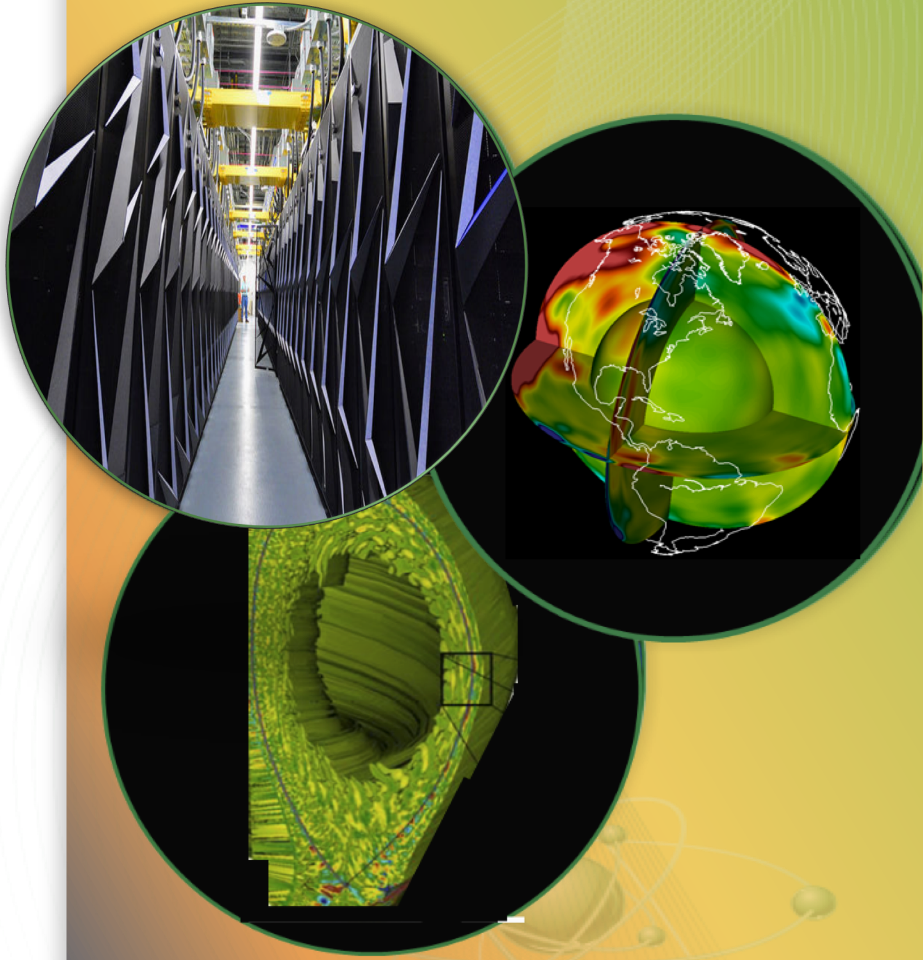
ADIOS 2Tutorial

IXPUG Software-Defined Visualization Workshop

7-10-2018

Norbert Podhorszki

Scientific Data Group
Scott Klasky (Group Leader)
Matthew Wolf (Deputy)
Scientific Data Management
Norbert Podhorszki –TL
Mark Ainsworth
Jong Choi
William Godoy
Tahsin Kurc
Qing Liu
Jeremy Logan
Kshitij Mehta
Eric Suchyta
Ruonan Wan
Jason Wang
Scientific Data Analytics
Dave Pugmire – TL
Mark Kim
James Kress
George Ostrouchov



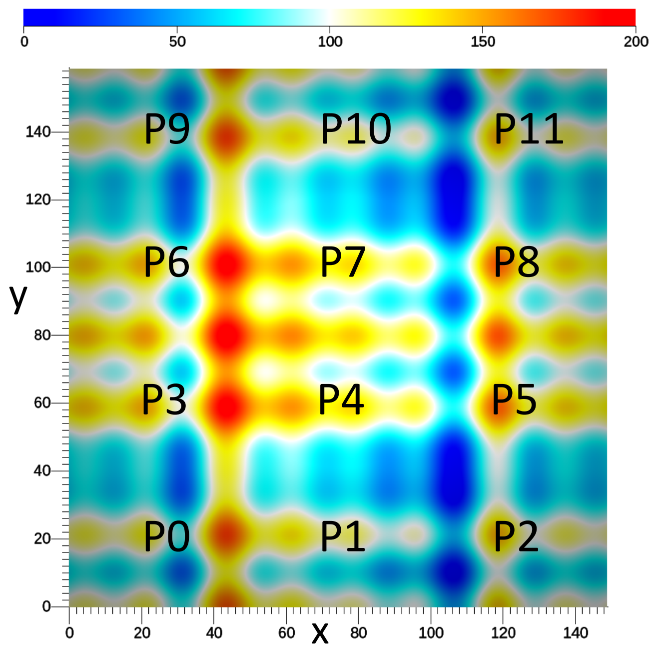
Georgia Tech , Rutgers, Kitware, ParaTools, HDF, PPPL, Sandia, LBNL, ANL, BNL, Oregon, Rutgers,, ++

Example

Heat Transfer 2D,
Fortran to C++ to Python pipeline

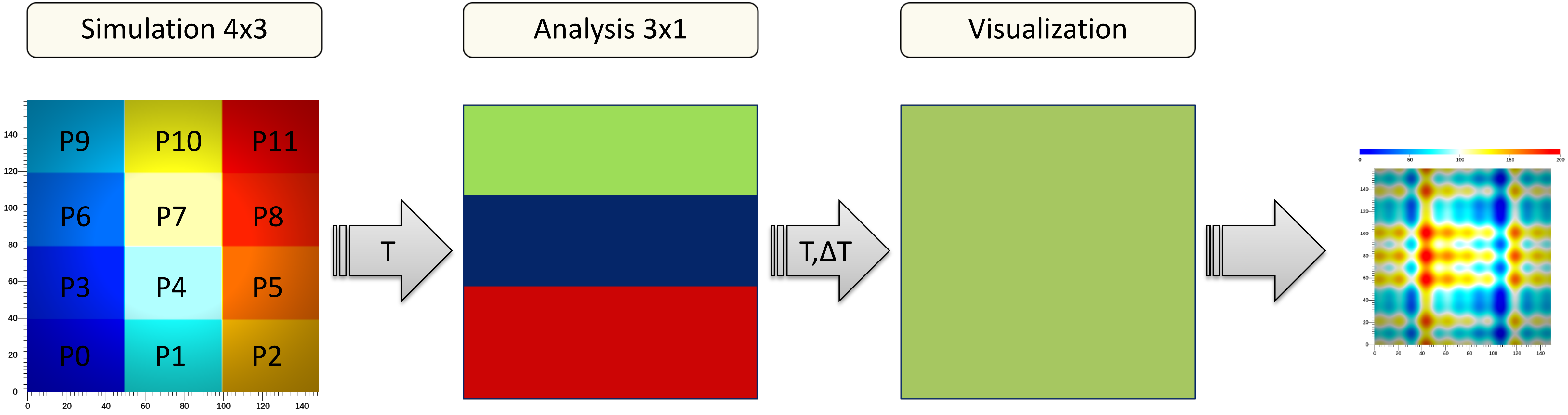
Heat Transfer Example

- In this example we start with a 2D code which writes data of a 2D array, with a 2D domain decomposition, as shown in the figure.
  - Heat transfer example with heating the edges
  - We write multiple time-steps, into a single output.
- For simplicity, we work on only 12 cores, arranged in a 4 x 3 arrangement.
- Each processor works on 40x50 subsets
- The total size of the output array = 4*40 x 3*50



Analysis and visualization

- Read with a different decomposition (1D)
  - Calculate ΔT
  - Write from 3 cores, arranged in a 3 x 1 arrangement.
- Plot T
  - image files

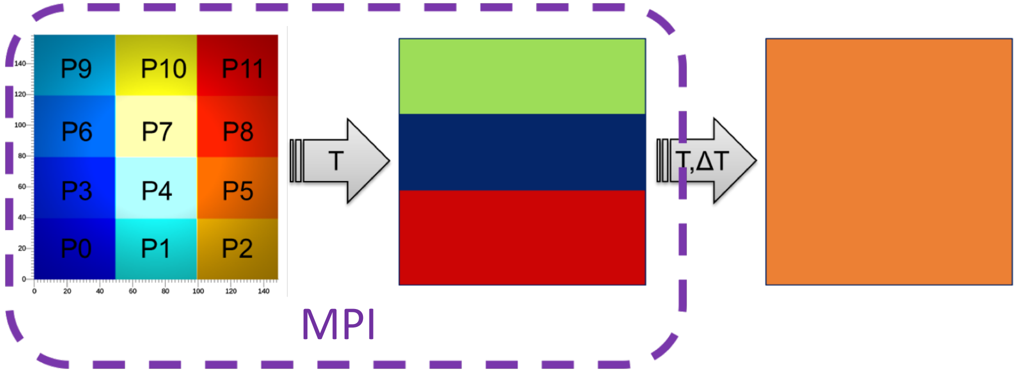


klasky@ornl.gov

Running the example in situ

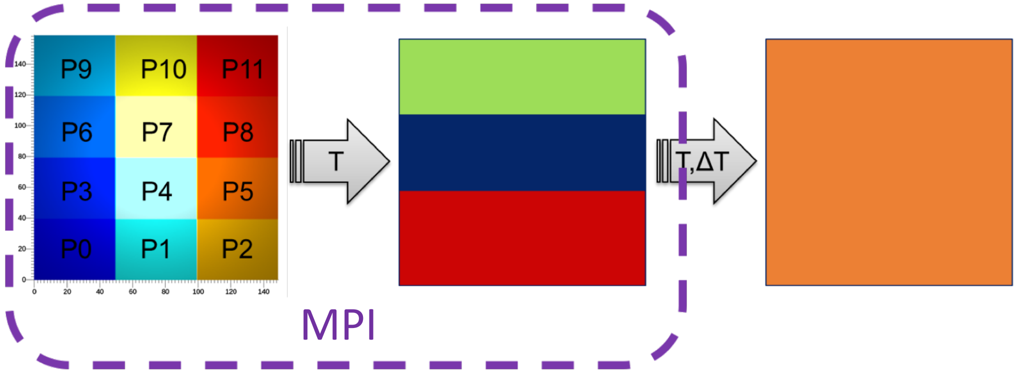
```
$ mpirun
-n 12 simulation/heatSimulation_adios2 heat
  4 3  40 50  100 1000
:
-n  3 ../cpp/heatAnalysis heat.bp analysis.bp   3 1
Simulation step 0: initialization
Analysis step 0 processing simulation st
Simulation step 1
Analysis step 1 processing simulation st
Simulation step 2
Analysis step 2 processing simulation st
Simulation step 3
Analysis step 3 processing simulation st
...
Simulation step 37
Analysis step 37 processing simulation s
Simulation step 38
Analysis step 38 processing simulation step 38
...
```

```
$ mpirun -n  1 ../python/heat_plot.py --in analysis.bp --out p
step:0, rank: 0, avg: 94.174, std: 32.927
step:1, rank: 0, avg: 97.161, std: 8.850
step:2, rank: 0, avg: 98.286, std: 4.823
^C
$ mpirun -n 1 ../python/heat_plot.py -i analysis.bp -o p
step:8, rank: 0, avg: 99.596, std: 0.549
step:9, rank: 0, avg: 99.663, std: 0.389
step:10, rank: 0, avg: 99.717, std: 0.285
^C
$ mpirun -n 1 ../python/heat_plot.py -i analysis.bp -o p
step:41, rank: 0, avg: 99.998, std: 0.001
step:42, rank: 0, avg: 99.998, std: 0.001
step:43
```



Running the example in situ

```
$ mpirun  -n 12 simulation/heatSimulation_adios2 heat   4  3   40  50   100  1000   : \
          -n   3 ../cpp/heatAnalysis heat.bp analysis.bp    3  1                    : \
          -n   1 ../python/heat_plot.py --in analysis.bp
Simulation step 0: initialization
Analysis step 0 processing simulation step 0
step:0, rank: 0, avg: 94.174, std: 32.927
Simulation step 1
Analysis step 1 processing simulation step 1
step:1, rank: 0, avg: 97.161, std: 8.850
Simulation step 2
Analysis step 2 processing simulation step 2
step:2, rank: 0, avg: 98.286, std: 4.823
Simulation step 3
Analysis step 3 processing simulation step 3
step:3, rank: 0, avg: 98.872, std: 2.885
...
```



How to develop such a pipeline?

- Most of us need testing and debugging
- Multiple teams may develop the separate applications

- Let's write the Simulation code first
- Output to files and examine the content
- Write the Analysis next and test it with reading from files
- Output to files and examine the content
- Write the Visualization and test it with reading from files
- Run some/all together at once with staging

Codes used for the tutorial

ADIOS 2.x source
https://github.com/ornladios/ADIOS2

Heat Transfer 2D example
https://github.com/pnorbert/adiosvm/tree/master/Tutorial/heat2d/cpp

OAK RIDGE
National Laboratory

ADIOS Overview

The need for online data analysis and reduction

Traditional approach:
Simulate, output, analyze

- Write simulation output to secondary storage; read back for analysis
- Decimate in time when simulation output rate exceeds output rate of computer

- Impossible to deal with!

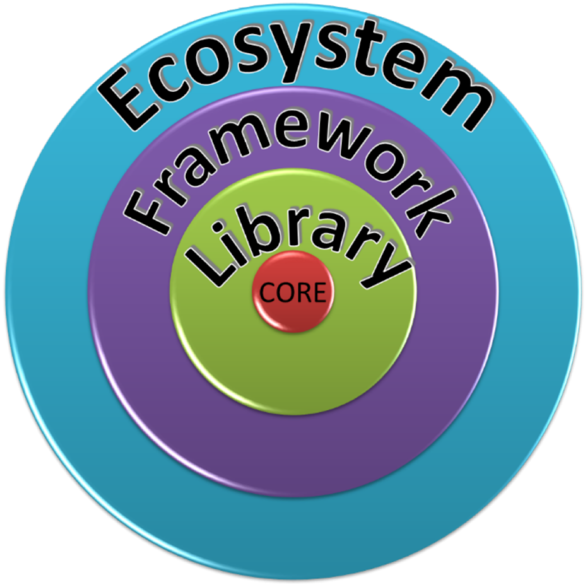
New approach:
Online data analysis and reduction

- Co-optimize simulation, analysis, reduction for performance and information output
- Substitute CPU cycles for I/O, via data (de)compression and/or online data analysis

Right bytes
in **right place**
at **right time**

What is ADIOS

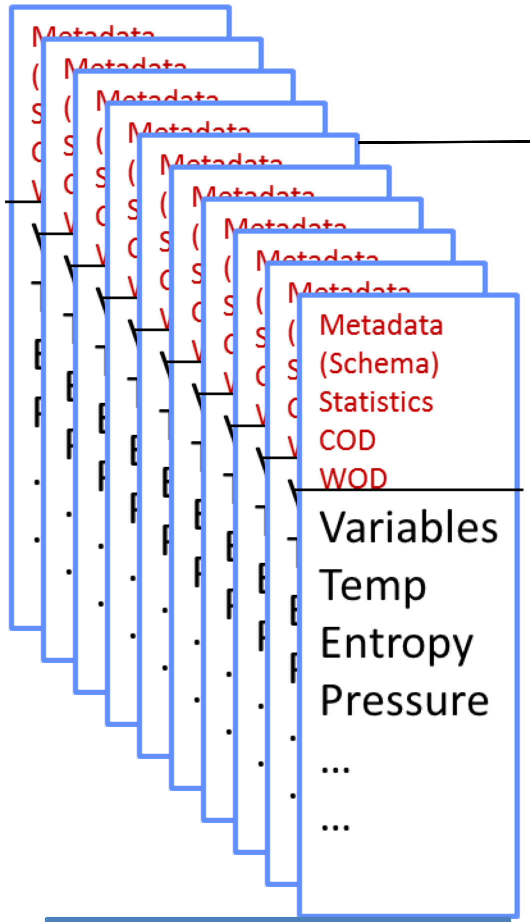
- An extendable **framework** that allows developers to *plug-in*
  - **I/O methods**: Aggregate, Posix, MPI
  - **Services**: Compression, Decompression
  - **File Formats**: HDF5, netcdf, …
  - **Stream** Format: ADIOS-BP
  - **Plug-ins**: Analytic, Visualization
  - **Indexing**: FastBit, ISABELLA-QA
- Incorporates the “best” practices in the I/O middleware layer
- Incorporates self describing data streams and files
  - https://www.olcf.ornl.gov/center-projects/adios/, https://github.com/ornladios/ADIOS
- Available at ALCF, OLCF, NERSC, CSCS, Tianhe-1,2 Pawsey SC, Ostrava



- ADIOS core – provides the basic infrastructure
  - BP stream format, Memory Buffering, Data Movement strategies
- ADIOS library - allow “best practice” from external components
  - Engines, Transformations, Indexing, Transports
- ADIOS Framework – allow scientific libraries to be used inside ADIOS
  - Staging libraries, reduction libraries, Indexing libraries, I/O libraries
- ADIOS ecosystem – Allow applications to interact with ADIOS codes/data
  - Analysis- Visualization services, Performance services, Living Miniapps

Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, et al.. Hello adios: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience* **2014**, *26*, 1453–1473.

Create a next-generation file/stream format

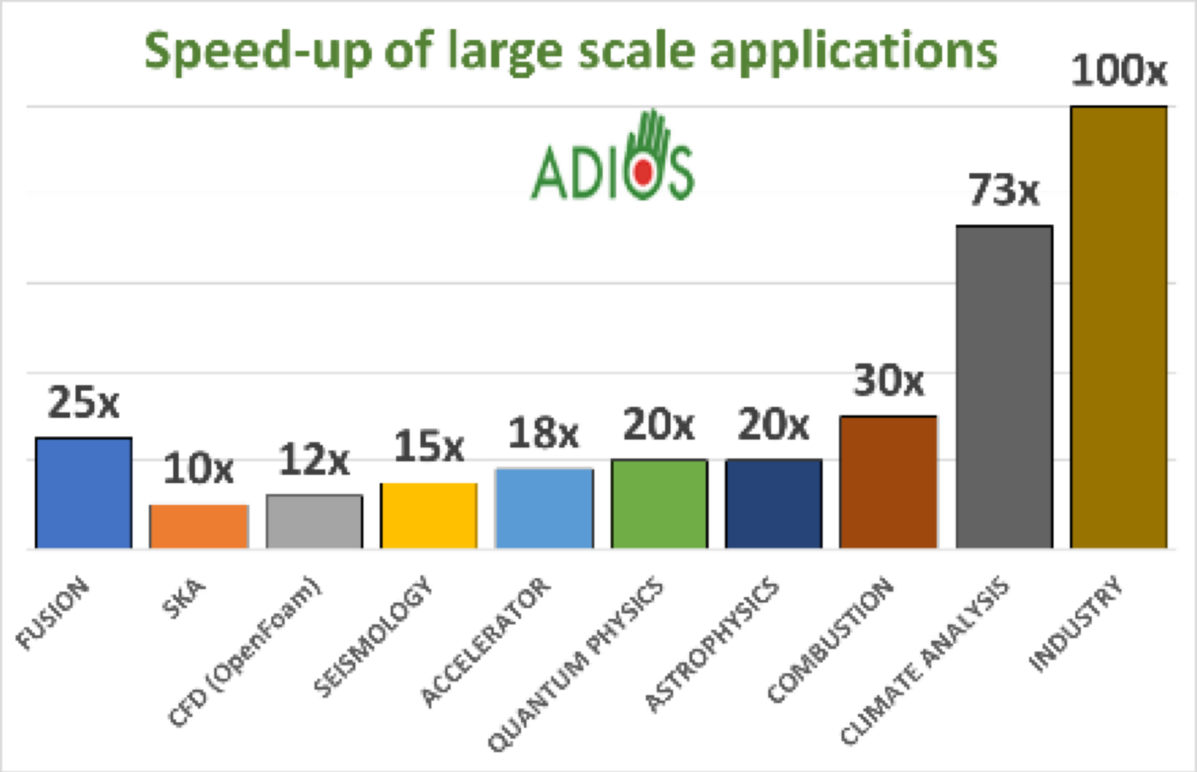


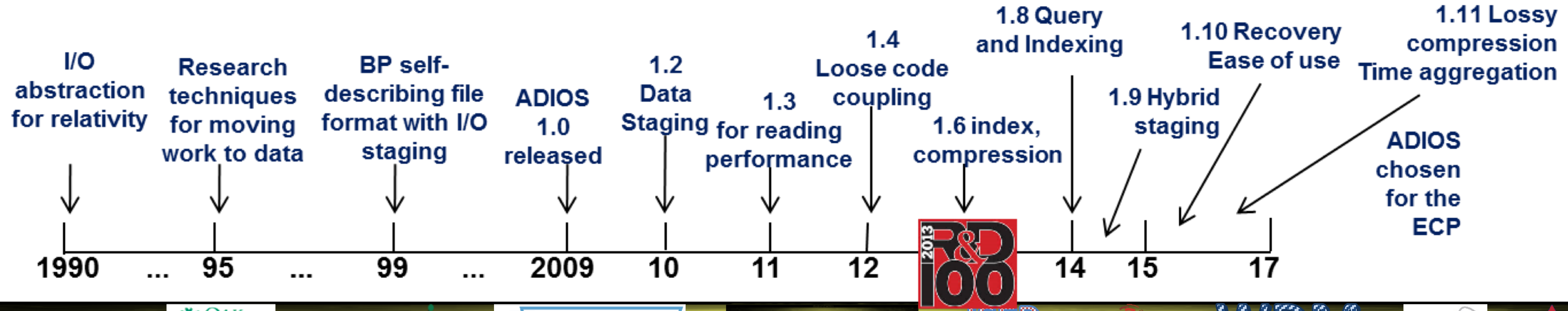
Ensemble of chunks = file

- All data chunks are from a single producer
  - MPI process, Single diagnostic
- Ability to create a separate metadata file when “sub-files” are generated
- Allows variables to be individually compressed
- Has a schema to introspect the information
- Has workflows embedded into the data streams
- Format is for “data-in-motion” and “data-at-rest”
- Log-like data format

J. Lofstead, F. Zheng, S. Klasky, K. Schwan, *Adaptable, metadata rich IO methods for portable high performance IO* in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, pp. 1–10.

I/O Framework for Data Intensive Science

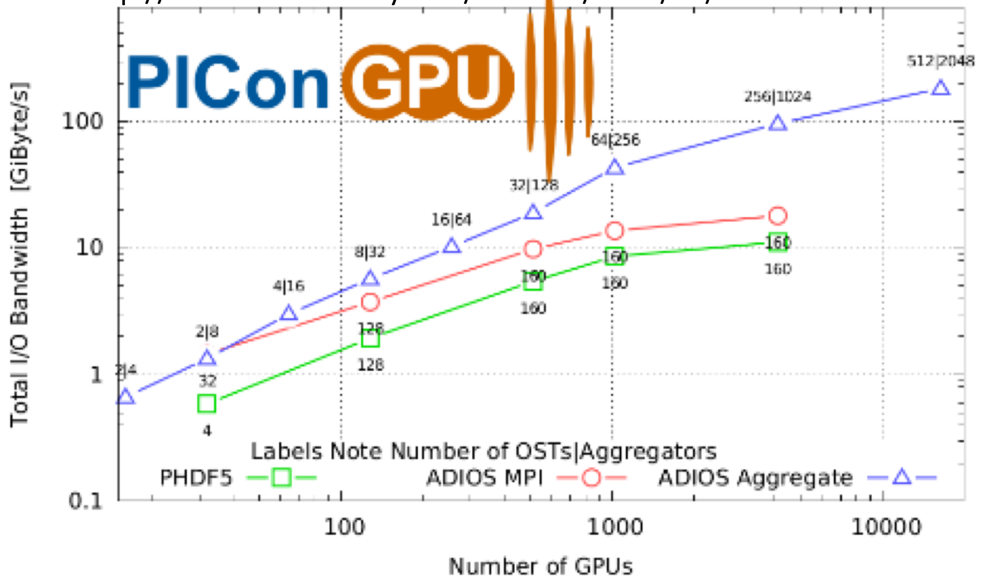


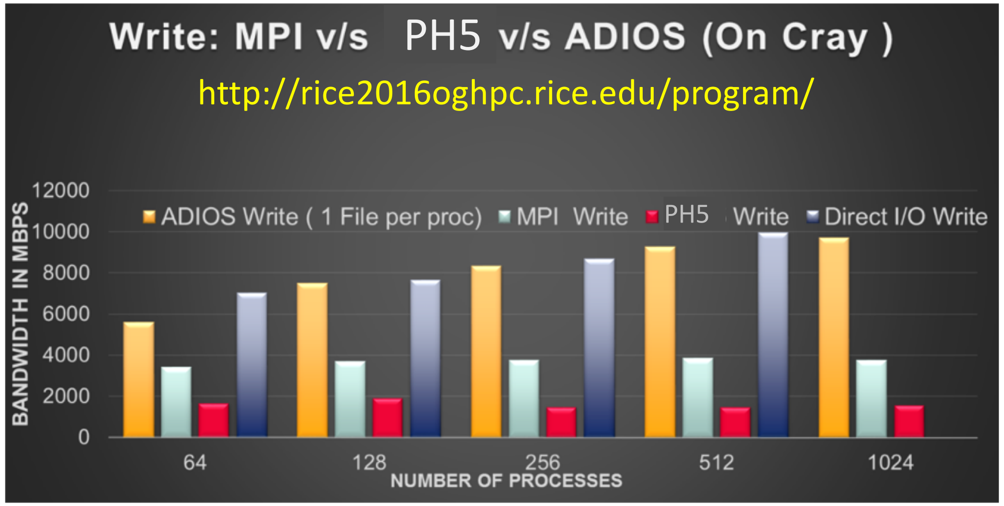


Impact to LCF applications

- Accelerators – PIConGPU
  - M. Bussmann, et al. - HZDR
  - Study laser-driven acceleration of ion beams and its use for therapy of cancer
  - Computational laboratory for real-time processing for optimizing parameters of the laser
  - Over 200 GB/s on 16K nodes on Titan
- Seismic Imaging – RTM by Total Inc.
  - Pierre-Yves Aquilanti, TOTAL E&P in context of a CRADA
  - TBs as inputs, outputs PBs of results along with intermediate data
  - Company conducted comparison tests among several I/O solutions. ADIOS is their choice for other codes: FWI, Kirchoff

http://www.sciencedaily.com/releases/2016/01/160126130823.htm





klasky@ornl.gov

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

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.

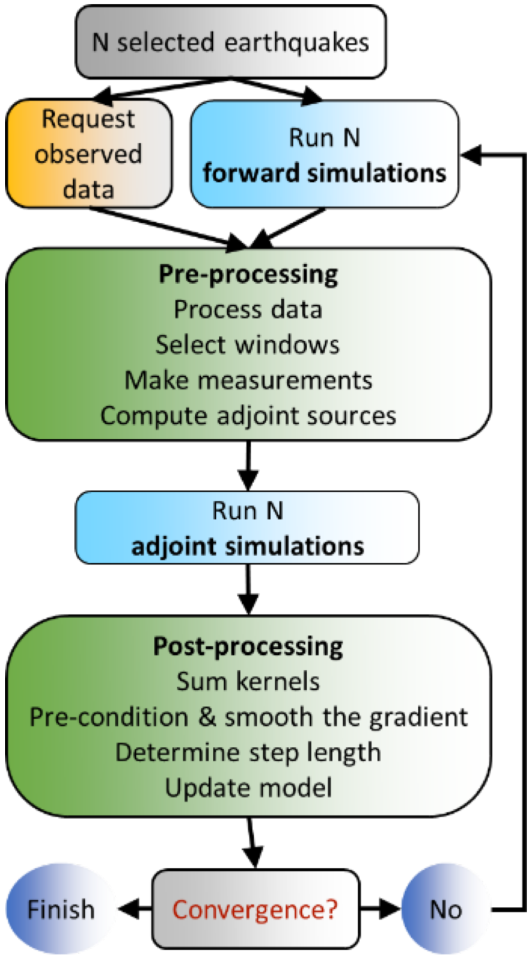
Significance and Impact

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

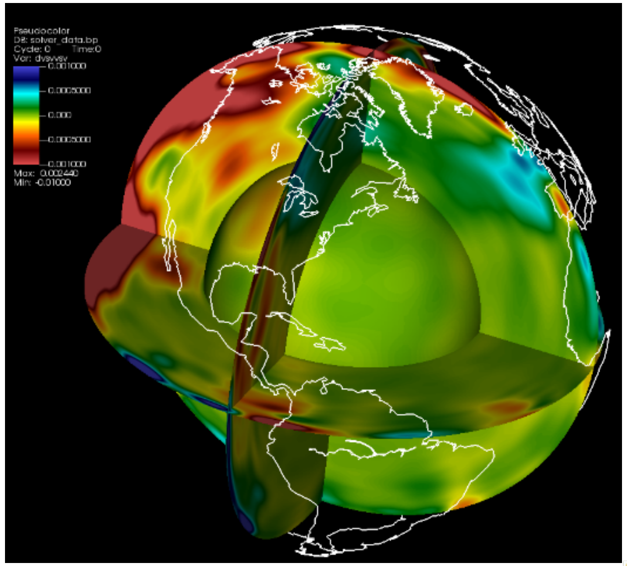
Research Details

- 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

A



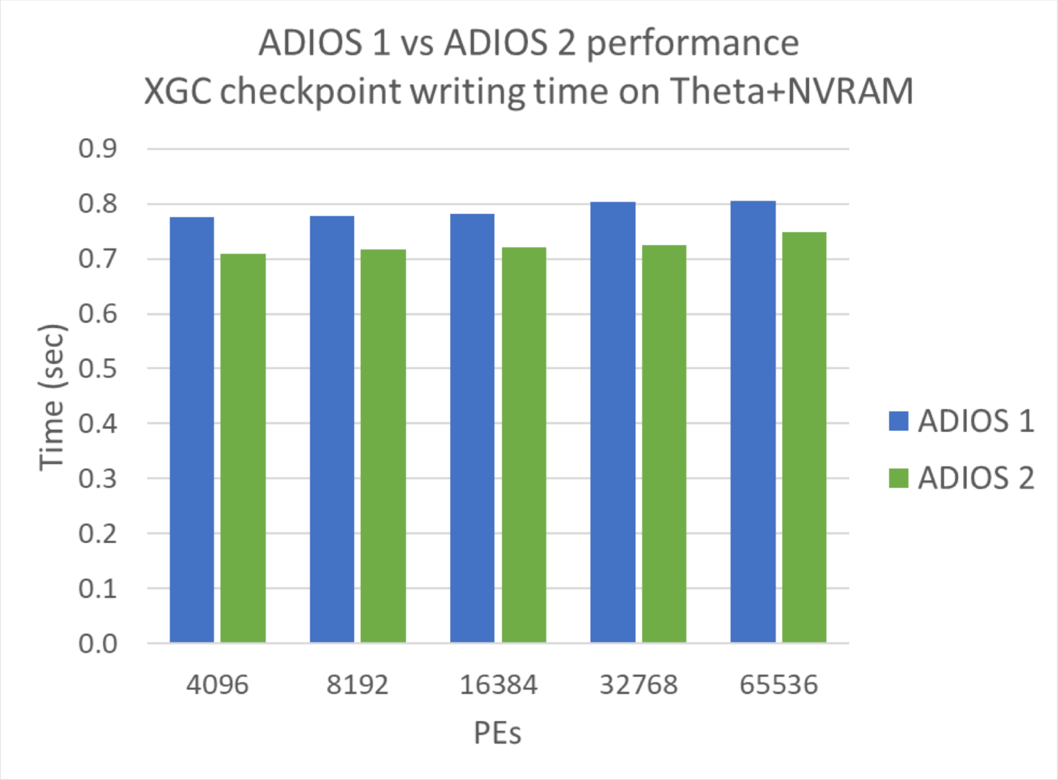
B



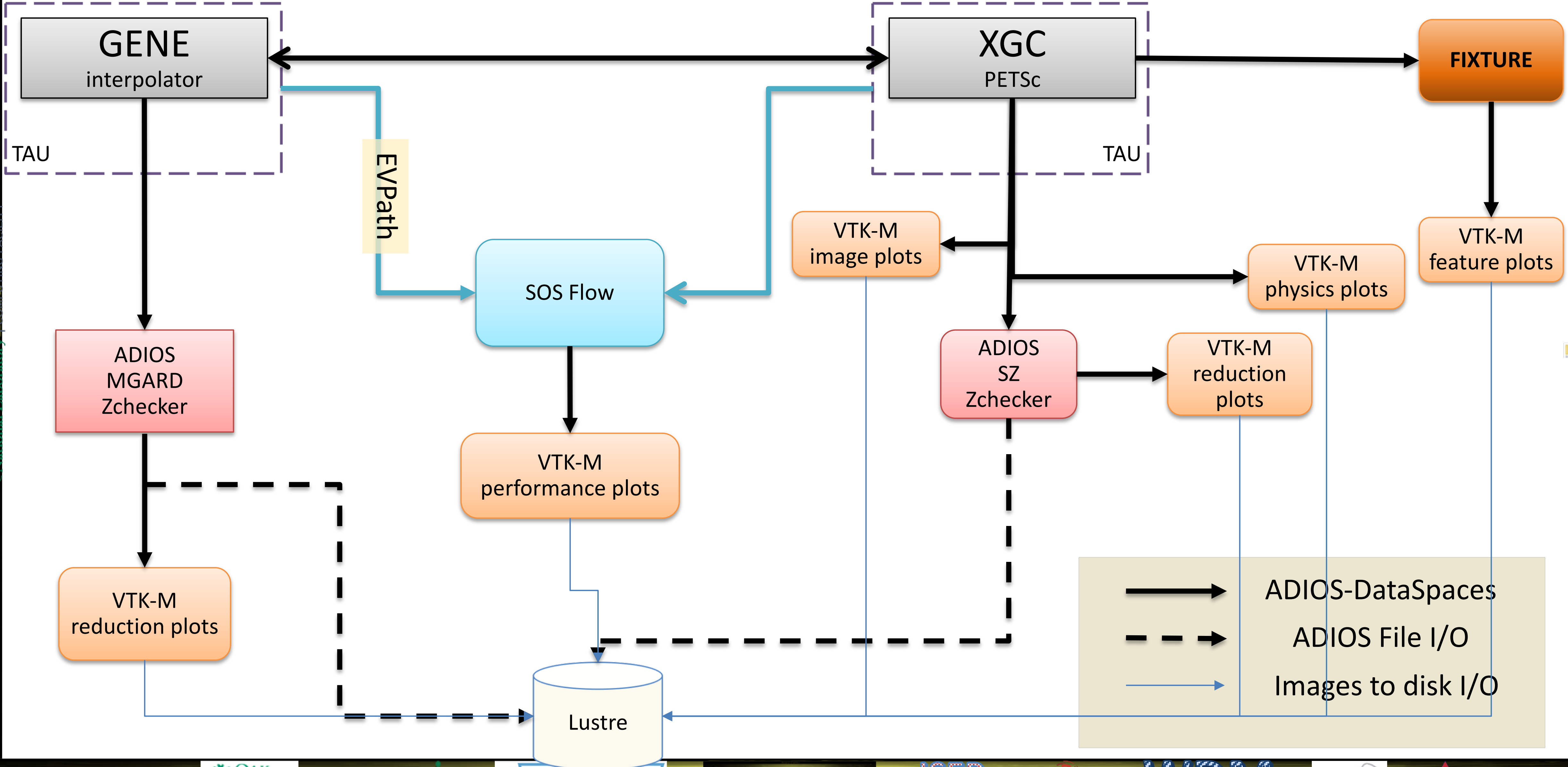
E. Bozdag; D. Peter; M. Lefebvre; D. Komatitsch; J. Tromp; J. Hill; N. Podhorszki; D. Pugmire. **Global adjoint tomography: first-generation model.** *Geophysical Journal International* 2016 207 (3): 1739-1766 https://doi.org/10.1093/gji/ggw356

ADIOS 2.2 testing on Theta

Average restart data size in GB					
nodes	64	128	256	512	1024
data size	152	304	609	1219	2438
Throughput GB/s					
PEs	4096	8192	16384	32768	65536
ADIOS 1	196.4	392.1	779.2	1519.5	3029.4
ADIOS 2	214.8	424.9	846.6	1680.3	3255.2



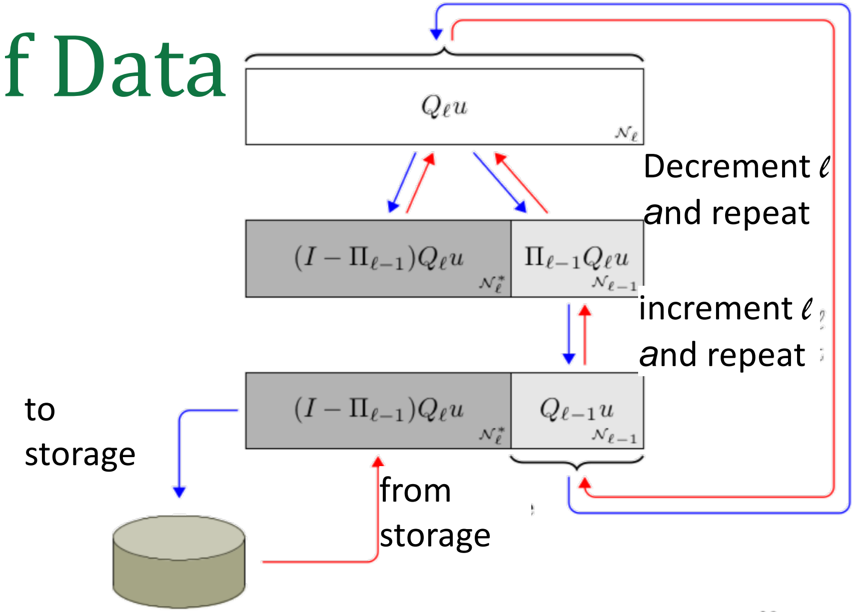
Whole Device Modeling workflow tasks and data flow



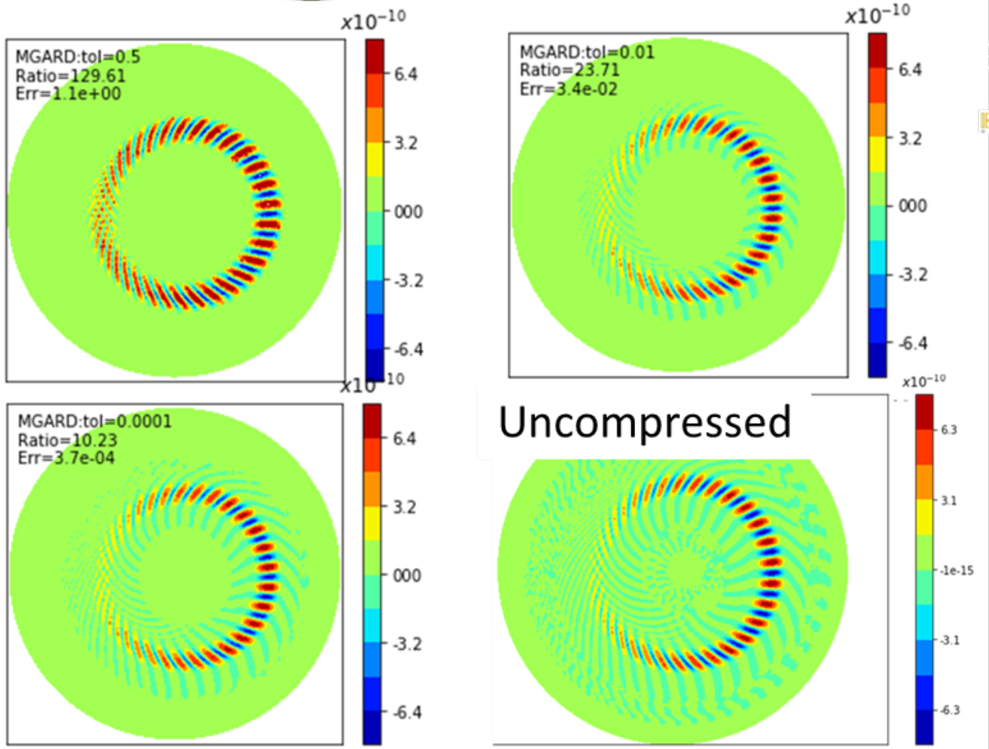
Give them faster I/O and they write more

- ADIOS has accelerated the I/O of many communities by 10-100X over competing State of the Art Solutions
- Most of our users typically have a "time budget" for I/O
  - Increasing the I/O throughput → more data being output → moving from TBs to PB
  - For Experiments and Observations this means that we can push more data
- This created a new problem for the community
  - Too much data and nowhere to store this for the long term
- Data Refactoring (Reduction + Re-ordering)
  - "Bucket the data" into different levels of importance
  - i.e. Save the information in one bucket and the rest of the data in another
- **Goal is to reduce data sizes by 1,000X with minimal loss of information for later post processing**

MGARD: MultiGrid Adaptive Reduction of Data



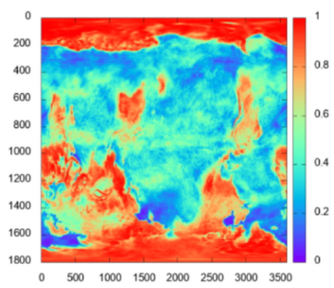
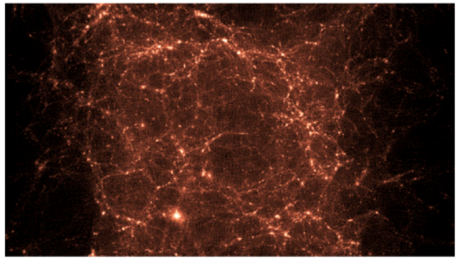
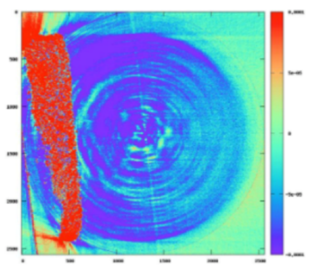
- Decomposes data into contributions from a hierarchy of meshes
- The hierarchical schema offers the flexibility to produce multiple levels of partial decompression of the data so users can work with reduced representations that require minimal storage while achieving the user specified tolerance
- Lossy data reduction based on discarding least important contributions
- Mathematically proven error bounds
- Applicable to structured (tensor product) grids with arbitrary spacing, integrated into ADIOS
- Aims to preserve structures present in input data



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.

ANL SZ Lossy compressor

- **Production quality** lossy compressor for scientific data respecting user set error bounds
- **For 1D, 2D, 3D structured and unstructured datasets.**
- **Strict error controls** (absolute error, relative error, PSNR, error distribution)
- **Thorough testing procedures**
- **Integrated in the ADIOS, HDF5, …**
- **Optimized compression ratios** (transforms, decompositions, multiple predictors, compression in time, lossless compression, etc.)
- **High compression/decompression speed** (MPI + OpenMP)

Mesh data	Particle data	Instrument data
		

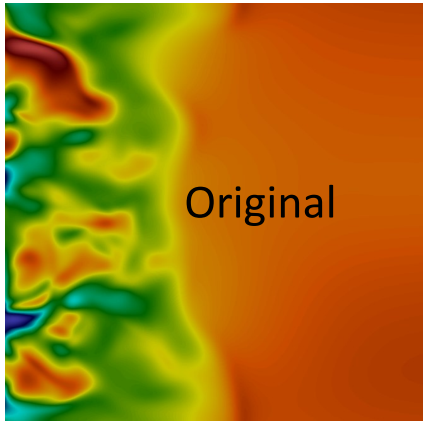
Compression performance

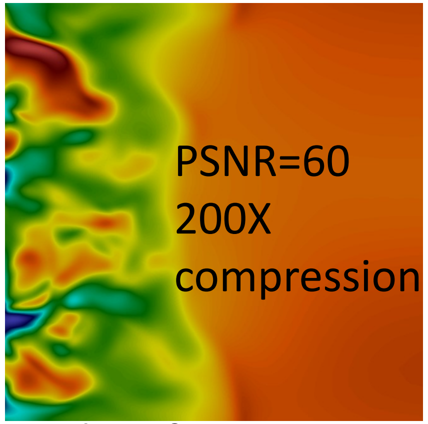
	Mesh data	Particle data	Instrument data
Compression rate	111 MB/s	62 MB/s	110 MB/s
Decompression rate	202 MB/s	182 MB/s	152 MB/s
Compression factor	**14.5**	**10.4**	**13.5**
Point Wise Error Bound	10^{-3}	10^{-3}	10^{-2}
GZIP compression factor	1.3	1.2	1.1

S. Di, F. Cappello, Fast Error-bounded Lossy HPC Data Compression with SZ, **IPDPS 2016**

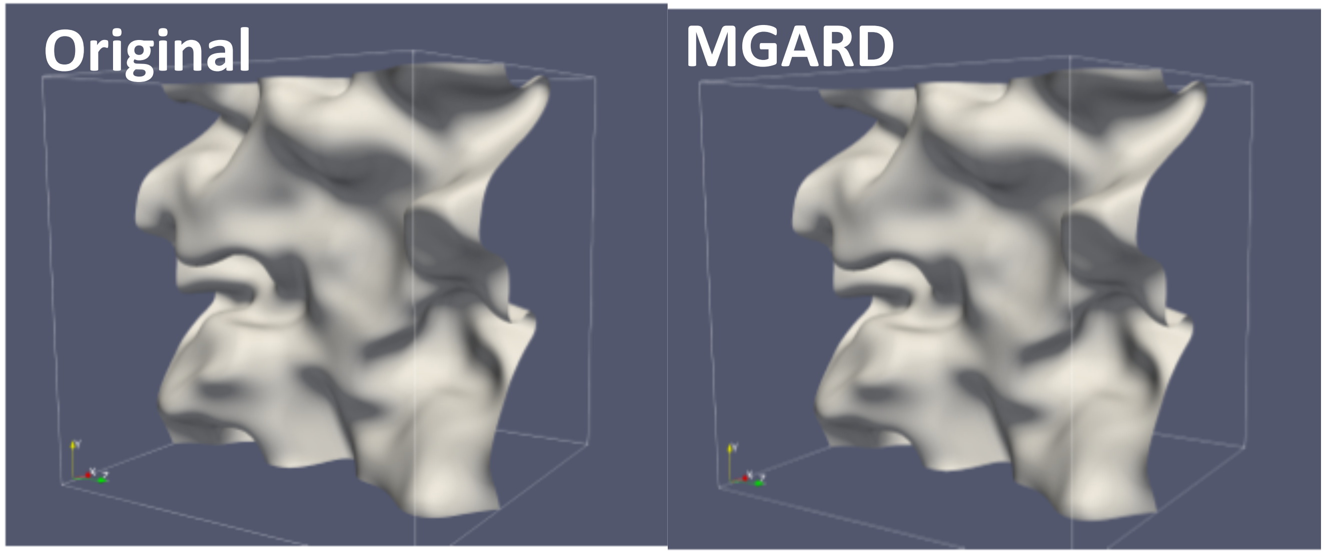
D. Tao, S. Di, Z. Chen, F. Cappello, Significantly Improving Lossy Compression for Scientific Datasets Based on Multidimensional Prediction and Error-Controlled Quantization, **IEEE IPDPS 2017**

Visualizations of data after lossy compression: often misleading



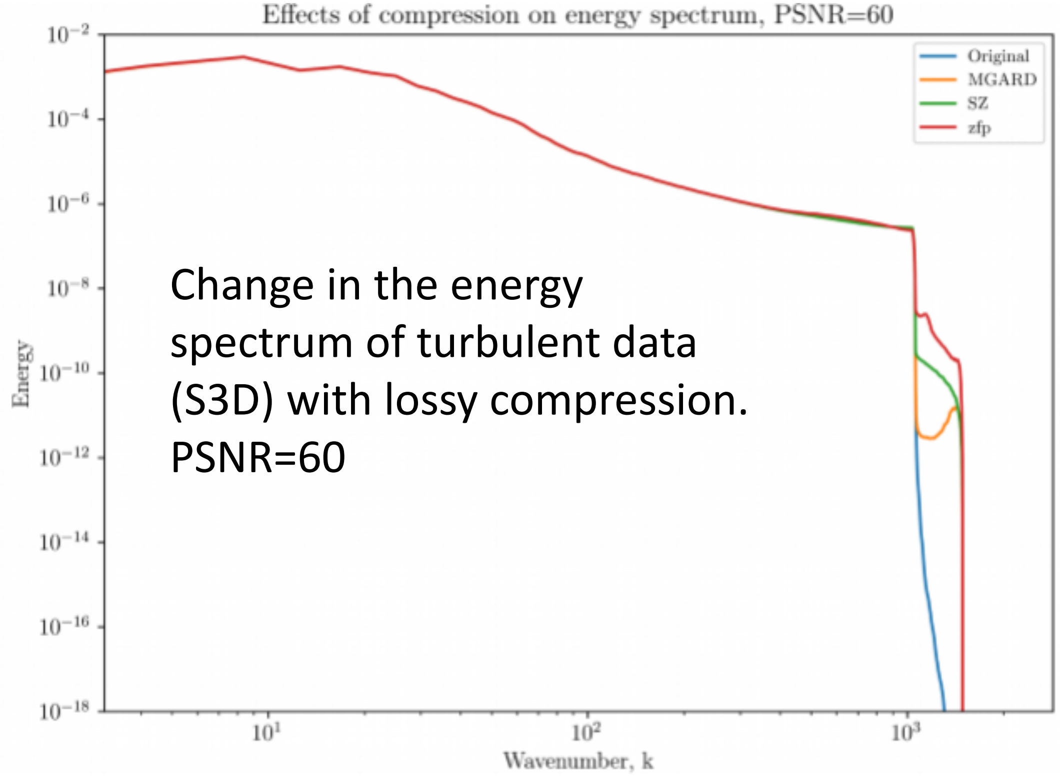


Contours of axial velocity, data from S3D



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

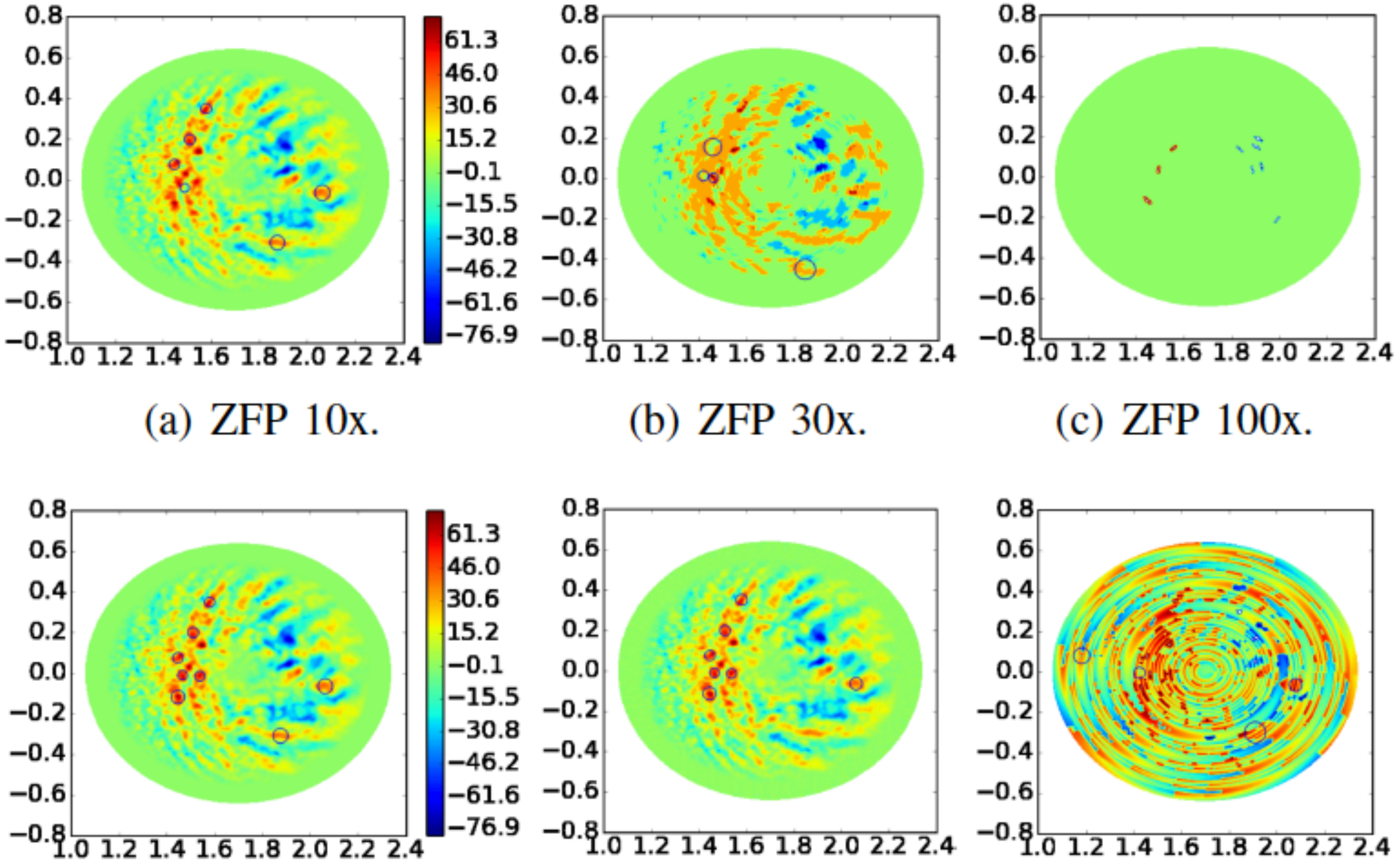
PSNR=60, 200X compression



it is hard to visually discern that energy is added to small flow structures by lossy compression

Understanding and Modeling Lossy Compression

- What is the impact of lossy compression on data fidelity and complex scientific data analytics?



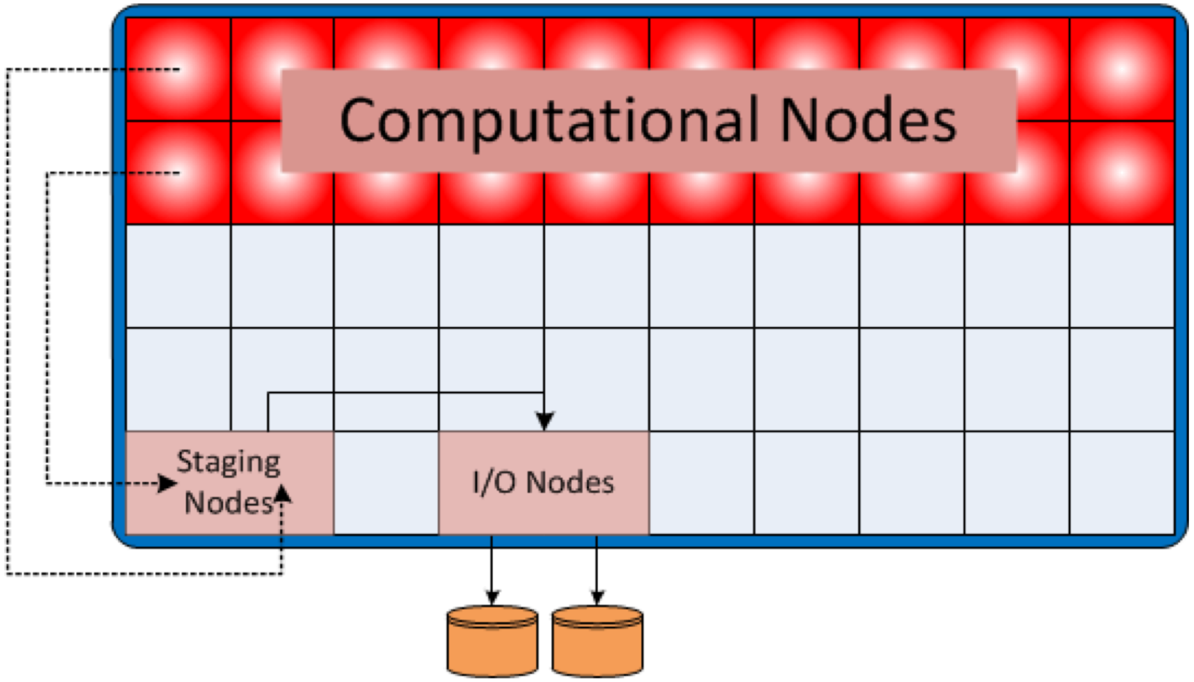
(a) ZFP 10x. (b) ZFP 30x. (c) ZFP 100x.

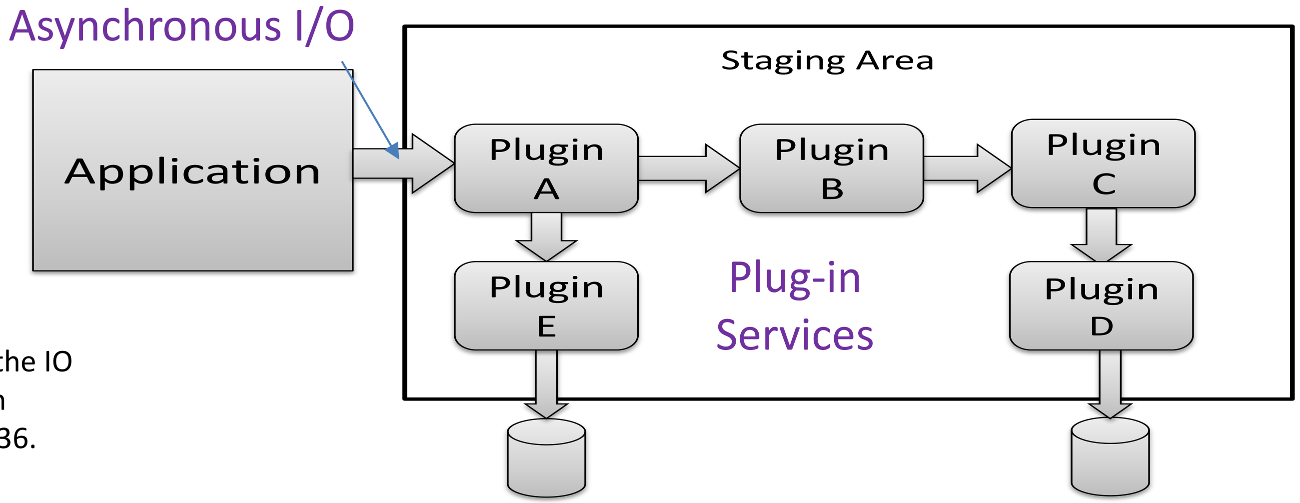
(d) SZ 10x. (e) SZ 30x. (f) SZ 100x.

Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Norbert Podhorszki, Scott Klasky, Matthew Wolf, Tong Liu, Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data, **IPDPS 18**, best paper nominee.

Using Self Describing Data for Staging

- Goal: enhance data services and communication among applications providing an intermediate common area (staging) that reduces file system access costs.
- Self-describing data is crucial for making decisions on-the-fly at every "stage".
- Imaging if this is done using only raw data?
- Components:
  - Asynchronous I/O buffers from Applications
  - Services provided as plugins:
    - Analytics & Visualization
    - Data Reduction
    - Data Transport (RDMA code coupling, files, WAN)

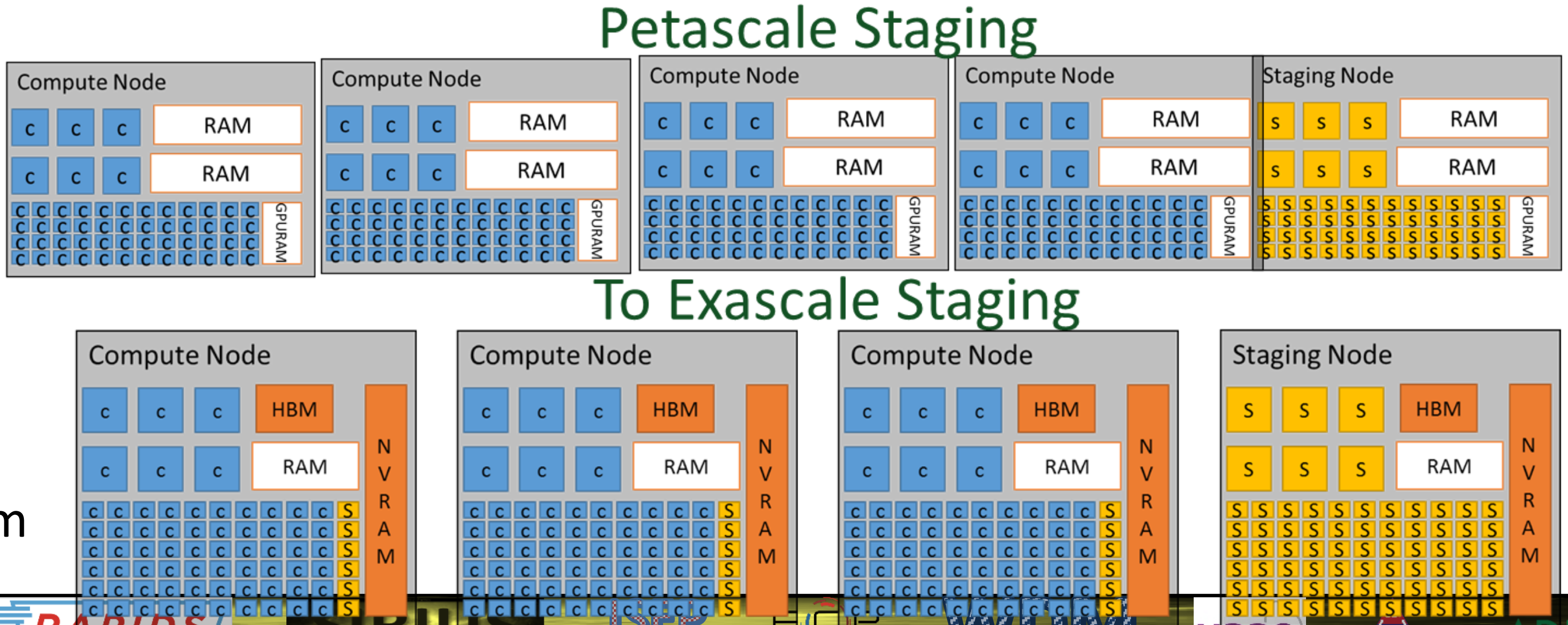




H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, S. Klasky, Just in time: adding value to the IO pipelines of high performance applications with JITStaging in Proceedings of the 20th international symposium on High performance distributed computing, ACM, pp. 27–36.

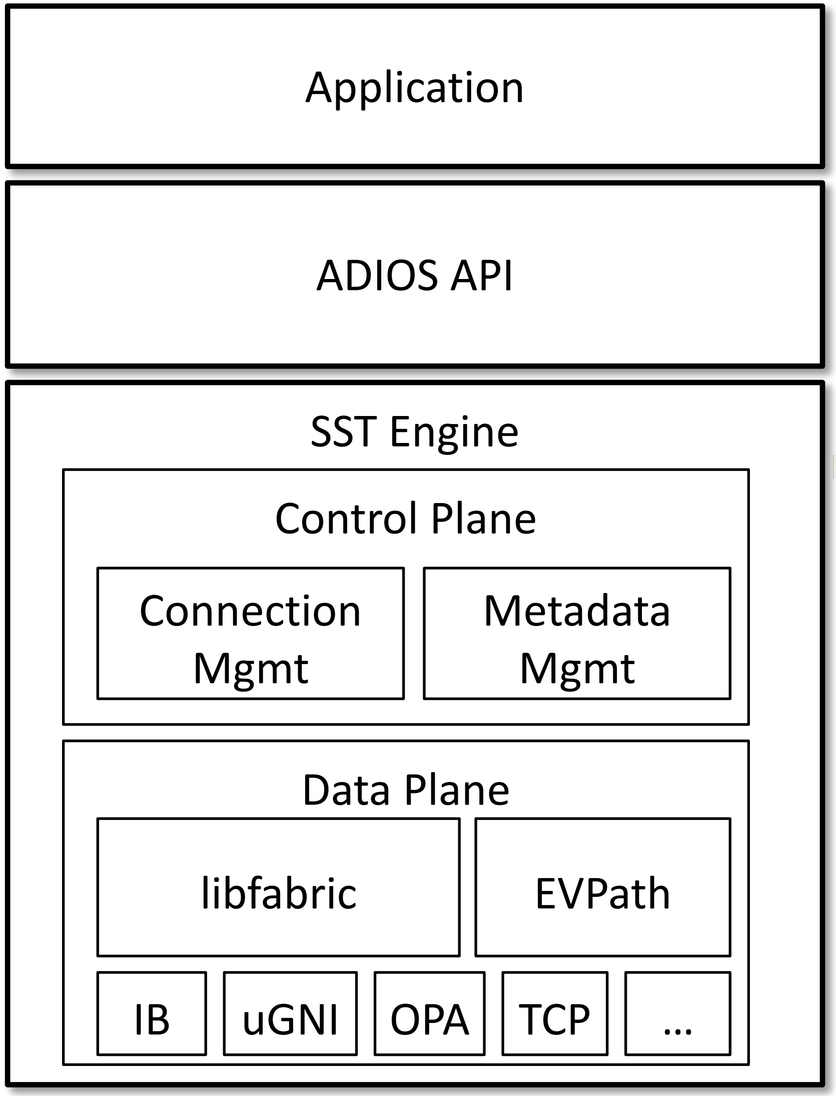
Staging

- Use compute and deep-memory hierarchies to optimize overall workflow for power vs. performance tradeoffs
- Abstract complex/deep memory hierarchy access
- Placement of analysis and visualization tasks in a complex system
- Impact of network data movement compared to memory movement
- Abstraction allows staging
  - On-same core
  - On different cores
  - On different nodes
  - On different machines
  - Through the storage system



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, 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



Good Software Engineering is critical to our success (CI)

Secure | https://open.cdash.org/index.php?project=ADIOS&date=2018-04-23

Login All Dashboards — Monday, April 23 2018 20:15:11

ADIOS

Dashboard | Calendar | Previous | Current | Project

2 hours ago: 4 warnings introduced on _1497_clang-analyzer
2 hours ago: 11 errors introduced on _1497_clang-analyzer
2 hours ago: 4 warnings introduced on master_925_vs2017
2 hours ago: 4 warnings introduced on master_925_vs2015
2 hours ago: 4 warnings introduced on master_2178_el7-gcc7-openmpi

See full feed

Nightly (5 builds)

Site	Build Name	Update	Configure		Build		Test				
		Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass	Start Time	Labels
aaargh.kitware.com	Linux-EL7_Intel18	d75f53	0	0	0	4	6	4^{+3}	71_{-3}	11 hours ago	(none)
aaargh.kitware.com	Linux-EL7_Intel17_IntelMPI	d75f53	0	0	0	4	7	2	114	11 hours ago	(none)
aaargh.kitware.com	Linux-EL7_Intel17	d75f53	0	0	0	4	6	1	74	11 hours ago	(none)
aaargh.kitware.com	Linux-EL7_GCC7	d75f53	0	0	0	4	6	1_{-1}	74^{+1}	11 hours ago	(none)
aaargh.kitware.com	Linux-EL7_GCC7_MPICH	d75f53	0	0	0	4	7	0	116	11 hours ago	(none)

Continuous Integration (16 builds)

Site	Build Name	Update	Configure		Build		Test				
		Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass	Start Time	Labels
AppVeyor	master_925_vs2015	b379ac	0	0	0	4	0	0	66	2 hours ago	(none)
CircleCI	master_2179_el7-gcc48	b379ac	0	0	0	4	0	0	92	2 hours ago	(none)
CircleCI	master_2178_el7-gcc7-openmpi	b379ac	0	0	0	4	0	0	120	2 hours ago	(none)
AppVeyor	close_bug_924_vs2015	7599aa	0	0	0	4	0	0	66	3 hours ago	(none)
CircleCI	close_bug_2177_el7-gcc7-openmpi	7599aa	0	0	0	4	0	0	120	3 hours ago	(none)
CircleCI	close_bug_2176_el7-gcc48	7599aa	0	0	0	4	0	0	92	3 hours ago	(none)
AppVeyor	close_bug_923_vs2015	316ca1	0	0	0	4	0	0	66	4 hours ago	(none)
CircleCI	close_bug_2174_el7-gcc48	316ca1	0	0	0	4	0	0	92	4 hours ago	(none)
CircleCI	close_bug_2175_el7-gcc7-openmpi	316ca1	0	0	0	4	0	0	120	4 hours ago	(none)
AppVeyor	fix-reorganize_922_vs2015	dc7d89	0	0	0	4	0	0	65	5 hours ago	(none)

First | Previous | 1 | 2 | Next | Last

Items per page 10

Analysis (14 builds)

Site	Build Name	Update	Configure		Build		Test				
		Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass	Start Time	Labels
TravisCI	close_bug_1495_cppcheck	316ca1	0	0	82	0				4 hours ago	(none)
TravisCI	_1497_cppcheck	b379ac	0	0	81	0				2 hours ago	(none)
TravisCI	close_bug_1496_cppcheck	7599aa	0	0	81	0				3 hours ago	(none)
TravisCI	fix-reorganize_1494_cppcheck	dc7d89	0	0	81	0				5 hours ago	(none)
TravisCI	_1497_clang-analyzer	b379ac	0	0	15	0				2 hours ago	(none)
TravisCI	close_bug_1496_clang-analyzer	7599aa	0	0	15	0				3 hours ago	(none)
TravisCI	close_bug_1495_clang-analyzer	316ca1	0	0	15	0				4 hours ago	(none)
TravisCI	fix-reorganize_1494_clang-analyzer	dc7d89	0	0	15	0				4 hours ago	(none)
aaargh.kitware.com	Linux-EL7_GCC7_Coverity	d75f53	0	0	0	4				10 hours ago	(none)
aaargh.kitware.com	Linux-EL7_GCC7_Valgrind	d75f53	0	0	0	0				11 hours ago	(none)

ADIOS API
Application Programming Interface

ADIOS Approach

- 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 subfiles, target filesystem 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

klasky@ornl.gov

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 / 2 / 0
real      /fluid_solution/domain14/sol1/Density              {8, 22, 52} = 0.610376 / 1.61812 / 1.18973
real      /fluid_solution/domain14/sol1/VelocityX            {8, 22, 52} = -135.824 / 135.824 / -6.254
real      /fluid_solution/domain14/sol1/VelocityY            {8, 22, 52} = -277.858 / 309.012 / 50.8434
real      /fluid_solution/domain14/sol1/VelocityZ            {8, 22, 52} = -324.609 / 324.609 / 64.7259
real      /fluid_solution/domain14/sol1/Pressure             {8, 22, 52} = 1 / 153892 / 86658.5 / 45590
real      /fluid_solution/domain14/sol1/Nut                  {8, 22, 52} = -0.00122519 / 1 / 0.0664823
real      /fluid_solution/domain14/sol1/Temperature          {8, 22, 52} = 1 / 362.899 / 243.302 / 121.
string    /fluid_solution/domain17/blockName/blockName       scalar = "rotor_flux_1_Main_Blade_shroudga

integer   /fluid_solution/domain17/sol1/Rind                 {6} = 2 / 2 / 2 / 0
real      /fluid_solution/domain17/sol1/Density              {8, 8, 52} = 0.615973 / 1.62794 / 1.16052
real      /fluid_solution/domain17/sol1/VelocityX            {8, 8, 52} = -135.824 / 66.1731 / -4.56457
...
```

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 basic concepts

Developer vs User of application

- Developer
  - Writes code
  - Defines variables and what is in the I/O
  - Sets default runtime parameters in code
- User
  - Runs the application
  - Specifies runtime parameters in configuration file
    - E.g. switches from file I/O to in situ processing

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

ADIOS C++ API

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/heat2d/cpp/IO_adios2.cpp   Line 25

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
  - a **name** is given to the IO object to identify it in the configuration

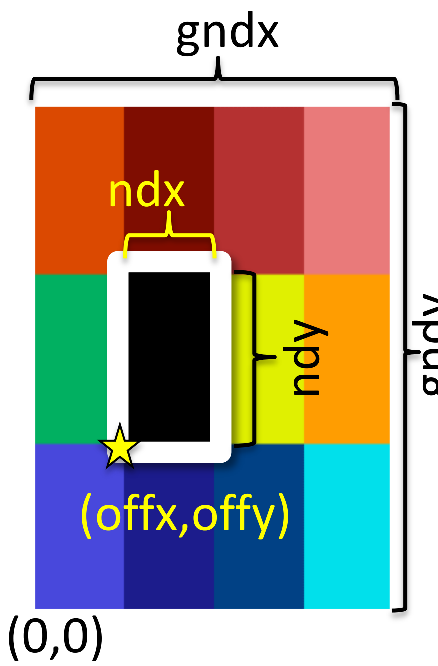
```
adios2::IO io = adios.DeclareIO("SimulationOutput");
if (!io.InConfigFile()) {
    io.SetEngine("BPFile");
}
```

/home/adios/Tutorial/heat2d/cpp/IO_adios2.cpp Line 27

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

/home/adios/Tutorial/heat2d/cpp/IO_adios2.cpp Line 46

Engine object

- To perform the IO

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

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

writer.Close()
```

/home/adios/Tutorial/heat2d/cpp/IO_adios2.cpp lines 55, 67, 73, 74, 60

Reading is similar

```
adios2::IO io = adios.DeclareIO("SimulationOutput");
adios2::Engine reader =
    io.Open("out.bp", adios2::Mode::Read);

  reader.BeginStep()
  {
    adios2::Variable<double> *vT =
      io.InquireVariable<double>("T");
    reader.Get(*vT, T.data());
  }
  reader.EndStep()

reader.Close()
```

Reserve memory for T before this

/home/adios/Tutorial/heat2d/cpp/analysis/heatAnalysis.cpp Lines 89, 98, 113, 128, 163, 166, 189

Put??? Where is Write?

```
writer.BeginStep()
writer.Put(varT, T.data());
writer.EndStep()
```

- Function means: here is the pointer to the data for my variable, which I want to see in the output sometime in the future.
  - Also, pointed data should not be modified until EndStep
- It does NOT mean: when it returns, the data is in the output and is ready for reading
- When it is present in the output depends on runtime settings and the engine
  - Usually EndStep() will flush/send data
  - Temporal aggregation may postpone flushing several steps at once

ADIOS Fortran90 API

Fortran Write API

```fortran
type(adios2_adios)    :: adios
type(adios2_io)       :: io
type(adios2_engine)   :: engine
type(adios2_variable) :: var

call adios2_init(adios, MPI_COMM_WORLD, adios2_debug_mode_on, ierr)
call adios2_declare_io(io, adios, "SimulationOutput", ierr)
call adios2_define_variable(var, io, "T", ndims, shape_dims, start_dims, &
                            count_dims, adios2_constant_dims, T, ierr)
call adios2_open(engine, io, "data.bp", adios2_mode_write, ierr)
call adios2_begin_step(engine, adios2_step_mode_append, 0.0, ierr)
call adios2_put(engine, var, T, ierr)
call adios2_end_step(engine, ierr)
call adios2_close(engine, ierr)
call adios2_finalize(adios, ierr)
```

Usually we define variables only during the first timestep

/home/adios/Tutorial/heat2d/fortran/simulation/io_adios2.F90

Writing with ADIOS I/O

```
call adios2_init_config (adios, "adios2.xml",  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", 2, shape_dims, &
                      start_dims, count_dims, &
                      adios2_constant_dims,  &
                      T, adios2_err )
call adios_close (adios_handle, adios_err)
…
call adios_finalize (rank, adios_err)
```

Fortran Read API

```
call adios2_init(adios, 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_T, 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 **Lines** 47, 48, 55, 59, 86, …

ADIOS Python Low-level API

ADIOS2 Python/Numpy/MPI4Py **Low-Level** API

- Python bindings are enabled for MPI and non-MPI builds of the ADIOS2 library
  - Thin layer to the native, compiled C++ library based on PyBind11 https://pybind11.readthedocs.io/en/stable/
  - Remember: Use “import adios2”
- Dependencies
  - Python 2 or 3 development toolchain (python-dev, python3-dev)
  - NumPy: Data to write and read will be represented by using Numpy array
  - MPI4Py: Need only if the ADIOS2 library was compiled with MPI

Python common

```
from mpi4py import MPI
import numpy
import adios2

adios = adios2.ADIOS("config.xml", comm, adios2.DebugON)

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

Python Write API

```
io = adios.DeclareIO("write")
var = io.DefineVariable("T", [npx*Nx, npy*Ny],
                              [posx*Nx, posy*Ny],
                              [Nx, Ny], adios2.ConstantDims, T)
engine = io.Open("data.bp", adios2.Mode.Write)
engine.BeginStep(adios2.StepMode.Append, 0.0f)
var.SetSelection([[posx*Nx, posy*Ny], [Nx, Ny]])
engine.Put(var, T)
engine.EndStep()
engine.Close()
```

/home/adios/Tutorial/heat2d/python

Python Read API

```
io = adios.DeclareIO("read")
engine = io.Open("data.bp", adios2.Mode.Read)
engine.BeginStep(adios2.StepMode.NextAvailable, 0.0f)
var = io.InquireVariable("T")
var.SetSelection([[2, 2], [4, 4]])
engine.Get(var, T)
engine.EndStep()
engine.Close()
```

ADIOS Python High-level API

ADIOS2 Python **High-Level** Bindings: "Hello Writer" Example

```
shape = [size* nx]
start = [rank* nx]
count = [nx]

# Open writer
fw = adios2.open("types_np.bp", "w", comm)

# Global values
if(rank == 0):
    fw.write("tag", "Testing ADIOS2 high-level API")
    fw.write("gvarR64", np.array(data.R64[0]))

# Arrays
for i in range(0, 3):
    fw.write("steps", "Step:" + str(i))
    fw.write("varR32", data.R32, shape, start, count)
    fw.write("varR64", data.R64, shape, start, count, True)

fw.close()
```

Open a "BP File" passing the name, mode, communicator….returns a "file" object

Write from strings (values only)

Write from single value numpy

Write from numpy float arrays

Write from numpy double arrays and ADVANCE Step to ALL Variables (endl = True)

Close your "File", data is on disk and fw becomes unreachable

Full example: https://github.com/ornladios/ADIOS2/blob/master/testing/adios2/bindings/python/TestBPWriteTypesHighLevelAPI.py

ADIOS2 Python High-Level Bindings: “Hello Reader” Example

```python
# Reader
fr = adios2.open("types_np.bp", "r", comm)

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")

inTag = fr.readstring("tag")
inR64 = fr.read("gvarR64")


while(not fr.eof()):
    instepStr = fr.readstring("steps")
    indataR32 = fr.read("varR32", start, count)
    indataR64 = fr.read("varR64", start, count, True)
    ….

fr.close()
```

Open a ”BP File” passing the name, read-only mode, communicator….returns a ”file” object

Inspect the file variables:
vars_info is a dict[var_name, metadata[key,value].
e.g.: variable_name: varR32
Type: float
Min: 0
Max: 9
AvailableStepsCount: 3
Shape: 20
SingleValue: false

Read returns single string

Read return single numpy double

eof: detects end of “file” is last step is reached

Read returns a numpy double arrays and ADVANCE Step to ALL Variables

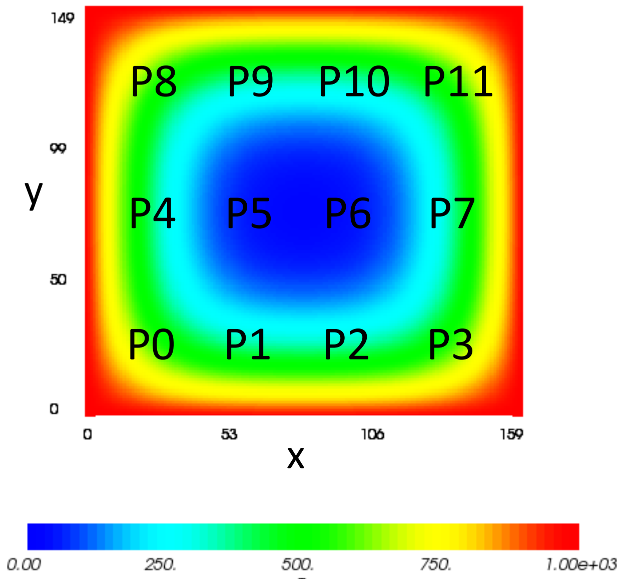
Close your read-only “File” fr becomes unreachable

Full example:
https://github.com/ornladios/ADIOS2/blob/master/testing/adios2/bindings/python/TestBPWriteTypesHighLevelAPI.py

Heat2D Example

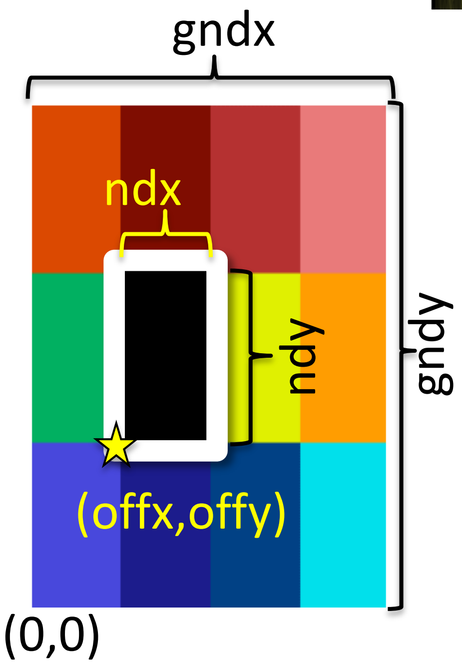
Write Example

- In this example we start with a 2D code which writes data of a 2D array, with a 2D domain decomposition, as shown in the figure.
  - Heat transfer example with heating the edges
  - We write out 5 time-steps, into a single file.
- For simplicity, we work on only 12 cores, arranged in a 4 x 3 arrangement.
- Each processor works on 40x50 subsets (T and dT).
- The total size of the output array = 4*40 x 3*50



Heat2D Global Array

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




```
integer*8                    :: var_T, io
integer*8, dimension(2) :: shape_dims, start_dims, count_dims
call adios2_define_variable (
    var_T, io, &
    "T", &                    !! name in output/input
    2, shape_dims, &          !! Global dimensions (2D here) (gndx, gndy)
    start_dims, &             !! offsets in global space (offx, offy)
    count_dims, &             !! local size (ndx, ndy)
    adios2_constant_dims,  &  !! true/false: dimensions will not change
    T, &                      !! Fortran variable to provide type
    adios2_err )(
```

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

The ADIOS XML configuration file

- Describe runtime parameters for each IO grouping
  - select the Engine for writing
    - **BPFile**, **HDF5**, **SST**, **InSituMPI**, **DataMan**
- see Tutorial/heat2d/fortran/adios2.xml
- XML-free: engine can be selected in the source code as well

The XML file

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

    <io name="SimulationOutput">
        <engine type="BPFile">
        </engine>
    </io>

</adios-config>
```

Writing with ADIOS I/O

```
call adios2_init_config (adios, "adios2.xml",  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", 2, shape_dims, &
                    start_dims, count_dims, &
                    adios2_constant_dims,  &
                    T, adios2_err )
call adios_close (adios_handle, adios_err)
…
call adios_finalize (rank, adios_err)
```

Compile ADIOS codes

- Makefile
  - use adios_config tool to get compile and link options

```
ADIOS_DIR = /opt/adios2/
ADIOS_CLIB = $(shell ${ADIOS_DIR}/bin/adios2-config --libs)
ADIOS_FLIB = ${ADIOS_CLIB}  -ladios2_f
```

  - 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}
```

Compile and run the code

VM

```
$ cd ~/Tutorial/heat2d/fortran/simulation
$ make adios2
$ cd ..
$ mpirun -n 12 simulation/heatSimulation_adios2   heat   4 3   256 256   5  100
Using BPFile engine for output
Process decomposition  : 4 x 3
Array size per process : 256 x 256
Number of output steps : 5
Iterations per step    : 100
Simulation step    0: initialization
Simulation step    1
Simulation step    2
Simulation step    3
Simulation step    4
$ du -hs *.bp*
16K    heat.bp
61M    heat.bp.dir
```

ADIOS Componentization

- ADIOS has many different engines
  - BPFile
    - File I/O with BP format, identical to adios 1.x format
  - HDF5
    - File IO with HDF5, requires building ADIOS with *(Parallel) HDF5*
  - SST (Sustainable Staging Transport)
    - N-to-M staging transfer using RDMA or TCP/IP, requires *libfabric*
  - DataMan
    - N-to-N staging transfer focusing on Wide-Area-Network transfers, requires *ZeroMQ*
  - InSituMPI
    - MPMD-style, MPI 2-way async comm, for in situ processing on separate cores

bpls

```
$ bpls -l heat.bp

double    T     5*{768, 1024} = 0 / 200 / null  / null
double    dT    5*{768, 1024} = -0.185194 / 0.130209 / null / null
```

y x

- bpls is a C program
  - dimensions are reported in C order

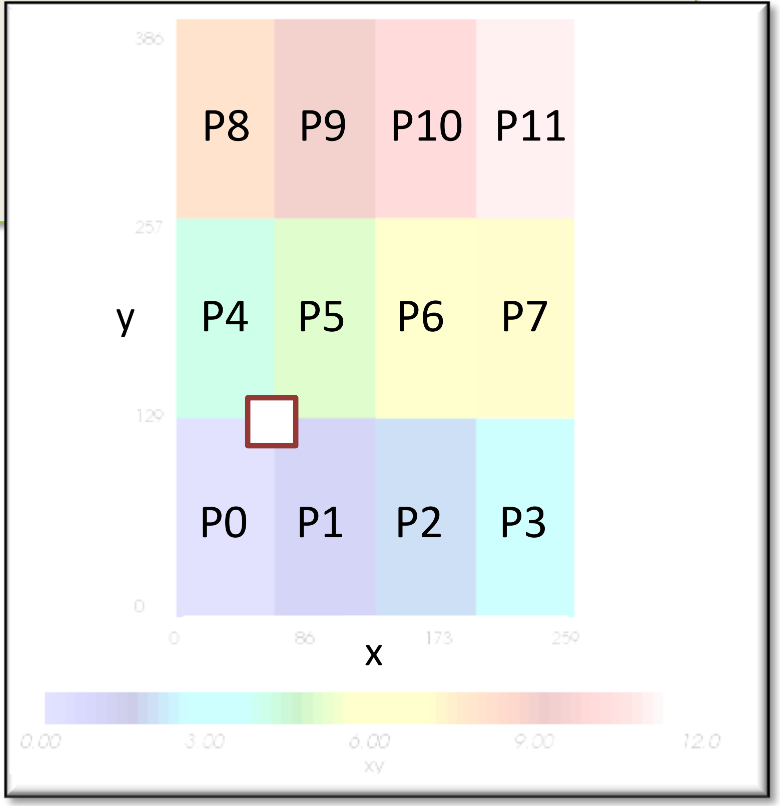
bpls (to show the decomposition of the array)

```
$ bpls -D heat.bp T
double    T      5*{768, 1024}
        step 0:
          block  0: [  0:255,    0: 255]
          block  1: [  0:255,  256: 511]
          block  2: [  0:255,  512: 767]
          block  3: [  0:255,  768:1023]
          block  4: [256:511,    0: 255]
          block  5: [256:511,  256: 511]
          block  6: [256:511,  512: 767]
          block  7: [256:511,  768:1023]
          block  8: [512:767,    0: 255]
          block  9: [512:767,  256: 511]
          block 10: [512:767,  512: 767]
          block 11: [512:767,  768:1023]
step 1:
          ...
```

bpls to dump: 2x2 read with bpls

- Use bpls to read in a 2D slice of the first output step

```
$ bpls heat.bp -d T -s "0,256,256" -c "1,2,2" -n 2
double   T     5*{768, 1024}
    slice (0:0, 256:257, 256:257)
    (0,256,256)    90.2494 90.3374
    (0,257,256)    88.8854 88.9734
```



- 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 heat.bp -d T -n 160 -f "%6.2f" | less –S
 double    T       5*{768, 1024}
    (0,   0,    0)     108.89 108.86 108.79 108.67 108.50 108.29 108.03 107.72 107.
    (0,   0, 160)     104.37 103.50 102.60 101.69 100.76  99.81  98.84  97.86  96.
    (0,   0, 320)      85.11  86.08  87.07  88.08  89.10  90.15  91.22  92.30  93.
    (0,   0, 480)     117.42 118.47 119.51 120.55 121.57 122.58 123.58 124.56 125.
    (0,   0, 640)     133.98 134.25 134.48 134.67 134.80 134.89 134.94 134.93 134.
    (0,   0, 800)      61.72  61.74  61.78  61.84  61.92  62.01  62.13  62.26  62.
    (0,   0, 960)      65.91  66.05  66.24  66.47  66.75  67.08  67.45  67.87  68
.
.
.
```

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

List the content

VM

```
$ h5ls -r heat.h5
/                        Group
/Step0                   Group
/Step0/T                 Dataset {1024, 768}
/Step0/dT                Dataset {1024, 768}
/Step1                   Group
/Step1/T                 Dataset {1024, 768}
/Step1/dT                Dataset {1024, 768}
/Step2                   Group
/Step2/T                 Dataset {1024, 768}
/Step2/dT                Dataset {1024, 768}
/Step3                   Group
/Step3/T                 Dataset {1024, 768}
/Step3/dT                Dataset {1024, 768}
/Step4                   Group
/Step4/T                 Dataset {1024, 768}
/Step4/dT                Dataset {1024, 768}
$ h5ls -d heat.h5/Step0/T
```

Compile and run the code

VM

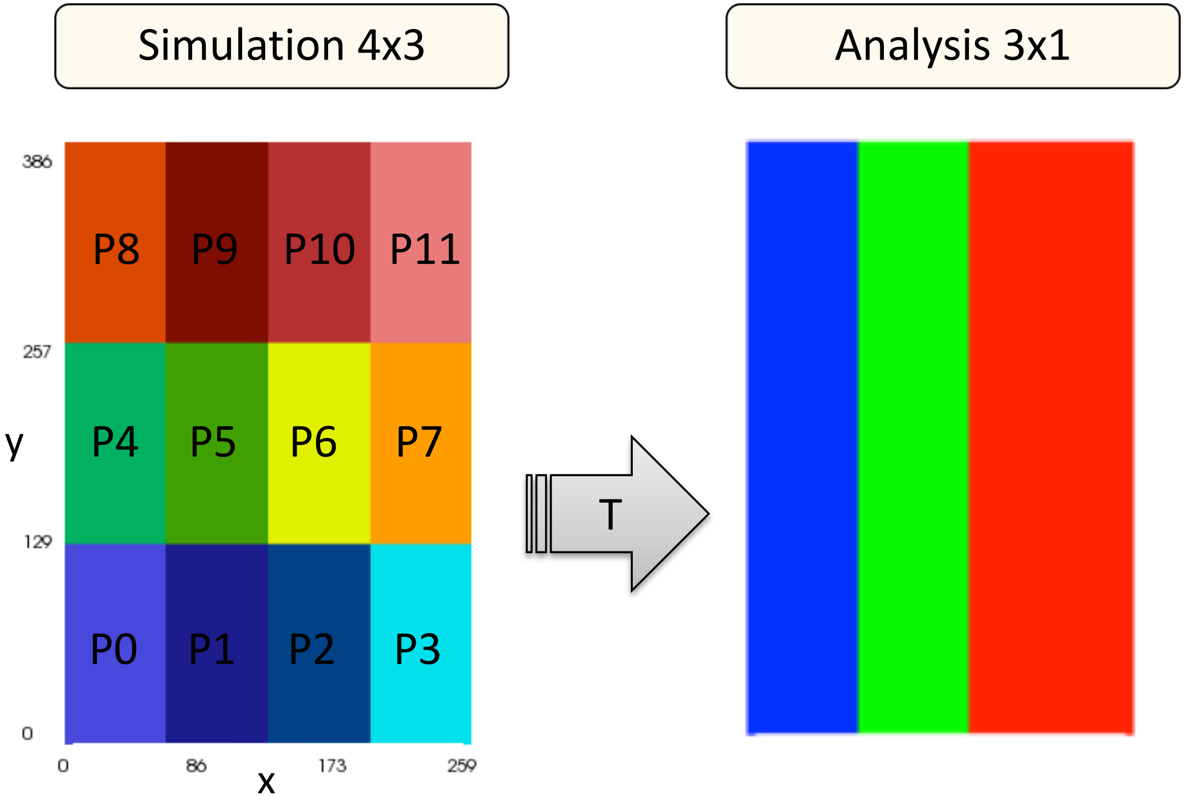
```
$ cd ~/Tutorial/heat2d/fortran/simulation
# make sure in adios2.xml SimulationOutput has the HDF5 engine
$ mpirun -n 12 simulation/heatSimulation_adios2 heat 4 3 256 256 5 100
Using HDF5 engine for output
Process decomposition  : 4 x 3
Array size per process : 256 x 256
Number of output steps : 5
Iterations per step    : 100
Simulation step    0: initialization
Simulation step    1
Simulation step    2
Simulation step    3
$ du -hs *.h5
 61M   heat.h5
```

Fortran Read API

```
integer(kind=8) :: adios, io, var, engine
call adios2_init(adios, MPI_COMM_WORLD, adios2_debug_mode_on, ierr)
call adios2_declare_io(io, adios, "reader", ierr)
call adios2_open(engine, io, "data.bp", adios2_mode_read, ierr)
call adios2_inquire_variable(var, io, "T", 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)
```

Analysis

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



Compile and run the reader

VM

$ cd ~/Tutorial/heat2d/fortran/**analysis**

$ make adios2_stream

$ cd ..

make sure in adios2.xml, SimulationOutput's engine is set to BPFile

$ mpirun -n 3 analysis/heatAnalysis_adios2_stream heat.bp

```
 Input file: heat.bp
 Using BPFile engine for input
 Process step: 0
 Global array size: 1024x768
 Process step: 1
 Process step: 2
 Process step: 3
 Process step: 4
Stream has terminated. Quit reader
```

Decomposes on the slow dimension

$ ls fort.10*

```
 fort.100  fort.101  fort.102
```

$ less -S fort.100

```
rank=0 size=1024x256 offsets=0:0 step=0
 time   row   columns 0...255
```

HDF5 engine to read/write HDF5 files

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

Compile and run the reader

VM

make sure in adios2.xml, SimulationOutput's engine is set to HDF5

$ mpirun -n 3 analysis/heatAnalysis_adios2_stream **heat.h5**

```
 Input file: heat.h5
 Using HDF5 engine for input
 Process step: 0
 Global array size: 1024x768
 Process step: 1
 Process step: 2
 Process step: 3
 Process step: 4
Stream has terminated. Quit reader
```

$ ls fort.10*

```
 fort.100  fort.101  fort.102
```

$ less -S fort.100

```
rank=0 size=1024x768 offsets=0:0 step=0
 time   row   columns 0...255
              0        1        2        3        4        5
--------------------------------------------------------------
```

Fortran Read API

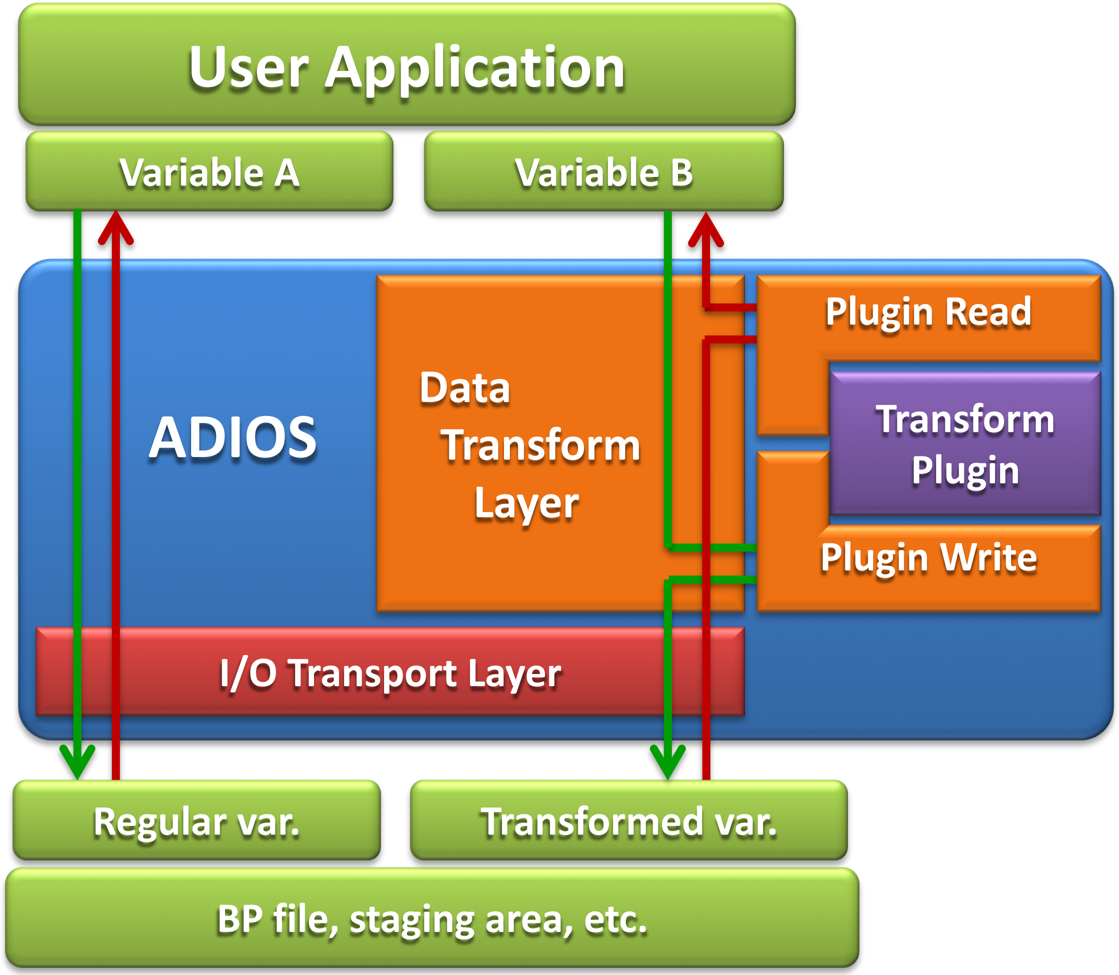
```fortran
integer(kind=8) :: adios, io, var, engine
call adios2_init(adios, MPI_COMM_WORLD, adios2_debug_mode_on, ierr)
call adios2_declare_io(io, adios, "reader", ierr)
call adios2_open(engine, io, "data.bp", adios2_mode_read, ierr)
call adios2_inquire_variable(var, io, "T", ierr )
call adios2_set_selection(var, 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)
```

Transformations

Compression/decompression

ADIOS Transforms

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



Examples…

- Edit the adios1.xml file to try all of the transforms for the following run
- $ mpirun -n 12 simulation/heatSimulation_adios1 heat 4 3 1024 1024 1 1
- NONE – 193 MB
- ZFP – 14 MB
- SZ – 832 KB
- MGARD – 3.1 MB
- blosc – 73 MB

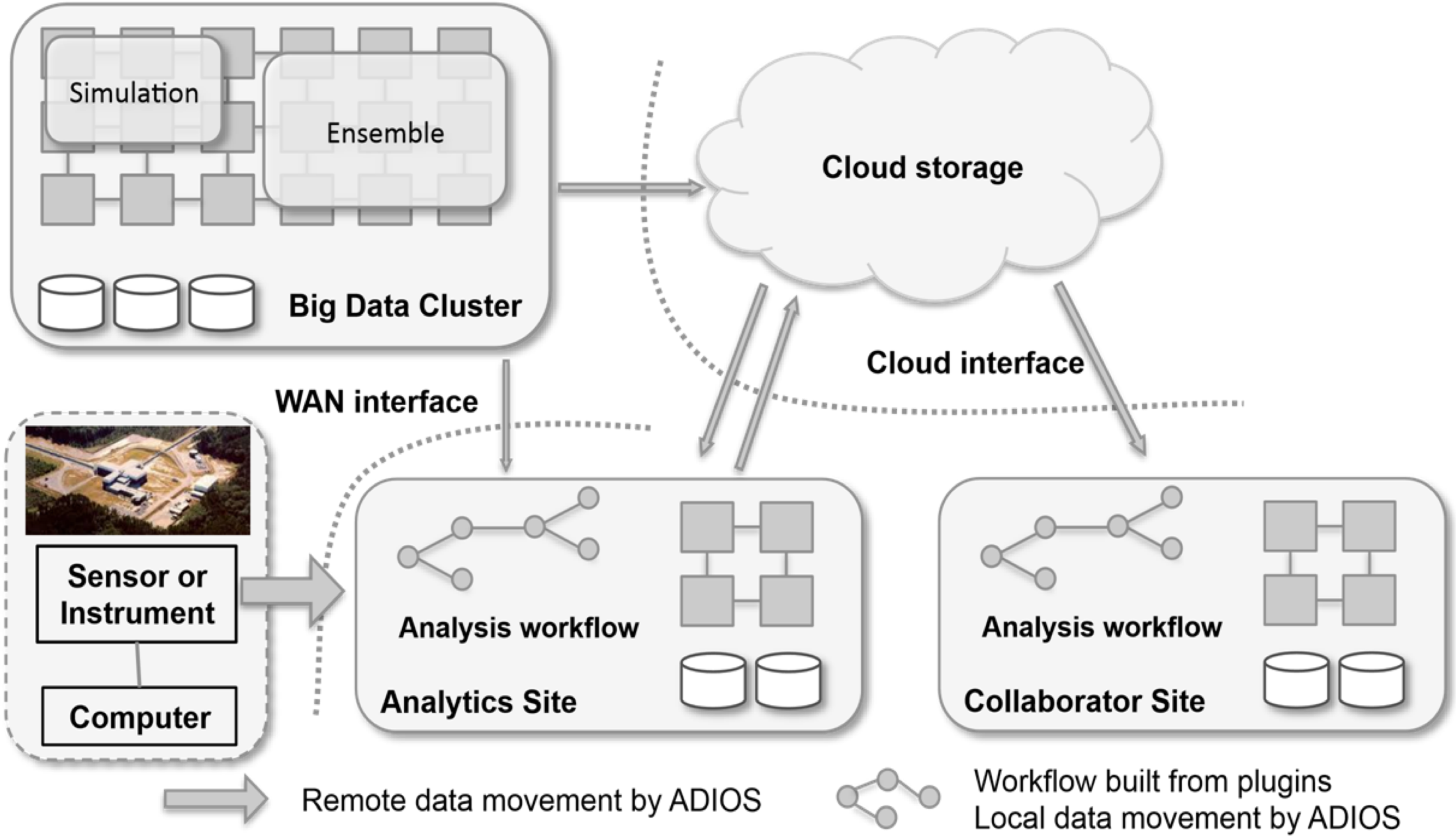
Selecting Transforms (no XML)

- There is a set_transform method, which takes the the variable id and settings string as arguments
-
```
MPI_Comm comm = MPI_COMM_WORLD
int64_t groupid, varid, fh;
char outfile[64] = ”demo-example.bp”
char groupname[64] = “demo”
char name[64] = “test”;
double data[100];

/* Compute data */

adios_init_noxml(comm);
adios_declare_group(&groupid, groupname, “”, adios_flag_yes);
adios_select_method(groupid, “MPI”, “”, “”);
vid = adios_define_var(groupid, vname, “”, adios_double, “100”, “”, “”);
set_transform(vid, “bzip2:5”);
adios_open(fh, groupname, outfile, “w”, comm);
adios_write(fh, vid, data);
adios_close(fh);
```

Staging I/O

Moving data without using the file system

Vision: building scientific collaborative applications



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)

```
call adios2_open(fh, io, streamname, adios2_mode_read, app_comm, ierr)
```

- Close once at the very end of streaming

```
call adios2_close(fh, ierr)
```

Advancing a stream

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

```
call adios2_begin_step(fh, adios2_step_mode_next_available, timeout_sec, ierr)
if (ierr /= adios2_step_status_ok) then
        exit
endif
```

  - advance to the "next available" or directly to the "latest available"
  - timeout_sec: block for this long if no new steps are available
- Release a step if not needed anymore
  - optimization to allow the staging method to deliver new steps if available

```
call adios2_end_step(fh, ierr)
```

Example of Read API: open a stream

```
call adios2_init_config(adios2obj, "adios2.xml", app_comm, .true., ierr)
call adios2_declare_io(io, adios2obj, "SimulationOutput", ierr)
call adios2_open(fh, io, streamname, adios2_mode_read, app_comm, ierr)

if (ierr .ne. 0) then
   print '(" Failed to open stream: ",a)', streamname
   print '(" open stream ierr=: ",i0)', ierr
   call exit(1)
endif
```

Example of Read API: read loop after open

```
ts = 0;
do
   call adios2_begin_step(fh, adios2_step_mode_next_available, -1.0, ierr)
   if (ierr /= adios2_step_status_ok)  exit
   call adios2_inquire_variable(var_T, io, "T", ierr)
   call adios2_variable_shape(var_T, ndim, dims, ierr)  ! dims(1), dims(2) for 2D
   ! Calculate per-process readsize and offset then allocate memory
   allocate( T(readsize(1), readsize(2)) )
   ! Create a 2D selection for the subset
   call adios2_set_selection(var_T, 2, offset, readsize, ierr)
   call adios2_get(fh, "T", T, ierr)
   ! Read all deferred and then release resources to this step
   call adios2_end_step(fh, ierr)
   ts = ts+1
   deallocate(T)
enddo
```

Example of Read API: clean-up

```
...
enddo

if (ierr==adios2_step_status_end_of_stream .and. rank==0) then
    print *, " Stream has terminated. Quit reader"
elseif (ierr==adios2_step_status_not_ready .and. rank==0) then
    print *, " Next step has not arrived for a while. Assume termination"
endif

call adios2_close(fh, ierr)
```

Staging I/O

adios2_reorganize tool

Oak Ridge National Laboratory

heat transfer example with adios2_reorganize

- Staged reading code
  - `/opt/adios2/bin/adios2_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
  - Convert to other formats (e.g. HDF5)

heat transfer example with staging

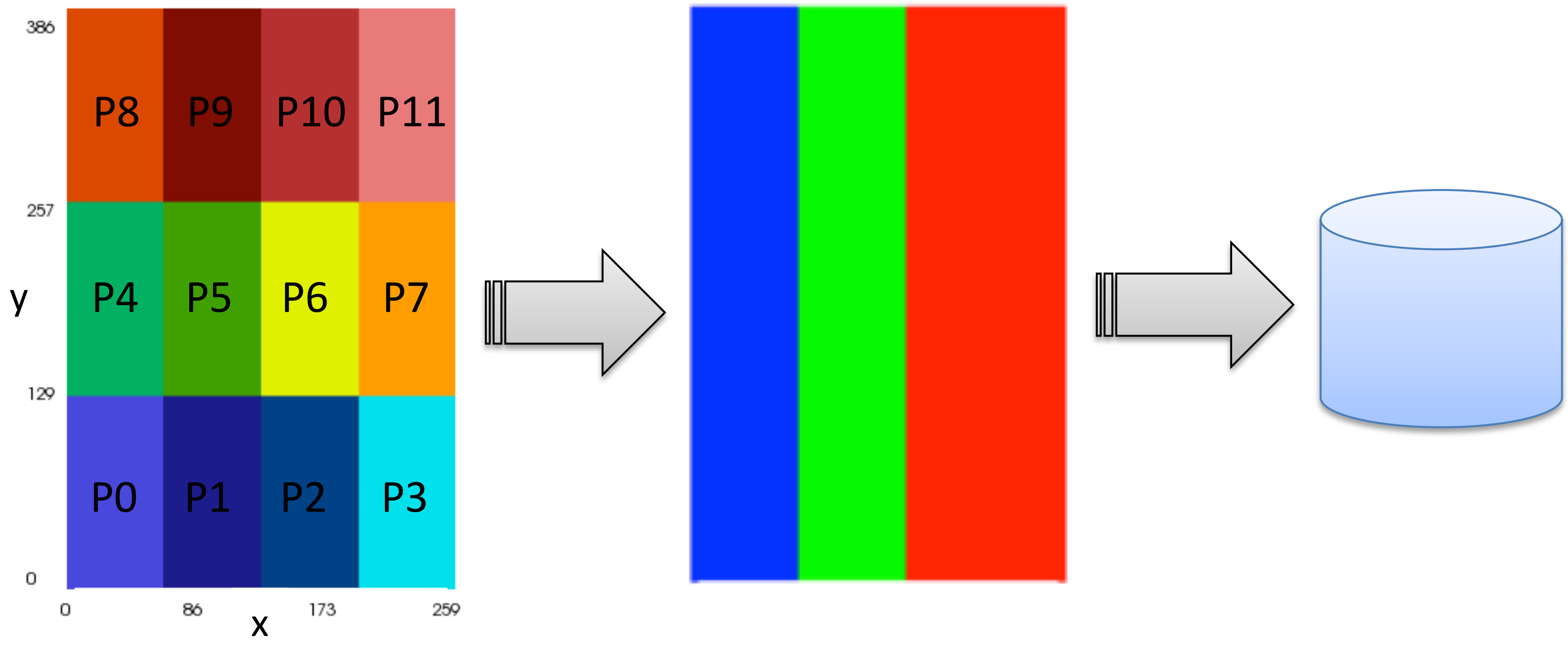
```
$ cd ~/Tutorial/heat2d/fortran
edit adios2.xml  (vi, gedit)
set engine to SST
   <io name="SimulationOutput">
      <engine type="SST">
      </engine>
   </io>
$ mpirun -np 4  simulation/heatSimulation_adios2    heat  2 2   300 300   10 600
```

In another terminal

```
$ cd ~/Tutorial/ heat2d/fortran
$ mpirun -np 2   /opt/adios2/bin/adios2_reorganize   heat.bp   staged.bp   SST  ""   BPFile  ""  2
Input stream               = heat.bp
Output stream              = staged.bp
Read method                = SST
Read method parameters     =
Write method               = BPFile
Write method parameters    =
Waiting to open stream heat.bp...
$ bpls -l staged.bp
```

N to M reorganization with adios2_reorganize

- heatSimulation + adios2_reorganize running together
  - Write out 6 time-steps.
  - Write from 12 cores, arranged in a 4 x 3 arrangement.
  - Read from 3 cores, arranged as 1x3



N to M reorganization with adios2_reorganize

$ cd ~/Tutorial/heat_transfer
edit heat_transfer.xml (vi, gedit)
set engine to BPFile

```
<io name="SimulationOutput">
    <engine type="BPFile">
    </engine>
</io>
```

$ mpirun -np 12 simulation/heatSimulation_adios2 heat 4 3 40 50 6 500
$ bpls -D heat.bp T

```
double   T            6*{150, 160}
       step 0:
         block  0: [  0: 49,    0: 39]
         block  1: [  0: 49,   40: 79]
         ...
         block 11: [100:149, 120:159]
```

$ mpirun -np 3 /opt/adios2/bin/adios2_reorganize **heat.bp** h_3.bp **BPFile** "" **BPFile** "" 3
$ bpls -D h_3.bp T

```
double   T            6*{150, 160}
       step 0:
         block 0: [  0: 49,   0:159]
         block 1: [ 50: 99,   0:159]
         block 2: [100:149,   0:159]
```

heat transfer example with staging

```
$ cd ~/Tutorial/heat2d/fortran
edit adios2.xml  (vi, gedit)
set engine to SST
   <io name="SimulationOutput">
      <engine type="SST">
      </engine>
   </io>
$  mpirun -np 6 simulation/heatSimulation_adios2 heat 2 3 300 300 10 600
```

In another terminal

```
$ cd ~/Tutorial/ heat2d/fortran
$ mpirun -np 4 /opt/adios2/bin/adios2_reorganize heat.bp staged.bp SST "" BPFile "" 4 1
Input stream             = heat.bp
Output stream            = staged.bp
Read method              = SST
Read method parameters   =
Write method             = BPFile
Write method parameters  =
Waiting to open stream heat.bp...
$ bpls  staged.bp –D
$ mpirun -np 4 /opt/adios2/bin/adios2_reorganize heat.bp staged.bp SST "" BPFile ""
$bpls staged.bp -D
```

Conversion from BP to HDF5 files with adios2_reorganize

```
$ cd ~/Tutorial/heat_transfer
edit heat_transfer.xml  (vi, gedit)
set engine to BPFile
  <io name="SimulationOutput">
      <engine type="BPFile">
      </engine>
   </io>
$ mpirun -np 12 simulation/heatSimulation_adios2   heat  4 3    40 50   6 500
$ bpls -D  heat.bp   T
 double    T              6*{150, 160}
        step 0:
          block  0: [  0: 49,    0: 39]
          block  1: [  0: 49,   40: 79]
          ...
          block 11: [100:149, 120:159]
$ mpirun -np 3  /opt/adios2/bin/adios2_reorganize   heat.h5   h_3.bp   BPFile ""   HDF5  ""   3
$ h5ls -r h_3.h5
/                        Group
/Step0                   Group
/Step0/T                 Dataset {150, 160}
/Step0/dT                Dataset {150, 160}
/Step1                   Group
...
```

The runtime config file: adios2.xml

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

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

Engine types
- BPFile
- HDF5
- SST
- **InSituMPI**
- DataMan

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

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

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

</adios-config>
```

Run them together – in a single MPI world

VM

```
$ cd /home/adios/Tutorial/heat2d/cpp
$ make clean-data
# edit adios2.xml and change engine from BPFile to InSituMPI for
# all io groups (SimulationOutput, AnalysisOutput, VizInput)
$ mpirun -n 4 ./heatSimulation    sim.bp  2 2  40 50  2 30   :
         -n  3 ./heatAnalysis      sim.bp  a.bp  3 1         :
         -n  1 ./heatVisualization  a.bp
Simulation step 0: initialization
Analysis step 0 processing simulation step 0
Simulation step 1
Visualization step 0 processing analysis step 0
Analysis step 1 processing simulation step 1
Simulation step 2
Visualization step 1 processing analysis step 1
...
$ ls *.bp* *.pnm
$ gpicview .
```

The runtime config file: adios2.xml

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

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

  <!--===================================
       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>
```

Engine types
BPFile
HDF5
SST
InSituMPI
DataMan

Run them together – separate programs

```
# edit adios2.xml and change engine to SST for
# all io groups (SimulationOutput, AnalysisOutput, VizInput)
```

```
$ mpirun -n 12 ./heatSimulation sim.bp 4 3 40 50 6 100
Simulation step 0: initialization
Simulation step 1
```

```
$ mpirun -n 3 ./heatAnalysis sim.bp a.bp 3 1
Analysis step 0 processing simulation step 0
Analysis step 1 processing simulation step 1
...
```

```
$ mpirun -n 1 ./heatVisualization  a.bp
Visualization step 0 processing analysis step 0
Visualization step 1 processing analysis step 1
...
```

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 engine for all IO groups to **HDF5**
- run the same writing example again
- then also read it back with the same reading example

The runtime config file: adios2.xml

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

  <!--====================================
        Configuration for the Simulation Output
    ====================================-->
  <io name="SimulationOutput">
    <engine type="HDF5">
    </engine>
  </io>

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

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

  <!--====================================
        Configuration for the Visualization Input
    ====================================-->
  <io name="VizInput">
    <engine type="HDF5">
    </engine>
  </io>

</adios-config>
```

Engine types
BPFile
HDF5
SST
InSituMPI
DataMan

Compile and run the Simulation with file I/O

VM

```
$ make clean-data
$ mpirun -n 12 ./heatSimulation   sim.h5  4 3  40 50   6 30
Process decomposition   : 4 x 3
Array size per process : 40 x 50
Number of output steps : 6
Iterations per step     : 30
Using HDF5 engine for output
Simulation step 0: initialization
Simulation step 1
...
Simulation step 5
Total runtime = 0.158174s
$ du -hs *.h5
1.2M  sim.h5
```

List the content

VM

$ h5ls -r sim.h5

```
/                        Group
/Step0                   Group
/Step0/T                 Dataset {160, 150}
/Step1                   Group
/Step1/T                 Dataset {160, 150}
/Step2                   Group
/Step2/T                 Dataset {160, 150}
/Step3                   Group
/Step3/T                 Dataset {160, 150}
/Step4                   Group
/Step4/T                 Dataset {160, 150}
/Step5                   Group
/Step5/T                 Dataset {160, 150}
```

$ h5ls -d sim.h5/Step0/T

Run the Analysis with file I/O

VM

```
$ mpirun -n 3  ./heatAnalysis  sim.h5  a.h5  3  1
Using HDF5 engine for input
Using HDF5 engine for output
gndx         = 160
gndy         = 150
rank 0 reads 2D slice 53 x 150 from offset (0,0)
rank 1 reads 2D slice 53 x 150 from offset (53,0)
rank 2 reads 2D slice 54 x 150 from offset (106,0)
Analysis step 0 processing simulation step 0
Analysis step 1 processing simulation step 1
...
Analysis step 5 processing simulation step 5
$ h5ls -r a.h5
```

Run the Visualization with file I/O

```
$ mpirun -n 1 ./heatVisualization   a.h5
Using HDF5 engine for input
gndx          = 160
gndy          = 150
Visualization step 0 processing analysis step 0
Visualization step 1 processing analysis step 1
...
Visualization step 5 processing analysis step 5
$ ls *.pnm
T.0.pnm  T.1.pnm  T.2.pnm  T.3.pnm  T.4.pnm  T.5.pnm
$ eog . &   ... or ...   $ gpicview &
```