High Performance Parallel I/O: Software Stack as Babel fish

Rob Latham
Mathematics and Computer Science Division
Argonne National Laboratory
robl@mcs.anl.gov
Data Volumes in Computational Science

Data requirements for select 2012 INCITE applications at ALCF (BG/P)

<table>
<thead>
<tr>
<th>PI</th>
<th>Project</th>
<th>On-line Data (TBytes)</th>
<th>Off-line Data (TBytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb</td>
<td>Supernovae Astrophysics</td>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>Khokhlov</td>
<td>Combustion in Reactive Gases</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>Lester</td>
<td>CO2 Absorption</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Jordan</td>
<td>Seismic Hazard Analysis</td>
<td>600</td>
<td>100</td>
</tr>
<tr>
<td>Washington</td>
<td>Climate Science</td>
<td>200</td>
<td>750</td>
</tr>
<tr>
<td>Voth</td>
<td>Energy Storage Materials</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Vashista</td>
<td>Stress Corrosion Cracking</td>
<td>12</td>
<td>72</td>
</tr>
<tr>
<td>Vary</td>
<td>Nuclear Structure and Reactions</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>Fischer</td>
<td>Reactor Thermal Hydraulic Modeling</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Hinkel</td>
<td>Laser-Plasma Interactions</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Elghobashi</td>
<td>Vaporizing Droplets in a Turbulent Flow</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Top 10 data producer/consumers instrumented with Darshan over the month of July, 2011. Surprisingly, three of the top producer/consumers almost exclusively read existing data.
Dataset Complexity in Computational Science

Complexity is an artifact of science problems and codes:

- Coupled multi-scale simulations generate multi-component datasets consisting of materials, fluid flows, and particle distributions.
- Example: thermal hydraulics coupled with neutron transport in nuclear reactor design.
- Coupled datasets involve mathematical challenges in coupling of physics over different meshes and computer science challenges in minimizing data movement.

Images from T. Tautges (ANL) (upper left), M. Smith (ANL) (lower left), and K. Smith (MIT) (right).

Model complexity:
Spectral element mesh (top) for thermal hydraulics computation coupled with finite element mesh (bottom) for neutronics calculation.

Scale complexity:
Spatial range from the reactor core in meters to fuel pellets in millimeters.
Leadership System Architectures

Mira IBM Blue Gene/Q System

49,152 Compute Nodes (786,432 Cores)

384 I/O Nodes

QDR Infiniband
Federated Switch

QDR IB
1 port per analysis node

Tukey Analysis System

96 Analysis Nodes
(1,536 CPU Cores, 192 Fermi GPUs, 96 TB local disk)

16 Storage Couplets (DataDirect SFA12KE)

560 x 3TB HDD
32 x 200GB SSD

Post-processing, Co-analysis, In-situ analysis engage (or bypass) various components

High-level diagram of 10 Pflop IBM Blue Gene/Q system at Argonne Leadership Computing Facility
I/O for Computational Science

**High-Level I/O Library**
maps application abstractions onto storage abstractions and provides data portability.

*HDF5, Parallel netCDF, ADIOS*

**I/O Forwarding**
bridges between app. tasks and storage system and provides aggregation for uncoordinated I/O.

*IBM ciod*

---

**Application**

---

**High-Level I/O Library**

---

**I/O Middleware**

---

**I/O Forwarding**

---

**Parallel File System**

---

**I/O Hardware**

---

**I/O Middleware**
organizes accesses from many processes, especially those using collective I/O.

*MPI-IO*

**Parallel File System**
maintains logical space and provides efficient access to data.

*PVFS, PanFS, GPFS, Lustre*

---

Additional I/O software provides improved performance and usability over directly accessing the parallel file system. Reduces or (ideally) eliminates need for optimization in application codes.
I/O Hardware and Software on Blue Gene/P

High-level I/O libraries execute on compute nodes, mapping application abstractions into flat files, and encoding data in portable formats. **I/O middleware** manages collective access to storage.

**I/O forwarding** software runs on compute and gateway nodes, bridges networks, and provides aggregation of independent I/O.

**Parallel file system** code runs on gateway and storage nodes, maintains logical storage space and enables efficient access to data.

**Drive management** software or firmware executes on storage controllers, organizes individual drives, detects drive failures, and reconstructs lost data.

---

**Compute nodes**
40,960 Quad core PowerPC 450 nodes with 2 Gbytes of RAM each

**Gateway nodes**
640 Quad core PowerPC 450 nodes with 2 Gbytes of RAM each

**Commodity network**
900+ port 10 Gigabit Ethernet Myricom switch complex

**Storage nodes**
136 two dual core Opteron servers with 8 Gbytes of RAM each

**Enterprise storage**
17 DataDirect S2A9900 controller pairs with 480 1 Tbyte drives and 8 InfiniBand ports per pair

Architectural diagram of the 557 TFlop IBM Blue Gene/P system at the Argonne Leadership Computing Facility.
High-Level I/O libraries

- Parallel-NetCDF: http://www.mcs.anl.gov/parallel-netcdf
  - Parallel interface to NetCDF datasets
- HDF5: http://www.hdfgroup.org/HDF5/
  - Extremely flexible; earliest high-level I/O library; foundation for many others
- NetCDF-4: http://www.unidata.ucar.edu/software/netcdf/netcdf-4/
  - netCDF API with HDF5 back-end
- ADIOS: http://adiosapi.org
  - Configurable (xml) I/O approaches
- SILO: https://wci.llnl.gov/codes/silo/
  - A mesh and field library on top of HDF5 (and others)
- H5part: http://vis.lbl.gov/Research/AcceleratorSAPP/
  - Simplified HDF5 API for particle simulations
- GIO: https://svn.pnl.gov/gcrm
  - Targeting geodesic grids as part of GCRM
- PIO:
  - Climate-oriented I/O library; supports raw binary, parallel-netcdf, or serial-netcdf (from master)
- ... Many more: my point: it's ok to make your own.
Application-motivated library enhancements

- FLASH checkpoint I/O
- Write 10 variables (arrays) to file
- Pnetcdf non-blocking optimizations result in improved performance, scalability
- Wei-keng showed similar benefits to Chombo, GCRM
File Access Three Ways

No hints: reading in way too much data

With tuning: no wasted data; file layout not ideal

HDF5 & new pnetcdf: no wasted data; larger request sizes
Additional Tools

- **DIY**: analysis-oriented building blocks for data-intensive operations
  - Lead: Tom Peterka, ANL([tpeterka@mcs.anl.gov](mailto:tpeterka@mcs.anl.gov))
  - [www.mcs.anl.gov/~tpeterka/software.html](http://www.mcs.anl.gov/~tpeterka/software.html)

- **GLEAN**: library enabling co-analysis
  - Lead: Venkat Vishnawath, ANL([venkatv@mcs.anl.gov](mailto:venkatv@mcs.anl.gov))

- **Darshan**: insight into I/O access patterns at leadership scale
  - Lead: Phil Carns, ANL([pcarns@mcs.anl.gov](mailto:pcarns@mcs.anl.gov))
  - press.mcs.anl.gov/darshan
DIY Overview: Analysis toolbox

Main Ideas and Objectives
- Large-scale parallel analysis (visual and numerical) on HPC machines
- For scientists, visualization researchers, tool builders
- In situ, coprocessing, postprocessing
- Data-parallel problem decomposition
- MPI + threads hybrid parallelism
- Scalable data movement algorithms
- Runs on Unix-like platforms, from laptop to all IBM and Cray HPC leadership machines

Features
- Parallel I/O to/from storage
- Domain decomposition
- Network communication
- Written in C++
- C bindings, can be called from Fortran, C, C++
- Autoconf build system
- Lightweight: libdiy.a 800KB
- Maintainable: ~15K lines of code

Benefits
- Researchers can focus on their own work, not on parallel infrastructure
- Analysis applications can be custom
- Reuse core components and algorithms for performance and productivity

Simulator
Flash, Nek5000, HACC

Visualization Tool
ParaView, VisIt

Analysis Library
ITL, Osuflow, Qhull, VTK

DIY

MPI

I/O
Read Data

Decomposition
Blocking
Assignment

Communication
Neighbor
Global

Utilities
Parallel Compression
Datatype Creation
Parallel Sort

DIY usage and library organization
DIY: Global and Neighborhood Communication

DIY provides 3 efficient scalable communication algorithms on top of MPI. May be used in any combination. Most analysis algorithms use the same three communication patterns.

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Tracing</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Global Information Entropy</td>
<td>Merge-based reduction</td>
</tr>
<tr>
<td>Point-wise Information Entropy</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Morse-Smale Complex</td>
<td>Merge-based reduction</td>
</tr>
<tr>
<td>Computational Geometry</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Region growing</td>
<td>Nearest neighbor</td>
</tr>
<tr>
<td>Sort-last rendering</td>
<td>Swap-based reduction</td>
</tr>
</tbody>
</table>

Example of swap-based reduction of 16 blocks in 2 rounds.

Benchmark of DIY swap-based reduction vs. MPI reduce-scatter.
Applications using DIY

- Particle tracing of thermal hydraulics flow
- Information entropy analysis of astrophysics
- Morse-Smale complex of combustion
- Voronoi tessellation of cosmology
GLEAN- Enabling simulation-time data analysis and I/O acceleration

- Provides I/O acceleration by asynchronous data staging and topology-aware data movement, and achieved up to 30-fold improvement for FLASH and S3D I/O at 32K cores (SC’10, SC’11[x2], LDAV’11)
- Leverages data models of applications including adaptive mesh refinement grids and unstructured meshes
- Non-intrusive integration with applications using library (e.g. pnetcdf) interposition

<table>
<thead>
<tr>
<th>Infrastructure</th>
<th>Simulation</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-analysis</td>
<td>PHASTA</td>
<td>Visualization using Paraview</td>
</tr>
<tr>
<td>Staging</td>
<td>FLASH, S3D</td>
<td>I/O Acceleration</td>
</tr>
<tr>
<td>In situ</td>
<td>FLASH</td>
<td>Fractal Dimension, Histograms</td>
</tr>
<tr>
<td>In flight</td>
<td>MADBench2</td>
<td>Histogram</td>
</tr>
</tbody>
</table>

- Scaled to entire ALCF (160K BG/P cores + 100 Eureka Nodes)
- Provides a data movement infrastructure that takes into account node topology and system topology – up to 350 fold improvement at scale for I/O mechanisms
Simulation-time analysis for Aircraft design with Phasta on 160K Intrepid BG/P cores using GLEAN

- Co-Visualization of a PHASTA simulation running on **160K cores** of Intrepid using ParaView on 100 Eureka nodes enabled by GLEAN
- This enabled the scientists understand the temporal characteristics. It will enable them to interactively answer “what-if” questions.
- GLEAN achieves **48 GiBps** sustained throughput for data movement enabling simulation-time analysis

Isosurface of vertical velocity colored by velocity and cut plane through the synthetic jet (both on 3.3 Billion element mesh). *Image Courtesy: Ken Jansen*
GLEAN: Streamlining Data Movement in Airflow Simulation

- PHASTA CFD simulations produce as much as ~200 GB per time step
  - Rate of data movement off compute nodes determines how much data the scientists are able to analyze
- GLEAN contains optimizations for simulation-time data movement and analysis
  - Accelerating I/O via topology awareness, asynchronous I/O
  - Enabling in situ analysis and co-analysis

Strong scaling performance for 1GB data movement off ALCF Intrepid Blue Gene/P compute nodes. GLEAN provides 30-fold improvement over POSIX I/O at large scale. Strong scaling is critical as we move towards systems with increased core counts.

Thanks to V. Vishwanath (ANL) for providing this material.
Darshan: Characterizing Application I/O

How are are applications using the I/O system, and how successful are they at attaining high performance?

Darshan (Sanskrit for “sight”) is a tool we developed for I/O characterization at extreme scale:

- No code changes, small and tunable memory footprint (~2MB default)
- Characterization data aggregated and compressed prior to writing
- Captures:
  - Counters for POSIX and MPI-IO operations
  - Counters for unaligned, sequential, consecutive, and strided access
  - Timing of opens, closes, first and last reads and writes
  - Cumulative data read and written
  - Histograms of access, stride, datatype, and extent sizes

http://www.mcs.anl.gov/darshan/
A Data Analysis I/O Example

- Variable size analysis data requires headers to contain size information
- Original idea: all processes collectively write headers, followed by all processes collectively write analysis data
- Use MPI-IO, collective I/O, all optimizations
- 4 GB output file (not very large)

<table>
<thead>
<tr>
<th>Processes</th>
<th>I/O Time (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,192</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>16,384</td>
<td>16</td>
<td>47</td>
</tr>
<tr>
<td>32,768</td>
<td>32</td>
<td>57</td>
</tr>
</tbody>
</table>

Why does the I/O take so long in this case?
Problem: More than 50% of time spent writing output at 32K processes. Cause: Unexpected RMW pattern, difficult to see at the application code level, was identified from Darshan summaries.

What we expected to see, read data followed by write analysis:

What we saw instead: RMW during the writing shown by overlapping red (read) and blue (write), and a very long write as well.
Solution: Reorder operations to combine writing block headers with block payloads, so that "holes" are not written into the file during the writing of block headers, to be filled when writing block payloads. Also fix miscellaneous I/O bugs; both problems were identified using Darshan.

Result: Less than 25% of time spent writing output, output time 4X shorter, overall run time 1.7X shorter.

Impact: Enabled parallel Morse-Smale computation to scale to 32K processes on Rayleigh-Taylor instability data. Also used similar output strategy for cosmology checkpointing, further leveraging the lessons learned.

<table>
<thead>
<tr>
<th>Processes</th>
<th>I/O Time (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,192</td>
<td>7</td>
<td>60</td>
</tr>
<tr>
<td>16,384</td>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>32,768</td>
<td>7</td>
<td>33</td>
</tr>
</tbody>
</table>
S3D Turbulent Combustion Code

- S3D is a turbulent combustion application using a direct numerical simulation solver from Sandia National Laboratory.
- Checkpoints consist of four global arrays:
  - 2 3-dimensional
  - 2 4-dimensional
  - 50x50x50 fixed subarrays

Thanks to Jackie Chen (SNL), Ray Grout (SNL), and Wei-Keng Liao (NWU) for providing the S3D I/O benchmark, Wei-Keng Liao for providing this diagram, C. Wang, H. Yu, and K.-L. Ma of UC Davis for image.
Impact of Optimizations on S3D I/O

- Testing with PnetCDF output to single file, three configurations, 16 processes
  - All MPI-IO optimizations (collective buffering and data sieving) disabled
  - Independent I/O optimization (data sieving) enabled
  - Collective I/O optimization (collective buffering, a.k.a. two-phase I/O) enabled

<table>
<thead>
<tr>
<th></th>
<th>Coll. Buffering and Data Sieving Disabled</th>
<th>Coll. Buffering Enabled (incl. Aggregation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX writes</td>
<td>102,401</td>
<td>5</td>
</tr>
<tr>
<td>POSIX reads</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MPI-IO writes</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Unaligned in file</td>
<td>102,399</td>
<td>4</td>
</tr>
<tr>
<td>Total written (MB)</td>
<td>6.25</td>
<td>6.25</td>
</tr>
<tr>
<td>Runtime (sec)</td>
<td>1443</td>
<td>6.0</td>
</tr>
<tr>
<td>Avg. MPI-IO time per proc (sec)</td>
<td>1426.47</td>
<td>0.60</td>
</tr>
</tbody>
</table>