SCALING AND PERFORMANCE

PROGRESS TOWARDS A COMPREHENSIVE THEORY OF STAR FORMATION – FROM BROWN DWARFS TO HIGH MASS STARS, CLUSTERS AND ON TO GIANT MOLECULAR CLOUDS

Richard I. Klein

Department of Astronomy, University of California, Berkeley, and Lawrence Livermore National Laboratory, P.O. Box 808, L-23, Livermore, CA 94550

A. Computational Methodology/Algorithms

First, the code employs a conservative higher-order Godunov scheme to solve the equations of compressible magneto-hydrodynamics using optimized approximate Riemann solvers (Toro, 1997). The algorithm is second-order accurate in both space and time for smooth flow problems, and it has a robust and accurate treatment of shocks and contact discontinuities. We have both pure HD and ideal MHD modules using an unsplit TVD scheme, equipped with different Riemann solvers including the Roe, and the HLL-family solvers (**Li** *et al.* 2012).

The second major component of the code is a highly scalable self-gravity solver developed by Martin *et al.* (2008). At each time step we solve a Poisson problem on the adaptive grid hierarchy to obtain the gravitational potential; we then apply the gradient of this potential as a source term in the momentum and energy equations (Truelove *et al.* 1998). A multigrid iteration scheme is used to solve the linear system of equations resulting from the discretization of the Poisson equation on a level. These level solutions are then iterated to convergence to obtain a solution for the gravitational potential on all levels.

The third component is an adaptive, coupled radiation hydrodynamics solver using single-frequency flux-limited diffusion. The radiation transfer module uses a split method optimized for physical conditions where radiation-gas energy exchange by emission/absorption dominates the work done by the radiation field on the gas. First, the code solves a fully implicit system consisting of the emission/absorption and diffusion parts of the radiation and gas energy equations (Howell & Greenough 1999). It uses a Newton-Raphson iteration method, with an adaptive parallel multigrid method to find provisional solutions in each loop of the iteration. Once the implicit system reaches convergence, the algorithm updates the gas and radiation states using explicit forms of the radiation pressure force, work and advection terms to $O(v/c)$. This allows the incorporation of radiation pressure with no significant increase in computational cost, while maintaining high accuracy in the regimes with which we are concerned.

The fourth component is an adaptive ray tracing method developed by Abel & Wandelt (2002) and Abel et al. (2007). In this method, we solve the radiative transfer equation along rays which split when the area they subtend exceeds a threshold, usually set to one-forth of a cell area. This allows excellent angular resolution everywhere without requiring an extraneous number of rays near the source. At the beginning of each ray trace, each source launches photon packets that travel until they reach a grid edge or lose 99.9% of their initial photon flux. Once photon packets are released, each processor loops over their packets, moving them from grid to grid until they are extinguished or reach the end of the grid domain for that particular processor. At grid edges, the photon packet uses ORION2's AMR framework to find its next grid. Each processor then communicates with the other processors to see if photon packets need to be exchanged. This process repeats until all photon packets are extinguished. Now with the photoionization rates computed, the ionization time step is set to be the maximum value that restrains any cell from changing its neutral fraction by more than 10%. Equivalently, this prevents an ionization front from moving through a cell in less than ten ray trace cycles. This choice follows **Krumholz,**

Stone, & Gardiner (2006), who showed this allows for accurate solutions because it ensures the change in the cell's thermal energy due to ionization heating or recombination cooling is constrained. The entire ray trace process is repeated until the hydrodynamic time step has elapsed. Then, the FLD solver computes a new solution for the non-ionizing radiation, hydrodynamics is updated via a TVD or PPM scheme, and the update is complete (**Oishi** *et al.* 2010).

These solvers are coupled together within the adaptive mesh refinement infrastructure using Chombo library. The adaptive mesh refinement scheme employs an automatic, dynamic regridding strategy based on an underlying rectangular discretization of the spatial domain (Berger & Oliger 1984, Berger & Colella 1989; Bell *et al.* 1994, Colella *et al.* 2000). The overall algorithm conserves total energy, mass, and momentum.

B. Scalability Tests & Timings

To characterize the scalability of the **ORION2** code, we ran a multiple blast wave test problem including MHD, self-gravity, and diffuse radiation on a single level of refinement for 10 coarse timesteps. For the 8 CPU version, we tiled a 128^3 domain with 8 64^3 grids, each containing identical blast waves. As we increased the number of CPUs, we expanded the domain with additional grids such that the total work per CPU remained constant. We measured the total time spent in different physics updates on the Stampede platform timers provided by Chombo. The CPU time per cell advance is shown for each module in Table 1. The results demonstrate excellent algorithmic scalability of the hyperbolic portion of the code to at least 4096 processor cores. The radiation module causes the greatest performance bottleneck owing to high communication overhead, but it too shows reasonable performance on up to 4096 processor cores.

Table 1: Stampede **ORION2** code scaling test in μs.

$N_{\rm CPU}$	$t_{\rm MHD}$	$t_{\rm gravity}$	$t_{\rm radiation}$	$t_{\rm total}$	MHD eff	Gravity eff	Radiation eff	Overall eff
8	5.53	7.94	11.33	24.8	1	1	1	1
64	5.57	8.41	12.66	26.64	0.99	0.94	0.89	0.93
512	5.87	9.54	15.14	30.56	0.94	0.83	0.75	0.81
1024	6.03	9.74	16.56	32.32	0.92	0.82	0.68	0.77
2048	6.18	11.70	20.83	38.71	0.9	0.68	0.54	0.64
4096	6.10	14.33	28.00	48.43	0.91	0.55	0.4	0.51

Next, we took one of our previous high mass models and timed the gravity, radiation, and MHD update times. On 256 processors, the CPU time per cell advance for the MHD, gravity and radiation modules were 43, 31, and 230 μs respectively. These results demonstrate that the actual cost of a cell update in a realistic problem may be substantially higher than those of our idealized estimate presented above. As a result, we use our realistic estimates when determining timing analysis (see below). We furthermore note that the benchmark timing results for the MHD module are typically about twice that of the purely hydrodynamic timings. We will therefore estimate that realistic applications with hydrodynamics only will require half the cell cycle time, or 22 μs.

Ray tracing is an inherently difficult problem to parallelize: in star formation problems, sources are typically clustered together and thus located on a small fraction of the processors. These processors begin the ray trace, but all other processors sit idle until rays advance into their domain. Load balancing and interprocessor communication ultimately become the limiting factors for all ray tracing implementations. Our implementation of asynchronous communication and load balancing have helped address these issues.

For weak and strong scaling studies, we used a highly simplified star formation model that is reminiscent of our actual calculations utilizing direct radiation and photoionization. This calculation resembles that done in **Krumholz et al. (2006)**. We embed a radiating source at the center of a (1 parsec)^3 box with a constant neutral gas density (= 10^4 cm^{-3}). The parameters that dictate how often rays divide is set to ensure that each cell is impacted by ~ 4 rays, which we find is required to achieve the appropriate accuracy in our radiation tests. Only the ray tracing module is used during the simulation. For these tests, the rays interact with the cells they cross but do not change the state of the cell. We artificially truncate the rays once they travel a certain distance, allowing each ray trace to be nearly identical for timing estimates.

For weak scaling tests, we find that communication begins to dominate the runtime, increasing the overall wall-clock time, at and above 128 processors. At this point each processor is stalled because it must send and receive rays to and from a considerable number of other processors. Since each processor has an identical workload, our asynchronous communication and load balancing improvements will not address this issue. However, our star formation problems will not exhibit this sort of load balancing for ray tracing, and thus weak scaling tests are not appropriate for our problems. We instead use the strong scaling results for our timing estimates.

For the strong scaling tests, we use a single source in a (1 parsec)^3 domain on a 256^3 grid. This grid is spread across a varying number of processors $p = 2^n$ ($n = 6, 7, 8$). The radial extent of the rays is set to either 0.1 pc, 0.2 pc, or 0.4 pc. These numbers are representative of typical star formation problems, where, in the case of ionization, the ionized regions surrounding the stars will constitute a small volume compared to the entire simulation domain.

To estimate the total CPU time for ray tracing, we've devised a formula that calculates the CPU time required for a single ray trace per cell as a function of the number of processors and the number of cells in the simulation that interact with the rays I_c. The latter accounts for how far rays travel when cast. Optically thick gas will halt the rays, while rays traveling through tenuous or ionized gas could travel far from their source. Our tests give us a two-dimensional grid of data, and a least-squares fit is made to the formula

$$r(p, I_c) \equiv \begin{bmatrix} \text{CPU time} \\ \text{per ray trace per cell} \end{bmatrix} = A \cdot \left(\frac{p}{256}\right)^B \cdot \left(\frac{I_c}{100,000}\right)^C , \quad (1)$$

yielding $A = 15\ \mu s$, $B = 0.2$, and $C = 0.7$.

The complete menu of physics modules includes hydrodynamics, magnetohydrodynamics, self-gravity, radiation diffusion, and photoionization. The cell update times for the first four physics modules required in realistic problems are summarized in table 2, with photoionization given by Equation (**??**). We again note that the radiation diffusion and gravity steps are implicit operators that require more iterations for convergence in problems where these processes dominate the flow dynamics. In practice problems including high mass source $M > 15\mathrm{M}_\odot$ require twice the radiation diffusion time of problems including only low mass sources. The time and that the cell update times reported in the table will be multiplied by the typical number of iterations required in similar scoping calculations.

C. Optimizations

Our code realizes great cost and memory efficiency by concentrating computational effort only where the flow demands it. In contrast to more conventional hydrodynamics and MHD codes, which use static uniform or graded grids, static nested grids, or a fixed number of hydrodynamic particles (SPH), AMR actively assesses the solution with respect to user-specified refinement criteria in order to guide the dynamic insertion and removal of rectangular fine grid patches. Dramatic speedups are found for 3-D AMR gravitational hydrodynamic calculations – indeed,

Table 2: Stampede cell update time for typical applications (μs).

Code Module	Cell Update Time
Hydrodynamics	22
Magnetohydrodynamics	43
Gravity	31
High Mass Star Radiation	230
Low Mass Star Radiation	115

with maximum refinements in linear extent in excess of order $10^4 - 10^5$ being not uncommon, a fixed grid code would require of order $10^{12} - 10^{15}$ cells throughout the *entire* duration of the simulation to cover the same region, where on such a calculation we typically use of order 10^6- several 10^7 cells, and even then, only after high-density structures have developed (Truelove *et al.* 1997).

In general, there are two bottlenecks to the implementation of parallel algorithms: communications costs and load balancing. Grids are distributed to processors in a round-robin fashion to do computation. Chombo uses the message passing interface (MPI) to handle all low-level parallelism aspects (data distribution, communication, etc.), while hiding the infrastructure complications from developers building physics solvers and from scientist end-users doing computations. While the elliptic and parabolic gravity and radiative diffusion solvers have higher communications overhead than hyperbolic solvers, with the most recent version of the Hypre library designed by Brown *et al.* (2000), it is now possible to achieve good scalability to at least 4096 processors. Efficient load balance also presents a challenging obstacle, due to the relatively broad distribution of grid sizes (which typically range from 5000 cells to $2.5 \cdot 10^5$ cells) which the AMR algorithm generates in order to minimize computation. However, as Rendleman *et al.* (1998) point out, it is possible to achieve very good load balance as long as there are many more grids than processors.

D. General Timing Analysis

We estimate the runtime $t_{\rm run}$, for any run on a single processor, simply as the product of the average cpu-time to advance a cell times the total number of cell advances :

$$t_{\rm run} = \frac{\rm seconds}{\rm cell\ cycle} \times \mathcal{N}_{\rm adv}^{\rm coarse} + r(1, I_c) \times (\text{total number of ray traces}) \times (\text{total number of cells}) \ , \quad (2)$$

where $\mathcal{N}_{\rm adv}^{\rm coarse}$ is the total number of cell advances over all levels per coarse timestep. If ray tracing is not included, the second term is omitted.

The number of coarse cycles required is simply the ratio of the total run time to the timestep on the coarsest level. The former is the characteristic timescale for the problem, which is the free-fall time for any collapse calculation in the absence of stellar winds. This is given by

$$t_{\rm ff} = \sqrt{\frac{3\pi}{32 G \rho_{\rm edge}}} = \frac{.54}{\sqrt{G\rho_{\rm edge}}}, \quad (3)$$

where $\rho_{\rm edge}$ is the smallest density at any point in the collapsing cloud. The coarse time step is determined by the Courant condition:

$$\Delta t^{\rm coarse} = C \frac{\Delta x^{\rm coarse}}{\max(v) + c_s}. \quad (4)$$

Here C is the Courant number (typically 0.5), $\max(v)$ is the largest 1-D velocity in the problem domain, and c_s is the sound speed.

The total number of cell advances over all levels for each coarse time step in an AMR calculation is determined by the distribution of adaptive grids. If there are N_k cells on level k ($k = 0$ .. $k_{\max}$), with level k a factor f (typically 2) more refined than level $k-1$, and $k_{\max}+1$ levels in total, then we may write

$$\mathcal{N}_{\rm adv}^{\rm coarse} = N_0(1 + fN_1/N_0 + f^2N_2/N_0 + f^3N_3/N_0 + ...) = N_0 \sum_{k=0}^{k_{\max}} f^k(N_k/N_0) \equiv \chi N_0. \quad (5)$$

Note the dimensionless factor χ in the above expression has a simple interpretation : it is how much longer all levels of an adaptive mesh refinement calculation take, per timestep, over a fixed grid calculation with base level 0 resolution. χ may be explicitly computed given an estimation of the number of refined cells on each level $k > 0$ and the base grid level $k = 0$ as

$$\chi = \sum_{k=0}^{k_{\max}} 2^k N_k/N_0. \quad (6)$$

To accurately couple with our diffuse radiation modules, it is necessary to call ray tracing on every AMR level. Comparatively, the timing of ray tracing on the finest AMR level greatly exceeds the time spent on the coarser levels, and therefore we estimate the total ray tracing time to be equal to the time spent tracing rays on the finest AMR level. The number of ray traces is estimated as the ratio of the total run time to the typical ray trace time step $\Delta t^{\rm ion}$ on this fine level. For the latter, this time step is equal to (and never greater than) the time step on the finest AMR level when ionization is not included. With ionization, sub-cycling is possible, since our ray tracing routine sets the ionization time step by limiting the ionization front from moving through a cell in less than ten ray traces. This timescale can be estimated by the ionization timescale for the predominately neutral cells at the front:

$$\Delta t^{\rm ion} = \frac{n_{\rm tot}}{n_{\rm H}\sigma_{\rm H}} \frac{4\pi r^2 e^\tau}{S} = \frac{4\pi r^2 e^\tau}{\sigma_{\rm H} S} = 1.9 \times 10^6 e^\tau \left(\frac{r}{3 \times 10^{18}\ {\rm cm}}\right)^2 \left(\frac{S}{10^{49}\ {\rm sec}^{-1}}\right) . \quad (7)$$

where r is the characteristic size of the ionized region around the source, S is the number flux of ionizing photons from the source, and $\tau \approx 9.4 \times 10^{-5} \cdot n_H \cdot (r/{\rm AU})$ is the optical depth of the zones between the source and first neutral zone at the edge of the ionization front. Here n_H is measured in units of ${\rm cm}^{-3}$. For highly dense regions near the star, e^τ can be substantial even when the neutral fraction is as low as $\sim 10^{-3}$.

For a single CPU we therefore have

$$t_{\rm run} = \frac{\rm seconds}{\rm cell\ cycle} \times \frac{t_{\rm ff}}{\Delta t^{\rm coarse}} \times \chi N_0 + r(1, I_c) \times \frac{t_{\rm ff}}{\Delta t^{\rm ion}} \times \chi N_0 \ . \quad (8)$$

If ray tracing is not included, the second term is omitted. If we run in parallel on p processors, with a speedup factor $s(p)$, the wall clock time needed is

$$t_{\rm run} = \frac{\rm seconds}{\rm cell\ cycle} \times \frac{t_{\rm ff}}{\Delta t^{\rm coarse}} \times \frac{\chi N_0}{p\, s(p)} + r(p, I_c) \times \frac{t_{\rm ff}}{p\, \Delta t^{\rm ion}} \times \chi N_0 \ , \quad (9)$$

and the total CPU time required is $t_{\rm CPU} = p t_{\rm run}$. In general, due to scalar overhead and finite problem size, the performance of the code will be less than optimal, and so $s(p) < 1$. By definition, $r(p, I_c)$ incorporates the ray trace's scaling efficiency.