Lawrence Berkeley National Laboratory

Lawrence Berkeley National Laboratory

Title
Performance and scaling of locally-structured grid methods for partial differential equations

Permalink
https://escholarship.org/uc/item/26k7w61v

Authors
Colella, Phillip
Bell, John
Keen, Noel
et al.

Publication Date
2008-04-18

Performance and Scaling of Locally-Structured Grid Methods for Partial Differential Equations

Phillip Colella, John Bell, Noel Keen, Terry Ligocki, Michael Lijewski, Brian Van Straalen

Computational Research Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720

E-mail: `PColella@lbl.gov`

Abstract. In this paper, we discuss some of the issues in obtaining high performance for block-structured adaptive mesh refinement software for partial differential equations. We show examples in which AMR scales to thousands of processors. We also discuss a number of metrics for performance and scalability that can provide a basis for understanding the advantages and disadvantages of this approach.

1. Introduction

A broad range of applied PDE problems exhibit multiscale behavior, i.e. variation in the solution over scales that are much smaller than the global large scales in the problem. Examples include flame fronts arising in the burning of hydrocarbon fuels and nuclear burning in supernovae; in geophysical problems, ocean currents, effects of localized features in orography or bathymetry, and tropical cyclones; and in plasma physics, a variety of small scale effects due to nonlinear instabilities and localized kinetic effects. In all of these problems, the fundamental mathematical description is given in terms of various combinations of PDE of classical type (elliptic, parabolic, hyperbolic). To effectively compute solutions to such problems, we need simulation capabilities with the following features.

- Multiresolution / adaptive methods: discretization methods that locally adjust the resolved length scales as a function of space, time, and the solution.
- Semi-implicit or fully-implicit methods for computing long-time dynamics in the presence of stiff fast dynamics.
- High-performance, scalable implementations.

In grid-based methods for numerical PDE, it is necessary to make a fundamental choice of discretization technologies. We have chosen to use locally structured grids, i.e. ones that are based on defining discrete unknowns on a rectangular discretization of the spatial independent variables. Specifically, we mostly use the finite-volume approach, in which the rectangular grid defines a collection of control volumes, for which a natural discretization of the divergence operator is obtained by integrating over the control volume. This leads to methods that satisfy discrete conservation laws, an essential feature if one is computing discontinuous solutions to PDE, and a desirable property for a much larger class of physical problems. There is a large

body of experience in how to construct stable and accurate discretizations to PDE based on this approach for a broad range of physics problems Regularity in space leads to regular data layouts, making it easier to optimize for various types of data locality. The regular geometric structure of the spatial distribution of unknowns leads to efficient iterative solvers for elliptic and parabolic problems.

Our adaptive mesh methods are based on the block-structured adaptive mesh refinement (AMR) algorithms of Berger and Oliger. In this approach, the regions to be refined are organized into rectangular patches of several hundred to several thousand grid points per patch. Thus one is able to use the high-resolution rectangular grid methods described above to advance the solution in time. Furthermore, the overhead in managing the irregular data is amortized over a relatively large amounts of floating point work on regular arrays. For time-dependent problems, refinement is performed in time as well as in space. Each level of spatial refinement uses its own stable time step, with the time steps at a level constrained to be integer multiples of the time steps at all finer levels. AMR is a mature technology for problems without geometry, with a variety of implementations and applications for various nonlinear combinations of elliptic, parabolic and hyperbolic PDE. In particular, the collection of applications here address most of the major algorithmic issues in developing adaptive algorithms for the applications described above, such as adaptive multigrid solvers for Poisson's equation, the representation of non-ideal effects in MHD, and the coupling of particle methods to AMR field solvers. AMR has also been successfully coupled to the mapped-grid and volume-of-fluid methods for treating irregular geometries.

A principal concern regarding this class of methods is whether they can be implemented to effectively use 10^4-10^5 processors. We want to address the following questions.

- Does AMR scale to this level of parallelism? Can the impact of irregular, dynamically varying loads on load balancing and communications intensity be controlled to a sufficient extent to obtain acceptable performance ?

- How does one understand and improve the performance of AMR algorithms and software (both scaling and absolute performance)?

- Under what circumstances does AMR provide the best scientific results for the least cost?

- How do we design a benchmark suite that encapsulates the usage patterns for a broad range of applications?

2. The AMR Programming Model

The BoxLib / Chombo libraries support a wide variety of applications that use AMR by means of a common software framework. The design approach used here is based on two ideas. The first is that the mathematical structure of the algorithm domain specified above maps naturally into a combination of data structures and operations on those data structures, which can be embodied in C++ classes. The second is that the mathematical structure of the algorithms can be factored into a hierarchy of abstractions, leading to an analogous factorization of the framework into reusable components, or layers. This reusability is realized by a combination of generic programming and sub-classing. A principal advantage to this design is the relative stability of the APIs as seen by the applications developer. While implementations may change considerably to enhance performance or in response to changes in the architecture, these changes are less likely to cause major upheavals to the applications programs. This is because the APIs reflect the mathematical structure of the algorithms, which remain relatively fixed targets.

The BoxLib / Chombo libraries provide high-level support for domain decomposition based on assigning rectangular patches to processors. All processors have access to processor assignment metadata, and distributed grid data is defined in terms of these metadata. There are two types of operations that can be performed on such distributed data. Local computation is performed

by iterating over patches assigned to the processor, which is able to access only those data. The second type of operation consists of communication primitives. Aggregate operations for exchanging ghost cell data among all of the patches on a given union of rectangles, and for copying from a data defined on a disjoint union of rectangles to data defined over some other union of rectangles. the other layers of the library are built using these primitive operations. These include interlevel operations that combine communication and irregular computation, such as interpolating boundary data, and averaging or interpolating between levels; control structures, such as multigrid iteration, or Berger-Oliger timestepping for refinement in time; and complete applications.

3. Defining Scalability and Performance

The focus of the present work is on methods for solving PDE that are *algorithmically scalable*, i.e. the number of floating-point operations and the amount of memory required for solving a problem (in the steady-state case) or advancing the solution by one time step (in the time-dependent case) are linear functions of the number of unknowns, with constants of proportionality independent of the mesh spacing. This is a property of the discretization methods and solution algorithms, rather than of parallel implementations, and represents a kind of algorithmic optimality, since the computational effort and memory for computing the solution is proportional to that required to evaluate the operator. Many modern methods for classical PDE are algorithmically scalable: explicit methods for hyperbolic problems, and multigrid-based methods for elliptic and parabolic problems.

There are two possible ways in which to apply more computing power to solving a problem. In strong scaling, we hold the size of the problem fixed, and increase the number of processors used to solve the fixed-size problem. Ideally, strong scaling leads to a reduction of the time to solution proportional to the number of processors. In weak scaling, we increase the size of the problem with the number of processors. For algorithmically scalable methods in numerical PDE, we are concerned with a specific form of weak scaling, in which the number of computational unknowns per processor is fixed. This is a scaling limit that is compatible with algorithmic scalability, since both the computational effort and the memory required / available scale linearly with the number of processors. In the weak scaling limit, we expect that the time to solution would be fixed, or at least bounded, independent of the number of processors, as the number of processors increases.

In many applications of numerical methods for PDE, the primary use of large number of processors on a single job is a weak scaling process. There are a number of reasons why this strategy makes sense.

- Even with the use of adaptive grids, many problems that drive the scientific agenda are out of reach due to inadequate grid resolution. In those cases, increasing the number of processors is used to increase the spatial grid resolution.
- Applications codes use the minimum number of processors so that the problem will fit into memory. Flops are free, memory is expensive.
- Any speedup obtained by increasing processor count is often lost due to realities of a multiuser environment. Doubling the number of processors to decrease the time to solution might cause a job to wait in the queue for longer than the time gained.

There are a few counterexamples to the primacy of weak scaling in applied PDE, mainly in settings such as numerical weather prediction, in which large-scale PDE simulations are used for real-time applications. Even for those applications, weak scaling studies are good at exposing bottlenecks.

Under weak scaling, Amdahl's law is replaced by much less restrictive conditions on load balancing. For example, for fixed-sized patches, a bound on t_{patch}, the execution time per patch,
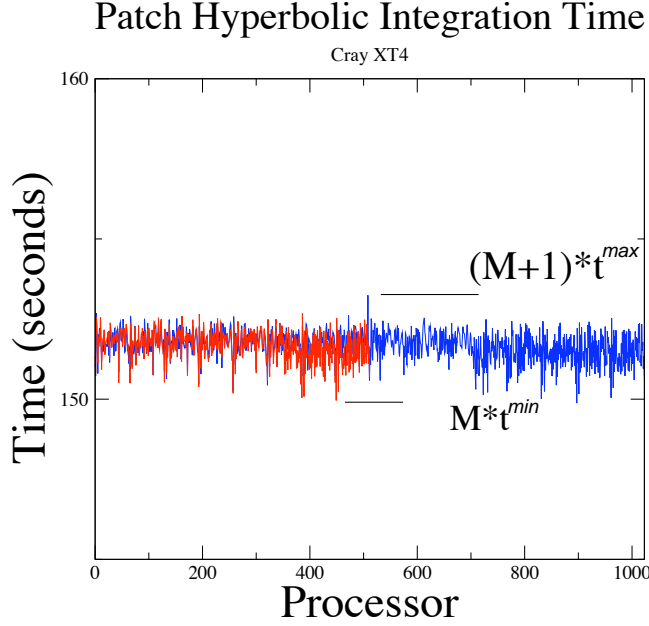
Figure 1. Time spent in the patch update as a function of processor number on a union of fixed-size patches for an unsplit PPM algorithm, for 512 processor (red) and 1024 processor (blue) calculations. Since the number of patches per processor is bounded independent of the number of processors, this calculation exhibits perfect weak scalability, with the time to execute this operation is bounded independent of the number of processors as we scale the problem size up.

that is independent of the number of processors, leads to a bound on the time to solution in the weak scaling limit. If $t_{min} \leq t_{patch} \leq t_{max}$, then the wall clock time t_{wall} to compute the solution is bounded above and below.

$$t_{min}M \leq t_{wall} \leq t_{max}(M+1) \ , \ M = \left\lfloor \frac{N_{patches}}{N_{proc}} \right\rfloor \tag{1}$$

Here $N_{patches}$ is the number of rectangular patches, N_{proc} is the number of processors, and M is to remain fixed as $N_{patches}$, N_{proc} vary (Figure 1).

In addition to scalability, there are other measures of performance.

- Stencil operations arising in PDE calculations typically can achieve a only small fraction (10% -20%) of the nominal peak performance on modern cache-based processors. We define *the operator peak performance* as the time to evaluate on a single processor a "typical" operator characterizing the problem being solved. We assess uniprocessor and overall aggregate performance in terms of flop rate as a fraction of the operator peak performance.

- We define the *adaptivity factor* to be the ratio of the time to perform a calculation on a uniform grid to the time to solution for the AMR calculation with the same accuracy, typically requiring the finest grid in the AMR hierarchy to have the same mesh spacing as the uniform grid calculation. The uniform grid performance is generally estimated from smaller runs, assuming perfect weak scaling, rather than computed directly. The adaptivity factor differs from the fraction of operator peak performance, in that it also takes into account the costs incurred due to parallelization of the uniform grid calculation.

- *Implementation efficiency* is the fraction of time spent on regular computation (e.g. in Fortran77 or other optimized single-patch operations). For the examples described here, implementation efficiency is very close to the fraction of operator peak performance.

4. AMR Parallel Benchmark Calculations

4.1. Replication Scaling Benchmarks

Classically, weak scaling studies for numerical methods for PDE on uniform grids have been done using mesh refinement: to scale up the problem, refine the grid by an integer factor in each direction, and increase the number of processors to hold the number of grid points per processor fixed. The corresponding exercise for AMR would be to refine the coarsest grid by an integer factor, and decrease the error tolerance so that the resolution at each level is increased by the same integer factor. In practice, such an approach leads to scaling behavior that is difficult to interpret. Under such a refinement scheme, the size of the refined regions at each level can change by significant amounts, most often to decrease the physical size of the refined region at a given level. This offsets the loss of scaling due to other causes, and makes it difficult to detect failures to scale, and to understand the causes of such failures. To eliminate these problems, we have used benchmarks based on *replication scaling*, in which we take a grid hierarchy and data for some number of processors, and scale up the problem by making identical copies (Figures 2,3). The full AMR code (processor assignment, remaining problem setup) is done without using the knowledge that the grids have been replicated. Replication scaling tests most aspects of weak scalability, is simple to define, and provides results that are easy to interpret. Thus it is a very useful tool for understanding and correcting scaling difficulties. Furthermore, it is a good proxy for some kinds of applications scaleup. For example, a large part of the simulation of a gas turbine will be the simulation of multiple identical burners arranged in a ring. Replication scaling does not fully test load balancing, in the sense that the parts of the calculation for which the replicated components are not coupled to one another may not increase the degree of load imbalance as we scale the calculation up. Thus the results obtained using replication scaling will need to be supplemented with other measurements to obtain definitive scaling behavior, such as showing that the $t_{wall} \times N_{proc}$ divided by the number of grid points ("grind time") is bounded in the weak scaling limit for a more traditional AMR mesh refinement study.

4.2. Gas Dynamics Benchmark

We benchmarked an explicit method for unsteady inviscid gas dynamics in three dimensions, based on the unsplit PPM algorithm [4, 7]. This algorithm requires about 6000 flops to update a grid point. Since it is an explicit method, ghost cell values are copied or interpolated only once per update. We used the implementation of this method "out of the box" from the Chombo software distribution, without significant modification. The operator peak performance for this method on the Cray XT4 was 530 Mflops / processor.

The single image used as the starting point for the replication benchmark is a spherical shock tube in 3D, with finest grids covering a spherical shell (Figures 4,5). There are two levels of refinement, factor of 4 each, with refinement in time proportional to refinement in space. We use fixed-sized 16^3 patches and a total of 6.2×10^7 grid points (five unknowns per grid point), with 10^9 grid point updates performed for the single coarse time step. In the results given here, we are only timing the cost of computing a single coarse time step, not the problem setup and initialization.

In Figure 6, we show plots of wall clock time for the total calculation on the Cray XT4, as well as for various phases of the calculation. In the plots, we normalize the times by the time to solve the smallest problem (2 images, 128 processors). We observe 96% efficient scaled speedup over a range of 128-8192 processors, corresponding to a wall clock time of 177 ± 4 seconds to compute 2×10^9 - 1.28×10^{11} grid-point updates. The fraction of operator peak for
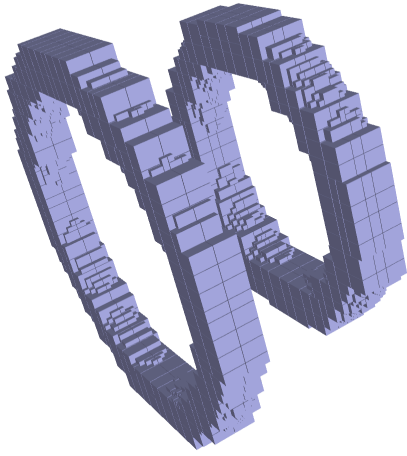
Figure 2. Single pair for rings for replication scaling of Poisson's equation. Image shows the collection of rectangular patches for this problem.
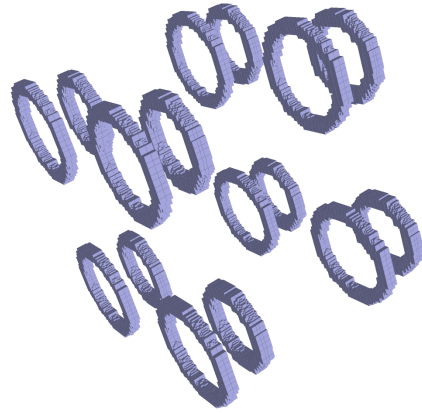

Figure 3. Ring grids replicated by a factor of two in each coordinate direction.
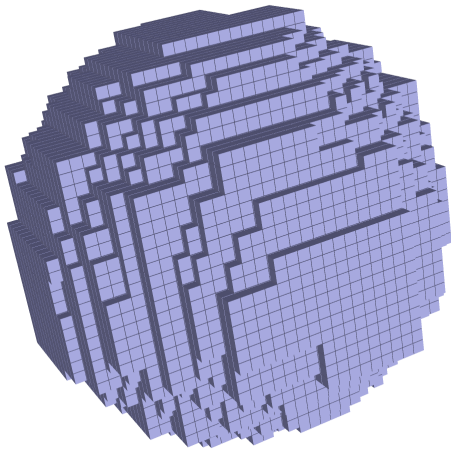

Figure 4. Single spherical shell grids for replication scaling of Poisson's equation. Image shows the collection of rectangular patches for this problem.
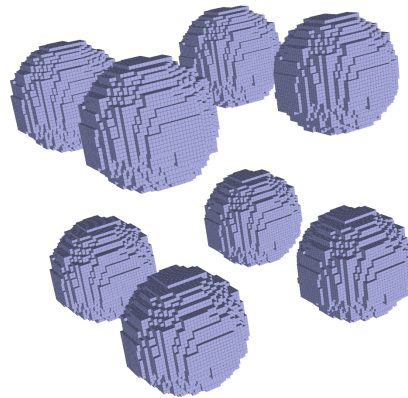

Figure 5. Spherical shell grids replicated by a factor of two in each coordinate direction.

these calculations was around 85% (450 Mflops / processor), with an adaptivity factor of 16. While we are concerned about the small fluctuations (of both signs) around perfect scaling, and the somewhat low single-processor performance as reflected in the operator peak performance, these results are consistent with the long-observed view that the use of AMR for hyperbolic problems scales well on large numbers of processors.
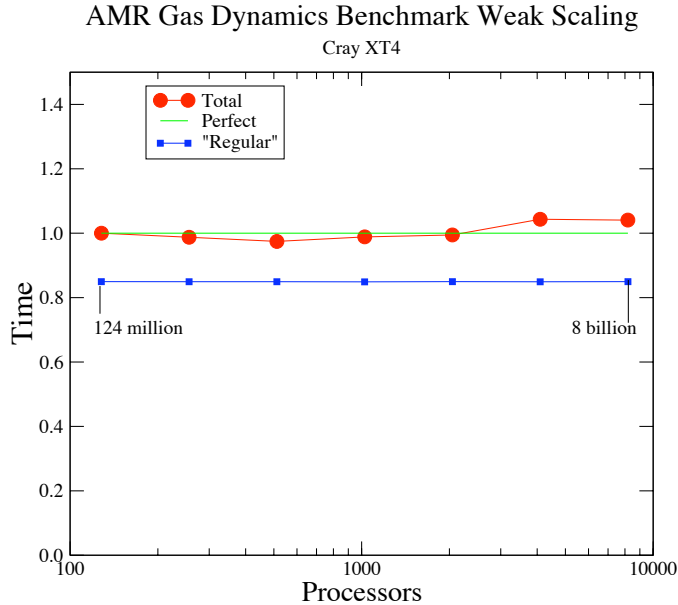
AMR Gas Dynamics Benchmark Weak Scaling

Figure 6. Gas dynamics weak scaling results. Red line shows wall clock time for a single coarse time step, and the blue line the time spent in regular grid computations. All times are scaled to the total time for the 128 processor results. Perfect scaling for the total calculation is shown as the horizontal green line.

4.3. Poisson Benchmark

We benchmarked an AMR solver for Poisson's equation, based on a cell-centered multilevel discretization of Laplacian in three dimensions [6]. The solver itself is based on multigrid iteration suitably modified for use on AMR grid hierarchies [9, 2]. The benchmark was run to apply ten iterations of the AMR-multigrid V-cycle, which is typical of the number of iterations required for acceptable reduction of the residual, and corresponds to 1700 flops / grid point. This is a much more demanding application from the standpoint of parallelism, requiring multiple computations of ghost points and other communications steps per multigrid interation. The algorithmic features of this benchmark are typical of broad range of elliptic solvers arising in applications using AMR.

The single image used as the basis of for the replication benchmark are the two rings (Figures 2, 3). The grids have two levels of refinement, with a refinement ratio of four for each. Patch size is allowed to vary between 8^3 and 32^3. there is one unknown per grid point, total of 15M grid points per image. An operator peak performance on XT4 of 840 Mflops / processor was measured by timing the application of the Laplacian operator to a single rectangular patch. As was the case for the gas dynamics benchmark, we are timing only the solver, not the setup time. To obtain the scaling results presented here required a good deal more effort than in the hyperbolic case. We required significant effort in code optimization (2 months), leading to 10X improvement in both per-processor performance and in scalability.

In Figure 7, we show plots of wall clock time for the total calculation on the Cray XT4, as well as for various phases of the calculation. In the plots, we normalize the times by the time to solve the smallest problem (32 images, 256 processors). We observed 87% efficient scaled speedup over range of 256-8192 processors, corresponding to a wall clock time of 8.4-9.5 seconds. the fraction of operator peak for these calculations was around 45% (375 Mflops / processor). The adaptivity factor was estimated to be 48. We did not have a uniform-grid method available
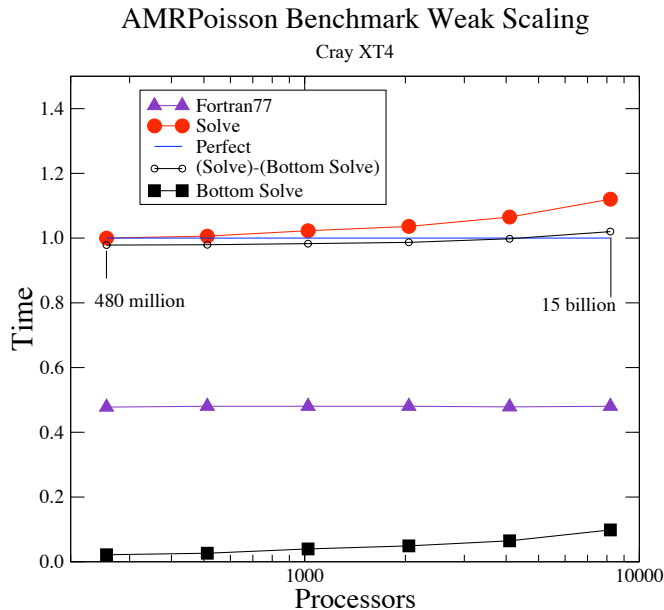
AMRPoisson Benchmark Weak Scaling

Figure 7. AMR Poisson weak scaling results. All times are scaled to the total time for the 256 processor results. The red line shows total time, and the purple line the time spent in regular grid computations. Pefect scaling for the total calculation is shown as the horizontal green line. The black squares shows the time spent using BiCGStab to solve on the coarsest multigrid level. The difference between the BiCGStab solver time and the total time to solution is plotted using black circles and follows very closely the perfect scaling line thus demonstrating that the BiCGStab solver time is the source of most of our lack of scalability in this calculation.

with which to compare, so our estimate was based on timings of the operator application on a uniform grid, and the assumption that the flop rate for the full solve would be the same as for the operator evaluation.

The detailed timing data shows that the deviation from perfect weak scaling can be attributed to a single part of the algorithm (Figure 7). We use BiCGStab, preconditioned with point relaxation, to solve the coarsened problem at the coarsest multigrid level. In our current implementation, the coarsest multigrid level corresponds to a coarsening of the coarsest AMR level, to the point where the domain consists of 2^3 patches, each on a different processor. This is not algorithmically scalable, and has relatively large communications costs. Our expectation is that we can eliminate this problem, for example, by moving the coarsest problem onto a much smaller number of processors, or even a single processor, where we can coarsen our multigrid levels down to a much smaller number of grid points, thus eliminating the lack of algorithmic scalability and reducing the communications overhead.

4.4. Methods for Obtaining High Performance
In order to obtain the scaling behavior reported here, particularly for Poisson's equation, we needed to address three major issues in our existing code base.

- Minimizing communications costs. We found it necessary to distribute patches in a way that minimizes communications costs using space-filling curves. If D is the spatial dimension of the problem, Morton ordering [8] is a 1-1 mapping of $\mathbb{Z}^D$ onto $\mathbb{Z}$ with good locality: the fraction of nearest neighbors in $\mathbb{Z}^D$ of the inverse image of an interval $\mathcal{I} \subset \mathbb{Z}$ of length M

whose Morton indices are not in $\mathcal{I}$ is $O(M^{-1/D})$. Load balancing is done by sorting the patches according to the Morton indices of their low corners, and dividing the linearly-ordered patches into intervals with equal workloads. The partitioning onto processors obtained using Morton ordering shows uniform distributions with only a small fraction of the patches having neighbors that are off-processor. This is in contrast to a recursive bisection approach that we had used previously, in which it is possible to have long thin partitions with all the patches requiring boundary data from off-processor. Morton ordering also makes it profitable at large numbers of processors (4096 and above) to overlap the local copying of ghost cell data from neighbors that are on the same processor and copying ghost-cell data from other processors using asynchronous MPI calls.

- Scalable computations of patch metadata. In current implementations of AMR, every processor has a copy of the metadata (assignment of patches to processors) for all unions of rectangles / processor assignments. These are used to compute intersection lists, e.g. from which patches is ghost-cell data to be copied. At 1000 processors and above, it is essential to use $O(log(N_{patch}))$ sorts and searches to compute intersection lists. Otherwise, there is a catastrophic failure to scale due to performing $O(N_{patch})$ computations on every processor. Even using fast methods, the cost of these computations are not negligible, so significant performance improvements were obtained from caching intersection lists.

- Optimizing coarse-fine boundary condition calculations Coarse-fine boundary conditions involve parallel communication and irregular computation. While these calculations are scalable, they can substantially increase the value of t_{max} in (1). We make agressive use of residual-correction form to minimize how often the coarse-fine boundary conditions are computed. For interpolation stencils that are actually regular, we call Fortran routines that implement them. Although we have not done so here, we could also use fixed-size patches to make nearly all such calculations regular, or develop fast irregular stencil operations.

5. Choosing AMR - A Case Study

APDEC supports the development of AMR for turbulent reacting flows based on the time-dependent low-Mach number fluid equations in three dimensions with detailed hydrocarbon chemistry and transport [5]. These are large systems. For example, there are approximately 60 primary dependent variables per grid point required to simulate methane combustion using the standard detailed chemical mechanism GRIMech 3.0, with most of these being chemical species variables. The low-Mach number combustion code LMC has been used to to investigate a variety of turbulent flames, and is being applied under SciDAC 2 to simulate syngas combustion under conditions characteristic of turbines used for power generation. For hydrocarbon combustion, the computational costs are dominated by those of solving ODEs at every grid point for chemistry, and of solving single-level variable-coefficient elliptic solvers that are used to impose divergence constraint that eliminates acoustic wave dynamics. The latter use multigrid-preconditioned BiCGStab, and pose a somewhat different set of problems than does the AMR Poisson benchmark discussed above. Because the LMC code uses refinement in time, the elliptic solvers are for problems that are defined on a single union of rectangles. At any AMR level except the coarsest one, there is a limit to how much we can coarsen the union of rectangles with the coarsened domain exactly covering the fine domain. This places additional demands on the scalability and performance of the solver at the coarsest level.

We benchmarked the LMC code using replication scaling (Figure 8). The single image is a wrinkled methane flame, using the chemical reaction mechanism referred to above. There are two levels of refinement, factor of 2 each, with a total of 4×10^6 grid points on the single image. For $N_{proc} \leq 1024$, the cost of the computation is dominated by the cost of solving the chemical rate equations. By rebalancing the data for this task on the fly, this part of the computation scales perfectly. There is a slight loss of weak scaling of the whole application for $N_{proc} = 4096$.

This lack of scalability is due to a lack of scalability of the variable-coefficient elliptic solvers used here. Similar scaling results for LMC were obtained on a Linux cluster (the LLNL Atlas machine), and on an SGI system (NASA Columbia).

The modest lack of scaling in principle could be viewed as a reason not to use LMC. However, algorithmic choices are driven by science requirements: of the available alternatives, what is going to provide the most scientific output for the least cost? The principal alternative to LMC is based on the use of a fully explicit method on a uniform grid. This approach, which has been the most extensively-used method for computing turbulent reacting flows for the last 20 years, uses explicit time discretization of the compressible Navier-Stokes equations with the chemistry and diffusive transport of species as in the LMC code. The fully explicit approach has the following properties.

- Explicit stencil operations scale perfectly.
- The time step is determined by CFL condition for acoustic waves (.02 μsec for the wrinkled-flame benchmark).
- For the chemical reaction terms, we use an explicit ODE method, and subcycle in time as needed.

These are to be compared to properties of the LMC code.

- Elimination of acoustic waves leads to a 300-fold increase in the time step (6 μsec for the wrinkled-flame benchmark). This comes at the cost of introducing elliptic solvers and an implicit treatment of the chemical reaction terms. The former leads to the small deviation from perfect weak scaling.
- AMR provides a tenfold reduction in the number of grid points over a uniform fine grid with the same resolution.

To see how these theoretical properties would work themselves out in practice, we implemented a fully explicit method and used it to solve the single image version of the wrinkled-flame benchmark, but using a uniform grid at the same finest resolution as the finest grid in the AMR LMC calculation. We extrapolated the results to larger number of processors assuming perfect scaling. The results are plotted in Figure 9. Overall, the introduction of implicitness and the use of AMR leads to an factor of 12-15 increase in the cost per grid point per time step over the fully explicit method. Nonetheless, we observe an improvement by a factor of 200-250 in time to solution by using LMC over fully explicit method on a uniform grid at the same effective resolution. The deviation from scalability is a miniscule effect relative to the difference between the approaches, and provides a compelling case for the use of the LMC approach from the standpoint of scientific productivity.

6. Conclusions and Future Plans

In this paper, we have presented two major results. The first is that AMR scales to 10^4 processors with per-grid-point performance comparable to the corresponding uniform-grid algorithms. The second is the performance study of the use of AMR in combustion simulations. We showed that AMR, combined with semi-implicit temporal discretizations, leads to a factor of 200 or more improvement to time to solution over fully explicit methods on uniform grids on a science application of importance to the DOE mission. To achieve these results, it was not necessary to make major changes in either the algorithm design or the software architecture. Due diligence and attention to details were sufficient, given that we had the computational resources to perform the measurements, and a suitably-designed AMR framework that localizes the software components that need to be modified to obtain high performance.

Over the next year, one of the principal goals will be for all of the libraries and applications supported by APDEC to scale to 10^4 processors. The results described here constitute a good
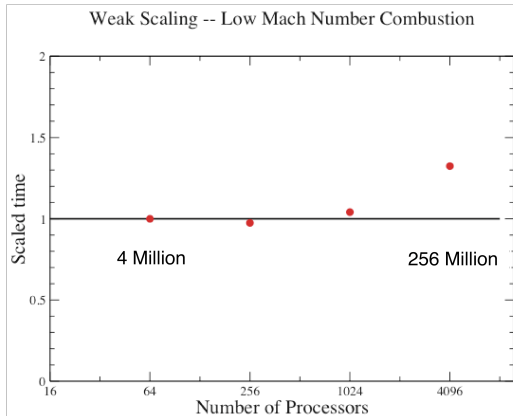
Figure 8. Scaling behavior for LMC code. The calculation consists of one coarse time step. The time is scaled by the time to compute the solution on 64 processors (1.2 seconds)
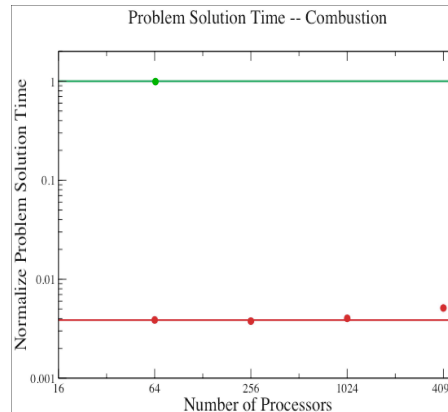
Figure 9. Comparison of performance of LMC code (red) and fully explicit code (green). Wall clock time is scaled by the time taken to compute the solution out to the same time as the LMC results on the left using the fully explicit code on 64 processors (300 seconds).

initial step in this direction, but there remains a considerable amount of work to be done. We need to address the scalability of the setup process for AMR, e.g. grid generation, load balancing, and other metadata operations. At 10^4 processors, tens of billions of grid points translates into millions of patches, for which all of the processors currently keep a copy of the metadata. At ~50 Bytes per patch, the size of the memory required to store the metadata becomes unacceptable. Possible approaches to reducing the memory footprint include sharing the metadata across several processors, or representing the patch metadata using a bitmap. In the latter case, this could be used in conjunction with a linear ordering of the data computable from knowing the bitmap information (e.g. Morton ordering), so that the metadata could be stored using one bit per patch, plus one long-long integer per processor.

The second area that will continue to receive attention is that of performance tuning of elliptic solvers, especially solvers defined on a single AMR level, i.e. a union of rectangles. As discussed in [1], the choice of parallelization strategies for the bottom solvers in multigrid can have a substantial impact on performance, and we expect that to be the case here, as well. We will also need to expand the range of elliptic solvers available to the users of APDEC software, including variable coefficient solvers, tensor solvers, and fourth-order accurate solvers; defined both on a single union of rectangles, and on multilevel AMR grid hierarchies.

Designing a benchmark suite that spans the usage patterns for a broad range of applications is a substantial undertaking. As well as the types of solvers and discretizations noted above, other issues include the variation of the layout of grids and grid hierarchies for different problem domains, and the development of metrics for judging solvers. Examples of the latter include scalability and performance along the lines discussed in this paper, as well as setup time and memory requirements. We plan to make our benchmark suite accessible to the larger scientific community to obtain feedback both from the applications users and other solver software developers.

Finally, we have not addressed in this paper the issues of extending the approach taken here to problems with anisotropies or significant amounts of irregular computation due to cut-cell treatment of geometries. In the latter case, we have a separate effort to improve the performance

and scaling of that algorithm domain, and we expect to obtain comparable scaling results to those seen here for AMR without geometry. AMR algorithm development for anisotropic solvers / problems (S_n radiation, geophysical fluid dynamics, thermal conduction in MHD) is still in its early stages. Moreover, parallelization for these problems promises to be considerably more difficult. For example, such problems often require the use of line solves in a stiff direction, leading to a loss of parallelism in that direction.

In considering the extension to 10^5 processors, the good news is that we still have plenty of parallelism left to exploit. If the increase in the number of processors is by introducing dozens of cores on a single chip, then hierarchical parallelization strategies of load balancing across chips, while using work-queues or fine-grain parallelism within loops executing over a single patch on a single chip, become attractive strategies. However, it is probably premature to make large investments in the development of software for this regime, until some of the major uncertainties regarding the computing environment at the petascale are resolved.

- Are we scaling to a flat processor space ? If so, what are the performance characteristics of MPI at 10^5 processors ? If not, What programming models will be used in the presence of hierarchical parallelism, and by what process will production-quality implementations of such programming models be developed ?
- The tools currently being used to write out, store, and access the data for analysis and visualization of time-dependent numerical PDE data not scale to the petabyte data sets that will be generated by 10^5 processors.
- Above 1000 processors, multiuser environments, intermittent hardware failures, and other artifacts can lead to unpredictable fluctuations in performance.
- Adequate access to large numbers of processors that enables software developers to run large jobs in a timely fashion is essential to software development for the high end. On the other hand, permitting such access may lead to lower utilization numbers for a production computing center.

Figure 10 provides an illustration of the last two points. In developing the gas dynamics benchmark, we noticed intermittent 10% fluctuations in the wall clock time for large numbers of processors. By performing repeated runs with detailed instrumentation, we were able to narrow the problem down to three specific hardware nodes that were experiencing single-bit memory errors [3], the correction of which caused the processor to slow down. Consequently the whole parallel section of code took 10% longer to finish. Once the problem with the memory was fixed, the spikes disappeared, and we obtained the near-perfect scaling results in Figure 6. As the number of processors increases, the risk of such intermittent hardware failures increases, with unpredictable impacts on performance. On the other hand, this experience shows the value of fast turnaround access for relatively short jobs on large numbers of processors. We were able to solve the problem within a few weeks, but only because we were able to perform repeated experiments using large numbers of processors to pinpoint the source of the difficulty.

We see no serious technical barriers specific to AMR to scaling to 10^5 processors and beyond. However, there are serious infrastructure issues to be resolved before the computational science community will be able to make effective use of such large numbers of processors. Software developers and applications users will need to be convinced that attempting to scale to such large numbers of processors will yield real science returns in a timely fashion.

References

[1] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos, "Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom", ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing, 2004.
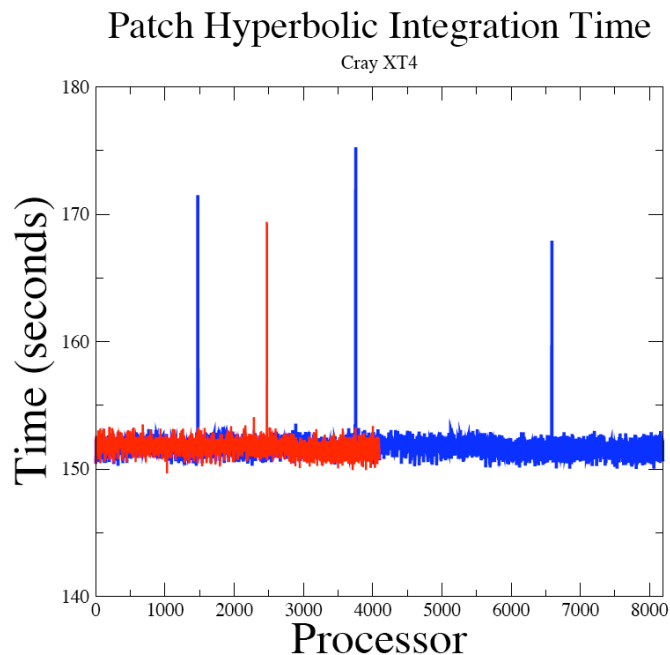
Patch Hyperbolic Integration Time
Cray XT4

Figure 10. Variation in loads for large numbers of processors. Red (4096 processor calculation) and blue (8192 processor calculation) show the time spent in the patch update as a function of processor number on a union of fixed-size patches for an unsplit PPM algorithm. The spikes correspond to specific hardware nodes, and were found to be caused by single-bit errors in the memory on those nodes [3].

[2] A. S. Almgren, T. Buttke, and P. Colella, "A fast adaptive vortex method in three dimensions" J. Comput. Phys., Vol.113, No.2 (1994), pp. 177-200.

[3] A. S. Bland, ORNL LCF, private communication, 2007.

[4] P. Colella and P. R. Woodward, "The piecewise parabolic method (PPM) for gas-dynamical simulations", J. Comput. Phys., Vol. 54, No.1 (1984), pp. 174-201.

[5] M.S. Day and J.B. Bell, "Numerical simulation of laminar reacting flows with complex chemistry", Combust. Theory Modelling Vol. 4, No. 1 (2000) pp. 535-556.

[6] D. F. Martin, P. Colella, and D. T. Graves, "A cell-centered adaptive projection method for the incompressible Navier-Stokes equations in three dimensions", LBNL report LBNL-62025, submitted to J. Comput. Phys.

[7] G. H. Miller and P. Colella, "A higher-order Godunov method for elastic-plastic flow in solids", J. Comput. Phys. Vol.167 (2001), pp. 131.

[8] G. M. Morton. "A computer oriented geodetic data base and a new technique in file sequencing". Technical report, IBM Ltd., Ottawa, Ontario, Mar. 1966.

[9] M. C. Thompson and J. H. Ferziger. "An adaptive multigrid technique for the incompressible Navier-Stokes equations", J. Comput. Phys. Vol. 82 (1989), p. 94-121.