Efficient Parallelization of RMCRT for Large Scale LES Combustion Simulations

Isaac Hunsaker*, Todd Harman†, Jeremy Thornock‡, Philip J. Smith§
University of Utah, Salt Lake City, UT 84112, USA

At the high temperatures inherent to combustion systems, radiation is the dominant mode of heat transfer. An accurate simulation of a combustor therefore requires precise treatment of radiative heat transfer. This is accomplished by calculating the radiative-flux divergence at each cell of the discretized domain. Reverse Monte Carlo Ray Tracing (RMCRT) is one of the few numerical techniques that can accurately solve for the radiative-flux divergence while accounting, in an efficient manner, for the effects of participating media. Furthermore, RMCRT lends itself to massive parallelism because the intensities of each ray are mutually exclusive. Therefore, multiple rays can be traced simultaneously at any given time step. We have created a parallelized RMCRT algorithm that solves for the radiative-flux divergence in combustion systems. This algorithm has been verified against a 3D benchmark case involving participating media. The error of this algorithm converges with an increase in the number of rays traced per cell, such that at 700 rays per cell, the L2 error norm of a 41^3 mesh is 0.49%. Our algorithm demonstrates strong scaling when run in parallel on 2 to 1536 processors for domains of 128^3 and 256^3 cells.

Nomenclature

$\hat{s}$	=	Direction vector
κ	=	Absorption coefficient
Ω	=	Solid angle
Φ	=	Scattering phase function
σ	=	Stefan-Boltzmann constant
σ_s	=	Scattering coefficient
G	=	Incident radiation function
I_{in}	=	Incoming intensity
I_{out}	=	Outgoing Intensity
N	=	Number of rays traced per cell
N_p	=	Number of processors used in a parallel simulation
N_x	=	Number of cells in the x direction
q	=	Radiative flux

I. Introduction

Historically, RMCRT has been solved on relatively coarse grids, which allows the entire domain to fit within the memory constraints of the processors of a cluster. As the demand for more refined meshes has grown, so has the demand to adequately resolve the radiative-flux divergence on these meshes. One method to handle these large domains involves decomposing them into smaller subdomains, called patches. Each processor is then passed a single patch. In combustion systems, this domain decomposition method has been very successful because the physical phenomena governing computational fluid dynamics is local by nature. Typical finite volume schemes use stencils that span only a few cells. Therefore the only cells that require values from outside patches are the few cells at or very near the patch boundaries. This means that a minimal amount of information is passed between patches, resulting in efficient parallelization. Radiation, conversely, is not a local phenomenon. The net radiative-heat flux at any given location within the domain is a function of all other fluxes throughout the domain[1]. This requires that the rays of RMCRT be allowed to traverse not just the patch, but the entire domain. We have designed a system that allows the

*PhD Candidate, Department of Chemical Engineering.
†Research Assistant Professor, Department of Mechanical Engineering.
‡Research Assistant Professor, Department of Chemical Engineering.
§Professor, Chair, Institue for Clean and Secure Energy.

processors to be passed information regarding the radiative properties of the entire domain, while only requiring each processor to trace rays from within their local patch. In this manner, we avoid the passing of rays between processors, and achieve strong scaling of our algorithm.

II. Discussion

A. Background

The Monte Carlo method was developed by Enrico Fermi, John von Neumann, and Nicholas Metropolis for the Manhattan Project during World War II[2]. The method modeled the behavior of neutrons and involved following the histories of these neutrons during fission. In the mid 1960's Howell and Perlmutter applied this method to thermal radiation heat transfer[3–5]. The method was later borrowed and greatly enhanced by the computer graphics community[6–8]. Since its inception, the Monte Carlo technique has been widely applied to practical problems with participating media[5]. The Monte Carlo method can handle the modeling of radiation within absorbing, emitting, and scattering media with relative ease, whereas most other methods handle them inefficiently at best[1]. Furthermore, the Monte Carlo method is the only method to date that can satisfactorily deal with the effects of irregular radiative properties such as nonideal directional or nongray behavior[5].

The Monte Carlo method is not without its drawbacks. Because it is a statistical technique, the variance is inversely proportional to the number of rays sampled. The standard deviation is therefore a function of the square root of the number of samples, resulting in slow convergence rates[5]. The Monte Carlo method is also computationally expensive when run in serial on a single processor. However, because of the uniqueness of the solutions to each of the rays, RMCRT is amenable to massive parallelization. This attribute may outweigh the low serial efficiency, especially in today's era of cloud computing.

B. The Monte Carlo Method for Participating Media

In general, the end result of radiation calculations in combustion systems is the coupling of the radiative-flux divergence to the energy equation. The radiative-flux divergence can be obtained from the radiative transport equation to yield

$$\nabla \cdot q = \kappa(4\pi I_{b,out} - \int_{4\pi} I_{in} d\Omega). \tag{1}$$

Because $d\Omega$ represents the solid angle of a ray, each ray has a solid angle of $\frac{4\pi}{N}$. Given the temperature and absorption coefficient of a cell of interest, one can readily compute the first term of the above equation, as $I_{b,out}$ is simply as follows

$$I_{b,out} = \frac{\sigma T^4}{\pi}. \tag{2}$$

The second term of Eq. (1) is not so readily obtained, and for this solution method, requires the tracing of numerous rays into the domain, then integrating back to the origin the contributions of the intensities of the cells along the path. This process takes place as follows.

To compute the second term of Eq. (1) for a given cell, we trace N number of rays, where N is statistically large enough to sample the entire domain. Randomly distributed throughout the cell, the rays begins at unique origins. For the determination of each origin, three random numbers (one for each Cartesian direction) are generated. The direction of each ray is determined by selecting two random numbers such that the probability distribution function of these directions is spherically distributed. For each ray, we therefore require a minimum of five random numbers. Then, for a given timestep in which the radiative flux divergenc is calculated, we require a minimum of $Nx \times Ny \times Nz \times N \times 5$ rays. With this magnitude of random numbers being pulled, a fast, effective random number generator is imperative. Section D describes the random number generator of choice for this application. Once a location and a direction have been selected, subsequent intersections and other physical phenomena can be calculated, and the ray may be traced through the domain until a sufficient fraction of the intensity has been attenuated. For an emitting, absorbing medium, intensity is picked up from the current cell in the path and traced back, with exponential attenuation according to Beer's Law, to the origin. The contributions from each of the N rays weighted by their corresponding solid angles are then used in the calculation of the flux divergence. For forward ray tracing, the number of rays traced will be a

function of the emissive power of the cell of origin. In reverse ray tracing, however, determining N for sufficient rays is not so intuitive because the flux divergence of the origin cell is dependent on the radiation of all other cells, whose emissive power may not be known a priori. Therefore, perhaps the best technique to determine N, is to use the variance of previous results[5,9].

C. Parallel Efficiency considerations

In the 1950s and 60s, users interacted more directly with low level computer algorithms. Because of the complexities that would be inherent to parallelizing algorithms, computer hardware was designed to handle code in a linear fashion. Today, however, there are several layers of computer languages separating the user from the low level algorithms, giving the user more abstraction. This allows the user to handle more complex algorithms such as code parallelization[10]. When this approach is implemented, the user can, in theory, attain a speedup that is proportional to the number of processors used. Ideal speedup occurs when the time spent passing information between processors is negligible compared to the work done by each processor, and if no computers sit idle while others complete their tasks. The former constraint is met by efficient code writing that ensures that all or most of the information a given processor needs to complete its computations is available to that processor. The latter constraint is met by proper load balancing to distribute the work load equally between processors.

The first attempts to parallelize Monte Carlo codes did so on single-instruction multiple-data stream (SIMD) machines. Vectors or groups of rays were distributed to the processors. This however, led to poor scalability as it necessitated the termination of all rays before generating a subsequent group. More recent algorithms, ours included, generate new rays at the onset of termination of a prior ray to avoid creating idle time amid processors[11].

RMCRT lends itself to massive parallelism because each ray's intensity is independent of every other ray's intensity, so multiple rays can be traced simultaneously at any given time step. In theory, one could simultaneously trace as many rays as one has processors or cores. This takes advantage of the existing framework common to parallelized LES codes such as the ARCHES code, which was developed by CRSIM and runs on the Uintah framework developed for the Center for the Simulation of Accidental Fires and Explosions[12–14]. In our modified patch domain decomposition, N number of rays are traced from each cell within the patch and allowed to traverse the entire domain until extinction. In this manner, each processor is responsible for solving the radiative-flux divergence of only the cells within its corresponding patch.

D. Sequential Efficiency Considerations

An efficient parallel ray tracing algorithm begins with an efficient sequential ray tracing algorithm. Before introducing any algorithm into a parallel scheme, it is best to optimize the code in serial mode. We therefore present several useful speedups for a ray tracing algorithm. At the heart of RMCRT is the algorithm that calculates the intersections between a ray and surfaces. This process is the most time consuming portion of the ray tracing algorithm, so care must be taken to abstain from any practice that would compromise its efficiency. In participating media, we are concerned not only with interactions of the domain boundaries, but also within the media. For non-homogeneous, emitting, absorbing media, we must know the absorption coefficient and temperature of every cell along the path of every ray. We therefore must know the cells through which a ray will pass. This is accomplished by implementing a ray marching algorithm similar to that outlined by Woo and Amanitides[15]. This ray marching algorithm efficiently handles the marching of the rays through the domain by minimizing the number of comparisons made, and by taking advantage of trigonometric and algebraic concepts.

Attention to detail is paramount in maintaining efficient code. Even simple operations such as function calls, if handled improperly, can lead to drastic reductions in efficiency. The elimination of all function calls is one way to avoid this trap, but this leads to other problems such as poor user readability. Two alternatives that preserve readability without sacrificing speed include passing by reference, and the use of inline functions. Another simple detail is the handling of loops. For instance, one may consider removing a statement out of a "for" loop that repeats fewer than 4 times. This negates the necessity of incrementing an index for each iteration. However insignificant these details may seem, neglecting opportunities for efficiency within deeply nested "for" loops can take a toll, especially when these operations occur on the order of 10^{12} times, as is the case in RMCRT.

A good random number generator is paramount to the efficiency to any Monte Carlo algorithm. To be effective, a random number generator must have an unbiased uniform distribution, low repeatability, and of course, fast execution time. Mersenne Twister, a pseudo random number generator developed by Matsumoto and Nishimura, guarantees a

non-repeating sequence of approximately $2^{20,000}$ [16]. We tested the Mersenne Twister algorithm on its ability to rapidly produce large samples of random numbers and demonstrate variance convergence.

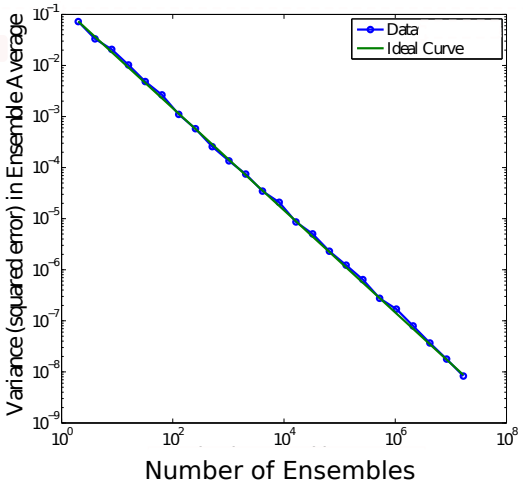

Figure 1. Variance convergence of Mersenne Twister. Each data point represents 100 random numbers summed to the previous data point.

Figure 1 demonstrates the approximately linear convergence of the variance of large samples of random numbers generated by Mersenne Twister. Each ensemble represents the variance of 100 random numbers, distributed between 0 and 1. The expected value of the ensemble is 0.5, and a variance can be readily calculated. Figure 1 represents the variance reduction as the variance of the sum of all previous ensembles reduces as the number of ensembles increases. As an example, the final data point representing 5×10^7 ensembles is a collection of 5×10^9 random numbers, and the variance of the collection of these random numbers is approximately 10^{-6}. Five billion samples is of the same order of magnitude as the number of random numbers required to run an RMCRT simulation on a 100^3 mesh using 1,000 rays per cell. We have observed that Mersenne Twister generates approximately 14 million random numbers per second when run on a single core of a 2.3 GHz processor. With its effectiveness and speed, Mersenne Twister has met our requirements for the RMCRT application.

Another sequential efficiency consideration is weighing the computational time for a given number of rays against the corresponding error inherent to that sample. In theory, the computational effort should scale approximately linearly with an increase in the number of rays. The variance of the data should also scale linearly, yet the error, the square root of variance, will decrease only as $N^{-0.5}$. Therefore, the increase in accuracy will come at a disproportionate computational price. Naively increasing accuracy beyond a reasonable metric by increasing the number of rays will result in poor efficiency.

E. Results

The described method of RMCRT was applied to a 3D benchmark case as outlined by Burns and Christon [17]. In this case, there is a known solution of the radiative-flux divergence along the centerlines of the domain. The domain contains a non-homogeneous absorbing, emitting medium, bounded by cold, black walls. The domain was discretized into 41^3 cells, from which 700 rays per cell were traced. After computing the radiative-flux divergence of each of the cells, a two dimensional slice of the domain was visualized as shown in Figure 2. The values at the centerline of the above figure were compared with the exact solution to yield the results as shown in Figure 3.

The L2 error norm of the data was calculated to be 0.49%. As demonstrated by Figure 3 and the low value of the L2 norm, RMCRT at 700 rays per cell produces results that agree well with the exact solution. To verify that the convergence of the error occurs at a rate proportional to $N^{-0.5}$ a series of simulations of the same benchmark case were run, varying the number of rays per cell from 8 to 1024. As expected, the log of the L2 norm plotted against the log of the number of rays produces a near-linear curve with a slope of $-\frac{1}{2}$ as indicated in Figure 4.

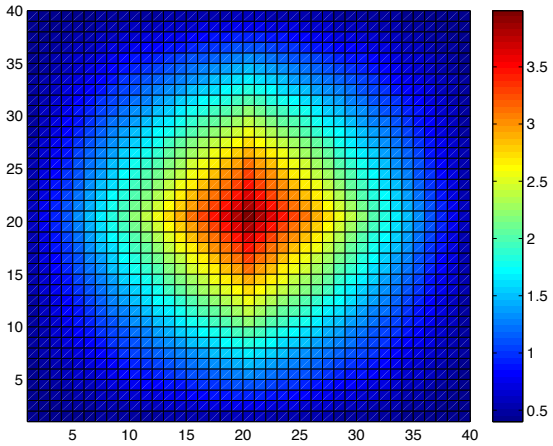

Figure 2. Radiative-flux divergence of a benchmark case as solved by RMCRT. Units of W/m^3.

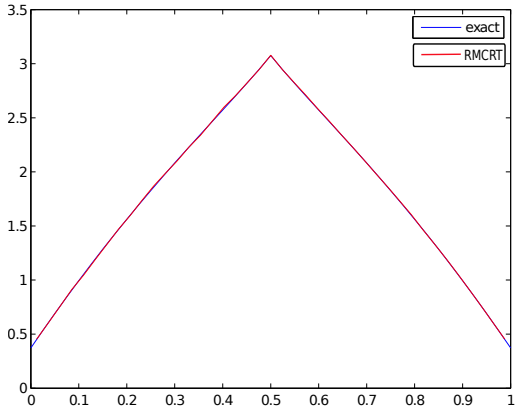

Figure 3. Exact solution of the flux divergence (blue) vs RMCRT (red) solution along the center line. Units of W/m^3.

Once the RMCRT algorithm had been verified on a single processor, we began testing its ability to perform in parallel. These studies were performed on a single level, i.e. the grid resolution was the same for each patch, including the patch from which a given processor would initiate rays. These studies were carried out with 25 and 100 rays per cell (see Figure 5) on a domains of 128^3 and 256^3 cells (see Figure 6). Strong scaling analysis was performed by varying the number of processors from 1 to 1536, and observing the total computational time. Figure 5 demonstrates the strong scaling of RMCRT with a fixed domain size of 128^3 with two cases: 25 rays per cell, and 100 rays per cell. The scaling closely matches the ideal curve, but begins to digress at high levels of parallelism. Figure 6 demonstrates the strong scaling with a fixed number of rays per cell for two cases: a domain of 128^3 cells and 256^3 cells. For the 128^3 case, strong scaling closely matched the ideal curve, but digressed as the number of processors increased above 384. For the 256^3 case, strong scaling was observed up to 768 processors. At 1536 processors, the simulation time actually increased above that of the 768 processor case. This is most likely due to an unbalanced work load, as some processors are handed patches very near the domain boundaries, and therefore trace rays only a short distance before they are terminated at the cold, black walls. These processors would then sit idle, while the other processors trace rays from locations farther from the walls, and therefore require more time to complete their computations.

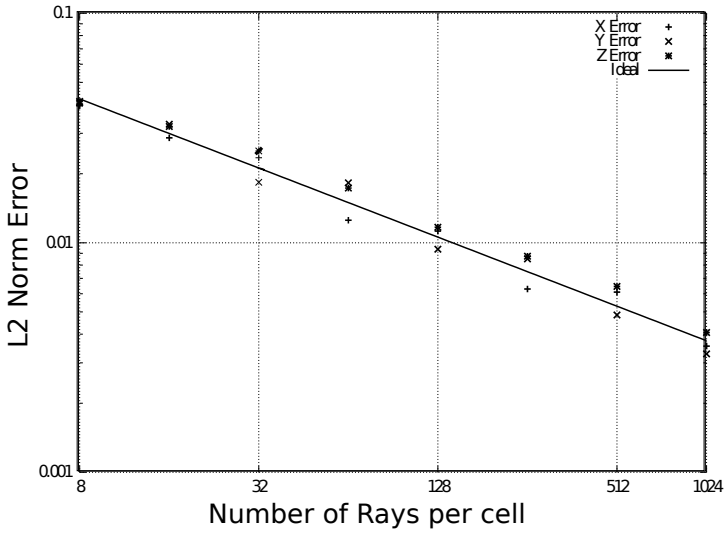

Figure 4. Convergence of the L2 error norm in the x,y, and z directions along a centerslice of the 3D domain. The ideal curve represents the -1/2 order scaling that is predicted by theory.

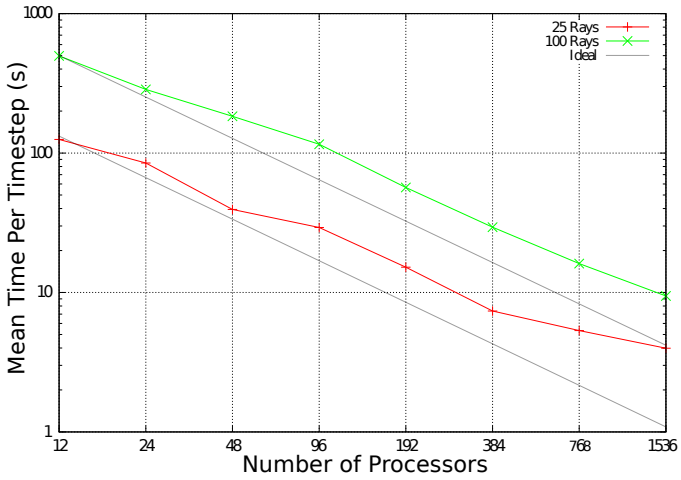

Figure 5. Strong scaling analysis of RMCRT on 12 to 1536 processors using 25 rays per cell(red) and 100 rays per cell (green). Domain size in both cases was 128^3.

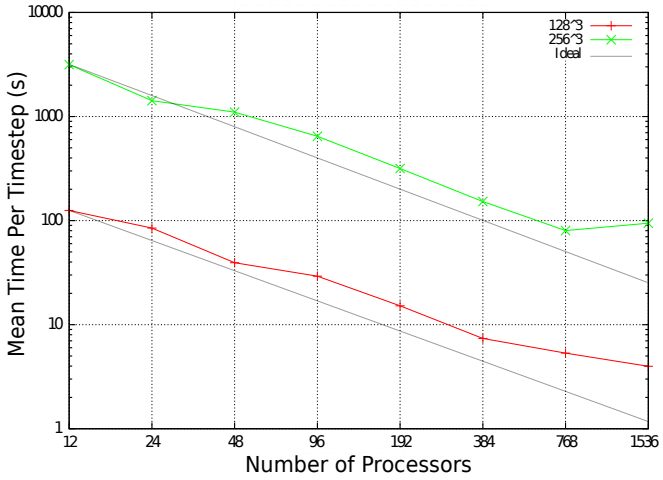

Figure 6. Strong scaling analysis of RMCRT on 12 to 1536 processors using 25 rays per cell on domains of 128^3 (red) and 256^3 (green).

III. Conclusions

Due to its high level of accuracy and its amenability to complex physics and parallelization, RMCRT is well suited to handle large scale, high resolution radiative transport calculations. For mesh resolutions that contain a large number of cells, to obtain results in a timely fashion, some form of parallelism becomes imperative. We have demonstrated that patch domain decomposition is a suitable form of parallelism for RMCRT, so long as each processor is given radiative information of the entire domain.

At present, further strong and weak scaling analyses are being performed to verify the scalability of RMCRT for larger domains. Memory constraints begin to restrict the size of the domain at approximately 300^3 cells. Future work will include investigation into a potential solution to this problem involving a composite mesh that allows each processor to be handed a fully resolved version of a subset of the domain and a coarsened version of the remainder of the domain (See Figure 7). We are optimistic that this technique will allow for radiation calculations on domains containing hundreds of millions of cells.

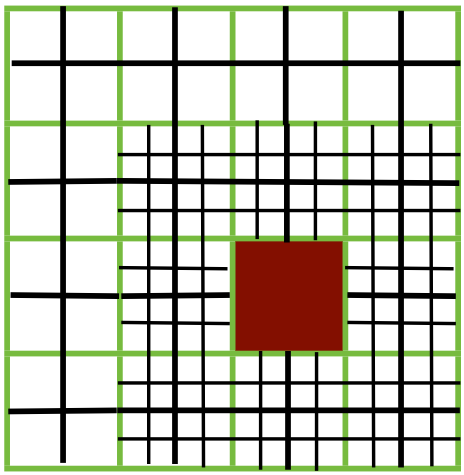

Figure 7. Composite mesh for parallelization of RMCRT. At the patch level, the processor is handed information on the finest level (red), a coarsened version of the information for proximal patches (black), and the coarsest information for distal patches (green).

Acknowledgments

The authors would like to thank Steven Parker, Charles Reid, Tony Saad, Sean Smith, and James Sutherland for their many contributions to the numerical work. This research was sponsored by the National Nuclear Security Administration under the Advanced Simulation and Computing program through DOE Research Grant #DE-NA0000740.

References

[1]Modest, M., *Radiative heat transfer*, Academic Pr, 2003.

[2]Sun, X., "Reverse Monte Carlo ray-tracing for radiative heat transfer in combustion systems," 2009.

[3]Howell, J. and Perlmutter, M., "Radiant Transfer Through a Gray Gas Between Concentric Cylinders Using Monte Carlo," *Journal of Heat Transfer*, Vol. 86, 1964, pp. 169–179.

[4]Howell, J. and Perlmutter, M., "Monte Carlo solution of thermal transfer in a nongrey nonisothermal gas with temperature dependent properties," *AlChE Jouranl*, Vol. 10, No. 4, 1964, pp. 562–567.

[5]Howell, J., "The Monte Carlo method in radiative heat transfer," *TRANSACTIONS-AMERICAN SOCIETY OF MECHANICAL ENGINEERS JOURNAL OF HEAT TRANSFER*, Vol. 120, 1998, pp. 547–560.

[6]Lee, M., Redner, R., and Uselton, S., "Statistically optimized sampling for distributed ray tracing," *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM, 1985, pp. 61–68.

[7]Pegoraro, V., Wald, I., and Parker, S., "Sequential Monte Carlo Adaptation in Low-Anisotropy Participating Media," *Computer Graphics Forum*, Vol. 27, Wiley Online Library, 2008, pp. 1097–1104.

[8]Pegoraro, V., Brownlee, C., Shirley, P., and Parker, S., "Towards interactive global illumination effects via sequential Monte Carlo adaptation," *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, IEEE, 2008, pp. 107–114.

[9]Almazan, P., "Accuracy control in Monte Carlo radiative calculations," *In NASA. Lewis Research Center, The Fifth Annual Thermal and Fluids Analysis Workshop p 47-62 (SEE N94-23634 06-34)*, Vol. 1, 1993, pp. 47–62.

[10]Almasi, G., "Research in highly parallel computer systems," *IEEE Electro Technology Review*, Vol. 2, 1986.

[11]Govaerts, Y. and Verstraete, M., "Raytran: A Monte Carlo ray-tracing model to compute light scattering in three-dimensional heterogeneous media," *Geoscience and Remote Sensing, IEEE Transactions on*, Vol. 36, No. 2, 2002, pp. 493–505.

[12]Rawat, R., Parker, S., Smith, P., and Johnson, C., "Parallelization and integration of fire simulations in the uintah pse," *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, pp. 12–14.

[13]Luitjens, J., Worthen, B., Berzins, M., and Henderson, T., "Scalable parallel AMR for the Uintah multiphysics code," *Petascale Computing Algorithms and Applications. Chapman and Hall/CRC*, 2007.

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

[15]Amanatides, J. and Woo, A., "A fast voxel traversal algorithm for ray tracing," *Eurographics*, Vol. 87, Citeseer, 1987, p. 10.

[16]Matsumoto, M. and Nishimura, T., "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 8, No. 1, 1998, pp. 3–30.

[17]Burns, S. and Christon, M., "Spatial domain-based parallelism in large-scale, participating-media, radiative transport applications," *Numerical Heat Transfer, Part B: Fundamentals*, Vol. 31, No. 4, 1997, pp. 401–421.