EXTENDING THE UINTAH FRAMEWORK THROUGH THE PETASCALE MODELING OF DETONATION IN ARRAYS OF HIGH EXPLOSIVE DEVICES

MARTIN BERZINS*, JACQUELINE BECKVERMIT†, TODD HARMAN ‡, ANDREW BEZDJIAN †, ALAN HUMPHREY *, QINGYU MENG §, JOHN SCHMIDT *, AND CHARLES WIGHT ¶

Abstract.
The Uintah software framework for the solution of a broad class of fluid-structure interaction problems has been developed by using a problem-driven approach that dates back to its inception. Uintah uses a layered task-graph approach that decouples the problem specification as a set of tasks from the adaptive runtime system that executes these tasks. Using this approach it is possible to improve the performance of the software components to enable the solution of broad classes of problems as well as the driving problem itself. This process is illustrated by a motivating problem that is the computational modeling of the hazards posed by thousands of explosive devices during a Deflagration to Detonation Transition (DDT) that occurred on Highway 6 in Utah. In order to solve this complex fluid-structure interaction problem at the required scale, substantial algorithmic and data structure improvements were needed to Uintah. These improvements enabled scalable runs for the target problem and provided the capability to model the transition to detonation. The solution to the target problem from these runs provided insight as to why the detonation happened, as well as demonstrating a possible remediation strategy that may have avoided detonation.

Key words. Uintah, software,detonation, scalability, parallel, adaptive, petascale

1. Introduction.

The move to multi-petaflop and eventually exascale computing over the next decade is seen as requiring changes in both the type of programs that written and how those programs will make use of novel computer architectures in order to perform large-scale simulations successfully. One approach that is seen as a candidate for successful codes at such scales uses a directed graph model of the computation to schedule work adaptively and asynchronously. The potential value of this methodology is expressed by [25, 16] *Exascale programming will require prioritization of critical-path and non-critical path tasks, adaptive directed acyclic graph scheduling of critical-path tasks, and adaptive rebalancing of all tasks with the freedom of not putting the rebalancing of non-critical tasks on the path itself.* Given such statements it is important to understand the value of this approach as used, for example, in the Uintah framework [35] when applied to challenging large-scale computational fluid-structure interaction problems. The development of the Uintah framework has, since its very inception been driven by such problems. This is possible as Uintah's task-graph based approach provides a clean separation between the problem specifications that defines the tasks and the runtime system that executes the tasks. Improvements to the runtime system thus have a potential impact on all Uintah applications.

The aim in this paper is to illustrate this application-driven process and to show that achieving scalable real world science and engineering calculations requires three steps. The first step is to develop a prototypical application code that exercises the kernel calculations of the algorithm and framework and the second step is to use this prototypical code to exercise the software in ways similar to the full application so as to expose algorithmic and data structure deficiencies in both the computational and communication methods. The third step is to run the full application itself. The motivating problem considered here is a hazard modeling problem involving energetic materials that resulted in a potentially catastrophic event on Highway 6 in Utah in 2005 when a truck carrying 36,000 pounds of seismic boosters

[1]SCI Institute, University of Utah, Salt Lake City UT 84112 , USA
[2]Department of Chemistry, University of Utah, Salt Lake City, UT, USA
[3]Department of Mechanical Engineering, University of Utah, Salt Lake City, UT, USA
[4]Google Inc
[5]Office of the President, Weber State University, Ogden, UT, USA

overturned, caught fire, and within minutes detonated, creating a crater 70 feet wide by 30 feet deep.

Energetic materials may be classified as propellants, pyrotechnics or explosives. The most prominent characteristic of these materials is the rate at which they can release energy, ranging from relatively slow and benign reactions to extremely fast and violent ones. Specifically, the slow rate of combustion (deflagration) is characterized by wave speeds of 10s-100s m/s while a detonation combustion front moves at 1000s m/s. These modes have been studied for single monolithic devices and are relatively well understood. What is less known, and is the focus of our research, is the cause of a Deflagration to Detonation Transition (DDT) in large arrays of small energetic devices. These arrays are used in the mining industry and are often being transported on highways. The open question is whether or not the explosives could have been transported in a safer manner so as to prevent a detonation. The goal of this research is to understand how DDT of multiple arrays of explosives can occur and to use computational models to help formulate packaging configurations to suppress it. In order to address these questions a DDT model has been developed in this work that has shown great promise in simulating reactive fluid-structure interactions. In parallel with this development the underlying software framework has been extended from a starting point of scalability on DOEs Titan [34] and Mira [36] to the combination of fluid-structure interaction and adaptive mesh models needed for this broad class of problems. One challenge in undertaking this extension is that algorithms that may have had hidden and potentially problematic dependencies with small constants at large core counts may only become visible at close to full machine capacity. In order to address these problems required a fundamental rewrite of many of the algorithms and data structures in Uintah so as to improve their efficiency.

Th main contribution of this paper is to show that after introducing these new, more efficient algorithms and data structures it was possible to demonstrate scalability on 700K cores on DOE's Mira and NSF's Blue Waters for a realistic model problem and to 512K cores on DOE's Mira for a real world, complex fluid-structure interaction problem that models DDT in a large array of explosives. This process is described as follows. In Section 2 the Uintah framework and its unique runtime system is described in outline. A discussion of the Uintah problem class and of the DDT modeling of a large array of explosive cylinders is presented in Section 3. Section 4 describes the main novel technical contributions of the paper with regard to the scalability challenges faced and the new algorithms and data structures introduced to achieve a scalable simulation. These improvements include a better AMR algorithm for large scale fluid-structure problems by improving the methods used in particle creation, task-graph compilation, load balancing and data copying after remeshing. In Section 5, the benefits of these contributions are shown through scalability and performance results at scales up to 700K cores. In Section 6 the principal computational science contribution is shown, namely that in the computational experiments with four DDT cases, developing detonations are demonstrated, as is the result that packing the explosives differently appears to make it much more difficult for detonation to occur. Section 7 describes related work using other similar computational frameworks.

Our conclusion in Section 8 is that these improvements have made it possible to model the detonation calculation at realistic scales. The results from this model have shown that detonation does occur in a prototypical simulation and that it looks likely that storing the explosives differently would have helped prevent detonation. Given the frequency with which explosives are transported by road, this is a potentially important result. In summary the Uintah adaptive DAG-based approach provides a very powerful abstraction for solving challenging multi-scale multi-physics engineering problems on some of the largest and most powerful computers available today.

2. Uintah Infrastructure.

The Uintah software framework originated in the University of Utah DOE Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [15, 40, 39]. Uintah has since been used to solve a variety of challenging fluid, solid, and fluid-structure interaction problems from a variety of domains described in [9], such as angiogenesis, tissue engineering, green urban modeling, blast-wave simulation, semi-conductor design and multi-scale materials research. Uintah's open-source license (MIT License) encourages community development and contributions from a variety of discplines for both non-commercial and commercial areas.

The Uintah framework is based on the fundamental idea of structuring applications drivers and applications packages as a Directed Acyclic Graph (DAG) of computational tasks, belonging to Uintah components that access local and global data from a *data warehouse* that is part of an MPI process and that deals with the details of communication. A runtime system manages the asynchronous and out-of-order (where appropriate) execution of these tasks and addresses the complexities of (global) MPI and (per node) thread based communication. Each Uintah component implements the algorithms necessary to solve partial differential equations (p.d.e.s) on structured adaptive mesh refinement (SAMR) grids. The runtime system provides a mechanism for integrating multiple simulation components and after analyzing the dependencies and communication patterns between these components efficiently executes the resulting multi-physics simulation. Four primary components have been developed and include: 1) a low and high-speed compressible flow solver, ICE [26]; 2) a Material Point Method algorithm, MPM [47] for structural mechanics; 3) a fluid-structure interaction (FSI) algorithm, MPMICE which combines the ICE and MPM components [23, 24]; and 4) a turbulent reacting CFD component, ARCHES [44] designed for simulation of turbulent reacting flows with participating media radiation.

Uintah components allow the developer to focus solely on developing the tasks for solving the partial differential equations on a local set of block structured meshes without using any specific MPI calls [40]. Such components are composed of C++ classes that follow a simple API to establish connections with other components in the system. The component itself is expressed as a sequence of tasks where data dependencies (inputs and outputs) are explicitly specified by the developer. For example in a standard p.d.e. stencil computation the component specifies the number of ghost-cell layers that are needed by a task. The tasks along with their data dependencies are then compiled into a DAG representation to express the parallel computation along with the underlying communications to satisfy the data needed by those tasks. The smallest unit of parallel work is a patch composed of a hexahedral cube of grid cells. Each task has a C++ method for the actual computation and each component specifies a list of tasks to be performed and the data dependencies between them [10]. Uintah executes these tasks in parallel by using a runtime system that is largely independent of the actual application itself. Each task is only ready for execution when the data it needs has arrived as a result of the automated communications system managed by the runtime system. As a single data warehouse per multi-core node is used to provide access to local variables and non-local variables through automatically instantiated MPI communications, the task itself only has to acquire the data from a local data warehouse on each compute node. The task then puts its results into a new data warehouse for the next time step. The new data warehouse at the end of one step becomes the old data warehouse for the next time step.

This division of labor between the application code and the runtime system allows the developers of the underlying parallel infrastructure to focus on scalability concerns such as load balancing, task (component) scheduling, communications, including accelerator or co-processor interaction. In addition, I/O is handled at this level with a design that facilitates the incorporation of efficient libraries such as PIDX [29].

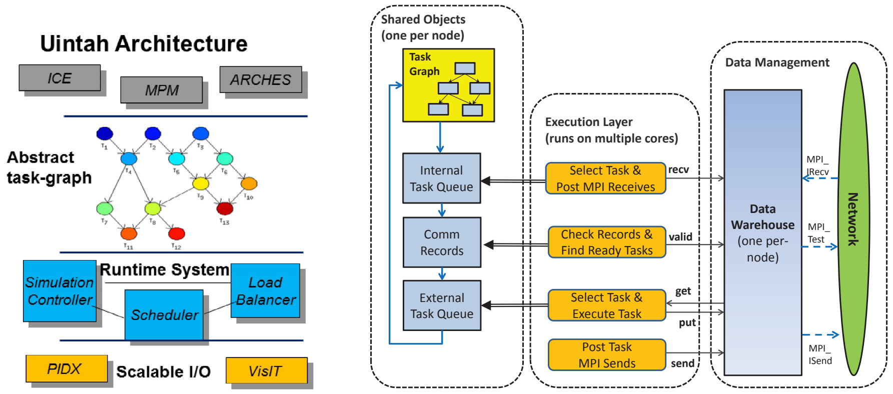

FIG. 1. *(a) Uintah Architecture and (b) Uintah Nodal Runtime System*

The separation between the application physiscs code and the runtime system is illustrated by Figure 1(a). The runtime system that is used on each compute node is shown in Figure 1(b). Overall this structure generally permits advances in the runtime system, such as scalability, to be immediately applied to applications without any additional work by the component developer. The nodal component of the runtime system has an execution layer that runs on each core and that queries the nodal data structures in order to find tasks to execute.

Each mesh patch that is executed on a node uses a local task graph that is composed of the algorithmic steps (tasks) that are stored along with various queues that determine which task is ready to run. Data management including the movement of data between nodes together with the storage of data in the data warehouse occurs on a per-node basis. The execution of the various tasks is distributed on a per-core level. Communications between the task queues, the tasks themselves and the data warehouse occur on a nodal level and are shown in Figure 1(b).

While this separation of concerns and indeed even some user-code has been unchanged since the first releases of Uintah, as computer systems have grown in complexity and scale, the runtime system has been substantially rewritten several times [11] to ensure continued scalability on the largest DOE, NSF and DOD computer systems available to Uintah. This scalability is achieved through several novel features in the code. The Uintah software makes use of scalable adaptive mesh refinement [32, 33, 31] and a novel load balancing approach [30], which improves on other cost models. While Uintah uses a DAG approach for task scheduling, the use of dynamic out-of-order task execution is important in improving scalability [36]. For systems with reduced memory per core, only one MPI process and only one data warehouse per node are used. Threads are used for task execution on individual cores. This has led to much improved memory use and better scalability [34]. Additional details surrounding Uintah's runtime system can be found in [36].

Even with this successful approach, however, the applications developer must still write code that ensures that both the computational and communications costs are sufficiently well-balanced, in order to achieve scalability. In the case where scaling is not achieved, Uintah's detailed monitoring system is often able to identify the source of the inefficiency, as will be illustrated in Sections 4.1 and 4.2.1 below.

Overall Uintah scales well on a variety of machines including those with Intel or AMD processors and Infiniband interconnects such as Stampede, the Cray machines such as Titan and Blue Waters and the Blue Gene/Q machines like Mira, [36]. Parts of Uintah also run on

GPU and Xeon Phi machines at present as part of an ongoing development activity. The separate runtime system that is clearly differentiated from the main component code allowed us to identify shortcomings at this level, (see Section 4) and so allowed us to improve the problematic algorithms thus resulting in better scalability (see Section 5) at the largest problem sizes and core counts without changing any applications code.

3. Target Scenario and Modeling a DDT. When modeling DDT in solid explosives there are three modes of combustion to consider, conductive deflagration, convective deflagration and detonation. Conductive deflagration occurs on the surface of the explosive material at low pressures and has a relatively slow flame propagation (on the order of a few cm/sec [45]). To model conductive deflagration, Uintah has adopted the WSB burn model [49] which has been validated over a wide range of pressures, temperatures and grid resolutions against experimental data [42, 41]. The WSB model is a global kinetics burn model which allows exothermic reactions to be represented at the macro-scale, enabling the use of coarser grid resolutions without the loss of fidelity. This is essential when trying to simulate problems requiring large physical domains.

Convective deflagration propagates at a much faster rate (a few hundred m/sec [7]) and is a very important combustion mode in the transition to detonation. Convective deflagration occurs when pressures are sufficient to decrease the flame stand-off distance allowing for the flame to penetrate into cracks or pores in the damaged explosive [3]. This deflagration process increases the surface area available for burning, thus increasing the mass rate converted from a solid to gas and the exothermic energy released, further increasing the pressure and burn rate. We model this process with an isotropic damage model (ViscoSCRAM [6]) to determine the extent of cracking in the solid. The localized pressure and material damage is used to determine where convective deflagration is occurring. The WSB burn model is then used to calculate the mass converted to gas within the solid.

In order for an explosive to transition into a detonation, a pressure threshold must be reached. For octahydro-1,3,5,7-tetranitro-1,3,5,7-tetrazocine (HMX), the explosive of interest, this pressure is 5.3 GPa [42]. Once the detonation pressure threshold is reached the JWL++ reactive flow model [46] is used to model detonation. One of our hypotheses for a DDT in an array of explosives is that inertial confinement and deformation of the reacting cylinders pressing together, forms a barrier that allows the local pressure to increase to that needed for detonation.

3.1. Multi-material governing equations. The governing multi-material model equations are stated and described, but not developed, here. Their development and the methods for solving them can be found in [22, 27, 23, 24]. The 8 quantities of interest and the equations (or closure models) which govern their behavior may now be identified. Consider a collection of N materials, and let the subscript r signify one of the materials, such that $\mathrm{r} = 1, 2, 3, \ldots, N$. In the simulation discussed in Section 6 two materials are used, a solid (PBX-9501) and a gas (products of reaction). In an arbitary volume $V(\mathbf{x}, t)$, the averaged thermodynamic state of a material is given by the vector $[M_\mathrm{r}, \mathbf{u}_\mathrm{r}, e_\mathrm{r}, T_\mathrm{r}, v_\mathrm{r}, \theta_\mathrm{r}, \boldsymbol{\sigma}_\mathrm{r}, p]$, where the elements are the r-material mass,velocity, internal energy, temperature, specific volume, volume fraction, stress, and the “equilibration” pressure. The r-material averaged density is $\rho_\mathrm{r} = M_\mathrm{r}/V$. The rate of change of the state in a volume moving with the velocity of r-material is:

$$(3.1) \qquad \frac{1}{V}\frac{D_{\rm r}M_{\rm r}}{Dt} = \sum_{s=1, s\neq r}^{N} S_{\rho}^{s\rightarrow r}$$

$$(3.2) \quad \frac{1}{V}\frac{D_{\rm r}(M_{\rm r}\mathbf{u}_{\rm r})}{Dt} = \theta_{\rm r}\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \boldsymbol{\nabla}\cdot\theta_{\rm r}(\boldsymbol{\sigma}_{\rm r}-\boldsymbol{\sigma}) + \rho_{\rm r}\mathbf{g} + \Sigma_{s=1}^{N}\mathbf{F}_{\rm rs} + \sum_{s=1, s\neq r}^{N} S_{\rho\mathbf{u}}^{s\rightarrow r}$$

$$(3.3) \quad \frac{1}{V}\frac{D_{\rm r}(M_{\rm r}e_{\rm r})}{Dt} = -\rho_{\rm r}p\frac{D_{\rm r}v_{\rm r}}{Dt} + \theta_{\rm r}\boldsymbol{\tau}_{\rm r} : \nabla\mathbf{u}_{\rm r} - \boldsymbol{\nabla}\cdot\mathbf{j}_{\rm r} + \Sigma_{s=1}^{N}Q_{\rm rs} + \sum_{s=1, s\neq r}^{N} S_{\rho e}^{s\rightarrow r}$$

Equations (3.1-3.3) are the averaged model equations for mass, momentum, and internal energy of r-material, in which $\boldsymbol{\sigma}$ is the mean mixture stress, taken here to be isotropic, so that $\boldsymbol{\sigma} = -p\mathbf{I}$ in terms of the hydrodynamic pressure p. The effects of turbulence have been omitted from these equations.

In Eq. (3.2) the term $\sum_{s=1}^{N}\mathbf{F}_{\rm rs}$ signifies a model for the momentum exchange among materials and is a function of the relative velocity between materials at a point. For a two material problem we use $\mathbf{F}_{12} = K_{12}\theta_1\theta_2(\mathbf{u}_1 - \mathbf{u}_2)$ where the coefficient K_{12} determines the rate at which momentum is transferred between materials. Likewise, in Eq. (3.3), $\sum_{\rm s=1}^{N} Q_{\rm rs}$ represents an exchange of heat energy among materials. For a two material problem $Q_{12} = H_{12}\theta_1\theta_2(T_2 - T_1)$ where $T_{\rm r}$ is the r-material temperature and the coefficient $H_{\rm rs}$ is analogous to a convective heat transfer rate coefficient. The heat flux is $\mathbf{j}_{\rm r} = -\rho_{\rm r}b_{\rm r}\boldsymbol{\nabla}T_{\rm r}$ where the thermal diffusion coefficient $b_{\rm r}$ includes both molecular and turbulent effects(when the turbulence is included).

The temperature $T_{\rm r}$, specific volume $v_{\rm r}$, volume fraction $\theta_{\rm r}$, and hydrodynamic pressure p are related to the r-material mass density, $\rho_{\rm r}$, and specific internal energy, $e_{\rm r}$, by way of equations of state. The four relations for the four quantites $(e_{\rm r}, v_{\rm r}, \theta_{\rm r}, p)$ are:

$$(3.4) \qquad e_{\rm r} = e_{\rm r}(v_{\rm r}, T_{\rm r})$$

$$(3.5) \qquad v_{\rm r} = v_{\rm r}(p, T_{\rm r})$$

$$(3.6) \qquad \theta_{\rm r} = \rho_{\rm r}v_{\rm r}$$

$$(3.7) \qquad 0 = 1 - \Sigma_{s=1}^{N}\rho_{\rm s}v_{\rm s}$$

Equations (3.4) and (3.5) are, respectively, the caloric and thermal equations of state. Equation (3.6) defines the volume fraction, θ, as the volume of r-material per total material volume, and with that definition, Equation (3.7), is referred to as the multi-material equation of state and so defines the unique value of the hydrodynamic pressure p that allows arbitrary masses of the multiple materials to identically fill the volume V. This pressure is called the "equilibration" pressure [28]. and is solved using a cell-wise iterative scheme. The initial guess for the dependent variables $v_{\rm r}$, P, comes from the previous timestep and the thermal equation of state is evaluated convergence, see equation (9) in [24].

A closure relation is still needed for the material stress $\boldsymbol{\sigma}_{\rm r}$. For a fluid $\boldsymbol{\sigma}_{\rm r} = -p\mathbf{I} + \boldsymbol{\tau}_{\rm r}$ where the deviatoric stress is well known for Newtonian fluids and where $\boldsymbol{\tau}_{\rm r}$ is the viscous shear stress tensor. For a solid, the material stress is the Cauchy stress. The Cauchy stress is computed using a solid constitutive model and may depend on the rate of deformation, the current state of deformation ($\mathbf{E}$), the temperature, and possibly a number of history variables:

$$(3.8) \qquad \boldsymbol{\sigma}_{\rm r} \equiv \boldsymbol{\sigma}_{\rm r}(\boldsymbol{\nabla}\mathbf{u}_{\rm r}, \mathbf{E}_{\rm r}, T_{\rm r}, \ \ldots)$$

Equations (3.1-3.8) form a set of eight equations for the eight state vector with components $[M_{\rm r}, \mathbf{u}_{\rm r}, e_{\rm r}, T_{\rm r}, v_{\rm r}, \theta_{\rm r}, \boldsymbol{\sigma}_{\rm r}, p]$, for any arbitrary volume of space V moving with the r-material velocity. This approach uses the reference frame most suitable for a particular material type. The Eulerian frame of reference for the fluid and the Lagrangian for the solid. There is no guarantee that the arbitrary volumes will remain coincident for the two materials. This problem is addressed by treating the specific volume as a material state which is integrated forward in time from the initial conditions. The total volume associated with all of the materials is given by:

$$V_t = \Sigma_{r=1}^{N} M_{\mathrm{r}} v_{\mathrm{r}} \tag{3.9}$$

where the volume fraction is $\theta_{\mathrm{r}} = M_{\mathrm{r}} v_{\mathrm{r}} / V_t$ (which sums to one by definition). An evolution equation for the r-material specific volume has been developed in [27] and is stated here as:

$$\begin{aligned}\frac{1}{V}\frac{D_{\mathrm{r}}(M_{\mathrm{r}} v_{\mathrm{r}})}{Dt} &= f_{\mathrm{r}}^{\theta} \boldsymbol{\nabla} \cdot \mathbf{u} + \left[v_{\mathrm{r}} S_{\rho_{\mathrm{r}}}^{s \to r} - f_{\mathrm{r}}^{\theta} \Sigma_{s=1}^{N} v_{\mathrm{s}} S_{\rho_{\mathrm{s}}}^{s \to r} \right] \\ &+ \left[\theta_{\mathrm{r}} \beta_{\mathrm{r}} \frac{D_{\mathrm{r}} T_{\mathrm{r}}}{Dt} - f_{\mathrm{r}}^{\theta} \Sigma_{s=1}^{N} \theta_{\mathrm{s}} \beta_{\mathrm{s}} \frac{D_{\mathrm{s}} T_{\mathrm{s}}}{Dt} \right].\end{aligned} \tag{3.10}$$

where $f_{\mathrm{r}}^{\theta} = \frac{\theta_{\mathrm{r}} \hat{\kappa}_{\mathrm{r}}}{\sum_{\mathrm{s}=1}^{N} \theta_{\mathrm{s}} \hat{\kappa}_{\mathrm{s}}}$, and $\hat{\kappa}_{\mathrm{r}}$ is the r-material bulk compressibility, β is the constant pressure thermal expansivity.

The evaluation of the multi-material equation of state (Eq. (3.7)) is required to determine an equilibrium pressure that results in a common value for the pressure, as well as specific volumes that fill the total volume identically.

3.2. Reaction Model. In Eq. (3.1) $S_{\rho}^{s \to r}$ is the rate of mass converted from s-material, or solid reactant, into r-material, gaseous products. Similarly, in Eqs. (3.2) and (3.3), $S_{\rho \mathbf{u}}^{s \to r}$ is the momentum and $S_{\rho e}^{s \to r}$ the energy converted between the s and r materials. These are simply the mean values of the donor material (PBX-9501) in the volume. The model for the mass conversion or mass burn rate is discussed below with full details provided in [4].

Our reaction model uses a simplified two phase chemistry model in which the solid explosive (A) is converted to gas phase intermediates (B) which react to form the final products (C). A(solid) $\rightarrow$ B(gas) $\rightarrow$ C(gas). Therefore only two phases of the combustion are modeled; the condensed and gas phases. The melt layer present in many explosives is assumed to have little impact on the overall combustion and is therefore ignored. This model has a large pressure dependence associated with the conductive heat transfer; as mentioned before, this greatly affects the rate of gas phase reactions. The mass burn rate $S_{\rho}^{s \to r}$, where ρ is the density, is computed using equations 3.11 and 3.12,

$$S_{\rho_{\mathrm{r}}}^{s \to r} = \left[\frac{\kappa_s \rho_s A_s R (\hat{T}_s)^2 exp(-E_s / R\hat{T}_s)}{C_p E_s [\hat{T}_s - T_0 - Q_s / 2C_p]} \right]^{1/2} \tag{3.11}$$

$$\hat{T}_s = T_0 + \frac{Q_s}{C_p} + \frac{Q_r}{C_p (1 + \frac{x_r(m_b, P)}{x_s(m_b)})} \tag{3.12}$$

where T_0 is the initial bulk solid temperature, κ is the thermal conductivity, E is the activation energy, R is the ideal gas constant C_P is the specific heat, Q is the heat released and x_r, x_s are physical lengths [4]. $\hat{T}_s$ is a sub-scale surface temperature, not to be confused with T_r or T_s in Eqs. (3.4, 3.5, 3.8, 3.10). Equations 3.11 and 3.12 are solved iteratively until convergence. This model for the mass burned (MB) has been modified to include three dimensional effects by including the Burn Front Area of a cell, BFA, [51], and evaluated over a given time, Δt, see Equation 3.13. This model has been validated against experimental data for a wide range of pressures at initial solid temperatures of 273K, 298K and 423K [42].

$$MB = \Delta t * BFA * S_{\rho_{\mathrm{r}}}^{s \to r} \tag{3.13}$$

The reaction model utilizes the crack propagation results from the ViscoSCRAM constitutive evaluation to model the transition into convective deflagration as defined by Berghout [8]. This model has been adjusted to match experimental relaxation times as determined by the visco-elastic response for PBX-9501 [5] At a pressure of 5.3 Gigapascals (GPa) or higher Uintahs DDT model switches from deflagration to detonation. This pressure threshold of 5.3 GPa for HMX and PBX-9501 was chosen for three main reasons. First, this threshold gave reasonable results for the run distance to detonation for aluminium impact experiments [42]. Second, it is well known that the reaction rate increases with increasing pressure. It was discovered there is a discontinuity in this increase at 5 GPa, and the reaction rate exhibits a large increase at this pressure. At pressures above 5 GPa the reaction rate continues to increase with pressure, but more dramatically than it did at pressures below the discontinuity [42, 18]. Third, the internal energy produced by the reversible adiabatic compression of solid HMX to 5.3 GPa is 138.9 kJ/mol . This amount of energy is relatively close to the activation energy found for HMX, which lies between 140 and 165 kJ/mol [42] and slightly higher for PBX-9501. More information about Uintah's validated reaction and material models can be found in [42].

4. Adaptive Mesh Refinement Challenges & Improvements. Modern, large-scale simulations such as our target problem (Section 3) require the use of massive parallelism and adaptive mesh refinement (AMR). It is well known that achieving a high degree of scalability for AMR based simulations is challenging due to poor scalability associated with the changing grid. In order to change the grid in response to a solution evolving in time, a number of steps must occur that do not occur in a fixed mesh calculation. These steps generally include regridding, load balancing and scheduling [31], as AMR requires that the grid and task schedule be recreated whenever regridding occurs. Poor performance in any of these steps can lead to performance problems at larger scales [31]. As we have gained access to larger and more diverse computational environments, we have greatly extended the scalability of the Uintah framework, necessitating continual improvements in the framework itself.

4.1. Standard Benchmark Problem. To understand and continually improve the scaling characteristics of Uintah and key components like MPMICE for each successive generation of machine, we have developed and used a standard benchmark problem that simulates a moving solid through a domain filled with air to represent key features of our benchmark problems as modeled using the MPMICE algorithm in the Uintah framework. In this work we will refer to two separate resolutions for our benchmark problem, *resolution-A* (192^3 cells) and *resolution-B* (384^3 cells). This benchmark is shown using *resolution-A* in [34] and [36], is representative of the detonation problem that is the focus of this work, exercises all of the main features of AMR, ICE and MPM, and also includes a model for the deflagration of the explosive along with the material damage model ViscoSCRAM. For *resolution-A*, three refinement levels are used for the simulation grid with each level being a factor of four more refined than the previous level. This problem has a total of 3.62 billion particles, 518 million cells and 277,778 total patches created on three AMR levels. While our benchmark problem with *resolution-A* achieved excellent scalability to 512K cores on the DOE Mira system [36], We observed a significant breakdown in scaling at 768K cores due to there being less than 0.3 patches per core and hence devised a much larger resolution problem, *resolution-B* (384^3 cells) by doubling the resolution in each direction resulting in nearly an order of magnitude increase in problem size. This problem, *resolution-B* uses a grid utilizing three refinement levels with each level being a factor of four more refined than the previous level, has a total of 29.45 billion particles, 3.98 billion cells created on three AMR levels, and 1.18 million total patches. As has been witnessed in the past, with each significant increase in problem size and successive generation of machine, we have discovered algorithmic challenges within the underlying framework that limit the scalability at some level. The scaling challenges faced in this work have only become apparent by running this large problem at high core counts that have stressed sections of infrastructure code in ways never before seen.

In this case the desire to solve this benchmark problem has required a substantial re-working of core algorithms (see Section 4.2), with extensive work on Uintah's task-graph compilation phase, load balancer and regridder. To achieve good scaling at high resolutions for our benchmark and detonation problem at high core counts on the DOE Mira system has required 3-4 man-months of work and millions of compute hours in debugging and testing at large scale.

In order to provide a better perspective on the amount of time and level of difficulty involved in debugging the problems described above, the first issue faced in improving Uintah's AMR capabilities on our standard benchmark problem with *resolution B* is described. Within the MPM particle creation routines (see Section 4.2.1), the lowest core count we were able to reproduce the bug we encountered at large scale was 64K cores. This turned out to test the limits of the large-scale commercial debugger Allinea DDT [13] on Mira. At these core counts on Mira, I/O nodes ran out of memory causing racks of the machine to crash. This was resolved only by the creation of special debug queues by Argonne staff that helped us to resolve this difficult and large-scale debugging issue.

4.2. Improvements. The Uintah framework has been improved to support the resolution required by (and hopefully beyond) this detonation problem, particularly in its particle system, load balancer and AMR infrastructure code. In order to identify key performance and scalability issues, Uintah's built-in monitoring functions to locate components needing improvement was used. Third-party profiling tools such the Google Performance Tools [21] and HPCToolkit [43] were then used to localize the exact code consuming the most CPU time. Manually inserted timers were also used to confirm profiling results and to verify the improvement once changes were made. The following four major areas of Uintah infrastructure code are discussed here to illustrate the scaling deficiencies that were discovered when running the benchmark problem using *resolution-B* (see Section 4.1) at extreme scale and how these problems were addressed.

TABLE 1
Particle Creator Improvement: Strong Scaling

Cores	8K	16K	32K	64K	128K	256K	512K
Before (Average)	2977.2	1475.9	705.9	332.5	147.6	55.9	15.7
Before (Maximum)	3339.6	1652.2	793.1	375.8	170.0	67.9	21.6
After (Average)	424.5	224.6	118.8	63.1	33.1	17.3	5.4
After (Maximum)	524.8	283.4	148.2	78.9	44.1	22.6	7.3

4.2.1. Particle Creation. As higher resolutions are now being used in the MPMICE simulation (Section 4.1), a dramatic slow down was observed during the initialization timestep. After resolving the large-scale debugging issues described in Section 4.1, it was possible to localize the problem source. By enabling Uintah's internal reporting for task execution times, it was found that this performance issue originated from the *MPM::actuallyInitialize* task. This task is designed to create particles and initialize particle data. By using the profiling methods described above, the particle creator code was found to be the primary source of this slow down. In Uintah, particles on each patch are created internally by the framework via a particle creator component. There are many internal variables defined within the particle creator component's global scope. Each time the particle creator processes a new patch, these temporary variables were being overwritten. The particle creator component was originally written ten years ago and worked well when using an MPI-only approach; the only approach available within Uintah at that time. When multi-threaded support was recently added [35], Pthread mutexes were added to protect these globally defined variables and generated significant overhead due to contention for the locks when particles are created on multiple patches concurrently.

This issue was resolved by redesigning the data structures within the particle creator code. This was accomplished by separating those variables that were globally defined into two categories; 1.) read-only variables that must remain globally defined and used by all patches, and 2.) thread-local variables which can be separated from one another and can be concurrently accessed without the need for locks. This is a typical problem when using locks on legacy data structures (from an MPI-only approach), whereby unnecessary shared data must be separated to get better performance. Table 1 shows the particle creation timing results, comparing strong scaling runs from 8K cores to 512K cores. After redesigning these legacy data structures to work in a multi-threaded environment, a 3X to 7X speedup in this portion of the code was observed.

TABLE 2
Resource Assignment Improvement: Weak Scaling

Cores	128	1K	82K	64K	512K
Before (Avg.)	0.039	0.269	2.39	18.25	60.96
After (Avg.)	0.010	0.011	0.010	0.009	0.009

4.2.2. Resource Assignment. Another component that showed significant performance degradation at large scale with high resolution was Uintah's load balancer. As described in Section 2, the load balancer partitions the simulation grid by using a history data-based forecast model. Tasks are then created on patches assigned to a local node. The profiling results obtained here revealed that scaling issues were centered around the load balancer's *AssignResources* method. This method assigns each patch in the grid with a rank ID. This rank information is then used for subsequent, automatic MPI communication generation between tasks on different nodes. From the weak scaling timing results as shown in the *before row* of Table 2, the cost of *AssignResources* grows when the number of patches per node stays constant. This timings show that there is an algorithm issue that has to be addressed. The original code looped though all the patches in the grid to assign a resource to it. This algorithm has an $O(n)$ complexity, where n is the number of patches. However as the MPI communications only happens locally in MPMICE, only the tasks that will communicate with the local tasks need be considered. Consequently, it was possible to restrict this method to only assign patches in the neighborhood of the local node. The new algorithm has an $O(n/p)$ complexity, where n/p is the number of patches in the neighborhood. For weak scaling tests, n/p is constant as the workload per node stays the same. The scaling results shown in the *after row* of Table 2 confirm the improved weak scaling and also show up to 6800X speedup when using this new algorithm.

4.2.3. Copy Data Timestep. The following two subsections will describe how the performance and scalability of Uintah's AMR infrastructure code has been greatly improved. As mentioned above, the efficiency of the regridding operation is very important for solving the detonation problem. The entire AMR regridding procedure includes three steps: 1) generating a new grid based on the refinement flags computed by the simulation component, 2) a copy-data timestep to determine differences between the old and new grid. For an already refined area, this means copying data from existing fine level data. For a newly refined area, this step calls a user-provided refine task to compute fine level data from coarse level data, and 3) compile a new task graph on the new grid for future simulation timesteps. Originally there was about 98% overhead for a single regridding operation on 512K cores if regridding were to occur every 50 to 60 timesteps. Profiling and timing measurements were obtained for the regridding operations to locate performance and scaling issues.

The current regridding algorithm has a linear complexity. The regridder timing is shown in Figure 2 -Regridder. The solid line shows timing results in terms of weak scaling. The dotted line shows a linear model that $T = \alpha p$ where $\alpha = 2.75 \times 10^{-4}$ and p are number of cores for the weak scaling runs. For the copy data timestep, the original algorithm computed the difference between the old and new grid by simply looping though the new grid patches and querying the related old grid patch. This algorithm runs in an $O(n \log(n))$ complexity, where n is the number of patches, as a bounding volume hierarchy (BVH) tree is used for querying a patch from the grid. Each query of this BVH tree costs $O(\log(n))$. It is important to have a consistent grid across all processors, so every node performs this computation. The cost of this copy data timestep is very small. It accounts for less than 0.2% of the total overhead on small scale runs, e.g. less than 10K cores. However, the overhead of this operation grows significantly when running with 512K cores. To improve the scalability while keeping the grid consistent across all the nodes, the difference between the old and new grids is computed and then all locally computed differences are gathered to obtain the difference across the entire grid. This new algorithms thus has two parts. The cost of the individually computed portions is $O(n \log(n)/p)$. When running weak scaling tests, n/p is constant. We then have approximately $O(log(p))$ complexity for the new code. The complexity for combining the individually computed portions together is $O(p)$. Figure 2-Copy Data shows the timing comparison between the new and old algorithms. A model of $T = \alpha \log(p) + \beta p$ where $\alpha = 1.60, \beta = 6.69 \times 10^{-6}$ for the new algorithm is shown in the dotted line. These results show about 10X speedup for the copy data timestep when using the new algorithm. This is clear evidence that a sub-optimal algorithm will become a significant performance issue at large scale, even when its cost appears negligible at low core counts.

4.2.4. Task Graph Compile. After new data has been copied to the new grid, the simulation needs to continue with this new grid. With Uintah's DAG based design, when the grid layout or its partition changes, a new task graph needs be to compiled and new MPI message tags are then generated by the framework. Task graph compilation is a complex operation with multiple phases, including creation of tasks themselves on local and neighboring patches, keeping a history of what these tasks are to compute, setting up connections of tasks (edges in the DAG), and finally assigning MPI tags to dependencies. Originally Uintah used a static scheduler where tasks were topologically sorted to compute a valid task execution order. This topological sort code also ensured the global reduction was called in a determined order across all the processors. However, this original code was written for a relatively small grid. When the sorting function decides which task should be executed before another task, it takes the union of a particular task's patches and then compares the union of patches from another task to determine the overlap. This is an $O(n^2)$ complexity, however it costs less than 0.2% of the total overhead and hence, was unnoticed until running at extreme scales as in this work. With the dynamic task scheduler, this sorting is no longer necessary. The global reduction ordering portion of this sorting which has a constant cost regardless of the number of processors or problem size was thus ultimately eliminated. As is shown in Figure 2-TaskGraph Compile, the task graph compiling code, there is a 42X speedup when running with 512K cores. The dotted line in this graph shows a constant scaling model. The overall AMR regridding cost including all three steps has improved by about 10X and its

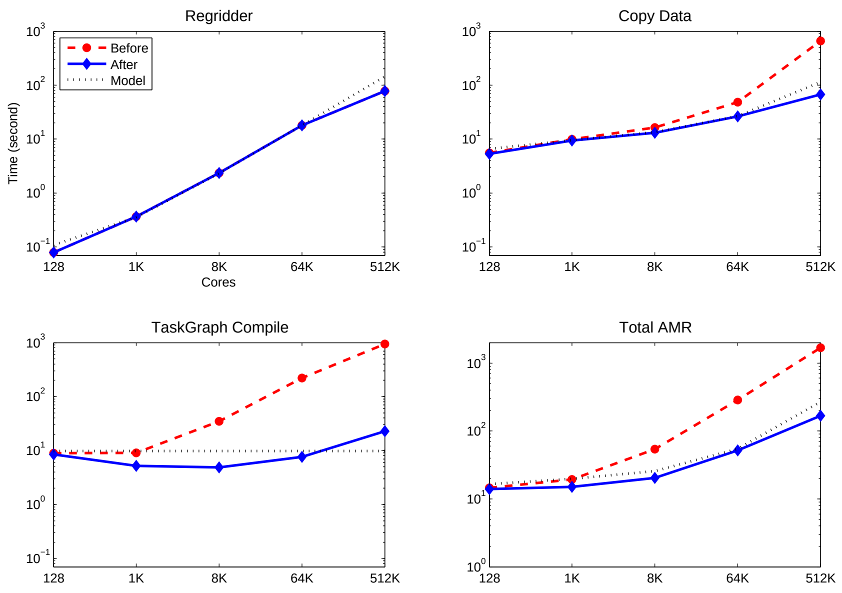

FIG. 2. *AMR Improvement Breakdown: Weak Scaling Timings for 128K to 512K Mira Cores*

overhead is less than 10% percent when running with 512K cores. The comparison of before and after timing and model results are shown in Figure 2-Total AMR. After these significant development efforts, the tiled regridder itself now contributes the most overhead of all three steps. Further improvements to this component will also be needed in the future.

5. Scaling Results. This section shows the scalability results for the AMR MPMICE simulations for both the standard benchmark problem (on both Mira and Blue Waters) using *resolution-A* and *resolution-B* (as defined in Section 4.1), as well as the actual detonation configuration for the array of multiple explosive devices (Mira only).

Strong scaling is defined as a decrease in execution time when a fixed size problem is solved on more cores, while weak scaling should result in constant execution time when more cores are used to solve a correspondingly larger problem.

Figure 3 demonstrates the overall strong scaling for the standard AMR MPMICE benchmark problem described in detail in Section 4.1 using both *resolution-A* from [36], [34] and *resolution-B* developed in this work. These tests were run on Blue Waters and Mira with up to 704K (Blue Waters) and 768K cores (Mira) and with 16 (Mira) and 32 (Blue Waters) threads per MPI node. It is interesting to observe that with the larger core count per node for Blue Waters (32 vs 16) the scaling is closer to being ideal. This can be attributed to the reduction in global communication. The strong scaling efficiency relative to 256K cores for Blue Waters on 704K cores is 89% and for Mira on 768K cores is 71% when running the benchmark problem of *resolution-B*.

In order to obtain scaling results shown, the optimal patch configuration was determined for our AMR MPMICE benchmark problem as needing to satisfy the following two requirements. 1) The number patches on each level should be tuned as closely as possible but should not exceed the number of cores on the largest run. 2) The patch should have at least 8x8x8 cells. The second requirement overrides the first one in that, without enough patches in a particular level for all CPU cores, it is not possible to further divide patches beyond 8x8x8. For patch sizes smaller than 8x8x8, the cost of a patch's MPI messages begins to exceed the cost of its computation, and hence the runtime system cannot overlap computation with communication. This lower bound on patch size should be considered as machine-dependent. In addition to choosing a good patch size for different AMR levels, it is also important to line up patch boundaries in finer levels to patch boundaries in coarser levels. An easy way to achieve this is to choose a finer level patch size that can evenly divide coarser level patch size in each dimension. For example, when coarse level patch size is 8x8x8, it is better to have a finer level patch size of 16x16x8 than 12x12x12. In fact the latter choice of patch size has been seen to lead to a greater MPI communication imbalance.

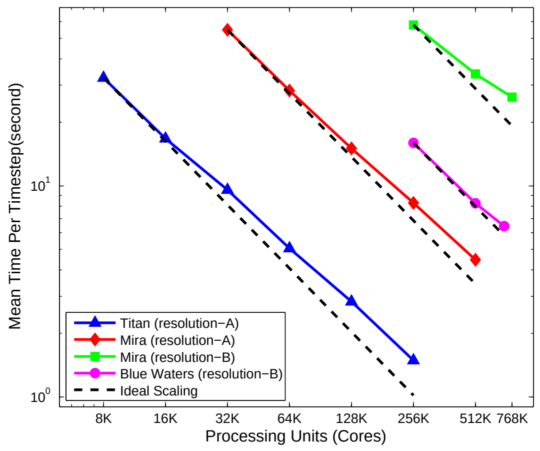

FIG. 3. *AMR MPMICE Strong Scaling for the Benchmark Problem:* resolution-A *with 192^3 cells and* resolution-B *with 384^3 cells*

5.1. Strong Scaling of MPMICE for Benchmark Problem. In the standard benchmark problem for both *resolution-A* and *resolution-B*, the simulation grid changes once every 50 to 60 timesteps as in [31]. A regridding operation occurs once the perticles reach the edge of the fine grid. The overhead of this regridding process, including creating the new grid, compiling a new task graph and moving old grid data to the newly created grid, accounts for less than 3% of the overall execution time with *resolution-A* and 10% with *resolution-B* when running with 512K cores. This is a result of the improvements described in Section 4.2 that have been made to reduce the cost of regridding process for AMR MPMICE simulations.

5.2. Weak and Strong Scaling of MPMICE for Detonation Problem. Using the benchmark problem (see Section 4.1) to understand the scaling characteristics of Uintah and its MPMICE simulation component, engineering guidelines to ensure scalability at the largest core counts of interest have been developed. In particular, it was determined that the patches should have sufficient resolution, minimally 512 cells per patch and that there should be approximately one patch per core. During the strong scalability performance runs, the total number of patches and resolution were fixed while the core count was increased. At the largest core count run, it was necessary to adjust the number of patches for the finest level such that the total count did not exceed the number of cores. In fact, good strong scaling was observed even when the number of patches was approximately 85% of the total core count.

Although considerable effort was spent characterizing our benchmark problem at varying resolutions, the real interest is to improve Uintah's performance on real engineering problems of interest. Scaling of benchmark problems has little value if the real problems do not scale. With that in mind, an example of the detonation problem described in Section 3 was used with the insight gained from our benchmark characterizations to demonstrate the scalability up to 512K cores on DOEs Mira. Figure 4 shows good strong scaling, through the four solid lines for four problem sizes (each a factor of eight larger than the last) and reasonable weak scaling (the four dashed lines) showing slightly increasing execution time as the workload per core is constant across the same four increasing problem sizes. The problem was only run for ten timesteps (without AMR, as this was exercised at scale in the previous case) but with a mesh that had three refinement levels were used with four different grid resolutions for the real detonation calculation. For the largest case, there were 446,880 patches and 1.37 billion cells and 7.75 billion particles.

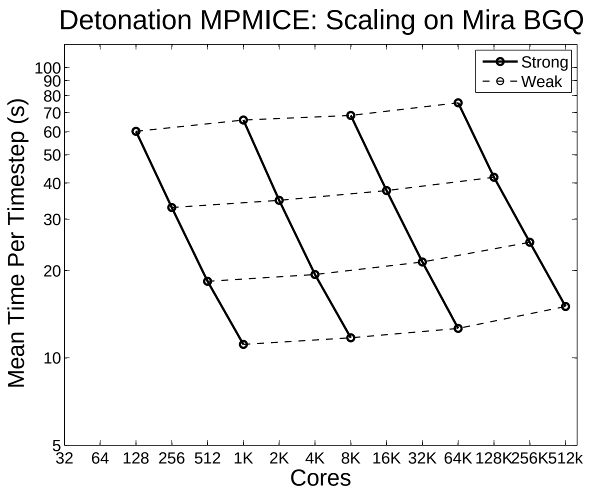

FIG. 4. *AMR MPMICE Strong and Weak Scaling for the Detonation Problem*

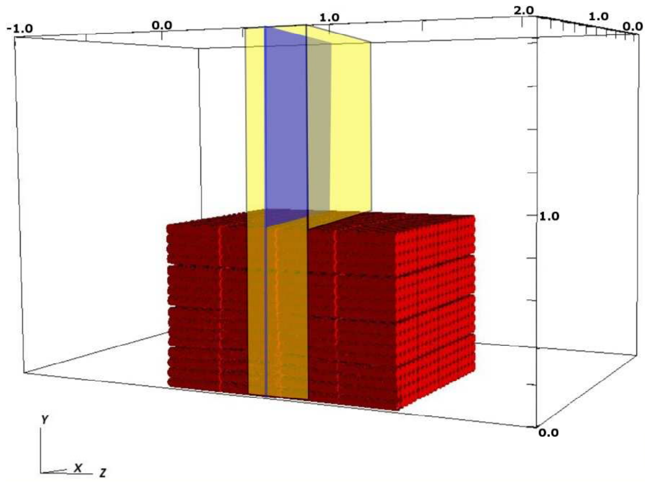

FIG. 5. *The initial set up of the three simulations. The large black box outlines the large 3D simulation, the yellow region shows the smaller 3D domain and the blue 2D slice shows the location of the 2D simulation plane.*

6. Computational DDT Modeling Results. In order to model the thousands of explosive devices, the grid resolution of the domain must be small enough to resolve the physical phenomena occurring in the three different modes of combustion. The domain must also be

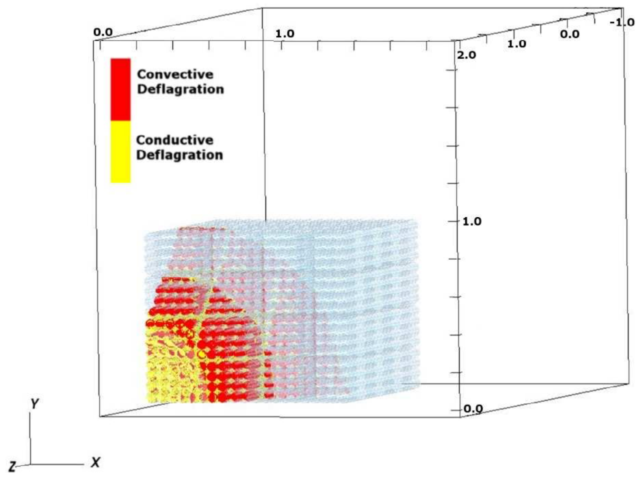

FIG. 6. *Preliminary results for the deflagration progression in the large 3D simulation. The full physical domain is shown. The light blue represents unburnt explosive cylinders and the red and yellow show the two modes of deflagration.*

large enough to ensure that the explosives are far away from the boundaries, to minimize any non-physical interactions with the numerical boundary conditions. To address the length scales (mm-m), Adaptive Mesh Refinement (AMR) was used with three levels and a refinement ratio of 4 between each mesh level. The results shown here are from simulating 1280 explosive cylinders, packed 4x5 to a crate, in a configuration similar to the packing of the 2005 transportation accident. The explosives are 54 mm in diameter and 0.33 m long and are ignited by hot gas along a confined boundary. Two of the boundaries ($x-, y-$) are reflective, the other four boundaries are "open", allowing product gases and particles to flow freely through them. The "open" boundaries are 1 m from the explosives in the $x+$, $y+$, $z+$, and $z-$ directions to minimize the boundary interactions (Figures 5 and 6). The domain for this simulation is 12 m^3 resulting in 350 million cells (2 mm cell spacing on the finest level) and 980 million particles to represent the explosives. Figure 6 shows the progression of burning within the cylinders. The light blue volume represents unburnt explosive cylinders, the red volume shows convective deflagration and yellow volume shows where conductive deflagration is occurring.

6.1. 2D and Axi-symmetric 3D Simulations. To investigate the possible mechanisms of DDT in an array of explosives, smaller 2D and 3D simulations were run. In these simulations the initial cylinder distribution was the same as the large 3D scenario described above, with 4x5 cylinders packaged in a "box" with 10 mm gaps, representing the spacing of the packing boxes. The main difference between all of the computational domains was the length of the domain in the z direction. A 2D simulation was run with 320 explosive cylinders and four highly confined boundaries. The location of the slice in the z direction is shown in Figure 5 by the blue slice. This numerical experiment demonstrated that a DDT was possible in this packing configuration when the explosive was highly confined. One proposed mechanism for this DDT is the inertial confinement created from the damaged cylinders forming a barrier that prevents the flow of product gases from exiting the domain, creating a pocket of high pressure and transitioning to detonation. Another possible mechanism is that the impact of the colliding cylinders in the high pressure environment produces a shock-to-detonation transition in the deflagrating material. Since this is a 2D simulation, the reacting gases and

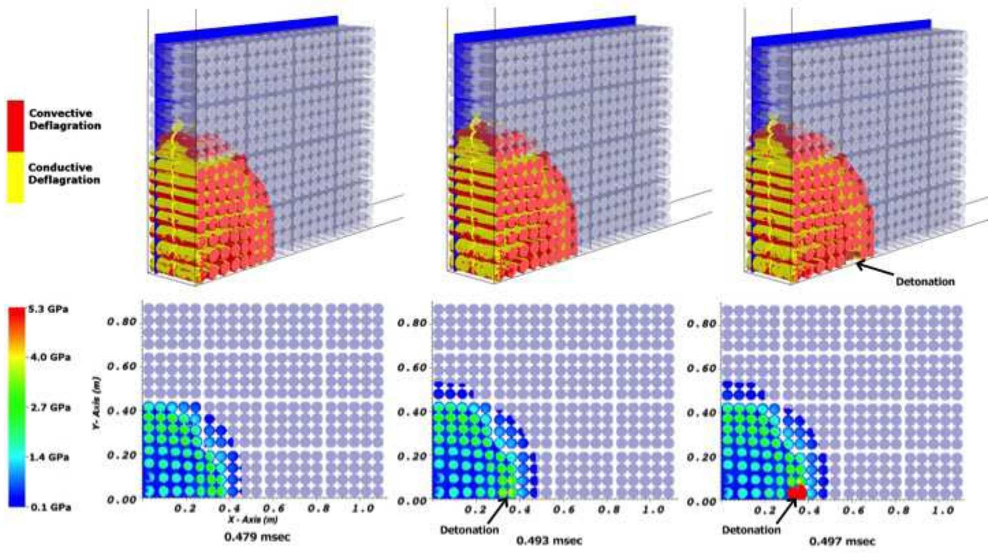

FIG. 7. *Top figure show the progression of deflagration through the explosives (light blue). The dark blue shows where the pressure slice (shown below) was taken. The bottom is a pressure profile of a DDT over time. Detonation can be seen at 0.710* $msec$

cylinders are artificially confined in the z direction, so no conclusions can be made and further tests are required in three space dimensions.

A smaller 3D simulation, shown in Figure 5 by the yellow region, with gaps in all directions allowed product gases to escape causing an increase in the time to detonation. Four of the boundaries were symmetric so gas could only escape out of the $x+$ and $y+$ boundaries. Figure 7 shows the burning modes and pressure distribution for a deflagration to detonation transition in the smaller 3D simulation. The top figure shows the progression of burning through the unburnt cylinders (light blue). The yellow volume represents conductive deflagration, the red volume shows convective deflagration and the dark blue slice shows where the pressure profile is taken. The lower contour plot shows the pressure distribution of a DDT in the array. Detonation occurred at $0.710\ msec$, and by $0.716\ msec$ the detonation had consumed a large portion of the explosive. It took approximately 40 microseconds longer for the smaller 3D simulation to detonate than the 2D simulation. This is the result of the product gases having more paths to escape in the 3D simulation.

6.2. Full 3D Simulations. In the case of the full 3D simulation, 64K cores were used in a calculation that ran from May 2014 until November 2014, with regular checkpointing and consumed about 30M CPU hours on Mira. While this simulation was not run at the full scales made possible by the scaling improvements shown above, it was not possible with the Mira allocation available to move to the next problem size up. This is solely due to the extra CPU hours needed beyond our allocation and the elapsed time needed beyond our allocation period. Figure 8 shows the maximum pressure trends for the 2D, smaller 3D and the large 3D simulations. The smaller array simulations give insight into the possible physical mechanisms. These mechanisms have been validated in the large 3D simulation, which also detonates, giving us a better understanding of how to suppress the transition to detonation in future transportation accidents. A first attempt at modeling this was made by changing the

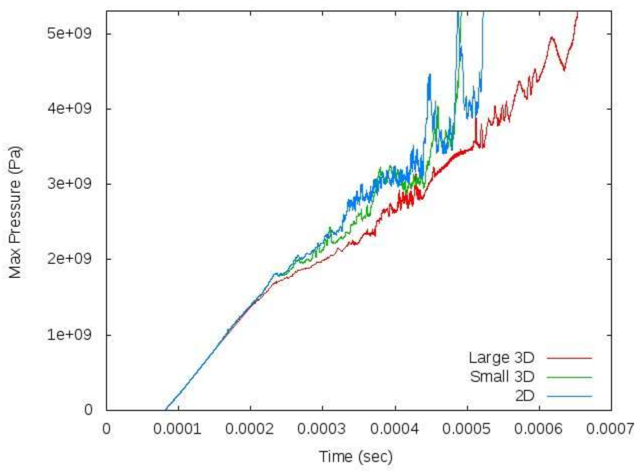

FIG. 8. *Maximum pressure on the finest level over time for the different simulations. Detonation occurs at 5.3 GPa.*

packing configuration of the detonator boxes to intersperse one empty box between two full ones in a checkerboard configuration. The results for a full 3D simulation of this case are shown in Figure 9. While it was not possible to run the simulation to completion, this preliminary result shows much lower pressures and suggests that this alternate packing approach may show promise as a means of more safely, but expensively, transporting the explosives. In this case the simulation was run on 200K cores on Mira and then on 16K Stampede cores.

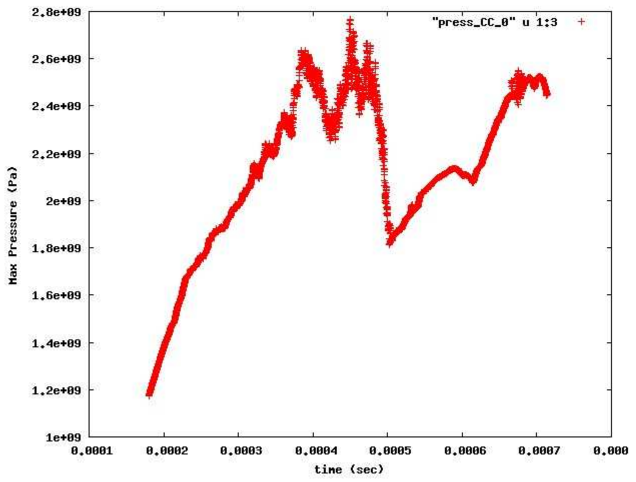

FIG. 9. *Maximum pressure on the finest level over time for the alternate packing configuration simulation. No detonation occurs.*

7. Related Work. There are several computational frameworks that use SAMR that are leveraged by application codes to solve similar types of problems as those for which Uintah was originally developed. These frameworks, including Uintah, are surveyed in a recent paper [17].

BoxLib [1] is a framework for building massively parallel SAMR application described by time-dependent PDEs, CASTRO [2] uses the BoxLib software for fully compressible radiation-hydrodynamics, while MAESTRO also uses BoxLib for low Mach number astrophysical calculations. Chombo [14] is an offshoot of the BoxLib framework that originated in 1998. Chombo has diverged from BoxLib in its implementation of data containers, but a number of applications build upon the framework including MHD, compressible CFD, CFD+EM, fluid-structure interaction, etc. Cactus [20] is a general-purpose software framework for high-performance computing with AMR as one of its features. The Einstein toolkit is the most prominent application. Enzo [38] is an astrophysical code that makes use of AMR for high resolution space and time requirements. A wide range of hydrodynamics and magneto-hydrodynamics solvers, radiation/diffusion and radiation transport have been incorporated. FLASH [19] was originally designed for simulation of astrophysical phenomena dominated by compressible reactive flows. Due to multiple physical scales, AMR was implemented using the octree-based PARAMESH packages. FLASH has undergone infrastructure improvements such that other applications including high energy physics, CFD and fluid-structure interactions leverage the FLASH framework. What distinguishes Uintah from other frameworks is both its underlying programming model and the development of a runtime environment with a DAG based taskgraph and application layer that makes it possible to achieve scalability at very large core counts.

There has been much related detonation work in the form of numerical modeling and experimental research on gas phase DDTs e.g. [37], but little is known about DDTs in a large collection of solid explosives. Significant amounts of experimental work has been done over the decades, on small scales, to better understand the role of convective deflagration in the transition into detonation for solid explosives [3]. Due to the hostile environment, the extreme pressures, temperatures and short time scales in a DDT, experiments have been relatively small scale (a few cm) [52, 12]. Other groups are modeling the transition to examine DDT mechanism which can not be seen experimentally [48, 50]. These mesoscale simulations have yet to produce a clear physical mechanism. Though these results will be beneficial to understanding the underlying mechanisms in a single monolithic device it will still be unclear how a DDT occurs in an unconfined array of explosives. To the best of the authors knowledge, the approach described here is a unique and novel attempt to understand DDT in a large collection of small explosive devices, especially on this scale.

8. Conclusions and Future Work. The main conclusion from this paper is that improving the supercomputer scalability of a complex DDT calculation required the removal of deficiencies that prevented scalability and that were not readily apparent at smaller or incremental changes in resolution. Discovering these key shortcomings in the runtime system algorithms and improving their overall algorithmic complexity resulted in dramatic improvements to the overall scalability of a challenging fluid-structure interaction benchmark problem. The immediate benefits to the runtime system resulted in our ability to demonstrate scalability for challenging complex fluid-structure interaction problems at at nearly the full machine capacity of Mira. This in turn made it possible to run a series of calculations that showed promise in improving our understanding of the detonation in the full highway 6 accident.

The general lessons from this work are that even when a substantial amount of work has been done to improve the scalability of a complex software framework, there are always challenges when trying to move to significantly larger problems and machines. Furthermore it is generally accepted by those working on the largest computers that these challenges will often involve technical innovation at the level of the algorithms, data structures and software architectures with a level of difficulty that is often unique to those scales.

9. Acknowledgments. 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 at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number ACI 1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This work was supported by the National Science Foundation under subcontracts No. OCI0721659, the NSF OCI PetaApps program, through award OCI 0905068. The support of the NSF XSEDE network is also acknowledged, particularly the support of TACC who also provided a Director's discretionary allocation to enable the final run to be made.

REFERENCES

[1] *BoxLib*, 2011. https://ccse.lbl.gov/Boxlib.

[2] A. Almgren, J. Bell, D. Kasen, M. Lijewski, A. Nonaka, P. Nugent, C. Rendleman, R. Thomas, and M. Zingale, *Maestro, castro, and sedona–petascale codes for astrophysical applications*, arXiv preprint arXiv:1008.2801, (2010).

[3] B. W. Asay, S. F. Son, and J. B. Bdzil, *The role of gas permeation in convective burning*, International Journal of Multiphase Flow, 22 (1996), pp. 923–952.

[4] J. Beckvermit, T. Harman, A. Bezdjian, and C. Wight, *Modeling Deflagration in Energetic Materials using the Uintah Computational Framework*, Accepted in Procedia Computer Science, (2015).

[5] J. G. Bennett, K. S. Haberman, J. N. Johnson, and B. W. Asay, *A constitutive model for the non-shock ignition and mechanical response of high explosives*, Journal of the Mechanics and Physics of Solids, 46 (1998), pp. 2303–2322.

[6] J. G. Bennett, K. S. Haberman, J. N. Johnson, B. W. Asay, and B. F. Henson, *A Constitutive Model for the Non-Shock Ignition and Mechanical Response of High Explosives*, Journal of the Mechanics and Physics of Solids, 46 (1998), pp. 2303–2322.

[7] H. L. Berghout, S. F. Son, L. G. Hill, and B. W. Asay, *Flame spread through cracks of PBX 9501 (a composite octahydro-1,3,5,7-tetranitro-1,3,5,7-tetrazocine-based explosive)*, Journal of Applied Physics, 99 (2006).

[8] H. L. Berghout, S. F. Son, C. B. Skidmore, D. J. Idar, and B. W. Asay, *Combustion of Damaged PBX 9501 Explosive*, Thermochimica Acta, 384 (2002).

[9] M. Berzins, *Status of release of the Uintah Computational Framework*, Tech. Report UUSCI-2012-001, Scientific Computing and Imaging Institute, 2012.

[10] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C.A. Wight, and J.R. Peterson, *Uintah - a scalable framework for hazard analysis*, in TG '10: Proc. of 2010 TeraGrid Conference, New York, NY, USA, 2010, ACM.

[11] M. Berzins, J. Schmidt, Q. Meng, and A. Humphrey, *Past, present, and future scalability of the uintah software*, in Proceedings of the Blue Waters Extreme Scaling Workshop 2012, 2013, p. Article No.: 6.

[12] P. B. Butler and H. Kriar, *Analysis of deflagaration to detonation transition in hight-energy solid propellants*, annual technical report, University of Illinois at Urabana-Champaign, September 1984.

[13] Allinea Software. Website by Versantus, *Allinea Web Page*, 2014. http://www.allinea.com/.

[14] P. Colella, D. Graves, T. Ligocki, D. Martin, D. Modiano, D. Serafini, and B. Van Straalen, *Chombo software package for AMR applications: design document*.

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

[16] D.L.Brown and P.Messina et al., *Scientific grand challenges: Crosscutting technologies for computing at the exascale*, Tech. Report Report PNNL 20168, US Dept. of Energy Report from the Workshop on February 2-4, 2010 Washington, DC, 2011.

[17] A. Dubey, A. Almgren, John Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Lffler, B. OShea, E. Schnetter, B. Van Straalen, and K. Weide, *A survey*

of high level frameworks in block-structured adaptive mesh refinement packages, Journal of Parallel and Distributed Computing, (2014).

[18] A. P. Esposito, D. L. Farber, J. E. Reaugh, and J. M. Zaug, *Reaction Propagation Rates in HMX at High Pressure*, Propellants, Explosives, Pyrotechnics, 28 (2003), pp. 83–88.

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

[20] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, *The Cactus framework and toolkit: Design and applications*, in Vector and Parallel Processing VECPAR 2002, Lecture Notes in Computer Science, Berlin, 2003, Springer.

[21] Google Project Hosting Google, *Google Performance Tools Web Page*, 2014. https://code.google.com/p/gperftools/wiki/GooglePerformanceTools.

[22] J.E. Guilkey, T.B. Harman, and B. Banerjee, *An eulerian-lagrangian approach for simulating explosions of energetic devices*, Computers and Structures, 85 (2007), pp. 660–674.

[23] 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, 2003, WIT Press.

[24] T. B. Harman, J. E. Guilkey, B. A Kashiwa, J. Schmidt, and P. A. McMurtry, *An eulerian-lagrangian approach for large deformationfluid-structure interaction problems, part 1:multi-physics simulations within a modern computationalframework*, in Fluid Structure Interaction II, Cadiz, Spain, 2003, WIT Press.

[25] J.Ang and K.Evans et al, *Workshop on extreme-scale solvers: Transition to future architectures*, Tech. Report USDept. of Energy, Office of Advanced Scientific Computing Research. Report of a meeting held on March 8-9 2012, Washington DC, 2012.

[26] B.A. Kashiwa and E.S. Gaffney., *Design basis for CFDLIB*, Tech. Report LA-UR-03-1295, Los Alamos National Laboratory, 2003.

[27] B. A. Kashiwa, *A multifield model and method for fluid-structure interaction dynamics*, Tech. Report LA-UR-01-1136, Los Alamos National Laboratory, 2001.

[28] B. A. Kashiwa and R. M. Rauenzahn, *A multimaterial formalism*, Tech. Report LA-UR-94-771, Los Alamos National Laboratory, Los Alamos, 1994.

[29] S. Kumar, A. Saha, J. Schmidt, V. Vishwanath, P. Carns, G. Scorzelli, H. Kolla, R. Grout, R. Ross, M. Papka, J. Chen, and V. Pascucci, *Characterization and Modeling of PIDX for Performance Prediction*, in Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2013, pp. 96:1–96:11.

[30] 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.

[31] J. Luitjens and M. Berzins, *Scalable parallel regridding algorithms for block-structured adaptive mesh refinement*, Concurrency and Computation: Practice and Experience, 23 (2011), pp. 1522–1537.

[32] J. Luitjens, M. Berzins, and T. Henderson, *Parallel space-filling curve generation through sorting*, Concurrency and Computation:Practice and Experience, 19 (2007), pp. 1387–1402.

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

[34] 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).

[35] Q. Meng, M. Berzins, and J. Schmidt, *Using hybrid parallelism to improve memory use in Uintah*, in Proceedings of the Teragrid 2011 Conference, ACM, July 2011.

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

[37] T. Ogawa, E. Oran, and V. Gamezo, *Numerical study of flame acceleration and DDT in an inclined array of cylinders using an AMR technique*, Computers and Fluids, 85 (2013), pp. 63–70.

[38] 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, vol. 41 of Lecture Notes in Computational Science and Engineering, Berlin, Heidelberg, 2005, Springer-Verlag, pp. 341–350.

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

[40] S. G. Parker, J. Guilkey, and T. Harman, *A component-based parallel infrastructure for the simulation*

of fluid-structure interaction, Engineering with Computers, 22 (2006), pp. 277–292.

[41] J. R. Peterson, J. Beckvermit, T. Harman, M. Berzins, and C. A. Wight, *Multiscale modeling of high explosives for transportation accidents*, in XSEDE '12: Proceedings of 2012 XSEDE Conference, New York, NY, 2012, ACM.

[42] J. R. Peterson and C. A. Wight, *An eulerian-lagrangian computational model for deflagration and detonation of high explosives*, Combustion and Flame, 159 (2012), pp. 2491–2499.

[43] Rice University * Rice Computer Science, *HPCToolkit Web Page*, 2014. http://hpctoolkit.org/index.html.

[44] 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.

[45] S. F. Son and H. L. Berghout, *Flame spread across surfaces of PBX 9501*, in American Institute of Physics Conference Proceedings, 2006, pp. 1014–1017.

[46] P. C. Souers, S. Anderson, J. Mercer, E. McGuire, and P. Vitello, *JWL++: A simple reactive flow code package for detonation*, Propellants, Explosives, Pyrotechnics, 25 (2000), pp. 54–58.

[47] D. Sulsky, Z. Chen, and H. L. Schreyer, *A particle method for history-dependent materials*, Computer Methods in Applied Mechanics and Engineering, 118 (1994), pp. 179–196.

[48] W. A. Trzcinski, *Numerical analysisis of the deflagration to detonation transition in primary explosives*, Centeral European Journal of Energetic Materials, 9 (2012), pp. 17–38.

[49] M. J. Ward, S. F. Son, and M. Q. Brewster, *Steady deflagration of HMX with simple kinetics: a gas phase chain reaction model*, Combustion and Flame, 114 (1998), pp. 556–568.

[50] L. Wei, H. Dong, H. Pan, X. Hu, and J. Zhu, *Study on the mechanism of the deflagration to detoantion transition process of explosive*, Journal of Energetic Materials, 32 (2014), pp. 238–251.

[51] C. A. Wight and E. Eddings, *Science-Based Simulation Tools for Hazard Assessment and Mitigation*, International Journal of Energetic Materials and Chemical Propulsion, 8 (2009).

[52] T. Zhang, Y. L. Bai, S. Y. Wang, and P. D. Liu, *Damage of a high-energy solid propellant and its deflagration-to-detonation transition*, Propellants, Explosives, Pyrotechnics, 28 (2003), pp. 37–42.