Optimizing Data Aggregation by Leveraging the Deep Memory Hierarchy on Large-scale Systems

François Tessier, Paul Gressier, Venkatram Vishwanath

Argonne National Laboratory, USA

Thursday 14th June, 2018
Computational science simulation in scientific domains such as in materials, high energy physics, engineering, have large performance needs

- In computation: the Human Brain Project, for instance, goes after at least 1 ExaFLOPS
- In I/O: 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>

- New workloads with specific needs of data movement
  - Big data, machine learning, checkpointing, in-situ, co-located processes, ...
  - Multiple data access pattern (model, layout, data size, frequency)
Massively parallel supercomputers supplying an increasing processing capacity

- The first 10 machines listed in the top500 ranking are able to provide more than 10 PFlops
- Aurora, the first Exascale system in the US (ANL!), will likely feature millions of cores

However, the memory per core or TFlop is decreasing...

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name, Location</td>
<td>BlueGene/L, USA</td>
<td>Sunway TaihuLight, China</td>
<td>N/A</td>
</tr>
<tr>
<td>Theoretical perf.</td>
<td>596 TFlops</td>
<td>125,436 TFlops</td>
<td>×210</td>
</tr>
<tr>
<td>#Cores</td>
<td>212,992</td>
<td>10,649,600</td>
<td>×50</td>
</tr>
<tr>
<td>Memory</td>
<td>73,728 GB</td>
<td>1,310,720 GB</td>
<td>×17.7</td>
</tr>
<tr>
<td>Memory/core</td>
<td>346 MB</td>
<td>123 MB</td>
<td>÷2.8</td>
</tr>
<tr>
<td>Memory/TFlop</td>
<td>124 MB</td>
<td>10 MB</td>
<td>÷12.4</td>
</tr>
<tr>
<td>I/O bw</td>
<td>128 GBps</td>
<td>288 GBps</td>
<td>×2.25</td>
</tr>
<tr>
<td>I/O bw/core</td>
<td>600 kBps</td>
<td>27 kBps</td>
<td>÷22.2</td>
</tr>
<tr>
<td>I/O bw/TFlop</td>
<td>214 MBps</td>
<td>2.30 MBps</td>
<td>÷93.0</td>
</tr>
</tbody>
</table>

Table: Comparison between the first ranked supercomputer in 2007 and in 2017.

Growing importance of movements of data on current and upcoming large-scale systems
Mitigating this bottleneck from an hardware perspective leads to an increasing complexity and a diversity of the architectures

- **Deep memory and storage hierarchy**
  - Blurring boundary between memory and storage
  - New tiers: MCDRAM, node-local storage, network-attached memory, NVRAM, Burst buffers
  - Various performance characteristics: latency, bandwidth, capacity

- **Complexity of interconnection network**
  - Topologies: 5D-Torus, Dragon-fly, fat trees
  - Partitioning: network dedicated to I/O
  - Routing policies: static, adaptive
Data Aggregation

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

Limitations:

- Inefficient aggregator placement policy
- Cannot leverage the deep memory hierarchy
- Inability to use staging data

Figure: Two-phase I/O mechanism
-Based on TAPIOCA, a library implementing the two-phase I/O scheme for topology-aware data aggregation at scale\(^1\) and featuring:
  - Optimized implementation of the two-phase I/O scheme (I/O scheduling)
  - Network interconnect abstraction for I/O performance portability
  - Aggregator placement taking into account the network interconnect and the data access pattern

- Augmented to include:
  - Abstraction including the topology and the deep memory hierarchy
  - Architecture-aware aggregators placement
  - Memory-aware data aggregation algorithm

---

\(^1\) F. Tessier, V. Vishwanath, and E. Jeannot. “TAPIOCA: An I/O Library for Optimized Topology-Aware Data Aggregation on Large-Scale Supercomputers”.
MA-TAPIOCA - Abstraction for Interconnect Topology

- Topology characteristics include:
  - Spatial coordinates
  - Distance between nodes: number of hops, routing policy
  - I/O nodes location, depending on the filesystem (bridge nodes, LNET, ...)
  - Network performance: latency, bandwidth

- Need to model some unknowns such as routing in the future

Listing 1: Function prototypes for network interconnect

```c
int networkBandwidth (int level);
int networkLatency ();
int networkDistanceToIONode (int rank, int IONode);
int networkDistanceBetweenRanks (int srcRank, int destRank);
```

Figure: 5D-Torus on BG/Q and intra-chassis Dragonfly Network on Cray XC30
(Credit: LLNL / LBNL)
MA-TAPIOCA - Abstraction for Memory and Storage

- Memory management API
- Topology characteristics including spatial location, distance
- Performance characteristics: bandwidth, latency, capacity, persistency
- Scope of memory/storage tiers (PFS vs node-local SSD)
  - On those cases, a process has to be involved at destination

**Listing 2:** Function prototypes for memory/storage data movements

```c
buff_t* memAlloc (mem_t mem, int buffSize, bool masterRank, char* fileName, MPI_Comm comm);
void memFree (buff_t *buff);
int memWrite (buff_t *buff, void* srcBuffer, int srcSize, int offset, int destRank);
int memRead (buff_t *buff, void* srcBuffer, int srcSize, int offset, int srcRank);
void memFlush (buff_t *buff);
int memLatency (mem_t mem);
int memBandwidth (mem_t mem);
int memCapacity (mem_t mem);
int memPersistency (mem_t mem);
```

MA-TAPIOCA

- Memory API (alloc, write, read, free, …)
- Abstraction layer (mmap, memkind, …)
- DRAM
- HBM
- NVRAM
- PFS
- …
Initial conditions: memory capacity for aggregation and destination.

\( \omega(u, v) \): Amount of data to move from memory bank \( u \) to \( v \)

\( d(u, v) \): distance between memory bank \( u \) and \( v \)

\( l \): The latency such as \( l = \max(l_{\text{network}}, l_{\text{memory}}) \);

\( B_{u \rightarrow v} \): The bandwidth from memory bank \( u \) to \( u \), such as \( B_{u \rightarrow v} = \min(B_{w_{\text{network}}}, B_{w_{\text{memory}}}) \).

\( A \): Aggregator, \( T \): Target

\[
\text{Cost}_A = \sum_{i \in V_C, i \neq A} \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right)
\]

\[
\text{Cost}_T = l \times d(A, T) + \frac{\omega(A, T)}{B_{A \rightarrow T}}
\]

\[
\text{MemAware}(A) = \min(\text{Cost}_A + \text{Cost}_T)
\]
\[ \text{Cost}_A = \sum_{i \in V_C, i \neq A} \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right) \]

\[ \text{Cost}_T = l \times d(A, T) + \frac{\omega(A, T)}{B_{A \rightarrow T}} \]

\[ \text{MemAware}(A) = \min(\text{Cost}_A + \text{Cost}_T) \]

<table>
<thead>
<tr>
<th>Value#</th>
<th>HBM</th>
<th>DRAM</th>
<th>NVR</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ms)</td>
<td>10</td>
<td>20</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>Bandwidth (GBps)</td>
<td>180</td>
<td>90</td>
<td>0.15</td>
<td>12.5</td>
</tr>
<tr>
<td>Capacity (GB)</td>
<td>16</td>
<td>192</td>
<td>128</td>
<td>N/A</td>
</tr>
<tr>
<td>Persistency</td>
<td>No</td>
<td>No</td>
<td>job lifetime</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table:** Memory and network capabilities based on vendors information
MA-TAPIOCA - Memory and topology aware aggregator placement

\[
\text{Cost}_A = \sum_{i \in V_C, i \neq A} \left( l \times d(i, A) + \frac{\omega(i, A)}{B_{i \rightarrow A}} \right)
\]

\[
\text{Cost}_T = l \times d(A, T) + \frac{\omega(A, T)}{B_{A \rightarrow T}}
\]

\[
\text{MemAware}(A) = \min (\text{Cost}_A + \text{Cost}_T)
\]

<table>
<thead>
<tr>
<th>Value#</th>
<th>HBM</th>
<th>DRAM</th>
<th>NVR</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ms)</td>
<td>10</td>
<td>20</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>Bandwidth (GBps)</td>
<td>180</td>
<td>90</td>
<td>0.15</td>
<td>12.5</td>
</tr>
<tr>
<td>Capacity (GB)</td>
<td>16</td>
<td>192</td>
<td>128</td>
<td>N/A</td>
</tr>
<tr>
<td>Persistency</td>
<td>No</td>
<td>No</td>
<td>job lifetime</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table: Memory and network capabilities based on vendors information

<table>
<thead>
<tr>
<th>P#</th>
<th>(\omega(i, A))</th>
<th>HBM</th>
<th>DRAM</th>
<th>NVR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>0.593</td>
<td>0.603</td>
<td>2.350</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>0.470</td>
<td>0.480</td>
<td>2.020</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0.742</td>
<td>0.752</td>
<td>2.710</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>0.503</td>
<td>0.513</td>
<td>2.120</td>
</tr>
</tbody>
</table>

Table: For each process, MemAware(A)
- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - MA-TAPIOCA_AGGTIER environment variable: topology-aware placement only
  - MA-TAPIOCA_PERSISTENCY environment variable to set the level of persistency required in case of a memory and topology aware placement
MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - MA-TAPIOCA_AGGTIER environment variable: topology-aware placement only
  - MA-TAPIOCA_PERSISTENCY environment variable to set the level of persistency required in case of a memory and topology aware placement
MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - MA-TAPIOCA_AGGTIER environment variable: topology-aware placement only
  - MA-TAPIOCA_PERSISTENCY environment variable to set the level of persistency required in case of a memory and topology aware placement

![Diagram showing MA-TAPIOCA algorithm]

- Processes
- Data
- Aggregators
- Target

- Buffering for DRAM, MCDRAM, NVRAM, BB, ...
- Non-blocking MPI calls
MA-TAPIOCA - Two-phase I/O algorithm

- Aggregator(s) selection according to the cost model described previously
- Overlapping of I/O and aggregation phases based on recent MPI features such as RMA and non-blocking operations
- The aggregation can be either defined by the user or chosen with our placement model
  - MA-TAPIOCA_AGGTIER environment variable: topology-aware placement only
  - MA-TAPIOCA_PERSISTENCY environment variable to set the level of persistency required in case of a memory and topology aware placement

**Algorithm 1: Collective MPI I/O**

```
1 n ← 5;
2 x[n], y[n], z[n];
3 ofst ← rank × 3 × n;
5
6 MPI_File_read_at_all (f, ofst, x, n, type, status);
7 ofst ← ofst + n;
9
10 MPI_File_read_at_all (f, ofst, y, n, type, status);
11 ofst ← ofst + n;
13
14 MPI_File_read_at_all (f, ofst, z, n, type, status);
```

**Algorithm 2: MA-TAPIOCA**

```
1 n ← 5;
2 x[n], y[n], z[n];
3 ofst ← rank × 3 × n;
5
6 for i ← 0, i < 3, i ← i + 1 do
7     count[i] ← n;
8     type[i] ← sizeof (type);
9     ofst[i] ← ofst + i × n;
11
12 MA-TAPIOCA_Init (count, type, ofst, 3);
14
15 MA-TAPIOCA_Read (f, ofst, x, n, type, status);
16 ofst ← ofst + n;
18
19 MA-TAPIOCA_Read (f, ofst, y, n, type, status);
20 ofst ← ofst + n;
22
23 MA-TAPIOCA_Read (f, ofst, z, n, type, status);
```
**Theta**
- Cray CX40 11.69 PFlops supercomputer at Argonne
  - 4,392 Intel KNL nodes with 64 cores
  - 16 GB of HBM, 192 GB of DRAM and 128 GB on-node SSD
- 10 PB parallel file system managed by Lustre
- Cray Aries dragonfly network interconnect

**Cooley**
- Intel Haswell-based visualization and analysis cluster at Argonne
  - 126 nodes with 12 cores and a NVIDIA Tesla K80
  - 384 GB of DRAM and a local hard drive (345 GB)
- 27 PB of storage managed by GPFS
- FDR Infiniband interconnect
S3D-IO

- I/O kernel of direct numerical simulation code in the field of computational fluid dynamics focusing on turbulence-chemistry interactions in combustion.
- 3D domain decomposition
- The state of each element is stored in an array of structure data layout
- The files as output are used for checkpointing and data analysis

Experimental setup

- Theta, a 11 PFlops Cray XC40 supercomputer with a Lustre filesystem
  - Single shared file collectively written every $n$ timesteps, stripped among OST.
  - Available tiers of memory: DRAM, HBM, on-node SSD
  - 96 aggregators for 256 nodes and 384 for 1024 nodes for both MPI-IO and MA-TAPIOCA
  - Lustre: 48 OST, 16MB stripe size, 4 aggr. per OST, 16MB buffer size
- Average and standard deviation on 10 runs
S3D-IO on Cray XC40 + Lustre

- Typical use-case with 134 and 537 millions grid points respectively distributed on 256 and 1024 nodes on Theta (16 ranks per node)
- Aggregation performed on HBM with MA-TAPIOCA
- I/O bandwidth increased by a factor of **3x** on 1024 nodes.

**Table:** Maximum write bandwidth (GBps).

<table>
<thead>
<tr>
<th>Points</th>
<th>Size</th>
<th>256 nodes</th>
<th>1024 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI-IO</td>
<td>134M</td>
<td>160 GB</td>
<td>3.02 GBps</td>
</tr>
<tr>
<td>MA-TAPIOCA</td>
<td>537M</td>
<td>640 GB</td>
<td>4.86 GBps</td>
</tr>
<tr>
<td>Perf. Improvement</td>
<td>N/A</td>
<td>N/A</td>
<td>+60.93%</td>
</tr>
</tbody>
</table>

- Experiments on 256 nodes (134 millions grid points) while artificially reducing the memory capacity.
- The capacity requirement not being fulfilled, our placement algorithm selects another aggregation layer (gray boxes)

**Table:** Maximum write bandwidth (GBps).

<table>
<thead>
<tr>
<th>Run</th>
<th>HBM</th>
<th>DDR</th>
<th>NVRAM</th>
<th>Bandwidth</th>
<th>Std dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16 GB</td>
<td>192 GB</td>
<td>128 GB</td>
<td>4.86 GBps</td>
<td>0.39 GBps</td>
</tr>
<tr>
<td>2</td>
<td>↓ 32 MB</td>
<td>192 GB</td>
<td>128 GB</td>
<td>4.90 GBps</td>
<td>0.43 GBps</td>
</tr>
<tr>
<td>3</td>
<td>↓ 32 MB</td>
<td>↓ 32 MB</td>
<td>128 GB</td>
<td>2.98 GBps</td>
<td>0.15 GBps</td>
</tr>
</tbody>
</table>
Experiments - HACC-IO

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
- Checkpointing files with data in an array of structure data layout

Experimental setup

- Theta, a 11 PFlops Cray XC40 supercomputer with a Lustre filesystem
  - Available tiers of memory: DRAM, HBM, on-node SSD
  - Lustre: 48 OST, 16MB stripe size, 4 aggr. per OST, 16MB buffer size
- Cooley, an Haswell-based visualization and analysis cluster with GPFS
  - Available tiers of memory: DRAM, on-node HDD
- Average and standard deviation on 10 runs
(a) One file per node on 1024 nodes while varying the data size per rank.

- Experiments on 1024 nodes on Theta
- Aggregation layer set with the MA-TAPIOCA_AGGTIER environment variable
- Regardless of the subfiling granularity, MA-TAPIOCA can use the local SSD as a shared file destination ($mmap + MPI_{Win}$)

(b) One file per node, 1MB/rank, while varying the number of nodes.
Experiments on 1024 nodes on Theta, one file per node

Comparison between aggregation on DRAM and HBM when writing on the local SSD

I/O performance achieved comparable

Predicted by our model

Figure: One file per node written on the local SSD. Aggregation on DRAM and HBM.
Typical workflow that can be seamlessly implemented with MA-TAPIOCA
Experiments on 256 nodes on Theta
Write time counter-balanced by the read time from the local storage
Total I/O time reduced by more than **26%**

Table: Max. Write and Read bandwidth (GBps) and total I/O time achieved with and without aggregation on SSD

<table>
<thead>
<tr>
<th>Agg. Tier</th>
<th>Write</th>
<th>Read</th>
<th>I/O time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA-TAPIOCA</td>
<td>DDR</td>
<td>47.50</td>
<td>38.92</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>DDR</td>
<td>32.95</td>
<td>37.74</td>
</tr>
<tr>
<td>MA-TAPIOCA</td>
<td>SSD</td>
<td>26.88</td>
<td>227.22</td>
</tr>
</tbody>
</table>

Variation

| Variation | -36.10% | +446.94% | -26.82% |

![Diagram showing the flow of data from application to parallel file system via MMAP and SSD with aggregation and I/O operations.]
HACC-IO on Cooley + GPFS

- Code and performance portability thanks to our abstraction layer
- Experiments on 64 nodes on Cooley (Haswell-based cluster)
- Same application code, same optimization algorithm using our memory and network interconnect abstraction
- Total I/O time reduced by **12%**

![Diagram](image)

**Table:** Max. Write and Read bandwidth (GBps) and total I/O time achieved with and without aggregation on local HDD

<table>
<thead>
<tr>
<th></th>
<th>Agg. Tier</th>
<th>Write</th>
<th>Read</th>
<th>I/O Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MA-TAPIOCA</strong></td>
<td>DDR</td>
<td>6.60</td>
<td>38.80</td>
<td>123.41 ms</td>
</tr>
<tr>
<td><strong>MPI-IO</strong></td>
<td>DDR</td>
<td>6.02</td>
<td>17.46</td>
<td>155.40 ms</td>
</tr>
<tr>
<td><strong>MA-TAPIOCA</strong></td>
<td>HDD</td>
<td>5.97</td>
<td>35.86</td>
<td>135.86 ms</td>
</tr>
<tr>
<td><strong>Variation</strong></td>
<td></td>
<td>-0.83%</td>
<td>+105.38%</td>
<td>-12.57%</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

MA-TAPIOCA, a data aggregation library able to take advantage of the network interconnect and the deep memory hierarchy for improved performance
- Architecture abstraction making possible to perform data aggregation on any type of memory or storage
- Memory and topology aware aggregators placement
- Efficient data aggregation algorithm

Good performance at scale, outperforming MPI I/O
- On a typical workflow, up to 26% improvement on a Cray XC40 supercomputer with Lustre and up to 12% on a visualization cluster

Code and performance portability on large-scale supercomputers
- Same application code running on various platforms
- Same optimization algorithms using our interconnect abstraction

Future Work
- As the memory hierarchy tends to be deeper and deeper, multi-level data aggregation is of interest
- Intervene at a lower level to capture any kind of data types
- Transfer to widely used I/O libraries
Acknowledgments

- Argonne Leadership Computing Facility at Argonne National Laboratory
- DOE Office of Science, ASCR
- Proactive Data Containers (PDC) project
Thank you for your attention!
ftessier@anl.gov