JLESC Project: Toward taming large and complex data flows in data-centric supercomputing

François Tessier*, Venkatram Vishwanath*, Emmanuel Jeannot†

*Argonne National Laboratory, USA
†Inria Bordeaux Sud-Ouest, France

November 29, 2016
Project’s status

Toward taming large and complex data flows in data-centric supercomputing

- Summary
  - Collaboration between Argonne National Lab. and Inria Bordeaux
  - Started in the beginning of 2015
  - Understand, characterize, and transform application data flows over a range of architectures

- Visits and meetings
  - E. Jeannot visited ANL on March 2015 and June 2016
  - I stayed 10 days at ANL on March 2015

- Impact and publications
  - I moved from Inria to ANL in February 2016 for a postdoc position
  - Our first results have been published and presented at the COM-HPC Workshop few weeks ago (SC’16)

Topology and Affinity Aware Hierarchical and Distributed Load-balancing in Charm++

- JLPC Inria-PPL project completed two years ago
- Recent publication at the COM-HPC Workshop (SC’16) thanks to the JLESC for Blue Waters allocations
Topology-Aware Data Aggregation for Intensive I/O on Large-Scale Supercomputers

François Tessier*, Preeti Malakar*, Venkatram Vishwanath*, Emmanuel Jeannot†, Florin Isaila‡

*Argonne National Laboratory, USA
†Inria Bordeaux Sud-Ouest, France
‡University Carlos III, Spain

November 29, 2016
Data Movement at Scale

- Computational science simulation such as climate, heart and brain modelling or cosmology have large I/O needs
  - Typically around 10% to 20% of the wall time is spent in I/O

**Table:** Example of I/O from large simulations

<table>
<thead>
<tr>
<th>Scientific domain</th>
<th>Simulation</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cosmology</td>
<td>Q Continuum</td>
<td>2 PB / simulation</td>
</tr>
<tr>
<td>High-Energy Physics</td>
<td>Higgs Boson</td>
<td>10 PB / year</td>
</tr>
<tr>
<td>Climate / Weather</td>
<td>Hurricane</td>
<td>240 TB / simulation</td>
</tr>
</tbody>
</table>

- Increasing disparity between computing power and I/O performance in the largest supercomputers

![Graph showing the ratio of I/O (TB/s) to Flops (TF/s) over years from 1997 to 2017. The ratio decreases over time.](image-url)
Complex Architectures

- Complex network topologies: multidimensional tori, dragonfly, ...
- Partitioning of the architecture to reduce I/O interference
  - IBM BG/Q with I/O nodes (Figure), Cray with LNET nodes
- New tiers of storage/memory for data staging
  - MCDRAM in KNL, NVRAM, Burst buffer nodes

Mira
- 49,152 nodes / 786,432 cores
- 768 TB of memory
- 27 PB of storage, 330 GB/s (GPFS)
- 5D Torus network
- Peak performance: 10 PetaFLOPS
Two-phase I/O

- Available in MPI I/O implementations such as ROMIO
- Improves I/O performance by writing larger data chunks
- Selects a subset of processes to aggregate data before writing it to the storage system

Limitations:
- Poor for small messages (from experiments)
- Inefficient aggregator placement policy
- Fails to take advantage of data model, data layout and memory hierarchy

Figure: Two-phase I/O mechanism
Two-phase I/O

- Available in MPI I/O implementations such as ROMIO
- Improves I/O performance by writing larger data chunks
- Selects a subset of processes to aggregate data before writing it to the storage system

Limitations:
- Poor for small messages (from experiments)
- Inefficient aggregator placement policy
- Fails to take advantage of data model, data layout and memory hierarchy

![Two-phase I/O mechanism](image)

**Figure:** Two-phase I/O mechanism
Two-phase I/O

- Available in MPI I/O implementations such as ROMIO
- Improves I/O performance by writing larger data chunks
- Selects a subset of processes to aggregate data before writing it to the storage system

Limitations:
- Poor for small messages (from experiments)
- Inefficient aggregator placement policy
- Fails to take advantage of data model, data layout and memory hierarchy

Figure: Two-phase I/O mechanism
<table>
<thead>
<tr>
<th></th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Context</td>
</tr>
<tr>
<td>2</td>
<td>Approach</td>
</tr>
<tr>
<td>3</td>
<td>Evaluation</td>
</tr>
<tr>
<td>4</td>
<td>Conclusion and Perspectives</td>
</tr>
</tbody>
</table>
**Approach**

**Improved aggregator placement while taking into account:**
- The topology of the architecture
- The data access pattern

**Efficient implementation of the two-phase I/O scheme**
- Captures the data model and the data layout to optimize the I/O scheduling
- Pipelining of aggregation phase and I/O phase to optimize data movement
- Leverage one-sided communication
- Uses non-blocking operation to reduce synchronization
Aggregator Placement - Topology-aware strategy

- $\omega(u, v)$: Amount of data exchanged between nodes $u$ and $v$
- $d(u, v)$: Number of hops from nodes $u$ to $v$
- $l$: The interconnect latency
- $B_{i \rightarrow j}$: The bandwidth from node $i$ to node $j$.

- $C_1 = \max \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right), i \in V_C$
- $C_2 = l \times d(A, IO) + \frac{\omega(A, IO)}{|V_C| \times B_{A \rightarrow IO}}$

Objective function:

- $\text{TopoAware}(A) = \min (C_1 + C_2)$

Computed by each process independently in $O(n), n = |V_C|$
Aggregator Placement - Topology-aware strategy

- $\omega(u, v)$: Amount of data exchanged between nodes $u$ and $v$
- $d(u, v)$: Number of hops from nodes $u$ to $v$
- $l$: The interconnect latency
- $B_{i\rightarrow j}$: The bandwidth from node $i$ to node $j$.

- $C_1 = \max \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i\rightarrow A}} \right), i \in V_C$
- $C_2 = l \times d(A, IO) + \frac{\omega(A, IO)}{|V_C| \times B_{A\rightarrow IO}}$

Objective function:
- $\text{TopoAware}(A) = \min (C_1 + C_2)$
- Computed by each process independently in $O(n), n = |V_C|$
Aggregator Placement - Topology-aware strategy

- $\omega(u, v)$: Amount of data exchanged between nodes $u$ and $v$
- $d(u, v)$: Number of hops from nodes $u$ to $v$
- $l$: The interconnect latency
- $B_{i \rightarrow j}$: The bandwidth from node $i$ to node $j$.

- $C_1 = \max \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right), i \in V_C$
- $C_2 = l \times d(A, IO) + \frac{\omega(A, IO)}{|V_C| \times B_{A \rightarrow IO}}$

Objective function:

- $\text{TopoAware}(A) = \min (C_1 + C_2)$

- Computed by each process independently in $O(n), n = |V_C|$
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \{round, aggregator, buffer\} for each process P
  - Let’s say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- 2\textsuperscript{nd} round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- and so on...
- Limitations: MPI\_Comm\_split, one aggr./node at most
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \{round, aggregator, buffer\} for each process P
  - Let’s say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
  - P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
  - 2\textsuperscript{nd} round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
  - and so on...
- Limitations: MPI\_Comm\_split, one aggr./node at most
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \{round, aggregator, buffer\} for each process P
  - Let's say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- 2\textsuperscript{nd} round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- and so on...
- Limitations: MPI\_Comm\_split, one aggr./node at most
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \{round, aggregator, buffer\} for each process P
  - Let’s say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- 2nd round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
  - and so on...
- Limitations: MPI_Comm_split, one aggr./node at most
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \{round, aggregator, buffer\} for each process P
  - Let’s say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- 2\textsuperscript{nd} round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- and so on...
- Limitations: MPI\_Comm\_split, one aggr./node at most
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \(\{\text{round, aggregator, buffer}\}\) for each process P
  - Let's say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- 2\(^{nd}\) round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- and so on...

- Limitations: MPI_Comm_split, one aggr./node at most
Algorithm

- Initialization: allocate buffers, create MPI windows, compute tuples \{round, aggregator, buffer\} for each process P
  - Let's say P1 is the aggregator
- P0, P1 and P2 put data in buffer 1 (round 1) of P1. P3 waits (fence)
- P1 writes buffer 1 in file and aggregates data from all the ranks in buffer 2
- 2\textsuperscript{nd} round. P1 writes buffer 2 and aggregates data from P1, P2 and P3
- and so on...
- Limitations: MPI\_Comm\_split, one aggr./node at most
Outline

1. Context
2. Approach
3. Evaluation
4. Conclusion and Perspectives
Micro-benchmark - Placement strategies

- Evaluation on Mira (BG/Q), 512 nodes, 16 ranks/node
- Each rank sends a data buffer chosen randomly between 0 and 2 MB
- Writes to /dev/null of the I/O node (aggregation and I/O phases only)
- Aggregation settings: 16 aggregators, 16 MB buffer size

Four tested strategies

- **Shortest path**: smallest distance to the I/O node
- **Longest path**: longest distance to the I/O node
- **Greedy**: lowest rank in partition (similar to the default MPICH strategy)
- **Topology-aware**
Micro-benchmark - Placement strategies

- Evaluation on Mira (BG/Q), 512 nodes, 16 ranks/node
- Each rank sends a data buffer chosen randomly between 0 and 2 MB
- Writes to /dev/null of the I/O node (aggregation and I/O phases only)
- Aggregation settings: 16 aggregators, 16 MB buffer size

Table: Impact of aggregators placement strategy

<table>
<thead>
<tr>
<th>Strategy</th>
<th>I/O Bandwidth (MBps)</th>
<th>Aggr. Time/round (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>1927.45</td>
<td>421.33</td>
</tr>
<tr>
<td>Longest path</td>
<td>2202.91</td>
<td>370.40</td>
</tr>
<tr>
<td>Shortest path</td>
<td>2484.39</td>
<td>327.08</td>
</tr>
<tr>
<td>Topology-Aware</td>
<td>2638.40</td>
<td>310.46</td>
</tr>
</tbody>
</table>

- I/O bandwidth increased by 37% in comparison to the Greedy strategy and 6% over the Shortest Path approach
HACC-IO

- I/O part of a large-scale cosmological application simulating the mass evolution of the universe with particle-mesh techniques
- Each process manages particles defined by 9 variables (38 bytes)
  - XX, YY, ZZ, VX, VY, VZ, phi, pid and mask
- One file per Pset (128 nodes) vs. one single shared file
- Aggregation settings: 16 aggregators per Pset, 16 MB buffer size (MPICH)
- Average and standard deviation on 10 runs

Figure: Data layouts in HACC-IO
Peak is estimated to 22.4 GBps (theoretical: 28.8 GBps)
Our approach achieves higher performance than the default strategies
- 5K particles (190 KB) and AoS data layout: $15 \times$ faster than MPI I/O
Sub-filing is an efficient approach for improved I/O performance.

Our topology-aware strategy achieves 90% of the peak I/O bandwidth (22.4 GBps).

- Significant improvement particularly for small messages.
Peak is estimated to 89.6 GBps (theoretical: 115.2 GBps)
90% of the peak I/O bandwidth achieved as on 1024 nodes
Improved I/O performance for both AoS and SoA layouts and significant improvement on smaller messages for the SoA case (up to 43%)
Outline

1. Context
2. Approach
3. Evaluation
4. Conclusion and Perspectives
Conclusion and Perspectives

Conclusion

- Optimized two-phase I/O library incorporating
  - Topology-aware aggregator placement
  - Optimized data movement and buffering (double-buffering, one-sided communication, block size awareness)
- Very good performance at scale, outperforming standard approaches
- On the I/O part of a cosmological application, up to $12 \times$ improvement on 65K ranks
- Architecture characteristics are critical for performance at scale

Next steps

- Take the routing policy into account
- Incorporate additional data models and layouts (2D, 3D-arrays)
- Hierarchical approach to tackle different tiers of storage
Acknowledgments

- Argonne Leadership Computing Facility at Argonne National Laboratory
- DOE Office of Science, ASCR
- NCSA-Inria-ANL-BSC-JSC-Riken Joint-Laboratory on Extreme Scale Computing
- European Union Seventh Framework Program
Thank you for your attention!

ftessier@anl.gov
Evaluation on Mira (BG/Q), 1024 nodes, 16 ranks/node
Each rank writes 1 MB
Write to /dev/null of the I/O node (performance of just aggregation and I/O phases)

Table: I/O Bandwidth (in MBps) achieved on a simple benchmark with a topology-aware aggregator placement while varying the number of aggregators and the buffer size.

<table>
<thead>
<tr>
<th>#Aggr/Pset</th>
<th>Buffer size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 MB</td>
</tr>
<tr>
<td>8</td>
<td>7652.49</td>
</tr>
<tr>
<td>16</td>
<td>7318.15</td>
</tr>
<tr>
<td>32</td>
<td>6329.95</td>
</tr>
</tbody>
</table>