DAG-Based Software Frameworks for PDEs

Martin Berzins, Qingyu Meng, John Schmidt, and James C. Sutherland [1]

Scientific Computing and Imaging Institute, [1]Institute for Clean and Secure Energy,
University of Utah, Salt Lake City, UT 84112, USA

Abstract. The task-based approach to software and parallelism is well-known and has been proposed as a potential candidate, named the silver model, for exascale software. This approach is not yet widely used in the large-scale multi-core parallel computing of complex systems of partial differential equations. After surveying task-based approaches we investigate how well the Uintah software and an extension named Wasatch fit in the task-based paradigm and how well they perform on large scale parallel computers. The conclusion is that these approaches show great promise for petascale but that considerable algorithmic challenges remain.

Keywords: Directed Acyclic Graph Task-Based Parallelism Scalability

1 Introduction

The task-based approach to parallel computing is both well-known and widely-discussed as a potentially useful approach, but is not so often employed at large scales on parallel architectures as of yet. The central idea is to use a Directed Acyclic Graph (DAG) based approach to express the structure of the underlying software, see [8, 11, 24]. While the leading edge of present large-scale computing is focused on petascale computations, the anticipated move to exascale computing, [30], over the next decade has led to a discussion of task-based approaches as potential candidates for exascale software over the next decade. For example, the Silver model, [1], aims to:

1. provide an abstraction of parallel computation that exposes and exploits a high degree of algorithm concurrency, particularly from dynamic directed graph structure-based applications,
2. enable intrinsic latency hiding through automatic overlap of computation and communication through message-driven work-queue multi-threaded execution,
3. minimize impact of synchronization and other overheads for efficient scalable execution through lightweight object-oriented semantics,
4. support dynamic global address space scheduling for adaptive resource management, and
5. unify heterogeneous structure computing for diversity of processing modalities and exploitation of accelerator micro-architectures.

With this in mind, the aim of this paper is to explore the usefulness of DAG based approaches, such as the Silver Model, for computational frameworks which solve large systems of partial differential equations (PDEs) on existing large scale computers. This

exploration will make use of the DAG-based parallel Uintah software framework for partial differential equations (PDEs) [21, 23] and its recent developments, [5, 14, 16, 19], to assess how well the Silver Model type approach works on present-day large-scale architectures for complex multi-physics multiscale applications. In order to assess how well the same approach also works when applied at multi-core level for complex physical applications, an approach, named Wasatch, proposed by one of the authors (Sutherland) and related to [20] will be considered. As a result of these investigations, a preliminary and tentative evaluation of the silver model type approach for PDE software infrastructures will be given.

2 A Brief Survey of Direct Acyclic Graph Approaches

The idea of the dataflow graph as an organizing structure for execution is well known and has been widely used in many different contexts. Only a non-exhaustive survey of a few salient approaches is given here. An appropriate starting point is Sarkar, [24], and the references within, while an example of a more recent discussion is [26]. One important distinction is the level of granularity at which the task-graph approach is applied. For example, the SISAL language compilers [25] used DAG concepts at a fine level of granularity to structure code generation and execution. More recently the PLASMA project uses these ideas at a sub-core level in a task-based linear algebra approach [13], to achieve speed-up over a conventional bulk-synchronous approach. The same ideas have also been extended to several thousand cores, [8]. An interesting language development explicitly designed for task-based paradigms is that of CnC [31]. A CnC approach has been shown to yield very good results in an automated way for linear algebra problems on multi-cores, [6]. While almost all of these approaches use data parallelism to achieve multi-core performance, that of [20] uses a functional decomposition of the complex system of PDEs being solved and will be discussed further below.

The use of a coarser granularity than the above examples makes it possible to apply the DAG approaches at a higher level. The SMARTS [28] dataflow engine used in the POOMA [2] toolkit shares a similar approach with Uintah, as described below, in that each caters to a particular higher-level presentation. SMARTS caters to POOMA's C++ implementation and its template-based approach. The Uintah software supports task graphs of C++ based mixed particle/grid algorithms on a structured adaptive mesh. Similar techniques are used by the well-known and successful Charm++ [11] framework which has a DAG-based dynamic runtime system and which has been successfully used on large scale parallel computers for a number of different applications.

3 Overview of Uintah Software

The Uintah Software was written in the University of Utah Center for the Simulation of Accidental Fires and Explosions (C-SAFE), a Department of Energy ASC center, which focused on providing science-based tools for the numerical simulation of accidental fires and explosions. Uintah is designed to solve complex multiscale multi-physics problems, by making use of a component design that enforces separation between large entities of software that can be swapped in and out, allowing them to be

independently developed and tested within the entire framework. This has led to a very flexible simulation package that has been able to simulate a wide variety of problems including a small cylindrical steel container filled with a plastic bonded explosive subjected to convective and radiative heat fluxes from a fire, [9], shape charges, stage-separation in rockets, the biomechanics of microvessels, the properties of foam under large deformation, and the evolution of large pool fires caused by transportation accidents. The application of Uintah to a petascale problem arising from "sympathetic" explosions in which the collective interactions of a large ensemble of explosives results in dramatically increased explosion violence, was described in [5].

Uintah currently contains four main simulation algorithms:
(i) ICE is a "multi-material" CFD algorithm that was originally developed by Kashiwa and others at LANL [12] for incompressible and compressible flow regimes. This method conserves mass, momentum, energy, and the exchange of these quantities between materials and is used here on adaptive structured hexahedral mesh patches.
(ii) The Material Point Method (MPM) is a particle method that is used to evolve the equations of motion for the solid materials applications involving complex geometries, large deformations, 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 adaptive mesh as a computational scratchpad.
(iii) MPMICE is fluid-structure solver that combines MPM and ICE [9, 22].
(iv) The fixed-mesh Arches component solves turbulent reacting flows with participating media radiation. It is a three-dimensional, Large Eddy Simulation (LES) code that uses a low-Mach number, variable density formulation to simulate heat, mass, and momentum transport in reacting flows, [10]. Where implicit solvers are needed Uintah components such as Arches or ICE use MPI-based solver libraries, PETSc [3] and Hypre [7]. Uintah was originally capable of running on 4K cores and has now also been released as software[1] and now runs on up to 196K cores on DOE's Jaguar at Oak Ridge Laboratory.

4 Uintah as viewed through the Silver Model

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, [22]. Uintah may be seen as a precursor of a Silver Model type code and so we now describe how the two approaches are related.

Parallel Computing Abstraction: Uintah utilizes an abstract task-graph representation 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, [21, 23] The task-graph allows the Uintah runtime system to

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

analyze the structure of the computation to automatically enable load-balancing, data communication, parallel I/O, and checkpoint/restart. The task-graph approach of Uintah also shares many features with the migratable object philosophy of Charm++ [11].
Overlap of Computation and Communication: The philosophy used in Uintah is that tasks should execute in an asynchronous manner as possible to provide as much overlap as is possible between computation and communication. 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. This procedure is completely asynchronous and has allowed parallelism to be integrated between multiple components while maintaining overall scalability, providing that there is enough work to keep each processor busy.

Patch Size	Scheduling Time	Regrid and Copydata	Task Wait MPI Time	Total Execution time
8	0.616	3.647	2.445	12.06
12	0.135	0.660	2.988	9.15
16	0.049	0.213	3.696	9.67
20	0.018	0.062	4.911	11.10

Table 1. AMR ICE Times as a function of mesh patch granularity.

Minimize Impact of Synchronization and other Overheads: Originally Uintah used a fixed execution pattern. After long MPI wait times were observed on large numbers of cores the internal task scheduler was rewritten so as to use both dynamic scheduling and out-of-order execution [19]. In particular the dynamic task scheduling mechanism in Uintah now allows tasks to run out of the sequential order that they are specified in the algorithm, if information obtained at runtime shows that this is permissible. Since Uintah is a general computational framework, it supports various tasks which may have asynchronous communications to different neighbors, write global variables, or even call third party libraries such as PETSc. The dynamic scheduler must be robust enough to guarantee that all these kinds of tasks are processed in such a way as to provide the correct result. This was accomplished by putting fine-grained computational tasks in a directed acyclic graph (DAG) and isolating the task memory. To achieve high scalability, a decentralized scheduling scheme was used; that is, each node schedules its tasks privately and communicates with other nodes regarding data dependencies only when necessary. Furthermore, Uintah's scheduler respects task priorities and supports scheduling global synchronization tasks. In order to create as many independent tasks as possible, we allow multiple versions of memory by adding a variable version table. This can help the system remove certain task dependencies and generate more independent tasks. Experiments in [19] varied mesh patch size on 24K cores and looked at trade-off between fewer larger patches, with fewer longer messages, less overhead and fewer parallel tasks and smaller patches with more tasks and more overhead and more shorter messages. Table 1 shows that a balance between sufficient parallel slackness and the associated overhead from scheduling regridding and wait time, was reached with patches of size 12x12x12 minimizing run time. This example illustrates some of the performance variations possible in a dynamically scheduled task-based execution

environment.

Adaptive Resource Management: A low-cost load balancing method is an important part of adaptive resource management in Uintah. Uintah's load balancer utilizes space-filling curves in order to cluster patches together [14, 16]. This algorithm was driven by using a simple model of computational cost on each patch. For more complex situations such as adaptive mesh refinement, and combinations of complex physics involving rapidly moving particles and adaptive meshes, it becomes increasingly difficult to have a reliable cost model that reflects potential changes at every timestep. In order to address this imbalance, a new measurement and feedback-based approach 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. This forecasting method is a simple exponential smoothing method [14], that has been used in a wide variety of applications because of its accuracy and simplicity. Although measurement based approaches have been used before, part of the reliability and robustness of this approach comes from the feedback loop.

Support for Heterogeneous computing: The standard message passing paradigm that Uintah initially operated under was that any data that needed to be shared to a neighboring core must be passed via MPI. For multi-core architectures, the process of passing data that is local to a node is both wasteful in terms of latency from MPI sends and receives and in the duplication of identical data that is shared between cores. For these reasons we have moved to an architecture in which only one copy of global data is stored per node in Uintah's data warehouse. The task scheduler now spins off tasks to be executed on, say, nc cores using a threaded model, [18] which results in the memory used in a single shared data warehouse being a fraction of only nc^{-1} of what is required for multiple MPI tasks, one per each of the nc cores. Figure 1 shows dramatic memory saving and even more dramatic memory decrease when the MPICH buffer sizes are constrained. The memory saving from eliminating the duplication of data within a node allows us to to expand the scope and range of problems that we have been unable to explore up until now. This architecture also offers the possibility of being extended to spin off tasks to be executed on other types of processors, such as GPUs, in the near future.

5 The Silver Model and Applications

Regardless of the support provided by task-based infrastructures such as Uintah, in order for the underlying problem to scale in both a weak and strong sense, sufficient parallel slackness, [29], and linear complexity are both required [17]. An important feature of Uintah is its adaptive meshing capability and so, as a result, Uintah has had to rethink algorithms for mesh refinement from the well-known Berger-Rigoutsos [4] algorithm to the tiled algorithm proposed by [15] in which regular hexahedral patches or tiles are uniformly refined, [5, 14]. As each tile is searched, in parallel, for refinement flags without the need for communication and then refined if it contains refinement

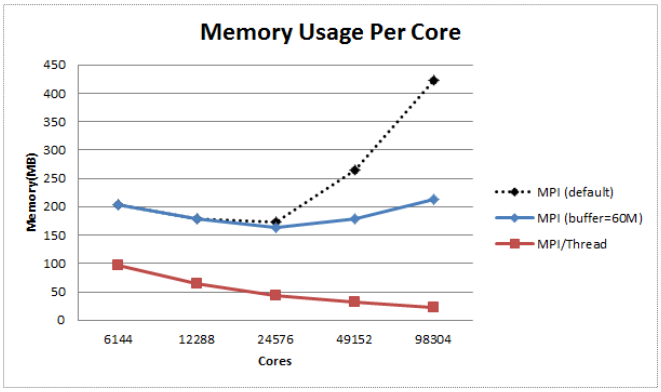

Fig. 1. Memory Reduction from Use of a Hybrid scheduler.

flags. This regridder is advantageous at large scales because cores only communicate once at the end of regridding when the patch sets are combined. Figure 3 (top) shows a simple example of Berger-Rigoutsos type patches and Figure 3 (bottom) shows the tiled approach applied to cylindrical refinement flags. Paradoxically the smaller number of patches resulting from the Berger-Rigoutsos algorithm makes it harder to distribute those patches evenly to large numbers of cores and the global communication required (or lack thereof) in each case is also significant, [15]. Figure 2 shows the performance

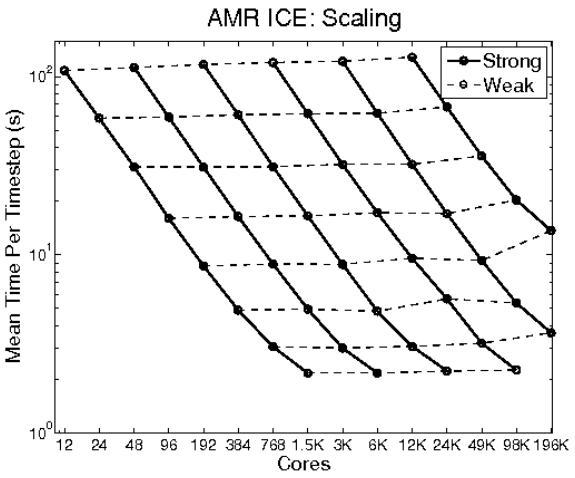

Fig. 2. Uintah Adaptive Mesh Scalability

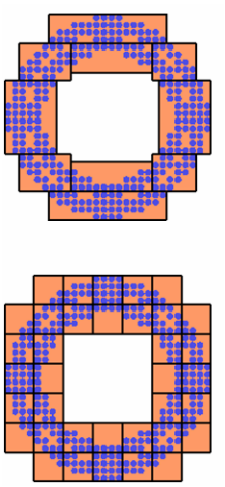

Fig. 3. BR and Tiled Patch Example

results for an extended version of the time-dependent adaptive mesh refinement problem

discussed in [5, 14]. The largest strong scaling case, as defined by the right most solid line, shows scaling to 196K cores. It was not previously possible to run this problem due to a shortage of memory per core on Jaguar. While these results are preliminary they are very promising. Similar scalability results to 196K cores have also been obtained for the combustion problem relating to sympathetic explosions in [5].

6 Using the DAG Approach at the multi-core level with Wasatch

The complexity of the problems to which simulation is applied naturally increases with available computing power. For example turbulent combustion simulation of typical fuels involves $\mathcal{O}(10) - \mathcal{O}(100)$ species and $\mathcal{O}(10) - \mathcal{O}(10^3)$ reactions. This requires solution of very large sets of highly coupled, nonlinear PDEs that span many orders of magnitude in both space and time. In such highly dynamic, multi-physics systems, one may not be able to determine *a priori* the most appropriate models. Therefore, programming models which allow significant flexibility in the complex couplings that may occur for different model sets in multi-physics applications are required. In the "expression" approach proposed by Sutherland, [20], the programmer writes pieces of code that calculate various mathematical expressions, explicitly identifying what data the code requires and produces/calculates. To create an algorithm, the programmer selects one or more expressions to be evaluated and the dependencies are recursively "discovered" resulting in a dependency graph. The dependency graph may be inverted to obtain the execution graph, which may be traversed in parallel if desired.

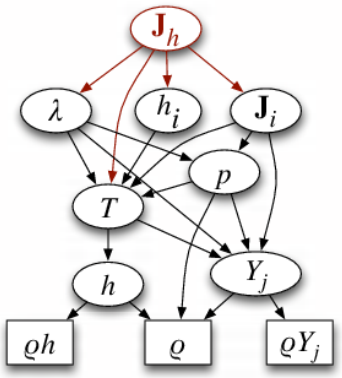

Fig. 4. Expression Tree for Heat Flux

As a simple example, consider a situation where we are solving PDEs for density (ρ), species mass (ρY_i), and enthalpy(ρh). One term in these equations is the energy diffusive flux, given by

$$\mathbf{J}_h = -\lambda \nabla T - \sum_{i=1}^{n_s} h_i \mathbf{J}_i, \qquad (1)$$

where λ is the thermal conductivity, h_i is the enthalpy of species i, and $\mathbf{J}_i$ is the mass diffusive flux of species i, given by another constitutive relationship. Defining the expression that calculates $\mathbf{J}_h$ as the root of the tree, we discover that we require expressions for λ, T, h_i and $\mathbf{J}_i$. We recursively obtain the graph shown in Figure 4. Each node in the graph represents a calculation performed over a subset of the mesh and can represent non-trivial operations. In Figure 4 boxes represent solution variables while ovals represent expressions and it is assumed that diffusivities are obtained from full kinetic theory and are functions of T, p, and Y_i. In the case of constant diffusion coefficients λ and D_{ij} do not depend on other values and the graph is simpler in structure. Moreover

for any such changes in dependencies, the expressions for λ and $\mathbf{J}_i$ are simply modified and the tree recompiled, with no logic changes in the application software. Similarly, the constitutive relationship for $\mathbf{J}_i$ can be easily modified without any direct implications on the expression for $\mathbf{J}_h$. While this is a very simple example, it illustrates the concept that complex relationships can be represented and abstracted well through the proposed expression approach.

In addition to the DAG expressions which expose the dependency and flow of the calculation, we employ an operator approach over strongly typed fields to form a domain specific language to achieve abstraction of field operations, including application of discrete operators such as interpolants, gradients, etc. This abstraction allows the programmer to work with fields in a MATLAB (vectorized) style while maintaining full compile-time type safety, ensuring that only valid field-field and operator-field operations can be performed. Furthermore, this level of abstraction also allows vectorized field operations to be automatically dispatched in parallel transparently to the programmer. Combined with the DAG strategy outlined above and the graph decomposition provided by Uintah, this provides three independent levels of parallelism that are easily exposed and exploited. Finally, when software is written using templated types, it is relatively simple to implement automatic differentiation techniques for C++ code. Figure 5 shows strong scaling up to 512 processes, using the DAG expression approach. There are several curves, representing an increasing number of threads. For example, the 8 thread curve at 512 processes implies 64 MPI processes running on 64 nodes with 8 cores per node and 8 threads per node. Similarly, 1 thread at 512 processes implies 512 MPI processes running on 64 nodes with 8 cores per node. The threads are doing the task graph decomposition using the expression approach. These results are for a set of 16 PDEs that have all-to-all coupling in their source terms and also have spatial operations going on. The key point is that overall this approach is very competitive against a straight domain decomposition approach.

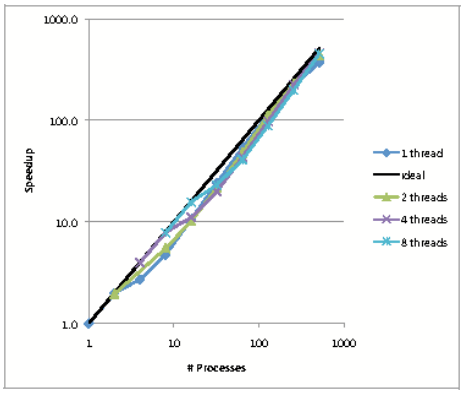

Fig. 5. Scalability of Wasatch Code.

7 Summary

The Uintah results show that the DAG approach has promise for complex adaptive mesh calculations and is worth pursuing. The recent work on Wasatch shows that the same idea also has great promise in simplifying the solution of very complex PDE problems and in automating several parts of the parallel computation pipeline in a multi-core environment . In particular, the conclusion is that algorithm-decomposition parallelism, enabled by the expression approach, has the potential to be used in conjunction with

MPI-based domain decomposition, as used in Uintah, to enable efficient scaling on modern multi-core architectures.

8 Acknowledgments

This work was supported by the National Science Foundation under subcontracts No. OCI0721659 and the NSF OCI PetaApps program, through award OCI 0905068, by DOE INCITE award CMB015 for time on Jaguar and by DOE NETL under NET DE-EE0004449.

References

1. S. Amarasinghe, D. Campbell, W. Carlson, A. Chien, W. Dally, E. Elnohazy, M. Hall, R. Harrison, W. Harrod, K.Hill, J. Hiller, S. Karp, C. Koelbel, D.Koester, P. Kogge, J.Levesque, D. Reed, V. Sarkar, R.Schreiber, M. Richards, A. Scarpelli, J.Shalf, A.Snavely, and T. Sterling. Exascale computing study: Software challenges in achieving exascale systems. Technical Report ECSS Report 101909, Georgia Institute of Technology, 2009.
2. S. Atlas, S. Banerjee, J.C. Cummings, P.J. Hinker, M. Srikant, J.V.W. Reynders, and M. Tholburn. POOMA: A high-performance distributed simulation environment for scientific applications. In *Supercomputing '95 Proceedings*, December 1995.
3. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Soft.Tools in Scien. Comput.*, pages 163–202. Birkhauser, 1997.
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. M. Berzins, J. Luitjens, Q. Meng, T. Harman, C.A. Wight, and J.R. Peterson. Uintah - a scalable framework for hazard analysis. In *TG '10: Proceedings of the 2010 TeraGrid Conference*, New York, NY, USA, 2010. ACM.
6. A. Chandramowlishwaran, K.Knobe, and R. Vuduc. Performance evaluation of Concurrent Collections on high-performance multicore computing systems. In *Proc. IEEE Int'l. Parallel and Distributed Processing Symp (IPDPS)*, Atlanta, GA, USA, April 2010.
7. R.D. Falgout, J.E. Jones, and U.M. Yangi. The design and implementation of hypre, a library of parallel high performance preconditioners. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 267–294. Springer-Verlag, 2006.
8. G.Bosilca, A. Bouteiller, A. Danalis, M. Faverge, H. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemariner, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra. Distibuted dense numerical linear algebra algorithms on massively parallel architectures: Dplasma. Technical report, Innovative Computing Laboratory, University of Tennessee, 2010.
9. 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.
10. J.Spinti, J. Thornock, E. Eddings, P.J. Smith, and A. Sarofim. Heat transfer to objects in pool fires, in transport phenomena in fires. In *Transport Phenomena in Fires*, Southampton, U.K., 2008. WIT Press.
11. 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.
12. 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.

13. J. Kurzak, H. Ltaief, J.Dongarra, and R. Badia. Scheduling dense linear algebra operations on multicore processors. *Concurrency and Computation: Practice and Experience*, Vol. 22, no. 1:pp. 15–44, 2010.
14. 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.
15. J. Luitjens and M. Berzins. Scalable parallel regridding algorithms for block-structured adaptive mesh renement. *Concurrency And Computation: Practice And Experience*, 2011.
16. 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.
17. 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.
18. Q. Meng, M. Berzins, and John Schmidt. Using hybrid parallelism to improve memory use in the Uintah framework. In *TG '11: Proceedings of the 2011 TeraGrid Conference*, New York, NY, USA, 2011. ACM (accepted).
19. Q. Meng, J. Luitjens, and M. Berzins. Dynamic task scheduling for the Uintah framework. In *Proceedings of the 3rd IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10)*, 2010.
20. P.K. Notz, R.P. Pawlowski, and J. C. Sutherland. Graph-based software design for managing complexity and enabling concurrency in multiphysics pde software. ACM Transactions on Mathematical Software (submitted).
21. S. G. Parker. A component-based architecture for parallel multi-physics pde simulation. *Future Gener. Comput. Syst.*, 22(1):204–216, 2006.
22. 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.
23. S. G. Parker, J. E. Guilkey, and T. Harman. A component-based parallel infrastructure for the simulation of fluid structure interaction. *Eng. with Comput.*, 22(3):277–292, 2006.
24. V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA, 1989.
25. V. Sarkar, S. Skedzielewski, and P. Miller. An automatically partitioning compiler for sisal. In *Proceedings of the conference on CONPAR 88*, pages 376–383, New York, NY, USA, 1989. Cambridge University Press.
26. O. Sinnen, L.A. Sousa, and E. S. Frode. Toward a realistic task scheduling model. *IEEE Trans. Parallel Distrib. Syst.*, 17:263–275, March 2006.
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. S. Vajracharya, S. Karmesin, P. Beckman, J. Crotinger, A. Malony, S. Shende, R. Oldehoeft, and S. Smith. Smarts: Exploiting temporal locality and parallelism through vertical execution, 1999.
29. L.G. Valiant. *Optimally universal parallel computers*, pages 17–20. Prentice Hall Press, Upper Saddle River, NJ, USA, 1989.
30. V.Sarkar, W.Harrod, and A.E Snavely. Scidac review: Software challenges in extreme scale systems. *Journal of Physics: Conference Series 180 012045*, 2009.
31. Z.Budimlic, M.Burke, V. Cavé, K. Knobe, G. Lowney, R.Newton, J.Palsberg, D.M. Peixotto, V.Sarkar, F.Schlimbach, and S.Tasirlar. Concurrent collections. *Scientific Programming*, 18(3-4):203–217, 2010.