1 Cosmological Structure Formation (Michael Norman and Greg Bryan)

1.1 Problem to be Solved

The universe is homogeneous and isotropic on scales exceeding one billion light years, but on smaller scales it is clumpy, exhibiting a hierarchy of structures which includes individual galaxies, groups and clusters of galaxies, and superclusters of galaxies. Understanding the origin and cosmic evolution of these structures is the goal of *cosmological structure formation–CSF*. CSF is inherently nonlinear, multidimensional, and involves a broad range of physical processes operating on a range of length– and time–scales. Numerical simulation is the only means we have of studying it in any detail.

Simulations of CSF have grown in size and complexity as computer power has grown. The largest N-body CSF simulations of the day has increased from N=32^3 particles on VAXs in the mid '80s to 1024^3 particles on today's MPPs—an astounding factor of over 32,000. Today, CSF simulations are among the largest consumers of supercomputer cycles at the NSF centers, rivaling CFD, condensed matter physics, and lattice gauge theory.

Two parallel applications described here simulate CSF in three spatial dimensions and time within an expanding background spacetime consistent with our understanding of the Big Bang origin of the universe. The first code, called *Kronos* [4], uses a uniform Cartesian grid comoving with the expanding universe as the basis for discretizing the equations of matter and gravitational dynamics. The second code, called *Enzo* [5, 7, 14], adds structured adaptive mesh refinement (SAMR) to the Kronos algorithm for improved spatial and temporal resolution in high density regions (galaxies, clusters, etc.) Sequential and parallel versions of both codes have been developed and optimized for vector multiprocessors, SMPs, MPPs, and clusters of PCs and SMPs. The message–passing parallel version Enzo, which can be run with and without mesh refinements, is our computational workhorse and is the main focus of this report.

1.2 Computational Issues

Matter in the universe is of two basic types: ordinary "baryonic" matter composed of nucleons and electrons out of which stars and galaxies are made, and non-baryonic "dark" matter of unknown composition, which is nevertheless known to be the dominant mass consituent in the universe on scales of galaxies and larger. Kronos and Enzo self-consistently simulate both components, which evolve according to different physical laws and therefore require different numerical algorithms.

Baryonic matter is evolved using a finite volume discretization of the Euler equations of gas dynamics cast in the comoving frame including energy source and sink terms due radiative heating and cooling processes, as well as changes in ionization state of the gas [4]. In some calculations involving nonequilibrium chemistry, separate chemical/ionic species are evolved by solving their kinetic rate equations [2]. Radiation fields are modeled as evolving but spatially homogeneous backgrounds; true radiative transfer is not yet included but is on the horizon [1].

Dark matter is assumed to behave as a collisionless phase fluid, obeying the Vlasov-Poisson equation. Its evolution is solved using particle-mesh algorithms for collisionless N-body dynamics [11]. Dark matter and baryonic matter interact only through their self-consistent gravitational field. The gravitational potential is computed by solving the Poisson equation on the uniform or adaptive grid hierarchy using Fourier transform techniques. In generic terms, our CSF codes are 3-D hybrid codes consisting of a multi-species hydrodynamic solver for the baryons coupled to a particle-mesh solver for the dark matter via a Poisson solver.

Matter evolution is computed in a cubic domain of length $L = a(t)X$, where X is the domain size in comoving coordinates, and $a(t)$ is the homogenous and isotropic scale factor of the universe which is an analytic or numerical solution of the Friedmann equation, a first order ODE. For sufficiently large L compared to the structures of interest, any chunk of the universe is statistically equivalent to any other, justifying the use of periodic boundary conditions. The speed of Fast Fourier Transform algorithms and the fact that they are ideally suited to periodic problems make them the Poisson solver of choice given the large grids employed—512^3 or larger.

CSF simulations require very large grids and particle numbers due to two competing demands: large boxes are needed for a fair statistical sample of the universe; and high mass and spatial resolution are needed to adequately resolve the scale lengths of the structures which form. For example, in order to adequately simulate the internal structure of galaxies and simultanenously describe their large scale distribution in space (large scale structure), a dynamic range of 10^4 per spatial dimension and 10^9 in mass is needed *at a minimum.*

The largest uniform grid simulation ever done including gas and dark matter is a Kronos simulation we carried out on 512 processors of the Con-

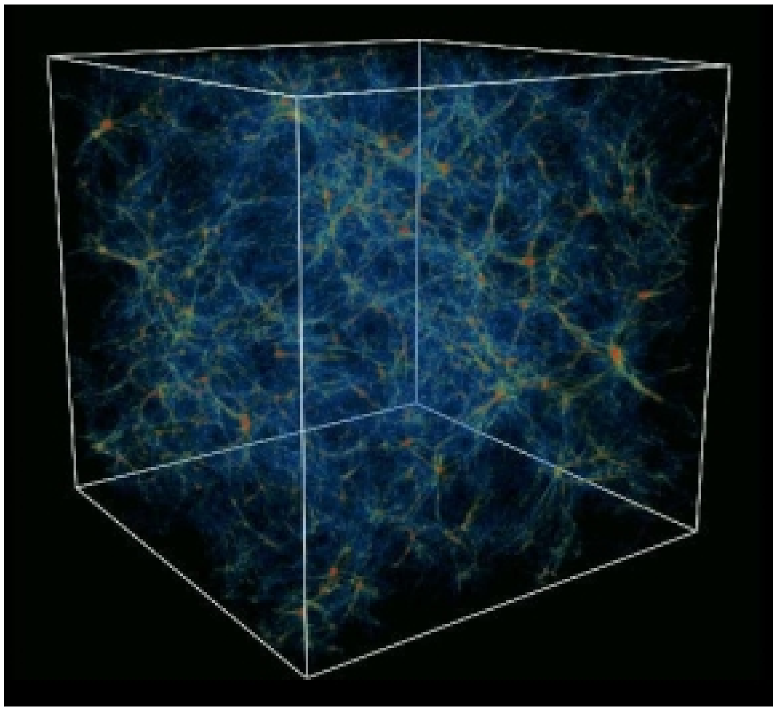

Figure 1: Kronos simulation of large scale cosmological structure. Shown is the distribution of gas density in a volume half a billion light years on a side. From [6].

nection Machine-5 at NCSA in 1994 (Figure 1). The simulation used a grid of 512^3 cells and 5×10^7 particles–far short of the requirements mentioned above. With the use of the adaptive mesh refinement code Enzo on the current generation of terascale computing systems, the desired resolutions are now achievable. In the next two sections, we discuss parallel computing aspects of these two codes.

1.3 Parallel Unigrid Code: Kronos

The Kronos code was developed from 1992-1994 by Greg Bryan for the Connection Machine-5 at the NCSA. The CM-5 had 512 processor nodes, each consisting of a SUN Sparc microprocessor, four vector processors, and 32 MB of memory. The theoretical peak speed of the system was quoted as 0.128 GFlop/sec/PN $\times$ 512 PN = 65 GFlop/sec, and the total memory was

16 GB.

Kronos was implemented in the data parallel Connection Machine Fortran (CMF) programming model. Conceptually, Kronos is the union of two codes: a 3-d Eulerian gas dynamics code (suitably modified for cosmology [4]), and a 3-d particle-mesh code (of which the FFT-based Poisson solver is a component) for the collisionless dark matter. The parallel challenges and solutions for each code are quite different, and so we discuss them individually.

The equations of gas dynamics are purely local: changes in cell quantities due to pressure forces and fluid advection involve only nearest neighbors. By assigning one virtual processor per cell in a 3-d cartesian lattice, nearest neighbor information was passed using the CM-5 NEWS data communication network via simple CSHIFT calls. This was the basis of our first implementation. Performance tests measured at $\approx$ 8 MFlops/sec/PN, or about 6% of peak. The reason for this poor performance was that the communication network was invoked between every computational cell regardless of whether they resided on the same physical processor or not.

In order to circumvent this, our second implementation abandoned the one virtual processor per cell model in favor of explicit domain decomposition. This was accomplished within the CMF data parallel programming model by declaring 6-d arrays for the fluid field variables; e.g., `d(:serial,:serial,:serial,:news,:ne`
the serial dimensions referring to the 3-d index of a cell within a given block, and the parallel dimensions referring to the indices of the block in a 3-d block decomposition of the computational domain. This had the advantage that serial operations on `d` within a block could proceed in parallel without invoking the communication network. Internal boundary values were copied from neighboring processors once per timestep into 5-d arrays which corresponded to the faces of the blocks. In this way, communication was isolated to one rather minor phase of the calculation. Performance improved threefold to $\approx$ 24 MFlop/sec or 18% of peak, which largely reflected the sustained speed of the purely local computations. Scaling tests with constant work per processor yielded ideal scaling up to NP=512 nodes, confirming that communication costs were minimal.

The particle-mesh code, on the other hand, is communication intensive. The PM algorithm consists of three phases, the first and third of which involve nonlocal communication between and among the 1-d particle list and 3-d field arrays. In the first *mass assignment* phase, the particles' mass is assigned to a density field array via a gather operation. In the second *field solve* phase, the Poisson equation is solved for the gravitational potential using 3-d FFTs—a nonlocal operation. The mesh force is computed from spatial diffences of the potential—a local operation. In the third *force interpolation* phase, the mesh force is interpolated to the particle positions via a scatter operation. Obviously, finding efficient parallel implementations which minimize communication costs is essential. An additional complication is that the particle distribution becomes highly inhomogeneous due to gravitational clustering, creating load imbalances in phases 1 and 3 even if the particle list and field arrays are uniformly distributed across processors.

We implemented the algorithm of Ferrell and Bertschinger [9] which elegantly solves all of these problems. Since the algorithm and its performance on the CM-5 are described in detail in [9], we merely summarize the key points. The gather-scatter portion of phases 1 and 3 are done in a completely load balanced way through the use of *Parallel Prefix Operations* on the particle list [10]. Parallel prefix operations, also referred to as *Scans*, are a method of turning certain kinds of global communications into regular, mostly local, communications. Briefly, the procedure is to sort the particle list so that all particles within a given processor are contiguous. An index list is introduced that contains the processor ID for each particle. Because the list has been sorted, the processor ID is constant in a segment, changing to another value in the next segment. We then use a *Segmented Scan Add* operation which computes a running sum of the masses of the particles within a given segment. This operation requires $O(\text{logNP})$ communication operations. The last element in each segment contains the total mass in the segment. We then have only one word of data to send to each virtual processor assigned to a grid cell. In step 2, three components of the gravitational acceleration on the grid is computed from the gridded mass densities using Fourier Transforms. For this purpose, we used the highly optimized 3-d FFT routines in the CMSSL library. The third force interpolation phase is essentially the inverse of the mass assignment phase. We use a *Segmented Scan Copy* to copy the gridded forces to a segmented force list. The operation also takes $O(\text{logNP})$ communication operations. The forces are then applied to the particles in parallel in a purely local fashion.

For a scaled work problem, the combined code exhibited linear speedup on the CM-5 to 512 processors, with a parallel efficiency of $T(1)/(NP * T(NP)) \sim 0.75$. Clearly, the communication overhead in the PM portion of the calculation is responsible for the lack of ideal scaling. Still, the fact that parallel speedup was roughly constant versus NP indicates that the combined algorithm was scalable.

1.4 Parallel AMR Code: Enzo

The demise of the CM-5 coupled with the need for higher resolution than afforded by uniform grids motivated the development of Enzo. Enzo uses structured adaptive mesh refinement (SAMR, [3, 5]) to achieve high resolution in gravitational condensations. The central idea behind SAMR is simple to describe but difficult to implement efficiently on parallel computers. While solving the desired set of equations on a coarse uniform grid, monitor the quality of the solution; when necessary, add an additional, finer mesh over the region that requires enhanced resolution. This finer (child) mesh obtains its boundary conditions from the coarser (parent) grid or from other neighboring (sibling) grids with the same mesh spacing. The finer grid is also used to improve the solution on its parent. As the evolution continues, it may be necessary to move, resize or even remove the finer mesh. Even finer meshes may be required, producing a tree structure that can continue to any depth.

To advance our system of coupled equations in time on this grid hierarchy, we use a recursive algorithm. For simplicity, we consider only the hydrodynamic portion of the algorithm; the dark matter dynamics and Poisson equation have a similar structure. The `EvolveLevel` routine is passed the level of the hierarchy it is to work on and the new time. Its job is to march the grids on that level from the old time to the new time:

```
EvolveLevel(level, ParentTime)
begin
  SetBoundaryValues(all grids)
  while (Time < ParentTime)
  begin
    dt = ComputeTimeStep(all grids)
    SolveHydroEquations(all grids, dt)
    Time += dt
    SetBoundaryValues(all grids)
    EvolveLevel(level+1, Time)
    RebuildHierarchy(level+1)
  end
end
```

Inside the loop which advances the grids on this level, there is a recursive call so that all the levels above (with finer subgrids) are advanced as well. The resulting order of timesteps is like the multigrid W cycle.

As with any hyperbolic equation, we must set the boundary conditions on the grids. This is done by first interpolating from a grid's parent and thing copying from sibling grids, where available. Once the boundary values have been set, we evolve the hydrodynamic field equations using procedure `SolveHydroEquations`. The final task of the `EvolveLevel` routine is to modify the grid hierarchy to the changing solution. This is accomplished via the `RebuildHierarchy` procedure, which takes a level as an argument and modifies the grids on that level and all higher levels. This involves three steps: First, a refinement test is applied to the parent grids of the current level to determine which cells need to be refined. Second, rectangular regions are chosen which cover all of the refined regions, while attempting to minimize the number of unnecessarily refined points. Third, the new grids are created and their values are copied from the old grids (which are deleted) or interpolated from parent grids. This process is repeated on the next refined level until the grid hierarchy has been entirely rebuilt.

1.5 Parallelization of Enzo

Other than the physical equations solved, Enzo bears no relation to Kronos. Virtually none of the CMF code was reusable because not only did we change algorithms, we changed programming models and languages as well. The code is mostly implemented in C++, with compute-intensive kernels in FORTRAN 77. Efficiently parallelizing SAMR is difficult, particularly for distributed memory systems. Grids have a relatively short life, so information must be updated frequently. Moreover, load balancing becomes crucial since small regions of the original grid eventually dominate the computational requirements.

Enzo development proceeded in two major steps. The first step, carried out by Greg Bryan from 1994-1996, was the implementation of a shared memory parallel code for the SGI Origin2000 employing SGI's PowerC compiler to concurrently execute grids at a given refinement level. The powerful, mature C development environment on the SGI was a major boon. However, since the workload is typically distributed nonuniformly across levels (cf. Fig. 3), and the algorithm dictates that levels must be processed sequentially, we found we could not efficiently use more than about 16 processors. Therefore, a second SPMD message-passing code for distributed memory systems was implemented from 1997-2000 wherein the root grid is domain decomposed into 3-d blocks. Each block plus its complement of subgrids are assigned to different processors, which work on them in parallel. Load balancing is achieved by sending grids from overloaded processors to underloaded ones, and optionally through the use of *grid splitting* [13].

We have used the MPI library to produce a code which is portable and efficient. In particular, we have used the following optimization techniques:

- *Distributed objects.* We leveraged the object-oriented design by distributing the objects over the processors, rather than attempting to distribute an individual grid.
- *Sterile objects.* Although distributing the objects results in good load balancing, it has the potential to greatly increase the amount of communication since each processor has to probe other processors to find out about neighboring grids. We solved this problem by creating a type of object which contained information about the location and size of a grid, but did not contain the actual solution arrays. These sterile objects are small and so each processor can hold the entire hierarchy. Only those grids which are truly local to that processor are non-sterile.
- *Pipelined communications.* One result of distribution is that all operations between two grids (e.g. obtaining boundary values) are potentially non-local. We optimize this by dividing each communication stage into two steps. First, all of the data are processed and sent. Since all processors have the location of all grids locally (thanks to the sterile objects), we can order these sends such that the data that are required first are sent first. Then, in the receive stage, the data needed immediately has had a chance to propagate across the network while the rest of the sends were initiated.

1.6 Performance

The performance of an AMR application is difficult to characterize because the workload and its distribution are dynamically changing throughout the calculation. The simplest measure is *time to solution* of a run versus NP. This necessitates running a job to completion over and over again, varying NP. This is computationally expensive for modest problem sizes and impractical for medium to large problems of interest. Nevertheless, this has been done; results are reported in [12]. We find that not only is parallel efficiency problem *size dependent*, as expected, but also *problem dependent* as well. For example, a survey calculation involving a large root grid and no subgrids

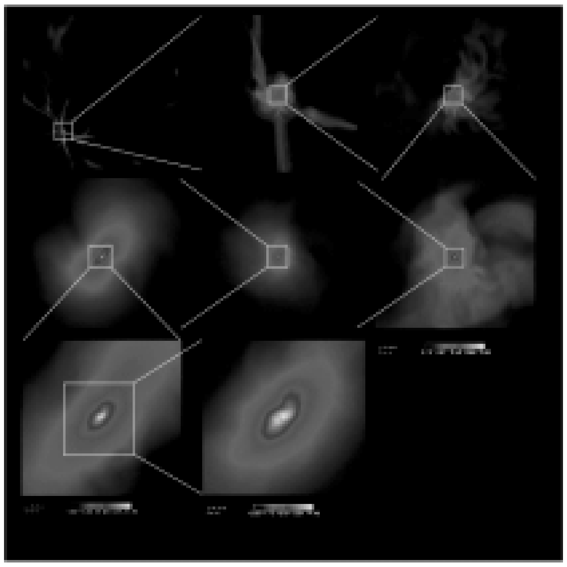

Figure 2: Enzo simulation of primordial star formation. Each image shows gas density in a region ten times smaller than the previous. From [8].

distributed over many processors will scale very differently than a calculation involving a large number of small, deeply nested subgrids focussing on a single collapsing object.

To illustrate the operation and performance of Enzo on the latter sort of problem, we show in Figure 2 an AMR simulation of primordial star formation which achieves a local resolution in space and time of 10^{12}. For comparison, 10^{12} is roughly the ratio of the diameter of the earth to the size of a human cell. Temporally, 10^{12} is roughly the ratio of time since the extinction of the dinosaurs to when you woke up this morning. Over 8000 subgrids are developed at 34 levels of refinement to achieve this unprecedented dynamic range.

In the top two panels of Figure 3, we show how the grid hierarchy grows as time progresses. Note the slow increase in the number of grids as the proto-star condenses and the final, very sudden jump in the depth of the grid tree at the end when the core of the cloud collapses to high density. This demonstrates how the data structures themselves adapt to fit the physical solution. Note also the extremely large number of memory allocations and frees, since the entire grid hierarchy is rebuilt thousands of times. This kind

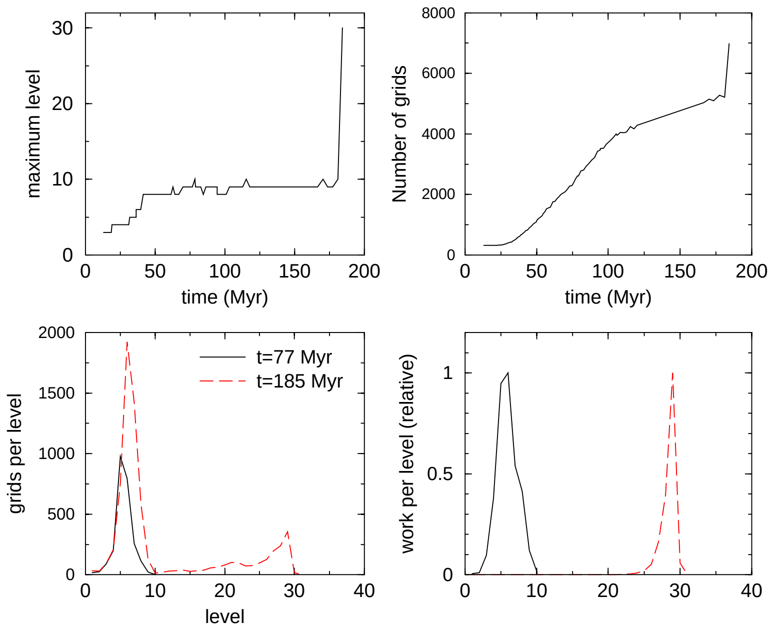

Figure 3: The top left and right panels show the depth of the hierarchy tree and the number of grids as a function of time (in millions of years), The bottom left and right panels plot the number of grids per level and an estimate of the computational work required per level (in each case normalized so that the maximum value is unity).

of method represents a new class of scientific computing that place great strain on the operating system infrastructure. Total memory usage is also substantial, often reaching up to 20 GB. With outputs in the 2-4 GB range, we require at least 50-100 GB disk storage and much more mass storage space.

In the bottom two panels of this figure, we have chosen two representative times and plotted the distribution of levels per grid. At early times, most of the grids are at moderate levels, representing the fact that relatively low resolution is sufficient to model the protostar. However, at late times, a large investment is required at the very highest levels of resolution.

Finally, we estimate the raw performance of the code in the following way. We have used the hardware floating-point counter on the SGI Origin2000 to determine the speed of a similar SAMR calculation. This provides a benchmark from which we can determine the speed of this calculation, which was run on the Blue Horizon IBM SP2 system at the San Diego Supercomputer Center. Running on 64 processors produced a speed approximately 125 times faster than a single Origin2000 processor (105 MFlop/s), yielding a total speed of approximately 13 Gflop/s. As an exercise, we can also ask how long this calculation would have taken with a traditional static grid code and compute an effective or virtual flop rate. To do this, we assume a grid with 10^{12} cells on each side, and assume the entire calculation would have taken (quite conservatively) 10^{10} timesteps. This works out to approximately 10^{50} floating point operations. Since the entire calculation took of order 10^6 seconds, this converts to a virtual flop rate of 10^{44} flop/s.

1.7 Future

In the near future we intend to carry out large scale simulations of galaxy formation resolving the internal structure of thousands of galaxies simultaneously. These will involve large global root grids (512^3 or larger) and deep mesh refinements around each forming galaxy. Computational requirements are in the sustained teraflop range, owing to the large number of timesteps required, with concommitantly large RAM and disk requirements. Currently, we are porting Enzo to terascale cluster architectures including the Compaq system at PSC, as well as Linux clusters at NCSA. Principal needs remain mature C and Fortran compilers, debugging tools, optimized mathematical subroutine libraries, and efficient parallel I/O subsystems. We plan to explore mixed mode parallel programming (threads + message passing) on the IBM SP2 with Power3 SMP nodes at the SDSC. Our experience with the CM-5 has taught us the hard way that language solutions to massive parallelism vanish as quickly as the hardware they rode in on.

Acknowlegements: This work was carried out under the auspices of the Grand Challenge Cosmology Consortium with partial support by NSF grants ASC-9318185 and AST-9803137, as well as support from the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign which is gratefully acknowledged.

References

[1] T. Abel, M. Norman & P. Madau, “Photon-Conserving Radiative Transfer Around Point Sources in Multidimensional Numerical Cosmology”, Astrophys. J., **523**, pp. 66-71, (1999).

[2] P. Anninos, Y. Zhang, T. Abel & M. Norman, "Cosmological Hydrodynamics with Multi-species Chemistry and Nonequilibrium Ionization and Cooling", New Astron. **2**, pp. 209-224, (1997).

[3] M.J. Berger and P. Colella, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," J. Comput. Phys., **82**, pp. 64-84, (1989).

[4] G.L. Bryan, M.L. Norman, J.M. Stone, R. Cen, and J.P. Ostriker, "A Piecewise Parabolic Method for Cosmological Hydrodynamics", Computer Physics Communication, **89**, pp. 149-168, (1995).

[5] G.L. Bryan and M.L. Norman, "A Hybrid AMR Application for Cosmology and Astrophysics," in IMA Volume 117 on *Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, eds. S.B. Baden, N.P. Chrisochoides, D. Gannon, and M.L. Norman, (Springer: New York), pp. 165-170 (2000).

[6] G. L. Bryan and M. L. Norman, "Statistical Properties of X-Ray Clusters: Analytic and Numerical Comparisons", Astrophys. J., **495**, pp. 80-90, (1998).

[7] G.L. Bryan "Fluids in the Universe: Adaptive Mesh Refinement in Cosmology", *Computing in Science and Engineering*, **1:2**, 46 (1999).

[8] G. Bryan, T. Abel & M. Norman, "Achieving Extreme Resolution in Numerical Cosmology Using Adaptive Mesh Refinement: Resolving Primordial Star Formation", submitted to *Proceedings of Supercomputing 2001*, `www.supercomp.org`

[9] R. Ferrell & E. Bertschinger, "Particle-Mesh Methods on the Connection Machine", Int. J. Mod. Phys. C **5**, pp. 933-956 (1994).

[10] W. D. Hillis & G. L. Steele, Jr., "Data Parallel Algorithms", Commun. ACM **29**, 12 (1986).

[11] R. W. Hockney & J. W. Eastwood, *Computer Simulation using Particles*, (McGraw-Hill, New York, 1988).

[12] Z. Lan, V. Taylor & G. Bryan, "Characterization of a Parallel Adaptive Mesh Refinement Application", in *Proceedings of Supercomputing2000*, `www.supercomp.org`.

[13] Z. Lan, V. Taylor & G. Bryan, "Dynamic Load Balancing for Adaptive Mesh Refinement Applications", in *Proceedings of the International Parallel and Distributed Processing Symposium 2001.*

[14] M.L. Norman and G.L. Bryan, "Cosmological Adaptive Mesh Refinement," in *Numerical Astrophysics*, eds. S. Miyama & K. Tomisaka, (Kluwer: Dordrecht, 1999), pp. 19-28.