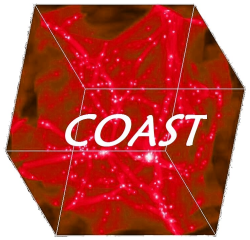
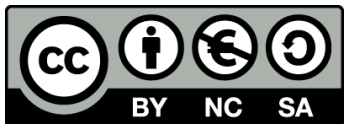
DUMSES-HYBRID

HOW TO USE AND DEVELOP IT

Marc Joos

2015, June 19th





Outline

Outline

What's new in Dumses-Hybrid?

Dumses is still:

- a 3D Eulerian second-order Godunov (magneto)hydrodynamic simulation code
- in cartesian, cylindrical and spherical coordinates
- with a fixed grid

But now:

- hybridized with OpenMP
- hybridized with OpenACC (for GPU)
- with parallel I/O
- with a new "user-friendly" configuration/compilation interface

Get the code:

it is now publicly available on SourceSup:

- `git clone git://git.renater.fr/dumses.git`

Outline

How to compile and launch the code

Do it in four steps:

- `./configure`
- `./make.py`
- `cp bin/dumses src/problem/your-problem/input $RUNDIR`
- `[mpirun -np N] ./dumses`

And don't forget to set your local variables:

- `export OMP_NUM_THREADS=N`
- `export ACC_DEVICE_TYPE='nvidia'`

Outline

Performances

Test: MRI with no dissipation, $128 \times 128 \times 128$

- with PGI compiler
- CPU: Intel SandyBridge
- GPU: NVIDIA K20c

	Architecture	# MPI th.	# OpenMP th.	$t_{elapsed}$ [s]
`dumses_mpi`	CPU	1	1	15.7
	CPU	4	1	4.1
`dumses_hybrid`	CPU	1	1	9.4
	CPU	1	4	2.9
	CPU	4	1	2.6
	GPU	1	1	0.81

Performances

Test: MRI with no dissipation, $128 \times 128 \times 128$

- with PGI compiler
- CPU: Intel SandyBridge
- GPU: NVIDIA K20c

	Architecture	# MPI th.	# OpenMP th.	$t_{elapsed}$ [s]
`dumses_mpi`	CPU	1	1	15.7
	CPU	4	1	4.1
`dumses_hybrid`	CPU	1	1	9.4
	CPU	1	4	2.9
	CPU	4	1	2.6
	GPU	1	1	0.81

Performances

Test: MRI with no dissipation, $128 \times 128 \times 128$

- with PGI compiler
- CPU: Intel SandyBridge
- GPU: NVIDIA K20c

	Architecture	# MPI th.	# OpenMP th.	$t_{elapsed}$ [s]
`dumses_mpi`	CPU	1	1	15.7
	CPU	4	1	4.1
`dumses_hybrid`	CPU	1	1	9.4
	CPU	1	4	2.9
	CPU	4	1	2.6
	GPU	1	1	0.81

Performances

Test: MRI with no dissipation, 128×128×128

- with PGI compiler
- CPU: Intel SandyBridge
- GPU: NVIDIA K20c

	Architecture	# MPI th.	# OpenMP th.	$t_{elapsed}$ [s]
`dumses_mpi`	CPU	1	1	15.7
	CPU	4	1	4.1
`dumses_hybrid`	CPU	1	1	9.4
	CPU	1	4	2.9
	CPU	4	1	2.6
	GPU	1	1	0.81

DUMSES-Hybrid vs. DUMSES:

- on CPU: 1.7× faster
- on GPU: 20× faster

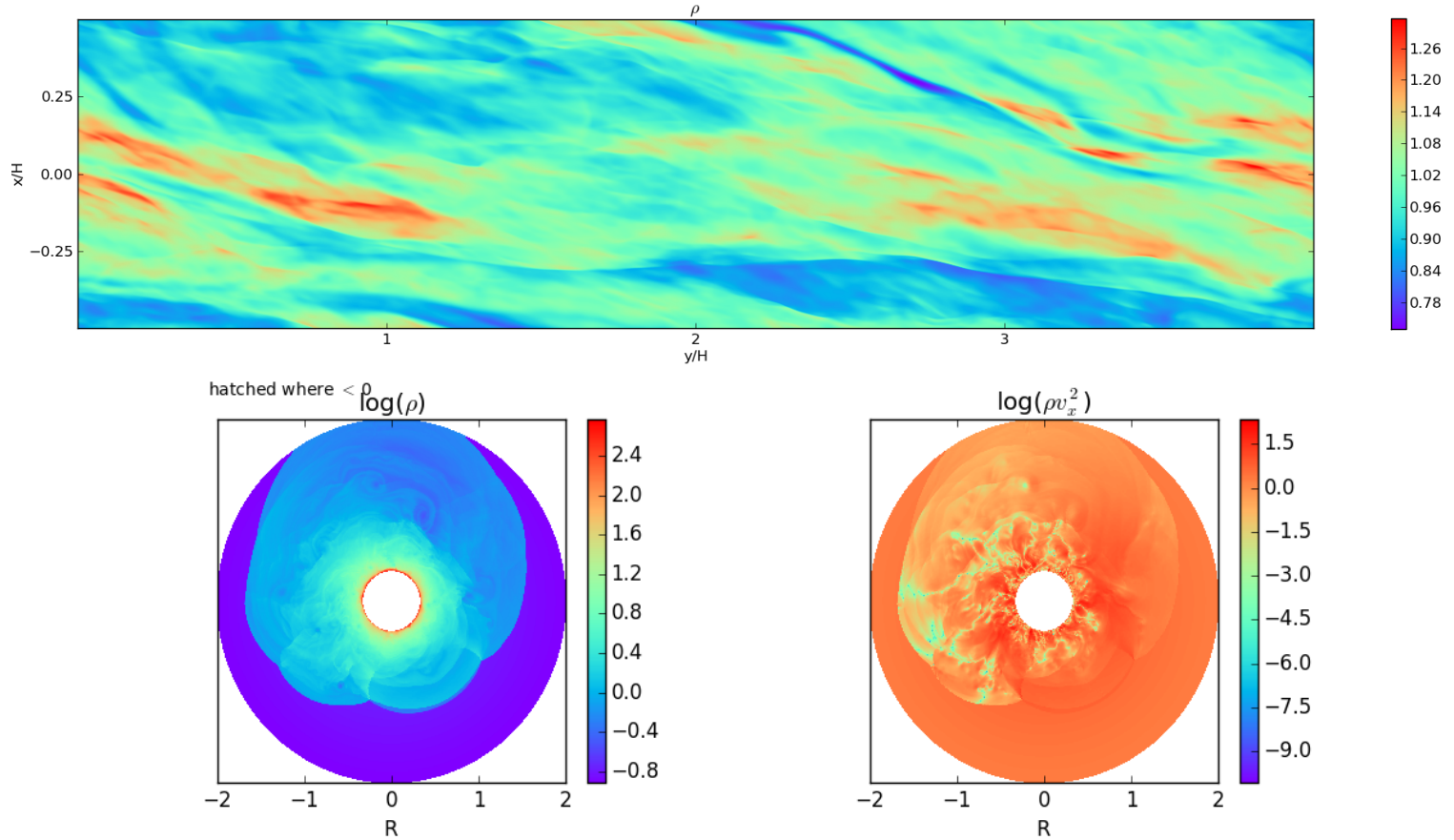
Outline

How to read'n'visualize your simulation

With Python!

If you use dumpy for the first time:

- `cd $DUMSES/utils/dumpy/`
- `python setup.py install`



Outline

Code architecture

Some highlights:

- as a general rule, never touch `src/dumses.f90`, `src/modules/*` and `src/subroutines/*` files. If you want to develop on DUMSES-Hybrid, just add your problem in `src/problem`

Code architecture

Some highlights:

- as a general rule, never touch `src/dumses.f90`, `src/modules/*` and `src/subroutines/*` files. If you want to develop on DUMSES-Hybrid, just add your problem in `src/problem`
- if you develop a new problem, you shouldn't worry about OpenMP: you can transparently add your code and it will work (though you won't get speed-up due to OpenMP)

```
1  !$OMP PARALLEL DO SCHEDULE(RUNTIME)
2  do k=1, khi
3     do j=1, jhi+1
4        do i=1, ihi+1
5           uin(i,j,k,iA) = uin(i,j,k,iA) &
6                         + (emfz(i,j+1,k) - emfz(i,j,k))/dy
7           uin(i,j,k,iB) = uin(i,j,k,iB) &
8                         - (emfz(i+1,j,k) - emfz(i,j,k))/dx
9        end do
10    end do
11 end do
12 !$OMP END PARALLEL DO
```

Code architecture

Some highlights:

- as a general rule, never touch `src/dumses.f90`, `src/modules/*` and `src/subroutines/*` files. If you want to develop on DUMSES-Hybrid, just add your problem in `src/problem`
- if you develop a new problem, you shouldn't worry about OpenMP: you can transparently add your code and it will work (though you won't get speed-up due to OpenMP)
- solvers are generated by a home-made Python preprocessor, as well as subroutine timing – but you'd probably never have to worry about it

```
1 !$py start_timing Timestep
2 call compute_dt(dt)
3 !$py end_timing Timestep
```

gives:

```
1 if (verbose) call system_clock(count=t0, count_rate=irate)
2 call compute_dt(dt)
3 if (verbose) then
4   call system_clock(count=t1, count_rate=irate)
5   print '("timestep:   ", F12.8, " s")', (t1 - t0)/(irate*1.d0)
6 endif
```

Hybridation on GPU

Goals:

- extend Dumses capabilities to prepare the future of HPC
- be as little invasive as possible and stay in Fortran

$\Rightarrow$ solution: OpenACC

Strategy:

- *à la* OpenMP: parallelization of external loops
- tricky point: data transfer to/from the device

```
1  !$acc data create(emfz)
2  !$acc kernels loop
3  do k=1, khi
4     do j=1, jhi+1
5        do i=1, ihi+1
6           uin(i,j,k,iA) = uin(i,j,k,iA) &
7                         + (emfz(i,j+1,k) - emfz(i,j,k))/dy
8           uin(i,j,k,iB) = uin(i,j,k,iB) &
9                         - (emfz(i+1,j,k) - emfz(i,j,k))/dx
10       end do
11    end do
12 end do
```

Development cycle and feedback

Development cycle:

- code refactoring and OpenMP hybridation:
  → $\sim$ 6 months for the compute core
- OpenACC hybridation:
  → $\sim$ 6 more months
  - more refactoring (no call in parallelized loops)
  - first *naive* step: parallelization following OpenMP: $\times 0.1$ speedup (yep, that shouldn't be called a "speedup")
  - second step caring about data transfer: $\times 10$ speedup (on the good days)
  - last step of optimization taking care of compute kernels configuration, register sizes and so on: $\times 20$ speedup (and that's solid!)

Feedback:

- debugging is painful
  → tools to dump random variables and manipulate them
- debbuging (and profiling) on GPU is even more painful
  - NVIDIA tools (they are cool, but the learning curve is steep)
  - PGI tools (profiling, parallelization informations at compile time...)

Outline

Tests suite & continuous integration

Suite of tests:

- basic 1D, 2D and 3D tests in all 3 directions in space (shock tube, Orszag-Tang and so on)
- shearing box and MRI tests
- support MPI, OpenMP, and OpenACC

Jenkins:

- run every night the tests suite on a server, on monoprocessor, with MPI, OpenMP and OpenACC
- send email with results in case of success
- send email with log in case of failure

Jenkins

Outline

Documentation

Code documentation with Doxygen

- basic header for every file (with a short description, authors, licenses & dates of creation/modification)
- short documentation for every subroutines

User manual

- code configuration, compilation and execution
- detailed input parameters
- visualization
- how to use the tests suite
- how to develop in Dumses-Hybrid
- how to convert output format, including from older version of the code