Accelerate MHD

Bijia PANG

Department of Physics, University of Toronto

Outline

- Magneto-hydrodynamics
- Physics problem (video)
- The algorithm of solving MHD
- Implementation on heterogeneous system
- CUDA on a mini GPU cluster
- Summary

Magneto-hydrodynamcs (MHD)

$$\frac{\partial \rho}{\partial t} + \nabla(\rho v) = 0$$ Mass conservation

$$\frac{\partial v}{\partial t} + (v \bullet \nabla) v = -\frac{1}{\rho}\nabla P - \frac{1}{4\pi\rho} B \times (\nabla \times B)$$ Momentum conservation

$$\frac{\partial}{\partial t}(\frac{1}{2}\rho v^2 + \rho\varepsilon + \frac{B^2}{8\pi}) = -\nabla(\rho v(\frac{1}{2}v^2 + \varepsilon + \frac{P}{\rho}) + \frac{B \times (v \times B)}{4\pi})$$ Energy conservation

$$\partial_t \vec{b} = \nabla \times (\vec{v} \times \vec{b})$$ Induction equation

$$\nabla \bullet \vec{b} = 0$$ No magnetic monopole

$$P = P(\rho, T)$$ Gas: equation of state

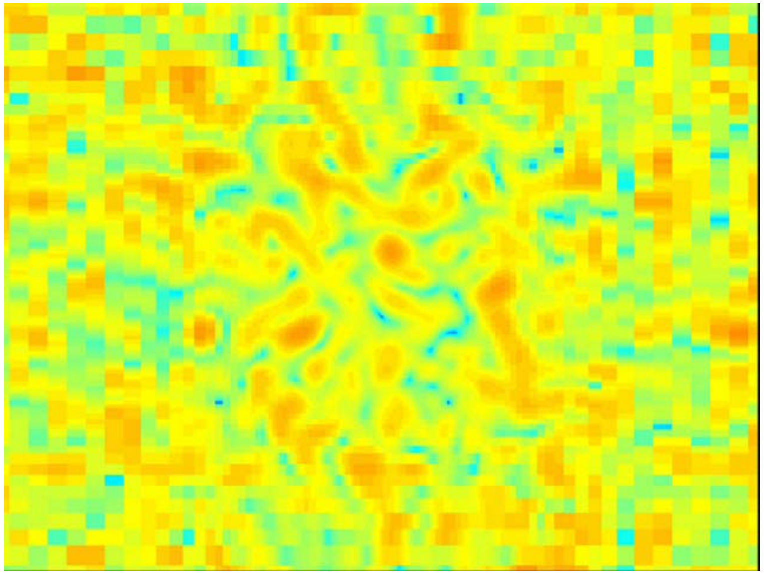
- Fluid dynamics under magnetic field

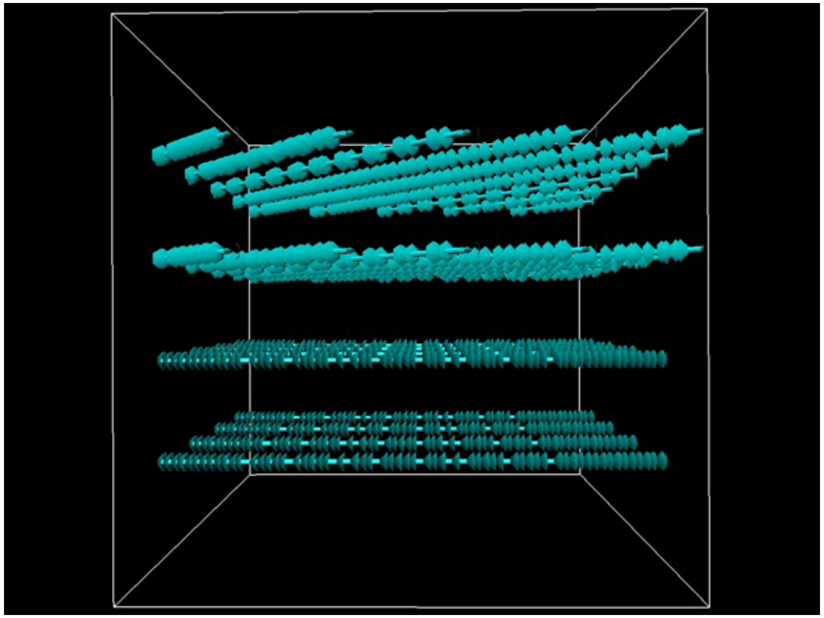
Landau & Lifshitz

MHD code

- The code was written by Ue-li Pen in 2003, and was expanded by Phil Arras, ShingKwong Wong, Hugh Merz, Matthias Liebendoerfer, Stephen Green, Bijia Pang.
- The code is a second-order accurate (in space and time) high-resolution total variation diminishing (TVD) MHD parallized code.
- Kinetic, thermal, and magnetic energy are conserved and divergent of magnetic field was kept to zero by flux constrained transport.
- The code is short and simple, easy for GPU acceleration.
- Three groups, H. Wong (arXiv:0908.4362), and H.-Y. Schive (arXiv:0907.3390), and B. Pang (arXiv:1004.1680) programmed it using CUDA on Nvidia GPU.

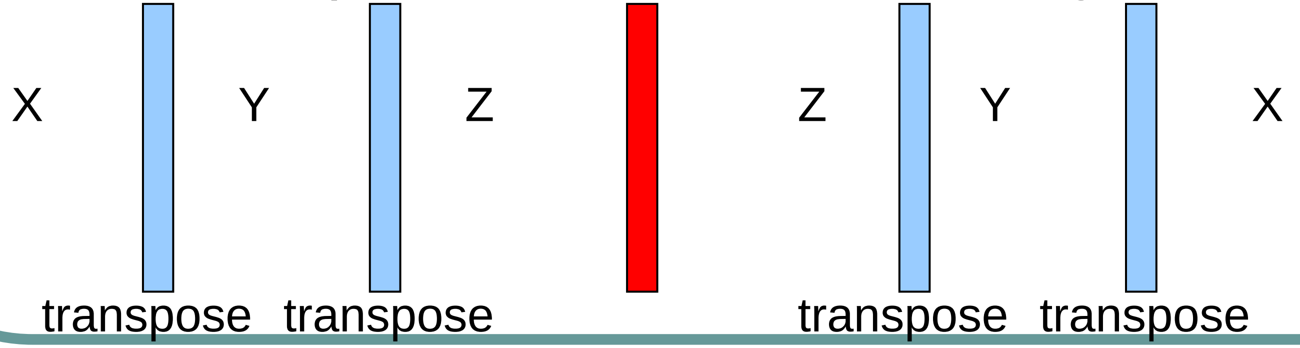
Simulation video





Algorithm of MHD code

- Finite difference + finite volume + time dependent
- u(5) fluid variable, stored on center
- b(3) magnetic variable, stored on cell face
- Second-order total variation diminishing scheme
- Dimension split for 3D box
- Fluid and magnetic update separately (1D)
- Matrix transpose is used for coalescing memory



Algorithm – fluid & magnetic

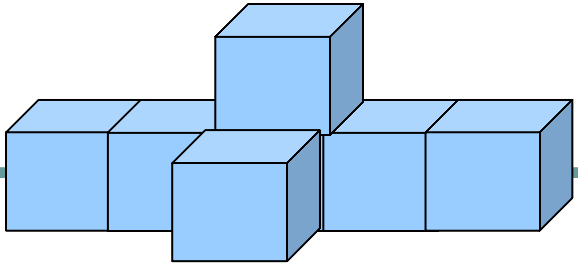
- Fluid: 1D advection equation

$$\partial_t \vec{u} + \nabla_x \vec{F} = 0$$

- Magnetic: 2D advection-constraint step. The same electro-motive force is used immediately in the constraint step to preserve zero divergent of magnetic field

$$\partial_t \vec{b} = \nabla \times (\vec{v} \times \vec{b}) \qquad \nabla \bullet \vec{b} = 0$$

Grid dependence



Pen, Arras, & Wong 2003

Heterogeneous platform

- Controlling processor + computing processors:

- CELL: Power Processor Element (PPE) + Synergistic Processing Elements (SPE)

  supported by IBM & MITACS
- GPU: CPU + unified shaders

- CELL: CELL SDK
- Nvidia: Compute Unified Device Architecture (CUDA)
- ATI: Open Computing Language (OpenCL)

The code is suited for acceleration

- The code is simple
- One dimensional update (same operation for every grid)
- Linear memory-access patterns (data transfer)

Results on different platforms

128^3 box on one CELL/GPU

Single-precision & milli-second

Architecture	x86(1)	x86(8)	Cell	N-GPU	A-GPU
Respective time	8770	1315	864	64	128
OpenCL time	N/A	6435	N/A	65	128
Peak Gflops	17	136	409.6	1030	2720
Peak GB/s	19.2	19.2	204.8	144	153.6
Power(Watts)	N/A	170	440	550	360
Code speed-up	1.0	6.7	10.2	137	68.5
Fractional speed-up	1.0	0.83	0.42	2.0	0.43
FLOPS fraction	3.1%	2.6%	1.3%	7.0%	1.3%
Bandwidth fraction	1.3%	8.8%	1.3%	24.2%	11.3%

XeonE5506@
2.13GHz

Cell blade Q22

Tesla C2050

HD 5870

Code speed-up: ratio on the platform compared to a single core x86

Fractional speed-up: ratio of code speed-up to theoretical peak performance ratio

FLOPS fraction: ratio of actual FLOPS to theoretical peak performance

Bandwidth fraction: ratio of actual data transfer to theoretical bandwidth(on-chip)

CUDA & openCL On Nvidia GPU

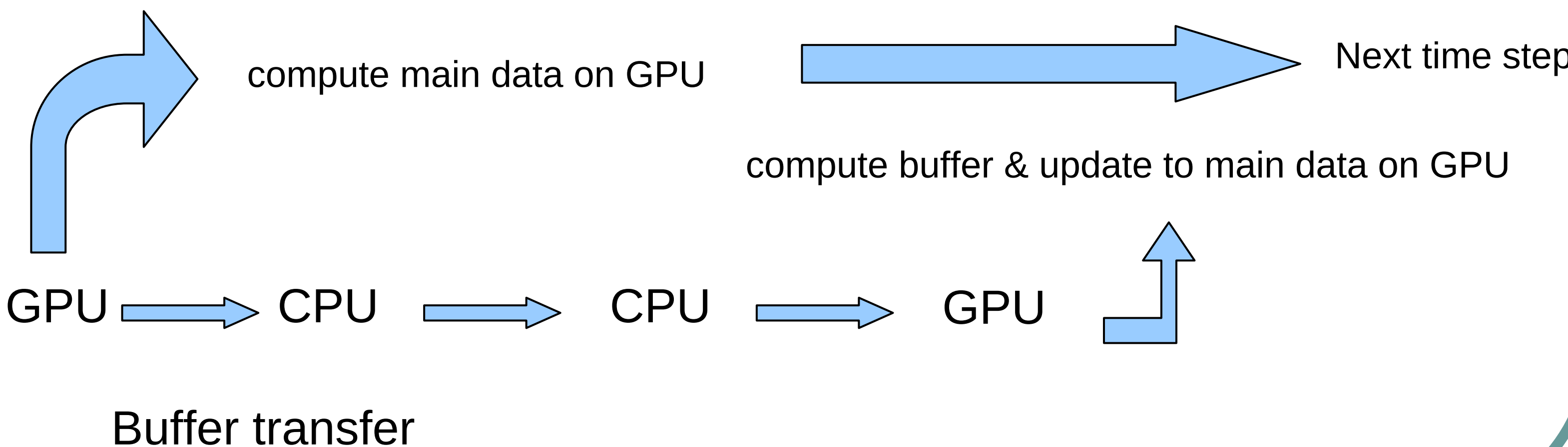
Tesla C2050

	Domain size			
Architecture	16^3	32^3	64^3	128^3
x86(1)	17.8	140	1096	8770
Nvidia (CUDA)	1.3	2.3	8.8	64
Nvidia (OpenCL)	1.5	2.5	9.3	65
Nvidia (CUDA) 2	1.9	3.7	17.9	136
Speedup (CUDA:x86)	13.1	61	125	137

Double precision

Fortran MPI + CUDA

- CUDA for Nvidia GPU
- Overcome low PCI-e bandwidth → let more data stay on GPU, less data for communication
- CPU to GPU (1 to 1)



Result on MPI + CUDA

MPI + cuda vs CPU MPI

speed-up(CUDA:openMP): 8.286/2.449=3.4

single precision (second)

time	cuda(2)	cuda(1)	fortran(2)	fortran(2) omp
two 218^3	2.449	1.489	31.08	8.286

note:
two cube for this table:
cuda(2): 1 GPU on 1 node, 2 nodes together
cuda(1): 2 GPU on 1 node
fortran(2): 1 CPU on 1 node, 2 nodes together, only fortran
fortran(2) omp: 1 CPU on 1 node, 2 nodes together, openMP fortran

lost on communication: 1.330/0.662=2

	cuda(1)	no comm
one 218^3	1.330	0.662

one cube for this table:
cuda(1): 1 GPU on 1 node, including the MPI communication
no comm: 1 GPU on 1 node, no communication

2 Tesla C1060 GPU + MPI

122^3 detail link

Summary

- Heterogeneous system can accelerate: Cell(10x), CUDA(137x), ATI(68x)
- ATI has a good theoretical peak performance, but CUDA on Nvidia perform better. (our openCL code not fully vectorized)
- CUDA & openCL perform the same on C2050
- CUDA + MPI can accelerate(3.4x to openMP)

- Future work: improve CUDA + MPI

Reference:

- H. Wong, U. Wong, X. Feng, and Z. Tang, "Magnetohydrodynamics simulations on graphics processing units," Imprint, 2009
- H.-Y. Schive, Y.-C, Tsai, and T. Chiueh, "GAMER: a GPU-Accelerated Adaptive Mesh Refinement Code for Astrophysics,"Astrophys. J. Suppl., vol. 186, pp. 457-484, 2010
- B. Pang, U. Pen, M. Perrone, "Magnetohydrodynamics on Heterogeneous architectures: a performance comparison," Imprint, 2010

Acknowledgement:

- We would like to thank Jonathan Dursi, Wenda Han, Qi Liu, Harald Pfeiffer, Scott Rostup, Daniele P. Scarpazza for helpful suggestions.
- We thank Scott Rostup, Hsi-Yu Schive, Tomoyoshi Shimobaba for providing their code for our reference, and Hon-Cheng Wong for providing the detail of their simulation results, and Kiyoshi Wesley Masui for reviewing the draft, and Gojko Vujanovic for providing the power consumption measurement.
- We acknowledge IBM TJ Watson Research Center for providing the IBM QS22 Cell blade. Part of Cell computations were performed on the Cell cluster at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto.
- The work of BP is supported by the MITACS ACCELERATE Scholarship.

Thank you!

How to update magnetic

- Find a second-order-accurate, upwind ElectroMotiveForce V_y*B_x
- In the advection step to update b_x and then immediately use the same EMF for the constraint step to update b_y.

$$\partial_t \boldsymbol{b} = \nabla \times (\boldsymbol{v} \times \boldsymbol{b}) ,$$

$$\nabla \cdot \boldsymbol{b} = 0 ,$$

$$\partial_t b_x + \partial_y (v_y b_x) = 0$$

$$\partial_t b_y = \partial_x (v_y b_x)$$

Pen, Arras, & Wong 2003

back

Upwind methods

- Upwind methods take into account the physical nature of the flow when assigning fluxes for the discrete solution.
- Excellent at capturing shocks and also highly stable.

Courant, Isaason, & Reeves (1952)

back

Total variation diminishing

- Nonlinear stability condition
- Overall number of oscillations is bounded
- A strongly nonlinear flux limiter that adds just enough diffusion to prevent numerical instabilities.

$$TV(u^t) = \sum_{i=1}^{N} | u_{i+1}^t - u_i^t | \qquad TV(u^{t+\Delta t}) \le TV(u^t)$$

$$\Delta F_{n+1/2}^t = \phi(\Delta F_{n+1/2}^{L,t}, \Delta F_{n+1/2}^{R,t})$$

Harten (1983)

back

Relaxing system

- Euler equation: momentum and energy fluxes depend on the pressure.
- The flow is considered as a sum of a right-moving wave u_R and a left-moving u_L.

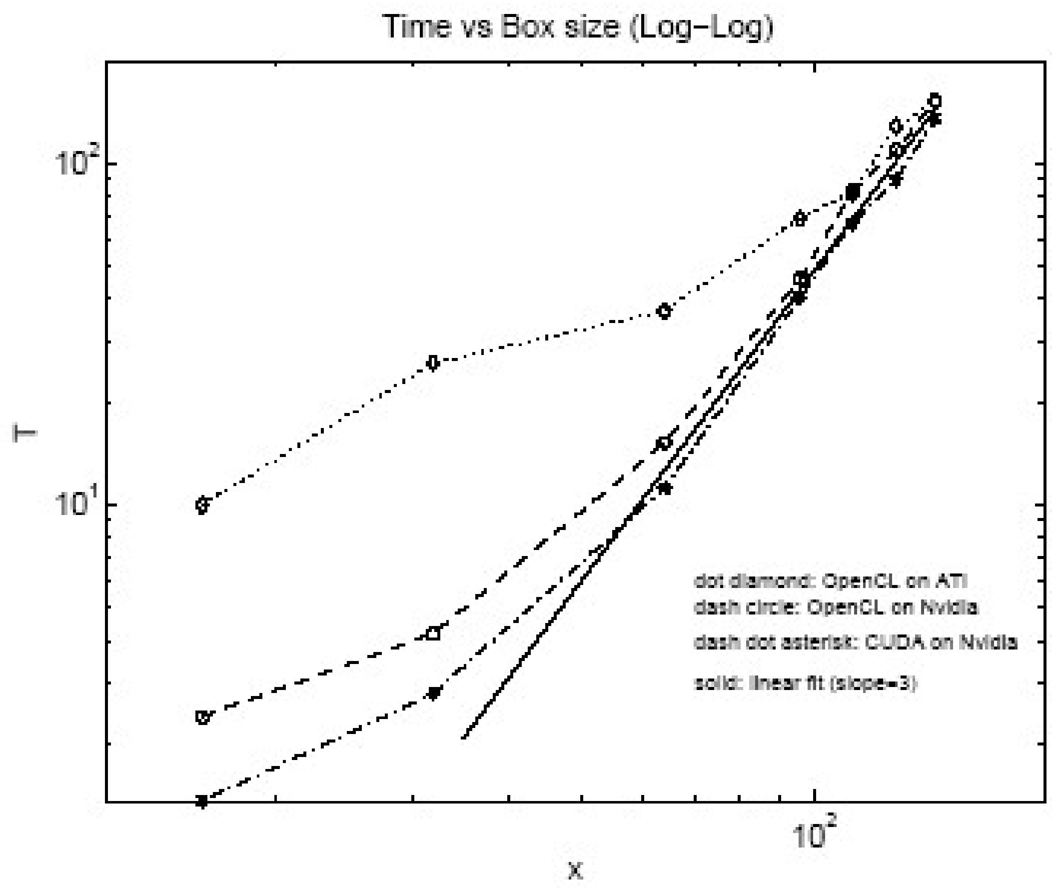
$$u^L = \left(\frac{1 - v/c}{2}\right)u \qquad u^R = \left(\frac{1 + v/c}{2}\right)u$$

$$\frac{\partial u}{\partial t} + \frac{\partial F^R}{\partial x} - \frac{\partial F^L}{\partial x} = 0$$

Jin & Xin 1995

back

Comparison between OpenCL & CUDA



Algorithm – fluid part

- 1D advection equation

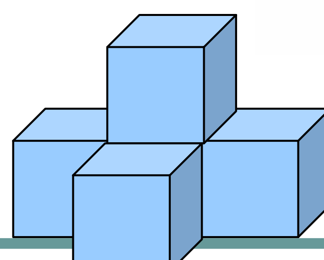
$$\partial_t \vec{u} + \nabla_x \vec{F} = 0$$

$$\boldsymbol{u} = (u_1, u_2, u_3, u_4, u_5)$$

$$(\rho, \rho v_x, \rho v_y, \rho v_z, e)$$

$$\boldsymbol{F} = \begin{pmatrix} \rho v_x \\ \rho v_x^2 + P_* - b_x^2 \\ \rho v_x v_y - b_x b_y \\ \rho v_x v_z - b_x b_z \\ (e + P_*) v_x - b_x \boldsymbol{b} \cdot \boldsymbol{v} \end{pmatrix}$$

Cell dependence



Pen, Arras, & Wong 2003

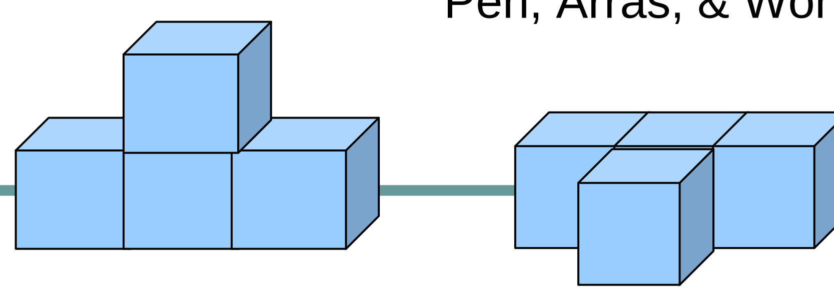
back

Algorithm – magnetic part

- Constrained transport(CT):
- 1. store magnetic field at cell faces;
- 2. the same electro-motive force is used immediately in the constraint step to preserve zero divergent of magnetic field.

$$\partial_t \vec{b} = \nabla \times (\vec{v} \times \vec{b}) \qquad \nabla \bullet \vec{b} = 0$$

Pen, Arras, & Wong 2003



Cell dependence

back

Detail for 122^3 box

```
single precision                                    (second)

---------------------------------------------------------------
                 | cuda(1)        | no comm                      |
---------------------------------------------------------------
one 122^3        | 0.342          | 0.104                        |
---------------------------------------------------------------
one cube for this table:
cuda(1):          1 GPU on 1 node, including the MPI communication
no comm:          1 GPU on 1 node, no communication


for one 122^3, extra 230 ms for communication
including:

device to host:                     70 ms
initiate buffer communication:      40 ms
copy inside gpu:                    20 ms (overlap)
wait for communication:             10 ms
host to device:                     50 ms
buffer calculation on device:       30 ms
update main matrix data:            30 ms
```

back