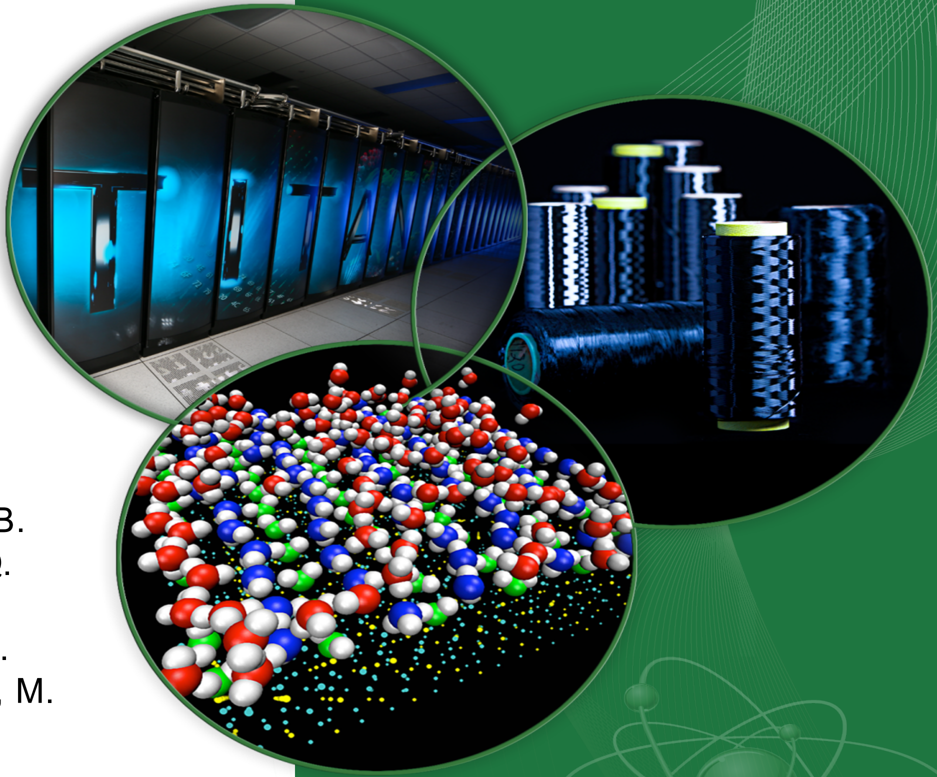
Scaling up your Application I/O using ADIOS

OLCF User Meeting
May 24-26, 2016

Norbert Podhorszki



Thanks to: H. Abbasi, C. S. Chang, S. Ethier, B. Geveci, J. Kim, T. Kurc, S. Klasky, J. Logan, Q. Liu, K. Mu, G. Ostrouchov, M. Parashar, D. Pugmire, J. Saltz, N. Samatova, K. Schwan, A. Shoshani, W. Tang, Y. Tian, M. Taufer, W. Xue, M. Wolf + many more

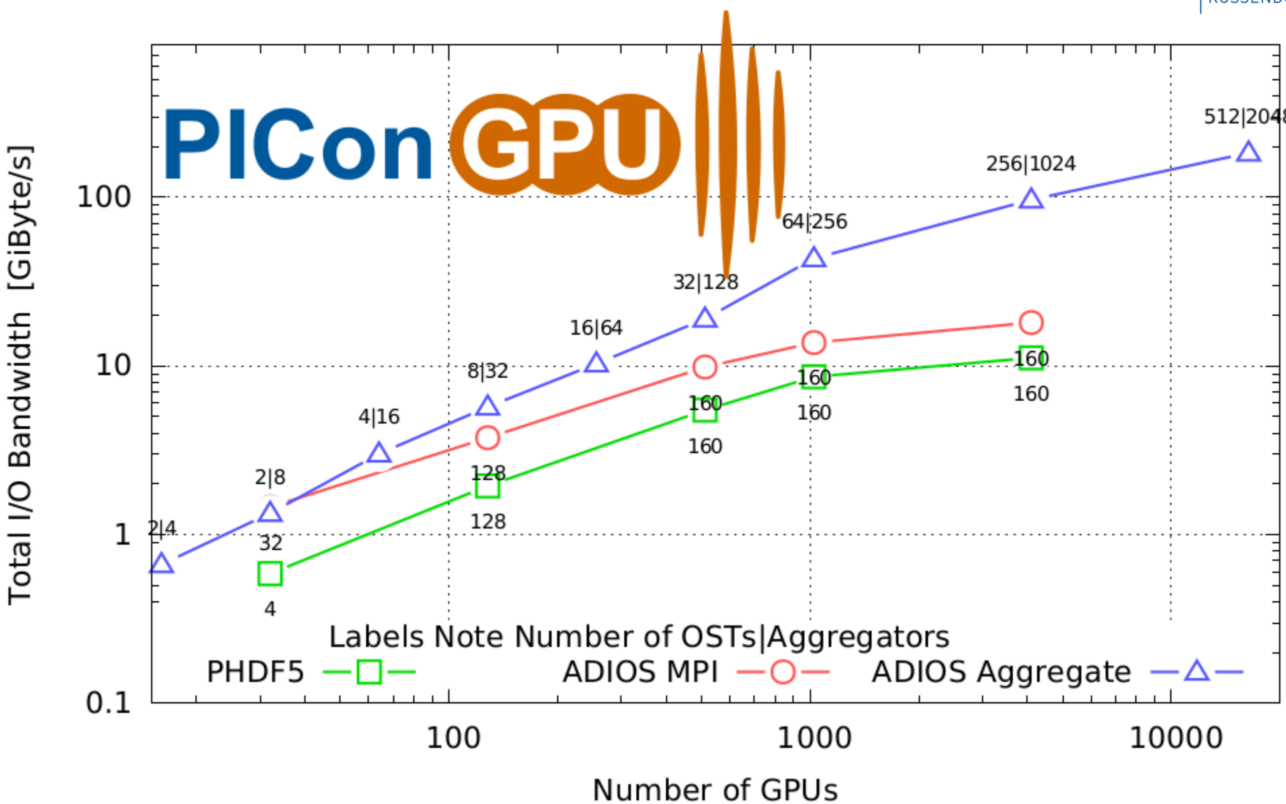
ORNL is managed by UT-Battelle
for the US Department of Energy

SDG

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

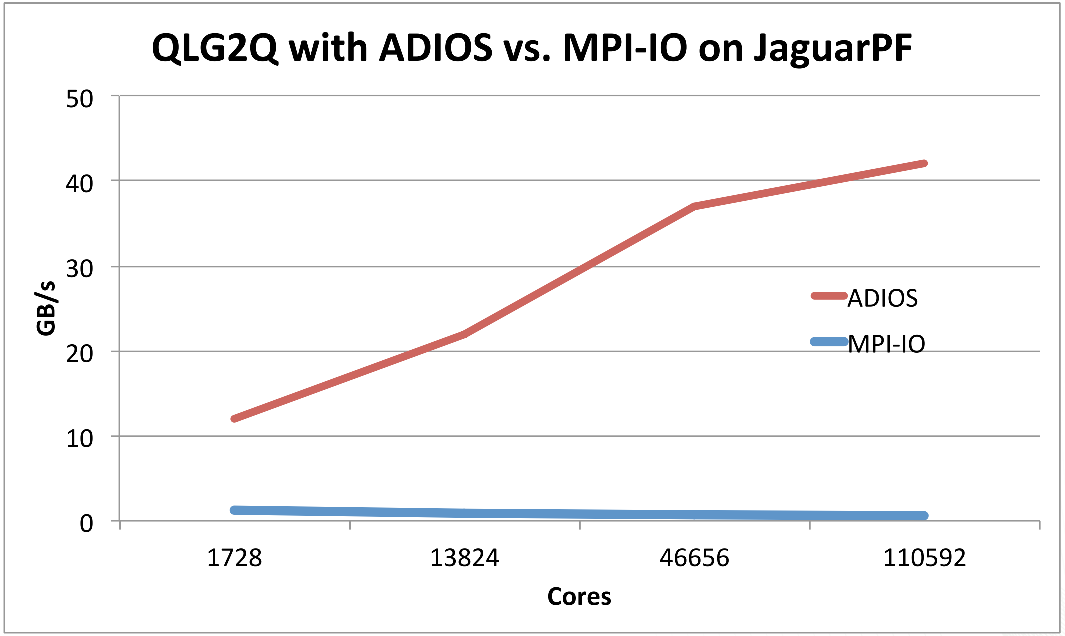
Warm up

- Large data size, ~5GB per node



Quantum Physics – QLG2Q

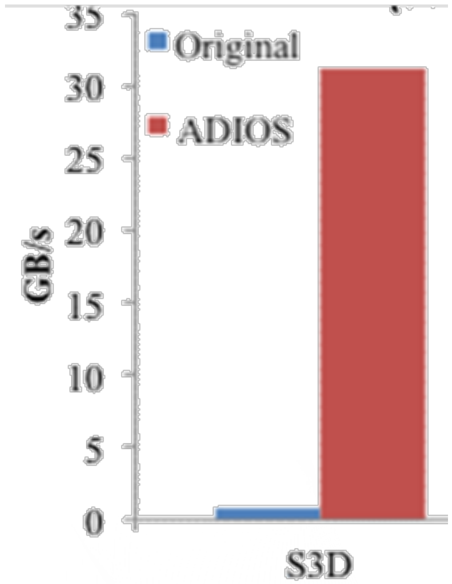
- Large data size + many processors
  - > 50 MB per core, >100K cores



- Later they achieved 98 GB/sec using ADIOS on ERDC, Garnet

Combustion - S3D

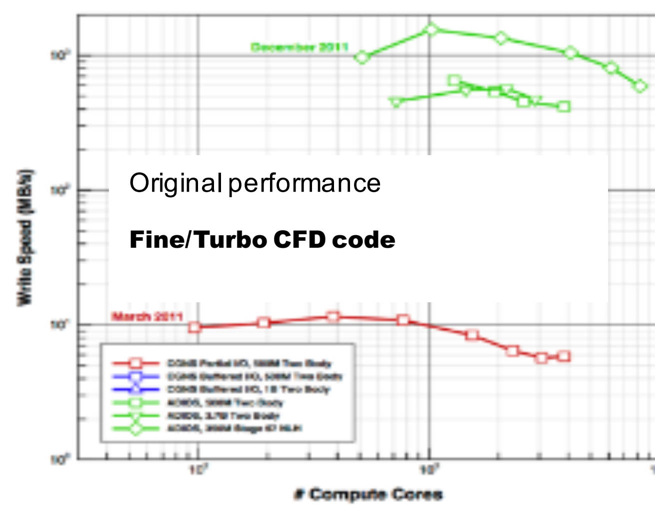
- Small data size + many processors
  - < 1 MB per core, >100K cores
- Individual process output is small, leading to low utilization of network bandwidth with other I/O solutions



Industrial application – RAMGEN & Numeca Fine/Turbo

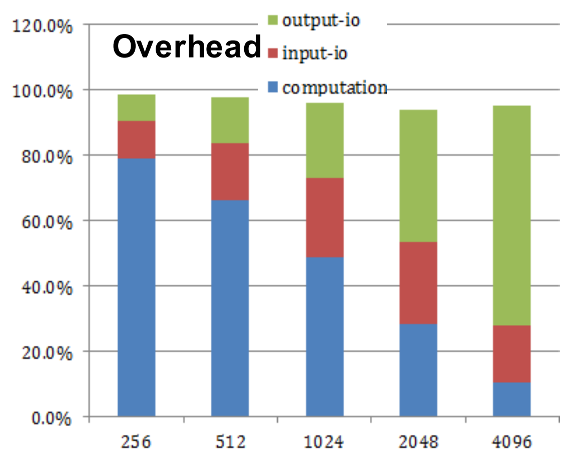
- Studying the time-varying interaction of turbomachinery-related aerodynamic phenomena (shock wave compression technology)
- Each processor handles a heterogeneous distribution of variable data based on a non uniform balancing of the structured model
- 2000 domains, 20 quantities = 40k output variables
- ADIOS allowed for running simulation on 3.7 billion grid cells, which was not possible before

ADIOS performance

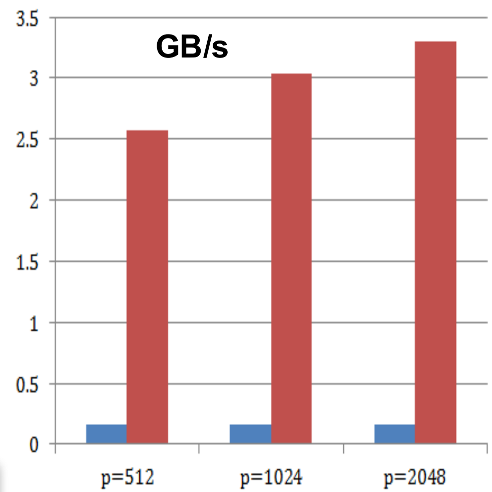


Weather - Global/Regional Assimilation and Prediction System

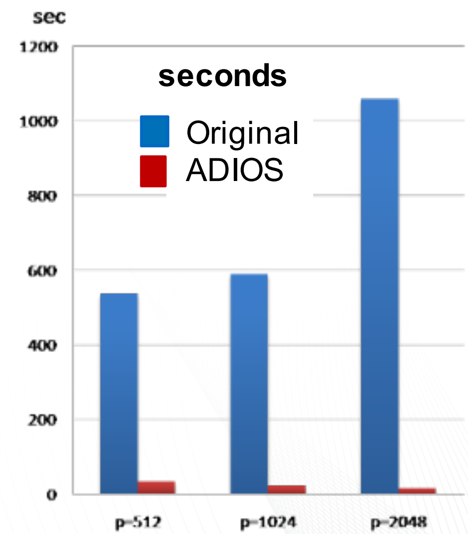
- China is the most natural disaster-prone country in the world
- GRAPES is the new generation NWP of CMA
- GRAPES is the official tool for weather prediction in China
- GRAPES was using MPI-IO, but was I/O dominated above 2K cores



IO dominates the time consumed of GRAPES when > 2048p (25km H-Res GFS Case)



Tianhe-1A, ADIOS AMR .vs. Original IO



Bluelight, GRAPES GFS 15km H-Res Case ADIOS .vs. Original IO

Applications push ADIOS Research Thrusts

- Research ideas moved to production because of the push from full scale science applications
- Many applications presented new challenges for the ADIOS team

ADIOS applications

Accelerator: PIConGPU, Warp

Astronomy: SKA

Astrophysics: Chimera

Combustion: S3D

CFD: FINE/Turbo, OpenFoam

Fusion: XGC, GTC, GTC-P, M3D,M3D-C1, M3D-K, Pixie3D

Geoscience: SPECFEM3D_GLOBE, AWP-ODC, RTM

Materials Science: QMCPack, LAMMPS

Medical Imaging: Cancer pathology

Quantum Turbulence: QLG2Q

Relativity: Maya

Weather: GRAPES

Visualization: Paraview, Visit, VTK, ITK, OpenCV, VTKm

GUARANTEE DOE LCF GUARANTEE

Impact on Industry :

- NUMECA (FINE/Turbo) – Allowed time-varying interaction of turbomachinery-related aerodynamic phenomena
- TOTAL (RTM) – Allowed running of higher fidelity seismic simulations
- FMGLOBAL (FireFoam)– Allowed running higher fidelity fire propagation simulations

Over 1B LCF hours from ADIOS enabled Apps 2015
Over 1,500 citations

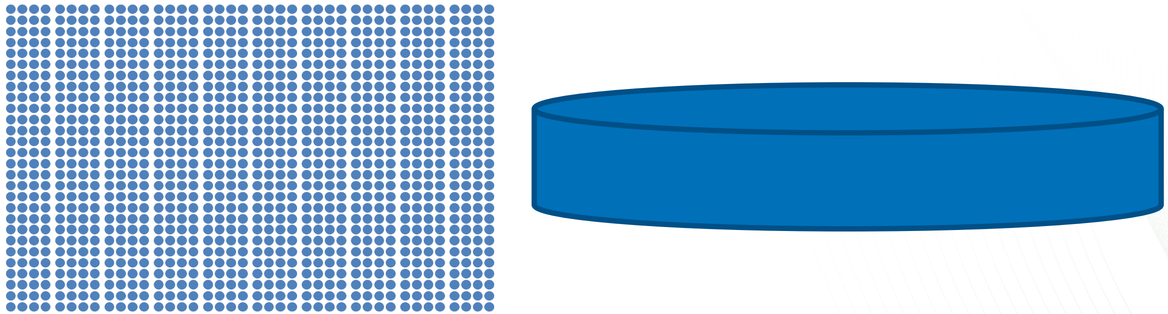
LCF/NERSC Codes in red

Impact at the HPC User facilities

- ALCF
- OLCF
- NERSC
- Tiahne-1A
- Tiahne-2
- Bluelight
- Singapore
- KAIST
- Ostrava
- Dresden
- ERDC
- CSCS
- Blue Waters
- EPFL
- Barcelona Supercomputing Center

Common Problems with I/O

- Many parameters that must be tuned **SIMULTANEOUSLY**
  - Scale out (many nodes) and scale in (many cores)
  - File systems are never 1 disk, RAID set on HPC systems (Lustre striping, anyone?)
  - File creates can be expensive
  - Memory copy is part of I/O
  - Network movement is part of I/O
  - Writing self-describing data introduces metadata movement
- Must examine **all** costs to understand best optimizations
- How do you maintain **performance portability** across different systems?

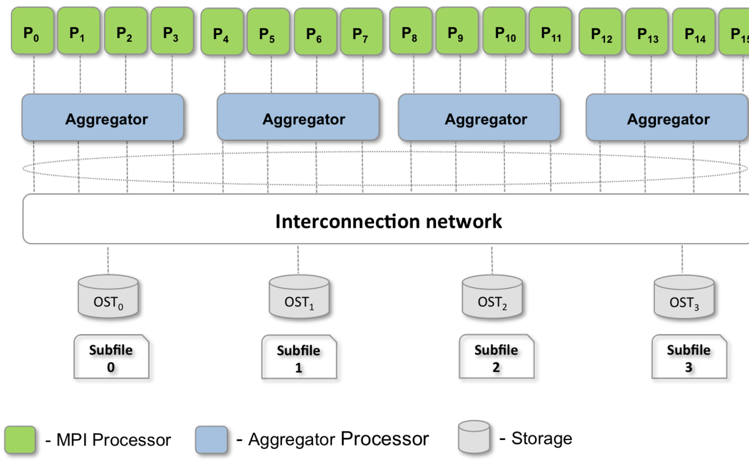


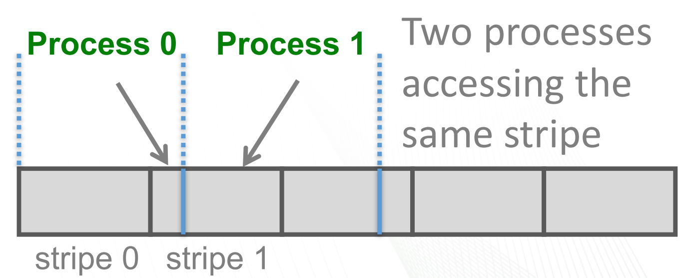
Common Techniques for I/O on HPC

- We do NOT recommend:
  - 1 Processor performs I/O (POSIX)
  - All processes write to their own file (POSIX)
  - All processes access one file ("Traditional" MPI-IO)
- We recommend to use a higher level library
  - Create self-describing, portable data format
  - Examples: NetCDF, pnetcdf, NetCDF-4, HDF5, ADIOS-BP, GRIB2, SEG-Y
- But this often leads to performance problems

Optimizations for a parallel file system

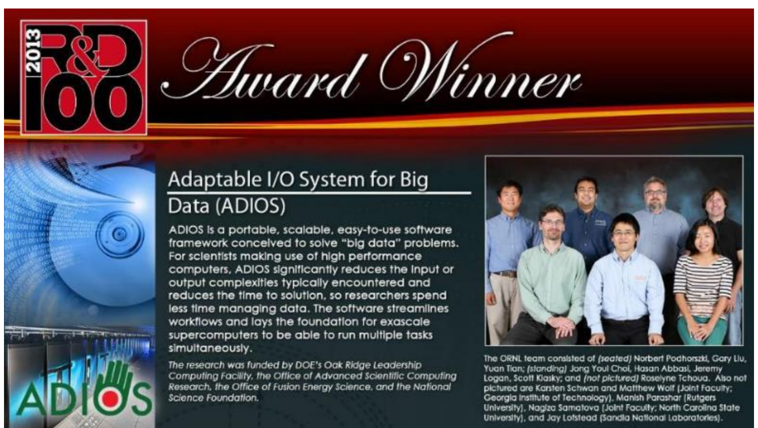
- Avoid latency (of small writes)
  - – Buffer data for large bursts
- Avoid accessing a file system target from many processes at once
  - – Aggregate to a small number of actual writers
    - proportionate to the number of file system targets, not MPI tasks
- Avoid lock contention
  - – by striping correctly
  - – or by writing to subfiles
- Avoid global communication during I/O
  - – ADIOS-BP file format





ADIOS

- An I/O abstraction framework
- Provides portable, fast, scalable, easy-to-use, metadata rich output
- Abstracts the API from the method
- Pick the I/O method at runtime - performance portability
- http://www.nccs.gov/user-support/center-projects/adios/



- Incorporates the "best" practices in the I/O middleware layer
- Applications are supported through OLCF INCITE program
- Outreach via on-line manuals, and live tutorials

ADIOS Approach

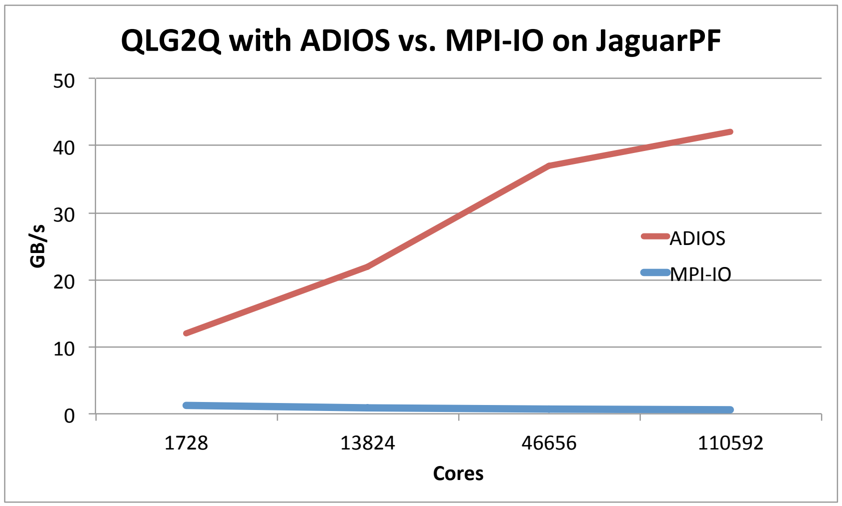
- I/O calls are of declarative nature in ADIOS
  - which process writes what
    - add a local array into a global space
  - adios_close() indicates that the process is done declaring all pieces that go into the particular dataset in that timestep
- I/O strategy is separated from the user code
  - aggregation,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

Writing with ADIOS

- XML approach
  - Easiest
  - Select I/O method at runtime by editing the XML file
  - Test skeletons can be generated from it
- Non-XML programming approach
  - Dynamically generating output variables
  - Multiple pieces of an array written from one process
- C/C++, F90, Numpy

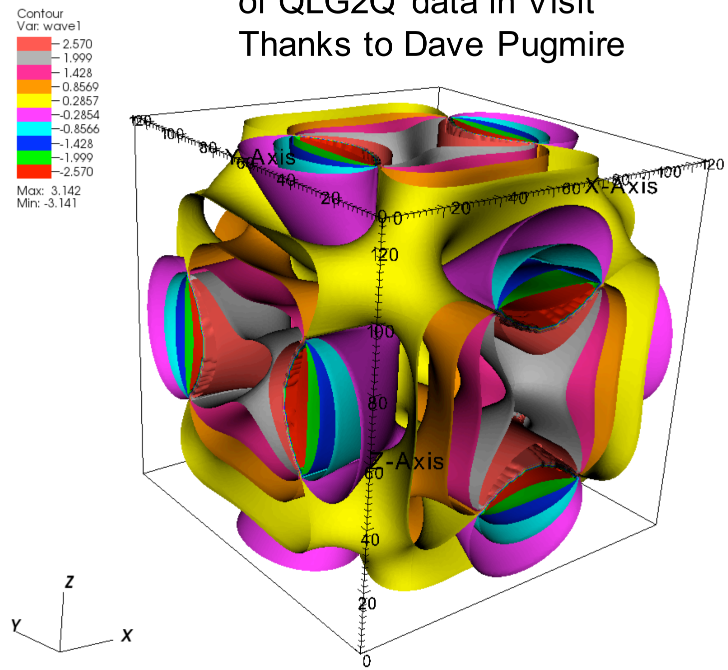
Writing with ADIOS: QLG2Q as example

- QLG2Q is a quantum lattice code developed in a DoD project.
- George Vahala (William & Mary), Min Soe (Rogers State)
- Large data size + many processors, > 50 MB per core, >100K cores



QLG2Q MPI-IO performance on JaguarPF @ OLCF

Isosurface visualization of QLG2Q data in Visit Thanks to Dave Pugmire

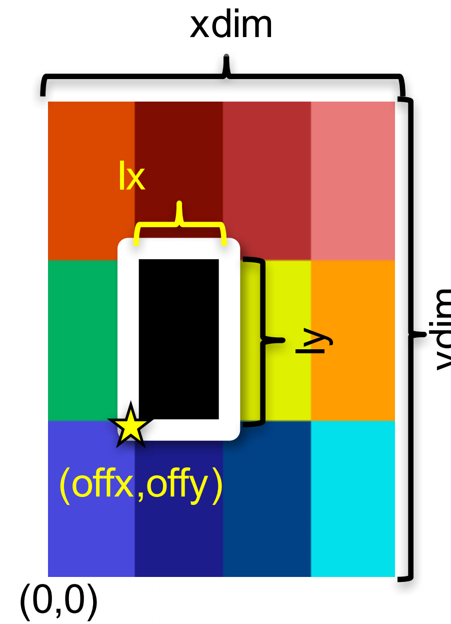


Source code to declare output action

```
call mpi_init (ierror)
call adios_init ("spin1.xml",mpi_comm_world, ierror)
...
call adios_open (adios_handle, "spin1", fname1, "w”, group_comm, ierr)
#include "gwrite_spin1.fh”
call adios_close (adios_handle,ierr)
...
call adios_finalize (rnk,ierror)
call mpi_finalize (ierror)
```

XML file (define output variables)

```
<?xml version="1.0"?>
<adios-config host-language="Fortran">
    <adios-group name="spin1”>
        <var name="xdim" gwrite="lg" type="integer"/>
        <!– ... Similar definitions for ydim, zdim, lx, ly, lz,
            offx, offy, offz -->
        <global-bounds dimensions="xdim,ydim,zdim”
                       offsets=”offx,offy,offz">
            <var name="qab1"
                gwrite="phia1(is:ie,js:je,ks:ke)"
                type="double complex" dimensions="lx,ly,lz"/>
            <!– ... Similar definitions for qab2, qab3, qab4, qab5, qab6 -->
        </global-bounds>
    </adios-group>
    ⋮
```



XML file to set runtime parameters

```
                    :
    <method group="spin1" method="MPI_AGGREGATE">
        num_aggregators=1024;num_ost=512
    </method>
    <buffer size-MB="256"
            allocate-time="now"/>
</adios-config>
```

XML file to set runtime parameters on Mira

```
⋮
    <method group="spin1" method="BGQ">


    </method>
    <buffer size-MB="60"
            allocate-time="now"/>
</adios-config>
```

- Topology-aware data movement was needed on BGQ
- With ADIOS BGQ method, QLG2Q achieves 120 GB/sec on 16 racks of Mira

Reading with ADIOS

- bpls
- C/C++, F90, Numpy, Matlab, pbdR, Java
- VisIt reads ADIOS .bp files

bpls (to show the mapping)

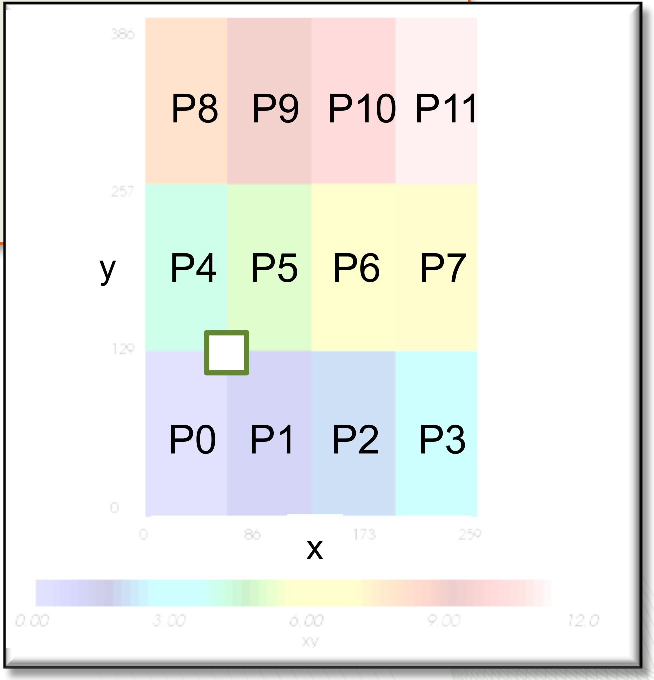
$ bpls -D heat.bp T   Min   /   Max   /   Avg   /   Std.dev

```
  double    T         6*{150, 160} = 0.000212009 / 999.47 / 442.536 / 318.995
        step 0:
          block  0: [  0: 49,   0: 39] = 5.0916 / 996.827/ 427.137/ 309.988
          block  1: [  0: 49,  40: 79] = 0.407082 / 943.871/ 216.189/ 275.313
          block  2: [  0: 49,  80:119] = 0.407082 / 943.871/ 216.189/ 275.313
          block  3: [  0: 49, 120:159] = 5.0916 / 996.827/ 427.137/ 309.988
          block  4: [ 50: 99,   0: 39] = 4.68652 / 943.676/ 269.445/ 283.684
          block  5: [ 50: 99,  40: 79] = 0.000212009 / 4.05808/ 0.482/ 0.868
          block  6: [ 50: 99,  80:119] = 0.000212009 / 4.05808/ 0.482/ 0.868
          block  7: [ 50: 99, 120:159] = 4.68652 / 943.676/ 269.445/ 283.684
          block  8: [100:149,   0: 39] = 5.0916 / 996.827/ 427.137/ 309.988
          block  9: [100:149,  40: 79] = 0.407082 / 943.871/ 216.189/ 275.313
          block 10: [100:149,  80:119] = 0.407082 / 943.871/ 216.189/ 275.313
          block 11: [100:149, 120:159] = 5.0916 / 996.827/ 427.137/ 309.988
        step 1:
          ...
```

bpls

- Use bpls to read in a 2D slice

```
$ bpls heat.bp -d T -s "0,49,39" -c "1,2,2" -n 2
double    T    6*{150, 160}
  slice (0:0, 49:50, 39:40)
  (0,49,39)    5.0916     4.15414
  (0,50,39)    4.99562  4.05808
```



- Note: both bpls and Matlab handle time as an extra dimension

F90

```
call adios_read_open_file (fh, filename, ADIOS_READ_METHOD_BP, group_comm, ierr)
call adios_get_scalar (fh, "gndx", gndx, ierr)     ! gets the value from metadata in memory
call adios_get_scalar (fh, "gndy", gndy, ierr)
readsize(1) = gndx
readsize(2) = gndy / nproc

! We can also inquire the dimensions, type and number of steps of a variable directly
call adios_inq_var (fh, "T", vartype, nsteps, ndim, dims, ierr)
ts = nsteps-1 ! Let's read the last timestep
offset(1)   = 0
offset(2)   = rank * readsize(2)
…
allocate( T(readsize(1), readsize(2)) )

! Create a 2D selection for the subset
call adios_selection_boundingbox (sel, 2, offset, readsize)
! Arrays are read by scheduling one or more of them and performing the reads at once
call adios_schedule_read (fh, sel, "T", ts, 1, T, ierr)
call adios_perform_reads (fh, ierr)
…
call adios_read_close (fh, ierr)
call adios_selection_delete (sel)
```

Numpy

```
$ python
>>> import adios as ad
>>> import numpy as np
>>> f = ad.file("heat.bp")
>>> f.printself()
...
>>> v = f.var['T']
>>> v.printself()
...
>>> T = v.read(offset=(49,39),count=(2,2),from_steps=0,nsteps=1)
>>> T
array([[ 5.09159701,  4.15414135],
       [ 4.99562066,  4.05807836]])

or just
>>> T = v.read((49,39), (2,2), 0, 1)
```

```
>>> v.printself()
=== AdiosVariable ===
      varid : 11
       type : float64
       ndim : 2
       dims : [150L, 160L]
     nsteps : 6
```

There is adios_mpi / mpi4py for parallel read

Matlab

```
>> data=adiosread('../heat.bp','T','Slice',[40 2;50 2;1 1])
data =
    5.0916    4.9956
    4.1541    4.0581
```

bpls result was

```
 (0,49,39)      5.0916   4.15414
 (0,50,39)      4.99562  4.05808
```

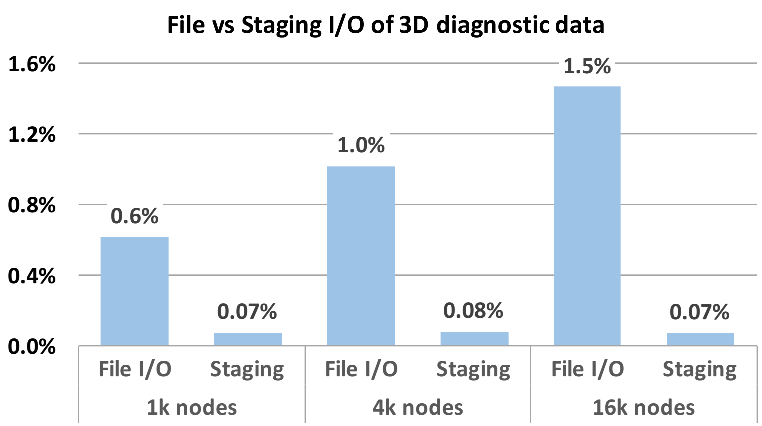
- Matlab is column-major, bpls (which is C) is row-major
  - 0, 49, 39 → 39, 49, 0
- Matlab array indices start from 1 (bpls/C starts from 0)
  - 0, 49, 39 → 40, 50, 1

```
>> f=adiosopen('../heat.bp');
>> T=adiosread(f.Groups,'T');
>> whos T
 Name       Size            Bytes   Class     Attributes
    T    160x150x6     1152000   double
>> adiosclose(f);
```

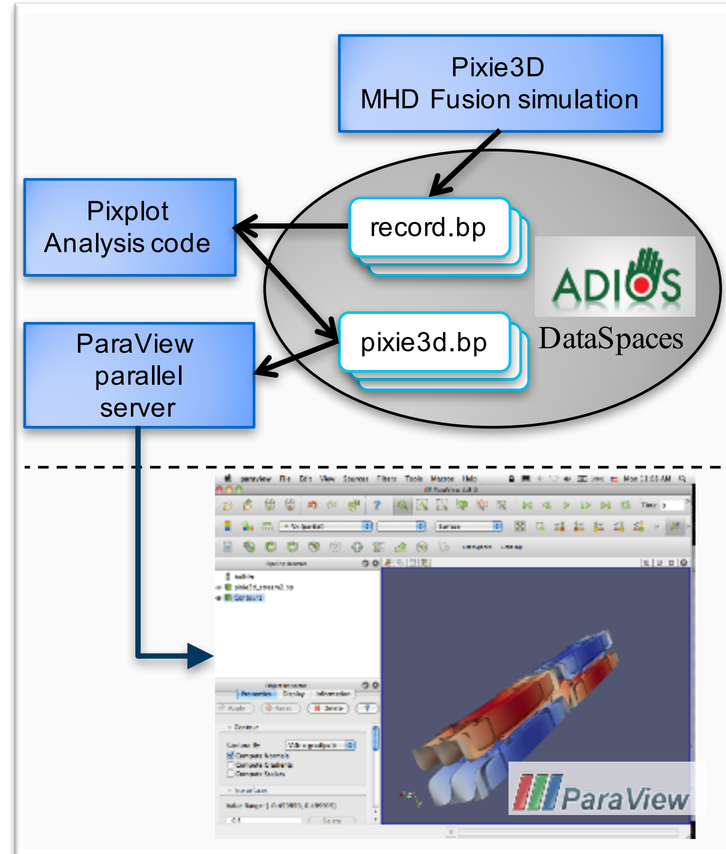
Coupling codes with ADIOS+staging method

ADIOS + DataSpaces/DIMES/FLEXPATH

+ asynchronous communication
+ easy, commonly-used APIs
+ fast and scalable data movement
+ not affected by parallel IO performance
- data aggregation/transformation at the coupler



Staging can eliminate output overhead
XGC on Titan, Diagnostics data 2 GB per simulation timestep



Interactive visualization pipeline of fusion simulation, analysis code and parallel viz. tool

What are Data Transformations?

- **Data transformations** are a class of technologies that change the format/encoding of data to optimize it somehow
  - Improve write performance
  - Reduce storage space
  - Accelerate read performance for analysis

Data Transformation	Purpose
Compression	Reduce I/O time and storage footprint
Filtering/sampling	Downsample data to reduce I/O and storage
Indexing	Speed up query-driven analytics/visualization
Level-of-detail encoding	Fast approximate reads, high-precision drilldown
Layout optimization	Speed up various read access patterns

Query

- Three approaches in ADIOS
  1. Use min/max (per-process statistics of arrays) to return list of process-blocks that can satisfy a query
  2. Use ALACRITY transform method to index data on-the-fly and get the exact points that satisfy the query
  3. Use FastBit to index data post-mortem and get the exact points that satisfy the query

ADIOS is a complex project

- ADIOS started as a project to solve I/O + analysis + visualization for fusion, but evolved
- Involves multiple institutions, multiple projects, many application areas
- ADIOS is a framework
  - CS researchers have a platform to place new I/O methods and try them for real codes
  - Application scientist can use “known” I/O methods as a backup when more advanced methods fail on new machines
- ADIOS is our research platform
  - Example: SC 2013: 4 papers, 5 posters

Questions