Exascale I/O tutorial

Presenters: N. Podhorszki, Scott Klasky
Oak Ridge National Laboratory, Computer Science and Mathematics Division

Excalibur - US extreme data workshop and tutorials

Cambridge, UK

July 18, 2023

Collaborators: Apps, Workflow, Data Management, Reduction, Viz

- **Scott Klasky**
- **Norbert Podhorszki**
- **Qing Liu**
- **Karsten Schwan**
- **Jay Lofstead**
- **Mark Ainsworth**
- **C.S. Chang**
- **Ana Gainaru**
- Hasan Abbasi
- Rick Archibald
- Chuck Atkins
- Vicente Bolea
- Phillipe Bonnet
- Michael Bussmann
- Jieyang Chen
- Hank Childs
- Jong Choi
- Michael Churchill
- Nathan Cummings
- Shaun de Witt
- Philip Davis
- Ciprian Docan
- Greg Eisenhauer
- Stephane Ethier
- Ian Foster
- Dmitry Ganyushin
- Kai Germaschewski
- Berk Geveci
- William Godoy
- Qian Gong
- Junmin Gu
- Jon. Hollocombe
- Kevin Huck
- Axel Huebl
- Toby James
- Chen Jin
- Mark Kim
- Brad King
- James Kress
- S.H. Ku
- Ralph Kube
- Tahsin Kurc
- Xin Liang
- Zhihong Lin
- Jeremy Logan
- Thomas Maier
- Kshitij Mehta
- Ken Moreland
- Todd Munson
- Manish Parashar
- Franz Pöschel
- Dave Pugmire
- Anand Rangarajan
- Sanjay Ranka
- Stefanie Reuter
- Caitlin Ross
- Nagiza Samatova
- Ari Shoshani
- Eric Suchyta
- Fred Suter
- Keichi Takahashi
- William Tang
- Roselyne Tchoua
- Nick Thompson
- Eric Suchyta
- Seiji Tsutsumi
- Ozan Tugluk
- Lipeng Wan
- Ruonan Wang
- Xinying Wang
- Ben Whitney
- Andreas Wicenec
- Matthew Wolf
- John Wu
- Bing Xie
- Fan Zhang
- Fang Zheng

Outline

- ADIOS 2 concepts and API (C++, Python, F90)
- BREAK
- Hands on with ADIOS 2 – Files
- QUICK BREAK
- ADIOS @ scale
- Staging with ADIOS
- Hands on with ADIOS 2 – Staging
- END

These slides are in https://tinyurl.com/adios-eied
https://users.nccs.gov/~pnorbert/adios-tutorial-eied-2023-uk.pdf
/rds/project/hpc/rds-hpc-training/rds-eied-workshop/shared/adios-tutorial.pdf

Outline

- ADIOS 2 concepts and API (C++, Python, F90)
- BREAK
- Hands on with ADIOS 2 – Files
- QUICK BREAK
- ADIOS @ scale
- Staging with ADIOS
- Hands on with ADIOS 2 – Staging
- END

ADIOS Concepts and C++ API

ADIOS Useful Information and Common tools

- ADIOS documentation: https://adios2.readthedocs.io/en/latest/index.html
- ADIOS Examples: https://adios2-examples.readthedocs.io/en/latest/
- ADIOS source code: https://github.com/ornladios/ADIOS2
  - Written in C++, wrappers for Fortran, Python, Matlab, C
  - Contains command-line utilities (bpls, adios_reorganize ..)
- This tutorial's code example (Gray-Scott): https://github.com/pnorbert/adiosvm.git
- Online help:
  - ADIOS2 GitHub Issues: https://github.com/ornladios/ADIOS2/issues

- Two movies showing the Tutorial for post-processing and on-line processing
- https://users.nccs.gov/~pnorbert/GrayScottPost.mp4
- https://users.nccs.gov/~pnorbert/GrayScottInsitu.mp4

ADIOS Approach: "How"

- I/O calls are of declarative nature in ADIOS
  - which process writes/reads what
    - add a local array into a global space (virtually)
  - EndStep() indicates that the user is done declaring all pieces that go into the particular dataset in that output step or what pieces each process gets
- I/O strategy is separated from the user code
  - aggregation, number of sub-files, target file-system hacks, and final file format not expressed at the code level
- This allows users
  - to choose the best method available on a system without modifying the source code
- This allows developers
  - to create a new method that's immediately available to applications
  - to push data to other applications, remote systems or cloud storage instead of a local filesystem

ADIOS basic concepts

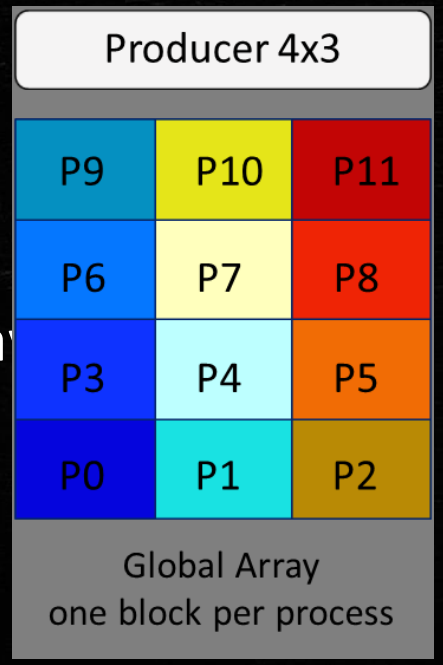
- Self-describing Scientific Data
- Variables
  - multi-dimensional, typed, distributed arrays
  - single values
    - Global: one process, or Local: one value per process
- Attributes
  - static information
    - for humans or machines
  - global, or assigned to a variable

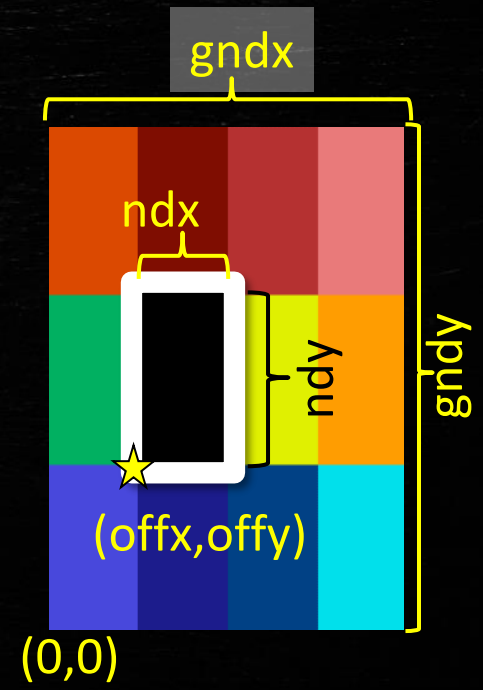
Self-describing Scientific Data

```
real     /fluid_solution/scalars/PREF                    scalar = 0
string   /fluid_solution/domain14/blockName/blockName    scalar = "rotor_flux_1_Main_Blade_skin"
integer  /fluid_solution/domain14/sol1/Rind              {6} = 2 / 2
real     /fluid_solution/domain14/sol1/Density           {8, 22, 52} = 0.610376 / 1.61812
real     /fluid_solution/domain14/sol1/VelocityX         {8, 22, 52} = -135.824 / 135.824
real     /fluid_solution/domain14/sol1/VelocityY         {8, 22, 52} = -277.858 / 309.012
real     /fluid_solution/domain14/sol1/VelocityZ         {8, 22, 52} = -324.609 / 324.609
real     /fluid_solution/domain14/sol1/Pressure          {8, 22, 52} = 1 / 153892
real     /fluid_solution/domain14/sol1/Nut               {8, 22, 52} = -0.00122519 / 1
real     /fluid_solution/domain14/sol1/Temperature       {8, 22, 52} = 1 / 362.899
string   /fluid_solution/domain17/blockName/blockName    scalar = "rotor_flux_1_Main_Blade_shroudga

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

Global Array: data produced by multiple processes

- N-dimensional array
  - **Shape**
- Has a type (int32, double, etc.)
  - **Type**
- Blocks of data are written into the arra
  - **Start** (offset)
  - **Count** (size of block)





Shape = {gndx, gndy}

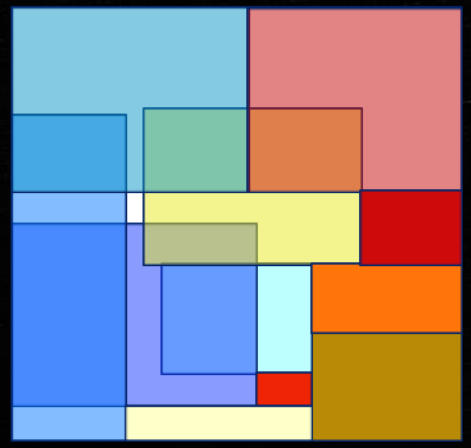
Start = {offx, offy}

Count = {ndx, ndy}

Global Array: data produced by multiple processes

- These are valid global arrays
  - One process can contribute more than one block
  - Some process may not write anything at all
  - Holes can be left in the global array
  - Overlapping of blocks is allowed

Global Array with overlapping blocks

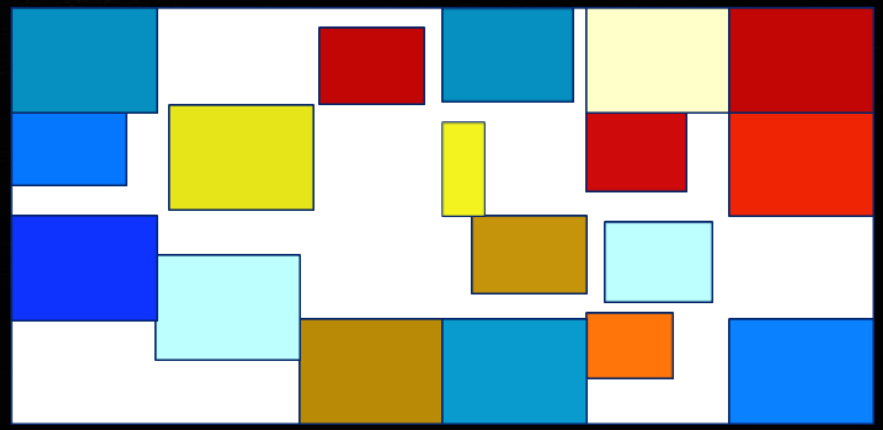


An overlapped cell has "one of the" values

12 Producers
multiple blocks per process

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

Global Array sparsely filled out



Read returns "nothing" for those cells.

ADIOS basic concepts

- Step
  - Producer outputs a set of variables and attributes at once
    - This is an **ADIOS Step**
  - Producer iterates over computation and output steps
- Producer outputs multiple steps of data
  - e.g. into multiple, separate files, or into a single file
  - e.g. steps are transferred over network
- Consumer processes step(s) of data
  - e.g. one by one, as they arrive
  - e.g. all at once, reading everything from a file
    - not a scalable approach

ADIOS Steps: Rules and constraints

- Step is not necessarily tied to the application timesteps
  - a Step can be constructed over time
- Entire content of a Step is either completely written or not at all
- A new Step can be very different from the previous step
  - may contain a completely different set of variables
  - array sizes can change
  - array decomposition can change
- Consumer is guaranteed to have access to entire content of Step as long as it wants it
- Entire content of a Step must fit into the producer's memory as a copy

ADIOS coding basics

- Objects
  - ADIOS
  - Variable
  - Attribute
  - IO
    - a group object to hold all variable and attribute definitions that go into the same output/input step
    - settings for the output/input
    - settings may be given before running the application in a configuration file
  - Engine
    - the output/input stream
  - Operator
    - a compression, reduction, data transformation operator for output variables

ADIOS object

- The container object for all other objects
- Gives access to all functionality of ADIOS

```
#include <adios2.h>
adios2::ADIOS  adios(configfile, MPI communicator);
```

- Notes:
  - both arguments are optional
    - `adios(), adios("config.xml"), adios(comm)`
  - Normally use 1 config file and then have different communicators for each I/O target

ADIOS2-Examples/source/cpp/gray-scott/simulation/main.cpp

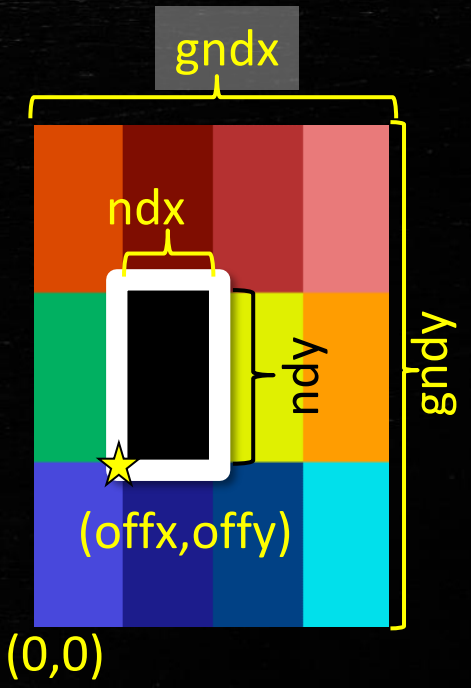
IO object

- Container for all variables and attributes that one wants to output or input at once
- Application settings for IO
- User run-time settings for IO – from configuration file (or input parameters)
  - a **name** is given to the IO object to identify it in the configuration

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

Variable

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




```
adios2::Variable<double> &varT = io.DefineVariable<double>
(
    "T",                // name in output/input
    {gndx, gndy},       // Global dimensions (2D here)
    {offx, offy},       // starting offsets in global space
    {ndx, ndy}          // local size
);
```

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

Hint: if it's only checkpoint restart, just use global dimensions {NPROC, N}, local offsets {Rank, 0},

Engine object

- To perform the IO (for a single output step)

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


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


writer.Close()
```

T is used in here!
Do not invalidate content of T before this!

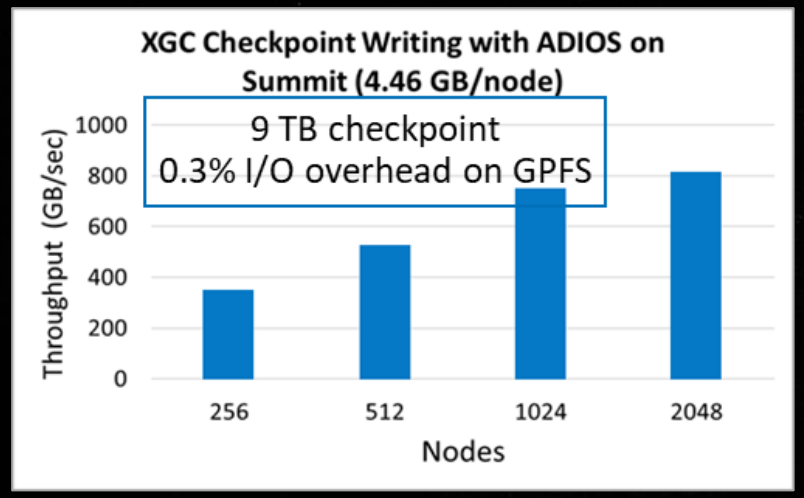
Reading is similar, but we can read from any number of procs

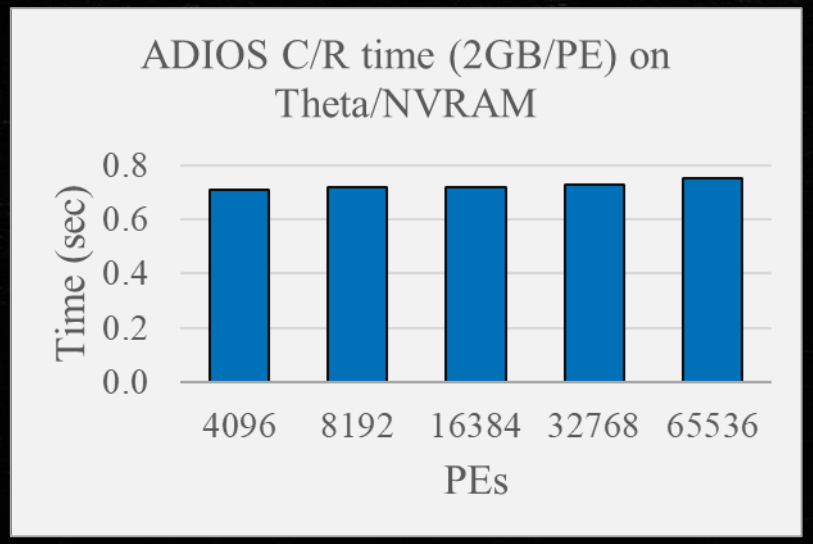
```
adios2::IO io = adios.DeclareIO("CheckpointRestart");
adios2::Engine reader =
     io.Open("checkpoint.bp", adios2::Mode::Read);


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

reader.Close()
```

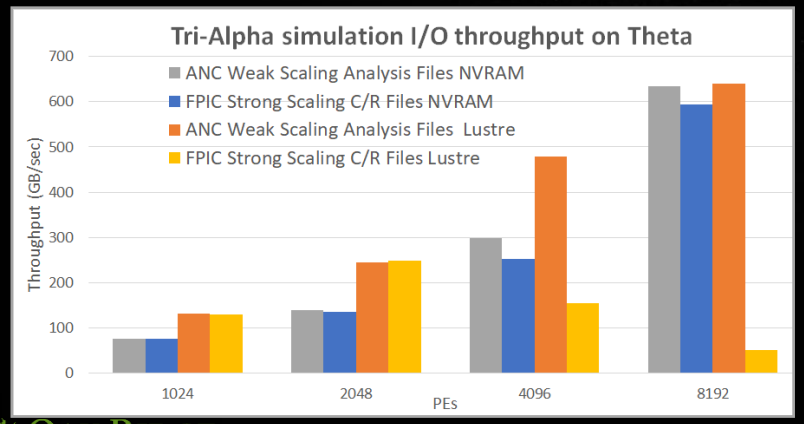
Reserve memory for T before this

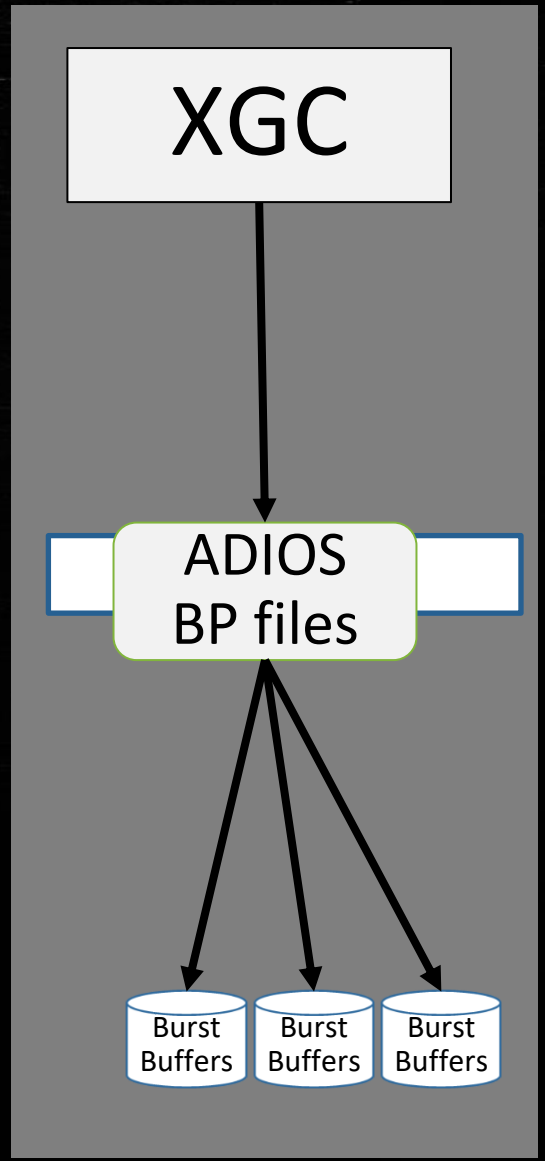
ADIOS APIs for self describing data output for C/R to NVRAM

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









Analysis/visualization data

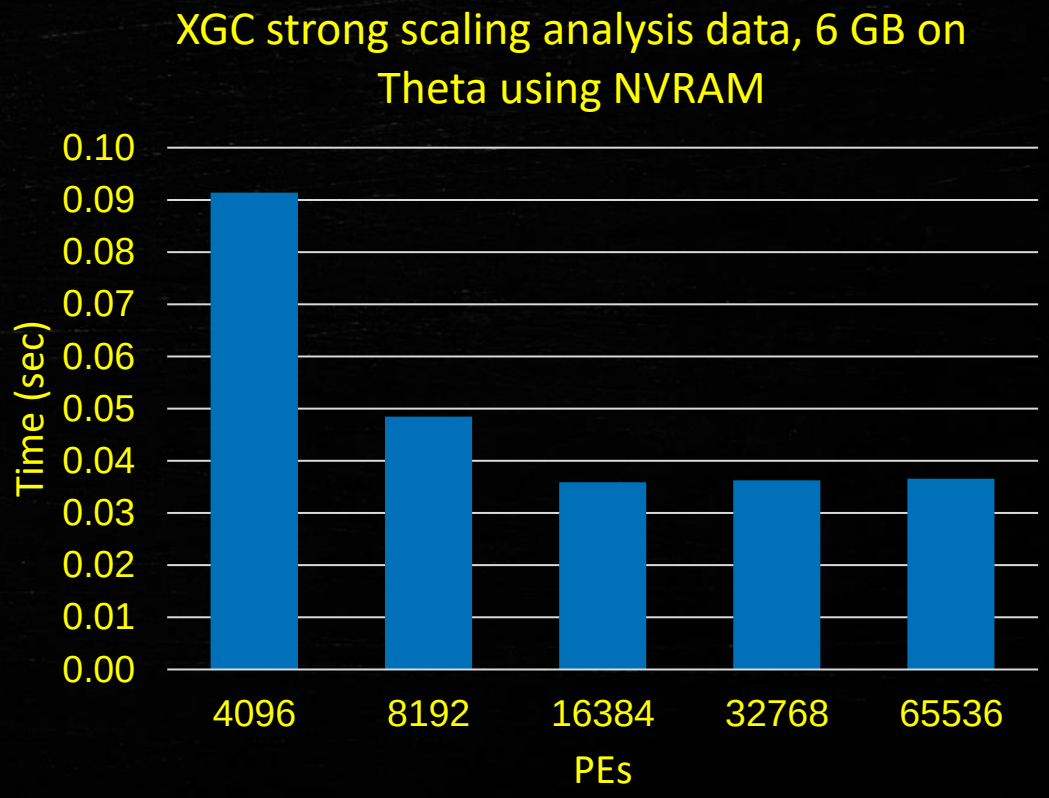
```
adios2::IO io = adios.DeclareIO("Analysis_Data");
if (!io.InConfigFile()) {
    io.SetEngine("FileStream");
}
adios2::Variable<double> varT = io.DefineVariable<double>
(
    "Temperature",          // name in output/input
    {gndx,gndy,gndz},       // Global dimensions (3D here)
    {offx, offy, offz},     // starting offsets in global space
    {nx,ny,nz}              // local size
);
io.DefineAttribute<std::string>("unit", "C", "Temperature");
```

double	Temperature	10*{20, 30, 40} = 8.86367e-07 / 200
string	Temperature/unit	attr = "C"

Engine object

- To perform the IO

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


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


writer.Close()
```



Put API explained

```
engine.Put(varT, T.data())
```

- Equivalent to

```
engine.Put(varT, T.data(), adios2::Mode:Deferred)
```

- This does NOT do the I/O (to disk, stream, etc.) once put return.
- you can only reuse the data pointer after calling `engine.EndStep()`

```
engine.Put(varT, T.data(), adios2::Mode:Sync)
```

- This makes sure data is flushed or buffered before put returns
- Get() works the same way
- The default mode is deferred
- BP5 specific:
  - Large Deferred Put() is NOT buffered
  - Use Sync mode only when you need it (when you need to modify the data pointer before EndStep)

ADIOS Python API
Basically, the C++ API in Python

ADIOS engines – change from BP5 to HDF5

1. <io name="SimulationOutput">
2. <engine type="**BP5**"/>
3. </io>

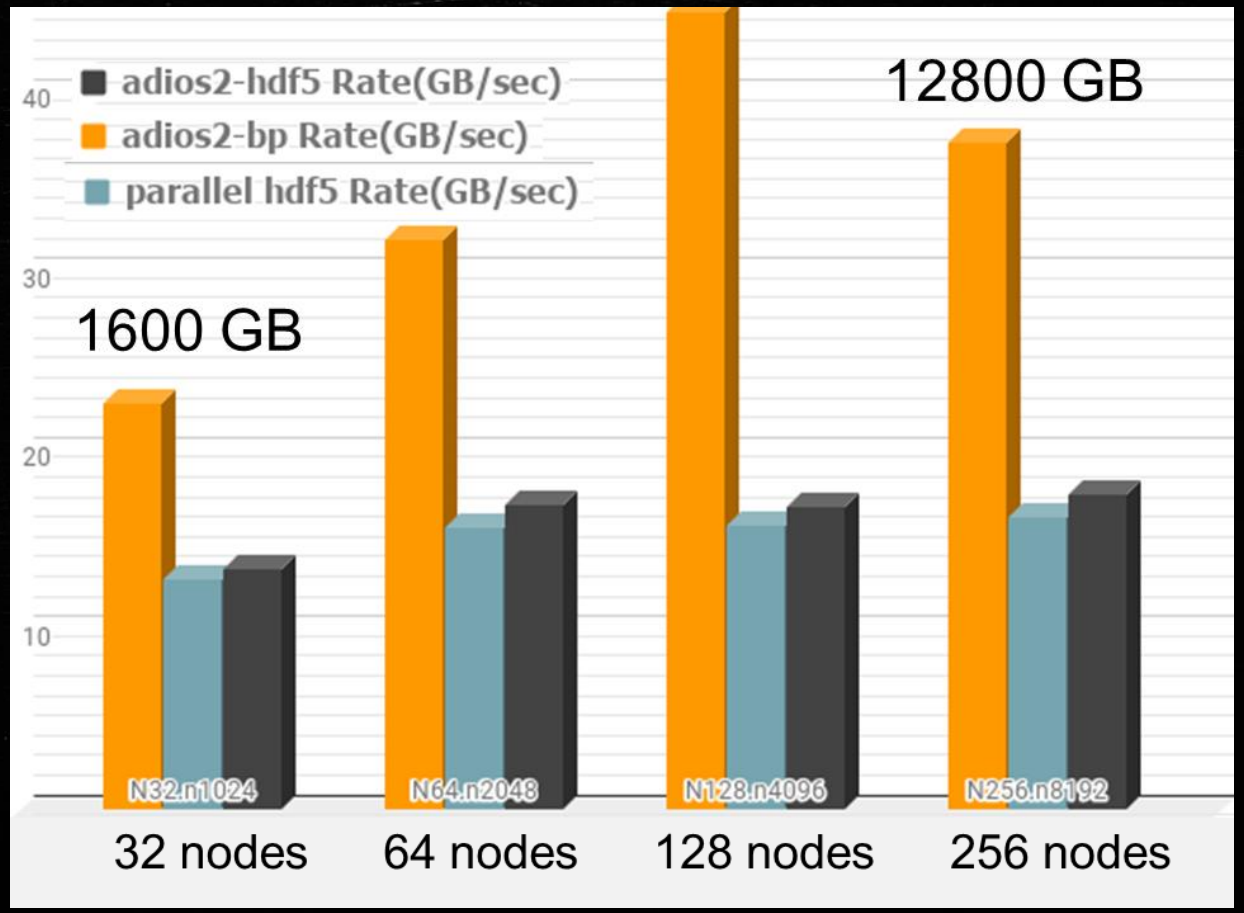
Change Engine name

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

- or in the source code

```
io.SetEngine("HDF5");
```

Cori + Lustre, for the heat equation



Python common

Serial python:

```
import numpy
import adios2

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

Parallel python with MPI:

```
from mpi4py import MPI
import numpy
import adios2

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

Python API start: ADIOS, IO and Engine objects

```
adios = adios2.ADIOS("adios2.xml")
adios = adios2.ADIOS("adios2.xml", comm, True)

io = adios.DeclareIO("SimulationOutput")
fr = io.Open("stream.bp", adios2.Mode.Read)

# adios2.Mode.Read - input stream step-by-step
# adios2.Mode.ReadRandomAccess - to see all steps at once (file)
# adios2.Mode.Write - to create an output stream
# adios2.Mode.Append - to append new steps to existing file

fr.Close()
```

Python API: Step-by-step reading

```
while True:
    status = fr.BeginStep()
    if status == adios2.StepStatus.EndOfStream:
        print("-- no more steps found --")
        break
    elif status == adios2.StepStatus.NotReady:
        sleep(1)
        continue
    elif status == adios2.StepStatus.OtherError:
        print("-- error with stream --")

    cur_step = fr.CurrentStep()
    ...
    fr.EndStep()
```

```
while True:
    status = fr.BeginStep()
    if status != adios2.StepStatus.OK:
        break
    cur_step = fr.CurrentStep()
    ...
    fr.EndStep()
```

Python Read API: List variables

```
vars_info = io.AvailableVariables()


for name, info in vars_info.items():
    print("variable_name: " + name)
    for key, value in info.items():
        print("\t" + key + ": " + value)
    print("\n")



# NOTE: list of variables may change
# from step to step
```

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

Variable access, selection and read

- var = io.InquireVariable("U")
- shape = var.Shape()
  - E.g. [64, 64, 64]
- Allocate Numpy array for reading
  - data = numpy.empty([shape[1], shape[2]], dtype=np.float64)
- Selection to read: tuple of start and count (e.g. 2D slice of 3D array):
  - var.SetSelection([[int(shape[0]/2),0,0], [1,shape[1],shape[2]]])
- Read data now
  - fr.Get(var, data, adios2.Mode.Sync)

ADIOS Python API
File reading with ReadRandomAccess

Python API start: Engine objects

```
fr = io.Open("stream.bp", adios2.Mode.ReadRandomAccess)
vars_info = io.AvailableVariables()
ustep = int(vars_info["U"]["AvailableStepsCount"])
vu = io.InquireVariable("U")
vu.SetStepSelection([0, ustep])
print("Number of var 'step' steps after SetStepSelection =
{0}".format(vstep.Steps()))

# read first 4 elements of a 1D array A for all steps
nelems = 4
data = np.zeros([ustep*nelems],  dtype=np.float64)
vA.SetSelection([ [0], [4] ])
fr.Get(vA, data, adios2.Mode.Sync)
```

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

ADIOS Fortran API

- See documentation at https://adios2.readthedocs.io/en/latest/api_full/api_full.html#fortran-bindings
- See API source code in ADIOS2 source
  - bindings/Fortran/modules
  - https://github.com/ornladios/ADIOS2/tree/master/bindings/Fortran/modules

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)
```

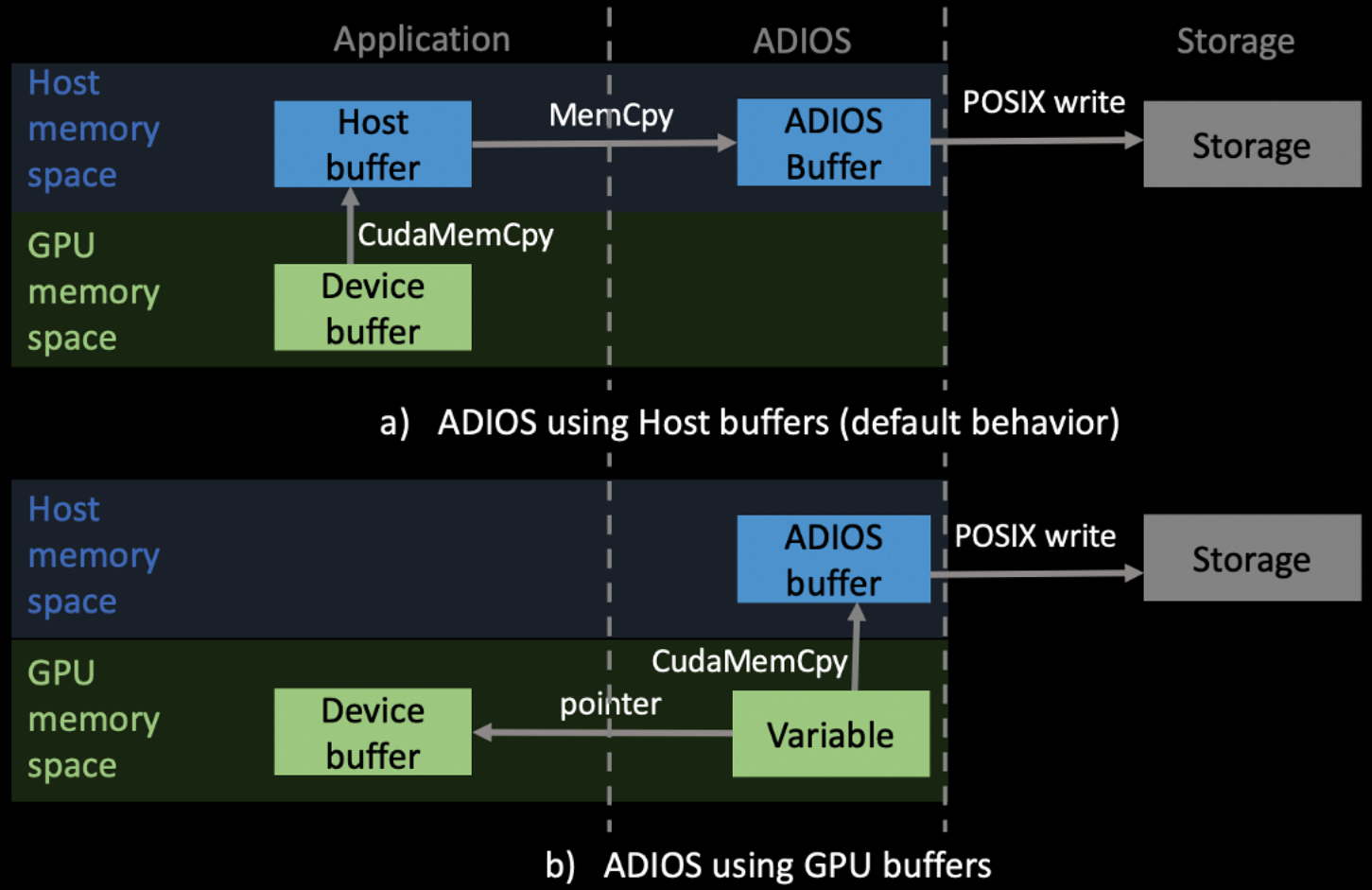
Fortran Read API

```
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

GPU-aware I/O

- Allow applications to give ADIOS GPU buffers
  - Decrease number of copies of the data
  - Transparent performance portability to different GPU architectures
  - Allow ADIOS to use GPU direct to storage, compression on GPU, or other optimizations



API for GPU-aware I/O

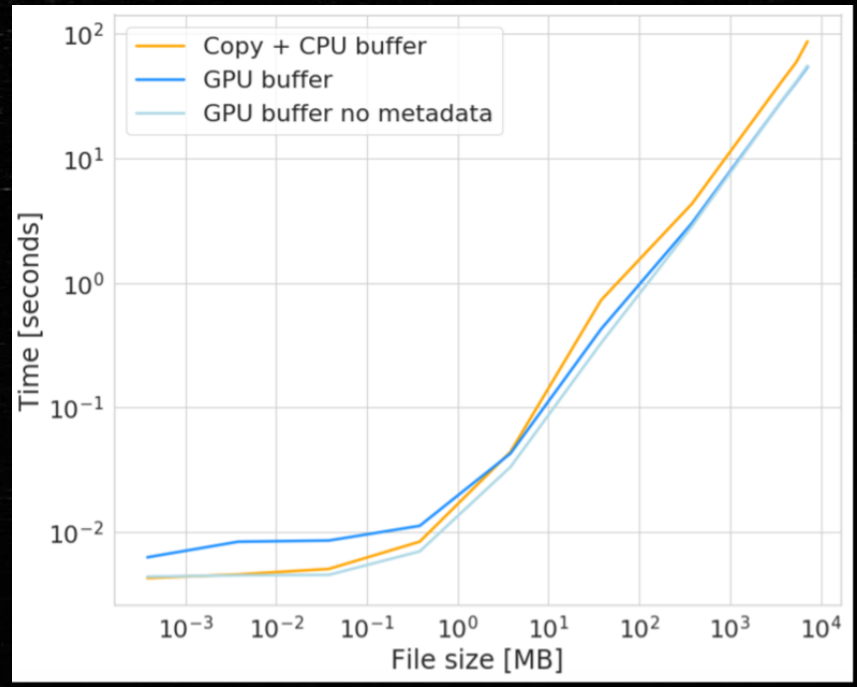
- Build ADIOS2 with CUDA support –D ADIOS2_USE_CUDA=ON
- The user provides a memory space associated with ADIOS2 variables
  - If not set ADIOS2 will detect automatically the memory space

```
adios2::Engine bpWriter;
...
auto data = io.DefineVariable<float>("data", shape, start, count);

bpWriter.Put(data, cpuData);

data.SetMemorySpace(adios2::MemorySpace::GPU);
bpWriter.Put(data, gpuData);
```

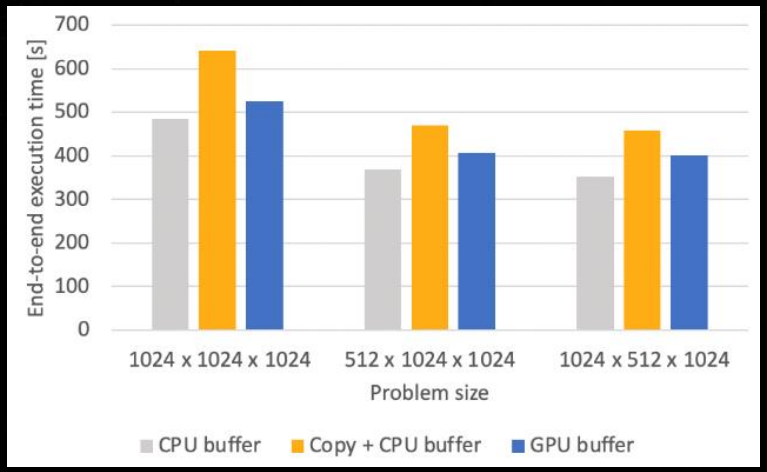
- ADIOS2 saves pointers to data and copies data to internal CPU buffers (in deferred or sync mode)
- Computes metadata for each Get/Put using CUDA kernels

Overhead for detecting where buffers are allocated

CPU STD vector	CUDA CPU buffer	CUDA GPU buffer
5-6 µs	1-2 µs	1-2 µs



Results on I/O kernels and OpenPMD



GPU backends in ADIOS2

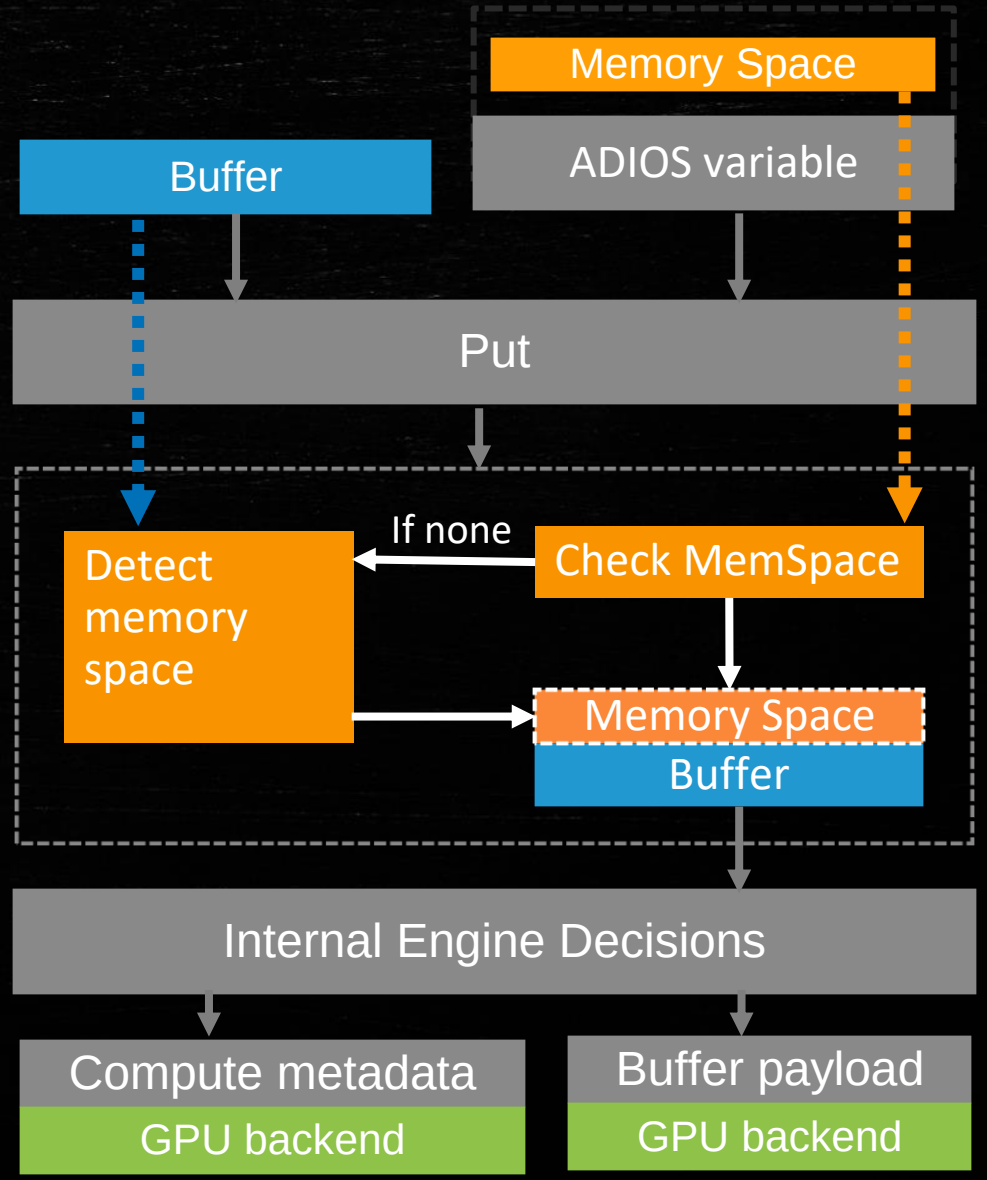
- Backends are chosen during build
  - ADIOS2 can have only one GPU backend active at a given time

```
cmake –D ADIOS2_USE_{BACKEND}=ON
```

  - Current supported backends: CUDA, Kokkos (CUDA), Kokkos (HIP), Kokkos (SYCL)
  - The Kokkos backend is chosen based on the Kokkos build linked to ADOIOS2

```
# build Kokkos
cmake -D Kokkos_ENABLE_CUDA=ON -D Kokkos_ARCH_VOLTA70=ON

# build ADIOS2
cmake -DKokkos_ROOT=${KOKKOS_HOME}/install
-D CMAKE_CUDA_ARCHITECTURES=70 -D ADIOS2_USE_KOKKOS=ON
```

  - The application source code does not change
    - Underneath ADIOS2 is using Kokkos or CUDA functions for computing the metadata and copying the user buffer to ADIOS2 internal buffers



Running an example

- Kokkos backend (same application code, simple write benchmark)
  - Build Kokkos on Summit with CUDA enabled

```
cmake -D CMAKE_CXX_COMPILER=${KOKKOS_HOME}/kokkos/bin/nvcc_wrapper
            -D Kokkos_ENABLE_CUDA=ON
            -D Kokkos_ARCH_VOLTA70=ON
            -D CMAKE_CXX_STANDARD=17
            -D CMAKE_POSITION_INDEPENDENT_CODE=TRUE
```

  - Build ADIOS2 with Kokkos backend

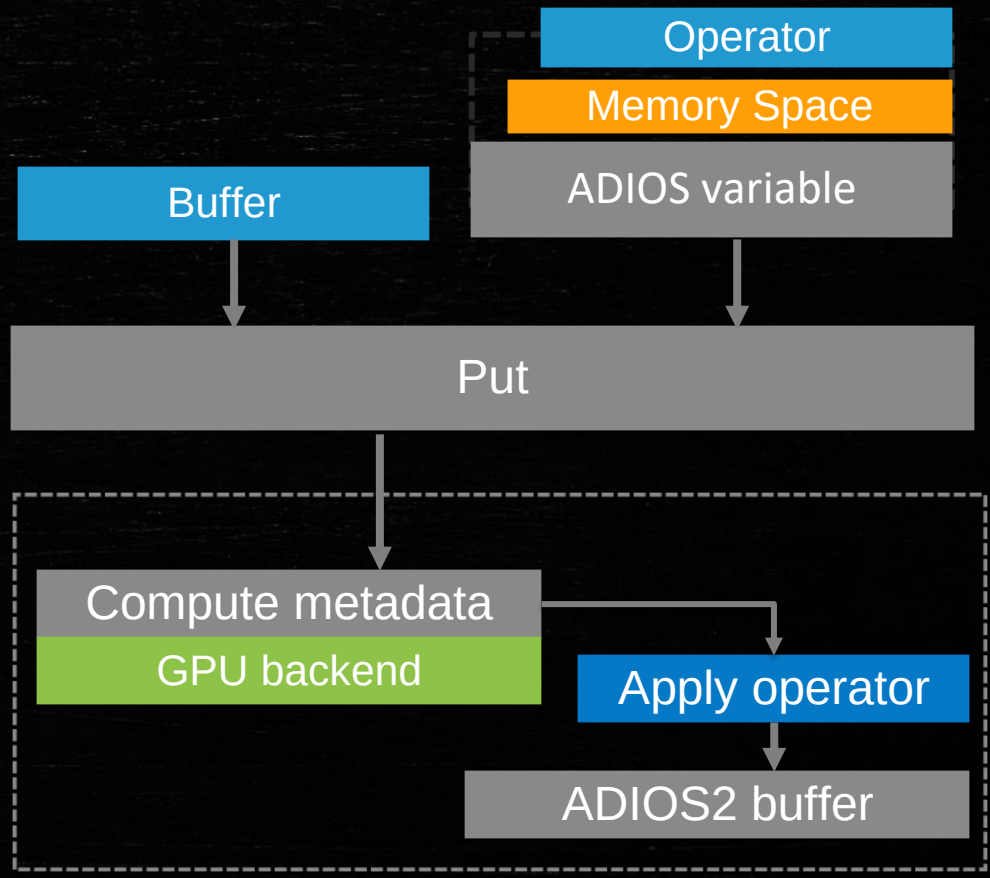
```
cmake -D CMAKE_CXX_STANDARD=17
            -D Kokkos_ROOT=${KOKKOS_HOME}/install
            -D CMAKE_CUDA_ARCHITECTURES=70
            -D CMAKE_CXX_COMPILER=${KOKKOS_HOME}/kokkos/bin/nvcc_wrapper
```

  - The two libraries need to use the same configuration
    - C++ standard 17, VOLTA70 architecture, nvcc_wrapper as the C++ compiler
    - Currently only supported for gcc >= 10

Compression with GPU-aware I/O

- No changes required in the source code
  - Operator attached to a variable
  - Memory space attached to a variable

```
auto var = io.DefineVariable<double>(”test", shape, start, count);

// define an operator
adios2::Operator varOp =
  adios.DefineOperator("mgardCompressor", adios2::ops::LossyMGARD);

//attach operator to variable
var.AddOperation(varOp, parameters);

var.SetMemorySpace(adios2::MemorySpace::GPU); // optional
bpWriter.Put(var, gpuSimData);
```



- Internal logic
  - Metadata is computed using the GPU backend
  - The operator is applied on the GPU buffer and the compressed data is copied directly in the ADIOS buffer

Operators that support GPU buffers:

- MGARD, ZFP
- The operators need to be built with GPU enable

Building applications with ADIOS

Compile ADIOS2 codes

- CMake
  - Use MPI_C and ADIOS2 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::cxx11_mpi   MPI::MPI_C)
```

  - Configure application by adding ADIOS installation to search path

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

  - Available ADIOS2 targets: cxx11 c, fortran, cxx11_mpi, c_mpi, fortran_mpi

Compile ADIOS2 codes

- Makefile
  - Add ADIOS2 library paths to LD_LIBRARY_PATH
  - Use adios2_config tool to get compile and link options

```
ADIOS2_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 ${ADIOS2_FINC} io_adios2.F90
	${FC}   -g -o heatSimulation heatSimulation heat_vars.o io_adios2.o  ${ADIOS2_FLIB}
```

Configure and build the ADIOS2-Examples repository (example)

```
$ cd ~/ADIOS2-Examples/
$ mkdir -p build-cmake
$ cd build-cmake
$ cmake \
  -DCMAKE_INSTALL_PREFIX=/home/adios/Tutorial \
  -DCMAKE_PREFIX_PATH="/opt/adios2;/opt/hdf5-parallel/default;/opt/zfp/default;/opt/sz/default“ \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo \
   ..
$ make -j 4
$ make install
$ cd /home/adios/Tutorial/share/adios2-examples/gray-scott

$ export PATH=$PATH:/home/adios/Tutorial/bin
$ export PYTHONPATH=/opt/adios2/lib/python3/dist-packages
$ export LD_LIBRARY_PATH=/opt/hdf5-1.13.0/parallel/lib::/opt/sz/default/lib:/opt/zfp/default/lib
```

Outline

- ADIOS 2 concepts and API (C++, Python, F90)
- BREAK
- Hands on with ADIOS 2 – Files
- QUICK BREAK
- ADIOS @ scale
- Staging with ADIOS
- Hands on with ADIOS 2 – Staging
- END

ADIOS2 tutorial on CSD3 in Cambridge

setup for users

These slides are in https://tinyurl.com/adios-eied

/rds/project/hpc/rds-hpc-training/rds-eied-workshop/shared/adios-tutorial.pdf

https://users.nccs.gov/~pnorbert/adios-tutorial-eied-2023-uk.pdf

login-icelake.hpc.cam.ac.uk

```
$ ssh -X -l <username> login-icelake.hpc.cam.ac.uk
$ source ~/rds/rds-eied-workshop/shared/setup.sh
$ adios2-config -v
ADIOS 2.9.0
Copyright (c) 2019 UT-BATTELLE, LLC
Licensed under the Apache License, Version 2.0
```

```
$ cd
$ cp -rp ~/rds/rds-eied-workshop/shared/tutorial .
$ cd tutorial
$ pwd
/home/username/tutorial
```

Gray-Scott example
From MPI-IO to ADIOS2

hands-on instructions
on CSD3

Software used in this tutorial

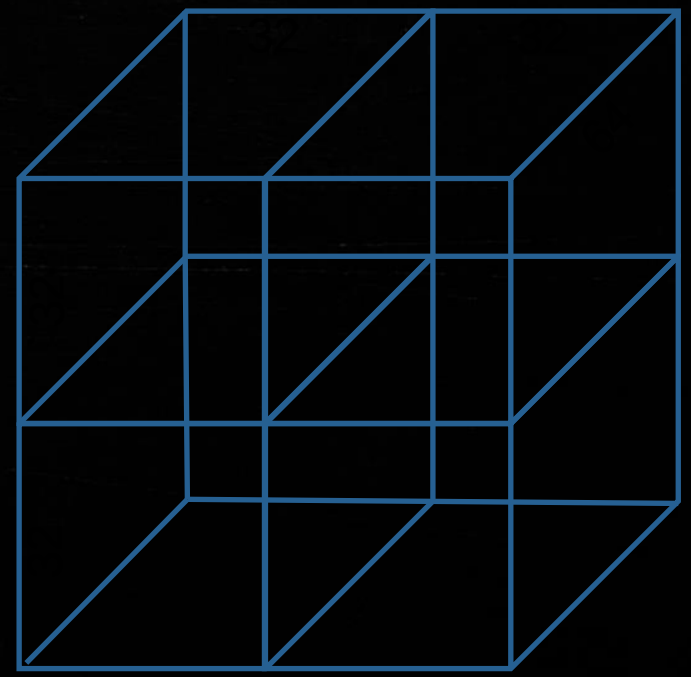
- ADIOS 2.9.0
  - https://github.com/ornladios/ADIOS2/releases/tag/v2.9.0
- adiosvm
  - https://github.com/pnorbert/adiosvm.git
  - Tutorial/gs-mpiio
  - Tutorial/gs-adios2

ADIOS Useful Information and Common tools

- ADIOS documentation: https://adios2.readthedocs.io/en/latest/index.html
- ADIOS Examples: https://adios2-examples.readthedocs.io/en/latest/
- ADIOS source code: https://github.com/ornladios/ADIOS2
  - Written in C++, wrappers for Fortran, Python, Matlab, C
  - Contains command-line utilities (bpls, adios_reorganize ..)
- Online help:
  - ADIOS2 GitHub Issues: https://github.com/ornladios/ADIOS2/issues

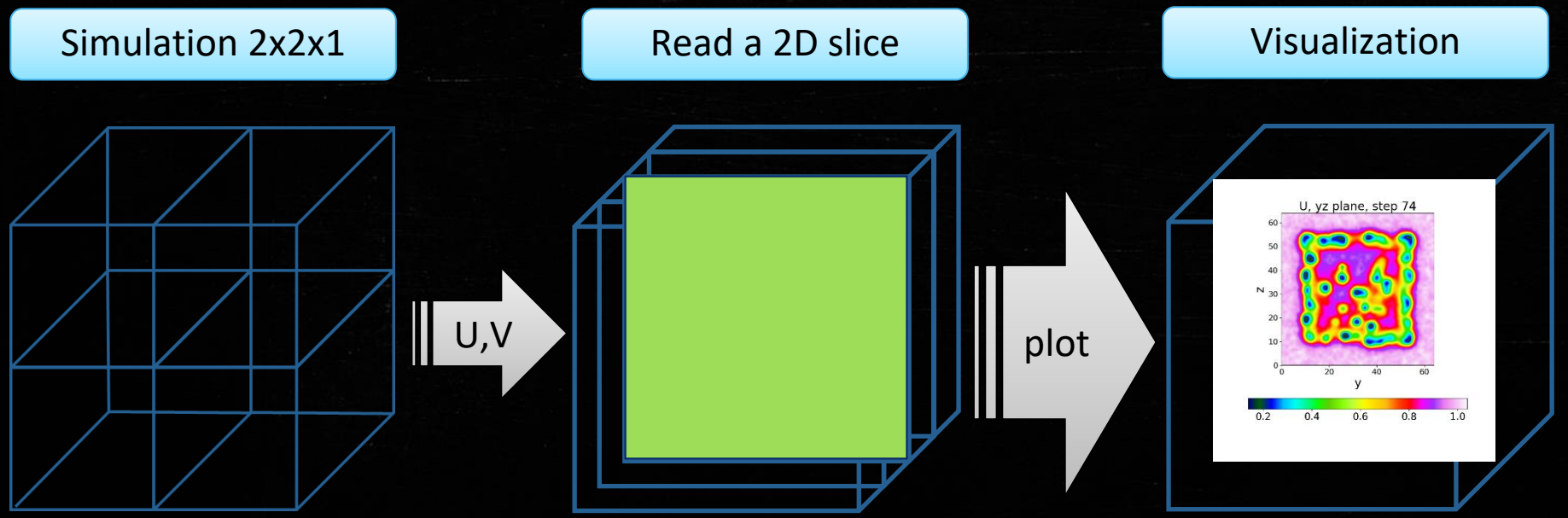
Gray-Scott Example

- 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



Visualization with plot/gsplot.py

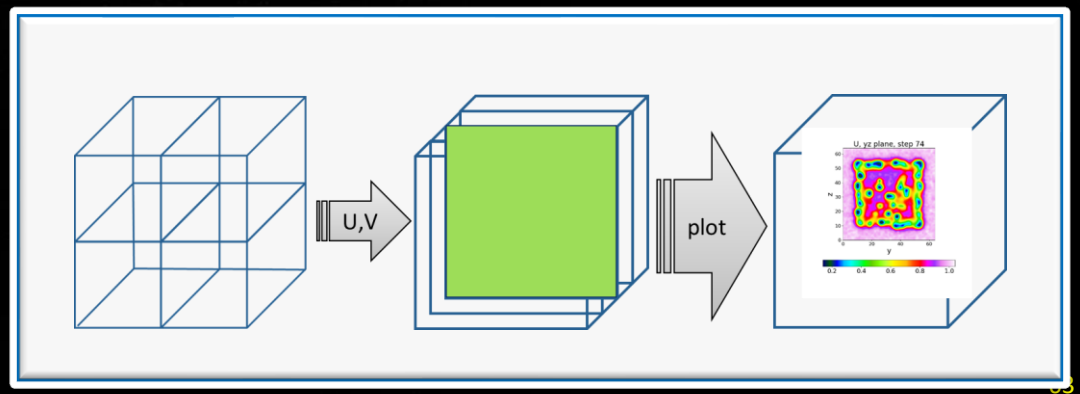
- Read a 2D slice and plot it with matplotlib



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

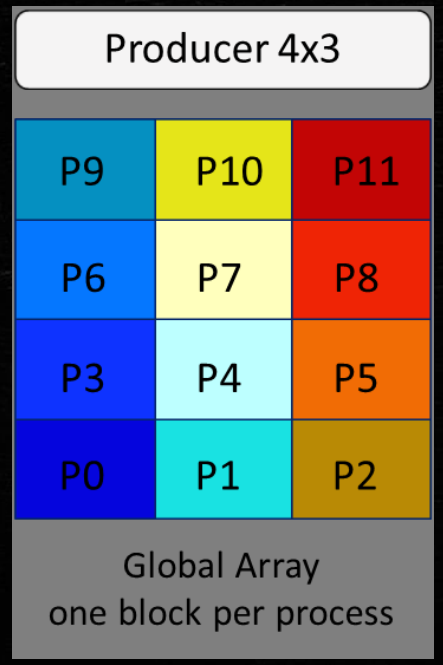
```
$ mpirun -n 4 ./gray-scott settings-staging.json
Simulation at step 10 writing output step     1
Simulation at step 20 writing output step     2
Simulation at step 30 writing output step     3
Simulation at step 40 writing output step     4
...
```

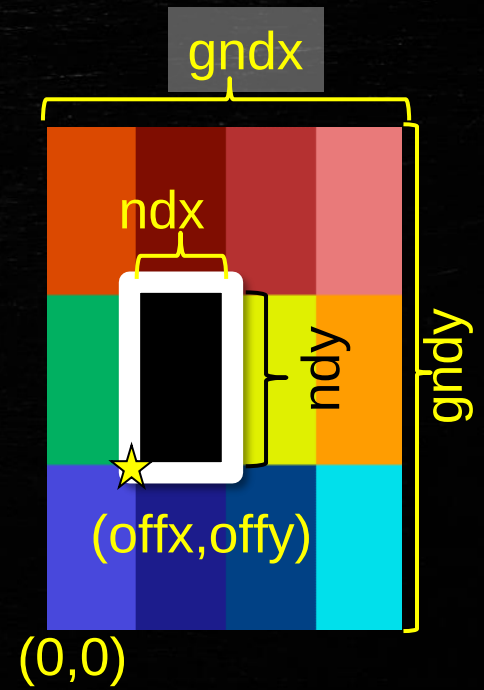
```
$ mpirun -n 1 python3 gsplot.py -i gs.bp
GS Plot step 0 processing stream step 0 sim step 10 minmax=0.0765537..1.05494
GS Plot step 1 processing stream step 1 sim step 20 minmax=0.111669..1.05908
...
```



Global Array: data produced by multiple processes

- N-dimensional array
  - **Shape**
- Has a type (int32, double, etc.)
  - **Type**
- Blocks of data are written into the array
  - **Start** (offset)
  - **Count** (size of block)





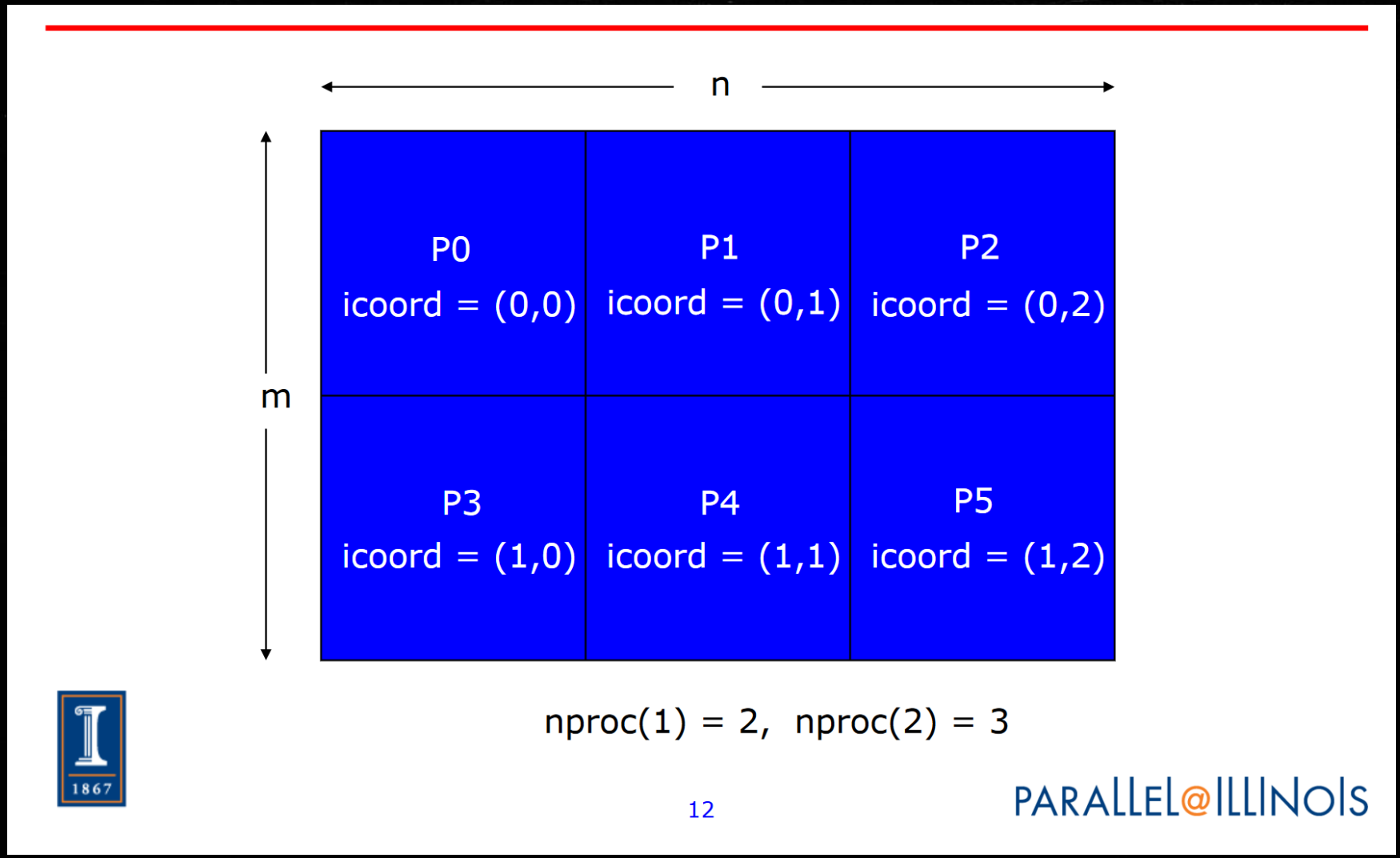
Shape = {gndx, gndy}

Start = {offx, offy}

Count = {ndx, ndy}

MPI-IO version IO with ghost cells

- https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture33.pdf



https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture33.pdf

CASE I: LOCAL ARRAY CONTIGUOUS IN MEMORY

```
iarray_of_sizes(1) = m
iarray_of_sizes(2) = n
iarray_of_subsizes(1) = m/nproc(1)
iarray_of_subsizes(2) = n/nproc(2)
iarray_of_starts(1) = icoord(1) * iarray_of_subsizes(1)     ! 0-based, even in Fortran
iarray_of_starts(2) = icoord(2) * iarray_of_subsizes(2)     ! 0-based, even in Fortran
ndims = 2
zeroOffset = 0

call MPI_TYPE_CREATE_SUBARRAY(ndims, iarray_of_sizes, iarray_of_subsizes,
        iarray_of_starts, MPI_ORDER_FORTRAN, MPI_REAL, ifiletype, ierr)
call MPI_TYPE_COMMIT(ifiletype, ierr)

call MPI_FILE_OPEN(MPI_COMM_WORLD, '/home/me/test',
        MPI_MODE_CREATE + MPI_MODE_RDWR, MPI_INFO_NULL, ifh, ierr)

call MPI_FILE_SET_VIEW(ifh, zeroOffset, MPI_REAL, ifiletype, 'native', MPI_INFO_NULL, ierr)

ilocal_size = iarray_of_subsizes(1) * iarray_of_subsizes(2)
call MPI_FILE_WRITE_ALL(ifh, local_array, ilocal_size, MPI_REAL, istatus, ierr)

call MPI_FILE_CLOSE(ifh, ierr)
```

Gray-Scott example MPI-IO version

hands-on instructions

gs-mpiio

Gray-Scott MPI-IO example (CSD3)

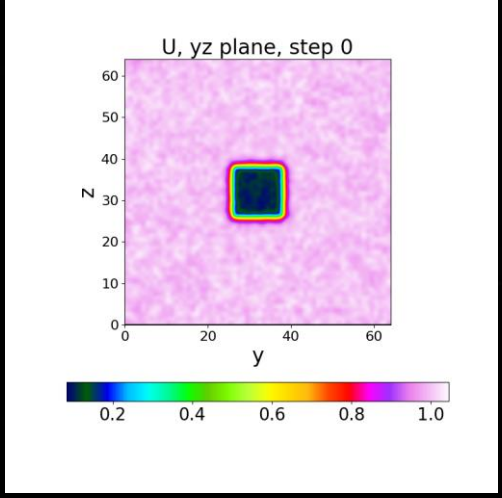
```
$ source ~/rds/rds-eied-workshop/shared/setup.sh
$ cd ~/tutorial/gs-mpiio
$ mkdir build
$ cd build
$ cmake -DCMAKE_PREFIX_PATH="$ADIOS2_DIR" \
-DCMAKE_INSTALL_PREFIX=${HOME}/rds/hpc-work/gs-mpiio ..
$ make
$ make install
```

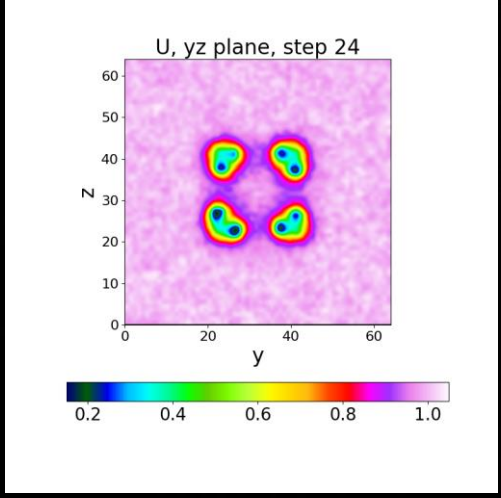
```
$ cd ~/rds/hpc-work/gs-mpiio
$ sintr_0 -p icelake -N 1 -n 76 -A TRAINING-CPU -t 04:00:00
```

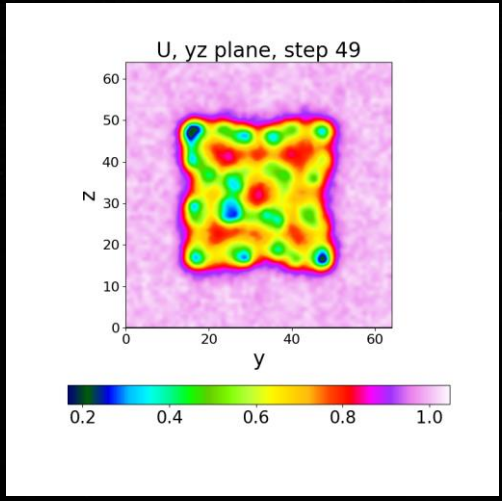
```
$ source ~/rds/rds-eied-workshop/shared/setup.sh
$ mpirun -n 16 ./gray-scott settings-files.json
$ ls -l data.u
$ mpirun -n 4 ./pdf-calc data.u data 100
$ ls -l data.*
```

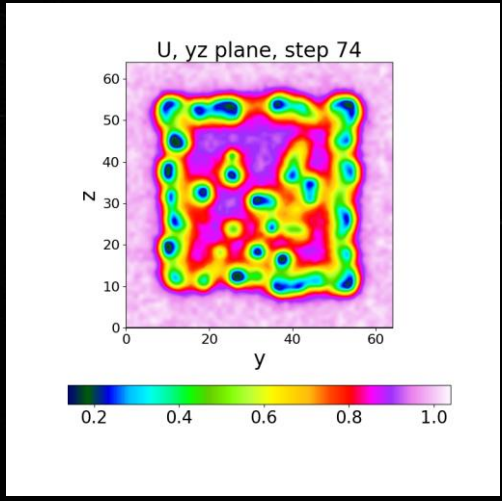
Gray-Scott MPI-IO example (CSD3)

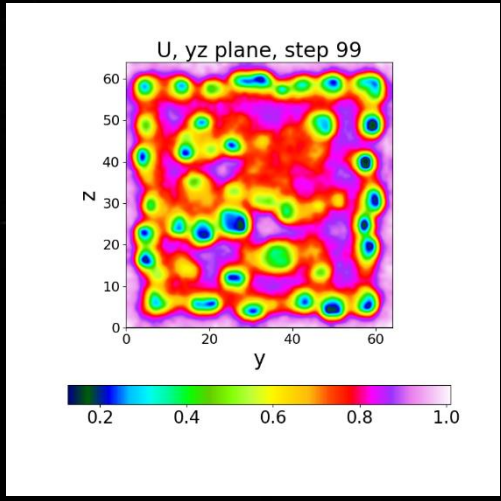
```
$ mpirun -n 1 python3 ./gsplot.py -i data.u -o u
$ ls -l u*.png
$ display u00099_yz.png &
```











Gray-Scott example
ADIOS2 version

step by step instructions to modify
gs-mpiio/simulation/
to an adios2 C++ writer

Preparation

```
$ source ~/rds/rds-eied-workshop/shared/setup.sh
$ cd ~/tutorial
$ cp -rp gs-mpiio mycode
$ cd mycode
$ rm -rf build
$ mkdir build
$ cd build
$ cmake -DCMAKE_PREFIX_PATH="$ADIOS2_DIR" \
-DCMAKE_INSTALL_PREFIX=${HOME}/rds/hpc-work/mycode  ..
$ make -j 4
```

```
$ cd ..
$ grep -r "#IO#"
simulation/writer.h:// #IO# include IO library
simulation/writer.h:    //#IO# declare ADIOS variables for engine, io, variables
simulation/writer.cpp:    // #IO# make initializations, declarations as necessary
simulation/main.cpp:// #IO# include IO library
simulation/main.cpp:    // #IO# Need to initialize IO library
CMakeLists.txt:# #IO# look for ADIOS2
CMakeLists.txt:# #IO# add link dependency adios2::cxx11_mpi
CMakeLists.txt:# #IO# add link dependency adios2::cxx11_mpi
```

Build changes (this is already done)

- We need to add ADIOS2 to the cmake build instructions
- cmake can find packages that were built with cmake if they are in the LD_LIBRARY_PATH or if told of the location
  - find_package(MPI REQUIRED)
  - find_package(ADIOS2 REQUIRED)
- Executables need to specify their dependency
  - target_link_libraries(gray-scott adios2::cxx11_mpi MPI::MPI_C)

$HOME/tutorial/mycode

```
$ cat CMakeLists.txt
cmake_minimum_required(VERSION 3.10)

set(CMAKE_CXX_STANDARD 11)
if(POLICY CMP0074)
  cmake_policy(SET CMP0074 NEW)
endif()

project(gray-scott C CXX)

find_package(MPI REQUIRED)
find_package(ADIOS2 REQUIRED)

# We are not using the C++ API of MPI, this will stop the compiler look for it
add_definitions(-DOMPI_SKIP_MPICXX -DMPICH_SKIP_MPICXX)

add_executable(gray-scott
  simulation/main.cpp
  simulation/gray-scott.cpp
  simulation/settings.cpp
  simulation/writer.cpp
)
target_link_libraries(gray-scott adios2::cxx11_mpi MPI::MPI_C)

add_executable(pdf-calc analysis/pdf-calc.cpp)
target_link_libraries(pdf-calc adios2::cxx11_mpi MPI::MPI_C)
```

simulation/main.cpp

- #include <adios2.h>
- Create an ADIOS object
- Create an IO object
- Build frequently to check for errors

$HOME/tutorial/mycode

```
$ vi simulation/main.cpp
```

```
// #IO# include IO library
#include <adios2.h>
#include <mpi.h>
```

```
GrayScott sim(settings, comm);
 sim.init();

 // #IO# Need to initialize IO library
 adios2::ADIOS adios("adios2.xml", comm);
 adios2::IO io = adios.DeclareIO("SimulationOutput");

 Writer writer(settings, sim, comm, io);
```

```
$ (cd build; make)
main.cpp:70:42: error: no matching function for call to
'Writer::Writer(Settings&, GrayScott&, MPI_Comm&, adios2::IO&)'
  70 |     Writer writer(settings, sim, comm, io);
```

simulation/writer.h

- #include <adios2.h>
- Add the IO object to the argument list of Writer constructor
- Declare ADIOS2 objects used in this class

$HOME/tutorial/mycode

```
$ vi simulation/writer.h
```

```
// #IO# include IO library
#include <adios2.h>
#include <mpi.h>
```

```
class Writer
{
public:
    Writer(const Settings &settings, const GrayScott &sim,
           const MPI_Comm comm, adios2::IO &io);
```

```
$ (cd build; make)
writer.cpp:17:1: error: no declaration matches 'Writer::Writer(const
Settings&, const GrayScott&, MPI_Comm)'
  17 | Writer::Writer(const Settings &settings, const GrayScott &sim,
     | ^~~~~~
```

```
    //#IO# declare ADIOS variables for engine, io, variables
    adios2::IO io;
    adios2::Engine writer;
    adios2::Variable<int> varStep;
    adios2::Variable<double> varU;

};
```

simulation/writer.cpp 1/3

- Add the IO object to the argument list of Writer constructor
- Define Variables and Memory Selections in constructor

$HOME/tutorial/mycode

```
$ vi simulation/writer.cpp
```

```
Writer::Writer(const Settings &settings, const GrayScott &sim,
               const MPI_Comm comm, adios2::IO &io)
: settings(settings), comm(comm), io(io)
{
```

```
    varStep = io.DefineVariable<int>("step");
    varU = 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}
    );
};
```

```
$ (cd build; make)
[100%] Built target gray-scott
```

simulation/writer.cpp 2/3

- Writer::open() function
  - Open adios2 output
- Writer::close() function
  - Close Engine object (the output)
- print_settings()
  - io.EngineType() string identifies the chosen and activated Engine for output (for debugging)

$HOME/tutorial/mycode

```
$ vi simulation/writer.cpp
```

```
void Writer::open(const std::string &fname)
{
    writer = io.Open(fname, adios2::Mode::Write);
}
```

```
void Writer::close()
{
    writer.Close();
}
```

```
void Writer::print_settings()
{
    std::cout << "Simulation writes data using engine type:              "
              << io.EngineType() << std::endl;
}
```

```
$ (cd build; make)
[100%] Built target gray-scott
```

simulation/writer.cpp 3/3

- Writer::write() function
- Declare an output step and pass the data pointers to the Engine
  - BeginStep
    - Put() for each variable deemed for output
    - note: don't need to write all defined variable
  - EndStep

$HOME/tutorial/mycode

```
$ vi simulation/writer.cpp
```

```
void Writer::write(int step, const GrayScott &sim)
{
   /* sim.u_ghost() provides access to the U variable as is */
   /* sim.u_noghost() provides a contiguous copy without the ghost cells */
   const std::vector<double> &u = sim.u_noghost();
   const std::vector<double> &v = sim.v_noghost();

   writer.BeginStep();
   writer.Put(varStep, step);
   writer.Put(varU, u.data());
   writer.EndStep();
}
```

```
$ (cd build; make)
[100%] Built target gray-scott
```

simulation/settings-files.json

$HOME/tutorial/mycode

- change output name from data to some .bp name
  - it can be anything though

```
$ vi simulation/settings-files.json
```

```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.02,
  "k": 0.048,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 1000,
  "noise": 0.01,
  "output": "gs.bp",
  "adios_config": "adios2.xml"
}
```

```
$ (cd build; make install)
```

Gray-Scott ADIOS2 example (CSD3)

```
$ (cd build; make install)
$ cd ~/rds/hpc-work/mycode
```

```
$ mpirun -n 16 ./gray-scott settings-files.json
Simulation writes data using engine type:              BP5
=========================================
grid:               64x64x64
steps:              1000
plotgap:            10
F:                  0.02
k:                  0.048
dt:                 2
Du:                 0.2
Dv:                 0.1
noise:              0.01
output:             gs.bp
adios_config:       adios2.xml
process layout:     4x2x2
local grid size:    16x32x32
=========================================
Simulation at step 10 publishing output step     1
…
Simulation at step 1000 publishing output step     100
```

Gray-Scott ADIOS2 example (CSD3)

```
$ ls -l gs.bp
total 205260
-rw-rw-r-- 1 hpcpodh1 hpcpodh1 209715200 Jul 10 14:07 data.0
-rw-rw-r-- 1 hpcpodh1 hpcpodh1    437536 Jul 10 14:07 md.0
-rw-rw-r-- 1 hpcpodh1 hpcpodh1     16325 Jul 10 14:07 md.idx
-rw-rw-r-- 1 hpcpodh1 hpcpodh1      1360 Jul 10 14:07 mmd.0
-rw-rw-r-- 1 hpcpodh1 hpcpodh1      5028 Jul 10 14:07 profiling.json
```

```
$ bpls -la gs.bp
  double   F     attr   = 0.01
  double   U     100*{64, 64, 64} = 0.0606895 / 1.06644
  int32_t  step  100*scalar = 10 / 1000
$ bpls -la gs.bp -d step -n 20
  int32_t  step  100*scalar = 10 / 1000
    ( 0)    10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
    (20)    210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400
    (40)    410 420 430 440 450 460 470 480 490 500 510 520 530 540 550 560 570 580 590 600
    (60)    610 620 630 640 650 660 670 680 690 700 710 720 730 740 750 760 770 780 790 800
    (80)    810 820 830 840 850 860 870 880 890 900 910 920 930 940 950 960 970 980 990 1000
```

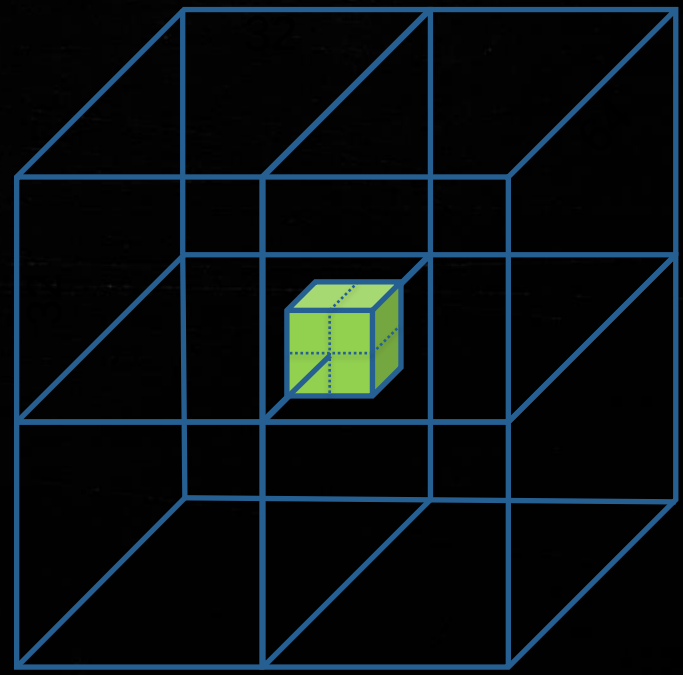
bpls (to show the decomposition of the array)

```
$ bpls -l -D gs.bp U
 double   U     100*{64, 64, 64} = 0.0907898 / 1
        step  0:
          block 0: [ 0:63,  0:31,  0:31] = 0.104002 / 1
          block 1: [ 0:63, 32:63,  0:31] = 0.104002 / 1
          block 2: [ 0:63,  0:31, 32:63] = 0.104002 / 1
          block 3: [ 0:63, 32:63, 32:63] = 0.104002 / 1
 ...
        step 99:
          block 0: [ 0:63,  0:31,  0:31] = 0.148308 / 0.998811
          block 1: [ 0:63, 32:63,  0:31] = 0.148302 / 0.998812
          block 2: [ 0:63,  0:31, 32:63] = 0.148335 / 0.998811
          block 3: [ 0:63, 32:63, 32:63] = 0.148302 / 0.998811
```

bpls to dump: 2x2x2 read with bpls

- Use bpls to read in the **center** 3D cube of the **last** output step

```
$ bpls gs.bp -d U -s "-1,31,31,31" -c "1,2,2,2" -n 2
double   U      100*{64, 64, 64}
   slice (99:99, 31:32, 31:32, 31:32)
    (99,31,31,31)    0.916973 0.916972
    (99,31,32,31)    0.916977 0.916975
    (99,32,31,31)    0.916974 0.916973
    (99,32,32,31)    0.916977 0.916976
```



- Note: bpls handles time as an extra dimension
- -s starting offset
  - first offset is the timestep, *-n to* count backwards
- -c size in each dimension
  - first value is how many steps
- -n how many values to print in one line

Gray-Scott example
ADIOS2 version

step by step instructions to modify
gs-mpiio/plot/gsplot.py
to an adios2 Python reader

Environment changes (this is already done)

- PYTHONPATH needs to include the path to the adios2 python library
  - library was built and installed with adios2
  - sourced env.sh set this path
- Make sure that you use the same version of python that was used to build adios2
- Example needs python3 that has
  - numpy
  - matplotlib
  - mpi4py (by gs-adios2/plot)

$HOME/tutorial/mycode

```
$ echo $PYTHONPATH
/rds/project/hpc/rds-hpc-training/rds-eied-workshop/shared/icl/intel/adios2/2.9.0/lib64/python3.6/site-packages:

$ which python3
/rds/project/hpc/rds-hpc-training/rds-eied-workshop/shared/adios-env/intel/bin/python3

$ python3
Python 3.6.8 (default, Feb 21 2023, 16:57:46)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-16)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> import matplotlib
>>> import mpi4py
>>> import adios2
```

plot/gsplot 1/4

- import adios2
- Create ADIOS/IO objects
- Open input with IO object

$HOME/tutorial/mycode

```
$ cd $HOME/tutorial/mycode
$ vi plot/gsplot.py
```

```
import numpy as np
import adios2
import matplotlib.pyplot as plt
```

```
# Read the data from this object
 print("open {0}...".format(args.instream))
 fr = open(args.instream, "rb")
 fileSize = fstat(fr.fileno()).st_size
 header = fr.read(headersize)
 shape3 = np.frombuffer(header, dtype=np.uint64, count=3, offset=0)
 print("size of U = {0}x{1}x{2}".format(shape3[0], shape3[1], shape3[2]))

 nelems = (int)(shape3[0]*shape3[1]*shape3[2])
 varsize = nelems*8
 nsteps_d = (fileSize-headersize)/varsize
 nsteps = (int)(nsteps_d)
 if ((np.float64)(nsteps) != (np.float64)(nsteps_d)):
    print("ERROR: file size {0} is unexpected".format(fileSize))
 print("found {0} steps".format(nsteps))
```

Remove all this code

```
adios = adios2.ADIOS("adios2.xml")
io = adios.DeclareIO("SimulationOutput")
fr = io.Open(args.instream, adios2.Mode.Read)
```

plot/gsplot 2/4

- Change for loop structure to an adios2 BeginStep..EndStep loop

$HOME/tutorial/mycode

$ vi plot/gsplot.py

```
    # Read through the steps, one at a time
    for step in range(0, nsteps):
        print("GS Plot step {0}".format(step), flush=True)
…
    fr.close()
```

```
    # Read through the steps, one at a time
    plot_step = 0
    while(True):
        status = fr.BeginStep()
        if status != adios2.StepStatus.OK:
            break

…  see next slide …

        fr.EndStep()
        plot_step = plot_step + 1

    fr.Close()
```

plot/gsplot 3/4

$HOME/tutorial/mycode

$ vi plot/gsplot.py

- Process one step 1/2
  - fr.CurrentStep() returns step in incoming stream
  - io.AvailableVariables() returns a dictionary of all variables in the step
  - Using Min/Max is an easy way to get the values of single-value variables (here "step")

```
if status != adios2.StepStatus.OK:
    break

cur_step = fr.CurrentStep()
vars = io.AvailableVariables()
sim_step = int(vars["step"]["Min"])
print("GS Plot step {0} processing stream step {1} sim step {2} minmax={3}..{4}".format(
    plot_step, cur_step, sim_step, float(vars["U"]["Min"]), float(vars["U"]["Max"])), flush=True)
```

plot/gsplot 4/4

$HOME/tutorial/mycode

$ vi plot/gsplot.py

- Process one step 2/2
  - Numpy arrays required for reading
  - io.InquireVariable() returns a variable object that can be used for reading

```
start_offset = headersize + step*varsize
u = np.fromfile(fr, dtype=np.float64,
            count=nelems).reshape(
   shape3[0], shape3[1], shape3[2])


...
if args.plane in ('yz', 'all'):
   data = u[int(shape3[0]/2), :, :]
   Plot2D('yz',  data, args, shape3, step, fontsize))
```

Just focus on the default YZ plane

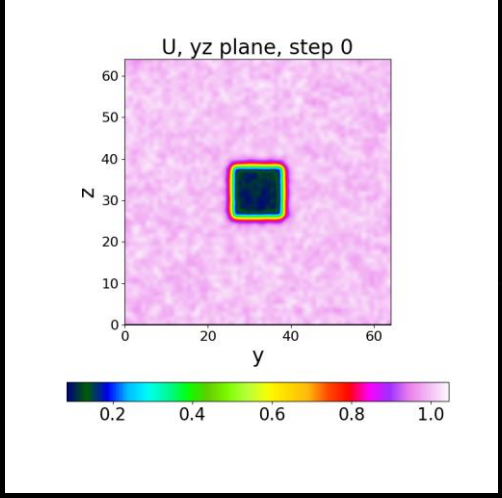
```
vu = io.InquireVariable("U")
shape3 = vu.Shape()
```

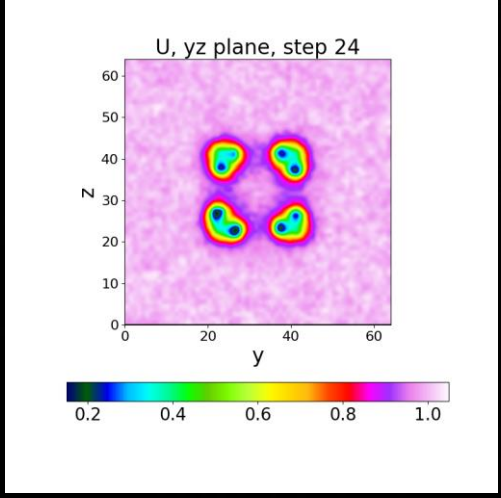
```
if args.plane in ('yz', 'all'):
   data = np.empty([shape3[1], shape3[2]], dtype=np.double)
   vu.SetSelection( [  [int(shape3[0]/2), 0, 0],
               [1, shape3[1], shape3[2]] ])
   fr.Get(vu, data, adios2.Mode.Sync)
   Plot2D('yz',  data, args, shape3, sim_step, fontsize)
```

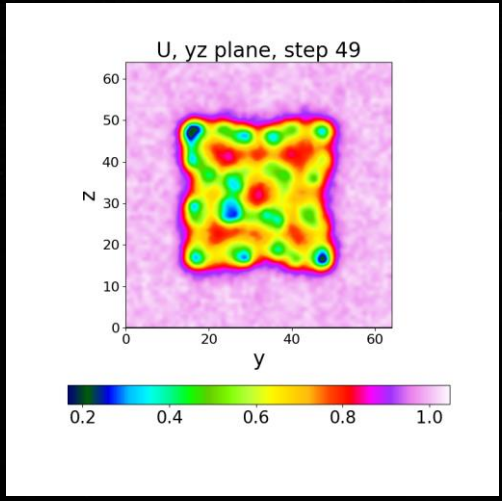
Home work: xy and xz planes. Plots will look the same as the yz plots because of the symmetry in the simulation

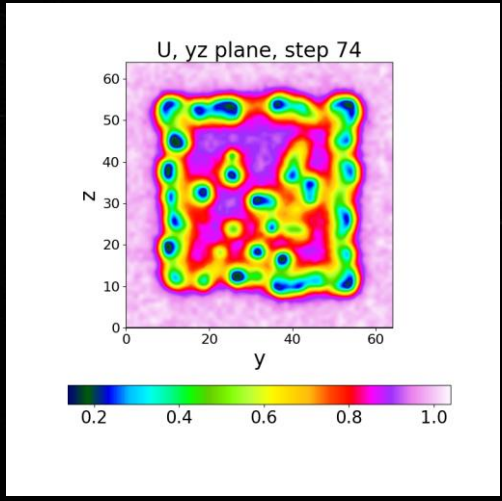
Gray-Scott MPI-IO example (CSD3)

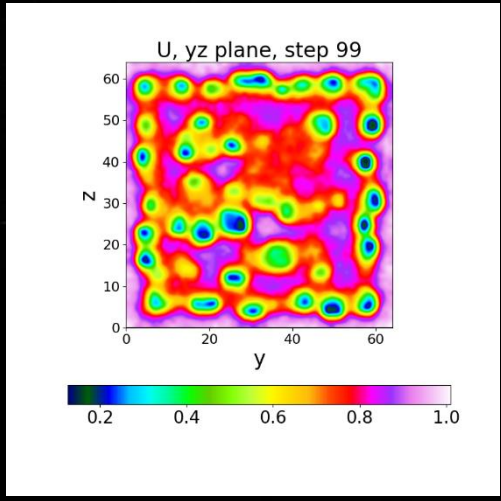
```
$ (cd build; make install)
$ cd ~/rds/hpc-work/mycode
$ mpirun -n 1   python3 ./gsplot.py -i gs.bp -o u
$ ls -l u*.png
$ display u00099_yz.png
```











Outline

- ADIOS 2 concepts and API (C++, Python, F90)
- BREAK
- Hands on with ADIOS 2 – Files
- QUICK BREAK
- ADIOS @ scale
- Staging with ADIOS
- Hands on with ADIOS 2 – Staging
- END

How to scale ADIOS I/O

ADIOS Scaling for large parallel file systems

- There are a **two basic** options for changing the performance of ADIOS for your code on all HPC systems
  - Aggregation – choosing the number of sub-files for maximizing the filesystem bandwidth
  - Appending multiple steps into a single output for minimizing the filesystem metadata overhead

ADIOS Scaling for large parallel file systems: number of files

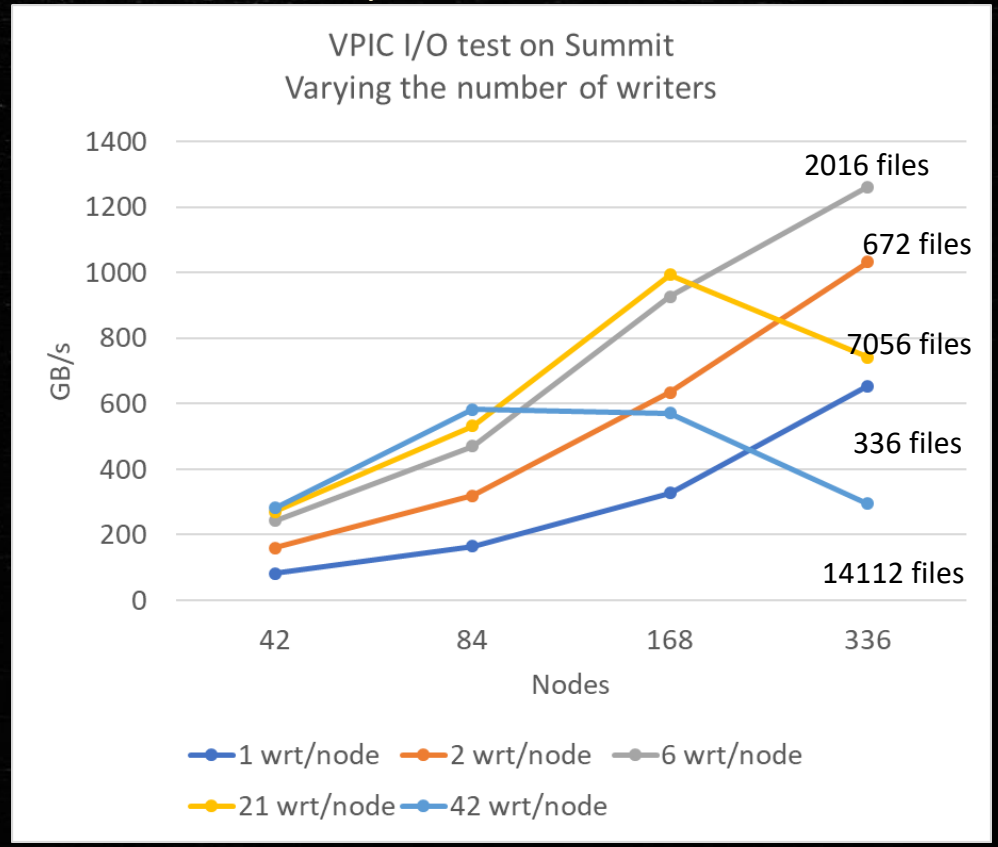
- Not good:
  - Single, global output file from many writers (or for many readers)
    - Bottleneck at file access
  - One file per process
    - Chokes the metadata server of the file system at create
    - Reading from different number of processes is very difficult
- Good:
  - Create K subfiles where K is proportioned to the capability of the file system, not the number of writers/readers
- ADIOS BP engine options
  - **NumAggregators**
  - **AggregatorRatio**
- ADIOS default settings
  - One file per compute node

```
<io name="SimulationOutput">
  <engine type="BP5">
    <parameter key="NumAggregators" value="256"/>
  </engine>
</io>
```

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 start but apps usually have one MPI task/GPU, so keep the total number of subfiles below 4000

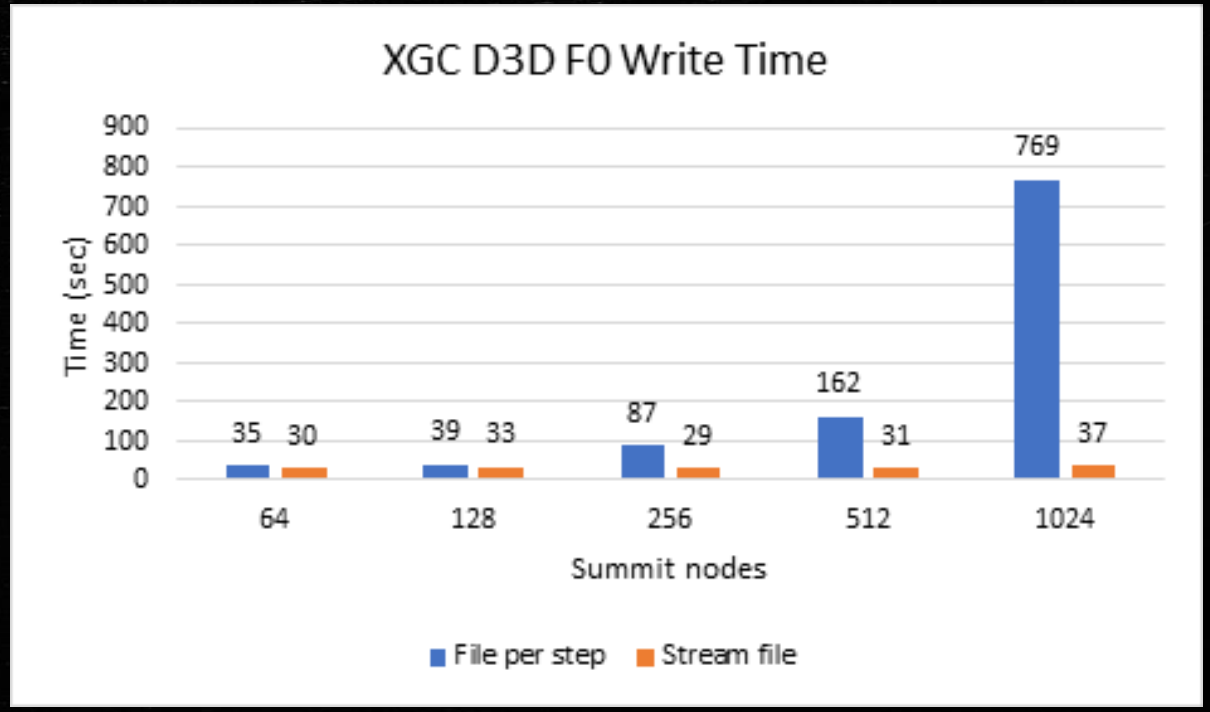
chart courtesy of Junmin Gu, LBNL



Application	Nodes/GPUs, 6 tasks/node	Data Size per step	I/O speed	ADIOS NumAggregators
SPECFEM3D_GLOBE	3200/19200	250 TB	~2 TB/sec	3200 (1:6 aggregation ratio)
GTC	512/3072	2.6 TB	~2 TB/sec	3072 (1:1 aggregation ratio)
XGC	512/3072	64 TB	1.2 TB/sec	1024 (1:3 aggregation ratio)
LAMMPS	512/3072	457 GB	1 TB/sec	512 (1:6 aggregation ratio)

ADIOS Scaling for large parallel file systems: number of steps

- Another aspect of number of files: number of output steps
- New output every step -> many files over the course of simulation
  - If the rate of steps stresses the file system, write performance will drop
  - Actually, the total time of creation of files will add up
- Appending multiple output steps into same file is better



XGC 100 output steps in 1245 second simulation, 40 GB per step (4TB total)

One file per node created in each step

Single output for all steps (one file per node)

Outline

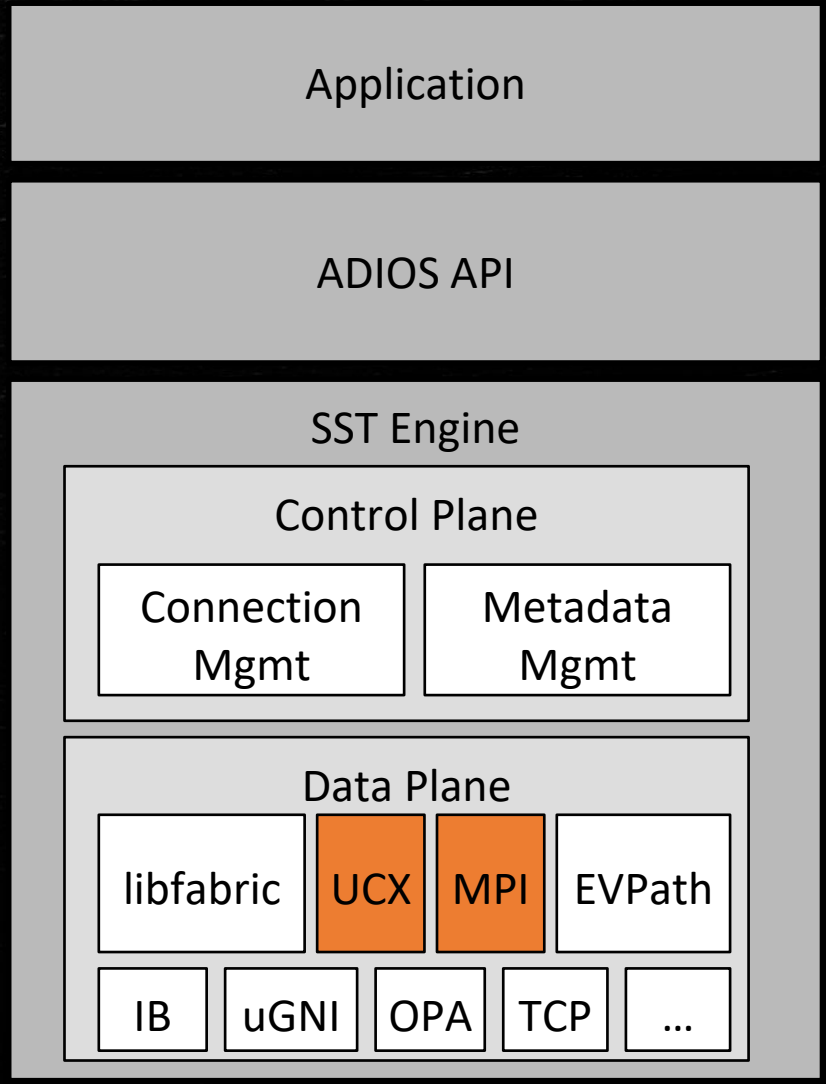
- ADIOS 2 concepts and API (C++, Python, F90)
- BREAK
- Hands on with ADIOS 2 – Files
- QUICK BREAK
- ADIOS @ scale
- Staging with ADIOS
- Hands on with ADIOS 2 – Staging
- END

Data Staging in ADIOS

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

Sustainable Staging Transport (SST)

- Direct connection 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
  - Uses EVPath library from Georgia Tech
  - Allows for dynamic connections, multiple readers and complex flow control management
- Data Plane
  - Exchange data using RDMA, or sockets, or MPI
  - Responsible for resource management for data transfer
  - Uses libfabric/ucx for portable RDMA support or MPI
  - Threaded to overlap communication with computation and for asynchronous progress monitoring



Selecting the data plane for SST

Enable threaded MPI in the applications

```
int provided;
MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
```

XML:

```
<io name="SimulationOutput">
  <engine type="SST">
    <parameter key="RendezvousReaderCount" value="0"/>
    <parameter key="QueueLimit" value="1"/>
    <parameter key="QueueFullPolicy" value="Discard"/>
    <parameter key="OpenTimeoutSecs" value="60.0"/>
    <parameter key="DataTransport" value="MPI"/>
  </engine>
</io>
```

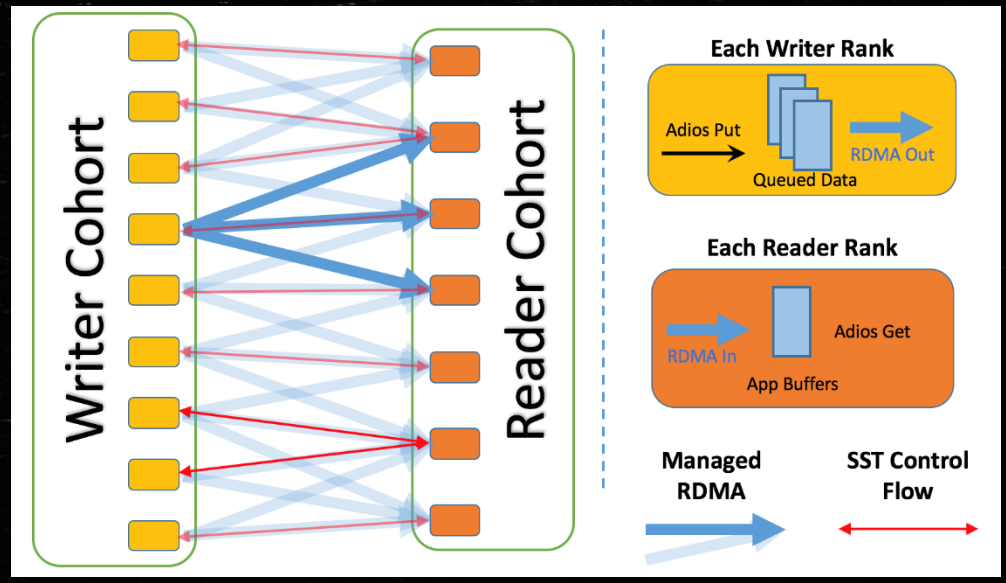
Options: MPI, UCX, WAN, fabric

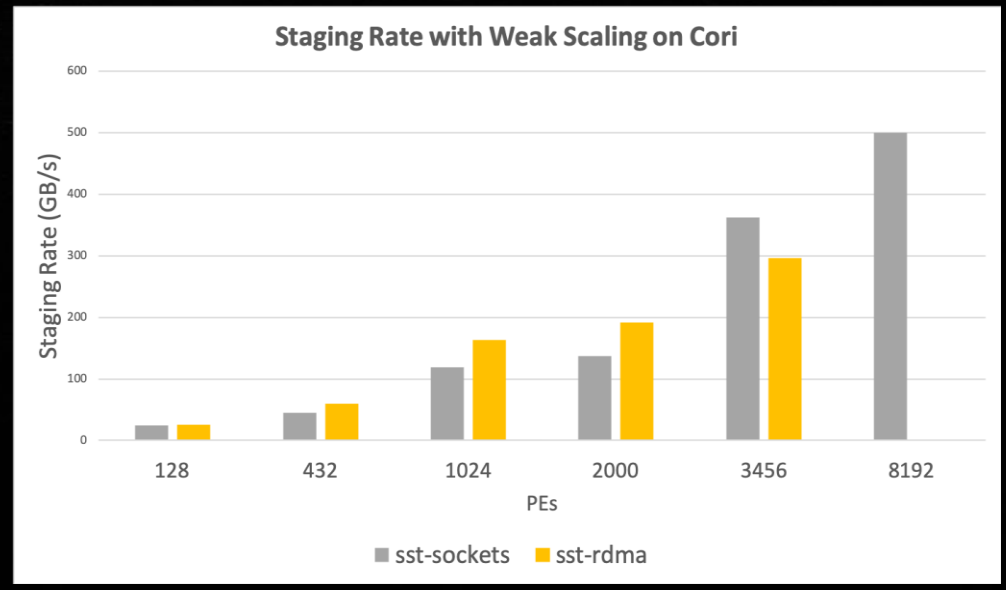
Source code:

```
adios2::ADIOS ad = adios2::ADIOS(configfile, comm);
adios2::IO io = ad.DeclareIO("SimulationOutput");
io.SetParameter("DataTransport", "MPI");
```

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).

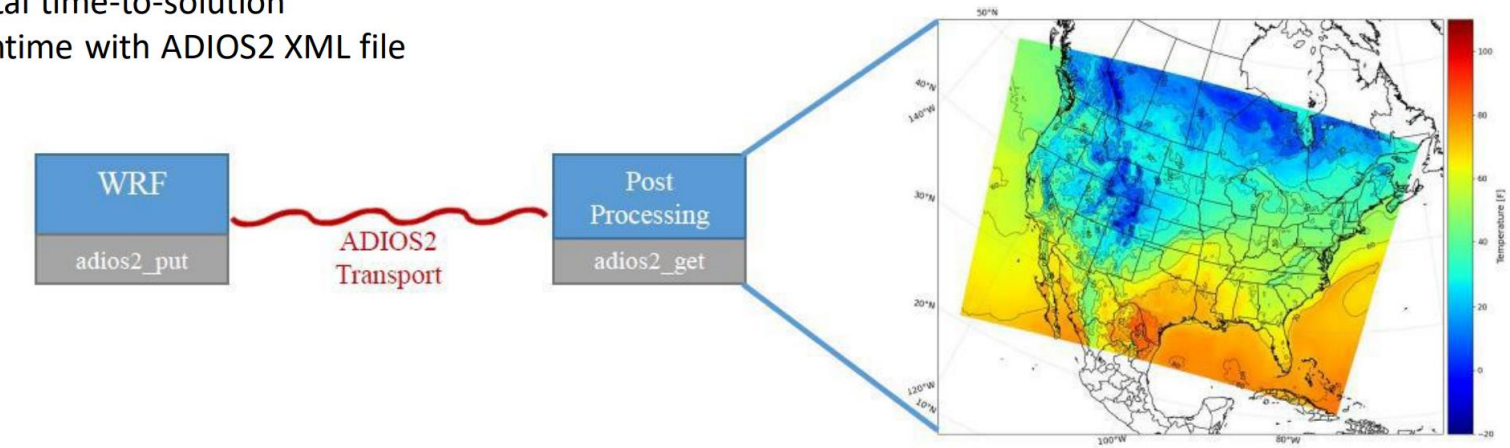


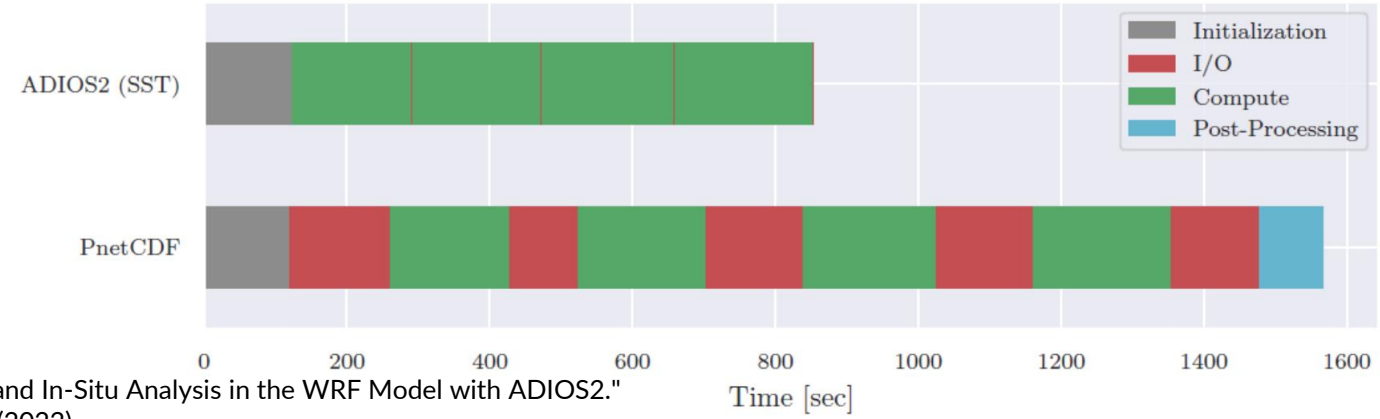


WRF-ADIOS2 In-Situ Analysis

M. Laufer, E. Fredj (2022)

- Proof-of-concept post processing analysis pipeline built using ADIOS2 high-level Python API
- Data is post-processed as soon as it is available over the network (not filesystem)
- Greatly reduces total time-to-solution
- Configurable at runtime with ADIOS2 XML file





Laufer, Michael, and Erick Fredj.
"High Performance Parallel I/O and In-Situ Analysis in the WRF Model with ADIOS2."
arXiv preprint arXiv:2201.08228 (2022).

Erick Fredj@2022
e-mail: fredj@jct.ac.il

Slide courtesy of:
Prof. Erick Fredj
LEV Academic Center, Israel
Rutgers University, New Jersey

SSC (Staging for Strong Coupling)

- An ADIOS 2 engine using MPI for portability and performance

- Features
  - Use a combination of one-sided MPI and two-sided MPI methods for flexibility and performance.
  - Use threads and non-blocking MPI methods to hide communication time.
  - Optimized for fixed I/O pattern – push data to consumer

- Target Applications
  - XGC, Gene, or Gem (Edge-core strong coupling)
  - Other ECP applications requiring strong coupling or always-on in-situ analysis/visualization

DataMan

- An ADIOS 2 engine subroutine designed for wide area network data transfer, data staging, near-real-time data reduction and remote in-situ processing
- Designed with following principles:
  - Flexibility: allowing transport layer switching (ADIOS BP file, ZeroMQ, google-rpc etc.)
  - Versatility: supporting various workflow modes (p2p, query, pub/sub, etc.)
  - Adaptability: allowing adaptive data compression based on near-real-time decision making
  - Extendibility: taking advantage of all ADIOS transports and operators, and other potential third-party libraries. For example, DataMan can use ZFP, Bzip, SZ that have been built into ADIOS, as well as any compression techniques that will be built into ADIOS in future.
- Features
  - Step aggregation to improve data rate, by sacrificing latency.
  - Lossy compression to reduce data size to be transferred, by sacrificing latency and precision.
  - Fast mode to improve latency and data rate, by sacrificing reliability.
  - Combinations of features above to achieve a certain set of performance requirements.
- Target Applications:
  - ECP applications requiring wide area network data transfer and adaptive data reduction
  - Square Kilometer Array observational data
  - KSTAR fusion experimental data

Application: Which staging method to use?

- System considerations
  - Staging between machines/over a WAN? **DataMan**
  - Co-located execution? **SST**
  - Availability of RDMA high-speed network? **SST** optimized for this case.
- Coupling considerations
  - Highly synchronized writer/reader? **SSC**
  - 1 writer, many readers streaming? **SST** optimized for this use case.
  - 1 reader, many small producers (e.g. AI training)? **SST**
- Performance considerations
  - Often application-specific, and not always obvious.
  - Here's where it helps to have a flexible I/O framework…

Inline engine

- Consumer runs inside the Producer code
- Consumer code has direct access to the Producer's data (zero-copy)
- No Global access (cross-communication) so must use Local Arrays only

Usage:

- Create two engines from one io object
  - Write mode for Producer
  - Read mode for Consumer
- Execute Consumer code right after Producer's EndStep()

examples/inlineMWE

```
unsigned long N = 256;
adios2::ADIOS adios;
adios2::IO io = adios.DeclareIO("InlineReadWrite");
io.SetEngine("Inline");
auto u = io.DefineVariable<double>("u", {}, {}, {N}, adios2::ConstantDims);
adios2::Engine writer = io.Open("writer", adios2::Mode::Write);
adios2::Engine reader = io.Open("reader", adios2::Mode::Read);
for (int64_t timeStep = 0; timeStep < 2; ++timeStep)
{
    writer.BeginStep();
    std::vector<double> v(N, 3.2);
    std::cout << "Putting data at address   " << v.data()
              << " into inline writer.\n";
    writer.Put(u, v.data());
    writer.EndStep();

    reader.BeginStep();
    double *data = nullptr;
    reader.Get(u, &data);
    std::cout << "Getting data from address " << data
              << " via inline reader\n";
    reader.EndStep();
}
```

```
$ ./bin/inlineMWE
Putting data at address   0x56444464cab0 into inline writer.
Getting data from address 0x56444464cab0 via inline reader
Putting data at address   0x56444464cab0 into inline writer.
Getting data from address 0x56444464cab0 via inline reader
```

ParaView In Situ Engine plugin

- ADIOS plugin that uses the inline engine internally, along with the Catalyst API to enable in situ visualization with ParaView

- Features
  - This plugin sets up the inline writer and reader on the same IO object for you
  - No code changes are now necessary to use the inline engine
  - The engine is instrumented with the Catalyst API calls – no need to add this to your application directly

Fides and ParaView Catalyst

- Need to set 3 environment variables:
  - ADIOS2_PLUGIN_PATH: set this to point to the directory containing libParaViewADIOSInSituEngine.so
  - CATALYST_IMPLEMENTATION_NAME=paraview
  - CATALYST_IMPLEMENTATION_PATHS: set this to point to the directory containing the ParaView Catalyst build. Most likely something like: /path-to-paraview-build/lib/catalyst
- Setting the Catalyst environment variables will swap in the ParaView Catalyst implementation at runtime so you can use the full

Configuring the in situ engine plugin

- Set engine type to plugin
- PluginName is a name given to it for ADIOS to keep track of it
- PluginLibrary is the name of the shared lib for the plugin
- DataModel is a JSON file describing the mesh/fields
- Script is a ParaView Catalyst script

```
<io name="SimulationOutput">
    <engine type="plugin">
        <!-- general plugin engine parameters -->
        <parameter key="PluginName" value="fides"/>
        <parameter key="PluginLibrary" value="ParaViewADIOSInSituEngine"/>
        <!-- ParaViewFides engine parameters -->
        <parameter key="DataModel" value="catalyst/gs-fides.json"/>
        <parameter key="Script" value="catalyst/gs-pipeline.py"/>
    </engine>
</io>
```

Outline

- ADIOS 2 concepts and API (C++, Python, F90)
- BREAK
- Hands on with ADIOS 2 – Files
- QUICK BREAK
- ADIOS @ scale
- Staging with ADIOS
- Hands on with ADIOS 2 – Staging
- END

Gray-Scott example ADIOS2 version

Run gray-scott and gsplot.py insitu

Gray-Scott ADIOS2 insitu through files (CSD3)

$MYSCRATCH/mycode

```
$ cd ~/rds/hpc-work/mycode
$ vi insitu.sh
./cleanup.sh data
mpirun -n 16 ./gray-scott settings-files.json &
mpirun -n 1 python3 ./gsplot.py -i gs.bp -o u &
wait
$ chmod +x insitu.sh
```

```
$ ./insitu.sh
Simulation at step 10 publishing output step     1
Simulation at step 20 publishing output step     2
GS Plot step 0 processing stream step 0 sim step 10 minmax=0.0744497..1.05815
…
Simulation at step 1000 publishing output step     100
…
GS Plot step 99 processing stream step 99 sim step 1000 minmax=0.0941736..1.03609
```

```
$ ls u*png | wc -l
100
```

Gray-Scott ADIOS2 insitu using SST (CSD3)

$HOME/tutorial/mycode

```
$ vi adios2.xml
Replace
   <io name="SimulationOutput">
      <engine type="BP5">
to
   <io name="SimulationOutput">
      <engine type="SST">
```

```
$ ./insitu.sh
Simulation at step 10 publishing output step     1
Simulation at step 20 publishing output step     2
GS Plot step 0 processing stream step 58 simulation step 590
…
Simulation at step 1000 publishing output step     100
```

```
$ ls u*png | wc -l
1
```

SST default policy is to NOT block the producer by a slow consumer.
in adios2.xml:

```
<parameter key="QueueFullPolicy" value="Discard"/>
```

Gray-Scott ADIOS2 insitu using blocking SST (CSD3)

$HOME/tutorial/mycode

```
$ vi adios2.xml
<io name="SimulationOutput">
      <engine type="SST">
        <parameter key="RendezvousReaderCount" value="1"/>
        <parameter key="QueueFullPolicy" value="Block"/>
```

```
$ ./insitu.sh
Simulation at step 10 publishing output step     1
GS Plot step 0 processing stream step 0 sim step 10 minmax=0.0765537..1.05494
Simulation at step 20 publishing output step     2
GS Plot step 1 processing stream step 1 sim step 20 minmax=0.111669..1.05908
…
Simulation at step 990 publishing output step     99
GS Plot step 98 processing stream step 98 sim step 990 minmax=0.118174..1.03925
Simulation at step 1000 publishing output step     100
GS Plot step 99 processing stream step 99 sim step 1000 minmax=0.0910717..1.03391
```

```
$ ls u*png | wc -l
100
```

Producer can be told to wait for N consumers and
to wait for slow consumers to read all steps
in adios2.xml:

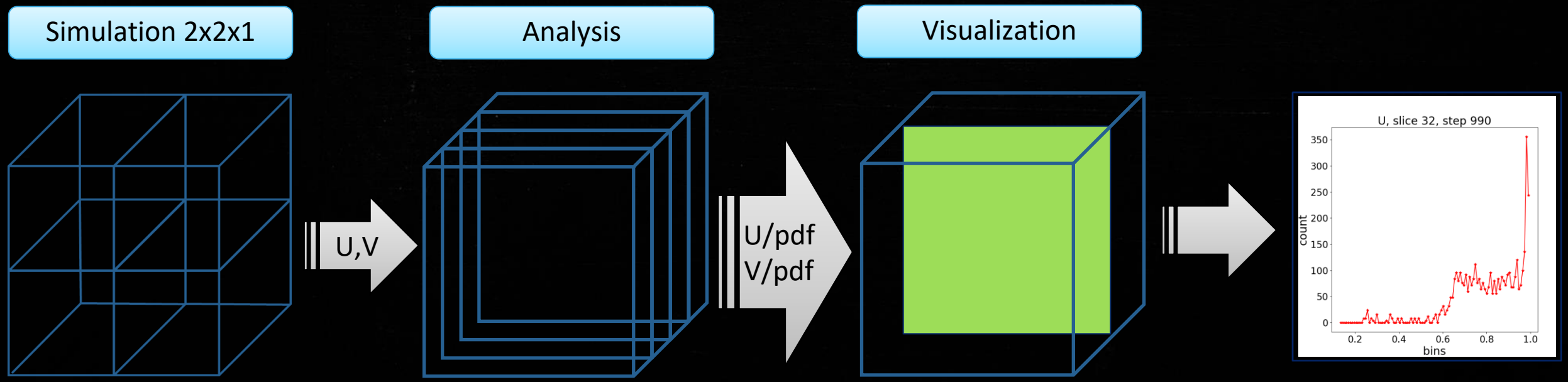
```
<parameter key="RendezvousReaderCount" value="1"/>
<parameter key="QueueFullPolicy" value="Block"/>
```

Home work

- Add V to the output of gray-scott
- Add xy and xz plane plots to gsplot.py
- Study gs-adios2/analysis/pdf-calc.cpp
  - for C++ reading code (also writes data with C++, no ghost cells)
- Study gs-adios2/pdfplot.py
  - a parallel python reader (using mpi4py)

Analysis and visualization (pdf-calc and pdfplot.py)

- 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

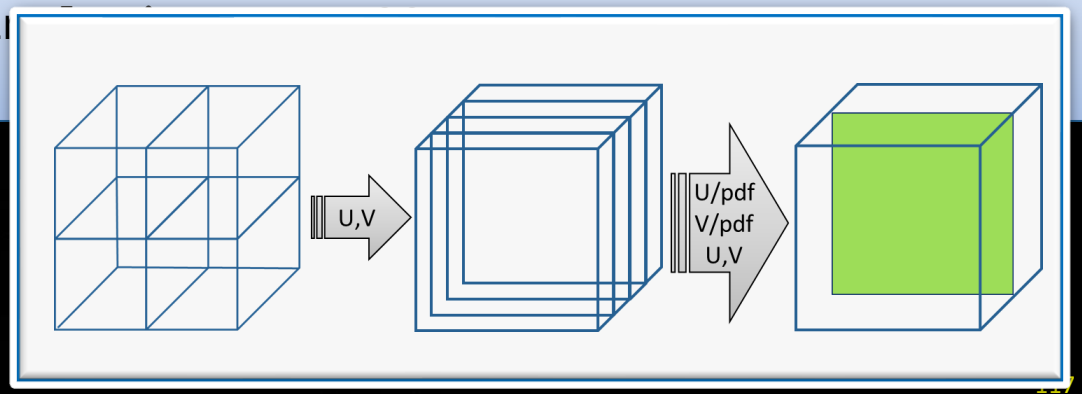


Goal by the end of the your homework: Running the example in situ

```
$ mpirun -n 4 ./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 ./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 si
...
```



Summary

ADIOS brings a programming interface and a framework of many solutions to the generic problem of producing and consuming data

- The interface frees scientists from the limited scope of file-based data processing
  - Being fully applicable to file-based data processing
- Offering a bridge from their scientific workflows that work now to the future, where they will extend their workflows with
  - More efficient data processing
  - Interactive visualization
  - Code coupling
  - On-the-fly AI training
  - Combining experimental data with simulation data