TAPIOCA: An I/O Library for Optimized Topology-Aware Data Aggregation on Large-Scale Supercomputers

François Tessier∗, Venkatram Vishwanath∗, Emmanuel Jeannot†

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

Wednesday 6th September, 2017

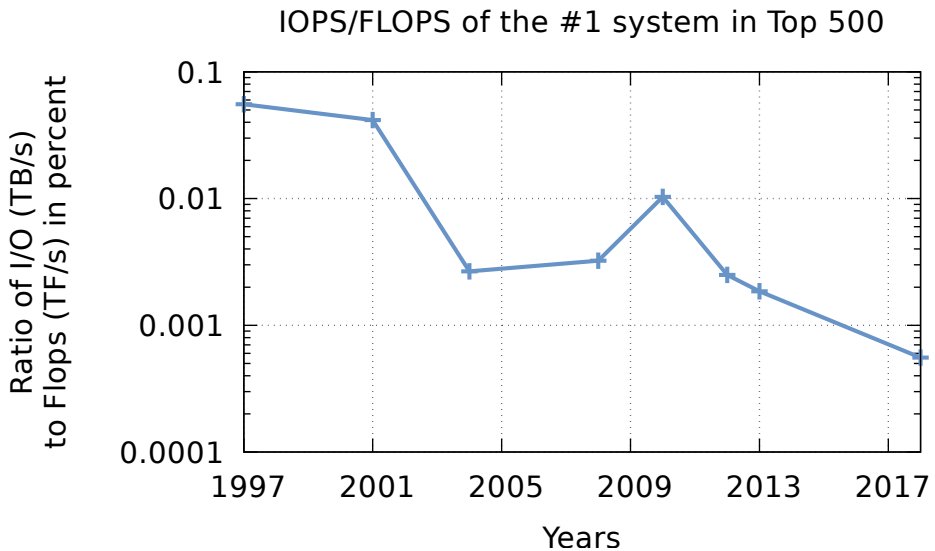
Data Movement at Scale

- Computational science simulation in scientific domains such as in materials, high energy physics, engineering, 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

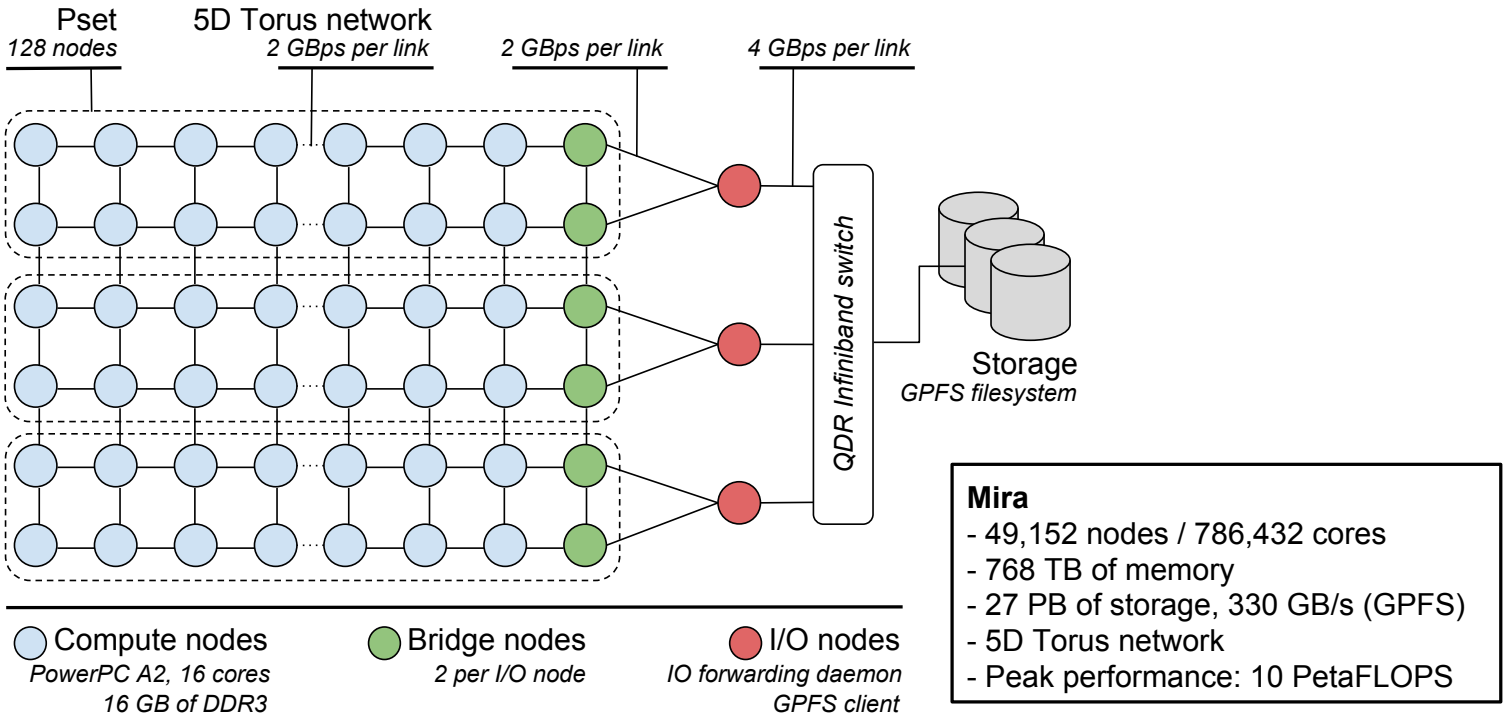
Scientific domain	**Simulation**	**Data size**
Cosmology	Q Continuum	**2 PB / simulation**
High-Energy Physics	Higgs Boson	**10 PB / year**
Climate / Weather	Hurricane	**240 TB / simulation**

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



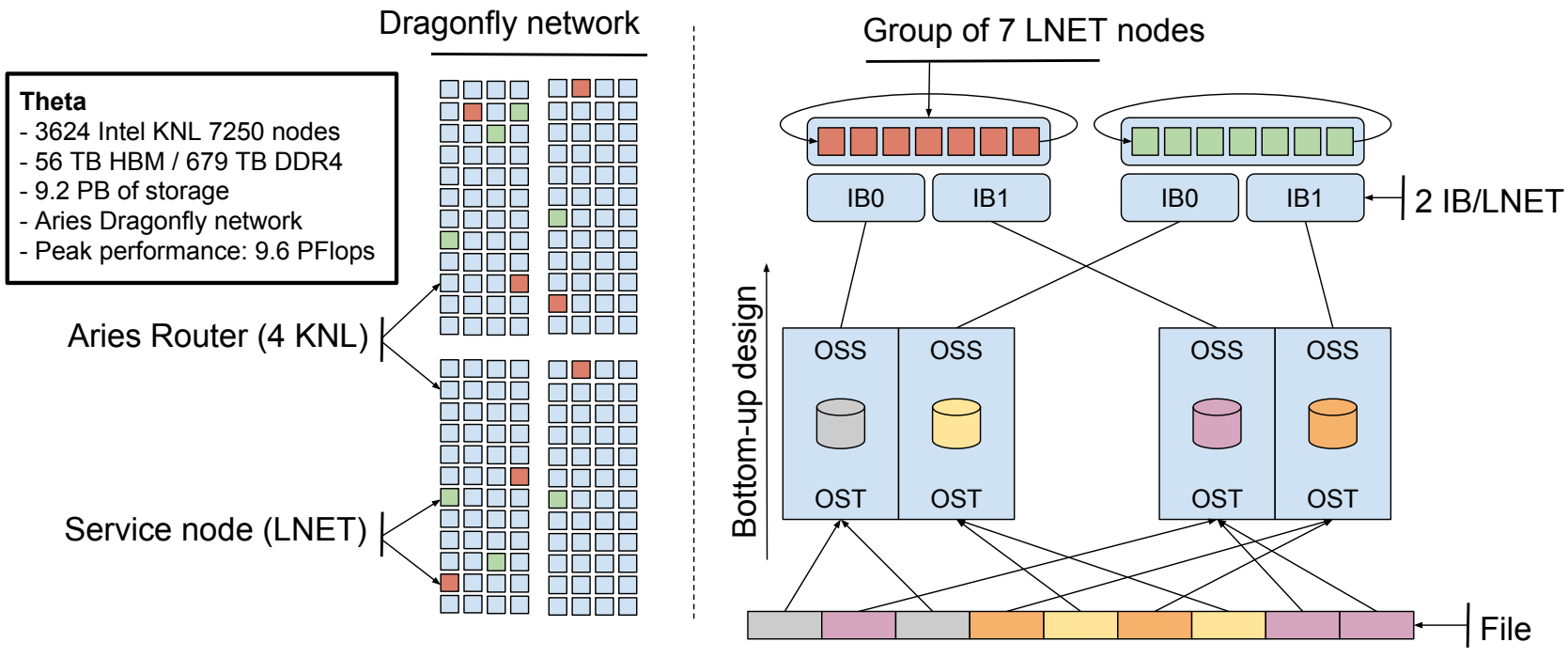
Complex Interconnect Hierarchies and Filesystems

- On BG/Q, data movement needs to fully exploit the 5D-Torus topology for improved performance
- Additionally, we need to exploit the placement of the I/O nodes for performance
- Cray supercomputers have similar challenges with dragonfly-based interconnects together with placement of LNET nodes for I/O
- We need to leverage filesystem specific features such as OSTs and striping in Lustre, among others.



Complex Interconnect Hierarchies and Filesystems

- On BG/Q, data movement needs to fully exploit the 5D-Torus topology for improved performance
- Additionally, we need to exploit the placement of the I/O nodes for performance
- Cray supercomputers have similar challenges with dragonfly-based interconnects together with placement of LNET nodes for I/O
- We need to leverage filesystem specific features such as OSTs and striping in Lustre, among others.



Two-phase I/O

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

Limitations:

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

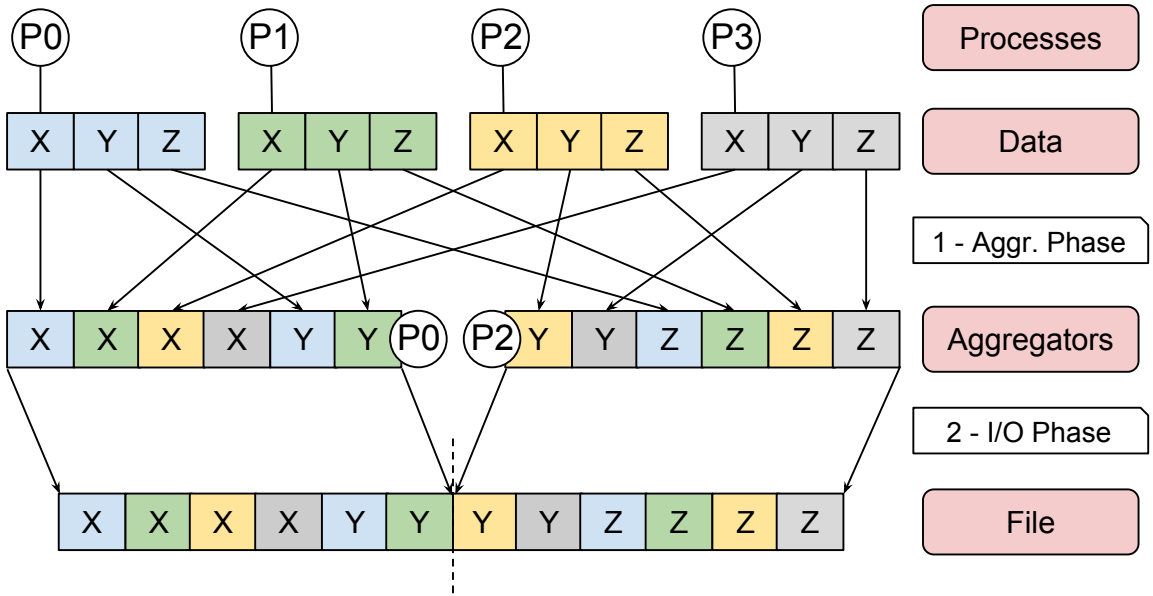
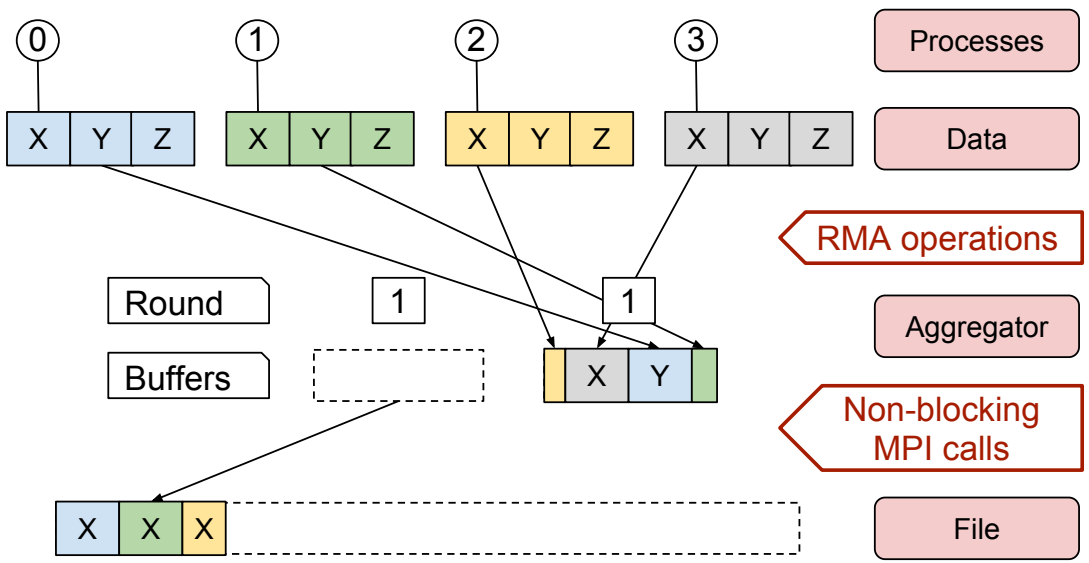


Figure: Two-phase I/O mechanism

TAPIOCA

- Library based on the two-phase I/O scheme for topology-aware data aggregation at scale featuring:
  - Efficient implementation of the Two-phase I/O scheme
    - Pipelining (RMA, non-blocking calls) of aggregation and I/O phases
    - Captures the data model and data layout to optimize the I/O scheduling
  - Interconnect architecture abstraction facilitating the I/O performance portability
  - Topology-aware aggregator placement taking into account
    - The topology of the architecture
    - The data access pattern



TAPIOCA

- Library based on the two-phase I/O scheme for topology-aware data aggregation at scale featuring:
  - Efficient implementation of the Two-phase I/O scheme
    - Pipelining (RMA, non-blocking calls) of aggregation and I/O phases
    - Captures the data model and data layout to optimize the I/O scheduling
  - Interconnect architecture abstraction facilitating the I/O performance portability
  - Topology-aware aggregator placement taking into account
    - The topology of the architecture
    - The data access pattern

Algorithm 1: Collective MPI I/O

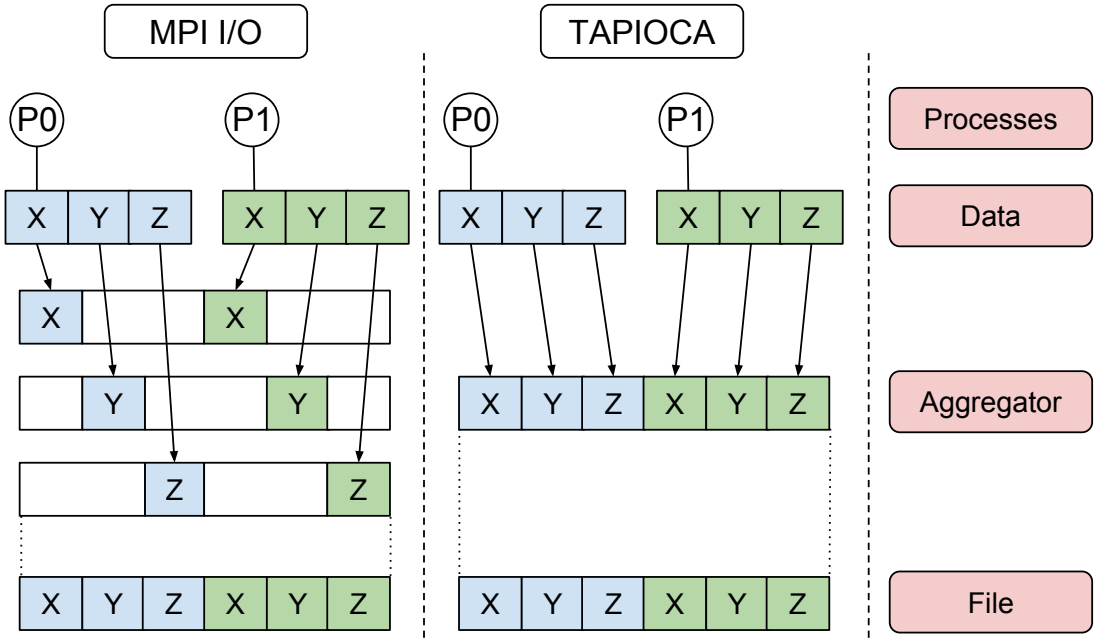
```
1  n ← 5;
2  x[n], y[n], z[n];
3  ofst ← rank × 3 × n;
5
6  MPI_File_write_at_all (f, ofst, x, n, type, status);
7  ofst ← ofst + n ;
9
10 MPI_File_write_at_all (f, ofst, y, n, type, status);
11 ofst ← ofst + n;
13
14 MPI_File_write_at_all (f, ofst, z, n, type, status);
```

Algorithm 2: 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 TAPIOCA_Init (count, type, ofst, 3);
14
15 TAPIOCA_Write (f, ofst, x, n, type, status);
16 ofst ← ofst + n ;
18
19 TAPIOCA_Write (f, ofst, y, n, type, status);
20 ofst ← ofst + n;
22
23 TAPIOCA_Write (f, ofst, z, n, type, status);
```

TAPIOCA

- Library based on the two-phase I/O scheme for topology-aware data aggregation at scale featuring:
  - Efficient implementation of the Two-phase I/O scheme
    - Pipelining (RMA, non-blocking calls) of aggregation and I/O phases
    - Captures the data model and data layout to optimize the I/O scheduling
  - Interconnect architecture abstraction facilitating the I/O performance portability
  - Topology-aware aggregator placement taking into account
    - The topology of the architecture
    - The data access pattern



TAPIOCA

- Library based on the two-phase I/O scheme for topology-aware data aggregation at scale featuring:
  - Efficient implementation of the Two-phase I/O scheme
    - Pipelining (RMA, non-blocking calls) of aggregation and I/O phases
    - Captures the data model and data layout to optimize the I/O scheduling
  - Interconnect architecture abstraction facilitating the I/O performance portability
  - Topology-aware aggregator placement taking into account
    - The topology of the architecture
    - The data access pattern

Abstractions for Interconnect Topology

- Performance portability on first-class supercomputers
  - Same application code running on various platforms
  - Same optimization algorithms using our interconnect abstraction
- Topology characteristics include:
  - Spatial coordinates, network dimensions
  - 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 and uncertainties such as routing

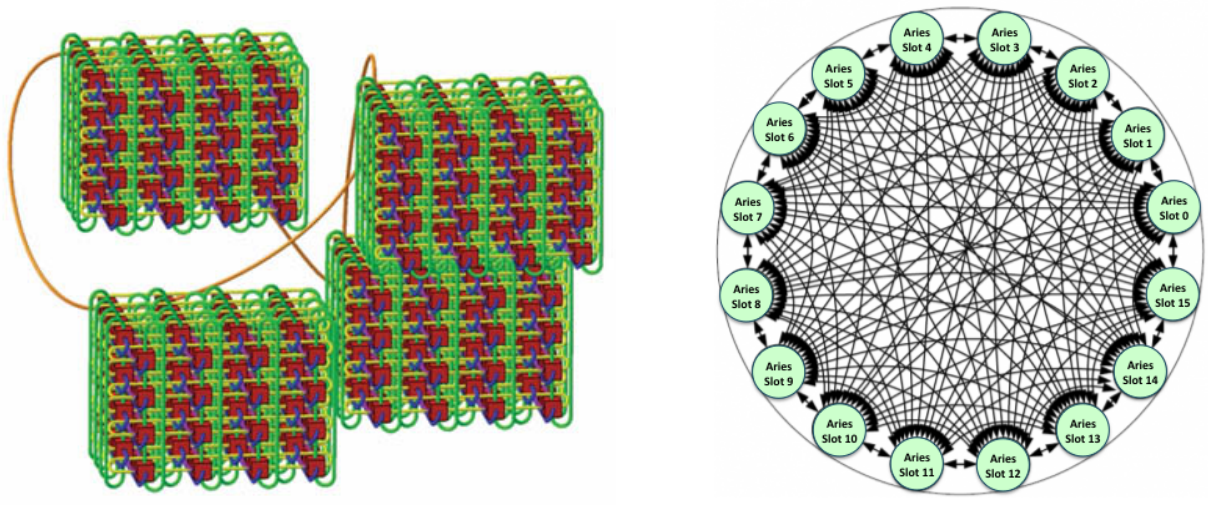


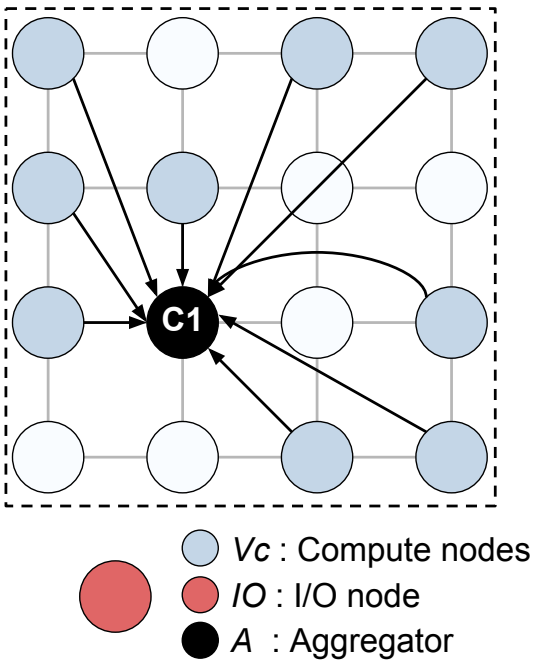
Figure: 5D-Torus on BG/Q and intra-chassis Dragonfly Network on Cray XC30

Source: LLNL / LBNL.

Abstractions for Interconnect Topology - Our current approach

- TAPIOCA features a topology-aware aggregator placement
- This approach is based on quantitative information easy to gather: latency, bandwidth, distance between nodes

- $\omega(u, v)$: Amount of data exchanged between nodes u and v
- $d(u, v)$: Number of hops from nodes u to v
- l: The interconnect latency
- $B_{i \to j}$: The bandwidth from node i to node j
- $\mathbf{C_1} = \sum_{i \in V_C, i \neq A} \left(l \times d(i, A) + \frac{\omega(i,A)}{B_{i \to A}} \right)$
- $\mathbf{C_2} = l \times d(A, IO) + \frac{\omega(A,IO)}{B_{A \to IO}}$
- $\mathbf{TopoAware(A)} = min\,(C_1 + C_2)$

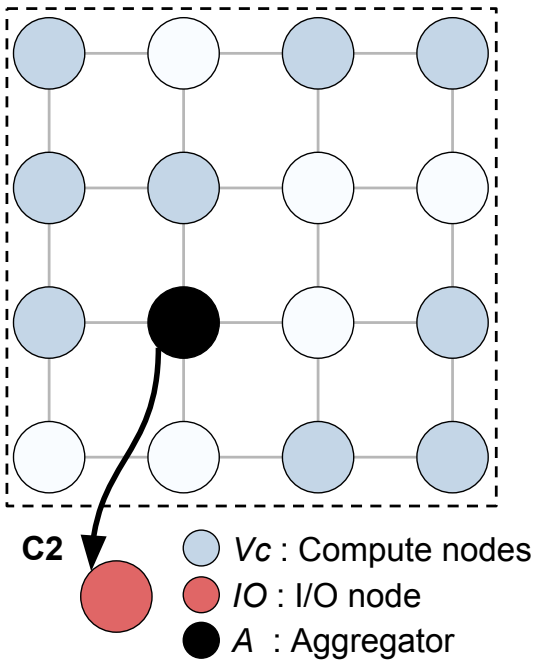


- Computed by each process independently in $O(n), n = |V_C|$
- Contention-aware algorithm: static and dynamic routing policies, unknown vendors information such as routing policy or data distribution among I/O nodes, ...

Abstractions for Interconnect Topology - Our current approach

- TAPIOCA features a topology-aware aggregator placement
- This approach is based on quantitative information easy to gather: latency, bandwidth, distance between nodes

- $\omega(u, v)$: Amount of data exchanged between nodes u and v
- $d(u, v)$: Number of hops from nodes u to v
- l: The interconnect latency
- $B_{i \to j}$: The bandwidth from node i to node j
- $\mathbf{C_1} = \sum_{i \in V_C, i \neq A} \left(l \times d(i, A) + \frac{\omega(i,A)}{B_{i \to A}} \right)$
- $\mathbf{C_2} = l \times d(A, IO) + \frac{\omega(A,IO)}{B_{A \to IO}}$
- $\mathbf{TopoAware(A)} = min\,(C_1 + C_2)$

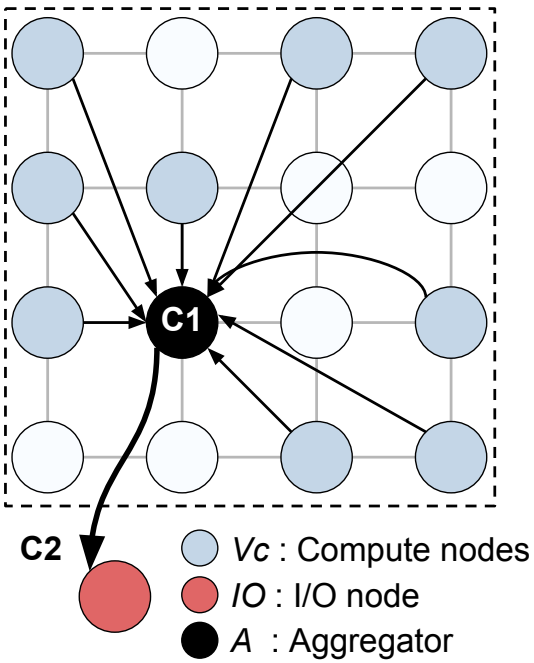


- Computed by each process independently in $O(n), n = |V_C|$
- Contention-aware algorithm: static and dynamic routing policies, unknown vendors information such as routing policy or data distribution among I/O nodes, ...

Abstractions for Interconnect Topology - Our current approach

- TAPIOCA features a topology-aware aggregator placement
- This approach is based on quantitative information easy to gather: latency, bandwidth, distance between nodes

- $\omega(u, v)$: Amount of data exchanged between nodes u and v
- $d(u, v)$: Number of hops from nodes u to v
- l: The interconnect latency
- $B_{i \to j}$: The bandwidth from node i to node j
- $\mathbf{C_1} = \sum_{i \in V_C, i \neq A} \left(l \times d(i, A) + \frac{\omega(i,A)}{B_{i \to A}} \right)$
- $\mathbf{C_2} = l \times d(A, IO) + \frac{\omega(A,IO)}{B_{A \to IO}}$
- **TopoAware(A)** $= min\,(C_1 + C_2)$

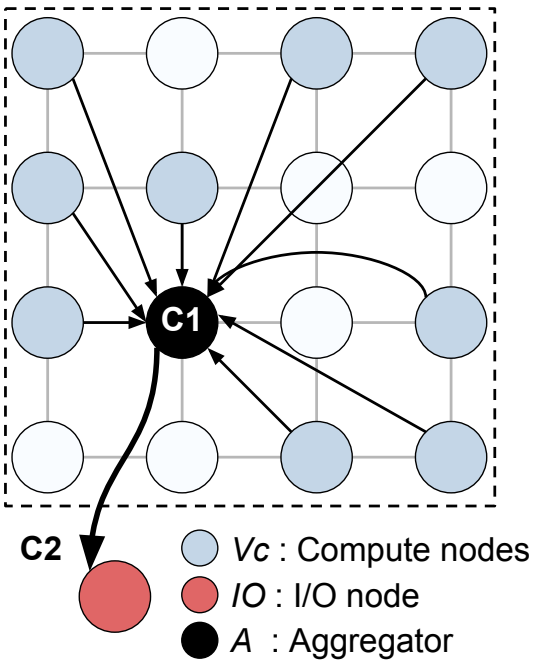


- Computed by each process independently in $O(n), n = |V_C|$
- Contention-aware algorithm: static and dynamic routing policies, unknown vendors information such as routing policy or data distribution among I/O nodes, ...

Abstractions for Interconnect Topology - Our current approach

- TAPIOCA features a topology-aware aggregator placement
- This approach is based on quantitative information easy to gather: latency, bandwidth, distance between nodes

- $\omega(u, v)$: Amount of data exchanged between nodes u and v
- $d(u, v)$: Number of hops from nodes u to v
- l: The interconnect latency
- $B_{i \to j}$: The bandwidth from node i to node j
- $\mathbf{C_1} = \sum_{i \in V_C, i \neq A} \left(l \times d(i, A) + \frac{\omega(i,A)}{B_{i \to A}} \right)$
- $\mathbf{C_2} = l \times d(A, IO) + \frac{\omega(A,IO)}{B_{A \to IO}}$
- **TopoAware(A)** $= min\,(C_1 + C_2)$



- Computed by each process independently in $O(n), n = |V_C|$
- Contention-aware algorithm: static and dynamic routing policies, unknown vendors information such as routing policy or data distribution among I/O nodes, ...

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*
- Two data layout: array of structures (AoS) and structure of arrays (SoA)

Experimental setup

- Theta, a **Cray XC40** supercomputer with a Lustre filesystem
  - Single shared file stripped among OST
  - 48 OST, 16MB stripe size, 4 aggr. per OST, 16MB buffer size (Table 2)
- Mira, an **IBM BG/Q** supercomputer with GPFS
  - One file per *Pset* (128 nodes)
  - 16 aggregators per Pset, 16MB buffer size (MPI I/O configuration)
- Average and standard deviation on 10 runs

Table: Lustre: ratio "Aggregator buffer size : Stripe size"

Ratio	1 : 8	1 : 4	1 : 2	1 : 1	2 : 1	4 : 1
I/O Bw (GBps)	0.36	0.64	0.91	1.57	1.08	1.14

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*
- Two data layout: array of structures (AoS) and structure of arrays (SoA)

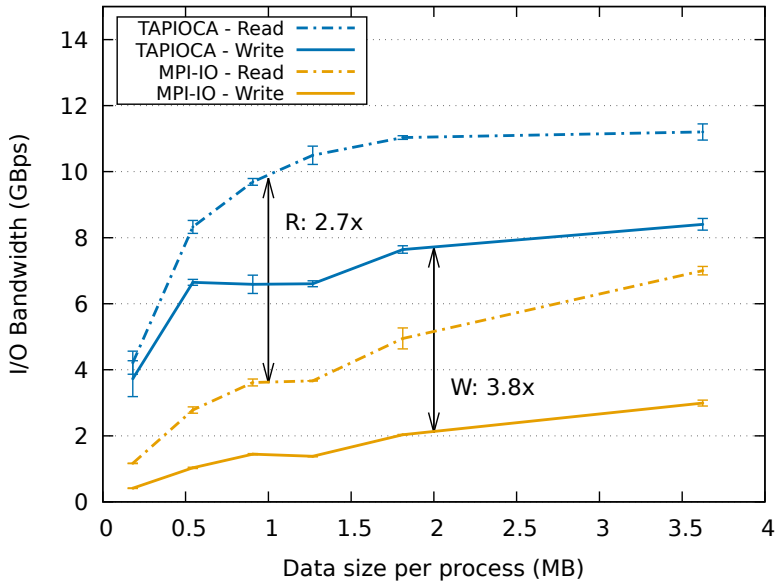
Experimental setup

- Theta, a **Cray XC40** supercomputer with a Lustre filesystem
  - Single shared file stripped among OST
  - 48 OST, 16MB stripe size, 4 aggr. per OST, 16MB buffer size (Table 2)
- Mira, an **IBM BG/Q** supercomputer with GPFS
  - One file per *Pset* (128 nodes)
  - 16 aggregators per Pset, 16MB buffer size (MPI I/O configuration)
- Average and standard deviation on 10 runs

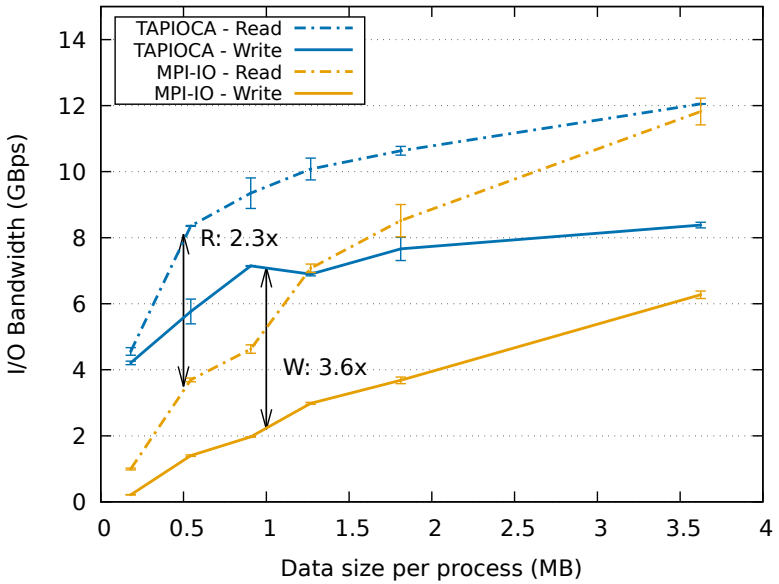
Table: Lustre: ratio "Aggregator buffer size : Stripe size"

Ratio	1 : 8	1 : 4	1 : 2	1 : 1	2 : 1	4 : 1
I/O Bw (GBps)	0.36	0.64	0.91	1.57	1.08	1.14

HACC-IO on Cray XC40 + Lustre



(a) Array of Structures data layout

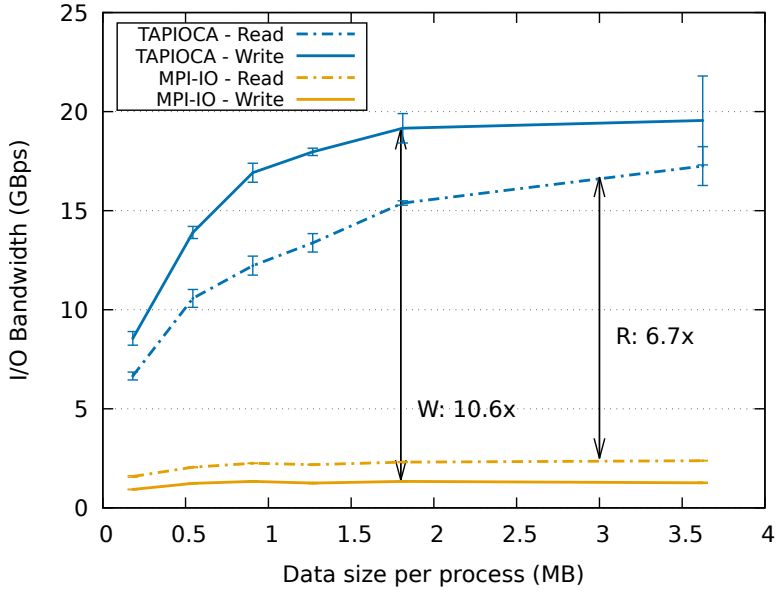


(b) Structure of Arrays data layout

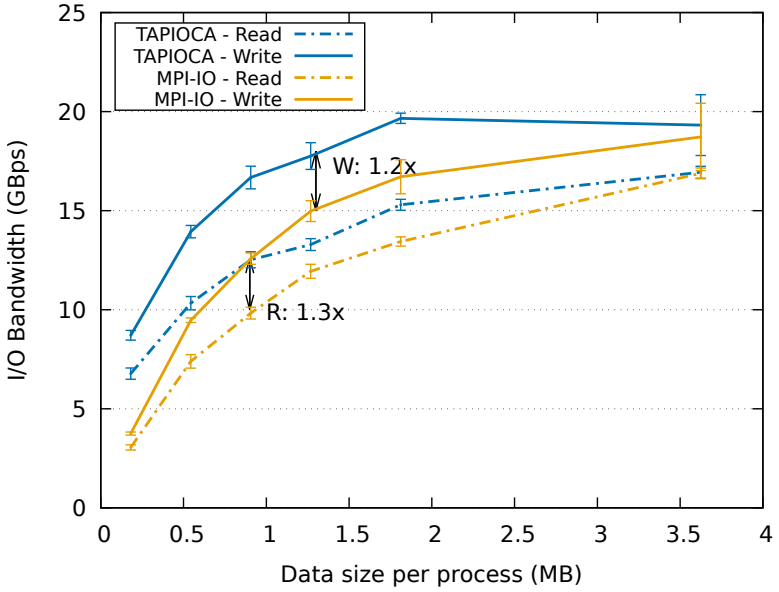
Figure: 1024 Theta-nodes (KNL), 16 ranks/node - 48 OSTs, 16MB stripe size, 192 aggregators - **TAPIOCA**: 16MB aggregators buffer size, 192 aggregators

- I/O bandwidth increased by a factor of **3x** with an AoS data layout
- Significant improvement on smaller messages for the SoA case (up to **3.6x**)
- Same I/O performance improvement on 2048 nodes

HACC-IO on IBM BG/Q + GPFS



(a) Array of Structures data layout



(b) Structure of Arrays data layout

Figure: 1024 Mira-nodes, 16 ranks/node, subfiling - 16 aggragators/*Pset*, 16MB aggregators buffer size

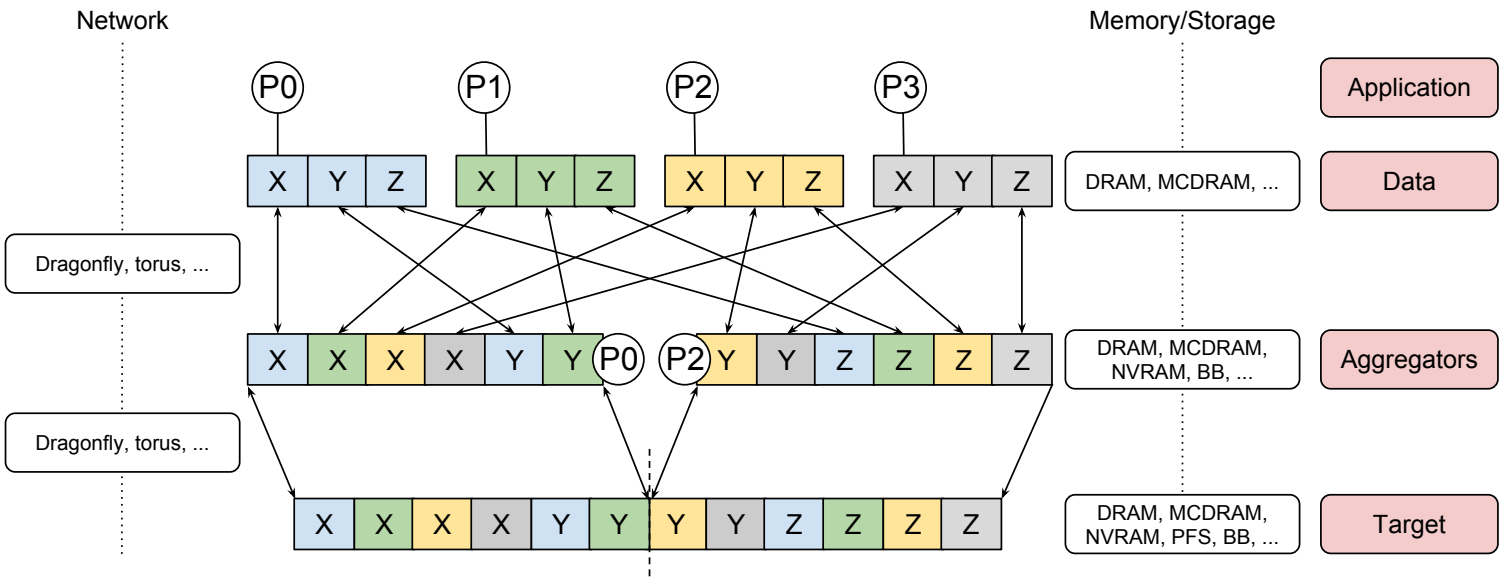
- **90%** of the peak I/O bandwidth (22.4 GBps) achieved with TAPIOCA
- Large gap between MPI I/O and TAPIOCA on an AoS data layout (I/O scheduling)
- Scalable algorithm tested on 4096 nodes with similar results

Conclusion

- TAPIOCA, an optimized I/O library incorporating
  - Topology-aware aggregator placement
  - Optimized data movement with I/O scheduling and pipelining
  - Interconnect abstraction
- Very good performance at scale, outperforming MPI I/O
- On HACC-IO, up to **10.6x** improvement on IMB BG/Q+GPFS and up to **3.8x** on a Cray XC40+Lustre supercomputer
- Scalability evaluated on more than 65K ranks
- Performance portability on two leadership-class supercomputers
  - Same application code running on both platforms
  - Same optimization algorithms using an interconnect abstraction

Future Work

- Move toward a **generic data movement library for data-intensive applications** exploiting deep memory/storage hierarchies as well as interconnect to facilitate I/O, in-transit analysis, data transformation, data/code coupling, workflows, ...



Conclusion

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 (JLESC)
- Proactive Data Containers (PDC) project

Thank you for your attention!

ftessier@anl.gov

Argonne NATIONAL LABORATORY