Enabling Transparent Asynchronous I/O using Background Threads

Houjun Tang, Quincey Koziol, Suren Byna, John Mainzer, Tonglin Li
HPC I/O

● **Synchronous**
  ○ Code executes in sequence.
  ○ Computation is blocked by I/O, waste system resources.

● **Asynchronous**
  ○ Code may execute out of order.
  ○ I/O is non-blocking, can overlap with computation.
Synchronous vs. Asynchronous

Sync: Compute → I/O → Compute → I/O → Compute → I/O

Async: Compute → Compute → Compute → I/O → I/O → I/O

Time saved
Existing Asynchronous I/O Solutions

- POSIX I/O: `aio_*`
- MPI-IO: `MPI_File_i`*
- ADIOS/DataSpaces
- PDC (Proactive Data Containers)

Limited number of low level asynchronous APIs

Requires extra server processes

Manual dependency management
Asynchronous I/O Design Goals

- Effective to execute all I/O operations asynchronously.
- Requires no additional resources (e.g. server processes).
- **Automatic** data dependency management.
- **Minimal** application code changes.
Implicit Background Thread Approach

- Transparent from the application, no major code changes.
- Execute I/O operations in the background thread.
  - Allow application to queue a number of operations.
  - Start execution when application is not busy issuing I/O requests.
- Lightweight and low overhead for all I/O operations.
- No need to launch and maintain extra server processes.
Dependency management

Application thread

Start
File Open
Create Obj
Write Obj
Compute / File Close
End

Background thread

Start
Asynchronous I/O Initialization
Task Queue

App status check
App thread idle?
Yes
Task Execution
End

No
Queue Management

- Regular task
- Dependent task
- Collective task
Dependency management

- File create/open execute first.
- File close waits for all existing tasks to finish.
- Any read/write operations execute after prior write to same object, in app’s order.
- Any write executes after prior reads of same object, in app’s order.
- Collective operations, in order, one at a time.
HDF5 Implementation

- VOL connector
- HDF5 I/O operations
- Additional functions
- Background thread w/ Argobots
- Error reporting

**Virtual Object Layer**
- HDF5 data model and API.
- Switch I/O implementation.

Enable by:
- Environmental variable, or
- \texttt{H5Pset_vol_async()}
**HDF5 Implementation**

- VOL connector
- HDF5 I/O operations
- Additional functions
- Background thread w/ Argobots
- Error reporting

**Metadata operations**
- *Initiation*: create, open.
- *Modification*: extend dimension.
- *Query*: get datatype.
- *Close*: close the file.

**Raw data operations**
- Read and write.
HDF5 Implementation

- VOL connector
- HDF5 I/O operations
- Additional functions
- Background thread w/ Argobots
- Error reporting

- H5Pset_vol_async
- H5Pset_dxpl_async_cp_limit
- H5Dtest
- H5Dwait
- H5Ftest
- H5Fwait
HDF5 Implementation

- VOL connector
- HDF5 I/O operations
- Additional functions
- Background thread w/ Argobots
- Error handling
## Experimental Setup

<table>
<thead>
<tr>
<th>System</th>
<th>Cori @ NERSC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benchmarks</strong></td>
<td></td>
</tr>
<tr>
<td>Workloads</td>
<td>Single process</td>
</tr>
<tr>
<td></td>
<td>Multiple process</td>
</tr>
<tr>
<td></td>
<td>Metadata heavy</td>
</tr>
<tr>
<td></td>
<td>Raw data heavy</td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
</tr>
<tr>
<td><strong>I/O kernels</strong></td>
<td></td>
</tr>
<tr>
<td>VPIC-IO</td>
<td>time-series plasma physics particle data write</td>
</tr>
<tr>
<td>BD-CATS-IO</td>
<td>time-series particle data read, analysis</td>
</tr>
</tbody>
</table>
Single Process - No Computation (Overhead)

Overhead
5% average
Single Process - With Computation

Speedup 2 - 9X
Multiple Process - Metadata Intensive Read

Speedup 1.1 - 3.5X
Multiple Process - Metadata Intensive Write

Speedup 1.1 - 2.1X
Multiple Process - VPIC-IO

The diagram shows the observed write time (s) for different numbers of processes. There are four curves representing different methods:

- **VPIC Write**
- **HDF5**
- **Async-0%**
- **Async-50%**
- **Async-100%**

The x-axis represents the number of processes, ranging from 32 to 8192. The y-axis represents the observed write time in seconds, ranging from 0 to 160.

The graph indicates that there is a significant speedup with increasing numbers of processes, achieving a speedup of 5 - 7X as per the text "Speedup 5 - 7X" located at the top right of the image.
Multiple Process - BD-CATS-IO

Speedup
5 - 9X
Conclusion

- **An asynchronous I/O framework**
  - Highly effective and low overhead.
  - Support all I/O operations.
  - Require no additional server processes.
  - Transparent from application.

- **Future work**
  - Apply this work to more applications and I/O libraries, further performance optimization.
  - “Event tokens” for explicit tracking and controlling the asynchronous I/O tasks.
Thanks!

Questions?