Uintah - a scalable framework for hazard analysis

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

Justin Luitjens
Scientific Computing and Imaging Institute
University of Utah
Salt lake City, UT 84112 USA
luitjens@cs.utah.edu

Qingyu Meng
Scientific Computing and Imaging Institute
University of Utah
Salt lake City, UT 84112 USA
qymeng@cs.utah.edu

Todd Harman
Department of Mechanical Engineering
University of Utah
Salt lake City, UT 84112 USA
T.Harman@utah.edu

Charles A. Wight
Department of Chemistry
University of Utah
Salt lake City, UT 84112 USA
Chuck.Wight@utah.edu

Joseph R. Peterson
Department of Chemistry
University of Utah
Salt lake City, UT 84112 USA
Joseph.R.Peterson@utah.edu

ABSTRACT

The Uintah Software system was developed to provide an environment for solving a fluid-structure interaction problems on structured adaptive grids on large-scale, long-running, data-intensive problems. Uintah uses a novel asynchronous task-based approach with fully automated load balancing. The application of Uintah to a petascale problem in hazard analysis arising from "sympathetic" explosions in which the collective interactions of a large ensemble of explosives results in dramatically increased explosion violence, is considered. The advances in scalability and combustion modeling needed to begin to solve this problem are discussed and illustrated by prototypical computational results.

Keywords

Uintah, scalability, parallel, adaptive, energetic materials

1. INTRODUCTION

The risks of manufacturing, transporting and storing energetic materials (explosives, propellants and pyrotechnics) are in most cases well understood. Devices containing such materials undergo extensive testing for hazard classification prior to transportation and deployment so that appropriate protocols can be implemented. This testing is most often done with single articles, or with a small number (for testing sympathetic detonation). However, there is now reason to believe that the behavior of large ensembles of explosive devices may be fundamentally different, and more dangerous.

This hazard is illustrated by an accident that took place on August 11, 2005 when a truck carrying 35,500 pounds of Pentolite explosives in Utah's Spanish Fork Canyon overturned and caught fire. Within 3 minutes the truck unexpectedly detonated, leaving behind a 70' wide x 30' deep crater. The normal response of energetic materials to heating from a fire is to undergo a thermal explosion (deflagration), in which the rate of reaction is limited by heat transfer. In a deflagration, only a small fraction of the material is consumed during the explosion resulting in a low violence explosion. The other limiting mode of combustion is detonation, which is driven by pressure rather than heat transfer, so reaction rates are typically many orders of magnitude faster resulting in more of the explosive material being consumed during the combustion process. Thus, the violence of explosion from a detonating energetic material is orders of magnitude higher than the same material deflagrating. Normally, detonations can only be initiated by a strong shock wave generated from a primary detonator or booster charge. In the accident it is believed that the reaction was initially a deflagration that transitioned to a detonation. This mode of combustion, deflagration-to-detonation transition (DDT) represents one of the most dangerous and least understood potential hazards involving energetic materials [1] and one of the foci of this research.

It is well known that damaged energetic materials have a propensity to undergo DDT because of the increased porosity and the ability to sustain convective burning inside of the material. This raises the possibility that large ensembles of individual explosive devices may undergo convective burning and DDT when damaged. By understanding the physical processes of this behavior and through simulation science it may be possible to design better methods of packing energetic materials that prevent convective burning and a catastrophic detonation in the event of a transportation accident.

We will address this problem through petascale simulation science. The target simulation scenario is an array of explosive devices (100's) that are heated non-uniformly as a result of a fire. The intention, and challenge, is to eventually predict the violence of explosion (as characterized by fragment velocities, pressures, etc.) and explosive yield (percentage of devices exploded) for common shipping configurations. This simulation capability will guide design improvements to shipping configurations that will mitigate the possibility of a DDT event in a transportation accident.

This goal requires advances in a) the computational infrastructure, b) the models used for the mechanical and reactive properties of the explosives, and c) the fluid-structure methodology. In this paper we describe, specific improvements including new sub-grid reaction models for the energetic materials, and continued evolution of the load-balancing and runtime components that facilitates exploiting adaptive multiple scales of parallelism within Uintah.

Figure 1 shows a preliminary simulation of the target scenario in which one exploding container (lower left) has caused a sym-

TeraGrid'10 TeraGrid10, August 25, 2010, Pittsburgh

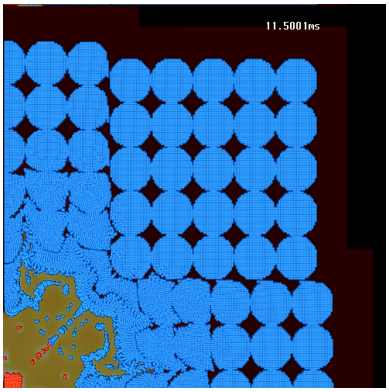

Figure 1: An array of 64 explosive cylinders ignited in the lower left corner.

pathetic explosion in the surrounding 64 containers. In this simulation only the deflagration mode of combustion was modeled. This low resolution prototyical simulation was run using the Uintah Software, a product of the University of Utah Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [9]. C-SAFE, a Department of Energy ASC center, focused on providing science-based tools for the numerical simulation of accidental fires and explosions, capable of running on 4K processors. Uintah has now been released as software[1] and extended to run on 98K processors through additional DOE and NSF funding. The benchmark C-SAFE problem was a multiphysics, large deformation, fluid-structure problem; a small cylindrical steel container filled with a plastic bonded explosive (PBX9501) subjected to convective and radiative heat fluxes from a fire. The incident heat flux caused the PBX to rapidly decompose into a gas above a critical temperature. The solid-to-gas reaction pressurized the interior of the steel container causing the shell to rapidly expand and eventually rupture. The gaseous products of reaction formed a blast wave that expanded outwards along with pieces of the container and unreacted PBX.

In order to solve complex multiscale multiphysics problems, such as this one, Uintah makes use of a component design that has also allowed it to excel as a research platform. Components enforce separation between large entities of software and can be swapped in and out, allowing them to be developed and tested within the entire framework, without affecting other components. This has led to a highly flexible simulation package that has been able to simulate a wide variety of problems including shape charges, stage-separation in rockets, the biomechanics of microvessels [13], the properties of foam under large deformation [6], and the evolution of large pool fires caused by transportation accidents [17], in addition to the exploding container described above.

Uintah currently contains three main simulation algorithms, or components, that are capable of using Adaptive Mesh Refinement (AMR): i) the ICE compressible multi-material CFD formulation, ii) the particle-based Material Point Method (MPM) for structural mechanics, and iii) the combined fluid-structure interaction algorithm MPMICE [12]. In addition, Uintah integrates numerous subcomponents including equations of state, constitutive models, and reaction models for deflagration and detonation.

ICE is a "multi-material" CFD algorithm that was originally developed by Kashiwa and others at LANL [16]. This technique can be used in both the incompressible and compressible flow regimes, which is necessary when modeling fires and explosions. This method conserves mass, momentum, energy, and the exchange of these quantities between materials and is used here on adaptive structured mesh meshes consisting of hexahedral patches often of 8^3 or 16^3 cells [18]. The Material Point Method is a particle method that is used to evolve the equations of motion for the solid materials. MPM is a powerful technique for computational solid mechanics, and has found favor in many applications involving complex geometries [13], large deformations [6], and fracture. Originally described by Sulsky, et al., [27], MPM is an extension to solid mechanics of the well-known particle-in-cell (PIC) method for fluid flow simulation, that uses the ICE mesh as a computational scratchpad. The fluid-structure methodology is a combination of the MPM and ICE [12, 25].

The Uintah component approach allows the application developers to only be concerned with solving the partial differential equations on a local set of block-structured adaptive meshes, without worrying about explicit message passing calls or notions of parallelization or load balancing. This approach also allows the developers of the underlying parallel infrastructure to focus on scalability concerns including load balancing, task (component) scheduling and communications. This component based approach to solving complex problems allows improvements in scalability to be immediately applied to applications without any additional work by the applications developer. The rest of this paper follows this philosophy in that the improvements in the parallel infrastructure are considered before the developments in combustion science.

Our experience with previous C-SAFE simulations of a single exploding container have shown that approximately 2K to 4K processors are required. To simulate our target scenario, 100 energetic devices, at a sufficient grid resolution will require petascale computations. It is thus important that the parallel infrastructure (load balancing, task/component scheduling and communication be able to scale to 98K cores and beyond. Sections 2-6 of this paper provide an overview of the improvements in load balancing and task scheduling needed to get closer to this goal. Despite the separation of components in Uintah, in Section 7 we describe a subtle example of how a mildly inefficient application code resulted in reduced scalability. Finally in Section 8 we describe the development of a DDT combustion model needed to begin to address our target simulation. The paper thus shows how computer science and combustion science combine to help to begin to address the challenge of modeling developing detonations in energetic materials.

2. THE UINTAH TASK-GRAPH

The heart of Uintah is a sophisticated computational framework that can integrate multiple simulation components, analyze the dependencies and communication patterns between them, and execute the resulting multi-physics simulation, [25]. The design of Uintah builds on the DOE Common Component Architecture (CCA) component model. Components are implemented as C++ classes that follow a very simple interface to establish connections with other components in the system. Uintah utilizes an abstract representation (called a task-graph) of parallel computation and communication to express data dependencies between multiple physics components. The task-graph is a directed acyclic graph of tasks. Each task consumes some input and produces some output (which is in turn the input of some future task). These inputs and outputs are specified for each patch in a structured AMR grid. Associated with each task is a C++ method which is used to perform the actual computation. Each component specifies a list of tasks to be performed and the data dependencies between them. The task-graph approach of Uintah shares many features with the migratable ob-

[1] see `http://www.uintah.utah.edu`

ject philosophy of Charm++ [15]. A scheduler component in Uintah sets up MPI communication for data dependencies and then executes the tasks that have been assigned to it. When a task completes, its outputs are sent to other tasks that require them.

These features allowed parallelism to be integrated between multiple components while maintaining overall scalability. The task-graph allows the Uintah runtime to analyze the structure of the computation to automatically enable load-balancing, data communication, parallel I/O, and checkpoint/restart.

A load balancer component is responsible for assigning each patch to a processor. Uintah's load balancer utilizes space-filling curves in order to cluster patches together [19]. In addition, it utilizes a cost-model associated with each patch to predict the workload which is then balanced across all processors.

After recent improvements in memory management and load balancing algorithms, Uintah has been shown to scale to tens of thousands of processors [20]. In scaling beyond this number of cores there are four main areas in which parallel performance has to be improved: (i) efficiency of task-graph execution, (ii) load balancing for fluid structure interaction problems, (iii) efficient scalable adaptive mesh refinement and (iv) elimination of algorithmic coding inefficiencies with regard to scalability. These areas will be addressed in turn in the following sections.

3. EFFICIENT TASK-GRAPH EXECUTION

Results from preliminary scaling studies on Kraken [2], showed that there was a substantial increase in MPI communication time at larger numbers of cores. We discovered that the time spent waiting for communication was due to dependencies between computing tasks distributed to different processors. In Uintah, the task scheduler component is responsible for a) computing the dependencies of tasks, b) determining the order of execution and c) ensuring that the correct inter-process communication is performed.

The Uintah task scheduler compiles all of the tasks and variable dependencies into a task-graph. Dependency edges are added between tasks based on the supplied variable dependencies. The computed dependency edges can be either internal or external. Internal dependencies are between patches on the same processor and external dependencies are between patches on different processors. Thus internal dependencies imply a necessary order where external dependencies specify required communication. The compilation process also combines external dependencies from the same source or to the same destination, thus coalescing messages.

Originally, Uintah used a static scheduler in which tasks were executed in a pre-determined order. This sometimes caused the simulation to sit idle when a single task was waiting for a message. Measurements showed that this type of delay was nearly 80 percent of total MPI waiting time in Uintah. The new dynamic scheduler changes the task order during the execution to overlap communication and computation, [22]. This scheduler required a large amount of development to support the out-of-order execution, which produced a significant performance benefit in lowering both the MPI wait time and the overall runtime. The dynamic scheduler utilizes two task queues: an internal ready queue and an external ready queue. If a task's internal dependencies are satisfied, then that task will be put in the internal ready queue where they will wait until all required MPI communication has finished. A counter of outstanding MPI messages is tracked for each task. When this counter reaches zero the communication is complete and the task is ready to be executed. At that point it is placed in the external ready queue.

[2]Kraken is an NSF supercomputer located at the University of Tennessee with 99,072 cores.

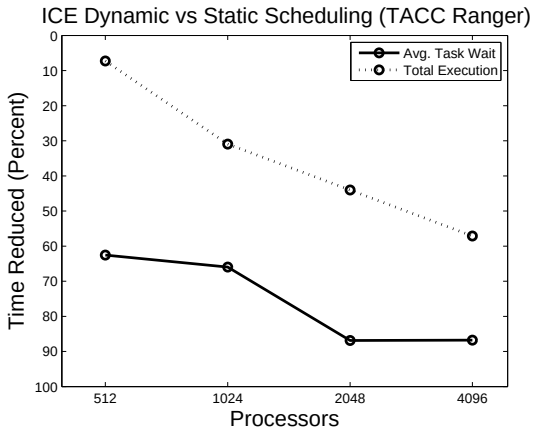

Figure 2: Reductions in Wait and Total Time from Dynamic Scheduling

When scheduling a task the scheduler chooses a task in the external ready queue based on a priority algorithm.

As long as the external queue is not empty, the processor always has tasks to run. This can help to overlap the MPI communication time with task execution. This approach reduces MPI wait times significantly. Figure 2 shows the percent reduction of both wait time (which is as high as 90% in some cases) and total execution time on a fixed mesh ICE problem on Ranger [3]. The example problem used is a two material compressible Navier Stokes type problem that models the movement of one material through another at high speed. 24,578 patches of 16^3 elements were used to solve this problem, [18].

Tasks that require a global communication, i.e., computation of the total mass of the system, require a specialized scheduling mechanism [22]. These tasks (which will be referred to as global tasks) are scheduled once per processor and execute on all patches assigned to that processor as opposed to once per patch. Due to the limitations of the MPI library, global communication requires a synchronization point, thus every processor must execute a global task at the same time. If global tasks are run out-of-order, a deadlock may occur.

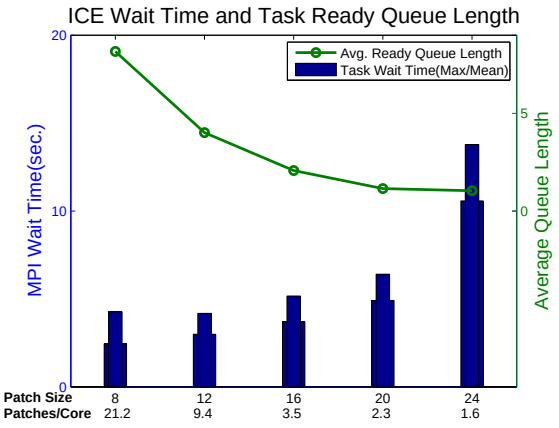

Figure 3: Queue length effects on wait times

[3]Ranger is an NSF supercomputer located at the University of Texas with 62,976 cores.

The Uintah dynamic scheduler uses the methods described in [22] to solve this problem. The tasks are divided into different phases where each phase contains only one global task. The scheduler only executes the global task if all of other tasks in its phase have completed. In addition, the scheduler allows non-global tasks to be executed in an earlier phase but not a later phase.

Uintah's patch design allows users to easily change the size and data layout so as to improve performance. Figure 3 shows that when running with the ICE component on 24K cores on Kraken, as the number of patches per core decreases so does the queue length of patches waiting to be executed, leading to an increase in the time spent waiting. When there are more patches-per-core the average length of the external ready queue is larger, creating an opportunity to reduce wait times and to overlap communications. Finally, Figure 4 shows that with smaller patches, the task wait time is small, but the overhead of regridding, data copying and scheduling is relatively large. As a result, the minimum overall execution time occurs when both wait time and regridding overhead are both relatively low.

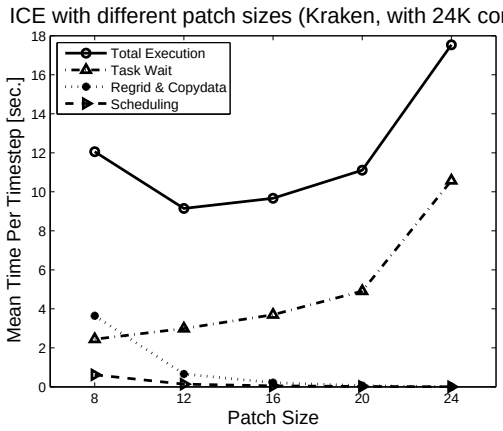

Figure 4: Granularity effects on execution time

4. EXPONENTIAL SMOOTHING LOAD BALANCER

Accurate workload prediction within Uintah is problematic with adaptive meshes. The complexity of the underlying physics complicates the process of deriving an accurate cost model. For example, MPM simulations have added load balancing complexity over ICE, because large numbers of computationally expensive particles move throughout the domain which may cause the workload to change at every timestep. To address this imbalance a new technique has been developed which uses forecasting methods to predict the cost of each patch based on observations made at runtime. During task execution, the time to complete each task is recorded and used to update a simple forecasting model which is then used to predict the time to execute on each patch in the future. This provides a mechanism to accurately predict the cost of each patch while requiring little information from the user or component developer.

One forecasting method used in Uintah is a simple exponential smoothing method [24], that has been used in a wide variety of applications because of its accuracy and simplicity. The model from [18] combines current core timing measurements and predictions as follows:

$$W_{r,t+1} = \alpha E_{r,t} + (1-\alpha) W_{r,t}, \tag{1}$$

where $\alpha = \frac{2.0}{T_s+1}$ and where $W_{r,t}$ is the predicted weight at timestep t for region r. $E_{r,t}$ is the actual execution time at timestep t for region r and α is a weighting factor in the range [0,1] which represents the rate of decay of past data. This method can also be viewed as a weighted moving average where the weight on past observations decreases exponentially. A smaller value for α places more weight on recent observations causing the forecast to respond more quickly to changes in the actual value but also causes the forecast to become more susceptible to noise. A larger value for α will cause that data to be smoother, thus eliminating noise but will also cause the forecast to react more slowly to changes in the actual value. The parameter α can be defined in terms of the size of a moving average window using the formula for α in equation (1), where T_s is the number of timesteps that will contain 99.9% of the total weight in the weighted average [24]. Uintah uses a default value of 20 for T_s. Further details of the algorithm are given in [18]. While measurement based load balancing has been used by others (e.g. by Charm++ [5]) our feedback mechanism appears to be novel.

5. A SCALABILITY METRIC FOR UINTAH

When attempting to scale codes that run well at small numbers of cores to large numbers of cores, such as the 98K cores available on Kraken, it is important to consider the efficiency and scalability of the code. Suppose that the problem execution time is defined by a function $T(n,p)$ where n is a measure of the problem size and p is the number of cores. The efficiency of a run of problem size of n running on p cores relative to execution on the smallest number of cores that the problem will fit on, say p_0, is given by the formula

$$E(n,p_0,p) = \frac{T(n,p_0)p_0}{T(n,p)p}, \tag{2}$$

where $0 < E(...) \leq 1$ unless there is superlinear speedup. Constant efficiency implies ideal strong scalability when the compute time for a fixed problem size is reduced by a factor of P when solved on p times as many cores. Weak scalability is defined by constant values of $W_s(n_0,p_0,p)$ where

$$W_s(n_0,p_0,p) = \frac{T(n_0p_0,p_0)}{T(n_0p,p)}, \tag{3}$$

and where n_0 is the problem size per core and p_0 is the lowest number of cores used with n_0p_0 being the total problem size on p_0 cores. Analysis of scalability and efficiency in [21] shows that *The system, is perfectly scalable if and only if the application has a linear computational complexity.* By perfect scalability we mean both strong and weak scalability. This may be measured by the product

$$SW_s(n_0,p_0,p) = W_s(n_0,p_0,p^*)E(p^*n_0,p^*,p) \times 100\% \tag{4}$$

for which a value of 100% implies perfect scalability with respect to the reference point of $T(n_0p_0,p_0)$ and where $T(p^*n_0,p^*)$ is an intermediate weak scaling value with p^* processors. By using equations (2) and (3) this measure may be re-written as:

$$SW_s(n_0,p_0,p) = \frac{T(n_0p_0,p_0)p^*}{T(n_0p^*,p)p} \times 100\% \tag{5}$$

The generalization of this is that a measure of the total scalability of a workload of $n_0\beta$ on p processors with respect to a workload of n_0p_0 on p_0 processors is given by:

$$SW_s(n_0,p_0,p) = \frac{T(n_0p_0,p_0)\beta}{T(n_0\beta,p)p} \times 100\%, p_0 \leq \beta \leq p. \tag{6}$$

This metric has the advantage that it is possible to quantify weak and strong scalability with only two runs of a problem. Calculating the exact workload $n_0\beta$ used on p cores may not always be easy for an adaptive mesh calculation, however. One requirement for achieving strong scalability is that the algorithms cannot have global data structures or operations that depend on the total number of cores p. Such restrictions and the need for linearity play a key role in the scalability of Uintah, see below.

6. SCALABLE AMR ALGORITHM

Uintah's task-graph structure of the computation makes it possible to improve scalability through adaptive self-tuning. However, the changing nature of the task-graph from adaptive mesh refinement poses extra challenges for scalability, [20].

Recent improvements within Uintah to the regridder and load balancer component have led to substantial improvements in the scalability of AMR. Previously, Uintah utilized the widely-used Berger-Rigoutsos algorithm, [4], to perform regridding. However, when using a large number of cores this algorithm performed poorly, requiring a new approach. The new regridder defines a set of fixed-sized tiles throughout the domain. Each tile is then searched, in parallel, for refinement flags without the need for communication. All tiles that contain refinement flags become patches. This regridder is advantageous at large scales because cores only communicate once at the end of regridding when the patch sets are combined, [18].

The scalability of Uintah's AMR infrastructure was tested in both the weak and strong sense using a test problem that required a dynamic grid with a constant total size, e.g., constant velocity, two material Navier Stokes calculation. The tests were run on Kraken and the time to complete 50 timesteps of the full AMR simulation was recorded. With 98K cores there are only one or two patches with 4096 cells per core. Experience suggests this is the point at which scalability begins to break down. This problem contained three mesh levels with each level being a factor of four more refined than the coarser level. Patches were uniformly sized with 16^3 cells in each patch. Regridding and load balancing were performed as needed and occurred around 5 times in each problem. The performance was tested for five problem sizes with each problem size containing approximately four times as many cells as the previous problem. The smallest problem contained 1.7 million cells and the largest problem contained 435 million cells. The numerical values

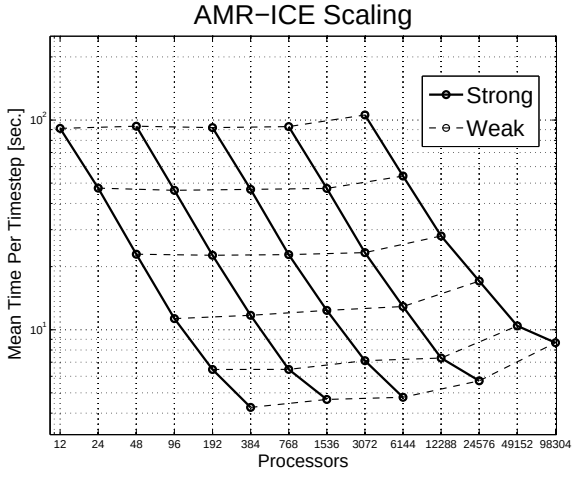

Figure 5: AMR Scaling Results for ICE

show that the rightmost set of results do not scale as well as the others. The weak and strong scaling results for up to 98,304 cores for

Weak	Strong	Run 3	Strong	Run 4	Strong	Run 5
Run	Cores	SWs	Cores	SWs	Cores	SWs
1	768	100	1536	99	3072	87
2	1536	98	3072	97	6144	85
3	3072	100	6144	98	12288	82
4	6144	92	12288	89	24576	67
5	12288	80	24576	78	49152	55
6	24576	59	49152	50	98304	33

Table 1: ICE with AMR SWs Metric values

ICE can be found in Figure 5 which shows six weak scaling sets of timings (horizontal dashed lines) and five strong scaling diagonal lines, [18]. Table 1 shows the SW_s metric for the three rightmost strong scaling runs and for the rightmost parts of the six weak scaling runs. The elimination of inefficiencies within Uintah and improvements to the load balancer have led to marked improvements in scalability. Good strong scaling occurred for every problem size tested. In each test scaling occurred down to approximately one patch per core. In the final run (rightmost solid line) (Strong 5) the scaling degenerates somewhat for large core counts. Using the data from [18], simple manipulation shows that the timings for the Strong 5 run follow the approximate formula

$$T(P) \approx T_0(P_0)\frac{P_0}{P} + log\left(\frac{P}{P_0}\right),$$

where the starting number of cores is $P_0 = 3072$ and $T_0(P_0) = 105.8$. The origin of the logarithmic term is not clear at present.

7. SCALABLE DATA STRUCTURES

One hurdle to scalability in AMR codes is a potential dependency on global meta-data, i.e. data that is replicated across the entire domain. Examples of this include the grid layout and the load balance information. Currently, every process must know the extents of every patch and which processors own which patches. As the number of patches grow the size of this global meta-data will also increase along with the time to create the data. This will eventually grow and become a significant problem.

Uintah has begun to reduce the amount of global meta-data used by using two methods. The first (and preferred) method is to eliminate the global dependency where possible. This has been done within the task-graph. Each core only creates tasks that are within its neighborhood, making the task-graph completely local. The second method has been to reduce the size of the data structures. For example, the patch data type has been divided into a light-weight and heavy-weight data type. When patches are being generated and communicated the light-weight data type is used. This data type only contains the coordinates of the low and high point of the patch. This allows the patches to be copied and communicated quickly. However, once the patch set has been finalized the light-weight patch data is turned into heavy-weight patch data, which include extra information that is required for the computation. Using this light-weight data structure helps minimize communication across the network. In addition, we have reduced the size of the heavy-weight patch data as much as possible. While this second method works well it is clear that it will insufficient when the number of patches is greater than, say, 100M. At this point we will have to move to either hierarchical or local algorithms.

In investigating the complexities of scaling with large numbers of cores it is important to realize that sometimes scaling fails be-

cause of quite innocent coding inefficiencies. For example Figure 6 shows the weak and strong scalability of the fixed mesh ICE

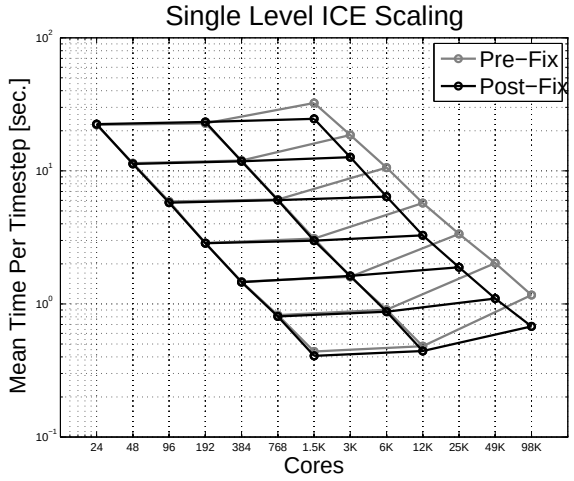

Figure 6: Before and after scaling results for static ICE

component for three problems sizes. Using profiling techniques we identified the cause of the poor scalability in the results labeled Pre-Fix as a query to the spatial extents of a level. This query would iterate through all N_p patches on the level and compute the spatial extents with a complexity of $O(N_p)$. This query itself is not a problem. However, an application developer was performing this query inside of a patch loop leading to an overall complexity of $O(N_p^2)$. These performance problems were resolved by computing and storing the spatial extents of a level at construction time making the query time $O(1)$. The results of this change can been seen in the Post-Fix results in Figure 6. At smaller numbers of processors this code was not detrimental to scalability but as the number of patches increased it quickly became one. In order to clarify the situation we apply the SW_s metric as defined by equation (6). the

ICE Pre-Fix Strong and Weak Efficiency						
Weak	Strong	Run 3	Strong	Run 4	Strong	Run 5
Run	Cores	SWs	Cores	SWs	Cores	SWs
1	24	100	192	97	1536	68
2	48	97	384	92	3072	59
3	96	94	768	92	6144	52
4	192	96	1536	89	12288	48
5	384	94	3072	86	24576	42
6	768	80	6144	72	49152	34
7	1536	78	12288	72	98304	29
ICE Post-Fix Strong and Weak Efficiency						
Weak	Strong	Run 3	Strong	Run 4	Strong	Run 5
Run	Cores	SWs	Cores	SWs	Cores	SWs
1	24	100	192	96	1536	91
2	48	99	384	95	3072	88
3	96	94	768	93	6144	87
4	192	98	1563	93	12288	85
5	384	96	3072	86	24576	74
6	768	81	6144	81	49152	74
7	1536	87	12288	79	98304	52

Table 2: Static mesh ICE with pre and post-fix SWs values.

metric values in Table 2 shows clearly that although scalability has greatly improved that there is still some way to go. This is a little less clear from Figure 6. This example shows the improvements that come from careful algorithmic analysis. We, and others, have observed that any doubling in the numbers of cores leads to having to either fix scaling inefficiencies or to make algorithmic changes. The above example suggests that there may be such problems in the AMR code close to 98K cores when there is comparatively little work per core.

8. PETASCALE SIMULATION OF SYMPATHETIC EXPLOSIONS.

An important goal of our NSF PetaApps project is to develop a science-based model of the deflagration-to-detonation transition (DDT) in high explosives. This transition from slow, thermally activated combustion to a fast pressure-driven detonation accounts for the majority of violent explosions in transportation accidents. It is therefore an essential component of our petascale simulations to enhance transportation safety.

In our recent work, we have combined the two validated Uintah combustion models for deflagration [10] and detonation [26] into a single model with a rudimentary switch to simulate the DDT. This model has been validated against impact experiments for PBX9501 conducted at Los Alamos [14] with generally good results. However, this model fails to capture much of the basic physics and mechanics of DDT, which relies on damage, porosity, convective burning and inertial confinement. The next stage of algorithm development will be to build these characteristics into the DDT model in ways that can be validated by existing experimental results. Results from [14] as well those of [8] are the main experimental data to be validated against.

Uintah incorporates reaction models that convert mass from energetic materials (e.g., the plastic-bonded explosive PBX9501) to product gases, with the appropriate release of heat. From a computational point of view, the models simply represent mass sources and sinks that take place on the Eulerian grid.

The numerical model for deflagration in energetic materials is based on a two-step global chemistry model described by Ward, Son and Brewster (WSB) [28]. As originally formulated, this model predicts the steady combustion rate of energetics as a function of pressure and initial temperature of the solid material. We extended the 1-D WSB analytic model to 3-D and validated the parameters against the experimental strand burner measurements of Atwood et al. [2]. Good agreement was obtained for the pressure-dependence of the burn rate as well as the dependence on initial (bulk) temperature of the PBX9501 [10].

The detonation component of the reaction model is adapted from Souers' JWL++ model [26]. The ignition and growth parameters of the model were adjusted to produce the desired detonation velocity (8800 m/s) and detonation pressure (35 GPa) appropriate for PBX9501. The constitutive model for PBX9501 uses the ViscoSCRAM formulation developed at Los Alamos National Laboratory [3]. The Murnaghan equation of state uses parameters published by Gibbs and Papolato [11]. Detonation product gases are modeled using a JWL equation of state using published parameters [23]. The detonation model was validated against low-amplitude impact test data (Steven's Test) [7,14]. The results showed excellent agreement between simulation and experiment when the detonation threshold pressure was set to 5 GPa. This value also gives good agreement for aluminum flyer plate shock detonation tests [8]. Run-to-detonation lengths were on the lower bound of the experimental values, but the slopes of the experiment and simulated Pop plots are identical to within experimental error.

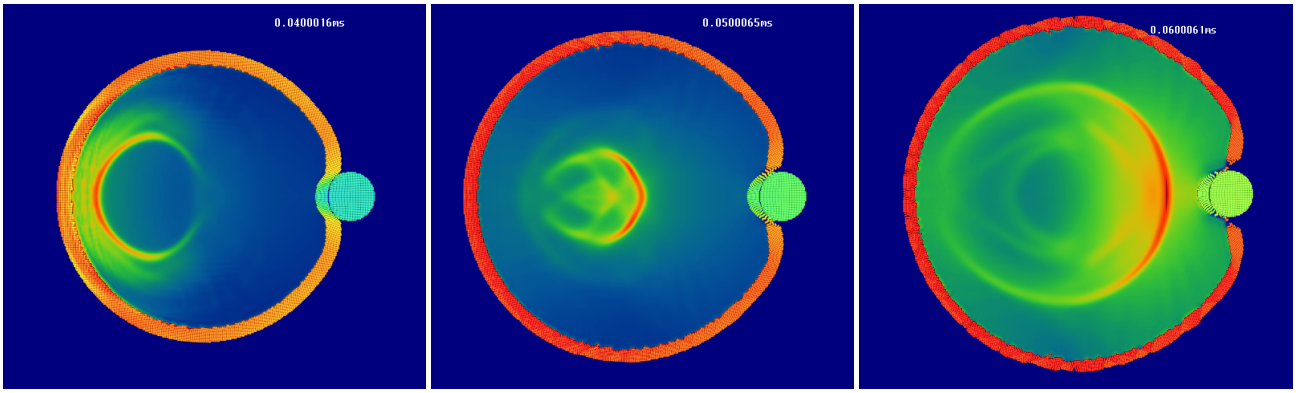

Figure 7: Montage of a 2-D steel container filled with a energetic material that is experiencing a detonation reaction.

One of the advantages of using the ViscoSCRAM constitutive model is that the crack radius is a temporally evolving quantity that is accessible by the reaction models. We are currently utilizing this variable as a method of allowing convective burning in the energetic material as it suffers damage due to stresses and strains that are imposed externally as well as caused by reaction and pressurization.

Figures 7 and 8 illustrate our current code capabilities. Figure 7 shows a 2-D simulation of a 4 inch diameter steel container initially filled with an energetic material undergoing a detonation reaction. To initiate the reaction a projectile (shown on the right) traveling at 500m/s collided with the steel shell creating a shock wave inside of the energetic material. This shock wave started a detonation reaction which moved from left to right rapidly pressurizing the container and contents. The container was modeled using MPM particles that are colored by the magnitude of the velocity of the particles. The contour plot inside of the container is colored by the pressure field. The top image shows the detonation wave approaching the left container wall. In the middle image the wave has reflected off of the left wall, focused to a point and is expanding as it travels to the right. In the rightmost image the wave is nearing the impact zone. A variable scale color map was used to accentuate the wave pattern. Figure 8 show a 3-D visualization of the scenario described above. In this image the steel case MPM particles are color coded by the temperature.

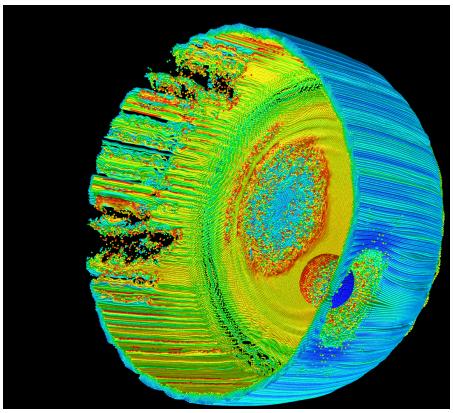

Figure 8: 3-D steel container filled with an energetic material experiencing a detonation reaction

These preliminary results require further work on both the combustion models and on the scalability in order for this scenario to be simulated with sufficient resolution and with multiple containers on 200-300K cores. Achieving this level of fidelity and scalability is the next challenge of this PetaApps project.

9. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under subcontract No. OCI0721659. Uintah was written by the University of Utah's Center for the Simulation of Accidental Fires and Explosions (C-SAFE) and funded by the Department of Energy, subcontract No. B524196. We would like to thank TACC and NICS for access to large numbers of cores.

10. REFERENCES

[1] B. W. Asay. *Shock Wave Science and Technology Reference Library, Volume 5, Non-shock initiation of explosives*. Springer-Verlag, Berlin, 2010.

[2] A. I. Atwood, T. L. Boggs, P. O. Curran, T. P. Parr, D. M. Hanson-Parr, C. F. Price, and J. Wiknich. Burning rate of solid propellant ingredients, part 1: Pressure and initial temperature effects. *Journal of Propulsion and Power*, 15:740–747, 1999.

[3] 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(12):2303–2322, 1998.

[4] M. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Trans. Systems Man Cybernet.*, 21(5):1278–1286, 1991.

[5] A. Bhatelé, L. V. Kalé, and S. Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 110–116. ACM, 2009.

[6] A. D. Brydon, S. G. Bardenhagen, E. A. Miller, and G. T. Seidler. Simulation of the densification of real open–celled foam microstructures. *J. Mech. Phys. Solids*, 53:2638–2660, 2005.

[7] S. K. Chidester, R. Garza, and C. M. Tarver. Low amplitude impact testing and analysis of pristine and aged solid high explosives. Technical report, UCRL-JC-127963, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 1998.

[8] S. K. Chidester, D. G. Thompson, K. S. Vandersall, D. J. Idar, C. M. Tarver, F. Garcia, and P. A. Urtiew. Shock

Initiation Experiments on PBX 9501 Explosive at Pressures below 3 GPa with Associated Ignition and Growth Modeling. In *Shock Compression of Condensed Matter- 2007: Proceedings of the Conference of the American Physical Society Topical Group on Shock Compression of Condensed Matter(2007 APS SCCM) Part Two*, volume 955, pages 903–906, 2007.

[9] 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*, pages 33–41. IEEE, Piscataway, NJ, November 2000.

[10] E. G. Eddings and C. A. Wight. Science based simulation tools for hazard assessment and mitigation. In *Advancements in Energetic Materials and Chemical Propulsion*, pages 921–937, 2008.

[11] T. R. Gibbs and A. Popalato, editors. *LASL Explosive Property Data*. University of California Press, 1980.

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

[13] J. E. Guilkey, J. B Hoying, and J. A. Weiss. Modeling of multicellular constructs with the material point method. *Journal of Biomechanics*, 39:2074–2086, 2007.

[14] D. J. Idar, R. A. Lucht, J. W. Straight, R. J. Scammon, R. V. Browning, J. Middleditch, J. K. Dienes, C. B. Skidmore, and G. A. Buntain. Low amplitude insult project: Pbx 9501 high explosive violent reaction experiments. Technical Report LA-UR–98-3366, Los Alamos National Laboratory, Los Alamos, 1998.

[15] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng. Programming petascale applications with Charm++ and AMPI. *Petascale Computing: Algorithms and Applications*, 1:421–441, 2007.

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

[17] G. Krishnamoorthy, S. Borodai, R. Rawat, J. P. Spinti, and P. J. Smith. Numerical modeling of radiative heat transfer in pool fire simulations. In *ASME International Mechanical Engineering Congress (IMECE), Orlando, Florida*, 2005.

[18] J. Luitjens and M. Berzins. Improving the performance of Uintah: A large-scale adaptive meshing computational framework. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS10)*, 2010.

[19] J. Luitjens, M. Berzins, and T. Henderson. Parallel space-filling curve generation through sorting: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(10):1387–1402, 2007.

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

[21] I. Martin and F. Tirado. Relationships between efficiency and execution time of full multigrid methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):562–573, 1997.

[22] Q. Meng, J. Luitjens, and M. Berzins. Dynamic task scheduling for scalable parallel AMR in the Uintah framework. SCI Technical Report UUSCI-2010-001, University of Utah, 2010.

[23] R. Menikoff and T. D. Sewell. Complete equation of state for beta-HMX and implications for initiation. *Shock Compression of Condensed Matter*, pages 157–160, 2004.

[24] D. C. Montgomery, L. A. Johnson, and J. S. Gardiner. *Forecasting and time series analysis*. McGraw-Hill, 1990.

[25] S. G. Parker, J. Guilkey, and T. Harman. A component-based parallel infrastructure for the simulation of fluid-structure interaction. *Engineering with Computers*, 22:277–292, 2006.

[26] P. C. Souers, R. Garza, and P. Vitello. Ignition & Growth and JWL++ Detonation Models in Coarse Zones. *Propellants, Explosives, Pyrotechnics*, 27(2):62–71, 2002.

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

[28] 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(3-4):556–568, 1998.