OAK RIDGE National Laboratory

ADIOS2 GPU aware
Defining application QoIs using derived variables

Ana Gainaru

ADIOS2 User Group

Feb 28, 2025

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

U.S. DEPARTMENT OF ENERGY

OAK RIDGE National Laboratory

ADIOS2 GPU aware

- GPU applications using ADIOS2
  - Using CUDA/HIP/Sycl/Kokkos
  - Using C++/Python

OAK RIDGE National Laboratory

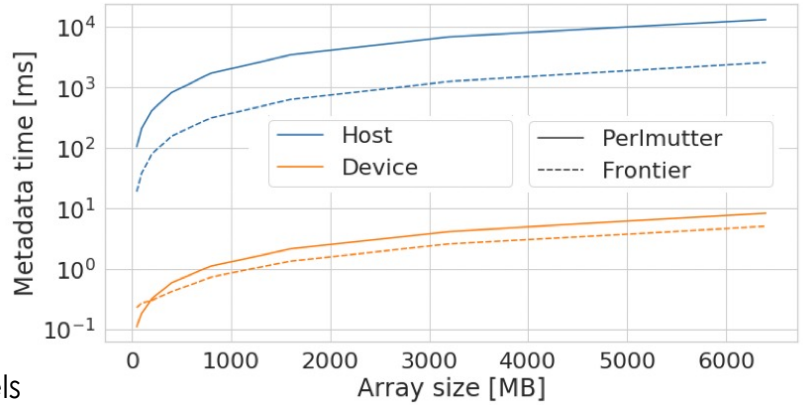
GPU-aware

- Allow applications to give ADIOS GPU buffers
  - Decrease number of copies of the data
  - Allow ADIOS to use GPU direct to storage, compression on GPU, or other optimizations
  - Transparent performance portability to different GPU architectures
- Build ADIOS2 with CUDA support –D ADIOS2_USE_CUDA=ON
- The user can provide a memory space
  - If not provided, ADIOS2 will detect automatically the memory space

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

- ADIOS2 saves pointers to data and copies data to internal CPU buffers
  - Computes metadata for each Get/Put using CUDA kernels

Performance of the GPU backend



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

Overhead for detecting where buffers are allocated

Compression with GPU-aware I/O

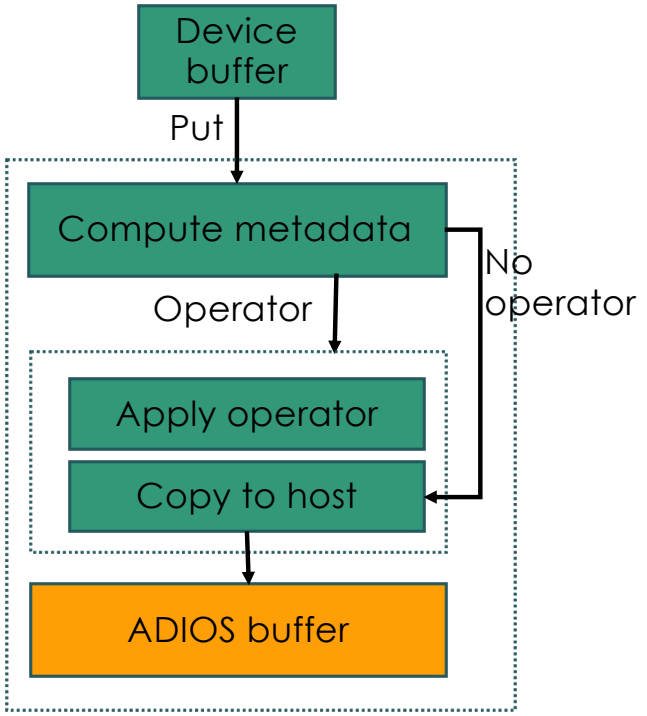
- No changes required in the source code
  - Operator attached to a variable
  - Memory space attached to a variable
- Internal logic
  - Metadata is computed using the GPU backend
  - The operator is applied on the GPU buffer and the compressed data is copied directly in the ADIOS buffer

```
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);
```

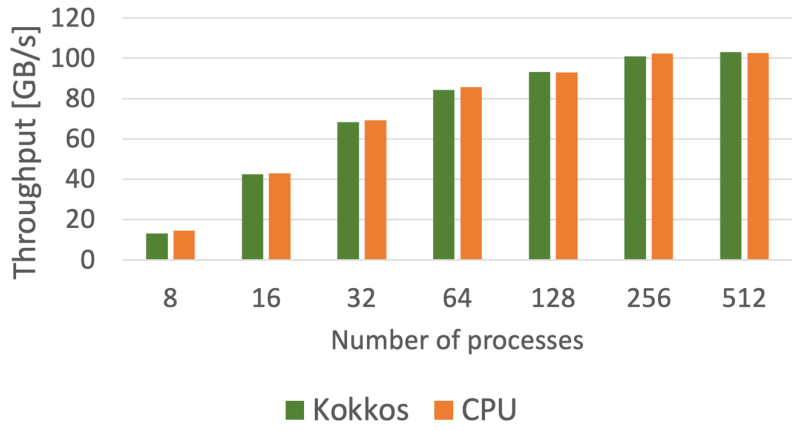


Operators that support GPU buffers:

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

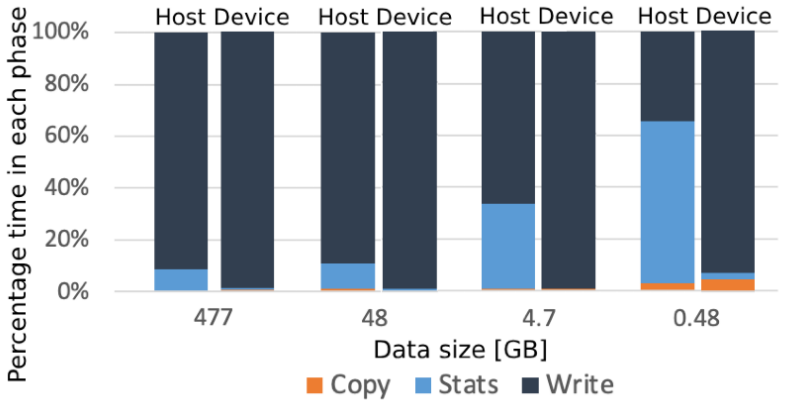
Performance

- When not collecting any metadata
  - The GPU backend has the same performance as the CPU backend



* Results for weak scaling on Summit, 64GB of data per node
* We measure the overall write throughput for all nodes

- Memory footprint
  - CPU backend
    - For chunks > 4MB keep user buffer
  - GPU backend uses internal buffers to hold the GPU data
  - Memory accessible from the Host
    - It's best to specify the memory space



* Single core performance breakdown

Backends and bindings

- Supported
  - Backend: CUDA and Kokkos (with CUDA, Sycl, HIP) backends
  - Engines: BP5, SST, dataman

```
$ ls ./examples/hello/:
bpStepsWriteReadCuda/                  sstKokkos/
bpStepsWriteReadHip/                   datamanKokkos/
bpStepsWriteReadKokkos/
```

  - Bindings: C++, Fortran, Python and in the next month C

```
$ ls ./examples/examples/hello/bpStepsWriteReadKokkos:
bpStepsWriteReadKokkos.cpp
bpStepsWriteReadKokkos.F90
bpWriteReadKokkosView.cpp

$ ls ./examples/examples/hello/bpStepsWriteReadCuda:
bpStepsWriteReadCuda.cu
bpStepsWriteReadCuda.py
```

Backends and bindings

CUDA C++

```
float * gpuData;
cudaMalloc(& gpuData, Nx * sizeof(float));


auto gpuVar = bpIO.DefineVariable<float>(”gpuArray",
  shape, start, count);

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

CUDA Python

```
import cupy as cp
gpuData = cp.array([[0, 1.0], [3.0, 4.0]], dtype=np.float32)


gpuVar = ioWriter.DefineVariable("gpuArray", npArrayWithSameType,
  shape, start, count)

gpuVar.SetMemorySpace(adios2.MemorySpace.GPU)
bpWriter.Put(gpuVar, gpuData.data.ptr)
```

Kokkos C++

```
Kokkos::View<float **, Layout> gpuData(”gpuArray", Nx,
Ny);

bpWriter.Put(gpuVar, gpuData);
```

Kokkos Fortran

```
call kokkos_allocate_view(ptr, gpuArray, ‘data’, int(N, c_size_t))

call adios2_define_variable(var_g, ioPut, &’gpuArray’,
     adios2_type_integer2, &2, ishape, istart, icount, ...)

call adios2_set_memory_space(var_g, adios2_memory_space_gpu, ierr)

call adios2_put(bpWriter, var_g, gpuArray, ierr)
```

OAK RIDGE National Laboratory

Defining application QoIs using derived variables

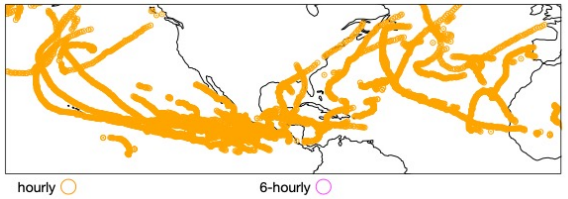
- Defining derived variables
- Writing/reading QoIs

What are derived quantities?

- Data or quantities of interest
  - Not specifically the result of the principal calculation of the application
  - Can be computed or extrapolated (derived) from primary data

- Why are they needed
  - Queries and analysis

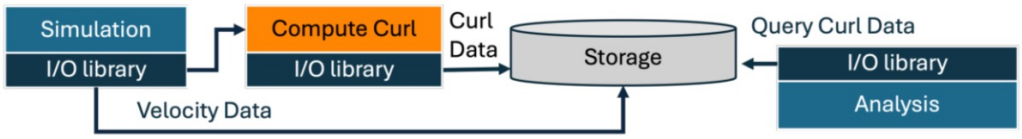
Cyclones found in 6-hourly data



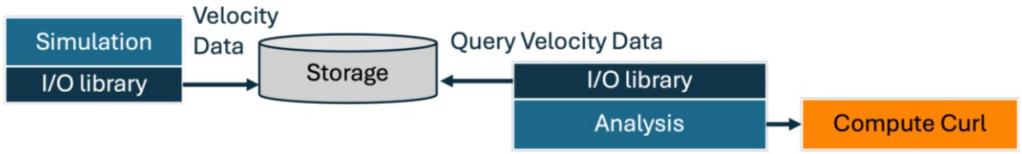
  - Typical query
    - Download 2D slices of the application output, manually choose areas of interest
    - Query on quantities of interest in the area of interest

Current solutions for derived variables

- Write side solutions
  - Workflows include analysis codes running with applications computing and storing the required derived data
- Read side solutions
  - Visualization/analysis technology capable of computing derived variables on the fly (e.g. Paraview)



(a) Write side solution



(b) Read side solution

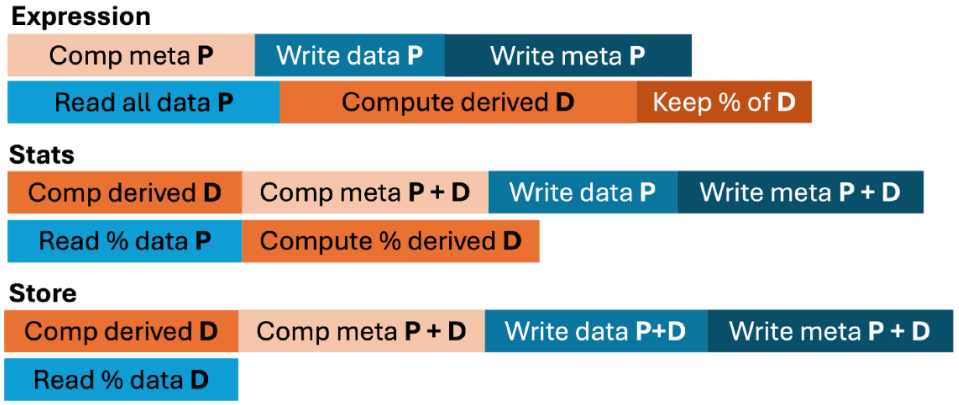
Current solutions for derived variables

- Write side solutions
  - Workflows include analysis codes running with applications computing and storing the required derived data
- Read side solutions
  - Visualization/analysis technology capable of computing derived variables on the fly (e.g. Paraview)
- **Offload this task to ADIOS2**
  - Choose for the application the best strategy for computing the derived variables
  - Hybrid solution
    - Write only metadata

```
IO::CreateDerived("Magnitude", velocityData);

for (i=0; i < simulationLoops; i++)
{
    // Compute new values for velocityData;
    IO::WriteToStorage(velocityData);
}
```

Trade-off between strategies



- **Store**
  - For low read network bandwidth
  - When storage is not an issue
  - When the analysis requires a lot of data
- **Expression**
  - For exploratory analysis (large amounts of data are investigated)
  - High read bandwidth
- **Stats**
  - For high compute units
    - Or trivial derived variables
  - When storage is an issue and data needs to be accessed remote
  - When the query is based on the quantity of interest but the analysis requires primary data

Derived variables in ADIOS2

- Define derived variable on **Write** side
  - Expression on ADIOS2 variables
  - Type of derived expression (Store, Stats, Expression)

```
adios2::DerivedVarType::StatsOnly,
adios2::DerivedVarType::ExpressionString,
adios2::DerivedVarType::StoreData
```

```
auto velocity = bpIO.DefineVariable<float>(
                            "velocity", shape, start, count);

bpIO.DefineDerivedVariable("derived/magnitude",
                        "v = velocity \n"
                        "magnitude(v)",
                        adios2::DerivedVarType::StatsOnly);
```

  - Call Put for primary variables in the normal way

- Inquire derived variable on **Read** side
  - Read data

```
$ bpls outputWithDerived.bp -l --show-derived

float    velocitu_X        10*{60000} = 0 / 45
float    velocity_Y        10*{60000} = 0 / 90
float    velocity_Z        10*{60000} = 0 / 60

float    derived/magnitude  10*{60000} = 0 / 100.623
   Derived variable with expression:
MAGNITUDE({velocity_X},{velocityY} ,{velocityZ})

 double   derived/sqrt      10*{60000} = 0 / 6.7082
```

  - Query on stats

Supported derived expressions

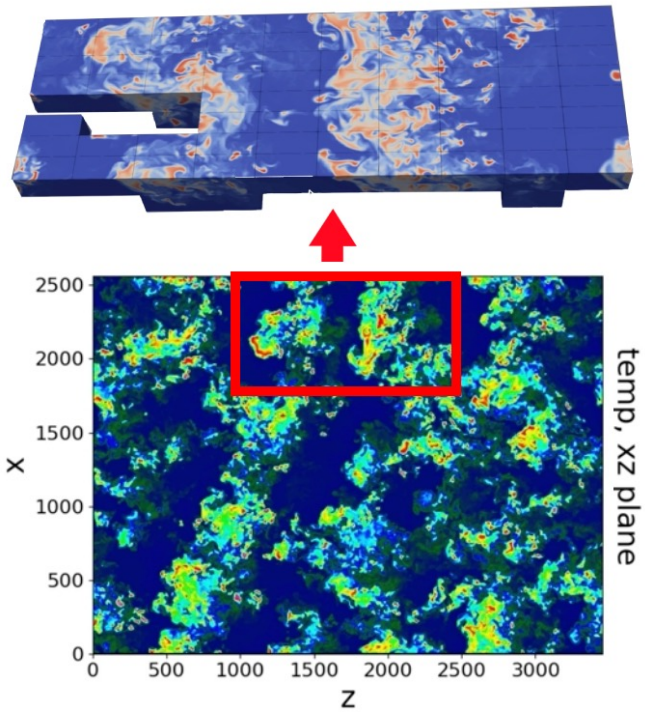
- **Scalar Math**
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Trig
  - Square Root
  - Pow
- **Vector Math**
  - Curl3D
  - Magnitude
  - Cross 3D
- **Statistics**
  - Mean
  - Median
  - Standard deviation
- Per process computations
- Limitations
  - No boundary exchanges
  - No timestamp aggregations

Aggregated expressions are supported (e.g. `sqrt(pow(x) + pow(y))`)

Performance

- The S3D simulation
  - Generates 1.5 TB of data in each step through 24 primary variables
  - Particles are stored in 3D arrays of 280x280x1280 size
    - Velocity is stored using 3 of separate variables, each requiring 64 GB on 900 ranks
  - **Query on magnitude either in-situ or on remote laptop, plot of temp**

- The e3sm simulation
  - Outputs model data at the 6-hourly interval generating around 24 GB through 9 primary variables on 96 ranks
  - Tropical cyclone track code queries the **magnitude of curl of velocity**

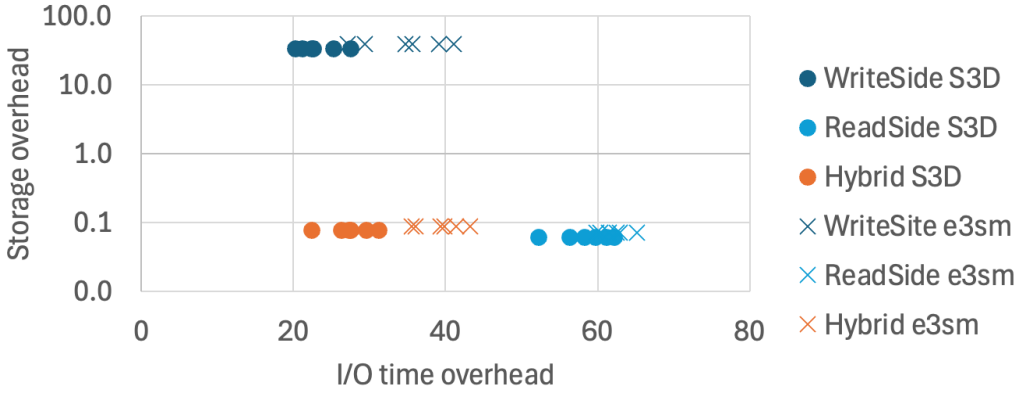


OAK RIDGE National Laboratory

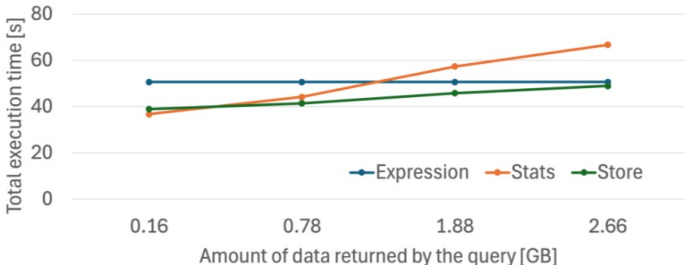
A. Gainaru et al. To Derive or Not to Derive: I/O Libraries Take Charge of Derived Quantities Computation, SBAC-PAD, 2024

Performance

- **S3D queries**
  - The **Write side** strategy adds 64 GB of data for each step
  - The **Read side** strategy requires storing 256 GB on the remote site
  - For 900 ranks the stats are12 MB
- **E3sm queries**
  - The size of the curl variables is 4 GB
  - The **Write side** strategy adds 28 GB
  - The stats for 96 ranks are 1 MB



The **Hybrid** strategy could be 1.5x slower due to curl having high complexity



OAK RIDGE National Laboratory

Send me your suggestions !

gainarua@ornl.gov

Wish list for future releases

- Derived variables on the GPU
- Metadata for GPU buffers
- More derived expressions supported
  - WarpX / GE / …
- Update the documentation

More performance results of the GPU backend will be presented at the Kokkos User Group meeting in May 2025 in Chicago

OAK RIDGE National Laboratory