Towards Parallel Access of Multi-dimensional, Multi-resolution Scientific Data

Sidharth Kumar

Argonne NATIONAL LABORATORY

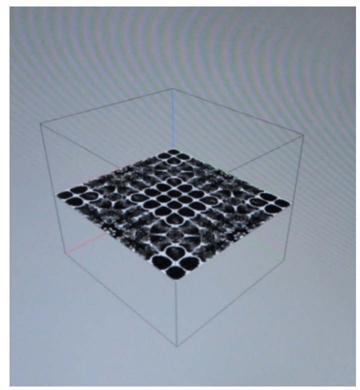
SC10 New Orleans, LA

SCI www.sci.utah.edu

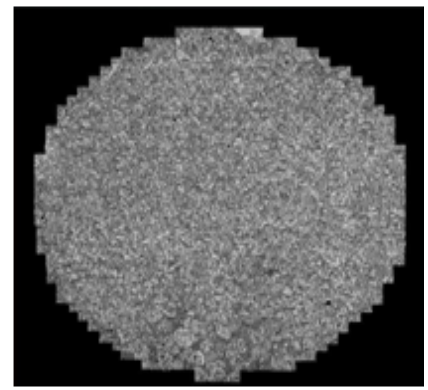
ViSUS : IDX Data Format

ViSUS : Technology to Analyze and Visualize Multi-dimensional data
IDX : Data type generated by ViSUS i/o API

iPhone Application



Visualizing 3D Data



Visualizing 2D Data

Applications in Digital Photography

IDX Data Type

- ***Cache Friendly***
  - **H**ierarchical **Z** Ordering
- ***Progressive access***
  - Multiple Levels of Resolution

HZ Ordering

Input Data stored in normal XY Ordering

13	12	14	15
8	9	10	11
4	5	6	7
0	1	2	3

XY Location

IDX Data Stored in HZ ordering

10(4)	11(4)	14(4)	15(4)
2(2)	5(3)	3(2)	7(3)
8(4)	9(4)	12(4)	13(4)
0(0)	4(3)	1(1)	6(3)

Assigned HZ Order (Level)

HZ Order = compute HZ(X, Y)

HZ Level = floor ((log$_2$ (HZ Order))) + 1

HZ Ordering

Input Data stored in normal XY Ordering

IDX Data Stored in HZ ordering

13	12	14	15
8	9	10	11
4	5	6	7
0	1	2	3

XY Location

10	11	14	15
2	5	3	7
8	9	12	13
0	4	1	6

Assigned HZ Order (Level)

4	4	4	4
2	3	2	3
4	4	4	4
0	3	1	3

HZ Level

HZ Order = compute HZ(X, Y)

HZ Level = floor ((log$_2$ (HZ Order))) + 1

IDX File Format

Progressive access : Multiple Levels of Resolution

Level 0

Level 1

Level 2

Level 3

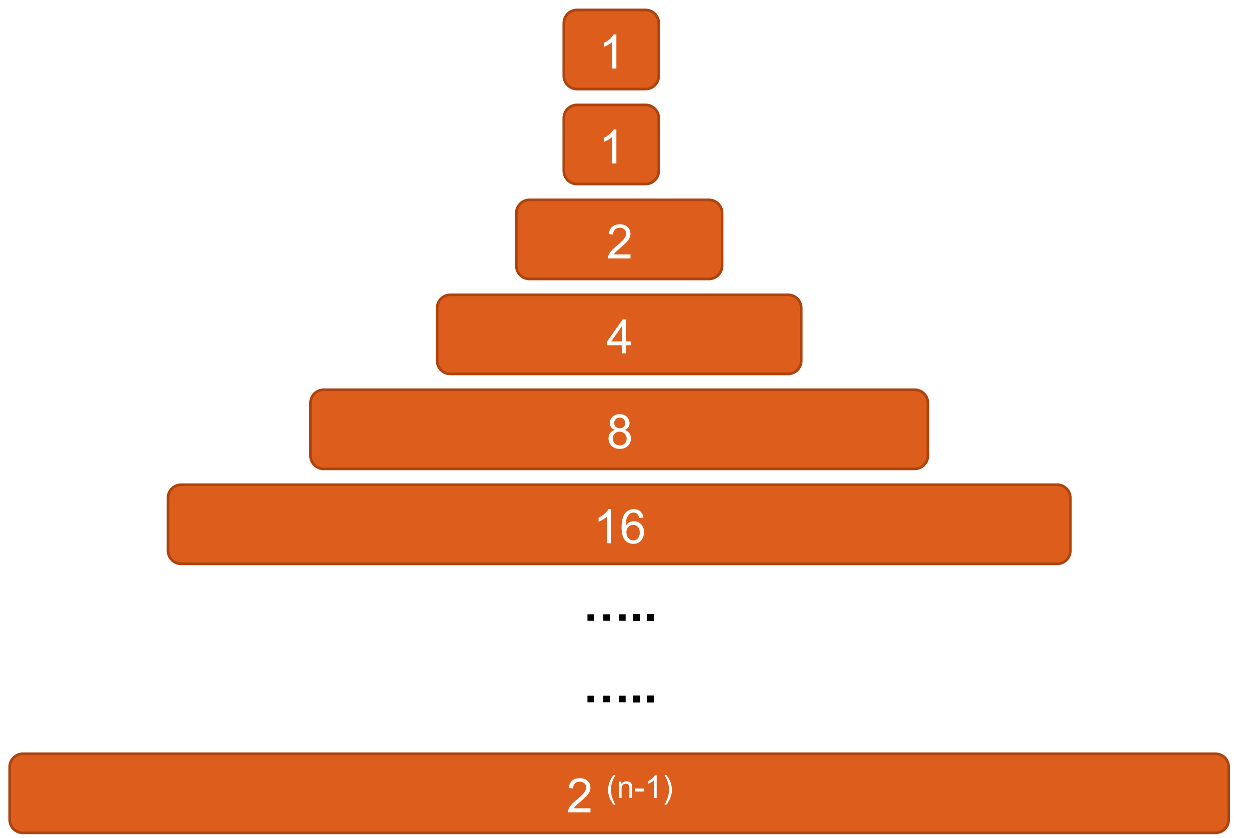
Level 4

Level 5

…..

…..

Level **n**



Motivation: IDX in HPC Application

HPC simulations generate enormous amounts of **Scientific Data**

Analysis and visualization of the data is a limiting factor in scientific research

IDX data format is promising in this scenario

- Interactive navigation of simulation data.
- Real-time Zoom in on regions of interest.

Motivation: Parallelizing ViSUS

Problem with current implementation

Existing tools for writing/reading IDX data only provides a serial interface.

HPC applications fails to utilize available parallel I/O resources.

Solution

Develop methods for writing IDX data in parallel

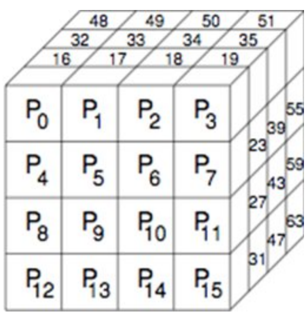
Enable HPC applications to write IDX data with scalable performance

Blue Gene/P : Making ViSUS scalable to run on Large Parallel Machines

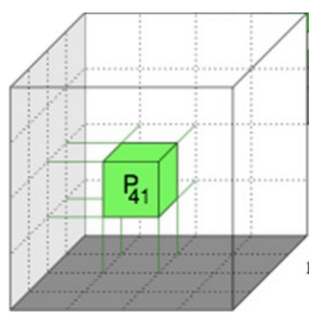
ViSUS : Serial Writer

Parallel application using ViSUS I/O to write directly into IDX format.

Divides the entire data volume into smaller 3D chunks



Each process independently writes to an IDX data set



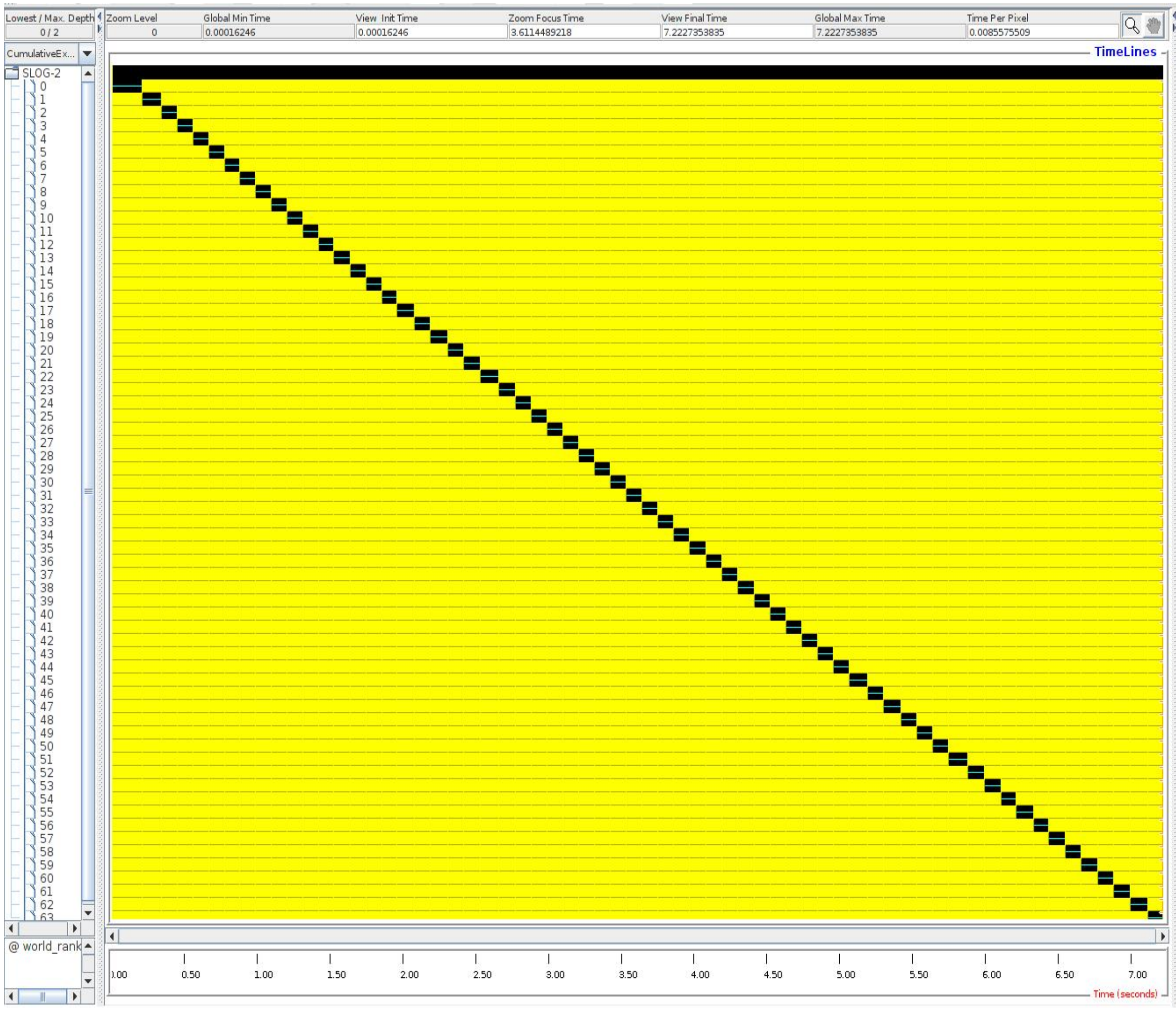
Visus Writes

Process with rank **r** writes to an IDX file only after the process with rank **r–1** has finished writing.

The processes cannot write concurrently due to conflicts in updating metadata and block layouts.

ViSUS Serial Writer : Performance

Processes



Time

MPI_Barrier

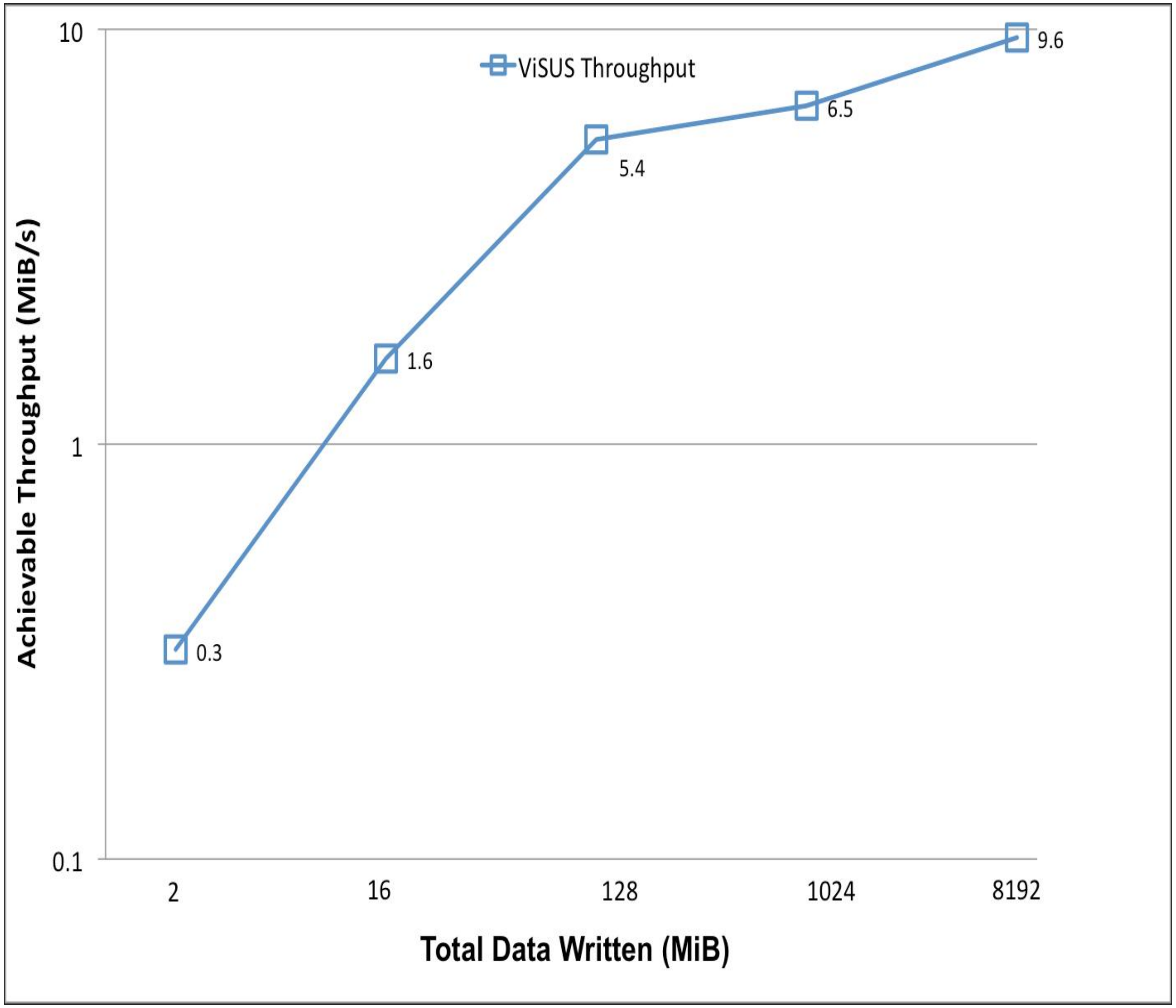
ViSUS Write

64 Processes : 2 MiB

Total Time = 7 Sec

Speed = 2.9 MiB/ Sec

ViSUS Serial Writer : Throughput



Best performance : 9.5MiB/s (8 GiB)

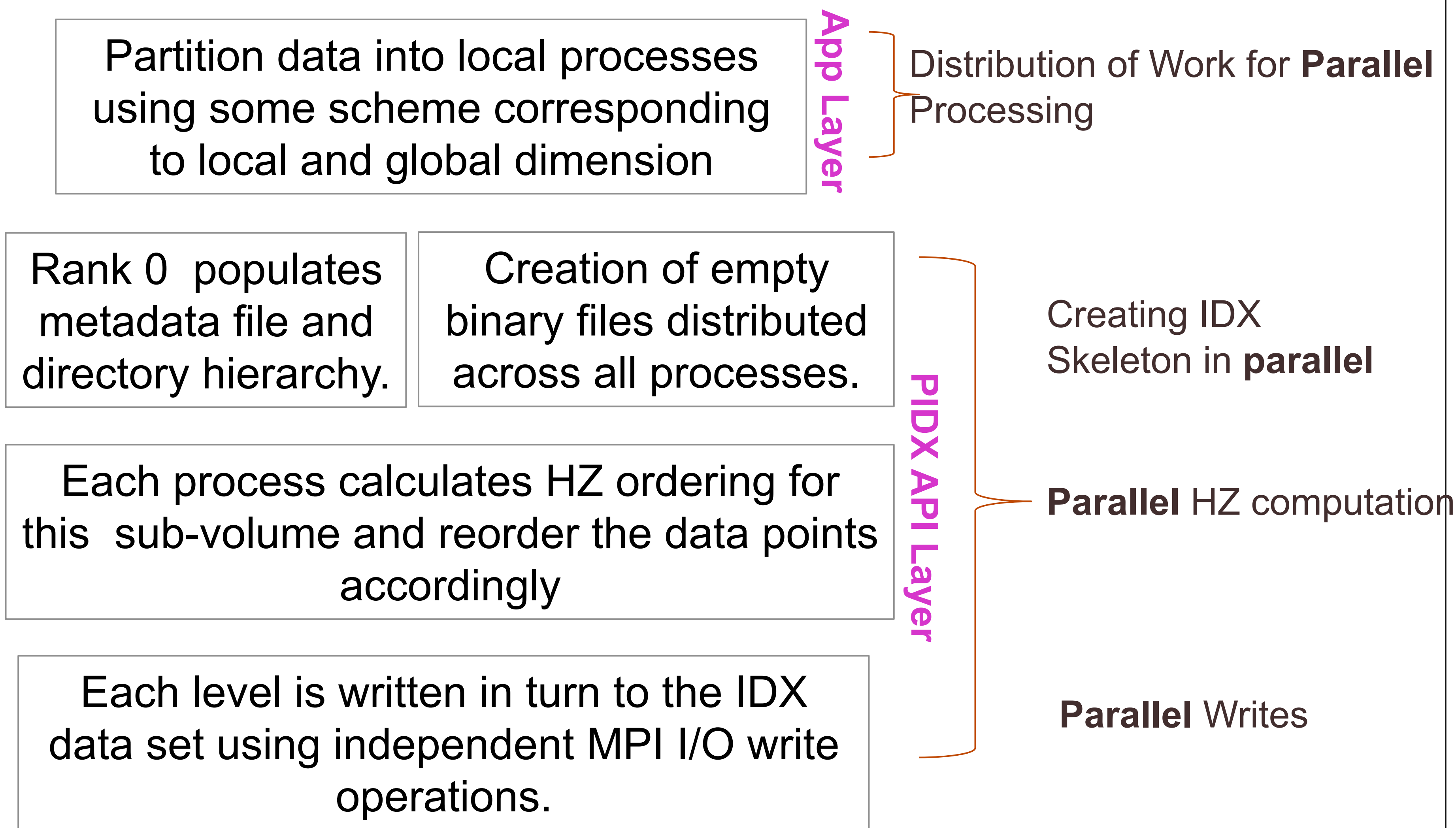
IOR Maximum Throughput: 218MiB/s (8GiB)

(4% of the max throughput)

PIDX : Prototype Parallel IDX Write API

- Concurrent I/O to an IDX data set.
- Functions patterned after ViSUS for ***creating***, ***opening***, ***reading***, and ***writing*** IDX data sets.
- PIDX functions performs collective operation by accepting an MPI communicator as an argument.

Parallel IDX Write



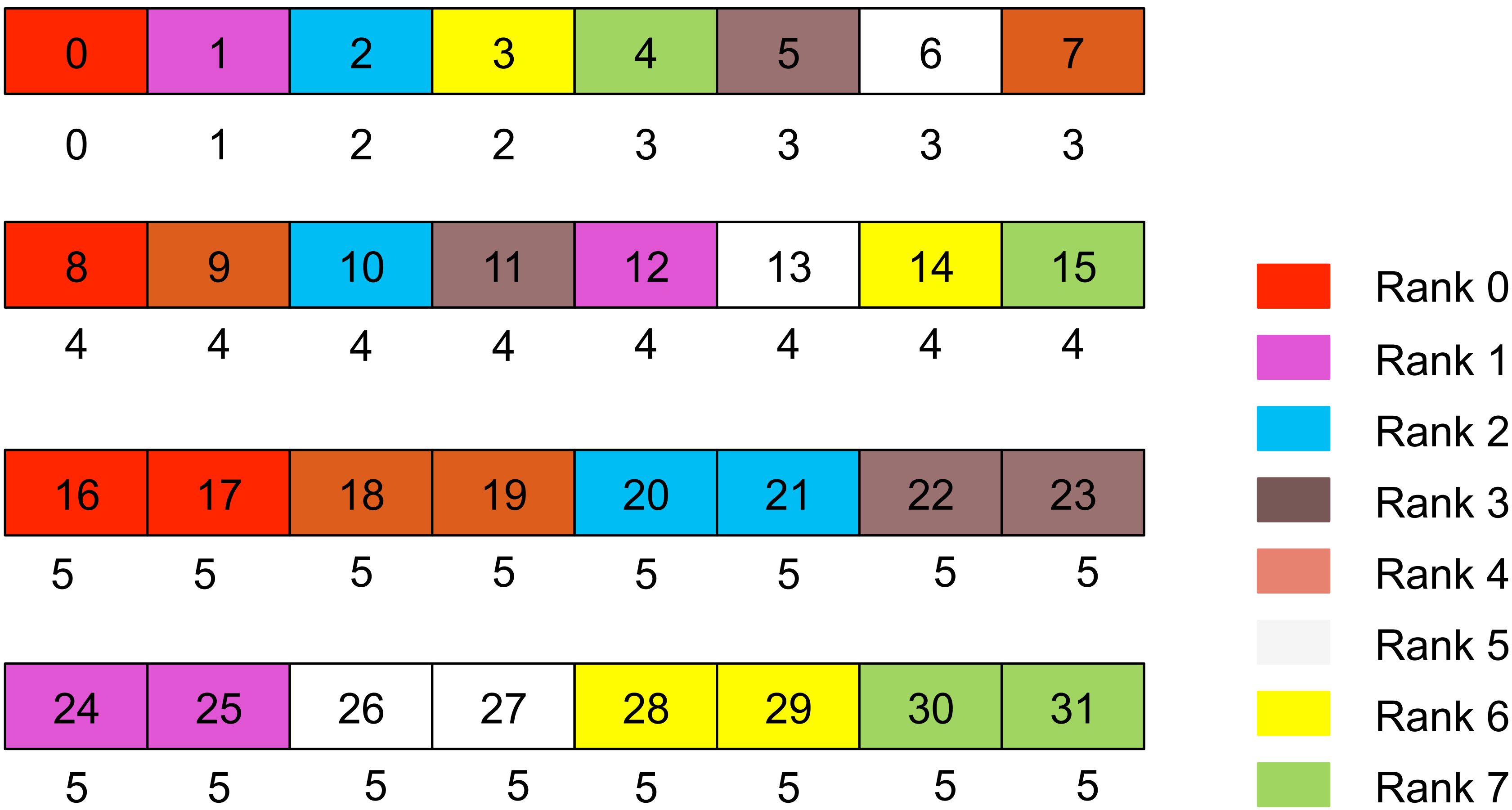
IDX : File Structure

0	1	2	3	4	5	6	7	HZ Order
0	1	2	2	3	3	3	3	HZ Level
8	9	10	11	12	13	14	15	
4	4	4	4	4	4	4	4	
16	17	18	19	20	21	22	23	
5	5	5	5	5	5	5	5	
24	25	26	27	28	29	30	31	
5	5	5	5	5	5	5	5	
32	33	34	35	36	37	38	39	
6	6	6	6	6	6	6	6	
40	41	42	43	44	45	46	47	
6	6	6	6	6	6	6	6	
48	49	50	51	52	53	54	55	
6	6	6	6	6	6	6	6	
56	57	58	59	60	61	62	63	
6	6	6	6	6	6	6	6	

64 Elements
7 Levels (0 inc)

PIDX : Discontinuous in File System

64 Elements **7** Levels (0 inc) **8** Processes **8** Elements / Proc



PIDX : Discontinuous in Memory

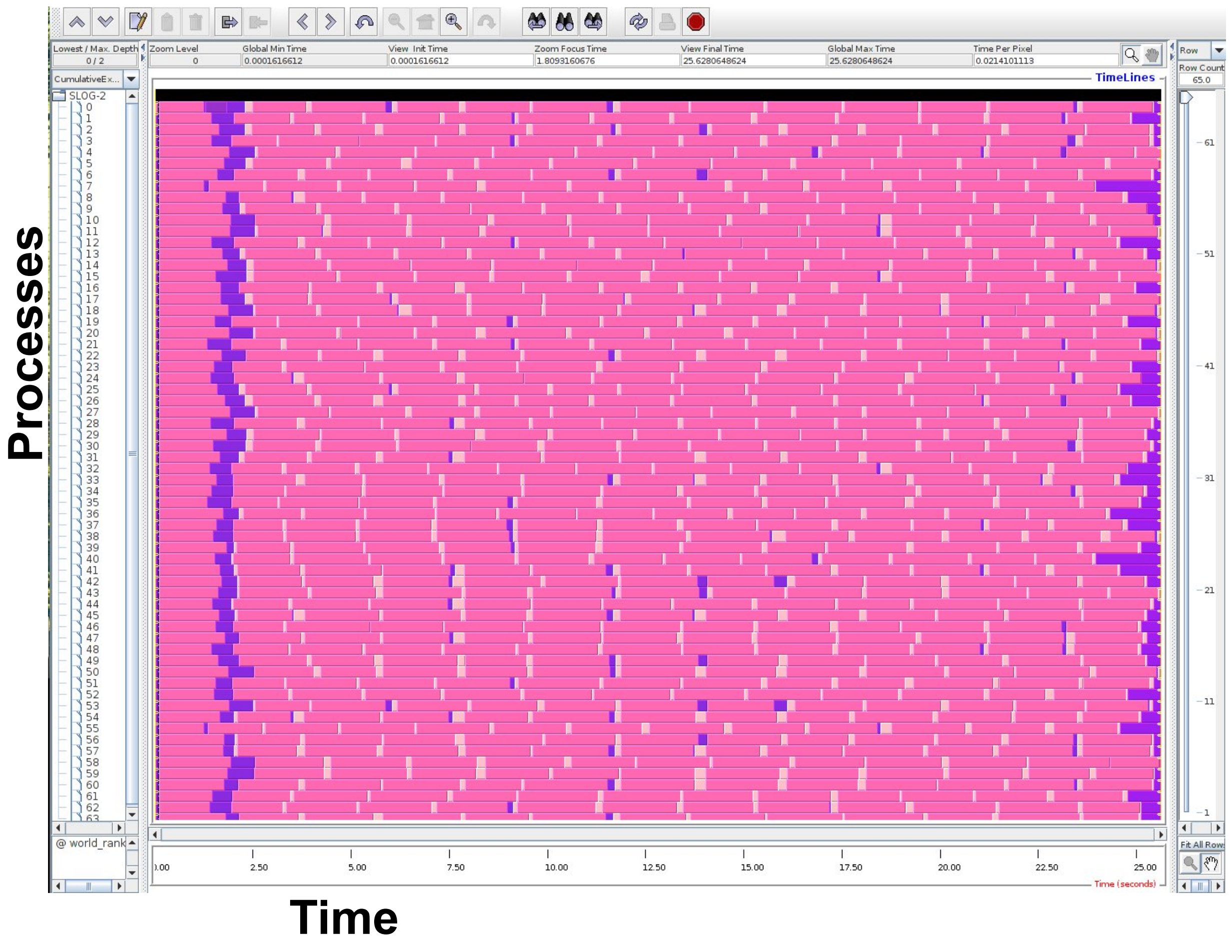
HZ Order	0	…	8	…	16	17	…	32	33	34	35
HZ Level	**0**		**4**		**5**	**5**		**6**	**6**	**6**	**6**

Data arrangement of **Rank 0**

Data is discontinuous in memory as well

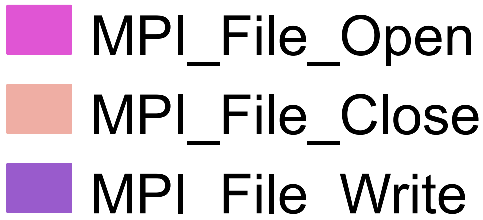
Continuous chunks of elements exists per level

ViSUS Parallel Writer : Performance



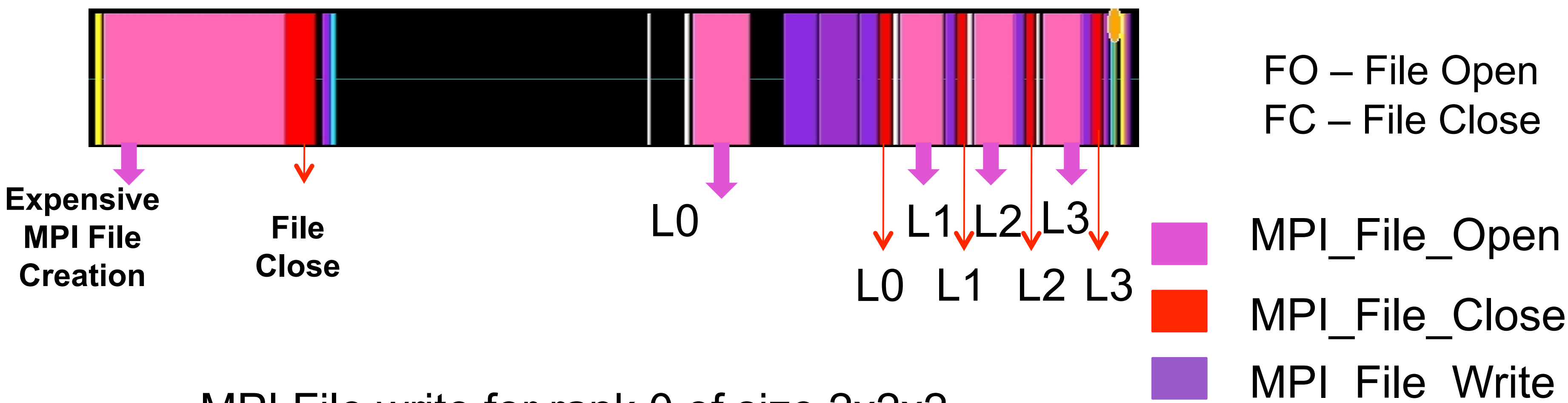
Concurrent Data Writes

Large Time Spent in File open and File Close

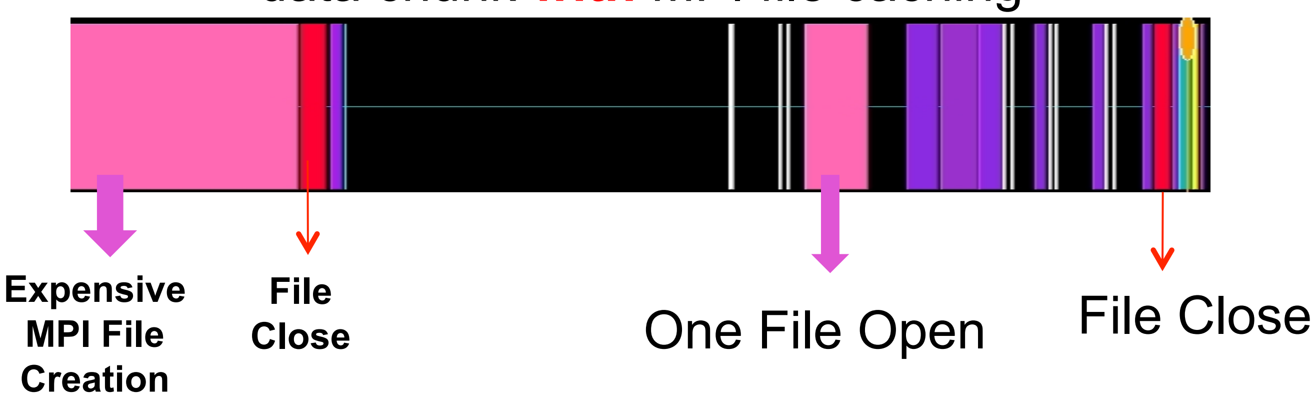


MPI File Caching

MPI File write for rank 0 of size 2x2x2 data chunk **without** MPI file caching

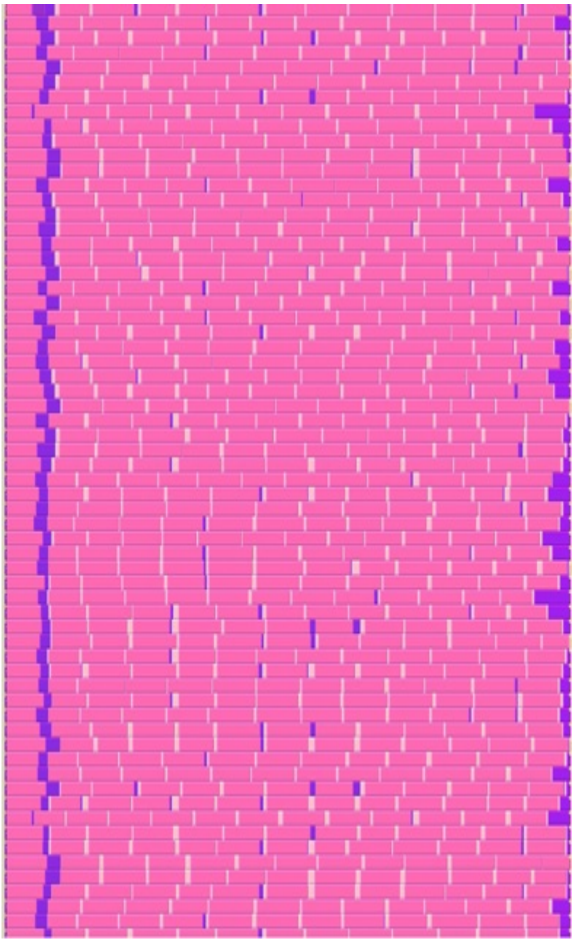


MPI File write for rank 0 of size 2x2x2 data chunk **with** MPI file caching

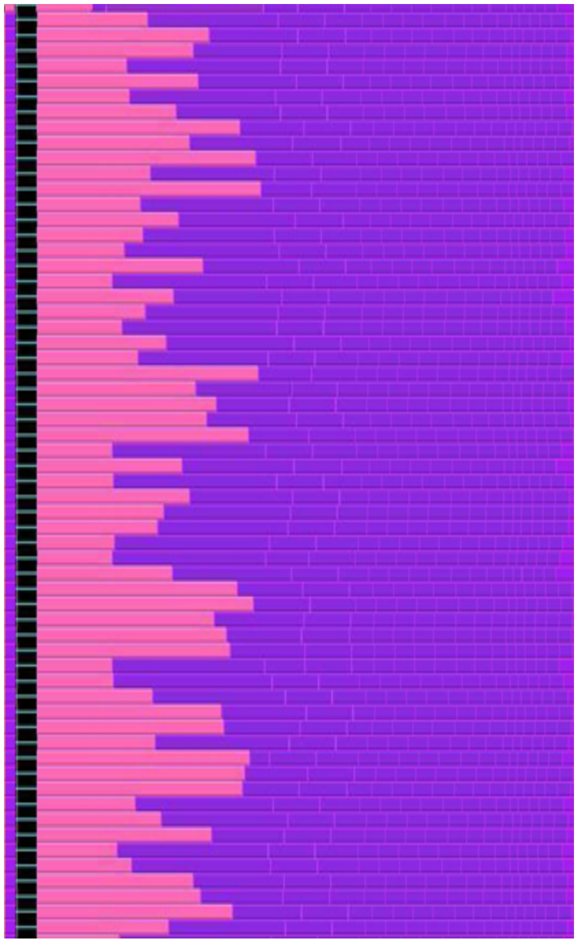


MPI File Caching Saves on 3 File opens.

Effect of MPI File Caching



Before



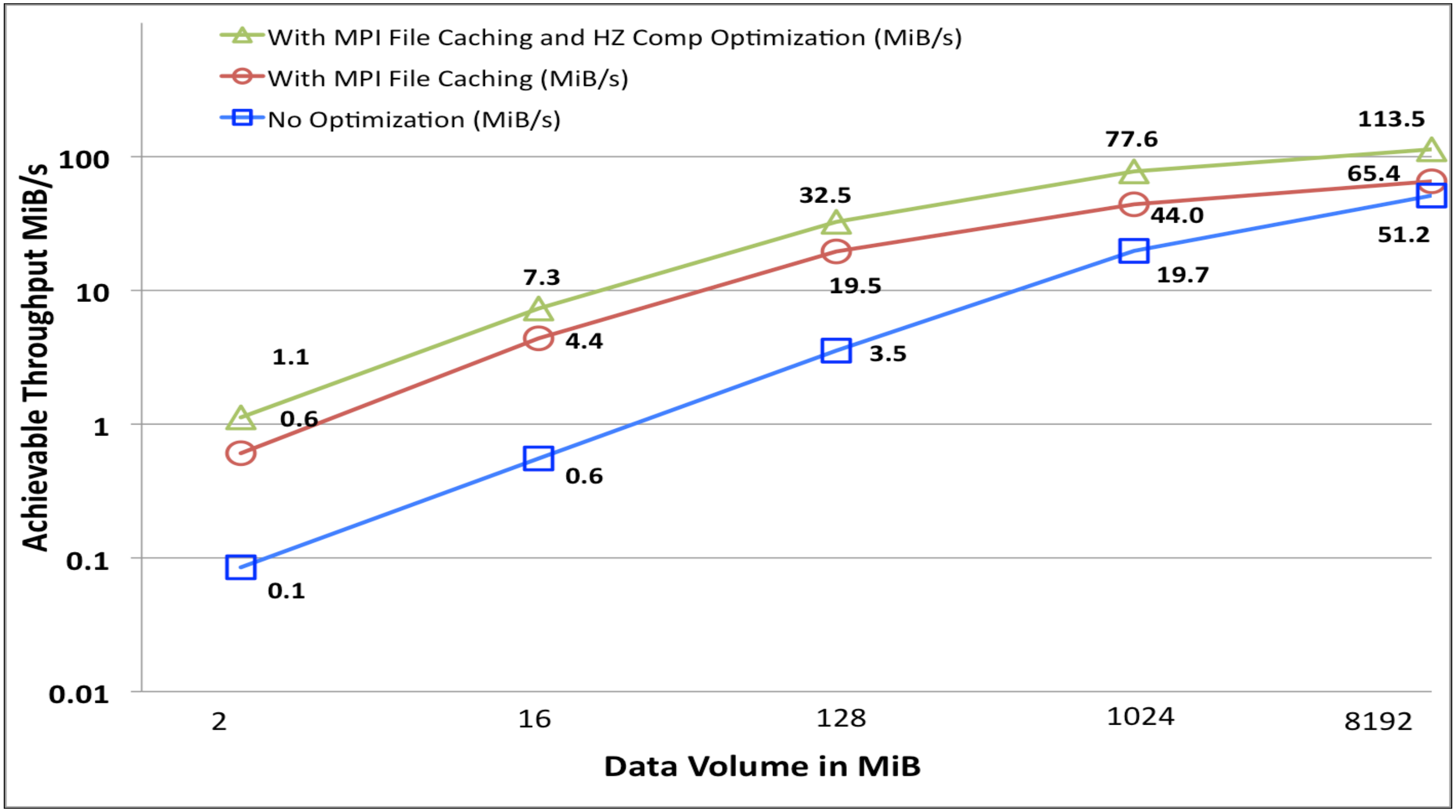
After

Total Data Written	Speed with MPI File Caching (MiB/S)	Speed with out MPI File Caching (MiB/S)
8 GiB	65	51
1 GiB	44	19
128 MiB	19.5	3.5

HZ optimization

- Significant amount of the I/O time spent in the computation to generate the HZ ordering.
- Identification of bottlenecks associated with redundant computations.
- 75% improvement in I/O throughput over the file handle improvements and up to a 10-fold improvement over the default implementation.

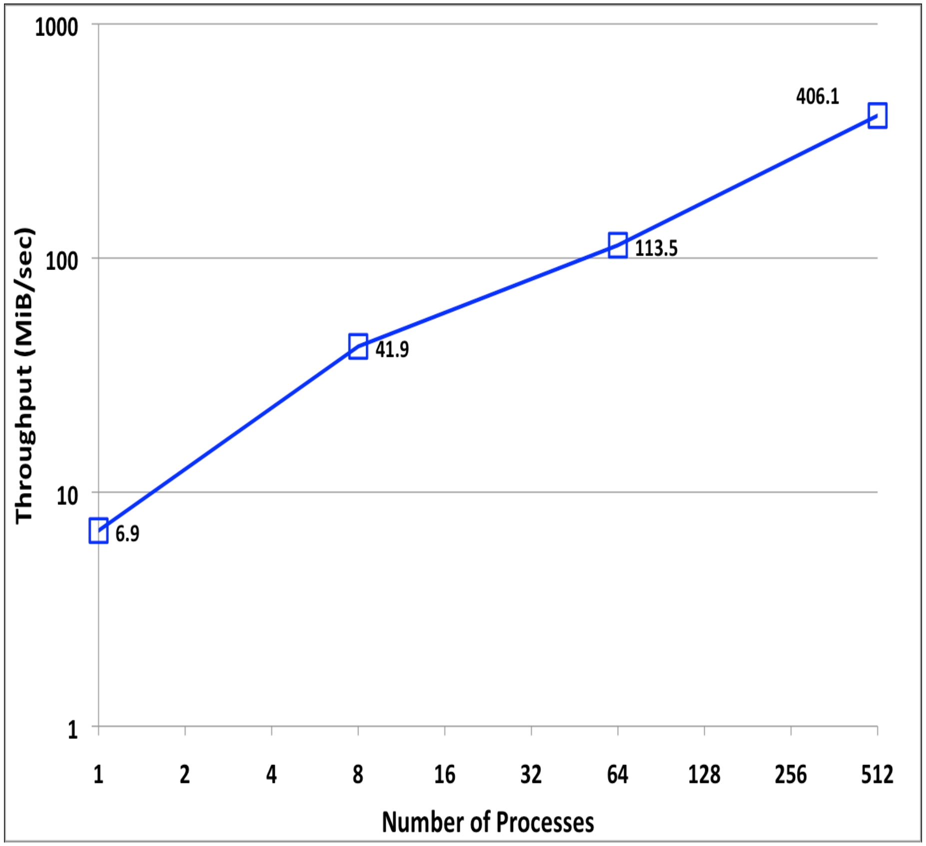
Optimizations



Two Fold improvement over default implementation writing 8GiB data using 64 nodes.

Scalability Analysis – Weak Scaling

- Constant load of 128 MB per process.
- Processes varied from 1 to 512.
- 1 process achieves 6.85 MiB/s, comparable to the speed of serial writer for an equal volume of data.
- The peak aggregate performance of 406 MiB/s is reached with 512 processes. This is approximately 60% of the peak IOR throughput achievable (667MiB/s) on 512 cores of surveyor.



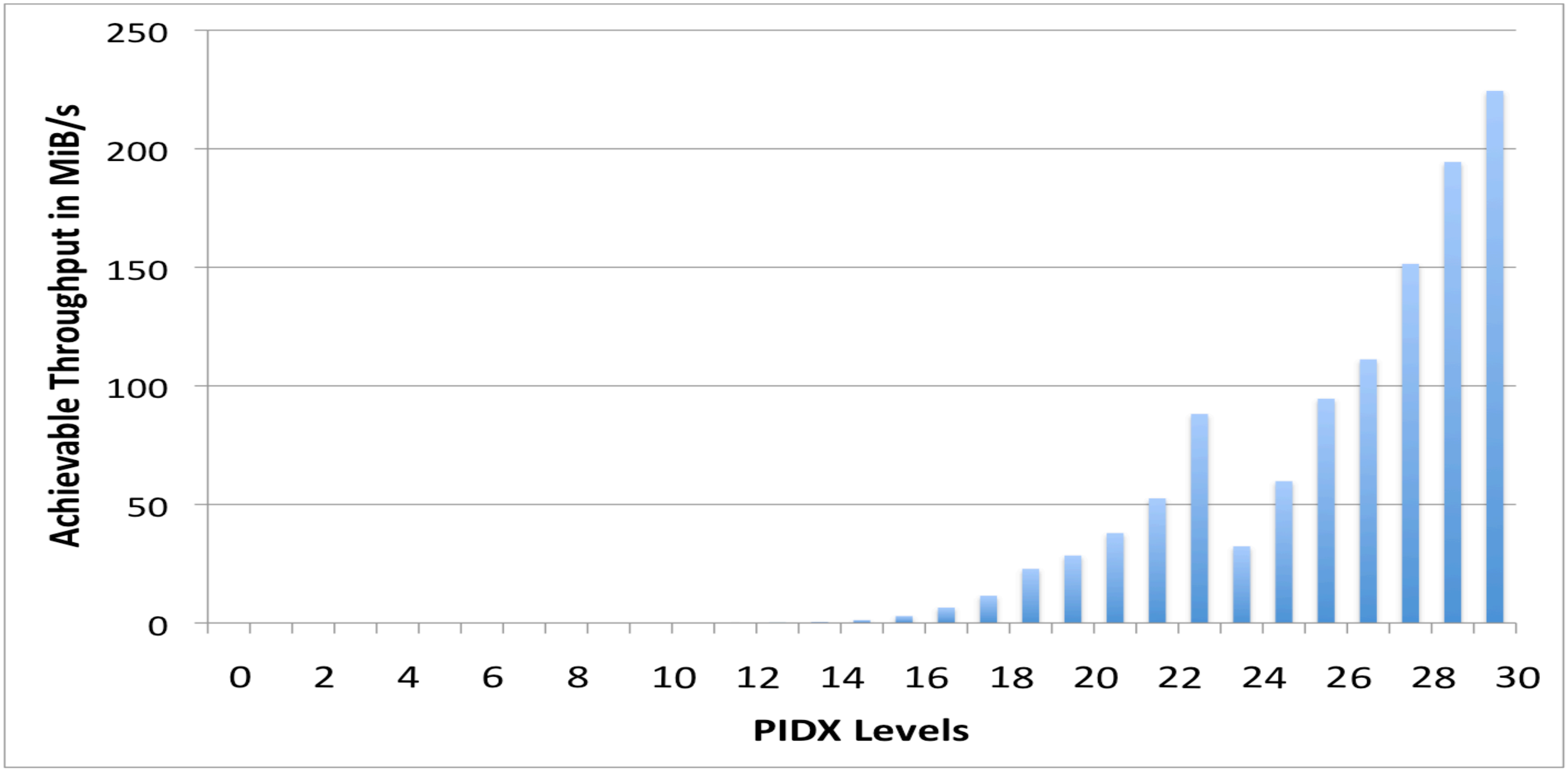
Scalability Analysis – Strong Scaling

Number of Processes	PIDX Throughput in MiB/s
64	120.3
512	143.9

Total Data Volume Of 8GiB

Maximum throughput achieved with 512 processes

PIDX Analysis



- Low throughput for levels up to 16.
- Limit hit on scalability with current implementation, falling short of the peak surveyor write performance achieved by IOR.
- Desired throughput achieved only at higher levels

Proposed Solution

Problem
Contention and metadata overhead caused levels 0 through 14 to take disproportionate amount of time relative to the amount of data that they were writing.

Solution
Plan to leverage aggregation strategies to better coordinate I/O in the first few cases where many processes contribute data to the same level of the IDX data set.

Conclusion

- Completed parallel write of IDX data format
  - Achieved a significant fraction of peak performance, at least at moderate scale
- Discovered overhead from unexpected sources in metadata operations and HZ computation
- More work needed to implement aggregation routines.

Project Members

SCI

Sidharth Kumar
Valerio Pascucci

Argonne National Laboratory

Venkatram Vishwanth
Phil Carns
Mark Hereld
Robert Latham
Tom Peterka
Michael E. Papka
Rob Ross

Thank You!!!!!!!!

Questions

?