Data Compression in ADIOS

Scott Klasky, Norbert Podhorszki, Ana Gainaru

EXASCALE COMPUTING PROJECT

National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline

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

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

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

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

BREAK

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

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

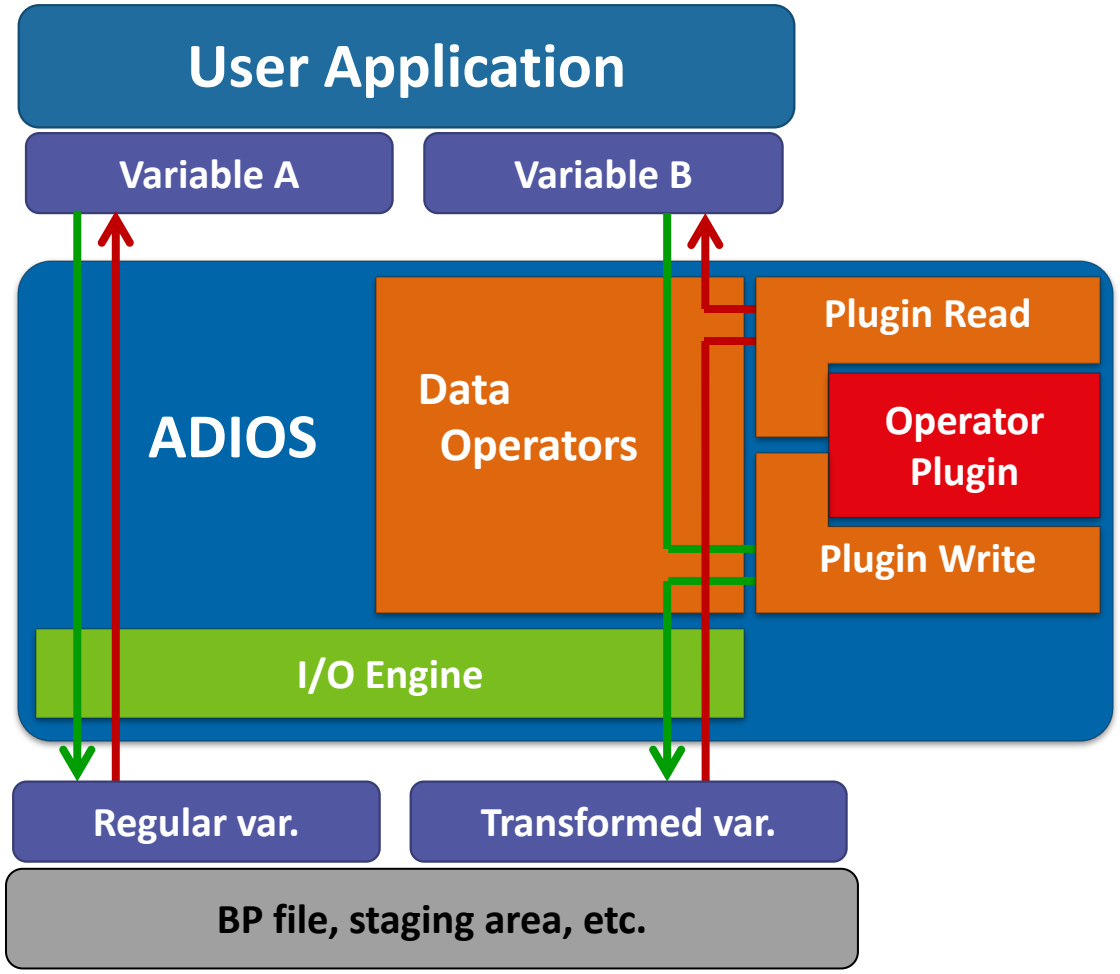
Part IV: In situ data analysis using I/O staging (1 hour) (Intermediate/Advanced)

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

Wrap-up

ADIOS Operators

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



Operator in ADIOS

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

Operators in source code

```
Operator ADIOS::DefineOperator( name, type )

Variable::AddOperation( Operator, {Parameter[,…]} )
```

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

auto varF = io.DefineVariable<float>("r32", shape, start, count, adios2::ConstantDims);

auto varD= io.DefineVariable<double>("r64", shape, start, count, adios2::ConstantDims);
```

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

```
varF.AddOperation(zfpOp, {{"accuracy", "0.01"}});

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

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

Operators in the ADIOS XML configuration file

- Describe **runtime** parameters for each IO grouping
  - Select the Engine for writing
    - BP4, SST, InSituMPI, DataMan, SSC engines support compression
    - HDF5 engine does not support compression
  - **Define an Operator**
  - **Select an Operation for Variables (operator + parameters)**
    - "zfp", "mgard", "sz" – all support "accuracy" parameter
- See https://github.com/ornladios/ADIOS2-Examples/blob/master/source/cpp/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
```