PARALLEL SOFTWARE TECHNOLOGIES LABORATORY PSTL

On Overlapping Communication and File I/O in Collective Write Operation

Raafat Feki and Edgar Gabriel

Parallel Software Technologies Lab
Department of Computer Science,
University of Houston, Houston, USA
Email: {rfeki, egabriel}@uh.edu

UNIVERSITY of HOUSTON

Motivation

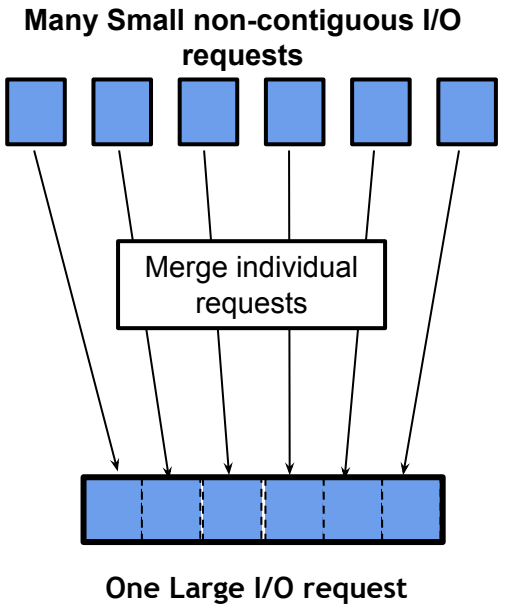
Many scientific applications operate on data sets that span hundreds of Gigabytes/Terabytes in size ⇒ A significant amount of time is spent in reading and writing data.

The Message Passing Interface (MPI) is the most widely used parallel programming paradigm for large scale parallel applications.

- **MPI I/O:** Parallel file I/O interface

Multiple processes can simultaneously access the same file to perform read and write operations using a shared-file access pattern:

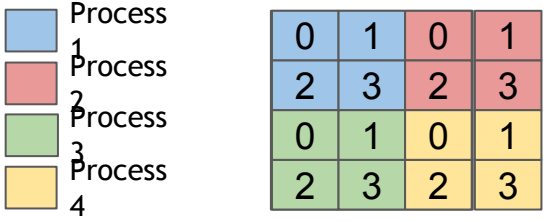
- **Individually:** Each process is responsible of writing/reading his local data independently of the others.
  - ➢ Poor performance for complex data layout.
- **Collectively:** Access information is shared between all processes.
  - ➢ Significantly reduced I/O time

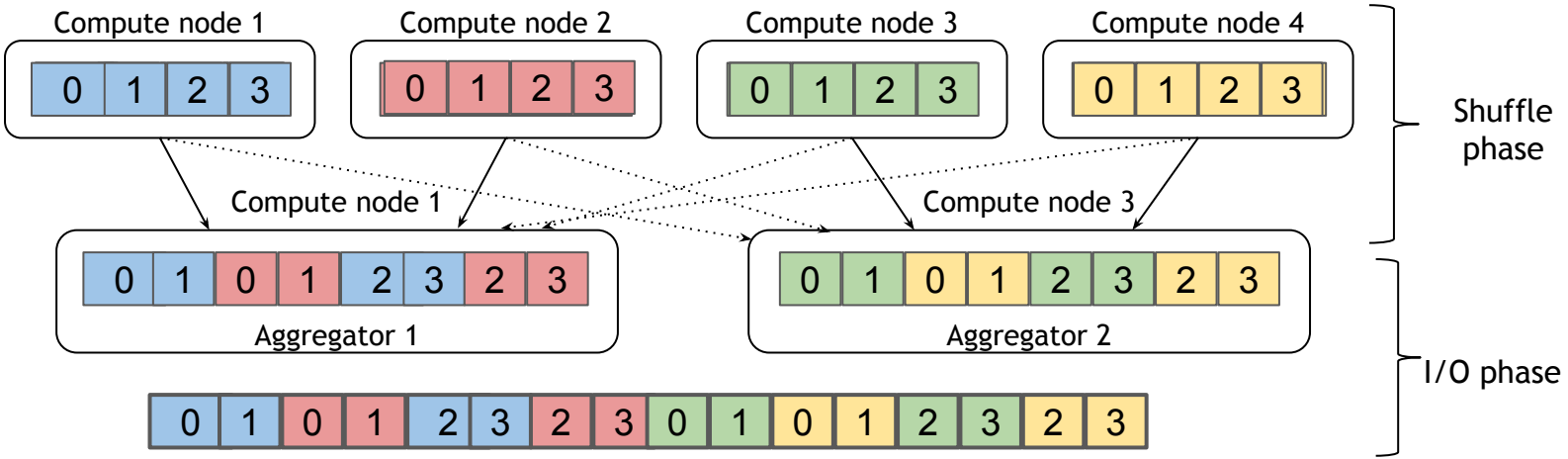


Two-Phase I/O

- Collective I/O algorithm used at the client level
- Only a subset of MPI processes will actually write/read data called Aggregators
- It consist of two Phases:
  - Shuffle/Aggregation phase:
    - Data shuffled and assigned to aggregators.
    - Data sent to aggregators.
  - Access/ I/O phase:
    - Aggregators write/read data into/from disk.

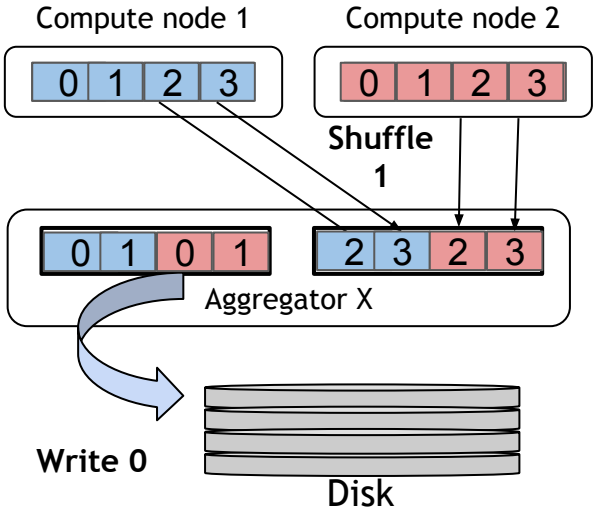
Example: 2D data decomposition





Overlapped Two-Phase (I)

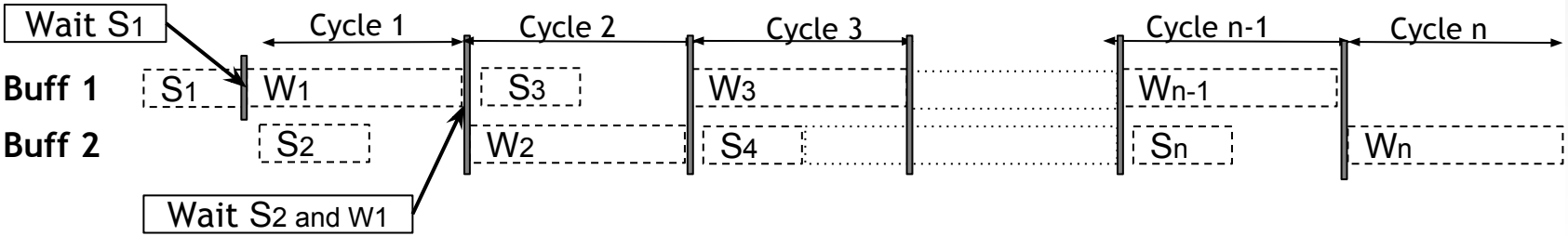
- Divide the buffer into two sub-buffers.
- **Overlap 2 phases:**
  - While the aggregator is writing the data from sub-buff 1 into disk, the compute nodes are shuffling the new data and send it into the second sub-buff.
    - By using Asynchronous operations



We define:

S_n: nth Shuffle phase

W_n: nth Access phase



Overlapped Two-Phase (II)

- There are multiple ways to implement the overlapping technique depending on the choice of:
  - The asynchronous functions (Communication or I/O).
  - The phases that will be overlapped (Aggregation or Access phase).

We proposed four different algorithms:

Algorithm	Overlapped Phases	Communication function	I/O function
1. Communication Overlap	2 Shuffle phases	Asynchronous	Synchronous
2. Writes Overlap	2 Access phases	Synchronous	Asynchronous
3. Write-Communication Overlap	1 Shuffle and 1 access phase	Asynchronous	Asynchronous

4.Write-Communication-2 Overlap: A revised version of the last algorithm that follows a data-flow model:

- ➢ The completion of any non-blocking operation is immediately followed by posting the follow-up operation first.
  - ✓ Two shuffles and two writes operations are handled in each iteration (2 cycles).

Evaluation (I)

We tested the original non-overlapped two-phase I/O and the four overlapping algorithms using:

- Platforms: Crill cluster (University of Houston) & Ibex cluster (KAUST university)
- File system: Beegfs
- Benchmarks: IOR(1D data), TileIO (2D data) and FlashIO (HDF5 output).

TABLE I
NUMBER OF RUNS AN OVERLAP ALGORITHM RESULTED IN BEST PERFORMANCE

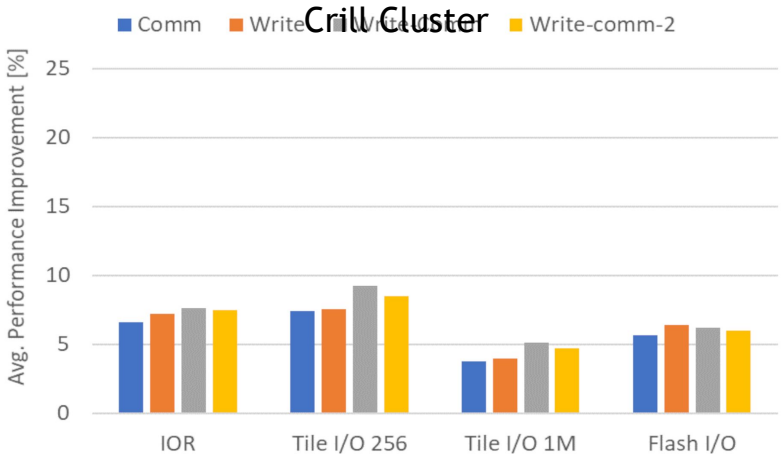
Benchmark	No Overlap	Comm Overlap	Write Overlap	Write-Comm Overlap	Write-Comm 2 Overlap
IOR	21	11	32	28	15
Tile I/O 256	17	13	18	31	26
Tile I/O 1M	10	6	18	20	17
Flash I/O	11	12	11	16	19
Total:	59	42	79	95	77

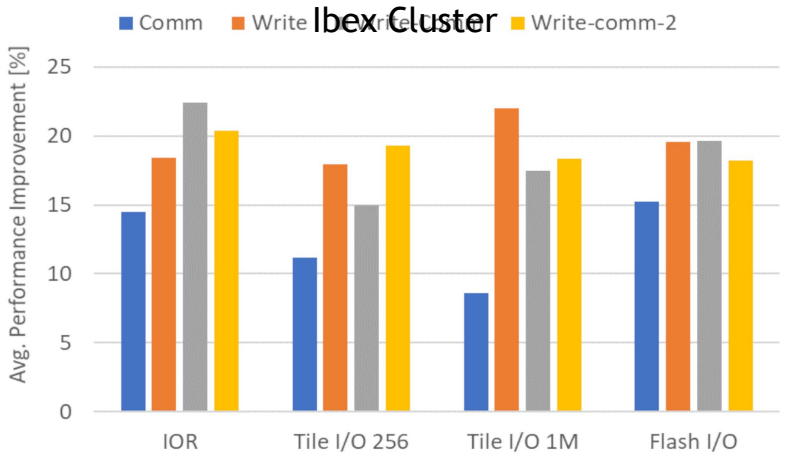
- We cannot identify a “winner” out of different algorithms.
- There is no benefit of overlapping technique in 16% of test cases.
- The algorithms incorporating asynchronous I/O outperformed the other approaches in 71% test series.

⇒ Better performance with **Asynchronous file I/O**

Evaluation (II)

In order to identify the best algorithm, we ran the 4 overlapping algorithms for all the benchmarks:





- The average performance improvement on the Crill clusters are very close for all versions with a slight advantage for the communication overlap version.
- The results on Ibex are more clear and shows a clear win of the communication overlap version

Data Transfer Primitives

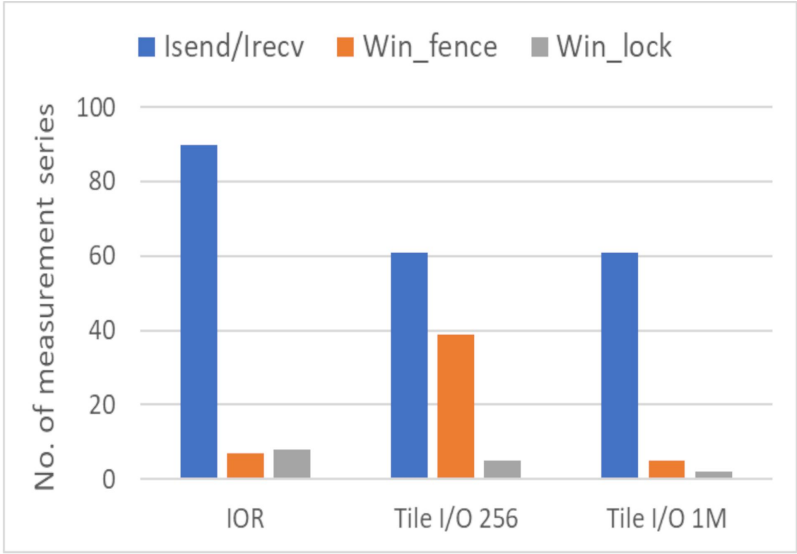
We investigated two communication models for the shuffle phase implementation:

- **Two Sided communication:** Currently implemented in the two-phase algorithm.
  - Data sent from MPI processes using MPI_send/Isend to receiver (aggregators).
  - Aggregators receive data into local buffers using MPI_recv/Irecv.

- **Remote memory access (RMA):**
  - Each process exposes a part of its memory to other processes
  - Only one side (Sender/Receiver) is responsible of data transfer (using resp Put()/Get()):
    - ➢ To alleviate the workload on the aggregator, we chose to make the senders “put” their data into the aggregators memory.
  - We used 2 synchronization methods to guarantee data consistency.
    - ➢ Active target synchronization (MPI_Win_fence)
    - ➢ Passive target synchronization(MPI_Win_lock/unlock())

Evaluation (III)

- Two-sided data communication outperformed the one-sided versions in 75% of the test-cases.
- When using Tile I/O with small element size of 256 Byte, the version using MPI_Win_fence achieved the best performance in 37% of the test cases.
  - The performance gain over two-sided communication was around 27% in these cases.
  - The benefits of using one-sided communication increased for larger process counts.



Number of times each of the three different data transfer primitives resulted in the best performance.

Conclusion and Future Work

- Conclusion:
  - Proposed various design options for overlapping two-phase I/O.
    - ➢ Overlap algorithms incorporating asynchronous I/O operations outperform other approaches and offer significant performance benefits of up to 22% compared to non-overlapped two-phase I/O algorithm.
  - Explored two communication paradigm for the shuffle phase:
    - ➢ One-sided communication did not lead to performance improvements compared to two-sided communication.
- Future work:
  - Running the same tests on the Lustre file system showed a total different results since it only supports blocking I/O functions.
    - ➢ Explore the lustre advanced reservations solution (Lockahead).