Parallelization and integration of fire simulations in the Uintah PSE

Rajesh Rawat[1], Steven G. Parker[2], Philip J. Smith[3], Christopher R. Johnson[4]
Center for Simulation of Accidental Fires and Explosions
University of Utah

Abstract

A physics-based stand-alone serial code for fire simulations is integrated in a unified computational framework to couple with other disciplines and to achieve massively parallel computation. Uintah, the computational framework used, is a component-based visual problem-solving environment developed at the University of Utah. It provides the framework for large-scale parallelization for different applications. The integration of the legacy fire code in Uintah is built on three principles: 1) Develop different reusable physics-based components that can be used interchangeably and interact with other components, 2) reuse the legacy stand-alone fire code (written in Fortran) as much as possible, and 3) use components developed by third parties, specifically non-linear and linear solvers designed for solving complex-flow problems. A helium buoyant plume is simulated using the Nirvana machine at Los Alamos National Laboratory. Linear scalability is achieved up to 128 processors. Issues related to scaling beyond 128 processors are also discussed.

Introduction

The Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [1] at the University of Utah is developing a general framework for large-scale massively parallel simulations of accidental fires and explosions involving hydrocarbons, structures, containers and high-energy materials. Realistic simulation of such complicated systems requires the representation of relevant physical processes such as turbulent reacting flows, convective and radiative heat transfer, structural mechanics and fundamental gas- and condensed-phase chemistry. Traditionally, such processes have been dealt with separately as stand-alone simulation tools. Coupling these processes presents a unique challenge from both research as well as software engineering perspective. Moreover, such complex systems are characterized by a wide range of continuum length scales (1mm-1km) and corresponding time scales (1 fs-1 s). This entire range cannot be directly computed even on the peta-flop computers, next generation ASCI machines. Nevertheless, important features of fire physics and its interaction with the structure engulfed in the fire can be captured by resolving large length and time scales responsible for controlling the dynamic features of fire. Resolving these length and time scales, however, requires using massively parallel computations. Therefore, to accomplish our goal of simulating complex large-scale fires in the presence of structures, we are developing software components reusing physics-based legacy Fortran codes that have been validated against experiments and that are computationally efficient and scalable. These components are created to allow one to freely choose options for different turbulent flow, turbulent mixing, radiation, and solvers and to easily couple them with Uintah Computational Framework [2] (UCF), a component-based visual Problem Solving Environment (PSE), to achieve parallel computations. UCF provides the framework for large-scale parallelization for different applications.

[1] Department of Chemical and Fuels Engineering, rawat@crsim.utah.edu
[2] Scientific Computing and Imaging Institute, School of Computing, sparker@cs.utah.edu
[3] Department of Chemical and Fuels Engineering, smith@crsim.utah.edu
[4] Scientific Computing and Imaging Institute, School of Computing, crj@cs.utah.edu

Uintah Computational Framework

The fire code simulates the propagation of an ignition event over a pool of liquid hydrocarbon fuel. To simulate this event, the code uses algorithms that couple turbulent computational fluid dynamics (CFD), turbulent mixing and reaction chemistry for fire (including soot), and radiative transport. Due to the wide range of physics chemical length and time scales involved in the simulations it is necessary to have the flexibility to explore different parallel algorithms. For instance, our CFD simulation uses large-eddy simulations in which large, energy carrying length and time scales are resolved and the smaller scales are modeled. A domain decomposition strategy is used to parallelize highly non-linear governing equations for such a system. While, this strategy works well for CFD systems, chemical reactions and turbulent sub-grid scale mixing models are functions of local state-space (chemical species) variables and are independent of the grid. Therefore, domain decomposition strategy in physical space may not yield the best performance for implementing chemical reactions. Instead, we partition our domain in state-space to yield good scalability. The same holds true for radiative heat transfer modeling which uses integro-differential radiative transport equations that can only be parallelized efficiently by dividing the tasks on the basis of different rays that need to be tracked on the whole grid rather than patches of grid. Moreover, the computation of radiative fluxes can be made efficient by taking advantage of the knowledge that the spatial grids for predicting radiative fluxes can be much more coarser that those for CFD. Thus, the flexibility in maintaining different grids will be very crucial for such simulation environment. Uintah is designed to incorporate such flexibility required to efficiently parallelize a multi-physics environment.

Uintah is a component-based visual Problem Solving Environment derived from the SCIRun PSE [3,4], combining the visual programming environment that SCIRun uses for shared-memory parallel computers with the distributed memory component model being developed by the Common Component Architecture (CCA) forum [5]. Uintah allows different types of components to be connected through different types of ports, which are general RPC-based method interfaces. Uintah has been designed and implemented to satisfy three goals: 1) To provide a general framework for massive scale simulations of fluid and particle physics, 2) to facilitate both MPI- and thread-based parallelism, and 3) to allow scientists from outside the computer field to have an intuitive method for easily inserting their algorithms into a parallel framework without being bogged down by all the details of parallel programming. UCF hides the complexity of parallel data management from the application developer by using two key abstractions: a component called the Data Warehouse, and a task graph representation of the numerical algorithm. The Data Warehouse presents developers with the abstraction of a global, single-assignment memory with automatic data lifetime management and storage reclamation. The task graph is designed to handle the wide range of loads due to different physics and architectural communication limitations.

Each algorithm is defined in terms of tasks with inputs and outputs provided. These tasks are described per patch or computational domain by variable names and spatial relations (including ghost cells). This information is used by a scheduler component defined in the UCF to create a task graph with edges of the graph representing (possible) communication required between different

processors. Based on the task graph, the scheduler decides which task will be executed by which processor and is guided by cost models for computation and communication. The disadvantages of the task graph approach are that an optimal solution is NP-hard (however, good solutions are not too hard), creation of the schedule can be expensive (however, it only needs to be recomputed periodically for load balance and the cost can be amortized over several time-steps). On the other hand, it accommodates flexible integration needs and workload profiles, offers a mix of static and dynamic load balancing, helps manage complexity of a mixed threads/MPI programming model, and allows the individual simulation components to evolve independently.

Integration and Parallelization

The fire code, Arches, that we developed and validated over the years was a stand-alone physics-based, serial code. To achieve the goal of developing a generalized framework for large-scale, massively parallel simulations of accidental fires and explosions, our focus in this work was on integrating the fire code in the UCF.

Integration Strategy
The integration of the Arches in UCF is built on three principles: 1) Develop different reusable physics-based components that can be used interchangeably and interact with other components, 2) reuse the legacy stand-alone fire code (written in Fortran) as much as possible, and 3) use components developed by third parties, specifically non-linear and linear solvers designed for solving complex-flow problems.

Our first step was to design a generalized, component-based architecture for fire simulations that can be incorporated in any computational framework that provides support for parallelization. First, we worked on the software design specifications to integrate with UCF. Figure 1 illustrates different design components and their relationships for fire simulations. These components are designed around real world concepts (such as subgrid scale micromixing as represented by the MixingModel component), and they encapsulate functionality found in multiphysics problems. Many of the design decisions were necessitated by the parallel-processing paradigm provided by UCF.

After finalizing the design, we added details to the design model to describe and optimize the implementation of the different physics-based, reusable components. These components fulfill two important features. First, they provide an interface to the UCF. An example of how these components interface with UCF for a CFD simulation is shown in Figure 2.

Boxes in the figure represent Arches components and the oval shapes represent the Data Warehouse component of the UCF. A request to advance simulations in time is passed from UCF to the Arches Integrator component. Arches Integrator reads the data provided by Data Warehouse and uses the Arches Solver component to solve the set of nonlinear equations defined at the time of the problem setup. It also creates a temporary Data Warehouse (referred to as "new dw" in the figure) to store data computed during the time step simulation. It can be seen that all the Arches components interact with the temporary Data Warehouse during the course of the simulation. At

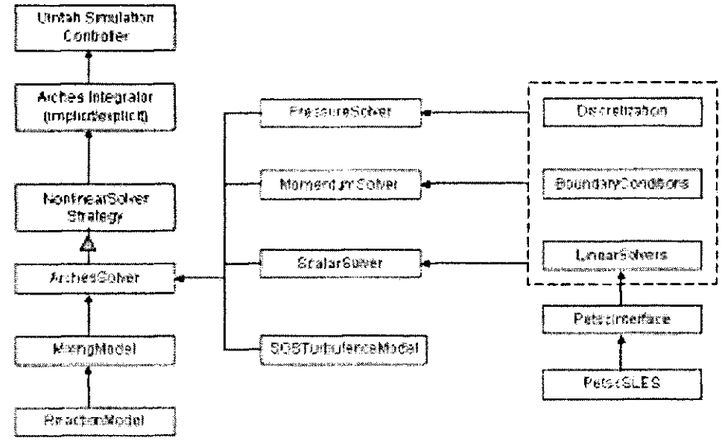

Figure 1: Component-based design for Arches.

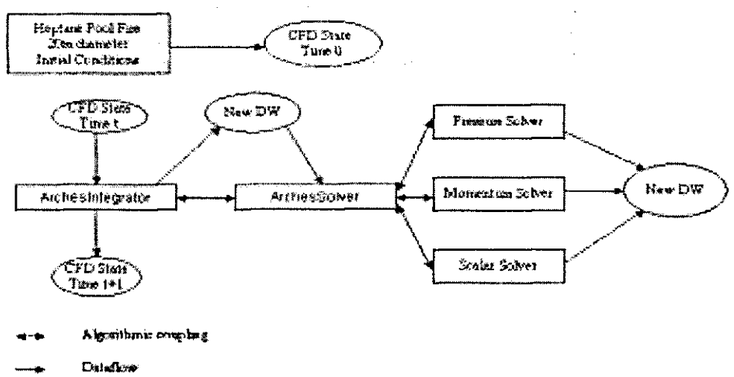

Figure 2: An example fire simulation

the end of one time step, Arches Integrator writes the data computed for the new time step back to the original Data Warehouse.

Second, the components define and encapsulate a family of algorithms. This design provides the flexibility to use and test different algorithms interchangeably in plug and play fashion. Figure 3 illustrates the SGSTurbulenceModel component for LES where different turbulence models are implemented and can be used interchangeably depending on the problem we are solving.

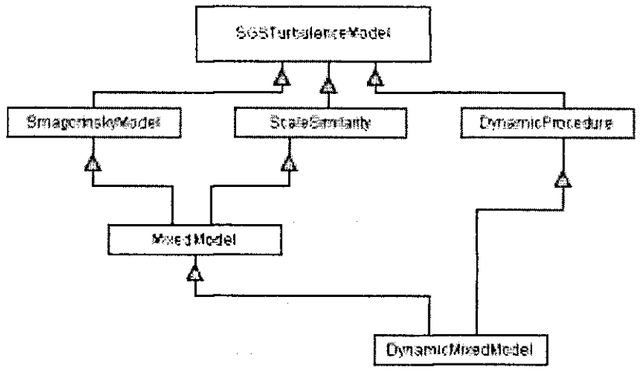

Figure 3: Subgrid scale turbulence model component of Arches

Code reuse was necessary to minimize the code development process since most of our stand-alone code was already validated. To accomplish this task, we spent significant time modularizing the Fortran subroutines for use in our components. This included reorganizing subroutines to fit within the design described above, and removing our dependency on common blocks.

Realistic fire simulations must account for relevant physical processes such as turbulent reacting flow, convective and radiative heat transfer, multi-phase interactions, and fundamental gas phase chemistry. Representations of these physical processes lead to very large, highly nonlinear, partial differential equations (PDEs). Robust nonlinear and linear solvers for these large-scale nonlinear sets of PDE on massively parallel ASCI platforms are required. To this end, we selected Portable Extensible Toolkit for Scientific Computation (PETSc) to provide general-purpose solvers. PETSc provides a suite of nonlinear and linear scalable solvers for scientific applications modeled using PDEs. [6] PETSc solvers are written to be independent of the underlying data structures that define

the problem. We have exploited this feature in the creation of the interface between PETSc and Uintah. The interface provides the capability to manipulate Uintah data structures as the vectors and the sparse matrices that are used by PETSc. The interface also allows us to write custom algorithms that can be used as preconditioners to the PETSc solvers.

Through the Uintah-PETSc interface, we now have access to the full range of solver capabilities provided by PETSc. To achieve scalability for big problem and large number of processors, we are working on developing customized preconditioners like Additive/Multiplicative Schwartz with PETSc solvers.

The fluid flow component of the Arches is now integrated and running into Uintah. An explicit solver is in place that uses the Uintah-PETSc interface for solving the pressure equation in an implicit fashion.

Parallelization Strategy

For parallel CFD computation, UCF creates a task graph using the components provided by the application developer and assigns them to different processors determined by the Scheduler based on the cost model for computation and communication, as described in the previous section.

Figure 4 shows a typical task graph created during the process of solving the pressure equation, one of the steps in a fire simulation.

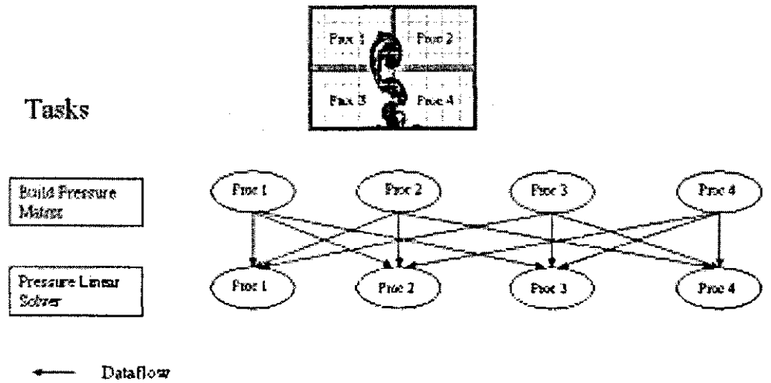

Figure 4: An example task graph for pressure solver of Arches

This process has been divided into two tasks. In the first task, a linear matrix is generated through discretization and the application of boundary conditions. Before the start of this task, each processor has the data it requires for computing the matrix. In the second task the matrix is solved using PETSc (see previous section). As shown in Figure 4, the computational domain is divided into four patches, and each patch is associated with a processor. Arrows in the figure indicate data dependencies between the processors. The Data Warehouse uses this task graph to determine dependencies and data transfer.

Results

Figure 5 illustrates the parallel scalability obtained with the integrated UCF/Arches code. The simulation consists of a buoyant helium plume of diameter 10cm. The problem domain is 1m x 1m x 1m, discretized into 3.4 million cells. The resulting code demonstrates linear scalability to 125 processors on the SGI Origin 2000 at Los Alamos National Laboratory. We have identified two bottlenecks to achieving scalability beyond 125 processors: First, the UCF has known limitations that are being address in a second version. Second, the preconditioner used in this computation does not exhibit good scaling characteristics beyond 125 processors. Multigrid preconditioners are being examined for future runs.

Other optimizations planned include the combination of MPI-based message-passing communication with a finer-grained thread-based communication. UCF facilitates this combination without complicated changes in the application code. Preliminary results indicate that performance improvements of 50% or more are achievable with this type of mixed communication model.

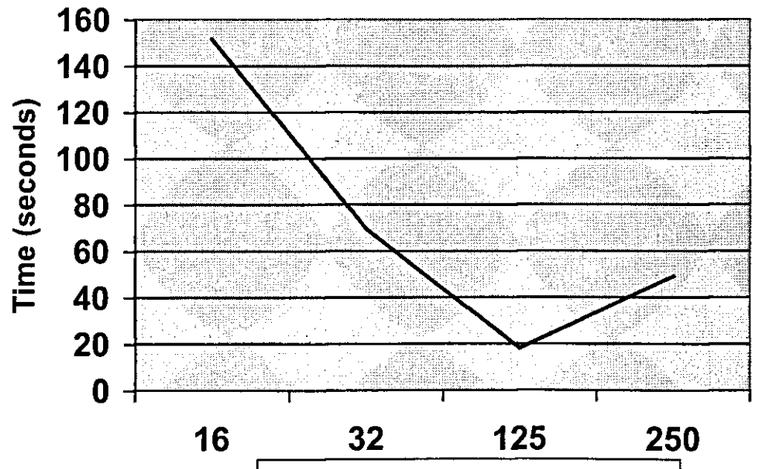

Figure 5: Scalability of Arches running in the Uintah Computational Framework

Conclusions

The Uintah PSE and Uintah Computational Framework provide a mechanism for achieving parallelism with an existing Fortran CFD code. In integrating the Arches code with UCF, we attempted to strike a balance between code re-use and performance obtained. While some changes were necessary to the underlying Fortran code, the bulk of the algorithmic sections of the code remained intact.

Acknowledgements

This work was supported by the Department of Energy Accelerated Strategic Computing Inititative (ASCI), Academic Strategic Alliance Partners (ASAP) program. This work is a collaboration of the Combustion Research Simulation (CRSIM) group in the Chemical and Fuels Engineering Department and the Scientific Computing and Imaging Institute in the School of Computing at the University of Utah. We would like to thank our C-SAFE colleagues for additional collaboration and assistance related to this effort.

References

[1] T. Henderson, P. McMurtry, P. Smith, G. Voth, C. Wight, and D. Pershing. "Simulating Accidental Fires and Explosions", IEEE Computational Science and Engineering, volume 7, number 2, pp. 64-76, 2000.

[2] J. Davison de St. Germain, John McCorquodale, Steven G. Parker, Christopher R. Johnson. Uintah: A Massively Parallel Problem Solving Environment. Ninth IEEE International Symposium on High Performance and Distributed Computing (HPDC) '00, August 2000.

[3] C. Johnson, S. Parker, C. Hansen, G. Kindlmann, and Y. Livnat. Interactive Simulation and Visualization. IEEE Computer, December 1999.

[4] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. Proceedings of High Performance Distributed Computing (HPDC) '99. August 1999.

[5] S. Parker, D. Beazley, and C. Johnson. Computational Steering software systems and strategies. IEEE Computational Science and Engineering, volume 4, number 4, pp. 50-59, 1997.

[6] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. The Portable Extensible Toolkit for Scientific Computing (PETSc) version 28. http://www.mcs.anl.gov/petsc/petsc.html, 2000.