# Efficient I/O and storage of adaptive resolution data

Sidharth Kumar,[*] John Edwards,[*] Peer-Timo Bremer,[*‡] Aaron Knoll,[*] Cameron Christensen,[*]
Venkatram Vishwanath,[†] Philip Carns,[†] John A. Schmidt,[*] Valerio Pascucci[*]

[*]Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA
[†]Argonne National Laboratory, Argonne, IL, USA
[‡]Lawrence Livermore National Laboratory, Livermore, CA, USA

*Abstract*—**We present an efficient, flexible, adaptive-resolution I/O framework that is suitable for both uniform and Adaptive Mesh Refinement (AMR) simulations. In an AMR setting, current solutions typically represent each resolution level as an independent grid which often results in inefficient storage and performance. Our technique coalesces domain data into a unified, multiresolution representation with fast, spatially aggregated I/O. Furthermore, our framework easily extends to importance-driven storage of uniform grids, for example, by storing regions of interest at full resolution and nonessential regions at lower resolution for visualization or analysis. Our framework, which is an extension of the PIDX framework, achieves state of the art disk usage and I/O performance regardless of resolution of the data, regions of interest, and the number of processes that generated the data. We demonstrate the scalability and efficiency of our framework using the Uintah and S3D large-scale combustion codes on the Mira and Edison supercomputers.**

## I. Introduction

As simulation sizes continue to grow rapidly, parallel I/O remains an ever increasing problem. There is currently a marked trend of simulations moving towards adaptive resolution techniques, e.g., Adaptive Mesh Refinement (AMR), to better manage multiple scales and couple detailed dynamics with large scale behaviors. Most current high-end I/O frameworks [1], [2] are optimized for uniform grids and, in fact, adaptive resolution grids are often simply represented and written as a collection of uniform grids at different resolutions. Such representations can result in fragmented and thus inefficient I/O. Furthermore, for convenience, many approaches unnecessarily replicate data on multiple levels, increasing the data footprint and decreasing I/O performance.

Uniform grid simulations are also growing in size, and writing intermediate data often takes up a nontrivial percentage of the total computation time. To reduce I/O time and disk usage, simulation runs frequently output the current solution only at certain iterations. A better approach is often to output a subset of the data at more frequent intervals. Ideally, the output data would be a region-of-interest (ROI), a reduced-resolution version of the grid, or a combination of the two. This technique would considerably reduce the disk usage and the I/O time of a simulation.

In this paper, we present extensions to the IDX file format [3], [4] that enable efficient storage of adaptive-resolution grids. The data is represented as a single adaptive grid, avoiding unnecessary replication, and providing both spatial and hierarchical locality. Our data format is agnostic to the type of simulation used to generate the data, and is thus general for both AMR-generated data and adaptive data derived from uniform grid simulation results. A single, unified format presents opportunities for re-use of I/O libraries regardless of simulation strategy. We also present extensions to the Parallel IDX (PIDX) I/O framework [5] that writes adaptive IDX files efficiently and supports AMR simulations. We discuss performance results of PIDX integrated into the Uintah block-structured AMR simulation environment [6], [7], [8]. We also show results using PIDX for data derived from S3D combustion simulation [9]. Regions of interest are extracted from the uniform grid data and written at higher resolution than the remaining regions. We also study the tradeoffs between performance and storage in both simulation environments and show that tuning between datasets and target machines can be done with a single parameter.

We have three specific contributions:

1) We extend the multiresolution IDX format to support adaptive resolution I/O. We also extend the Parallel IDX (PIDX) framework to support adaptive IDX in a parallel setting.
2) Using PIDX, we write IDX files for AMR simulations, which coalesces AMR levels into a single, space-efficient format that shows excellent spatial and hierarchical locality characteristics. We specifically demonstrate improved I/O performance over the commonly-used I/O format of Uintah.
3) We propose a novel, adaptive, region-of-interest (ROI) storage methodology for dumps of uniform simulation data. Using PIDX, we demonstrate this methodology to be more efficient than current techniques that store data in its entirety.

We discuss previous work in Section II. We then describe the IDX format for adaptive data in Section III followed by consideration of adaptive IDX for parallel applications in Section IV. In Section V, we describe our experiment platforms. We show experimental results of I/O throughput, disk usage, and visualization experiments for AMR in Section VI and for uniform simulations in Section VII.

## II. Related Work

AMR and uniform grid simulations generally have different I/O and storage requirements and thus methodologies tend to focus on one or the other. We discuss each in turn. We additionally directly compare our technique with the commonly used parallel HDF5 [2] I/O library.

## A. *AMR*

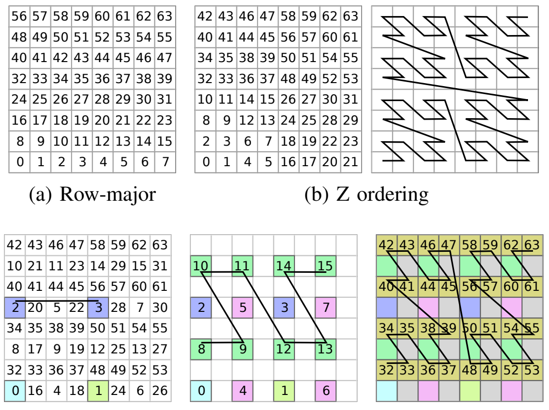
I/O in a block-structured AMR simulation environment is a challenge with various existing approaches. One approach is for each processor to write a single file, or $N$-$N$ output, where $N$ is the number of processors. This is commonly called file-per-process I/O. This approach is simple and efficient, but data at different hierarchy levels is duplicated, there is no mechanism for decoupling the data storage resolution from AMR resolution, and a large number of files are produced, causing a burden on downstream visualization and analysis packages. An $N$-1 approach is one where all processors write to a single file, such as the popular HDF5 format [2]. The Chombo [10] and FLASH [11] multiphysics applications use AMR for their simulation and these codes use HDF5 to write their AMR data to storage. HDF5 is a self-describing hierarchical representation with chunked storage and parallel I/O using MPI-I/O. Yu et al. [12] target cell-based AMR rather than block-based. Their algorithm bridges the gap between $N$-1 and $N$-$N$ by doing $N$-$M$ where $M$ is user-tunable number of files, and it achieves spatial locality by using space-filling curves.

## B. *Uniform simulations*

Solutions from simulations that are uniformly gridded are intuitively straightforward to store. Typical data storage techniques range from a single shared file ($N$-1), as used in PnetCDF [1] and HDF5 [2] to a file-per-process I/O ($N$-$N$). Single shared file approaches, such as PnetCDF, are optimized for dense, regular datasets, and are inefficient for ROI data in both performance and storage. File-per-process methods can be efficient in storage, but do not scale well on all parallel file systems. Subfiling [13] is a mechanism between these two extremes wherein the data is written out to a few files, $N$-$M$ wherein $M << N$, to overcome the locking and metadata overheads associated with the parallel filesystem. Another popular library used to manage parallel I/O for scientific applications is ADIOS [14]; it supports a variety of back-end formats and plug-ins that can be selected at run time. An important feature utilized by I/O libraries in general is aggregation, which is a stage in the write pipeline that passes data between nodes such that aggregator nodes can do more efficient block-based writes. PIDX [5], [15], [16], [17] is a parallel I/O API that stores data in the IDX format. PIDX also uses aggregators, and recently added a restructuring phase that increases efficiency on writes of data in grids that are not powers of $2^D$ in size.

## C. *Parallel HDF5*

PIDX differs from HDF5 in terms of the storage format and in the read and write techniques. Whereas IDX is a data format naturally suited to multiresolution AMR datasets as well as visualization and analysis, HDF5 is, in contrast, a container, making data layout specification the application developer's responsibility. A well-designed HDF5 layout can be suitable, but is not included in the HDF5 specification. Further, IDX does not need to store metadata associated with AMR levels or adaptive ROI; hierarchical and spatial layout characteristics are implicit, whereas HDF5 requires metadata to describe data layout and extents.



(a) Row-major (b) Z ordering

(c) HZ level 2 (d) HZ level 4 (e) HZ level 6

Fig. 1. Different index orderings. (a) Row-major ordering has poor spatial locality. (b) Z ordering shows good spatial locality but has no concept of hierarchy or resolution adaptivity. (c)-(e) HZ ordering has both spatial and hierarchical locality. For an example of hierarchical locality, note that obtaining a $1/2^2$ resolution version of the grid requires a single disk read of elements 0-15 (best seen in (d)).

The read and write techniques of PIDX also contribute to better performance. While HDF5 performs shared file I/O, PIDX breaks data into multiple files. The number of files to generate can be adjusted based on the file system. This approach extracts more performance out of parallel file systems, and is customizable to specific file systems. Further, PIDX utilizes a customized aggregation phase that spans all AMR levels, leveraging concurrency and leading to more optimized file access patterns.

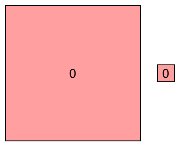
This paper describes extensions to PIDX that not only make it a viable option for parallel AMR simulation I/O, but also enable region-of-interest (ROI) stores in a uniform setting.

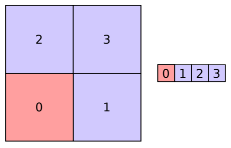
## III. Adaptive Resolution IDX

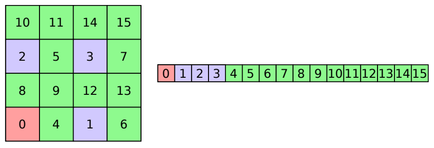
The IDX format [3], [4] was originally designed to support fast, multiresolution reads of uniform grids [18], [19]. In this section, we discuss the suitability of the IDX format to adaptive data and also an extension to the IDX format to support such datasets. The extension has little to no impact on read performance. Section IV will discuss the necessary extensions to the PIDX I/O framework to support fast writes of adaptive data.

## A. *Balance of spatial and hierarchical locality*

Structured data can be encoded into a single-dimensional array for disk storage in a variety of ways (see Fig. 1). A simple approach is to use row- or column-major ordering, but spatial locality is reasonable along only one dimension


(a) $RR_0$


(b) $RR_2$


(c) $RR_4$

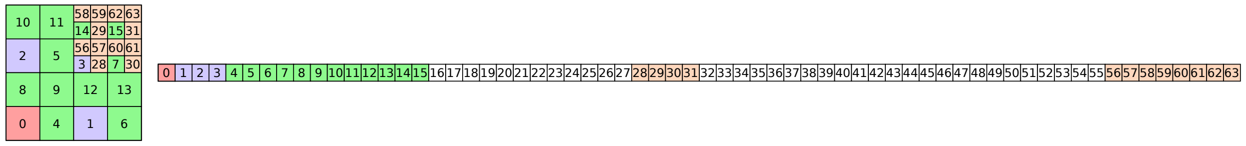
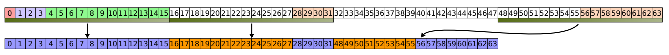
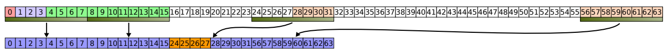

(d) $RR_6$

Fig. 2. HZ encoding of resolution regions ($RR$). Elements are colored by hierarchy level (HZ level in parentheses): pink (0), blue (1-2), green (3-4), orange (5-6). (a)-(c) Because of hierarchical locality, there is no wasted space when storing the entire domain at reduced resolution. (d) Storing an ROI smaller than the domain introduces fragmentation that will be handled when writing to disk. Elements in white need not be written to disk.


(a) Block size = 16; disk usage = 48; efficiency = 0.58


(b) Block size = 8; disk usage = 32; efficiency = 0.88

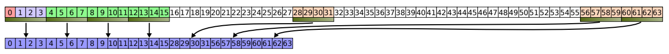

(c) Block size = 4; disk usage = 28; efficiency = 1.0

Fig. 3. Compaction of HZ-indexed elements. The single-dimensional array (above) is divided into write blocks and skip blocks. Only write blocks are stored on disk (below). Elements in the HZ-indexed array are shown colored by HZ level, and elements in the disk array are colored blue for primary and orange for secondary elements. Decreasing block size decreases disk space usage and increases efficiency.

(the x-axis in the figure). Z (also known as Morton) ordering [20] shows better spatial locality. The Z index is efficient to compute – simply interleave the bits of the Cartesian coordinates. Hierarchical Z (HZ) ordering extends Z ordering by introducing hierarchy. The HZ index is also efficient to compute using bit operations as described in [21].

The IDX storage format uses HZ ordering, which shows good locality both spatially and hierarchically, as shown in Fig. 2(a-c). Conceptually, the scheme acts as a storage pyramid with each level of resolution (called HZ level) laid out in Z-order. However, unlike traditional pyramids, IDX avoids replicating samples, which not only reduces the storage requirements but also organizes the data hierarchically. Fig. 2(a-c) shows an example of a $4 \times 4$ grid mapped to a linear index using IDX. Fig. 2d shows an adaptively refined version of this grid alongside the now partially occupied index space. Note that even for this adaptively sampled grid, a lower resolution version of the entire domain can be obtained by reading the contiguous 0-15 block.

To understand the impact of the partially occupied index space, we define a resolution region $RR_i$ at HZ level $i$ to be a (spatial) region in the domain stored at resolution $i$ (see Fig. 2). The region may be of any shape and need not be connected. Since IDX does not replicate samples, each element $e$ of an arbitrary resolution grid with HZ index $HZ(e)$ has a uniquely defined HZ level, $HZLevel(e)$, and can potentially be part of all $RR_i$s with $HZLevel(e) \leq i$. Due to the fragmentation of the index space and the disk blocking discussed in the next section, files may contain some "samples" not part of the original grid. In the discussion, we call these *secondary* samples, whereas samples that are part of the original grid are referred to as *primary* samples (see Figs. 3).

Consider again Fig. 2d: $RR_4$ covers the entire domain and $RR_6$ only the top-right corner. Even with the good spatial locality of the HZ order, $RR_6$ gets split into two blocks of memory with indices 28-31 and 56-63, respectively. The number of secondary samples can vary based on the block size. We define the (storage) efficiency of a particular scheme as:

$$E = \frac{P}{P+S} \tag{1}$$

with $P$ the number of primary and $S$ the number of secondary elements. Note that any lower resolution write will always have perfect efficiency as long as it covers the entire domain. Furthermore, if there exists a $RR_i$ that does not cover the domain, adding lower resolution data actually improves the efficiency.

## B. IDX blocks

Similar to most other large scale file formats, IDX does not store the entire data in a single chunk but instead breaks the array into blocks that are written to (and read from) disk. In the case of adaptive IDX files, each block that contains at least one primary sample is written and all other blocks are skipped. Fig. 3 shows the blocking and the resulting data on disk for different block sizes for the example of Fig. 2d. Note that, as the block size decreases, the efficiency increases.

The IDX format supports missing blocks. Many downstream tools such as ViSUS [22], [23] and VisIt [24] are block-aware and thus can simply ignore missing blocks. Dealing with secondary samples is slightly more involved. One option is to upsample secondary elements from primary elements at lower resolutions. This option is attractive because it does not require a change to the IDX format. However, it may

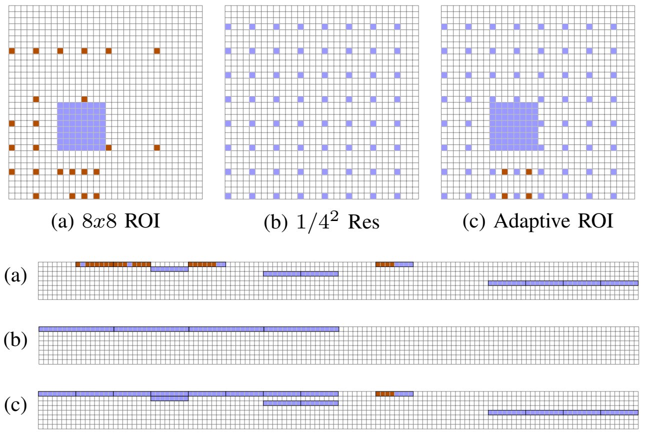


Fig. 4. ROI and reduced-resolution layouts. Blue elements are primary elements; orange elements are secondary elements, or elements that are written as a by-product. The top row shows spatial grids with different write patterns. The lower grids show the element layout on disk. Only primary and secondary elements are actually written to disk – blank elements are not written. (a) ROI write. Efficiency is 0.73. (b) Reduced-resolution write. Data on disk is not fragmented for reduced resolution writes, so efficiency is 1. (c) Adaptive ROI, combining ROI and reduced-resolution. Secondary elements for the ROI overlap with primary elements of the reduced-resolution write, so data is less fragmented, for efficiency of 0.97.

require expensive interprocess communication to obtain lower-resolution primary element values. Furthermore, the visualization or analysis tool reading the data has no way of knowing if a value was interpolated or if it was generated by the simulation. The preferred solution is to store a bitmask for each written block that indicates which elements are valid. This uses a small amount of additional storage and requires a change to the IDX format, but it is general and suitable for all scenarios. Addition of the bitmask is the only extension we make to the IDX format in order to support adaptive data.

## IV. Parallel IDX

The PIDX framework has previously been applied only to uniform data [5]. We describe the fundamental PIDX principles of HZ encoding and aggregation, and then discuss extensions that must be made for adaptive resolution datasets (AMR and ROI I/O).

### A. *HZ encoding and aggregation*

HZ encoding is the process of copying elements into single-dimensional arrays in HZ order. Each node computes the min and max HZ indices of its elements, allocates an array of size $\max - \min + 1$, and copies each element into the HZ array. Depending on resolution and spatial bounds of the node, the HZ-encoded array may have dummy values for elements not contained in the node's spatial domain (see Fig. 2d). Because this fragmentation can cause I/O inefficiencies due to multiple writes, we adopt the aggregation strategy of Kumar et al. [5], which we briefly describe here.

The first step is to set up the aggregation buffers. A set of nodes, or aggregators, allocate buffers such that each buffer stores a unique, HZ-contiguous section of elements. Each processing node sends elements to the appropriate aggregators. Using aggregation has been shown to reduce I/O times significantly [16], since each aggregator can perform a single, contiguous disk write.

### B. *AMR simulations*

We present two I/O algorithms for use in AMR simulations that write data in IDX format. Given an AMR level $l$, define $\mathit{HZLevel}(l)$ to be the highest HZ level of an element in $l$.

**Algorithm Store1**

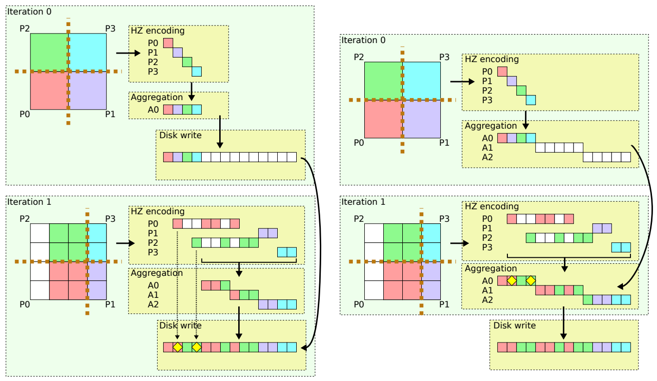
```
1: $hz_{prev} \leftarrow 0$
2: **for** Each AMR level $l \in \{0, \ldots, n\}$ **do**
3:     Determine write blocks
4:     HZ encoding
5:     Initialize aggregation buffers
6:     $hz_l \leftarrow \mathit{HZLevel}(l)$
7:     **for** Each HZ level $hz \in \{hz_{prev}+1, \ldots, hz_l\}$ **do**
8:         Aggregation
9:     **end for**
10:    Write aggregated data to disk
11:    **for** Each HZ level $hz \in \{0, \ldots, hz_{prev}\}$ **do**
12:        Write primary elements in $hz$ to disk
13:    **end for**
14:    $hz_{prev} \leftarrow hz_l$
15: **end for**
```

The Store1 algorithm (Fig. 5a) writes each AMR level $l$ in turn. After HZ encoding, all elements $e$ for which $\mathit{HZLevel}(e) > hz_{prev}$ are aggregated and written. Writing secondary elements is safe because HZ ordering guarantees that, for elements $a$ and $b$, HZ$(a) <$ HZ$(b)$ if $\mathit{HZLevel}(a) < \mathit{HZLevel}(b)$. We bypass aggregation and write directly to disk all primary elements $e$ for which $\mathit{HZLevel}(e) \leq hz_{prev}$ to avoid overwriting existing primary elements with secondary elements.

**Algorithm Store2**

```
1: **for** Each AMR level $l \in \{0, \ldots, n\}$ **do**
2:     Determine write blocks
3: **end for**
4: Initialize aggregation buffers
5: **for** Each AMR level $l \in \{0, \ldots, n\}$ **do**
6:     HZ encoding
7:     Aggregation
8: **end for**
9: Disk write
```

The Store2 algorithm (Fig. 5b) sets up aggregation buffers for all AMR levels and then aggregates the entire dataset before performing the disk write. Store2 requires more memory, as the aggregation buffers store elements of all AMR levels at once, but gains performance advantages because it avoids the multiple fragmented disk writes of primary elements at HZ levels $hz \leq hz_{prev}$, and instead makes fragmented data



(a) Algorithm Store1 (b) Algorithm Store2

Fig. 5. Store algorithms. Elements are colored by their processing node, which is denoted P*i*, or are colored white if they need not be written to disk. We label aggregation nodes as A*j*. Elements that are overwritten either on disk (Store1) or in memory (Store2) are marked with a yellow diamond.

transfers from processing nodes to aggregation nodes. Store2 is the default algorithm used by PIDX.

We compared the two algorithms with an experiment on Edison (see Section V) with a coarse AMR level with $25^3$ elements per core and a refined level at a refinement ratio of 2. Store2 performed I/O roughly an order of magnitude faster in all cases. With 32/64/128 cores, Store1 took 4/8/12 seconds, and Store2 took 0.6/1/2 seconds.

In both the Store1 and Store2 algorithms, the order of iteration through AMR levels is important. Levels must be processed from low resolution to high resolution so that more refined data overwrites coarser approximations rather than vice versa.

### C. ROI I/O and Uniform simulations

Writing reduced-resolution or ROI data in a uniform simulation setting is more straighforward than AMR simulations. Reduced-resolution stores simply restrict HZ levels to a specified maximum and then use the PIDX algorithm published previously [5]. ROI stores also use the PIDX algorithm, but restrict HZ indices when setting up the aggregator buffers. Details of the PIDX algorithm are described by Kumar et al [5].

One step common to both AMR I/O and ROI I/O is identification of IDX blocks that need to be written to disk and accordingly perform aggregation and disk I/O. For uniform resolution datasets, there are no secondary elements, hence, all blocks gets written to the disk. As a result, the write-load by default gets equally balanced amongst the aggregators. This is not true with adaptive resolution data (AMR and ROI), where blocks have to be skipped from being written to the disk (see Figs. 3 and 4). Hence, for adaptive resolution data, before performing aggregation and actual disk writes, it is essential to identify the blocks that need to be written (first step in both Store1 and Store2) and then uniformly send the data corresponding to these write-blocks to the aggregators. To this end we label the blocks as a write or a skip block: We first allocate a buffer to act as a bitmask for which blocks to write. The size of the buffer is determined by the highest HZ level. For a given processor $P$, let $h$ be the highest HZ level represented by an element. $P$ then iterates through all blocks in all HZ levels $\leq h$. If a block $B$ intersects the spatial region covered by $P$, then $B$ is marked as a write block. With the write-blocks identified, the information is used to set up the aggregation buffers for the aggregation step. With this step we are able to avoid any unnecessary I/O (for skip-blocks), and achieve load balance.

## V. Results overview and experiment platform

We report results of experiments in both AMR and uniform grid settings. The AMR experiment (Section VI) demonstrates weak scaling performance of PIDX within the Uintah simulation framework. Uniform grid experiments (Section VII) include a study of time and storage efficiencies of various layouts of regions of interest; data and weak scaling results on reduced-resolution writes; and time and disk usage results from combined ROI and reduced resolution writes of combustion simulation output using S3D.

The experiments presented in this work were performed on Edison at the National Energy Research Scientific Computing (NERSC) Center and Mira at the Argonne Leadership Computing Facility (ALCF). Edison is a Cray XC30 with a peak performance of 2.39 petaflops, $124,608$ compute cores, 332 TiB of RAM, and 7.5 PiB of online disk storage. We used Edison Lustre file system (168 GiB/s, 24 I/O servers and 4 Object Storage Targets). Default striping was used with the Lustre file system. Mira system contains 48 racks and 768K cores, and has a theoretical peak performance of 10 petaflops. Each node has 16 cores, with 16 GB of RAM per node. I/O and interprocessor communication travels on a 5D torus network. Every 128 compute nodes has two 2 GB/s bandwidth links to two different I/O nodes, making 4 GB/s bandwidth for I/O at most. I/O nodes are connected to file servers through QDR IB. Mira uses a GPFS file system with 24 PB of capacity and 240 GB/s bandwidth.

## VI. AMR simulations

Our AMR experiments are done within the Uintah simulation environment. While Uintah is a block-structure code [25], the IDX format is suitable for any hierarchical structured data, including octree and overlapping grids.

### A. Uintah simulation and I/O framework

The Uintah framework uses a structured adaptive mesh refinement (SAMR) grid to execute the solution of partial differential equations (PDEs). The component developer typically uses either a finite difference or finite volume algorithm to discretize the PDEs on the grid. The grid can be thought of as a container of AMR levels. Each level is described by a collection of patches that are distributed to individual processors/cores via a load balancing algorithm in the runtime system. See Fig. 6. Each patch can be thought of as the

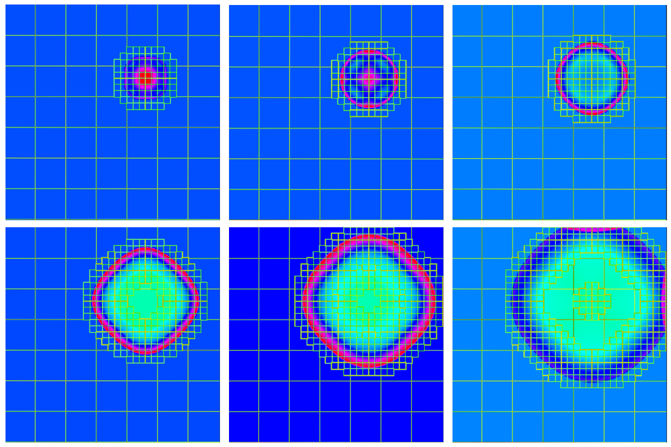


Fig. 6. Progression of an AMR Uintah simulation of a 2-level problem in time. Data was written to disk using PIDX and visualized with ViSUS. The experiment was run on Edison with a coarse grid domain of size 200×200×200. The figure demonstrates regridding of the finer AMR level, leading to an increasing number of patches. Each rectangle corresponds to a single patch. Note that the last timestep corresponds to two disconnected regions: center and ring of the fine AMR level.

fundamental unit of work that contains a region of cells at a given resolution.

In the existing I/O framework of Uintah, every process writes data belonging to a patch into a separate file. This form of I/O is an extension to the file-per process style of I/O commonly adopted by many simulations. Each MPI rank collects all of the patches that it is assigned. The simulation variable data for each patch is written out into a separate file along with its associated metadata file. The metadata file stores type, extents, and bounds of all the variables. For AMR with multiple levels, a directory structure is created. For relatively small numbers of patches ($< 2K$) and core counts, the I/O framework works well. However, I/O performance degrades significantly for typical simulations with several hundreds of thousands of patches/processors. The overwhelmingly large number of small files causes both writes and reads to become extremely expensive.

Including metadata files, the total number of files created every timestep is twice the total number of patches across all AMR levels. The number of patches usually exceeds the product of cores and AMR levels, leading to an allocation of multiple patches per AMR level for every process. Creation of such a large number of files leads to a big overhead in metadata management. The problem intensifies as a simulation progresses, leading to generation of newer patches at finer AMR resolutions (see Fig. 6). Note that it is structurally not possible to coalesce patches into a bigger buffer to obtain better disk access patterns and fewer files. This is because patches within an AMR level are not guaranteed to be spatially close to each other, and further, may form irregular shapes (see Fig. 6).

We have integrated PIDX I/O library with Uintah, enabling the simulation code to write AMR data directly in the IDX format. IDX/PIDX fills the need for both an efficient file format and a parallel I/O library.

### B. *Performance results in Uintah*

In this section, we evaluate the weak scaling performance of PIDX when writing AMR data with two levels of refinement from Uintah simulations on both Mira and Edison. The simulation used a refinement ratio of 2, i.e., each voxel in the coarse level was refined into 8 voxels at the finer refinement level. In each run, Uintah wrote out 15 timesteps consisting of 4 variables: pressure, temperature, density and mixture fraction. The simulation dumps data at every 10th time-step. The patch size for the coarse refinement level is $25^3$, and the patch size for the finer refinement level is $12^3$. All the processes uniformly span the coarse-level grid, hence writing one coarse-level patch of size $25^3$. For this particular simulation, the number of fine-level patches at the start of the run is approximately equal to the number of cores, but as the simulation progresses the number gradually increases with regridding of the fine level.

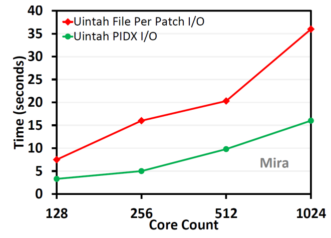
We performed experiments on Edison and Mira. On Edison, we varied the number of processes from 1024 to 8192 and used a block size of $2^{14}$. On Mira, processes ranged from 128 to 1024 with a block size of $2^{17}$. The weak scaling results comparing Uintah with PIDX I/O and Uintah with the traditional file-per-patch on Mira and Edison can be seen in Figs. 7a and 7b, respectively.

On Mira, it can be seen in Fig. 7a that at all core counts, Uintah with PIDX I/O performs better than the traditional file-per-patch mechanism. At 1024 cores, the file-per-patch scheme takes approximately 35 seconds compared to 15 seconds by PIDX I/O. Broadly, there are two reasons for this performance behavior. 1) Metadata congestion in creating the hierarchy of files for the file-per-patch method leads to degraded non-scalable performance. 2) Fewer file creations along with the custom Store2 algorithm used in PIDX lead to a better disk access pattern and improved scalibility.

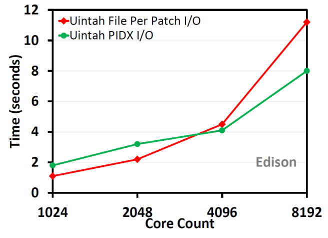
It can be seen in Fig. 7b that PIDX I/O trails in performance on Edison at core counts of 1024 and 2048. This is primarily because the Lustre filesystem of Edison is more adept at handling large numbers of files compared to the GPFS file system of Mira. Hence, for lower-core counts the number of files generated by file-per-patch approach is within reasonable limits, but as the number of cores increases, the number of files generated reaches a limit that starts to saturate the metadata server. Hence, there is an observed degradaton in performance at 4096 core counts. PIDX, on the other hand, generates roughly two orders of magnitude fewer files than file-per-patch I/O schemes (see Table 7c) and this ratio remains steady even as the number of cores increases. The improved performance with PIDX can again be attributed to the custom aggregation phase that assures favorable disk access patterns.

## VII. Uniform simulation and adaptive ROI I/O

With uniform grid simulations increasing in size and complexity, it is difficult for the I/O and storage systems to keep pace with the increasing amount of data that scientists need



(a) Mira

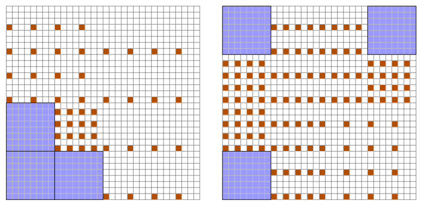


(b) Edison

| Total processes | Avg patch count | File count | |
|---|---|---|---|
| | L0 + L1 | PIDX | File-per-patch |
| 1024 | 1024 + 1412 | 24 | 4872 |
| 2048 | 2048 + 3201 | 44 | 10498 |
| 4096 | 4096 + 5835 | 88 | 19682 |
| 8192 | 8192 + 10857 | 156 | 38098 |

(c)

Fig. 7. (a) Mira results for weak scaling of Uintah with PIDX I/O, and traditional file-per-patch I/O using a two-level AMR simulation. We do not report numbers to 8192 cores on Mira because the default Uintah I/O scheme fails at higher core counts (Uintah's file-per-patch approach overwhelms the I/O nodes). This problem is currently being addressed by Uintah developers. (b) Edison results for weak scaling of Uintah with PIDX I/O, and traditional file-per-patch I/O using a two-level AMR simulation (c) Table showing the number of patches generated for Edison runs, and corresponding number of files generated with PIDX I/O and file-per-patch I/O approach. With the latter approach, there is a metadata file for every patch taking the file count tally to twice the total number of patches. With PIDX, there are fewer number of files controlled by parameters block size and blocks per file, which are set as 32768 and 128, respectively.



(a) Clustered; E = 0.80. (b) Scattered; E = 0.63.

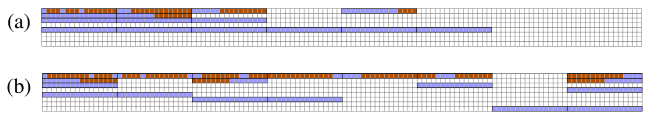


Fig. 8. Effect of scattering of processes on efficiency. If regions of interest are located apart from each other, then efficiency goes down as potential secondary elements from one ROI are less likely to correspond to primary elements of another ROI. 32 × 32 grid; 8 × 8 regions; block size = 16. Efficiency is denoted as E.

to store. Here we present an I/O strategy incorporating ROI and reduced resolution writes that we call Adaptive ROI I/O, which is an attractive alternative to traditional raw file dumps, especially for analysis and visualization. Many simulations are heavily padded to avoid boundary artifacts and often the phenomena of interest, e.g., ignition, extinction, etc., are confined to a comparatively small part of the domain. Therefore, writing distinct analysis or visualization snapshots that either only store the regions of interest or grade the saved resolution according to some importance measure may significantly reduce the overall datasize without impacting the results. Here we use simple range thresholding to identify ROIs but more sophisticated techniques such as merge or contour trees [26], entropy-driven classification [27], or wavelets [28] could be adapted as well. We focus our discussion in this section on uniform resolution grids but the same principles can be applied to more aggressively subset AMR meshes as well.

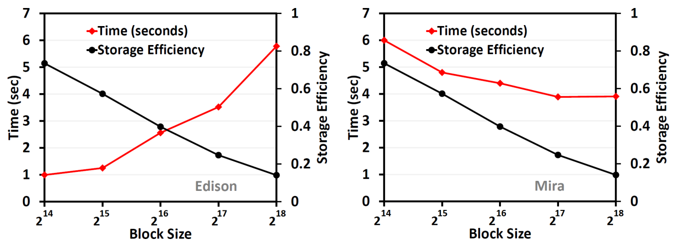
The primary technical challenge of Adaptive ROI I/O is to attain a high throughput even though the data is distributed unevenly and potentially consists of many small isolated regions, which can lead to fragmented accesses to both memory and disk. Here we demonstrate how using different block sizes and the on-the-fly aggregation capabilities of PIDX reduces these problems and leads to high throughput I/O. We report results from experiments done on both Mira and Edison.

For analysis purposes, we break down Adaptive ROI I/O into two orthogonal capabilities: first, full resolution ROI I/O, where a subset of the domain is written at the native resolution; and second, reduced-resolution I/O, where the entire domain is written at reduced resolution. Both stress different parts of the storage layout as shown in Figs. 4a and 4b. While the former relies on using blocks and Z-order locality to attain storage efficiency, the latter exploits the hierarchial nature of the file format. Both components are equally influenced by the aggregation phase. Adaptive ROI storage, i.e., storing varying resolutions according to some importance measure, can be seen as a combination of both components. Nevertheless, even separately there exist valid use cases such as writing data only around features of interest and storing the entire domain at low resolution for exploratory visualization.

## A. *Region-of-interest I/O*

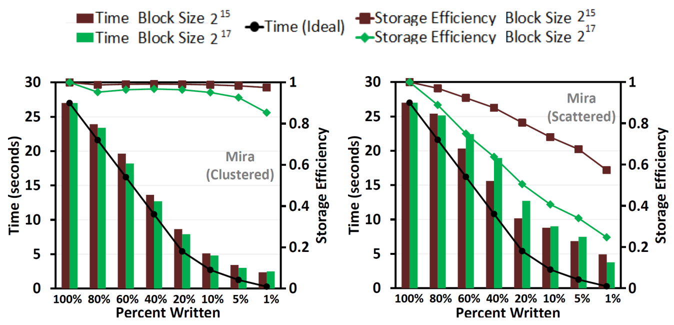
This section provides empirical analysis of full resolution ROI writes, with the goal of understanding the tradeoffs between storage efficiency and performance. For most of the experiments conducted in this section, the amount of data written is varied between 1%, 5%, 10%, 20%, 40%, 60%, and 80% of the entire data volume. This is achieved by making the given percentage of processes generate and write data, whereas others remain idle. The IDX block size is varied among $2^{14}$, $2^{15}$, $2^{16}$, $2^{17}$, and $2^{18}$. Recall that smaller block size leads to better storage efficiency (see Fig. 3). Furthermore, we identify two scenarios for the distribution of processes in the global domain: *clustered*, where processes generating data are spatially close to each other (see Fig. 8a), and *scattered*, where processes generating data are spatially far from each other (see Fig. 8b).

For a given percentage of data to write, the clustered layout improves the storage efficiency. This is due to the existence of fewer number of secondary elements across all processes, because most of the secondary elements of a process are primary elements of an adjacent process (see Fig. 8a). Recall from section III-B that secondary elements are an artifact of usage of blocks. Similarly, the scattered case leads to poor storage efficiency, due to the presence of large numbers of secondary elements across all processes (see Fig. 8b). Storage efficiency improves with reduced block sizes for both layouts.



(a) Clustered (b) Scattered

Fig. 9. Uniform simulation / ROI-based output. (a) Performance evaluation (time) and storage efficiency on Edison with varying block sizes for writing 1% of scattered layout data using 4096 cores. (b) Performance evaluation (time) and storage efficiency on Mira with varying block sizes for writing 1% of scattered layout data using 4096 cores.



(a) Clustered (b) Scattered

Fig. 10. Uniform simulation / ROI-based output. (a) Performance and storage measured with varying percent write for block size $2^{15}$ and $2^{17}$ using the clustered layout. (b) Performance and storage measured with varying percent write for block size $2^{15}$ and $2^{17}$ using the scattered layout.

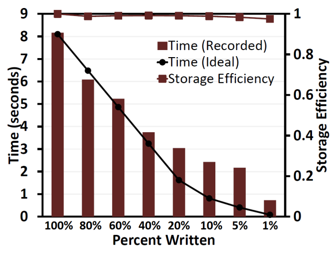
*1) Storage and performance tradeoffs:* In the first set of experiments we measure the tradeoffs between storage efficiency and performance for writing ROI data as a result of variation in block size. For this purpose, we use the scattered layout and fix the amount of data written at 1%. This configuration in particular is interesting. It can be treated as the worst case ROI I/O scenario: a small percent of data is being written, and storage efficiency is low because the writing processes are spatially scattered. Block size is exponentially varied from $2^{14}$ to $2^{18}$.

All experiments are conducted at fixed core count of 4096 using the S3D I/O simulator. The S3D I/O extracts just the portion of S3D combustion simulation code concerned with restart dumps, allowing us to focus exclusively on I/O characteristics. For our evaluation, we used an S3D I/O configuration wherein each of the 40 processes (1% of 4096) produced a $64^3$ sub-volume of double precision floating point data. This configuration produced 32MB of data within each of the 40 processes. The number of blocks per file was fixed to 256 for all runs, implying fewer files with a large block size and vice versa. Results for both Mira and Edison can be seen in Fig. 9. Performance measured as time taken to perform the I/O operation (red trendline) is shown on the primary Y-axis (left), and the corresponding storage efficiency (black trendline) is plotted on the secondary Y-axis. A key insight is that storage efficiency improves with a smaller block count no matter what machine is used. Hence, the black trendline for storage efficiency shows an inclining trend with decreasing block size for both Mira and Edison. Interestingly the performance (time to write) improves with storage efficiency on Edison, as opposed to a decline on Mira. This is largely due to the metadata contention associated with writing large numbers of files on Mira. For example, a block size of $2^{14}$ writes 256 files as opposed to writing only 16 files for $2^{18}$ block size.

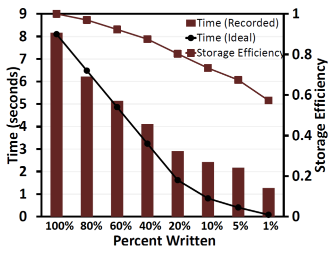
*2) Storage efficiency vs. performance with varied percent writes:* In this section we evaluate the efficacy of PIDX when performing full resolution ROI writes with varying region sizes. We again used the S3D I/O simulator to carry out all the experiments. The number of core counts was fixed to 4096, but we varied the percent of processes performing I/O to be 1%, 5%, 10%, 20%, 40%, 80%, and 100%. For each of these cases, the processes generating data each contributed a $64^3$ block of double-precision data (32MB). On Mira, block sizes of $2^{15}$ and $2^{17}$ were used that correspond to relatively lower and higher storage efficiency, respectively. For both Mira and Edison, we used the scattered and clustered layout for each percentage of data written. The results for Edison and Mira can be seen in Figs. 10 and 11, respectively. For Mira results, the brown and green histograms correspond to I/O time for blocks of size $2^{15}$ and $2^{17}$, respectively, and are shown on the primary Y-axis (left). Similarly, the green and brown trendlines correspond to the storage efficiency for the blocks sizes $2^{17}$ and $2^{15}$ shown on the secondary Y-axis. The black trendline corresponds to ideal time showing the ideal, linear decrease in time with decreasing write percentage.

For Mira with decreasing write volumes, the percent reduction in time for the clustered layout is better compared to the scattered layout. This stems from the fact that the clustered layout has a better storage efficiency. Another observation is that for the clustered layout, performance for writes up to 40% of the entire volume is comparable to the ideal time. Performance starts to decrease for smaller write percents, mainly due to lack of workload. Comparing the performance pattern across block sizes with similar storage efficiencies (i.e., the workload is similar for both layouts), it can be seen that with the clustered layout, larger block size ($2^{17}$) results in slightly better performance compared to block size $2^{15}$. This again can be attributed to less metadata contention associated with creating fewer files with block size $2^{17}$. The scattered layout, on the other hand, presents slightly different behavior for the two block sizes. Due to a large difference in the storage efficiency of the two block sizes, the I/O performance is dominated by the total load involved. Hence, for percent writes of 60%, 40%, and 20%, the smaller block size with better storage efficiency and relatively lesser work load shows better performance. The trend again starts to reverse around 10% writes, when the overall work load becomes small (smaller write %), resulting in better performance for block size of $2^{15}$ compared to $2^{17}$.

Edison results can be seen in Fig. 11. Block size of $2^{15}$ was used for all the experiments. Similar to Mira, greater per-



(a) Clustered



(b) Scattered

Fig. 11. Uniform simulation / ROI-based output. (a) Edison results for performance and storage efficiency measured with varying percent write for block size $2^{15}$ and $2^{17}$ using the clustered layout. (b) Edison results for performance and storage measured with varying percent write for block size $2^{15}$ and $2^{17}$ using the scattered layout.

formance degradation is observed for smaller percent writes. Also, the rate of degradation in performance for the scattered layout is observed to be slightly more than the clustered layout.

Overall, our ROI experiments demonstrate that excellent storage and performance results can be obtained by adjusting a single parameter (block size) to tune for the specific target system, ROI size, and ROI layout.

## B. Reduced resolution I/O

Storing simulation results at reduced resolution can be used for fast exploratory visualization. In addition, this capability can also allow simulations to dump more frequent visualization checkpoints. The majority of simulations perform I/O dumps at intervals of several hundred timesteps, partially due to the expense incurred in parallel I/O. With PIDX, we can effectively have several reduced resolution dumps inserted between any two full resolution dumps. The more frequent dumps can lead to better tracking and monitoring of the simulation. Fig. 15 shows the lifted ethylene jet dataset stored at full (top) and 1/64 (bottom) resolutions. The lower resolution image is more efficient to store and requires less disk space, and still can give a general idea of how the simulation is progressing.

The layout of data in IDX format makes it inherently efficient for performing parallel I/O at reduced resolution. Data is laid out in increasing resolution, so access up to a given resolution level does not encounter any secondary elements, which leads to contiguous access of data (see Figs. 2 and 4b). Absence of any secondary element also ensures storage efficiency is ideal. In the following we evaluate the performance of reduced resolution writes.

*1) Data scaling:* In this section, we evaluate the efficacy of PIDX writes at varying resolutions. Similar to ROI, experiments in this section were also carried out using the S3D I/O simulator. Twenty timesteps were written for each run. We fixed the number of cores at 4096, while each process contributed a $64^3$ block of double-precision data (32MB at full resolution). Resolution level was exponentially decreased from 1 to $1/64$. At full resolution, each process dumps $64^3$ elements; at $1/64$ resolution each process dumps $16^3$ elements. The experimental results for Mira and Edison are shown in

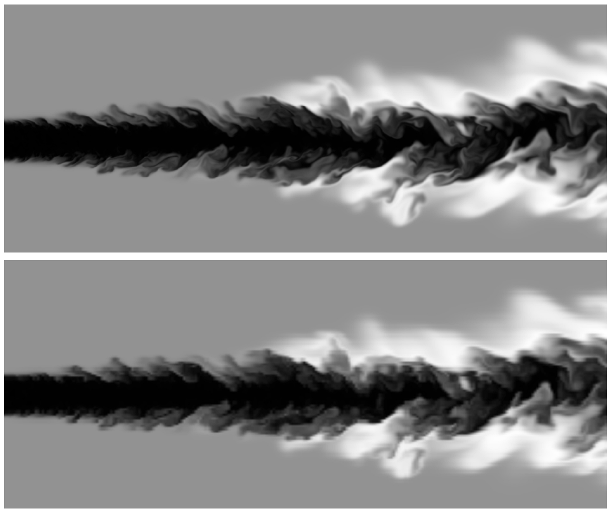
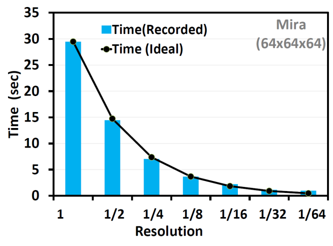
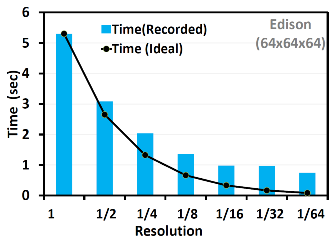


Fig. 12. Slice rendering of the temperature field of the lifted ethylene jet. (Top) Full resolution data; (Bottom) Data at $1/64$ resolution level.



(a) Mira

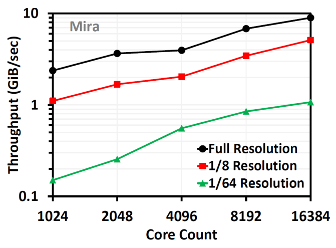


(b) Edison

Fig. 13. Reduced resolution I/O. The histogram corresponds to the time recorded to write the dataset at varying resolution levels. The trendline corresponding to ideal time shows the ideal, linear decrease in time with decreasing resolution, and is calculated by dividing the recorded time for full resolution writes by two for every resolution level. (a) With 4096 cores on Mira, the plot shows reduced resolution write timings where each core has $64^3$ elements. (b) With 4096 cores on Edison, the plot shows reduced resolution write timings where each core has $64^3$ elements.

Fig. 13. Except for the last two refinements of $1/32$ an $1/64$, it can be seen that for Mira, write time is cut almost in half with each resolution level, demonstrating near perfect efficiency. On the other hand, with Edison, while write time does reduce with resolution, the improvement is not as high as that of Mira. This difference in behavior of the two machines can largely be attributed to the presence of dedicated I/O nodes on Mira, as opposed to shared I/O channel on Edison.

*2) Weak scaling:* In this section, we evaluate the weak scaling performance of PIDX when writing S3D datasets at reduced resolution on both Mira and Edison. In each run, S3D writes out 20 timesteps. With Edison, each process contributes a $64^3$ block of double-precision data and for Mira, a process contributes a $32^3$ block of double-precision data. On Edison, we varied the number of processes from 1024 to 32768, thus varying the amount of data generated per timestep from 32GB to 1 TB. On Mira, we varied the number of processes from 1024 to 16384, thus varying the amount of data generated per timestep from 4 GB to 128 GB. Weak scaling runs are



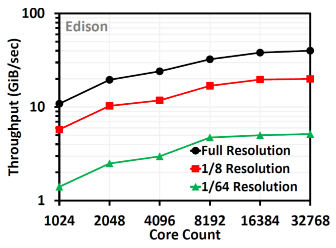


Fig. 14. Reduced resolution I/O. (a) Weak scaling on Mira. The number of cores increases from 1024 to 32768, while keeping the per process load fixed at $64^3$, and varying the resolution as 1, $1/8$ and $1/64$. (b) Weak scaling on Edison. The number of cores increases from 1024 to 16384, while keeping the per process load fixed at $32^3$, and varying the resolution as 1, $1/8$, and $1/64$.

conducted for three resolution levels: full resolution, $1/8$ and $1/64$ . The results for Mira and Edison can be seen in Fig. 14. Two key trends can be observed in these results. First, parallel I/O performance scales with all resolution writes, and second, decline in throughput for $1/8$ resolution writes is much more prominent than the decline in throughput of $1/64$. This behavior can be attributed to the lack of enough load to leverage the benefits of the optimizations of the PIDX API.

Our weak scaling experiments show that dumps at reduced resolution are scalable, and our data scaling experiments indicate that runtime performance gains can be achieved for varying loads. When combined with the fact that disk usage has ideal efficiency, reduced resolution writes become a compelling option for simulation monitoring and other applications where a full data dump is not necessary.

## *C. Adaptive ROI I/O*

The primary usefulness of ROI and reduced-resolution I/O is when they are combined into Adaptive ROI I/O. To demonstrate Adaptive ROI, we use the lifted ethylene jet, one of the largest combustion simulations performed by S3D (see Fig. 15). In particular, we use the temperature field as an initial test case. We use two different thresholds to define regions of high, medium, and low temperature and save these at full, $1/64$, and $1/512$ resolution, respectively. This results in a very complex arrangement well-suited to stress the system. The two flame sheets most easily distinguished on the left side of Fig. 15 burn very hot and thus get preserved at full resolution. The outside coflow on the top and bottom is heated by the central flame and thus resides in the medium temperature region. Finally, the channel in between the sheets contains the relatively cool fuel stream which gets classified as low temperature. Together, this configuration creates many sharp resolution drops and isolated regions as well as a significantly uneven data distribution. The resulting adaptive resolution IDX output takes only $39\%$ of the full resolution output time, while writing $30\%$ of the $12.8$ GB of full resolution data. As shown in the middle of Fig. 15, the resulting volume rendering (using up-sampling to create a uniform resolution grid) preserves the ROI almost perfectly while showing the expected artifacts

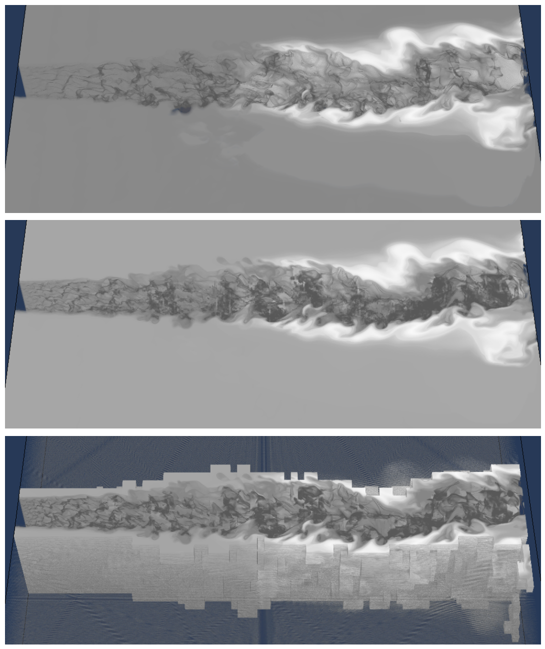


Fig. 15. Volume rendering of the temperature field of the lifted ethylene jet. (top) Full resolution data; (middle) Adaptively sampled data, up-sampled to create a uniform image; and (bottom) Adaptively sampled data without up-sampling to highlight the data distribution.

especially in the center of the flame. The bottom of Fig. 15 shows the same adaptive data without up-sampling to highlight the block distribution.

## VIII. Conclusions

Motivated by the growing need for data formats and I/O frameworks that support massive, adaptive datasets in a parallel setting, we have demonstrated IDX to be a viable format and PIDX to be an effective framework for data I/O in both AMR and uniform grid simulation environments. Our experiments show that IDX is storage-efficient, amenable to I/O optimizations, and scalable. Because of its previously-demonstrated spatial and hierarchical locality, reads are resolution-progressive and cache-efficient, and it is thus an excellent choice for visualization, analysis, and monitoring applications.

We have presented algorithms that build on existing aggregation methodologies for efficient writes, and shown that our extensions to the PIDX framework are viable for adaptive data. Further, we have shown PIDX to be an effective tool in adaptive ROI dumps of uniform data.

**Acknowledgments** This work is supported in part by BNSF CISE ACI-0904631, NSG IIS-1045032, NSF EFT ACI-0906379,DOE/NEUP 120341, DOE/Codesign P01180734, DOE/SciDAC DESC0007446, and CCMSC DE-NA0002375. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

## References

[1] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A high-performance scientific I/O interface," in *Proceedings of SC2003: High Performance Networking and Computing*. Phoenix, AZ: IEEE Computer Society Press, November 2003.

[2] "HDF5 home page," http://www.hdfgroup.org/HDF5/.

[3] V. Pascucci and R. J. Frank, "Global static indexing for real-time exploration of very large regular grids," in *Conference on High Performance Networking and Computing, archive proceedings of the ACM/IEEE Conference on Supercomputing*, 2001.

[4] V. Pascucci, D. E. Laney, R. J. Frank, F. Gygi, G. Scorzelli, L. Linsen, and B. Hamann, "Real-time monitoring of large scientific simulations," in *ACM Symposium on Applied Computing*, 2003, pp. 194–198.

[5] S. Kumar, V. Vishwanath, P. Carns, J. Levine, R. Latham, G. Scorzelli, H. Kolla, R. Grout, R. Ross, M. Papka, J. Chen, and V. Pascucci, "Efficient data restructuring and aggregation for I/O acceleration in PIDX," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[6] J. D. de St. Germain, J. McCorquodale, S. G. Parker, and C. R. Johnson, "Uintah: A massively parallel problem solving environment," in *Ninth IEEE International Symposium on High Performance and Distributed Computing*. IEEE, Piscataway, NJ, November 2000, pp. 33–41.

[7] S. G. Parker, J. Guilkey, and T. Harman, "A component-based parallel infrastructure for the simulation of fluid-structure interaction," *Engineering with Computers*, vol. 22, pp. 277–292, 2006.

[8] S. G. Parker, "A component-based architecture for parallel multi-physics PDE simulation." *Future Generation Computer Systems*, vol. 22, pp. 204–216, 2006.

[9] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. M. Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, "Terascale direct numerical simulations of turbulent combustion using s3d," in *Computational Science and Discovery Volume 2*, January 2009.

[10] P. Colella, D. T. Graves, D. Modiano, D. B. Serani, and B. van Straalen, "Chombo software package for AMR applications," Lawrence Berkeley National Laboratory, Tech. Rep., 2000.

[11] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo, "FLASH: An adaptive mesh hydrodynamics code for modelling astrophysical thermonuclear flashes," *Astrophysical Journal Supplement*, vol. 131, pp. 273–334, 2000.

[12] Y. Yu, D. H. Rudd, Z. Lan, N. Y. Gnedin, A. Kravtsov, and J. Wu, "Improving parallel I/O performance of cell-based AMR cosmology applications," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 933–944.

[13] K. Gao, W.-K. Liao, A. Nisar, A. Choudhary, R. Ross, and R. Latham, "Using subfiling to improve programming flexibility and performance of parallel shared-file I/O," in *International Conference on Parallel Processing, 2009. ICPP '09*, September 2009, pp. 470–477.

[14] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE '08*. New York: ACM, June 2008, pp. 15–24.

[15] S. Kumar, V. Pascucci, V. Vishwanath, P. Carns, R. Latham, T. Peterka, M. Papka, and R. Ross, "Towards parallel access of multi-dimensional, multiresolution scientific data," in *Proceedings of 2010 Petascale Data Storage Workshop*, November 2010.

[16] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout, "PIDX: Efficient parallel I/O for multi-resolution multi-dimensional scientific datasets," in *IEEE International Conference on Cluster Computing*, 2011.

[17] S. Kumar, A. Saha, V. Vishwanath, P. Carns, J. A. Schmidt, G. Scorzelli, H. Kolla, R. Grout, R. Latham, R. Ross *et al.*, "Characterization and modeling of pidx parallel I/O for performance optimization," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 67.

[18] S. Kumar, C. Christensen, J. Schmidt, P.-T. Bremer, E. Brugger, V. Vishwanath, P. Carns, H. Kolla, R. Grout, J. Chen, M. Berzins, G. Scorzelli, and V. Pascucci, "Fast multiresolution reads of massive simulation datasets," in *Supercomputing*, ser. Lecture Notes in Computer Science, J. Kunkel, T. Ludwig, and H. Meuer, Eds. Springer International Publishing, 2014, vol. 8488, pp. 314–330.

[19] B. Summa, G. Scorzelli, M. Jiang, P.-T. Bremer, and V. Pascucci, "Interactive editing of massive imagery made simple: Turning atlanta into atlantis," *ACM Trans. Graph.*, vol. 30, pp. 7:1–7:13, April 2011.

[20] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company, 1966.

[21] V. Pascucci and R. J. Frank, "Hierarchical indexing for out-of-core access to multi-resolution data," in *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer, 2003, pp. 225–241.

[22] V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, S. Philip, and S. Kumar, "The ViSUS visualization framework," in *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, E. W. Bethel, H. Childs, and C. Hansen, Eds. CRC Press, 2012.

[23] V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, and S. Kumar, "Scalable visualization and interactive analysis using massive data streams," *Advances in Parallel Computing: Cloud Computing and Big Data*, vol. 23, pp. 212–230, 2013.

[24] "Visit home page," https://wci.llnl.gov/codes/visit/.

[25] Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins, "Investigating applications portability with the Uintah DAG-Based runtime system on PetScale supercomputers," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, pp. 96:1–96:12.

[26] G. Weber, P.-T. Bremer, M. Day, J. Bell, and V. Pascucci, "Feature tracking using reeb graphs," in *Topological Methods in Data Analysis and Visualization*. Springer, 2011, pp. 241–253.

[27] C. Wang and H.-W. Shen, "Information theory in scientific visualization," *Entropy*, vol. 13, no. 1, pp. 254–273, 2011.

[28] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens, "Revisiting wavelet compression for large-scale climate data using jpeg 2000 and ensuring data precision," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011, pp. 31–38.