LowFive: In Situ Data Transport for High-Performance Workflows

Tom Peterka, ANL
Dmitriy Morozov, LBNL
Arnur Nigmetov, LBNL
Orcun Yildiz, ANL
Bogdan Nicolae, ANL
Philip Davis, U. Utah

“Somewhere, something incredible is waiting to be known.”
—Carl Sagan

LowFive Library

Producer
write(step1.h5/group1/grid)
write(step1.h5/group2/particles)

HDF5 VOL plugin

In-memory distributed metadata mirroring
HDF5 data model

Consumer 1
read(step1.h5/group1/grid)

Optional passthrough

Consumer 2
read(step1.h5/group2/particles)

Storage
step1.h5

An example of three tasks coupled through the LowFive in situ data transport library.

github.com/diatomic/LowFive

RAPIDS Deep Dive
June 7, 2023

Tom Peterka
tpeterka@mcs.anl.gov
Mathematics and Computer Science Division
Executive Summary

In Situ Data Transport for Workflows

- Move data efficiently in parallel between tasks (codes)
- Minimize modification to tasks
- Select and map data between local subsets and a global model
- Redistribute data among different decompositions of tasks
- Synchronize data accesses between tasks
- Fan-in and fan-out among different numbers of tasks

LowFive

- In situ data transport layer for workflows
- HDF5 data model
- Built as an HDF5 VOL plugin
- Allows bypassing storage and sending data over MPI
- Redistributes data between producer and consumer tasks
- Standalone software library that workflow systems can use
Design Choices

A balance between user’s view of data (productivity) and the workflow’s efficient movement of data (performance)

<table>
<thead>
<tr>
<th>Design Criteria</th>
<th>LowFive Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>User’s view of data (model or schema)</td>
<td>HDF5 data model</td>
</tr>
<tr>
<td>In situ transport mechanism (direct, staging)</td>
<td>Direct, parallel, MPI point to point messages</td>
</tr>
<tr>
<td>Software stack intercept location</td>
<td>High-level HDF5 metadata</td>
</tr>
<tr>
<td>Software design</td>
<td>Standalone HDF5 VOL plugin</td>
</tr>
</tbody>
</table>
User’s View of Data

Objectives

• Self describing
• Flexible and able to handle complex models
• Supports wide variety of HPC, big data, AI, ML tasks
• Compatible with many existing codes

HDF5

• Hierarchical, sufficient complexity, self-describing
• Widely used
• Many APIs with HDF5 back-end: HighFive, h5py, NetCDF-4, SCORPIO, TensorFlow, Keras
• Rich metadata to support efficient parallel communication
• VOL plugin availability

Ref: HDF5 User’s Guide
In Situ Data Transport Mechanism

**Staging**
- Dedicated resources for transport
- Decouple producer from consumer (could allow overlap)
- May require launching a separate service
- Shared access (could also involve locking)

**Direct**
- No additional resources or services
- Simple, point-to-point communication
- Tightly coupled producer and consumer (synchronous)
- A staging area could still be a producer/consumer task

Logically, LowFive looks like a staging area, and it could have been implemented this way.

The actual implementation of LowFive, however, is direct point-to-point communication.
Software Stack Intercept Location

- POSIX level (Burst buffer systems)
  - Catch all I/O
  - No metadata
Software Stack Intercept Location

- Application level (Conduit, Bredala)
  - All metadata
  - Change user code

- POSIX level (Burst buffer systems)
  - Catch all I/O
  - No metadata
Software Stack Intercept Location

- Application level (Conduit, Bredala)
  - All metadata for data transport
  - Change user code

- High-level I/O API (LowFive)
  - Rich metadata for data transport
  - Little/no change to user code

- POSIX level (Burst buffer systems)
  - Catch all I/O
  - No metadata
Method
Software Stack

Scientific Simulations, AI, ML Frameworks
  Applications

HDF5, NetCDF-4, HighFive, H5Py
  I/O libraries

HDF5
  Data model
  Virtual Object Layer (VOL)

LowFive
  Data transfer

DIY
  Block parallelism

MPI
  Message passing

• DistMetadataVOL
• MetadataVOL
• VOLBase
LowFive Metadata Tree

HDF5 Data Model
- Hierarchical data model much like a UNIX file system
- Root is the file
- Internal nodes are groups
- Leaves are datasets or other objects (e.g., attributes)

LowFive Data Model
- Our in-memory replica of HDF5 metadata
- One object for every HDF5 object
- Shallow or deep data pointer or copy

Our own LowFive in-memory replica of HDF5 data model.
LowFive Properties and Options

• Optional file storage in addition to in-memory
  • Unique per file / dataset
  • Allow wildcard patterns
  • In-memory is default
• Deep / shallow data copy options
  • Deep copy: LowFive owns a copy
  • Shallow copy: LowFive points to user’s copy
  • Per file and per dataset
  • Allow wildcard patterns
  • Deep copy (LowFive owns a copy) is default
• Time- and space-partitioning
  • Everything goes through MPI messages
Data Redistribution: Problem

Example of data redistribution from a producer task with 6 processes decomposed row-wise to a consumer task with 4 processes decomposed column-wise. The problem is that neither the producer nor the consumer task knows anything about the other’s decomposition.
Data redistribution for previous example, showing the common decomposition on which both producer and consumer agree. The common decomposition is a virtual overlay on the producer side. Gray rectangles in the center panel represent the intersections of the producer and consumer decompositions with the common decomposition.
Synthetic Benchmarks

Different experiment scenarios

<table>
<thead>
<tr>
<th>Total # MPI Procs.</th>
<th># Producer Procs.</th>
<th># Consumer Procs.</th>
<th>Total # Grid Points</th>
<th>Total # Particles</th>
<th>Total Data Size (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3.0e6</td>
<td>3.0e6</td>
<td>0.06</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>4</td>
<td>1.2e7</td>
<td>1.2e7</td>
<td>0.22</td>
</tr>
<tr>
<td>64</td>
<td>48</td>
<td>16</td>
<td>4.8e7</td>
<td>4.8e7</td>
<td>0.99</td>
</tr>
<tr>
<td>256</td>
<td>192</td>
<td>64</td>
<td>1.9e8</td>
<td>1.9e8</td>
<td>3.54</td>
</tr>
<tr>
<td>1024</td>
<td>768</td>
<td>256</td>
<td>7.7e8</td>
<td>7.7e8</td>
<td>14.34</td>
</tr>
<tr>
<td>4096</td>
<td>3072</td>
<td>1024</td>
<td>3.0e9</td>
<td>3.0e9</td>
<td>55.88</td>
</tr>
<tr>
<td>16384</td>
<td>12288</td>
<td>4096</td>
<td>1.2e10</td>
<td>1.2e10</td>
<td>223.51</td>
</tr>
</tbody>
</table>

Number of processes and data sizes for synthetic benchmark experiments
Synthetic Benchmarks: In Situ vs. Storage

Time to write/read grid and particles between 1 producer task and 1 consumer task, comparing LowFive file and memory modes, in a weak scaling regime.
Synthetic Benchmarks: Overhead of Using LowFive vs. Pure HDF5 for File I/O

Time to write/read grid and particles, comparing LowFive file mode with pure HDF5 file, in a weak scaling regime.
Synthetic Benchmarks: Overhead of Using LowFive vs. Pure MPI for Message Passing

Time to write/read grid and particles comparing LowFive memory mode, with pure MPI communication, in a weak scaling regime.
Synthetic Benchmarks: 10X Data Size

- $10^7$ regularly structured grid points + $10^7$ particles per producer process
- 190 MiB of data per producer process
- 0.55 GiB of data per consumer process (3:1 producer:consumer procs)
- Total data size at the largest scale tested is 0.55 TiB.

Time to write/read large size grid and particles, comparing LowFive memory mode, DataSpaces, and pure MPI, in a weak scaling regime.
Time to write/read data between Nyx and Reeber using LowFive memory mode, HDF5 files, and AMReX plotfiles demonstrates that LowFive in situ data transport is 20X faster at scale than the best I/O solution (AMReX plotfile format).
Next Steps

• Finish implementing missing functions in our metadata
• Continue to test on applications and their software stacks
• Producer – consumer synchronization and flow control
• Integrate in workflow systems driving further development
  • Henson can use LowFive (Nyx + Reeber use case)
  • We are also developing a new workflow system---Wilkins---on top of Henson and LowFive
Use Cases and Deeper Software Stacks

Climate modeling software stack using NetCDF data model

- Wilkins
- Henson
- E3SM Climate Codes
- SCORPIO I/O Library
- NetCDF-4
- HDF5
- LowFive
- MPI

Cosmology software stack using AMR data model

- Wilkins
- Henson
- Nyx Cosmology Code
- AMReX AMR Library
- HDF5
- LowFive
- MPI

AI software stack using tensor data model

- Wilkins
- Henson
- Keras
- H5py
- HDF5
- LowFive
- MPI
Acknowledgments

Facilities
- Argonne Leadership Computing Facility (ALCF)
- Argonne Laboratory Computing Resource Center (LCRC)
- Oak Ridge Leadership Computing Facility (OLCF)
- National Energy Research Scientific Computing Center (NERSC)

Funding
- DOE ASCR Research Program
- Margaret Lentz

People
- Tom Peterka, Dmitriy Morozov, Arnur Nigmetov, Orcun Yildiz, Bogdan Nicolae, Philip Davis