Engineering with Computers (2006) 22:277–292
DOI 10.1007/s00366-006-0047-5

ORIGINAL ARTICLE

A component-based parallel infrastructure for the simulation of fluid–structure interaction

Steven G. Parker · James Guilkey · Todd Harman

Received: 19 April 2005 / Accepted: 1 February 2006 / Published online: 7 November 2006
© Springer-Verlag London Limited 2006

Abstract The Uintah computational framework is a component-based infrastructure, designed for highly parallel simulations of complex fluid–structure interaction problems. Uintah utilizes an abstract representation of parallel computation and communication to express data dependencies between multiple physics components. These features allow parallelism to be integrated between multiple components while maintaining overall scalability. Uintah provides mechanisms for load-balancing, data communication, data I/O, and checkpoint/restart. The underlying infrastructure is designed to accommodate a range of PDE solution methods. The primary techniques described here, are the material point method (MPM) for structural mechanics and a multi-material fluid mechanics capability. MPM employs a particle-based representation of solid materials that interact through a semi-structured background grid. We describe a scalable infrastructure for problems with large deformation, high strain rates, and complex material behavior. Uintah is a product of the University of Utah Center for Accidental Fires and Explosions (C-SAFE), a DOE-funded Center of Excellence. This approach has been used to simulate numerous complex problems, including the response of energetic devices subject to harsh environments such as hydrocarbon pool fires. This scenario involves a wide range of length and time scales including a relatively slow heating phase punctuated by pressurization and rupture of the device.

Keywords Fluid–structure interaction · Parallel computing · Software framework · Material point method

S. G. Parker (✉)
School of Computing and Scientific Computing and Imaging (SCI) Institute, University of Utah,
Salt Lake City, UT, USA
e-mail: sparker@cs.utah.edu

J. Guilkey · T. Harman
Department of Mechanical Engineering,
University of Utah, Salt Lake City, UT, USA

1 Introduction and motivation

The University of Utah Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [1] is a Department of Energy ASC center that focuses on providing state-of-the-art, science-based tools for the numerical simulation of accidental fires and explosions, especially within the context of handling and storing highly flammable materials. The primary objective of C-SAFE is to provide a software system in which fundamental chemistry and engineering physics are fully coupled with nonlinear solvers, visualization, and experimental data verification, thereby integrating expertise from a wide variety of disciplines. Simulations using this system will help to better evaluate the risks and safety issues associated with fires and explosions in accidents involving both hydrocarbon and energetic materials.

The Uintah software system was primarily designed to support the solution of a wide range of highly dynamic physical processes on a large number of processors. Specifically, our target simulation is the heating of a explosive device placed in a large hydrocarbon pool fire and the subsequent deflagration explosion and blast wave. The explosive device is a small cylindrical steel container (4'' outside diameter) filled with plastic bonded explosive (PBX-9501). Convective and radiative heat fluxes from the fire heat the outside of the container and the PBX. After some period of time, the critical temperature in the PBX is reached and the explosive begins to rapidly decompose into a gas. The solid $\rightarrow$ gas reaction pressurizes the interior of the steel container causing the shell to rapidly expand and eventually rupture. The gaseous products of reaction form a blast wave that expands outward along with pieces of the container and the unreacted PBX. The physical processes in this simulation have a wide range in time and length scales from microseconds and microns to minutes and meters. An example of this simulation is depicted in Fig. 1, and discussed further in Sect. 5.2. Uintah is designed to be a general-purpose fluid–structure code that will simulate not only this scenario but a wide range of related problems.

Complex simulations such as this require both immense computational power and complex software. Typical simulations include solvers for structural mechanics, fluids, chemical reactions, and material models. All of these aspects must be integrated in an efficient manner to achieve the scalability required to perform these simulations. The heart of Uintah is a sophisticated computational framework that can integrate multiple simulation components, analyze the dependencies and communication patterns between them, and efficiently execute the resulting multi-physics simulation. Uintah also provides mechanisms for automating load-balancing, checkpoint/restart, and parallel I/O. While this document focuses on two major components in Uintah, it has also been used for the Arches fire simulation code [2], and a handful of demonstration components. The Uintah core is designed to be general, and would be appropriate to use for a wide range of PDE algorithms based on structured (possibly adaptive) grids and particle-in-cell (PIC) algorithms.

We discuss the approach we use for the Uintah fluid solver and the fluid–structure interaction algorithm in Sect. 2. The Uintah computational framework is described in Sect. 3, and the details of particular Uintah components that build on this framework are discussed in Sect. 4. Results from these simulations are presented in Sect. 5.

2 Fluid–structure interaction methodology

Here, we describe our approach to ‘‘full physics’’ simulations of fluid–structure interactions involving large deformations and phase change. By ‘‘full physics’’, we refer to problems involving strong coupling between the fluid and solid phases with a full Navier–Stokes representation of fluid phase materials and the transient, nonlinear response of solid phase materials, which may include chemical or phase transformation between the solid and fluid phases.

2.1 Modeling equations

The methodology upon which our approach is built is a full ‘‘multi-material’’ approach, in which each material is given a continuum description and defined over the complete computational domain. Although at any point in space the material composition is uniquely defined, the multi-material approach adopts a statistical viewpoint, whereby the material (either fluid or solid) resides with some finite probability. To determine the probability of finding a particular material at a specified point in space, together with its current

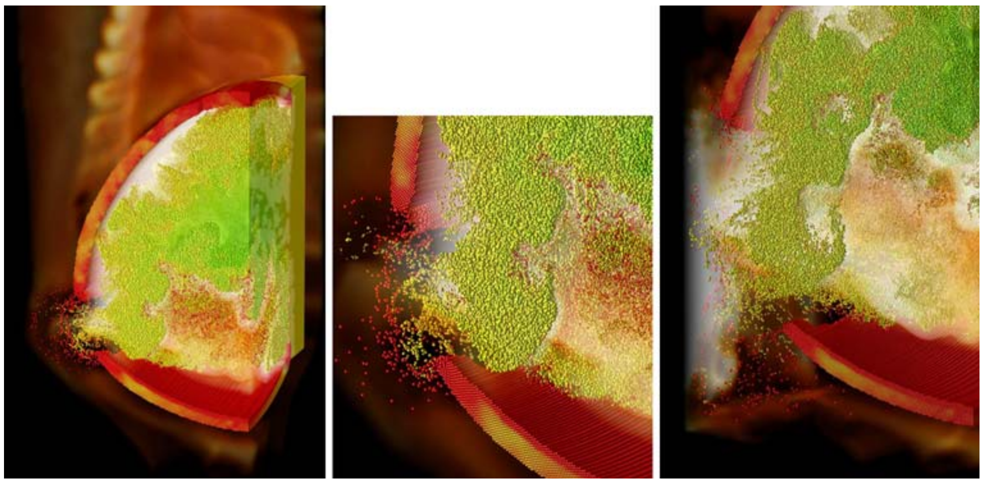
Fig. 1 Inside view of an energetic device at the point of rupture

state (i.e., mass, momentum, energy), multi-material equations are used. We describe below an algorithm that uses a common framework to treat the coupled response of a collection of arbitrary materials. This follows the ideas previously presented by Kashiwa and colleagues [3, 4].

Our description of the methodology begins by stating and describing the multi-material conservation equations for mass, momentum and energy. A thorough description of the development of these equations can be found in [5, 6]. The volume and ensemble averaged equations are

$$\frac{1}{V}\frac{D_r M_r}{Dt} = \Gamma_r \tag{1}$$

$$\frac{1}{V}\frac{D_r(M_r\mathbf{u}_r)}{Dt} = \theta_r\nabla\cdot\boldsymbol{\sigma} + \nabla\cdot\theta_r(\boldsymbol{\sigma}_r - \boldsymbol{\sigma}) + \rho_r\mathbf{g} + \sum_{s=1}^{N}\mathbf{f}_{rs} + \sum_{s=1}^{N}\mathbf{u}_{rs}\Gamma_{rs} \tag{2}$$

$$\frac{1}{V}\frac{D_r(M_r e_r)}{Dt} = -\rho_r p\frac{D_r v_r}{Dt} + \theta_r\boldsymbol{\tau}_r : \nabla\mathbf{u}_r - \nabla\cdot\theta_r\mathbf{q}_r + \rho_r\varepsilon_r + \sum_{s=1}^{N}h_{sr} + \sum_{s=1}^{N}(e+pv)_{rs}\Gamma_{rs} \tag{3}$$

where the right side of each equation is the averaged rate occurring in V. Also, the averaged r-field internal energy comes from $E_r = e_r + \frac{1}{2}u_r^2$; in which e_r typically includes all internal modes (translational + vibrational + rotational + chemical).

Equations 1, 2 and 3 are the averaged model equations for mass, momentum and internal energy of r-material in the volume V, (which can be thought of as a computational cell). M_r is the mass contained in that volume. In Eq. 1, Γ_r refers to the rate at which r-material mass is generated or depleted from the volume due to conversion from other materials, typically due to chemical reaction, while Γ_{rs} in Eqs. 2 and 3 refers to the rate of conversion of mass between the r-material and another s-material. $\mathbf{u}_r$, θ_r, $\boldsymbol{\sigma}_r$ and ρ_r refer to the r-material velocity, volume fraction, stress and density, respectively. The non-subscripted $\boldsymbol{\sigma}$ is the mean mixture stress, taken here to be isotropic, so that $\boldsymbol{\sigma} = -p\mathbf{I}$ in terms of the hydrodynamic pressure p. In this, the deviatoric part of the material stress is given by $\boldsymbol{\tau}_r = \boldsymbol{\sigma}_r - \frac{1}{3}\mathbf{I}Tr(\boldsymbol{\sigma}_r)$.

In Eqs. 2 and 3 quantities related to the multi-material Reynolds stress, which must be modeled, are neglected and were not included in the current work. In Eq. 2 the term $\sum_{s=1}^{N}\mathbf{f}_{rs}$ signifies a model for the momentum exchange among materials. This term results from the deviation of the r-field stress from the mean stress, averaged. This is typically modeled as a function of the relative velocity between materials at a point. (For a two material problem this term might look like $\mathbf{f}_{12} = K_{12}\theta_1\theta_2(\mathbf{u}_1 - \mathbf{u}_2)$ where the coefficient K_{12} determines the rate at which momentum is transferred between materials.) Likewise, in Eq. 3, $\sum_{s=1}^{N}h_{sr}$ represents an exchange of heat energy among materials and for a two- material problem typically takes the form $h_{12} = H_{12}\theta_1\theta_2(T_2 - T_1)$ where T_r is the r-material temperature and the coefficient H_{rs} is analogous to a convective heat transfer rate coefficient. The heat flux is given by $\mathbf{q}_r = -\rho_r b_r\nabla T_r$ where the kinematic thermal diffusion coefficient b_r is an effective one that contains both molecular and turbulent effects (when the turbulence is included).

In Eqs. 2 and 3, the velocity $\mathbf{u}_{rs}$ and the enthalpy $(e + pv)_{rs}$ are the velocity and enthalpy converted to/or from the s-material to the r-material. The conversion of a solid propellant to gaseous products of reaction according to a model is an example of such a process.

As usual, individual equations of state are needed for each material to determine the relationships between pressure, density, temperature and internal energy. Constitutive models are also required to describe the stress for each material, based on appropriate input parameters (deformation, strain rate, history variables, etc.). In addition to those, the multi-material nature of the equations also requires closure for the volume fraction of each material, θ_r. The equations that provide this closure are

$$\theta_r = \rho_r v_r \tag{4}$$

$$0 = 1 - \sum_{s=1}^{N}\rho_s v_s(p_{eq}) \tag{5}$$

where p_{eq} is the unique pressure that satisfies Eq. 5. We now turn our attention to the algorithmic approach used to solve the above relationships.

The FSI method described here involves integrating a Lagrangian particle method into a general multi-material CFD formulation. A brief description of the main elements of this approach follows.

2.2 Multi-material CFD approach

The basis of the multi-material CFD formulation used here is the ICE (for Implicit, Continuous-fluid, Eulerian) method [7], further developed by Kashiwa and others at Los Alamos National Laboratory [8]. The use of a cell centered, finite volume approach is convenient for multi-material simulations in that a single control volume is used for all materials. This is particularly important in regions where a material volume goes to zero. By using the same control volume for mass and momentum it can be ensured that as the material volume goes to zero, the mass and momentum go to zero at the same point. The technique is fully compressible, allowing wide generality in the types of problems that can be simulated.

Our implementation of the ICE technique invokes operator splitting, in which the solution consists of a separate Lagrangian phase where the physics of the conservation laws is computed (e.g., right hand side of Eqs. 2, 3), and an Eulerian phase, where the material state is transported via advection to the surrounding cells. The general solution approach is well developed and described in [8–10]. Where appropriate, details specific to the solution of FSI problems are expounded upon in Sect. 2.4.

A physical and numerical requirement of this multi-material approach is that the total volume fraction of material must sum to one in each cell. This is achieved by solving Eq. 5 in an iterative manner in each computational cell each timestep, giving a so-called equilibrium pressure in that cell, as well as values for the specific volume for each material. This is one of the unique features of the multi-field formulation, relative to standard CFD techniques. The other, is the transfer of mass, momentum and energy between materials. In this algorithm, arbitrary amounts of all materials may be present in any computational cell (as opposed to other FSI approaches that segregate materials into distinct regions of the computational domain). Thus, instead of the transfer between materials occurring at boundaries, it takes place within computational cells. Mass transfer is typically "one-way" and evolves according to governing equations that describe a reaction. Momentum and heat transfer between materials is driven (in our formulation) by relative differences in velocity and temperature, respectively. A point-wise implicit solution for each component of velocity and temperature is carried out as the last step in the Lagrangian phase of the calculation.

2.3 The material point method

Here, we briefly describe a particle method known as the material point method (MPM) which 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 applications involving complex geometries [11], large deformations [12] and fracture [13], to name a few. After the description of MPM, its incorporation within the multi-material CFD approach is described in Sect. 2.4.

Originally described by Sulsky et al. [14, 15], MPM is a particle method for structural mechanics simulations. MPM is an extension to solid mechanics of FLIP [16], which is a PIC method for fluid flow simulation. Lagrangian particles or material points are used to discretize the volume of a material, each particle carries state information (e.g. mass, volume, velocity, and stress) about the portion of the volume that it represents. The method typically uses a Cartesian grid as a computational scratchpad for computing spatial gradients. This same grid also functions as an updated Lagrangian grid that moves with the particles during advection and thus, eliminates the diffusion problems associated with advection on an Eulerian grid. At the end of a timestep, the grid is reset to the original, regularly ordered, position.

In explicit MPM, the equations of motion are cast in the form [15]:

$$\mathbf{ma} = \mathbf{F}^{\text{ext}} - \mathbf{F}^{\text{int}} \tag{6}$$

where $\mathbf{m}$ is the mass matrix, $\mathbf{a}$ is the acceleration vector, $\mathbf{F}^{\text{ext}}$ is the external force vector (sum of the body forces and tractions), and $\mathbf{F}^{\text{int}}$ is the internal force vector resulting from the divergence of the material stresses. An implicit formulation of MPM is discussed in [17].

The solution procedure begins by projecting the particle state to the nodes of the computational grid using a suitable shape function, to form the mass matrix $\mathbf{m}$ and to find the nodal external forces $\mathbf{F}^{\text{ext}}$, and velocities, $\mathbf{v}$. In practice, a lumped mass matrix is usually used. These quantities are calculated at individual nodes by the following equations, where the $\sum_p$ represents a summation over all particles:

$$m_i = \sum_p S_{ip} m_p \tag{7}$$

$$\mathbf{v}_i = \frac{\sum_p S_{ip} m_p \mathbf{v}_p}{m_i} \tag{8}$$

$$\mathbf{F}_i^{\text{ext}} = \sum_p S_{ip} \mathbf{F}_p^{\text{ext}} \tag{9}$$

and i refers to individual nodes of the grid. m_p is the particle mass, $\mathbf{v}_p$ is the particle velocity, and $\mathbf{F}_p^{\text{ext}}$ is the external force on the particle. S_{ip} is the shape function of the ith node evaluated at $\mathbf{x}_p$.

A velocity gradient, $\nabla \mathbf{v}_p$ is computed at the particles using the velocities projected to the grid:

$$\nabla \mathbf{v}_p = \sum_i \mathbf{G}_{ip} \mathbf{v}_i \tag{10}$$

where $\mathbf{G}_{ip}$ is the gradient of the shape function of the *ith* node evaluated at $\mathbf{x}_p$.

This is used as input to a constitutive model which is evaluated on a per particle basis, the result of which is the Cauchy stress at each particle, $\boldsymbol{\sigma}_p$. With this, the internal force due to the divergence of the stress is calculated via

$$\mathbf{F}_i^{\text{int}} = \sum_p \mathbf{G}_{ip} \boldsymbol{\sigma}_p V_p, \tag{11}$$

where V_p is the particle volume.

Equation 6 can then be solved for $\mathbf{a}$. An explicit forward Euler method is used for the time integration:

$$\mathbf{v}^L = \mathbf{v} + \mathbf{a}\Delta t \tag{12}$$

and the particle position and velocity are explicitly updated by

$$\mathbf{v}_p(t+\Delta t) = \mathbf{v}_p(t) + \sum_i S_{ip} \mathbf{a}_i \Delta t \tag{13}$$

$$\mathbf{x}_p(t+\Delta t) = \mathbf{x}_p(t) + \sum_i S_{ip} \mathbf{v}_i^L \Delta t \tag{14}$$

This completes one MPM timestep. This process is depicted graphically in Fig. 2. This process involves several particle-based operations, grid-based computations, and mass-weighted interpolations between the grids and particles.

2.4 Integration of the material point method within the multi-material CFD approach

As described above, MPM makes use of a computational grid for the solution of the governing equations. The grid used is arbitrary and can be the same grid used by the accompanying multi-material CFD component. Then, a further projection of the physical state of the solid from the computational nodes to the cell centers colocates the solid material state with that of the fluid. This common reference frame is used for all physics that involve mass, momentum, or energy exchange among the materials. This allows for a tight coupling between the fluid and solid phases. The coupling occurs through terms in the conservation equations, rather than explicitly through specified

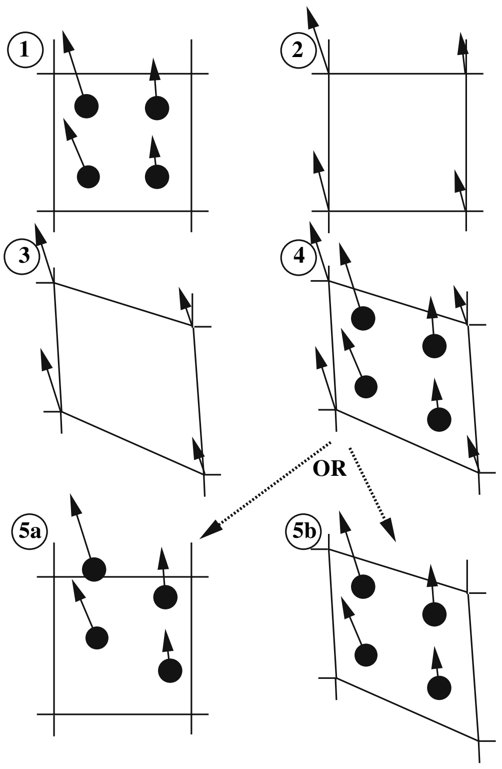

Fig. 2 Graphical depiction of the material point method

boundary conditions at interfaces between materials. Since a common multi-field reference frame is used for interactions among materials, typical problems with convergence and stability of solutions for separate domains communicating only through boundary conditions are alleviated. While the primary description of the solid phase materials remains at the particles, during the course of a single timestep, it also has a representation in the same volume as the Eulerian based materials. Once the Lagrangian phase of the solution of the multi-field equations is complete (including momentum and heat transfer), increments to the solid materials' state are interpolated to the nodes and ultimately to the particles and the state at the particles, including the position, is updated.

It is primarily in advection, as well as in the computation of internal forces, that the use of the separate reference frames becomes important. Eulerian advection is typically subject to significant diffusion. Therefore if the Eulerian frame is used exclusively for both solid and fluid materials, the interface between the materials will become smeared and nonphysical behavior may result. The use of a particle description for the solid advection minimizes this problem. Furthermore, while straining history does not typically play a role in the stress field in a fluid, it is important in many engineering solid materials to describe phenomenon such as plasticity. The particle description of the solid provides a convenient frame to evaluate the solid material stress, and to store and carry forward in time the relevant history variables. This role of the particle is similar to the role that Gaussian integration points play in finite element method formulations. On the other hand, if a particle description is used for fluid phases, the random behavior of general fluid motion will generally result in very random particle distributions. This limits the utility of the MPM for fluid calculations. However, the integration of the two, where part of the calculation takes place in a common reference frame, allows each material phase to enjoy its optimum description and achieves a tight coupling.

3 Computational framework

The Uintah computational framework consists of a set of software components and libraries that facilitate the solution of partial differential equations (PDEs) on structured AMR (SAMR) grids using hundreds to thousands of processors.

One of the challenges in designing a parallel, component-based multi-physics application is determining how to efficiently decompose the problem domain. Components, by definition, make local decisions. Yet parallel efficiency is only obtained through a globally optimal domain decomposition and scheduling of computational tasks. Typical techniques include allocating disjoint sets of processing resources to each component, or defining a single domain decomposition that is a compromise between the ideal load balance of multiple components. However, neither of these techniques will achieve maximum efficiency for complex multi-physics problems.

Uintah uses a non-traditional approach to achieving parallelism, employing an abstract taskgraph representation to describe computation and communication. The taskgraph is an explicit representation of the computation and communication that occur in the coarse of a single iteration of the simulation (typically a timestep or nonlinear solver iteration). Uintah components delegate decisions about parallelism to a scheduler component, using variable dependencies to describe communication patterns and characterizing computational workloads to facilitate a global resource optimization. The taskgraph representation has a number of advantages, including efficient fine-grained coupling of multi-physics components, flexible load balancing mechanisms and a separation of application concerns from parallelism concerns. However, it creates a challenge for scalability, which we overcome by creating an implicit definition of this graph and representing it in a distributed fashion.

We describe the major pieces in the Uintah framework and then describe how those pieces are assembled to perform the fluid–structure simulations described above.

3.1 Components

The fundamental design methodology in Uintah is a software component. Components enforce separation between large entities of software and enable large-scale applications to be constructed out of smaller, isolated pieces. 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.

The interfaces between components are simplified because the components do not explicitly communicate with one another. A typical component implements a handful of methods, including:

- problemSetup: parse input parameters from a section of an XML document;
- scheduleComputStableTimestep: schedule tasks that compute a stable timestep for the next integration interval;
- scheduleTimeAdvance: schedule tasks that complete a timestep integration.

It is important to note that no computation is performed in these methods; the component simply defines the steps in the algorithm that will be done later and specifies the C++ method that will perform each of those steps. The details of this mechanism will be described in the following section.

The components for a target simulation are assembled from an XML input file specification.

The primary advantage of a component-based approach is that it facilitates the separate development of simulation algorithms, models, and infrastructure. Components of the simulation can evolve independently. Since C-SAFE is a research project, we need to accommodate the fact that most of the software is still under development. The component-based architecture allows pieces of the system to be implemented in a rudimentary form at first and then evolve as the technologies mature. Most importantly, Uintah allows the aspects of parallelism (schedulers, load-balancers, parallel input/output, and so forth) to evolve independently of the simulation components. This approach allows the computer science effort to focus on these problems without waiting for the completion of the scientific applications or vice-versa. Furthermore, components enable replacement of computation pieces without complex decision logic in the code itself.

3.1.1 Tensor product task graphs

Uintah enables integration of multiple simulation algorithms by adopting an execution model based on ‘‘macro’’ dataflow. Each component specifies the steps in the algorithm and the data dependencies between those steps. These steps are combined into a single graph structure (called a *taskgraph*). The taskgraph represents the computation to be performed in a single timestep integration, and the data dependencies between the various steps in the algorithm. Graphs may specify numerous exchanges of data between components (fine-grained coupling) or few (coarse-grained coupling), depending on the requirements of the underlying algorithm. The fluid–structure algorithm described above requires several points of data-exhcange in a single timestep to achieve the tight coupling between the fluid and solids. This contrasts with approaches that exchange boundary conditions at fluid–solid interfaces. The taskgraph structure allows fine-grained interdependencies to be expressed in an efficient manner.

The taskgraph is a directed acyclic graphs of tasks, each of which produces some output and consumes some input (which is in turn the output of some previous task). These inputs and outputs are specified for each patch in a structured, possibly AMR, grid. Associated with each task is a C++ method which is used to perform the actual computation. Uintah data structures are compatible with Fortran arrays, so that the application writer can also use Fortran subroutines to provide numeric kernels on each patch.

A taskgraph representation by itself works well for the coupling of multiple computational algorithms, but presents challenges for achieving scalability. A taskgraph that represents all communication in the problem, would require time, proportional to the number of computational elements to create. Creating this on a single processor, or on all processors would eventually result in a bottleneck. Uintah addresses this problem by introducing the concept of a ‘‘tensor product taskgraph’’. Uintah components specify tasks for the algorithmic steps only, which are independent of the problem-size or number of processors. Each task in the taskgraph is then implicitly repeated on a portion of patches in the decomposed domain. The resulting graph, or tensor product taskgraph, is created collectively; each node contains only the tasks that it owns and those that it communicates with. The graph exists only as a whole across all computational elements, resulting in a scalable representation of the graph. Communication requirements between tasks are also specified implicitly through a dependency algebra described in Sect. 3.1.4.

Each execution of a taskgraph integrates a single timestep, or a single non-linear iteration, or some other coarse algorithm step. Taskgraphs may be assembled recursively, with a typical Uintah simulation containing one for time integration and one for nonlinear iteration. An AMR simulation may contain several more for implementing time subcycling and refinement/coarsening operators on the AMR grid.

Consider the taskgraph in Fig. 3. Ovals represent tasks, are a simple array program and easily treated by traditional compiler array optimizations. Edges represent named values stored by Uintah. Solid edges have values defined at each material point (Particle Data) and dashed edges have values defined at each grid vertex (Grid Data). Variables denoted with a prime (′) have been updated during the time step. The figure shows the slice of the actual Uintah material point method (MPM, see Sect. 2.3) taskgraph concerned with advancing Newtonian material point motion on a single patch for a single timestep. This graph would be repeated on each patch in the

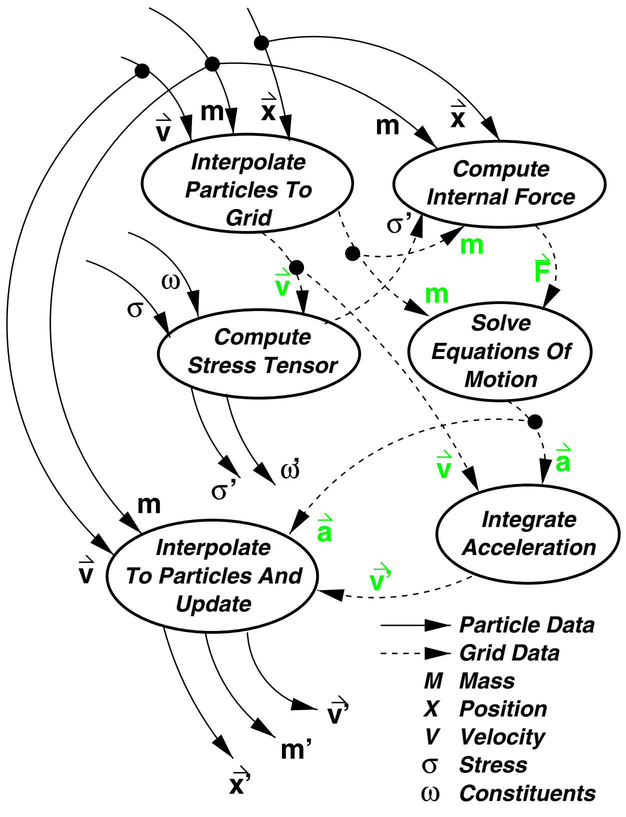

Fig. 3 An example of Uintah taskgraph

domain that contained particles, resulting in a tensor product graph the defined the specific communication patterns for the parallel execution of this portion of the algorithm.

The idea of the dataflow graph as an organizing structure for execution is well known. The SMARTS [18] dataflow engine that underlies the POOMA [19] toolkit shares similar goals and philosophy with Uintah. SISAL compilers [20] used dataflow concepts at a much finer granularity to structure code generation and execution. Dataflow is a simple, natural and efficient way of exposing parallelism and managing computation, and is an intuitive way of reasoning about parallelism. What distinguishes implementations of dataflow ideas is that each caters to a particular higher-level presentation. SMARTS caters to POOMA's C++ implementation and stylistic template-based presentation. The SISAL compiler was, of course, developed to support the SISAL language. Uintah is implemented to support a presentation catering to C++ and Fortran based mixed particle/grid algorithms on a structured adaptive mesh.

3.1.2 Communication

Tasks ''communicate'' with each other through an entity called the *DataWarehouse*. The DataWarehouse is an abstraction of a global single-assignment memory, with automatic data lifetime management and storage reclamation. The DataWarehouse is accessed through a simple name-based dictionary mechanism and provides each task with the illusion that all memory is global. Since the taskgraph must correctly describe all data dependencies, the data stored in the DataWarehouse will always contain the data required by the task (for the specified variable and region of space). Latency in requesting data from the DataWarehouse is not an issue; the correct data is deposited into the DataWarehouse before the task is executed. Requesting data from the DataWarehouse typically does not require a copy operation.

Values stored in the DataWarehouse are typically array-structured, but also include reductions and global data. These type are described further in Sect. 3.1.3.

The DataWarehouse abstraction is sufficiently high-level that it can be efficiently mapped onto both message-passing and shared-memory communication mechanisms. Threads sharing a memory can access their input data directly; single-assignment dataflow semantics eliminate the need for any locking of values. Threads running in disjoint address spaces communicate by message-passing protocol, and Uintah is free to optimize such communication by message aggregation. Tasks need not be aware of the transports used to deliver their inputs and thus Uintah has complete flexibility in control and data placement to optimize communication both between address spaces or within a single shared-memory SMP node.

3.1.3 Particle and grid support

Consider Fig. 4. We define several terms which we use in discussing SAMR grids:

- *Patch:* A contiguous rectangular region of index space and a corresponding region of simulated physical space. The domain on the right of Fig. 4 is the same as the domain on the left, except that is has been decomposed into two patches.
- *Cell:* A single coordinate in the integer index space, also corresponding to the smallest unit in simulated physical space. A variable centered at the cells in the simulation would have a value corresponding to each of the **X**'s in Fig. 4.
- *Node:* An entity at the corners of each of the cells. A variable centered at the nodes in the simulation would have a value corresponding to each of the **O**'s in Fig. 4.
- *Face:* The faces join two cells. Uintah represents values on *X*, *Y*, and *Z* faces separately.
- *Ghost cell*: Cells (or nodes) that are associated with a neighboring patch, but are copied locally to fulfill data dependencies from outside of the patch.

Uintah simulations are performed using a strict ''owner computes'' strategy. This means that each topological entity (a node, cell or face) belongs to exactly one patch. There are several variable types that represent data associated with these entities. An **NCVariable** (node-centered variable) contains a single value at each **Node** in the domain. Similarly, **CCVariables** contain values for each cell-center, and **XFCVariables**, **YFCVariables** and **ZFCVariables** are face-centered values for the faces corresponding to the *X*, *Y*

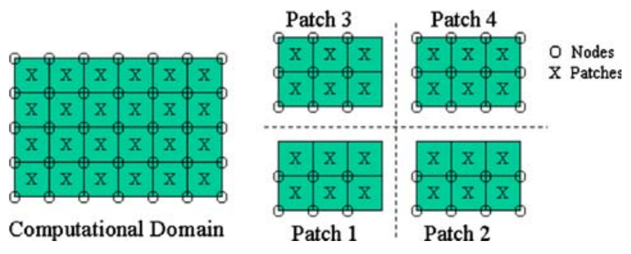

Fig. 4 A simple computational domain and a four-patch decomposition of that domain

and Z axes. Each of these variables types is C++ template class, therefore a node/cell/face-centered value can be any arbitrary type. Typically, values are a double-precision number representing pressure, temperature, or some other scalar throughout the field, but values may also be a more complex entity, such as a vector representing velocity, or a tensor representing a stress.

In addition to the topological based variables described above, there is one additional variable type: **ParticleVariable**. This variable contains values associated with each particle in the domain. A special particle variable contains the position of the particle. Other particle variables are defined by the simulations, and in the case of the MPM algorithm, include quantities like mass, volume, velocity, temperature, stress, and so forth. For the purposes of the discussions below, particles can be considered a fancy type of cell-centered variable, since each particle is associated with a single cell. It is important however, to point out that explicit lists of particles within a cell are not maintained. We have found it more efficient to determine particle/cell associations as they are needed instead of paying the high cost of maintaining lists of particles for each cell.

It is also worth noting that the data structures described here are different from those that many particle-based algorithms employ. Instead of associating all of the data associated with a single particle in a C++ struct, we create arrays for each property. This is called a "structure of arrays", or vertical storage whereas the typical method is an "array of structures", or horizontal storage. We have found that vertical storage can result in improved performance for most particle based simulations. In addition, the vertical storage mechanism allows the properties to vary for different groups of particles. In Uintah simulations, MPM particles may have different constitutive models for different materials) (e.g. the explosive may have very different properties than a steel container).

3.1.4 Describing variable dependencies

Tasks describe data requirements in terms of their computations on node, cell and face-centered quantities. A task that computes a cell-centered quantity from the values on surrounding nodes would establish a requirement for one layer of nodes around the cells at the border of a patch. This is termed nodes around cells in Uintah terminology. As shown in Fig. 5, a layer of *ghost nodes* would be copied from neighboring patches on the top and right edges of the lower-left patch. In a four-processor simulation,

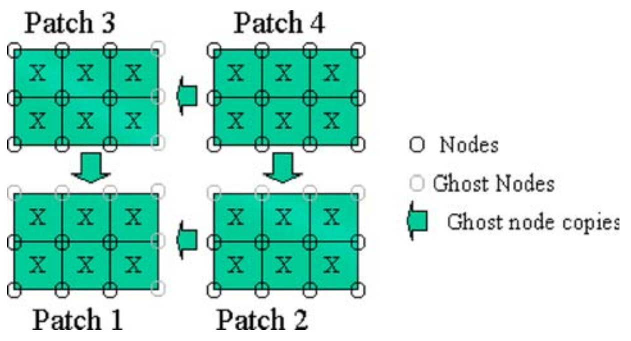

Fig. 5 Communication of ghost nodes in a simple four-patch domain

this copy would involve MPI messages from each of the other three processors. It is important to note the asymmetry in this process; data are often not required from all neighbors to satisfy a computation. Symmetry in the algorithm comes when a subsequent step uses "cells around nodes" to satisfy another data dependency, but each communication step is asymmetric.

The region of the domain over which data are required, is termed the halo region. Similarly, each task specifies the data that it will compute, but in this case, no ghost cells are necessary (or allowed). By defining the halo region with this simple relationship, one can specify the communication patterns in a complex domain without resorting to explicit definition of communication needed. These "computes & requires" lists for each task are collected to create the full taskgraph. Subsequently, the specification of the halo region is combined with the details of the patches in the domain to create the tensor product taskgraph.

A task could specify that it requires data from the entire computational domain. However, for typical scalable algorithms, the tasks ask for only one (or possibly two) layers of data outside of the patch.

Data dependencies can also be specified between refinement levels in an AMR mesh. This can often create very complex communication patterns, but they are still specified by the simulation component using the "X around Y" mechanism.

3.1.5 Task programs

Each component specifies a list of tasks to be performed and the data dependencies between them. These tasks may also include dependencies on quantities from other components. Each of task in Uintah is a C++ method that typically goes through the following sequence of steps:

1. Retrieve data from the data warehouse. This includes specifying the number of ghost cells using the mechanism described in Sect. 3.1.4. These operations never require communication because the appropriate data have been deposited in the data warehouse before the task was executed.
2. Allocate memory for data outputs in the data warehouse.
3. Perform the computation appropriate for particular algorithmic step. In the tasks shown in Fig. 3, the task iterates over either grid cells or particles to perform the MPM algorithm computations. This phase may call a fortran subroutine or the low-level computational kernel.

When the task completes, the infrastructure will send data to other tasks that require the output from this task.

3.1.6 Execution

On a single processor, execution of the taskgraph is simple. The tasks are simply executed in the topologically sorted order. This is valuable for debugging, since multi-patch problems can be tested and debugged on a single processor. In most cases, if the multi-patch problem passes the taskgraph analysis and executes correctly on a single processor, then it will execute correctly in parallel.

In a multi-processor machine the execution processes is more complex. In an MPI-only implementation, there are a number of ways to utilize MPI functionality to overlap communication and I/O. We describe one way that is currently implemented in Uintah.

We process each detailed task in a topologically sorted order. For each task, the scheduler posts non-blocking receives (using MPI_Irecv) for each of the data dependencies. Subsequently, we call MPI_Waitall to wait for the data to be sent from neighboring processors. After all data have arrived, we execute the task. When the task is finished, we call MPI_Isend to initiate data transfer to any dependent tasks. Periodic calls to MPI_Waitsome for these posted sends ensure that resources are cleaned up when the sends actually complete.

The mixed MPI/thread execution is somewhat different. First, non-blocking MPI_Irecvs are posted for *all* of the tasks assigned to the processor. Then each thread will concurrently call MPI_Waitsome and will block for internal data dependencies (i.e. from other tasks) until the data dependencies for any task are complete. That task is executed and data that it produces is sent out. The thread then goes back and tries to complete a next task. This implements a completely asynchronous scheduling algorithm. Preliminary results for this scheduler indicate that a performance improvement of approximately 2X is obtainable. However, thread-safety issues in vendor MPI implementations have slowed this effort.

It can be seen that dramatically different communication styles can be employed by simply changing out the scheduler component. The physics-based application components are completely insulated from these variations. This is a very important aspect that allows the Computer- Science-teams to focus on the best way to utilize the communication software and hardware on a particular machine, without requiring sweeping changes in the application. Each scheduler implementation consists of less than 1,000 lines of code, so it is relatively easy to write one that will take advantage of the properties of the communication hardware available on a machine. Often, the only difficult part is getting the correct information from the vendor in order to determine the best strategy for communicating data.

To accommodate software packages that were not written using the Uintah execution model, we allow tasks to be specially flagged as "using MPI". These tasks will be gang-scheduled on all processors simultaneously, and will be associated with all of the patches assigned to each processor. In this fashion, Uintah applications can use available MPI-based libraries, such as PETSc (2005 http://www.mcs.anl.gov/petsc/) and Hypre (2005 http://www.llnl.gov/CASC/linear_solvers/).

3.2 Infrastructure features

The taskgraph representation in Uintah enables compiler-like analysis of the computation and communication steps in a timestep. This analysis is performed at runtime, since the combination of tasks required to compute the algorithm may vary dramatically based on problem parameters.

Through analysis of the taskgraph, Uintah can automatically create checkpoints, perform load balancing and eliminate redundant communication. This analysis phase, which we call "compiling" the taskgraph, is what distinguishes Uintah from most other component-based multi-physics simulations. The taskgraph is compiled when the grid changes, when the nature of the algorithm changes, or when load imbalance is detected.

Uintah also has the ability to modify the set of components in use during the course of the simulation.

This is used to transition between solution algorithms, such as a fully explicit or semi-implicit formulation, based on conditions in the simulation.

3.2.1 *Parallel I/O and checkpointing*

Data output is scheduled by creating tasks in the taskgraph just like any other component. Constraints specified with the task allow the load balancing component to direct those tasks (and the associated data) to the processors where data I/O should occur. In typical simulations, each processor writes data independently for the portions of the dataset which it owns. This requires no additional parallel communication for output tasks. However, in some cases this may not be ideal. Uintah can also accommodate situations where disks are physically attached to only a portion of the nodes, or a parallel filesystem where I/O is more efficient when performed by only a fraction of the total nodes.

Checkpointing is obtained by using these output tasks to save all of the data in the DataWarehouse at the end of the timestep. Data lifetime analysis ensures that only the data required by subsequent iterations will be saved. If the simulation components have been correctly written to store all of their data in the DataWarehouse, restart is a trivial process. During restart, the components process the XML specification of the problem that was saved with the datasets, and then Uintah creates input tasks that load the DataWarehouse from the checkpoint files. If necessary, data redistribution is performed automatically during the first execution of the taskgraph. In a similar fashion, changing the number of processors is possible. The current implementation does not redistribute data among the patches when the number of processors are changed. Patch redistribution is a useful component even beyond changing the processor count, and will be implemented in the future.

3.2.2 *Load balancing*

A load balancer component is responsible for assigning each detailed task to one processor. For typical Uintah simulations, the most significant source of load imbalance is the existence of particles, and the associated work, in only a portion of the computational domain. The equilibration pressure iteration in ICE also contributes very mildly to load imbalance.

To date, we have implemented only simple static load-balancing mechanisms. However, Uintah was designed to allow very sophisticated load-balance analysis algorithms required by the large-scale motion of particles through the domain. In particular, a cost-model associated with each task will allow an optimization process to determine the optimal assignment of tasks to processing resources. Cost models associated with the communication architecture of the underlying machine are also available. One interesting aspect of the load-balance problem is that the integrated performance analysis in Uintah will allow the cost-models to be corrected at run-time to provide the most accurate cost information possible to the optimization process.

The mixed thread/MPI scheduler described above implements a dynamic load-balancing mechanism (i.e. a work queue) within an SMP node, and uses a static load-balancing mechanism between nodes. We feel that this is a powerful combination, which we will pursue further.

Careful readers will pick up on the fact that the creation of detailed tasks require knowledge of processor assignment. However, sophisticated load-balance components may require this detailed information before they can optimize the task/processor assignments. We use a two-phase approach, where tasks are assigned arbitrarily, then an optimization is performed and the final assignments are made to the tasks. Subsequent load-balance iterations use the previous approximation as a starting point for the optimization process.

3.2.3 *Adaptive mesh refinement*

Many multi-physics simulations require a broad span of space and timescales. C-SAFE's primary target simulation scenario includes a large scale fire (size of meters, time of minutes) combined with an explosion (size of microns, time of microseconds). To capture this wide range of time and length scales efficiently and accurately, the Uintah architecture has been has been designed to support adaptive mesh refinement on structured adaptive grids in the style of Berger and Colella [21]. Some aspects of this capability are still under development, so many of the simulations are performed on a non-adaptive mesh. Particle refinement and coarsening has not been implemented but is being actively studied.

Each individual component can flag individual cells for refinement. A separate component, called the *Regridder*, creates an adaptive mesh that includes refined regions for all of the flagged cells. Each simulation component uses the taskgraph structure to create operators for refinement and coarsening between levels using variable descriptions similar to Sect. 3.1.4.

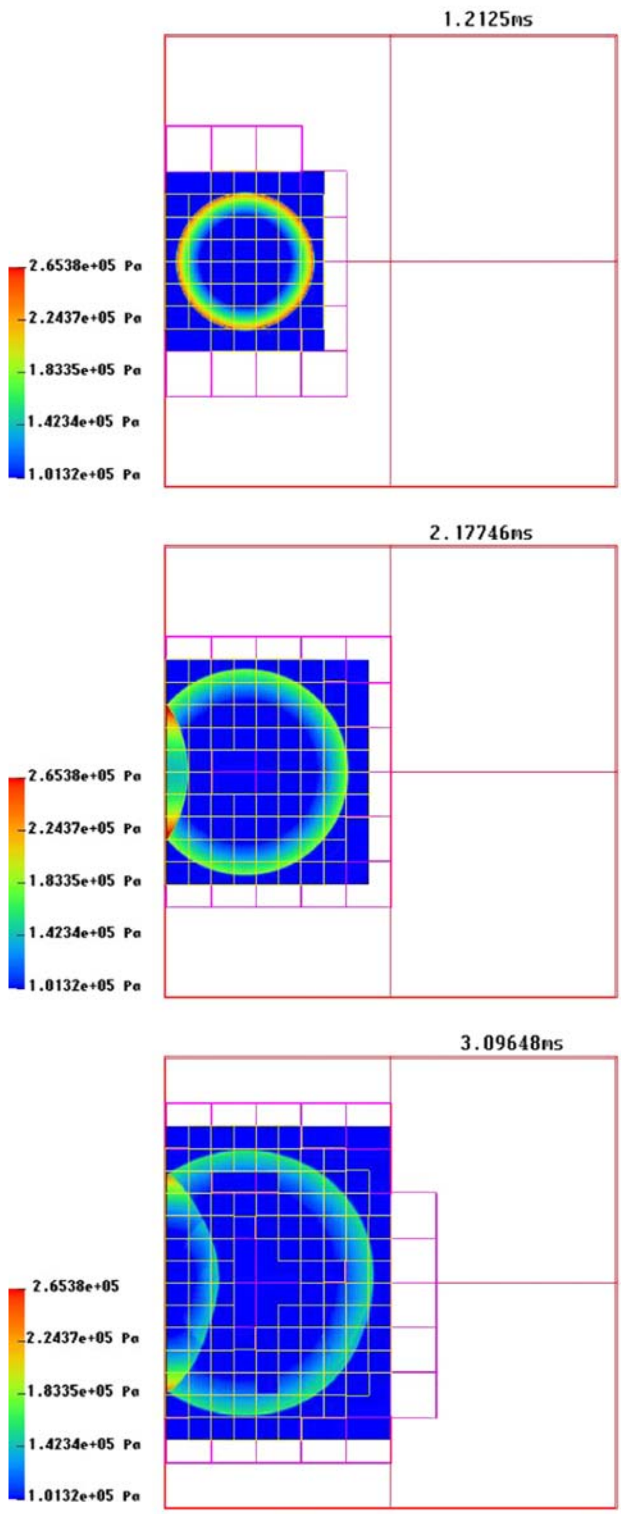

Fig. 6 A pressure blast wave reflecting off of a solid boundary

Figure 6 shows a blast wave reflecting off of a solid boundary with an AMR mesh using the explicit ICE algorithm and refinement in both space and time. For the slower fire phase, we are in the process of implementing a multi-resolution pressure solve that works with the semi-implicit ICE component (See Sect. 2.2).

4 Uintah components

Figure 7 shows the main components involved in a typical FSI simulation using the algorithms described in Sect. 2. The *SimulationController* is the component that manages restart files and controls the time integration tasks. First, it reads the specification of the problem from an XML input file. After setting up the initial grid, it passes the description to the simulation component. The simulation component can be a number of different things, including the multi-material CFD algorithm, the MPM algorithm, or a coupled MPM-CFD algorithm. Each simulation component defines a set of tasks to the *Scheduler*, where a task-graph is created that represents the global computation (See Sect. 3.1.1). In addition, the *DataArchiver* component describes a set of output tasks to the Scheduler. These tasks will save a user specified set of variables to disk, including automatically derived checkpoint information when needed. Once all tasks are known to the scheduler, the *LoadBalancer* component uses the configuration of the machine (including processor counts, communication topologies, etc.) to assign tasks to processing resources. The Scheduler uses MPI to communicate the data to the right processor at the right time and then executes callbacks into the simulation or DataArchiver components to perform the actual work. The DataWarehouse described in Sect. 3.1.2 is managed by the scheduler to communicate data to the tasks. This process continues until the taskgraph is fully executed. The execution process is repeated to integrate further timesteps.

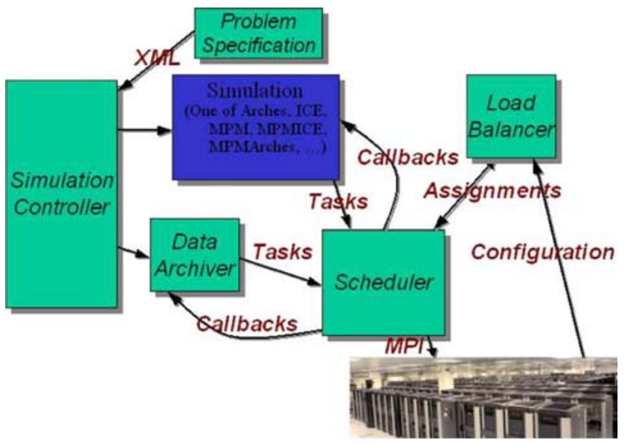

Fig. 7 Uintah simulation components

Each of these components runs concurrently on each processor. The components communicate with their counterparts on other processors using MPI. However, the Scheduler is typically the only component that needs to communicate with other processors.

4.1 Physical and subgrid scale models

Each of the Uintah simulation components requires some number of models to make it physically relevant. Examples of these include equations of state for the CFD components as well as constitutive relations and contact models for the MPM component. Generally, these models assume that the resolution of the computational grid is sufficient to capture relevant physical processes (exceptions include some constitutive models which attempt to homogenize micro-scale behavior). In contrast, other physical processes such as gas phase chemical reactions, solid $\rightarrow$ gas reactions, turbulence and mixing are phenomena, which typically are not adequately resolvable . All of these model types are available within Uintah.

The component architecture of Uintah is employed to allow these models to be self-contained units. While each model has a well-defined function (such as computing a stress), they may each require very different data to accomplish this computation. The models describe their data requirements, which the framework will compile into the taskgraph and subsequently determine the data communication required for that combination of models. Consequently, submodels can have a profound effect on the communication patterns in the simulation but they are still completely encapsulated.

We briefly describe a few of these models here.

Equations of state One of the characteristics of the CFD component used in the FSI simulations is that it is multi-material formulation. Thus, the governing equations are solved separately for each material, with rules for interaction between the materials, and simulations involving arbitrary numbers of materials are possible. One of the characteristics of a material is its equation of state (EOS), the relationship between pressure, volume and temperature alluded to near the end of Sect. 2.2. Uintah makes available a number of equations of state including those for an ideal gas, solid explosives, explosive products of reaction and soil.

Constitutive models Constitutive models are, generally speaking, relationships between stress and a material's state of deformation and rate of deformation. A potentially large number of history variables give information about the loading path a material has undergone to arrive at its current state. Uintah has a large number of constitutive models available for use with the MPM component. Currently available are numerous models each for elastic materials, metals, explosives and biological tissue.

Contact models One of the hallmarks of MPM is the relative ease with which contact between solid bodies can be modeled, particularly when compared to traditional solid mechanics methods such as the finite element method (FEM). A no-slip, no-interpenetration contact model comes for "free" when using MPM, and with a relatively small amount of additional computational effort, frictional contact can also be modeled [22]. The Uintah-MPM component is also a multi-material formulation, with different materials (or identical materials, between which advanced contact is to be modeled) represented by distinct sets of particles. Each material has a separate field representation (velocity, temperature, mass, etc.) on the grid as well, and these interact via contact models for momentum and heat exchange. This approach requires no surface descriptions and the expense increases only linearly with both the number of nodes and the number of materials.

Subgrid reaction models The Uintah CFD component contains a plugin capability to carry arbitrary transported quantities and provide sources of mass, momentum and energy. These models can be used to describe a wide range of physical processes, including turbulent mixing, radiative heat transfer, gas phase chemical reactions and exothermic solid $\rightarrow$ gas reactions. For the types of scenarios that Uintah was designed to address, each of these is vital. Uintah contains several models for gas phase hydrocarbon combustion as well as models for both detonation and deflagration of high energy (HE) materials.

5 Results and discussion

The Uintah infrastructure is used regularly throughout the C-SAFE center for most development and production runs. The results presented here are representative of the problems being examined by C-SAFE. Each of these results uses the full Uintah infrastructure, including the facilities for scheduling, load-balancing and parallel communication. However, none of these results employ AMR, since the AMR version of the FSI algorithm in still under development. In addition, we have performed numerous simulations of other complex phenomena [11, 23–26] using this infrastructure.

5.1 Material point method—compression of foam microstructure

The material point method was chosen as the structural mechanics technique because of its compatibility with the multi-material CFD formulation. The fact that the MPM can use the same structured grid, instead of requiring the use of its own unstructured mesh, greatly simplifies issues of geometric compatibility between the solid and fluid materials. However, MPM is also a powerful tool for solid mechanics simulations in its own right. It avoids issues of mesh creation, distortion and entanglement, simplifies contact and lends itself to parallelism.

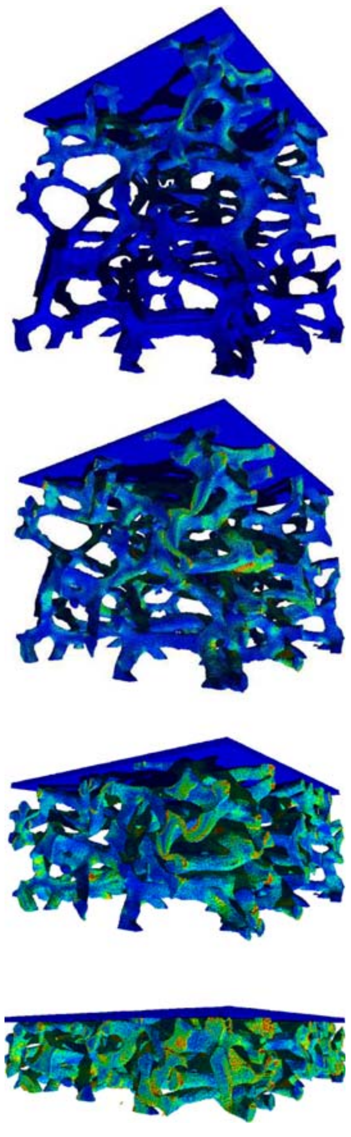
Fig. 8 Time series depicting crushing of a small sample of foam using the material point method

Figure 8 depicts a simulation that demonstrates many of MPM's strengths. Here a small foam microstructure (0.5 mm^3) is compressed by a rigid plate [12]. The initial geometry is taken from a micro-CT scan of an actual sample of foam. The use of particles, instead of an unstructured mesh, makes generation of a computational representation of the geometry trivial. As the compression proceeds, the individual struts that comprise the structure are subjected to severe distortion and contact each other, either of which render the finite element method useless in this application.

5.2 Fluid–structure interaction–explosion of a high energy device

The C-SAFE target scenario is the simulation of metal container filled with a plastic-bonded explosive (PBX) subject to heating from a hydrocarbon pool fire. Figure 9 shows results from a simulation that is approaching that goal. Here, a jet of hot air is impinging on a steel container filled with PBX-9501, heating the container and the PBX. Once the temperature of the PBX reaches a threshold, it ignites in an exothermic solid $\rightarrow$ gas reaction. The products of this reaction pressurize the container, causing it to bulge and eventually rupture. At this point, the high-pressure product gases are free to escape the container and interact with the surrounding environment. Shock waves resulting from this process are evident in the middle and left panels. The purple mass in the center of the domain is the remaining unburned propellant.

There are some unique and significant features in this simulation that warrant further description. First, there is no notion of surfaces separating different regions of the domain. Second, the energetic device initially contains no gas and no void space, the products of reaction occupy the space left by the consumed reactant and the space created due to expansion. Lastly, the interaction of gases from two domains initially separated by a steel container is a unique feature that we have not seen demonstrated in any other FSI capability.

Another simulation of a similar scenario can be seen in Fig. 1, which shows an explosive device subjected to a jet fuel fire as the device to rupture. The individual steel and PBX particles are shown with a color-map indicating temperature, along with a volume rendering of the temperature products of the solid $\rightarrow$ gas reaction. In the rightmost pane, the products of reactions can be seen impacting the edge of the computational domain. This simulation required 600 CPUs (2.4 Ghz Intel Xeon, Myrinet interconnect) for approximately 120 h, and was completed on a different

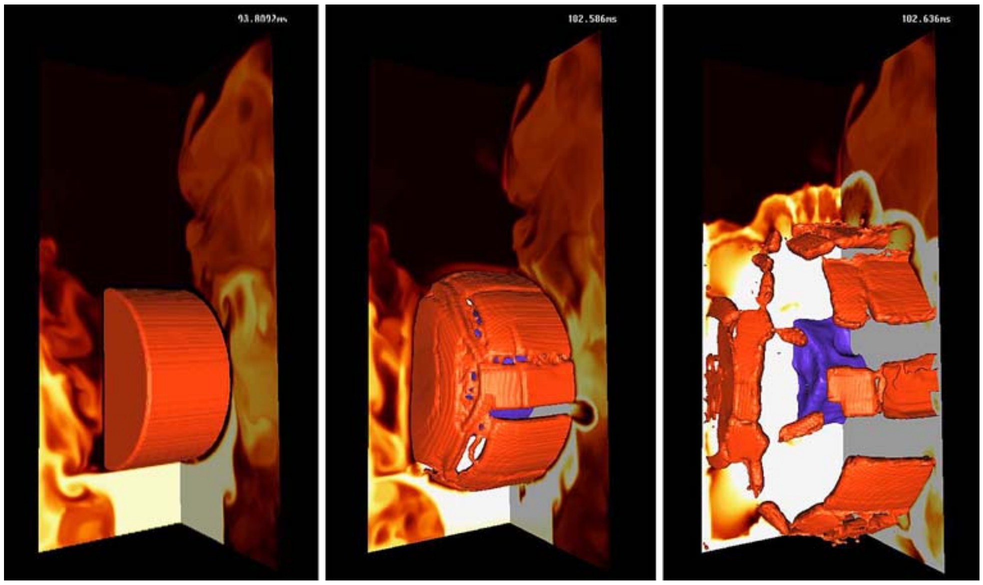

Fig. 9 Hot jet impinging on an energetic device leading to its explosion

machine with 600 CPUs (1.5 Ghz Intel Itanium) for about 30 h. The automatic checkpointing facility was used to restart the simulation on the second machine after our time allocation was exhausted on the first.

This simulation also employed a mechanism to transition between different solution algorithms. The first phase used a semi-implicit algorithm to simulate the dynamics of the evolution of a jet fuel fire for several seconds to obtain a heat flux. The second phase performed a fully implicit heat conduction algorithm inside the steel container for approximately 1 hr until the explosive reached the ignition temperature. The third phase switched to a fully explicit calculation to simulate the dynamics of the rapid pressurization of the container and subsequent failure of the device. This explosion phase represents less than a millisecond of time, with the timestep severely restricted. Each of these phases was triggered automatically by conditions in the simulation. Approximately half of the computational time was devoted to the final phase of the simulation.

5.3 Scalability

The system described here has been used to carry out a variety of simulations. Figure 10 shows a log-log plot of the time to solution versus processor count of a fixed-size benchmark MPM application running on a variety of different machines. It should be noted that the problem shown is relatively small, with timesteps completing in much less than one second for the large processor configurations. Simulations are routinely executed on hundreds of processors including the results shown in Fig. 9, which was executed on 600 processors. This benchmark simulation employs a fixed grid of uniform resolution. Performance analysis of the AMR algorithms mentioned in Sect. 3.2.3 are not yet available. However, the infrastructure is able to scale to thousands of processors while employing the machinery to enable AMR, including the complex taskgraph structure created by a resource of that scale.

6 Conclusions and acknowledgments

We presented the Uintah computational framework and its strategies for integrating multiple physics components in a scalable simulation, and presented

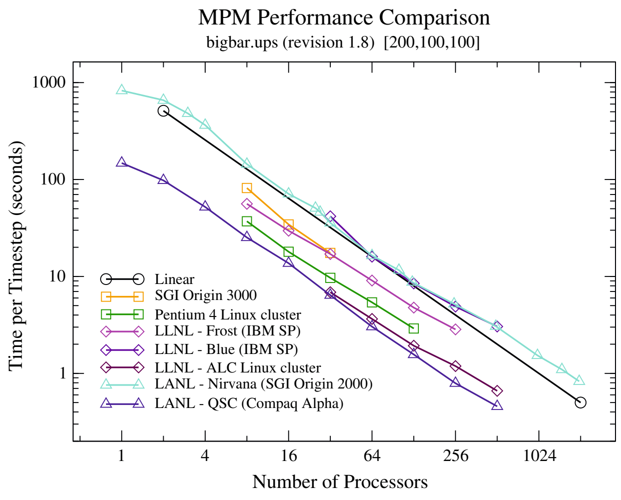

Fig. 10 Parallel performance of a typical C-SAFE problem. This is a 16 million particle material point method (MPM) computation, executed on a variety of machines at DOE national laboratories

results from representative simulations. This component-based system enables multiple physics simulations to be combined into an integrated scalable simulation of fires and explosions.

The authors gratefully acknowledge Dr. Bryan A. Kashiwa of Los Alamos National Laboratory for sharing with us the theoretical and algorithmic foundations upon which this work is based. Without the personal interactions that he made time to provide, none of this would be possible. This work was supported by the U.S. Department of Energy through the Center for the Simulation of Accidental Fires and Explosions, under grant W-7405-ENG-48, and NSF-ITR Grant # CTS0218574 . We thank each of the members of the C-SAFE team at the University of Utah for their contributions to the Uintah software.

References

1. Henderson T, McMurtry P, Smith P, Voth G, Wight C, Pershing D (1994) Simulating accidental fires and explosions. Comp Sci Eng 2:64–76
2. Krishnamoorthy G, Borodai S, Rawat R, Spinti J, Smith P (2005) Numerical modeling of radiative heat transfer in pool fire simulations. ASME International Mechanical Engineering Congress (IMECE), Orlando, Florida
3. Kashiwa B, Rauenzahn R (1994) A multimaterial formalism. Technical Report LA-UR-94-771, Los Alamos National Laboratory, Los Alamos
4. Kashiwa B, Lewis M, Wilson T (1996) Fluid–structure interaction modeling. Technical Report LA-13111-PR, Los Alamos National Laboratory, Los Alamos
5. Kashiwa B (2001) A multified model and method for fluid–structure interaction dynamics. Technical Report LA-UR-01-1136, Los Alamos National Laboratory, Los Alamos
6. Kashiwa B, Gaffney E (2003) Design basis for cfdlib. Technical Report LA-UR-03-1295, Los Alamos National Laboratory, Los Alamos
7. Harlow F, Amsden A (1968) Numerical calculation of almost incompressible flow. J Comp Phys 3:80–93
8. Kashiwa B, Rauenzahn R (1994) A cell-centered ice method for multiphase flow simulations. Technical Report LA-UR-93-3922, Los Alamos National Laboratory, Los Alamos
9. Guilkey J, Harman T, Xia A, Kashiwa B, McMurtry P (2003) An eulerian–lagrangian approach for large deformation fluid–structure interaction problems, part 1: Algorithm development. In: Fluid structure interaction II. WIT Press, Cadiz
10. Harman T, Guilkey J, Kashiwa B, Schmidt J, McMurtry P (2003) An eulerian–lagrangian approach for large deformation fluid–structure interaction problems, part 2: Multi-physics simulations within a modern computational framework. In: Fluid structure interaction II. Cadiz, Spain, WIT Press
11. Guilkey J, Hoying J, Weiss J (2006) Modeling of multicellular constructs with the material point method. J Biomech 39:2074–2086
12. Brydon A, Bardenhagen S, Miller E, Seidler G (2005) Simulation of the densification of real open-celled foam microstructures. J Mech Phys Solids 53:2638–2660
13. Guo Y, Nairn J (2004) Calculation of j-integral and stress intensity factors using the material point method. Comput Model Eng Sci 6:295–308
14. Sulsky D, Chen Z, Schreyer H (1994) A particle method for history-dependent materials. Comp Methods Appl Mech Eng 118:179–196
15. Sulsky D, Zhou S, Schreyer H (1995) Application of a particle-in-cell method to solid mechanics. Comput Phys Commun 87:236–252
16. Brackbill J, Ruppel H (1986) Flip: a low-dissipation, particle-in-cell method for fluid flows in two dimensions. J Comp Phys 65:314–343
17. Guilkey J, Weiss J (2003) Implicit time integration for the material point method: quantitative and algorithmic comparisons with the finite element method. Int J Num Meth Eng 57:1323–1338
18. Vajracharya S, Karmesin S, Beckman P, Crotinger J, Malony A, Shende S, Oldehoeft R, Smith S (1999) Smarts: exploiting temporal locality and parallelism through vertical execution. In: Proceedings of the 13th international conference on supercomputing
19. Atlas S, Banerjee S, Cummings J, Hinker P, Srikant M, Reynders J, Tholburn M (1995) POOMA: a high-performance distributed simulation environment for scientific applications. In: Supercomputing ’95 proceedings
20. Feo J, Cann D, Oldehoeft R (1990) A report on the sisal language project. J Parallel Distrib Comput 10(4):349–366
21. Berger M, Colella P (1989) Local adaptive mesh refinement for shock hydrodynamics. J Comput Phys 82:64–84
22. Bardenhagen S, Guilkey J, Roessig K, Brackbill J, Witzel W, Foster J (2001) An improved contact algorithm for the material point method and application to stress propagation in granular material. CMES 2:509–522
23. Banerjee B (2005) The mechanical threshold stress model for various tempers of ansi 4340 steel. Int J Solids Struct (in press)
24. Banerjee B (2005) Validation of a multi-physics code: plasticity models and taylor impact. In: Proceedings of joint ASME/ASCE/SES conference on mechanics and materials. Baton Rouge, LA
25. Banerjee B (2004) Material point method simulations of fragmenting cylinders. In: Proceedings of the 17th ASCE engineering mechanics conference. Newark, DE
26. Banerjee B, Guilkey J, Harman T, Schmidt J, McMurtry P (2005) Simulation of impact and fragmentation with the material point method. In: Proceedings of the 11th international conference on fracture, Turin, Italy, p 689