HYDRA-MPI: An Adaptive Particle-Particle, Particle-Mesh code for conducting Cosmological Simulations on MPP Architectures

Robert Thacker[a], Gavin Pringle[b], H. M. P. Couchman[a], Stephen Booth[b]

[a]Department of Physics and Astronomy, McMaster University, 1280 Main St. West, Hamilton, Ontario, L8S 4M1, Canada
thacker@physics.mcmaster.ca

[b]Edinburgh Parallel Computing Center, Edinburgh University, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK

We discuss the design and implementation of a parallel adaptive P^{3}M code, for massively parallel architectures. The code, although primarily designed for conducting gravitational simulations in a cosmological context, also includes a Smoothed Particle Hydrodynamics solver. The code is written in a combination of C, FORTRAN 77 and FORTRAN 90. The MPI-2 remote memory access standard is used for one-sided message passing thereby enabling more time to be spent in raw computation. The design decision has lead to a number of interesting problems and circumstances, which we discuss in detail. Raw performance figures are presented in terms of numbers of particles per second to compare to alternative codes.

Nous discutons de la conception et de l'implémentation d'un code P^{3}M parallèle et adaptif pour les architectures massivement parallèles. Le code, quoique conçu principalement pour les simulations gravitationnelles dans le contexte de la cosmologie, comprend aussi un code hydrodynamique basé sur des particules. Le code est écrit en C, FORTRAN 77 et FORTRAN 90. Le standard MPI-2 est utilisé pour le passage unidirectionnel des messages, ce qui permet de laisser plus de temps aux calculs proprement dits. La plan adopté a mené à certains problèmes et circonstances, que nous discutons en détail. Les performances sont présentées en fonction du nombre de particules par seconde, dans le but de comparer à d'autres codes existants.

1 Introduction

The current generation of surveys of the Cosmic Microwave Background (CMB) radiation (*e.g.* MAXIMA [1]) and large scale structure (*e.g.* DEEP2 [2]) are providing an unpredented amount of highly accurate data about the Universe: the era of 'Precision Cosmology' has truely begun. While solving many long standing questions, the new data pose new and tougher challenges. The enormous volumes surveyed by modern galaxy surveys require highly accurate simulated 'mock surveys' to help understand both theory and error analysis. Accuracy and sampling errors are reduced by simulating as large a volume as possible with high resolution. The combined requirements of large volumes and high resolution necessitates the use of billions of mass elements in the simulation, which in turn requires massively parallel computing.

Observations of the CMB and supernova distance surveys [3] indicate that the evolution of structure in the Universe is dominated by two competing phenomena, the repulsion of the vacuum energy of universe Einstein's 'Cosmological Constant' and an unseen, massive component, 'Dark Matter'. The cosmological constant acts globally and hence the growth of structure on cosmological scales is dominated by the evolution of gravitational perturbations. Collapse of these perturbations proceeds rapidly, with cosmic expansion acting as a drag term. The resulting collapsed objects, such as galaxies, are many thousands of times denser than their surroundings. Modeling such large density contrasts, and the interactions that occur when structures merge to form larger ones, requires the use of Lagrangian 'particle' descriptions. Hydrodynamics can be included in particle-based simulations by using Smoothed Particle Hydrodynamics (SPH,[4]).

Particle-based solvers of interest in cosmology can be broadly divided into mesh-based solvers such as particle-particle, particle-mesh (P^{3}M, [5]) and "Tree" methods which rely an approximation method to evaluate the force for distant particles [6]. The adaptive P^{3}M algorithm [8] is an advancement of the P^{3}M algorithm designed to cope with highly inhomogeneous particle distributions. Special purpose hardware such as GRAPE [7] has rendered the direct PP method competitive in small simulations (less than 16 million particles), but it remains unlikely that it will ever be used for larger simulations. For conducting larger scale simulations the P^{3}M algorithm has proven very popular, with the first billion particles simulations being conducted by the Virgo Consortium in 1998 [9].

The design of our code has been heavily influenced by our research on the publically available serial [10] and earlier parallel versions of the adaptive P^{3}M code. We first developed a parallel implementation [11] on the Cray T3D using the CRAFT API which greatly aided development by providing a global address space. Shortly after, we developed a highly efficient implementation of the code in OpenMP which we currently use for a large body of our research [12]. On distributed memory systems the dynamic nature of the data movement in the algorithm necessitates high performance message passing to be efficient. Initially we chose to use Cray SHMEM for message passing [13] but we have since converted to the more portable remote memory access facilities provided by MPI-2[14].

The layout of this paper is as follows: we first review the physical system being studied and the AP^{3}M algorithm. This is followed by a discussion of our parallelization scheme and performance figures. Finally we comment upon the status of MPI-2 RMA support and conclude with a brief summary.

2 The Adaptive P^{3}M Algorithm

Given a set of initial conditions, which are usually constrained by observational data (*e.g.* the MAXIMA data-set) we must solve the following gravito-hydrodynamic equations,

1. the continuity equation,
$$\frac{d\rho}{dt} + \rho \nabla . \mathbf{v} = 0, \tag{1}$$
2. the Euler equation (equation of motion),
$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho} \nabla P - \nabla \phi, \tag{2}$$
3. the Poisson equation,
$$\nabla^2 \phi = 4\pi G \rho, \tag{3}$$
4. the internal energy equation,
$$\frac{du}{dt} = -\frac{P}{\rho} \nabla . \mathbf{v}, \tag{4}$$

with the dynamical system being closed by the equation of state $P = P(\rho, s)$. Dissipation, viscosity and thermal conductivity are usually ignored (*i.e.* an ideal fluid) with the only source entropy coming from an artificial viscosity.

The purpose of the adaptive P^{3}M algorithm is to solve Poisson's equation. Euler's equation and the energy equation are solved using SPH. The continuity equation is recovered immediately by using a particle based method.

The starting point of the P^{3}M algorithm is the idea that the gravitational force can be separated into short and long range components. The long-range component can be efficiently calculated by a Fourier-based solver and the short-range component can be calculated by performing a direct sum over particles within a given short-range radius. The long-range component is usually termed the 'PM force', for Particle-Mesh, while the short-range force is termed the 'PP force', for Particle-Particle.

Calculating the gravitational force on a mesh requires solving Poisson's equation. To do this a mesh representation of the particle masses must first be made, which is done using interpolation kernels (often called 'assignment functions'). We use a 27-point quadratic function, the 'Triangular Shaped Cloud', which strongly suppresses aliasing of frequencies from above the Nyquist frequency. The gravitational potential can be found from the density mesh simply by conducting a Fourier convolution with the Green's function[1] $1/k^2$. Once the potential has been recovered the gravitational force can be found by numerically differencing the mesh, and we use a 10-pt stencil. Finally, the acceleration on a particle is determined by interpolating from the force mesh using the inverse procedure to the initial mass assignment. The number of operations for this algorithm is αN+βL^{3}log L where N is the number of particles, L is the size of the (cubic) mesh and α and β are constants.

The PP force component used in the P^{3}M algorithm only needs to be calculated out to a small radius (typically 2.2 potential mesh cells). Thus the P^{3}M algorithm has an operation count that scales as αN+βL^{3}log L +$\gamma \sum$N$_{pp}^2$, where γ is a constant and N$_{pp}^2$ corresponds to the number of particles in the short range force calculation within a specified region. The summation is performed over all the PP regions, which are identified using a chaining mesh of size close to the short range cutoff. However, P^{3}M suffers the drawback that as clustering of matter develops more particles contribute to the N$_{pp}^2$ term, producing a drastic slowdown in solution time for a given number of particles. It is possible to see slowdowns of the order of 100-fold.

Adaptive P^{3}M remedies the slow-down under clustering of P^{3}M by isolating regions where the N$_{pp}^2$ term dominates and solving for the short range force in these regions, termed 'refinements', using FFT methods on a sub-mesh, supplemented by less expensive short-range calculations. This process is a repeat of the P^{3}M algorithm except that the FFT is now calculated with isolated boundary conditions. At the expense of a little additional bookkeeping, this method dramatically circumvents the clustering slowdown. The operation count is now,α N + β L^3 log L + $\sum_j^R$(α_j N$_j$ + β_j L$_j^3$ log L + $\gamma_j \sum N_{j_{pp}}^2$), where R is the number of refinements. The α_j and γ_j are all expected to be very similar to the α and γ of the main solver, while the β_j are approximately four times larger than β due to the isolated Fourier transform. Dependent upon clustering, AP^{3}M is as much as 10 times, or more, faster than P^{3}M.

The basis of the SPH method is the smoothing kernel, $W(\Delta r, h)$ which depends on a local smoothing scale, h, and the radial separation, Δr. The concept of the smoothing scale comes from the following discreteness approximation;

$$< A(\mathbf{r}) >= \int d^3\mathbf{r}' A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h)$$
$$\rightarrow \sum_j m_j \frac{A(\mathbf{r}_j)}{\rho(\mathbf{r}_j)} W(\mathbf{r} - \mathbf{r}_j, h), \tag{5}$$

where m_j is the mass of a neighbour particle and ρ the density. In the limit $h \rightarrow 0$, $W(\mathbf{r} - \mathbf{r}', h) \rightarrow \delta(\mathbf{r} - \mathbf{r}')$. Thus the smoothing kernel can be viewed as a mechanism to interpolate continuous fields between discrete points. The smoothing kernel also has the property that, by using integration by parts, all derivatives of fields can be calculated using derivatives of the smoothing kernel. Therefore, terms

[1]More complex Green's functions are used to remove noise from the interpolation process, but the procedure is the same.

in the equation of motion can be calculated by summing over products of fields and kernel derivatives without having to do explicit local differencing. A number of detailed reviews [15,16] of the SPH method are available.

Shock capturing is facilitated by using an artificial viscosity term in the Euler equation. We also apply 'shear-correction' [17] to prevent spurious angular momentum transfer by the artificial viscosity. Written in terms of sums of neighbouring particles the Euler equation and energy equations we use are;

$$\frac{d\mathbf{v}_i}{dt} = -\sum_j^N m_j \left(\frac{P_i}{\rho_i^2} + \frac{\Pi_{ij}}{2}\right) \nabla_i \overline{W}(\Delta\mathbf{r}_{ij}, h_i, h_j) + \sum_j^N m_j \left(\frac{P_j}{\rho_j^2} + \frac{\Pi_{ji}}{2}\right) \nabla_j \overline{W}(\Delta\mathbf{r}_{ij}, h_j, h_i), \quad (6)$$

and,

$$\frac{du_i}{dt} = \sum_j^N m_j \left(\frac{P_i}{\rho_i^2} + \frac{\Pi_{ij}}{2}\right) \Delta\mathbf{v}_{ij}.\nabla_i \overline{W}(\Delta\mathbf{r}_{ij}, h_i, h_j). \quad (7)$$

where $\Delta\mathbf{X}_{ij} = \mathbf{X}_i - \mathbf{X}_j$, the bar denotes arithmetic averaging of a quantity and Π_{ij} is the artificial viscosity described in [18].

Finding solutions to the Euler and energy equations is comparatively straightforward. It is first necessary to create a list of neighbours for each particle. The contribution to the local density of each particle from neighbours is then evaluated using the smoothing kernel. Because we write the equation of motion in a form that does not depend on the density of neighbouring particles we do not need to calculate *all* particle densities first. Instead we store the list of neighbours during the density calculation and then reuse this list to calculate the kernel derivatives used in the Euler and energy equations.

The (greatly simplified) solution cycle of one time-step may be summarized as follows: (1) Place refinements within top level mesh, (2) Accumulate all (low cost) PP forces, (3) Assign mass to the density mesh, (4) Calculate forces via Fourier convolution, (5) Apply mesh force and accelerate particles, (6) Repeat 1-5 on the sub-meshes, (7) Calculate densities, hydrodynamic forces and energy changes for gas particles. Note that the procedure of placing meshes is hierarchical in that a further sub-mesh may be placed inside a sub-mesh. This procedure can continue to an arbitrary depth but, typically, speed-up only occurs to a depth of six levels.

3 Parallel Implementation

As can be seen from the solution cycle AP^{3}M is a sophisticated multi-part algorithm. Load balance strategies adopted in one part of the algorithm are not guaranteed to work in another. Thus we have had to use a combination of static and dynamic load balancing approaches. Our static load balancing tactic is the same as that discussed in [13], although, for completeness, we review this approach in the following paragraphs.

Both the PM and PP algorithms require a particle-in-cell bookkeeping method. A linked or ordered list may be used, or particle indices can be reordered to be contiguous within chaining cells (which optimizes data locality). We use particle reordering and have observed up to a two-fold performance improvement for clustered particle distributions [12] versus using a linked list. Note that these clustered regions are potentially millions of times denser than their surroundings. Naturally, such dense regions can lead to potentially huge load imbalances because processors store particles by geographic regions. Recursive orthogonal bisection is commonly used in treecodes to domain decompose particles to avoid this issue. However, bookkeeping becomes complex and some regions can develop obscure geometries (*e.g.* very long thin boxes) and strenuous communication patterns.

The solution adopted in our P^{3}M code is to use a 2-d block cyclic decomposition of columns of chaining cells over processors. We arrived at this distribution primarily since it represents a good balance between the desire to minimize memory overheads (determined by the surface area of any stored region) while at the same time providing a comparatively small quantum of work, thereby enabling good load balance. Load balance is ultimately a function of how many columns are assigned to a given processor, and good load balance requires a reasonably large number of blocks. Our decomposition also has the benefit of enabling a fixed communication pattern for ghost particles that can be optimized during code development. Four barriers are used in the computation corresponding to transfers of the left,right,top, and bottom ghost cell regions. Full details of the communication pattern are given in [13].

The PM calculation is more complex. Although particles are administered using the 2-d decomposition of chaining cells, the Fourier convolution is best calculated using a slab decomposition. To address this issue we first interpolate the particles on to a density mesh that is distributed in the same 2-d fashion as the particles. We then perform a 2-d to 1-d domain decomposition transformation of the density mesh. Once complete, the Fourier convolution is calculated in planes and then we perform the inverse 1-d to 2-d decomposition to ensure that all particles can interpolate forces from a local copy the potential mesh. The domain decomposition transformations are extremely fast on the T3E (less than 2% of the time of the solution cycle).

Our recent work has focused on adding adaptivity. Early on during development it was decided that one sided communication (via MPI-2) was necessary for refinements so that data could be retrieved without interrupting the calculation on the host PE. Without one sided communication it is necessary to use either polling or a multi-threaded implementation to check for incoming requests at regular intervals. Because placement of the sub-meshes is adaptive no fixed communication strategy can be adopted, and hence we were forced to use the `MPI_WIN_LOCK` primitives in

MPI-2. This primitive allows fine grained locking of individual PEs to enable asynchronous communication.

To implement the solution for a sub-mesh, which uses an isolated FFT, it was necessary to implement a new convolution routine. To aid code reuse we wrote a routine that can compute both periodic and isolated convolutions. This process was comparatively straightforward since we had previously built the full 3-d FFT from a series of 1-d FFTs. To ensure fast execution of the FFT we use the FFTW [19] computational kernel. We have also added a parallelized Green's function routine since each sub-mesh requires a different Green's function (the base mesh function can be saved).

Perhaps the most significant intellectual challenge in developing the parallel code has been the task of determining optimal refinement placement. This is a difficult global optimization problem since it is possible that regions that are spatially close may be more optimally calculated via a larger mesh that covers both of them. However, the problem is at least well posed; we need to determine the set of refinements that minimize the cost function for the entire mesh. Thus we must consider the cost of the PP work in each cell versus placing it into a refinement while, at the same time, considering the effect on local cells which may or may not be placed in refinements. The algorithm we use in the serial code proved to be implicitly serial and hence a new algorithm had to be developed. The steps in our new sub-mesh placing algorithm are as follows: (1) For each chaining cell evaluate the cost of doing PP work, (2) Identify all cells above a maximum work threshold - these are cells that ideally should be placed in sub-meshes and are called 'target' cells, (3) 'Flood fill' regions that have been flagged into cuboidal shapes, (4) Sub divide any regions which are (a) too long or (b) contain too few target cells (5) Ensure boundary regions around sub-meshes are correctly clipped. Self-tuning of cost parameters on different platforms could potentially aid performance in this section of code and is being considered.

Because the placement of sub-meshes is hierarchical, at some point mesh boundaries must be defined by coordinates that do not map to chaining cell boundaries. One must then test whether remote particles lie inside or outside a refinement boundary. To avoid rounding errors we use a global integer coordinate system and map refinement boundaries and particle positions on to this grid when testing for location.

Once the sub-meshes on the top level mesh have been placed, calculation of the solutions for the refinements proceeds using a task-farm controlled by a master PE. The master PE carries a list of all outstanding refinements which is updated by slave PEs which calculate the placement of new refinements during the calculation of any refinement they have been given. As refinements are calculated they are erased from the master list so that memory is not wasted on calculations which have already been performed. Each slave may place up to 200 new refinements and the master PE can carry a maximum of 1000 refinements at any particular time, although both these limits can be changed given available memory resources.

To ensure good load balance we divide up the placed sub-meshes into the following categories: (1) Refinements large enough to be calculated across the entire machine, (2) Refinements too large to be calculated on a single processor, but are sufficiently small to be calculated on a 'sub-machine' with n processors (where n$<$ machine size), (3) Refinements that can be calculated on one PE. The calculation order is as follows: first calculate all refinements requiring the whole machine, then decompose machine into a combination of sub-machines and single processors. Calculate a combination of sub-machine refinements and single processor refinements. When all sub-machine refinements are completed further decompose sub-machines into single processors and calculate all remaining single processor refinements. We also ensure that refinements are calculated in the order of work load. Therefore as the calculation progresses load balance is likely to be assured since the work quanta are getting progressively smaller as the end of the calculation is approached.

In the serial code and OpenMP versions we have an explicit barrier between levels of refinements. For example all sub-meshes placed within the top level mesh are calculated before proceeding to sub-meshes placed within the sub-meshes. The situation arose because refinement placement must be performed before calculating a sub-mesh however, preventing race conditions means waiting until the sub-mesh calculation has been performed before proceeding to calculate the sub-mesh placed within the sub-mesh. In the MPI code we only return the new refinement details once the sub-mesh has been calculated, therefore calculation of the new refinements may proceed once the master PE has received any new refinements.

The communication involved in calculating a refinement is as follows: the machine/sub-machine master PE, or individual PE receives the geographic location of the refinement and the number of particles from the global communicator master PE. The remote particle data is then retrieved by a series of `MPI_GET` operations within an access window defined by `MPI_WIN_LOCK` and `MPI_WIN_UNLOCK`. For small refinements all the data may be contained on one PE, while for larger refinements a number of messages may be required. The master PE in the *local* communicator is responsible both retrieving and redistributing the data within the sub-machine. Once the data has been distributed the calculation proceeds in precisely the same fashion as for the main mesh (no optimizations for refinements calculated on one PE have been included). Once the calculation cycle is complete the particle forces are then accumulated back to PEs from whence they came. The local master PE is again responsible for this operation and the `MPI_ACCUMULATE` operation is used within another `MPI_WIN_LOCK(UNLOCK)` access epoch.

Once the AP^{3}M algorithm was complete we decided to implement SPH in a separate module. In both the serial and OpenMP versions of the code the SPH and AP^{3}M solvers

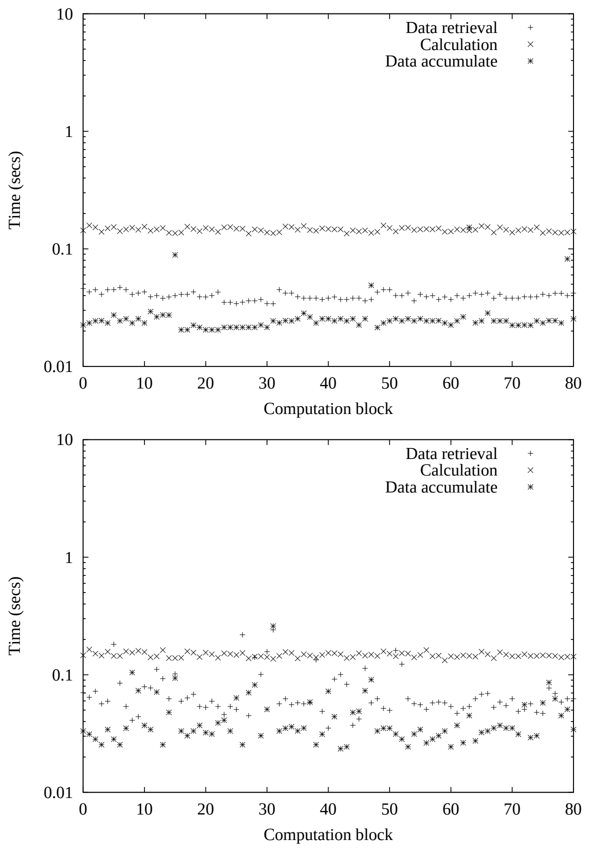

Figure 1. Comparison of MPI-2 performance for fully loaded versus half loaded SMP nodes. The top graph was run using only 2 cpus out of 4 in each SMP node, while the bottom graph was run using all 4 cpus in each SMP node. Both runs used a total of 16 processors. When the SMP nodes are fully loaded communication times become highly inconsistent and large fluctuations in execution time result. Although the messaging fluctuations appear innocuous in the fully loaded run, parallel efficiency is strongly impacted by the slowest PE.

are intertwined, with the SPH solver borrowing the list mechanism from the gravity solver. This is a highly efficient way of doing things since the SPH code can reuse the neighbour list calculated for the AP^{3}M solver. However, it limits the maximum neighbour search radius of the SPH to be one chaining cell which is insufficient for a number of cosmological simulations and it also icreases code complexity. Hence we separated the two solvers.

Because SPH is approximately an order N algorithm we can rely upon the block cyclic distribution of particles for load balance. Each PE must calculate the SPH for each column of chaining cells after first retrieving all the of the ghost particles. We retrieve two layers of chaining cells for the ghost particles which would be expensive memory-wise if we used the same calculation pattern as the PP part of the code. However, instead we pay a small communication latency to trade-off memory use, and the overhead is still significantly smaller than the calculation load. In figure 1 we plot the calculation and communication overheads for a number of columns in the initial configuration of a simulation. This plot also shows overheads associated with MPI-2 communication threads.

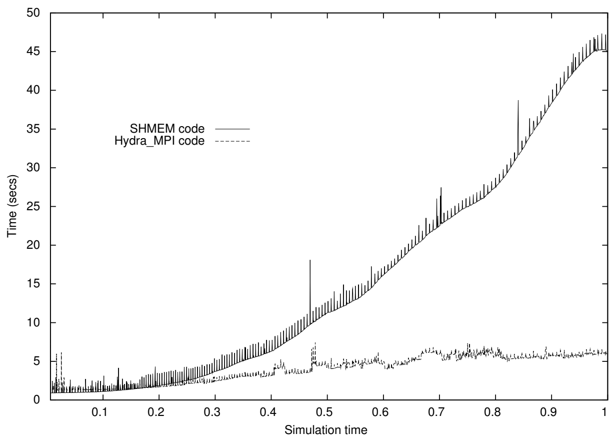

Figure 2. Comparison of solution cycle time for P^{3}M (SHMEM) compared to AP^{3}M (HYDRA-MPI). Each simulation used 16 PE's on a Cray T3E-900 and 2×10^5 particles. Spikes on both plots are associated with I/O overheads for checkpointing.

At all stages during the development we have paid great attention to correctness checking. Unfortunately the complexity of the algorithm can make this difficult. For example, the distribution of optimal refinements is different for the serial versus parallel codes, and force errors will be subtly different between the two codes. However, because the refinements are calculated to preserve a given force accuracy both codes return forces with errors within a given RMS bound and hence both sets of forces are acceptable. Therefore, to establish correctness we have compared forces on a particle-by-particle basis to the full P^{3}M solution. All tests show the MPI code reproduces errors in agreement with the RMS limits for a variety of different particle configurations.

4 Parallel Performance

Performance details for the non-adaptive parts of the MPI-2 code are the same as the Cray SHMEM code [13]. To test the adaptive parts of the algorithm we have conducted two simulations, one examining the performance of the raw gravity solver, and the other the performance of the combined AP^{3}M+SPH solver. The speed of the adaptive gravity solver for a simulation with 2×10^5 particles is compared to the unadaptive algorithm in figure 2. Under clustering the adaptive algorithm slows only by a factor of around 6, while the unadaptive algorithm slows by a factor of almost 50. Improving the placement of refinements could probably speed the adaptive algorithm up further, the serial code slows only by a factor of around 3 on a similar particle distribution. Refinement placing in the parallel code is not yet fully optimized.

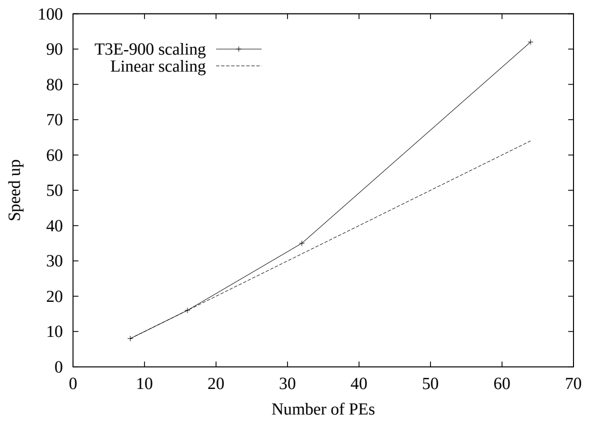

Figure 3. Parallel scaling for the initial configuration of the Santa Barbara cluster simulation (2×64^3 particles, half dark matter, half gas) on the T3E-900. Results are super-linear due to the reduced messaging thread overhead as the amount of data per PE is reduced.

To test the full algorithm we ran the 'Santa Barbara' galaxy cluster simulation [20]. Parallel speed up figures for the beginning of the simulation are shown in figure 3. The scaling is super-linear because the overhead of threads used in MPI-2 RMA becomes smaller as the amount of data per PE is reduced (a PE deals with fewer messages). Under clustering the speed of the code slows down by around a factor of 6 on 8 PEs, we have not yet tested on higher numbers of PEs. It should be borne in mind that the number of particles per PE is expected to be in the region of 2-3$\times 10^5$ in full production runs, while our figures show close to this number distributed across a large number of PEs. We expect excellent parallel scaling due to the large work quantum in production runs. In terms of the number of particle updates per second (PUPS), we achieve a maximum speed of around 15,000 PUPS/PE for gravity and around 6,000 PUPS/PE for the SPH on the T3E-900. Results on the SC cluster at McMaster are 70,000 PUPS/PE for gravity, and 30,000 PUPS/PE for SPH. These numbers are reduced by clustering in the particle distribution.

5 MPI-2 RMA implementations

Our decision to use MPI-2 was strongly motivated by the nature of the AP^{3}M algorithm. The need to load and unload data to remote processors, that may be involved in a calculation at that time, necessitates one-sided communication or a multi-threaded implementation to work effectively. For simplicity we chose to embark on the one-sided route. As adaptive mesh algorithms become more popular we believe that other researchers will be faced with similar dilemmas.

This design decision has created significant problems: at the time of writing we have only worked on 1 platform that has a fully compliant and *stable* version of MPI-2, namely the Cray T3E (although both Fujitsu and IBM claim to have fully compliant MPI-2 libraries). Neither is there a publically available version of MPI-2, although progress on MPICH-2 [21] is moving forward. On our SC cluster we have been able to use Compaq's 'Alaska MPI' distribution. However, as of time of writing (March 2003) a fully stable version of 'Alaska MPI' has only just been released, and unfortunately it remains an unsupported product. We are working with Compaq/HP to address this issue.

Implementing MPI-2 efficiently entails hidden costs. Because it is impossible to map non-simply connected regions to contiguous memory addresses our remote memory requests are non-contiguous[2]. Implementing this operation is difficult since a stencil of the required memory addresses must be passed to a thread on the target processor which then retrieves and packs the required addresses before sending the message back to the origin processor. Also, because any particular particle may receive contributions to its force update from a number of other remote particles we use the `MPI_ACCUMULATE` operation extensively. This operation is used in a mode equivalent to performing a remote addition on the target processor. However, implementation of this operation again requires a remote thread that consumes memory and CPU time. Figure 1 gives a clear representation of these overheads.

It is interesting to note that similar problems relating to copy overheads and TCP/IP stacks are appearing in high performance networking. The latest high performance Gigabit ethernet cards have 'TCP off-load engines' (TOEs) to remove TCP kernel overheads from the CPU. The overheads associated with copying memory (which is a slow operation due to limitations of memory bandwidth) have lead researchers to examine 'zero copy' implementations of MPI as well as remote direct memory access (RDMA) over TCP/IP. It is our belief that the best way of ensuring high performance for *all* the RMA operations possible within MPI-2 is to provide some kind of hardware RMA off-load engine. Given the growth of TOEs and that current high performance networking cards, *e.g.* the Quadrics Elan3 card, contain a programmable co-processor, hardware assistance appears to be more than a possibility.

6 Summary and Discussion

We have described the design and implementation of an MPI-2 version of the AP^{3}M+SPH algorithm: HYDRA-MPI. The code has been executed in a non-adaptive form on a number of different architectures: Cray T3E, Compaq SC, Sun E3500 SMP, SGI Origin 2000 and Beowulf systems. At present only platforms supporting the MPI-2 `MPI_WIN_LOCK` RMA functionality can run the full adaptive algorithm. This limits the number of architectures that we can currently utilize due to the slow adoption of MPI-2.

The non-adaptive code is known to scale well to hundreds of processors. The adaptive code is as yet untested

[2]Of course non-contiguous regions of memory can be broken into a series of contiguous regions, but maximal network bandwidth is not usually achieved with small messages.

on large problems, although for small problems we observe excellent, in some cases super-linear, scaling. The MPI code contains improvements that were observed to speed up both the serial and parallel OpenMP codes.

Over the coming months we plan to test the code on both the (US) Terascale and NERSC facilities. Unfortunately, at present, there are no large scale facilities in Canada with the MPI-2 support necessary to run our code. Once we gain experience of working with the code in an MPP environment we plan to embark on an aggressive simulation program. Our near-term goal is to conduct simulations with 10^{10} particles. This calculation will require close to 10^{18} Flops, or about 10 days on a *sustained* Teraflop class facility. Detailed analysis and visualization of the multi-Terabyte data-set that will result will present a whole new set of challenges.

This project has greatly benefitted from the input of J-C Desplat, Adrian Jenkins, Tom MacFarland, Peter Thomas, Frazer Pearce and Richard Warren of HP. RJT is supported by the Canadian Computational Cosmology Consortium. HMPC acknowledges the support NSERC and the CIAR.

References

1. MAXIMA Collaboration
http://cosmology.berkeley.edu/group/cmb/
2. DEEP2 Galaxy Survey
http://deep.berkeley.edu/ ˜marc/deep/
3. Supernova Cosmology Project
http://www-supernova.lbl.gov/
4. Gingold R. A. & Monaghan J. J. Monthly Notices of the Royal Astronomical Society, 181, 375 (1977).
5. Hockney R. W. & Eastwood J. W. Computer Simulation Using Particles, McGraw-Hill (1988).
6. Barnes J. & Hut P. Nature 324, 446 (1986).
7. Sugimoto D. *et al.* Nature 345, 33 (1990).
8. Couchman H. M. P., Astrophysical Journal, 368, L23 (1991).
9. Evrard A. *et al.* , Astrophysical Journal, 573, 7 (2002).
10. The HYDRA Consortium
http://coho.mcmaster.ca/hydra
11. Pearce F. R. & Couchman H. M. P. New Astronomy, 2, 411 (1997).
12. Thacker R. J. PhD Thesis, University of Alberta, (1999).
13. MacFarland T., Couchman H. M. P., Pearce F. R. & Pichlmeier J., New Astronomy, 3, 687 (1998).
14. Official documentation of the MPI standard
http://www.mpi-forum.org/docs/docs.html
15. Monaghan J. J. Annual Review of Astronomy and Astrophysics, 30, 543 (1992).
16. Steinmetz M. Monthly Notices of the Royal Astronomical Society, 278, 1005 (1996).
17. Balsara D., Journal of Computational Physics, 121, 357 (1995).
18. Thacker R. J., Tittley E. R., Pearce F. R., Couchman H. M. P. & Thomas P. A., Monthly Notices of the Royal Astronomical Society, 319, 619 (2000).
19. FFTW subroutine library
http://www.fftw.org
20. Frenk C. S. *et al.* , Astrophysical Journal, 525, 554 (1999).
21. MPICH-2 status
http://www-unix.mcs.anl.gov/ ˜
gropp/projects/parallel/MPICH/mpich2/