Retour d'expérience sur le développement du code RamsesGPU

Applications de type CFD pour l'astrophysique: Turbulence MHD

Pierre Kestener

CEA Saclay, DRF
Maison de la Simulation

Forum ORAP
17 mars 2016

Résumé

- **Depuis 2007, Nvidia Tesla 1.0**, les architectures matérielles de GPU ont énormément évolué et surtout **relâché** beaucoup de **contraintes de programmation** qui rendait l'obtention de bonnes performances parfois très difficiles.
- **Les outils logiciels ont énormément évolué:** debug (`cuda-gdb`, profiler (`nvvp`), intégration dans des outils d'analyse de performance (PAPI, TAU, ...), environnement de développement (eclipse et sa toolbox parallèle, ...)
- **En 2009-10: il pouvait y avoir une très grosse différence de performance entre une version GPU optimisée et non-optimisée :**
  - A-t-on respecté les contraintes liées à la coalescence mémoire ?
  - A-t-on respecté les recommendations sur le layout mémoire ? Structure-of-Array *vs* Array-of-Structure ?
  - Utilise-t-on la mémoire partagée pour optimiser la réutilisation des données ?

Résumé

- Aujourd'hui, sur les architectures Nvidia ≥ Kepler, arch 3.5:
  - **Certaines contraintes sont beaucoup moins *raides*** (notamment e.g. la coalescence mémoire, plus besoin d'utiliser `CudaMallocPitch` pour l'alignement mémoire des tableaux nD)
  - **Relâchement des contraintes sur les registres:** on peut implémenter des kernels CUDA plus complexes, avant d'atteindre les problèmes de *register spilling*
  - L'utilisation de la mémoire partagée est toujours importante, mais moins cruciale (ça dépend du type de kernel)
  - Introduction de l'UVA (Unified Virtual Adressing)
  - Les difficultés liées au *static linkage* de certaines variables (scope interne au fichier CUDA) disparaissent.
- C'est devenu **de plus en plus *facile* d'écrire du code CUDA**, en ayant déjà des performances non ridicules dès les premiers tests.
- On peut plus rapidement se focaliser sur ce qui importe: **reformuler (*refactoring*) l'implantation d'un algorithme pour exprimer du parallélisme et utiliser au mieux les différents niveaux de hiérarchie (gang, worker, ...) adapté à l'architecture matérielle.**

Content

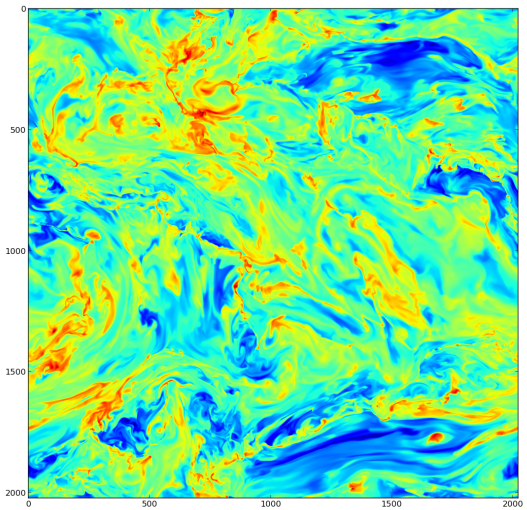
- **Astrophysics motivation (Protoplanetary disk)**
  GPU Implementation of high-order finite volumes schemes
- **Optimization strategies**
  - Kepler optimization: register and read-only data cache
  - Memory footprint optimization
- **Performance measurements on real-case applications**
  - Magneto-Rotational Instability (MRI) on CURIE hybrid partition (up to 256 GPUs) and on BLUE WATERS (576 GPUs)
  - MHD Turbulence on KEENELAND system (up 486 GPUs)
- **Perpectives / Future applications and developments**

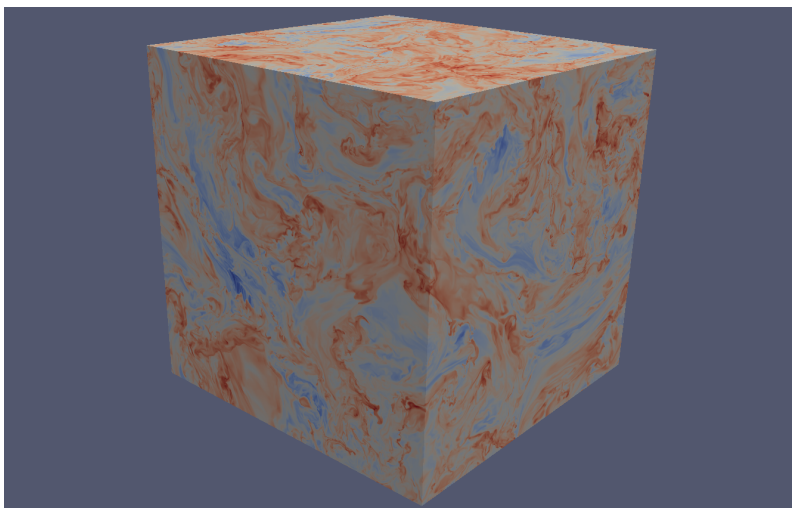
Motivation - Stencil applications - sciences cases

- **Porting finite volumes / finite differences schemes** (for *regular grid*) **on GPU** is reportedly said to be **easy**, **but** what is the **impact on performance of**:
  - complex numerical schemes : **CFD can have large amount of FLOPS / cell** (⇒ **Large register counts per thread**)
  - data precision : single / double
  - intrinsics instructions : e.g. fast math
  - algorithm : split / unsplit schemes, 2D vs 3D
  - physics : hydro / MHD / Riemann solver
  - other numerical part: shear border condition
- **Present code:** take the full MHD numerical scheme (from astrophysics code RAMSES) for GPU porting; **all refactoring allowed / put everything on GPU**
- Make the code production ready for science case on existing large clusters of GPU :
  - **magneto-rotationnal instability (MRI)**
  - **interstellar turbulence**

Code Ramses-GPU : MHD astrophysics applications

- RAMSES-GPU is developped in CUDA/C for astrophysics MHD applications of regular grid
- http://www.maisondelasimulation.fr/projects/RAMSES-GPU/html/
- ~ 70k lines of code (out of which ~ 15k in CUDA)
- Sciences cases applications :
  - **MRI in accretion disk** ($Pm = 4$): ~ 500000h on CURIE (256 GPU) at $800 \times 1600 \times 800$
  - **MHD Driven turbulence: (Mach** ~ 10): ~ 150000h on KEENELAND (U. Tenessee) up to resolution 2016^3 (486 GPUs)

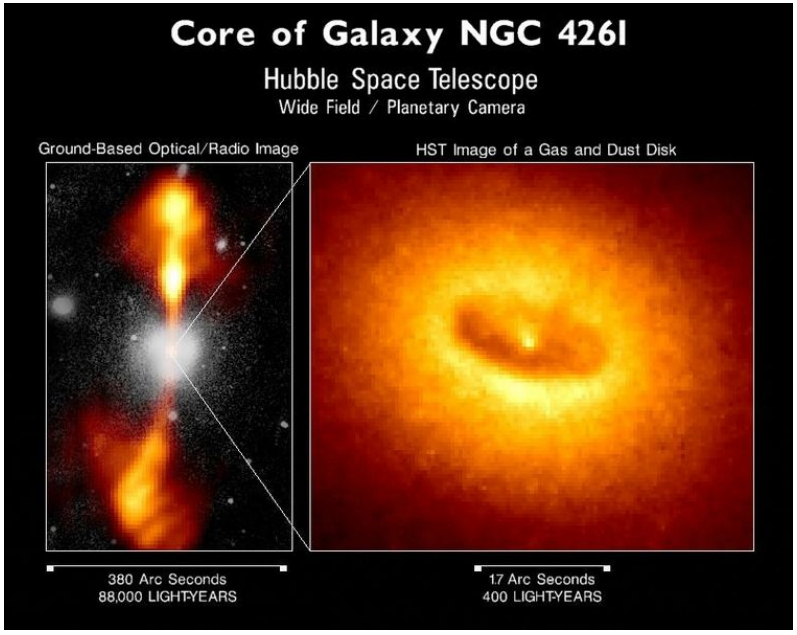




Summary

1. Astrophysics motivations - HPC applications
   - What is MRI (Magneto-rotational Instability) ?
   - High resources requirements : need for GPU acceleration
   - Compressible (M)HD and finite volume methods
2. CPU/GPU performances
   - Memory footprint optimization
   - MHD
   - Parallel IO
   - Teaching
   - Conclusion

Accretion disks



AGN NGC4261, source : HST/NASA/ESA

- Objects difficult to observe
- Main problem : matter falls (spiral) towards central object but there is an outgoing net transfert ofangular momentum (AM).

Systems with an accretion disk

- protostellar disk
- accretion disk in binary star systems
- Active galactic nuclei (AGN) (most luminous persistent sources in the universe)

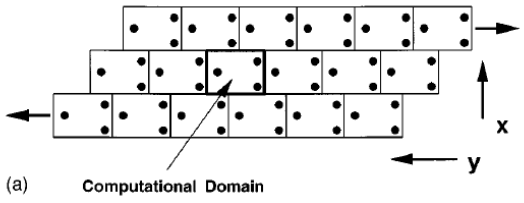
Understanding accretion disks

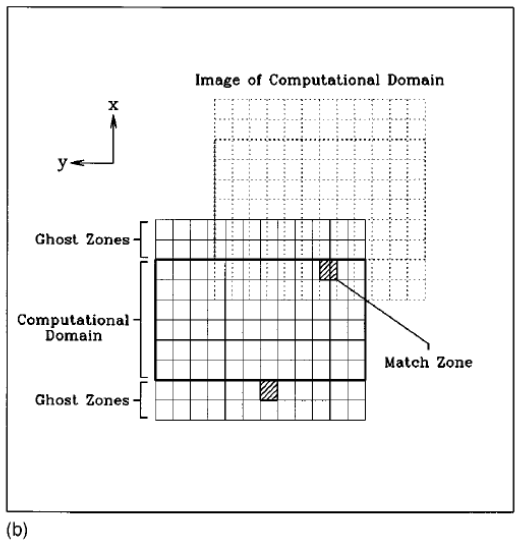
- Shear viscosity not sufficient; effective viscosity due to turbulence can provide an explanation to observed mass accretion rate
- How turbulence is driven ? Estimate AM transfert rates. What about **B** ?

MRI - Shearing box setup

Magneto-Rotationnal Instability (MRI)

- Long-term behavior study ⇒ long simulation runs (several 10^6 time steps); numerical scheme must be stable !
- use Cartesian grid x, y, z to model R, ϕ, z in the neighborhood of R_0
- Write MHD equations in a rotating frame Ω_0
- differential rotation is modeled by shear border conditions in azimutal direction and periodic in y and z





Ref: Balbus & Hawley, Rev. Mod. Phys. 70, 1-53 (1998)

MRI - Shearing box setup

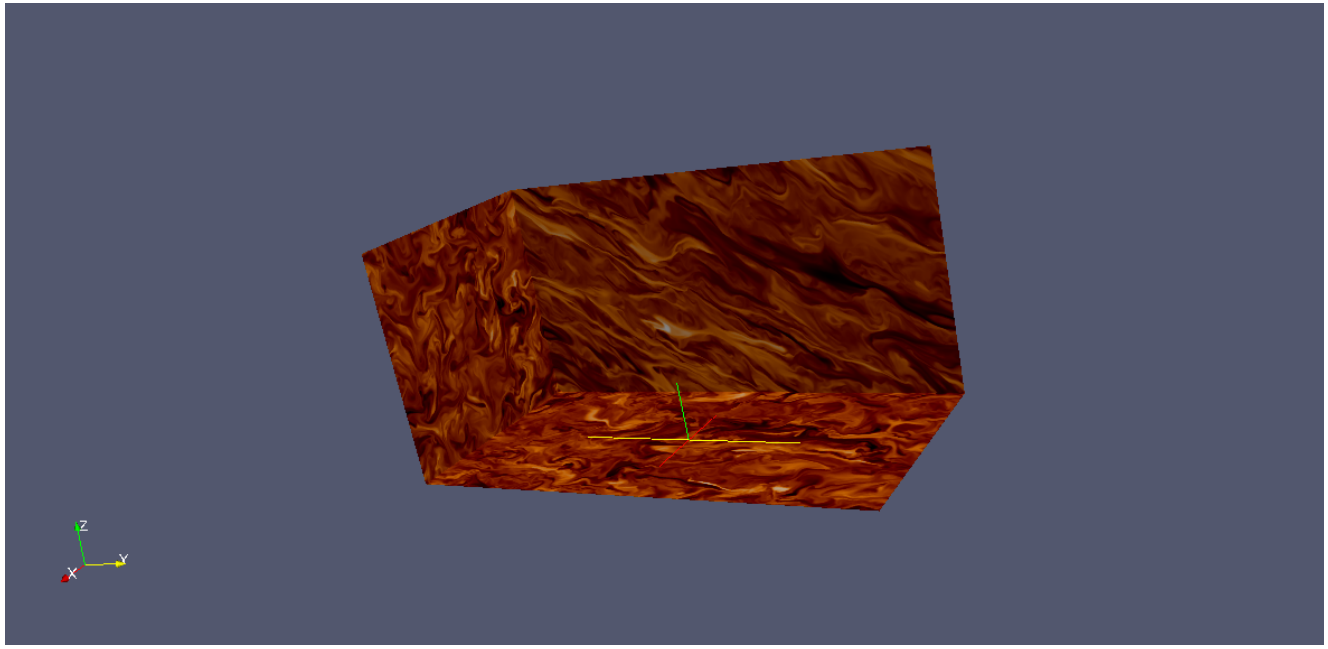
- **Shearing-box setup**: **MHD** equations in a **rotating frame** (Ω_0)
  - **compressible fluid dynamics**
  - **magnetic field**: induction equation
  - **inertial forces terms**
  - **(static) gravitational terms**
  - **dissipative terms** (viscosity + resistivity)
  - **shearing box border conditions** (in radial direction)

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \mathbf{v}) = 0$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \rho(\mathbf{v}.\nabla)\mathbf{v} + 2\rho\Omega_0 \times \mathbf{v} = \frac{(\nabla \times \mathbf{B}) \times \mathbf{B}}{4\pi} - \nabla P + \nabla T + \rho \mathbf{g}$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B} - \eta \nabla \times \mathbf{B})$$

MRI - Simulations



Movie representing **magnetic field** B_y evolution during 1 orbit
resolution is $\mathbf{800 \times 1600 \times 800}$, $\mathbf{Pm = 4}$, $\mathbf{Re = 25000}$
1 orbit is 1 day wall clock time on 256 GPU (CURIE system - Fermi M2090)
One of the largest MRI simulation ever done; same resolution on
BlueGene/Q with 32k cores (128k threads)

MHD - MRI - simulation run setup

- **Technical features of a CPU *run* of MHD SB with dissipative terms** [1] of a highly resolved MHD (Magneto-Rotational Instability) problem on the French machine TITANE (CCRT in 2010 [2]):
  - resolution : $256 \times 512 \times 256 \sim$ **33M cells**
  - time steps : $\mathbf{10^6}$
  - MPI processes : **1024**
  - *wall clock time* : 4 days, i.e. $\sim 100 hours$ (**100.000** compute hours)
  - effective performance in *cell updates per seconds* : $256 * 512 * 256 * 10^6/(100 * 3600) =$ **93.2M updates/s** (that is 93200 update/s per CPU core) - today's CPU perf is closer to **170M updates/s**.

[1] Intel Nehalem, quadcore, 2.93 GHz
[2] using the RAMSES (CPU) code

MHD - MRI - simulation run setup

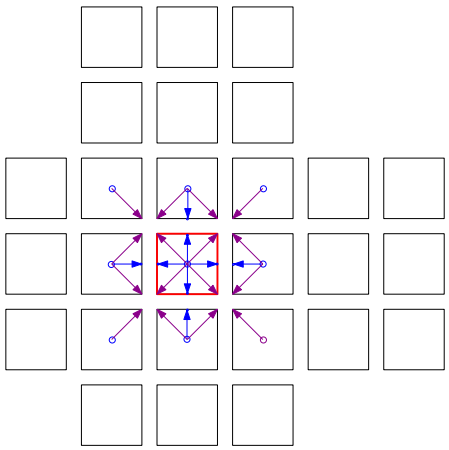
- **Problem :** double space resolution
  - space resolution : $512 \times 1024 \times 512 \Rightarrow$ total cells number is 8 times larger
  - time steps : $2 * 10^6$ (same dynamics)
  - $\Rightarrow$ NEED **16 times mode computing resources** ($1.6Mh$), just for one *run*
  - By keeping the same sub-domain size per MPI proc, we need 8000 MPI proc during 200 hours (one week) ! Or 2000 MPI processes for one month.
  - If perfect *weak scaling*, $\Rightarrow$: $8000 * 93200 = 746$Mupdate/s for the entire system
- **Can we implement a GPU-based solution ?**
- **What for ?**
  - Solve the same problem faster.
  - Solve bigger problems for smaller amount of time
- **What would be the amount of computing resources ?**
- **Solution: develop code Ramses-GPU using CUDA/C**

Godunov finite volume scheme on GPU - stencil algorithms

- **Conservative form** :
  $\mathbf{U} = [\rho, \rho u, \rho v, \rho w, E, B_x, B_y, B_z]$

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x + \mathbf{G}(\mathbf{U})_y + \mathbf{H}(\mathbf{U})_z = 0$$

- After **time and space integration** between t_n and t_{n+1} over a cell:



$$\frac{\mathbf{U}_{i,j,k}^{n+1} - \mathbf{U}_{i,j,k}^{n}}{\Delta t} + \frac{\mathbf{F}_{i+1/2,j,k}^{n+1/2} - \mathbf{F}_{i-1/2,j,k}^{n+1/2}}{\Delta x} + \ldots = 0$$

- **induction equation discretization**, e.g. for B_x

$$\frac{B_{x,i-1/2,j,k}^{n+1} - B_{x,i-1/2,j,k}^{n}}{\Delta t} = \frac{E_{z,i-1/2,j+1/2,k}^{n+1/2} - E_{z,i-1/2,j-1/2,k}^{n+1/2}}{\Delta y} + \ldots$$

Godunov finite volume scheme on GPU - stencil algorithms

Pseudo-algorithm for $\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x = 0$

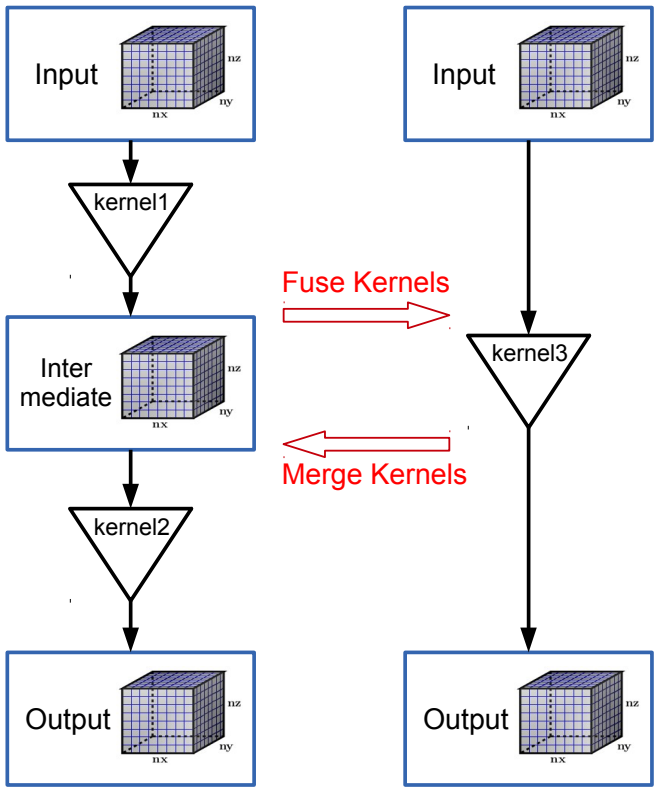
for k **do**
  for j **do**
    for i **do**
      - Get Hydro State $\mathbf{U}_{i,j,k}$
      - Convert To Primitive Variables using Equation of State
      - Compute slopes Δ_i
      - Interpolate states at $i-1/2$, $i+1/2$, evolve $\Delta t/2$
      - Solve local Riemann problem, compute fluxes $\mathbf{F}^{n+1/2}_{i+1/2}$
      - Update

$$\mathbf{U}^{n+1}_{i,j,k} = \mathbf{U}^{n}_{i,j,k} + \Delta t(\mathbf{F}^{n+1/2}_{i+1/2,j,k} - \mathbf{F}^{n+1/2}_{i-1/2,j,k})/\Delta x$$

    end for
  end for
end for

- Very large FLOPS / cell / time step : ~ 700 FLOPS in Hydro and ~ 3500 FLOPS in MHD ⇒ lots of local variables, large register requirements ⇒ potential register spilling problems
- CUDA/C parallelization strategy: loops over i and j are unfolded; each thread receives a z-column to deal with (see P. Micikevicius, ACM, 2009)
- Allow some refactoring between CPU and GPU version: split the for loops (then need to store intermediate results in GPU external memory)

Godunov finite volume scheme on GPU - stencil algorithms



- Very large FLOPS / cell / time step : ~ 700 FLOPS in Hydro and ~ 3500 FLOPS in MHD ⇒ lots of local variables, large register requirements ⇒ potential register spilling problems
- CUDA/C parallelization strategy: loops over i and j are unfolded; each thread receives a z-column to deal with (see P. Micikevicius, ACM, 2009)
- Allow some refactoring between CPU and GPU version: split the for loops (then need to store intermediate results in GPU external memory)

Godunov finite volume scheme on GPU - stencil algorithms

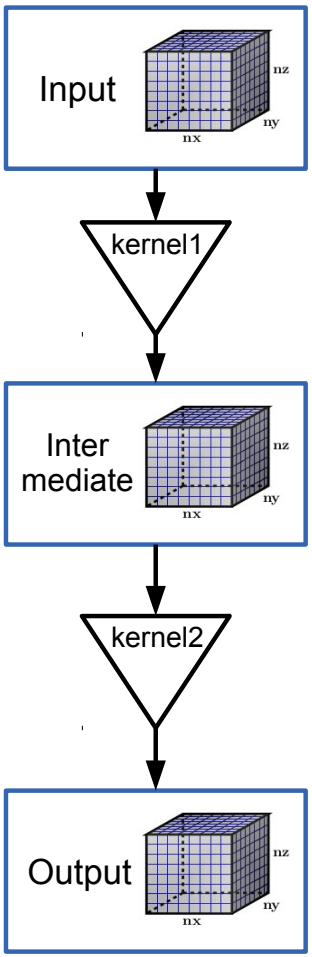
- Directional **splitted** scheme ⇒ **stencil 1D** : **only one GPU kernel** (low memory storage required)
- Directional **unsplit** scheme ⇒ **stencil 3D** : better perf when split computation into 3 GPU kernels (larger memory storage)
  - Sub-domain decomposition ⇒ strongly improve memory storage required.
- **GPU implementation/parallelization :** naturally unfold for loops over i and j; loop over z is performed inside one GPU thread (efficient shared memory usage).

Summary

1. Astrophysics motivations - HPC applications
   - What is MRI (Magneto-rotational Instability) ?
   - High resources requirements : need for GPU acceleration
   - Compressible (M)HD and finite volume methods

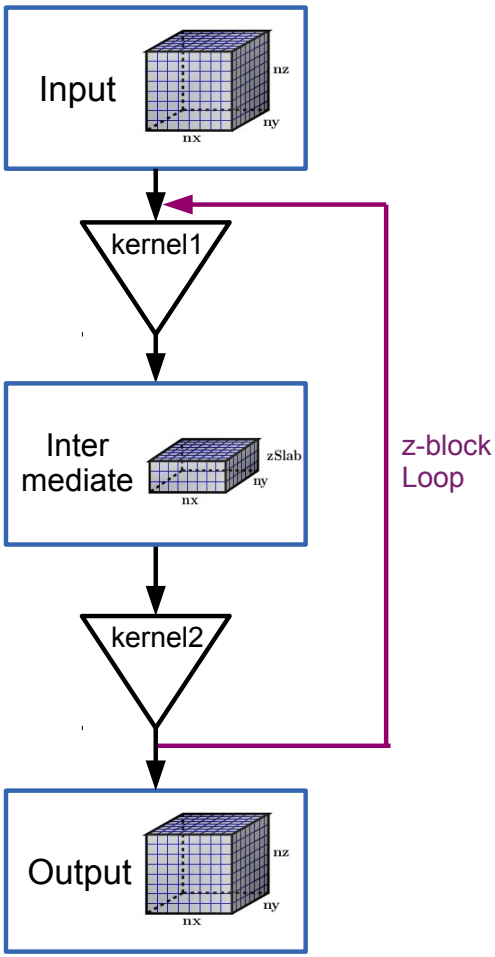
2. CPU/GPU performances
   - Memory footprint optimization
   - MHD
   - Parallel IO
   - Teaching
   - Conclusion

Memory footprint optimization - MHD scheme



- MHD scheme is much more memory consumming than pure hydrodynamics
- **Goal: Enable the largest ever done MRI simulation at resolution** $800 \times 1600 \times 800$
- With original GPU implementation, memory requirement was more than **5.8 GBytes / per GPU** (using 256 GPUs) $\Rightarrow$ can't do it on the CURIE system ☹

Memory footprint optimization - MHD scheme



- MHD scheme is much more memory consumming than pure hydrodynamics
- **Goal: Enable the largest ever done MRI simulation at resolution** $800 \times 1600 \times 800$
- With original GPU implementation, memory requirement was more than **5.8 GBytes / per GPU** (using 256 GPUs) $\Rightarrow$ can't do it on the CURIE system ☹
- **Memory footprint optimization:** apply a **z-blocking technique** (sub-domain decomposition) to all kernels involved $\Rightarrow$ **1.9 GBytes / per GPU** (using 256 GPUs) $\Rightarrow$ ☺

MHD - 3D Orszag-Tang - performances

Hardware configuration : Fermi M2090 - Single GPU

Single-GPU/CPU run - Number of 10^6 cell updates / second in single precision

size	CPU	CPU (dissip)	GPU	GPU (dissisp)	$\frac{GPU}{CPU}$
32x32x32	0.184	0.161	6.03	4.97	**32.7**
48x48x48	0.195	0.168	9.63	8.49	**49.4**
64x64x64	0.201	0.175	11.94	10.43	**59.4**
96x96x96	0.204	0.176	12.60	11.20	**61.7**
128x128x128	0.210	0.184	13.64	12.08	**64.5**
150x150x150	0.218	0.183	14.60	13.00	**67.0**

- On CPU, dissipative terms (~ 12%) whatever domain size
- On GPU, dissipative terms impact is 18% (32^3) and 11% (150^3); dissipative terms computation needs to call border routine (larger cost)
- Double-precision has a strong impact on GPU perf: ~ /4 $\Rightarrow$ **reg. spilling problems**

MHD - 3D Orszag-Tang - performances

Hardware configuration : Fermi M2090 - Single GPU

Single-GPU/CPU run - Number of 10^6 cell updates / second in single precision

size	CPU	CPU (dissip)	GPU	GPU (dissisp)	$\frac{GPU}{CPU}$
32x32x32	0.184	0.161	6.03	4.97	**32.7**
48x48x48	0.195	0.168	9.63	8.49	**49.4**
64x64x64	0.201	0.175	11.94	10.43	**59.4**
96x96x96	0.204	0.176	12.60	11.20	**61.7**
128x128x128	0.210	0.184	13.64	12.08	**64.5**
150x150x150	0.218	0.183	14.60	13.00	**67.0**

- Fermi M2090 : Double-precision has a strong impact on GPU perf: $\sim /4 \Rightarrow$ **reg. spilling problems**
- Kepler: reg. spilling problem strongly reduced.
- Kepler: **1 K20 GPU** ~ 2.5 **faster than a 16-core CPU node (2 Proc Intel Xeon E5-2670)**

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

Hardware configuration: Fermi M2090 - Single GPU

Number of 10^6 cell updates / second in single precision

size	CPU	GPU	$\frac{GPU}{CPU}$ total
16x32x16	0.138	1.65	**11.8**
32x64x32	0.156	5.56	**37.0**
48x96x48	0.163	8.02	**50.1**
64x96x64	0.169	8.96	**52.7**
64x128x64	0.168	9.12	**53.6**
96x150x96	0.175	10.1	**59.4**
96x192x96	0.170	10.2	**60.0**
128x128x64	0.173	9.3	**53.7**

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

Hardware configuration: M2090 - Multi GPU (1 GPU per MPI proc)

Mpi perf : Number of 10^6 cell updates / second / MPI proc in single precision. Sub-domain size is $128 \times 128 \times 64$

# proc MPI	global size	CPU	Efficiency CPU	GPU	Efficiency GPU	$\frac{GPU}{CPU}$ *total*
1	$128 \times 128 \times 64$	0.173	-	8.0	-	**50.1**
8	$256 \times 256 \times 128$	1.39	100.6 %	38.4	60.0 %	
64	$512 \times 512 \times 256$	11.1	100.1 %	306.7	59.9 %	
256	$512 \times 1024 \times 512$	43.8	98.7 %	1066.0	52.1 %	

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

Hardware configuration: M2090

MPI perf : Number of 10^6 cell updates / second / MPI proc in single precision using sub-domain size ($128 \times 128 \times 64$)

# proc MPI	global size	CPU	Efficiency CPU	GPU	Efficiency GPU
256	$512 \times 1024 \times 512$	43.8	98.7 %	1066.0	52.1 %

- **Simulation sizing (estimated):** CURIE/CPU, 2.10^6 time steps, domain size $512 \times 1024 \times 512 \Rightarrow$ **1 week, 4000 cores**
- **Simulation sizing (measured):** CURIE/GPU, 2.10^6 time steps, domain size $512 \times 1024 \times 512 \Rightarrow$ **5.8 days, 256 GPU**
- **Simulation sizing (actual):** CURIE/GPU (**DP**), $0.5x10^6$ time steps, domain size $800 \times 1600 \times 800 \Rightarrow$ **14 days, 256 GPU**, largest ever performed

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

- **largest ever performed** MRI simulation $800 \times 1600 \times 800$ (also done on BlueGene, but requiring 32k processors).
- **GPU implementation has a quite low MPI parallelization efficiency**; this due to communications (MPI + PciE) for borders update (normal and shear); this strongly modify profiling and decreases maximum possible acceleration (Amdahl law).
- There is room for optimization in the MHD scheme: **CUDA kernels are register limited on Fermi**.
- **Need to use very large sub-domain**; strong scaling should drop faster (but we only have 288 GPU on CURIE !)

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

***Profiling* for a 64 MPI proc configuration with $48 \times 96 \times 48$ sub-domain size (CURIE)**

Function	% of total time
Godunov scheme	22.9 %
dissipative terms	17.5 %
borders (internal/external)	22.7 %
shearing border	36.9 %

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

***Profiling* for a 64 MPI proc configuration with $48 \times 96 \times 48$ sub-domain size (CURIE)**

Function	% of total time
Godunov scheme	22.9 %
dissipative terms	17.5 %
borders (internal/external)	22.7 %
shearing border	36.9 %

MHD - MRI - 3D performances

MRI with dissipative terms (viscosity and resisitivity) and shearing-box

- borders update are dominated by MPI and CPU-GPU PciE communications.
- borders update is ~ 60% of total time ($48 \times 96 \times 48$ sub-domaine) !! It was only 10 % for single-GPU ...
- Once MPI + PciE communications are integrated into profiling, strong scaling is quite good

Ramses-GPU sur TITAN - parallel IO - PNetCDF

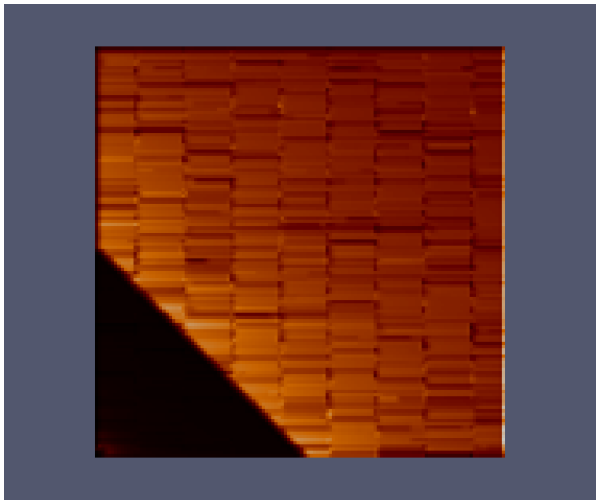
Number of GPU	Global size	Total output size (GBytes)	Local size per GPU	Local size per GPU (GBytes)	Lustre stripe count	time (s)	Effective bandwidth (GBytes/s)
512	2048^3	554	$256 \times 128 \times 512$	1.07	64	45	12.60
512	2048^3	554	$256 \times 128 \times 512$	1.07	128	25	22.95
4096	4096^3	4417	$256 \times 128 \times 512$	1.07	64	291	15.50
4096	4096^3	4417	$256 \times 128 \times 512$	1.07	128	**158** [1]	**28.80**

- Using PNetCDF collective IO
- **How good is IO perf we obtained ?**
- **Total Lustre aggregated bandwidth:** 2×500 **GBytes/s**
- THE most important parameter is Lustre stripe count
  `mkdir data; lfs setstripe -c 128 -s 1m data`
- https://www.olcf.ornl.gov/kb_articles/lustre-basics/

[1] avec les paramêtres LUSTRE par defaut: **plusieurs heures !**

Ramses-GPU - CUDA teaching

- Since 2014, **a simplified 2D version of RamsesGPU** (Euler equation, 2nd order finite volume scheme), is used for training / **teaching CUDA programming to 1st year PhD students (ED127).**
- CPU code written **fortran** (**astrophysics community still very Fortran-aholic**)
- 3 days mini-project to port euler dynamics to GPU using CUDA/Fortran



- **Usually very fruitfull:** students make CUDA error, we iteratively converge to a working solution. On the left, an exemple with a **data-race problem**.

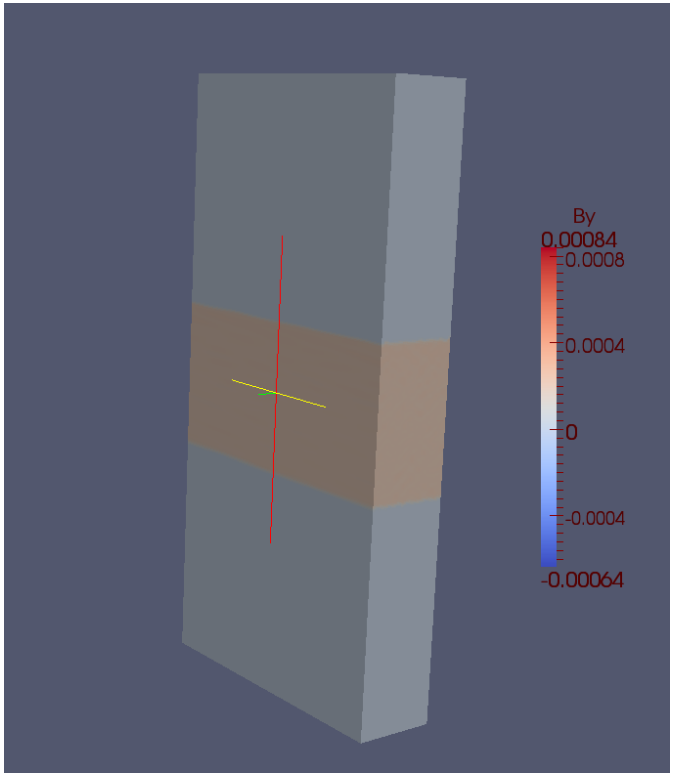
Ongoing and future work

- **Work in progress:**
  - New software developments / new physics modules(ongoing): **Poisson solver** for self-gravitating flows, cooling/heating terms
  - Code refactoring: use CUDA 5.0 *separate compilation*; problem: it uses more registers, so more register spilling and in turn ~ 25% perf drop
  - More on-line analysis to avoid writing output file too often
  - Coupling to Paraview/Catalyst for in-situ visualization (large domain)
  - New science case application: stratified MRI, ongoing work with C. Gammie and B. Ryan at Univ. Illinois at U.-C. on BlueWaters
- **Other possible developments:**
  - cylindrical/spherical coordinate (almost done)
  - higher-order schemes: weno5 ? Discontinuous Galerkin for MHD.
  - Radiative HD/MHD ?
  - AMR ? Interesting work by CLAMR (cell-based AMR on GPU with OpenCL) Coupling with an external library (e.g. p4est)

Conclusion

- Provide GPU Implementation for multiple finite volume schemes (hydrodynamics and MHD)
- **Performed some of the largest ever done MHD simulations**
- **Entire application on GPU**
- **Double-precision MHD implementation:**
  - strong performance drop on **Fermi** due to **register spilling** / shear border condition;
  - situation stronly improved on **Kepler**
- **Very good multi-GPU perf** provided using large sub-domain size
- Cuda/C++ code available :

http://www.maisondelasimulation.fr/projects/RAMSES-GPU/html/index.html



stratified MRI

On-poing work with C. Gammie and B. Ryan (2015)