Silo/HDF5 Modifications for Dawn

Mark C. Miller

Presented at the Dawn User Forum, April 15, 2010
Silo Background

Benefits (= flexibility)
- platform independent, self-describing, archiveable data
- random access (more true of post-processors than simulation codes)

Drawbacks (= performance degradation)
- metadata (data a lib writes on behalf of its caller)
- caller is far removed from actual disk I/O behavior/control
Poor Man’s Parallel I/O

Concurrent, parallel writes work ONLY FOR simple I/O patterns

- Size, shape, distribution of data across MPI tasks is ‘simple’ to describe
- The global monolithic “whole” object is decomposed on read, re-composed on write
- Example: 1D table of particle types, positions, velocities ==> good candidate

Large, multi-physics simulations are more complex

- size, shape, distribution and existence of data from task to task varies significantly
- All tasks have piece of (main) mesh...
- but some tasks have only some variables, materials, particles, tracers, time histories

Solution: Poor Man’s Parallel I/O

- Decompose into N GROUPS -- N totally independent of MPI_Comm_size()
- Only one MPI-task in each group has write access at any one time
- Serial I/O to multiple files, simultaneously
- Very flexible with what each MPI-task needs to do in the way of I/O
- Do not pay cost of “decomposing on read” and “recomposing on write”
- When N==1, get completely serial I/O (doesn’t scale too well!!!)
- When N==MPI_Comm_size() (Ares), get a file per MPI-task
- Ale3d typically chooses N==# I/O channels
- **Note:** Looking up from Lustre, you can’t tell the difference between this and MPI-IO
I/O Performance

Histogram

<table>
<thead>
<tr>
<th>request size</th>
<th>writes</th>
<th>bytes</th>
<th>%writes</th>
<th>cum.%writes</th>
<th>%bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;10^1 bytes:</td>
<td>48</td>
<td>217</td>
<td>20.1680</td>
<td>20.1680</td>
<td>.0001</td>
</tr>
<tr>
<td>&lt;10^2 bytes:</td>
<td>41</td>
<td>1485</td>
<td>17.2268</td>
<td>37.3949</td>
<td>.0009</td>
</tr>
<tr>
<td>&lt;10^3 bytes:</td>
<td>116</td>
<td>22474</td>
<td>48.7394</td>
<td>86.1344</td>
<td>.0136</td>
</tr>
<tr>
<td>&lt;10^4 bytes:</td>
<td>8</td>
<td>30540</td>
<td>3.3613</td>
<td>89.4957</td>
<td>.0186</td>
</tr>
<tr>
<td>&lt;10^5 bytes:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>89.4957</td>
<td>0</td>
</tr>
<tr>
<td>&lt;10^6 bytes:</td>
<td>3</td>
<td>1092492</td>
<td>1.2605</td>
<td>90.7563</td>
<td>.6655</td>
</tr>
<tr>
<td>&lt;10^7 bytes:</td>
<td>22</td>
<td>162989412</td>
<td>9.2436</td>
<td>100.0000</td>
<td>99.3010</td>
</tr>
</tbody>
</table>
Strategies for Improving Performance?

Aggregation
- Gather many smaller requests into fewer larger ones
- Need memory (buffer) to do this.
- Try aggregating as much as possible \textit{WITHIN} one MPI-task first.
- Failing that, start aggregating \textit{ACROSS} MPI-tasks.
Simplest Aggregation Strategy: Ram Disk

HDF5’s “Core” Virtual File Driver (VFD):

- Stores everything to a growing buffer in memory.
- Writes buffer to file on close.
- Reads ENTIRE file to memory buffer on open.
- Represents upper-bound of what is possible at expense of (a lot) of memory.
- Only works if when code does I/O, it is dumping less than 50% of available memory.
- Not a good long term solution

HDF5’s “Split” VFD:

- Splits data into two classes; raw and meta, writing each to its own file.
- Keep all metadata in memory using core vfd
- Write raw data using sec2 vfd.
- This results in good performance too.
- But, you wind up with two files for every one “file” that application creates.
New HDF5 Virtual File Driver for Silo

Breaks file’s address space into blocks

Does I/O only in blocks
  • Allocates enough memory to keep N blocks in memory

Two Parameters set by code
  • SILO_BLOCK_SIZE
  • SILO_BLOCK_COUNT

Good Values for Dawn
  • SILO_BLOCK_SIZE = (1<<20)
  • SILO_BLOCK_COUNT=16 (16 Megabytes total)
Other VFDs We May Write

Aggregate blocks across MPI-tasks
- Wind up with a SINGLE file at the bottom even though application thought it was writing many.
- But the file will still be a valid, HDF5 file

Remote-Core VFD
- Use extra MPI-tasks just for I/O
- Code “writes” to memory in these extra MPI tasks just like core VFD does now.
- Code goes back to compute while data drains to files from the extra MPI-tasks
- This could be fastest as code would NOT have to wait for I/O to complete before returning to compute.

Smart-Split VFD:
- Only one file is produced
- Raw data is block buffered as in new Silo VFD
- Metadata is kept in memory until file close, then tacked onto end of file.