpage

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

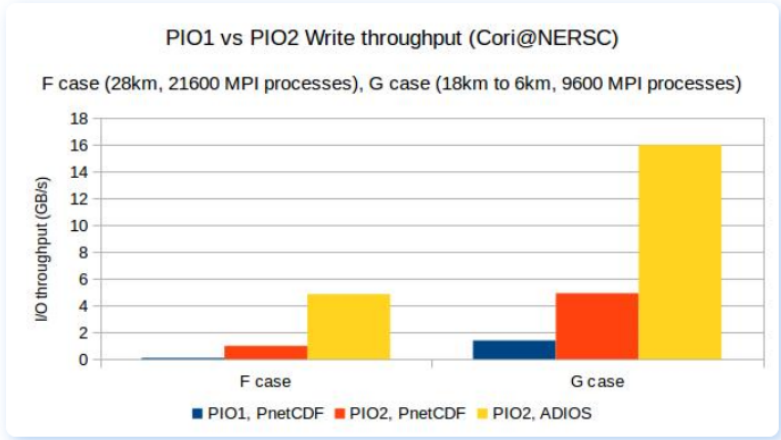
PIO2 + ADIOS = PERFORMANCE IMPROVEMENT

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/

What makes ADIOS "special"

- ADIOS "limits" the choices to perform I/O
  - E.g. you can't have unlimited large arrays for attributes
  - You don't specify stripe size/count to get performance
- ADIOS has **1** **free** parameter which optimizes I/O
  - This is the number of aggregators (explained later in this tutorial)
- For example, in the VPIC I/O benchmark, we don't talk about collective vs. independent I/O, and how we add steps.
  - There is only 1 way to implement this with ADIOS
  - The choice of the engine (implementation) determines the I/O bandwidth along with
    - The total size of the data (per rank)
    - The frequency of I/O

SCIENTISTS REPLICATE A LASER EXPERIMENT FROM INITIAL CONDITIONS

- Doctors have used radiation therapy to treat cancer since 1899.
- Until recently, most radiation used photon beams in the form of x-rays to kill cancer cells
- A team led by HZDR studied ion acceleration driven by high-intensity lasers using TITAN
- They want an understanding of how to optimize the laser-driven ion acceleration process
- The group ran PIConGPU on 8,000 of Titan's GPUs over a 16-hour period.
- They needed good I/O performance since the simulations generated 4 petabytes of data
- At first, the team used state-of-the-art parallel I/O to analyze the data, but time to process a single time step was too long.
- "Taking just a single look into the simulation required 25 minutes of I/O only—no computing," Huebl said. "It was unacceptable for us."
- Using ADIOS they reduced the time to 30 seconds.

https://www.olcf.ornl.gov/2018/07/17/titan-helps-scientists-fine-tune-laser-interactions-to-advance-cancer-treatments/

Schellhammer, Sonja M., Sebastian Gantz, Armin Lühr, Bradley M. Oborn, Michael Bussmann, and Aswin L. Hoffmann. "Experimental verification of magnetic field induced beam deflection and Bragg peak displacement for MR-integrated proton therapy." Medical physics (2018).

Observational: SKA

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

Telescope	Raw Data Rate	Archive Growth
MWA	1.4 TB/hour	5 PB/year
LSST	1.5 TB/hour	6 PB/year
ASKAP	9 TB/hour	5.5 PB/year
SKA1-LOW	1,400 TB/hour	150 PB/year

Science impact

Ran a simulation on Summit to generate SKA-LOW data which required extreme I/O performance

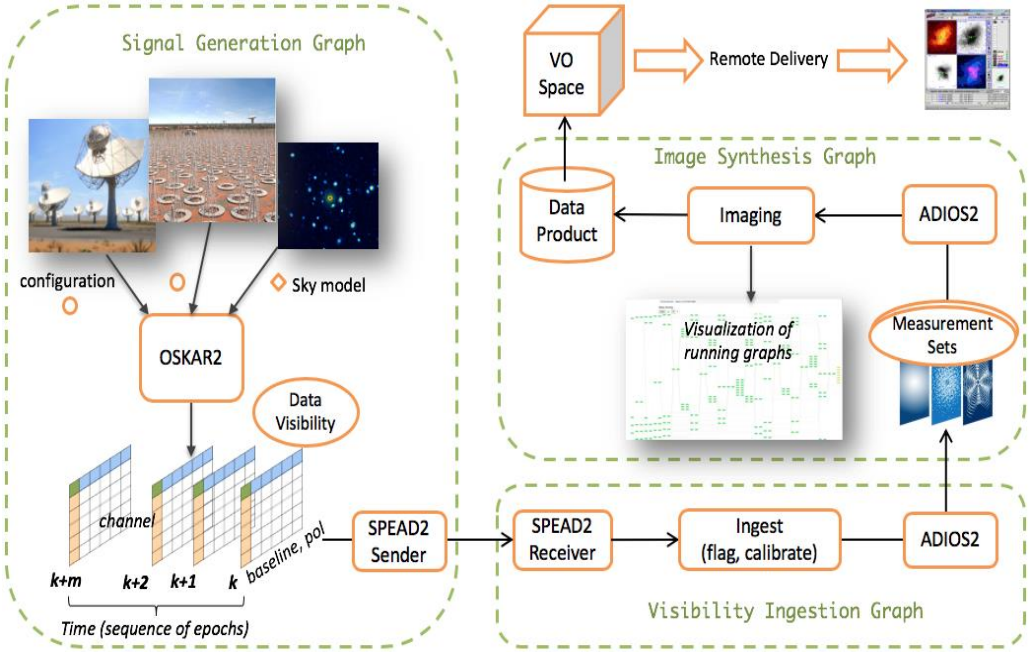
Significance and Impact

The data output of the SKA is limited by the achievable I/O bandwidth, with over 25 EB per year being generated.

Results showed that SKA-LOW data can be processed on a Summit scale machine

Technical Approach

- The full SKA will use a million antennas to enable astronomers to monitor the sky in unprecedented detail and survey the entire sky much faster than any system currently in existence
- One of SKA's greatest challenges is in the ability to move, process, and store data, without losing information
- Used 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
- Used ADIOS for the I/O inside of DALiuGE
- The test run used a cosmological simulation of the early Universe at a time known as the Epoch of Reionisation, when the first stars and galaxies formed and became visible



New I/O patterns for "extreme performance": PB/s requiring Near Real Time processing

SKA

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

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

SHARE ARTICLE: TWITTER FACEBOOK

TAGS

BIG DATA

OAK RIDGE NATIONAL LABORATORY

SHANGHAI ASTRONOMICAL OBSERVATORY

SHAO

SKA-LOW

SQUARE KILOMETRE ARRAY

SUMMIT

SUPERCOMPUTING

October 22, 2019

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.

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.

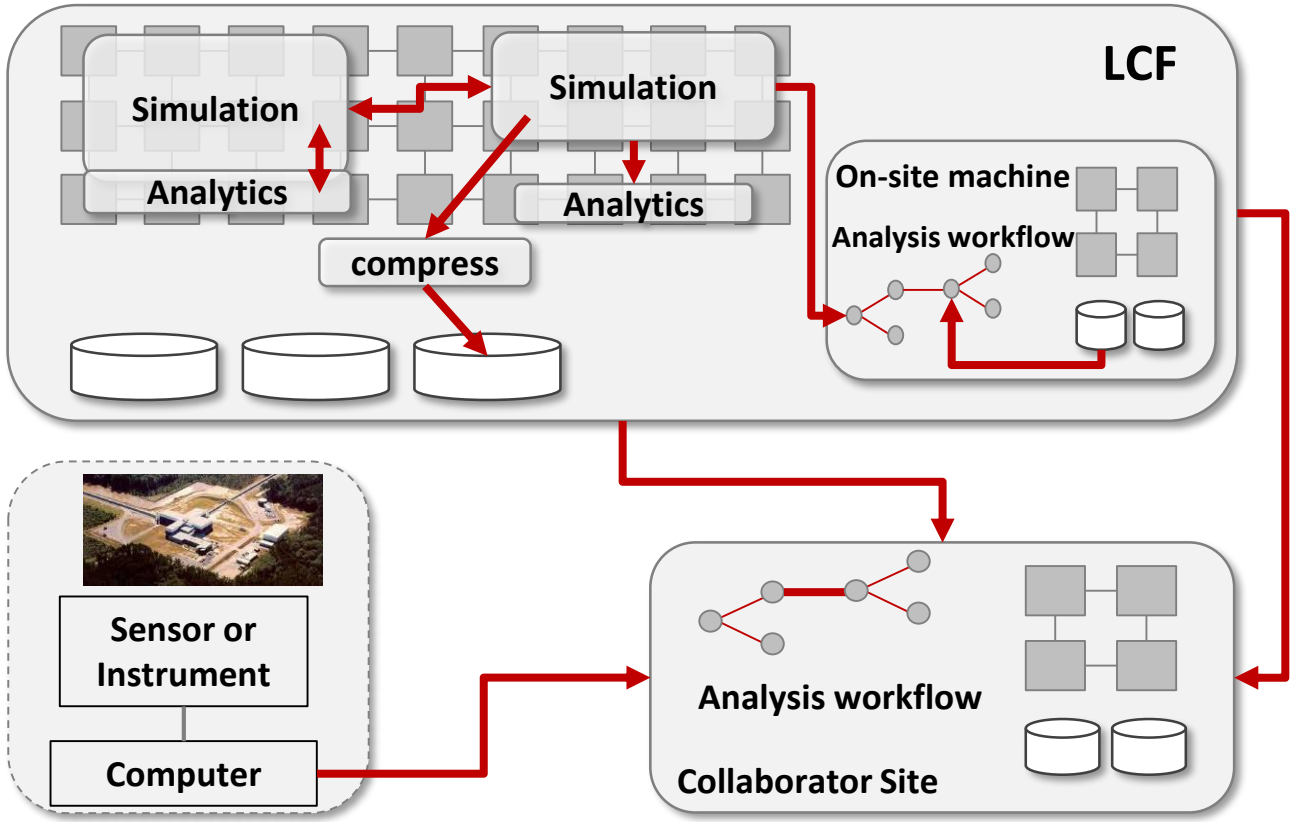
"So we used another piece of software written by ICRAR, called the Data Activated Flow Graph Engine (DALiuGE), to distribute one of these simulators to each of the 27,648 graphics processing units that make up Summit.

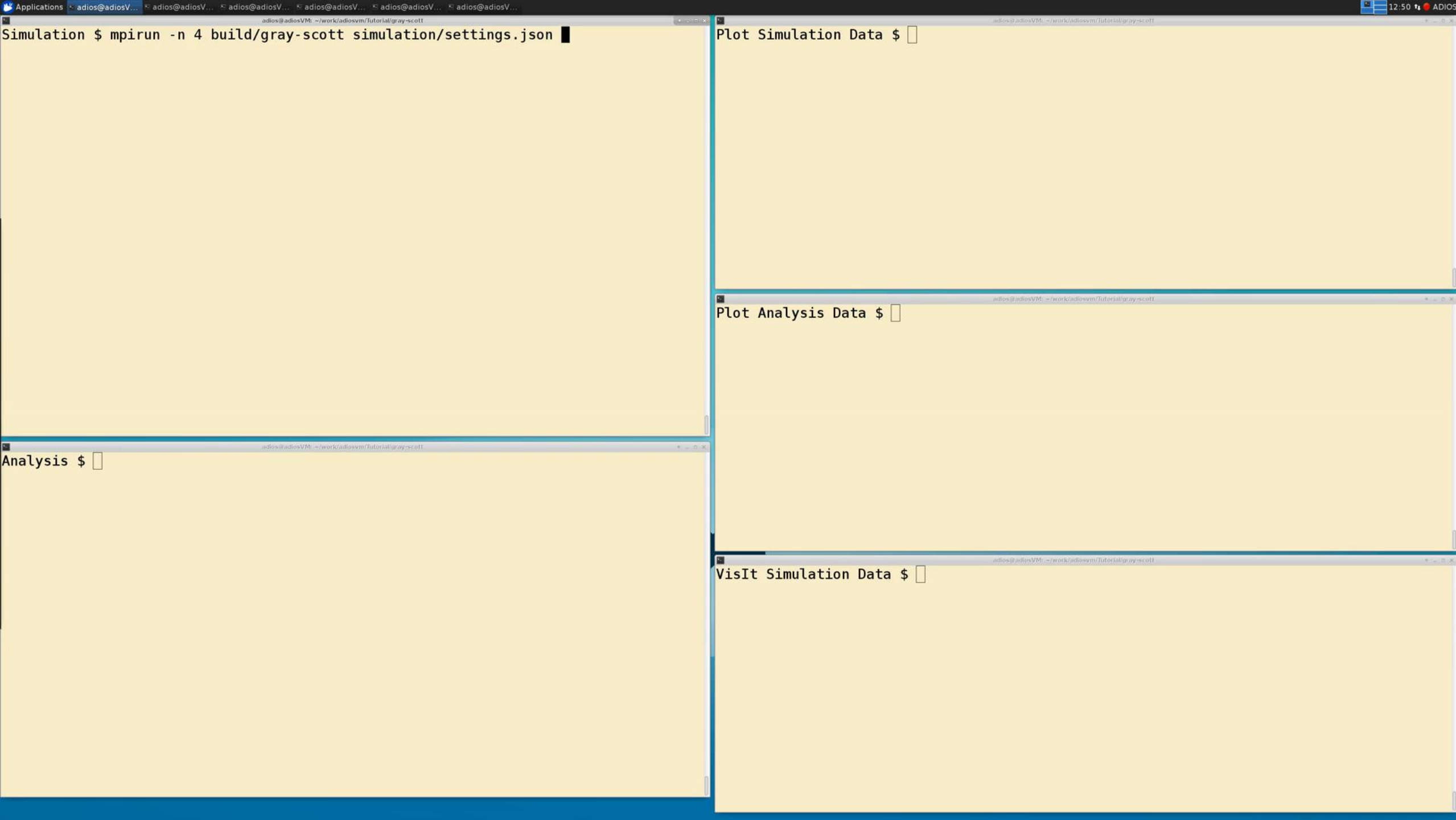
"We also used the Adaptable IO System (ADIOS), developed at the Oak Ridge National Laboratory, to resolve a bottleneck caused by trying to process so much data at the same time."

The test run used a cosmological simulation of the early Universe at a time

Our Vision: Enabling High Performance pub/sub I/O

- Create a high performance I/O abstraction to allow for on-line memory/file data subscription service
- Create an abstraction for self-describing I/O for files, streams, etc.
- Work at all scales (laptops, desktops, clusters, exascale-platforms)
- Scale out to the number of nodes, sites, timesteps, variables, etc.




Applications
adios@adiosV...
adios@adiosV...
adios@adiosV...
adios@adiosV...
adios@adiosV...
adios@adiosV...
12:50
ADIOS
adios@adiosVM: ~/work/adiosvm/Tutorial/gray-scott
Simulation $ mpirun -n 4 build/gray-scott simulation/settings.json
adios@adiosVM: ~/work/adiosvm/Tutorial/gray-scott
Plot Simulation Data $
adios@adiosVM: ~/work/adiosvm/Tutorial/gray-scott
Plot Analysis Data $
adios@adiosVM: ~/work/adiosvm/Tutorial/gray-scott
Analysis $
adios@adiosVM: ~/work/adiosvm/Tutorial/gray-scott
VisIt Simulation Data $

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: High-Performance Publisher/Subscriber I/O framework

Vision

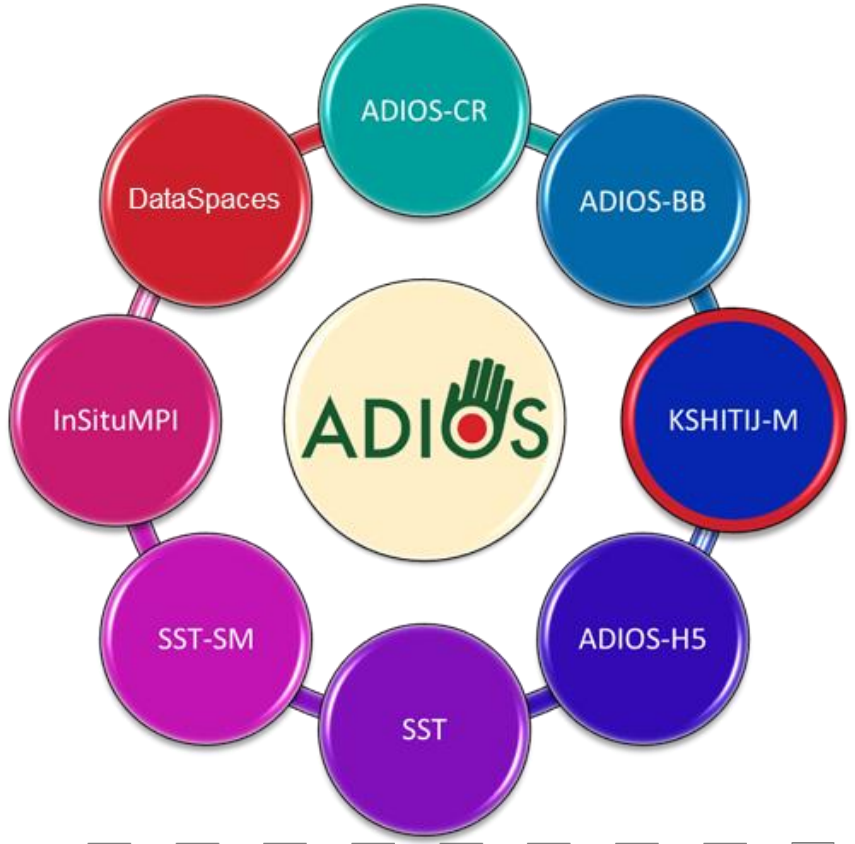
Create a high performance I/O abstraction to allow for on-line/off-line memory/file data subscription service

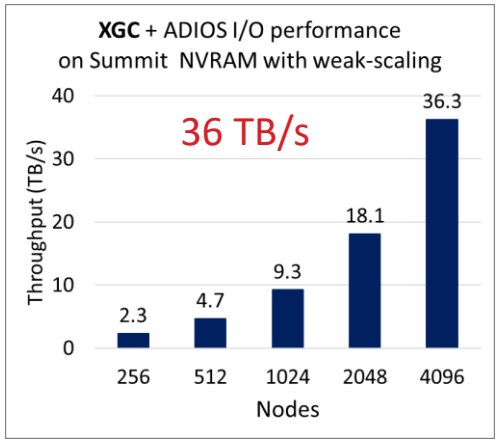
Create a sustainable solution to work with multi-tier storage and memory systems

Research Details

- Declarative, publish/subscribe API is separated from the I/O strategy and use of multi-tier storage
- Multiple implementations (engines) provide functionality and performance in different use cases
- Rigorous testing ensures portability
- Data reduction techniques are incorporated to decrease storage cost

https://github.com/ornladios/ADIOS2





XGC writing particles on Summit's local NVRAM: 18.3 TB in 0.5 seconds on 4K nodes

ADIOS Useful Information and Common tools

- ADIOS documentation: https://adios2.readthedocs.io/en/latest/index.html
- 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: https://github.com/ornladios/ADIOS2-Examples
- 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 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

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
...
```

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 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

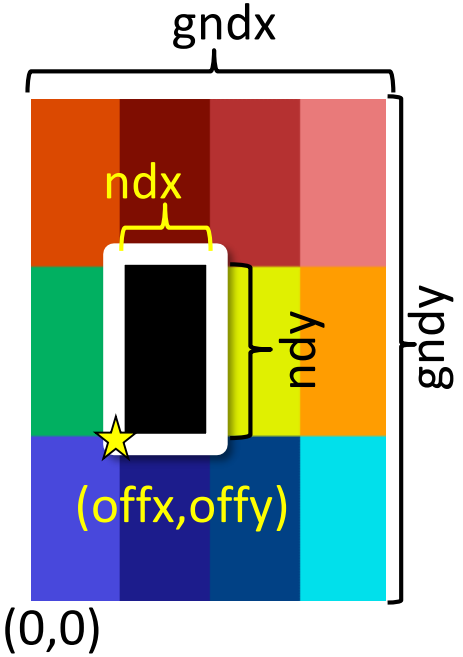
IO object

- 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

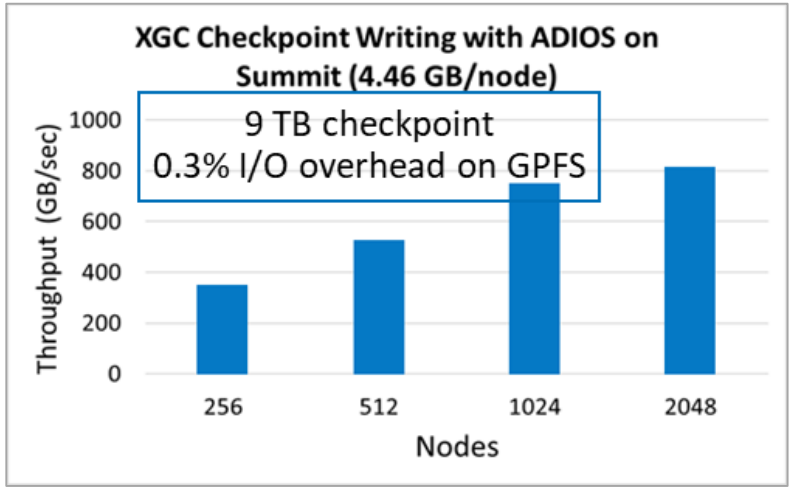
```
adios2::Engine writer =
io.Open("checkpoint.bp", adios2::Mode::Write);


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


writer.Close()
```

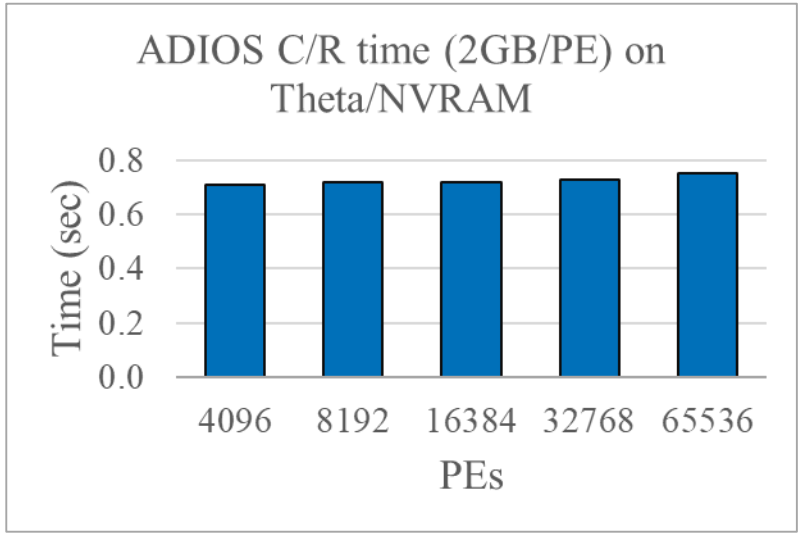
Reading is similar, but we can read from any number of procs

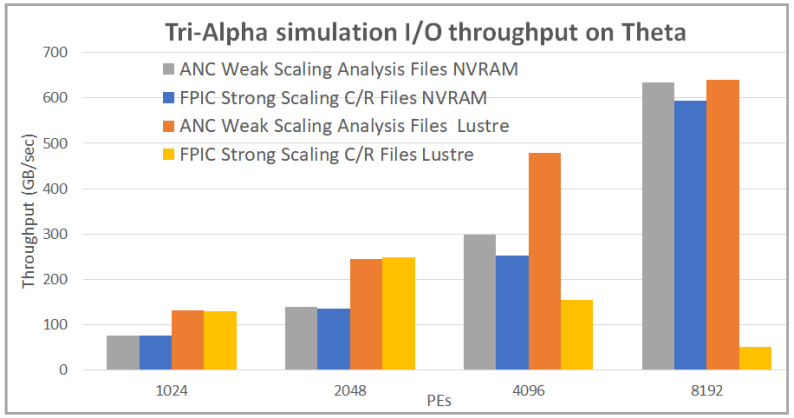
```
adios2::IO io = adios.DeclareIO("CheckpointRestart");

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



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

                                          <-- Reserve memory for T before this

reader.Get(*vT, T.data());



reader.Close()
```

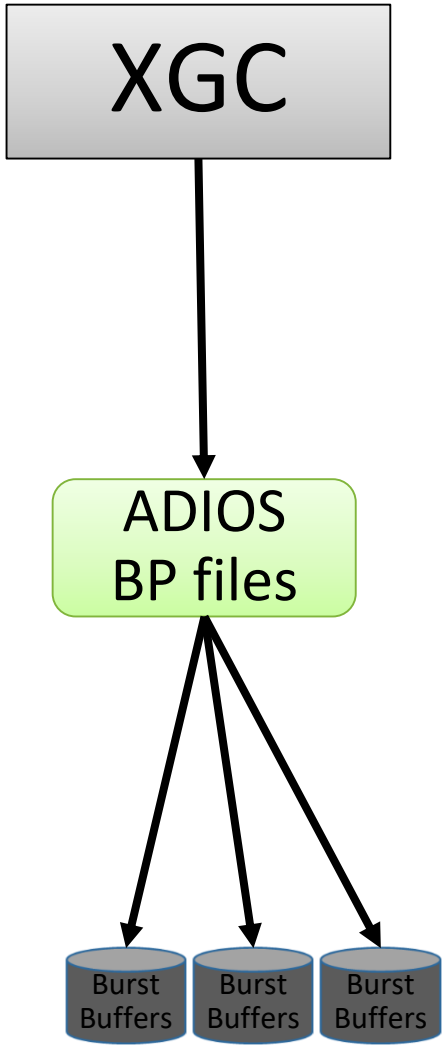
ADIOS APIs for self describing data output for C/R to NVRAM

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









Analysis/visualization data

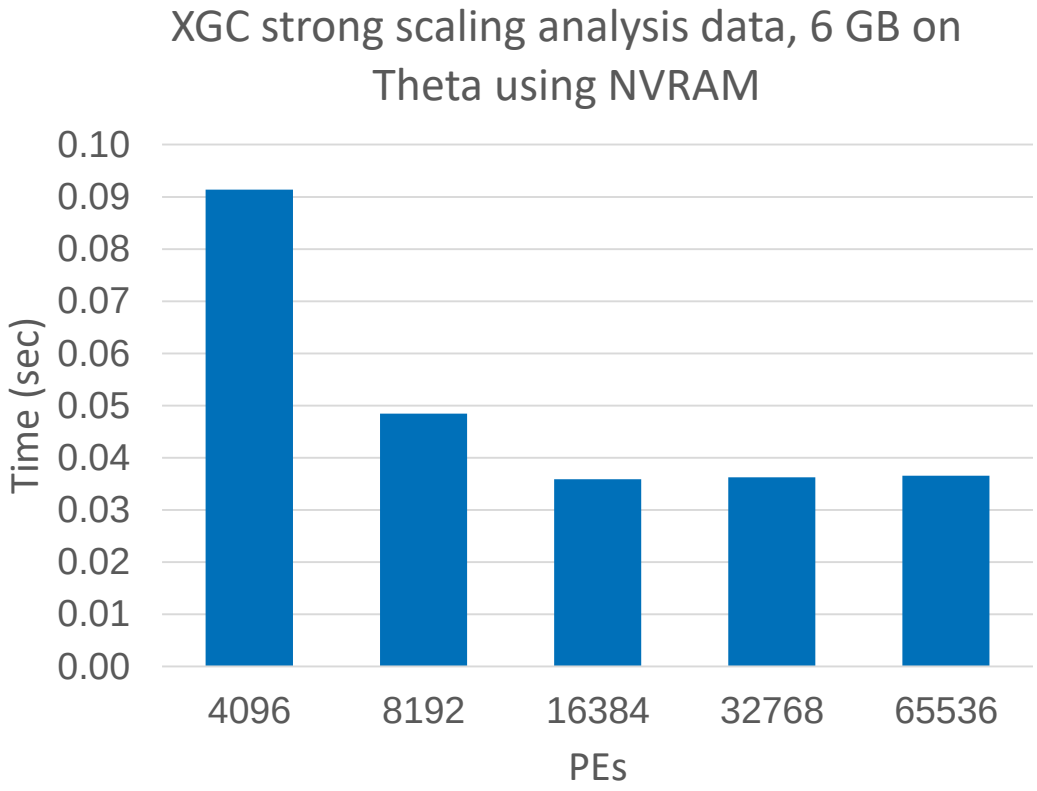
```
adios2::IO io = adios.DeclareIO("Analysis_Data");

if (!io.InConfigFile()) {

     io.SetEngine("BP4");

}

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

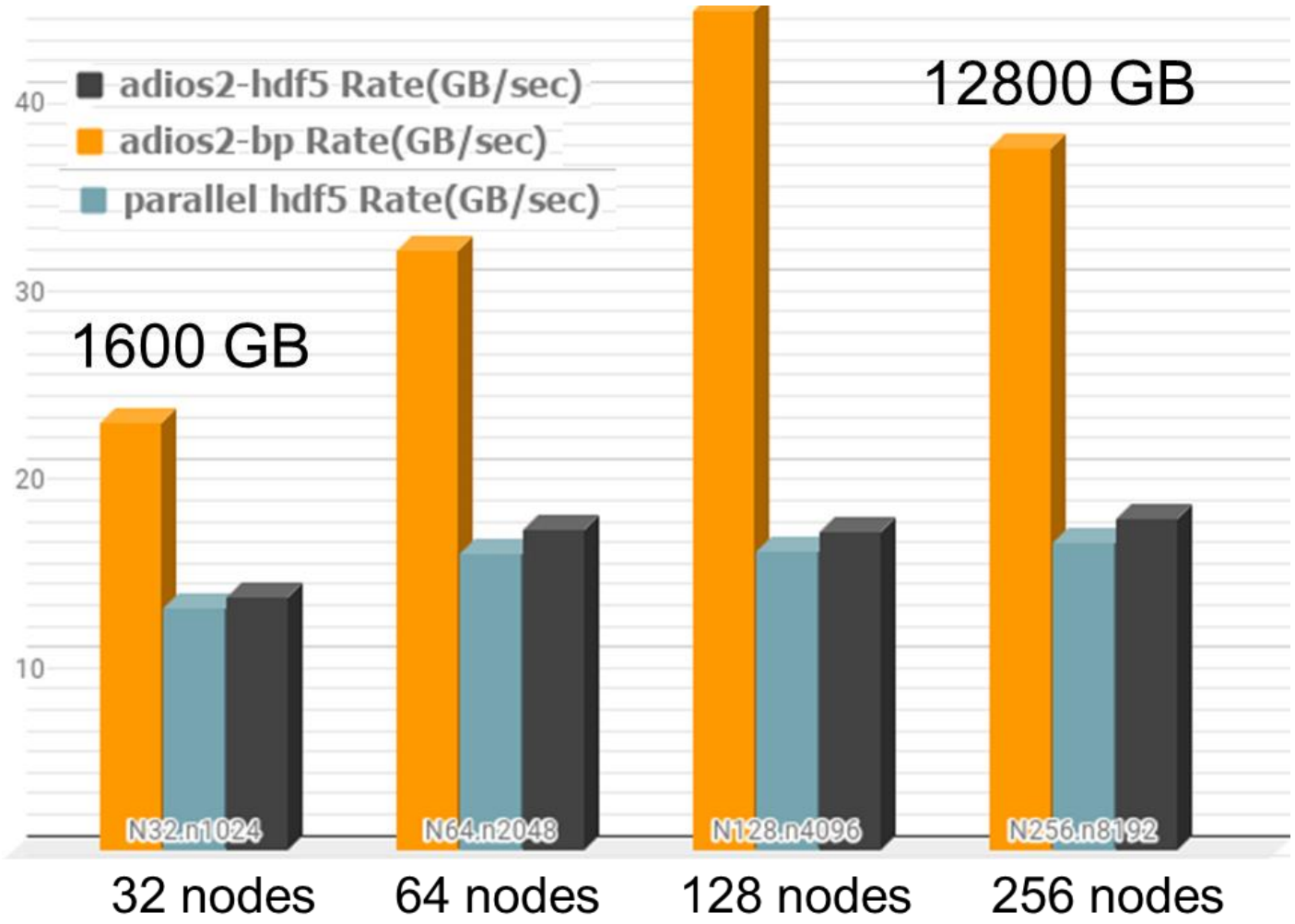
ADIOS engines – change from BP4 to HDF5

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

Change Engine name

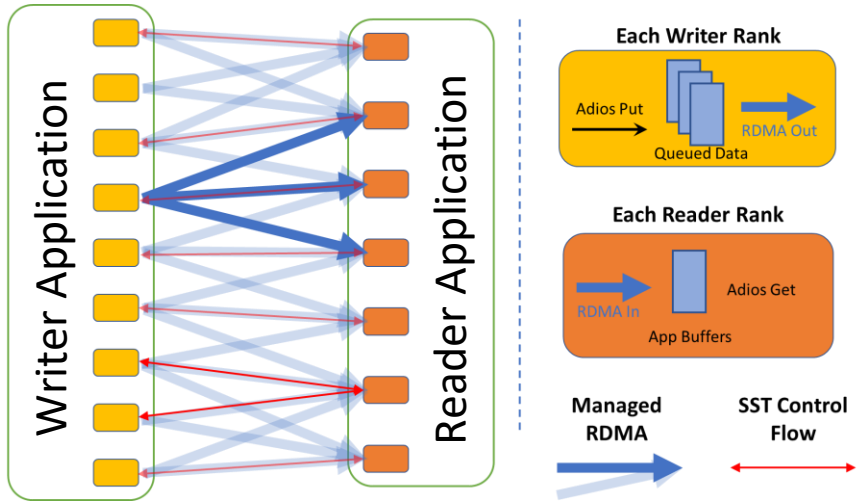
```
1. <io name="SimulationOutput">
2.         <engine type="HDF5"/>
3. </io>
```

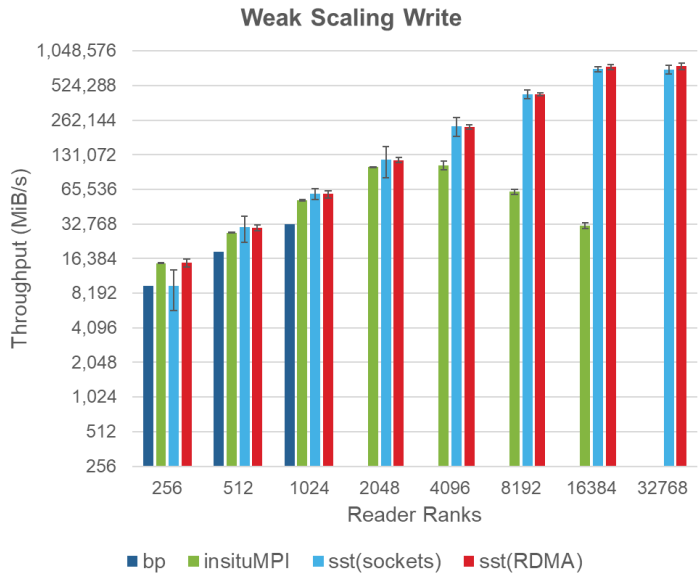
Cori + Lustre, for the heat equation



Sustainable Staging Transport (SST)

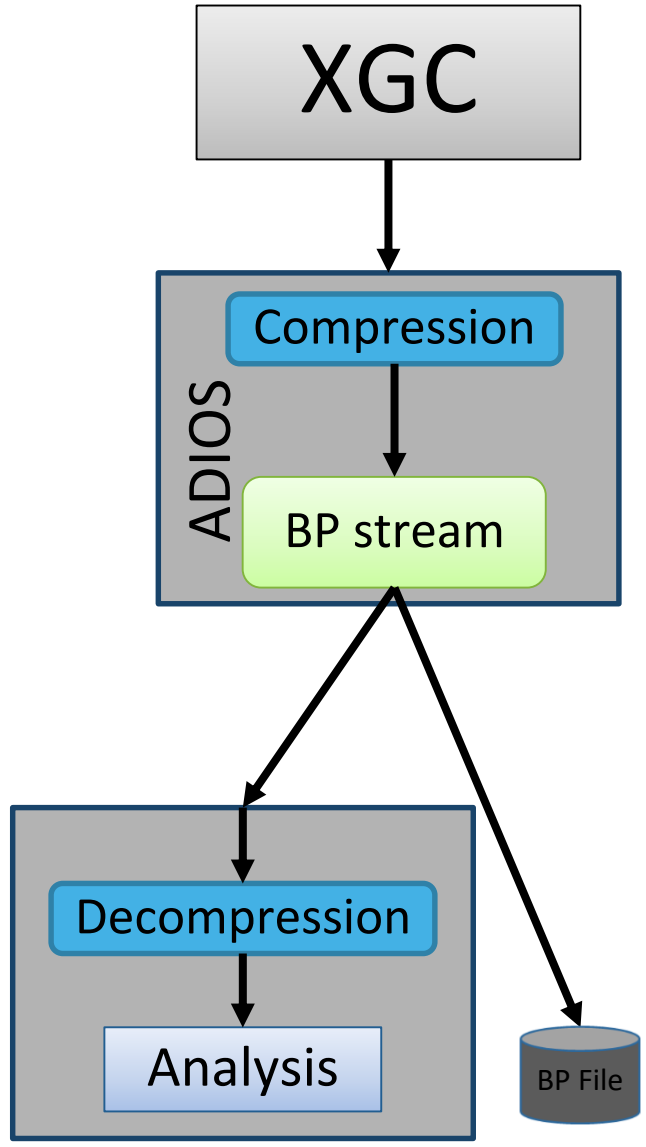
- Provide a direct and high performance connection between an ADIOS writer and one or more readers
  - Communication between separate applications
  - Multiple transport mechanisms using libfabric
- Efficiently handle data transfer and necessary refactoring
- Avoid interfering with original computation as much as possible
- Support dynamic behavior (readers come and go)
- Be robust to client failure and resource availability changes





ADIOS interface to data reduction (compression/decompression)

- Compression & Decompression methods inside ADIOS are 1st to 3rd party software components
  - Supported routines are: NONE, type-conversion, ZFP, SZ, MGARD, BZIP2, blosc
- Each variable can be compressed using different compression mechanisms
- Decompression is done automatically by the ADIOS complaint engine
- Next version: support for indexing/queries



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)

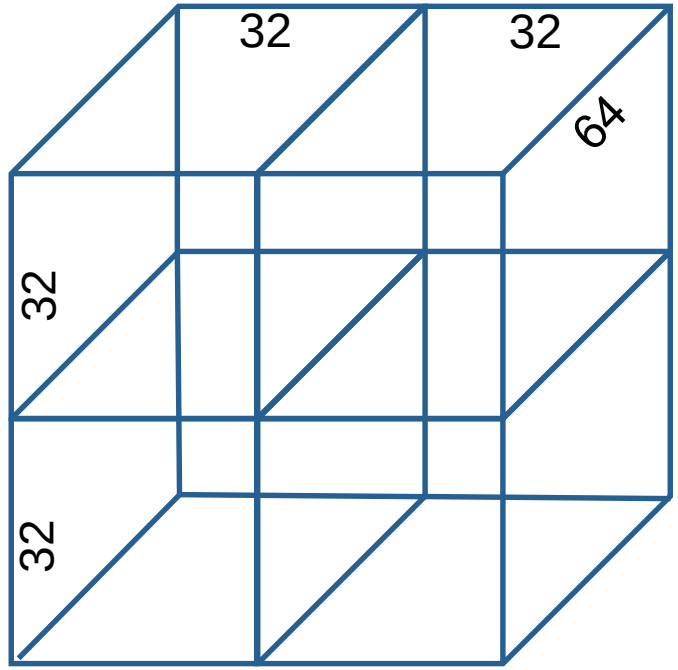
- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization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

Wrap-up

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

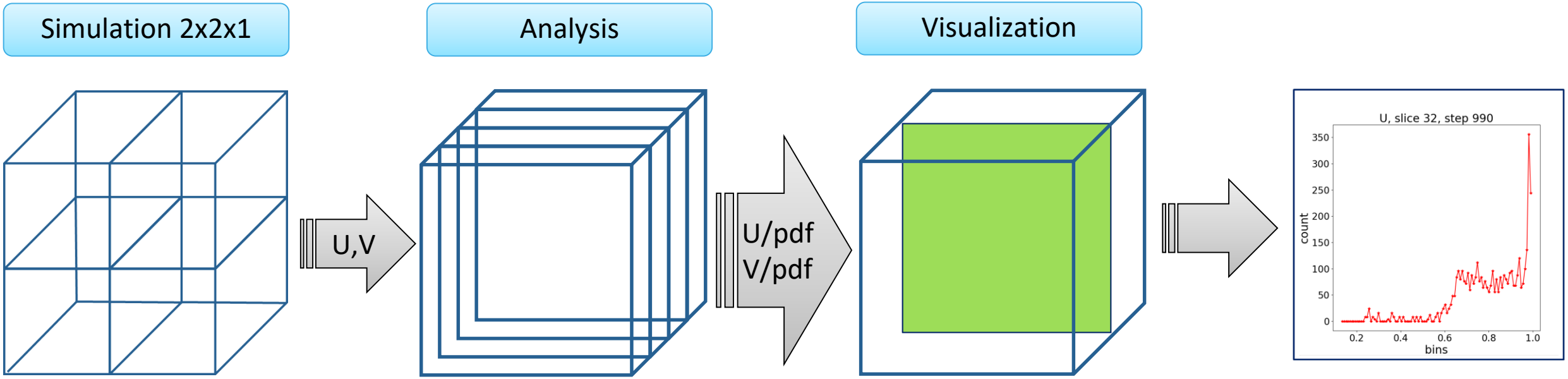
Gray-Scott Example

- 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
  - 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

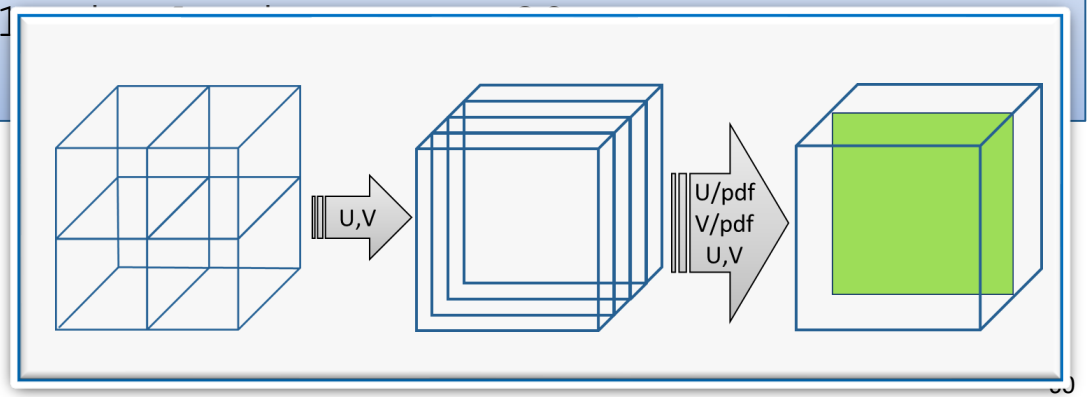


Goal by the end of the day: Running the example in situ

```
$ 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

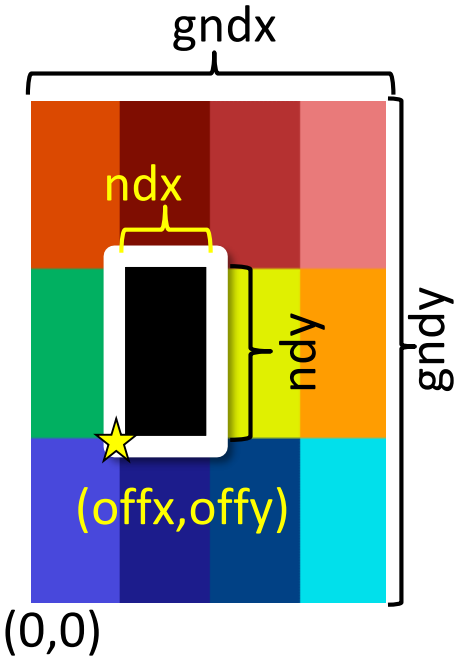
```
$ cd ~/ADIOS2-Examples/
$ mkdir -o build-cmake
$ cd build-cmake
$ cmake \
  -DCMAKE_PREFIX_PATH=/home/adios/Tutorial \
  -DADIOS2_DIR=/opt/adios2   \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo   \
  ..
$ make -j 4
$ make install
$ export PATH=$PATH:/home/adios/Tutorial/bin
$ cd /home/adios/Tutorial/share/adios2-examples/gray-scott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

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)

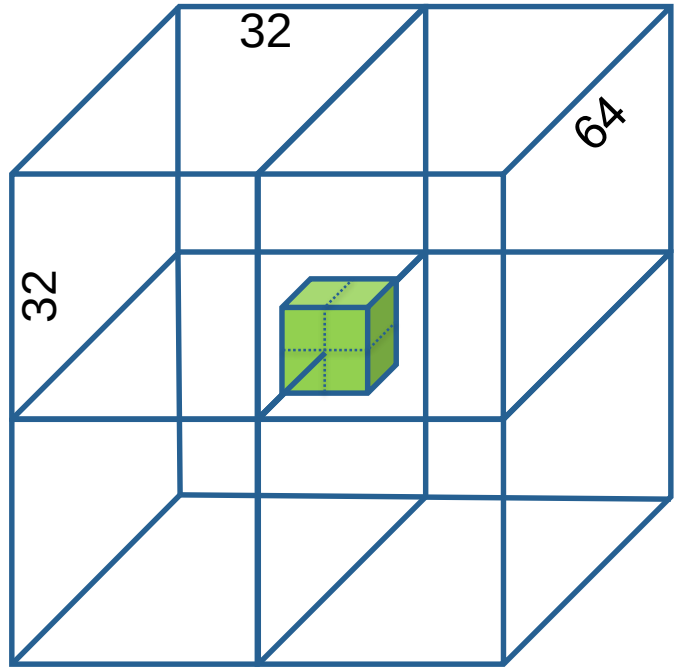
```
$ bpls -D gs.bp U
 double   U      100*{64, 64, 64} = 0.0907898 / 1
         step  0:
           block 0: [ 0:63,  0:31,  0:31] = 0.104002 / 1
           block 1: [ 0:63, 32:63,  0:31] = 0.104002 / 1
           block 2: [ 0:63,  0:31, 32:63] = 0.104002 / 1
           block 3: [ 0:63, 32:63, 32:63]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

Engine types

BP4

HDF5

SST

InSituMPI

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
SST
InSituMPI
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)

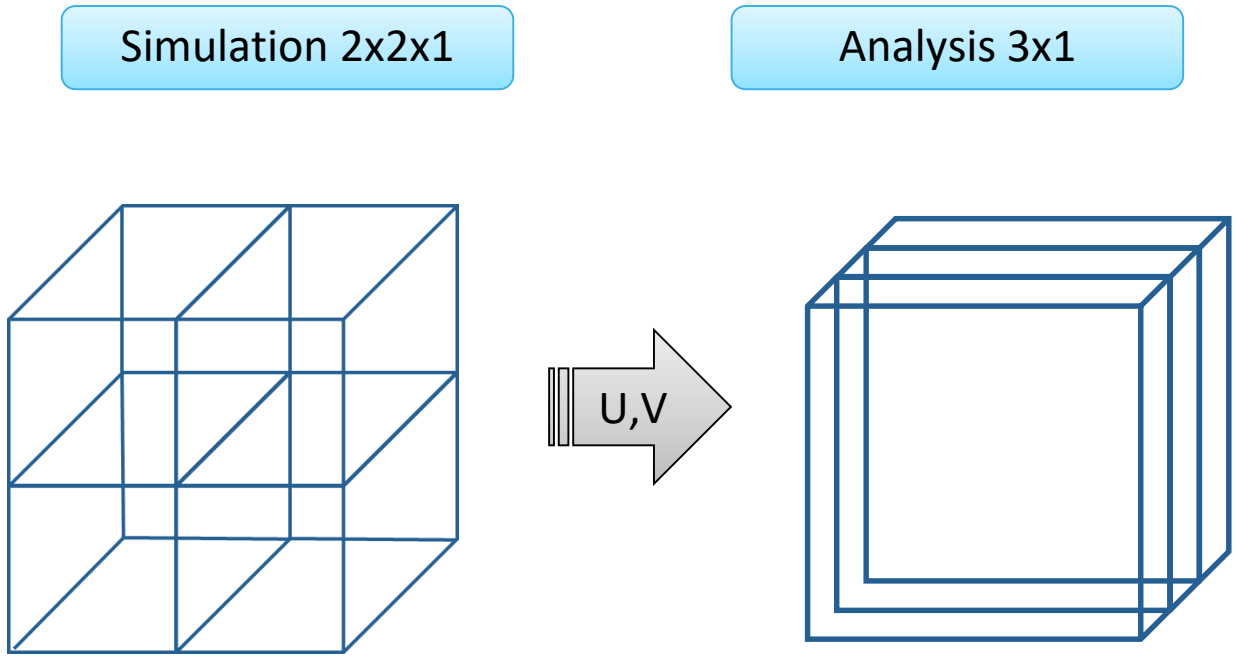
- **Lecture**: Introduction to "data staging" for in situ analysis and code coupling
- **Hands-on:** Create a simple pipeline using the MiniApp that computes, and visualizes a derived variable using data staging
- **Hands-on**: Add data reduction to the pipeline
- **Demonstration**: In situ visualization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

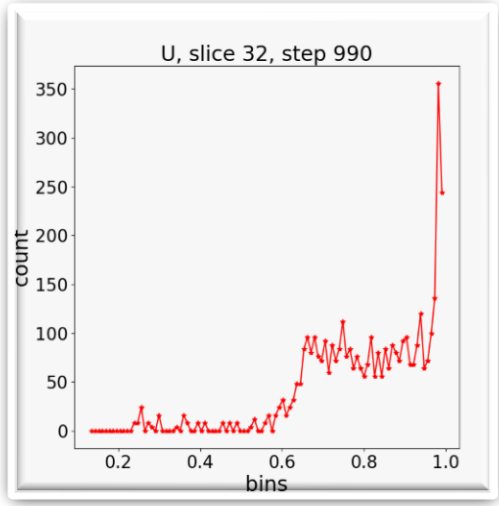
$ bpls -l pdf.**h5** -D U/pdf

```
  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
```

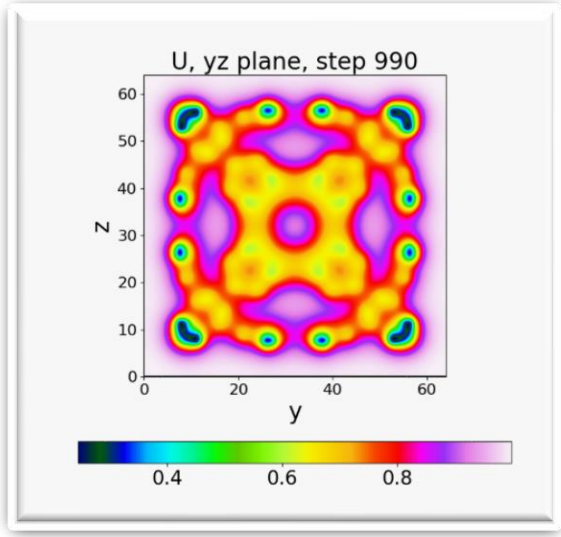
HDF5 format: array seen as one big block

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
SST
InSituMPI
DataMan

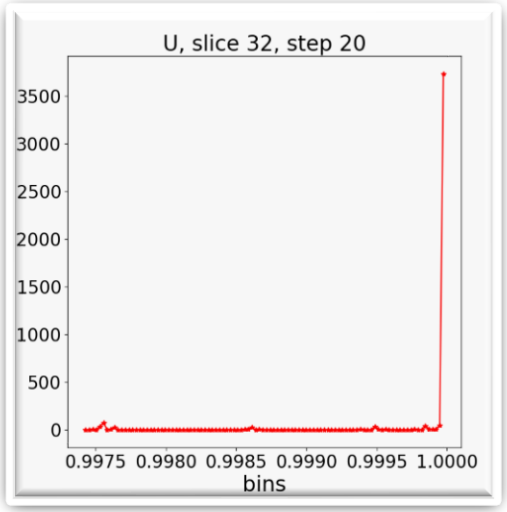
```
   <!--================================
       Configuration for PDF calc and PDF Plot
     ================================-->

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

</adios-config>
```

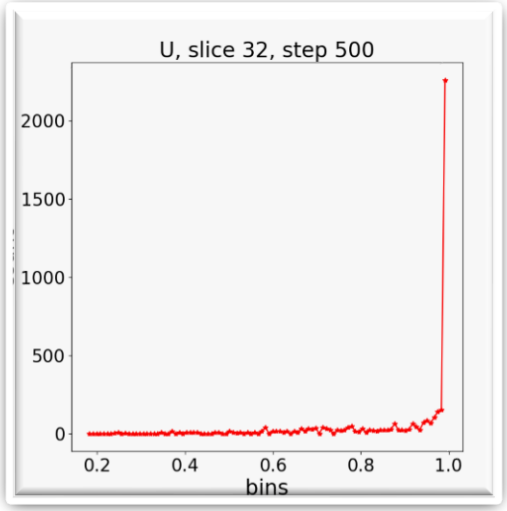
Run the code again if you removed gs.h5

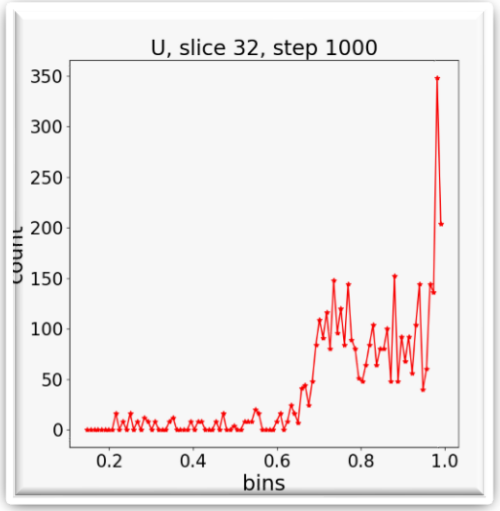
```
$ cd ~/Tutorial/share/adios2-examples/gray-scott
$ mpirun -n 4 adios2-gray-scott settings-files.json
Simulation writes data using engine type:              HDF5
=========================================
grid:               64x64x64
steps:              1000
plotgap:            10
F:              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

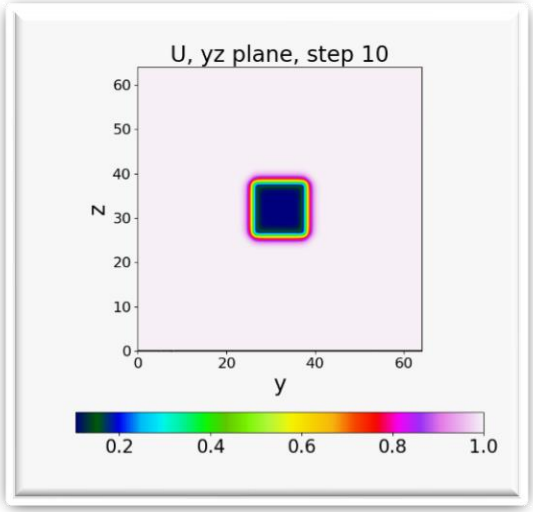


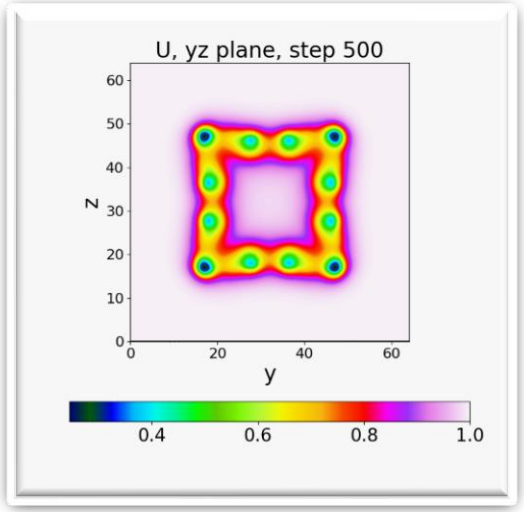


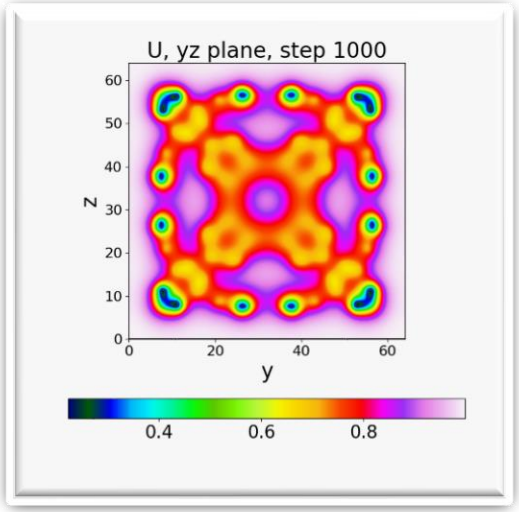


Run the plotting script with file I/O

```
$ 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

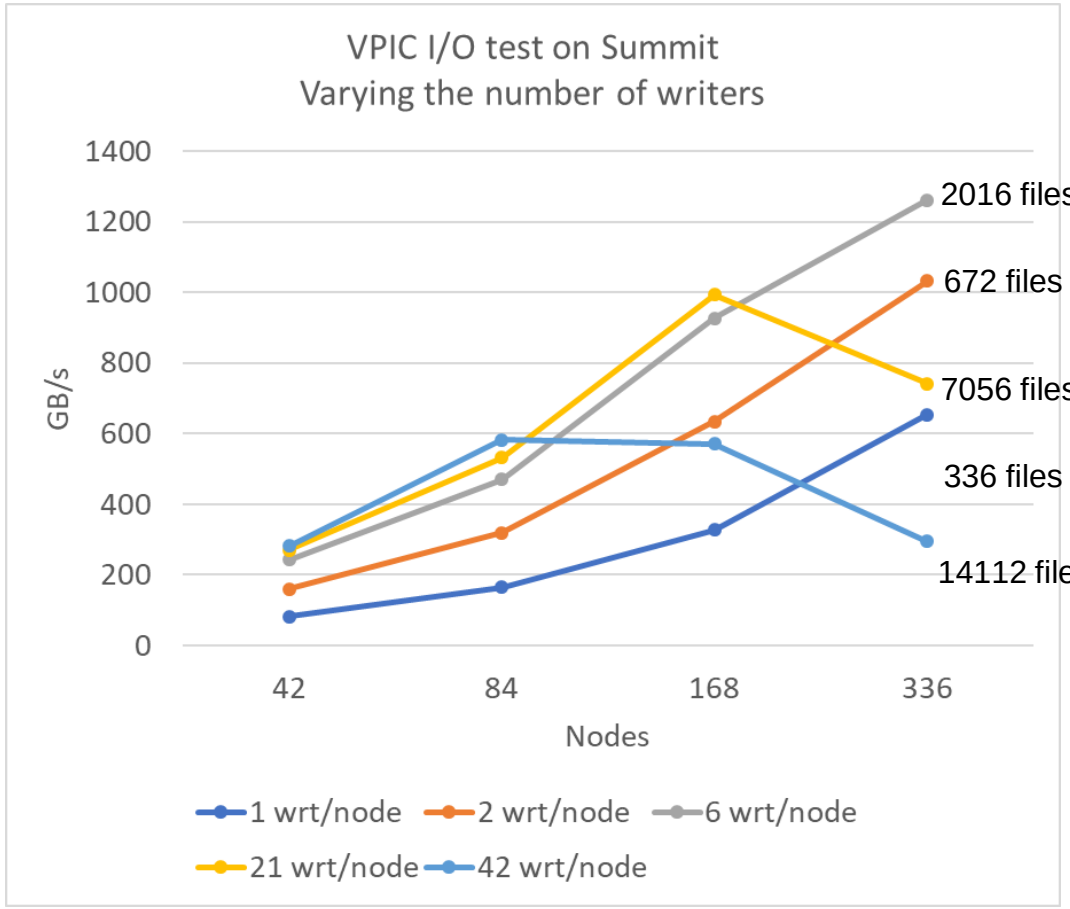
ADIOS Scaling for large parallel file systems

- 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
    - 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 option: SubStreams

```
<io name="SimulationOutput">
  <engine type="BP4">
    <parameter key="SubStreams" 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



Application	Nodes/GPUS	Data Size per step	I/O speed	ADIOS SubStreams
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)

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

Data compression

- Two types of compression techniques integrated into our software
  - Lossless
  - Lossy

Survey's of compression techniques

Reference	Year	Classification
Holtz (1999)	1997	Shannon entropy, Huffman code, LZ code and Self-Learning Autopsy data trees
Kimura and Latifi (2005)	2005	Compression techniques for WSN
Sudhakar et al. (2005)	2005	Ten wavelet coding techniques
Peng et al. (2005)	2005	3D mesh compression techniques
Chew and Ang (2008)	2008	Eight image compression algorithms
Smith (2010)	2010	Lossy compression
Lakshminarasimhan (2011)	2011	Compressing the incompressible with ISABELA: in-situ reduction of spatio-temporal data
Sridevi et al. (2012)	2012	Short survey on medical image compression
Srisooksai et al. (2012)	2012	Distributed data compression and local data compression
Hosseini (2012)	2012	Huffman, RLE, LZ and Arithmetic coding with its applications
Rehman et al. (2014)	2014	Image compression techniques
Li et al. (2014)	2014	Mesh compression methods with random accessibility
Ian et al. (2014)	2014	Various standards for remote sensing data compression
ZainEldin et al. (2015)	2015	Image compression techniques in WMSN
Rekha and Samundiswary (2015)	2015	Low power adaptive image compression techniques in WSN
Kavitha (2016)	2016	RLE, LZW and Huffman coding, Transform coding, DCT and DWT
Tarek et al. (2016)	2016	DCT and DWT image compression techniques
Mittal and Vetter (2016)	2016	Compression in cache and main memory systems
Rana and Thakur (2017)	2017	Computer vision applications
Patel and Chaudhary (2017)	2017	DCT, DWT and hybrid approaches in WMSN

Data compression

- A form of data reduction
- ~40 years old (LZ77, 1977), ~70 years (Shannon's information theory, 1948)
- Systematically (almost) used for files (e.g. GZIP), digital photos (e.g. JPEG), movies (e.g. MP4), music (e.g. MP3)
- Very effective on images
- A priori compression algorithms are generic and applicable to many applications
  - In practice they are combined and optimized for specific usages

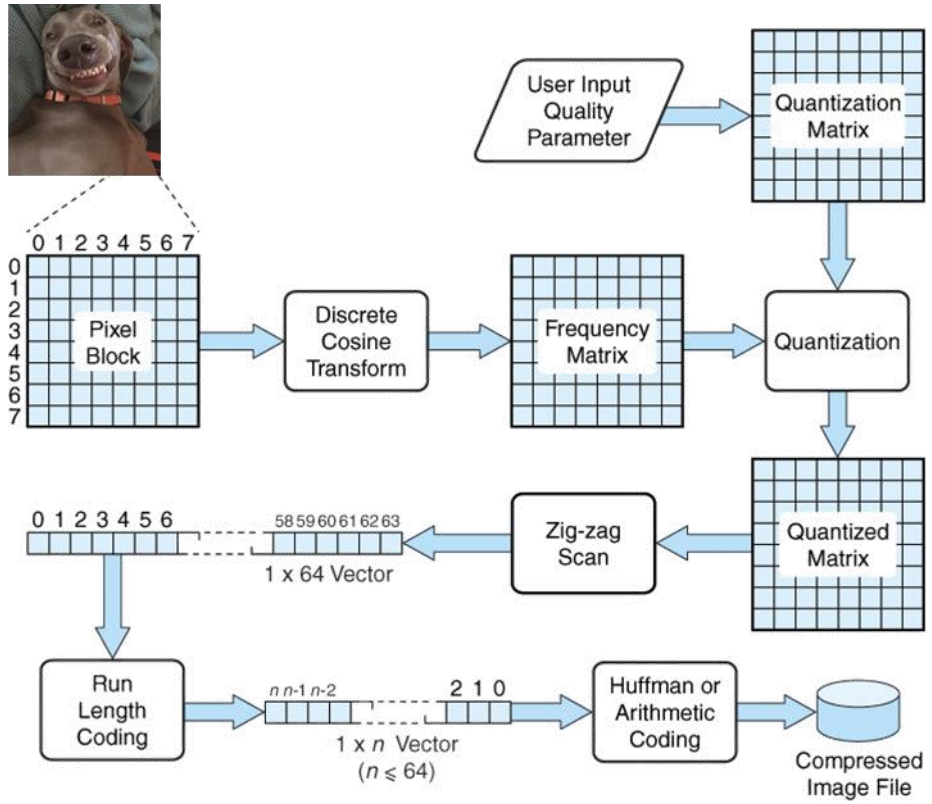
From F. Cappello

Common compression/reduction technique

- Lossy
  - SZ - ANL
  - ZFP - LLNL
  - MGARD – Brown-ORNL
  - Reduce-constrain-particles – LANL
  - ALACRITY – NCSU-ORNL
  - PCA
  - ISABELLA – NCSU-ORNL
- Image/movies
  - Jpeg, mpeg, …

- Lossless
  - LZ4
  - SZIP
  - GZIP

JPEG compression



Compression

- Compression is achieved by removing data redundancy while preserving information content.
- The information content of a group of bytes (a message) is its *entropy*.
  - Data with low entropy permit a larger compression ratio than data with high entropy.
- Entropy, *H*, is a function of symbol frequency.
- Entropy is the weighted average of the number of bits required to encode the symbols of a message:

$$H = -P(x) \times \log_2 P(x_i)$$

- The entropy of the entire message is the sum of the individual symbol entropies.

$$\sum -P(x) \times \log_2 P(x_i)$$

- The average redundancy for each character in a message of length *l* is given by:

$$\sum P(x) \times l_i - \sum -P(x) \times \log_2 P(x_i)$$

Lossless compression

- A class of data compression that allows the original data to be perfectly reconstructed from the compressed data
- Redundant data is removed in compression and added during decompression
- Most algorithms do two things in the sequence
  - Generate a statistical model for the input data
  - Use the model to map the input data to bit sequences in such a way that “probable” data will produce shorter output than “improbable data”
- Common method are run-length encoding, Huffman encoding, dictionary-based encoding

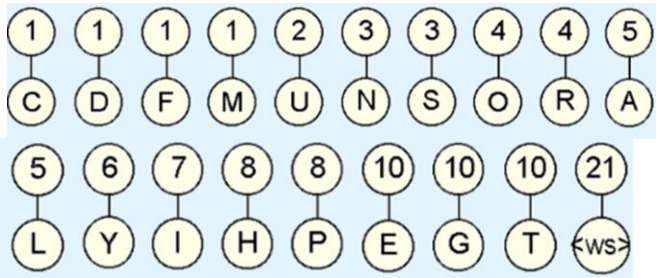
Statistical Encoding (homepage.cs.uiowa.edu/~ghosh/7A.pp)

- The entropy metric just described forms the basis for statistical data compression
- Two widely-used statistical coding algorithms are *Huffman coding* and *arithmetic coding*
- Huffman coding builds a binary tree from the letter frequencies in the message
  - The binary symbols for each character are read directly from the tree
- Symbols with the highest frequencies end up at the top of the tree, and result in the shortest codes

First build the tree by counting the occurrences of each symbol in the text to be encoded

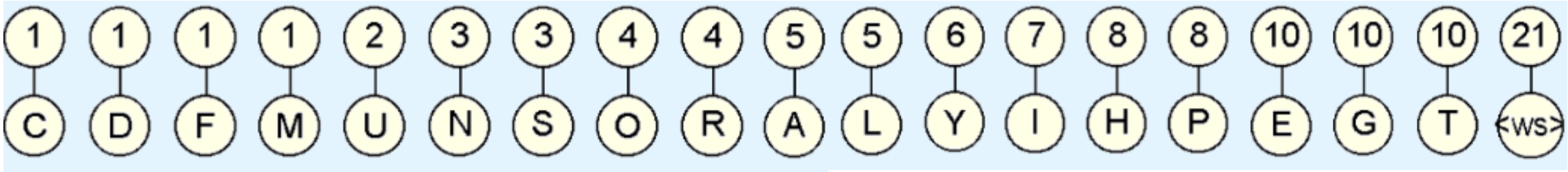
```
HIGGLETY PIGGLTY POP
THE DOG HAS EATEN THE MOP
THE PIGS IN A HURRY
      THE CATS IN A FLURRY
HIGGLETY PIGGLTY POP
```

Letter	Count	Letter	Count
A	5	N	3
C	1	O	4
D	1	P	8
E	10	R	4
F	1	S	3
G	10	T	10
H	8	U	2
I	7	Y	6
L	5	<ws>	21
M	1		

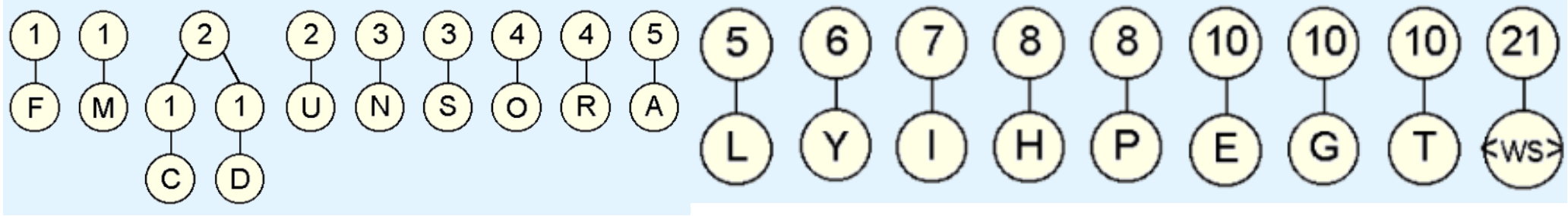


Place the letters and their frequencies into a forest of trees that each have two nodes: one for the letter, and one for its frequency.

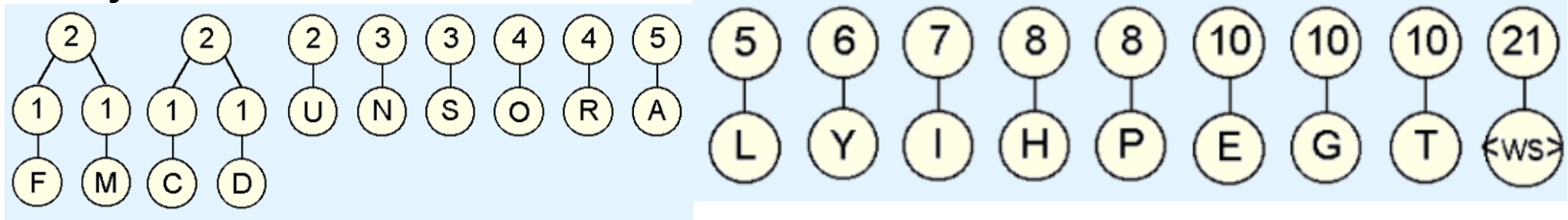
- Place the letters and their frequencies into a forest of trees that each have two nodes: one for the letter, and one for its frequency



- Start building the tree by joining the nodes having the two lowest frequencies

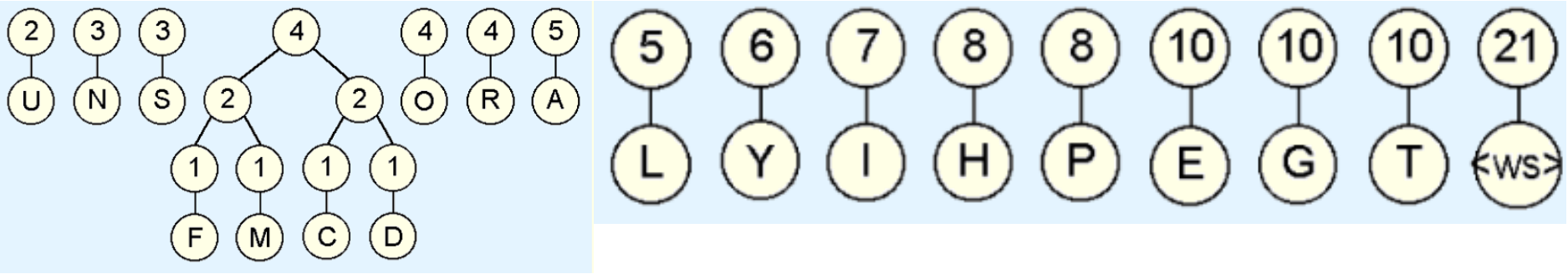


- Next join the nodes with two lowest frequencies

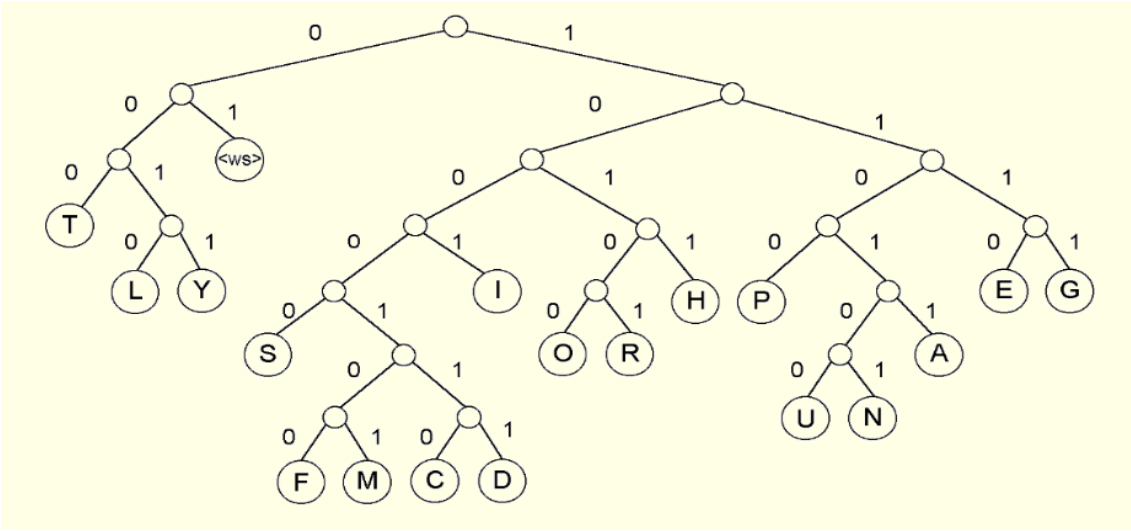


Statistical coding

- Repeat



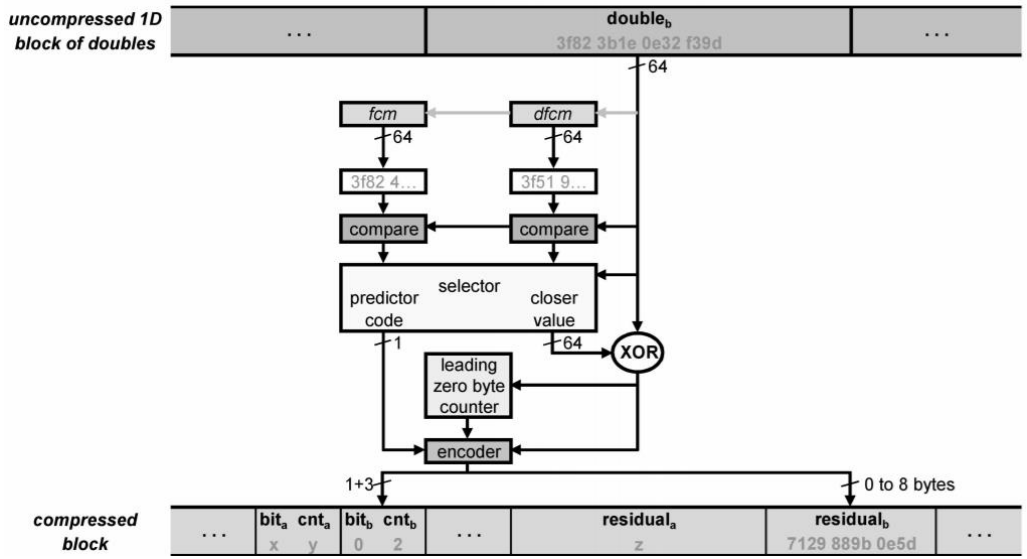
- And keep repeating until we have the finished tree, and the code from the tree



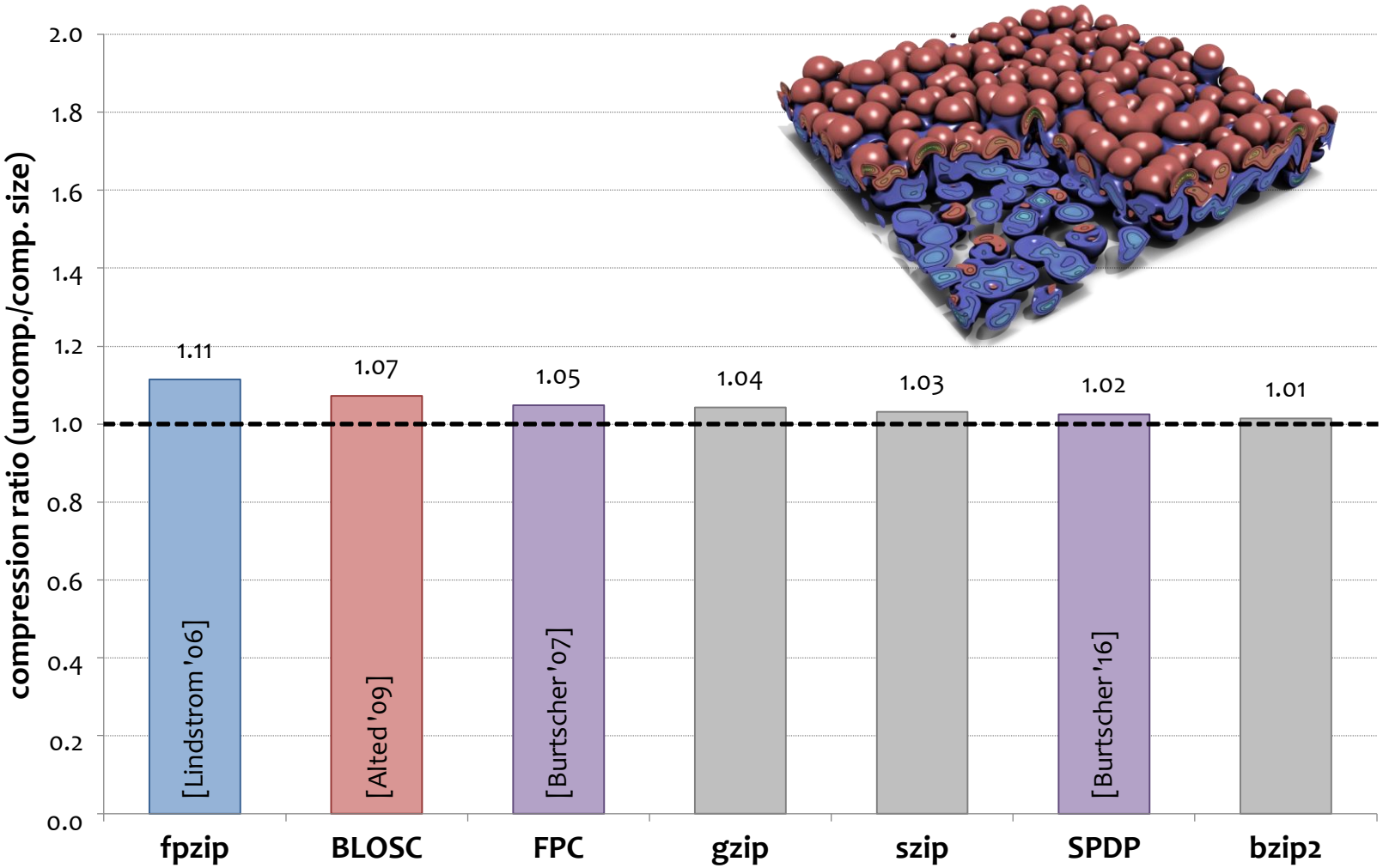
Letter	Code	Letter	Code
<ws>	01	O	10100
T	000	R	10101
L	0010	A	11011
Y	0011	U	110100
I	1001	N	110101
H	1011	F	1000100
P	1100	M	1000101
E	1110	C	1000110
G	1111	D	1000111
S	10000		

Lossless data compression

- Zlib
  - zlib was written by Jean-loup Gailly and Mark Adler and is an abstraction of the DEFLATE compression algorithm used in their gzip file compression program
- Bzip: algorithm runs as follows
  - Run-length encoding (RLE) of initial data
  - Burrows–Wheeler transform (BWT) or block sorting
  - Move to front (MTF) transform
  - Run-length encoding (RLE) of MTF result
  - Huffman coding
  - Selection between multiple Huffman tables
  - Unary base 1 encoding of Huffman table selection
  - Delta encoding (Δ) of Huffman code bit-lengths
  - Sparse bit array showing which symbols are used
- Szip
- FPC



Numerical data is challenging to compress losslessly (P. Lindstrom)



Compression/reduction techniques in ADIOS or coming soon

- SZ - ANL
- ZFP - LLNL
- MGRD – Brown-ORNL
- LZ4
- SZIP
- GZIP
- ALACRITY – NCSU-ORNL
- ISABELLA – NCSU-ORNL

Time decimation

- Most simulations write 1/N time steps
- Don’t consider possible errors, and often don’t say this is a compression technique
- Use a “linear auditing approach to save the deltas
- Missing data can be re-generated
- M. Ainsworth, S. Klasky, and B. Whitney, “Compression using lossless decimation: analysis and application,” SIAM J. Sci. Comp., 2017

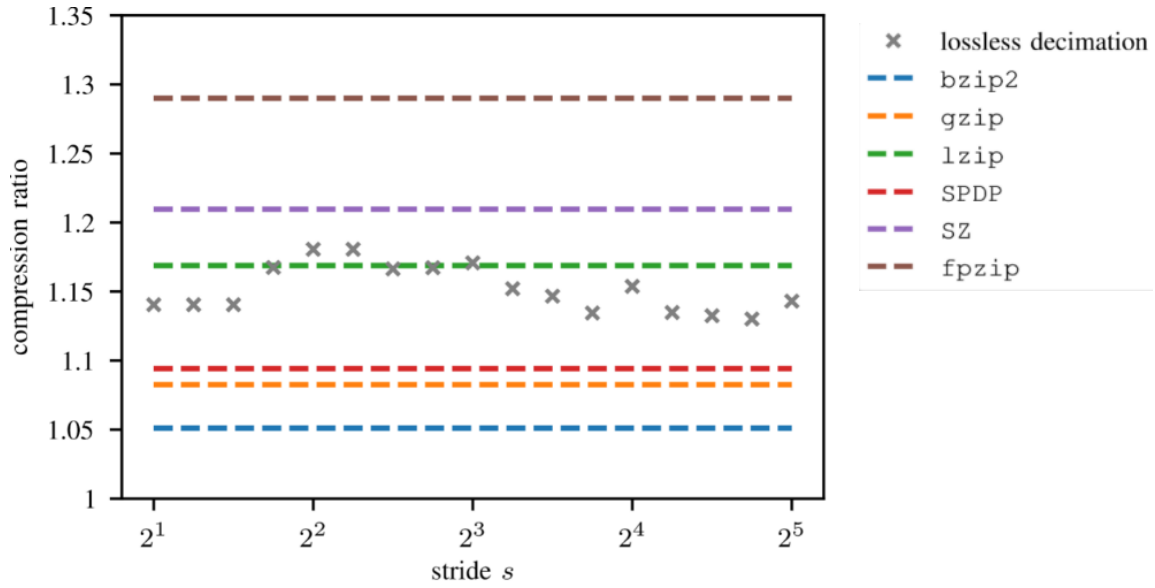


Fig. 5. Comparison of lossless decimation and a selection of general-purpose and specialized compressors. The data used is a $1024 \times 256 \times 768$ timeslab of single-precision pressure values (768 MB in total) taken from a simulation of flow in a turbulent channel [42]–[44]. For each stride, the data is decimated, linear interpolation is used to calculate a surrogate for each discarded value, and the residuals are compressed with `lzip` [45]. For this data, lossless decimation achieves a larger compression factor than `bzip2` [46], `gzip` [47], and `SPDP` [48], and is comparable to that of `lzip`. The specialized floating point compressors `SZ` [6] (run in near-lossless mode by requiring that the absolute error be at most 2^{-149}) and `fpzip` [49] (run in lossless 1D mode) outperform lossless decimation in compression ratio. A unique advantage of lossless decimation is that the decimated data, which is stored uncompressed in the output, can be interpolated to generate a surrogate dataset, without the need to decompress the entire dataset when full reproduction is not required.

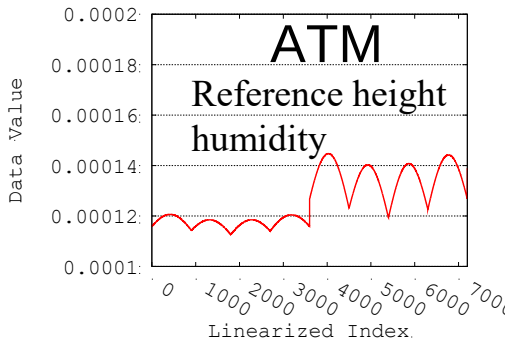
Steps to compression, look for reduction (F. Cappello)

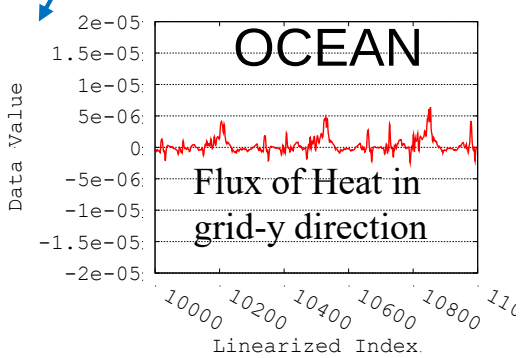
Similarities, Smoothness

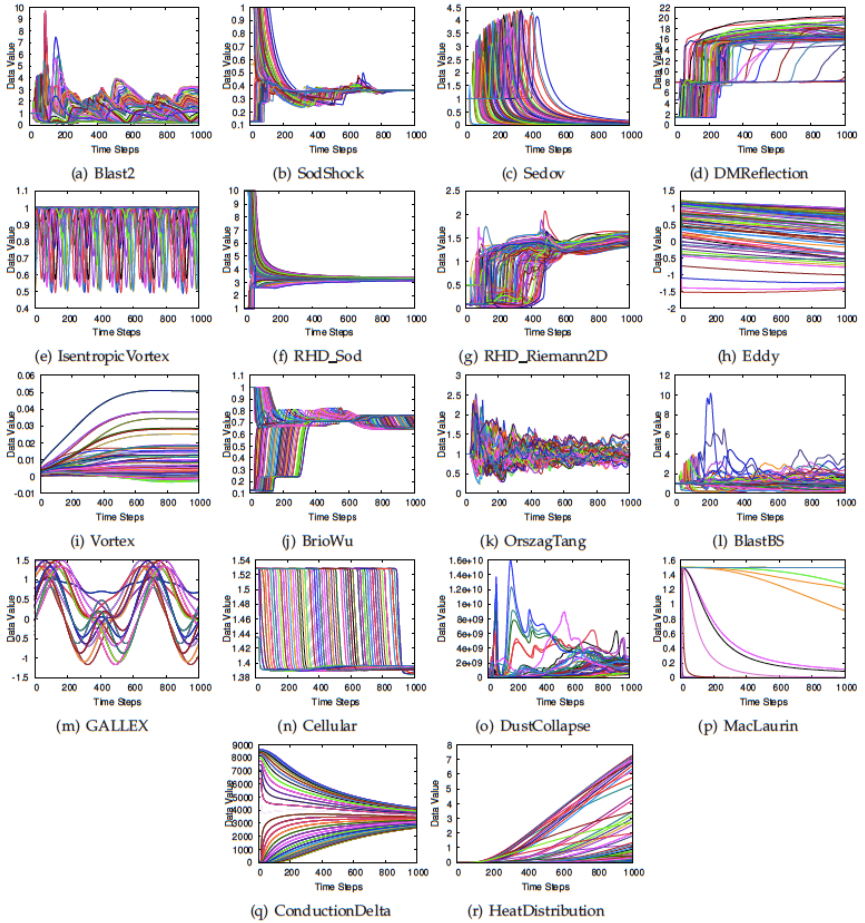
Autocorrelation,

Time/ Space

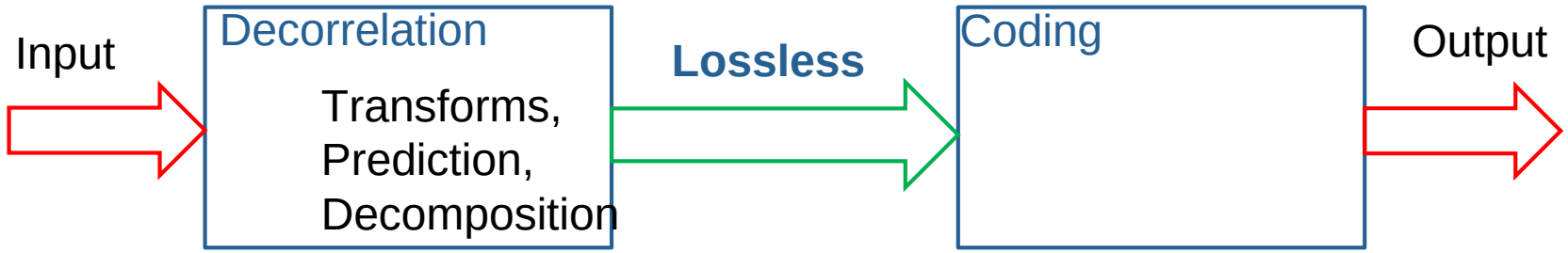
Plotting the values as time series







Main stages of a lossless compressor:



ANL SZ Lossy compressor **SZ**: **S**cientific data compressor

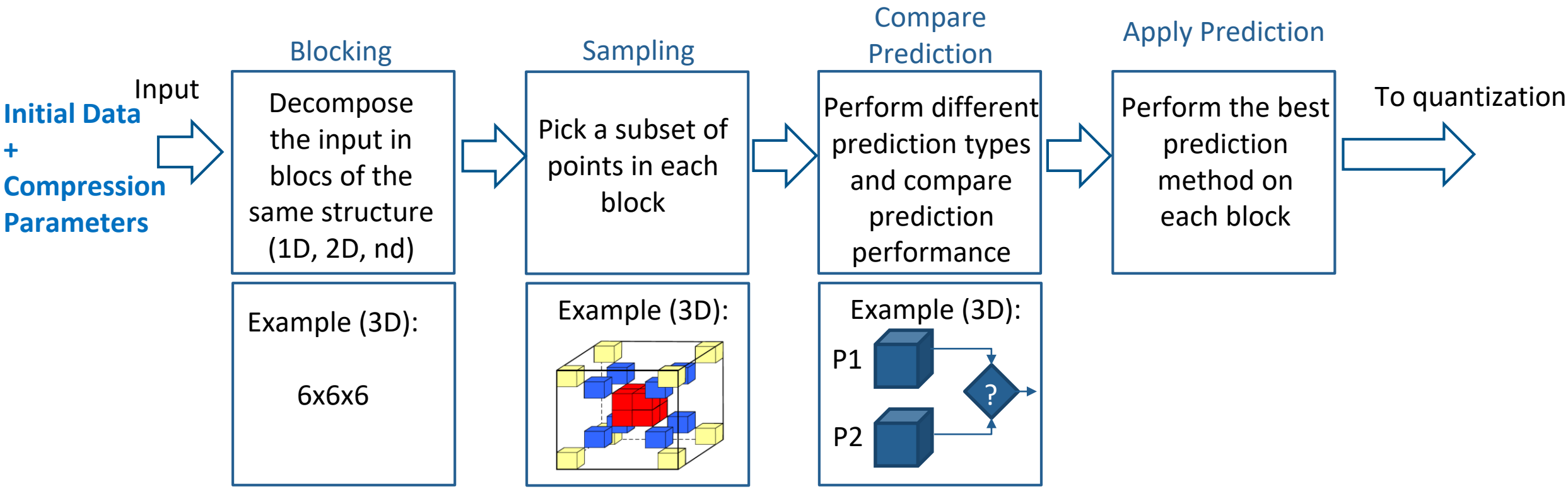
Key features:

- **Production quality** lossy compressor for scientific data respecting user set error bounds
- **For 1D, 2D, 3D structured and unstructured datasets**. E.g. 3D simulation fields, instruments data, time series.
- For **floating point** and **integer** data
- **Strict error controls** (absolute error, relative error, PSNR, *error distribution*)
- **Thorough testing procedures**, bug tracking, tests on the CORAL systems
- **Integrated in the ADIOS, HDF5 and PnetCDF I/O libraries**.
- Reader/Writer for ADIOS, HDF5, netCDF
- **Optimized compression ratios** (optional transform, multiple predictors, compression in time, lossless compression, etc.)
- **High compression/decompression speed** (*MPI + OpenMP*)

SZ is supported by the DOE/NNSA **ECP EZ and ExaSky** Projects and the **NSF Aletheia** Project.

SZ 2.0 Prediction stage

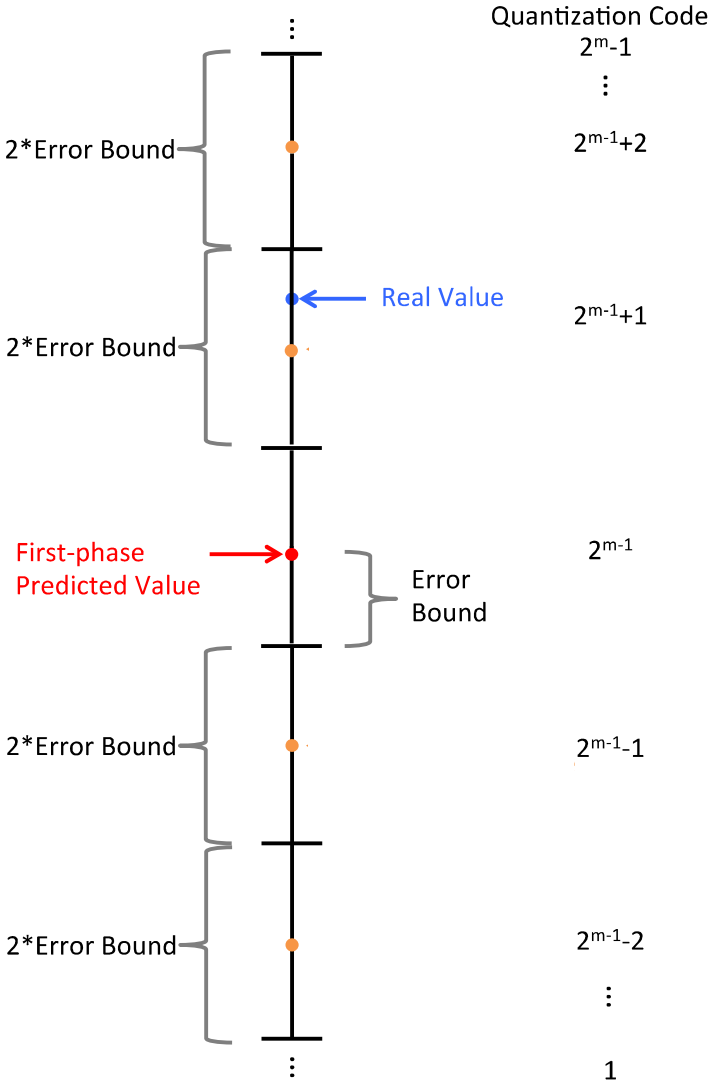
- SZ 2.0 Block based, 2 decorrelation schemes (Lorenzo and regression), on-line, per block selection of the best decorrelation scheme

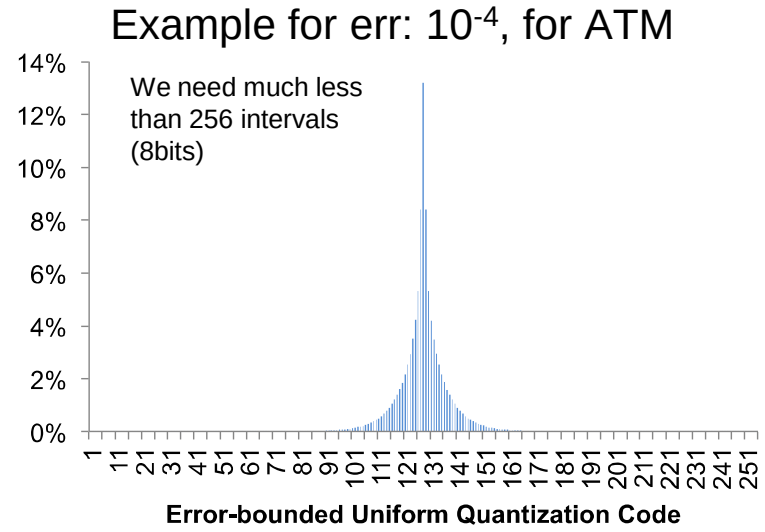


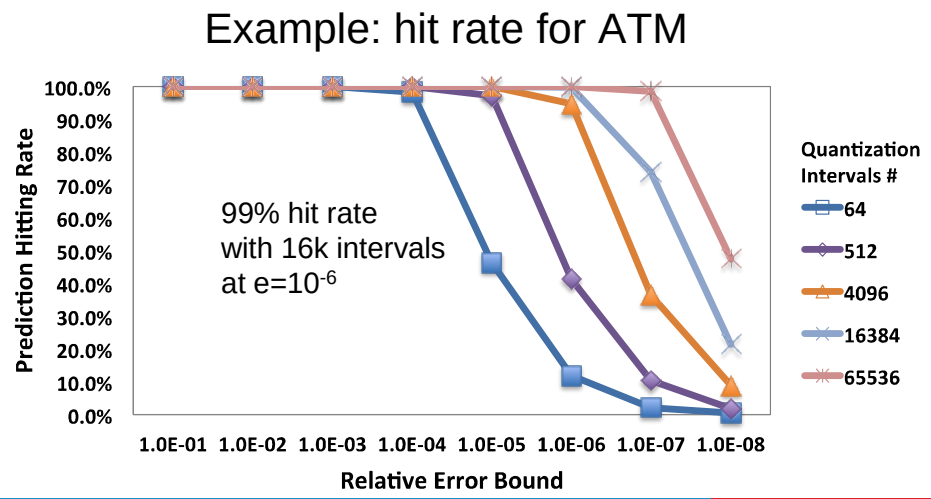
X. Liang , S. Di, D. Tao, S. Li, S. Lix, H. Guo , Z. Chen, F. Cappello, Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets, submitted

SZ 2.0 Quantization Stage

Linear Quantization of prediction error (map data into quantization bins)



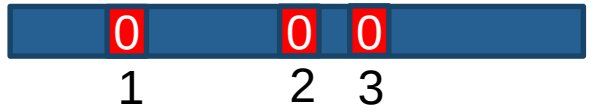




- Transforms each predicted data into 1 integer value (with loss)
- If data is out of scale, keep it in a separate array

Initial data (FP numbers)

Array of quantization (integers)



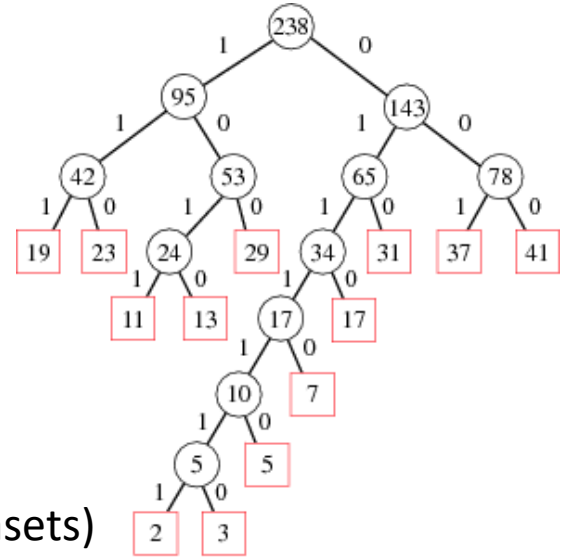
Array of unpredictable data (FP)

Data are ordered in the arrays in their initial order

SZ 2.0 Coding Stage

Variable length coding (Huffman)

We built the Huffman tree using a symbol size corresponding to the number of bits needed to code the bin numbers

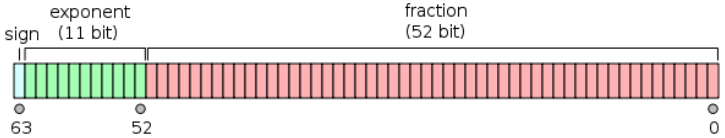


➡ Reduce the number of bits needed to represent values

Huffman can be very expensive (building the tree, tree size)

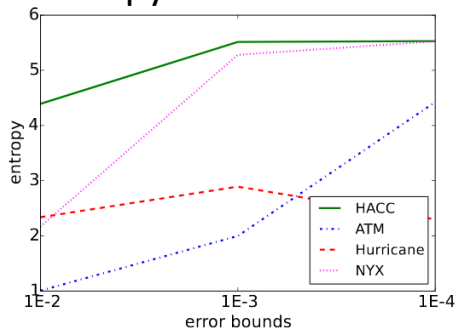
➡ Future: will add a less expensive (less efficient) coding for small datasets)

Unpredictable data analysis



➡ Reduce the number of bits needed to represent unpredictable data (<1%)

Entropy after Huffman



Optional Zstandard (L77 + Finite State Entropy): improve the compression by about 10%

3) Reducing Storage Footprint

PSNR:

$$X = \{x_1, x_2, ..., x_N\} \qquad R_X = x_{max} - x_{min}$$
$$\tilde{X} = \{\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_N}\} \qquad e_{abs_i} = x_i - \tilde{x_i}$$

$$rmse = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(e_{abs_i})^2}$$

$$psnr = 20 \cdot log_{10}(\frac{R_X}{rmse})$$

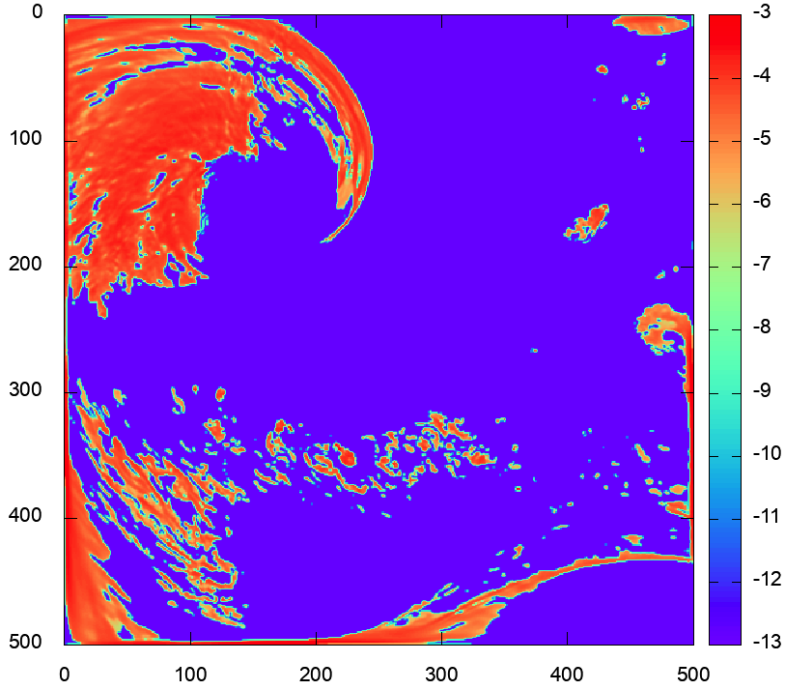
SSIM:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

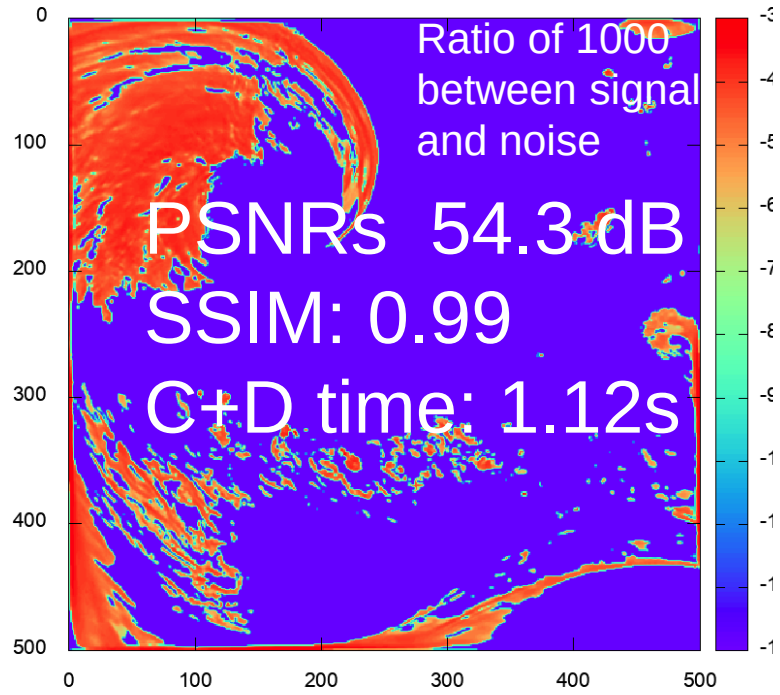
μ_x the average of x;
μ_y the average of y;
σ_x^2 the variance of x;
σ_y^2 the variance of y;
σ_{xy} the covariance of x and y;

X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, F. Cappello, Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets**, IEEE BigData** 2018

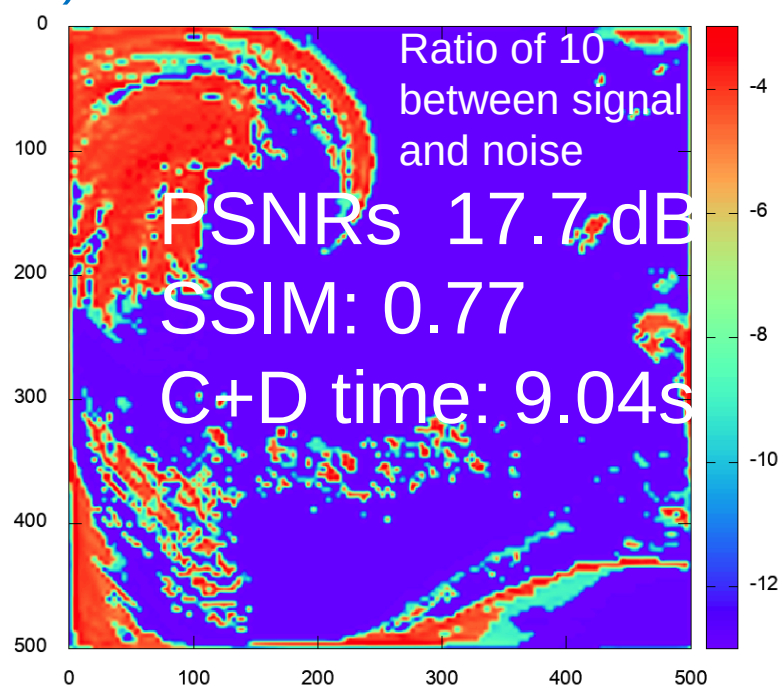
Hurricane dataset: CLOUDf field (SZ2.0 absolute error bound set to 10^{-2}): **SZ CR= 62**



Original



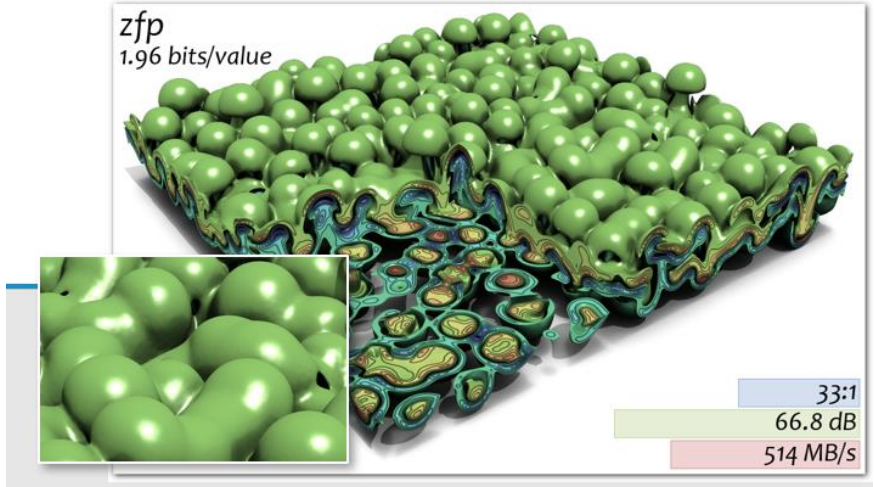
SZ 2.0

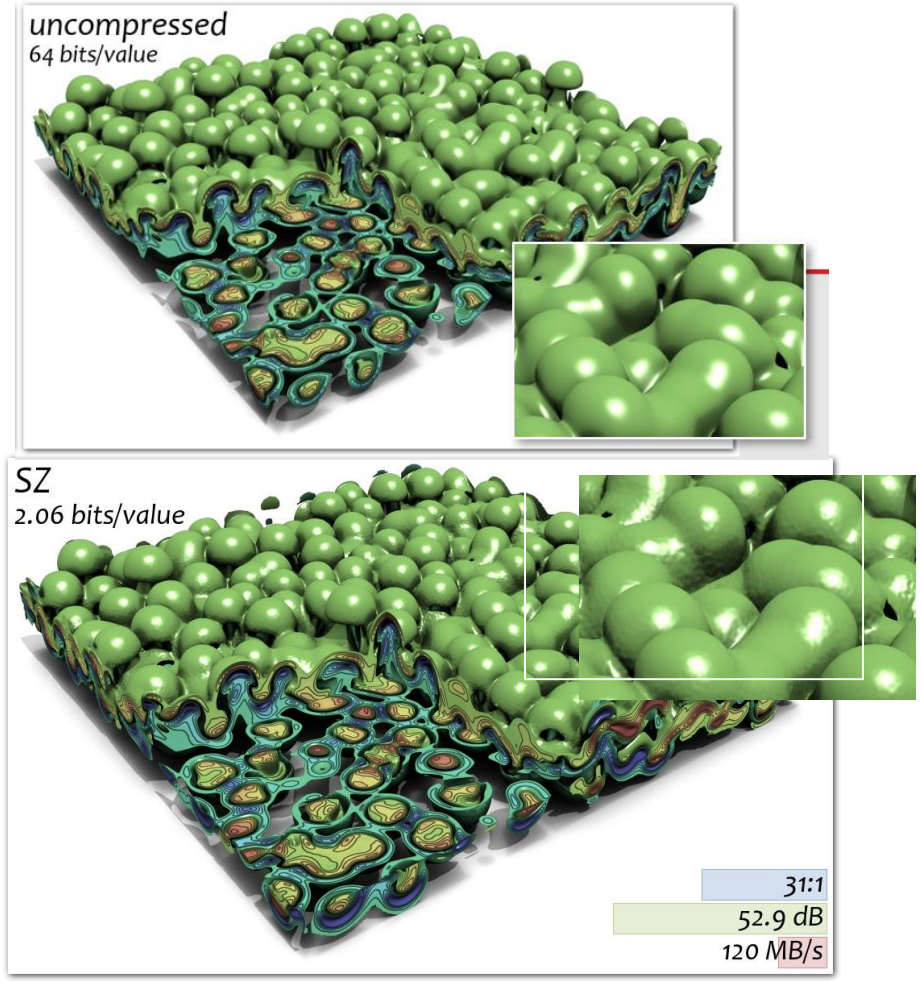


Spatial sampling + tricubic interpolation

ZFP

- ZFP random-accessible compressed arrays
  - Partition *d*-dim. array into blocks of 4^d values
    - Blocks are independently (de)compressed on demand
  - Can truncate compressed bit stream anywhere
    - Variable rate enables **absolute-error bound**
    - Fixed rate enables read/write **random access** to block
- github.com/LLNL/zfp
  - BSD licensed open source





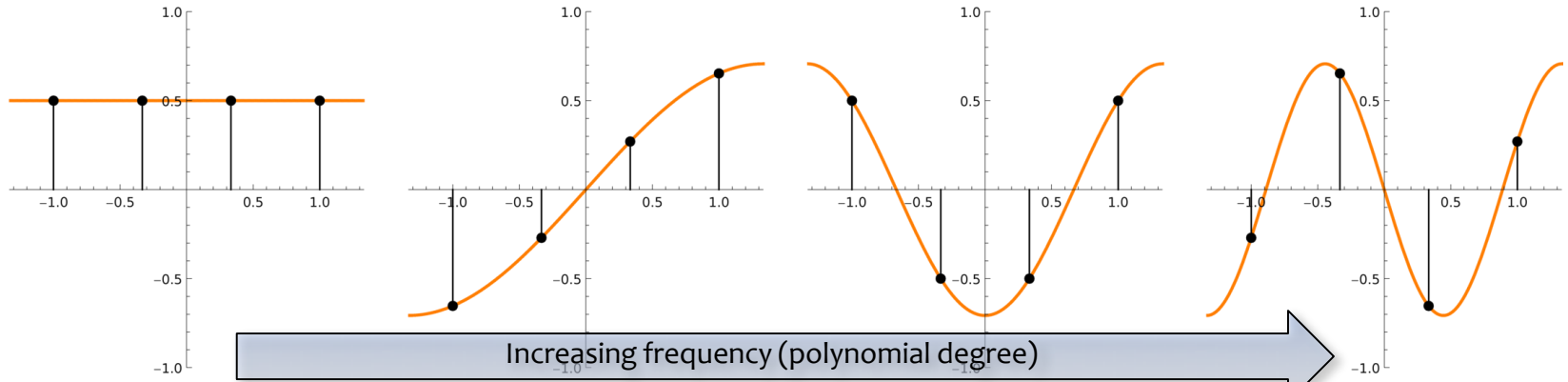
ZFP decorrelates d-dimensional block of 4d values using an orthogonal transform

$$\underbrace{\begin{pmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \hat{f}_4 \end{pmatrix}}_{\text{coefficients}} = \underbrace{\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ c & s & -s & -c \\ 1 & -1 & -1 & 1 \\ s & -c & c & -s \end{pmatrix}}_{\text{orthogonal transform}} \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}}_{\text{block}}$$

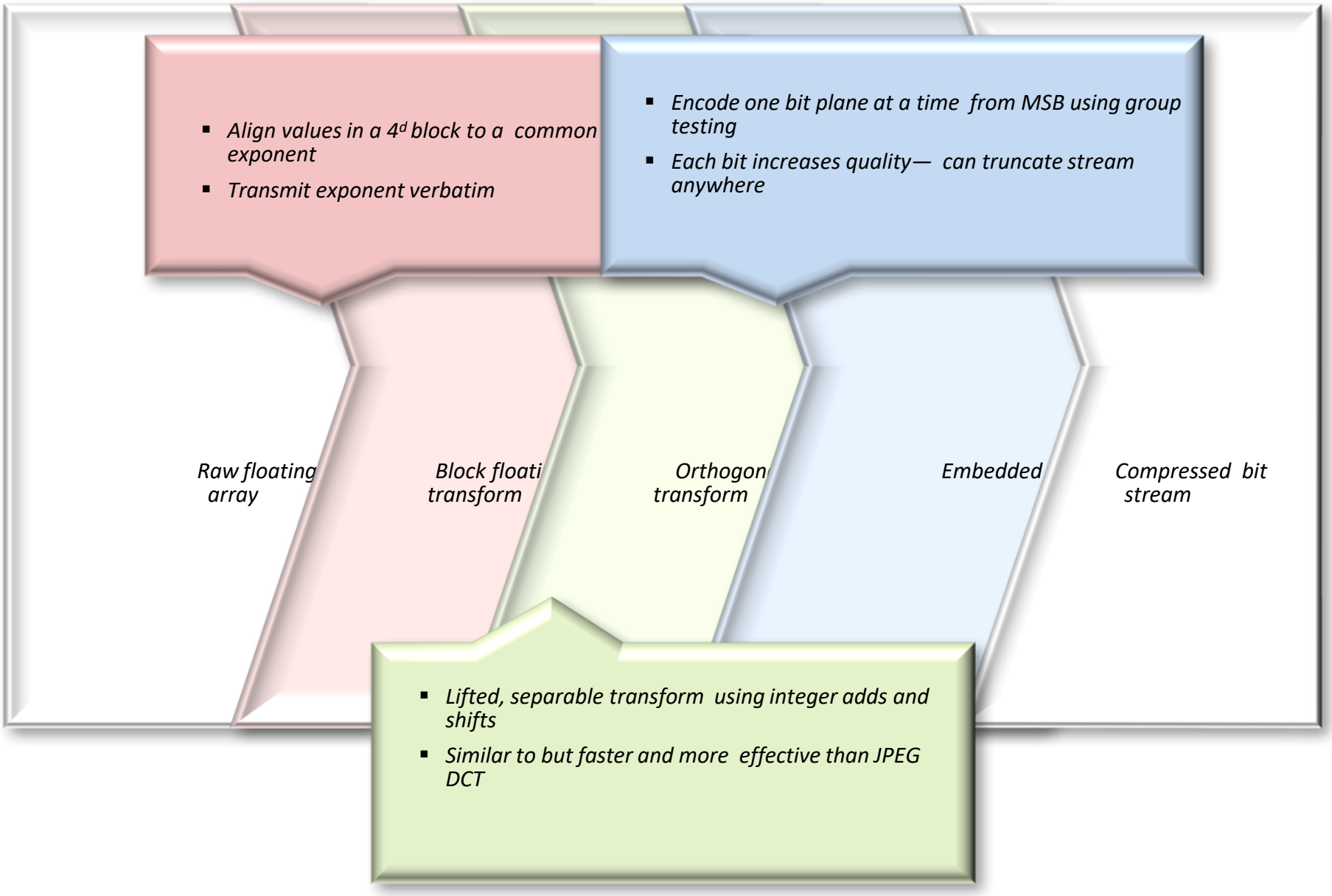
Free parameter t

$$s = \sqrt{2}\sin\tfrac{\pi}{2}t \qquad c = \sqrt{2}\cos\tfrac{\pi}{2}t$$

Basis functions for $t = 1/4$



ZFP: Compressed floating-point array that support random access on demand



MGARD: MultiGrid Adaptive Reduction of Data

Decomposes data into contributions from a hierarchy of meshes,

$$u = Q_0 u + (Q_1 - Q_0)u + \ldots + (Q_L - Q_{L-1})u$$

Adaptive reduction of data based on discarding least important contributions

$$u = \sum_{\ell=0}^{L} \sum_{n \in N_\ell} \alpha_n^l \varphi_n^\ell \qquad \tilde{\alpha}_n^\ell = \begin{cases} \alpha_n^\ell, & \text{for } |\alpha_n^\ell| \geqslant \tau_c \\ 0, & \text{otherwise} \end{cases}$$

$$\tilde{u} = \sum_{\ell=0}^{L} \sum_{n \in N_\ell} \tilde{\alpha}_n^\ell \varphi_n^l$$

Mathematically proven error bounds

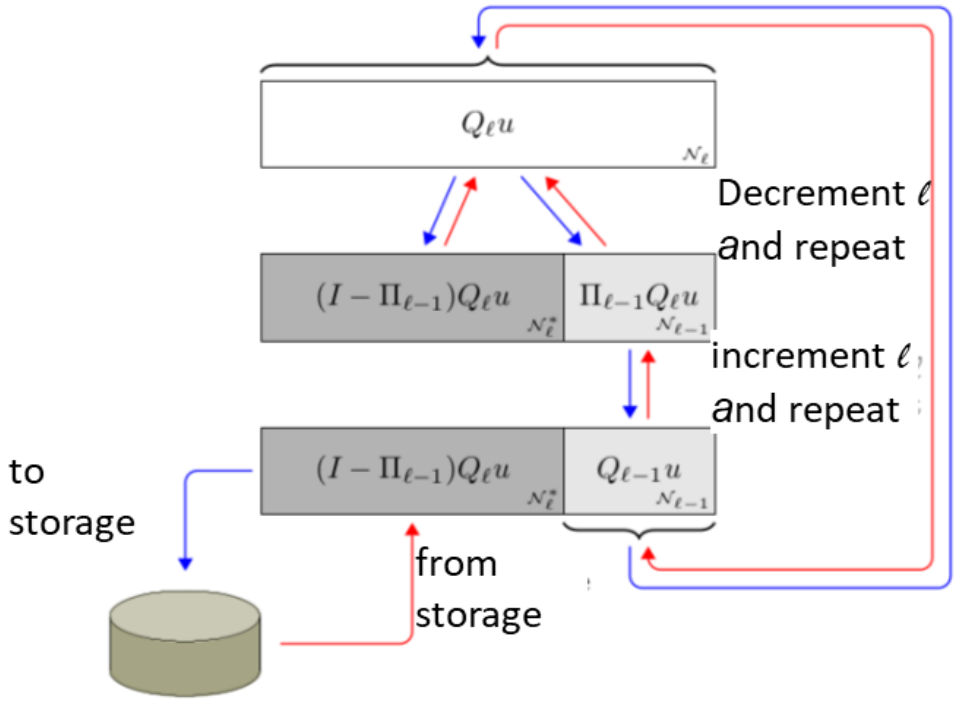
$$C_d \eta_s(v) \leq \|v\|_s \leq \eta_s(v)$$

where

$$\eta_s(v)^2 = \sum_{\ell=0}^{L} 2^{2s\ell} \|\Delta_\ell v\|^2$$

Applicable to structured (tensor product) grids with arbitrary spacing, integrated into ADIOS

Able to preserve quantities of interest (spectrum, averages…)



MGARD decomposition and recomposition workflow

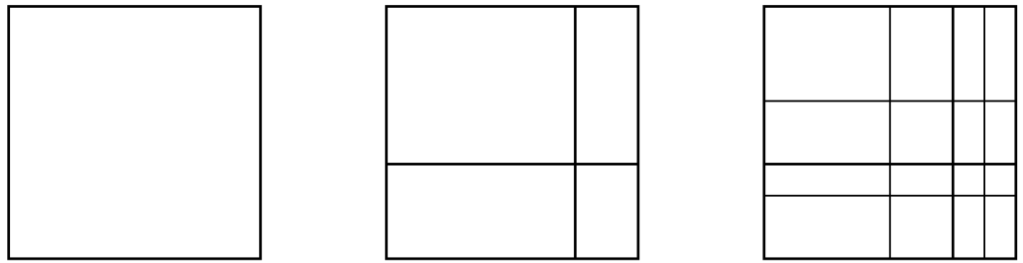


Illustration of a tensor product mesh hierarchy

M. Ainsworth, O. Tugluk, B. Whitney, S. Klasky, "Multilevel Techniques for Compression and Reduction of Scientific Data---Quantitative Control of Accuracy in Derived Quantities", *SIAM Journal on Scientific Computing,* Submitted for publication 2018.

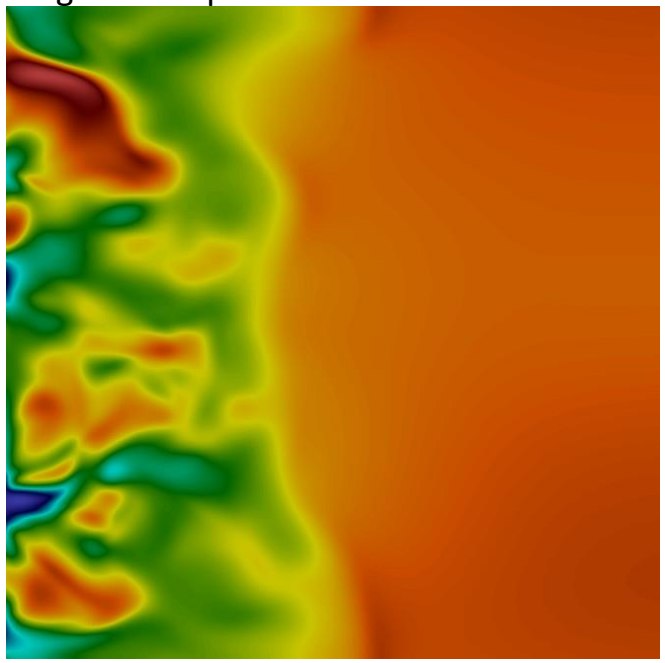
M. Ainsworth, O. Tugluk, B. Whitney, S. Klasky, "Multilevel Techniques for Compression and Reduction of Scientific Data --The Multivariate Case", *SIAM Journal on Scientific Computing,* Submitted for publication 2018.

M. Ainsworth, O. Tugluk, B. Whitney, S. Klasky, "Multilevel Techniques for Compression and Reduction of Scientific Data --The Univariate Case", *Computing and Visualization in Science,* Accepted for publication 2018.

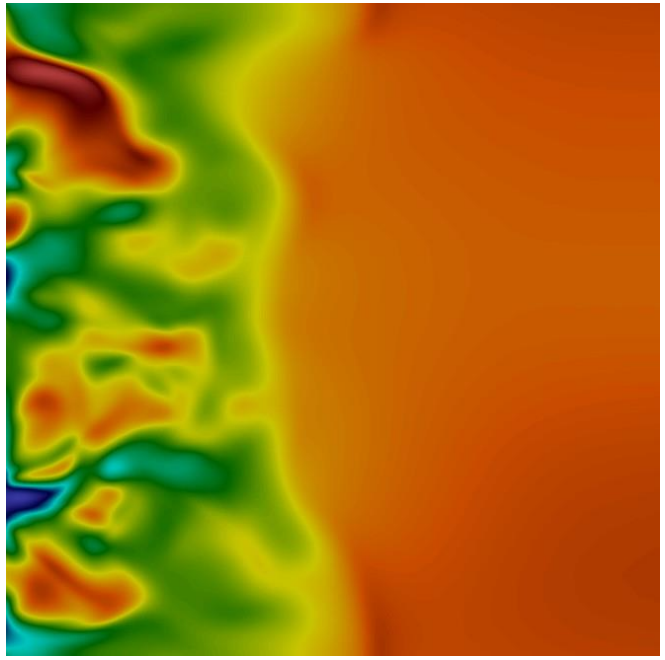
MGARD: Data Application to ECP APP S3D

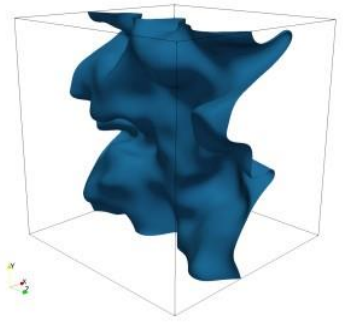
Contours of axial velocity, data from S3D simulation.

Original Compression

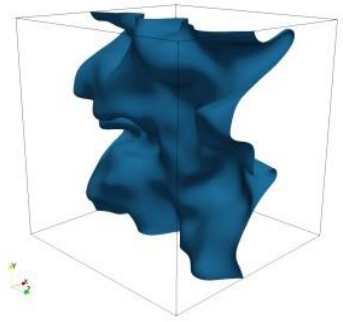


MGARD with PSNR=60, 200X compression

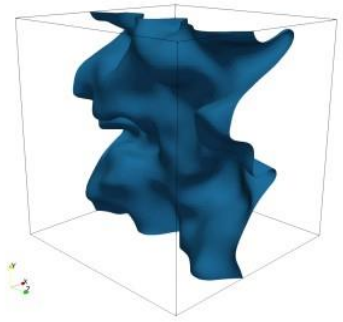




(a) Original dataset.



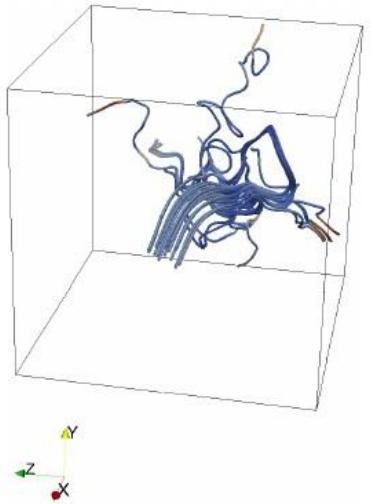
(b) Reduced dataset (target PSNR 60 dB and 277 × compression).



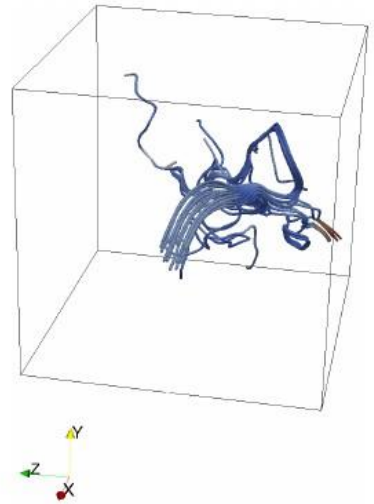
(c) Reduced dataset (target PSNR 40 dB and 1401 × compression).

Figure: Flame front (T=1200K), data from S3D simulation

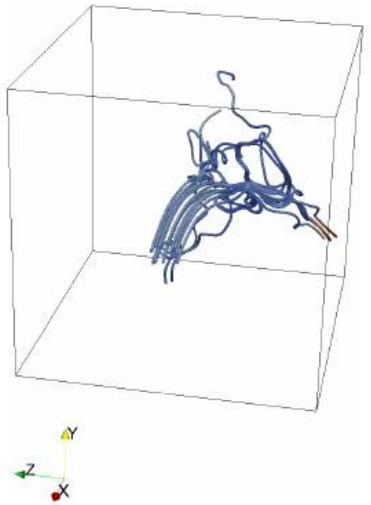
Figure : Streamlines generated for the original flowfield (a), and various modes of MGARD compression (b, c, d)



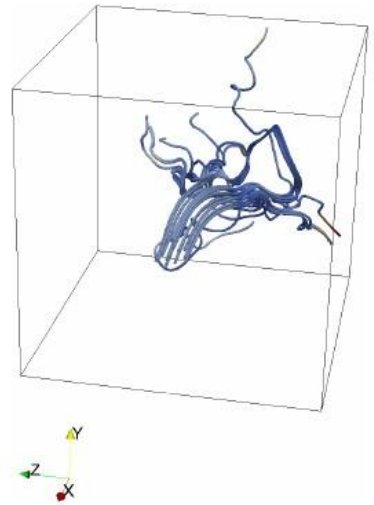
(a) Original dataset.



(b) Reduced dataset using MGARD with PSNR 40 dB (1403 × compression).



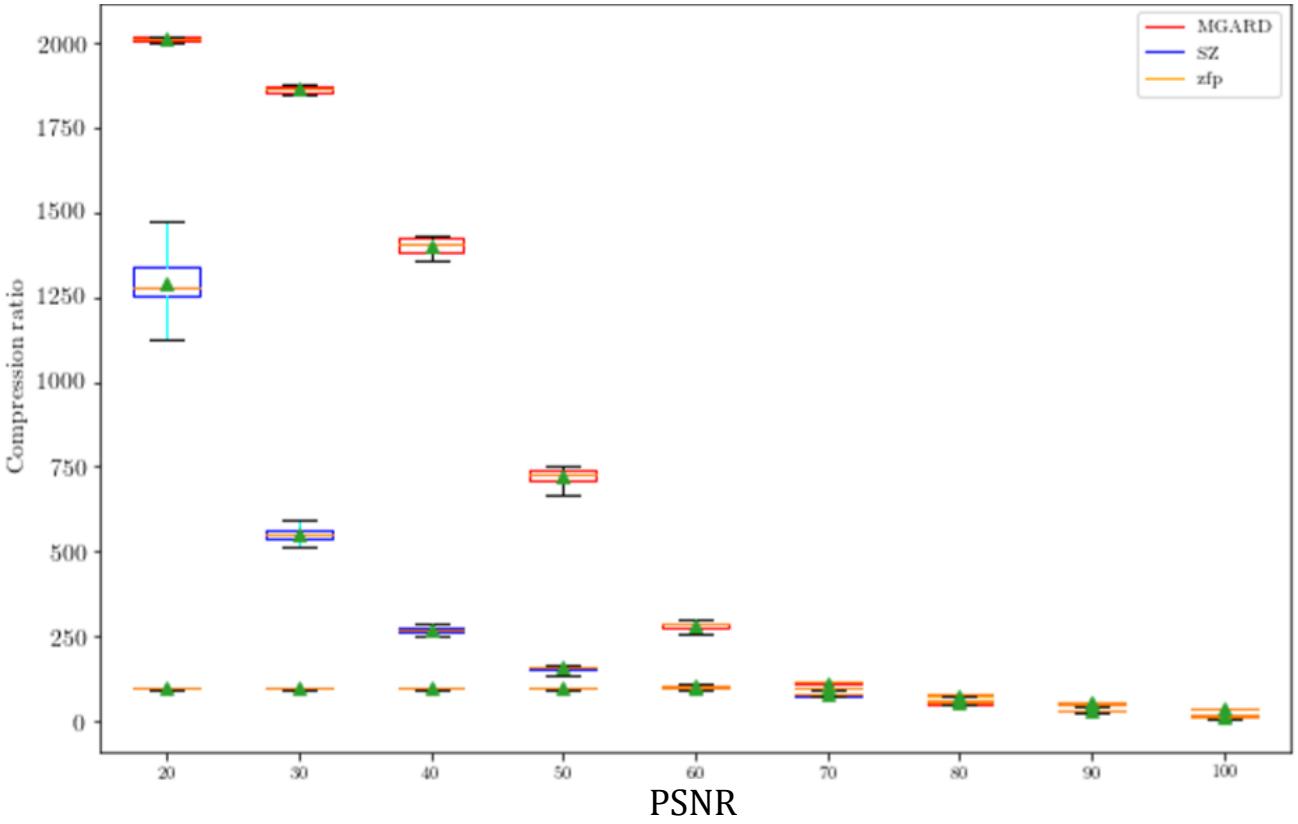
(c) Reduced dataset using MGARD with $s = \frac{1}{2}$ and $\tau = 5 \times 10^{-4}$ (1214 × compression).



(d) Reduced dataset using MGARD with $s = 1$ and $\tau = 5 \times 10^{-4}$ (1701 × compression).

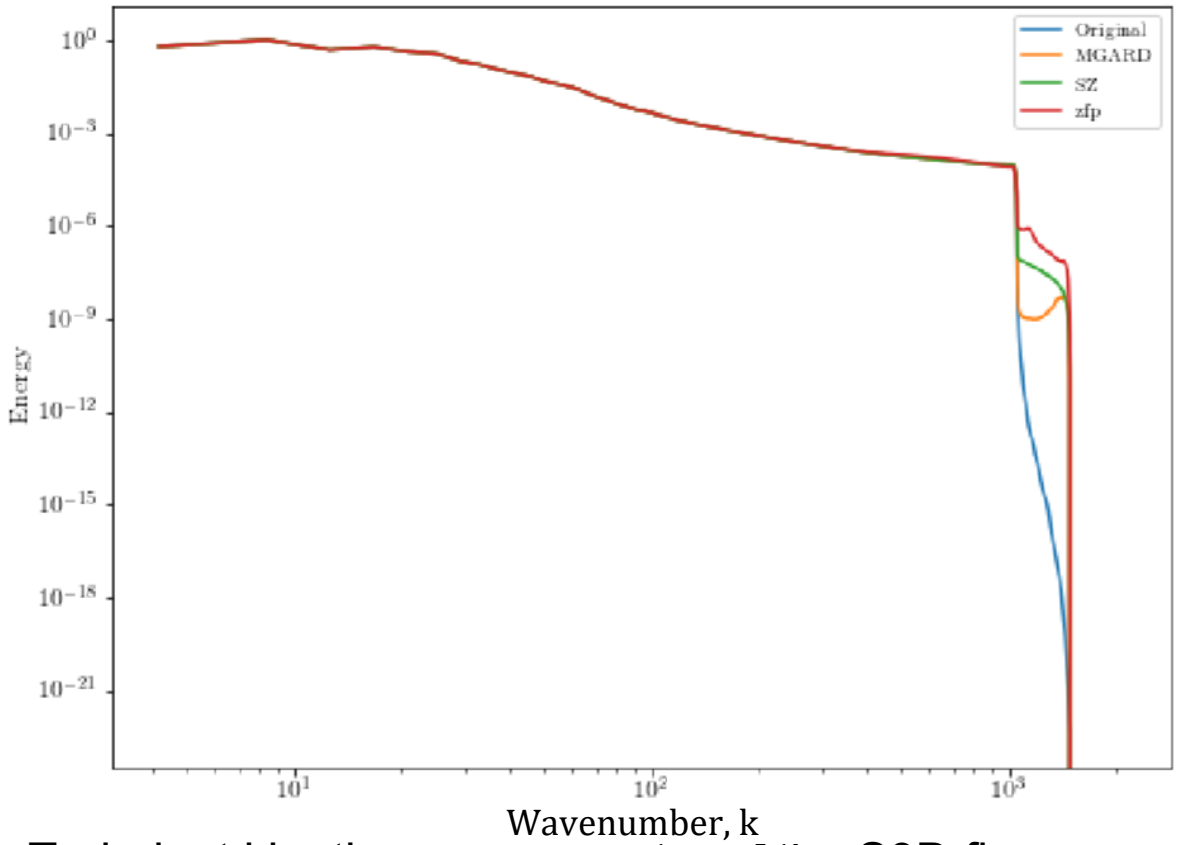
MGARD PSNR & energy spectrum compared to SZ and ZFP for the S3D

Compression ratios achieved by MGARD, SZ, and zfp on the S3D dataset



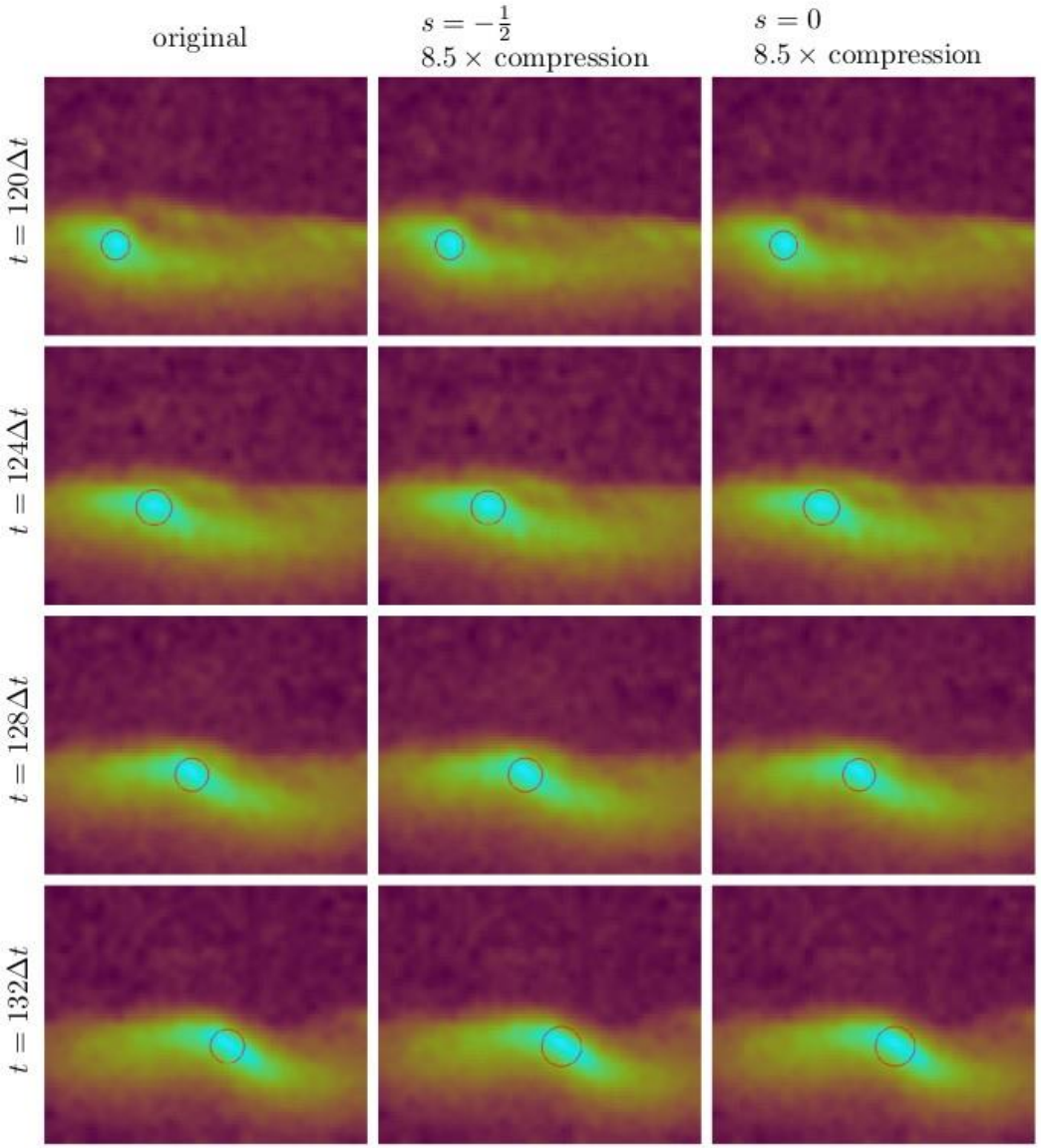
Compression ratios achieved by MGARD, SZ, and zfp on the S3D dataset at various PSNR tolerances.
At each PSNR, a boxplot of the compression ratios of each species of the output is shown. The target PSNR is set directly for MGARD and SZ, while for zfp the fixed-rate mode is used and the resulting PSNR is calculated.

Effects on compression on energy spectrum, PSNR=60

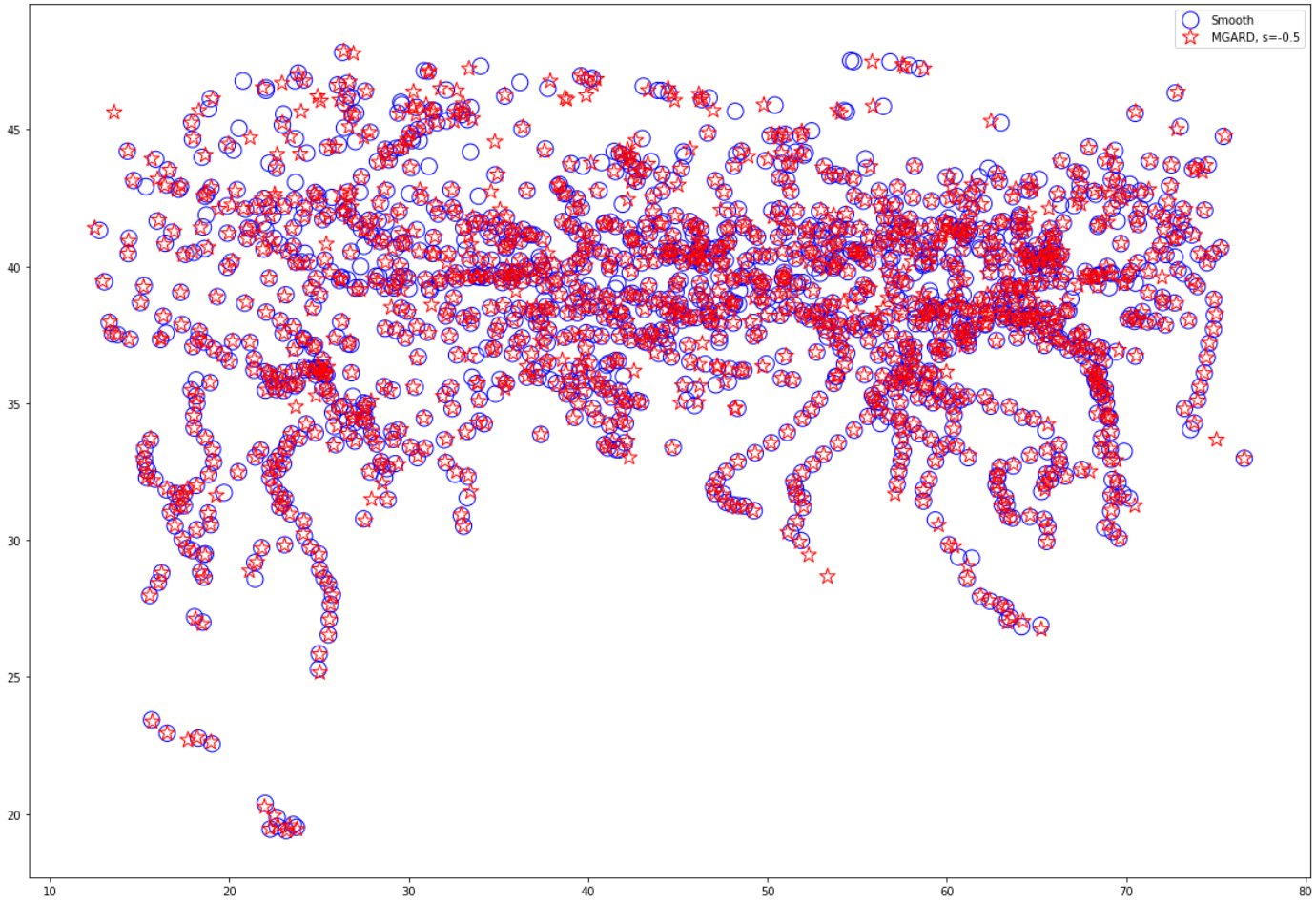


Turbulent kinetic energy spectra of the S3D flow field at time 2.999×10−3. The reduced datasets have target peak signal-to-noise ratios of 60dB. The distortion in the spectra is concentrated in the smaller scales of the flow. The larger scales (lower wavenumbers) are faithfully reproduced.

MGARD: Data Application to Fusion Experimental Data



Blobs detected in original and reduced datasets.



First 1500 blob centers as identified by openCV library.
Circles: blob centers found in the original data, stars: blob centers found in the reduced dataset

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

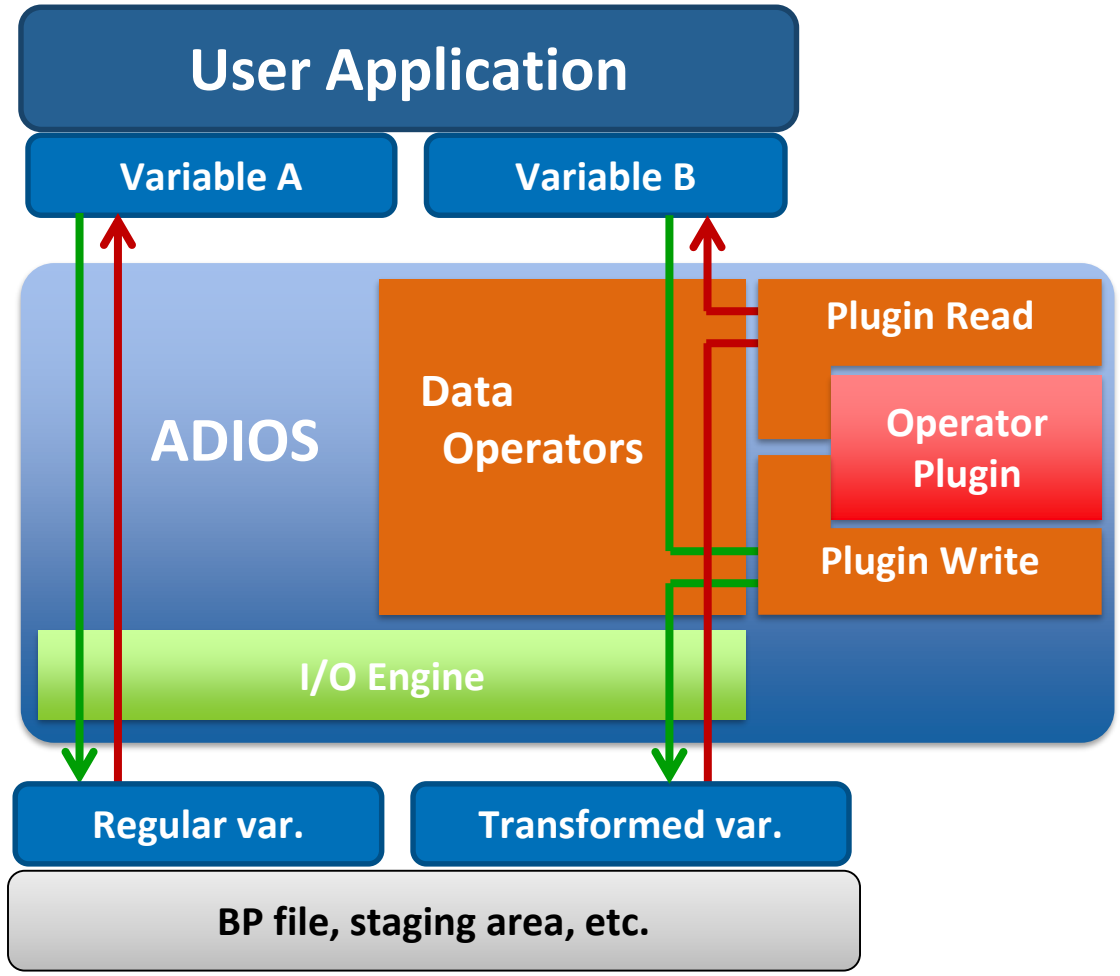
Wrap-up

Data reduction operators in ADIOS

Compression/decompression
with Gray-Scott Example

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

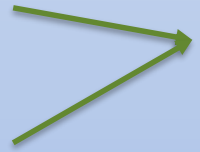
- Using the BP4 file output engine
- Run the pdf-calc example
  - with dumping the input data
  - with compressing U and V
- With SZ
  - with different accuracy levels (0.01, 0.0001, 0.000001)
- Compare the size of gs.bp and pdf.bp
- Run
  `python3 gsplot.py -i <file> -o <picname>`
  to plot data and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

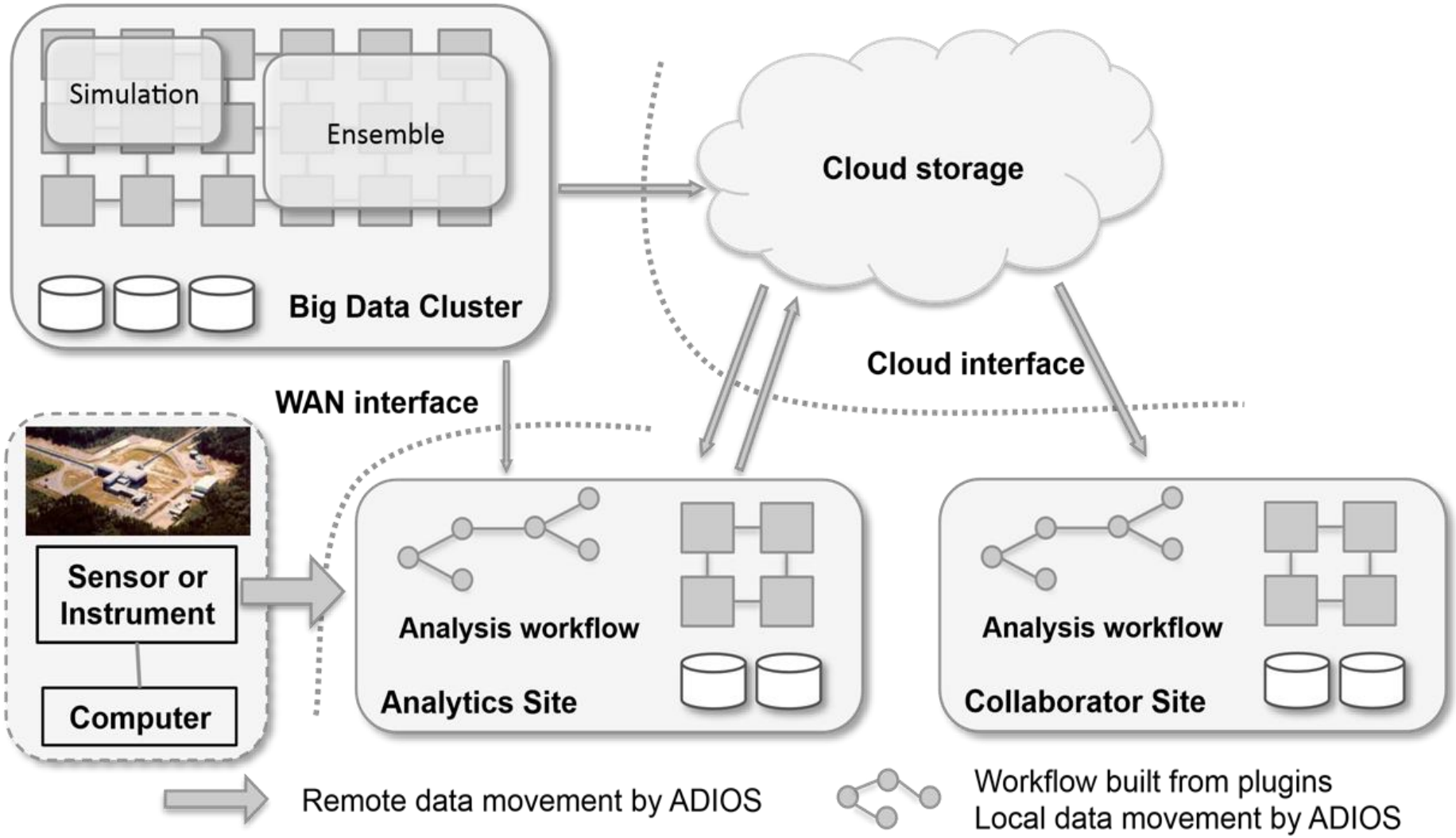
- **Lecture**: Overview of common data reduction techniques for scientific data
  - Introduction to compression
  - Introduction to lossy compression techniques: MGARD, SZ, and ZFP
- **Hands-on**: Adding compression to previous examples

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

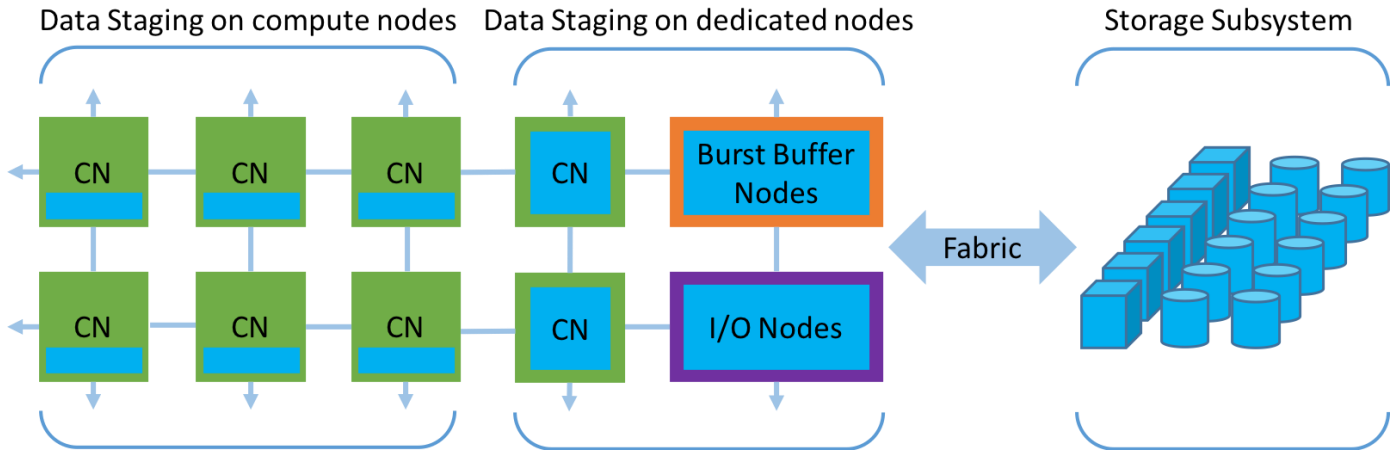
Wrap-up

Vision: building scientific collaborative applications



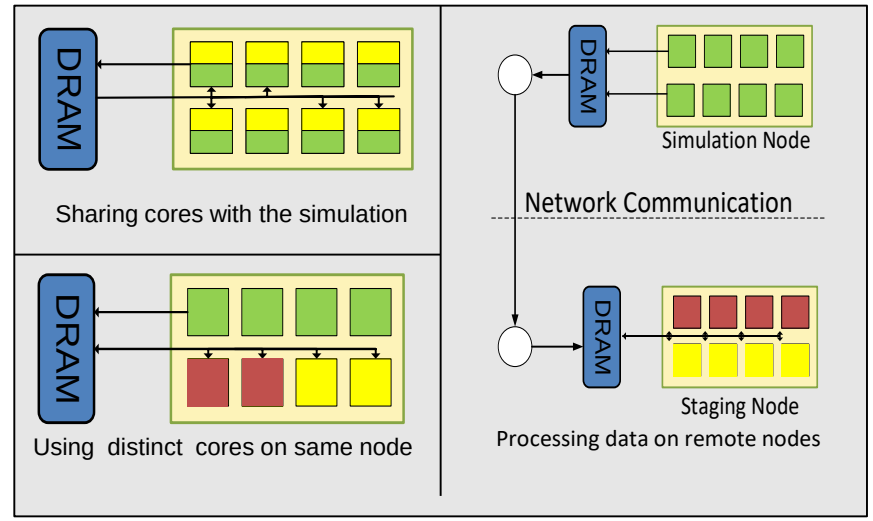
Rethinking the Data Pipeline – Data Staging

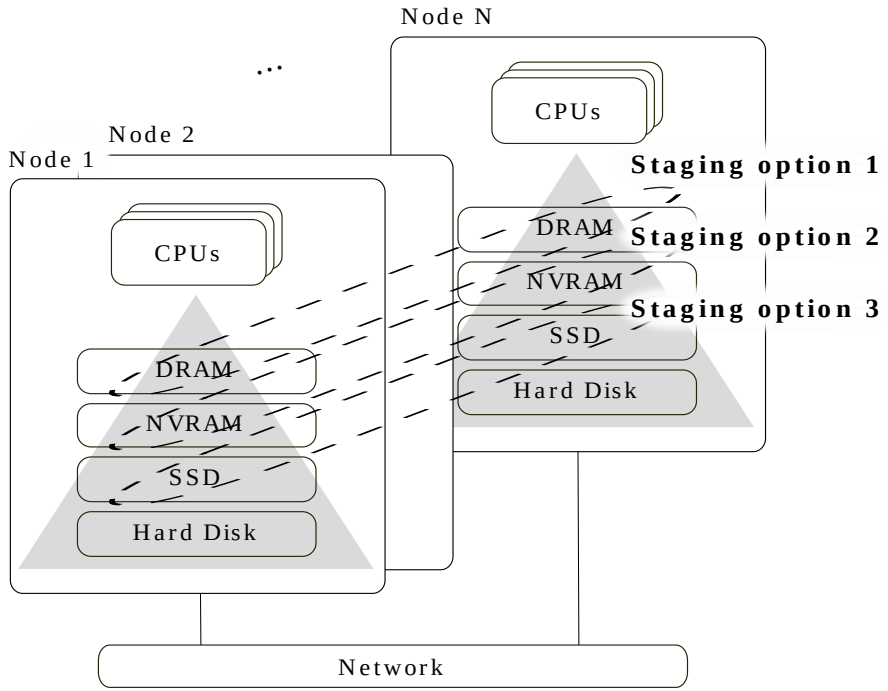
- Leverage resources (storage CPU, accelerators, etc.) for data management
  - Reduce data movement
  - Move computation/analytics closer to the data source
  - Process, transform data along the data path
  - Minimize us of the lower tiers of the memory hierarchy (e.g., file system)
- Support in-situ workflows with flexible data coupling / data-exchange behaviors
- A number of staging solutions/services have emerged representing different design choices, programming abstractions, runtimes, e.g., FlexPath, DataSpaces, etc.
- Multiple staging options in ADIOS: SST, SST-SM, DataMan, InSituMPI



Rich Design Space for Staging

- Location of the compute resources
  - Same cores as the simulation (in situ)
  - Some (dedicated) cores on the same nodes
  - Some dedicated nodes on the same machine
  - Dedicated nodes on an external resource
- Data access, placement, and persistence
  - Direct access to simulation data structures
  - Shared memory access via hand-off / copy
  - Shared memory access via non-volatile near node storage (NVRAM, BB)
  - Data transfer to dedicated nodes or external resources (decoupled in space)
- Synchronization and scheduling
  - Execute synchronously with simulation every n^{th} simulation time step
  - Execute asynchronously (decoupled in time)
  - Dynamic execution



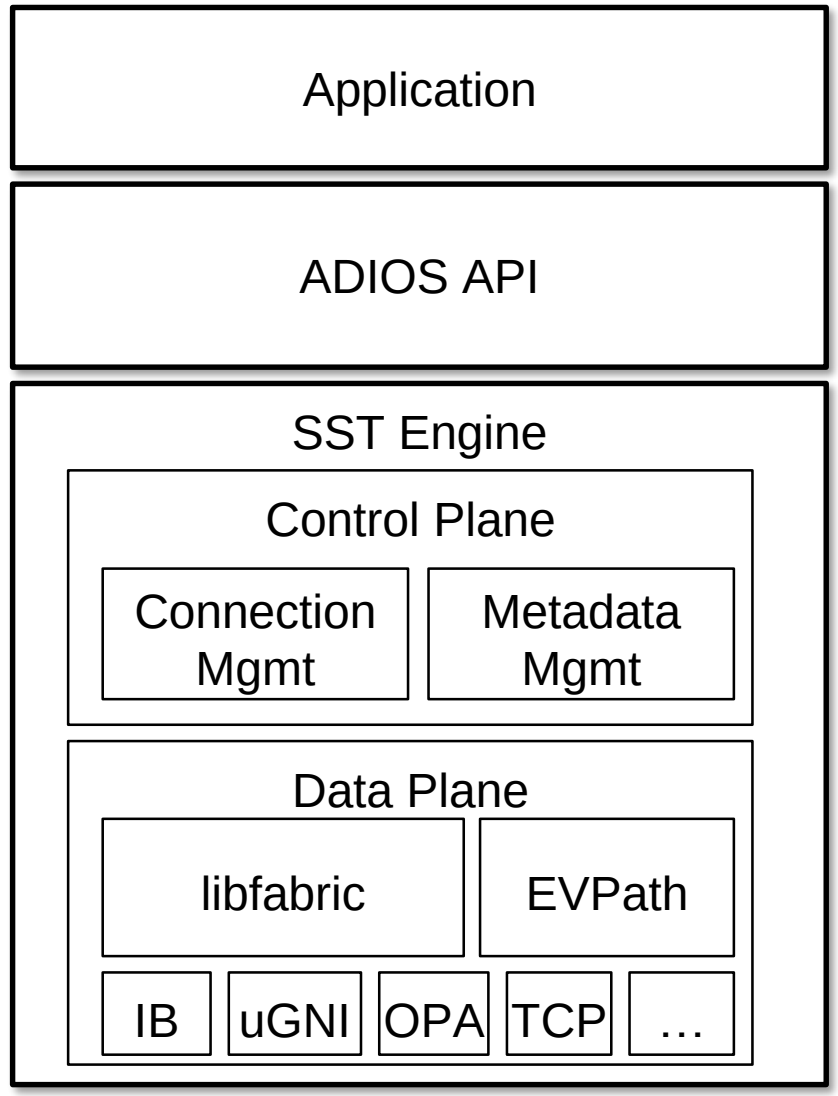


Data Staging in ADIOS

- Sustainable Staging Transport (SST)
  - In-situ infrastructure for staging in a streaming-like fashion using RDMA, SOCKETS
- InSituMPI
  - One way staging for MPMD applications
- DataMan
  - WAN transfers using sockets and ZeroMQ
- SSC (coming in v2.6)
  - One-sided MPI for strong coupling of codes
- DataSpaces
  - Staging infrastructure providing a shared memory abstraction

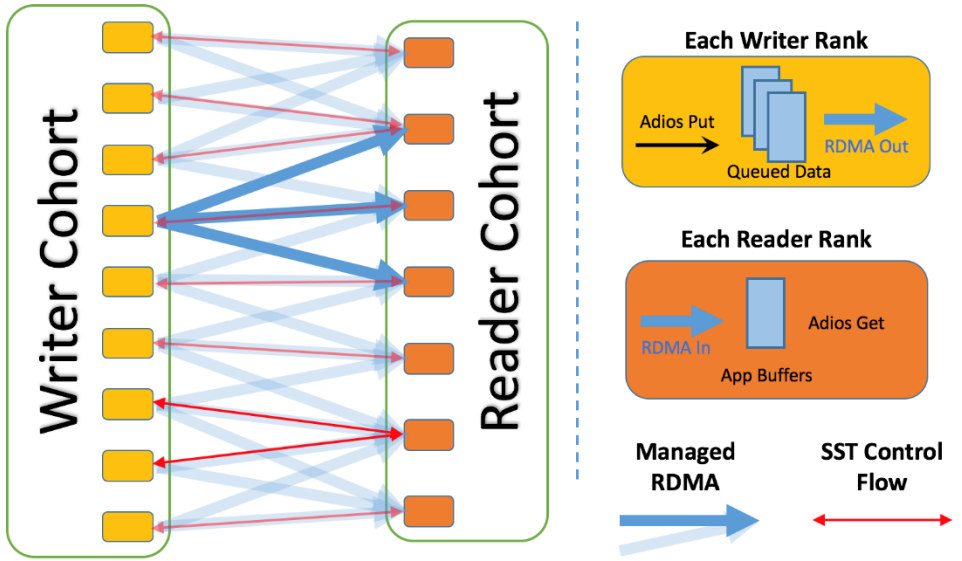
Sustainable Staging Transport (SST)

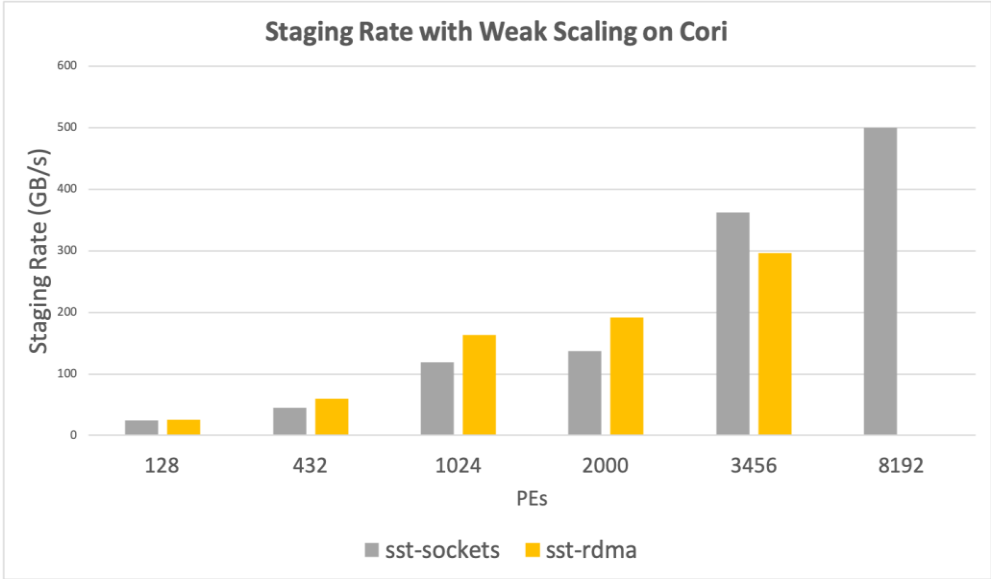
- Direct coupling between data producers and consumers for in-situ/in-transit processing
- Designed for portability and reliability.
- Control Plane
  - Manages meta-data and control using a message-oriented protocol
  - Inherits concepts from Flexpath, DIMES, uses EVPath
  - Allows for dynamic connections, 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).



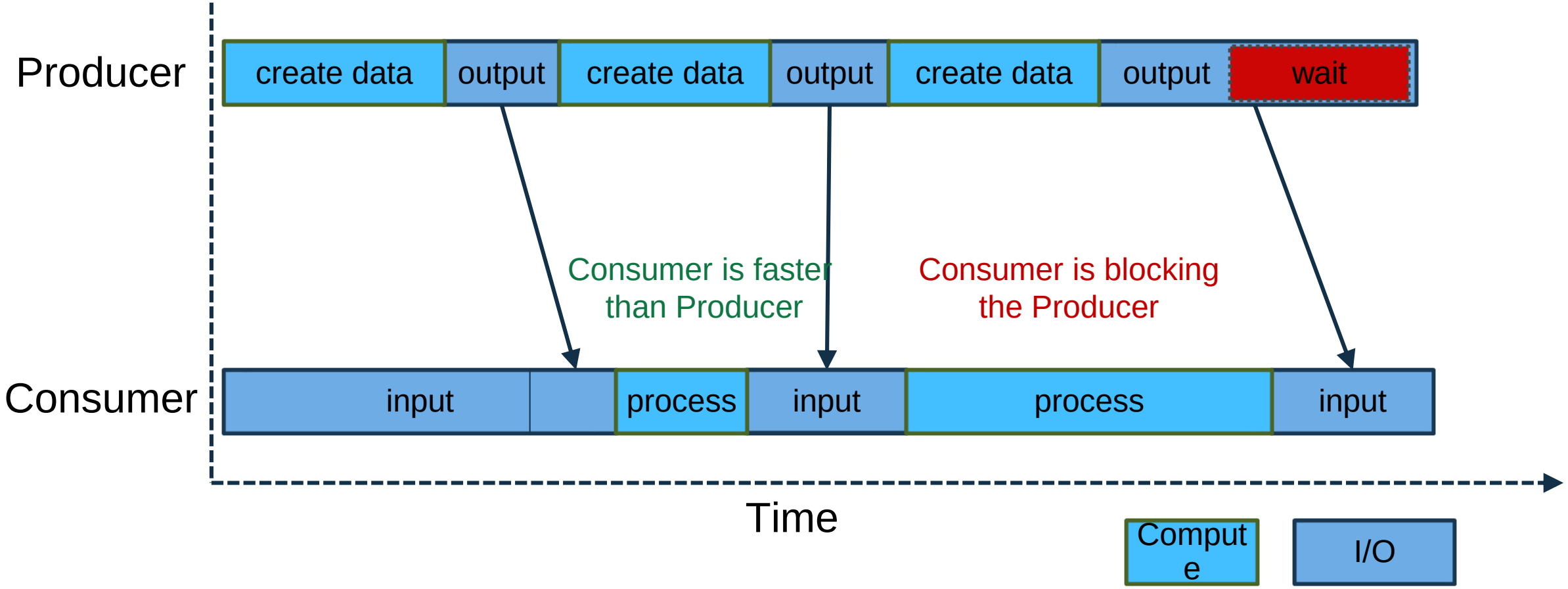


InSituMPI: Non-buffering, blocking data transfer

- Designed for
  - In situ analysis (one-way staging) for MPMD applications
  - Using MPI asynchronous point-to-point messaging (isend, irecv)
- Producer waits until the Consumer receives the data
  - Because it cannot overwrite the variables with new computation
- Optimized for memory usage
  - No buffering on producer side
  - With good read pattern, there is no temporary buffering on consumer side either
- Optimized for fixed write-read communication patterns
  - From second output step, all data is immediately sent to corresponding consumers
  - For non-fixed patterns, metadata is exchanged every step to establish the pattern
- Buffered version in development, using MPI one-sided operations (SSC)

InSituMPI engine

- Good for in situ analysis, but consumer may block producer



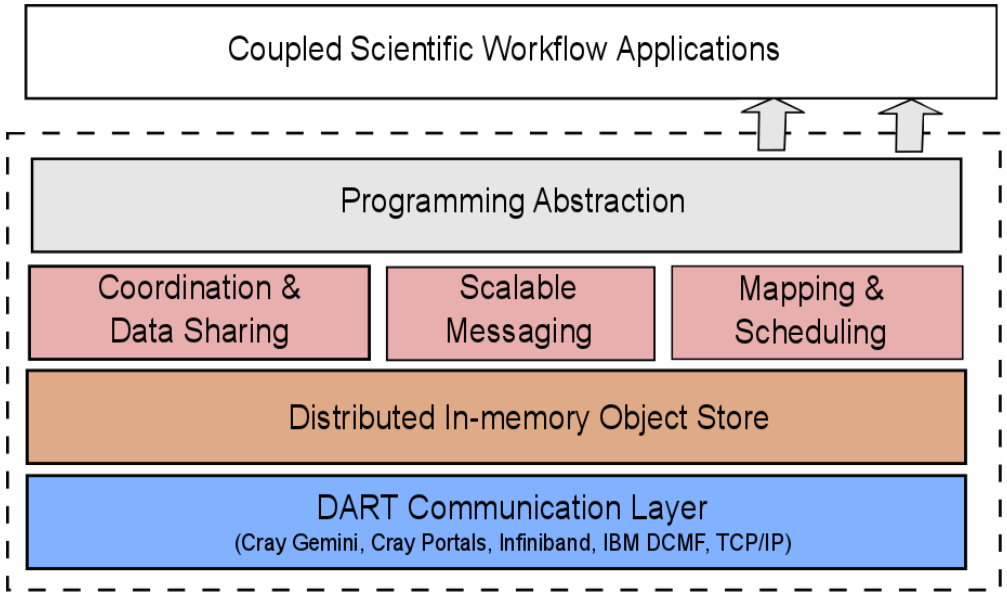
DataMan

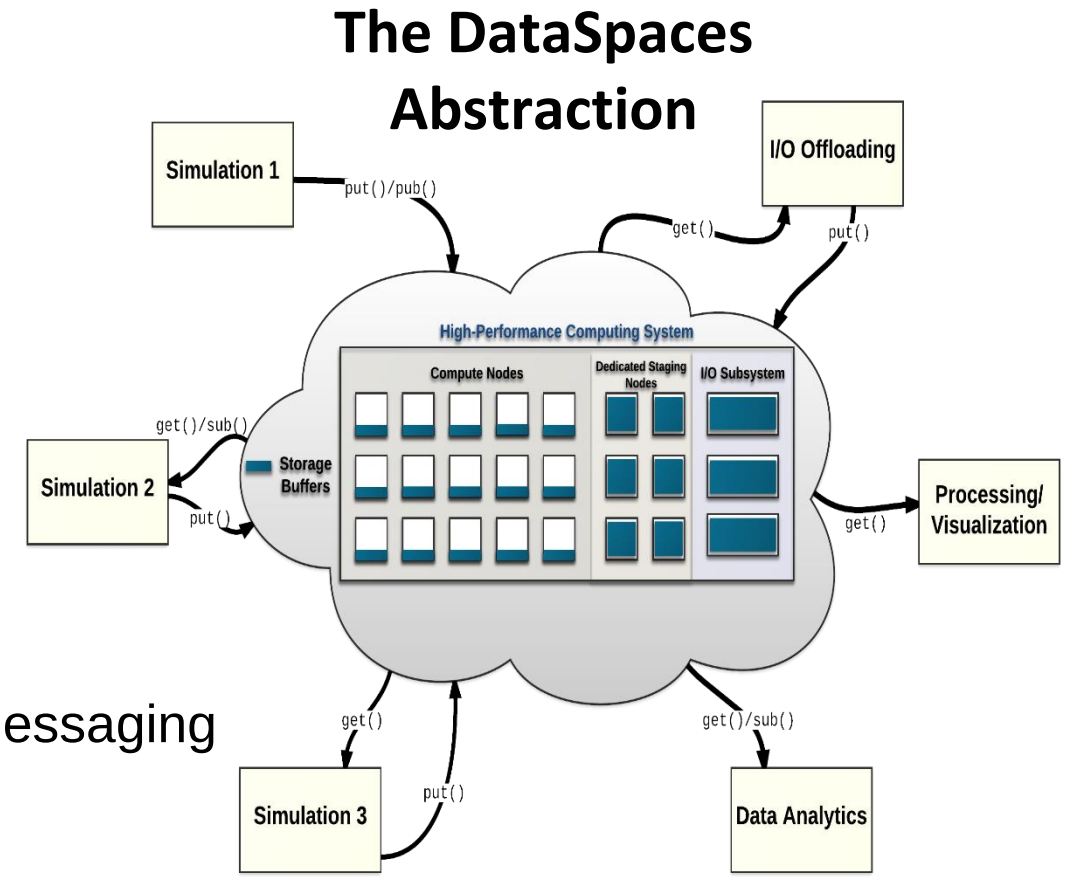
- An ADIOS 2 engine subroutine designed for wide area network data transfer, data staging, near-real-time data reduction and remote in-situ processing
- Designed with following principles:
  - Flexibility: allowing transport layer switching (ADIOS BP file, ZeroMQ, google-rpc etc.)
  - Versatility: supporting 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 2 transports and operators, and other potential third-party libraries. For example, DataMan can use ZFP, Bzip, SZ that have been built into ADIOS2, as well as any compression techniques that will be built into ADIOS 2 in future.
- 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



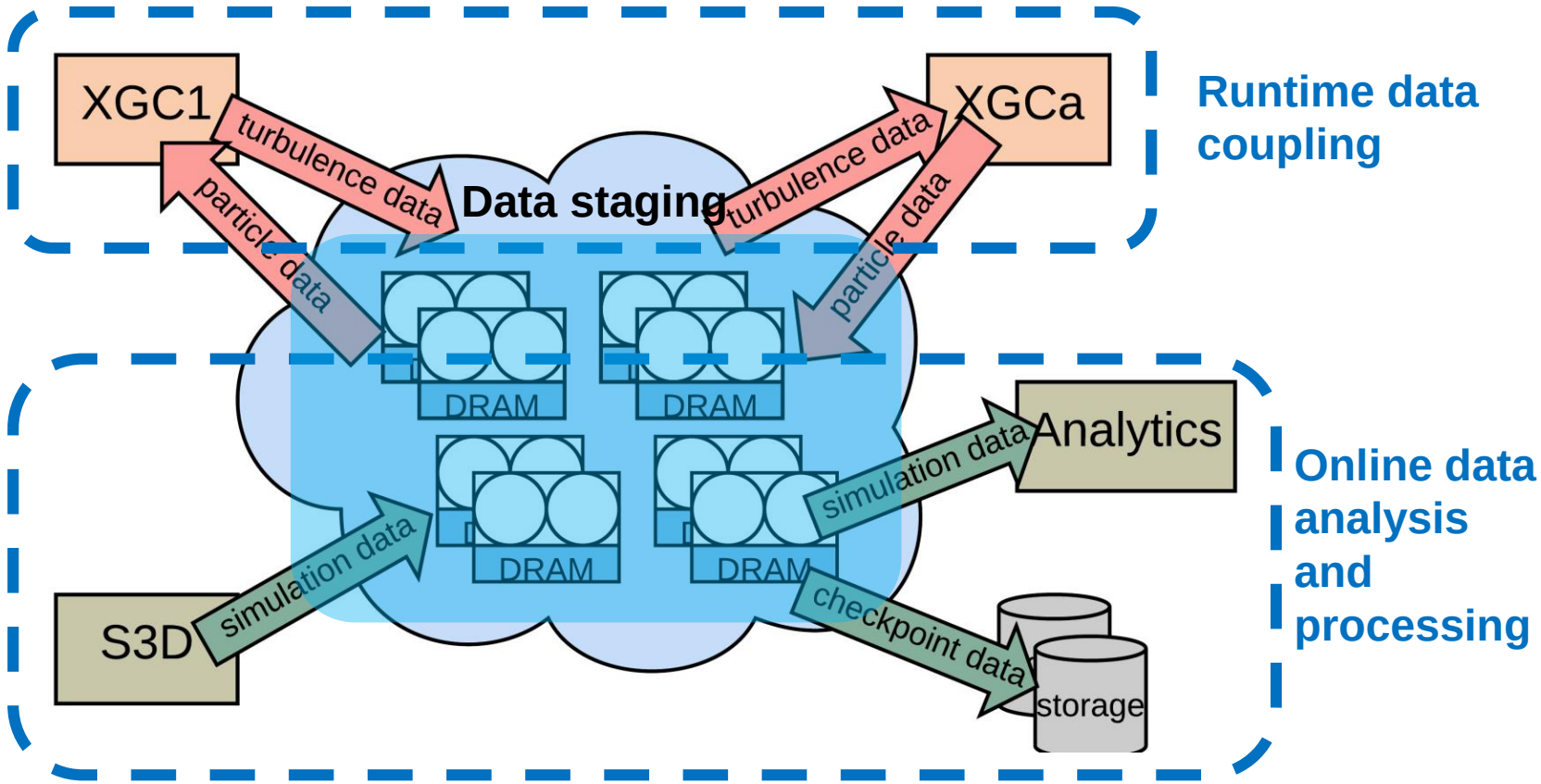


- Virtual shared-space programming abstraction
  - Simple API for coordination, interaction and messaging
- Distributed, associative, in-memory object store
  - Online data indexing, flexible querying
- Adaptive cross-layer runtime management
  - Hybrid in-situ/in-transit execution
- Efficient, high-throughput/low-latency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

www.dataspaces.org

The DataSpaces Staging Abstraction

- Storage distributed across memory hierarchy and a dynamic set of cores/node
- Enable in-staging data processing, querying sharing, and exchange



Application: Which staging method to use?

- System considerations
  - Staging between machines/over a WAN? **DataMan**
  - Co-located execution? **SST-SM+X**
  - Availability of RDMA high-speed network? **SST** (and **DataSpaces**) optimized for this case.
- Coupling considerations
  - Highly synchronized writer/reader and a highly optimized MPI? **InsituMPI**
  - 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.
- Performance considerations
  - Often application-specific, and not always obvious.
  - Here's where it helps to have a flexible I/O framework…

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

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.

Recall read API

- 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

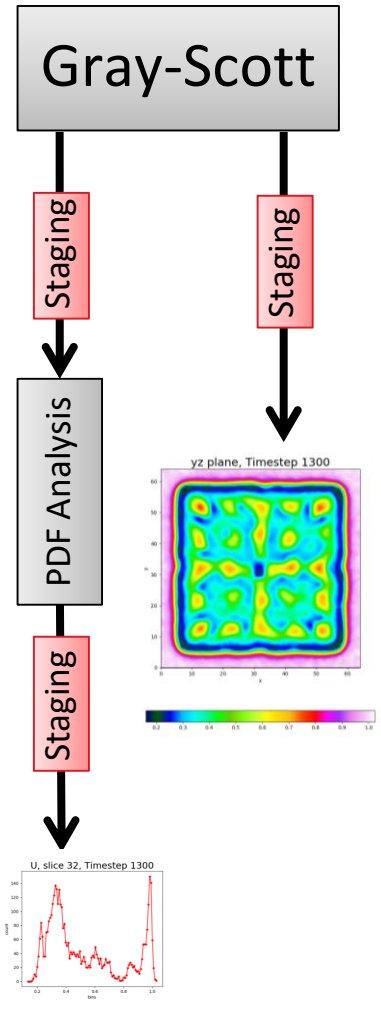
```
adios2::StepStatus read_status =
        reader.BeginStep(adios2::StepMode::Read, timeout);
if (read_status != adios2::StepStatus::OK) {
    break;
}
```

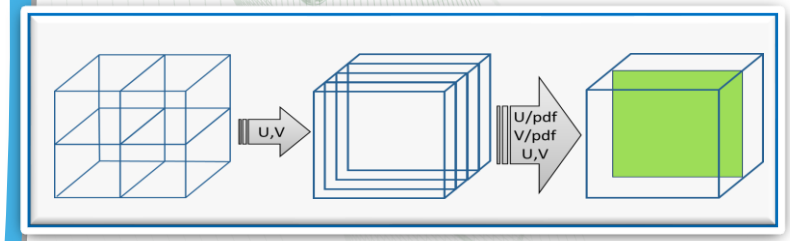
  - float timeout: wait for this long for a new step to arrive
- Release a step if not needed anymore
  - optimization to allow the staging method to deliver new steps if available

```
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

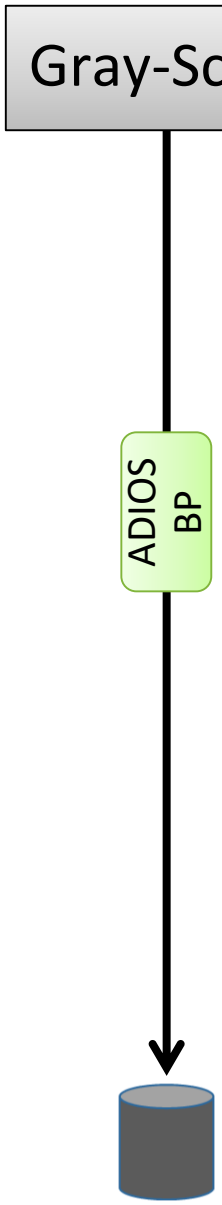
SST

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

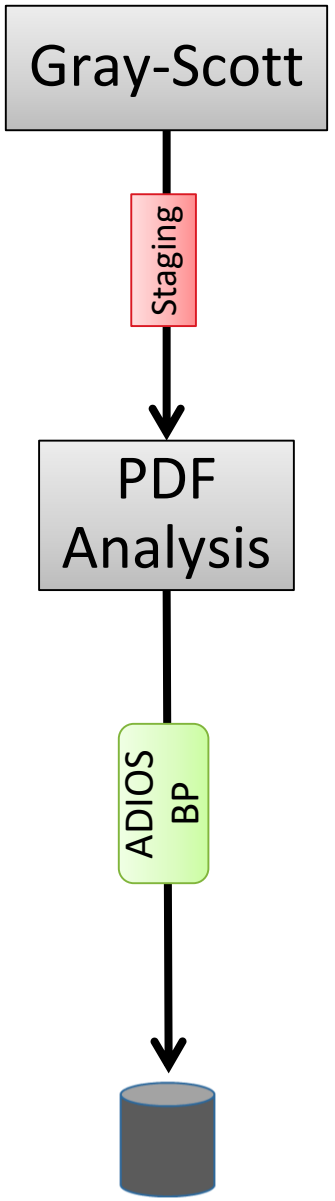


edit **adios2.xml: SimulationOutput, PDFAnalysisOutput** uses **BP4** engine

```
$ ./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
```

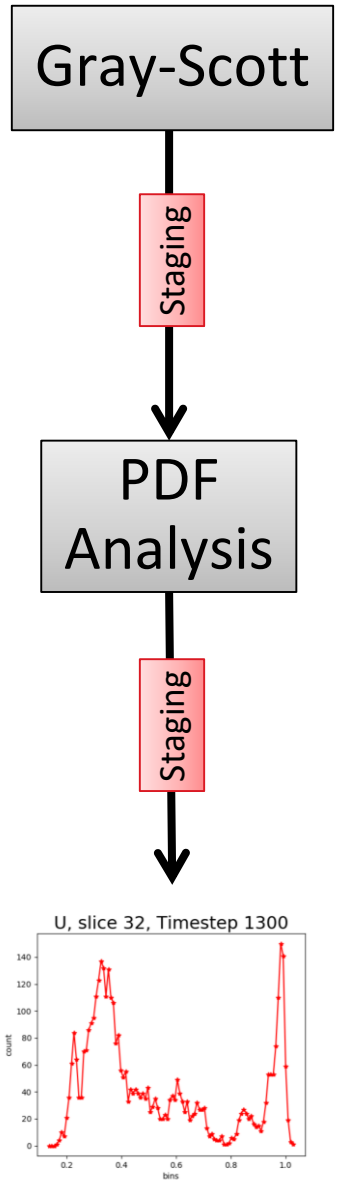
In Situ Visualization with ADIOS




```
$ 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

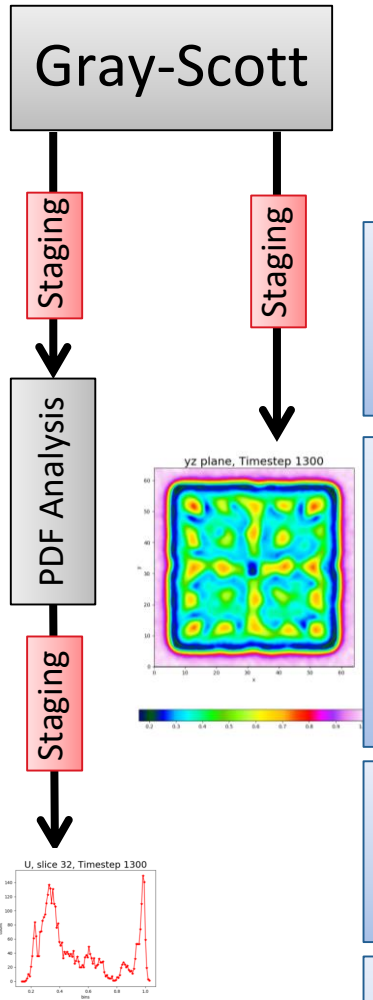



```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:                BP4
=========================================
grid:                 64x64x64
steps:                60000
 …
```

```
$ 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

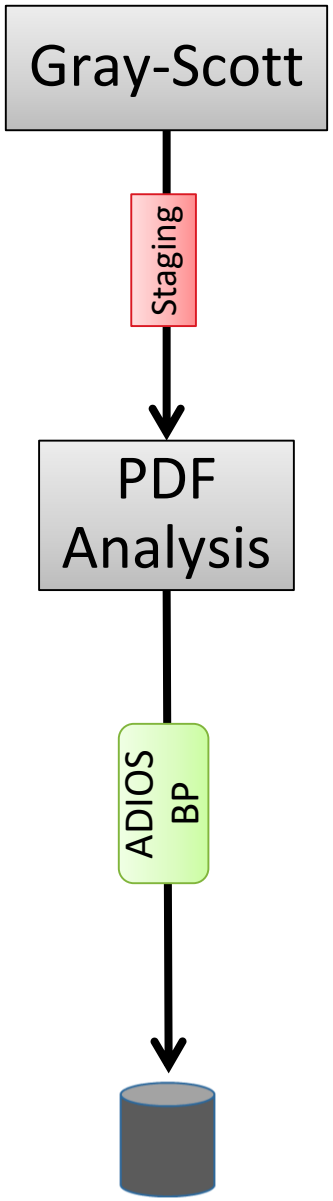
The runtime config file: adios2.xml

```
<?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

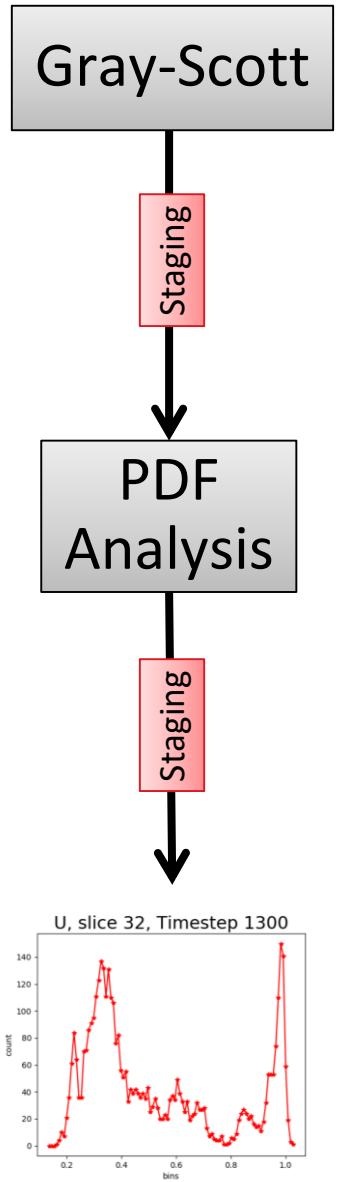



```
edit adios2.xml and change SimulationOutput to SST
                           PDFAnalysisOutput   to BP4
$ ./cleanup.sh data
```

```
$ 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

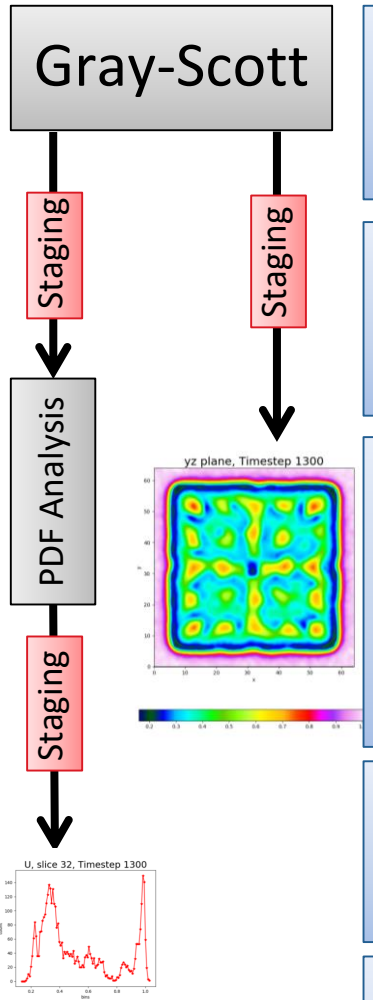
```
$ ./cleanup.sh data
```

```
$ 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
...
```

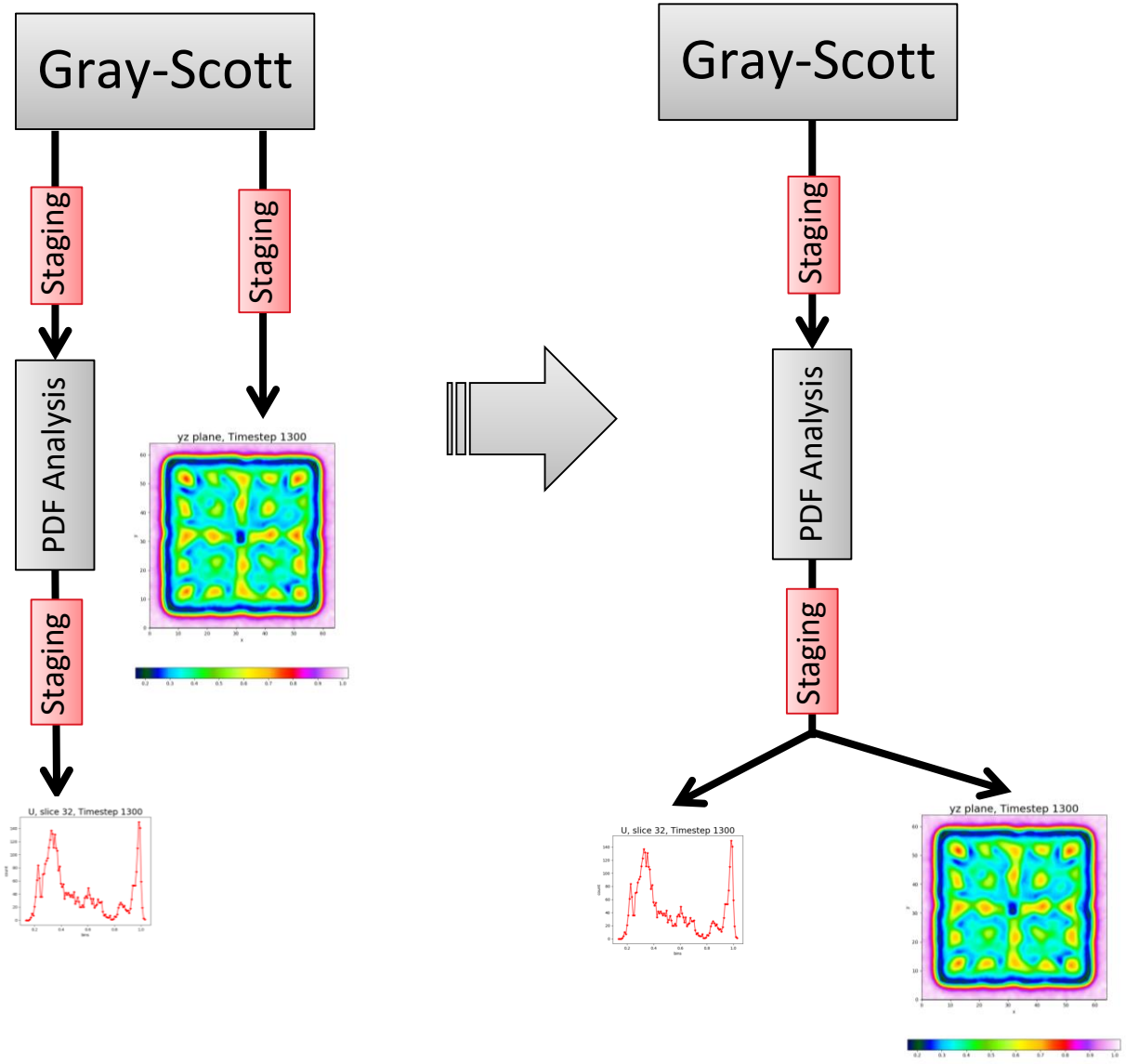
Home Work

- 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



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: Overview of self-describing I/O using HDF5 and ADIOS **(2.5 hours) (Beginner/Intermediate)**

- **Lecture:** HDF5 framework, I/O abstraction, file format
- **Lecture:** ADIOS framework, I/O abstraction, file format

Hands-on: use a parallel MiniApp to write self-describing data

- Use HDF5 write API to write data in parallel
- Use ADIOS write API to write data in parallel

BREAK

- Write HDF5 files using the ADIOS API
- Read HDF5 files using the HDF5 API

Hands-on: Parallel data reading

- ADIOS read API in Fortran90, C++, and Python
- Using the ADIOS API to read HDF5 data

LUNCH

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 examples in previous section for writing and reading

Part IV: In situ data analysis using I/O staging **(2 hours) (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

BREAK

- **Hands-on**: Staging and converting with adios_reorganize tool
- **Demonstration**: Additional data staging methods

Part V: Data Indexing and Querying (0.25 hour) **(Intermediate/Advanced)**

- **Lecture**: Introduction of indexing data in ADIOS and HDF5

Wrap-up

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

Demonstration

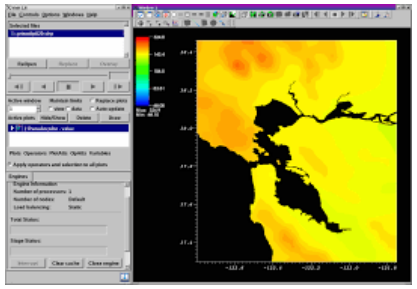
In situ visualization with Visit and ParaView

using the Gray-Scott MiniApp

Visualization Services Software Stack

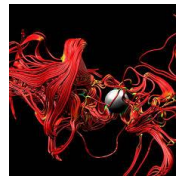
VISIT

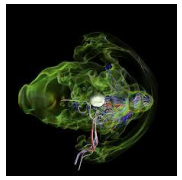
ParaView



Interactive Visualization Tools

Visualization Micro Services





Portable Visualization Algorithms

VTK-m

Data Model

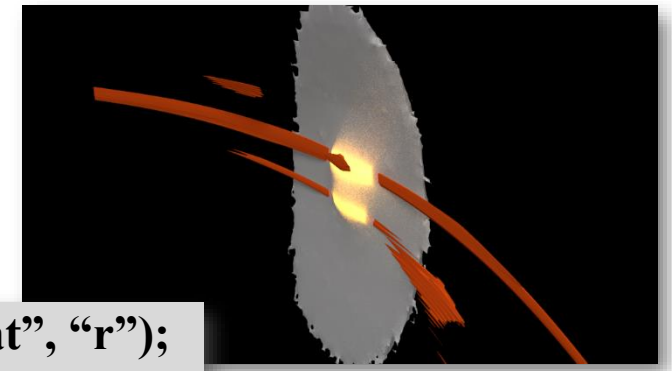
ADIS

Data Management and Movement

ADIOS

Data Management and Movement: ADIOS

Leverages something already being done: **File I/O**



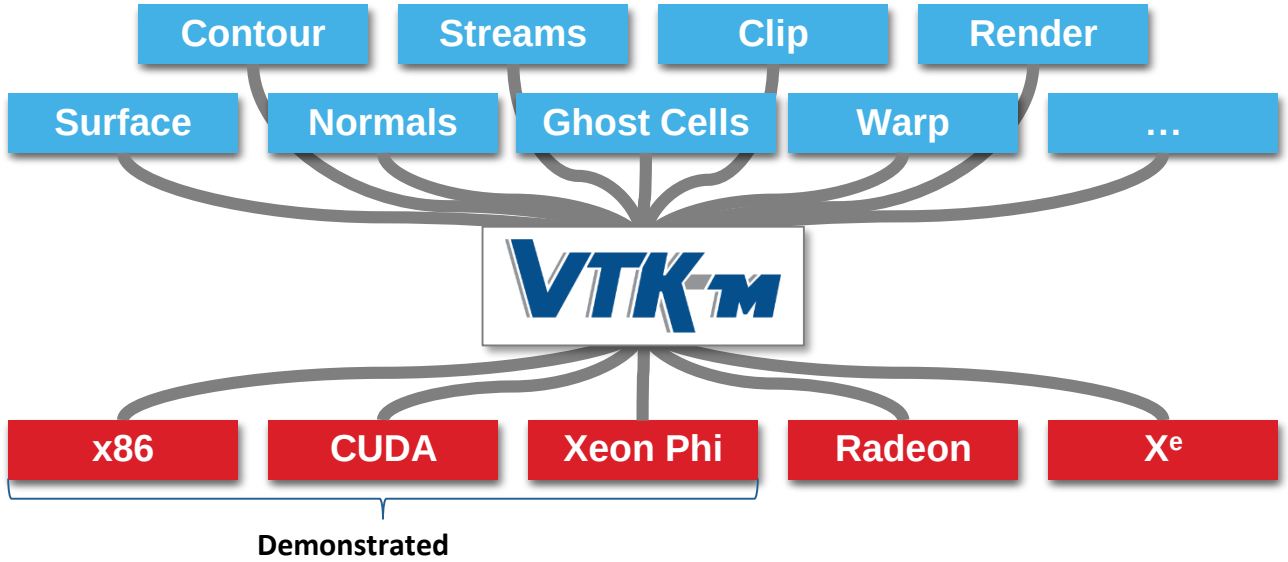
```
fp = fopen("out.dat", "w");
fwrite(fp, ……);
```

```
fp = fopen("out.dat", "r");
fread(fp, ……);
```

VTK-m Scientific Visualization Toolkit

Scientific Visualization Toolkit for heterogenous architectures

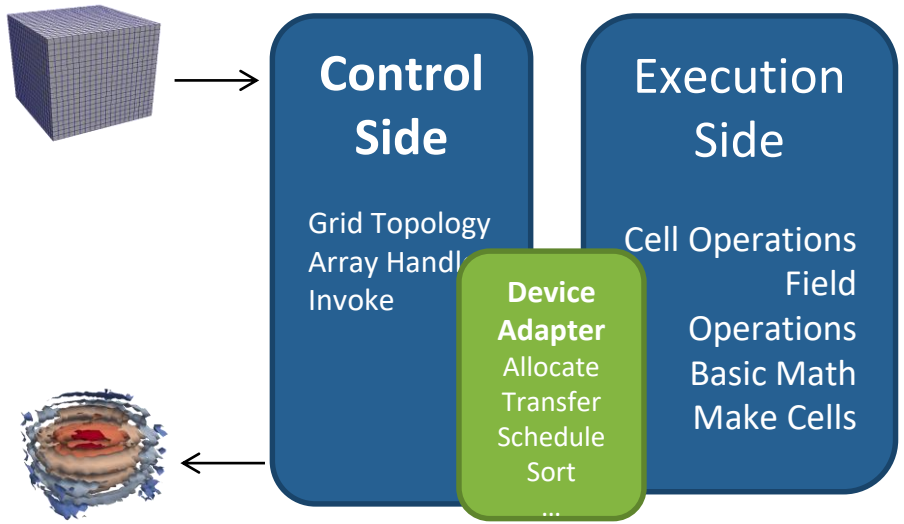
Overall strategy: Write once, run everywhere



System Overview:

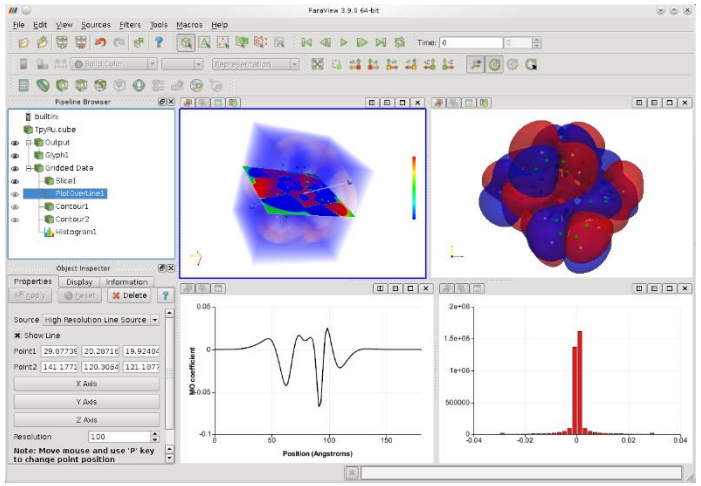
Control side specification of grids and data parallel operations

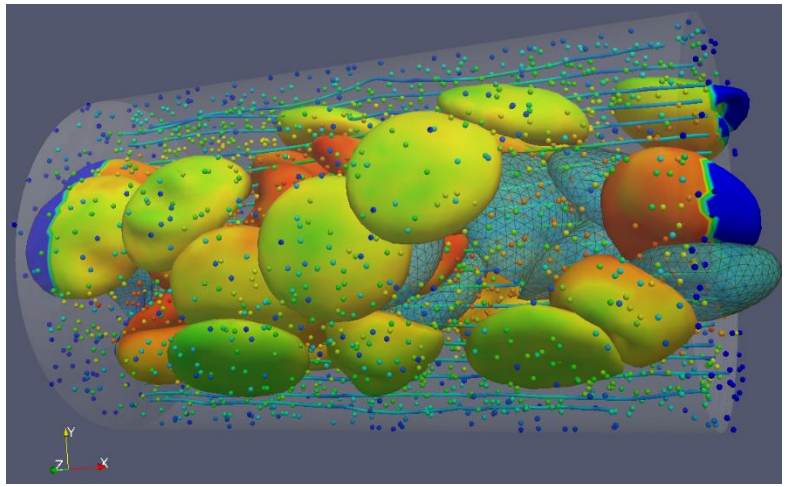
Execution side mapping of operations onto specific hardware

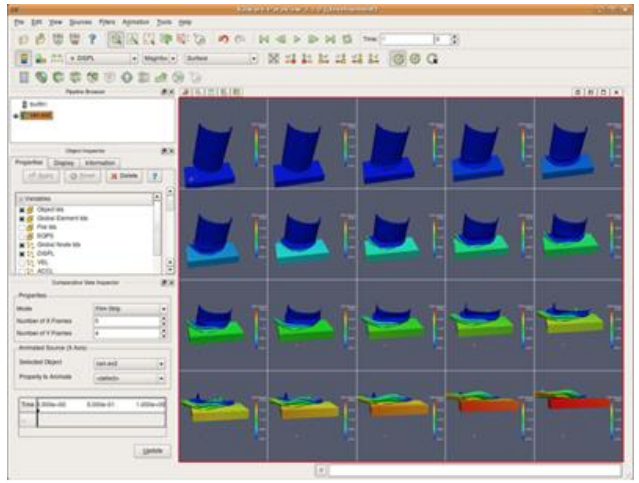


Paraview

ParaView is an **open-source**, **multi-platform**, **data analysis and visualization** application for analyzing **extremely large datasets** using desktop & distributed memory computing resources.
See https://paraview.org

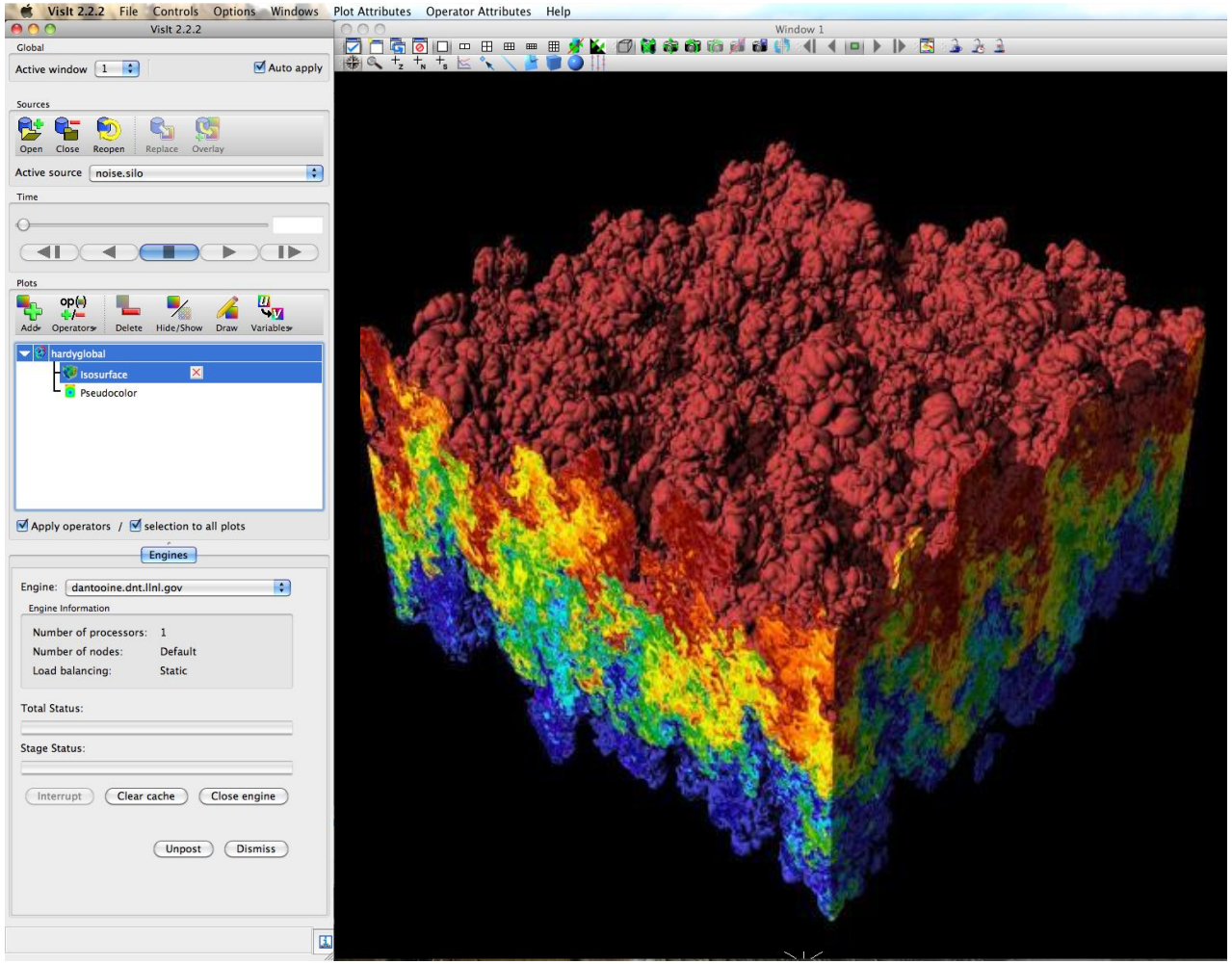




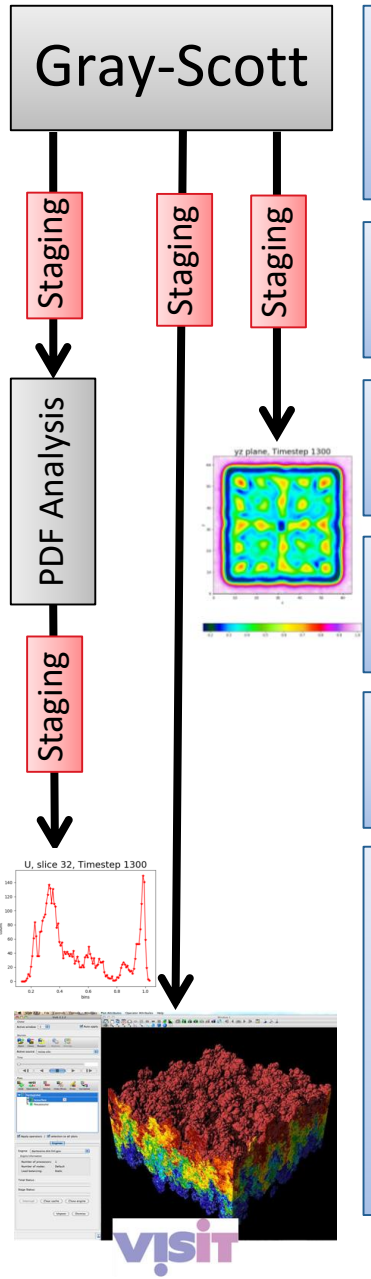


VisIt

- Production end-user tool supporting scientific and engineering applications
- Provides an infrastructure for parallel post-processing that scale from desktops to massive HPC systems
- Source released under a BSD style license
- See: **visitusers.org**



In Situ Visualization with ADIOS




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

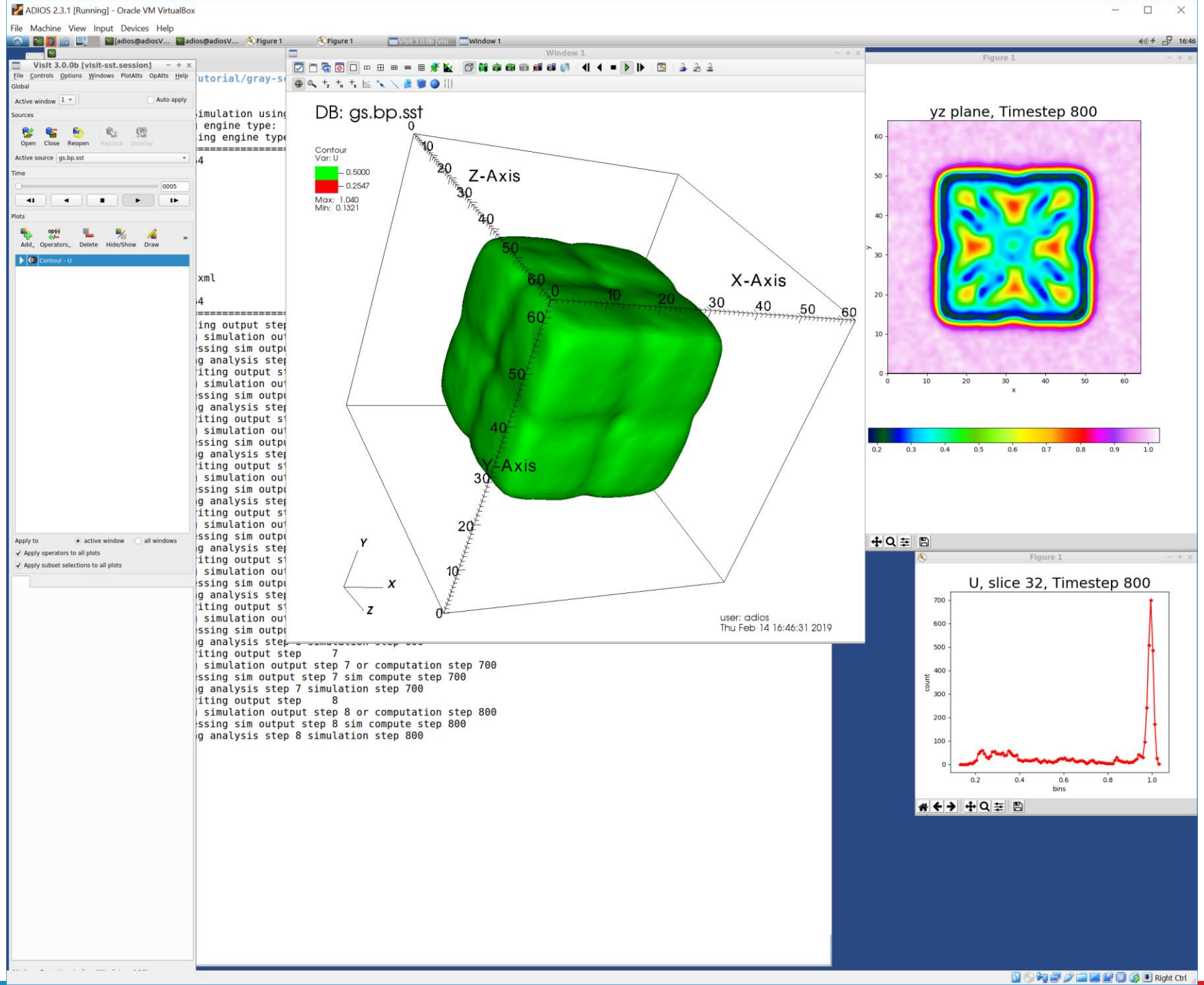
```
$ mpirun -n 4 adios2-gray-scott settings-staging.json
 Simulation writes data using engine type:                SST
```

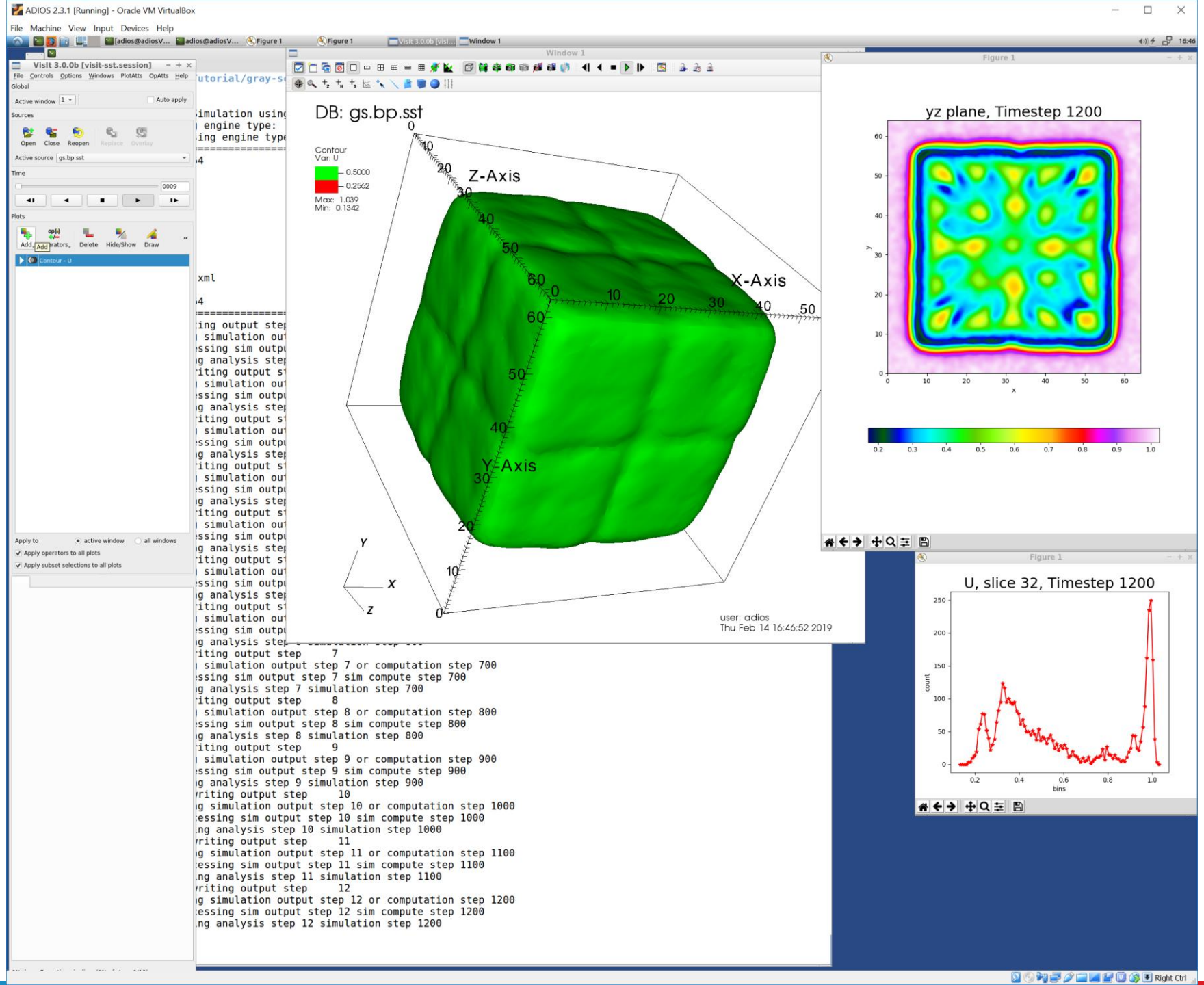
```
$ mpirun -n 2 adios2-pdf-calc gs.bp pdf.bp 100
PDF analysis reads from Simulation using engine type:  SST
```

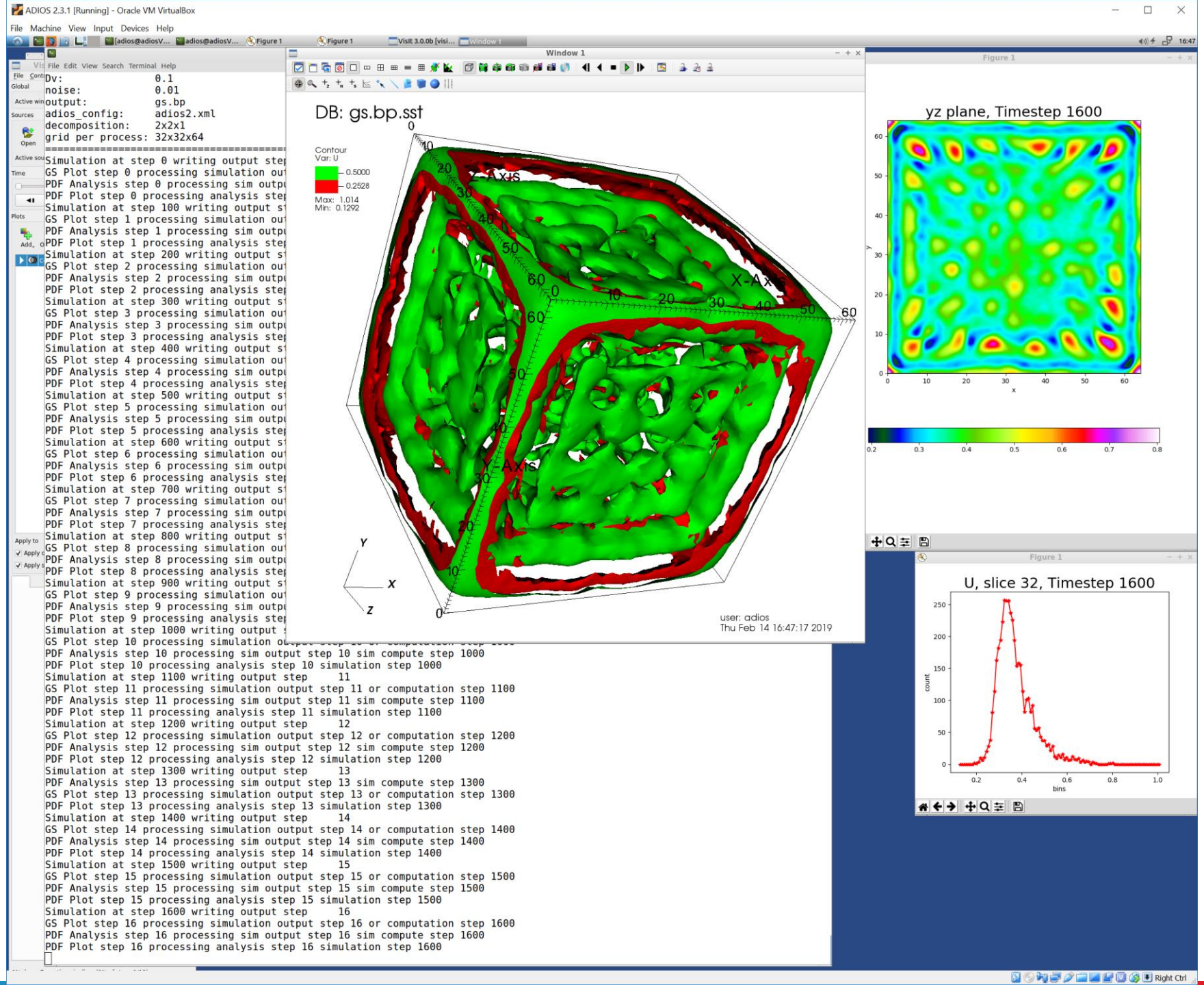
```
$ python3 pdfplot.py -i pdf.bp
PDF Plot step 0 processing analysis step 0 simulation step 0
```

```
$ 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
ssing sim output step 7 sim compute step 700
g analysis step 7 simulation step 700
iting 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


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
simulation output step 7 or computation step 700
ssing sim output step 7 sim compute step 700
g analysis step 7 simulation step 700
iting 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
iting 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
riting 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
riting 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
riting 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
PDF Analysis step 0 processing sim output
PDF Plot step 0 processing analysis step
Simulation at step 100 writing output step
GS Plot step 1 processing simulation output
PDF Analysis step 1 processing sim output
PDF Plot step 1 processing analysis step
Simulation at step 200 writing output step
GS Plot step 2 processing simulation output
PDF Analysis step 2 processing sim output
PDF Plot step 2 processing analysis step
Simulation at step 300 writing output step
GS Plot step 3 processing simulation output
PDF Analysis step 3 processing sim output
PDF Plot step 3 processing analysis step
Simulation at step 400 writing output step
GS Plot step 4 processing simulation output
PDF Analysis step 4 processing sim output
PDF Plot step 4 processing analysis step
Simulation at step 500 writing output step
GS Plot step 5 processing simulation output
PDF Analysis step 5 processing sim output
PDF Plot step 5 processing analysis step
Simulation at step 600 writing output step
GS Plot step 6 processing simulation output
PDF Analysis step 6 processing sim output
PDF Plot step 6 processing analysis step
Simulation at step 700 writing output step
GS Plot step 7 processing simulation output
PDF Analysis step 7 processing sim output
PDF Plot step 7 processing analysis step
Simulation at step 800 writing output step
GS Plot step 8 processing simulation output
PDF Analysis step 8 processing sim output
PDF Plot step 8 processing analysis step
Simulation at step 900 writing output step
GS Plot step 9 processing simulation output
PDF Analysis step 9 processing sim output
PDF Plot step 9 processing analysis step
Simulation at step 1000 writing output step
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
X-Axis
Y-Axis
Y
X
Z
user: adios
Thu Feb 14 16:47:17 2019
Figure 1
yz plane, Timestep 1600
Figure 1
U, slice 32, Timestep 1600
count
bins

In-line processing

- Demo of using the ADIOS-API with in-line processing of gray-scott data
  - Gray-scott + VTK-isosurface visualization
    - Memory is directly shared between the simulation and the visualization
    - Visualization images created, which we will view in Paraview
- We will also show Paraview reading ADIOS-BP4 data and visualizing the data from Gray-Scott

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

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
```

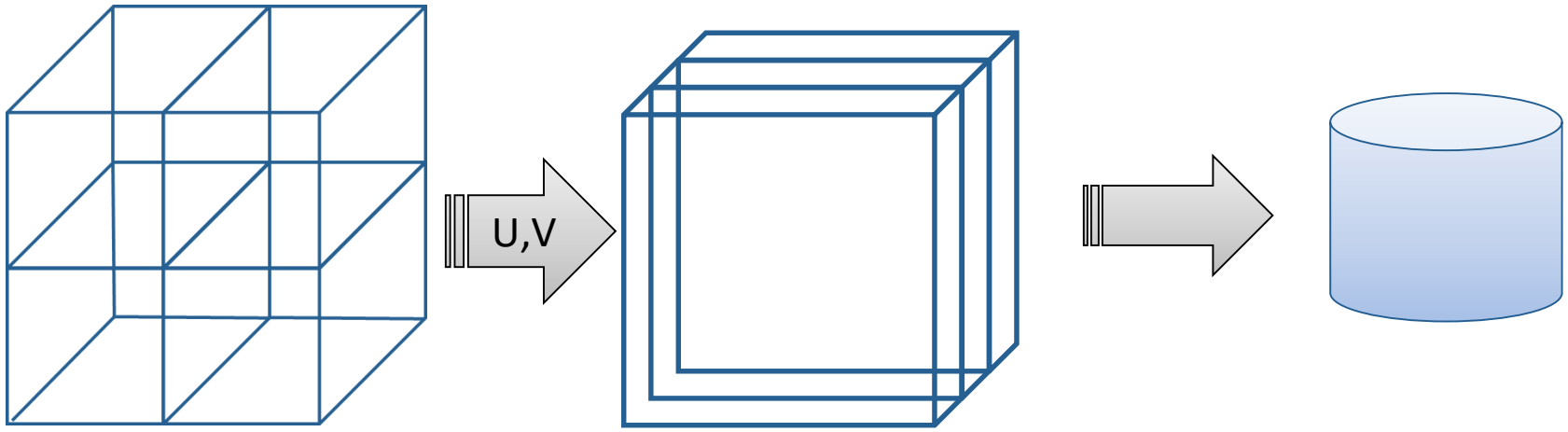
In another terminal

```
$ 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)