Decoupled I/O for Data-Intensive High Performance Computing

Chao Chen \(^1\)    Yong Chen \(^1\)    Kun Feng \(^2\)    Yanlong Yin \(^2\)
Hassan Eslami \(^3\)    Rajeev Thakur \(^4\)    Xian-He Sun \(^2\)
William D. Gropp \(^3\)

\(^1\)Department of Computer Science, Texas Tech University
\(^2\)Department of Computer Science, Illinois Institute of Technology
\(^3\)Department of Computer Science, University of Illinois Urbana-Champaign
\(^4\)Mathematics and Computer Science Division, Argonne National Laboratory

Sep 12th, 2014
High performance computing is a strategic tool for scientific discovery and innovation
- Climate Change: Community Earth System Model (CESM)
- Astronomy: Supernova, Sloan Digital Sky Survey
- etc..

Utilizing HPC system to simulate events and analyze the output to get insights

Figure 1: Climate modeling and analysis
Figure 2: Typical scientific workload
Big Data Problem

- Many scientific simulations become highly data intensive
- Simulation resolution desires finer granularity both spatial and temporal
  - e.g. climate model, 250KM ⇒ 20KM; 6 hours ⇒ 30 minutes
- The output data volume reaches tens of terabytes in a single simulation, the entire system deals with petabytes of data
- The pressure on the I/O system capability substantially increases

<table>
<thead>
<tr>
<th>PI</th>
<th>Project</th>
<th>On-Line Data</th>
<th>Off-Line Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamb, Don</td>
<td>FLASH: Buoyancy-Driven Turbulent Nuclear Burning</td>
<td>75TB</td>
<td>300TB</td>
</tr>
<tr>
<td>Fischer, Paul</td>
<td>Reactor Core Hydrodynamics</td>
<td>2TB</td>
<td>5TB</td>
</tr>
<tr>
<td>Dean, David</td>
<td>Computational Nuclear Structure</td>
<td>4TB</td>
<td>40TB</td>
</tr>
<tr>
<td>Baker, David</td>
<td>Computational Protein Structure</td>
<td>1TB</td>
<td>2TB</td>
</tr>
<tr>
<td>Worley, Patrick H.</td>
<td>Performance Evaluation and Analysis</td>
<td>2TB</td>
<td>1TB</td>
</tr>
<tr>
<td>Wolverton, Christopher</td>
<td>Kinetics and Thermodynamics of Metal and Complex Hydride Nanoparticles</td>
<td>5TB</td>
<td>100TB</td>
</tr>
<tr>
<td>Washington, Warren</td>
<td>Climate Science</td>
<td>10TB</td>
<td>345TB</td>
</tr>
<tr>
<td>Tsigelny, Igor</td>
<td>Parkinson’s Disease</td>
<td>2.5TB</td>
<td>50TB</td>
</tr>
<tr>
<td>Tang, William</td>
<td>Plasma Microturbulence</td>
<td>2TB</td>
<td>10TB</td>
</tr>
<tr>
<td>Sugar, Robert</td>
<td>Lattice QCD</td>
<td>1TB</td>
<td>44TB</td>
</tr>
<tr>
<td>Siegel, Andrew</td>
<td>Thermal Striping in Sodium Cooled Reactors</td>
<td>4TB</td>
<td>8TB</td>
</tr>
<tr>
<td>Roux, Benoit</td>
<td>Gating Mechanisms of Membrane Proteins</td>
<td>10TB</td>
<td>10TB</td>
</tr>
</tbody>
</table>

Figure 3: Data volume of current simulations

Figure 4: Climate Model Evolution: FAR (1990), SAR (1996), TAR (2001), AR4 (2007)
Gap between Applications’ Demand and I/O System Capability

- Gyrokinetic Toroidal Code (GTC) code
  - Outputs particle data that consists of two 2D arrays for electrons and ions, respectively
  - Two arrays distributed among all cores, particles can move across cores in a random manner as the simulation evolves

- A production run with the scale of 16,384 cores
  - Each core outputs roughly two million particles, 260GB in total
  - Desires $O(100\text{MB}/s)$ for efficient output

- The average I/O throughput of Jaguar (now Titan) is around 4.7MB/s per node

- Large and growing gap between the application’s requirement and system capability
Decoupled I/O

A new way of moving computations near to data to minimize the data movement and address the I/O bottleneck issue

- A runtime system design for our Decoupled Execution Paradigm
- Providing a set of interface for users to decouple their applications, and map into different sets of nodes

Figure 5: Decoupled Execution Paradigm and System Architecture
Overview of Decoupled I/O

- An extension to MPI library, managing both Compute nodes and Data nodes in the DEP architecture.
- Internally splits them into compute group and data group for normal applications and data-intensive operations respectively.

Figure 6: Overview of Decoupled I/O
Overview of Decoupled I/O

Involves 3 improvements to existing MPI library:

- Decoupled I/O APIs
- Improved MPI compiler (mpicc)
- Improved MPI process manager (hydra)

![Diagram of Decoupled I/O at runtime](image)

Figure 7: Decoupled I/O at runtime
Decoupled I/O API

- Abstracting each data-intensive operation with two phases: traditional I/O and data processing
- Providing APIs to treat them as an ensemble with different file handler design, and *data_op* argument

Table 1: Decoupled I/O APIs

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_File_decouple_open(MPI_Decoupled_File fh, char * filename, MPI_Comm comm);</td>
<td>Open a file for decoupled I/O.</td>
</tr>
<tr>
<td>MPI_File_decouple_close(MPI_Decoupled_File fh, MPI_Comm comm);</td>
<td>Close a decoupled file.</td>
</tr>
<tr>
<td>MPI_File_decouple_read(MPI_Decoupled_File fh, void *buf, int count, MPI_Datatype data_type, MPI_Op data_op, MPI_Comm comm);</td>
<td>Read from a decoupled file.</td>
</tr>
<tr>
<td>MPI_File_decouple_write(MPI_Decoupled_File fh, void *buf, int count, MPI_Datatype data_type, MPI_Op data_op, MPI_Comm comm);</td>
<td>Write to a decoupled file.</td>
</tr>
<tr>
<td>MPI_File_decouple_set_view(MPI_Decoupled_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char * datarep, MPI_Info info, MPI_Comm comm);</td>
<td>Set view for a decoupled file.</td>
</tr>
<tr>
<td>MPI_File_decouple_seek(MPI_Decoupled_File fh, MPI_Offset offset, int whence, MPI_Comm comm);</td>
<td>Seek within a decoupled file.</td>
</tr>
</tbody>
</table>
Decoupled I/O API Example

**Traditional Code**

```c
int buf;
MPI_File_read(fh, buf, ...);
for(i = 0; i < bufsize; i++) {
    sum += buf[i];
}
```

**Decoupled I/O Code**

```c
/* define operation */
int sum_op(buf, bufsize) {
    for (i = 0; i < bufsize; i++ )
        sum += buf[i];
}

MPI_op myop;
MPI_Op_create(myop, sum_op);
MPI_File_decoupled_read(fh, sum, myop, ....);
```
Process/Node management

- Data nodes and compute nodes are at the same level belonging to two groups
- "mpirun -np n -dp m -f hostfile ./app" to run an application with $n$ compute processes and $m$ data processes
- All of them belong to the MPI_COMM_WORLD communicator with distinguished rank
- Each group has its own group communicator MPI_COMM_LOCAL as an intra-communicator,
- MPI_COMM_INTER communicator as a group-to-group inter-communicator between the compute processes group and data processes group.
Identify the process type, compute process or data process, with its rank in MPI_COMM_WORLD to execute different codes.

Data process code is automatically generated by mpicc with hints defined by macros MPI_DECOUPLE_START and MPI_DECOUPLE_END.

MPI_Op for defining offloaded operations that have to be registered at the before MPI_DECOUPLE_START.
Decoupled I/O Implementation and Prototyping

- Completely based on MPI library
- Gather the tasks from compute processes, and scatter them to data process.

![Decoupled I/O prototype](image)

Figure 8: Decoupled I/O prototype
## Platform and Setup

### Platform:

<table>
<thead>
<tr>
<th>Name</th>
<th>DISCFarm Cluster(small), Hrothgar Cluster(large)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of nodes</td>
<td>DISCFarm: 16 nodes, Hrothgar: 640 nodes</td>
</tr>
<tr>
<td>CPU</td>
<td>DISCFarm: Xeon 2.6GHz, 8 cores, Hrothgar: Westmere 2.8GHz, 12 cores</td>
</tr>
<tr>
<td>Memory</td>
<td>DISCFarm: 4GB/node, Hrothgar: 24GB/node</td>
</tr>
</tbody>
</table>

### Evaluated Operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data assimilation (ENKF)</td>
<td>read the data, and apply EnKF algorithm (including 6 matrix multiplications, 1 matrix addition, and 1 matrix substitution), then write almost the same size data</td>
</tr>
<tr>
<td>Flow-routing</td>
<td>compute the direction where fluids flow to</td>
</tr>
<tr>
<td>Summation</td>
<td>calculates the total value of all specified data elements</td>
</tr>
<tr>
<td>Lookup</td>
<td>searches for and returns all elements that meet given criteria</td>
</tr>
</tbody>
</table>
Results and Analysis

- Compared against Active Storage (AS)
- 2 storage nodes, 4 data nodes and 8 compute nodes for DEPIO, 12 compute nodes for AS
- Around 13% improvements

Figure 9: Performance Comparison of Decoupled I/O and Active Storage I/O (Observed CPU usage on storage nodes: 1.3%)
Results and Analysis (with resource contention)

- Workload on storage nodes has great impact on Active Storage performance
- DEPIO keeps better performance than AS with less impact from workload on storage nodes

Figure 10: Performance of Decoupled I/O under Different CPU Usages on storage nodes
Results and Analysis

- Up to 60 nodes in the Hrothagar cluster
- Compared against traditional storage I/O (TS)
- Observed 25% performance improvements

![Emulation Performance of Decoupled I/O](image-url)

**Figure 11: Emulation Performance of the Decoupled I/O**
Overhead of the Decoupled I/O

- Primary overhead comes from the communication, Gather, Scatter, etc...
- As the data size of each I/O request increases, this overhead is observed to decrease steadily

![Communication Overhead of Decoupled I/O](image)

Figure 12: Overhead of the Decoupled I/O Operation
Big data computing brings new opportunities but also poses big challenges.

Dedicating data nodes for data-intensive operations can be helpful and critical for system performance.

An initial investigation of runtime system design to decouple a task into compute-intensive and data-intensive phases:
- Beneficial because of less resource contention, and reduced system wide data movements.

Prototyping were conducted to evaluate the potential of Decoupled I/O.

Plan to investigate the feasibility of the integration with the MapReduce and in-memory computing model.
Thank You

For more information, please visit: http://discl.cs.ttu.edu

This research is sponsored in part by the National Science Foundation under the grants CNS-1338078, CNS-1162540, CNS-1162488, and CNS-1161507.