Collective I/O Optimizations for Adaptive Mesh Refinement Data Writes on Lustre File System

Dharshi Devendran, Suren Byna, Bin Dong, Brian van Straalen, Hans Johansen, Noel Keen, Nagiza Samatova

Lawrence Berkeley National Laboratory

* North Carolina State University

Cray User Group 2016
May 10, 2016
Overview

- Complex I/O patterns result in poor performance
- Adaptive Mesh Refinement (AMR) I/O is complex
- Poor I/O stalls AMR simulations
- Contributions of this paper
  - Identification of AMR I/O bottleneck in the Chombo library
  - Collective buffering optimizations
    - MPI-IO Collective buffering
    - Novel Aggregated Collective Buffering (ACB) strategy
  - I/O performance improvement with ACB
Data-driven science

- **Simulations**
  - Multi-physics (FLASH) – 10 PB
  - Cosmology (NyX) – 10 PB
  - Plasma physics (VPIC) – 1 PB

- **Experimental and Observational data**
  - High energy physics (LHC) – 100 PB
  - Cosmology (LSST) – 60 PB
  - Genomics – 100 TB to 1 PB

- **Scientific applications rely on efficient access to data**
Background – Adaptive Mesh Refinement (AMR)

Dynamically adapts the spatial resolution of geometric meshes

- Improved efficiency of computational resources while meeting desirable error levels

Block-structured AMR

- A hierarchy of levels of resolutions
- Boxes/Patches: non-overlapping, logically-rectangular regions

Sample AMR data structure
AMR use cases – Ice sheet simulations

- **BISICLES** is an AMR ice sheet model aimed at large (continental)-scale ice sheets, and is built on the Chombo framework.

- Projections of future sea level rise resulting from impacts of climate change on large ice sheets.

A rectangular region of ice (~30km x 26km in real size) detaches from the main shelf.

Antarctica glacier, surrounded by ocean (dark blue).
AMR use cases – Climate simulations

- **The AMR Dycore** is a high-accuracy AMR climate model “dynamical core” based on Chombo
- **Significance**: Identify atmospheric features (tropical cyclones, atmospheric rivers, etc.), and track them in time and at high resolution

Atmospheric River, fig from ESRL
A brief intro to Chombo

- Software package for solving PDE-based physics models on AMR grids at large scales
- Utilities for simulating in domains with complex geometries, or on mapped grids

(A) An example of a space-time adaptive mesh refinement calculation in a cubed sphere geometry, for Climate applications. (B) The hierarchical levels of mesh refinement are used to capture moving features (e.g., multiple overlapping pressure waves). (C) Each level of refinement consists of a group of patches, each containing multiple data points, and each point belongs to a single patch. Tracked features can span patches or multiple levels as they evolve in time.
A brief intro to Chombo, cont.

- Implements block-structured AMR
  - Consists of hierarchy of uniform meshes, with resolution of 2 consecutive levels related by the refinement ratio
  - Each grid level divided into rectangular “boxes”

3 level AMR grid with 3 variables (u, v, p). The refinement ratio is 2 in each direction.
Distribution of data in Chombo applications

- Boxes distributed across MPI processes to balance loads across processes as much as possible
  - Load for box is usually proportional to number of points in a box

- Typical load balancing procedure:
  - Sort by Morton ordering (lists spatially adjacent boxes together)
  - Apply Kernighan-Lim algorithm to distribute boxes
    - Resulting distribution of boxes may appear random
Chombo’s I/O pattern

- Boxes arranged in lexicographic order in file
  - Lexicographic order: Box B0 <= Box B1 if lower left corner of B0 <= lower left corner of B1 in grid
- Processes write independently to non-contiguous regions in the file
- Separate write for each box
- Results in several small independent write calls

Boxes distributed across MPI processes to balance loads

Boxes in lexicographic order in file
Performance bottleneck with current implementation

- Separate write for each box results in several write calls overall
  - Large scale Chombo simulations can have \(\sim 10^5\) boxes
  - Large overhead for processing many write calls

- Each call only writes small amount of data (\(\sim 1-4\) MB per box)

- Each call performs new seek to find file location for writing
Experimental Evaluation – Systems

• **NERSC Edison**
  - Cray XC30 supercomputer with Lustre file system
  - The *scratch2* file system has 96 OSTs with 72 GB/s peak I/O bandwidth
    • 4 OSTs per I/O server (OSS)

• **NERSC Cori (Phase 1)**
  - Cray XC40 supercomputer with Lustre file system
  - 248 OSTs with 744 GB/s peak I/O bandwidth
    • 1 OST per I/O server (OSS)
Experimental Evaluation– Chombo I/O Benchmark

• Isolates Chombo’s write functionality

• Provides control over amount of data written through a replication factor parameter.
  - Parameter indicates number of times to replicate unit grid in each direction
  - In the experiments, we set this replication factor to write out 61 GB, 494 GB, and 987 GB data files

2 views (front and side) of one unit of AMR grid used in experiments. Grid has 3 levels. Refinement factor between levels is 4 (in each direction).
# Darshan stats for existing I/O pattern

<table>
<thead>
<tr>
<th>Darshan Counter</th>
<th>Independent I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MPI-IO writes</td>
<td>115268</td>
</tr>
<tr>
<td>Number of POSIX writes</td>
<td>115628</td>
</tr>
<tr>
<td>Most common access size</td>
<td>4 M</td>
</tr>
<tr>
<td>Count of most common access size</td>
<td>115201</td>
</tr>
<tr>
<td>2\textsuperscript{nd} most common access size</td>
<td>272 bytes</td>
</tr>
<tr>
<td>Count of 2\textsuperscript{nd} most common access size</td>
<td>15</td>
</tr>
</tbody>
</table>

Number and size of writes determined by number and size of boxes (total number of boxes = 115628, each box ~ 4 MB)

Stats for run on Edison with 2304 processes (96 nodes).
File striped across 96 OSTs with stripe size of 8 MB. 494 GB was written.
Aggregation with MPI-IO collective buffering

- Idea: aggregate contiguous data into buffers to reduce number of write calls
- Subset of MPI processes assigned to perform the aggregation
- CB2 mode of MPI-IO collective buffering optimizes for the Lustre file system
Collective buffering I/O pattern

- Aggregators (A0, …) collect boxes and reshuffle them into buffers
- Each process only sends a single box in each collective call
  - Each process can contain several boxes, resulting in many collective write calls
## Darshan stats for MPI-IO Collective buffering

<table>
<thead>
<tr>
<th>Darshan Counter</th>
<th>Independent I/O</th>
<th>Collective buffering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MPI-IO writes</td>
<td>115268</td>
<td>119808</td>
</tr>
<tr>
<td>Number of POSIX writes</td>
<td>115628</td>
<td>164270</td>
</tr>
<tr>
<td>Most common access size</td>
<td>4 M</td>
<td>4 M</td>
</tr>
<tr>
<td>Count of most common access size</td>
<td>115201</td>
<td>42689</td>
</tr>
<tr>
<td>2(^{nd}) most common access size</td>
<td>272 bytes</td>
<td>8 M</td>
</tr>
<tr>
<td>Count of 2(^{nd}) most common access size</td>
<td>15</td>
<td>4830</td>
</tr>
</tbody>
</table>

- Get some larger writes with collective buffering
- Number of CB aggregators = number of OSTs = 96
### Darshan stats for MPI-IO Collective buffering

<table>
<thead>
<tr>
<th>Darshan Counter</th>
<th>Independent I/O</th>
<th>Collective buffering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MPI-IO writes</td>
<td>115268</td>
<td>119808</td>
</tr>
<tr>
<td>Number of POSIX writes</td>
<td>115628</td>
<td>164270</td>
</tr>
<tr>
<td>Most common access size</td>
<td>4 M</td>
<td>4 M</td>
</tr>
<tr>
<td>Count of most common access size</td>
<td>115201</td>
<td>42689</td>
</tr>
<tr>
<td>2\textsuperscript{nd} most common access size</td>
<td>272 bytes</td>
<td>8 M</td>
</tr>
<tr>
<td>Count of 2\textsuperscript{nd} most common access size</td>
<td>15</td>
<td>4830</td>
</tr>
</tbody>
</table>

- Still have many small writes
- Get some larger writes with collective buffering
- Number of CB aggregators = number of OSTs = 96
Aggregated Collective Buffering (ACB)

- Aggregate all boxes into one buffer on each process
  - Number of boxes copied into ACB buffer is a parameter, which can be tuned to balance performance and memory usage (future direction)
- MPI-IO aggregators reshuffle boxes for large contiguous writes
Darshan stats for ACB

<table>
<thead>
<tr>
<th>Darshan Counter</th>
<th>Independent I/O</th>
<th>Collective buffering</th>
<th>ACB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MPI-IO writes</td>
<td>115268</td>
<td>119808</td>
<td>6912</td>
</tr>
<tr>
<td>Number of POSIX writes</td>
<td>115628</td>
<td>164270</td>
<td>63241</td>
</tr>
<tr>
<td>Most common access size</td>
<td>4 M</td>
<td>4 M</td>
<td>8 M</td>
</tr>
<tr>
<td>Count of most common access size</td>
<td>115201</td>
<td>42689</td>
<td>63177</td>
</tr>
<tr>
<td>2\textsuperscript{nd} most common access size</td>
<td>272 bytes</td>
<td>8 M</td>
<td>272 bytes</td>
</tr>
<tr>
<td>Count of 2\textsuperscript{nd} most common access size</td>
<td>15</td>
<td>4830</td>
<td>16</td>
</tr>
</tbody>
</table>

ACB significantly reduces number of POSIX writes
## Darshan stats for ACB

<table>
<thead>
<tr>
<th>Darshan Counter</th>
<th>Independent I/O</th>
<th>Collective buffering</th>
<th>ACB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MPI-IO writes</td>
<td>115268</td>
<td>119808</td>
<td>6912</td>
</tr>
<tr>
<td>Number of POSIX writes</td>
<td>115628</td>
<td>164270</td>
<td>63241</td>
</tr>
<tr>
<td>Most common access size</td>
<td>4 M</td>
<td>4 M</td>
<td>8 M</td>
</tr>
<tr>
<td>Count of most common access size</td>
<td>115201</td>
<td>42689</td>
<td>63177</td>
</tr>
<tr>
<td>2\textsuperscript{nd} most common access size</td>
<td>272 bytes</td>
<td>8 M</td>
<td>272 bytes</td>
</tr>
<tr>
<td>Count of 2\textsuperscript{nd} most common access size</td>
<td>15</td>
<td>4830</td>
<td>16</td>
</tr>
</tbody>
</table>

ACB significantly reduces number of POSIX writes

Most writes are relatively large
Performance on Edison

- 61 GB test: 576 procs (24 nodes), striped across 24 OSTs with 4 MB stripe size
- 494 GB test: 2304 procs (96 nodes), striped across 96 OSTs with 8 MB stripe size
- 987 GB test: 5760 procs (240 nodes), stripes across 96 OSTs with 16 MB stripe size

- ACB is 2.6x to 3.8x faster than independent I/O
- ACB is 2x to 2.6x faster than collective buffering
- Striping affects performance of collective buffering on 987 GB test case
Performance on Cori

- ACB is 5.7x to 9.6x faster than independent I/O
- ACB is 1.6x to 1.8x faster than collective buffering

61 GB test: 576 procs (24 nodes), striped across 24 OSTs with 4 MB stripe size
494 GB test: 3072 procs (96 nodes), striped across 96 OSTs with 8 MB stripe size
987 GB test: 5856 procs (244 nodes), stripes across 244 OSTs with 16 MB stripe size
Evaluation with Lustre striping - Edison

Experiment specifics:
• 2304 procs (96 nodes)
• 494 GB file
• Striped across 96 OSTs
• Stripe size = 8 MB

ACB is 2x to 3.4x faster than independent I/O, and 2x to 2.6x faster than collective buffering
Evaluation with Lustre striping - Cori

Experiment specifics:
- 3072 procs (96 nodes)
- 494 GB file
- Striped across 96 OSTs
- Stripe size = 8 MB

ACB is 3.9x to 9.1x faster than independent I/O, and 1.5x to 1.8x faster than collective buffering
Performance of BISICLES w/ ACB

• Simulates evolution of Antarctic ice sheet over time
  • Solves nonlinear equation for ice velocity and advects the ice
  • Results in many boxes of varying sizes
    • Finest level contains 896 boxes; number of cells in a box ranges between 256 and 4096 cells; each process has at least one box; maximum number of boxes on a process is 6; average number of boxes is 1.5

Antarctic ice-sheet velocity field. The inset shows the refined meshes around the Pine Island Glacier in the Amundsen Sea Embayment. Pine Island Glacier (PIG) is the single greatest Antarctic contributor to sea level rise at the moment.
Performance of BISICLES w/ ACB, cont

- For checkpoint files, ACB is **20X faster** than independent I/O, and **13X faster** than collective buffering.

- For plot files, ACB is **13X faster** than independent I/O, and **6.4x faster** than collective buffering.

Run on 576 procs (24 nodes) on Edison. Files striped across 24 OSTs with stripe size of 4 MB.
Darshan analysis of BISICLES I/O w/ ACB

- Independent I/O and collective I/O perform several small writes, especially for checkpoint files
- ACB writes out large chunks of data

Top four write sizes (ACCESS SIZE) and corresponding counts from Darshan logs

<table>
<thead>
<tr>
<th>ACCESS SIZE</th>
<th>Ind I/O</th>
<th>Coll I/O</th>
<th>ACB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS SIZE 1</td>
<td>8736</td>
<td>5408</td>
<td>4 M</td>
</tr>
<tr>
<td>COUNT 1</td>
<td>4085</td>
<td>3182</td>
<td>477</td>
</tr>
<tr>
<td>ACCESS SIZE 2</td>
<td>5408</td>
<td>17920</td>
<td>272</td>
</tr>
<tr>
<td>COUNT 2</td>
<td>3229</td>
<td>3059</td>
<td>69</td>
</tr>
<tr>
<td>ACCESS SIZE 3</td>
<td>7680</td>
<td>19488</td>
<td>3848</td>
</tr>
<tr>
<td>COUNT 3</td>
<td>3080</td>
<td>2646</td>
<td>51</td>
</tr>
<tr>
<td>ACCESS SIZE 4</td>
<td>17920</td>
<td>7680</td>
<td>13160</td>
</tr>
<tr>
<td>COUNT 4</td>
<td>1333</td>
<td>1281</td>
<td>40</td>
</tr>
</tbody>
</table>

Checkpoint file statistics

<table>
<thead>
<tr>
<th>ACCESS SIZE</th>
<th>Ind I/O</th>
<th>Coll I/O</th>
<th>ACB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS SIZE 1</td>
<td>73440</td>
<td>73440</td>
<td>4 M</td>
</tr>
<tr>
<td>COUNT 1</td>
<td>1429</td>
<td>1529</td>
<td>322</td>
</tr>
<tr>
<td>ACCESS SIZE 2</td>
<td>204000</td>
<td>38880</td>
<td>272</td>
</tr>
<tr>
<td>COUNT 2</td>
<td>1378</td>
<td>1422</td>
<td>23</td>
</tr>
<tr>
<td>ACCESS SIZE 3</td>
<td>38880</td>
<td>204000</td>
<td>544</td>
</tr>
<tr>
<td>COUNT 3</td>
<td>1372</td>
<td>1298</td>
<td>13</td>
</tr>
<tr>
<td>ACCESS SIZE 4</td>
<td>300000</td>
<td>396000</td>
<td>40</td>
</tr>
<tr>
<td>COUNT 4</td>
<td>709</td>
<td>604</td>
<td>9</td>
</tr>
</tbody>
</table>

Plot file statistics
Conclusions and Future Work

- ACB issues fewer write calls than collective buffering and independent I/O
  - Each ACB write call sends relatively large chunks of data
- ACB speeds up independent I/O implementation by 2x to 9.1x, and collective buffering by 1.5x to 2.6x
- Apply and analyze ACB performance on EBChombo and cubed sphere climate application (CAMR)
- Eliminate extra buffer copy in ACB, and use unions of hyperslabs to specify locations of boxes in memory
- Explore ACB performance on burst buffers
Thanks!

Contact:
Suren Byna [SByna@lbl.gov]
Nagiza Samatova [nagiza.samatova@gmail.com]

Projects: **In situ AMR Indexing and Querying**
**ExaHDF5**

Thanks to DOE ASCR Program Manager Dr. Lucy Nowell