Radiative Heat Transfer Calculation on 16384 GPUs Using a Reverse Monte Carlo Ray Tracing Approach with Adaptive Mesh Refinement

Alan Humphrey
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT, 84112 USA
Email: ahumphrey@sci.utah.edu

Daniel Sunderland
Sandia National Laboratories
PO Box 5800 / MS 1418
Albuquerque, NM, 87175 USA
Email: dsunder@sandia.gov

Todd Harman
Department of Mechanical Engineering
University of Utah
Salt Lake City, UT, 84112 USA
Email: t.harman@utah.edu

Martin Berzins
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT, 84112 USA
Email: mb@sci.utah.edu

***Abstract*—Modeling thermal radiation is computationally challenging in parallel due to its all-to-all physical and resulting computational connectivity, and is also the dominant mode of heat transfer in practical applications such as next-generation clean coal boilers, being modeled by the Uintah framework. However, a direct all-to-all treatment of radiation is prohibitively expensive on large computers systems whether homogeneous or heterogeneous. DOE Titan and the planned DOE Summit and Sierra machines are examples of current and emerging GPU-based heterogeneous systems where the increased processing capability of GPUs over CPUs exacerbates this problem. These systems require that computational frameworks like Uintah leverage an arbitrary number of on-node GPUs, while simultaneously utilizing thousands of GPUs within a single simulation. We show that radiative heat transfer problems can be made to scale within Uintah on heterogeneous systems through a combination of reverse Monte Carlo ray tracing (RMCRT) techniques combined with AMR, to reduce the amount of global communication. In particular, significant Uintah infrastructure changes, including a novel lock and contention-free, thread-scalable data structure for managing MPI communication requests and improved memory allocation strategies were necessary to achieve excellent strong scaling results to 16384 GPUs on Titan.**

***Keywords*-Uintah; Radiation Modeling; Titan; Reverse Monte Carlo Ray Tracing; Mesh Refinement; GPU; Scalability**

I. Introduction

The need to solve larger and more complex simulation problems while at the same time not incurring higher and higher power costs has led to an increasing focus on GPU and Intel Xeon Phi-based architectures. Many existing and most emerging high performance computing (HPC) systems rely on such architectures. In the case of the DOE Titan system, with a theoretical peak performance of 27 petaflops, over 90% of the computational power come from its 18,688 GPUs. These heterogeneous systems pose significant challenges in terms of programmability due to deep memory hierarchies, vendor-specific language extensions and memory constraints, e.g. less device-side memory compared to host memory per node. To preserve current capabilities on upcoming machines and to solve larger and more complex simulations on existing machines, HPC codes must effectively leverage manycore architectures. In this paper we focus on the changes to Uintah needed to leverage GPU-based architectures such as DOE Titan and the proposed DOE Summit, for large-scale calculations. To achieve good performance on these architectures, it is important that algorithms and codes effectively leverage an arbitrary number of GPUS on-node while simultaneously utilizing all GPUs available in an allocation, potentially thousands.

The Uintah development effort aimed at using these GPU-based machines at scale is driven by the target problem of the University of Utah Carbon Capture Multidisciplinary Simulation Center (CCMSC), funded by the NNSA Predictive Science Academic Alliance Program (PSAAP) II. The CCMSC aims to simulate through petascale and eventually exascale, a 1000MWe oxy-fired clean coal boiler being developed by Alstom Power to deliver high efficiency electric power generation with carbon capture. A primary CCMSC focus is on using extreme-scale computing for reacting, large eddy simulation (LES)-based codes within the Uintah open source framework, using machines like Titan and the upcoming Summit system in a scalable manner. The physical size of the CCMSC target boiler simulations necessitates the use of systems like DOE Titan at near-capacity to adequately resolve the computational domain in a tractable amount of time.

Radiation is the dominant mode of heat transfer in these boiler simulations, and because radiative heat transfer rates are generally proportional to the fourth power of the temperature

[1], applications such as the CCMSC boiler simulations that simulate a turbulent combustion process, are highly influenced by the accuracy of the radiation models used.

The CCMSC has been actively pursuing the use of photon Monte Carlo (PMC) methods, originally considered in [1] and more specifically reverse Monte Carlo ray tracing (RMCRT), initially developed in [2] and [3] to compute radiative heat flux and its divergence. RMCRT naturally incorporates scattering physics and also lends itself to scalable parallelism due to the mutual exclusivity of the rays. Each ray or ray bundle may be traced independently, making PMC approaches an ideal candidate for GPU parallelization. This is in stark contrast to the discrete ordinates method (DOM) [4], currently used within Uintah, which is computationally expensive, involves multiple global, sparse linear solves and presents challenges with the incorporation of scattering physics.

A principal challenge in modeling radiative heat transfer is the strong nonlocal nature of radiation, with potential propagation of radiation across the entire domain from any point. For our RMCRT model, this translates to an all-to-all communication requirement that replicates the boiler geometry on each node to facilitate local ray tracing. We address this challenge by leveraging Uintah's adaptive mesh refinement (AMR) capabilities, using Cartesian mesh patches to generate a fine mesh that is only used locally (close to each grid point) and a successively coarser mesh is used further away, via a level-upon-level approach. This approach is fundamental to the CCMSC target problem, where the entire computational domain needs to be resolved to adequately model the radiative heat flux. Using this approach, we previously showed excellent strong scaling to over 256k CPU cores on the DOE Titan system for problem sizes that were previously intractable with a single fine mesh (single-level) RMCRT approach due to on-node memory constraints [5]. This scaling was consistent with the communication and computation model in [5].

The challenges in moving from a CPU to a GPU-based multi-level RMCRT algorithm using this mesh refinement approach have extended well beyond what a typical GPU port of a CPU code might entail. In the case of the Uintah open-source framework, additional complexities are posed by these architectures based on a core Uintah design that focuses on insulating the application developer from the underlying architecture. Thus in the context of heterogeneous systems, Uintah's asynchronous task-based paradigm requires that all host-to-device and device-to-host data copies for computational task dependencies (inputs and outputs), as well as device context management must be handled automatically in the same way MPI messages are generated by the Uintah runtime system, as shown in [6] and [7].

The current Uintah model has departed from an MPI-only approach and now employs a shared memory model on-node [7], [8]. This combination of MPI + Pthreads, and in the presence of GPUs, also Nvidia CUDA, all coupled with shared data structures and the use of MPI_THREAD_MULTIPLE (where all CPU threads perform their own MPI sends and receives), creates an environment for potential race conditions and deadlock scenarios, some of which only manifest at larger scale in our experience and are routinely difficult to debug.

This work focuses on how we have addressed the challenges involved with this mixed concurrency environment to scale this difficult globally-coupled, all-to-all problem to 16,384 GPUs on the DOE Titan system, showing this approach to be feasible in production boiler calculations on current and future GPU-based heterogeneous architectures. This result has been achieved through a focused progression, starting first with our work shown in [6] to achieve basic GPU task scheduling and execution. This work implemented a proof-of-concept, single-level GPU RMCRT algorithm and heterogeneous task scheduler and runtime system within Uintah, and was the origin of this work. Second in this progression was our preliminary multi-level RMCRT work, focusing on CPU scaling, where in [5] we demonstrate excellent strong scaling to over 256K CPU cores on the DOE Titan system. The specific contributions made by this work in moving from a CPU to a GPU-based multi-level RMCRT algorithm are the extensive modifications to the Uintah infrastructure necessary to achieve the GPU scaling results shown in Section V. A more in-depth study of CPU vs GPU performance will be part of future work.

These contributions are:

(i) Leveraging Uintah's AMR infrastructure in a novel way to reduce the volume of communication sufficiently so as to allow scalability. Uintah's AMR capabilities are introduced in Section II, along with an overview of Uintah.

(ii) Changing the way that AMR meshes are stored on the GPU to overcome the limited available GPU global memory. This has entailed a significant extension of the Uintah `GPU DataWarehouse` system [9] to support a mesh-level database, a repository for shared, per-mesh-level variables such as global radiative properties. This has allowed multiple mesh patches, each with associated GPU tasks, to run concurrently on the GPU while sharing coarse, radiation mesh data. This extension of the `GPU DataWarehouse` is discussed in Section III, which also gives an overview on radiation transport and describes our GPU-based multi-level RMCRT model.

(iii) The introduction of novel non-blocking, thread-scalable data structures for managing asynchronous MPI communication requests, replacing previously problematic Mutex-protected vectors of MPI communication records. To be non-blocking a wait, failure, or resource allocation by one thread cannot block progress on any other thread. Non-blocking data-structures are lock-free if at all steps at least one thread is guaranteed to make progress, and are wait-free if at any step all threads are guaranteed to make progress [10]. Section IV describes these changes and their motivation, and also shows speedups in local MPI communication times made possible through these infrastructure improvements.

(iv) A vastly improved memory allocation strategy to reduce heap fragmentation is covered in Section IV, that allows running simulations at the edge of the nodal memory footprint on machines like Titan.

(v) Determining optimal fine mesh patch sizes to yield GPU performance while maintaining over-decomposition of the computational domain to hide latency. This is covered in Section V where we provide strong scaling results over a wide range of GPU counts up to 16,384 GPUs, and also show the results of differing patch configurations across this range of GPUs

An overview of related work is given in Section VI, and the paper concludes in Section VII with future work in this area.

II. The Uintah Code

The Uintah open-source (MIT License) software has been widely ported and used for many different types of problems involving fluids, solids and fluid-structure interaction problems [11], with the latest release in January 2015 [12]. Uintah consists of a set of parallel software components and libraries that facilitate the solution of partial differential equations on structured AMR grids. Uintah presently contains four main simulation components: 1.) the multi-material ICE [13] code for compressible flows; 2.) the particle-based code MPM [14] for structural mechanics; 3.) the combined fluid-structure interaction (FSI) algorithm MPM-ICE [15] and 4.) the ARCHES turbulent reacting CFD component [16] that was designed for simulating turbulent reacting flows with participating media radiation. Uintah is highly scalable [17], runs on many National Science Foundation (NSF), Department of Energy (DOE) and Department of Defense (DOD) parallel computers on a broad class of problems.

Uintah is unique in its methods and its use of a directed acyclic graph (DAG) approach as part of a production-strength code in a way that is coupled to a runtime system. Uintah's design maintains a clear partition between applications code and its runtime system, making it possible to achieve great increases in scalability through changes to the runtime system *without changes to the applications themselves*.

Particular advances made in Uintah include highly scalable AMR using Cartesian mesh patches [17]. A key factor in improving performance has been the reduction in MPI wait time through the dynamic and even out-of-order execution of task-graphs [18]. The need to reduce memory use in Uintah led to the adoption of a nodal shared memory model in which there is only one MPI process per multicore node, and execution of tasks is on individual cores through Pthreads [19]. As a result, Uintah has demonstrated scalability to 768K cores on complex fluid-structure interactions with AMR. Uintah's thread-based runtime system [19] uses decentralized execution of the task-graph, implemented by each CPU core requesting work itself and performing its own MPI. A lock-free shared memory abstraction through Uintah's `DataWarehouse` approach [19] was implemented using atomic operations, allowing efficient access by all cores to the shared data on a node. Finally, the nodal architecture of Uintah has been extended to run tasks on one or more on-node accelerators [6] by using a multi-stage queue architecture to organize work for CPU cores and GPUs in a dynamic way, and is the starting point for this paper.

A. The ARCHES Combustion Simulation Component

The ARCHES component within the Uintah computational framework was designed for the simulation of turbulent reacting flows with participating media radiation. It is a three-dimensional, Large Eddy Simulation (LES) code described in [20]. ARCHES uses a low-Mach number ($M < 0.3$), variable density formulation to model heat, mass, and momentum transport in reacting flows.

ARCHES is the primary CCMSC simulation component, and solves the coupled mass, momentum and energy conservation equations on a staggered finite-volume mesh for the gas and solid phase with combustion [16], [21]. The discretized equations are integrated in time using an explicit, strong-stability preserving second or third-order Runge-Kutta method [22]. Spatial discretization is handled with central differencing where appropriate for energy conservation or flux limiters (eg, scalar mixture fractions) to maintain numerical accuracy. The low-mach, pressure projection formulation requires a solution of sparse linear system at each timestep using the *Hypre* linear solver package [23]. The turbulent subgrid velocity and species fluctuations [24] are modeled with the dynamic Smagorinsky closure model. The solution procedure solves the intensity equation over a discrete set of ordinates and, like the pressure equation, is formulated as a linear system that is solved using *Hypre*. Research using ARCHES has been done on radiative heat transfer using the parallel discrete ordinates method [4] (DOM, a modeling method developed at Los Alamos National Laboratory for neutron transport) and the P1 approximation to the radiative transport equation [25]. Work done by Sun [2] and Hunsaker [3] has shown that Monte Carlo ray tracing methods are potentially more efficient and offer an alternative to DOM.

III. RMCRT Model

Scalable radiation modeling plays a key computational role in applications such as heat transfer in combustion simulations [20], neutron transport modeling [26] in nuclear reactors and astrophysics modeling, and is generally considered one of the most challenging problems in large-scale computational science and engineering due to the global nature of radiation. For heat transfer problems such as the CCMSC boiler simulations, coupling combustion and radiation poses several numerical challenges. The fluid mechanics of combustion are an inherently local phenomena, wherein conservation laws may be applied over a finite volume. Radiation however, is a long-distance phenomenon due to strong nonlocal effects. Because of these nonlocal effects, conservation laws cannot be applied over an infinitesimal volume, but must be applied over the entire computational domain, creating difficulties for domain decomposition due to the need for nonlocal data.

A. Radiation Transport Models

The heat transfer problem arising from the clean coal boilers being modeled by the ARCHES component [16] within the Uintah framework has thermal radiation as the dominant heat transfer mode and involves solving the conservation of energy equation (1) and radiative heat transfer equation (RTE) 2 simultaneously. A critical quantity of interest for all boiler simulations is the heat flux to the surrounding walls, as the major mode of heat transfer in the coal-fired boiler is radiation. In the context of the CCMSC, the design of new boiler facilities utilizing ultra super critical air-combustion technology will require accurate radiative heat flux estimates in environments with increased CO_2 concentrations, higher temperatures and different radiative properties for new metal alloys. Thermal radiation in the target boiler simulations is loosely coupled to the computational fluid dynamics (CFD) due to time-scale separation.

ARCHES is designed to solve the mass, momentum, mixture fraction, and thermal energy governing equations inherent to coupled turbulent reacting flows. ARCHES has relied primarily on a DOM solver [4] to compute the radiative source term in the energy equation shown by:

$$c_v \frac{dT}{dt} = -\nabla \cdot (\kappa \nabla T) - p\nabla \cdot v + \Phi + Q''' - \nabla \cdot q_r \quad (1)$$

where c_v is the specific heat, T is the temperature field, p is the pressure, κ is the thermal conductivity, v is the velocity vector, Φ is the dissipation function, Q''' is the heat generated within the medium, e.g. chemical reaction, and $\nabla \cdot q_r$ is the net radiative source [5]. A radiatively participating medium can emit, absorb and scatter thermal radiation. The energy equation is then conventionally solved by ARCHES (finite volume) and the temperature field, T is used to compute the net radiative source term. This net radiative source term is then fed back into the energy equation (for the ongoing CFD calculation) which is solved to update the temperature field, T [5].

A particular limitation of DOM is false scattering. This is a due to spatial discretization error, similar to numerical diffusion in CFD calculations. A ray that is traced through the enclosure by DOM will gradually widen as it moves farther away from its point of origin. False scattering can be addressed by using a finer mesh of control volumes, but at greater computational cost [1].

Recent work has shown that Monte Carlo ray tracing (MCRT) methods are potentially more efficient [3], [2]. Traditional forward MCRT approaches are inefficient though, in that large numbers of traced rays may not reach the subdomain of interest. Both DOM and MCRT methods aim to approximate the radiative transfer equation (2), the equation describing the interaction of absorption, emission and scattering for radiative heat transfer, which is an integro-differential equation with three spatial variables and two angles that determine the direction of $\hat{s}$ [27]. For MCRT methods, a statistically significant number of rays (photon bundles) are traced from a computational cell to the point of extinction, that is, until their radiative intensity falls below a specified threshold. This method is then able to calculate energy gains and losses for every element in the computational domain.

Reverse Monte Carlo ray tracing (RMCRT), the focus of this work, is an emission-based reciprocity method, where rays are traced backwards from the detector, thus eliminating the need to track ray bundles that never reach the detector [28]. Rather than integrating the energy lost as a ray traverses the domain as in forward MCRT approaches, RMCRT integrates the incoming intensity absorbed at the origin, where the ray was emitted. RMCRT is more amenable to domain decomposition, and thus Uintah's parallelization scheme due to the backward nature of the process [2], and the mutual exclusivity of the rays themselves. The process is considered reverse through the Helmholtz Reciprocity Principle, e.g. incoming and outgoing intensity can be considered as reversals of each other [29].

$$\begin{aligned} \frac{dI(\hat{s})}{ds} &= \hat{s}\nabla I(\hat{s}) \\ &= k_\eta I - \beta I(\hat{s}) \\ &+ \frac{\sigma_s}{4\pi}\int_{4\pi} I(\hat{s})\Phi(\hat{s_i},\hat{s})d\Omega_i, \end{aligned} \quad (2)$$

In equation 2, k_η is the absorption coefficient, σ_s is the scattering coefficient, dependent on the incoming direction s. β is the extinction coefficient that describes total loss in radiative intensity, I is the change in intensity of incoming radiation from point s to point $s + ds$ and is determined by summing the contributions from emission, absorption and scattering from direction $\hat{s}$ and scattering into the same direction $\hat{s}$. $\Phi(\hat{s_i},\hat{s})$ is the phase function that describes the probability that a ray coming from direction s_i will scatter into direction $\hat{s}$ and integration is performed over the entire solid angle Ω_i [28], [27]. Though a method for modeling spectral effects has been considered, currently we are using a mean absorption coefficient approximation (σ_s) in combination with the mean optical path length, and hence not resolving spectral frequencies, e.g. η for wavelength. Adding spectral frequencies to RMCRT would entail adding a loop over wave-lengths, η and is part of future work.

B. *RMCRT and Ray Tracing Overview*

The principal motivation for the development of a GPU-based RMCRT radiation calculation arises from the computational intensity of the radiation solve in the CCMSC production runs, consuming as much as 50% of the overall CPU time per timestep when using DOM. Additionally this work is motivated by access to large-scale GPU-based machines like DOE Titan, where over 90% of the available FLOPS are on the GPUs. Many of the CCSMSC target simulations will run on Titan over its life span. Beyond this, utilization of the planned DOE Summit system is planned.

RMCRT uses rays more efficiently than forward MCRT, but it is still an *all-to-all* method, for which all of the geometric information and radiative properties for the entire computational domain must be accessible by every ray [2]. These radiative properties consist of; κ, the absorption coefficient, a property of the medium the ray is traveling through, σT^4, a physical constant $\sigma\cdot$ temperature field, T^4 and, $cellType$ (boundary or flow cell), a property of each computational cell in the domain. In our approach, the boiler geometry is replicated on each node and ray tracing takes place without the need to pass ray information across nodal boundaries (via MPI) as rays traverse the computational domain. Our RMCRT approach is afforded the choice of replication due to the relative simplicity of the boiler geometry.

To address these communication challenges, we have developed a multi-level AMR approach for both CPU [5] and now GPU, in which a fine mesh is only used close to each grid point and a successively coarser mesh is used further away, significantly reducing MPI message volume and nodal memory footprint. This algorithm allows for the radiation computation to be performed with an appropriate mesh resolution while still being coupled with other physics components. The LES CFD, particle transport and particle reactions are solved on a different mesh resolution appropriate to their physics and models. This balanced approach to coupling multiphysics is made possible by Uintah's AMR design. The amount of data stored on every computational patch is significantly reduced, and the computational overhead for successively finer computation is eliminated when not needed.

C. Multi-Level GPU Implementation

Following our original proof-of-concept GPU task scheduler introduced in [6], a single-level CPU and GPU RMCRT approach was initially considered. This approach was to begin comparisons against the current DOM solver within the Uintah ARCHES component, using the benchmark problem described by Burns and Christon in [30]. Accuracy studies of this single-level RMCRT approach are shown in [3] for this benchmark, which examines the accuracy of the computed divergence of the heat flux and shows expected Monte Carlo convergence when compared to the published data in [30]. In this approach, the quantity of interest, the divergence of the heat flux, ∇q is calculated for every cell in the computational domain. The entire domain was replicated on every node (with all-to-all communication) for the radiative properties. This replication occurred on the single fine mesh, which for N_{total} mesh cells, the amount of data communicated is $\mathcal{O}(N_{total}^2)$.

Though this single, fine mesh approach was highly accurate and effective at lower core and GPU counts, problem sizes beyond 256^3 were intractable for highly resolved domains, especially on machines with less than 2GB of memory per core. GPU scalability results were shown up to 64 GPUs through the work done in [6] to achieve basic accelerator task scheduling and execution. Using a problem size of 128^3, the volume of communication coupled with the PCIe transfers begins to dominate, and the GPUs were starved for work with only a single patch per GPU. These difficulties led to the use of an AMR approach that uses a mesh hierarchy to limit the amount of communication on CPU architectures [5].

Figure 2 in [5] best illustrates this approach with a 2-D diagram of three-level mesh refinement scheme, illustrating how a ray from a fine-level patch (right) might be traced across a coarsened domain (left). In general, the data required by our multi-level RMCRT algorithm from the fine CFD mesh, is projected to all coarse levels subject to a user-defined refinement ratio (typically 2 or 4), where each coarse level spans the entire domain. Our general multi-level RMCRT ray marching process is described in detail in [5], which includes a precise model of communication and computation.

A significant challenge in moving to a GPU-based, multi-level RMCRT algorithm is the limited amount of global memory available on the current generation of GPUs found on Titan. These Nvidia K20X models have 6GB compared to 32GB CPU host-side. The Uintah `DataWarehouse` design automatically generates MPI messages and keeps multiple versions of variables for out-of-order scheduling and execution [18], as different tasks may require the same variable on the same neighboring patch multiple times for differing ghost cell requirements. Tasks may also need input variables prior to modification. In order to support these and other scenarios, the "*on-demand*" `DataWarehouse` provides the application the illusion it has access to memory it does not actually own (via the task input specification, where the ghost cell requirement is specified). In the context of our multi-level RMCRT radiation model, this is a global halo, or "infinite ghost cell" requirement on all coarse levels. Because of this design, data from the coarser levels is retrieved from the Uintah `DataWarehouse` for each fine level patch on a node. This presents problems for a limited memory footprint as on Titan's K20X GPUs.

Our solution to this problem has been to effectively short-circuit the creation of these redundant global copies of the radiative properties on the host and their subsequent transfer across the PCIe bus to the GPU. This has been achieved by a significant extension of the Uintah `GPU DataWarehouse` system [9] to support a level database that stores a single copy of shared global radiative properties (per-mesh level based on Uintah's level-upon-level approach to AMR). Our solution has effectively minimized PCIe transfers and ultimately allowed multiple mesh patches, each with GPU tasks, to run concurrently on the GPU while sharing data from the coarse radiation mesh. This design leverages the two copy engines available on the K20X GPUs and also makes use of support for running multiple, concurrent kernels. Using these features, Uintah can copy data for multiple fine-mesh patches to the GPU, each sharing a global copy of the coarsened radiative properties.

Data for these GPU tasks can be simultaneously copied to-and-from the device as multiple RMCRT kernels run simultaneously. `CUDA Streams`, managed by the Uintah infrastructure provide additional concurrency, as operations from different streams can be interleaved.

IV. Infrastructure Improvements

Uintah uses an "MPI + *X*" approach, a combination of MPI + (Pthreads + Nvidia CUDA). This mixed concurrency model has the potential for problematic race conditions and deadlock scenarios, some of which only manifest at larger scale in our experience. Significant infrastructures changes were necessary to improve nodal throughput and to expose more concurrency while maintaining correctness within this complex environment. In particular it was necessary to choose optimal data structures and algorithms to efficiently expose concurrency, as well as to maintain critical sections around legacy serial data structures. Furthermore, it was vital for Uintah to manage limited resources such as nodal memory through the use of custom allocators that allow frameworks like Uintah to choose more optimal allocation policies for different objects to better utilize available resources and improve nodal throughput.

A. Multi-Threaded Processing of Asynchronous MPI

Uintah currently uses MPI_THREAD_MULTIPLE (which to the best of our knowledge is rarely adopted by MPI users), which allows individual threads to perform their own MPI sends and receives. Initial attempts to run at large scale with accelerators in this environment exposed a subtle race condition in the shared vector used to process outstanding MPI_Requests via `MPI_Testsome()`, which was protected by a Pthread write-lock. This race scenario involved multiple threads simultaneously processing the same received message, with all threads allocating a buffer for the same MPI message, and only one thread actually processing the message and invoking the callback to deallocate its buffer. Other threads may have allocated buffers which were never released, resulting in a severe memory leak in the Uintah infrastructure, causing the application to quickly fail at large-scale due to *out of memory* errors on the compute nodes.

Though this scenario was present in other simulations, it was only evident at large scale, and only significant within our RMCRT radiation model due to the high volume and size of MPI messages. Despite this, the approach to multi-threaded processing of asynchronous MPI within Uintah had worked seemingly well for all cases until now.

A more coarse-grained critical section was not feasible as it would have serialized a substantial portion of the algorithm. The solution ultimately required a fundamental redesign in the data structure and algorithm used to manage MPI communication records in a multi-threaded environment. The new algorithm leverages a novel wait-free pool, which is thread-scalable and contention-free, to store individual MPI requests. The wait-free pool iterator is implemented as a unique, move-only object which toggles an atomic flag to protect access to the referenced value to prevent data races, i.e. multiple threads modifying the same value. Using `C++11` features (*atomics, move constructor, move assignment, and disabling copy construction and copy assignment*) to implement a unique protected iterator, that guarantees no two threads can have iterators which dereference to the same object. `MPI_Test()` is then used on each request individually in contrast to the prior design which used `MPI_Testsome()` to test a collection of requests. This solution, outlined in Algorithm 1 results in much simpler code with fewer allocations, eliminates the complexity of managing the previously used locked vectors of `MPI_Request` objects and their related critical sections.

In our experience, achieving multi-threaded correctness and performance requires different algorithm and data structure choices. This is illustrated in our example by moving to the use of `MPI_Test()` from multiple/many threads and away from the complexity of managing data structures designed for the use of `MPI_Testsome()`.

Algorithm 1 Wait-free MPI_Request pool

```
RecvCommList& recv_list = m_recv_lists[id];
auto ready_request =
  [](CommNode const& n)→bool{return n.test();};
iterator = recv_list.find_any(ready_request);
if (iterator) then
    MPI_Status status;
    iterator→finishCommunication(m_comm, status);
    recv_list.erase(iterator);
end if
```

B. Memory Allocation and Management Strategy

After identifying and addressing the race condition described above, our RMCRT benchmark problem [30] still failed at scale due to memory-related issues, though it ran longer before failure. Further investigation revealed that extreme heap fragmentation was occurring when running our RMCRT benchmark problem. Persistent small allocations mixed with transient large allocations fragmented the heap such that it grew continually, acting as though a significant memory leak still existed. Using Google's tcmalloc [31], a highly scalable memory allocator for multi-threaded applications, reduced heap fragmentation but the mixture of persistent and transient allocations still resulted in unacceptable fragmentation. Furthermore, frequent small allocations from multiple threads also caused a performance degradation due to contention of shared resources. The performance of the infrequent large allocations was not a factor in the overall performance.

1) Custom Allocators to Reduce Fragmentation: Developing and using custom allocator classes for Uintah's MPI buffers and `GridVariables` (simulation variables that reside on Uintah's Cartesian mesh patches at cell centers, nodes or faces x,y,z), allowed us to leverage our knowledge of how a data structure would be used to distinguish between large/small and transient/persistent allocations which greatly improved memory utilization and reduced fragmentation.

To eliminate the observed heap fragmentation, we developed allocators to address the range of allocation sizes which were causing the fragmentation. For large allocations, we completely avoided the heap by implementing a specialized allocator that uses `mmap` to allocate anonymous virtual memory. While `mmap` is a system call and can be slower than a standard `malloc`, it was more important to avoid fragmenting the heap than to optimize the throughput of large allocations. Throughput was not a concern for the performance of large allocations, but it is critical for frequent small allocations. To manage our small transient objects, i.e. objects that are frequently created and destroyed, we developed a lock-free memory pool on top of our `mmap` allocator to avoid the heap and to maximize throughput. All other infrequent allocations are still managed using the heap.

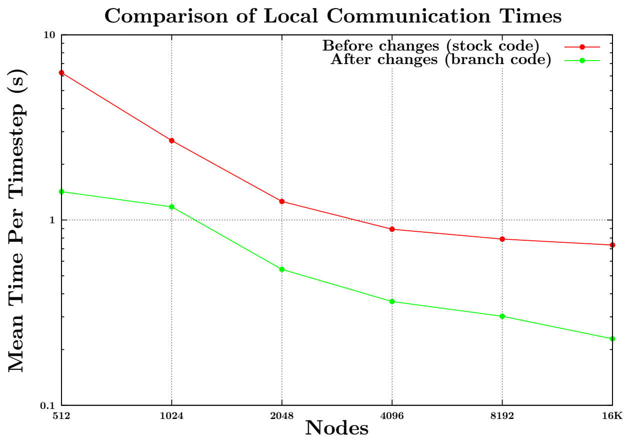

Fig. 1. Comparison of the local communication time (sec) before and after infrastructure improvements.

With these custom allocators, Uintah is now better able to manage memory requirements by designing for specific needs and requirements to reduce fragmentation and increase throughput when necessary. As the scope of problem sizes increase and simulations are pushed to the full capacity of available resources, specific management strategies and algorithms must be developed to better use available resources. Generic algorithms are no longer sufficient to accommodate the bleeding edge that high performance computing lives on.

Using these techniques, portions of Uintah infrastructure code related to communication were significantly simplified, and nodal throughput was improved by a factor of 2-4X in processing local MPI communication (the time spent posting MPI messages for individual threads). Figure 1 shows the time spent doing local communication, before and after our infrastructure improvements for our CPU implementation of the Burns and Christon [30] RMCRT benchmark on Titan. These runs were from 512 to 16,384 nodes, with a 2-level problem with 136.31M cells, 512^3 on the fine CFD mesh and 128^3 on the coarse radiation mesh. There were 262k total mesh patches in this problem.

Table I lists data from Figure 1 with times before and after infrastructure improvements and the associated speedups. More detailed information on communication frequency, as well as average message volume, and latency for this radiative heat transfer calculation is shown in [5]. The speedups shown in Table I are a direct result of removing only a single Mutex and related critical sections. We expect refactoring other sections of infrastructure code to yield similar improvements.

TABLE I
LOCAL COMMUNICATION DATA SHOWN IN FIGURE 1.

Comparison in Local Communication Times						
#Nodes	512	1k	2k	4k	8k	16k
Time (s) before	6.25	2.68	1.26	0.89	0.79	0.73
Time (s) after	1.42	1.18	0.54	0.36	0.30	0.23
Speedup (X)	4.40	2.27	2.33	2.47	2.63	3.17

V. Scaling Studies

In this section, we show strong scalability results on the DOE Titan XK7 [1] system for the Burns and Christon [30] benchmark problem using the GPU implementation of the multi-level mesh refinement approach. We define *strong scaling* as a decrease in execution time when a fixed size problem is solved on more cores, and *weak scaling* as the change in execution time as the number of processors and problem size vary proportionally to each other. We define parallel efficiency, E as:

$$E = \frac{T_{serial}}{N * T_{parallel}(N)}, \tag{3}$$

where T_{serial} is the time to solution using 1 processing unit, N is the number of processing units and $T_{parallel}(N)$ is the time to solve the same problem with N processing units. Weak scaling results are not shown here due to the nature of the growth in communication for this problem, specifically that radiation or any globally coupled algorithm grows quadratically as $\mathcal{O}(N^2)$ (N is the number of communicating MPI ranks) with respect to the problem size. Strong scaling is important in our case as the CCMSC seeks to solve a fixed target problem in a tractable amount of time using more compute resources. To achieve this, the CCMSC needs the whole of machines like Titan.

Figures 2 and 3 each show the performance and scalability of the multi-level RMCRT:GPU algorithm for three patch sizes. In each fine level cell in both problems, 100 rays were used to compute the divergence of the heat flux. The number of cells in a patch was varied, 16^3 (red), 32^3 (green), and 64^3 (blue). Each of the simulations consisted of a grid with 2 levels, and used a refinement ratio of 4 between the levels. All simulations were run on the DOE Titan system, leveraging the single GPU per node with Uintah's hybrid, multi-threaded task scheduler and runtime system originally designed and tested in [7], [9] using 16 threads and 1 GPU per node. This scheduler and runtime system has been heavily modified as outlined in Section IV to achieve the results shown here.

For the simulation results shown in Figure 2, the total number of cells in the domain was 17.04 million. The fine level contained 256^3 cells and the coarse level contained 64^3 cells. For the larger simulation results shown in Figure 3, the total number of cells in the domain was 136.31 million. The fine level contained 512^3 cells and the coarse level contained 128^3 cells. Using equation 3, the strong scaling efficiency of the large benchmark problem (Figure 3) is 96% going from 4096 to 8192 GPUs, and 89% going from 4096 to 16,384 GPUs

[1] Titan is a Cray KX7 system located at Oak Ridge National Laboratory, where each node hosts a 16-core AMD Opteron 6274 processor running at 2.2 GHz, 32 GB DDR3 memory and 1 NVIDIA Tesla K20x GPU with 6 GB GDDR5 ECC memory. The entire machine offers 299,008 CPU cores and 18,688 GPUs (1 per node) and over 710 TB of RAM. Titan uses a Cray Gemini 3D Torus network, 1.4 μs latency, 20 GB/s peak injection bandwidth, and 52 GB/s peak memory bandwidth per node.

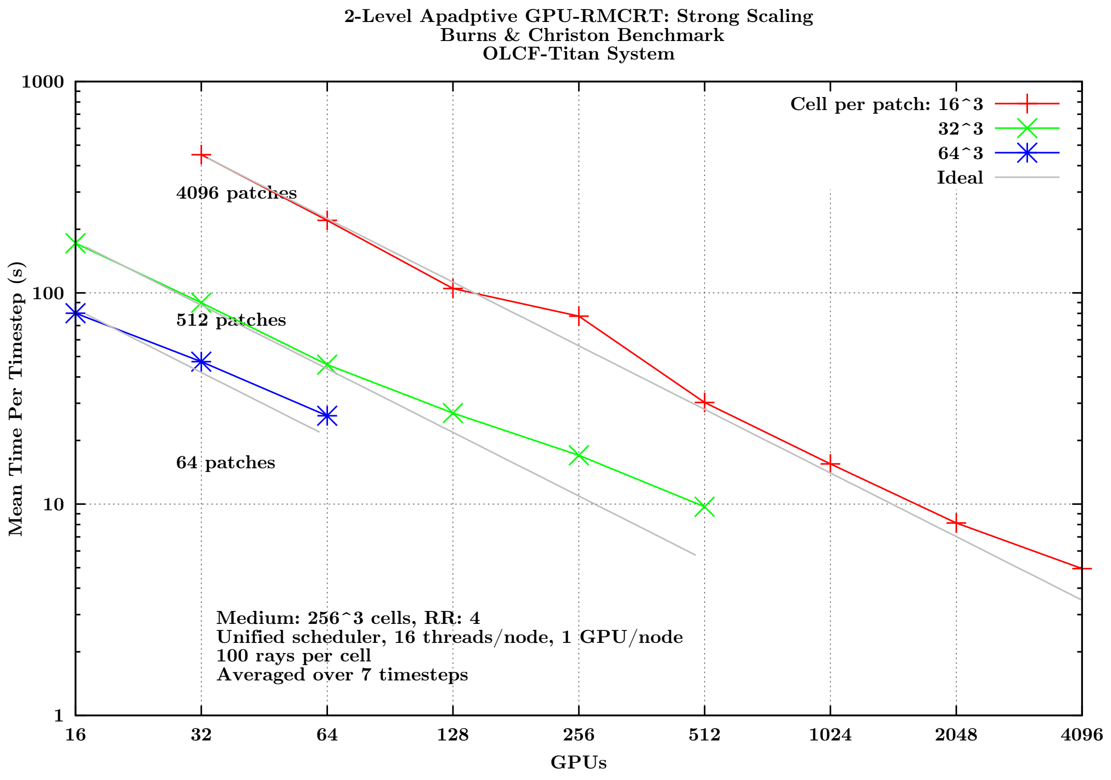

Fig. 2. GPU Strong scaling of the MEDIUM 2-level benchmark RMCRT problem for 3 patch sizes on the DOE Titan system. Refinement ratio of 4 between levels (RR:4). The fine CFD mesh contains 256^3 cells, coarse radiation mesh contains 64^3 cells. Different patch sizes illustrate GPU speedup.

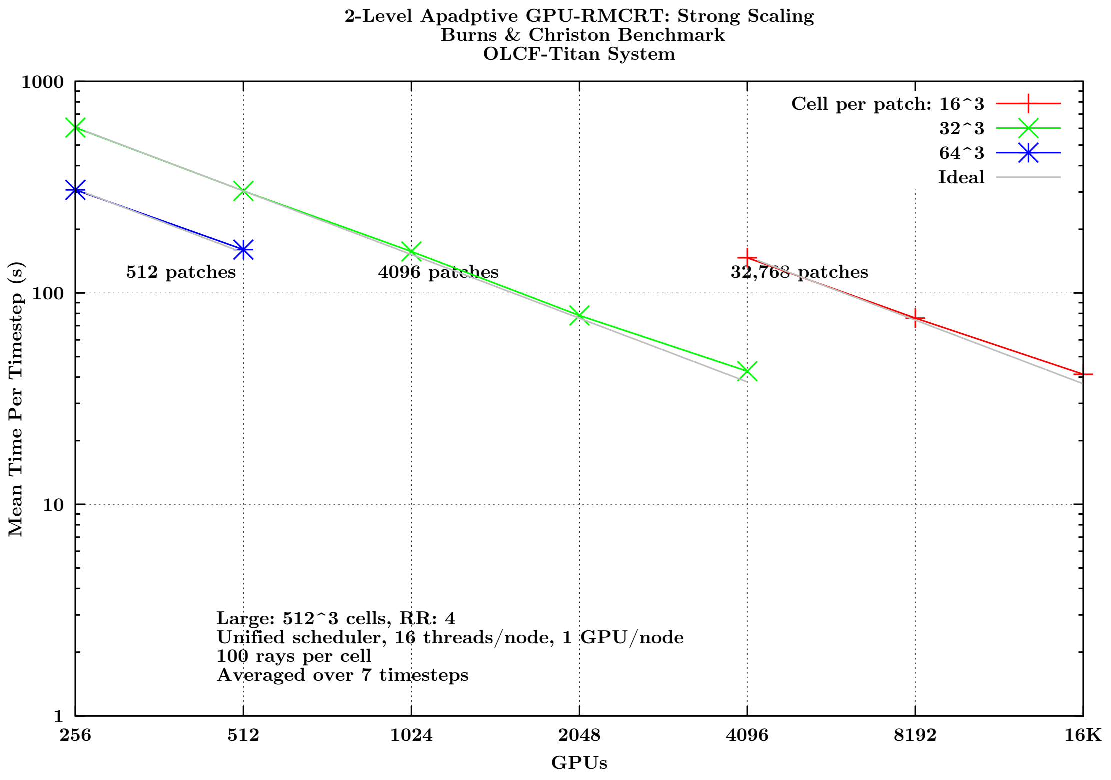

Fig. 3. GPU Strong scaling of the LARGE 2-level benchmark RMCRT problem for 3 patch sizes on the DOE Titan system. Refinement ratio of 4 between levels (RR:4). The fine CFD mesh contains 512^3 cells, coarse radiation mesh contains 128^3 cells. Different patch sizes illustrate GPU speedup.

These results show in general that 1.) using larger patches provides more work per GPU and yields a more significant speedup, 2.) with the improvements to the Uintah infrastructure outlined in this work, we observe excellent scaling through processing multiple patches per GPU and 3.) the algorithm and implementation scales well to 16384 GPUs as a result of the improvements made to Uintah through our work. These results also offer a promising and scalable approach to radiative heat transfer calculations for the CCMSC target boiler problem on current and emerging heterogeneous architectures.

VI. Related Work

Much of the work done toward developing scalable radiation transport models can be found in computational astrophysics and cosmology, involving problems such as neutron star merger, supernova and high energy density plasma. This work is in the context of codes like ARWIN, the AZEuS adaptive mesh refinement, magnetohydrodynamics fluid code, the more general AMR-based FLASH code [32], based on oct-tree meshes and the physics AMR code Enzo, [33]. At the national labs, radiation codes such as RAMSES and PARTISN [34] exist, but are not generally available or target problems like neutron transport, as found in CRASH [35], a block adaptive mesh code for multi-material radiation hydrodynamics. There are also radiation transport problems that use CFD codes and AMR techniques [36], [37], however, a broad range of problems exist that require the concept of tracing rays or particles, such as the simulation of light transport and electromagnetic waves [5]. Much of the available literature on GPU-based Monte Carlo ray tracing approaches to radiation can be found in the Oncology community where GPUs are used for radiation dose calculation [38].

There are overall very few cases of GPU usage at the scale reported here. Gaburov, et al [39] have published results on the evolution of the Milky Way galaxy, a calculation done using 18600 GPUs on DOE Titan. Gray, et al [40] were one of the first to take advantage of at least 8192 GPUs in parallel with their Ludwig soft matter physics application. Many of these reports involve stories concerning code-related issues scaling on Titan and offer detailed discussion on how software teams overcame significant code and algorithmic challenges in porting their applications to GPU-based architectures.

VII. Conclusions and Future Work

We have demonstrated through this work that radiative heat transfer problems can be made to scale within Uintah on current petascale heterogeneous systems through a combination of reverse Monte Carlo ray tracing (RMCRT) techniques combined with adaptive mesh refinement, to reduce the amount of global communication. The results presented here offer a promising approach to modeling radiative heat transfer within Uintah. This approach aims to enable the Utah CCMSC to run the target 1000MWe boiler problem on current and emerging GPU-based architectures at large scale.

Through this work, we have shown the necessity of choosing optimal data structures and algorithms to efficiently expose concurrency. We have also illustrated how maintaining critical sections around serial data structures in legacy code increases code complexity and the likelihood of the introduction of difficult race conditions and deadlock scenarios, especially when using mixed concurrency models, namely *MPI + (Pthreads + Nvidia CUDA)*. Furthermore, we have shown the necessity for frameworks like Uintah to better manage limited memory through the use of custom allocators that allow us to choose better allocation policies for different objects and to better utilize available resources, improving nodal throughput.

The specific contribution in this work is the development of a scalable radiation model for current and emerging heterogeneous architectures, made widely available through the Uintah open-source framework. This contribution may also address related problems with pervasive all-to-all type communications and will be of importance to a broad class of users, developers, scientists and students for whom such problems are presently a bottleneck.

As part of future work, we plan to refactor more algorithms within the Uintah infrastructure to use non-blocking data structures over the mutex-like synchronization primitives. Additionally we will extend the use of our custom memory allocators and trackers to implement ways of tracking memory allocations between scaling runs to identify allocation patterns that do not scale. Work is currently underway to address coprocessor architectures, namely Intel Xeon Phi in preparation for machines like DOE Cori and Aurora. This work will leverage the Kokkos library [41] to achieve performance portability, requiring the extension of the Uintah runtime system to support multi-threaded task execution.

VIII. Acknowledgments

This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We would like to acknowledge Oak Ridge Leadership Computing Facility ALCC awards CMB109, “Large Scale Turbulent Clean Coal Combustion” and CSC188, “Demonstration of the Scalability of Programming Environments By Simulating Multi-Scale Applications” for time on Titan. Additionally, we would like to thank all those involved with Uintah past and present.

References

[1] I. Veljkovic and P. E. Plassmann, “Scalable photon monte carlo algorithms and software for the solution of radiative heat transfer problems,” in *Proceedings of the First International Conference on High Performance Computing and Communications*, ser. HPCC’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 928–937. [Online]. Available: http://dx.doi.org/10.1007/11557654_104

[2] X. Sun and P. J. Smith, “A parametric case study in radiative heat transfer using the reverse monte-carlo ray-tracing with full-spectrum k-distribution method,” *Journal of Heat Transfer*, vol. 132, no. 2, p. 024501, 2010.

[3] I. Hunsaker, T. Harman, J. Thornock, and P. Smith, "Efficient Parallelization of RMCRT for Large Scale LES Combustion Simulations," paper AIAA-2011-3770. 41st AIAA Fluid Dynamics Conference and Exhibit, 2011.

[4] G. Krishnamoorthy, R. Rawat, and P. Smith, "Parallel Computations of Radiative Heat Transfer Using the Discrete Ordinates Method," numerical Heat Transfer, Part B: Fundamentals, 47 (1), 19-38, 2005.

[5] A. Humphrey, T. Harman, M. Berzins, and P. Smith, "A scalable algorithm for radiative heat transfer using reverse monte carlo ray tracing," in *High Performance Computing*, ser. Lecture Notes in Computer Science, J. M. Kunkel and T. Ludwig, Eds. Springer International Publishing, 2015, vol. 9137, pp. 212–230.

[6] A. Humphrey, Q. Meng, M. Berzins, and T. Harman, "Radiation Modeling Using the Uintah Heterogeneous CPU/GPU Runtime System," in *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment (XSEDE 2012)*. ACM, 2012.

[7] Q. Meng, A. Humphrey, and M. Berzins, "The Uintah Framework: A Unified Heterogeneous Task Scheduling and Runtime System," in *Digital Proceedings of Supercomputing 12 - WOLFHPC Workshop*. IEEE, 2012.

[8] Q. Meng, M. Berzins, and J. Schmidt, "Using Hybrid Parallelism to Improve Memory Use in the Uintah Framework," in *Proc. of the 2011 TeraGrid Conference (TG11)*, Salt Lake City, Utah, 2011.

[9] Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins, "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*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 96:1–96:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503250

[10] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[11] M. Berzins, "Status of Release of the Uintah Computational Framework," Scientific Computing and Imaging Institute, Tech. Rep. UUSCI-2012-001, 2012.

[12] Scientific Computing and Imaging Institute, "Uintah Web Page," 2015, http://www.uintah.utah.edu/.

[13] B. Kashiwa and E. Gaffney., "Design basis for cfdlib," Los Alamos National Laboratory, Tech. Rep. LA-UR-03-1295, 2003.

[14] D. Sulsky, S. Zhou, and H. L. Schreyer, "Application of a particle-in-cell method to solid mechanics," *Computer Physics Communications*, vol. 87, pp. 236–252, 1995.

[15] J. E. Guilkey, T. B. Harman, A. Xia, B. A. Kashiwa, and P. A. McMurtry, "An Eulerian-Lagrangian approach for large deformation fluid-structure interaction problems, part 1: Algorithm development," in *Fluid Structure Interaction II*. Cadiz, Spain: WIT Press, 2003.

[16] J.Spinti, J. Thornock, E. Eddings, P. Smith, and A. Sarofim, "Heat transfer to objects in pool fires," in *Transport Phenomena in Fires*. Southampton, U.K.: WIT Press, 2008.

[17] J. Luitjens and M. Berzins, "Improving the performance of Uintah: A large-scale adaptive meshing computational framework," in *Proc. of the 24th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS10)*, 2010.

[18] Q. Meng, J. Luitjens, and M. Berzins, "Dynamic task scheduling for the uintah framework," in *Proceedings of the 3rd IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10)*, 2010.

[19] Q. Meng and M. Berzins, "Scalable large-scale fluid-structure interaction solvers in the Uintah framework via hybrid task-based parallelism algorithms," *Concurrency and Computation: Practice and Experience*, 2013. [Online]. Available: http://dx.doi.org/10.1002/cpe.3099

[20] P. J. Smith, R.Rawat, J. Spinti, S. Kumar, S. Borodai, and A. Violi, "Large eddy simulation of accidental fires using massively parallel computers," in *18th AIAA Computational Fluid Dynamics Conference*, June 2003.

[21] J. Pedel, J. N. Thornock, and P. J. Smith, "Large eddy simulation of pulverized coal jet flame ignition using the direct quadrature method of moments," *Energy & Fuels*, vol. 26, no. 11, pp. 6686–6694, 2012. [Online]. Available: http://dx.doi.org/10.1021/ef3012905

[22] S. Gottlieb, C. Shu, and W. Tadmor, "Strong stability-preserving high-order time discretization methods," *Siam Review*, vol. 43, no. 1, pp. 89–112, 2001.

[23] R. Falgout, J. Jones, and U. Yang, "The design and implementation of hypre, a library of parallel high performance preconditioners," in *Numerical Solution of Partial Differential Equations on Parallel Computers*, ser. Lecture Notes in Computational Science and Engineering, A. Bruaset and A. Tveito, Eds. Springer Berlin Heidelberg, 2006, vol. 51, pp. 267–294. [Online]. Available: http://dx.doi.org/10.1007/3-540-31619-1_8

[24] S. B. Pope, *Turbulent Flows*. Cambridge Press, 2000.

[25] G. Krishnamoorthy, R. Rawat, and P. Smith, "Parallelization of the P-1 Radiation Model," numerical Heat Transfer, Part B: Fundamentals, 49 (1), 1-17, 2006.

[26] C. Clouse, "Parallel deterministic neutron transport with amr," in *Computational Methods in Transport*, ser. Lecture Notes in Computational Science and Engineering, F. Graziani, Ed. Springer Berlin Heidelberg, 2006, vol. 48, pp. 499–512. [Online]. Available: http://dx.doi.org/10.1007/3-540-28125-8_25

[27] K. Viswanath, I. Veljkovic, and P. E. Plassmann, "Parallel load balancing heuristics for radiative heat transfer calculations." in *CSC*, 2006, pp. 151–157.

[28] M. F. Modest, "Backward Monte Carlo Simulations in Radiative Heat Transfer," *Journal of Heat Transfer*, vol. 125, no. 1, pp. 57–62, 2003. [Online]. Available: http://link.aip.org/link/JHTRAO/v125/i1/p57/s1&Agg=doi

[29] B. Hapke, *Theory of Reflectance and Emittance Spectroscopy*. Cambridge University Press, 1993, cambridge Books Online. [Online]. Available: http://dx.doi.org/10.1017/CBO9780511524998

[30] S. P. Burns and M. A. Christen, "Spatial domain-based parallelism in large-scale, participating-media, radiative transport applications," *Numerical Heat Transfer, Part B: Fundamentals*, vol. 31, no. 4, pp. 401–421, 1997.

[31] S. Lee, T. Johnson, and E. Raman, "Feedback directed optimization of tcmalloc," in *Proceedings of the Workshop on Memory Systems Performance and Correctness*, ser. MSPC '14. New York, NY, USA: ACM, 2014, pp. 3:1–3:8. [Online]. Available: http://doi.acm.org/10.1145/2618128.2618131

[32] B.Fryxell, K.Olson, P.Ricker, F.X.Timmes, M.Zingale, D.Q.Lamb, P.Macneice, R.Rosner, J. Rosner, J. Truran, and H.Tufo, "FLASH an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes," *The Astrophysical Journal Supplement Series*, vol. 131, pp. 273–334, November 2000.

[33] B. O'Shea, G. Bryan, J. Bordner, M. Norman, T. Abel, R. Harkness, and A. Kritsuk, "Introducing Enzo, an amr cosmology application," in *Adaptive Mesh Refinement - Theory and Applications*, ser. Lecture Notes in Computational Science and Engineering, vol. 41. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 341–350.

[34] L. Los Alamos National Security, "Los Alamos National Laboratory Transport Packages," 2014, http://www.ccs.lanl.gov/CCS/CCS-4/codes.shtml.

[35] B. van der Holst, G. Toth, I. Sokolov, K. Powell, J. Holloway *et al.*, "Crash: A Block-Adaptive-Mesh Code for Radiative Shock Hydrodynamics - Implementation and Verification," *Astrophys.J.Suppl.*, vol. 194, p. 23, 2011.

[36] J. P. Jessee, W. A. Fiveland, L. H. Howell, P. Colella, and R. B. Pember, "An adaptive mesh refinement algorithm for the radiative transport equation," *Journal of Computational Physics*, vol. 139, no. 2, pp. 380–398, 1998.

[37] M. Pernice and B. Philip, "Solution of equilibrium radiation diffusion problems using implicit adaptive mesh refinement," *SIAM J. Sci. Comput.*, vol. 27, no. 5, pp. 1709–1726, 2005.

[38] R. W. Townson, X. Jia, Z. Tian, Y. J. Graves, S. Zavgorodni, and S. B. Jiang, "Gpu-based monte carlo radiotherapy dose calculation using phase-space sources," *Physics in Medicine and Biology*, vol. 58, no. 12, p. 4341, 2013. [Online]. Available: http://stacks.iop.org/0031-9155/58/i=12/a=4341

[39] J. Bédorf, E. Gaburov, M. S. Fujii, K. Nitadori, T. Ishiyama, and S. P. Zwart, "24.77 pflops on a gravitational tree-code to simulate the milky way galaxy with 18600 gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 54–65. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.10

[40] A. Gray and K. Stratford, *Ludwig: multiple GPUs for a complex fluid lattice Boltzmann application*. Chapman and Hall/CRC, 2013.

[41] H. C. Edwards and D. Sunderland, "Kokkos array performance-portable manycore programming model," in *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, ser. PMAM '12. New York, NY, USA: ACM, 2012, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/2141702.2141703