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

Sidharth Kumar, Venkatram Vishwanath, Philip Carns,
Valerio Pascucci ,Robert Latham, Tom Peterka, Michael Papka, Robert Ross

Motivation

IDX enables fast and efficient access to large scale scientific data.

Problem with current implementation

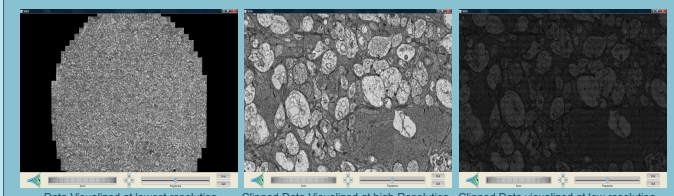
- Existing tools for writing IDX data only provides only a serial interface.
- HPC applications fails to utilize available parallel I/O resources.

Solution

- Investigate methods for writing IDX data in parallel
- Enable HPC applications to write IDX data with scalable performance

Visus (IDX Format)

Multiple levels of resolution : Easy to query data of desired resolution and dimension.



Data Visualized at lowest resolution | Clipped Data Visualized at high Resolution | Clipped Data visualized at low resolution

(530 Mib Data Set of Rat's retina scan)

Hierarchical Z (HZ) ordering - The crux behind IDX data format, computation requires spatial coordinates of data.

X	Y	Z	XYZ Value	Z Order	H Z	Level
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	2	2	2	2
0	1	1	3	3	3	2
1	0	0	4	4	4	3
1	0	1	5	5	6	3
1	1	0	6	6	5	3
1	1	1	7	7	7	3

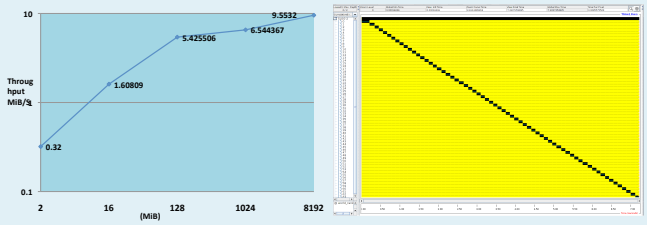
Data is then reorganized into levels corresponding to the formulation:

$$Level = floor\ ((log2\ (HZ\ index))) + 1$$

Level corresponds to different resolution level the data is rearranged into.

Visus Serial IDX Writer

A micro benchmark that divides entire data volume into smaller 3D chunks. Each process independently writes to the IDX data set. No concurrent writes due to conflicts in updating metadata and layout. MPI barriers and tokens used to maintain order amongst processes

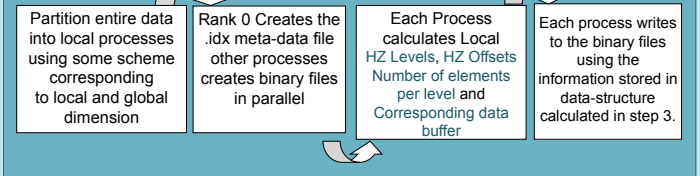


•The maximum performance is only 9.5 MiB/s. Using IOR, we obtain a peak performance of 218MiB/s for 64 processes writing a total of 8GiB. Thus, we are able to achieve only 4% of the maximum throughput.

•Jumpshot profile of 64 processes writing an IDX file using ViSUS. Each horizontal line represents a process, the yellow regions correspond to MPI barrier waits and black regions correspond to time spent writing data. As expected, we notice that a large portion of the runtime for each process is spent waiting for the I/O token.

PIDX : Prototype API For Parallel IDX Writes

This API is called Parallel IDX (PIDX) and includes functions patterned after ViSUS for creating, opening, reading, and writing IDX data sets. Each of the PIDX functions is a collective operation that accepts an MPI communicator as an argument.



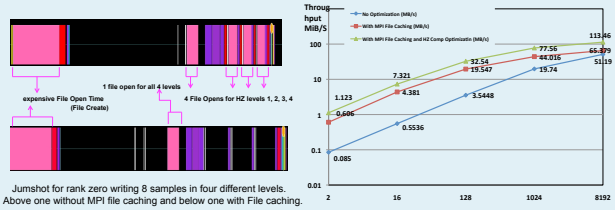
The data in this case is provided in the form of a contiguous, row-major ordered data buffer. Each process calculates an HZ ordering for its sub-volume, reorder the data points accordingly, and write the data points to interleaved portions of the IDX data set.

Optimization Strategies

The PIDX prototype described in the previous section greatly improved I/O performance over serial use of the ViSUS

MPI File Caching

Following figure highlights the time spent by rank 0 when writing data into the four initial HZ levels. The Pink regions represent File Open Time.



Jumshot for rank zero writing 8 samples in four different levels. Above one without MPI file caching and below one with File caching.

Performance of the API is improved by performing MPI file caching while writing data, which prevents unnecessary file opens corresponding to each HZ level.

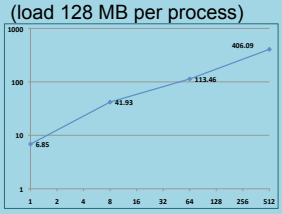
HZ computation Optimization

Indentified bottlenecks associated with redundant computations as well as inadequate use of the floating point double hummers.

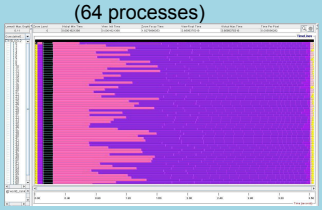
From the graph above: there is almost 220% increase in performance of the API after the two different optimizations.

Performance Testing

Weak scaling
(load 128 MB per process)



64x64x64 sample set
(64 processes)



•Single process achieves approximately 6.85 MiB/s, comparable to the speed of a 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.

•Pink corresponding to file opens and purple corresponds to file close, all the processes writes to the IDX file in parallel.

Future Work

Both with weak and strong scaling, a limit is hit on scalability with our current implementation, falling short of the peak surveyor write performance achieved by IOR.

Level	Time	Level	Time	Level	Time
0	**1.596**	11	0.129	22	0.304
1	**1.444**	12	0.151	23	0.363
2	**1.544**	13	0.151	24	1.979
3	**1.530**	14	0.148	25	2.140
4	**1.653**	15	0.102	26	2.706
5	**1.640**	16	0.085	27	4.605
6	**1.628**	17	0.077	28	6.760
7	0.148	18	0.086	29	10.533
8	0.132	19	0.087	30	18.251
9	0.134	20	0.140		
10	0.132	21	0.211		

Time Taken To Write The Various Idx Levels For A 8 Gib Data Volume On 64 Processes

We found that contention and metadata overhead caused levels 0 through 6 to take a disproportionate amount of time relative to the amount of data that they were writing. We 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.

Publication

S. Kumar, V. Vishwanath, P. Carns, V. Pascucci, R. Latham, T. Peterka, M. Papka, R. Ross. "Towards Parallel Access of Multi-dimensional, Multi-resolution Scientific Data," 5th Petascale Data Storage Workshop, Supercomputing 2010