Magnetohydrodynamic Turbulence in Accretion Discs

A test case for petascale computing in astrophysics

Marc Joos

Sébastien Fromang

Collaborators:
Pierre Kestener, Geoffroy Lesur, Héloïse Méheut, Daniel Pomarède & Bruno Thooris
Patrick Hennebelle, Andrea Ciardi, Romain Teyssier...

Service d'Astrophysique - CEA Saclay

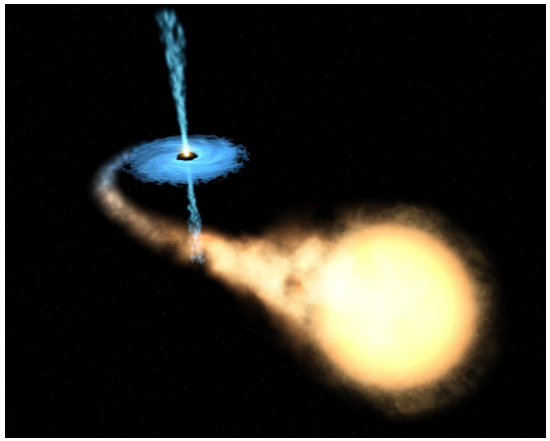
Outline

Outline

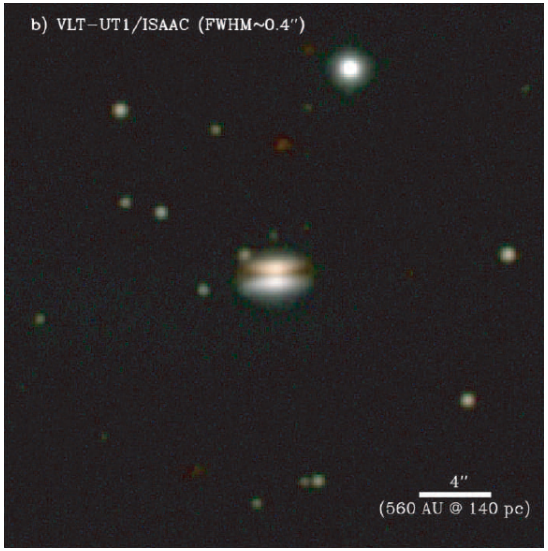
Accretion discs

What is it?

- discs of diffuse material (gas mostly)
- rotating around central object
- observed at all scales
  - protostars
  - neutron stars
  - supermassive black holes
  - etc.



(NASA)

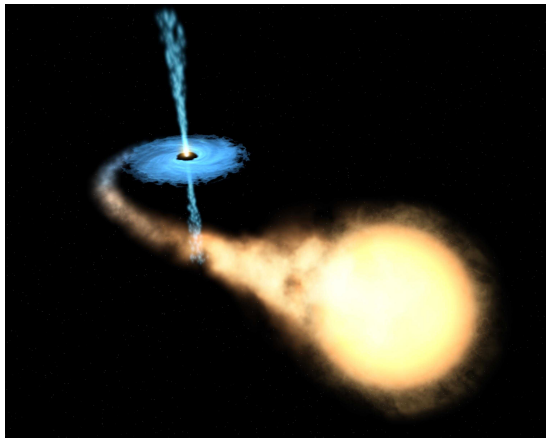


(Grosso *et al.*, 2003)

Accretion discs

What is it?

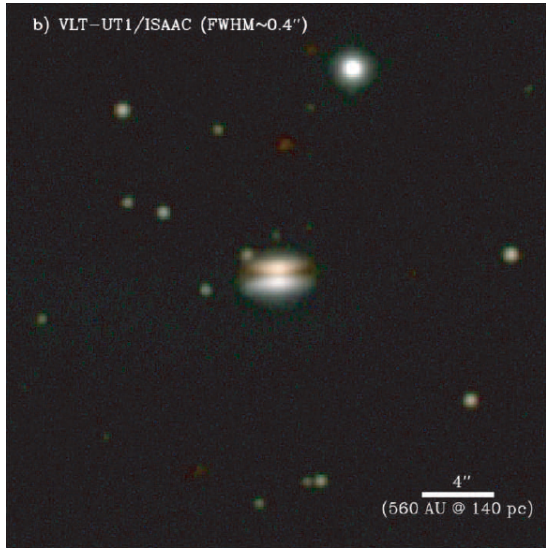
- discs of diffuse material (gas mostly)
- rotating around central object
- observed at all scales
  - protostars
  - neutron stars
  - supermassive black holes
  - etc.



(NASA)

Angular momentum:

- Material accretion
  $\Rightarrow$ angular momentum loss

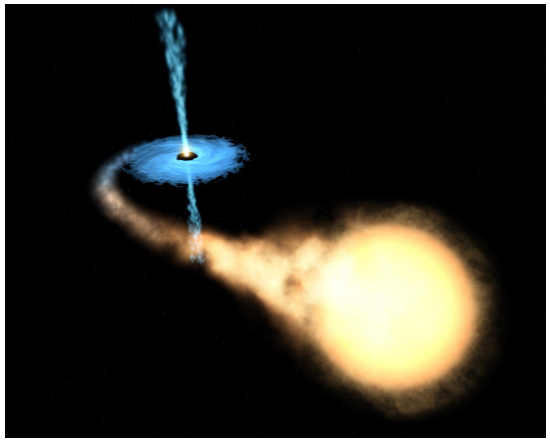


(Grosso *et al.*, 2003)

Accretion discs

What is it?

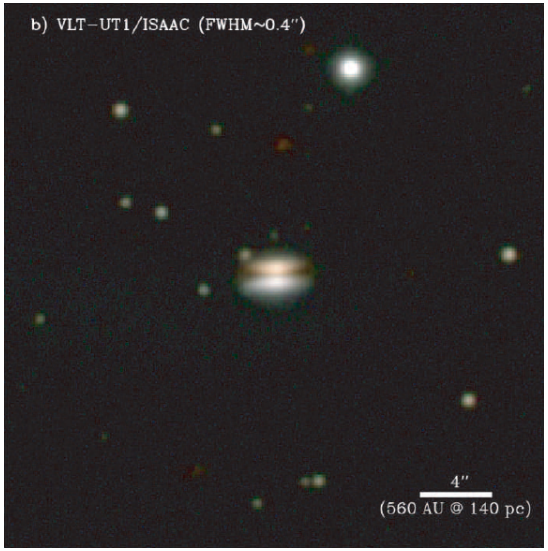
- discs of diffuse material (gas mostly)
- rotating around central object
- observed at all scales
  - protostars
  - neutron stars
  - supermassive black holes
  - etc.



(NASA)

Angular momentum:

- **Which mechanism to transport efficiently angular momentum?**

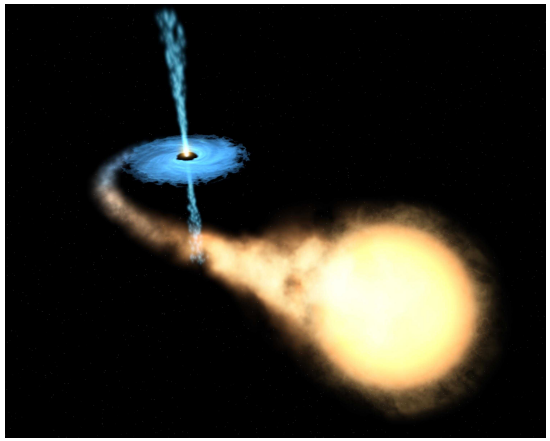


(Grosso *et al.*, 2003)

Accretion discs

What is it?

- discs of diffuse material (gas mostly)
- rotating around central object
- observed at all scales
  - protostars
  - neutron stars
  - supermassive black holes
  - etc.

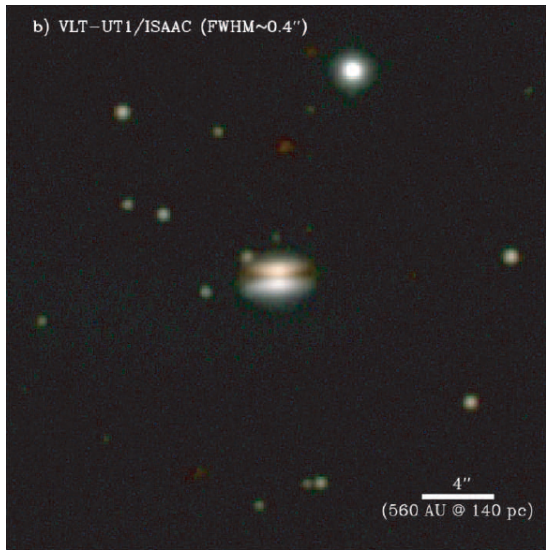


(NASA)

Angular momentum:

- *ad hoc* prescription: $\boxed{\nu_t = \alpha c_s H}$

(Shakura & Synyaev 1973; Lynden-Bell & Pringle 1974)



(Grosso *et al.*, 2003)

The magneto-rotational instability (MRI)

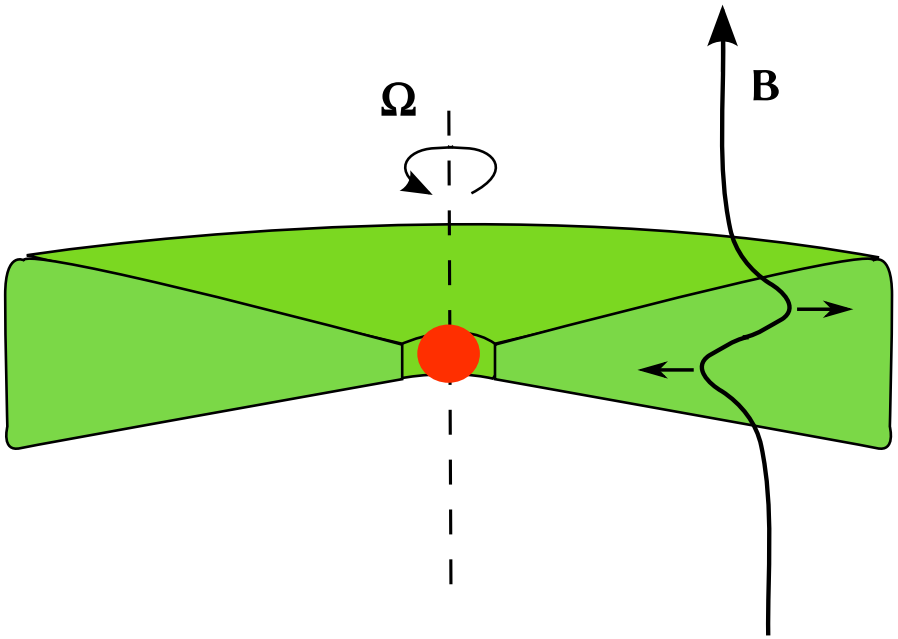
What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$



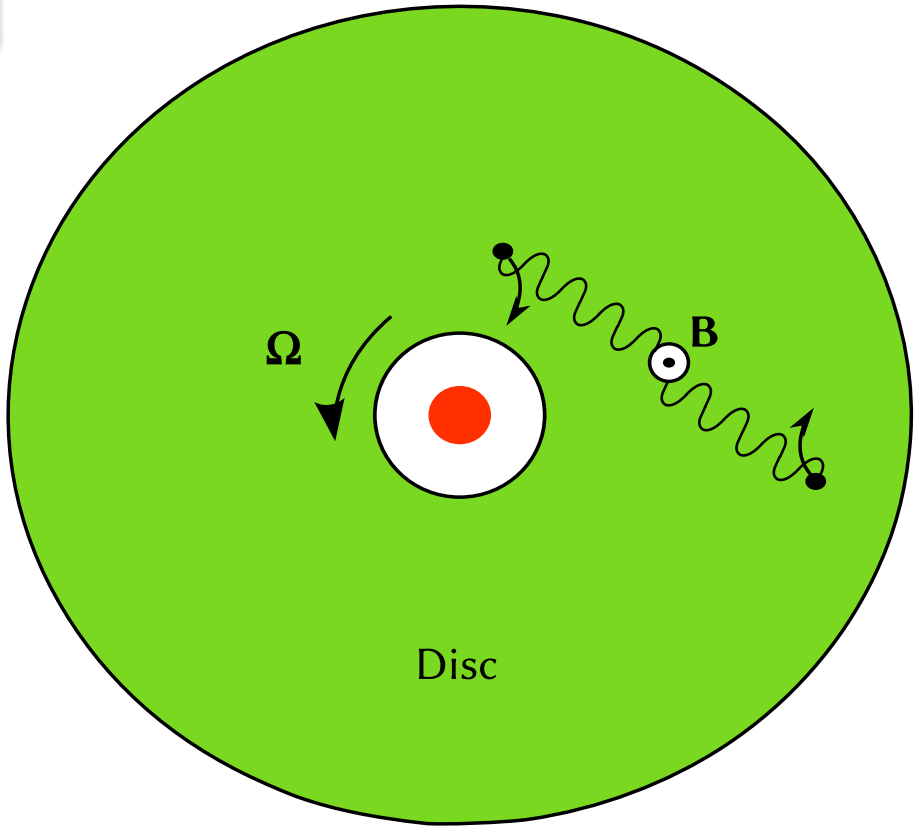


Fig.: Magneto-rotational instability principle

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

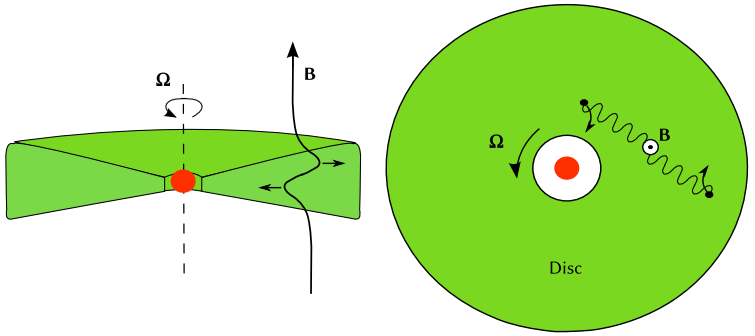


Fig.: MRI principle

Some dimensionless numbers...

- **Magnetic intensity:**

$$\beta \sim (c_s/v_a)^2$$

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

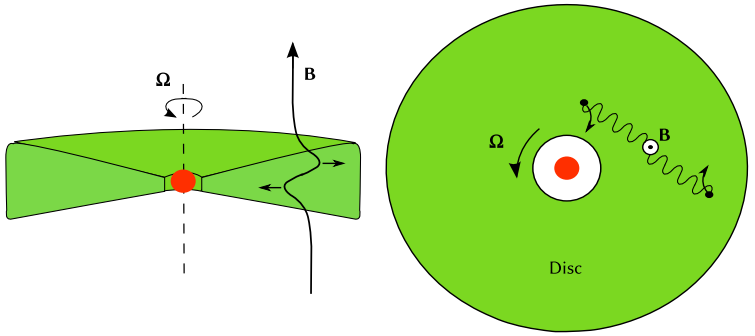


Fig.: MRI principle

Some dimensionless numbers...

- **Magnetic intensity:**

$$\beta \sim (c_s/v_a)^2$$

- **Viscosity:**

$$R_e = c_s H/\nu$$

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

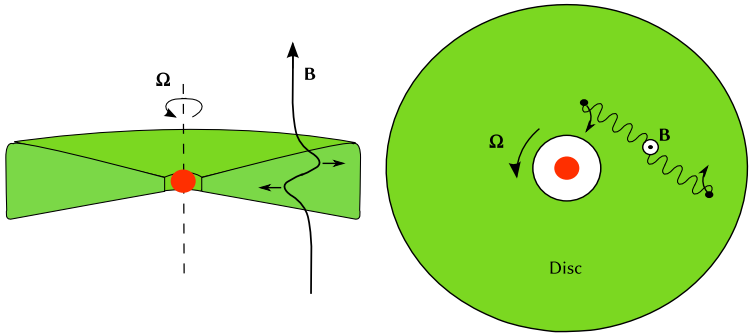


Fig.: MRI principle

Some dimensionless numbers...

- **Magnetic intensity:**

$$\beta \sim (c_s/v_a)^2$$

- **Viscosity:**

$$R_e = c_s H/\nu$$

- **Résistivity:**

$$R_m = c_s H/\eta$$

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

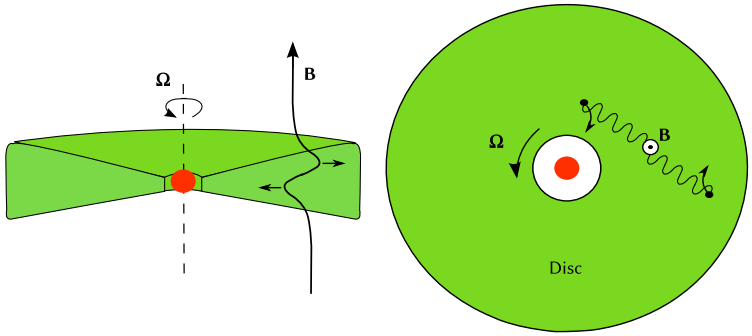


Fig.: MRI principle

Some dimensionless numbers...

- **Magnetic intensity:**
$$\beta \sim (c_s/v_a)^2$$
- **Viscosity:**
$$R_e = c_s H/\nu$$
- **Résistivity:**
$$R_m = c_s H/\eta$$
- **Magnetic Prandtl number:**
$$P_m = R_m/R_e = \nu/\eta$$

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

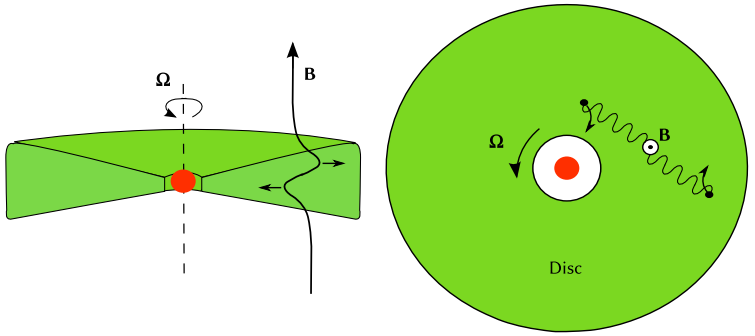


Fig.: MRI principle

Some dimensionless numbers...

- **Magnetic intensity:**
$$\beta \sim (c_s/v_a)^2$$
- **Viscosity:**
$$R_e = c_s H/\nu$$
- **Résistivity:**
$$R_m = c_s H/\eta$$
- **Magnetic Prandtl number:**
$$P_m \ll 1 \text{ in accretion discs (Balbus \& Henri 2008)}$$

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

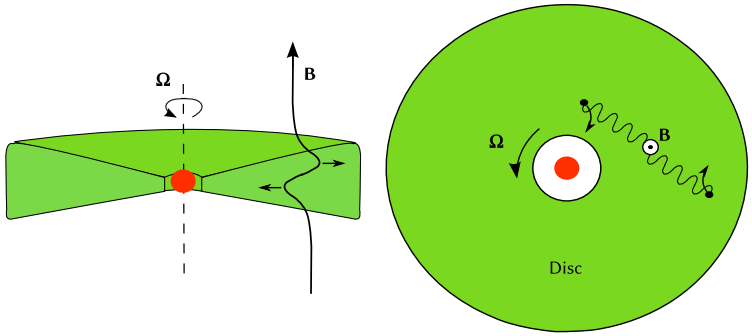
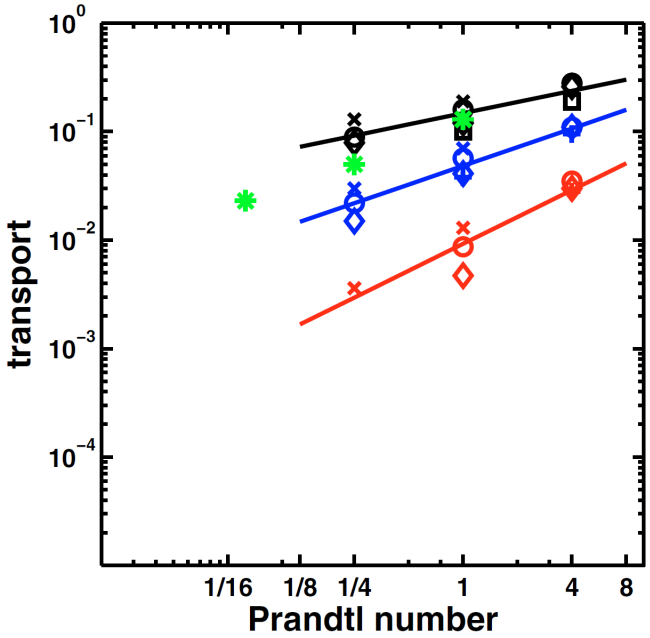


Fig.: MRI principle

Evolution of α with P_m?



— $\beta = 10^2$
— $\beta = 10^3$
— $\beta = 10^4$

□: $R_e = 400$
+: $R_e = 800$
◇: $R_e = 1600$
○: $R_e = 3200$
×: $R_e = 6400$

* $R_e = 20\,000$ & $\beta = 10^3$

(Lesur & Longaretti 2010)

The magneto-rotational instability (MRI)

What is it?

- MHD instability
  (Balbus & Hawley 1991)
- weak **B** field
- $d_r\Omega < 0$

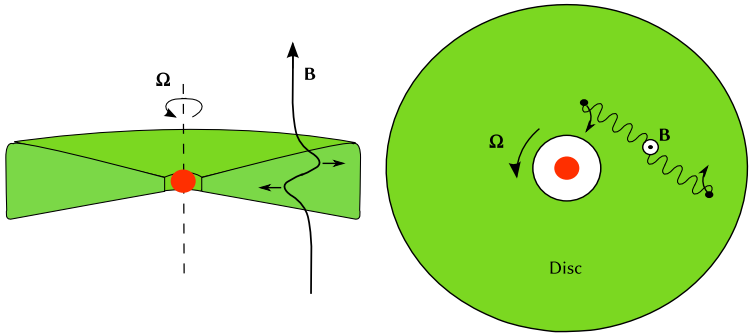
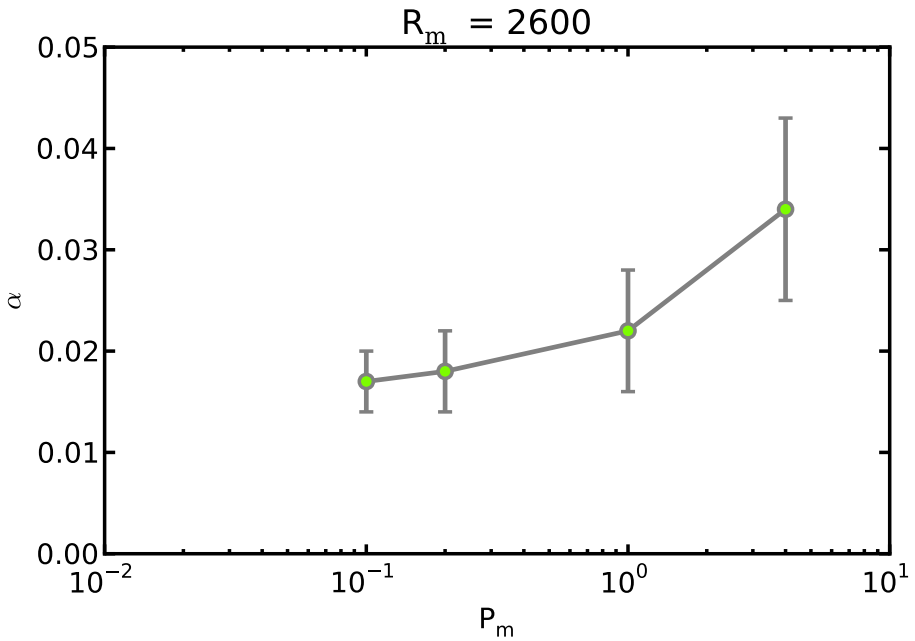


Fig.: MRI principle

Evolution of α with P_m?

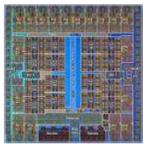


Outline

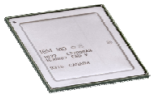
The BlueGene/Q hierarchy

Simulation performed on *Turing@IDRIS*

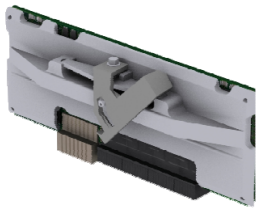
1. Chip
16 cores



2. Module
Single Chip



3. Compute Card
One single chip module,
16 GB DDR3 memory



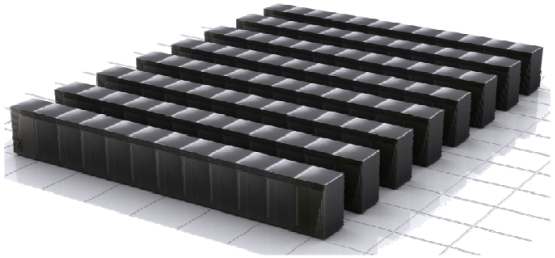
4. Node Card
32 Compute Cards

5a. Midplane
16 Node Cards

5b. I/O Drawer
8 I/O cards

6. Rack
2 Midplanes
1, 2 or 4 I/O Drawers

7. System



Per rack:

- peak performance: **209 TF**
- power efficiency: **~2.1 GF/W**

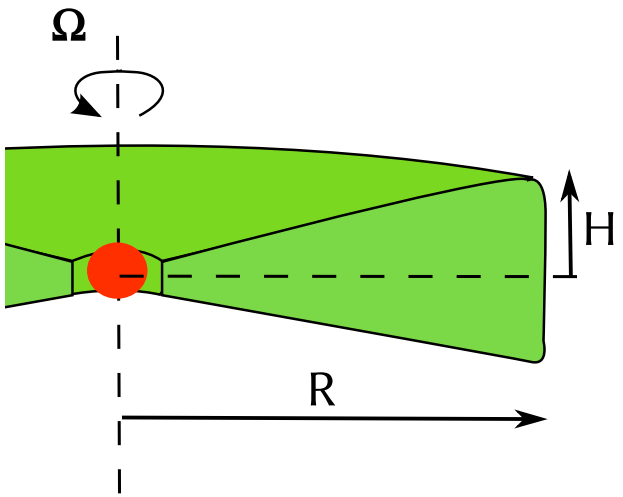
(adapted from IBM BG/Q white papers)

Local approach

Resolution issue

- turbulent scale: $\ell_{\text{turb}} \sim H$
- few 100's fluid elements to resolve H
- $H/R \sim 0.1$
- ~ 25 H to cover the radial range

$\rightarrow$ **computationally expensive!**

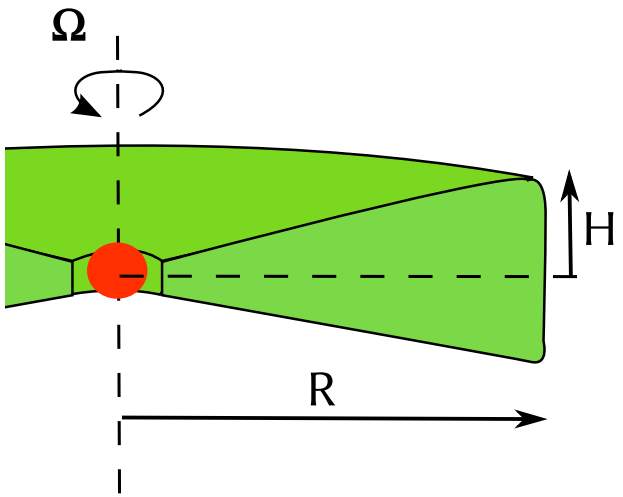


Local approach

Resolution issue

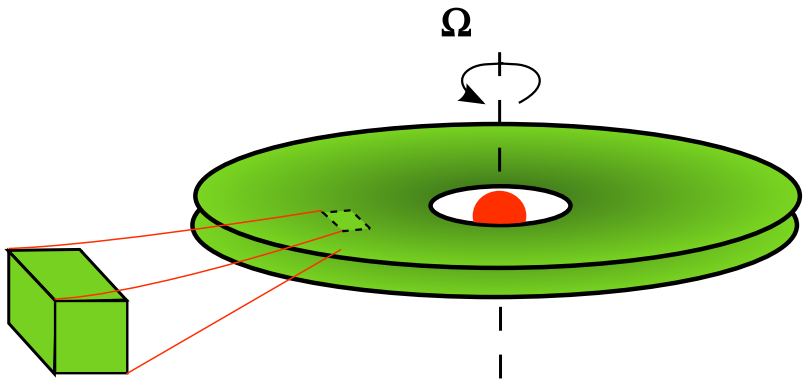
- turbulent scale: $\ell_{\text{turb}} \sim H$
- few 100's fluid elements to resolve H
- $H/R \sim 0.1$
- ~ 25 H to cover the radial range

$\rightarrow$ **computationally expensive!**



The local approach

- MHD equations
- dissipation (or not)
- EOS

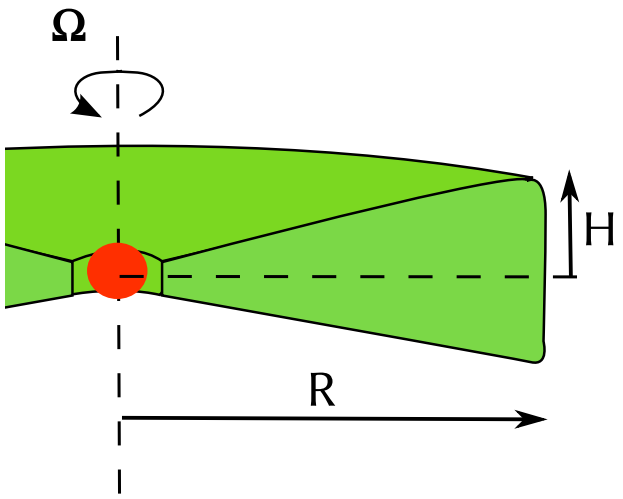


Local approach

Resolution issue

- turbulent scale: $\ell_{\text{turb}} \sim H$
- few 100's fluid elements to resolve H
- $H/R \sim 0.1$
- ~ 25 H to cover the radial range

$\rightarrow$ **computationally expensive!**



The local approach

$$(1) \qquad \partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0$$

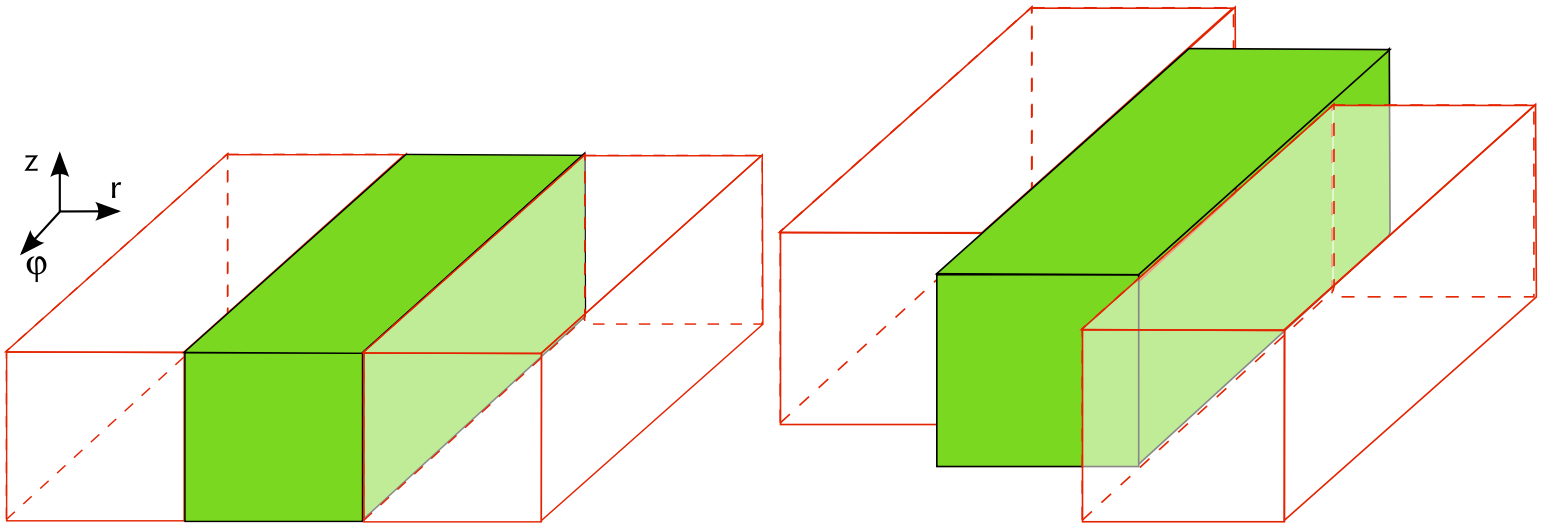
$$(2) \qquad \rho \left(\partial_t \mathbf{v} + (\mathbf{v} \cdot \nabla)\, \mathbf{v}\right) = -\nabla P + (\nabla \times \mathbf{B}) \times \mathbf{B} - 2\rho \boldsymbol{\Omega} \times \mathbf{v} + 2q\rho\Omega_0^2 x \mathbf{e}_x$$

$$(3) \qquad \partial_t \mathbf{B} = \nabla \times (\mathbf{v} \times \mathbf{B})$$

+ energy eq. or EOS

Local approach

The *shearing box*



(a) $t = 0$ (b) $t > 0$

Boundary conditions:

- **Azimuthal direction:** periodic
- **Vertical direction:** periodic
- **Radial direction:** periodic in shearing coordinates

Numerical methods

The RAMSES code (Teyssier 2002; Fromang *et al.* 2006)

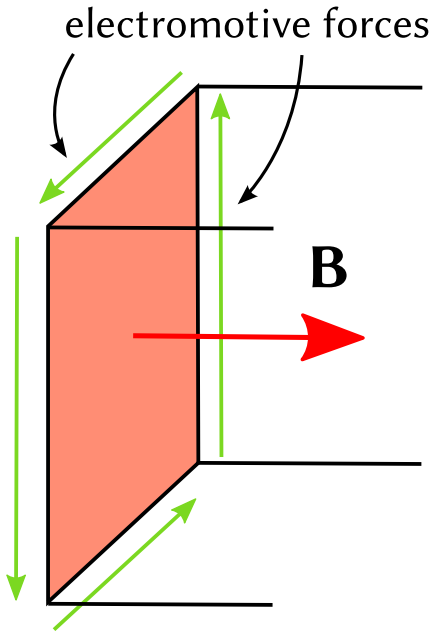
- **Finite volume method:** (Godunov's scheme)

$$\partial_t \mathbf{u} + \nabla \cdot \mathbf{F}(\mathbf{u}) = 0 \quad \Rightarrow \quad \mathbf{u}_i^{n+1} = \mathbf{u}_i^n - \frac{\mathbf{F}_{i+1/2}^{n+1/2} - \mathbf{F}_{i-1/2}^{n+1/2}}{\Delta x} \Delta t$$

  $\Rightarrow$ Riemann problem to solve at cells interface

- **upwind scheme:** stable if $|a\Delta t/\Delta x| \leq 1$
- **Constrained transport:**
  using Stokes theorem, the induction eq. becomes

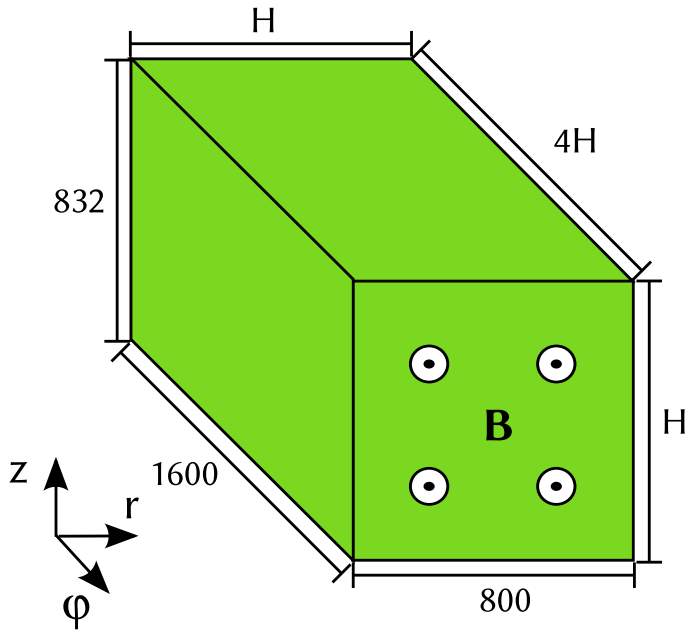
$$\partial_t \int_S \mathbf{B} \cdot \mathrm{d}\mathbf{S} + \int_L (\mathbf{B} \times \mathbf{v}) \cdot \mathrm{d}l = 0$$



Initial conditions

Parameters:

- **Resolution: 800×1600×832**
- **∼800 000 timesteps**
  (∼9 millions CPU hours, 25 orbits)
- on **32 768 CPUs**
  (131 072 sub-grids of 25×25×13)
- Toroidal **B**
- Homogeneous ρ

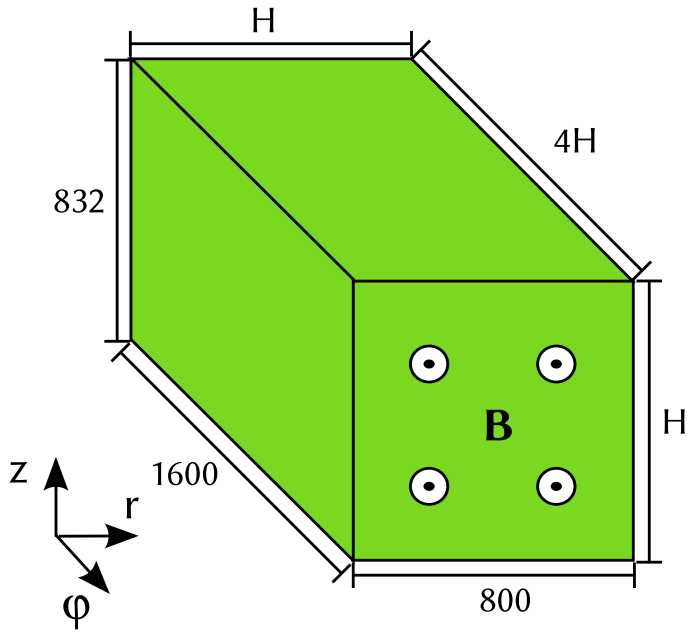


Initial conditions

Parameters:

- **Resolution: 800×1600×832**
- **∼800 000 timesteps**
  (∼9 millions CPU hours, 25 orbits)
- on **32 768 CPUs**
  (131 072 sub-grids of 25×25×13)
- Toroidal **B**
- Homogeneous ρ

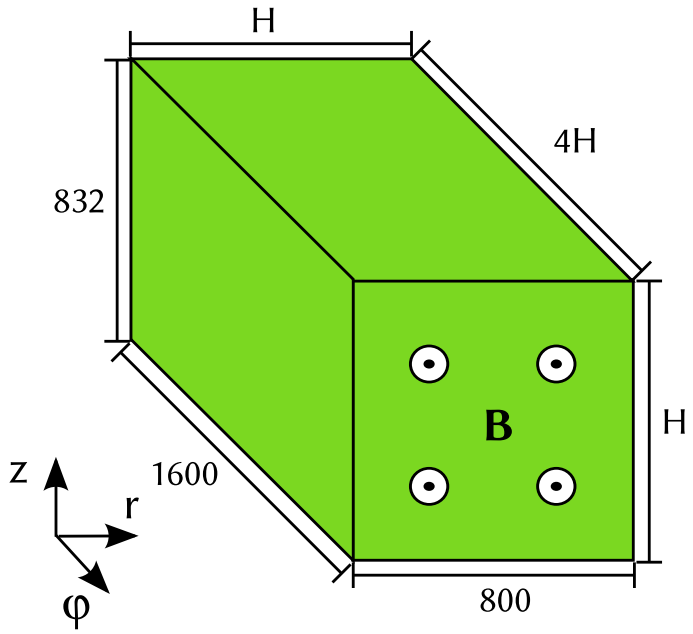
- **ideal MHD**
  $\rightarrow P_m \sim 1$



Initial conditions

Parameters:

- **Resolution: 800×1600×832**
- **∼800 000 timesteps**
  (∼9 millions CPU hours, 25 orbits)
- on **32 768 CPUs**
  (131 072 sub-grids of 25×25×13)
- Toroidal **B**
- Homogeneous ρ

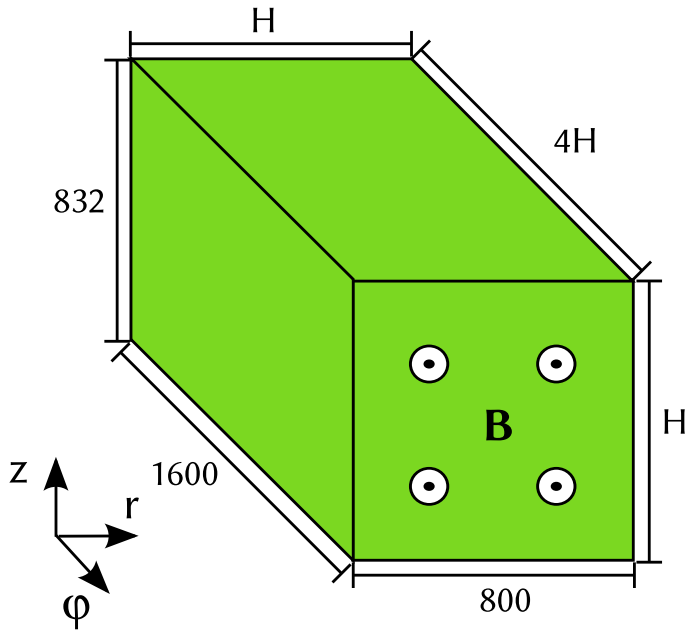


- **ideal MHD**
  → $P_m \sim 1$
- **non-ideal MHD**
  $R_e = 85\,000$ & $R_m = 2600$
  ⇒ $P_m = 0.03$

Initial conditions

Parameters:

- **Resolution: 800×1600×832**
- **∼800 000 timesteps**
  (∼9 millions CPU hours, 25 orbits)
- on **32 768 CPUs**
  (131 072 sub-grids of 25×25×13)
- Toroidal **B**
- Homogeneous ρ



- **ideal MHD**
  → $P_m \sim 1$
- **non-ideal MHD**
  $R_e = 85\,000$ & $R_m = 2600$
  ⇒ $P_m = 0.03$
  highest R_e ever reach!

What challenges at the petascale?

1. **Weak scaling**

Weak scaling

# CPUs	$t^{\text{2th./CPU}}_{\text{elapsed}}$ [s]	$t^{\text{4th./CPU}}_{\text{elapsed}}$ [s]
4096	$\sim$0.58	$\sim$0.55
8192	$\sim$0.82	$\sim$0.78
32768	$\sim$0.84	$\sim$0.80

What challenges at the petascale?

1. **Weak scaling**

Weak scaling

# CPUs	$t^{\text{2th./CPU}}_{\text{elapsed}}$ [s]	$t^{\text{4th./CPU}}_{\text{elapsed}}$ [s]
4096	∼0.58	∼0.55
8192	∼0.82	∼0.78
32768	∼0.84	∼0.80

- **∼70% efficiency on 32 768 CPUs**
- **∼5% faster with 4 threads/CPU**

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**

Parallel I/O

- 131 072 MPI processes, no hybridation
  $\Rightarrow$ if sequential I/O: **131 072 files to write (and read)!**
- different libraries tested, in particular parallel HDF5 and parallel NetCDF
- more details later!

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**

Parallel I/O

- 131 072 MPI processes, no hybridation
  $\Rightarrow$ if sequential I/O: **131 072 files to write (and read)!**
- different libraries tested, in particular parallel HDF5 and parallel NetCDF
- more details later!
- *(note however that GPFS holds on well even with so many files to deal with...)*

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**
3. **Visualization & data processing**

Visualization & data processing

How to visualize the data? (200 Go outputs!)

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**
3. **Visualization & data processing**

Visualization & data processing

How to visualize the data? (200 Go outputs!)

- high frequency outputs: sides of the domain
  $\rightarrow$ every 3200 timesteps

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**
3. **Visualization & data processing**

Visualization & data processing

How to visualize the data? (200 Go outputs!)

- high frequency outputs: sides of the domain
  - → every 3200 timesteps
  - → **fast visualization, 3D movies**

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**
3. **Visualization & data processing**

Visualization & data processing

How to visualize the data? (200 Go outputs!)

- high frequency outputs: sides of the domain
  - → every 3200 timesteps
  - → **fast visualization, 3D movies**
- low frequency outputs: whole domain
  - → every 32 000 timesteps

What challenges at the petascale?

1. **Weak scaling**
2. **Parallel I/O**
3. **Visualization & data processing**

Visualization & data processing

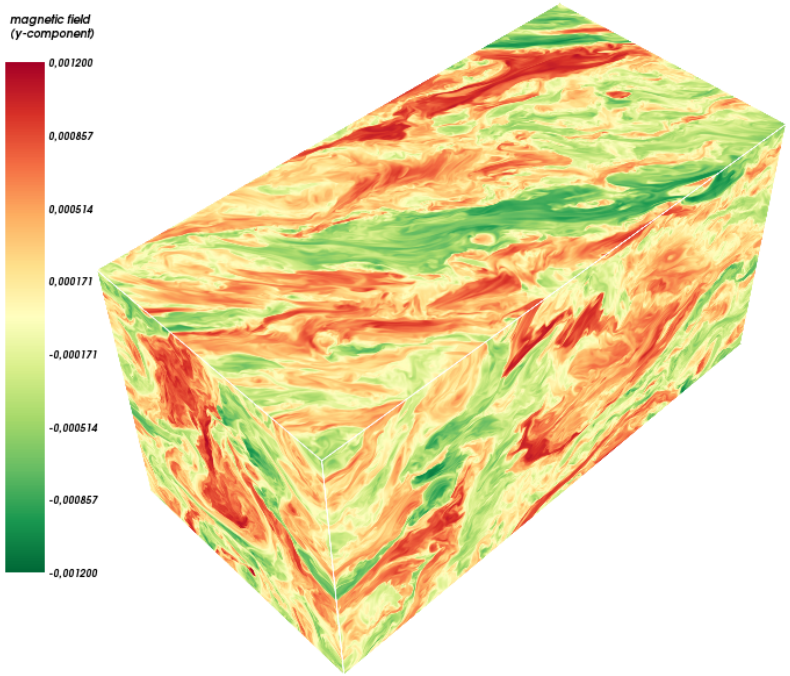
How to visualize the data? (200 Go outputs!)

- high frequency outputs: sides of the domain
  - → every 3200 timesteps
  - → **fast visualization, 3D movies**
- low frequency outputs: whole domain
  - → every 32 000 timesteps
  - → **science (averages, power spectra...)**

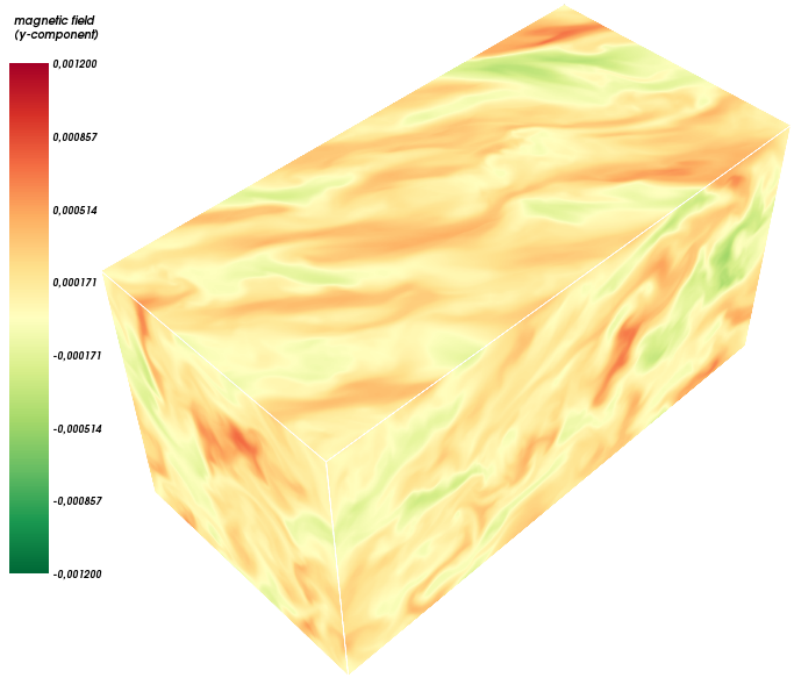
Outline

Overview

- ideal vs. non-ideal MHD: dissipation effects



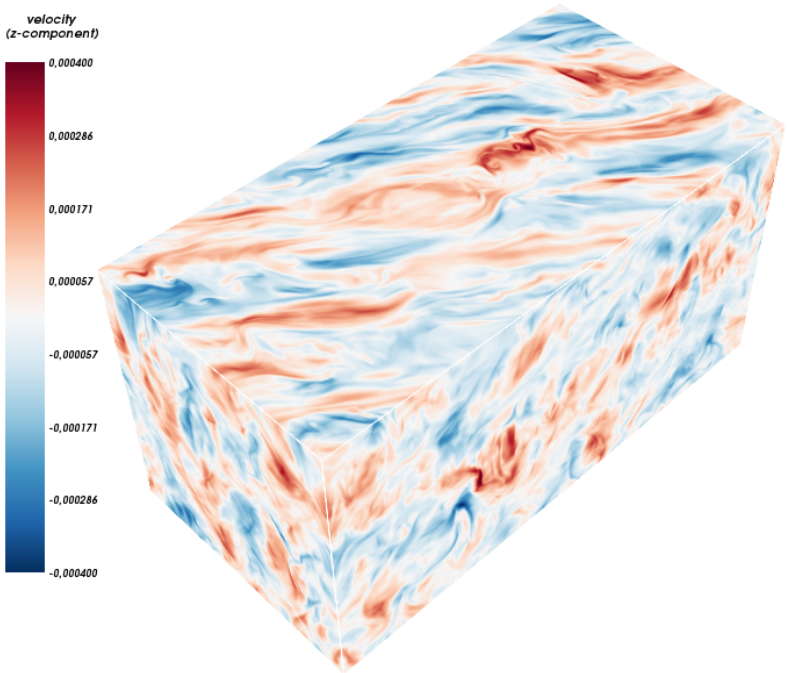
Ideal MHD – B_y



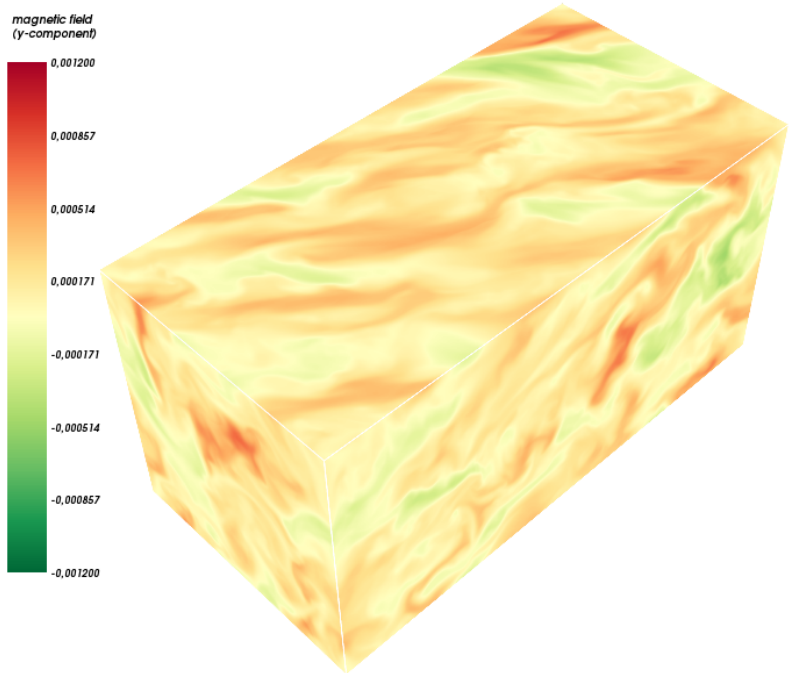
non-ideal MHD – B_y

Overview

- ideal vs. non-ideal MHD: dissipation effects
- kinetic vs. magnetic: dissipation scales



non-ideal MHD – v_z

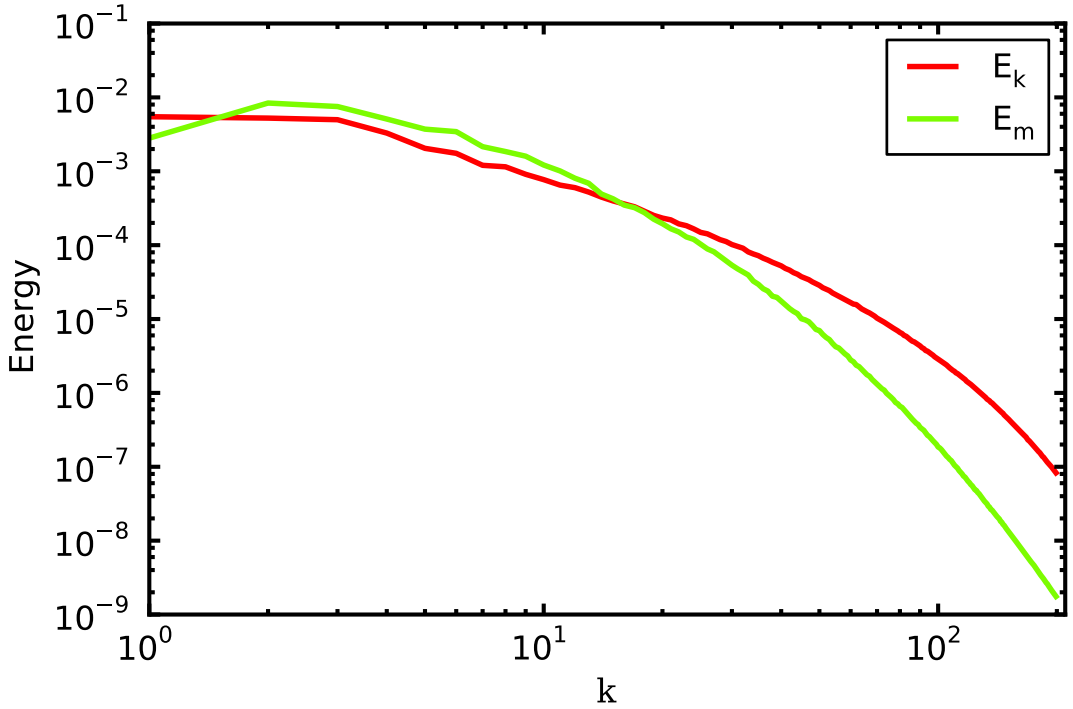


non-ideal MHD – B_y

Power spectra

Kinetic & magnetic energies

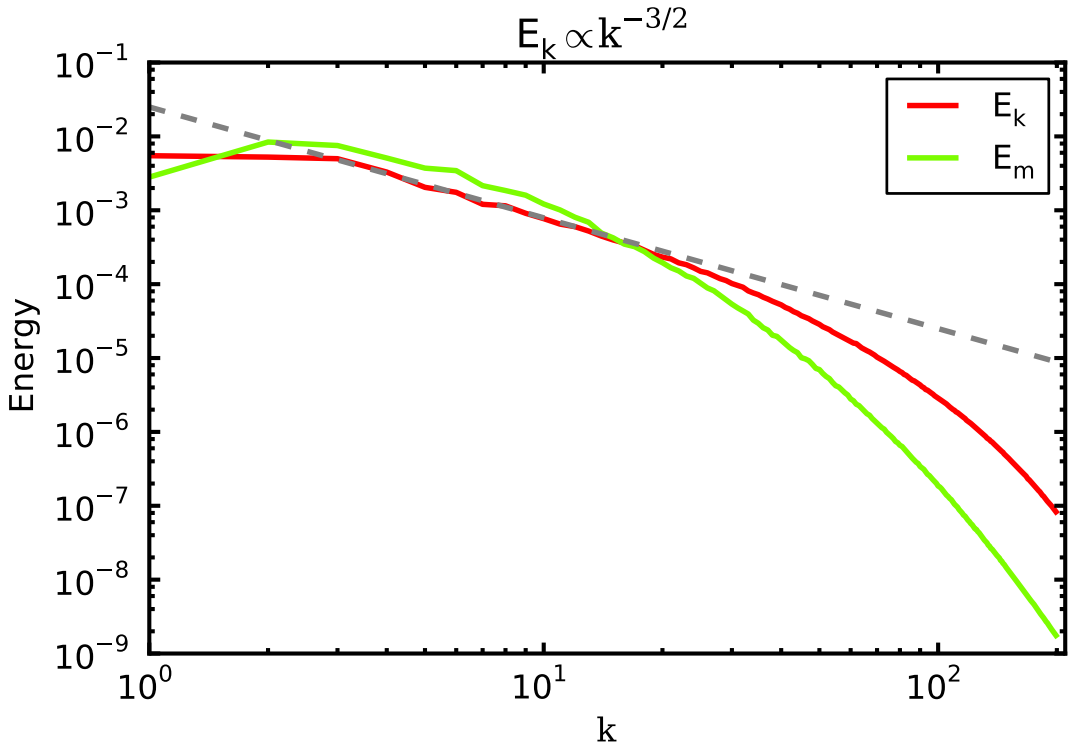
- $E_k(k_z) = \frac{1}{2}\rho_0 \left|\widetilde{\mathbf{v}}(k_z)\right|^2$
- $E_{\mathrm{mag}}(k_z) = \frac{1}{8\pi}\left|\widetilde{\mathbf{B}}(k_z)\right|^2$



Power spectra

Kinetic & magnetic energies

- $E_k(k_z) = \frac{1}{2}\rho_0 \left|\widetilde{\mathbf{v}}(k_z)\right|^2$
- $E_{\mathrm{mag}}(k_z) = \frac{1}{8\pi}\left|\widetilde{\mathbf{B}}(k_z)\right|^2$



Angular momentum transport rate

How to measure the turbulence efficiency?

$$\mathcal{T}_{\text{Reynolds}} = \left\langle \rho \left(v_x - \bar{v}_x\right)\left(v_y - \bar{v}_y\right) \right\rangle$$

$$\mathcal{T}_{\text{Maxwell}} = \left\langle -\frac{B_x B_y}{4\pi} \right\rangle$$

Angular momentum transport rate

How to measure the turbulence efficiency?

$$\mathcal{T}_{\text{Reynolds}} = \left\langle \rho \left(v_x - \bar{v}_x\right)\left(v_y - \bar{v}_y\right) \right\rangle$$

$$\mathcal{T}_{\text{Maxwell}} = \left\langle -\frac{B_x B_y}{4\pi} \right\rangle$$

$$\Rightarrow \boxed{\alpha = \frac{\mathcal{T}_{\text{Reynolds}} + \mathcal{T}_{\text{Maxwell}}}{P_0}}$$

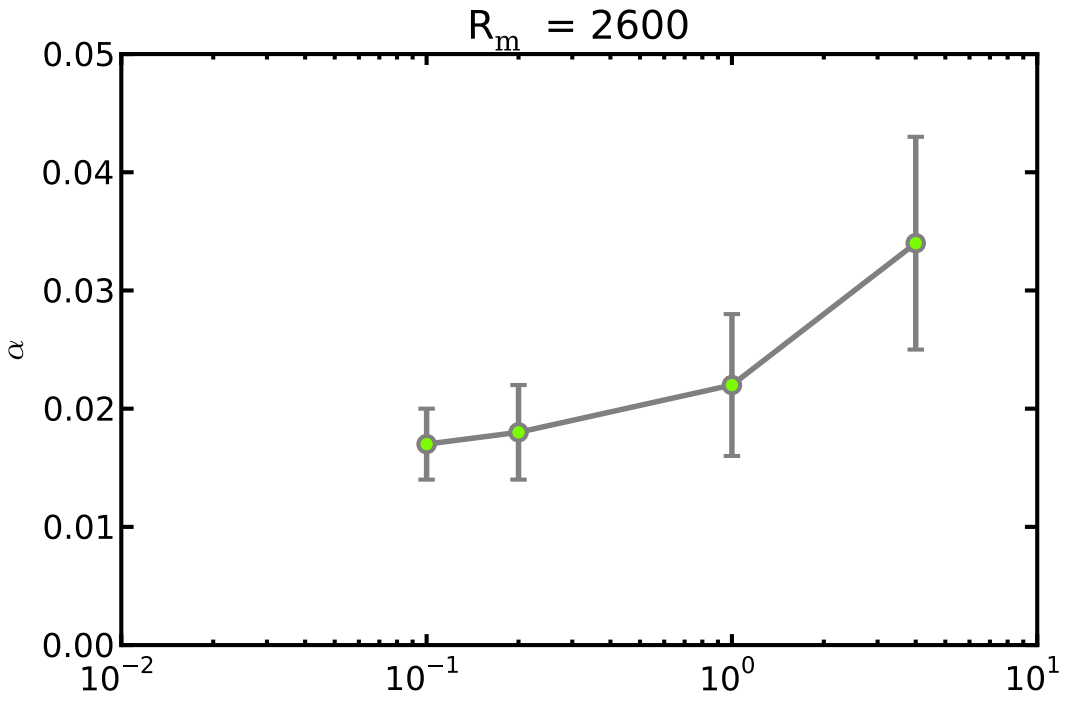
Angular momentum transport rate

How to measure the turbulence efficiency?

$$\mathcal{T}_{\text{Reynolds}} = \left\langle \rho \left(v_x - \bar{v}_x\right)\left(v_y - \bar{v}_y\right) \right\rangle$$

$$\mathcal{T}_{\text{Maxwell}} = \left\langle -\frac{B_x B_y}{4\pi} \right\rangle$$

$$\Rightarrow \boxed{\alpha = \frac{\mathcal{T}_{\text{Reynolds}} + \mathcal{T}_{\text{Maxwell}}}{P_0}}$$



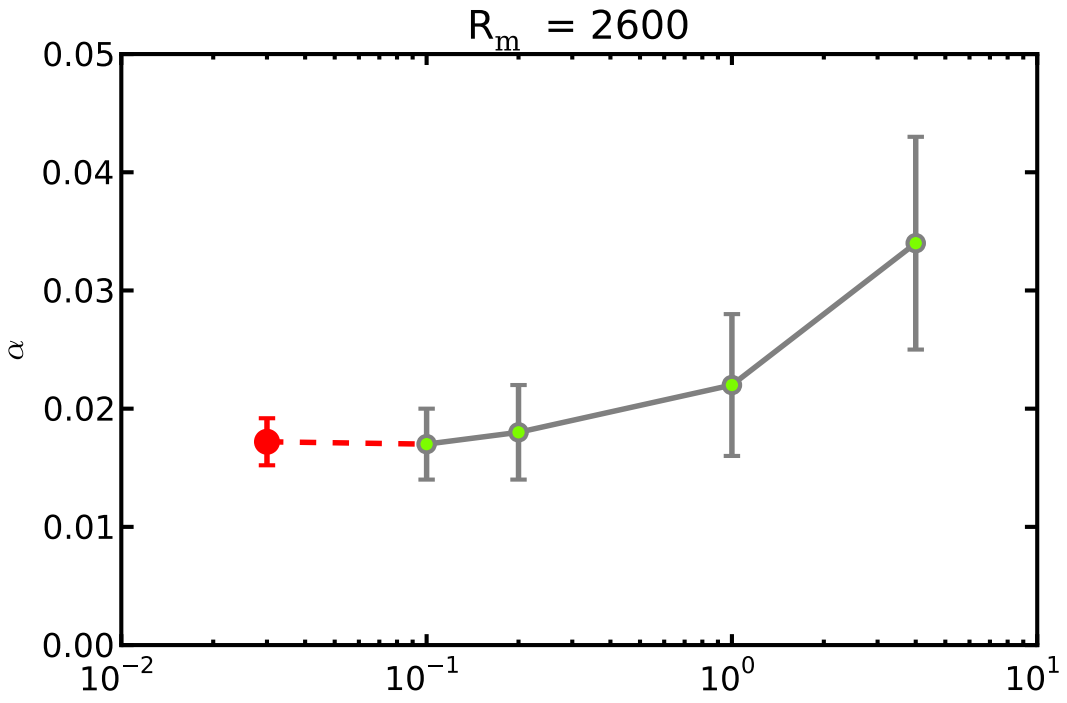
Angular momentum transport rate

How to measure the turbulence efficiency?

$$\mathcal{T}_{\text{Reynolds}} = \left\langle \rho \left(v_x - \bar{v}_x\right)\left(v_y - \bar{v}_y\right) \right\rangle$$

$$\mathcal{T}_{\text{Maxwell}} = \left\langle -\frac{B_x B_y}{4\pi} \right\rangle$$

$$\Rightarrow \boxed{\alpha = \frac{\mathcal{T}_{\text{Reynolds}} + \mathcal{T}_{\text{Maxwell}}}{P_0}}$$



Outline

Why do we care?

On-going evolution

- Computing power increases;
- Number of cores increases rapidely;
- Memory per core stays constant or decreases;
- Storing capacity is growing faster than the access speed.

Why do we care?

On-going evolution

- Computing power increases;
- Number of cores increases rapidely;
- Memory per core stays constant or decreases;
- Storing capacity is growing faster than the access speed.

Consequences

- Data generated increases with the computing power;
- More core but less memory: more files!;
- One file per process approach:
  - saturation of filesystems;
  - pre- & post-processing steps heavier;
- Time spent in I/O increases.

Why do we care?

On-going evolution

- Computing power increases;
- Number of cores increases rapidely;
- Memory per core stays constant or decreases;
- Storing capacity is growing faster than the access speed.

Consequences

- Data generated increases with the computing power;
- More core but less memory: more files!;
- One file per process approach:
  - saturation of filesystems;
  - pre- & post-processing steps heavier;
- Time spent in I/O increases.

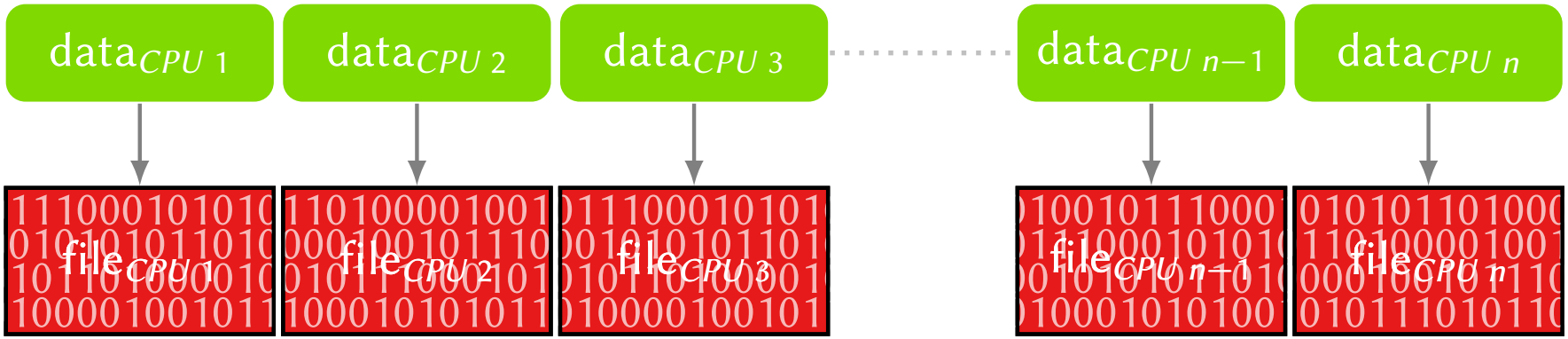
⇒ **Need for parallel I/O with sustainable performance on supercomputers**

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

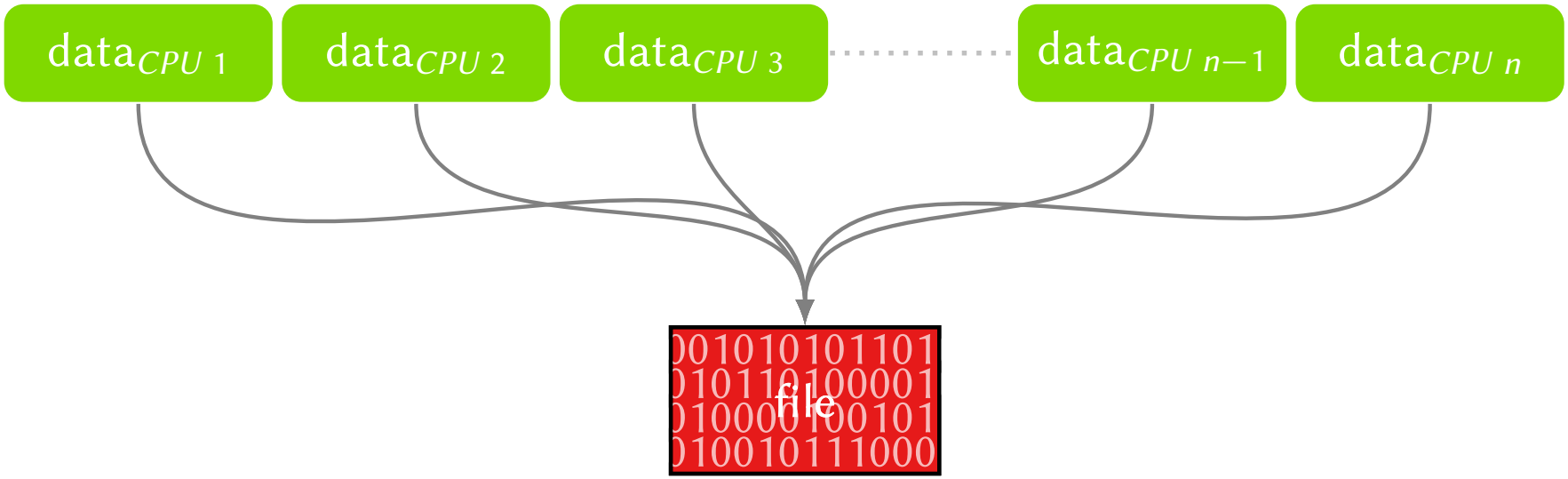
Parallel I/O approaches: POSIX



Parallel I/O approaches: POSIX

```
1 real(8), dimension(xdim,ydim,zdim,nvar) :: data
2 character(LEN=80) :: filename
3
4 call get_filename(myrank, 'posix', filename)
5
6 open(unit=10, file=filename, status='unknown', form='unformatted')
7 write(10) data
8 close(10)
```

Parallel I/O approaches: MPI-IO



Parallel I/O approaches: MPI-IO

```
integer :: xpos, ypos, zpos, myrank, i
real(8), dimension(xdim,ydim,zdim,nvar) :: data
integer, dimension(3) :: boxsize, domdecomp
character(LEN=13) :: filename

! MPI variables
integer :: fhandle, ierr
integer :: int_size, double_size
integer(kind=MPI_OFFSET_KIND) :: buf_size
integer :: written_arr
integer, dimension(3) :: wa_size, wa_subsize, wa_start

! Create MPI array type
wa_size    = (/ nx*xdim, ny*ydim, nz*zdim /)
wa_subsize = (/ xdim, ydim, zdim /)
wa_start   = (/ xpos, ypos, zpos /)*wa_subsize
call MPI_Type_Create_Subarray(3, wa_size, wa_subsize, wa_start &
     & , MPI_ORDER_FORTRAN, MPI_DOUBLE_PRECISION, written_arr, ierr)
call MPI_Type_Commit(written_arr, ierr)

call MPI_Type_Size(MPI_INTEGER, int_size, ierr)
call MPI_Type_Size(MPI_DOUBLE_PRECISION, double_size, ierr)

filename = 'parallelio.mp'
! Open file
call MPI_File_Open(MPI_COMM_WORLD, trim(filename) &
     & , MPI_MODE_WRONLY + MPI_MODE_CREATE, MPI_INFO_NULL, fhandle, ierr)
```

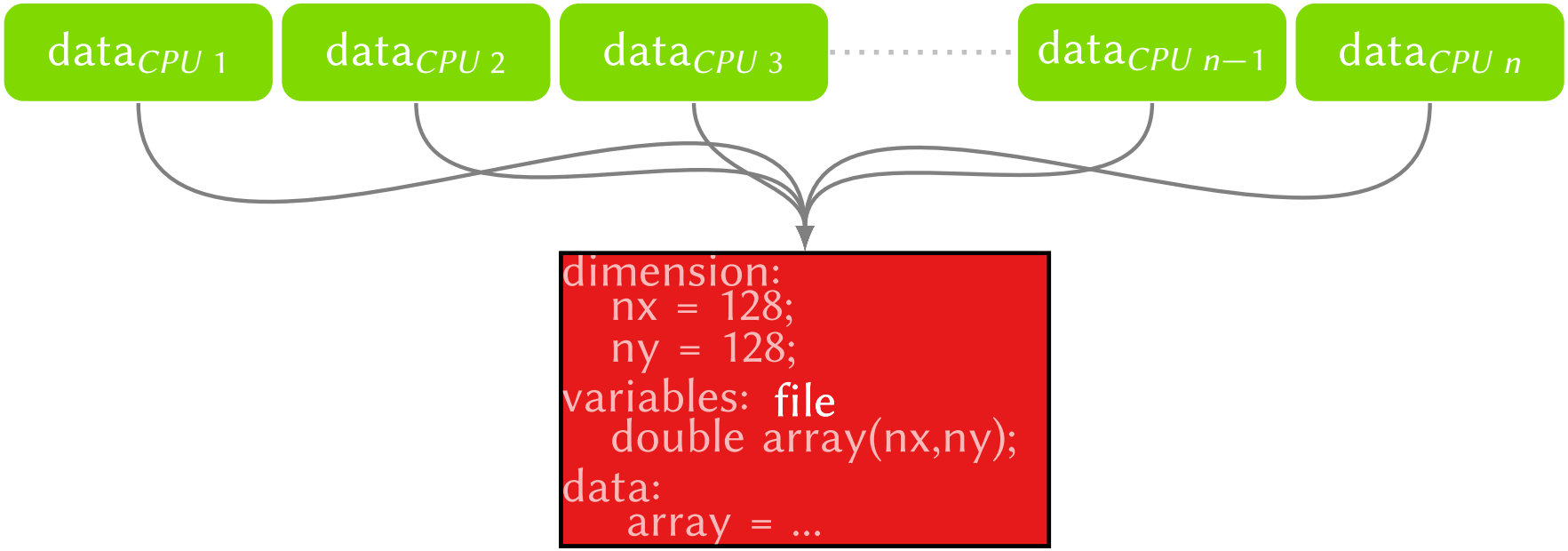
Parallel I/O approaches: MPI-IO

```
29 ! Write data
30 buf_size = 6*int_size + xdim*ydim*zdim*double_size*myrank
31 call MPI_File_Seek(fhandle, buf_size, MPI_SEEK_SET, ierr)
32 call MPI_File_Write_All(fhandle, data(:,:,:,1), xdim*ydim*zdim &
33         , MPI_DOUBLE_PRECISION, MPI_STATUS_IGNORE, ierr)
34
35 ! Close file
36 call MPI_File_Close(fhandle, ierr)
```

Parallel I/O approaches: Parallel NetCDF

NetCDF: **Net**work Common **D**ata **F**orm

→ **self-documented**, **portable** format



Parallel I/O approaches: Parallel NetCDF

```
integer :: xpos, ypos, zpos, myrank
real(8), dimension(xdim,ydim,zdim,nvar) :: data
character(LEN=13) :: filename

! PnetCDF variables
integer(kind=MPI_OFFSET_KIND) :: nxtot, nytot, nztot
integer :: nout, ncid, xdimid, ydimid, zdimid, vid1
integer, dimension(3) :: sdimid
integer(kind=MPI_OFFSET_KIND), dimension(3) :: dims, start, count
integer :: ierr

dims = (/ xdim, ydim, zdim /)

! Create file
filename = 'parallelio.nc'
nout = nfmpi_create(MPI_COMM_WORLD, filename, NF_CLOBBER, MPI_INFO_NULL &
        , ncid)

! Define dimensions
nout = nfmpi_def_dim(ncid, "x", nxtot, xdimid)
nout = nfmpi_def_dim(ncid, "y", nytot, ydimid)
nout = nfmpi_def_dim(ncid, "z", nztot, zdimid)
sdimid = (/ xdimid, ydimid, zdimid /)

! Create variable
nout = nfmpi_def_var(ncid, "var1", NF_DOUBLE, 3, sdimid, vid1)

! End of definitions
nout = nfmpi_enddef(ncid)
```

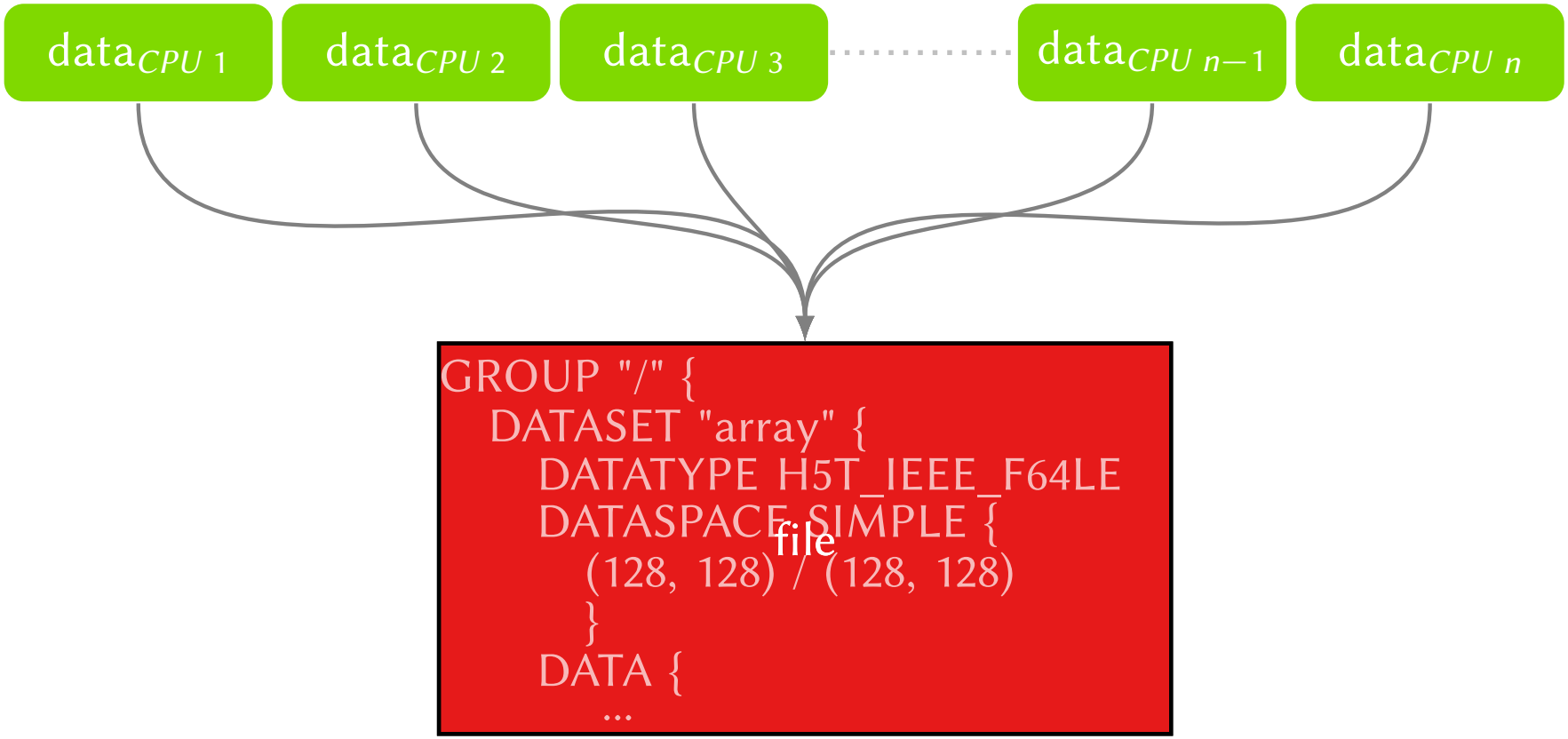
Parallel I/O approaches: Parallel NetCDF

```
31 start = (/ xpos, ypos, zpos /)*dims+1
32 count = dims
33
34 ! Write data
35 nout = nfmpi_put_vara_double_all(ncid, vid1, start, count, data(:,:,:,1))
36
37 ! Close file
38 nout = nfmpi_close(ncid)
```

Parallel I/O approaches: Parallel HDF5

HDF5: **H**ierarchical **D**ata **F**ormat

→ **self-documented**, **hierarchical**, **portable** format



Parallel I/O approaches: Parallel HDF5

```
1  integer :: xpos, ypos, zpos, myrank
2  real(8), dimension(xdim,ydim,zdim,nvar) :: data
3  character(LEN=13) :: filename
4
5  ! HDF5 variables
6  integer :: ierr
7  integer(HID_T) :: file_id, fapl_id, dxpl_id
8  integer(HID_T) :: h5_dspace, h5_dset, h5_dspace_file
9  integer(HSIZE_T), dimension(3) :: start, count, stride, blockSize
10 integer(HSIZE_T), dimension(3) :: dims, dims_file
11
12 ! Initialize HDF5 interface
13 call H5open_f(ierr)
14
15 ! Create HDF5 property IDs for parallel file access
16 filename = 'parallelio.h5'
17 call H5Pcreate_f(H5P_FILE_ACCESS_F, fapl_id, ierr)
18 call H5Pset_fapl_mpio_f(fapl_id, MPI_COMM_WORLD, MPI_INFO_NULL, ierr)
19 call H5Fcreate_f(filename, H5F_ACC_RDWR_F, file_id, ierr &
20      , access_prp=fapl_id)
21
22 ! Select space in memory and file
23 dims      = (/ xdim, ydim, zdim /)
24 dims_file = (/ xdim*nx, ydim*ny, zdim*nz /)
25
26 call H5Screate_simple_f(3, dims, h5_dspace, ierr)
27 call H5Screate_simple_f(3, dims_file, h5_dspace_file, ierr)
```

Parallel I/O approaches: Parallel HDF5

```
29 ! Hyperslab for selecting data in h5_dspace
30 start     = (/ 0, 0, 0 /)
31 stride    = (/ 1, 1, 1 /)
32 count     = dims
33 blockSize = (/ 1, 1, 1 /)
34 call H5Sselect_hyperslab_f(h5_dspace, H5S_SELECT_SET_F, start, count &
35      , ierr, stride, blockSize)
36
37 ! Hyperslab for selecting location in h5_dspace_file (to set the
38 ! correct location in file where we want to put our piece of data)
39 start     = (/ xpos, ypos, zpos /)*dims
40 stride    = (/ 1,1,1 /)
41 count     = dims
42 blockSize = (/ 1,1,1 /)
43 call H5Sselect_hyperslab_f(h5_dspace_file, H5S_SELECT_SET_F, start, count &
44      , ierr, stride, blockSize)
45
46 ! Enable parallel collective IO
47 call H5Pcreate_f(H5P_DATASET_XFER_F, dxpl_id, ierr)
48 call H5Pset_dxpl_mpio_f(dxpl_id, H5FD_MPIO_COLLECTIVE_F, ierr)
49
50 ! Create data set
51 call H5Dcreate_f(file_id, trim(dsetname), H5T_NATIVE_DOUBLE &
52      , h5_dspace_file, h5_dset, ierr, H5P_DEFAULT_F, H5P_DEFAULT_F &
53      , H5P_DEFAULT_F)
```

Parallel I/O approaches: Parallel HDF5

```
55 ! Finally write data to file
56 call H5Dwrite_f(h5_dset, H5T_NATIVE_DOUBLE, data, dims, ierr &
57      , mem_space_id=h5_dspace, file_space_id=h5_dspace_file &
58      , xfer_prp=dxpl_id)
59
60 ! Clean HDF5 IDs
61 call H5Pclose_f(dxpl_id, ierr)
62 call H5Dclose_f(h5_dset, ierr)
63 call H5Sclose_f(h5_dspace, ierr)
64 call H5Sclose_f(h5_dspace_file, ierr)
65 call H5Fclose_f(file_id, ierr)
66 call H5Pclose_f(fapl_id, ierr)
67 call H5close_f(ierr)
```

Parallel I/O approaches: ADIOS

```
1  integer :: xpos, ypos, zpos, myrank
2  real(8), dimension(xdim,ydim,zdim,nvar) :: data
3  character(LEN=17) :: filename
4
5  ! MPI & ADIOS variables
6  integer    :: adios_err
7  integer(8) :: adios_handle, offset_x, offset_y, offset_z
8  integer :: xdimglob, ydimglob, zdimglob
9  integer :: ierr
10
11 ! Init ADIOS
12 call ADIOS_Init("adios_BRIO.xml", MPI_COMM_WORLD, ierr)
13
14 ! Define offset and global dimensions
15 offset_x = xdim*xpos
16 offset_y = ydim*ypos
17 offset_z = zdim*zpos
18 xdimglob = xdim*nx; ydimglob = ydim*ny; zdimglob = zdim*nz
19
20 ! Open ADIOS file & write data
21 call ADIOS_Open(adios_handle, "dump", "parallelio_XML.bp", "w" &
22      & , MPI_COMM_WORLD, ierr)
23 ! Write I/O
24 # include "gwrite_dump.fh"
25
26 ! Close ADIOS file and interface
27 call ADIOS_Close(adios_handle, ierr)
28 call ADIOS_Finalize(myrank, ierr)
```

Parallel I/O approaches: ADIOS

with the following XML file for definitions:

```
1  <?xml version="1.0"?>
2  <adios-config host-language="Fortran">
3     <adios-group name="dump" coordination-communicator="MPI_COMM_WORLD">
4     <global-bounds dimensions="xdimglob, ydimglob, zdimglob"
5             offsets="offset_x, offset_y, offset_z">
6        <var name="var1" gwrite="data(:,:,:,1)" type="double"
7             dimensions="xdim, ydim, zdim"/>
8     </global-bounds>
9  </adios-group>
10
11 <method group="dump" method="MPI"/>
12
13 <buffer size-MB="10" allocate-time="now"/>
14
15 </adios-config>
```

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library
—
MPI-IO
PHDF5
PnetCDF
ADIOS

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library	ease of use
—	✓
MPI-IO	–
PHDF5	X
PnetCDF	–
ADIOS	✓✓

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library	ease of use	1 file
—	✓	X
MPI-IO	–	✓
PHDF5	X	✓
PnetCDF	–	✓
ADIOS	✓✓	✓

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library	ease of use	1 file	portability
—	✓	X	X
MPI-IO	–	✓	X
PHDF5	X	✓	✓
PnetCDF	–	✓	✓
ADIOS	✓✓	✓	✓

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library	ease of use	1 file	portability	self-documented
—	✓	X	X	X
MPI-IO	–	✓	X	X
PHDF5	X	✓	✓	✓
PnetCDF	–	✓	✓	✓
ADIOS	✓✓	✓	✓	✓

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library	ease of use	1 file	portability	self-documented	flexibility
—	✓	X	X	X	✓
MPI-IO	–	✓	X	X	–
PHDF5	X	✓	✓	✓	✓
PnetCDF	–	✓	✓	✓	✓
ADIOS	✓✓	✓	✓	✓	✓✓

Parallel I/O approaches

Possible approaches:

- sequential I/O
- master process distributing data
- MPI-IO
- Parallel HDF5
- Parallel NetCDF
- ADIOS
- ...

library	ease of use	1 file	portability	self-documented	flexibility	interface
—	✓	X	X	X	✓	X
MPI-IO	–	✓	X	X	–	X
PHDF5	X	✓	✓	✓	✓	✓
PnetCDF	–	✓	✓	✓	✓	✓
ADIOS	✓✓	✓	✓	✓	✓✓	–

BRIO: a benchmark for parallel I/O

Tested libraries:

- sequential I/O
- MPI-IO
- parallel HDF5
- parallel NetCDF
- ADIOS

What does it do?

- write/read data distributed on a cartesian grid
- compute writing/reading time
- few parameters:
  - size of the grid
  - domain decomposition
  - contiguity of data
  - XML/noXML interface (for ADIOS only)
- under GNU/GPL license, available at `https://bitbucket.org/mjoos`

Results on Turing (BG/Q)

# MPI threads	library	contiguous	t_{writing} [s]
4096	—	—	9.602
	HDF5	—	9.337
	parallel HDF5	✓	29.226
	parallel NetCDF	X	8.394
		✓	5.941
16 384	—	—	12.129
	parallel HDF5	✓	109.419
	parallel NetCDF	X	12.165
		✓	9.557
131 072	—	—	100.197
	parallel NetCDF	✓	47.592

Results on Turing (BG/Q)

# MPI threads	library	contiguous	t_{writing} [s]
4096	—	—	9.602
	HDF5	—	9.337
	parallel HDF5	✓	29.226
	parallel NetCDF	X	8.394
		✓	5.941
16 384	—	—	12.129
	parallel HDF5	✓	109.419
	parallel NetCDF	X	12.165
		✓	9.557
131 072	—	—	100.197
	parallel NetCDF	✓	47.592

Results on Turing (BG/Q)

# MPI threads	library	contiguous	t_{writing} [s]
4096	—	—	9.602
	HDF5	—	9.337
	parallel HDF5	✓	29.226
	parallel NetCDF	X	8.394
		✓	5.941
16 384	—	—	12.129
	parallel HDF5	✓	109.419
	parallel NetCDF	X	12.165
		✓	9.557
131 072	—	—	100.197
	parallel NetCDF	✓	47.592

Outline

Why hybridize codes?

Advantages

- better match of modern architectures:
  interconnected nodes with shared memory

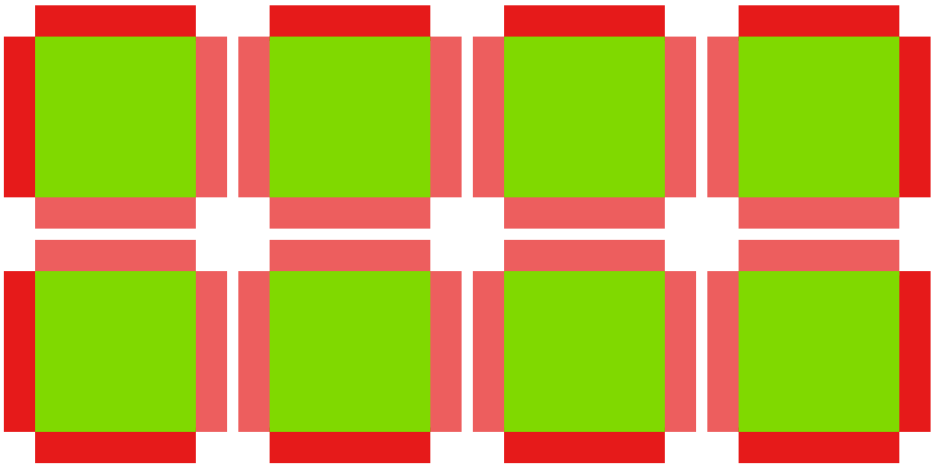
Why hybridize codes?

Advantages

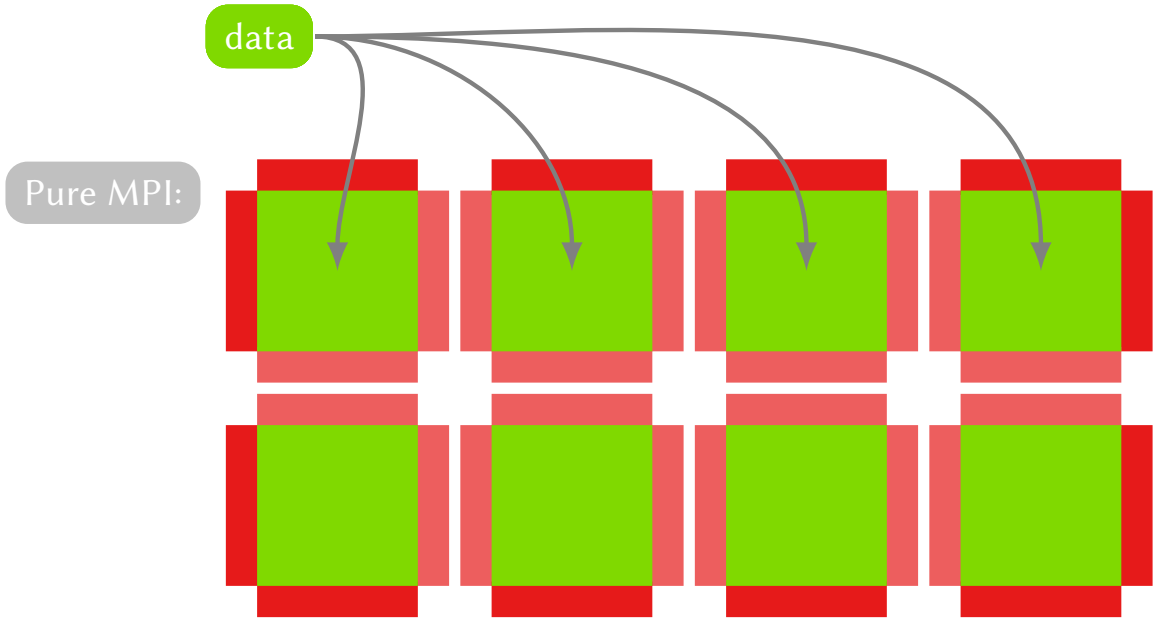
- better match of modern architectures:
  interconnected nodes with shared memory
- optimized memory usage:
  - less data duplicated by MPI processes
  - lower memory footprint

Why hybridize codes?

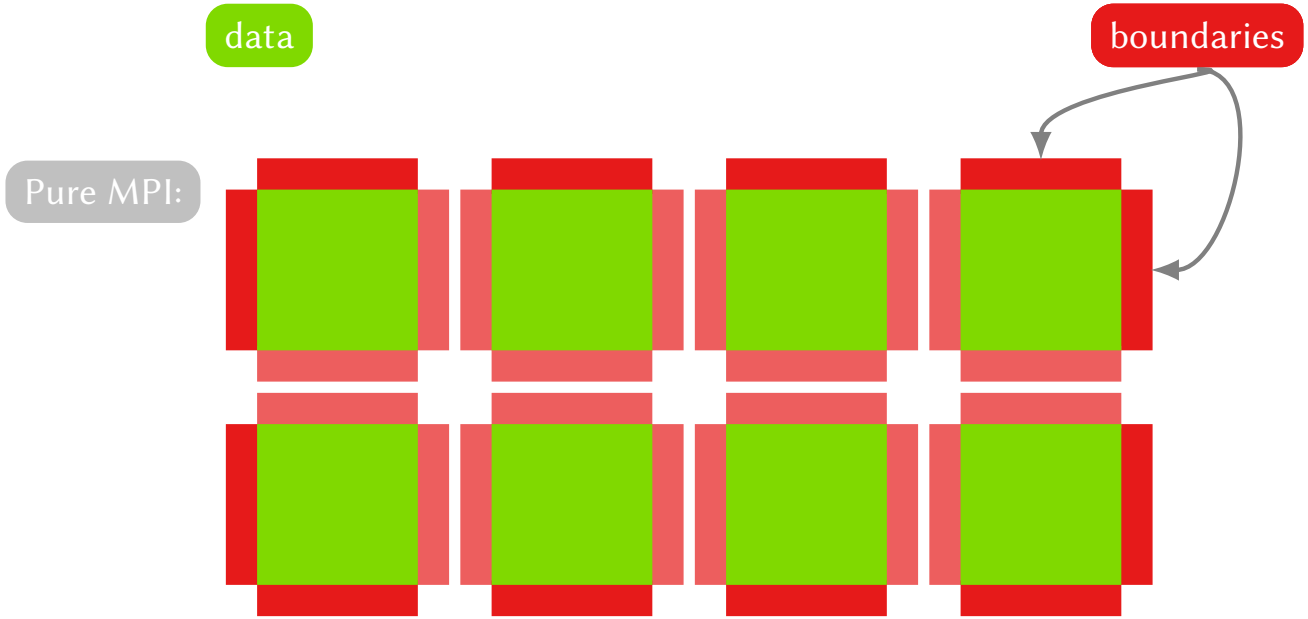
Pure MPI:



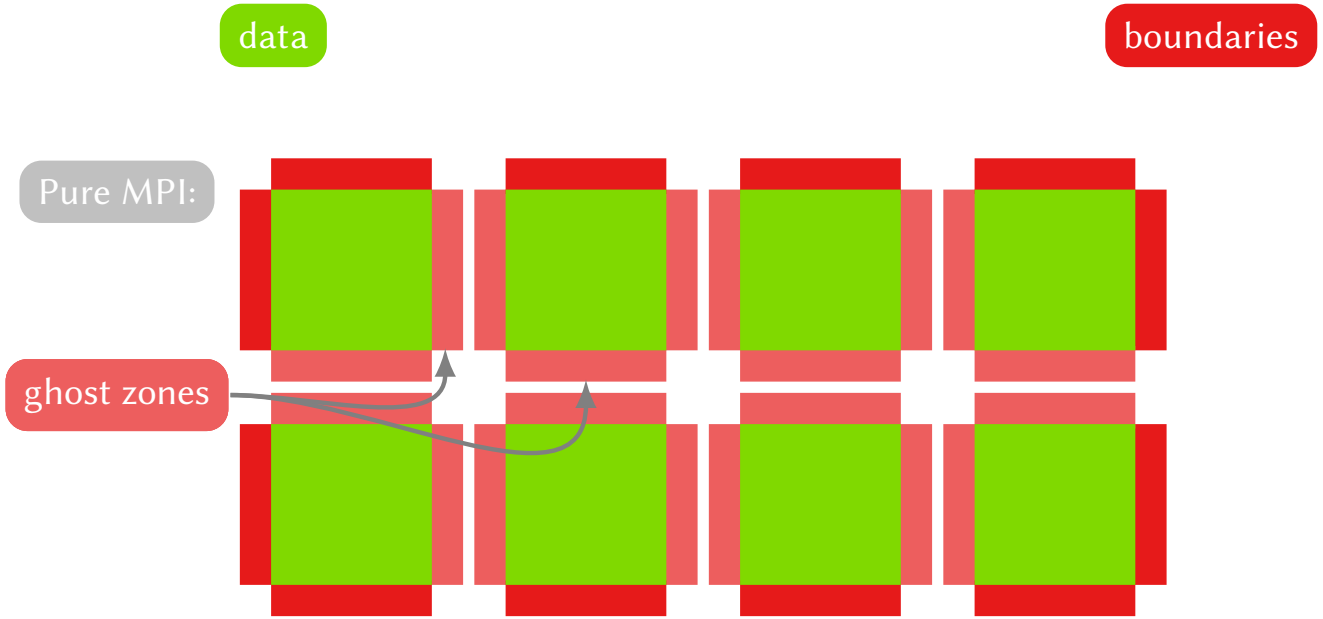
Why hybridize codes?



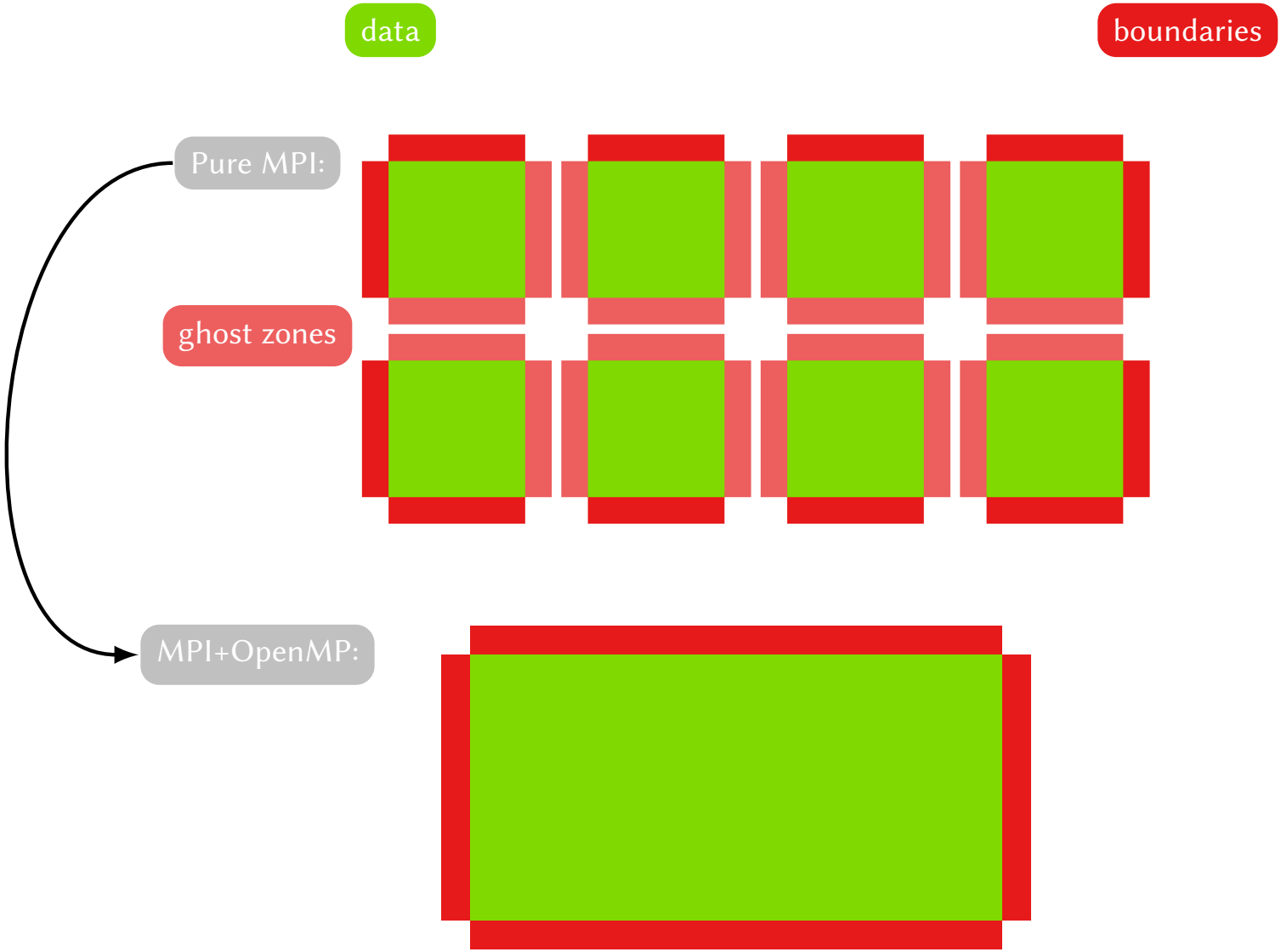
Why hybridize codes?



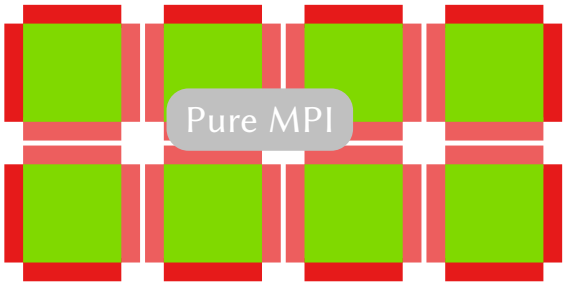
Why hybridize codes?

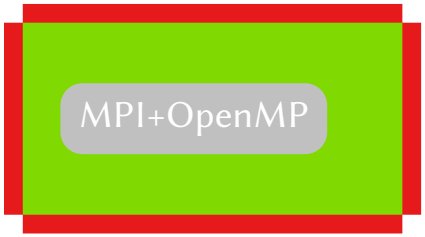


Why hybridize codes?



Why hybridize codes?





Ex: $800 \times 1600 \times 832$ domains, 11 variables (double precision)

# MPI processes	# OpenMP threads	Size (in Go)
131 072	1	197
16 384	8	135

memory gain: $>30\%$

Why hybridize codes?

Advantages

- better match of modern architectures:
  interconnected nodes with shared memory
- optimized memory usage:
  - less data duplicated by MPI processes
  - lower memory footprint
- better I/O performances:
  - less simultaneous access
  - less operations with bigger datasets
  - less files (without parallel I/O)

Why hybridize codes?

Advantages

- better match of modern architectures:
  interconnected nodes with shared memory
- optimized memory usage:
  - less data duplicated by MPI processes
  - lower memory footprint
- better I/O performances:
  - less simultaneous access
  - less operations with bigger datasets
  - less files (without parallel I/O)
- better granularity:
  - MPI program: compute and communicate
  - **granularity**: ratio between computing and communication steps
  - the larger the granularity, the better the extensivity

Hybridation of Ramses

Main steps:

1. **reorganize the code**
   → from 42 to 10 source files, reorganization of the modules etc.
2. **Fine-Grain approach**: parallelize external loops
3. **MPI communications**

```
1  integer, parameter :: nx=512, ny=512, nz=1024
2  real(8), dimension(:,:,:), allocatable :: array
3  integer :: i, j, k
4
5  allocate(array(nx,ny,nz))
6
7  do k = 1, nz
8     do j = 1, ny
9        do i = 1, nx
10          array(i,j,k) = (k*j + i)*1.
11       enddo
12    enddo
13 enddo
14
15 deallocate(array)
```

```
1  integer, parameter :: nx=512, ny=512, nz=1024
2  real(8), dimension(:,:,:), allocatable :: array
3  integer :: i, j, k
4
5  allocate(array(nx,ny,nz))
6
7  !$OMP PARALLEL
8  !$OMP DO SCHEDULE(RUNTIME)
9  do k = 1, nz
10    do j = 1, ny
11       do i = 1, nx
12          array(i,j,k) = (k*j + i)*1.
13       enddo
14    enddo
15 enddo
16 !$OMP END DO
17 !$OMP END PARALLEL
18
19 deallocate(array)
```

Hybridation of Ramses

Preliminary results:

Poincaré@Maison de la Simulation (16 cores per node, Intel Sandy Bridge):

# MPI proc.	# OpenMP threads	t_{elapsed} [s]
16	1	17.49
8	2	17.04
4	4	16.85
2	8	20.07

Turing@IDRIS (1024 cores per node, PowerPC A2):

# MPI proc.	# OpenMP threads	t_{elapsed} [s]
2048	1	355.9
1024	2	337.9
512	4	366.1

Auto-parallelization: does it worth it?

What is it?

- compilers can detect serial portions of the code that can be multithreaded
  → equivalent to an OpenMP parallelization

How to use it?

compiler	option
Intel	`-parallel`
IBM	`-qsmp=auto`
PGI	`-Mconcur`

How to help your compiler?

compiler	pragma
Intel	`parallel`
IBM	—
PGI	`concur[/noconcur]`

Auto-parallelization: does it worth it?

Simple example:

```
1  integer, parameter :: nx=512, ny=512, nz=1024
2  real(8), dimension(:,:,:), allocatable :: array
3  integer :: i, j, k
4
5  allocate(array(nx,ny,nz))
6
7  !$OMP PARALLEL
8  !$OMP DO SCHEDULE(RUNTIME)
9  do k = 1, nz
10    do j = 1, ny
11       do i = 1, nx
12          array(i,j,k) = (k*j + i)*1.
13       enddo
14    enddo
15 enddo
16 !$OMP END DO
17 !$OMP END PARALLEL
18
19 deallocate(array)
```

Results:

Intel compiler:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	0.5174	0.5187
2	0.2454	0.2383
4	0.1302	0.1272
8	0.07820	0.06650

PGI compiler:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	0.6510	1.049
2	0.3092	0.5067
4	0.1559	0.2580
8	0.08200	0.1332

Auto-parallelization: does it worth it?

Intermediate example:

```
1  !$omp parallel shared(a,anew,error,iter)
2  do
3     error = 0.d0
4
5     !$omp do reduction(max:error) schedule(runtime)
6     do j = 2, m-1
7        do i = 2, n-1
8           anew(i,j) = 0.25*(a(i,j+1) + a(i,j-1) &
9                + a(i-1,j) + a(i+1,j))
10          error = max(error,abs(anew(i,j)-a(i,j)))
11       enddo
12    enddo
13
14    !$omp do schedule(runtime)
15    do j = 2, m-1
16       do i = 2, n-1
17          a(i,j) = anew(i,j)
18       enddo
19    enddo
20
21    if((error .lt. tolerance) .or. &
22    (iter-1 .gt. iter_max)) exit
23    !$omp single
24    if(mod(iter,10).eq.0) print*, iter, error
25    iter = iter + 1
26    !$omp end single
27 enddo
28 !$omp end parallel
```

Results:

Intel compiler:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	0.0627	0.0856
2	0.0593	0.0469
4	0.0518	0.0251
8	0.0581	0.0154
16	0.0796	0.0122

PGI compiler:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	0.175	0.194
2	0.101	0.0967
4	0.0643	0.0509
8	0.0440	0.0285
16	0.0344	0.0177

Auto-parallelization: does it worth it?

"Real" example:

Tested on RAMSES:

- Intel compiler, with 4 MPI processes:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	656.5	777.6
2	540.5	424.2
4	491.4	246.4

Auto-parallelization: does it worth it?

"Real" example:

Tested on RAMSES:

- Intel compiler, with 4 MPI processes:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	656.5	777.6
2	540.5	424.2
4	491.4	246.4

- IBM compiler, with 1024 MPI processes:

# threads	t_{auto} [s]	t_{OpenMP} [s]
1	587.7	582.3
2	340.3	337.9
4	218.5	216.0

Outline

Why do want to do astrophysics on GPUs?

Pro:

✓ sheer computing power

hardware	processing power [GFLOPS]
Intel Sandy Bridge (single core)	24.6
Intel Sandy Bridge (whole chip)	157.7
NVIDIA Tesla Kepler 20 (SP)	4106
NVIDIA Tesla Kepler 20 (DP)	1173

Why do want to do astrophysics on GPUs?

Pro:

- ✓ sheer computing power
- ✓ weak scaling on RAMSES *(RAMSES-GPU doc., P. Kestener)*

# MPI proc.	Global size	perf$_{CPU}$ [update/s]	perf$_{GPU}$ [update/s]
1	128×128×128	0.21	13.6
8	256×256×256	1.68	95.3
64	512×512×512	13.4	750.3
128	1024×512×512	26.8	1498.3
256	1024×1024×512	52.5	2969.3

Why do want to do astrophysics on GPUs?

Pro:

- ✓ sheer computing power
- ✓ weak scaling on RAMSES *(RAMSES-GPU doc., P. Kestener)*

# MPI proc.	Global size	perf_{CPU} [update/s]	perf_{GPU} [update/s]
1	$128\times128\times128$	0.21	13.6
8	$256\times256\times256$	1.68	95.3
64	$512\times512\times512$	13.4	750.3
128	$1024\times512\times512$	26.8	1498.3
256	$1024\times1024\times512$	52.5	2969.3

Cons:

- X CUDA is a C library
  - → need to translate scientific codes in C/C++

Why do want to do astrophysics on GPUs?

Pro:

- ✓ sheer computing power
- ✓ weak scaling on RAMSES *(RAMSES-GPU doc., P. Kestener)*

# MPI proc.	Global size	perf_{CPU} [update/s]	perf_{GPU} [update/s]
1	128×128×128	0.21	13.6
8	256×256×256	1.68	95.3
64	512×512×512	13.4	750.3
128	1024×512×512	26.8	1498.3
256	1024×1024×512	52.5	2969.3

Cons:

- X CUDA is a C library
  - → need to translate scientific codes in C/C++
- X CUDA is not *memory management*-friendly
  - → need to rethink the algorithms

Why do want to do astrophysics on GPUs?

Pro:

- ✓ sheer computing power
- ✓ weak scaling on RAMSES *(RAMSES-GPU doc., P. Kestener)*

# MPI proc.	Global size	perf_{CPU} [update/s]	perf_{GPU} [update/s]
1	128×128×128	0.21	13.6
8	256×256×256	1.68	95.3
64	512×512×512	13.4	750.3
128	1024×512×512	26.8	1498.3
256	1024×1024×512	52.5	2969.3

Cons:

- X CUDA is a C library
  - → need to translate scientific codes in C/C++
- X CUDA is not *memory management*-friendly
  - → need to rethink the algorithms

⇒ 1.5 year to recode RAMSES

OpenACC: the way to go?

What is it?

- Compiler directives to specify loops and regions to offload from CPU to accelerator
- no need to explicitely manage data
- Fortran friendly!

OpenACC: the way to go?

```
tol      = 1.d-6
iter_max = 1000

do while ( error .gt. tol .and. iter .lt. iter_max )
   error=0.d0

   do j = 1, m-2
      do i = 1, n-2
         Anew(i,j) = 0.25 *(A(i+1,j) + A(i-1,j) &
                   + A(i,j-1) + A(i,j+1))
         error = max( error, abs(Anew(i,j) - A(i,j)))
      end do
   end do

   if(mod(iter,100).eq.0) print*, iter, error
   iter = iter + 1

   do j = 1, m-2
      do i = 1, n-2
         A(i,j) = Anew(i,j)
      end do
   end do

end do
```

- $t_{CPU} = 0.177$ s

OpenACC: the way to go?

```
tol      = 1.d-6
iter_max = 1000

do while ( error .gt. tol .and. iter .lt. iter_max )
   error=0.d0

   !$acc kernels loop
   do j = 1, m-2
      do i = 1, n-2
         Anew(i,j) = 0.25 *(A(i+1,j) + A(i-1,j) &
                   + A(i,j-1) + A(i,j+1))
         error = max( error, abs(Anew(i,j) - A(i,j)))
      end do
   end do

   if(mod(iter,100).eq.0) print*, iter, error
   iter = iter + 1

   !$acc kernels loop
   do j = 1, m-2
      do i = 1, n-2
         A(i,j) = Anew(i,j)
      end do
   end do

end do
```

- $t_{CPU} = 0.177$ s
- $t_{GPU} = 0.149$ s

OpenACC: the way to go?

```
1  tol      = 1.d-6
2  iter_max = 1000
3
4  !$acc data copy(A) create(Anew)
5  do while ( error .gt. tol .and. iter .lt. iter_max )
6     error=0.d0
7
8     !$acc kernels loop
9     do j = 1, m-2
10       do i = 1, n-2
11          Anew(i,j) = 0.25 *(A(i+1,j) + A(i-1,j) &
12                     + A(i,j-1) + A(i,j+1))
13          error = max( error, abs(Anew(i,j) - A(i,j)))
14       end do
15    end do
16
17    if(mod(iter,100).eq.0) print*, iter, error
18    iter = iter + 1
19
20    !$acc kernels loop
21    do j = 1, m-2
22       do i = 1, n-2
23          A(i,j) = Anew(i,j)
24       end do
25    end do
26
27 end do
28 !$acc end data
```

- $t_{CPU} = 0.177$ s
- $t_{GPU} = 0.149$ s
- $t_{\text{GPU data}} = 0.00667$ s

Conclusions & prospects

Physics:

1. **Asymptotic convergence** of α at low P_m
2. **Hydrodynamics cascade** at small scales
3. Simulation still running on Turing to **confirm the result**

Conclusions & prospects

Physics:

1. **Asymptotic convergence** of α at low P_m
2. **Hydrodynamics cascade** at small scales
3. Simulation still running on Turing to **confirm the result**

Numerics:

1. **parallel I/O** are mature enough to be used:
   - (very) good performance
   - **more and more easy to use**
   - unavoidable to go to exascale
2. **need hybridation** (OpenMP/OpenACC + MPI) to go to keep going with the increasing computational power
3. **GPU are not so hard to use**
4. **Don't worry!** your compilers become smarter and smarter

Conclusions & prospects

Physics:

1. **Asymptotic convergence** of α at low P_m
2. **Hydrodynamics cascade** at small scales
3. Simulation still running on Turing to **confirm the result**

Numerics:

1. **parallel I/O** are mature enough to be used:
   - (very) good performance
   - **more and more easy to use**
   - unavoidable to go to exascale
2. **need hybridation** (OpenMP/OpenACC + MPI) to go to keep going with the increasing computational power
3. **GPU are not so hard to use**
4. **Don't worry!** your compilers become smarter and smarter

Thank you for your attention!