Managing Large-Scale Data
Using ADIOS

Aug. 6 2013
Knoxville, TN

Norbert Podhorszki
pnorbert@ornl.gov
Jeremy Logan
jlogan7@utk.edu
University of Tennessee
Outline

- ADIOS Introduction
- ADIOS Write API
- Hands-on 1, Write data with ADIOS
- Hands-on 1, Tools
- ADIOS Read API
- Hands-on 1, Read data with ADIOS
- Hands-on 1++, Spatial aggregation
- Hands-on 2, ADIOS Write API (Non-XML version)
- Hands-on 4, Multi-block writing with non-XML API
- Hands-on 3, Staging example
- Hands-on 4, Visualization of ADIOS data
- Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
- Hands-on 8 Python
- Hands-on 9, Java
- Summary
ADIOS Participants

• ORNL: Hasan Abbasi, Jong Youl Choi, **Scott Klasky**, Qing Liu, Kimmy Mu, Norbert Podhorszki, Dave Pugmire, Roselyne Tchoua
• Georgia Tech: Greg Eisenhauer, Jay Lofstead, Karsten Schwan, Matt Wolf, Fang Zhang
• UTK: Jeremy Logan, Yuan Tian
• Rutgers: C. Docan, Tong Jin, Manish Parashar, Fan Zhang
• NCSU: Drew Boyuka, Z. Gong, Nagiza Samatova,
• LBNL: Arie Shoshani, John Wu
• Emory University: Tahsin Kurc, Joel Saltz
• Sandia: Jackie Chen, Todd Kordenbock, Ken Moreland
• NREL: Ray Grout
• PPPL: C. S. Chang, Stephane Ethier, Seung Hoe Ku, William Tang
• Caltech: Julian Cummings
• UCI: Zhihong Lin
• Tsinghua University (China): Wei Xue, Lizhi Wang

*Red* marks major research/developer for the ADIOS project, *Blue* denotes student, *Green* denotes application scientists
Thanks for our Current Funding 😊

1. OLCF: ADIOS, Barb Helland (Buddy Bland)
2. Runtime Staging, Exascale-SDM, ASCR: Lucy Nowell
3. Scalable Data Management, Analysis and Visualization Institute, ASCR: Lucy Nowell
5. NSF, An Application Driven I/O Optimization Approach for PetaScale Systems and Scientific Discoveries: Almadena Chtchelkanova
7. NSF, Remote Data Analysis and Visualization: Barry Schneider
9. OFES, Center for Nonlinear Simulation of Energetic Particles in Burning Plasmas (CSEP), John Mandrekas
10. OFES, Energetic Particles, John Mandrekas
11. BES, Network for ab initio many-body methods: development, education and training, Hans M. Christen
12. NSF-NSFC, High Performance I/O Methods and infrastructure for large-scale geo-science applications on supercomputers., D. Katz
Application Partners

Over 158 Publications from 2006 - 2013

<table>
<thead>
<tr>
<th>Application</th>
<th>Code</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrophysics</td>
<td>Chimera</td>
<td>T. Mezzacappa</td>
</tr>
<tr>
<td>Combustion</td>
<td>S3D</td>
<td>J. Chen</td>
</tr>
<tr>
<td>AMR</td>
<td>Chombo</td>
<td>B. Van Straalen</td>
</tr>
<tr>
<td>CFD</td>
<td>Fine/Turbo</td>
<td>M. Gontier</td>
</tr>
<tr>
<td>Fusion-Edge</td>
<td>XGC1</td>
<td>C. S. Chang</td>
</tr>
<tr>
<td>Fusion-Edge</td>
<td>XGC0</td>
<td>S. H. Ku</td>
</tr>
<tr>
<td>Geoscience</td>
<td>AWP-ODC</td>
<td>Y. Cui</td>
</tr>
<tr>
<td>Materials</td>
<td>LAMMPS</td>
<td>A. Franchioni</td>
</tr>
<tr>
<td>Weather</td>
<td>GRAPES</td>
<td>W. Xue</td>
</tr>
<tr>
<td>Nuclear</td>
<td>HFODD</td>
<td>H. Nam</td>
</tr>
<tr>
<td>Relativity</td>
<td>Maya</td>
<td>P. Laguna</td>
</tr>
<tr>
<td>Sub surface</td>
<td></td>
<td>M. Wheeler</td>
</tr>
<tr>
<td>Materials</td>
<td></td>
<td>M. Parashar</td>
</tr>
<tr>
<td>Fusion</td>
<td>GTC-P</td>
<td>S. Ethier</td>
</tr>
<tr>
<td>Fusion</td>
<td>GTS</td>
<td>W. Wang</td>
</tr>
<tr>
<td>Fusion</td>
<td>M3D-C1</td>
<td>S. Jardin</td>
</tr>
<tr>
<td>Fusion</td>
<td>M3D-K</td>
<td>G. Y. Fu</td>
</tr>
<tr>
<td>Fusion</td>
<td>GTC</td>
<td>Z. Lin</td>
</tr>
<tr>
<td>Fusion</td>
<td>GEM</td>
<td>S. Parker</td>
</tr>
<tr>
<td>Quantum</td>
<td>QLG2Q</td>
<td>M. Soe</td>
</tr>
<tr>
<td>Image Analysis</td>
<td></td>
<td>T. Kurc</td>
</tr>
<tr>
<td>AMR</td>
<td>Boxlib</td>
<td>J. Bell</td>
</tr>
<tr>
<td>Relativity</td>
<td>Cactus</td>
<td>G. Allen</td>
</tr>
<tr>
<td>Weather</td>
<td>NASA</td>
<td>T. Clune</td>
</tr>
<tr>
<td>Materials</td>
<td>QMC</td>
<td>J. Kim</td>
</tr>
<tr>
<td>Climate</td>
<td></td>
<td>J. Fu</td>
</tr>
<tr>
<td>Climate</td>
<td>CAM</td>
<td>K. Evans</td>
</tr>
</tbody>
</table>
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
Extreme scale computing

• Trends
  – More FLOPS
  – Limited number of users at the extreme scale

• Problems
  – Performance
  – Resiliency
  – Debugging
  – Getting Science done

• Problems will get worse
  – Need a “revolutionary” way to store, access, debug to get the science done!

## File Systems

<table>
<thead>
<tr>
<th>File system</th>
<th>Hard links</th>
<th>Symbolic links</th>
<th>Block journaling</th>
<th>Metadata-only journaling</th>
<th>Case-sensitive</th>
<th>Case-preserving</th>
<th>File Change Log</th>
<th>Snapshot</th>
<th>XIP</th>
<th>Encryption</th>
<th>COW</th>
<th>Integrated LVM</th>
<th>Data deduplication</th>
<th>Volumes are realtimeable</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPM file system</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>DECtape</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>Level-D</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>RT-11</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>DOS (DEC)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>OS/4000</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>VF5S</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>V7FS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>FAT12</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Offline</td>
</tr>
<tr>
<td>FAT16</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Offline</td>
</tr>
<tr>
<td>FAT32</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Offline</td>
</tr>
<tr>
<td>exFAT</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Unknown</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>GFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>GPFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Unknown</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>HFS</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>HFS Plus</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>FF5</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>UF51</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>UF52</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>LFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ext2</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ext3</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ext4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>Lustre</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>NILFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ReiserFS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ReiserNT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>OCFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>OCFS2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>Reliance</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>Reliance Nitro</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>XFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>JFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>QFS</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>Be File System</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>unknown</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>NS5</td>
<td>Yes</td>
<td>Yes</td>
<td>Unknown</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>NWFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ODS-2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ODS-5</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>UDF</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>VxFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>Posix</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>ZFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>VMFS2</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>VMFS3</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
<tr>
<td>Btrfs</td>
<td>Yes</td>
<td>Yes</td>
<td>Unknown</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

**Note:** The table contains information about various file systems, including their support for hard links, symbolic links, block journaling, metadata-only journaling, case-sensitive, case-preserving, file change log, snapshotting, XIP, encryption, COW, integrated LVM, data deduplication, and volumes that are realizable.
The SOA philosophy for HPC/ADIOS

• The overarching design philosophy of our framework is based on the **Service-Oriented Architecture**
  – Used to deal with system/application complexity, rapidly changing requirements, evolving target platforms, and diverse teams
• Applications constructed by assembling services based on a universal view of their functionality using a **well-defined API**
• Service implementations can be changed easily
• Integrated simulation can be assembled using these services
• **Manage complexity while maintaining performance/scalability**
  – Complexity from the problem (complex physics)
  – Complexity from the codes and how they are
  – Complexity of underlying disruptive infrastructure
  – Complexity from coordination across codes and research teams
  – Complexity of the end-to-end workflows
Our Goals for sustainable software development

- Ease of use
- High Performance
- Scalable
- Portable
- Easy to master
ADIOS

- An I/O abstraction framework
- Provides portable, fast, scalable, easy-to-use, metadata rich output
- Change I/O method on-the-fly
- Abstracts the API from the method http://www.nccs.gov/user-support/center-projects/adios/

- Typical 10X performance improvement for synchronous I/O over other solutions

Georgia Tech, Rutgers, NCSU, Emory, Auburn, Sandia, LBL, PPPL
Contributions in bridging the data gap for high fidelity science

Performance Improvement over other parallel I/O solutions

<table>
<thead>
<tr>
<th>Application Areas</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fusion</td>
<td>1000%</td>
</tr>
<tr>
<td>Square Kilometer Array</td>
<td>1000%</td>
</tr>
<tr>
<td>Seismology</td>
<td>1500%</td>
</tr>
<tr>
<td>Quantum Physics</td>
<td>2000%</td>
</tr>
<tr>
<td>Astrophysics</td>
<td>2000%</td>
</tr>
<tr>
<td>Combustion</td>
<td>3000%</td>
</tr>
<tr>
<td>Climate Analysis</td>
<td>7300%</td>
</tr>
<tr>
<td>Industrial Engineering</td>
<td>10000%</td>
</tr>
</tbody>
</table>
Most cited ADIOS-related publications

- 96 Flexible io and integration for scientific codes through the adaptable io system (adios)
- 81 Datastager: scalable data staging services for petascale applications
- 76 Adaptable, metadata rich IO methods for portable high performance IO
- 53 PreDatA preparatory data analytics on peta-scale machines
- 46 Managing variability in the IO performance of petascale storage systems
- 43 Grid-based parallel data streaming implemented for the gyrokinetic toroidal code
- 36 DataSpaces: An interaction and coordination framework for coupled simulation workflows
- 35 High performance threaded data streaming for large scale simulations
- 32 Workflow automation for processing plasma fusion simulation data
- 26 Plasma edge kinetic-MHD modeling in tokamaks using Kepler workflow for code coupling, data management and visualization
- 26 An autonomic service architecture for self-managing grid applications
- 23 Extending i/o through high performance data services
- 19 EDO: improving read performance for scientific applications through elastic data organization
- 19 Six degrees of scientific data: reading patterns for extreme scale science IO
- 19 Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data
ADIOS information

• ADIOS is an I/O framework
  – Similar software philosophy as Linux: there is no single owner
  – Provides multiple methods to stage data to a staging area (on node, off node, off machine)
  – Data output can be anything one wants
    • Different methods allow for different types of data movement, aggregation, and arrangement to the storage system or to stream over the local-nodes, LAN, WAN
  – It contains our own file format if you choose to use it (ADIOS-BP)
  – In the next release, it will be able to compress/decompress data in parallel: plugged into the transform layer
  – In the next release: it will contain mechanisms to index and then query the data
The History of ADIOS

- General Relativity: enabled interactive analysis
- Z. Lin: Allowed fusion code to scale from 100’s to 1000’s of cores
- Ramgen/Numeca: First commercial code to use ADIOS, allowed them to scale CFD simulations from .5B cells to 3.7B cells
- GEOS5 climate code reading performance improved 78X over parallel HDF5
- J. Chen, combustion, allows us to scale beyond the MPI-IO barrier at 30K cores
- Workflow streams from KSTAR to ORNL for 10X faster data processing of KSTAR ECEI data
- ADIOS 1.0 Released
- Data Staging
- ADIOS 1.3 for reading performance
- Move work to data with JIT for analysis
- ADIOS: 3 Phase Parallel I/O
- abstract I/O visualization for relativity
- thread and buffer data
- BP file format
- Predata, DataSpaces-I/O pipelines
**ADIOS latest Release**

- **1.5.0 Release June 2013**
- New staging methods:
  - DIMES
  - FLEXPATH
- CMAKE build files (besides Automake files)
- New write method VAR_MERGE for spatial aggregation of small per-process-output into larger chunks. It improves both write and read performance for applications
- Please give us suggestions for future releases after the next one at SC-2013.

**Aggregated file read method**

**Staged write C code**
- Uses any staging or file method to read data, writes with any ADIOS method
Favorite highlights

• "I appreciate the clean yet capable interface and that from my perspective, it just works," Grout.
• “ADIOS has allowed us to couple fusion codes to solve new physics problems”, Chang.
• “So far I have found that even using ADIOS in the way you said is bad and silly in every aspect, the I/O performance is still improved by about 10X”, J. Wang ICAR

• “... thanks to Dr. Podhorszki, the developers at Numeca have had excellent support in integrating ADIOS into FINE/Turbo. We have observed a 100x speedup of I/O, which is difficult to achieve in commercial codes on industrial applications”, Grosvenor.
LUSTRE

- Lustre consists of four major components
  - MetaData Server (MDS)
  - Object Storage Servers (OSSs)
  - Object Storage Targets (OSTs)
  - Clients

- Performance: Striping, alignment, placement

GPFS is similar, but ...

Writing to file system

• Avoid latency (of small writes)
  – **Buffer** data for large bursts

• Avoid accessing a file system target from many processes at once
  – **Aggregate** to a small number of actual writers
    • proportionate to the number of file system targets, not MPI tasks

• Avoid lock contention
  – by **striping** correctly
  – Our method writes to subfiles

• Avoid global communication during I/O
  – ADIOS-BP file format
• All data chunks are from a single producer
  – MPI process
  – Single diagnostic

• Ability to create a separate metadata file when “sub-files” are generated

• Allow code to be integrated to streams

• Allows variables to be individually compressed

• Format is for “data-in-motion” and “data-at-rest”
Reading performance

- Aggregate and place chunks on file system with an Elastic Data Organization
- Ability to optimize for common read patterns (e.g. 2D slice from 3D variable), space-time optimizations
- **Achieved a 73x speedup for read performance, and 11x speedup for write performance in mission critical climate simulation GEOS-5 (NASA), on Jaguar.**

First place ACM student Research Competition 2012
Introduction to Staging
• Initial development as a research effort to minimize I/O overhead
• Draws from past work on threaded I/O
• Exploits network hardware support for fast data transfer to remote memory


Automatic Benchmark Generation

- **Skel** addresses various issues with I/O kernels
  - Automatic generation reduces development burden and enhances applicability
  - Skeletals are easily kept up to date with application changes
  - Skel provides a consistent user experience across all applications
    - No scientific libraries are needed
    - No input data is needed
    - Measurements output from skeletals are standard for all applications
- The correctness of Skel has been validated for several applications and three synchronous I/O methods
Outline

- ADIOS Introduction
- **ADIOS Write API**
  - Hands-on 1, Write data with ADIOS
  - Hands-on 1, Tools
- ADIOS Read API
  - Hands-on 1, Read data with ADIOS
  - Hands-on 1++, Spatial aggregation
  - Hands-on 2, ADIOS Write API (Non-XML version)
  - Hands-on 2, Multi-block writing with non-XML API
  - Hands-on 3, Staging example
  - Hands-on 4, Visualization of ADIOS data
  - Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
  - Hands-on 8 Python
  - Hands-on 9, Java
- Summary
Writing setup/cleanup API

- Initialize/cleanup
  - `#include “adios.h”`
  - `adios_init (‘config.xml’, comm)`
    - parse XML file on each process
    - setup transport methods
    - `MPI_Comm`
      - only one process reads the XML file
      - some staging methods can connect to staging server
  - `adios_finalize (rank)`
    - give each transport method opportunity to cleanup
    - particularly important for asynchronous methods to make sure they have completed before exiting
    - call just before `MPI_Finalize()`
  - `adios_init_noxml (comm)`
    - Use instead of `adios_init()` when there is no XML configuration file
    - Extra APIs allows for defining variables
API for writing 1/3

• Open for writing
  – adios_open (fh, “group name”, “file name”, mode, comm)
    • int64_t fh handle used for subsequent calls for write/close
    • “group name” matches an entry in the XML
      – identifies the set of variables and attributes that will be written
    • Mode is one of ‘w’ (write) or ‘a’ (append)
    • Communicator tells ADIOS what processes in the application will perform I/O on this file

• Close
  – adios_close (fh)
    • handle from open

Fortran
integer*8 :: fd
call adios_open (fd, group_name, filename, mode, comm, err)
call adios_close (fd, err)
API for writing 2/3

• Write
  – adios_write (fh, “varname”, data)
    • fh is the handle from open
    • Name of variable in XML for this group
    • Data is the reference/pointer to the actual data
  – NOTE: with a XML configuration file, adios can build
    Fortran or C code that contains all of the write calls for all
    variables defined for a group

• Must specify one call per variable written

Fortran
call adios_write (fd, varname, data, err)
Important notes about the write calls

- **adios_write()**
  - usually *does not write* to the final target (e.g. file)
  - most of the time it only buffers data locally
  - when the call returns, the application can re-use the variable’s memory

- **adios_close()**
  - takes care of getting all data to the final target
  - usually the buffered data is *written at this time*
API for writing 3/3

One final piece required for buffer overflows

- **adios_group_size (int64_t fh, uint64_t data_size, uint64_t &total_size)**
  - called after adios_open(), before any adios_write()
  - fh is the handle returned from open
  - data_size is the size of the data in bytes to be written
    - by this particular process
    - i.e. the size of all writes between this particular open/close step
  - total_size is returned by the function
    - how many bytes will really be written (your data + all metadata) from this process

- gpp.py generates
  - the adios_group_size and all adios_write statements
  - 2 files per group (1 for read, 1 for write) in the language specified in the XML file (C style or Fortran style)
    - you need to include the appropriate file in your source after adios_open

**Fortran**
call adios_group_size (fd, data_size, total_size, err)
Outline

• ADIOS Introduction
• ADIOS Write API
• **Hands-on 1, Write data with ADIOS**
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
• Goals
  – Write an Fortran90/MPI code with ADIOS I/O
  – How to compile ADIOS codes.
  – How to run ADIOS codes.
  – How to create global arrays.
Write Example

• In this example we start with a 2D code which writes data of a 2D array, with a 2D domain decomposition, as shown in the figure.
  • $xy = 1.0 \cdot \text{rank} + 1.0 \cdot ts$

• We write out 2 time-steps, in separate files.
• For simplicity, we work on only 12 cores, arranged in a 4 x 3 arrangement.
• Each processor allocates a 65x129 array ($xy$).
• The total size of the array = 4*65, 3*129
The ADIOS XML configuration file

• Describe each IO grouping.
• Maps a variable in the code, to a variable in a file.
• Map an IO grouping to transport method(s).
• Define buffering allowance
• “XML-free” API are also provided
XML Overview

- **writer.xml** describes the output variables in the code

```xml
<adios-group name="writer">
  <var name="nx_global" type="integer"/>
  <var name="ny_global" type="integer"/>
  <var name="nx_local" type="integer"/>
  <var name="ny_local" type="integer"/>
  <var name="offs_x" type="integer"/>
  <var name="offs_y" type="integer"/>
  <global-bounds dimensions="nx_global, ny_global"
                   offsets="offs_x, offs_y">
    <var name="xy" type="double"
         dimensions="nx_local, ny_local"/>
  </global-bounds>
</adios-group>
<transport group="writer" method="MPI"/>
```
XML overview (global array)

• We want to read in xy from an arbitrary number of processors, so we need to write this as a global array.

• Need 2 more variables, to define the offset in the global domain
  – `<var name="offs_x" type="integer"/>
  – `<var name="offs_y" type="integer"/>

• Need to define the xy variable as a global array
  – Place this around the lines defining xy in the XML file.
  – `<global-bounds dimensions="nx_global,ny_global"
    offsets="offs_x,offs_y">
  – `</global-bounds>`
Calculating sizes and dimensions

\[ \text{posx} = \text{mod(rank, npx)} \] ! 1st dim: 0, npx, 2npx... are in the same X position
\[ \text{posy} = \text{rank/npx} \] ! 2nd dim: npx processes belong into one dim
\[ \text{offs}_x = \text{posx} \times \text{ndx} \] ! The processor offset in the x dimension for the global dimensions
\[ \text{offs}_y = \text{posy} \times \text{ndy} \] ! The processor offset in the x dimension for the global dimensions
\[ \text{nx}_\text{local} = \text{ndx} \] ! The size of data that the processor will write in the x dimension
\[ \text{ny}_\text{local} = \text{ndy} \] ! The size of data that the processor will write in the y dimension
\[ \text{nx}_\text{global} = \text{npx} \times \text{ndx} \] ! The size of data in the x dimension for the global dimensions
\[ \text{ny}_\text{global} = \text{npy} \times \text{ndy} \] ! The size of data in the y dimension for the global dimensions
XML Overview

• Need to define the method, we will use MPI.
  – <transport group="writer" method="MPI"/>

• Need to define the buffer
  – <buffer size-MB="4" allocate-time="now"/>
  – Can use any size, but if the buffer > amount to write, the I/O to disk will be faster.

• Need to define the host language (C or Fortran ordering of arrays).
  – <adios-config host-language="Fortran">

• Set the XML version
  – <?xml version="1.0"?>

• And end the configuration file
  – </adios-config>
The final XML file

1. <?xml version="1.0"?>
2. <adios-config host-language="Fortran">
3.  <adios-group name="writer">
4.   <var name="nx_global" type="integer"/>
5.   <var name="ny_global" type="integer"/>
6.   <var name="offs_x" type="integer"/>
7.   <var name="offs_y" type="integer"/>
8.   <var name="nx_local" type="integer"/>
9.   <var name="ny_local" type="integer"/>
10.  <global-bounds dimensions="nx_global,ny_global" offsets="offs_x,offs_y">
11.   <var name="xy" type="real*8" dimensions="nx_local,ny_local"/>
12.  </global-bounds>
13.  </adios-group>
14.  <transportgroup="writer" method="MPI"/>
15.  <buffer size-MB="4" allocate-time="now"/>
16.  </adios-config>
gpp.py

• Converts the XML file into F90 (or C) code.
• > gpp.py writer.xml
• > cat gwrite_writer.fh

adios_groupsize = 4 &
+ 4 &
+ 4 &
+ 4 &
+ 4 &
+ 4 &
+ 8 * (nx_local) * (ny_local)
call adios_group_size (adios_handle, adios_groupsize, adios_totalsize, adios_err)
call adios_write (adios_handle, "nx_global", nx_global, adios_err)
call adios_write (adios_handle, "ny_global", ny_global, adios_err)
call adios_write (adios_handle, "offs_x", offs_x, adios_err)
call adios_write (adios_handle, "offs_y", offs_y, adios_err)
call adios_write (adios_handle, "nx_local", nx_local, adios_err)
call adios_write (adios_handle, "ny_local", ny_local, adios_err)
call adios_write (adios_handle, "xy", xy, adios_err)
Writing with ADIOS I/O (simplest form)

call adios_init ("writer.xml", group_comm, adios_err)
...
call adios_open (adios_handle, "writer", trim(filename),
    "w", group_comm, adios_err)

#include "gwrite_writer.fh"

call adios_close (adios_handle, adios_err)
...
call adios_finalize (rank, adios_err)

Source file extension should be .F90 (instead of .f90)
to enforce macro preprocessing at compile time
Compile ADIOS codes

• Makefile
  – use adios_config tool to get compile and link options

```latex
ADIOS_DIR = /sw/xk6/adios/1.5.0/cle4.0_pgi12.10.0
ADIOS_INC = $(shell ${ADIOS_DIR}/bin/adios_config -c -f)
ADIOS_FLIB = $(shell ${ADIOS_DIR}/bin/adios_config -l -f)
ADIOSREAD_FLIB := $(shell ${ADIOS_DIR}/bin/adios_config -l -f -r)
```

– Codes that write and read

```latex
writer: writer.F90 gwrite_writer.fh
   ${FC} -g -c -o writer.o ${ADIOS_INC} writer.F90
   ${LINKER} -g -o writer writer.o ${ADIOS_FLIB}

gwrite_writer.fh: writer.xml
   ${GPP} writer.xml
```
Compile and run the code

Chester

```bash
$ cd /tmp/work/<username>/adios-1.5.0-tutorial-chester
$ qsub -I -X -l walltime=2:00:00,nodes=1
$ cd $PBS_O_WORKDIR
$ cd write_read
$ make
$ aprun -n 12 ./writer
  ts= 0
  ts= 1
$ ls -l *.bp
  -rw-r--r-- 1 esimmon esimmon 815379 2013-06-12 09:44 writer00.bp
  -rw-r--r-- 1 esimmon esimmon 815628 2013-06-12 09:44 writer01.bp
```
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
ADIOS Hands on: Tools

• Goals
  – Learn how to look at an ADIOS-BP file
  – Learn how to convert the data to HDF5 and NetCDF-3 files
ADIOS Tools

• **bpls**
  – Similar to h5dump+h5ls and ncdump
  – Also shows array min/max values
  – Metadata performance independent of data size

• **bp2h5, bp2ncd**
  – Convert BP format into HDF5 or NetCDF
$ module load adios/1.5.0

$ bplS -lv writer00.bp

File info:
- of groups: 1
- of variables: 7
- of attributes: 0
- time steps: 0 - 0
- file size: 796 KB
- bp version: 1
- endianness: Little Endian
- statistics: Min / Max / Avg / Std_dev

Group writer:
- integer /nx_global scalar = 260
- integer /ny_global scalar = 387
- integer /offs_x scalar = 0
- integer /offs_y scalar = 0
- integer /nx_local scalar = 65
- integer /ny_local scalar = 129
- double /xy {387, 260} = 0 / 11 / 5.5 / 3.45205
Use bpls to read in a 2D slice

$ bpls writer00.bp -d xy -s "128,64" -c "2,2" -n 2

double   /xy   {387, 260}
slice (128:129, 64:65)
   (128,64)   0 1
   (129,64)   4 5

- **-d var**
  - dump var
- **-s / -start**
  - define beginning offset
- **-c / -count**
  - define size in each dimension
- **-n 2**
  - print 2 values per row
bp2h5, bp2ncd

$ bp2ncl writer00.bp
$ ncdump -h writer00.nc

```plaintext
netcdf writer00 {
  dimensions:
    nx_global = 260 ;
    ny_global = 387 ;
    nx_local = 65 ;
    ny_local = 129 ;
    offs_x = 65 ;
    offs_y = 129 ;
  variables:
    double xy(nx_global, ny_global) ;
}
```

$ bp2h5 writer00.bp writer00.h5
$ h5ls writer00.h5

```
   nx_global        Dataset {1}
   nx_local         Dataset {1}
   ny_global        Dataset {1}
   ny_local         Dataset {1}
   offs_x           Dataset {1}
   offs_y           Dataset {1}
   xy               Dataset {387, 260}
```
Block layout of the example

• map.c
  – uses ADIOS API to inquire the local dimensions and offsets of each writer

```bash
$ ./map writer00.bp
...
double /xy[387, 260]: min=0 max=11
step 0:
  block 0: [ 0:128, 0: 64]
  block 1: [ 0:128, 65:129]
  block 2: [ 0:128, 130:194]
  block 3: [ 0:128, 195:259]
  block 4: [129:257, 0: 64]
  block 5: [129:257, 65:129]
  block 6: [129:257, 130:194]
  block 7: [129:257, 195:259]
  block 8: [258:386, 0: 64]
  block 9: [258:386, 65:129]
  block 10: [258:386, 130:194]
  block 11: [258:386, 195:259]
```

slow dimension, fast dimension from:to, from:to
ADIOS Componentization

• ADIOS can allow many different I/O methods
  – POSIX
  – POSIX1
  – MPI_AGGREGATE: needs num_ost, and num_aggregators
    <transport group="writer" method="MPI_AGGREGATE">num_aggregators=4;num_ost=2</transport>
  – PHDF5:
    • Limited functionality in ADIOS 1.5, but will be improved in future releases.
    • Must remove attributes and PATHS for this to work
  – NC4: same expectations as PHDF5

• Rule of thumb:
  – Try Posix, then move to MPI, then MPI_AGGREGATE
Outline

- ADIOS Introduction
- ADIOS Write API
- Hands-on 1, Write data with ADIOS
- Hands-on 1, Tools
- **ADIOS Read API**
- Hands-on 1, Read data with ADIOS
- Hands-on 1++, Spatial aggregation
- Hands-on 2, ADIOS Write API (Non-XML version)
- Hands-on 2, Multi-block writing with non-XML API
- Hands-on 3, Staging example
- Hands-on 4, Visualization of ADIOS data
- Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
- Hands-on 8 Python
- Hands-on 9, Java
- Summary
Read API basics

- Common API for reading files and streams (with staging)
  - In staging, one must process data step-by-step
  - Files allow for accessing all steps at once
- Schedule/perform reads in bulk, instead of single reads
  - Allows for optimizing multiple reads together
- Selections
  - bounding boxes, list of points, selected blocks and automatic
- Chunking (optional)
  - receive and process pieces of the requested data concurrently
  - staging delivers data from many producers to a reader over a certain amount of time, which can be used to process the first chunks
Read API basics

• **Step**
  – A dataset written within one adios_open/.../adios_close

• **Stream**
  – A file containing of, or a staged, series of steps of the same dataset
  – not a byte stream!
  – the step is quite a large unit

• **Read API is designed to read data from one step at a time, then advance forward**
  – alternative API allows for reading all steps at once from a file
Read API

• Initialization/Finalization
  – One call per each read method used in an application
  – Staging methods usually connect to a staging server / other application at init, and disconnect at finalize.

```c
int adios_read_init_method (enum ADIOS_READ_METHOD method, MPI_Comm comm, const char * parameters);
```

```c
int adios_read_finalize_method (enum ADIOS_READ_METHOD method);
```

**Fortran**

```fortran
use adios_read_mod
call adios_read_init_method (method, comm, parameters, err)
call adios_finalize_method (method, err)
```
Read API

- Open as a stream or as a file
  - for step-by-step reading (both staged data and files)
    ADIOS_FILE * adios_read_open (const char * fname, enum ADIOS_READ_METHOD method, MPI_Comm comm, enum ADIOS_LOCKMODE lock_mode, float timeout_sec)

  - for seeing all timesteps at once (files only!)
    ADIOS_FILE * adios_read_open_file (const char * fname, enum ADIOS_READ_METHOD method, MPI_Comm comm)

- Close
  int adios_read_close (ADIOS_FILE *fp)

Fortran
use adios_read_mod
call adios_read_open (fp, fname, method, comm, lockmode, timeout_sec, err)
call adios_read_open_file (fp, fname, method, comm, err)
call adios_read_close (fp, err)
Locking options

• ALL
  – lock current and all future steps in staging
  – ensures that reader can read all data
  – reader’s priority, it can block the writer

• CURRENT
  – lock the current step only
  – future steps can disappear if writer pushes more newer steps and staging needs more space
  – writer’s priority
  – reader must handle skipped steps

• NONE
  – no assumptions, anything can disappear between two read operations
  – be ready to process errors
Advancing a stream

• One step is accessible in streams, advancing is only forward
  
  int adios_advance_step (ADIOS_FILE *fp, int last, float timeout_sec)
  
  – last: advance to “next” or to latest available
    • “next” or “next available” depends on the locking mode
    • locking = all: go to the next step, return error if that does not exist anymore
    • locking = current or none: give the oldest, still available step after the current one
  
  – timeout_sec: block for this long if no new steps are available

• Release a step if not needed anymore
  
  – optimization to allow the staging method to deliver new steps if available
  
  int adios_release_step (ADIOS_FILE *fp)
Reading data

• Read is a scheduled request, ...
  
  int64_t adios_schedule_read ( 
  const ADIOS_FILE * fp, 
  const ADIOS_SELECTION * selection, 
  const char * varname, 
  int from_steps, 
  int nsteps, 
  void * data) 

  – in streaming mode, only one step is available

• ...executed later with other requests together
  
  int adios_perform_reads (const ADIOS_FILE *fp, int blocking)

Fortran
call adios_schedule_read ( 
  fp, selection, varname, 
  from_steps, nsteps, data, err)
call adios_perform_reads ( 
  fp, err)
Inquire about a variable (no extra I/O)

- **ADIOS_VARINFO** * adios_inq_var (ADIOS_GROUP *gp, const char * varname)
  - Inquiry about a variable in a group.
  - Value of a scalar variable can be retrieved with this function instead of reading.
  - This function does not read anything from file but processes info already in memory after open.
  - It allocates memory for the ADIOS_VARINFO struct and content, so you need to free resources later with adios_free_varinfo().
- ADIOS_VARINFO variables
  - int ndim Number of dimensions
  - uint64_t *dims Size of each dimension
  - int nsteps Number of steps of the variable in file. Streams: always 1
  - void *value Value (for scalar variables only)
  - int nblocks Number of blocks that comprise this variable in a step
  - void *gmin, *gmax, gavg, gstd_dev Statistical values of the global arrays
  - see adios_read_v1.h for more information
- int adios_inq_var_stat (ADIOS_FILE *fp, ADIOS_VARINFO * varinfo, int per_step_stat, int per_block_stat)
  - Get statistics about an array variable (min, max, avg, std_dev)
    - globally, per step, per written block
- int adios_inq_var_blockinfo (ADIOS_FILE *fp, ADIOS_VARINFO * varinfo)
  - Get writing layout of an array variable (bounding boxes of each writer)

**Fortran**
call adios_get_scalar (fp, varname, data, err)
call adios_inq_file (fp, vars_count, attrs_count, current_step, last_step, err)
call adios_inq_varnames (fp, vnamelist, err)
call adios_inq_var (fp, varname, vartype, nsteps, ndim, dims, err)
Read an attribute (no extra I/O)

- int adios_get_attr (ADIOS_FILE * fp,
  const char * attrname,
  enum ADIOS_DATATYPES * type,
  int * size,
  void ** data)

  - Attributes are stored in metadata, read in during open operation.
  - It allocates memory for the content, so you need to free it later with free().

Fortran

call adios_inq_attr (fp, attrname, attrtype, attrsize, err)
Select a bounding box

• In our example we need to define the selection area of what to read from an array.

• ADIOS_SELECTION * adios_selection_boundingbox (int ndim, uint64_t * offsets, uint64_t * readsize)

Fortran

call adios_selection_boundingbox (integer*8 sel, -- return value
     integer ndim, integer*8 offsets(:), integer*8 readsize(:))
Outline

- ADIOS Introduction
- ADIOS Write API
- Hands-on 1, Write data with ADIOS
- Hands-on 1, Tools
- ADIOS Read API
- Hands-on 1, Read data with ADIOS
- Hands-on 1++, Spatial aggregation
- Hands-on 2, ADIOS Write API (Non-XML version)
- Hands-on 2, Multi-block writing with non-XML API
- Hands-on 3, Staging example
- Hands-on 4, Visualization of ADIOS data
- Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
- Hands-on 8 Python
- Hands-on 9, Java
- Summary
ADIOS Hands on: Read

• Goals
  – Learn how to read in data from an arbitrary number of processors.
Compile and run the read code

- We can read in data from 1 – 260 processors with a 1D domain decomposition

```bash
$ cd ../write_read
$ make
$ mpirun -n 1 ./reader
$ ls fort.*; tail -n 4 fort.100
  fort.100
  1 256 386 12.0
  1 257 386 12.0
  1 258 386 12.0
  1 259 386 12.0

$ mpirun -n 7 ./reader
$ ls fort.*; tail -n 4 fort.100
  fort.100 fort.101 fort.102 fort.103 fort.104 fort.105 fort.106
  1 33 386 9.0
  1 34 386 9.0
  1 35 386 9.0
  1 36 386 9.0
```

Each line contains:
- timestep
- x (global)
- y (global)
- xy
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
Hierarchical Spatial Aggregation

• Small chunks need to be consolidated to reduce seek overhead
• Simple concatenation doesn’t change the number of seeks
• Spatial Aggregation with hierarchical topology merges spatially adjacent chunks into one
  – Spatial locality of every data point is reserved
  – Writing: less writers, less contention at storage during output
  – Reading: improve read performance for common spatial access patterns

Original 16 chunks

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

HSA to 4 Processes (0, 2, 8, 10)

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing the consolidation process from 16 chunks to 4 processes.
VAR_MERGE transport method

• Applies Hierarchical Spatial Aggregation to consolidate small data variables

• It can/should be combined with other ADIOS methods
  – to achieve good I/O performance on a specific system

• Supports up to three dimensional variable with any kind of domain decomposition

• Currently supports up to 2 levels of aggregation
  – One level merges $2^n$ processes (for n-dimensional variables)
  – Merging 64 chunks into 1 for a 3-D variable with 3-D domain decomposition in two steps
How to use VAR_MERGE

• chunk_size: the minimum of merged chunk size.
  – For example, for a 3-D variable with 3-D domain decomposition, spatial aggregation will be applied if the chunk_size=1MB, and each process has a chunk <128KB (1024KB/8).
  – Default chunk_size is 2MB.
• io-method: the underlying I/O transport method.
  – Default is ADIOS MPI method.
• io_parameters: the parameters required for using the specified “io_method”
  – Must be a comma-separated string of options, not a semicolon-separated one

```
<method group="genarray" method="VAR_MERGE">
  chunk_size=1048576;io_method=MPI_LUSTRE;
  io_parameters=stripe_count=4,stripe_size=1048576</method>
```
VAR_MERGE on write_read example

• Only aggregator processes write the variable to file

<transport group="writer" method="VAR_MERGE">io_method=MPI
</transport>

• 2 merges:
  – 12 → 4 then 4 → 1

<transport group="writer" method="VAR_MERGE">
  chunk_size=300000;io_method=MPI
</transport>

• 1 Level of merge
  – 12 → 4
    – Due to the limited buffer in the chunk_size

$ cd ../write_read
$ vi writer.xml
$ mpirun -np 12 ./writer
$ ./map writer00.bp
...
double /xy[387, 260]: min=0 max=11
  step 0:
    block 0: [ 0:386, 0:259]

$ mpirun -np 12 ./writer
$ ./map writer00.bp
...
double /xy[387, 260]: min=0 max=11
  step 0:
    block 0: [ 0:257, 0:129]
    block 1: [ 0:257, 130:259]
    block 2: [258:386, 0:129]
    block 3: [258:386, 130:259]
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
ADIOS non-XML APIs

• Limitation of the XML approach
  – Must have all variables defined in advance

• Approach of non-XML APIs
  – Similar operations to what happens internally in adios_init
  – Define variables and attributes before opening a file for writing
  – The writing steps are the same as XML APIs
    • open file
    • set group size (size the given process will write)
    • write variables
    • close file
Non-XML API functions

• Initialization
  – init adios, allocate buffer, declare groups and select write methods for each group.

  adios_init_noxml ();
  adios_allocate_buffer (ADIOS_BUFFER_ALLOC_NOW, 10);
  – when and how much buffer to allocate (in MB)
  adios_declare_group (&group, "restart", "iter", adios_flag_yes);
  – group with name and optional timestep indicator (iter) and whether statistics should be generated and stored
  adios_select_method (group, "MPI", "", "");
  – with optional parameter list, and base path string for output files
Non-XML API functions

• Definition

```c
int64_t adios_define_var (group, "name", "path", type,
    "local_dims", "global_dims", "offsets")
```

– Similar to how we define a variable in the XML file
– returns a handle to the specific definition

• Dimensions/offsets can be defined with

  – scalars (as in the XML version)
    • `id = adios_define_var (g, "xy", ",", adios_double,
      "nx_global, ny_global",
      "nx_local, ny_local",
      "offs_x,offs_y")`
    • need to define and write several scalars along with the array

  – actual numbers
    • `id = adios_define_var (g, "xy", ",", adios_double,
      "100,100", "20,20", "0,40")`
Multiple blocks per process

• AMR codes and load balancing strategies may want to write multiple pieces of a global variable from a single process

• ADIOS allows one to do that but
  – one has to write the scalars defining the local dimensions and offsets for each block, and
  – group size should be calculated accordingly
  – This works with the XML API, too, but because of the group size issue, pre-generated write code cannot be used (should do the adios_write() calls manually)
  – Array definition with the actual sizes as numbers saves from writing a lot of scalars (and writing source code)
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
ADIOS Hands on Non-XML Write API

• Goals
  – Use Non-XML API of ADIOS
  – How to write multiple blocks of a variable from a process
Multi-block example (Fortran)

- Here we repeat the first tutorial example, except that each process writes 2 blocks (2\textsuperscript{nd} block shifted in the X offset).
  - 12 cores, arranged in a 4 x 3 arrangement.
  - 24 data blocks

- Each processor allocates an 65x129 array (xy)
  - we write the same array to two places in the output

- Total size of the array
  - 2*4*65 x 3*129

- Use the non-XML API
  - define and write xy array without scalars
noxml example

$ cd ../noxml_F90
$ make
$ mpirun -np 12 ./writer_noxml
  ts= 0
ts= 1
$ ls -l *.bp
-rw-r--r-- 1 esimmon esimmon 1618482 2013-06-12 09:44 writer00.bp
-rw-r--r-- 1 esimmon esimmon 1618731 2013-06-12 09:44 writer01.bp
$ bplsl -l writer00.bp
  double /xy {387, 520} = 0 / 11 / 5.5 / 3.45205
$ bplsl -l writer01.bp
  double /xy {387, 520} = 1 / 12 / 6.5 / 3.45205
Block layout of the example

- map.c
  - uses ADIOS API to inquire the local dimensions and offsets of each writer

```
$ ../write_read/map writer00.bp
.. /write_read/map writer00.bp
... double /xy[387, 520]: min=0 max=11
    step 0:
        block 0: [ 0:128, 0: 64]
        block 1: [ 0:128, 65:129]
        block 2: [ 0:128, 130:194]
        block 3: [ 0:128, 195:259]
        block 4: [129:257, 0: 64]
        block 5: [129:257, 65:129]
        block 6: [129:257, 130:194]
        block 7: [129:257, 195:259]
        block 8: [258:386, 0: 64]
        block 9: [258:386, 65:129]
        block 10: [258:386, 130:194]
        block 11: [258:386, 195:259]
```

```
$ ../write_read/map ../write_read/writer00.bp
... double /xy[387, 260]: min=0 max=11
    step 0:
        block 0: [ 0:128, 0: 64]
        block 1: [ 0:128, 65:129]
        block 2: [ 0:128, 130:194]
        block 3: [ 0:128, 195:259]
        block 4: [129:257, 0: 64]
        block 5: [129:257, 65:129]
        block 6: [129:257, 130:194]
        block 7: [129:257, 195:259]
        block 8: [258:386, 0: 64]
        block 9: [258:386, 65:129]
        block 10: [258:386, 130:194]
        block 11: [258:386, 195:259]
```
noxml example

Quit the interactive job here. We will run the next example as a batch job.

$ exit
$ module load adios/1.5.0
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• ADIOS + Matlab
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
ADIOS Hands on

• Goals
  – Couple the write and read example with a staging method
  – How to write the streaming read code.
  – How to run a staging code.
Coupling Example

- The first write and read examples running together
  - We write out 5 time-steps, into the same file(name).
  - Write from 12 cores, arranged in a 4 x 3 arrangement.
  - Read from 3 cores, arranged as 1x3
coupling example

```bash
$ cd ../coupling
$ make
$ qsub job.chester
-- wait for who knows how long --
-- commands: showq, showstart, showbf
$ ls -l *.bp
-rw-r--r-- 1 pnorbert ccsstaff 1615244 Aug  4 09:56 reader_001.bp
-rw-r--r-- 1 pnorbert ccsstaff 1615493 Aug  4 09:56 reader_002.bp
-rw-r--r-- 1 pnorbert ccsstaff 1615742 Aug  4 09:56 reader_003.bp
-rw-r--r-- 1 pnorbert ccsstaff 1615991 Aug  4 09:56 reader_004.bp
-rw-r--r-- 1 pnorbert ccsstaff 1616240 Aug  4 09:56 reader_005.bp
$ bpls -l reader_001.bp
  integer /info/rank  scalar = 0
  integer /gdx  scalar = 260
  integer /gdy  scalar = 387
  integer /ldx  scalar = 86
  integer /ldy  scalar = 387
  integer /ox  scalar = 0
  integer /oy  scalar = 0
  double /xy  {387, 260} = 0 / 11 / 5.5 / 3.45205
  double /xy2 {387, 260} = 0 / 2.55 / 1.28269 / 0.884548
```
Previous Approaches for Sharing Data between Coupled Applications

Disk files
+ asynchronous communication
+ easy, commonly-used APIs
- affected by parallel IO performance

Coupler approach
+ asynchronous communication
+ data aggregation/transformation at the coupler
- centralized coupler(s) can become a bottleneck

Direct parallel data transfer (DIMES, FLEXPATH)
+ fast and scalable data movement
- programming complexity (hand coded)
- need for asynchronous interactions
Coupling codes with ADIOS+staging method

ADIOS + DataSpaces/DIMES/FLEXPATH
+ asynchronous communication
+ easy, commonly-used APIs
+ fast and scalable data movement
+ not affected by parallel IO performance
- data aggregation/transformation at the coupler

Fusion code coupling (CPES project)

Interactive visualization pipeline of fusion simulation, analysis code and parallel viz. tool
Code which reads file data

call adios_read_init_method (ADIOS_READ_METHOD_BP, group_comm, "", ierr);
call adios_read_open_stream (inh, infn, 0, group_comm,
    ADIOS_LOCKMODE_CURRENT, 60.0, ierr)
do while (ierr != err_stream_terminated)
    call adios_selectionBoundingbox (sel, 0, offset, readsize)
    call adios_schedule_read (inh, sel, "gdx", 1, 1, gdx, ierr)
    call adios_schedule_read (inh, sel, "gdy", 1, 1, gdy, ierr)
    call adios_perform_reads (inh, ierr)
    ! ... calculate offsets and sizes of xy to read in...
    call adios_selectionBoundingbox (sel, 2, offset, readsize)
    call adios_schedule_read (inh, sel, "xy", 1, 1, xy, ierr)
    call adios_perform_reads (inh, ierr)
    call adios_advance_step (inh, 0, 60.0, ierr)
endo
call adios_read_close (inh, ierr)
ADIOS Read API for streams

Code which reads stream data

call adios_read_init_method (ADIOS_READ_METHOD_DATASPACES, group_comm,"",ierr);
call adios_read_open_stream (inh, infn, 0, group_comm,
                    ADIOS_LOCKMODE_CURRENT, 60.0, ierr)
do while (ierr != err_stream_terminated)
    call adios_selection_boundingbox (sel, 0, offset, readsize)
call adios_schedule_read (inh, sel, "gdx", 1, 1, gdx, ierr)
call adios_schedule_read (inh, sel, "gdy", 1, 1, gdy, ierr)
call adios_perform_reads (inh, ierr)
    ! ... calculate offsets and sizes of xy to read in...
call adios_selection_boundingbox (sel, 2, offset, readsize)
call adios_schedule_read (inh, sel, "xy", 1, 1, xy, ierr)
call adios_perform_reads (inh, ierr)
call adios_advance_step(inh, 0, 60.0, ierr)
enddo
call adios_read_close (inh, ierr)
Outline

- ADIOS Introduction
- ADIOS Write API
- Hands-on 1, Write data with ADIOS
- Hands-on 1, Tools
- ADIOS Read API
- Hands-on 1, Read data with ADIOS
- Hands-on 1++, Spatial aggregation
- Hands-on 2, ADIOS Write API (Non-XML version)
- Hands-on 2, Multi-block writing with non-XML API
- Hands-on 3, Staging example
- Hands-on 4, Visualization of ADIOS data
- Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
- Hands-on 8 Python
- Hands-on 9, Java
- Summary
Skel: Automatic generation of I/O Skeletal Benchmarks

- Skeletal applications perform the same I/O operations as an application, but eliminate computation and communication
- Created from ADIOS XML file and a handful of additional parameters
- Easy to create, run, and update, compared to instrumented apps or I/O kernels
Skel: Users and Impact

- Extensible collection of relevant I/O benchmarks
- Focus less on measuring and more on improving I/O performance
- Create a comprehensive and relevant set of I/O benchmarks for new or existing hardware platforms
- Demonstrated to work well with these applications (so far):
  - S3D (Combustion)
  - GTS (Fusion)
  - CHIMERA (Astrophysics)
  - GRAPES (Weather)
  - GEOS-5 (Climate)

1 app, 2 machines, 3 methods
Skel Tutorial

# Begin with the XML file for the application
$ cd ../skel/
$ mkdir –p grapes
$ cd grapes
$ cp ../grapes.xml .
$ ls
  grapes.xml

# First, generate the XML file for the skeletal app.
# just to filter out from xml file what would not work with skel
$ skel xml grapes
# skel sub_command project_name
$ ls
  grapes_skel.xml  grapes.xml
# Now create a default parameter file

$ skel params grapes

$ ls

  grapes_params.xml.default  grapes_skel.xml  grapes.xml

# Copy the default parameters (this makes it more difficult to accidentally overwrite your settings)

$ cp grapes_params.xml.default grapes_params.xml
Skel Tutorial

# Edit the parameters
$v$ vi grapes_params.xml

# “target” defines the platform you are running on.
# <scalars> are used to define array sizes
# <arrays> can be initialized randomly or with specific
# patterns (helpful for debugging)
# <tests> allow multiple runs using different methods

# or, just use a prepared run
$cp$ ../grapes_params.xml .
Skel Tutorial

# Generate the Makefile
$ skel makefile grapes

# Generate the source file
$ skel source grapes

# look at the generated code
$ vi grapes_skel_post_process_write.f90

# Build the skeletal application
$ make
# executable: grapes_skel_post_process_write
$ mpirun -np 4 ./grapes_skel_post_process_write

**************************
  Groupsize: 375892
  Open Time: 1.88302893774414063E-003
  Access Time: 8.32605361938476563E-003
  Close Time: 9.05203819274902344E-003
  Total Time: 1.54860019683837891E-002
**************************

# output file: out_post_process_write_1
Outline

• ADIOS Introduction
• ADIOS Write API
• Hands-on 1, Write data with ADIOS
• Hands-on 1, Tools
• ADIOS Read API
• Hands-on 1, Read data with ADIOS
• Hands-on 1++, Spatial aggregation
• Hands-on 2, ADIOS Write API (Non-XML version)
• Hands-on 2, Multi-block writing with non-XML API
• Hands-on 3, Staging example
• Hands-on 4, Visualization of ADIOS data
• Hands-on 5, I/O skeleton generation with Skel
• **ADIOS + Matlab**
• Hands-on 8 Python
• Hands-on 9, Java
• Summary
• Use bpls to read in a 2D slice

```
$ bpls writer00.bp -d xy -s "128,64" -c "2,2" -n 2
```

double /xy {387, 260}

slice (128:129, 64:65)
(128,64) 0 1
(129,64) 4 5

• How do we the same in Matlab?
Matlab startup on Chester

$ module load matlab
$ matlab -nodesktop –nosplash

< M A T L A B (R) >
Copyright 1984-2012 The MathWorks, Inc.
R2012b (8.0.0.783) 64-bit (glnxa64)
August 22, 2012

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>>
Matlab

• Matlab is column-major, bpls (which is C) is row-major
  – 128, 64 → 64, 128
• Matlab array indices start from 1 (bpls/C starts from 0)
  – 64, 128 → 65, 129

```matlab
>> data=adiosread('writer00.bp','xy','Slice',[65 2; 129 2])
data =
  0   4
  1   5
```

```matlab
>> f=adiosopen('writer00.bp');
>> data=adiosread(f.Groups,'/xy');
>> whos data
Name      Size    Bytes  Class          Attributes
data     260x387  804960 double

>> adiosclose(f);
```
Matlab API functions

• Open
  
  INFO = ADIOSOPEN (FILE);
  INFO = ADIOSOPEN (FILE, 'Verbose', LEVEL)
  • see definition in matlab: help adiosopen

• Close
  
  ADIOSCLOSE (STRUCT)
  • STRUCT is the return value of ADIOSOPEN

• Read from an opened file
  
  DATA = ADIOSREAD (STRUCT, VARPATH)
  • STRUCT is the return value of ADIOSOPEN

• Read from an unopened file
  
  DATA = ADIOSREAD (FILE, VARPATH)
  • File is the path string here
function reader (file)
    % f=adiosopen(file);
    f=adiosopen(file, 'Verbose',0);

    % list metadata of all variables
    for i=1:length(f.Groups.Variables)
        f.Groups.Variables(i)
    end

    % read in the data of xy
    data=adiosread(f.Groups,'/xy');

    adiosclose(f)

    % export the last variable in the file as 'xy' in matlab
    assignin('base','xy',data);

    % check out the variable after the function returns
    % whos('xy')
>> reader('writer00.bp');
...
ans =
    Name: '/xy'
    Type: 'double'
    Dims: [260 387]
    Timedim: 0
    GlobalMin: 0
    GlobalMax: 11

>> whos
Name        Size            Bytes  Class       Attributes
xy    260x387         804960  double

>> imagesc(xy)
Outline

- ADIOS Introduction
- ADIOS Write API
- Hands-on 1, Write data with ADIOS
- Hands-on 1, Tools
- ADIOS Read API
- Hands-on 1, Read data with ADIOS
- Hands-on 1++, Spatial aggregation
- Hands-on 2, ADIOS Write API (Non-XML version)
- Hands-on 2, Multi-block writing with non-XML API
- Hands-on 3, Staging example
- Hands-on 4, Visualization of ADIOS data
- Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
  - Hands-on 8 Python
- Hands-on 9, Java
- Summary
ADIOS Python Wrapper Build

• Prerequisite modules and tools
  – NumPy: array and matrix representation
  – MPI4Py: MPI for Python
  – Cython: provide C/C++ interface to Python
  – Cmake: makefile generator

• Build
  – Run cmake which will detect python libraries and generate Makefile
    $ cmake /dir/to/adios-source/wrapper/numpy
  – Type make which will make adios.so
    $ make
  – Copy adios.so where python can access (E.g., ~/.local/lib/python2.6/site-packages)

• Use in Python
  >>> import adios
Writer Example

• Use the same Fortran example (01_write_read)

• Import necessary modules

```python
import numpy as np
from mpi4py import MPI
import adios

```

• Use NUMPY for array/matrix

```python
xy = np.zeros((ndx, ndy), dtype=np.float64)
```

• Call adios write functions

```python
fd = adios.open("writer", filename, "w")
adios.set_group_size(fd, groupsize)
adios.write_int(fd, "nx_global", nx_global)
...
adios.write(fd, "xy", xy)
adios.close(fd)
```
Python Write API functions

- Init and finalize
  ```python
  Init(CONFIG);
  finalize()
  ```
  - CONFIG is a filename for Adios XML config file

- Open & Close
  ```python
  FOBJ = open(GRPNAME, FILENAME, MODE);
  close (FOBJ)
  ```
  - FOBJ is adios file object

- Write to an opened file
  ```python
  write_int (FOBJ, VARNAME, VALUE)
  write (FOBJ, VARNAME, NUMPY_ARR)
  ```
  - NUMPY_ARR is a NumPy array/matrix object
• We will get the same result with MPI

$ mpirun -np 12 ./writer.py

• Use bpls to read in a 2D slice

$ bpls writer00.bp -d xy -s "128,64" -c "2,2" -n 2
double /xy {387, 260}
slice (128:129, 64:65)
(128,64) 0 1
(129,64) 4 5
Read Example

• Import necessary modules

```python
import numpy as np
from mpi4py import MPI
import adios
```

• Call adios read functions

```python
f = adios.AdiosFile(filename)
g = f.group['writer']
## Scalar reading
nx_global = g.var['/nx_global'].read().item()
## Array/Matrix reading
xy = g.var['/xy'].read(offset, offset + count)
f.close()
```

• read() will return NumPy object (array/matrix)
read

- Run with MPI

$ mpirun –np 3 ./reader.py

- Each process writes python.NNN (NNN is rank)
- Full matrix (387-by-260) will be decomposed by column-wise
Python Read API functions

- **File Open & Close**
  
  \[
  
  \text{FOBJ} = \text{AdiosFile}(\text{FILENAME});
  \]
  
  \[
  \text{FOBJ.close()}
  \]
  
  - FOBJ is adios file object

- **Group open**
  
  \[
  \text{GOBJ} = \text{FOBJ.group}[\text{GRPNAME}]
  \]
  
  - GOBJ is adios group object
  - GRPNAME is a group name

- **Read variables**
  
  \[
  \text{VOBJ} = \text{GOBJ.var}[\text{VARNAME}]
  \]
  
  \[
  \text{NUMPY.ARR} = \text{VOBJ.read()}
  \]
  
  - VARNAME is a variable name
  - VOBJ is Adios variable object
  - NUMPY.ARR is a NumPy array/matrix object
Outline

- ADIOS Introduction
- ADIOS Write API
- Hands-on 1, Write data with ADIOS
- Hands-on 1, Tools
- ADIOS Read API
- Hands-on 1, Read data with ADIOS
- Hands-on 1++, Spatial aggregation
- Hands-on 2, ADIOS Write API (Non-XML version)
- Hands-on 2, Multi-block writing with non-XML API
- Hands-on 3, Staging example
- Hands-on 4, Visualization of ADIOS data
- Hands-on 5, I/O skeleton generation with Skel
- ADIOS + Matlab
- Hands-on 8 Python
- **Hands-on 9, Java**
- Summary
ADIOS Java Wrapper Build

• Prerequisite modules and tools
  – Java JDK: need Java Native Interface (JNI)
  – Cmake: makefile generator

• Build
  – Run cmake which will detect Java libraries and generate Makefile
    $ cmake /dir/to/adios-source/wrapper/java
  – Type make which will make libAdiosJava.so and AdiosJava.jar
    $ make
  – Copy libAdiosJava.so and AdiosJava.jar to a shared location (E.g., $ADIOS_DIR/lib)

• Use in Java
  import gov.ornl.ccs.Adios;    // For Writer
  import gov.ornl.ccs.AdiosFile; // For Reader
  import gov.ornl.ccs.AdiosGroup;
  import gov.ornl.ccs.AdiosVarinfo;
Java Write API functions

- **MPI Init and finalize**
  
  ```
  MPI_Init();
  MPI_Finalize()
  ```
  
  - Adios Java wrapper contains own MPI init and finalize routine

- **Adios Init and finalize**
  
  ```
  Init()
  Finalize(ID)
  ```
  
  - ID is a rank

- **Open & Close**
  
  ```
  H = open(GRPNAME, FILENAME, MODE);
  close (H)
  ```
  
  - H is adios file handler

- **Write to an opened file**
  
  ```
  Write (H, VARNAME, VAL)
  ```
  
  - Overloaded. VAL can be scalar or array (int, double, byte, ...)

Writer Example

• Same tasks with Fortran example (01_write_read)
• Import necessary class

```java
import gov.ornl.ccs.Adios;
import gov.ornl.ccs.AdiosFile;
import gov.ornl.ccs.AdiosGroup;
import gov.ornl.ccs.AdiosVarinfo;
```

• Call adios write functions

```java
long adios_handle = Adios.Open("writer",
    String.format("writer%02d.bp", ts), "w", comm);
Adios.SetGroupSize (adios_handle, groupsize);

Adios.Write (adios_handle, "nx_global", nx_global);
...
Adios.Write (adios_handle, "xy", xy);
Adios.Close (adios_handle);
```
mpiexec and bplS

- We will get the same result with MPI

```sh
$ mpiexec -n 12 java -Djava.library.path=/opt/adios/1.5.0/lib -classpath /opt/adios/1.5.0/lib/AdiosJava.jar:. Writer
```

  - Add AdiosJava.jar in the classpath
  - Specify the directory contains libAdiosJava.so (shared lib) in java.library.path

- Use bplS to read in a 2D slice

```sh
$ bplS writer00.bp -d xy \  
-s "128,64" -c "2,2" -n 2
```

```sh
double /xy {387, 260}  
slice (128:129, 64:65)  
(128,64) 0 1  
(129,64) 4 5
```
Java Read API functions

• File open & close
  AdiosFile FH = new AdiosFile();
  FH.open(FILENAME, COMM);
  FH.close()
  • H is adios file handler

• Group open & close
  AdiosGroup GH = new AdiosGroup(FH);
  GH.group(GRPNAME)
  • GH is adios group handler
  • GRPNAME is a group name

• Read variables
  AdiosVarinfo VH = new AdiosVarinfo(GH);
  VH.inq(VARNAME)
  VH.readIntValue()
  VH.read(OFFSET, COUNT, ARR)
  • VARNAME is a variable name
  • VH is Adios variable handler
  • After inq(), call readIntValue() for scalar or read() for array
Read Example

• Import necessary library

```java
import gov.ornl.ccs.AdiosFile;
import gov.ornl.ccs.AdiosGroup;
import gov.ornl.ccs.AdiosVarinfo;
```

• Open file, group, and varinfo

```java
AdiosFile file = new AdiosFile();
file.open(String.format("writer%02d.bp", ts), comm);
AdiosGroup group = new AdiosGroup(file);
group.open("writer");
AdiosVarinfo var1 = new AdiosVarinfo(group);
var1.inq("nx_global");
```

• Read scalar or array

```java
int nx_global = var1.readIntValue();
double[] xy = new double[...];
var3.read(offset, readsize, xy);
```
mpiexec and bplS

• Run with MPI

$ mpiexec -n 3 java -Djava.library.path=/opt/adios/1.5.0/lib -classpath /opt/adios/1.5.0/lib/AdiosJava.jar:. Reader

  – Add AdiosJava.jar in the classpath
  – Specify the directory contains libAdiosJava.so (shared lib) in java.library.path

• Each process writes java.NNN (NNN is rank)

• Full matrix (387-by-260) will be decomposed by column-wise
Summary

• ADIOS is an abstraction for data pipelines
  – Over 60 publications
• Typically the ADIOS team creates new I/O methods when applications require them
  – Fusion applications (Particle-in-cell, MHD)
  – Combustion (DNS, LES, AMR)
  – Astrophysics
  – Relativity
  – Climate
  – Weather
  – QCD
  – High Energy Physics
• ADIOS 1.5 includes staging services
  – I/O abstraction allows codes to be executed in staging area
• ADIOS 2.0 will be a new version of ADIOS using SOA methodologies