Spatially-aware Parallel I/O for Particle Data

Sidharth Kumar
sid14@uab.edu
University of Alabama at Birmingham

Steve Petruzza
spetruzza@sci.utah.edu
SCI Institute, University of Utah

Will Usher
will@sci.utah.edu
SCI Institute, University of Utah

Valerio Pascucci
pascucci@sci.utah.edu
SCI Institute, University of Utah

ABSTRACT

Particle data are used across a diverse set of large scale simulations, for example, in cosmology, molecular dynamics and combustion. At scale these applications generate tremendous amounts of data, which is often saved in an unstructured format that does not preserve spatial locality; resulting in poor read performance for post-processing analysis and visualization tasks, which typically make spatial queries. In this work, we explore some of the challenges of large scale particle data management, and introduce new techniques to perform scalable, spatially-aware write and read operations. We propose an adaptive aggregation technique to improve the performance of data aggregation, for both uniform and non-uniform particle distributions. Furthermore, we enable efficient read operations by employing a level of detail re-ordering and a multi-resolution layout. Finally, we demonstrate the scalability of our techniques with experiments on large scale simulation workloads up to 256K cores on two different leadership supercomputers, Mira and Theta.

ACM Reference Format:
Sidharth Kumar, Steve Petruzza, Will Usher, and Valerio Pascucci. 2019. Spatially-aware Parallel I/O for Particle Data. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337875

1 INTRODUCTION

The continuing growth in computational power available on HPC systems has allowed for increasingly higher resolution simulations. As these simulations grow in size, the amount of data produced grows correspondingly, challenging existing I/O strategies. To address these issues, a large body of work has explored scalable I/O strategies for uniform grid data [1–5], and AMR data [6]; however, relatively little has focused on particle data. Particles are widely used in the simulation community, with large scale runs producing datasets of millions to billions of particles [7–9].

When considering an I/O strategy for particle simulations, portability and scalability are key concerns. The requirement of portability is straightforward: different I/O systems and network architectures may require different strategies, and the I/O system must provide the flexibility to adjust for these differences. With regard to scalability, it is desirable that the I/O strategy support both high-throughput parallel *writes*, essential for checkpointing and saving results, as well as low-latency *reads*, required for post-processing analysis and visualization tasks. However, developing an efficient end-to-end I/O strategy is difficult, as the needs of writes and reads are often at odds with each other. The two tasks have contrasting file-access patterns, and are often executed on different systems, with different levels of parallelism. These contrasting demands are difficult to satisfy simultaneously, and existing I/O methods for particles [1, 3, 9–12] struggle to do so.

To achieve high-throughput writes, current I/O libraries—both for particles and in general—use two-phase I/O [6, 13, 14] and subfiling [10–12]. Two-phase I/O is a technique in which data is moved (or aggregated) across the fast interconnect onto a few chosen aggregator processes before being written to disk. Subfiling is a similar optimization technique, which controls the total number of files across which data is written. However, neither subfiling nor two-phase I/O explicitly enforce spatial locality when writing data to files (see Figure 1). As a result, subsequent visualization and analysis reads performed on fewer processes may need to make unaligned, suboptimal accesses to the files to load the data. Furthermore, particle data can have a highly non-uniform distribution, which a spatially unaware approach will be unable to account for and adapt to.

When working with large-scale simulations, it is often the case that the data does not fit in the available memory, e.g., when using a single workstation for visualization. To allow low latency and low memory visualization of such large-scale datasets, it is common to employ a level of detail [15–19] and multiresolution [15–18, 20–23] strategy, which organizes the data to enable fast reads of representative subsets. However, existing I/O strategies for particle data do not re-organize the particles to support such functionality when writing to disk. Thus, a typical scientific workflow for particle simulations involves a costly data conversion step [15–18, 23], to transform the data into a format suitable for analysis and visualization.

An ideal system for scalable parallel I/O should support both scalable writes and reads across a range of levels of parallelism and HPC system architectures. To reduce loading latency and allow for data streaming and progressive access, the file format used should also support fast spatial and hierarchical read queries. In this paper, we address these needs with a spatially-aware parallel

ICPP 2019, August 5–8, 2019, Kyoto, Japan

ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337875

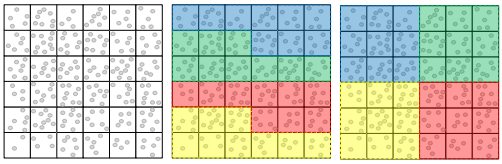


Figure 1: A distributed simulation on 36 processes (left). The particle data is aggregated from 36 to 4 processes without spatial awareness (middle, grouped by color), producing output files with poor spatial locality. Our spatially aware aggregation (right), produces files with a spatial layout better suited to visualization reads.

I/O framework, which keeps communication to a minimum and ensures that only spatially near particles are grouped together during aggregation. The output of this approach are files with good spatial locality, which are well suited for visualization and analysis tasks. In particular, this is the first work that tries to solve the problem of simultaneously attaining both high-bandwidth parallel I/O and low-latency visualization-appropriate reads for particle data. Our contributions are:

- A file format which allows efficient visualization and analysis queries, by leveraging data locality and hierarchy in the file; and
- A spatially aware adaptive aggregation I/O strategy to write particle data in this format, while still achieving high throughput.

Finally, we report a detailed evaluation using datasets representative of common particle-based simulations, where our approach yielded maximum achievable throughput on Theta [24], and 50% of the maximum throughput on Mira [25] using 1/3 of the system.

2 RELATED WORK

While scalable I/O for grid-based and structured datasets has been widely explored, less work has focused on particle data, in part due to the challenges of working with the unstructured and unevenly distributed nature of the data. Common approaches for I/O of particle data use either a single shared file [8, 12, 26], or a file per-process [7]. While such approaches can work well on a single HPC resource, or provide tolerable performance at some scales, both methods are known to not scale to large core counts, or be portable to machines with different I/O architectures.

To improve I/O performance and reduce latency for visualization and analysis tasks, a wide range of visualization focused file formats have been proposed. A common trend among all such formats is that they provide both a hierarchical representation, and reorganize the data for better spatial coherence. To visualize large cosmological datasets, Fraedrich et al. [15] construct a multi-resolution hierarchy as a post-process task, which optimizes for level of detail and fast read performance. Similar efforts by Reichl et al. [17], Schatz et al. [16] and Rizzi et al. [18] have presented interactive, level of detail based visualization methods for large scale cosmology simulations, all based on either a post-process data conversion [16, 17] or a re-sorting process, which is run when the visualization task starts [18]. Wald et al. [23] presented a low memory overhead acceleration structure and file format for particle visualization, which is also constructed by re-sorting the data in a post-processing step.

Although such visualization focused file formats have been shown to be highly effective, they each require some up front data conversion or processing step to be run, either as a pre-process or when the application launches. These pre-processing steps can take time ranging from hours [15], to a day [17] or a week [16], depending on the output format and data size. This additional conversion step is time consuming, and requires making a duplicate copy of the data, both of which become more expensive as the data size increases. Thus, outputting the data from the simulation in a format which is natively optimized for visualization tasks is clearly desirable, to eliminate this bottleneck.

Recent work on scalable I/O for particle data has leveraged sub-filing [9, 10] to improve scalability. Habib et al. [9] use an approach on IBM Blue Gene/Q which groups processes by their I/O node to determine the sub-filing assignment, to aggregate data for cosmology simulations with up to 1 trillion particles (47TB) and report achieving up to 118GB/s bandwidth.

2.1 Parallel HDF5

Byna et al. [10] examine tuning HDF5's built in support for sub-filing on two particle simulation I/O kernels. While HDF5's sub-filing improves performance over a single shared file, Byna et al. report encountering significant limitations with HDF5's sub-filing. Specifically, they report errors running at more than 32K processes or beyond certain sub-file counts. They report reaching 67% and 28% of peak bandwidth at 16,384 cores on Edison and Cori respectively, with our approach (Section 5) we achieve 50% and 100% of peak bandwidth at 256k cores on Mira and Theta, respectively. Furthermore, the files written by Byna et al. are not suitable for post-process visualization at lower core counts, as they report the number of reader processes and sub-filing factor must match the write configuration. Our approach (Section 5) on the other hand allows reads with different core counts than were used to write the data. Furthermore, Our approach allows users to read data at varying resolution, a feature not provided by other I/O systems. We also compare HDF's performance with our system in Section 5.2.

3 SPATIALLY AWARE TWO-PHASE I/O

Sub-filing and two-phase I/O are commonly employed aggregation techniques to balance the tradeoff between file per-process and single file (i.e., collective) I/O. However, existing I/O systems typically treat particle data as a stream of bytes, and ignore spatial correspondence in the data when applying sub-filing and two-phase I/O. Without spatial domain knowledge, it is possible that processes with particle data from distant regions of the domain will be grouped into the same file, producing a poor spatial mapping of the data on disk. Such groupings can occur when using methods which only consider the network topology in determining the aggregators, as the best decomposition in this sense is not necessarily a good decomposition of the particles (Figure 1). Files written without spatial knowledge can incur in significant costs when used for post-process analysis and visualization. For example, if the user wants to perform distributed rendering on four nodes the spatially-aware mapping of the data in Figure 1 requires only a single file to be read by each process, while the spatially unaware mapping requires each process to read two files and perform more seek operations

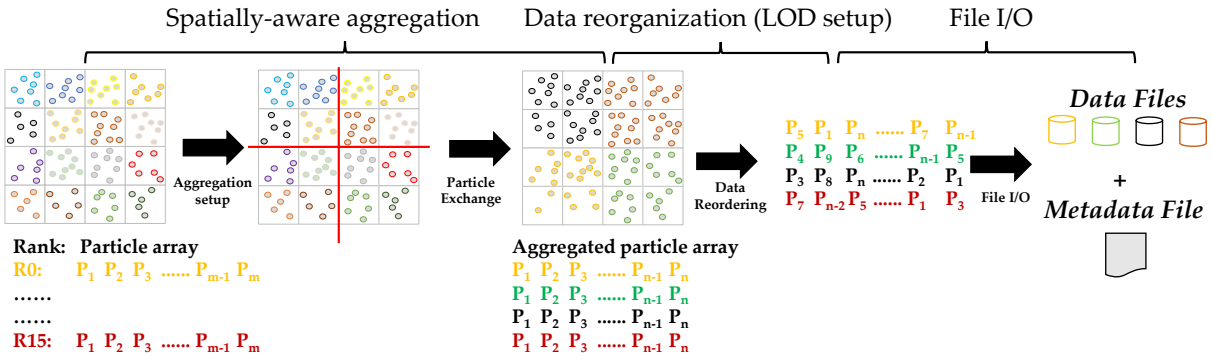


Figure 2: An illustration of our two-phase I/O approach, which takes spatial locality into consideration.

within those files. It is important to note that along with rendering, a range of standard analysis and visualization tasks are dependent on region-based queries, e.g.: nearest neighbour search, vector field integration, stencil operations, image processing.

There are two main challenges in designing a spatially aware I/O strategy: creating a mapping between the location of a particle and its location in the file, and using the mapping to create a spatially aware data aggregation phase. For structured data (i.e., grids) there exists a direct, one-to-one mapping between a voxel's 3D index to a location in the file. Voxels then can be mapped to locations in the file using some ordering scheme, e.g., row-order, Z-order, or HZ-order. However, the unstructured nature and unbalanced distribution of particle data makes it difficult to define a similar fine-grained mapping in general. Furthermore, moving particles over the network during the aggregation phase is more challenging for structured data. While for structured data the bounds of any data packet can be directly used to identify which aggregator process it belongs to and where in that aggregator's buffer it will be placed, there is no such correspondence for bulk communication of particles. If a process's data is split into two aggregators, it must loop through the particles to determine which aggregator they belong to.

We tackle these challenges by first building a correspondence between a particle's spatial location and its position in the file. To this end we begin by imposing a rectilinear 3D-grid—referred to as the aggregation-grid—on the simulation domain. The aggregation-grid spatially partitions the simulation domain into a mesh of axis-aligned 3D-boxes, which we refer to as aggregation partitions. We assign a processor-rank r to each of these aggregation partitions. The aggregation-grid spans the entire simulation domain, placing each particle within a unique aggregation partition. With this setup we initiate the particle exchange phase, where all particles residing within an aggregation partition are transferred to the process r assigned to the partition. At the end of this phase, all particles residing within an aggregation partition are aggregated to a single process. Next, we re-order the aggregated particles into a level of detail ordering. The level of detail ordering is used when reading data to support multi-resolution queries, for streaming and from low-memory systems. Finally, each process holding the aggregated data writes the buffer out to an independent file. These output files contain particles which are spatially local, and contained within a disjoint sub-region of the domain.

Our system is flexible enough to handle aggregation-grids that are not aligned with simulation's partitioning of the domain, in which case a process might have to send data to multiple aggregators. However, this requires each process loop through its set of particles to determine which aggregator they belong to. For a uniform resolution simulation, where every process works on patches of equal size, we ensure that the aggregation-grid is always aligned with the simulation grid. By aligning the aggregation-grid with the simulation's we avoid needing to filter the particles, while preserving communication locality and minimizing data movement during the aggregation phase. We note that our aggregation scheme is not limited to uniform domain decompositions, and can easily adjust to adaptive resolution simulations, or simulations with uneven distributions of particles (see Section 6).

We discuss an example of our I/O scheme (in 2D) in Figure 2. The simulation partitions the domain into a uniform grid, with subsets of particles within each patch, colored by their rank. In the aggregation step, we impose an aggregation-grid composed of four aggregation partitions on the domain, with each aggregation partition owned by a specific rank. The particles within each aggregation partition are colored by the aggregator assigned to the region (black, yellow, green, purple). The particles within each region are then sent to the owner of the aggregation partition, which re-orders the particles into the desired level of detail ordering. Finally, each aggregator writes its data independently to a separate file. The I/O scheme can be broken down into the following steps.

(1) Setup aggregation-grid (Section 3.1)
(2) Select aggregators (Section 3.2)
(3) Exchange metadata and data (Section 3.3)
(4) Allocate aggregation buffer (Section 3.3)
(5) Particle exchange (Section 3.3)
(6) Particle shuffling for LOD (Section 3.4)
(7) Write data to file (Section 3.4)
(8) Write spatial metadata (Section 3.5)

3.1 Aggregation-Grid Setup

We begin by imposing an aggregation-grid on the simulation domain. To align the imposed grid with the simulation's domain partitioning we enforce that the aggregation-grid starts at the same coordinate as the simulation grid, and set the aggregation partition size to an integer multiple of the per-process domain size.

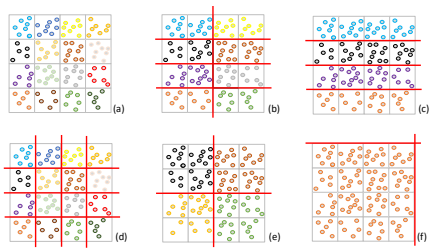


Figure 3: The choice of aggregation-grid size is directly tied to the number of files produced, and the amount of communication. Aggregating the particles (a) with configuration: (b) 2x4 outputs 8 files, (c) 1x4 outputs 4, (d) 4x4 is equivalent to file per-process, (e) 2x2 also outputs 4 files, (f) is equivalent to shared file I/O. The best configuration is both machine and workload dependent, and is exposed as a tuning parameter to users.

We define the aggregation partition factor, abbreviated as P_x, P_y, P_z, as the ratio of the size of an aggregation partition to the simulation's patch size in the x, y and z dimensions, respectively. The aggregation partition factor (P_x, P_y, P_z) impacts both the degree of network traffic during the aggregation phase and the total number of outputted files. In particular the total number of outputted files is $f = (n_x/P_x) \times (n_y/P_y)$. For example with $4 \times 4 = 16$ processes and $P_x \times P_y = 2 \times 2$, the total number of generated files will be $(4/2) \times (4/2) = 4$ (see Figure 3e). Alternatively, choosing $P_x \times P_y \times P_z = 1 \times 1 \times 1$ corresponds to an extreme where each of the aggregation partitions and the process patches are the same, and is equivalent to file per-process I/O (see Figure 3d). At the other extreme, the aggregation partition can be set to be the same size as the entire domain, resulting in an all-to-one aggregation which will save out a single file, equivalent to single shared file I/O. For most scenarios, the latter configuration is not feasible due to limitations in the available memory on a single core. Note that a process's patch offset and size in a particle simulation represents the spatial boundary within which all the particles on that process are contained. Although this size and offset information is typically available from the simulation, the I/O system can easily compute this information by finding the bounding box of the particles on the process.

The amount of communication performed during aggregation is also dependent on the partition factor. As shown in Figure 3, communication during the data aggregation phase is localized to each aggregation partition, confined to a group of $P_x \times P_y \times P_z$ processes. Therefore, increasing the partition factor along an axis will lead to communication between a larger number of processes. The best partition factor is dependent on multiple factors, such as the machine's I/O architecture and network topology, and the number of particles and variables in the simulation. To allow users to adjust performance to best suit their platform and simulation, we expose the partition factor as a tuneable parameter, and conduct a detailed sensitivity study in our evaluation (Section 5).

3.2 Aggregator Selection

With the aggregation partition factor set, the next step is to select the aggregator processes. We assign one process for each aggregation partition to be responsible for receiving data from the other processes, and saving it to disk. These aggregators are chosen uniformly from the rank space, to ensure even utilization of the network. For example, with 16 participating processes and 4 aggregation partitions, we assign processes with ranks 0, 4, 8 and 12, to be the aggregators. While it can be the case that the chosen aggregator for a partition's data resides in a different partition, spatially neighboring processes may not be close in the network topology, and hence, we choose a scheme which ensures a more even utilization of the network [27].

3.3 Metadata and Data Exchange

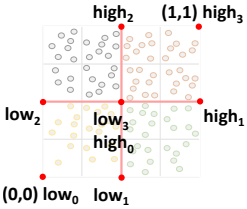
Once the aggregation-grid is in place and aggregators assigned, the processes initiate the data aggregation phase. However, unlike grid-based data, for unstructured particle data the aggregators do not know a-priori how many data packets to expect, nor how big a buffer to allocate to hold the set of particles they will receive. Similarly, the transmitting processes do not know a-priori the size of data packets they will send to their assigned aggregator(s). Thus, before the actual particle data exchange phase, we perform a metadata exchange where processes send to their aggregator(s) the number of particles they will be sending during the actual aggregation phase.

For a uniform-resolution simulation, where all processes have the same size patches, we accelerate this phase by aligning aggregation partitions with the simulation's domain partitioning. By aligning our imposed aggregation-grid we ensure that each process will send all its particles to a single aggregator. In general, if the aggregation partitions are not aligned with the simulation, each process must first identify the aggregation partitions it intersects with and perform a scan through its particles to group them by the partition they fall into.

After the metadata exchange phase, we perform the actual exchange of particles. For aligned aggregation-grids, the domain of each process is always contained inside a single partition, in which case each process can simply send all of its particles to the process which owns the partition. For non-aligned aggregation-grids, each process instead sends the bundle of particles compiled during the metadata exchange phase to the respective aggregators for each partition. We use non-blocking MPI point-to-point communication for both metadata and data exchange.

3.4 Level of Detail Particle Data Layout

After the particles are collected by their respective aggregators, they are reorganized to provide a layout suited for multiresolution access for post-processing analysis and visualization tasks. These tasks are typically performed on small clusters or workstations, where smaller memory capacities and slow data access can heavily affect overall performance. In order to facilitate fast analytics and visualization-appropriate low-latency reads, we employ a level of detail reordering of the particles. Level-of-detail (LOD) ordering provides multiresolution access allowing users to perform analytics at varying scales. The order of particles used to create the levels of detail can be defined using different kinds of heuristics such as density or random. As an example, our format implements reordering as a random reshuffling. The particles are reordered in-place, producing a sequence of subsets that represent different levels of detail. The reordered particles are then written out as one long



Box #	Agg rank	File name	Low	High
0	0	File_0	0.0, 0.0	0.5, 0.5
1	4	File_4	0.5, 0.0	1.0, 0.5
2	8	File_8	0.0, 0.5	0.5, 1.0
3	12	File_12	0.5, 0.5	1.0, 1.0

Figure 4: An example of a spatial metadata file. The columns highlighted in grey are written to the file. Agg rank is used to derive the name of the data file.

sequence where each level follows the previous. When reading from this data format, applications can read subsequent levels by simply reading further in the file.

As a result of random shuffling, the data format created is dynamic in nature, with the number of particles (x) in every level (l) depending on the number of processes (n) accessing the data (during *reads*). We define each level of detail l (i.e., LOD) as a subset of at most $x(n, l) = n \cdot P \cdot S^l$ particles, where n is the number of processors reading the data, P is the number of particles per reading process contained in the first level, and s is a resolution scale factor that can be varied by the user (S defaults to 2). As can be seen from the formula, the resolution scale S acts as a multiplier that exponentially increase the number of particles going from one level to the next. For example, when reading a dataset containing 100 particles on one core (i.e., $n = 1$) with $P = 32$ and $S = 2$, the first level (level 0) will contain 32 particles ($1 \cdot 32 \cdot 2^0$), the second (level 1) 64 ($1 \cdot 32 \cdot 2^1$) and the third the remaining four particles ($100-64-32$). The different LODs are simply subsets of the data, and as such we do not incur any additional storage overhead compared to strategies which compute additional representative particles for the coarser LODs. With this format users can read a file linearly up to some desired level of detail. The LOD upto which one reads data can be based on the available memory on the machine, or some other metric (e.g., screen- space projected area, RMSE).

The computation time for reordering the particles is not a significant bottleneck, we find that for 32K particles it requires 33 msec on Mira and 80 msec on Theta. We note that while Theta has more cores per processor, the single core performance is lower than Mira's, and our reordering is not currently parallelized.

3.5 Writing Spatial Metadata

In addition to writing a hierarchy of data files, we also write an additional metadata file. The metadata file stores a list of bounding boxes corresponding to the particles stored in the data files. An example of the metadata file for the particles written in Figure 2 is shown in Figure 4. Our aggregation scheme ensures that the bounding boxes of particles in the data files are unique and non-overlapping. The metadata file holding the bounding box of data files is used when *reading* the data for analysis and visualization tasks. We populate this metadata file on rank 0, using an MPI_Allgather to collect the bounding boxes of all aggregator-processes. The collected data is then written to a binary file with the format shown in Figure 4. We plan to further extend the metadata format by storing, e.g., the minimum and maximum values of scalar fields of the region as well. Such metadata can be used to narrow down range-queries on these non-spatial attributes (e.g., density, pressure or temperature). We discuss we leverage the metadata file information to facilitate fast reads in the following section.

4 SCALABLE PARALLEL READS FOR ANALYSIS AND VISUALIZATION

In this section we present three key factors that work together to make our parallel reads fast and scalable. In particular, we focus on reads for analysis and visualization tasks where the number of processes reading the data is typically much smaller than the number of processes that were originally used to write.

First, our I/O format generates fewer data files compared to the traditional file-per-process I/O, thus parallel reads on our format require opening and accessing fewer files. For example, generating a dataset at 64K processes with our data format and configuration of $P_x \times P_y \times P_z = 2 \times 2 \times 2$, produces 8K files, whereas file-per-process I/O would generate 64K files. Reading the dataset with say 512 processes will have every process open 128 ($64K/512$) file with file-per-process I/O as opposed to opening only 16 ($8K/512$) files with our I/O scheme.

Second, our format writes data in a spatially coherent manner, where files are written with spatial knowledge of particles. This further reduces the number of file accesses when reading back the data. For example, if the user wants to perform distributed rendering on four nodes the spatially-aware mapping of the data in Figure 1 requires only a single file to be read by each process, while the spatially unaware mapping requires each process to read two files and perform additional seek operations within those files.

Finally, our spatial metadata file lets processes pick which data file to read. As mentioned in Section 3.5, along with data files we also write a metadata file storing the spatial extents (bounding-box) of the data files. If we consider a generic parallel read operation to perform an analysis or visualization task, processes typically need to read particle data from some specific spatial regions (box-query) of the domain. With our format, any process making such reads simply uses the bounding box information stored in the metadata file to select exactly which file to read. Note, reading and parsing the metadata file is a lightweight I/O task. Reading data from a format devoid of such metadata requires every process to read all particles across all the files and then cherry-pick the relevant particles. As a matter of fact, this solution will produce undesirable performance.

Furthermore, our format not only allows selective access of files but it also grants the ability to read particles within the files at increasing levels of details. Recall from section Section 3.4, we deploy reordering to create an implicit LOD hierarchy within particles. Every file contains particles in a order that identifies a sequence of level of details. A file can be read linearly up to a desired LOD, which can be chosen, for example, based on the available memory on the machine, or some other metric (e.g., screen-space projected area, RMSE). If more accuracy is desired or more memory is available, the application can read and append another level of data to the previously loaded particles to provide progressive refinement. The layout makes our format especially useful for visualization applications which leverage progressive refinement to work with large datasets. For example, an application can query a low level of detail to quickly display a representative subset to the user, and

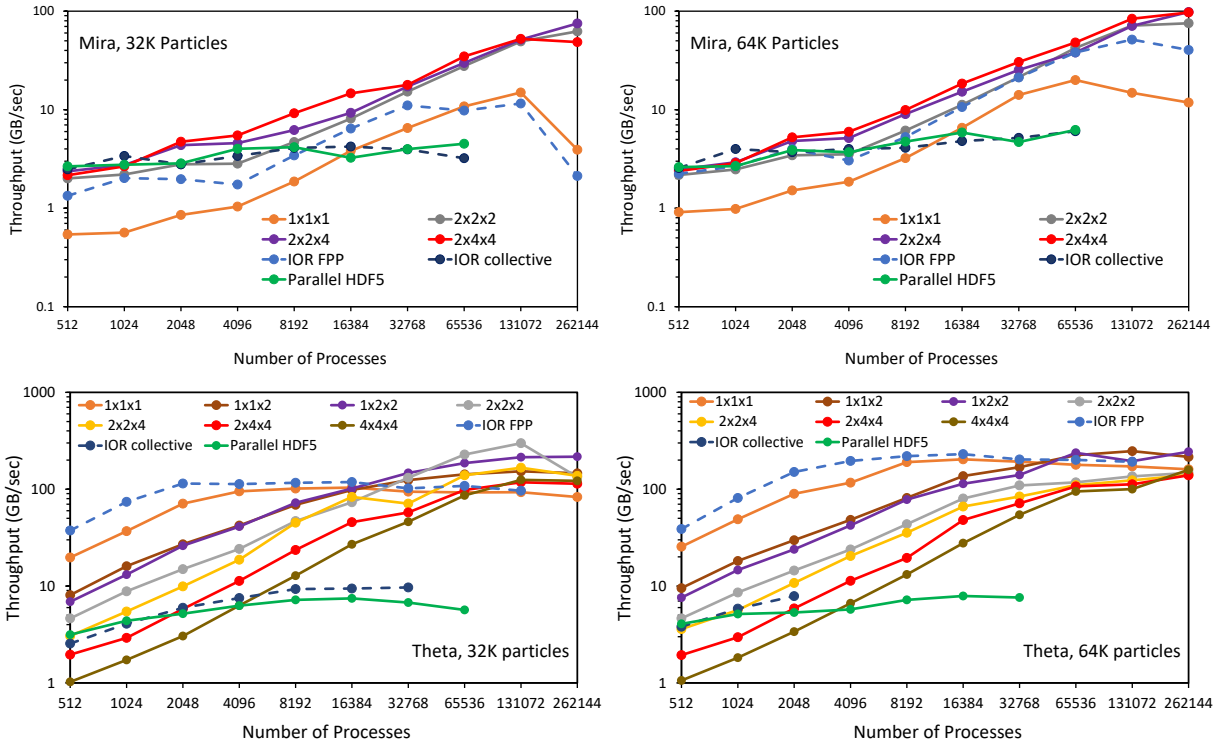


Figure 5: Parallel write weak scaling for different configurations of our spatially aware two-phase I/O strategy on Mira (first row) and Theta (second row). The ideal choice of aggregation configuration is highly machine and workload dependent, and is left exposed to users as a tuning parameter. We also report IOR and Parellel HDF5 experiments as reference for file per process and collective I/O performance.

over time in the background, load subsequent levels to refine the image until some set quality threshold or memory limit is reached.

5 EVALUATION

In this section we report the evaluation of our I/O system on Mira [25] and Theta [24] supercomputers. First, we demonstrate the scalability of our system in performing parallel writes (Section 5.2), and discuss how our system can be effectively tuned for the two platforms. We then demonstrate the efficacy of our system in performing full resolution (Section 5.3), and LOD reads (Section 5.4).

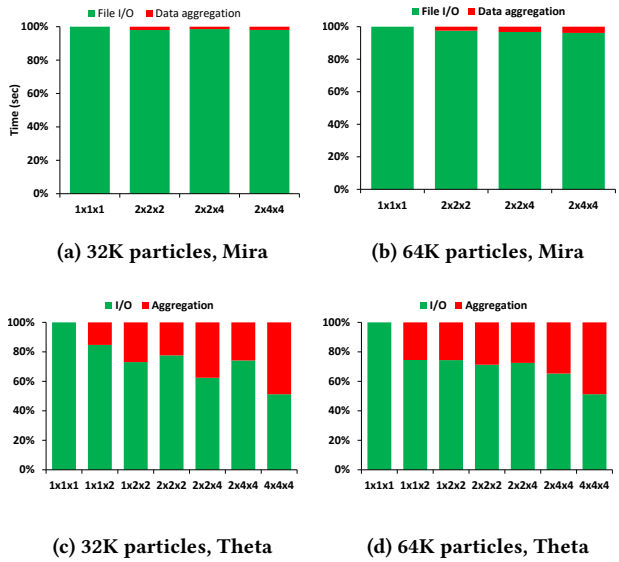
5.1 Experimental setup

For all our experiments, we used datasets representative of the I/O workload of the Uintah simulation Framework [7]. Uintah is a general purpose software framework, used in the development of components for numerical modeling of fluid-structure interactions, computational fluid dynamics, solid mechanics and multi-physics simulation. Specifically of interest for our work is Uintah's support for a range of particle based models, e.g., the material point method [7] and Lagrangian particle transport [28], providing a range of datasets representative of common use cases of particle data. The simulation framework has shown scalability to core counts approaching 768K. In our experiments we used two workloads with 32,768 and 65,536 particles per core. Each particle is represented by 15 double precision values (i.e., position vector with 3 components, stress tensor with 9 components, density, volume, ID), and 1 single precision variable (i.e., type). For the two workloads this configuration corresponds to 4 and 8 MB respectively, data per core for each timestep. To provide baseline file per process and collective I/O comparisons for our results we also run IOR [29] and Parallel HDF5 (i.e., using the *h5perf* tool) [30] benchmarks to write data equivalent to the 32K and 64K particle per core configurations. As we are more interested in the performance a simulation would see when saving data, our benchmarks are run without fsync, with the consideration that a simulation would not wait to fsync before continuing on to the next timestep. Similarly, the IOR benchmarks are also run without fsync, for both file per process and shared file I/O. This configuration should yield peak performance for file-per-process I/O on a Lustre file system. On the Lustre system we use 48 stripes (48 OSTs), with the stripe size set to 8MB, as recommended by the ALCF guidelines for I/O performance on Theta [31].

We ran our experiments on two platforms with different network and filesystem architectures: ALCF's Mira [25], an IBM BlueGene/Q using a 5D Torus network topology and GPFS filesystem [32]; and Theta [24], a Cray machine with Intel Xeon Phi processors with Dragonfly topology network and Lustre file system. We also used a workstation with 4 Intel Xeon CPUs (18 cores each), 3TB of RAM and two SSD drives for read performance evaluation.

5.2 Parallel Writes

On both Mira and Theta we perform two sets of weak scaling experiments, one for 32K particles per core, and one for 64K particles



(a) 32K particles, Mira

(b) 64K particles, Mira

(c) 32K particles, Theta

(d) 64K particles, Theta

Figure 6: Time profiles for different aggregation configurations on Mira (a,b) and Theta (c,d). We observe that on Theta more time is spent in aggregation (communication) compared to Mira, for the same configurations. This indicates that fewer partitions, and thus less communication, should be preferred on Theta.

per core. On both machines, we varied the number of processes from 512 to 262,144, thus varying the amount of data generated per timestep from 2 GB to 1TB (32K particles per core), 4 GB to 2 TB (64K particles per core). For all our runs we varied P_x, P_y, P_z, using configurations $(1, 1, 1)$, $(1, 1, 2)$, $(1, 2, 2)$, $(2, 2, 2)$, $(2, 2, 4)$, $(2, 4, 4)$ and $(4, 4, 4)$. Recall from Section 3.1 P_x, P_y, P_z, controls both the extent of communication performed during aggregation, and the number of generated files. Larger values of P_x, P_y, P_z correspond to more communication and fewer output files. In some preliminary scaling runs, we observed that Mira performed better with larger P_x, P_y, P_z compared to Theta, which had better performance at smaller P_x, P_y, P_z. Based on this observation, we reduced the number of experiments performed on Mira and did not run experiments for configurations $(1, 1, 2)$ and $(1, 2, 2)$.

We plot the results of our experiments on Mira in Figure 5 (top row), along with the IOR (dashed trend lines) and Parallel HDF5 (green trend line) results. We observe that both IOR file per process and our I/O scheme's file per process configuration $(1, 1, 1)$, starts to saturate at very high process counts (at 131,072 processes for 32K particles, and 65,536 processes for 64K particles). At this scale file per process I/O encounters issues with the network file-system failing to handle the large number of files efficiently. IOR's shared file I/O and PHDF5 also do not scale, as collective I/O starts to overload the network when gathering data, dramatically decreasing its performance at scale. We observe that configurations $(2, 2, 4)$ (purple line) and $(2, 4, 4)$ (red line) of our spatially aware two-phase I/O scales all the way to 262,144 processes. At 262,144 processes we obtain a maximum throughput of 98 GB/second while writing a total of $\approx$ 17 billion particles. This high performance can be attributed to two factors, a scalable aggregation phase that takes a small portion of total I/O time, and the generation of fewer, larger files providing a bigger I/O burst size compared to file-per-process I/O.

The first claim can be verified from Figure 6a and b, which shows a breakdown of the percentage of time spent aggregating data over the network and the actual time spent writing files to disk, for the experiments at 32K cores with 32K and 64K particles per core. As expected, for both runs we observe an increase in aggregation time with more aggregation partitions; however this percentage remains small compared to the actual file I/O time. This demonstrates that on Mira the time spent in writing files to the storage system is significantly higher than the time spent collecting the data over the network.

Our second claim is supported by computing the aggregated file size for each configuration, using the equation discussed in Section 3.1. As the aggregation-grid increases, we generate fewer and larger files, with bigger I/O burst size. For example, with 32K particles per-process at 4096 process, file per-process I/O will produce 4096 files, each 4MB; however, aggregating with a (2, 2, 4) grid will produce 128 files, each 128MB.

We plot our results for Theta in Figure 5(second row), and find that Theta exhibits completely different performance characteristics compared to Mira. Theta's Lustre filesystem is known to scale well with file per process I/O, and thus IOR with fsync disabled should give the filesystem's peak performance. As expected, we observe that both IOR's file-per-process (dashed light blue lines) and our I/O scheme's file-per-process (dark yellow) configuration $(1, 1, 1)$ scales much better than Mira. In fact, file per process reaches peak performance at most core counts. However, at high core counts file per process I/O's performance starts to flatten out, as the file creation time for the large number of files begins to dominate the actual I/O time.

While the $(1, 2, 2)$ configuration (purple line) is outperformed by file per process at lower process counts, it continues to scale at higher process counts, finally, outperforming file-per-process I/O at 65,536 processes. Using $(1, 2, 2)$ configuration for 32K and 64K particles per core, we achieve a maximum throughput of 216 GB/second and 243 GB/second at 262,144 processes, respectively. File-per-process I/O on the other hand yields a throughputs of 83 GB/second and 160 GB/second. Shared file I/O on Theta yields sub-optimal performance, which can be attributed to a more expensive aggregation phase.

To determine how the different configurations affect the overall performance on Theta, and differences between Theta and Mira, we examine the time breakdown between communication and file I/O for different configurations (Figure 6). On Theta we observe that the aggregation of data over the network is far more expensive than on Mira, and takes a greater percentage of time overall as the aggregation-partition size (and thus number of communicating processes) increases. This observation, combined with the Lustre system on Theta preferring more independent files explains why we observe file per process I/O performing so well compared to Mira. In line with this preference for file per process I/O over communication, we find better performance when aggregating among smaller groups of processes on Theta (i.e., $(1, 1, 2)$, $(1, 2, 2)$). These results are in contrast to our findings on Mira, where aggregating among larger groups of processes gives better results. By exposing

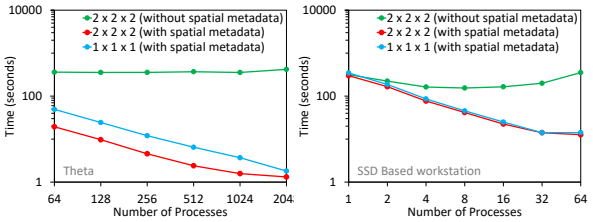


Figure 7: Visualization style read strong scaling on Theta and a single workstation, reading a 2 billion particle dataset written on 64K cores. Our approach, using spatial correspondence, provides good strong scaling and read performance, even at far fewer cores than the data was written with. On the contrary, reading from file formats with poor or no spatial correspondence are significantly slower and their performance decrease at scale.

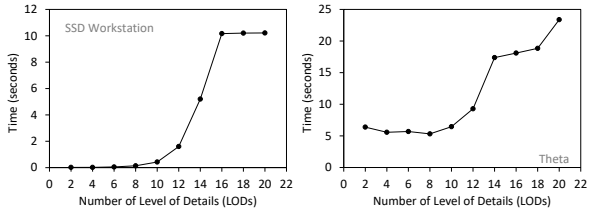


Figure 8: Level of detail read performance on an SSD Based Workstation (a) and Theta (b) using 64 cores to read progressively higher levels from a 2 billion particle dataset. Our ordering allows fast reads for low levels of detail, and does not increase significantly until reading large portions of the dataset.

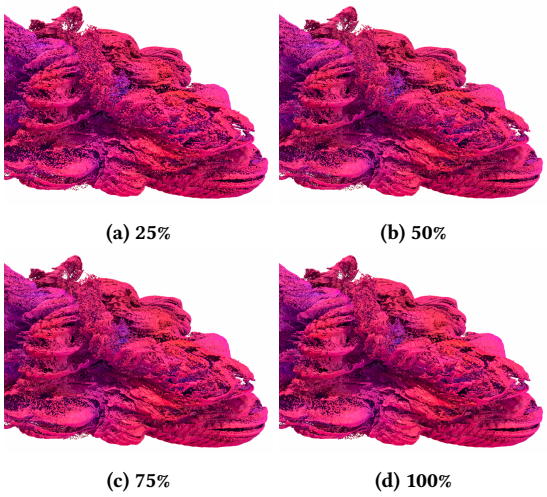


Figure 9: A zoomed in view of a coal particle injection simulation dataset, with 55 million particles, written using a random reshuffling for level of detail. From left to right: progressively more particles are read, from (a) 25% to (d) 100% of the data. Lower resolutions using this LOD ordering can still provide a good representation of the data, and be read quickly using our I/O strategy.

the aggregation partition factor as a user tuneable parameter, our approach can be used effectively on both architectures.

5.3 Visualization Reads

In contrast to simulations, which perform massive read and write operations, visualization and analysis tasks are less compute bounded, and instead are often performed on more modest machines and core counts than the simulation was run on. For example, the output of a simulation run on Mira or Theta at hundreds of thousands of cores could then be visualized on ALCF's visualization clusters like Cooley, which has just 1512 cores total; or using a smaller job on Mira or Theta; or even on a user's workstation.

To evaluate the scalability of our approach we conducted a series of strong scaling experiment. We generated three datasets of the same size, using 64K processes with a per-process load of 32K particles. In the first case the dataset was written with an aggregation configuration of (2,2,2) with no spatial metadata file. In the second case the dataset used was written with the same aggregation configuration of (2,2,2), but with the spatial metadata file. Note that in both these cases the dataset contains 8K files. Finally in the third case, we read a dataset written using a file-per-process aggregation configuration of (1,1,1) (total of 64K files), but with the spatial metadata file. We ran our parallel reads on two different platforms: Theta and a local workstation (configuration in Section 5.1) equipped with an SSD drive. On Theta we varied the number of processes from 64 to 2048 and for the workstation we varied the number of processes from 1 to 64. The results are plotted in Figure 7.

For both machines, as expected the second case (red line), which uses spatial metadata and has fewer files, provides the best performance and scaling. For the first case (green line), the lack of spatial information forces every process to read the entire set of particles (i.e., from 8K files). Therefore, in this case adding more processes does not reduce the per-process I/O load; as a result it exhibits the worst performance on both machines. Finally, we observe for the 3rd case (1,1,1 configuration, blue line) reading large number of files has a stronger impact on Theta as compared to the SSD based workstation. On the workstation with SSDs the time taken to read dataset across a large hierarchy of files (i.e., in this case 64K) is almost comparable to reading from a smaller hierarchy of files (2nd case, 8K files). For the third case we also observe that, although the large number of files reduces the overall performance, the spatial information present in the metadata (and in the format data layout) still allows this approach to scale well (i.e., time reduces with more cores).

5.4 Level of Detail Reads

In this section we report the time to read progressively levels of detail from a 2 billion particle (2^{31}) dataset using 64 cores ($n = 64$). The dataset was originally written at 64K cores using the $2 \times 2 \times 2$ aggregation configuration (generating 8K = 64K/$2 \times 2 \times 2$ files). For our experiments we set P to be 32 and S to be 2, recall P is the number of particles per reading process contained in the first level and S is the resolution scaling factor. Working with this configuration ($n = 64$, $P = 32$, $S = 2$) we can have upto 20 level of details (i.e., using the formula $x = n \cdot p \cdot s^l$ defined in Section 3.4, where $x = 2^{31}$ particles we find that $l = log_2[2^{31}/(64 \cdot 32)] = 20$). We measure the time taken to read progressively increasing number of levels (starting with reading two levels all the way to reading all

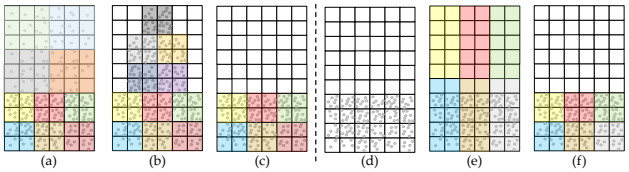


Figure 10: (a-c) Examples of non-uniform particle distributions typically found in simulations. The colored boxes correspond to the adaptive aggregation-grid which takes into account the spatial distribution of the particles to load balance I/O and network traffic, while ignoring regions without particles. (d-e) A non-uniform distribution of particles (d), written without adaptivity would assign aggregators to regions without any particles (e). Our spatially adaptive aggregation scheme takes into account the particle distribution, and adapts the aggregation grids correspondingly (f).

20 levels). In Figure 8 we report the results of our experiments on Theta and an SSD based workstation.

On Theta we observe that the first few levels can be read in about the same time, this is mainly because the total I/O time is dominated by large number of file opening operations rather than actual reads (every process opens approximately 8k/64 files). On the other end, after level of detail 8 we observe a time increase that is proportional to the number of particles until the last level (i.e., 20) where the timing is equivalent to reading the entire dataset using 64 cores (as seen in the Figure 7). On the SSD workstation we see a slightly different trend. As opposed to Theta, for initial lower levels we observe time increasing proportionally with the number of particles being read. This result is in accordance to the trend we saw in Figure 7, where the workstation with SSDs was showing very minimal performance deviation with respect of number of files being read. Overall we note that the time spent to load lower LOD on the SSD workstation is small enough to enable interactive analysis and visualization tasks.

Using this ordering, level of detail can be implemented in a visualization application by loading progressively larger subsets of the data over time. The lower levels of detail can still provide an accurate view of the data, and the structure can be preserved by increasing the particle radius [19]. To evaluate the use of our LOD framework to perform progressive visualization we produced some particle rendering at different resolutions. In Figure 9, we show how when higher levels of detail are loaded, the radius is decreased to provide a smooth progression from low to high detail visualization. We can also observe how most of the features are still visible even using only 25% of the particle data.

6 PARTICLE DISTRIBUTION AND ADAPTIVE AGGREGATION

Simulations tend to balance particles evenly across processes; however, it is common to have a non-uniform distribution of particles across the spatial domain. In particular, we consider two common non-uniform distribution of particles: one where some spatial region of the simulation domain has a lower particle density compared to others (Figure 10a), and one where some spatial region of the domain has no particles at all (Figure 10b and c). These scenarios are especially common in simulations where particles move toward

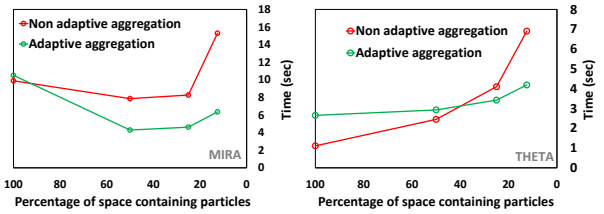


Figure 11: Write time comparison of adaptive aggregation vs. non-adaptive aggregation for increasingly non-uniform particle distributions. On both Mira and Theta we find our adaptive approach improves performance.

specific regions, or are used to represent physical materials, or are injected over time.

Layout-agnostic aggregation schemes do not take into account the spatial distribution of the particles, and as a result, may assign aggregators for regions that do not have particles at all (Figure 10e), or assign more aggregators than are needed in regions with a low density of particles, underutilizing the I/O system and network. However, our spatially-aware I/O scheme is layout-aware and takes into consideration the particle density over the simulation domain, creating an adaptive aggregation-grid to properly load balance the particle data (Figure 10f). The adaptive grid places aggregators uniformly across the entire rank space, and ensures that no aggregator is assigned to empty simulation domain.

To create the adaptive grid, processes perform an all-to-all exchange and send each other their spatial extents, and the number of particles within their extents. The adaptive grid is built by determining the subregion of the simulation domain where the particles reside, based on the exchanged spatial extents and density information. The aggregation-grid is then adjusted to partition just those regions which contain particles, after which the aggregation and I/O phases are performed as described previously. Processes without particles do not participate in the subsequent stages at all.

6.1 Adaptive Aggregation Writes

To study how well our adaptive aggregation works to handle uneven particle distributions, we simulate various levels of uneven distributions using 4096 cores. We divided the domain into 4096 regions of equal size, within which we generate particles that are distributed over progressively smaller portions of the domain, ranging from covering the entire domain, to 50%, 25%, down to only 12.5% of the domain. With particles occupying smaller percentage of the domain, some processes will have a higher density of particles, while others may have none at all (see Figure 10d). We compare the performance of our layout-aware *adaptive aggregation* scheme (Figure 10f) against a layout agnostic *non-adaptive aggregation* scheme (Figure 10e). The results are plotted in Figure 11.

Overall we find that adaptive aggregation (green line) yields improvement over non-adaptive aggregation (red line). On Mira, we observe that as the domain occupied by particles decreases from 100% to 50%, I/O time reduces significantly with adaptive aggregation. The reduction in time with non-adaptive aggregation is not as significant. On Theta instead, we observe a completely different trend, where the I/O time remains constant. This difference in performance trend can be attributed to the presence of dedicated

I/O nodes on Mira as opposed to shared I/O nodes on Theta. With adaptive aggregation scheme, the aggregators are uniformly spread across the rank space, thus evenly utilizing all available I/O nodes. Also, given that the total number of particles are same across all configurations and placement of aggregators do not have significant impact on Theta, we observe almost constant performance on Theta (green line). For highly localized domain distributions (12.5%) our aggregation scheme starts to saturates in performance, unable to load balance effectively. In the future we plan to explore this direction of work in more detail.

7 CONCLUSIONS AND FUTURE WORK

General purpose I/O libraries fails to take into account the requirements of post-processing analysis and visualization. In this paper, we introduced an I/O system specifically designed for particles, which writes data in a format that preserves spatial locality, while simultaneously supporting spatial read queries at different LOD. The I/O system is suitable for tasks such as distributed rendering, stencil operations, topological analysis, etc. as they can take advantage of spatial locality in the data format. In our experimental evaluation, we achieve maximum throughput on Theta, and 50% of the maximum throughput on Mira using only 1/3 of the machine. These performance results demonstrate that our techniques can perform scalable, spatially aware two-phase I/O, while providing performance portability on different machines.

However, our current work is not without limitations. Although our existing LOD structure is well-suited to multi-resolution spatial queries, it may not be the best choice for other styles of range or filter queries used in analysis tasks. Integrating more advanced or adaptable structures would make for valuable future work. Similarly, our current adaptive aggregation phase is applicable to only a subset of uneven distributions found in practice, and extending it to support a wider range of configurations would be useful, to support a broader range of simulations. This could be done by creating an adaptive grid on the fly, which can re-balance the grid partition size and placement based on the particle distribution.

ACKNOWLEDGEMENTS

This work is supported in part by the Intel Graphics and Visualization Institutes of XeLLENCE program, NSF:CGV Award: 1314896, NSF:IIP Award: 1602127, NSF:ACI Award:1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375 and NSF:OAC Award: 1842042. This work used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] "HDF5 Home Page," http://www.hdfgroup.org/HDF5/.
[2] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka, "Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.
[3] J. Li, W. keng 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 *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Nov 2003, pp. 39–39.
[4] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, Metadata Rich IO Methods for Portable High Performance IO," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium On*, 2009.
[5] S. Kumar, J. Edwards, P.-T. Bremer, A. Knoll, C. Christensen, V. Vishwanath, P. Carns, J. A. Schmidt, and V. Pascucci, "Efficient I/O and storage of adaptive-resolution data," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014.
[6] 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 *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE, 2011.
[7] M. Berzins, J. Beckvermit, T. Harman, A. Bezdjian, A. Humphrey, Q. Meng, J. Schmidt, and C. Wight, "Extending the Uintah Framework through the Petascale Modeling of Detonation in Arrays of High Explosive Devices," *SIAM Journal on Scientific Computing*, 2016.
[8] S. W. Skillman, M. S. Warren, M. J. Turk, R. H. Wechsler, D. E. Holz, and P. M. Sutter, "Dark Sky Simulations: Early Data Release," *arXiv:1407.2600*, 2014.
[9] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, and others, "HACC: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, 2016.
[10] S. Byna, M. Chaarawi, Q. Koziol, J. Mainzer, and F. Willmore, "Tuning HDF5 subfiling performance on parallel file systems," in *Cray User Group*, 2017.
[11] S. Byna, J. Chou, O. Rubel, H. Karimabadi, W. S. Daughter, V. Roytershteyn, E. W. Bethel, M. Howison, K.-J. Hsu, K.-W. Lin, and others, "Parallel I/O, analysis, and visualization of a trillion particle simulation," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference For*, 2012.
[12] M. Howison, A. Adelmann, E. W. Bethel, A. Gsell, B. Oswald, and others, "H5hut: A high-performance I/O library for particle-based simulations," in *Cluster Computing Workshops and Posters, 2010 IEEE International Conference On*, 2010.
[13] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective I/O in ROMIO," in *Seventh Symposium on the Frontiers of Massively Parallel Computation*, 1999.
[14] J. M. del Rosario, R. Bordawekar, and A. Choudhary, "Improved parallel I/O via a two-phase run-time access strategy," *SIGARCH Comput. Archit. News*, 1993.
[15] R. Fraedrich, J. Schneider, and R. Westermann, "Exploring the Millennium Run–Scalable Rendering of Large-Scale Cosmological Datasets," *IEEE Transactions on Visualization and Computer Graphics*, no. 6, 2009.
[16] K. Schatz, C. Muller, M. Krone, J. Schneider, G. Reina, and T. Ertl, "Interactive Visual Exploration of a Trillion Particles," in *LDAV*, 2016.
[17] F. Reichl, M. Treib, and R. Westermann, "Visualization of big SPH simulations via compressed octree grids," in *IEEE International Conference On Big Data*, 2013.
[18] S. Rizzi, M. Hereld, J. Insley, M. E. Papka, T. Uram, and V. Vishwanath, "Large-Scale Parallel Visualization of Particle-Based Simulations using Point Sprites and Level-Of-Detail," 2015.
[19] M. Le Muzic, J. Parulek, A.-K. Stavrum, and I. Viola, "Illustrative visualization of molecular reactions using omniscient intelligence and passive agents," in *Computer Graphics Forum*, 2014.
[20] C. Baldwin, G. Abdulla, and T. Critchlow, "Multi-resolution modeling of large scale scientific simulation data," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03. ACM, 2003.
[21] V. Pascucci and R. J. Frank, "Global static indexing for real-time exploration of very large regular grids," in *ACM/IEEE Conference on High Performance Networking and Computing*, 2001.
[22] Y. Tian, S. Klasky, W. Yu, B. Wang, H. Abbasi, N. Podhorszki, and R. Grout, "Dynam: Dynamic multiresolution data representation for large-scale scientific analysis," in *Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on*. IEEE, 2013.
[23] I. Wald, A. Knoll, G. P. Johnson, W. Usher, V. Pascucci, and M. E. Papka, "CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees," in *2015 IEEE Scientific Visualization Conference (SciVis)*, 2015.
[24] ANL. (2019) Theta home page. [Online]. Available: https://www.alcf.anl.gov/theta
[25] (2017) Mira home page. [Online]. Available: https://www.alcf.anl.gov/mira
[26] Sandia National Laboratories, "LAMMPS Molecular Dynamics Simulator."
[27] S. Kumar, D. Hoang, S. Petruzza, J. Edwards, and V. Pascucci, "Reducing network congestion and synchronization overhead during aggregation of hierarchical data," in *2017 IEEE 24th International Conference on High Performance Computing*.
[28] T. Saad and J. C. Sutherland, "Wasatch: An architecture-proof multiphysics development environment using a domain specific language and graph theory," *Journal of Computational Science*, vol. 17, pp. 639 – 646, 2016, recent Advances in Parallel Techniques for Scientific Computing.
[29] H. Shan, K. Antypas, and J. Shalf, "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Nov 2008.
[30] h5perf home page. [Online]. Available: https://support.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Perf
[31] P. Coffman, F. Tessier, P. Malakar, and G. Brown, "Parallel I/O on Theta with Best Practices," Talk at ALCF Simulation, Data, and Learning Workshop, 2018.
[32] D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, B. Steinmacher-Burow, A. Choudhury, Y. Sabharwal, S. Singhal, and J. J. Parker, "Looking Under the Hood of the IBM Blue Gene/Q Network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12, 2012.