



# Scalable Data Management of the Uintah Simulation Framework for Next-Generation Engineering Problems with Radiation

Sidharth Kumar<sup>1</sup>(✉), Alan Humphrey<sup>1</sup>, Will Usher<sup>1</sup>, Steve Petruzza<sup>1</sup>, Brad Peterson<sup>1</sup>, John A. Schmidt<sup>1</sup>, Derek Harris<sup>2</sup>, Ben Isaac<sup>2</sup>, Jeremy Thornock<sup>2</sup>, Todd Harman<sup>2</sup>, Valerio Pascucci<sup>1</sup>, and Martin Berzins<sup>1</sup>

<sup>1</sup> SCI Institute, University of Utah, Salt Lake City, UT, USA  
{sidharth, ahumphrey}@sci.utah.edu

<sup>2</sup> Institute for Clean and Secure Energy, Salt Lake City, UT, USA

**Abstract.** The need to scale next-generation industrial engineering problems to the largest computational platforms presents unique challenges. This paper focuses on data management related problems faced by the Uintah simulation framework at a production scale of 260K processes. Uintah provides a highly scalable asynchronous many-task runtime system, which in this work is used for the modeling of a 1000 megawatt electric (MWe) ultra-supercritical (USC) coal boiler. At 260K processes, we faced both parallel I/O and visualization related challenges, e.g., the default file-per-process I/O approach of Uintah did not scale on Mira. In this paper we present a simple to implement, restructuring based parallel I/O technique. We impose a restructuring step that alters the distribution of data among processes. The goal is to distribute the dataset such that each process holds a larger chunk of data, which is then written to a file independently. This approach finds a middle ground between two of the most common parallel I/O schemes—file per process I/O and shared file I/O—in terms of both the total number of generated files, and the extent of communication involved during the data aggregation phase. To address scalability issues when visualizing the simulation data, we developed a lightweight renderer using OSPRay, which allows scientists to visualize the data interactively at high quality and make production movies. Finally, this work presents a highly efficient and scalable radiation model based on the sweeping method, which significantly outperforms previous approaches in Uintah, like discrete ordinates. The integrated approach allowed the USC boiler problem to run on 260K CPU cores on Mira.

## 1 Introduction

The exponential growth in High performance computing (HPC) over the past 20 years has fueled a wave of scientific insights and discoveries, many of which

---

S. Kumar and A. Humphrey—Authors contributed equally.

© The Author(s) 2018

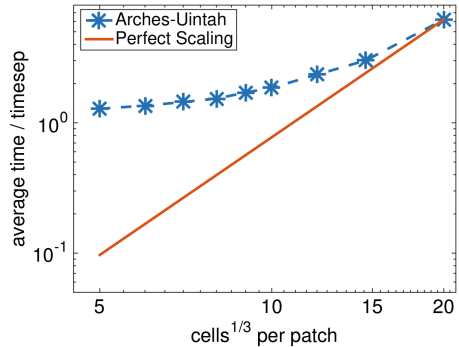
R. Yokota and W. Wu (Eds.): SCFA 2018, LNCS 10776, pp. 219–240, 2018.

[https://doi.org/10.1007/978-3-319-69953-0\\_13](https://doi.org/10.1007/978-3-319-69953-0_13)

would not be possible without the integration of HPC capabilities. This trend is continuing, for example, the DOE Exascale Computing Project [18] lists 25 major application focus areas [20] in energy, science, and national security. The primary challenge in moving codes to new architectures at exascale is that, although present codes may have good scaling characteristics on some current architectures, those codes may likely have components that are not suited to the high level of parallelism on these new computer architectures, or to the complexity of real-world applications at exascale. One of the major challenges faced by modern scalable scientific codes is with regard to data management. As the gap between computing power and available disk bandwidth continues to grow, the cost of parallel I/O becomes an important concern, especially for simulations at the largest scales. Large-scale simulation I/O can be roughly split into two use cases: checkpoint restarts in which the entire state of a simulation must be preserved exactly, and analysis dumps in which a subset of information is saved. Both checkpointing and analysis dumps are important, yet due to poor I/O scaling and little available disk bandwidth, the trend of large-scale simulation runs is to save fewer and fewer results. This not only increases the cost of faults, since checkpoints are saved less frequently, but ultimately may affect the scientific integrity of the analysis, due to the reduced temporal sampling of the simulation. This paper presents a simple to implement method to enable parallel I/O, which we demonstrate to efficiently scale up to 260K processes.

For most applications, the layout of data distributed across compute cores does not translate to efficient network and storage access pattern for I/O. Consequently, performing naive I/O leads to significant underutilization of the system. For instance, the patch or block size of simulations is typically on the order of  $12^3$  to  $20^3$  voxels (cells), mainly because a scientist typically works under a restricted compute budget, and smaller patch sizes lead to faster execution of individual computational timesteps (see Fig. 1), which is critical in completion of the entire simulation. Small patch sizes do not bode well for parallel I/O, with either file-per-process I/O or shared file I/O.

We find a middle ground by introducing a restructuring-based parallel I/O technique. We virtually regrid the data by imposing a restructuring phase that alters the distribution of data among processes in a way such that only a few processes end up holding larger patches/blocks, which are then written to a file independently. The efficacy and scalability of this approach is shown in Sect. 3.



**Fig. 1.** Time taken for execution of a timestep for different patch sizes. Execution time starts to increase rapidly after a patch size of  $12^3$ .

In order to gain scientific insight from such large-scale simulations, the visualization software used must also scale well to large core counts and datasets, introducing additional challenges in performing scientific simulations at scale for domain scientists. To address I/O challenges on the read side of the scientific pipeline, we also use our scalable parallel I/O library in combination with the ray tracing library OSPRay [24] to create a lightweight remote viewer and movie rendering tool for visualization of such large-scale data (Sect. 4).

Finally, we introduce a new, efficient radiation solve method into Uintah based on spatial transport sweeps [2, 4]. The radiation calculation is central to the commercial 1000 megawatt electric (MWe) ultra-supercritical (USC) coal boiler being simulated in this work, as radiation is the dominant mode of heat transfer within the boiler itself. To improve parallelism within these spatial sweeps, the computation is split into multiple stages, which then expose spatial dependencies to the Uintah task scheduler. Using the provided information about the stage's dependencies, the scheduler can efficiently distribute the computation, increasing utilization. For the target boiler problem discussed in this paper, we find this method up to 10× faster than previous reverse Monte Carlo ray tracing methods (Sect. 5) due to this increased utilization.

This work demonstrates the efficacy of our approach by adapting the Uintah computational framework [8], a highly scalable asynchronous many-task (AMT) [7] runtime system, to use our I/O system and spatial transport sweeps within a large-eddy simulation (LES). This work is aimed at predicting the performance of a commercial 1000 MWe USC coal boiler, and has been considered as an ideal exascale candidate given that the spatial and temporal resolution requirements on physical grounds give rise to problems between 50 to 1000 times larger than those we can solve today.

The principal contributions of this paper are:

1. A restructuring-based parallel I/O scheme.
2. A data parallel visualization system using OSPRay.
3. A faster approach to radiation using a spatial transport sweeps method.

## 2 Background

### 2.1 Uintah Simulation Framework

Uintah [22] is a software framework consisting of a set of parallel software components and libraries that facilitate the solution of partial differential equations on structured adaptive mesh refinement (AMR) grids. Uintah currently contains four main simulation components: (1) the multi-material ICE code for both low- and high-speed compressible flows; (2) the multi-material, particle-based code MPM for structural mechanics; (3) the combined fluid-structure interaction (FSI) algorithm MPM-ICE; and (4) the Arches turbulent reacting CFD component that was designed for simulating turbulent reacting flows with participating media radiation. Separate from these components is an AMT runtime, considered as a possible leading alternative to mitigate exascale challenges at the

runtime system-level, which shelters the application developer from the complexities introduced by future architectures [7]. Uintah’s clear separation between the application layer and the underlying runtime system both keeps the application developer insulated from complexities of the underlying parallelism Uintah provides, and makes it possible to achieve great increases in scalability through changes to the runtime system that executes the taskgraph, without requiring changes to the applications themselves [17].

Uintah decomposes the computational domain into a structured grid of rectangular cuboid cells. The basic unit of a Uintah simulation’s Cartesian mesh (composed of cells) is termed a *patch*, and simulation variables that reside in Uintah’s patches are termed *grid* or *particle* variables. The Uintah runtime system manages the complexity of inter-nodal data dependencies, node-level parallelism, data movement between the CPU and GPU, and ultimately task scheduling and execution that make up a computational algorithm [19], including I/O tasks. The core idea is to use a directed acyclic graph (DAG) representation of the computation to schedule work, as opposed to, say, a bulk synchronous approach in which blocks of communication follow blocks of computation. This graph-based approach allows tasks to execute in a manner that efficiently overlaps communication and computation, and includes out-of-order execution of tasks (with respect to a topological sort) where possible. Using this task-based approach also allows for improved load balancing, as only nodes need to be considered, not individual cores [8].

## 2.2 Related Work

Many high-level I/O libraries have been developed to help structure the large volumes of data produced by scientific simulations, such as HDF5 and PnetCDF. The hierarchical data format (HDF5) [10] allows developers to express data models organized hierarchically. PnetCDF [14] is a parallel implementation of the network common data form (netCDF), which includes a format optimized for dense, regular datasets. Both HDF5 and PnetCDF are implemented using MPI and both leverage MPI-IO collective I/O operations for data aggregation. In practice, shared file I/O often does not scale well because of the global communication necessary to write to a single file. ADIOS [15] is another popular library used to manage parallel I/O for scientific applications.

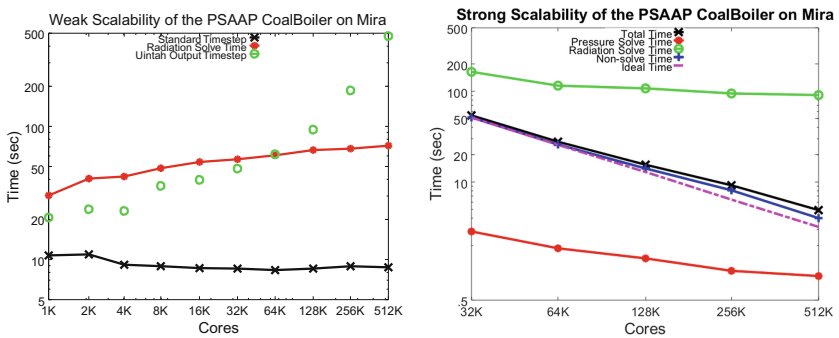
On the visualization front, VisIt [9] and ParaView [3] are popular distributed visualization and analysis applications. They are typically executed in parallel, coordinating visualization and analysis tasks for massive simulation data. The data are typically loaded at full resolution, requiring large amounts of system memory. Moreover, both can be used for remote visualization, where a remote server (or servers) is responsible for rendering and operating on the data and the user interacts remotely through a lightweight client. Our renderer works similarly, allowing for remote visualization and offline movie rendering. Furthermore, our viewer supports running on any CPU architecture, via OSPRay [24], which is now integrated in ParaView and being integrated into VisIt. However, in contrast to these larger visualization packages, our viewer is tuned specifically

for interactive visualization and movie rendering, allowing for better rendering performance and enabling us to take better advantage of parallel I/O for fast data loading.

The 1000 MWe USC coal boiler being modeled by Uintah in this work has thermal radiation as a dominant heat transfer mode, and involves solving the conservation of energy equation and radiative heat transfer equation (RTE) simultaneously. This radiation calculation, in which the radiative-flux divergence at each cell of the discretized domain is calculated, can take up to 50% of the overall CPU time per timestep [11] using the discrete ordinates method (DOM) [6], one of the standard approaches to computing radiative heat transfer. Using a reverse Monte Carlo ray tracing approach combined with a novel use of Uintah’s adaptive mesh refinement infrastructure, this calculation has been made to scale to 262K cores [11], and further adapted to run on up to 16K GPUs [12]. The spatial transport sweeps method discussed in Sect. 5 shows great promise for future large-scale simulations.

### 2.3 System Configuration

The Mira supercomputer [1] at Argonne National Laboratory is an IBM Blue Gene/Q system that enables high-performance computing with low power consumption. The Mira system has 16 cores per node, 1024 nodes per rack, and 48 racks, providing a total of 768K cores. Each node has 16 GB of RAM and the network topology is a 5D torus. There are two I/O nodes for every 128 compute nodes, with one 2 GB/s bandwidth link per I/O node. Mira uses the GPFS file system. Ranks are assigned with locality guarantees on the machine, which our I/O system can also take advantage of. Initial results of weak and strong scaling studies of the target problem on Mira are shown in Fig. 2.

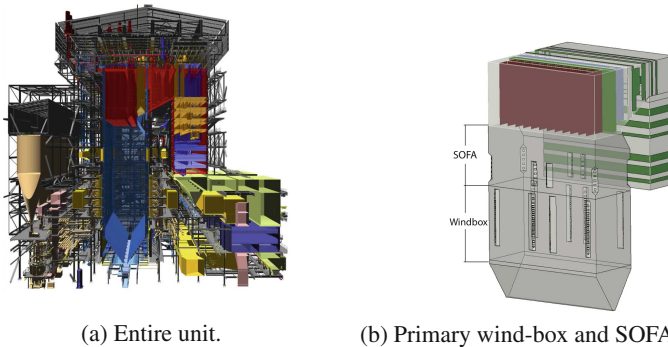


**Fig. 2.** Strong and weak scalability of the coal boiler simulation on Mira using the discrete ordinates solver. In these initial studies, we found scaling issues with the I/O and radiation solve components of Uintah that needed to be addressed for the production runs. Note the radiation solve is not executed each timestep, and not included in the total time for a timestep.

## 2.4 Target Boiler Problem

GE Power is currently building new coal-fired power plants throughout the world and evaluating new designs for these boilers. Many of these units will potentially be 1000 MWe, twin-fireball USC units (Fig. 3a). Historically, twin-fireball (or 8-corner) units became part of the GE Power product offering because of the design uncertainty in scaling 4-corner units from a lower MWe rating to a much higher MWe rating. In order to decrease risk, both from GE Power’s perspective and the customer’s, two smaller units were joined together to form a larger unit capable of producing up to 1090 MWe.

Simulation plays a key role in designing new boilers, allowing engineers to build, test, and optimize new designs at very low cost. When viewed as a large-scale computational problem, there are considerable challenges to simulating the boiler at acceptable accuracy and resolution to gain scientific insight about the design. The geometric complexity of the boiler is considerable, and presented a significant challenge for the combustion modelers. The boiler measures  $65\text{ m} \times 35\text{ m} \times 15\text{ m}$  and contains 430 separated over-fired air (SOFA) inlets (Fig. 3b), which inject pulverized coal and oxygen into the combustion chamber. Moreover, the boiler has division panels, plates, super-heaters and re-heater tubing with about 210 miles of piping walls, and tubing made of 11 metals with varying thickness. Both the 8-corner units and 4-corner units have different mixing and wall absorption characteristics that must be fully understood in order to have confidence in their respective designs. One key piece of the design that must be understood is how the SOFA inlets should be positioned and oriented, and what effect this has on the heat flux distribution throughout the boiler.

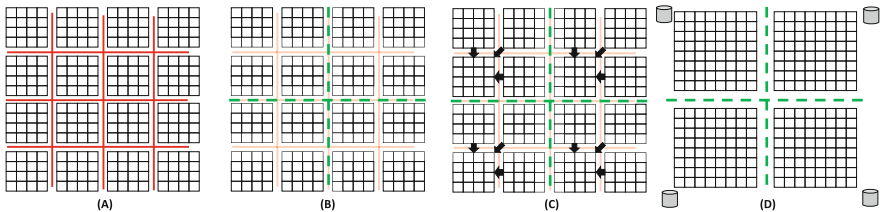


**Fig. 3.** CAD rendering of GE Power’s 1000 MWe USC two-cell pulverized coal boiler.

## 3 Restructured Parallel I/O

File-per process I/O and single shared file I/O are two of the most commonly used parallel I/O techniques. However, both methods fail to scale at high core counts. An I/O-centric view of the typical simulation pipeline is as follows: the

simulation domain is first divided into cells/elements (usually pixels or voxels), and then the elements are grouped into patches. Each patch is assigned a rank or processor number. Rank assignment is done by the simulation software using a deterministic indexing scheme (for example, row-order or Z-order). When performing file-per-process I/O, each process creates a separate file and independently writes its patch data to the file. This approach works well for relatively small numbers of cores; however, at high core counts this approach performs poorly, as the large number of files overwhelms the parallel file system. When using single shared file I/O, performance also decreases as the core count increases, as the time spent during data exchange involved in the aggregation step becomes significant, impeding scalability [5]. In this work we tackle both the communication bottleneck of the aggregation phase in single shared file I/O and the bottleneck of creating a hierarchy of files in file-per-process I/O by proposing a middleground approach through a restructuring-based parallel I/O technique. The main idea is to regrid the simulation domain in a restructuring phase.



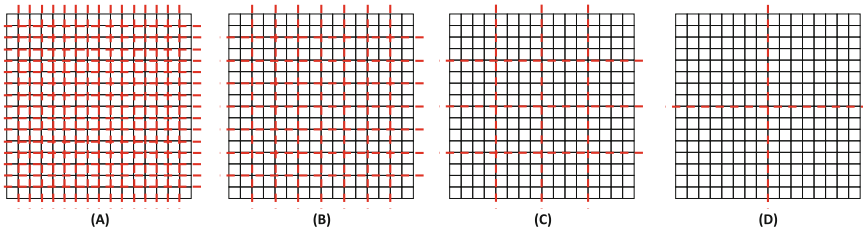
**Fig. 4.** Schematic diagram of restructuring-based parallel I/O. (A) The initial simulation patch size is  $4 \times 4$ . (B) A new grid of patch size  $8 \times 8$  is imposed. (C) The restructuring phase is executed using MPI point-to-point communication. (D) Finally, using the restructured grid, every patch is written to a separate file.

Starting from the original simulation grid (Fig. 4(A)), we begin restructuring by imposing a new grid on the simulation domain (Fig. 4(B)). The patch size of the imposed grid is larger than the initial patch size assigned by the simulation. As mentioned in Fig. 1, the patch size assigned by the simulation is on the order of  $12^3$  to  $20^3$ , while the patch size of the new restructured grid is typically twice that in each dimension. The simulation data is then restructured-based on the new grid/patch configuration (Fig. 4(C)). During the restructuring, MPI point-to-point communication is used to move data between processes [13]. Note that the communication is distributed in nature and confined to small subsets of processes, which is crucial for the scalability of the restructuring phase. At the end of the restructuring phase, we end up with large-sized patches on a subset of processes (Fig. 4(D)). Given that the new restructured patch size is always bigger than the patch size assigned by the simulation (or equal in size at worst), we end up with fewer patches held on a subset of the simulation processes. Throughout the restructuring phase the data remains in the application layout. Once the restructuring phases concludes, each processes holding the restructured

patches create a file and writes its data to the file. This scheme of parallel I/O finds a middle-ground between file-per-process-based parallel I/O and shared file I/O, both in terms of the total number of files generated and the extent of communication required during the data aggregation phase. With our approach, the total number of files generated is given by the following formula:

$$\text{number\_of\_files} = \left\lceil \frac{\text{bounds\_x}}{\text{nrst\_x}} \right\rceil \times \left\lceil \frac{\text{bounds\_y}}{\text{nrst\_y}} \right\rceil \times \left\lceil \frac{\text{bounds\_z}}{\text{nrst\_z}} \right\rceil$$

Based on the restructuring box size ( $\text{nrst\_x} \times \text{nrst\_y} \times \text{nrst\_z}$ ), we can have a range of total number of outputted files. The number of files will be equal to the number of processes (i.e., file-per-process I/O) when the restructuring patch size is equal to the simulation patch size. The number of files will be one (i.e., shared-file I/O) when the restructuring patch size is equal to the entire simulation domain ( $\text{bounds\_x} \times \text{bounds\_y} \times \text{bounds\_z}$ ). For most practical scenarios, the latter situation is not feasible due to limitations on the available memory on a single core. With file-per-process I/O, there is no communication among processes, whereas with collective I/O associated with shared file I/O, the communication is global in nature. With the restructuring approach, all communication is localized. The restructuring approach not only helps tune the total number of outputted files, but also increases the file I/O burst size, which in general is a requirement to obtain high I/O bandwidth. Our approach exhibits good scaling characteristics, as shown in the following two sections.



**Fig. 5.** Three variations of restructuring-based parallel I/O. (A) No restructuring, each patch is held by a process and is written out separately to a file. (B) Restructuring phase with new patches containing  $2^2$  simulation patches, creating  $4\times$  fewer files. (C) New patch size of  $4^2$  simulation patches, creating  $16\times$  fewer files (D) New patch size of  $8^2$  simulation patches, creating  $64\times$  fewer files. Communication is limited to groups of (B) 4, (C) 16, and (D) 64 processes.

### 3.1 Parameter Study

The tunable parameter in our proposed I/O framework that has the greatest impact on performance is the patch size used for restructuring. The patch size affects both the degree of network traffic and the total number of outputted files. To understand the impact of the parameter, we wrote a micro-benchmark

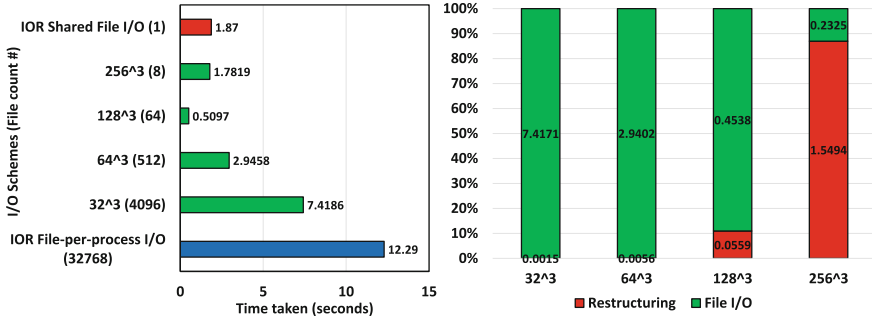


to write out a 3D volume. In our evaluations on the Mira supercomputer, we kept the number of processes fixed at 32768. Each process wrote a  $16^3$  sub-volume of double precision floating point data to generate a total volume of 1 gigabyte ( $512^3$ ). We used four restructuring box sizes –  $32^3$ ,  $64^3$ ,  $128^3$  and  $256^3$ ; the number of files generated, respectively, varied as  $4096$  ( $512^3/32^3$ ),  $512$  ( $512^3/64^3$ ),  $64$  ( $512^3/128^3$ ),  $8$  ( $512^3/256^3$ ). The number of processes involved in communication during the restructuring phase increases with the box size. For example, with a restructuring box of size  $32^3$ , communication is limited to groups of 8 processes ( $32^3/16^3$ ), whereas with a restructuring box of size  $256^3$ , communication takes place with a group of 4096 processes. See Fig. 5 for an example. In order to provide a baseline for the results obtained, we also ran IOR benchmarks. IOR is a general-purpose parallel I/O benchmark [23] that we configured to perform both file-per-process as well as shared file I/O. For shared file I/O, all the processes wrote to a single file using MPI collective I/O. The results can be seen in Fig. 6. The file-per-process I/O performs the worst and this is because the underlying GPFS [21] parallel file system of Mira is not adept at handling large numbers of files. Although we did not run any benchmarks of the Lustre PFS [16], it is more suited to handling large numbers of files, especially at low core counts. This is mainly because GPFS is a block-based distributed filesystems where the metadata server controls all the block allocation, whereas the Lustre filesystem has a separate metadata server for pathname and permission checks. Furthermore, the Lustre metadata server is not involved in any file I/O operations, which avoids I/O scalability bottlenecks on the metadata server. The constraint on the number of files makes our approach highly suitable for GPFS file systems. Note that both file systems start to get saturated with file per-process I/O at high core counts.

The performance of restructuring-based parallel I/O improves with larger box sizes, reaching peak performance with a box size of  $128^3$ . At that patch size, the restructuring approach achieves a  $3.7\times$  improvement over the IOR benchmark’s shared file I/O approach (using MPI collective I/O) because our approach’s *aggregation* phase is localized in nature, involving only small groups of processes, as opposed to MPI collective I/O’s underlying global communication. However, we observe performance degradation at a restructuring box size of  $256^3$ . The reason for this degradation can be understood by looking at a time breakdown between the restructuring (communication) phase and the file I/O phase (Fig. 6). As can be seen, communication time (red) increases with larger restructuring boxes. Although the file I/O time continues to decrease with increasing box size, the restructuring time begins to dominate at  $256^3$ , and as a result overall performance suffers. We believe our design is flexible enough to be tuned to generate small numbers of large shared files or a large number of files, depending on which is optimal for the target system.

### 3.2 Production Run Weak Scaling Results

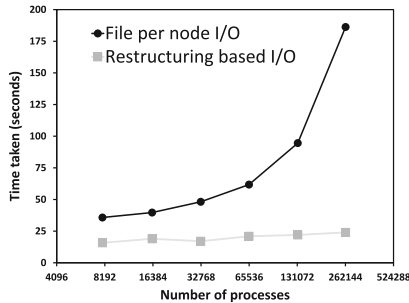
With Uintah’s default I/O subsystem, every node writes data for all its cores into a separate file. Therefore, on Mira there is one file for every 16 cores (16 cores



**Fig. 6.** (Left) Performance of restructuring-based parallel I/O with varying box sizes. (Right) Time breakdown between restructuring (communication) and file I/O for different box sizes. (Color figure online)

per node). This form of I/O is an extension to the file-per process style of I/O commonly adopted by many simulations. An XML-based meta-data file is also associated with every data file that stores type, extents, bounds, and other relevant information for each of the different fields. For relatively small core counts this I/O approach works well. However, I/O performance degrades significantly for simulations with several hundreds of thousands of patches/processors. The cost of both reads and writes for large numbers of small files becomes untenable.

We extended the Uintah simulation framework to use the restructuring-based I/O scheme, and evaluated the weak scaling performance of the I/O system when writing data for a representative Uintah simulation on Mira. In each run, Uintah wrote out 20 timesteps consisting of 72 fields (grid variables). The patch size for the simulation was  $12^3$ . The number of cores was varied from 7,920 to 262,890. Looking at the performance results in Fig. 7, our I/O system scales well for all core counts and performs better than the original Uintah UDA I/O system. The restructuring-based I/O system demonstrates almost linear scaling up to 262,890 cores whereas the performance of file-per-node I/O starts to decline after



**Fig. 7.** Weak scaling results of restructuring-based parallel I/O compared to Uintah’s file per node I/O approach. Our I/O system outperforms Uintah’s default I/O at all core counts, attaining 10× performance improvement at 260K cores.

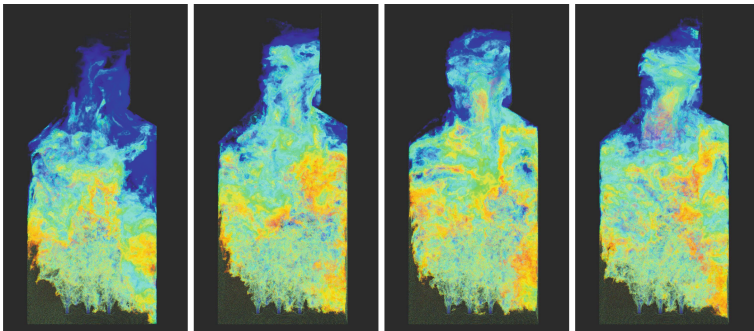
16,200 cores. At 262,890 cores, our I/O system achieves an approximate speed-up of  $10\times$  over Uintah’s default file-per-node I/O.

The restructuring-based I/O system was then used in production boiler simulations, carried out at 260K cores on Mira. Due to the improved performance of the I/O system, scientists were able to save data at a much higher temporal frequency. In terms of outputs, close to 200 terabytes of data was written which, using our new restructuring I/O strategy, required only 2% of the entire simulation time. If the simulation were run using the Uintah’s default file-per-process node output format, nearly 50% of the time of the computation would be spent on I/O, reducing the number of timesteps that could be saved, or increasing the total computation time significantly.

## 4 Scalable Visualization with OSPRay

When trying to visualize the data produced on Mira using the Cooley visualization cluster at ANL, VisIt rendered at interactive framerates for smaller datasets; however, when trying to visualize the recent large simulations using all the cores on each node would consume too much memory, resulting in crashes or significantly reduced performance due to swapping. To address these issues and allow for quick, interactive visualization and high-quality offline movie rendering, we wrote a lightweight renderer using OSPRay [24] which uses the restructuring-based parallel I/O to read the data. OSPRay is a CPU-based open-source ray tracing library for scientific visualization, and includes high-quality and high-performance volume rendering, along with support for rendering distributed data with MPI.

OSPRay includes support for two modes of MPI-parallel rendering: an offload mode, where data is replicated across nodes, and subregions of the image are distributed; and a distributed mode, where different ranks make OSPRay API calls independently to setup their local data, and then work collectively to render



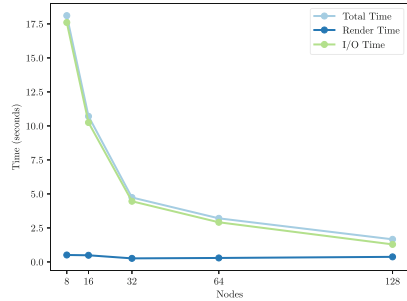
**Fig. 8.** Frames from the movie showing the  $O^2$  field over time. Using restructuring-based parallel I/O backend and OSPRay we were able to render an animation of the full 1030 timesteps in two hours using 128 KNL nodes on Theta.

the distributed data using sort-last compositing [24]. To leverage the benefits of the restructuring-based parallel I/O in the viewer, we implemented our renderer using the distributed mode of OSPRay, with each rank responsible for loading and rendering a subregion of the dataset. To properly composite the distributed data OSPRay requires the application to specify a set of regions on each rank, which bound the data owned by that rank. In our case this is trivially the bounds of the single subregion owned by the rank. The renderer supports two usage modes, allowing for interactive remote visualization and offline movie rendering for creating production animations of the evolution of the boiler state.

The interactive viewer runs a set of render worker processes on the compute nodes, with one per node as OSPRay uses threads for on-node parallelism. The user then connects over the network with a remote client and receives back rendered JPG images, while sending back over the network camera and transfer function changes to interact with the dataset. To decouple the interface from network latency effects and the rendering framerate, we send and receive to the render workers asynchronously, and always take the latest frame and send the latest application state. With this application, users can explore the different timesteps of the simulation and different fields of data interactively on their laptop, with the rendering itself performed on Theta or Cooley. When rendering on 16 nodes of Theta with a  $1080 \times 1920$  framebuffer (oriented to match the vertical layout of the boiler), the viewer was able to render at 11 FPS, allowing for interactive exploration.

The offline movie renderer is run as a batch application and will render the data using a preset camera path. The movies produced allow for viewing the evolution of the boiler state smoothly over time, as the timesteps can be played through at a constant rate, instead of waiting for new timesteps or fields to load. A subset of frames from this animation is shown in Fig. 8, which was rendered using 128 nodes on Theta. The majority of the time spent in the movie rendering is in loading the data, which scales well with the presented I/O scheme (Fig. 9). The animation is rendered at  $1080 \times 1920$  with a high number of samples per pixel to improve image quality.

While our lightweight viewer is valuable for visual exploration, it is missing the large range of additional analysis tools provided by production visualization and analysis packages like VisIt. To this end, we are working on integrating OSPRay into VisIt as a rendering backend, enabling scalable interactive visualization for end users.



**Fig. 9.** Strong scaling of movie rendering on Theta.

## 5 Radiation Modeling: Spatial Transport Sweeps

The heat transfer problem arising from the clean coal boilers being modeled by the Uintah framework has thermal radiation as a dominant heat transfer mode and involves solving the conservation of energy equation and radiative heat transfer equation (RTE) simultaneously [11]. Scalable modeling of radiation is currently one of the most challenging problems in large-scale simulations, due to the global, all-to-all nature of radiation [17], potentially affecting all regions of the domain simultaneously at a single instance in time. To simulate thermal transport, two fundamental approaches exist: random walk simulations and finite element/finite volume simulations, e.g., discrete ordinates method (DOM) [6], which involves solving many large systems of equations. Additionally, the algorithms used for radiation can be used recursively with different spatial orientations and different spectral properties, requiring hundreds to thousands of global, sparse linear solves. Consequently, the speed, accuracy, and limitations of the method must be appropriate for a given application.

Uintah currently supports two fundamentally different approaches to solving the radiation transport equation (RTE) to predict heat flux and its divergence (operator) in these domains. We provide an overview of these supported approaches within Uintah and introduce a third approach, illustrating its performance and scaling with results up to 128K CPU cores on Mira for a radiation benchmark problem.

### 5.1 Solving the Radiation Transport Equation

The heat flux divergence can be computed using:

$$\nabla Q = 4 * \pi * S - \int_{4\pi} I_{\Omega} d\Omega, \quad (1)$$

where  $S$  is the local source term for radiative intensity, and  $I_{\Omega}$  is computed using the RTE for grey non-scattering media requiring a global solve via:

$$\frac{dI_{\Omega}}{ds} = k * (S - I_{\Omega}) \quad (2)$$

Here,  $s$  is the 1-D spatial coordinate oriented in the direction in which intensity  $I_{\Omega}$  is being followed, and  $k$  is the absorption or attenuation coefficient. The lack of time in the RTE implies instantaneous transport of the intensity, appropriate for most applications. The methods for solving the RTE discussed here aim to solve for  $I_{\Omega}$  using Eq. 2, which can then be integrated to compute the radiative flux and divergence.

### 5.2 Discrete Ordinates

The discrete ordinates method [6], used in our Mira simulations, solves the RTE by discretizing the left-hand side of Eq. 2, which results in a 4 or a 7-point stencil,

depending on the order of the first derivative. Instabilities arise when using the higher order method, so often the 7-point stencil is avoided, or a combination of the two stencils is used. The 4-point stencil results in numerical diffusion that impacts the fidelity of the solve, but for low ordinate counts can improve solution accuracy. As shown by Fig. 2, this method has been demonstrated to scale, but it is computationally expensive, due to the numerous global sparse linear solves. In the case shown, as many as 30–40 backsolves were required per radiation step, with up to an order of magnitude more solves required in other cases. It should be noted that, due to their computational cost, the radiation solves are computed roughly once every 10 timesteps, as the radiation solution does not change quickly enough to warrant a more frequent radiation calculation.

### 5.3 Reverse Monte Carlo Ray Tracing

Reverse Monte Carlo Ray Tracing (RMCRT) [11] has been implemented on both CPUs and GPUs [11, 12], and is a method for solving Eq. 2 by tracing radiation rays from one cell to the next, as described in detail in [11, 12]. Reverse Monte Carlo (as opposed to forward) is desirable because rays are then independent of all other ray tracing processes, and are trivially parallel. RMCRT exhibits high accuracy with sufficient ray sampling and is can easily simulate various scattering effects. However, RMCRT can have a very large memory footprint when geometry is replicated on each node to facilitate local ray tracing. To reduce this memory footprint and the required communication, RMCRT leverages the AMR support within Uintah to use a fine mesh locally and a coarse mesh distally. RMCRT uses Monte Carlo processes to model scattering physics and is a direct method, outperforming discrete-ordinates with significant scattering. Numerical diffusion is non-existent for RMCRT. When computation resources are abundant and the solid-angle [11] can be well resolved, RMCRT is preferred to discrete-ordinates, where numerical diffusion hurts accuracy. This method has been made to scale up to 256K CPU cores and up to 16K GPUs within the Uintah framework [11, 12, 19], and is used for the large-scale, GPU-based boiler simulations, as detailed in [19].

### 5.4 Spatial Transport Sweeps Method

The Uintah infrastructure allows for solving for the intensities with a 4-point stencil known as *spatial transport sweeps*, a *sweeping method* [2, 4], or simply *sweeps*. It is this method that we cover in detail in this work.

Sweeps is a lightweight spatially serial algorithm in which spatial dependencies dictate the speed of the algorithm. These dependencies impose serialized inter-nodal communication requirements, and account for the bulk of the algorithm's cost. While the sweeping method is inherently serial, it can be parallelized over many ordinate directions and spectral frequencies. The radiation sweeping mechanism uses the older A-matrix construction from the linear solve, and performs recursive back substitution on the A-matrix to solve for the intensities. This process is done in stages for each intensity and phase, both of which

are defined below. Although this staging process is serialized by the reliance of corner-to-corner dependencies, it can show good performance when sweeping a large quantity of independent solves. For a non-scattering medium, the angular and spectral intensities are all independent of each other, allowing for parallelization of the solve.

On large, distributed memory systems, the intensities are stored on multiple compute nodes, making communication between them expensive and inefficient. To address this problem, one processor (or node) needs to operate only on intensities that have satisfied their spatial dependency. The method shown here is based on the algorithm for a simple rectangular domain; however, it further supports identification of these dependencies for complex domains with non-rectangular shapes. To most easily convey the methodology used, we start by describing the algorithm on a rectangular domain.

Consider a domain with  $3 \times 3$  sub-units. Within Uintah, these sub-units are referred to as *patches*. A diagram showing how these patches are divided is shown in Fig. 10. The number labeling each patch (Fig. 10a) designates the phase in which a sweep is relevant for a single intensity, from the  $x + y + z + octant$ , with a single wave number. Note that these phases are defined as:

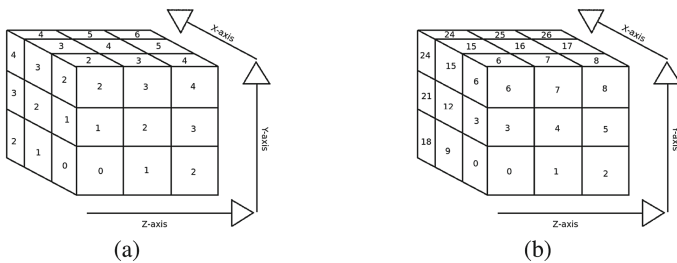
$$P = x_i + y_i + z_i \tag{3}$$

where  $x_i, y_i,$  and  $z_i$  are the patch indices in the x, y, and z directions. The patch indices are defined as the number of patches away from the origin patch. Hence, the total number of phases required to complete a single complete full-domain sweep is:

$$P_{max} = x_{max} + y_{max} + z_{max} \tag{4}$$

where  $x_{max}, y_{max}, z_{max}$  are the maximum. Numbers designate the designated phase indices of the patches within the domain. We determine the patch indices using the sub-domain with the patch ID provided by Uintah.

Uintah numbers its patches in the order of  $z, y, x$  (Fig. 10b). From this we can determine the point in space in which the sweep is currently located using modulo operators, the patch dimensions, and the patch ID. The patch index is then converted to the patch indices  $x_i, y_i, z_i$  for each patch. Using the Uintah



**Fig. 10.** A rectangular domain divided into 27 sub-domains, labeled by the designated phase (a) and Uintah patch ID (b).

task scheduler, we can indicate to a task what this phase is. This process is more complicated when conducting sweeps with multiple intensity directions. First, consider additional intensities that are in the  $x^+, y^+, z^+$  directions. To keep as many processors busy as possible in the computation, we create stages.

A stage  $S$  is defined as  $S = I + P$ , where  $I$  is the intensity index relevant to a single octant. We know the maximum number of stages via the equation  $S_{max} = I_{max} + P_{max}$ . Now we have an algorithm that describes the sweeping in a single direction, for intensities of the same octant: next, we will discuss how to extend this to all octants. The phase equation for the  $x^-, y^-, z^-$  octant results in:

$$P = x_{max} - x_i + y_{max} - y_i + z_{max} - z_i \quad (5)$$

Hence, a total of eight phase equations are possible, depending on the combination of directions. We discuss two equations in detail in this paper. The task designates the stage and intensity, and then computes a function mapping its patch ID to its spatial patch index using a series of modulus. If the patch and intensity are relevant to the local processor, then it executes, otherwise it exits the task.

**Spatial Scheduling-Supporting Sweeps Within Uintah.** In order to spatially schedule the sweeping tasks, a Uintah patch subset must be identified during the Uintah task-scheduling phase. It is convenient to use Eq. 5 to accomplish this. Iterating over the phase  $P$  and two patch indices  $y_i$  and  $z_i$  allows us to collect the relevant patches to a sweeping phase  $P$ , which results in  $P_{max}$  patch subsets per octant. These patch subsets can be reused for each intensity solve. The patch subsets are used in the Uintah task-requires call to the infrastructure that manages ghost cells, which greatly reduces communication costs. The sweep is propagated across the domain by having one independent task per stage. To propagate information from patch to patch within Uintah, a requires modifies dependency chain is created, where the requires is conditioned on a patch subset only relevant to the patches on which the sweep is occurring. The patch subset is defined as all patches with the same phase number  $P$ , as shown in Fig. 10a.

## 5.5 Sweeps: Scaling and Performance Results

Table 1 illustrates the performance and weak scaling within Uintah of the sweeping method for radiation transport on a benchmark radiation problem, run on Mira up to 128K CPU cores. Note this method is experimental and although DOM was used for the radiation solve in the full-scale Mira boiler simulations, the sweeping method introduced in this work shows great promise for radiation calculations within future boiler simulations.

For the target boiler problems in question, the use of sweeps reduces radiation solve times by a factor of 10 relative to RMCRT and linear solve methods. The differences in the solution times between the linear solve (DOM) in Table 1 and Fig. 2 are attributed to differences between the benchmark calculation and the target boiler simulation. Within the target boiler (CFD) simulation, the previous



**Table 1.** Weak scaling results on the model radiation problem.

Cores	16	256	4056	65536	131072	262144
Time DO	91	189	399	959	1200	1462
DO iter.	90	180	400	900	1300	1500
Sweeps time	1.9	3.4	4.4	9.09	13.9	

solve can also be used as an initial guess, thus accelerating convergence and making it possible for DOM to use as few as 30–40 iterations as compared to the much larger number of backsolves shown in Table 1. In contrast to a static problem, no initial guess is available, and significantly more iterations are used with DOM than is the case in a full boiler simulation (as shown in Table 1), in which as many as 1500 iterations are used. However, we note that for this problem each DOM iteration takes about a second. Hence, the best that DOM could achieve would be about 40s, even if a good initial guess is available. In this way sweeps outperforms both the actual observed cost and the optimistic estimated cost of DOM with its linear solve using 40 iterations by a factor of between 4 and 10. However, the sweeping algorithm has not currently scaled beyond 128K cores due to its large memory footprint and, additionally, it can be slower than the linear solver for systems with very high attenuation. This is because the sparse linear solvers are iterative, but they converge quickly for systems with large attenuation, as the impact of radiation can be isolated to a subset of the domain for these systems.

For systems with scattering, DOM typically lags the scattering term and then resolves until the intensities converge to within a certain tolerance. The convergence can be costly for systems where the scattering coefficients are significantly larger than the absorption coefficients. Given these very encouraging results, applying sweeps to the full problem and improving its memory use are clearly the next steps.

## 6 Mira Production Cases – Results

In making use of the improvements to scalability of the entire code, two production cases were considered using the geometry, inlet parameters and operating parameters of a GE Power 8 Corner Unit. The first case represented the operation of the commercial unit that is currently in production, whereas the second case represented alterations to the inlet parameters to investigate a more uniform energy distribution. Each case was run for approximately 20s of physical time, which is considered sufficient for the boiler to achieve a steady state distribution.

Table 2 shows the computational aspects of the 2 cases that were run on Mira, simulating the 8-corner unit. Each production case was run at 260K cores with 455 M grid cells at a resolution of 4.25 cm with 16 MPI ranks per node using a timestep of  $8e^{-5}$  and about 400 MB of memory per rank. Table 2 shows that between the first and second cases additional speed-ups were achieved in the

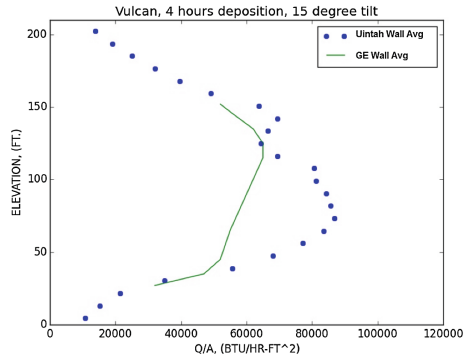
**Table 2.** Computational aspects of 8-corner boiler simulations

Item	Original inlets	Modified inlets
Number of cells	455 M	455 M
Cell resolution	4.25 cm	4.25 cm
MPI ranks	260,712	260,712
MPI ranks/node	16	16
Memory/rank	412 MB	372 MB
$dt$	$8e-5$	$8e-5$
Time/timestep	4.5 s	3.0 s
Pressure solve	1.9 s	1.0 s
Radiation solve	101 s	79.3 s
Data to disk	5.5 min	33.2 s
Data dumps	77	1030
Data size	9.9 TB	180 TB
# Timesteps	236,500	220,979
Simulated time	19.38 s	17.92 s
CPU hours	97 M	110 M

pressure solve due to work being done in Uintah/Arches. The most significant performance improvement was the switch of the I/O library, with the presented restructuring-based I/O, which resulted in 33 s write times, compared to the 5.5 min required on the Original Inlets case which used the legacy Uintah I/O system. Ultimately, the Modified Inlets case wrote 1030 datasets allowing for the creation of 3D rendered movies of the simulation.

Though validation of the simulation data against experimental data was performed, the proprietary nature of both the simulation and experimental data makes publication of these comparisons problematic. However, working closely with the GE Power engineers made it possible to validate the results of these simulations against their previous results. Figure 11 depicts the heat absorption profile (x-axis) as a function of the elevation in the boiler (y-axis), and shows the average absorption profile predicted in the unmodified inlet configuration (Original Inlets) is different from the tentative estimates due to the higher fidelity modeling performed with Arches, but it is in relatively good agreement with the actual absorption profile based on discussions with GE Power engineers and the existing proprietary data provided. The second case was run with changes to the inlet geometry parameters to optimize gas-side energy imbalance (GSEI) by changing the flow pattern in the wind-box as well as the SOFA inlets.

The key result from this work is the confidence that has been established with GE Power to demonstrate that high resolution LES simulations are a useful tool for exploring a range of operating conditions, with the potential to be used for future designs. This is the first time that computational design at this scale has



**Fig. 11.** Heat absorption profile as a function of the elevation. The solid green line shows GE Power’s wall-averaged absorption profile tentative estimates for the expected operating conditions in the unit. The blue dots show the average absorption profile computed from unmodified inlet case. (Color figure online)

been used for such a complex combustion problem with petascale simulations. Future studies of the unit will investigate design and operation adjustments to achieve incremental improvements in gas-side energy imbalance. GE will consider testing the new conditions in the existing unit when significant improvements are discovered.

## 7 Conclusions

This work has introduced an excellent exascale candidate problem through the successful simulation of a commercial, 1000 megawatt electric (MWe) ultra-supercritical (USC) boiler, the largest currently in production worldwide, using Large-Eddy Simulation (LES) predictions, requiring 280 Million CPU hours on Mira. The overall objective of this work was in understanding how we can solve such a problem through the use of an AMT runtime to efficiently schedule and execute computational tasks, including I/O, and to leverage scalability improvements in the runtime itself, linear solvers, I/O, and computational algorithms. To achieve the results shown in this work for production-level petascale computations significant code and algorithmic innovations were required, including novel adaptations of I/O system that achieved a nearly order of magnitude improvement in I/O performance.

Through this work, we have exposed areas even within an advanced, scalable runtime system that need careful design consideration for post-petascale and eventually exascale platforms, particularly when globally coupled problems such as radiation are considered. For example, while existing radiation methods used in Uintah scale, it is clear from the results presented that the use of the sweeps method for problems of this scale and size needs to be investigated further, to see if it is possible to reduce the overall simulation time significantly. A key lesson this work conveys is that the success of large, production-scale simulations

depends upon scalability at every level of the code. If any single component within the simulation pipeline does not scale, the problem cannot be solved. It is through the integration of these scalable components and subsystems that the next generation of problems may be solved on exascale systems. Finally, our results have demonstrated the potential role that LES simulations can have on analysis and design of an operational commercial boiler and that simulations can be used as a design tool for future systems, and that choosing fast scalable and hardware appropriate algorithms, for key areas such as radiation is important in achieving scalable results.

**Acknowledgements.** This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. An award of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357. The authors would like to thank the Intel Parallel Computing Centers program. We would like to thank all those involved with Uintah past and present.

## References

1. Mira home page. <https://www.alcf.anl.gov/mira>
2. Adams, M.P., Adams, M.L., Hawkins, W.D., Smith, T., Rauchwerger, L., Amato, N.M., Bailey, T.S., Falgout, R.D.: Provably optimal parallel transport sweeps on regular grids. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2013)
3. Ayachit, U.: The ParaView Guide: A Parallel Visualization Application. Kitware Inc., New York (2015)
4. Bailey, T., Hawkins, W.D., Adams, M.L., Brown, P.N., Kunen, A.J., Adams, M.P., Smith, T., Amato, N., Rauchwerger, L.: Validation of full-domain massively parallel transport sweep algorithms. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2014)
5. Balaaji, P., Chan, A., Thakur, R., Gropp, W., Lusk, E.: Toward message passing for a million processes: characterizing MPI on a massive scale blue gene/P. *Comput. Sci. Res. Dev.* **24**(1), 11–19 (2009)
6. Balsara, D.: Fast and accurate discrete ordinates methods for multidimensional radiative transfer. Part I, basic methods. *J. Quant. Spectrosc. Radiat. Transf.* **69**(6), 671–707 (2001)
7. Bennett, J., Clay, R., Baker, G., Gamell, M., Hollman, D., Knight, S., Kolla, H., Sjaardema, G., Slattengren, N., Teranishi, K., Wilke, J., Bettencourt, M., Bova, S., Franko, K., Lin, P., Grant, R., Hammond, S., Olivier, S., Kale, L., Jain, N., Mikida, E., Aiken, A., Bauer, M., Lee, W., Slaughter, E., Treichler, S., Berzins, M., Harman, T., Humphrey, A., Schmidt, J., Sunderland, D., McCormick, P., Gutierrez, S., Schulz, M., Bhatele, A., Boehme, D., Bremer, P., Gamblin, T.: ASC ATDM level 2 milestone #5325: asynchronous many-task runtime system analysis and assessment for next generation platforms. Technical report, Sandia National Laboratories (2015)

8. Berzins, M., Beckvermit, J., Harman, T., Bezdjian, A., Humphrey, A., Meng, Q., Schmidt, J., Wight, C.: Extending the uintah framework through the petascale modeling of detonation in arrays of high explosive devices. *SIAM J. Sci. Comput.* **38**(5), 101–122 (2016)
9. Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G.H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E.W., Camp, D., Rübel, O., Durant, M., Favre, J.M., Navrátil, P.: VisIt: an end-user tool for visualizing and analyzing very large data. In: *High Performance Visualization-Enabling Extreme-Scale Scientific Insight*, pp. 357–372 (2012)
10. HDF5 home page. <http://www.hdfgroup.org/HDF5/>
11. Humphrey, A., Harman, T., Berzins, M., Smith, P.: A scalable algorithm for radiative heat transfer using reverse Monte Carlo ray tracing. In: Kunkel, J.M., Ludwig, T. (eds.) *ISC High Performance 2015*. LNCS, vol. 9137, pp. 212–230. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-20119-1\\_16](https://doi.org/10.1007/978-3-319-20119-1_16)
12. Humphrey, A., Sunderland, D., Harman, T., Berzins, M.: Radiative heat transfer calculation on 16384 GPUs using a reverse Monte Carlo ray tracing approach with adaptive mesh refinement. In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1222–1231, May 2016
13. Kumar, S., Vishwanath, V., Carns, P., Levine, J., Latham, R., Scorzelli, G., Kolla, H., Grout, R., Ross, R., Papka, M., Chen, J., Pascucci, V.: Efficient data restructuring and aggregation for I/O acceleration in PIDX. In: *2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, November 2012
14. Li, J., Liao, W.-K., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., Zingale, M.: Parallel netCDF: a high-performance scientific I/O interface. In: *Proceedings of SC 2003: High Performance Networking and Computing*, Phoenix, AZ. IEEE Computer Society Press, November 2003
15. Lofstead, J., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: 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 2008*, pp. 15–24. ACM, New York, June 2008
16. Lustre home page. <http://lustre.org>
17. Meng, Q., Humphrey, A., Schmidt, J., Berzins, M.: Investigating applications portability with the Uintah DAG-based runtime system on petascale supercomputers. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC 2013*, pp. 96:1–96:12. ACM, New York (2013)
18. U. D. of Energy: Exascale computing project (2017). <https://exascaleproject.org/>
19. Peterson, B., Humphrey, A., Schmidt, J., Berzins, M.: Addressing global data dependencies in heterogeneous asynchronous runtime systems on GPUs. In: *Third International Workshop on Extreme Scale Programming Models and Middleware, ESPM2*. IEEE Press (2017, submitted)
20. Russell, J.: Doug Kothe on the race to build exascale applications (2017). <https://www.hpcwire.com/2017/05/29/doug-kothe-race-build-exascale-applications/>
21. Schmuck, F., Haskin, R.: GPFS: a shared-disk file system for large computing clusters. In: *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pp. 231–244 (2002)
22. Scientific Computing and Imaging Institute: Uintah web page (2015). <http://www.uintah.utah.edu/>

23. Shan, H., Antypas, K., Shalf, J.: Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In: Proceedings of Supercomputing, November 2008
24. Wald, I., Johnson, G., Amstutz, J., Brownlee, C., Knoll, A., Jeffers, J., Günther, J., Navratil, P.: OSPRay - a CPU ray tracing framework for scientific visualization. *IEEE Trans. Visual. Comput. Graph.* **23**(1), 931–940 (2017)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

