AMR and GRAVITY in RAMSES

Patrick Hennebelle
Patch-based versus tree-based

Courtesy Romain Teyssier
Cell-centered variables are updated level by level using linked lists.

Cost = 2 integer per cell.

Optimize mesh adaptation to complex flow geometries, but CPU overhead compared to unigrid can be as large as 50%.

2 type of cell: “leaf” or active cell
“split” or inactive cell
Refrinement rules for graded octree

Step 1: mesh consistency
if a split cell contains at least one split or marked cell, then mark the cell with flag = 1 and mark its 26 neighbors

Step 2: physical criteria
quasi-Lagrangian evolution, Jeans mass
geometrical constraints (zoom)
Truncation errors, density gradients...

Step 3: mesh smoothing
apply a dilatation operator (mathematical morphology) to regions marked for refinement → convex hull

Courtesy Romain Teyssier
Godunov schemes and AMR

Berger & Oliger (84), Berger & Collela (89)

Prolongation (interpolation) to finer levels
- fill buffer cells (boundary conditions)
- create new cells (refinements)

Restriction (averaging) to coarser levels
- destroy old cells (de-refinements)

Flux correction at level boundary

\[
\left( F_{i+1/2,j}^{n+1/2} \right) = \frac{\left( F_{i+1/2,j+1/4}^{n+1/2} \right) + \left( F_{i+1/2,j-1/4}^{n+1/2} \right)}{2}
\]

Careful choice of interpolation variables (conservative or not ?)

Several interpolation strategies (with \( R^T P = I \)):
- straight injection
- tri-linear, tri-parabolic reconstruction

Courtesy Romain Teyssier
Buffer cells provide boundary conditions for the underlying numerical scheme. The number of required buffer cells depends on the kernel of the chosen numerical method. The kernel is the ensemble of cells on the grid on which the solution depends.

- First Order Godunov: 1 cell in each direction
  \[ u^{n+1}_i = u^n_i (1 - C) + u^n_{i-1} C \]

- Second order MUSCL: 2 cells in each direction
  \[ u^{n+1}_i = u^n_i (1 - C) + u^n_{i-1} C - \frac{C}{2} (1 - C) (\Delta u_i - \Delta u_{i-1}) = 0 \]

- Runge-Kutta or PPM: 3 cells in each direction

Simple octree AMR requires 2 cells maximum. For higher-order schemes (WENO), we need to have a different data structure (patch-based AMR or augmented octree AMR).
Adaptive Time Stepping

Time integration: single time step or recursive sub-cycling

- froze coarse level during fine level solves (one order of accuracy down !)
- average fluxes in time at coarse fine boundaries

\[
(F^{n+1/2,\ell}_{i+1/2,j}) = \frac{1}{\Delta t_{\ell}^{+1} + \Delta t_{2}^{+1}} \left( \Delta t_{1}^{+1} \frac{(F^{n+1/4,\ell+1}_{i+1/2,j-1/4}) + (F^{n+1/4,\ell+1}_{i+1/2,j+1/4})}{2} + \Delta t_{2}^{+1} \frac{(F^{n+3/4,\ell+1}_{i+1/2,j-1/4}) + (F^{n+3/4,\ell+1}_{i+1/2,j+1/4})}{2} \right)
\]
The AMR catastrophe

First order scheme: the AMR extension is \textit{not consistent} at level boundary.

Second order scheme: for $\alpha=1.5$, AMR is \textit{unstable} at level boundary.

Solutions: 1- refine gradients, 2- enforce first order, 3- add artificial diffusion

Assume $a$ and $C>0$. 

First order scheme:
\[
\frac{u_i^{n+1} - u_i^n}{\Delta t} + a \frac{u_{i+1/2}^{n+1/2} - u_{i-1/2}^{n+1/2}}{\Delta x} = 0
\]

Second order scheme:
\[
\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = a \frac{\Delta x}{2} (\alpha^2 - C) \frac{\partial^2 u}{\partial x^2} + O(\Delta t^2, \Delta x^2)
\]

At level boundary, we lose one order of accuracy in the modified equation.

Romain Teyssier
Shock wave propagating through level boundary

Courtesy Romain Teyssier
Shock-tube test with regular grid

100 Eulerian mesh points with PLM

Courtesy Romain Teyssier
Shock-tube test with AMR

120 AMR mesh points with RAMSES

Courtesy Romain Teyssier
Maximum numerical dissipation occurs at the 2 fluids interface.

The optimal refinement strategy is based on density gradients.

The number of required cells is directly related to the fractal exponent $n$ of the 2D surface.

$$N_{cell} \propto (\Delta x)^{-n}$$

Courtesy Romain Teyssier
Particle-Mesh on AMR grids:
Cloud size equal to the local mesh spacing

Poisson solver on the AMR grid
Multigrid or Conjugate Gradient
Interpolation to get Dirichlet boundary conditions (one way interface)

Quasi-Lagrangian mesh evolution:
roughly constant number of particles per cell
\[ n = \frac{\rho_{DM}}{m_{DM}} + \frac{\rho_{gas}}{m_{gas}} + \frac{\rho_*}{m_*} \]

Trigger new refinement when \( n > 10-40 \) particles. The fractal dimension is close to 1.5 at large scale (filaments) and is less than 1 at small scales (clumps).

Courtesy Romain Teyssier
RAMSES: a parallel graded octree AMR

- Tree-based AMR (octree structure): the cartesian mesh is recursively refined on a cell by cell basis.
- Full connectivity: each “oct” have direct access to neighboring parent cells and to children “octs”. (memory overhead: 2 integers per cell).

→ Optimize the mesh adaptivity to complex geometries, but CPU overhead can be as large as 50%.

N body module: Particle-Mesh method on AMR grids (similar to the ART code).

Hydro module: Unsplit second order Godunov method: Riemann solver with piecewise linear reconstruction (option: MUSCL or PLMDE).

Time integration: Single time step or W cycle (fine levels subcycling)

Other: Cooling & UV heating, Zoom simulation technology

MPI based parallel implementation → Space Filling Curves

Courtesy Romain Teyssier
Domain decomposition for parallel computing

Parallel computing using the MPI library with a domain decomposition based on the Peano-Hilbert curve.

Algorithm inspired by gravity solvers (tree codes).
*Use locally essential tree.*

Tested and operational up to 6144 core.
Scaling depends on problem size and complexity.


*Courtesy Romain Teyssier*
Locally essential trees

Each processor octree is surrounded by ghost cells (local copy of distant processor octrees) so that the resulting local octree contains all the necessary information.

Locally essential tree in processor #1

Locally essential tree in processor #2

Locally essential tree in processor #3

Salmon 90, Warren 92, Dubinsky 96

Courtesy Romain Teyssier
Dynamic partitioning in RAMSES

Several cell ordering methods:
1- column, row or plane major
2- Hilbert or Morton
3- User defined (angular, column+Hilbert…)

Dynamic partitioning is performed every N steps by sorting each cell along chosen ordering and redistributing the mesh across processors. Usually, a good choice is N=10 (10% overhead).

 Courtesy Romain Teyssier
Are there any optimal load balancing strategy?

Column major

Angular

Hilbert

Courtesy Romain Teyssier
Are there any optimal load balancing strategy?

Recursive bisection

Courtesy Romain Teyssier
Are there any optimal load balancing strategy?

Hilbert ordering

Courtesy Romain Teyssier
Are there any optimal load balancing strategy?

Angular ordering

Courtesy Romain Teyssier
Gravity in RAMSES

\[ \Delta \Phi = 4\pi G\rho \]

\[
\Rightarrow \Phi_{i+1} - 2\Phi_i + \Phi_{i-1} = 4\pi G dx^2 \rho_i \quad \text{(in 1D)}
\]

\[
\begin{pmatrix}
-2 & 1 & 0 & 0 \\
1 & . & 0 \\
0 & . & 1 \\
0 & 0 & 1 & -2
\end{pmatrix}
\begin{pmatrix}
\Phi_1 \\
\Phi_2 \\
\Phi_3 \\
\Phi_n
\end{pmatrix}
=
\begin{pmatrix}
4\pi G dx^2 \rho_1 \\
4\pi G dx^2 \rho_2 \\
4\pi G dx^2 \rho_3 \\
4\pi G dx^2 \rho_n
\end{pmatrix}
\]

Because it is linear it can usually be solved efficiently in a grid using Fourier transform. However, when the mesh structure is complex, this is not a good strategy.

Gravity in RAMSES is used on the grid using iterative methods.

Two methods have been developed:
- conjugate gradient
- multi-grid (relaxion method)
The conjugate gradient method

Generic method to solve the system $Ax=b$ when $A$ is symmetrical.

Since $A$, is symmetrical, there is a basis of normal eigenvectors.

$$p_i^T Ap_j = 0, \ i \neq j$$

$$x = \sum_{i=1}^{n} \alpha_i p_i$$

$$=> Ax = \sum_{i=1}^{n} \alpha_i Ap_i$$

$$p_k^T Ax = \sum_{i=1}^{n} \alpha_i p_k^T Ap_i = \alpha_k p_k^T Ap_k$$

$$=> \alpha_k = \frac{p_k^T Ax}{p_k^T Ap_k} = \frac{p_k^T b}{p_k^T Ap_k}$$

If the eigenvectors of $A$ are known, $x$ is easily found.
However, this is quite a difficult task to get them all since $A$ is a very big matrix.

The idea of the method is to find a good approximation by considering only a few of them.
This leads to an iterative procedure.
The conjugate gradient method

The idea is then to try minimizing the quantity: \[ f(x) = \frac{1}{2}x^T Ax - x^T b \]

Indeed the gradient of \( f \) is \( Ax-b \). Thus a minimum of \( f \) corresponds to a solution.

Let us start with some guess \( x_0 \). We can calculate the residual:

\[ r_0 = b - Ax_0 \] which is the gradient of \( f \) in \( x_0 \). More generally if we have an approximation \( x_k \), the residual is simply \[ r_k = b - Ax_k \]

If we were trying to use to minimize the system directly along the gradient, the algorithm would be very inefficient. Instead one can construct a new direction that will be orthogonal. It can be shown that the vectors

\[ p_k = r_k - \sum_{i<k} \frac{p_i^T Ar_k}{p_i^T Ap_i} p_i \]

are all orthogonal to each other and thus provide an efficient set of direction for minimisation.
The new value of $x$ is minimised along $p_k$. It is going to be efficient because the direction along which it is done is perpendicular to all the previous directions along which the minimisation has been done.

$$x_{k+1} = x_k + \alpha_k p_k$$

The optimal choice turns out to be:

$$\alpha_k = \frac{p_k^T b}{p_k^T A p_k}$$

This provides an efficient minimisation algorithm not too difficult to implement. In particular, there is no need to store large matrix.

The gradient conjugate method can be used to solve second order terms in other equations (e.g. heat and radiative diffusion).