Current Status of the Athena++ code

Kengo TOMIDA (Princeton)

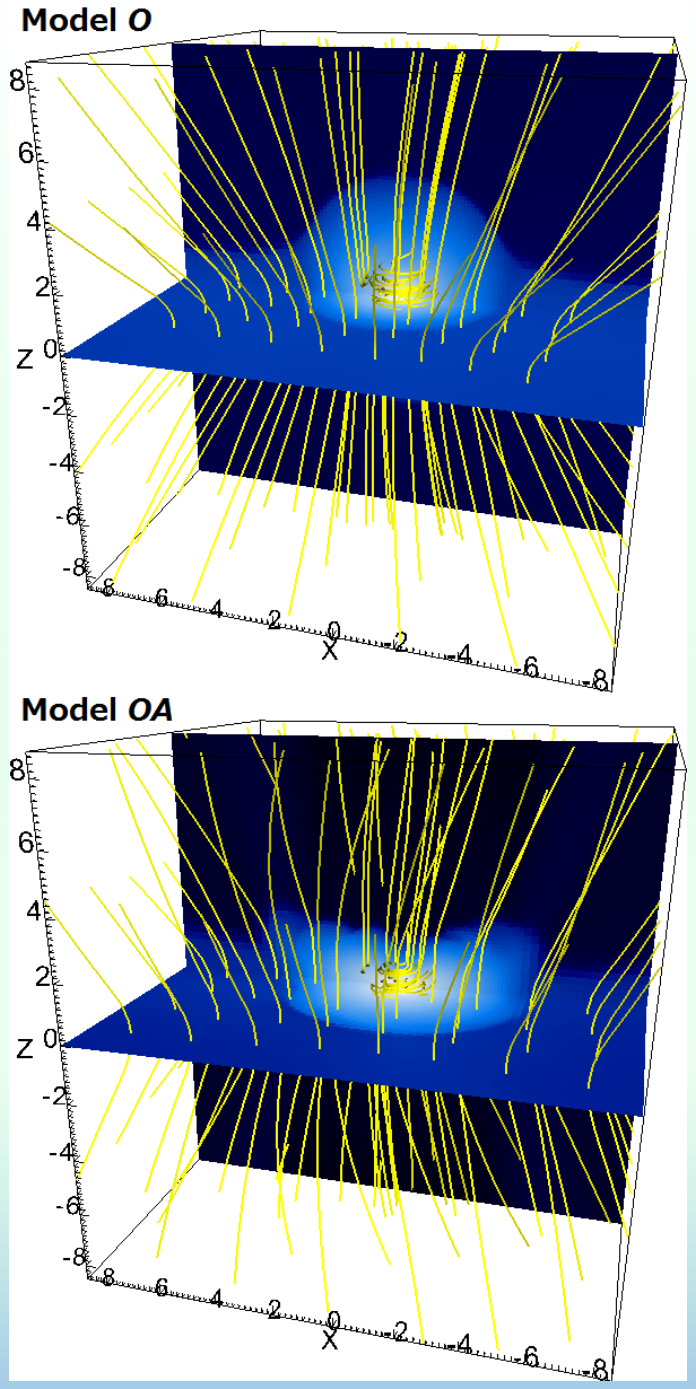
James Stone (Princeton)
and
The Athena++ Development Team

Before Starting…

Tomida, Okuzumi & Machida 2015
accepted by ApJ, arXiv: 1501.04102

3D nested-grid RMHD simulations of protostellar collapse with ambipolar diffusion and Ohmic dissipation

Non-ideal MHD effects remove magnetic flux efficiently and enable early disk formation even before a protostar forms. The disk size in the early phase remain small, which is consistent with recent interferometric observations.



The Athena++ Project

2 Theoretical and Computational Astrophysics Networks:

- "Black Hole Accretion" (Princeton, Illinois and UC Berkeley)
- "From the ISM to the IMF: Multi-Scale, Multi-Physics Modeling of Star Formation"

Collaboration between Princeton, UC Berkeley and Santa Cruz
i.e. collaboration between Athena and Orion

(Scientific) Goal: study the origin of core/initial mass function using multi-scale simulations covering from galactic scale to cloud scale

(Computational) Goal: develop useful, efficient, scalable multi-physics radiation MHD code with adaptive mesh refinement (AMR)

→We are now developing the Athena++ code from scratch

Why Writing It from Scratch?

Problems of Athena = old design :

- Not flexible grid configuration: uniform mesh spacing only
- The **Array-of-Structure** data structure is not suitable for vectorization (Intel AVX = 4 double) ⇒ **Structure-of-Array**
- Many contributions → difficulty in maintenance
- **No AMR**: only static mesh refinement (SMR)
- Flat MPI parallelization

Note that, however, Athena was quite successful and efficient. For example, for uniform grid ideal MHD on IBM Blue Gene/Q, Athena (lightly tuned) achieved 128,000 cells / sec / core and about 12% of the theoretical peak performance

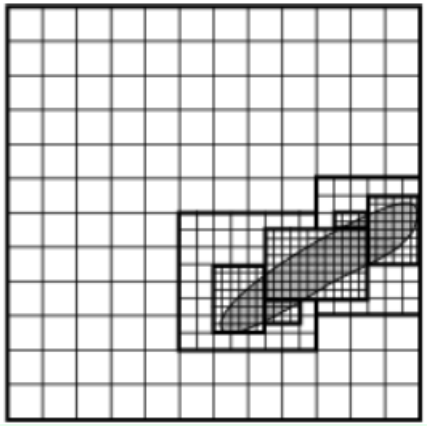
(lightly tuned = optimized by myself)

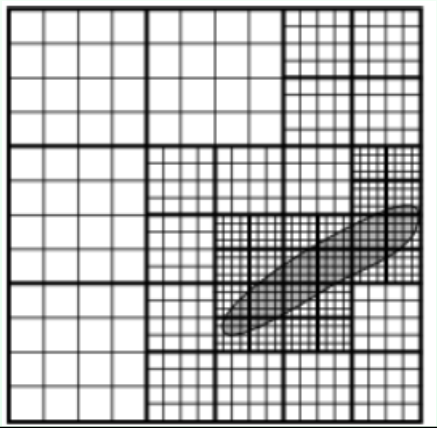
Design Policy

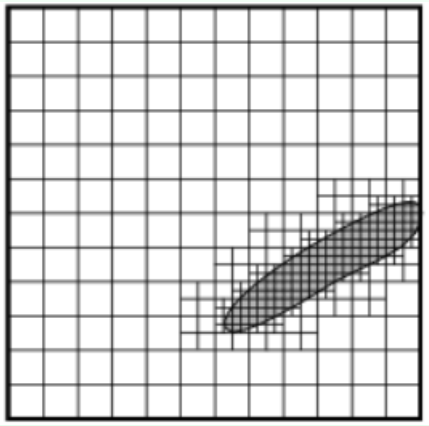
- Conform the industry standards (C++11, MPI-2 or 3, etc.)
- No reliance on external libraries as much as possible
- Support many systems (Intel, GNU, Cray, IBM, and Xeon Phi)
- **Hybrid parallelization with MPI and OpenMP**
- Support parallel IO (MPI-IO) as well as conventional IO
- **Performance** and **clarity** are the top priorities
- Flexible coordinate systems, including SMR and **AMR**
- Support standard MHD algorithms based on Athena
  - PLM, PPM (MP5?)
  - Constrained Transport
  - HLLE, HLLC, HLLD, Roe
  - 2nd and 3rd order time integrators
- Many physics modules
  - Self-gravity
  - Radiation Transfer
  - Chemistry
  - Particles (dust / sink)
  - Non-ideal MHD
  - SR / GR (fixed metric at first)

(Figures are from Matsumoto -san's presentation, thanks!)

AMR Design: Grid Structure

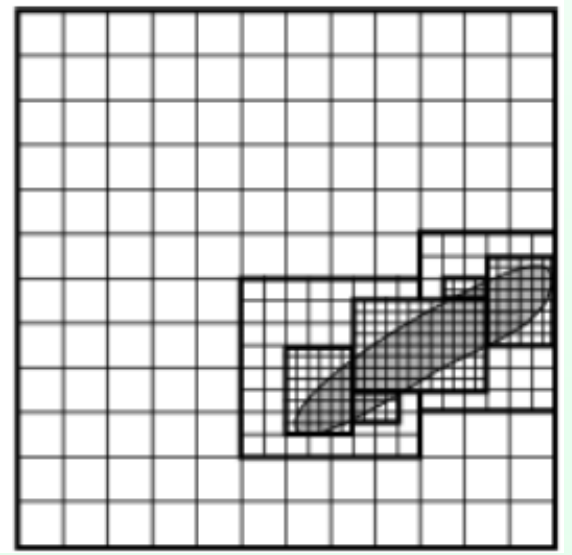


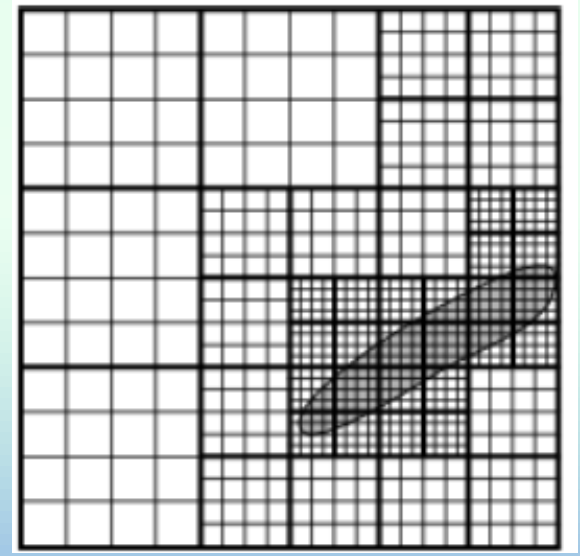




	(A) Block (Patch) Based	(B) Oct-Tree-Block Based	(C) Cell(Tree)-Based
Pros	High adaptivity **Uniform within block** **Use of existing scheme**	**Simple relation btw blocks** **Uniform within block** **Use of existing scheme** Parallelization by space-filling curve	Highest adaptivity Logically beautiful Parallelization by space-filling curve
Cons	Grids are not unique **Non-trivial grid generation** **Complex parallelization**	Lower adaptivity (depending on patch size)	**Performance Issue** **Complicated grids** **(non-trivial neighbor cell)** **Hard to write,read,analyze**
Examples	Original: Berger & Colella 1989 Orion, PLUTO(Chombo), Enzo, **Athena SMR**,…	FLASH(PARAMESH) Nirvana, SFUMATO,…	RAMSES, ART

AMR Design: Overlapping Blocks





Q: Should we solve the coarse region overlapping with the finer block?

For type A, yes, it makes sense because otherwise there is a “hole” in the coarse grid.

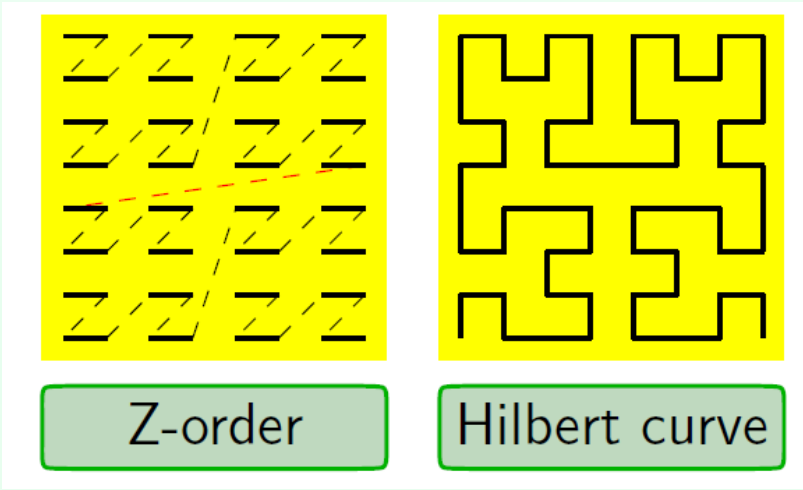
For type B, no, but many codes do. Typical argument: the overlapping region is small, so it is not expensive to solve it.

Wrong. Restriction operation requires MPI communications of large data, and it has dependency on finer levels, therefore it will cause poor performance. Definitely **NO**.

The figure is taken from presentation by Herman Haverkort
http://www.win.tue.nl/~hermanh/stack/dagstuhl08-talk.pdf

AMR Design

Parallelization and load balancing:
Use a self-similar space-filling curve over all the grid levels.
Currently Z-ordering is implemented.
This ordering is also used to create unique IDs to identify a block.
Hilbert-like curves will be considered in the future, but they can be costly.



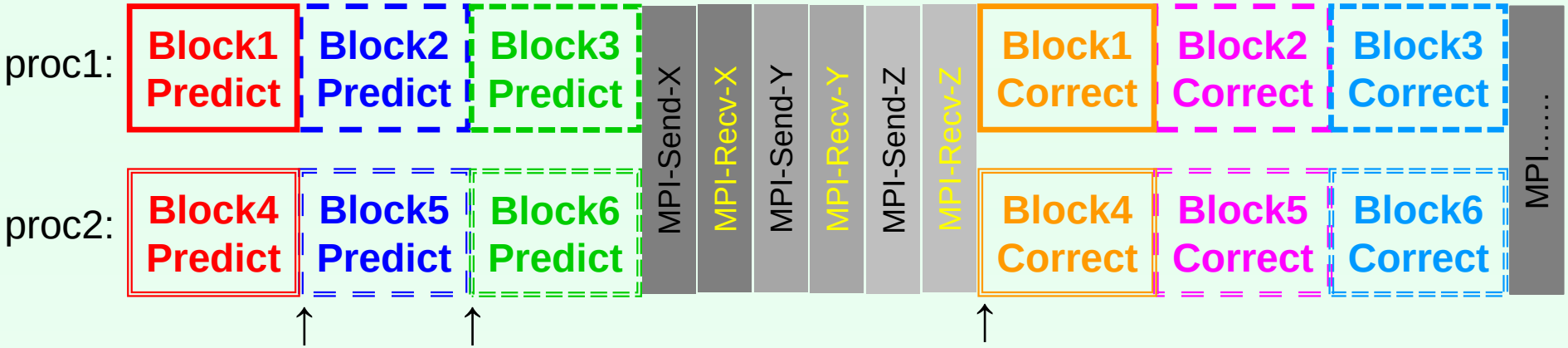
Time Stepping:
Currently we focus on the shared time stepping.
Adaptive time stepping is useful only for explicit schemes, and there will be a problem in load balancing.

Input and Output:
MPI IO is implemented for restarting, which creates only one file.
HDF5 is being considered for analysis.

Computation / Communication Overlap: Dynamic Scheduling

In order to get the maximum performance, we want to hide MPI communication behind computation. But how?

Typical integration scheme is…



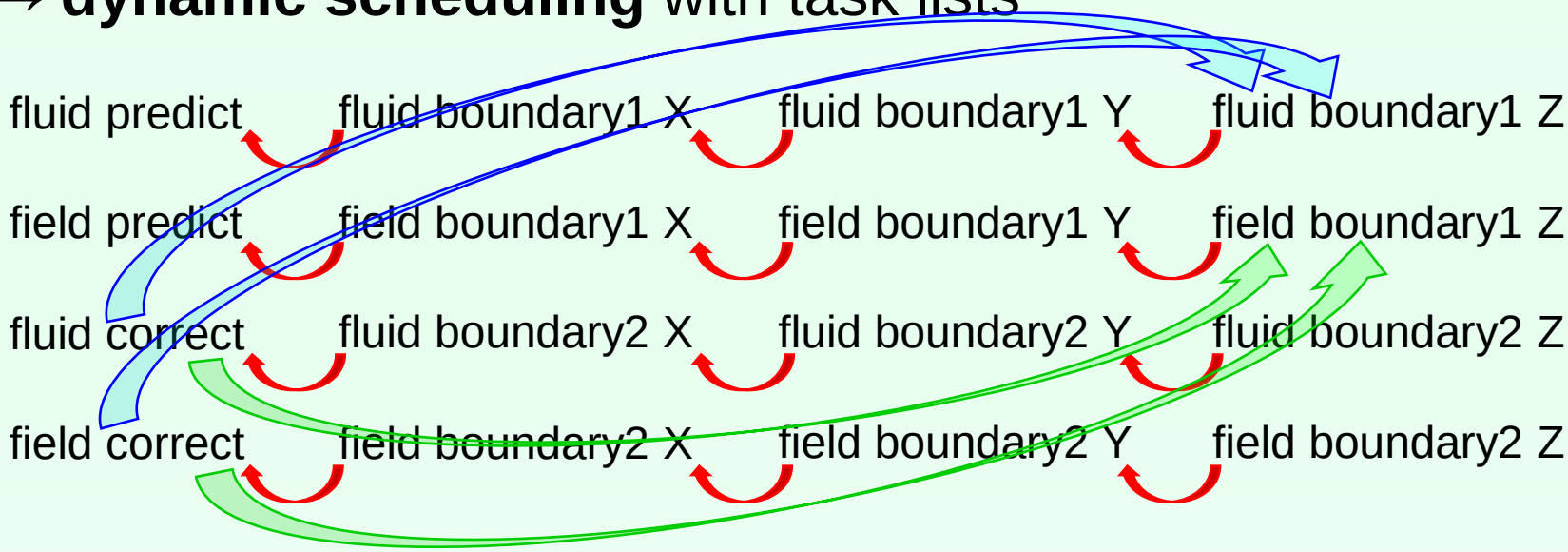
We can start communications.

We do not have to wait all the communications here.
But the boundaries do not necessarily arrive in this order

Note:
We receive X data before starting Y.

Computation / Communication Overlap: Dynamic Scheduling

So, we should not fix the ordering of the blocks and tasks
→ **dynamic scheduling** with task lists



If there is a task in the list whose dependency is clear, do it.
If not, go to the next block, and loop until all the tasks completed.
MPI communications are non-blocking (Isend/Irecv),
MPI_Test() is used to check if the communication is completed.

Inside MPI

We found MPI performance is improved by calling **MPI_Iprobe**.
How does this work? (Caution: this is implementation dependent)

Even with non-blocking (Isend/Irecv) communications, MPI does NOT necessarily have a background process which transfers the data between memories on different nodes.
Instead, it does some communication tasks when MPI-related functions are called. Calling MPI functions give MPI chances to advance the communication. I guess this works especially when there are multiple messages at the same time.

MPI_Iprobe checks availability of a message, harmless & cheap.

For details, ask me, and read these articles:
http://stackoverflow.com/questions/20999299/why-does-mpi-iprobe-return-false-when-message-has-definitely-been-sent
https://www.xsede.org/documents/271087/586927/Woodward.pdf

Uniform Grid Performance: Serial Performance

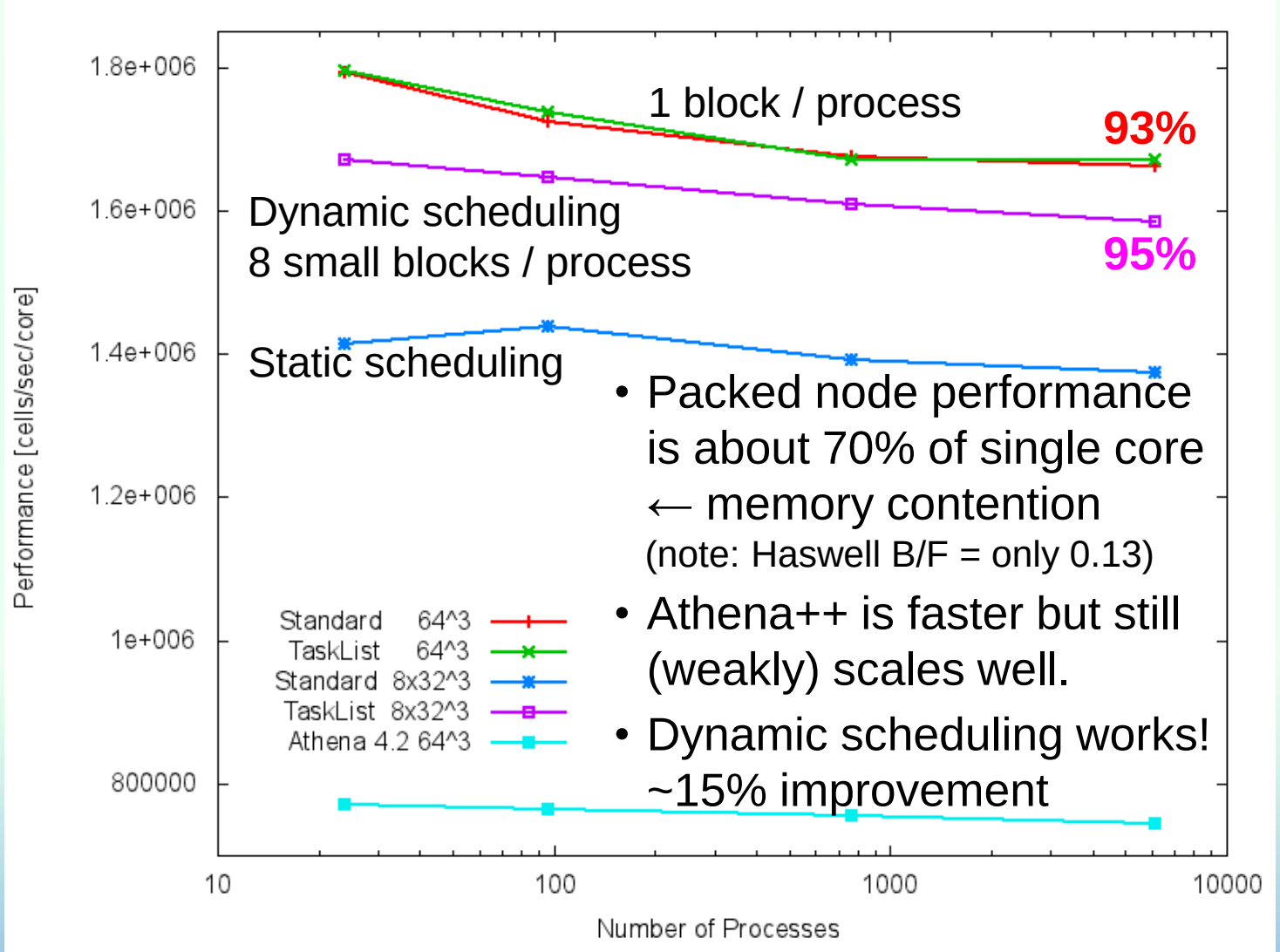
pure hydro , HLLE, 2nd order primitive reconstruction, 64^3 cells
(note: this is the cheapest configuration)

IvyBridge Xeon E5-2670v2, 2.5GHz, 10 cores, DDR3 memory
GNU (-O3) : 1.16e6 cells / sec / core
Intel (-ipo -fast) : **1.41e6** cells / sec / core

XC30, Haswell Xeon E5-2697v3, 2.6GHz, 12 cores, DDR4
Cray (-O3 -h aggress -h vector3 -hfp3 –hwp) : 1.39e6 cells/s/core
Intel (-ipo -fast) : **2.52e6** cells / sec / core（**!**）

1. Haswell is faster (higher bandwidth, higher AVX throughput).
2. $\gtrsim$ 20% of the theoretical peak performance is achieved.
3. Intel Compiler is fast (maybe better performance with AVX?)
   … or is there any better compiler option for crayCC?

Uniform Grid Performance: (Weak) Scalability (up to 6144 cores)



Current Development Status

Hydro: 99%, almost done, being tested

MHD: 80%, debug and MPI parallelization in progress

OpenMP: 40%, currently only for hydro

AMR: 20%, design fixed, some features implemented

Self-Gravity: 1%, thinking about algorithm

Radiation: 50%, being imported from Athena

GR: 80%, hydro and MHD implemented, but w/ HLLE

Particles: 20%, serial implementation in progress

IO: 50%, table, VTK, parallel restarting done, HDF5?

Chemistry, non-ideal MHD, sink particles: 0%

Future Plan

- Version 1.0 (= fully parallelized uniform-grid MHD) will be ready in one month or so
- Radiation and (fixed metric) GR are being implemented
- I promised that I will write AMR MHD before April 10th.
- Then self-gravity.

- The code will be delivered to collaborators at first
- A limited version (equivalent to current Athena but faster) will be distributed publicly
- The full version (including new features) will be publicly available later

Summary

Athena++: new AMR RMHD code

- Nothing fundamentally new, but redesigned thoroughly
- Highly optimized AND scalable (target > million cores)
- Many physics & various applications will be supported
- Some new techniques including dynamic scheduling
- Oct-Tree-Block based AMR without overlapping

A few remarks for XC30 users:

- Benchmark your code (not only scalability)
- Try Intel Compiler; it can be faster
- Choose or develop your code wisely