CfCA Users' Meeting
2017/11/29

# Development of a Full-Multigrid Gravity Solver for Athena++

**Kengo TOMIDA**

(Osaka University)

# Features of Athena++

| Already available | Not public yet | Being implemented | Planned |
|---|---|---|---|
| HD/MHD | Non-ideal MHD (Ohmic, Hall, AD) | Particles (star / dust) | General EOS |
| Curvilinear coordinate | Radiation transfer (Direct ray tracing) | Self-gravity (Multigrid on SMR/AMR) | Post-processing radiation transfer (ALMA Science Proj.) |
| SMR/AMR | Self-gravity (FFT on uniform grid) | Heterogeneous parallelization | Hybrid PIC Plasma |
| Special Relativity | Chemical reactions | Developer's guide | Radiation transfer (VTEF+implicit etc.) |
| General Relativity (Fixed metric) | Shearing Box | Code paper | Full General Relativity (dynamic metric) |
| MPI + OpenMP | 4th-order scheme | | |
| Parallel IO (MPI/HDF) | Self-gravity (Multigrid on uniform grid) | | |
| User-defined functions | | | |
| Support for Intel, GCC, IBM, Cray, incl. KNL | | | |
| Website / tutorial | | | |

赤字: Features (being) developed at Osaka
青字: Osaka is involved

# Development of Multigrid Solver

Some physics requires solutions to global/implicit PDE.

- Self-gravity
- Radiation transfer (FLD, Variable Eddington Factor)
- Conduction, Diffusion, Viscosity

We need a good parallel elliptic/parabolic PDE solver

- fast and scalable
- robust and accurate
- compatible with AMR
- flexible and versatile → C++ derived classes

But how “fast” should it be?

→ at least faster than the MHD part.

# Hyperbolic vs Ellipitical Eqs

**Hydrodynamics:** $$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F} = 0$$

Complicated, but well established (Riemann solvers).
Information propagates at characteristic speeds
→ Finite Volume Method with local explicit update works well.

**Gravity:** $$\nabla^2 \varphi = 4\pi G \rho$$

This is very simple, but one of the worst equations numerically.

- Information propagates instantaneously

→ Global, consistent solution is required

- Boundary conditions matter

→ Physically consistent boundaries are required

- Computationally cheap - but memory / network intensive

→ Difficult to achieve good performance / scalability

# Poisson Solvers

Discretized equation (2D, NxN cells):

$$\frac{\varphi_{j,i+1} - 2\varphi_{j,i} + \varphi_{j,i-1}}{\Delta x^2} + \frac{\varphi_{j+1,i} - 2\varphi_{j,i} + \varphi_{j-1,i}}{\Delta y^2} = 4\pi G \rho_{j,i}$$

$$A\boldsymbol{\varphi} = \boldsymbol{\rho}$$

It is hopeless to solve this equation directly: $O(N^6)$ ($N^9$ in 3D).

Common numerical methods:

- Fast Fourier Transform (already implemented on Athena++)
  + Deterministic (i.e. require only one sweep) and robust
  + Computationally efficient
  - Efficient only on uniform and $2^N$ cells with periodic boundaries
  - Require global communication including global transpose
  - **O(NlogN)** - logN is actually not a small factor
- Tree solver (e.g. Wünsh et al. 2017 for FLASH)
- **Iterative solvers**

# Basic Iterative Solvers

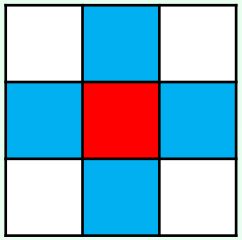
$$A\boldsymbol{\varphi} = \boldsymbol{\rho} \rightarrow (L + D + U)\boldsymbol{\varphi} = \boldsymbol{\rho}$$

D = diagonal, L = lower triangle, U = upper triangle

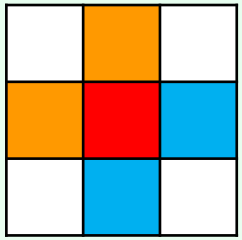
Jacobi iteration: $\boldsymbol{\varphi}^{k+1} = D^{-1}\{\boldsymbol{\rho} - (L + U)\boldsymbol{\varphi}^{k}\}$

Gauss-Seidel iteration: $\boldsymbol{\varphi}^{k+1} = D^{-1}\{\boldsymbol{\rho} - L\boldsymbol{\varphi}^{k+1} - U\boldsymbol{\varphi}^{k}\}$

Jacobi:
Update a **target cell**
using **old** information

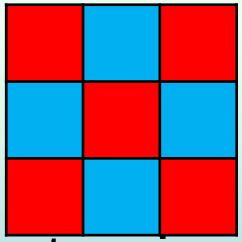


Gauss-Seidel:
Use **updated cells**
and **old** information

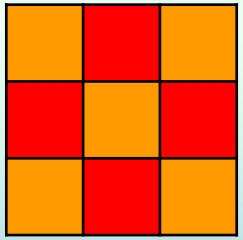


Improvement: Red-Black Gauss-Seidel iteration

Step1:
Update **red cells**
using **old** information



Step 2:
Update **red cells**
using **updated cells**



⇒ Less dependent, isotropic scheme (but still slow)

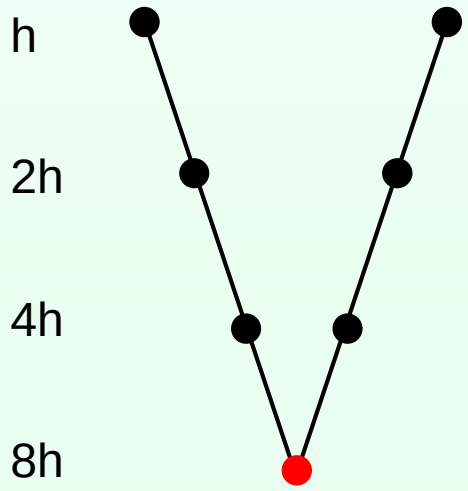
# Multigrid Concept

Iterative solvers = essentially **smoothing** (diffusion) of residual
Diffusion "timescale" $\tau \sim \lambda^2 / D \rightarrow$ small-scale noises diffuse faster
$\Rightarrow$ Accelerate convergence by applying different resolutions

2-level algorithm:
1. Coarsen (restrict) the grid to the coarse level
2. Apply diffusion (RBGS smoothing) on the coarse level
3. Project (prolongate) the result onto the fine level
4. Apply diffusion (RBGS smoothing) on the fine level
5. Repeat until the solution converges

Configuration of our Multigrid:
V(1,1) Cycle = 1 smoothing before restriction, 1 after prolongation
Restriction: volume-weighted average
Prolongation: trilinear or triquadratic interpolation

# Multigrid Poisson Solver




- Red-Black Gauss-Seidel Smoothing
- \ Restriction
- / Prolongation
- Solve Coarsest Level (exact or iterative)

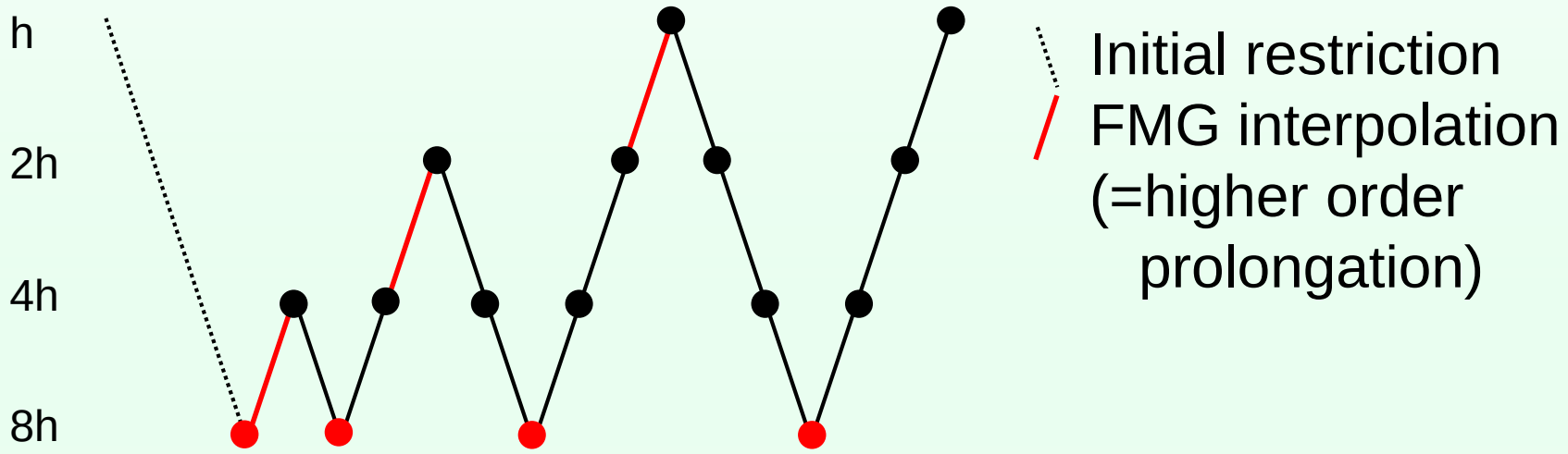
Note: # of communications
GSRB: 2 boundary communications
Restriction/Prolongation: 1 communication

V-Cycle MG solver can be applied as an iterative solver

- Reduces error in all the wavelengths
- Computationally efficient: O(N) (but O(NlogN) if communication dominates)
- Typically one sweep reduces the error by a factor of ~10
- Need a good initial guess to get fast convergence
- Memory and network intensive

(especially latency as it requires a lot of small messages)

# Full Multigrid Method

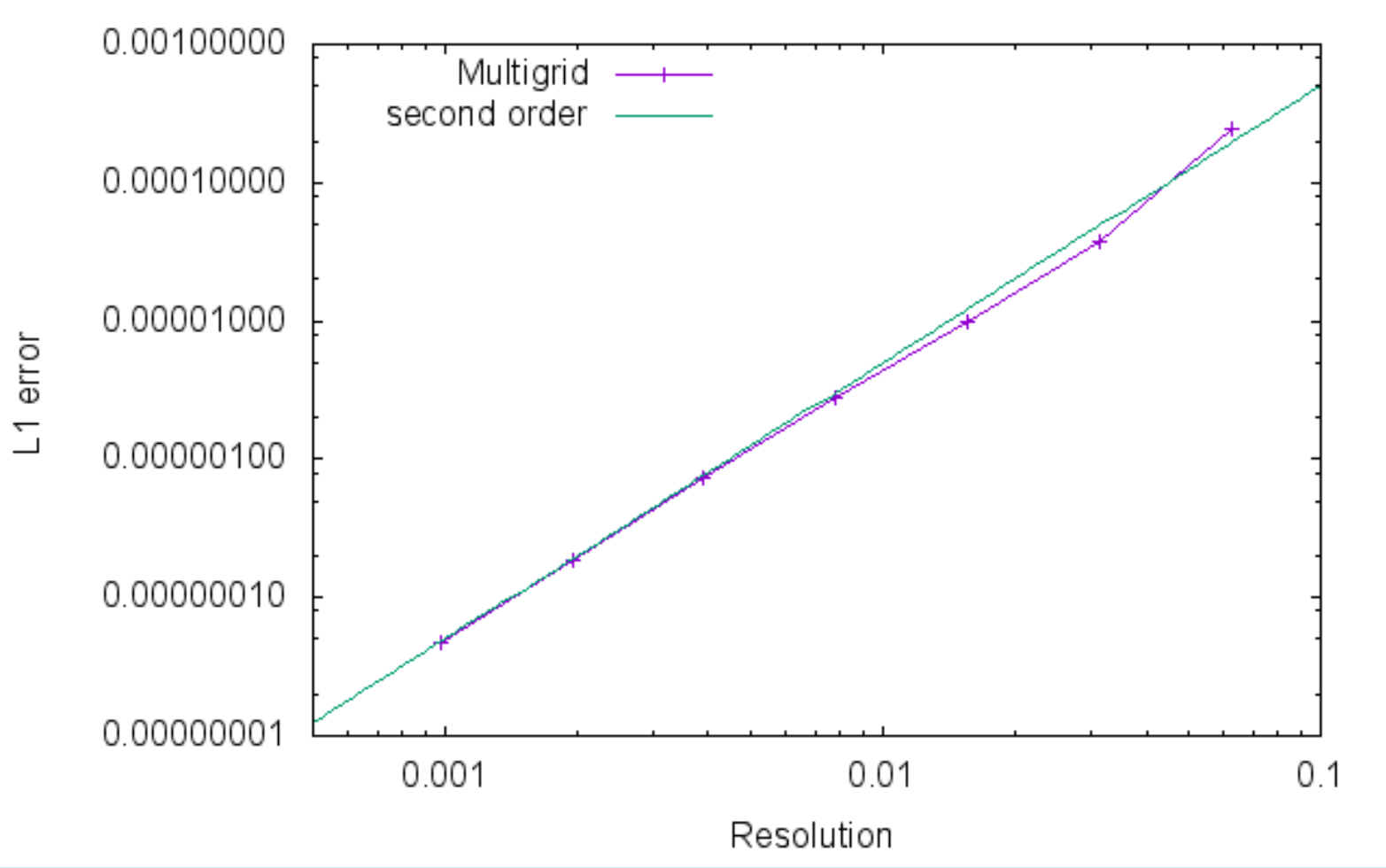



FMG solver: starting from the coarsest level, and use the result of the V-cycle MG solver as the initial guess on the finer grid.

- One FMG sweep is sufficient to achieve ~ discretization error
- Extremely efficient: cost ~ 2 full V-cycles
- Higher order interpolation is needed

(trilinear for normal prolongation, triquadratic for FMG)

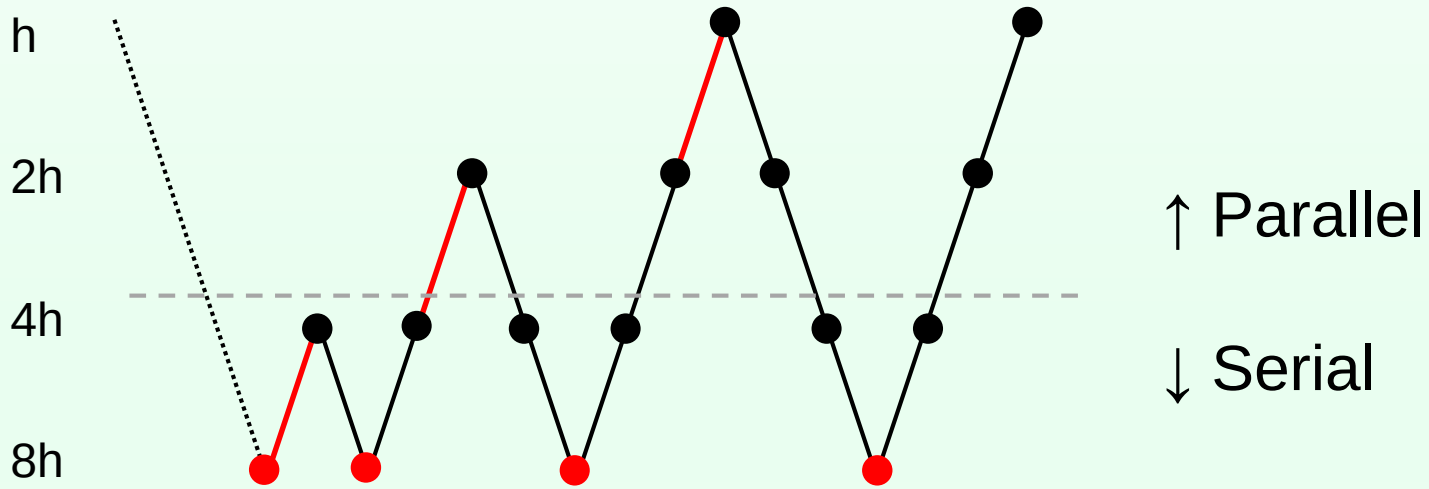
# FMG Convergence




3D sinusoidal waves, $16^3 \rightarrow 1024^3$ / wavelength

Second-order convergence is achieved with one FMG sweep.

# Parallel Multigrid

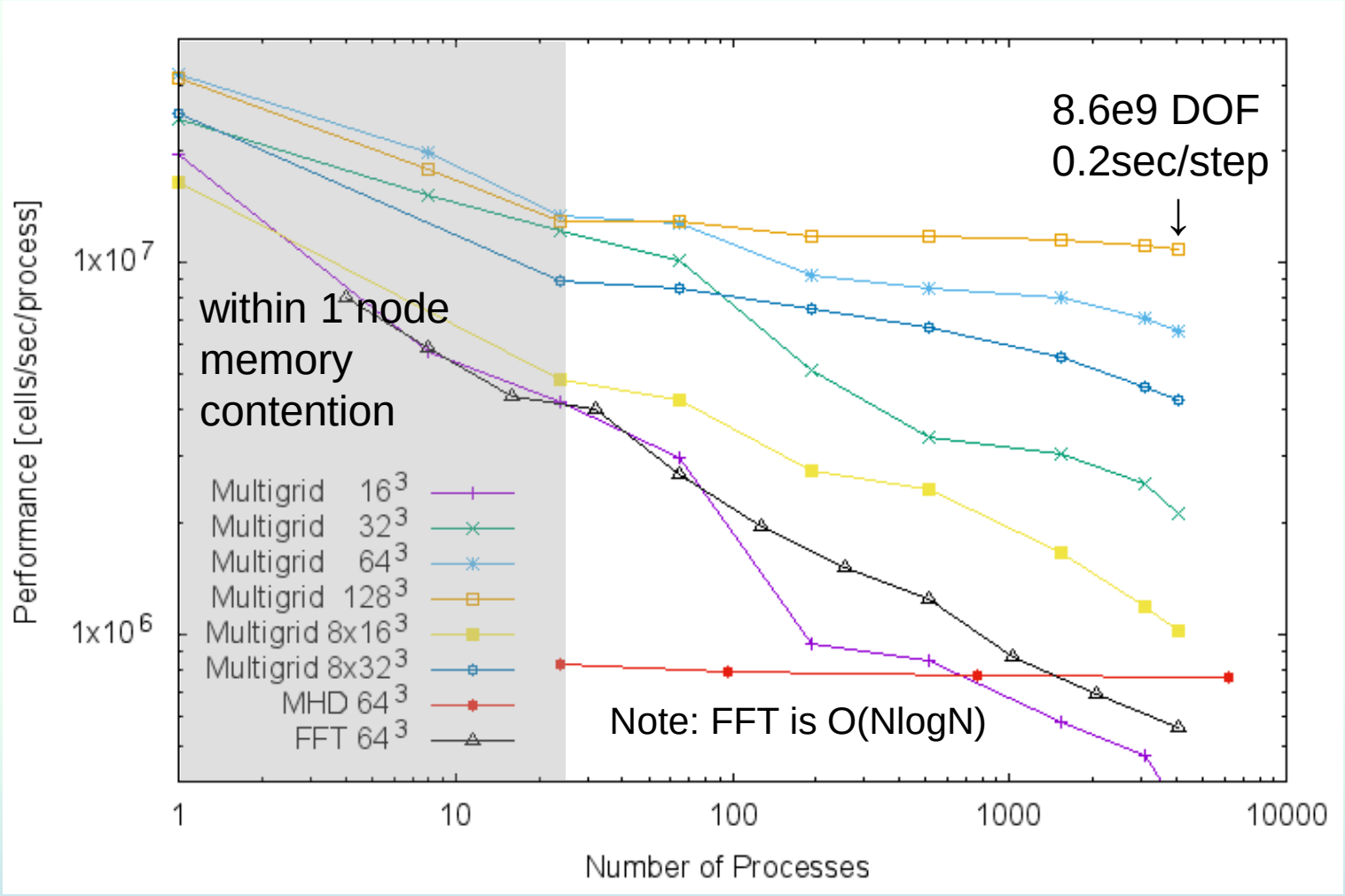



In Athena++, the domain is decomposed into MeshBlocks.
At somepoint, a MeshBlocks is derefined into 1x1x1 cell
→ collect data using MPI_Allgather and apply Multigrid,

Gauss-Seidel smoother is cheap → network latency matters
Using Athena++'s dynamic scheduling with TaskList, we interleave (overlap) communication and computation
(This works only with more than one MeshBlocks/process)

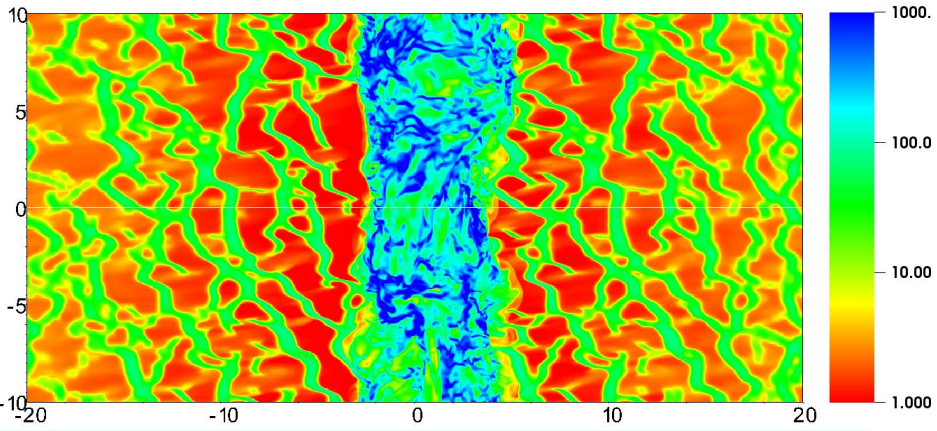
# Performance

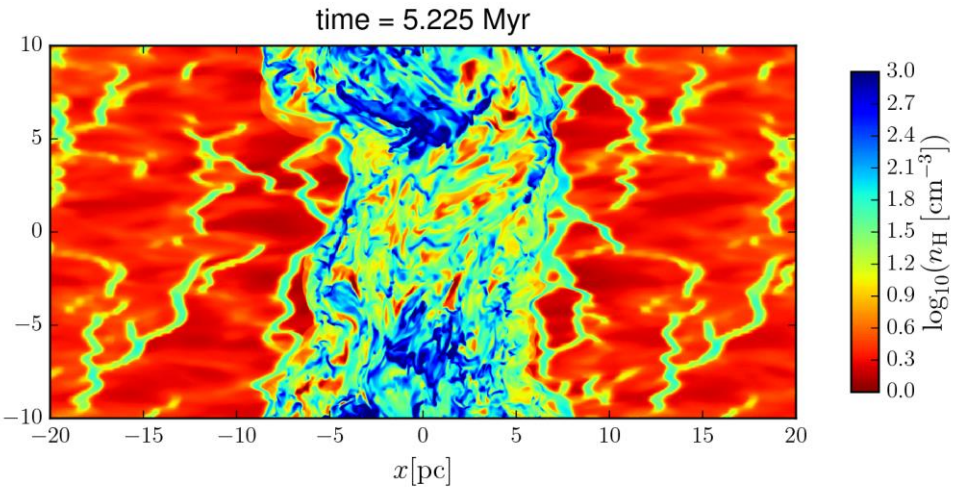
Cray XC30 @ NAOJ
Flat MPI




Multigrid performs sufficiently good at least up to 4096 cores
Small MeshBlocks are slow, but it is OK with many MeshBlocks

# Molecular Cloud Formation




Left: with self-gravity Right: without self-gravity

Caution: this is not a fair comparison - the density is twice higher.

1024x512x512, <n>=10 or 5 / cc, v=20km/s, B=5μG, θ=11°

After ~5Myr the maximum density reaches >$10^6$/cc and collapses.

The accumulated mass is so large that self-gravity is significant.

→ to follow formation of cloud cores, we need SMR / AMR.

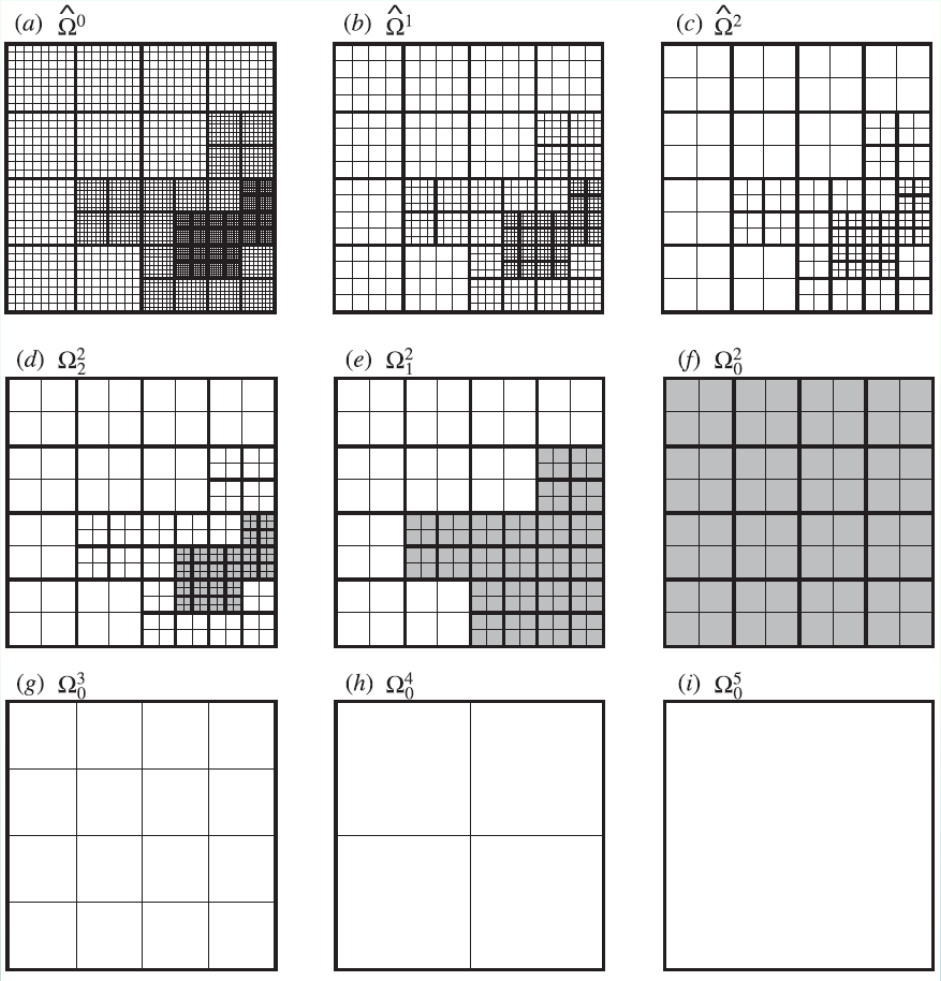
(see Iwasaki-san's talk for details) **Only 20% additional cost!**

# Next Step: SMR/AMR

Basically, we will follow SFUMATO's approach→

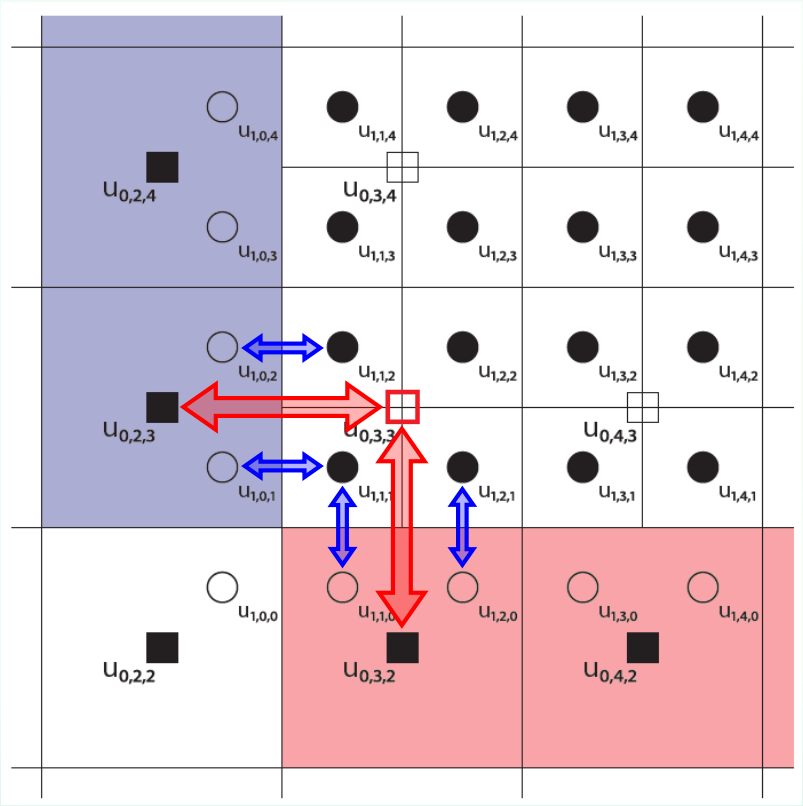
On SMR/AMR grids, the Full Approximation Scheme (FAS) must be adopted.

Mass conservation formula (Feng et al., JCoPh, 2017, 352, 463) consistently discretize the level boundary so that the fluxes on the coarse and fine levels match → no need of correction, possibly improve convergence




(Matsumoto 2007, SFUMATO)

# Mass Conservation Formula




(Feng et al. JCoPh, 2017, 352, 463)

■ : Coarse active cells

● : Fine active cells

○ : Ghost cells

□ : Zombie cells

Ghost cells ○ are set by quadratic interpolation using active cells ■ ●. Zombie cells are calculated so that the fluxes between levels match.

Note that the double zombie cell □ has two different values – but it is OK as this cell appears only as a boundary value between the levels.

# Summary

Multigrid works!

- Full Multi Grid method works very well
- Achieved sufficiently high performance
- Flexible interface using derived classes: can be applied to other physics such as radiation transfer
- Heterogeneous parallelization will improve performance

Athena++

- Planning next public release soon
- Non-ideal MHD, shearing box, chemistry, higher order etc…
- Even more physics: gravity, full GR, radiation, …
- Method paper in progress