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

github.com/diatomic/LowFive
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>
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
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.
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.
Synthetic Benchmarks

Different experiment scenarios

<table>
<thead>
<tr>
<th>Producer</th>
<th>LowFive Memory</th>
<th>Pure MPI</th>
<th>Data Spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure HDF5</td>
<td>LowFive File</td>
<td>PFS</td>
<td></td>
</tr>
</tbody>
</table>

Number of processes and data sizes for synthetic benchmark experiments

<table>
<thead>
<tr>
<th>Total # MPI Procs.</th>
<th>Total # Grid Points</th>
<th>Total # Particles</th>
<th>Total Data Size (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.2e7</td>
<td>1.2e7</td>
<td>0.06</td>
</tr>
<tr>
<td>12</td>
<td>4.8e7</td>
<td>4.8e7</td>
<td>0.22</td>
</tr>
<tr>
<td>48</td>
<td>1.9e8</td>
<td>1.9e8</td>
<td>0.99</td>
</tr>
<tr>
<td>192</td>
<td>7.7e8</td>
<td>7.7e8</td>
<td>3.54</td>
</tr>
<tr>
<td>768</td>
<td>1.2e10</td>
<td>1.2e10</td>
<td>14.34</td>
</tr>
<tr>
<td>3072</td>
<td>3.0e9</td>
<td>3.0e9</td>
<td>55.88</td>
</tr>
<tr>
<td>12288</td>
<td></td>
<td></td>
<td>223.51</td>
</tr>
</tbody>
</table>
Synthetic Benchmarks: In Situ vs. Storage

Weak Scaling LowFive File vs Memory Mode

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

We compare the completion time of writing and reading grid and particles in LowFive memory mode versus pure MPI communication in a weak scaling regime.

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.
Science Workflow: Cosmology

Both Nyx and Reeber were used “off the shelf” with no modifications to use LowFive (in the Henson workflow system).

<table>
<thead>
<tr>
<th>Data Size</th>
<th>LowFive Write Time</th>
<th>LowFive Read Time</th>
<th>HDF5 Write Time</th>
<th>HDF5 Read Time</th>
<th>Plotfiles Write Time</th>
<th>LowFive vs HDF5</th>
<th>LowFive vs Plotfiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>256^3</td>
<td>2.87</td>
<td>0.106</td>
<td>5.46</td>
<td>0.37</td>
<td>4.42</td>
<td>1.9</td>
<td>1.54</td>
</tr>
<tr>
<td>512^3</td>
<td>2.00</td>
<td>0.287</td>
<td>104.20</td>
<td>0.69</td>
<td>18.10</td>
<td>52.01</td>
<td>9.03</td>
</tr>
<tr>
<td>1024^3</td>
<td>2.87</td>
<td>0.628</td>
<td>920.44</td>
<td>3.02</td>
<td>35.00</td>
<td>320.00</td>
<td>12.17</td>
</tr>
<tr>
<td>2048^3</td>
<td>7.69</td>
<td>3.205</td>
<td>x</td>
<td>x</td>
<td>154.52</td>
<td>x</td>
<td>20.09</td>
</tr>
</tbody>
</table>

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).
Recap

**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
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

github.com/diatomic/LowFive