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

Phillip Colella, Noel Keen, Terry Ligocki, Brian van Straalen
Applied Numerical Algorithms Group, LBNL

John Bell, Mike Lijewski
Center for Computational Sciences and Engineering, LBNL

SciDAC Applied Partial Differential Equations Center for Enabling Technologies (APDEC)

Office of Science
U.S. DEPARTMENT OF ENERGY

SciDAC
Scientific Discovery through Advanced Computing

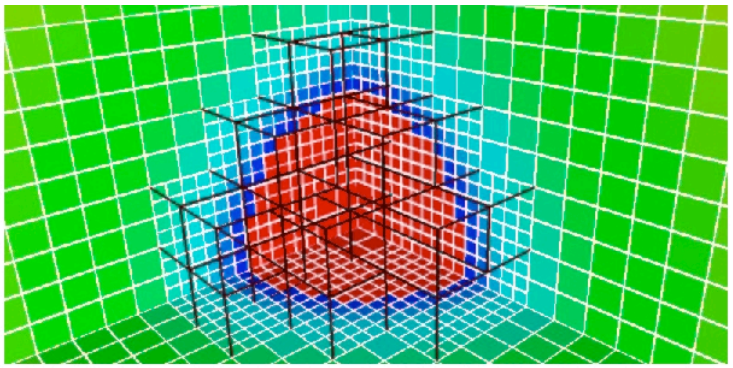
Multiscale Problems and Multiresolution Methods

- A broad range of applied PDE problems exhibit multiscale behavior.
  - Combustion: flames.
  - Astrophysics and cosmology: galaxy / star formation, nuclear burning.
  - Geophysical fluid dynamics: localized currents, orography / bathymetry, tropical cyclones.
  - Plasma physics: kinetic models of plasmas; nonlinear MHD instabilities of various kinds; fueling an MFE reactor.
  - Subsurface flows: geological features, fronts.

- Typically, these are represented mathematically by various combinations of PDE of classical type (elliptic, parabolic, hyperbolic). To effectively compute solutions to such problems, we need to meet the following requirements.
  - 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.

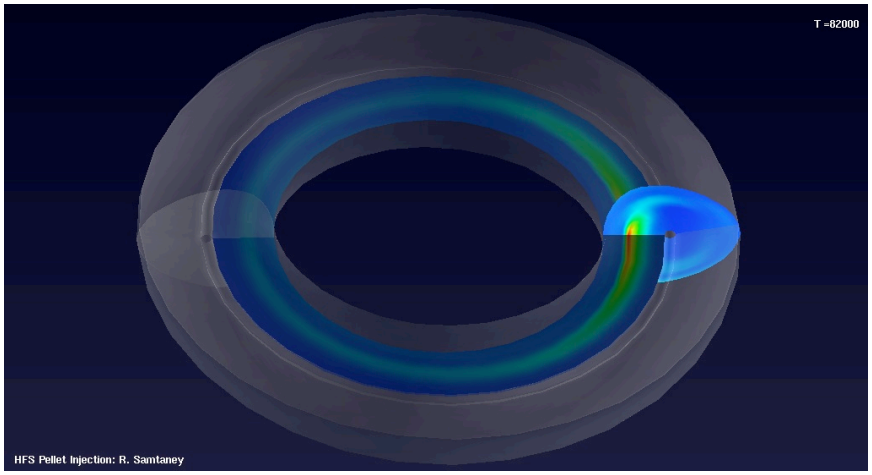
Block-Structured Adaptive Mesh Refinement (AMR)

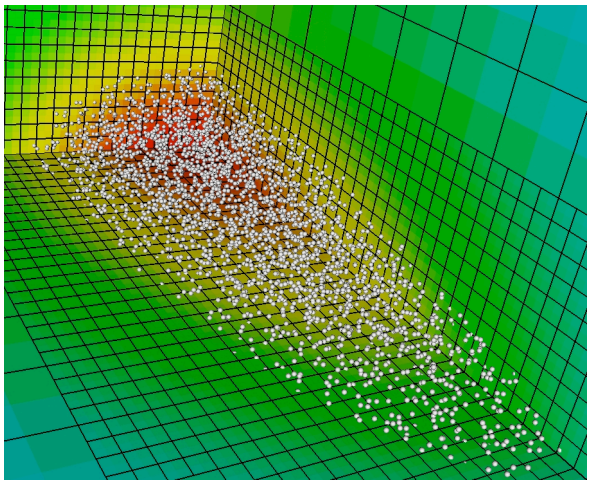
• Refined regions are organized into rectangular patches.



• Refinement in time as well as in space for time-dependent problems.

• Local refinement can be applied to any structured-grid data, such as bin-sorted particles.

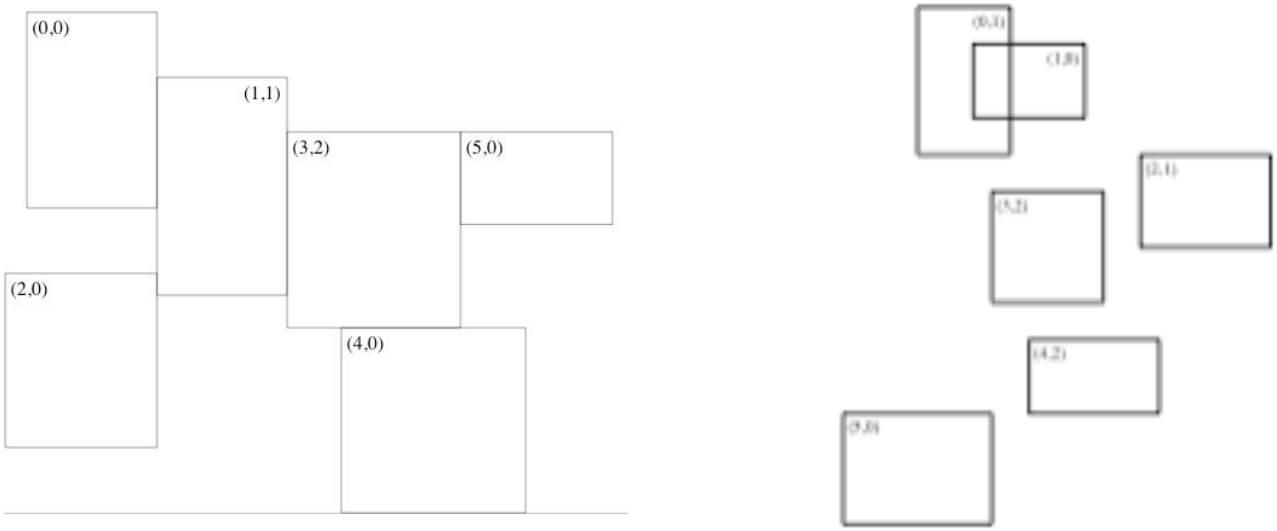




An AMR Software Framework

- The BoxLib / Chombo libraries support a wide variety of applications that use AMR by means of a common software framework.
  - Mixed-language programming: C++ for high-level abstractions, Fortran for calculations on rectangular patches.
  - Reuseable components, based on mapping of mathematical abstractions to classes. Components are assembled in different ways to implement different applications capabilities.
  - Layered architecture, that hides different levels of detail behind interfaces.
- In this approach, high performance is obtained by optimizing components, but in the context of specific applications usage patterns. Leads to demand-driven performance optimization.
- For the applications we are supporting, implicit methods (and hence solvers) are essential.

AMR Programming Model



- Domain decomposition that assigns rectangular patches to processors. All processors have access to processor assignment metadata. Distributed grid data built on top of these metadata.
- Local computation: iterate over patches owned by processor. Processor has access only to local data.
- Communication primitives: exchange of ghost cell data, copying from a disjoint union of rectangles to some other union of rectangles.
- Interlevel operations: interpolating boundary data, averaging / interpolation between levels combine communication and irregular computation.

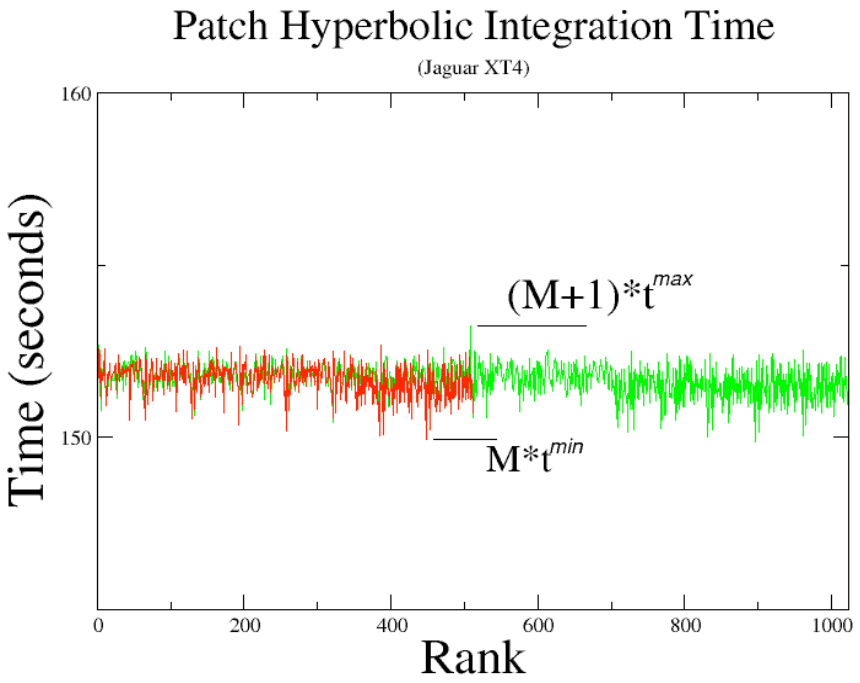
Performance / Scalability: Top Level Questions

- Does AMR scale ?
- 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 (Science / Megawatt-Hour)?
- How do we design a benchmark suite that encapsulates the usage patterns for a broad range of applications?

Understanding Scalability and Performance

Defining Scalability, Performance

- For many PDE applications, scalability ´ weak scalability.
  - Use 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 beyond that point is often lost due to realities of a multiuser environment.
  - Real-time applications, e.g. numerical weather prediction, are counterexamples. Even in 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, a bound on the execution time per patch independent of the number of processors.



$t^{min} \cdot t^{patch} \cdot t^{max}$

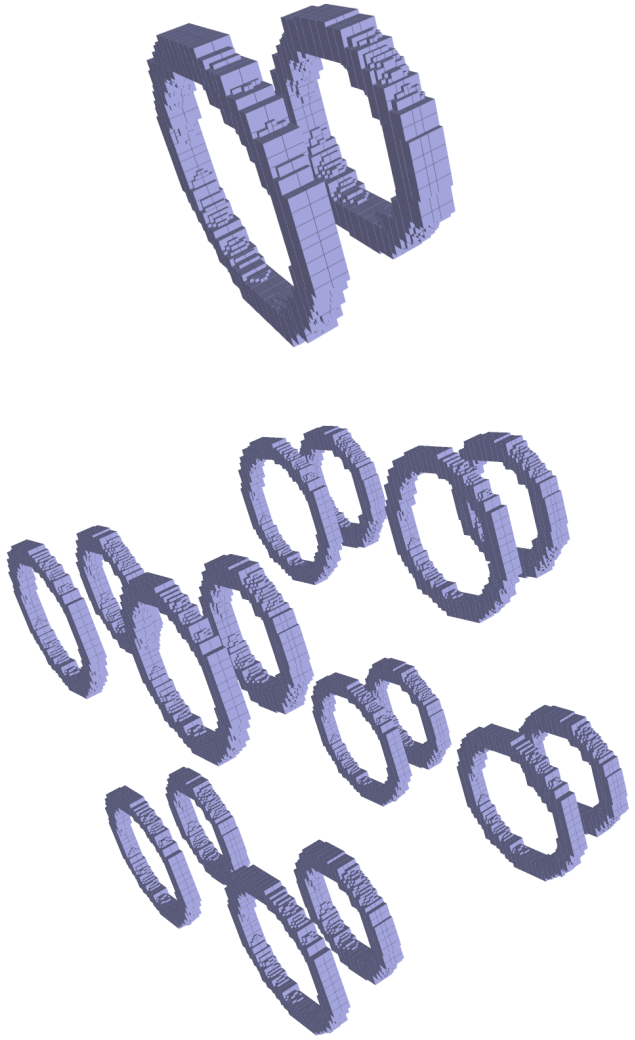
$M = N_{patches}/N_{procs}$

Defining Scalability, Performance

- Operator peak performance: maximum performance for evaluating an operator on a uniform grid on a single processor. For stencil operations, this has been typically 10%-20% of the nominal peak. We assess performance in terms of a fraction of operator peak performance.
- Adaptivity factor: ratio of the time to perform the calculation on a uniform grid at the finest resolution to the time to solution for the AMR calculation. The former is generally estimated from smaller runs and assuming perfect weak scaling, rather than computed directly.
- Implementation efficiency: what fraction of time is 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.

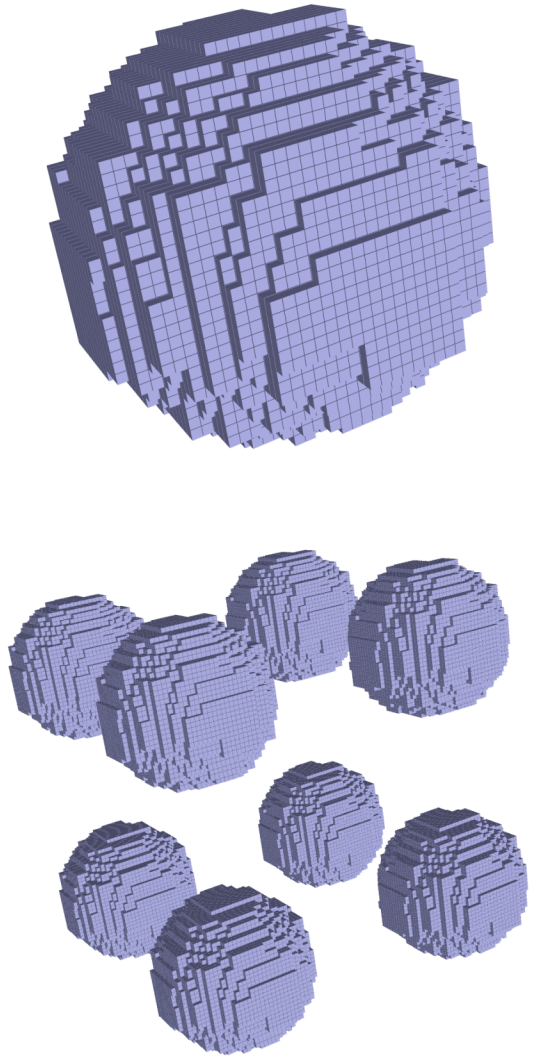
Replication Scaling Benchmarks

- Take a single grid hierarchy, and scale up the problem by making identical copies. Full AMR code (processor assignment, remaining problem setup) is done without knowledge of replication.
  - Good proxy for some kinds of applications scaleup.
  - Tests algorithmic weak scalability and overall performance.
  - Does not test load balancing.
  - Avoids problems with interpreting scalability of more conventional mesh refinement studies with AMR.
  - Variations: random offsets of images; replicate the set of cells tagged as needing refinement.



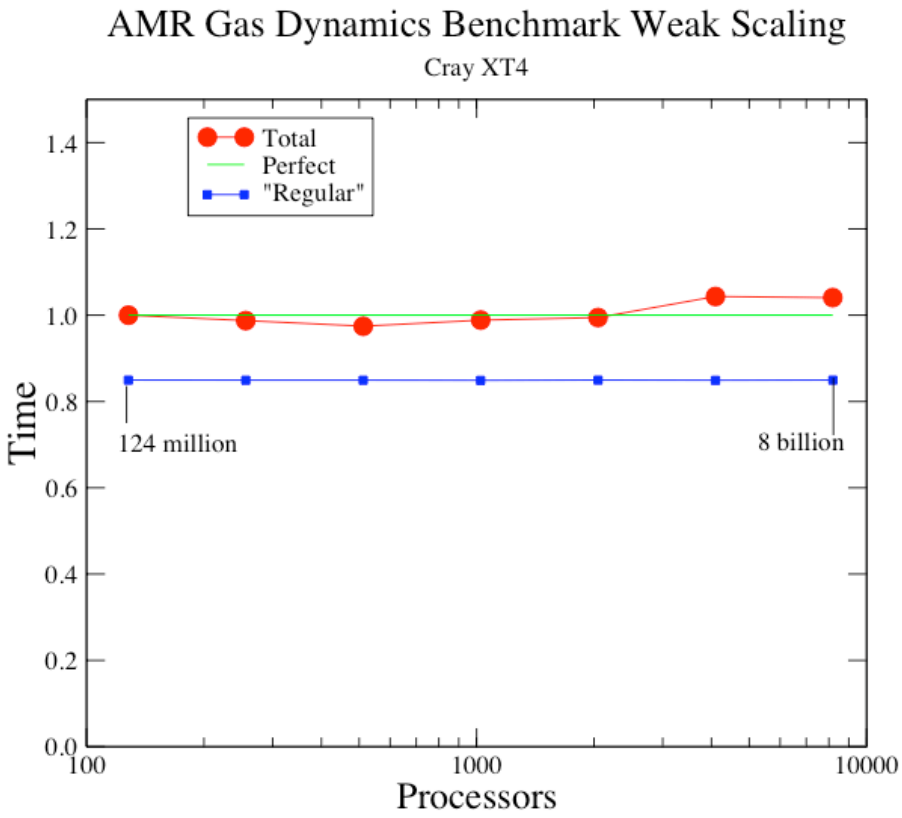
AMR Gas Dynamics Benchmark

- Unsplit PPM solver - 6K flops / grid point to update a cell. Explicit method, so ghost cell values copied / interpolated only once per update. Easiest case.
- Single image is a spherical shock tube in 3D, with finest grids covering a spherical shell.Two levels of refinement, factor of 4 each. Refinement in time proportional refinement in space. Fixed-sized (16x16x16) patches. Five unknowns / cell, 62M grid points, with1B grid point updates performed per coarse time step.
- Operator peak performance on XT4 is 530 Mflops / processor.
- Timing only the update step - no initialization, regridding, etc.
- Results obtained with hyperbolic code “out of the box” from Chombo distribution.



Gas Dynamics Benchmark:Results

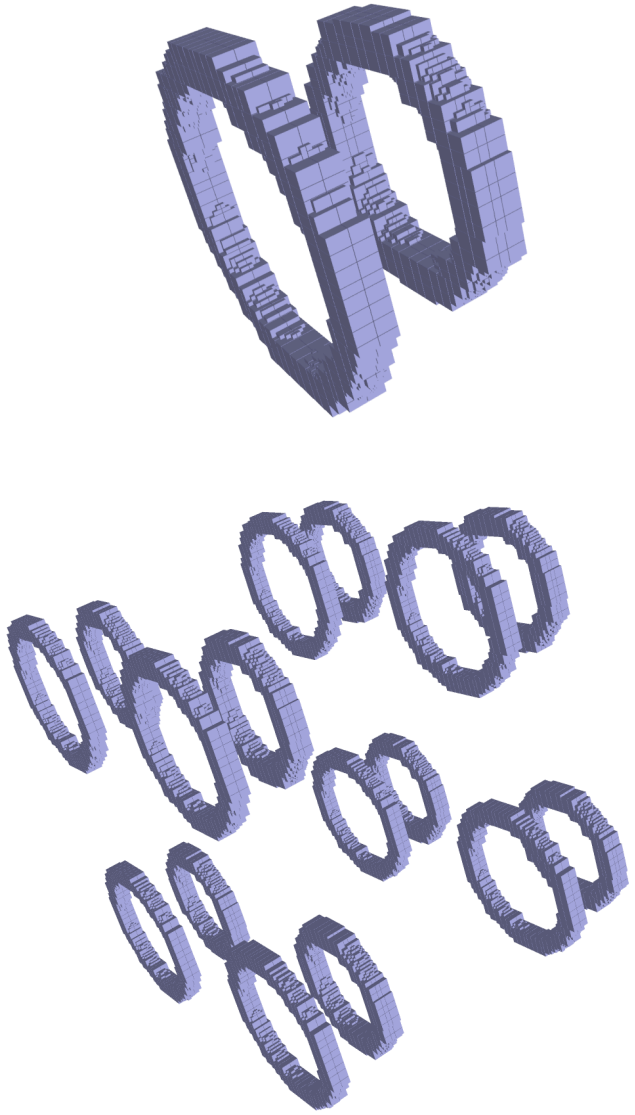
- 96% efficient scaled speedup over range of 128-8192 processors (173-181 seconds).
- Fraction of operator peak: 85% (450 Mflops / processor).
- Adaptivity factor: 16.
- Outstanding issues: small fluctuations in scaling, somewhat lower single-processor performance than expected.



Scaling of AMR for explicit methods is relatively easy.

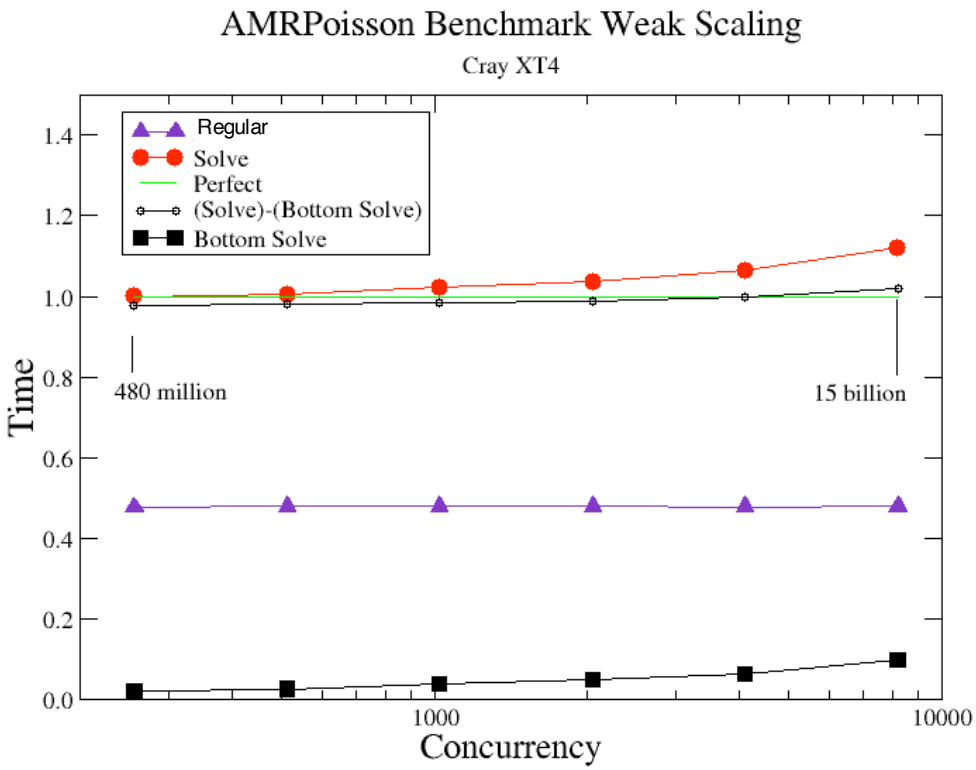
AMR Poisson Benchmark

- Multilevel discretization of Laplacian, with AMR multigrid algorithm used as solver. 10 iterations of AMR-MG V-cycle = 1700 Flops / grid point. Over 100 calls to communication (exchange / copyTo) per iteration. Typical of broad range of elliptic solvers on AMR grids.
- Single image is two rings. Two levels of refinement, factor of 4 each. Patch size is allowed to vary between 8^3 and 32^3. One unknown per cell, total of 15M grid points per image.
- Operator peak performance on XT4 is 840 Mflops / processor.
- Timing only the solver - no initialization.
- Results obtained after significant effort in code optimization (2 months), leading to 10X improvement in per-processor performance and in scalability.



Poisson Benchmark:Results

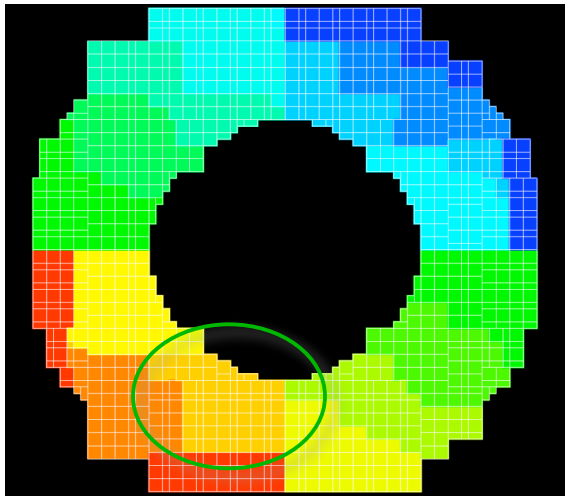
- 87% efficient scaled speedup over range of 256-8192 processors (8.4-9.5 seconds).
- Fraction of operator peak: 45% (375 Mflops / processor).
- Adaptivity factor: 48 (estimated).
- Principal bottleneck: level 0 BiCGStab. Level 0 grids are coarsened down to where domain is covered by a collection of 2x2x2 patches, each on a different processor. This problem is easy to fix.



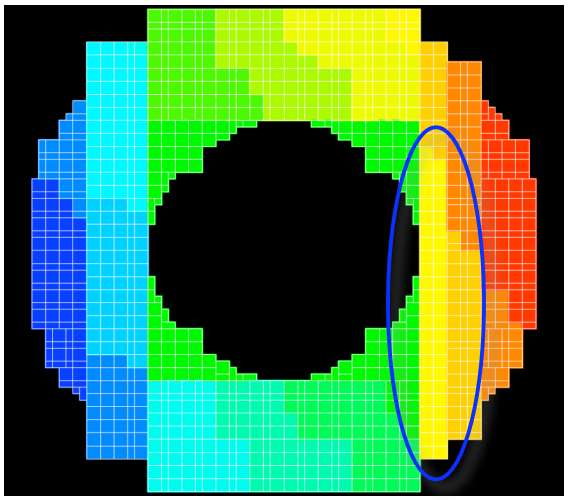
Development of scalable Poisson solvers is one of the most challenging goals for AMR. How did we accomplish this ?

Minimizing Communications Costs

- Distributing patches to processors to maximize locality. Sort the patches by Morton ordering, and divide into equal-sized intervals.
- Overlapping local copying and MPI communications in exchanging ghost-cell data (only has an impact at 4096, 8192).
- Exchanging ghost-cell data less frequently in point relaxation.



Morton-ordered load balancing (slice through 3D grids).



Berger-Rigoutsos + recursive bisection.

Metadata, Irregular Computations

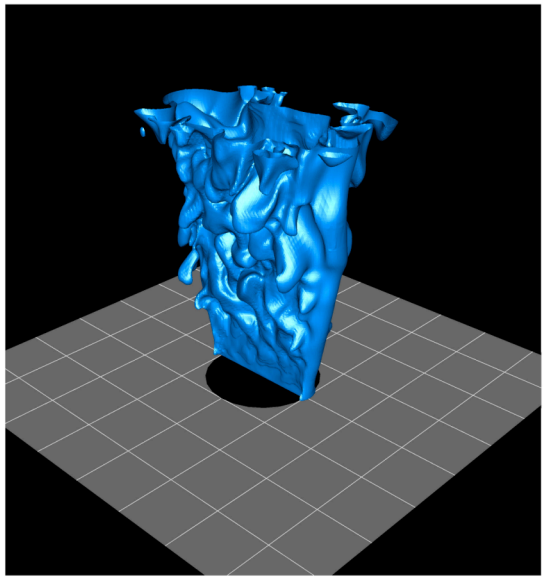
- Every processor has a copy of the metadata (assignment of patches to processors) for all unions of rectangles / processor assignments. These are used to compute intersection lists, e.g. from which patches ghost-cell data is copied.
  - Storing the metadata (not an issue except for the largest problems).
  - Fast sorts / searches to compute intersection lists - otherwise, catastrophic failure to scale due to $O(N_{patch})^2$ computations.
  - Caching intersection lists.

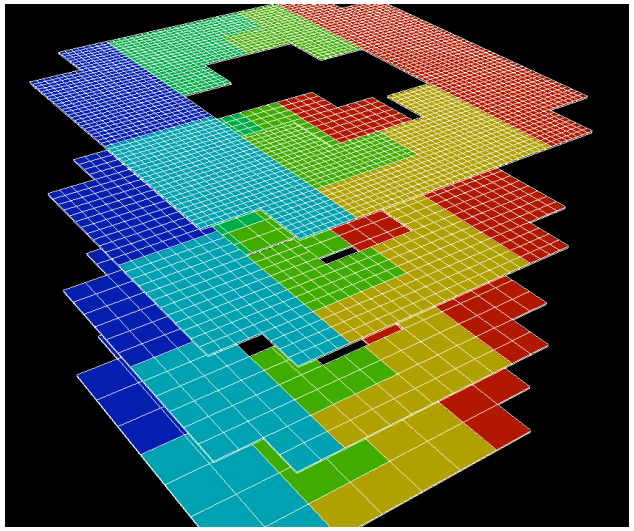
- Coarse-fine boundary conditions involve parallel communication and irregular computation.
  - Use of residual-correction form to minimize how often they are called.
  - For stencils that are actually regular, call Fortran. Could also use fixed-size patches to make such calculations regular, or develop fast irrefular stencil operations.
  - Cache stencil communications data (copiers).

Algorithm Choices: A Case Study

AMR Low-Mach-Number Combustion (LMC) Algorithm

- Discretization of the Ma ! 0 fluid equations with detailed hydrocarbon chemistry and transport. For methane (GRIMech 3.0, EGLib), 60 unknowns per grid point.
- Used to investigate a variety of turbulent flames. Extensions to simulate syngas combustion, nuclear burning in supernovae.
- Computational time dominated by solving ODEs at every grid point for chemistry using an implicit solver. Single-level variable-coefficient elliptic solvers to impose divergence constraint that replaces acoustic wave dynamics. The latter use multigrid-preconditioned BiCGStab.

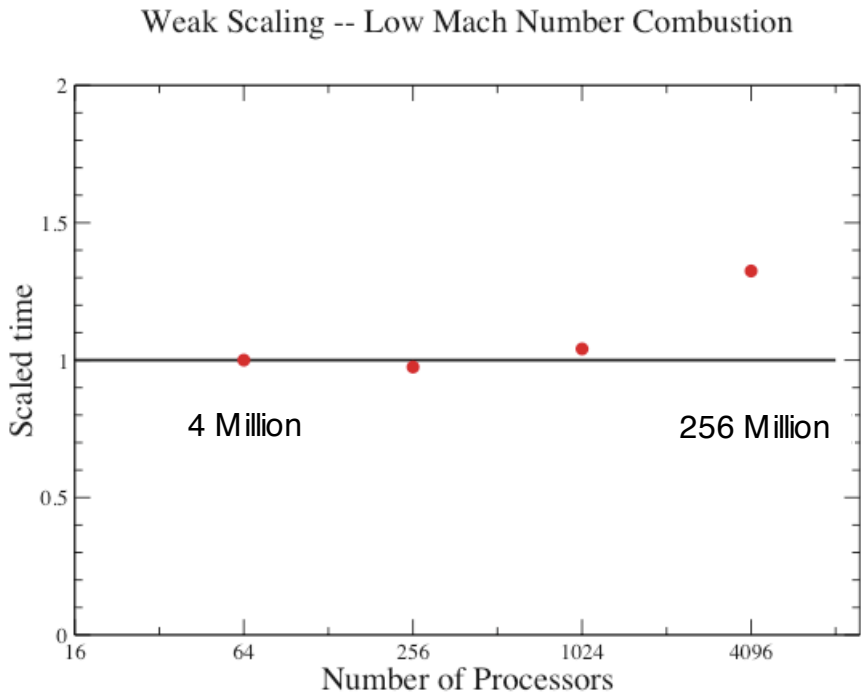




LMC Benchmark

• LMC Replication Benchmark: Single image is a wrinkled flame. Two levels of refinement, factor of 2 each, refinement in time. Total of 4M grid points.

- For $N_{proc} \cdot 1024$, 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.
- Variable-coefficient elliptic solvers used here are leading to a loss of weak scaling of the whole application for larger numbers of processors.
- Similar scaling results obtained on a Linux cluster, and on an SGI system (NASA Columbia).



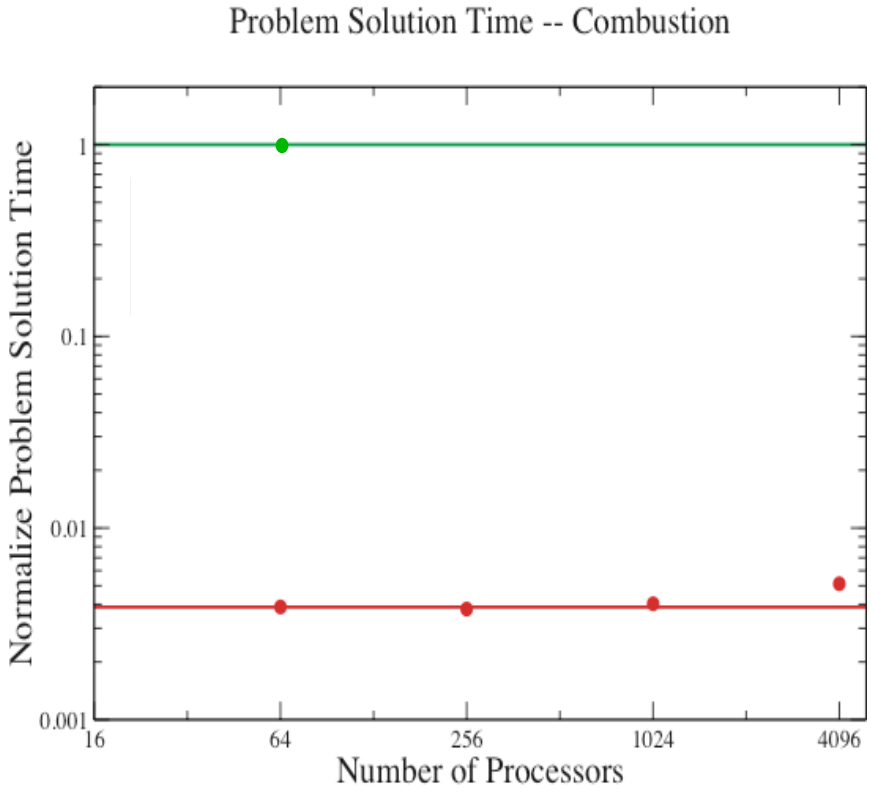
LMC vs. Fully Explicit Method

- Algorithmic choices are driven by science requirements: of the available alternatives, which is going to provide the most scientific output for the least cost ? To address this question, we compared LMC to a version of what has been the standard approach to solving these problems for the last 20 years.

- Fully explicit method for viscous compressible flow on a uniform grid:
  - Explicit stencil operations scale perfectly.
  - Time step is determined by CFL condition for acoustic waves (.02 μ sec).
  - For chemistry, use explicit ODE method, subcycle in time as needed.
- LMC:
  - Elimination of acoustic waves leads to a 50X increase in the time step (1 μsec). This comes at the cost of introducing elliptic solvers and the accompanying loss of ideal scaling.
  - AMR provides 10X reduction in the number of grid points over a uniform fine grid with the same resolution.
  - Chemistry ODEs integrated with an implicit solver.

LMC vs. Fully Explicit Method

- 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.
- Deviation from scalability is a miniscule effect relative to the difference between the approaches.



Green: fully explicit method.
Red: LMC.

Future Plans

Outstanding Issues at 10K Processors

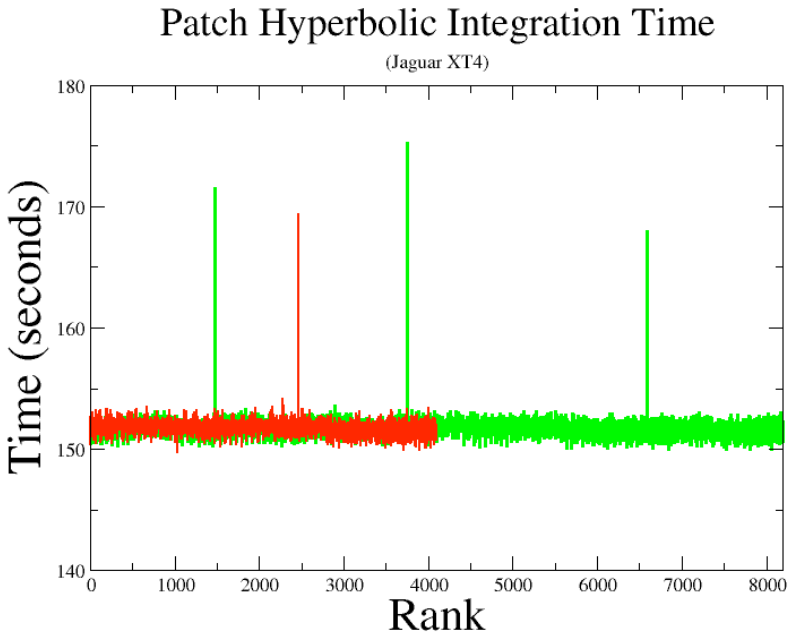
- Parallel grid generation (Gunney, et. Al., LLNL; fixed-size patches).
- Analysis tools for weak scaling load balance.
- Propagation of fast intersection calculations into entire library. Scalability of setup process.
- Scalability of level solvers for variable-coefficient elliptic problems.
- Design of solver performance test suite that spans the usage patterns for full range of applications:
  - Grid layouts (e.g. cosmology vs. combustion).
  - Choice of operators (variable coefficient, tensor, high-order).
  - Level solvers vs. multilevel hierarchies.
  - Metrics: those given above, plus memory usage, setup time.
  - Make benchmarks accessible to larger solver community.
- Complex geometries using embedded boundaries. Load balancing based on run-time measurements (already done in LMC).
- Anisotropic solvers / problems (S_n radiation, GFD, thermal conduction in MHD). Line solves **$** loss of parallelism in that direction.

Potential Issues at ¸ 100K Processors

- Good news: We still have plenty of parallelism left to exploit, particularly on multicore/manycore architectures.
- Representation of patch metadata. Current implementation uses ~50 Bytes / patch, which too large for 1M patches or more. Possible approaches include:
  - Minimum storage representation: bitmap (1 bit / patch), + one long-long per processor **!** 2 MB for 100K processors.
  - Distribute patch metadata over multiple processors. For manycore nodes, store one copy of the metadata per node.
- Parallel load balancing algorithm, based on bitmap representation. Exploit the recursive structure of Morton ordering to do divide-and-conquer.

Computing Environment at the Petascale

- Does MPI scale to a flat 100K processor space ?
- Production-quality implementations of hierarchical programming models for multicore/manycore systems.
- I/O. Visualization and analytics. Mass storage.
- Above 1000 processors, multiuser environments and other artifacts can lead to unpredictable fluctuations in performance.
- Adequate access to large numbers of processors for library development.



The spikes correspond to specific physical nodes and are due to correction of single-bit memory errors on those nodes (A. S. Bland, ORNL). After this problem was corrected, we obtained the scaling results given in the previous slides.

Conclusions

- AMR scales to 10K processors with per-grid-point performance comparable to the corresponding uniform-grid algorithms. We see no serious technical barriers specific to AMR to scaling to 100K processors and beyond.
- Due diligence and attention to details is necessary and sufficient for the development of scalable AMR software (provided the other resources are there).
- The development of such software components should be driven by the requirements of applications.
- An suitably-designed AMR framework is essential for performance tuning.

Developers and users need to be convinced that attempting to scale beyond 10K processors will yield real science returns in a timely fashion.