Master in
High Performance Computing

MHPC BPiSSD

Graziano Giuliani – ICTP ESP

https://github.com/graziano-giuliani/MHPC_BPSSD

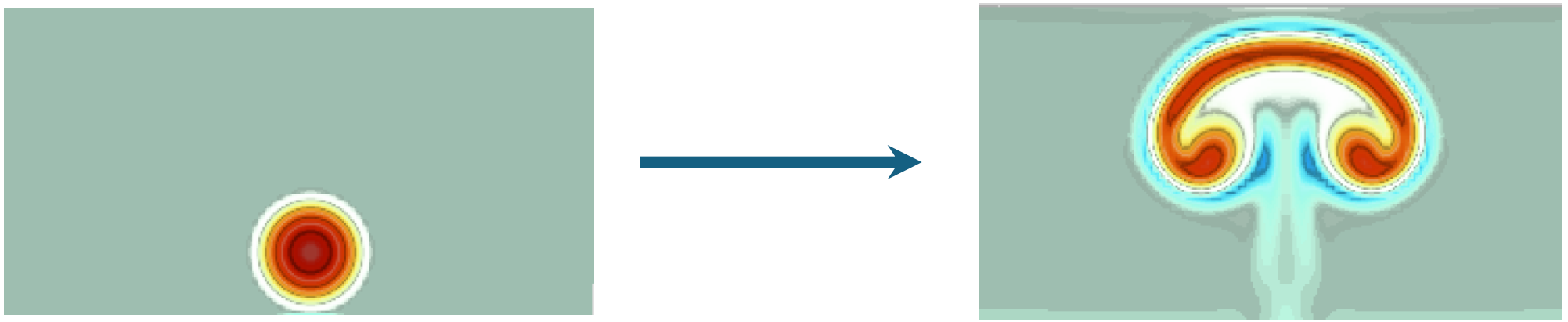
The Abdus Salam
International Centre
for Theoretical Physics

Target for this week

- Group work on a mock Scientific Software project
  - Project Manager and Developers
  - Interaction with Domain Expert Scientists
  - Roadmap, milestones and Release
- Apply languages, techniques and tools from completed courses
  - Fortran programming language
  - Parallelization of a code
  - Build environment and libraries
  - Collaborative development version control
  - Testing and regression analysis

The Thermal Blob code

- 2D geo fluid dynamic code in Fortran
  - X dimension is periodic
  - Serial solver with simple discretization
  - NetCDF output with use of library



The Developer's task (the exam pass)

- Parallelize the code using any of the parallel execution framework you have seen so far in the courses, or any valid possible combinations thereof
  - OpenMP, OpenAcc, MPI
- Constraints
  - Cannot change programming language
  - Results cannot change more than type precision
  - Each developer works on a branch
  - Only the PM can merge after checking test results
  - You must have ready by Friday afternoon a presentation with benchmark(s) to show results and describe the development process.

The BONUS tasks

- Implement more than one parallel version
- Use cmake/fpm for the build
- Document the code inline (Doxygen/FORD)
- Code the parallel output by using a parallel netCDF implementation (netCDF4, PnetCDF)

The Scientific Problem

The set of governing equations for the atmosphere and the ocean are examples of system of nonlinear partial differential equations (PDE's), to be solved on some spatial domain $\mathscr{D}$ and time interval $[0, t_f]$, given suitable boundary (BC) and initial (IC) conditions. Therefore the typical problem to be solved in ESM is an intial/boundary value problem of the (general) form:

$$\begin{aligned} \frac{\partial \psi}{\partial t} &= \mathscr{L}(\psi) \\ \psi(0) &= \psi_0 \\ \psi(\mathcal{B}) &= \Psi_{\mathcal{B}}(t) \end{aligned} \tag{3}$$

where

- $\psi(\mathbf{x}, t)$ is a status variable funcion of space and time
- $\mathscr{L}$ is a generally non linear differential operator.

The Space discretization

The original domain $\mathscr{D}$ is replaced by a spatially discretized domain $\mathscr{D}_h$ and the original problem is replaced by:

$$\frac{\partial \phi_i}{\partial t} = \mathscr{L}_h(\phi)_i, \quad i = 1, 2, \ldots m \tag{4}$$

$$\phi_i(0) = (\phi_0)_i, \quad i = 1, 2, \ldots m \tag{5}$$

$$\phi(\mathcal{B}_h) = \Phi_{\mathcal{B}_h}(t) \tag{6}$$

where:

- $\mathscr{L}_h$denotes a discrete approximation of the continuous differential operator $\mathscr{L}$
- h denotes the tipical size of the discrete spatial elements and determine the resolution of the spatial discretization.

The finite differences

- $\mathscr{D} = x_i, \quad i = 1, \ldots m$, called grid, is a regular array of discrete locations, called nodes. $h = \Delta x$ is the average spacing between nodes.
- $\phi_i(t) \approx \psi(x_i, t)$
- $\mathscr{L}_h$ is obtained by replacing derivatives present in $\mathscr{L}$ with finite difference quotients

Example in 1D:

- If $\mathscr{D} = [0, L]$ and $\mathscr{L}(\psi) = \psi'$, then
  - $\mathscr{D} = \{x_i = ih, \quad i = 1, \ldots m\}$ with $h = \Delta x = L/m$
  - $\mathscr{L}_h(\phi)_i = {}^{(\phi_{i+1} - \phi_{i-1})}/_{2h}$

The time discretization

The space discretized time continuous solution $\psi_i(t), \quad t \in [0, t_f]$ is approximated by introducing a timestep $\Delta t = {}^{t_f}/_{n}$ and a set of discrete time levels $t_k = k\Delta t, \quad k = 0, \ldots, n$. The choice of Δt must comply with numerical CFL stability condition.
Numerical ODE's methods give the fully discrete approximated solution:

$$\psi_i^k, \quad k = 0, \ldots, n; \quad i = 1, \ldots, m \tag{17}$$

Main approches include:

1. explicit schemes
2. semi-implicit schemes
3. semi-Lagrangian shemes

The Leapfrog scheme

The method do not require the solution of a system at each timestep to update the solution from one discrete time level to the next: decoupled.
One of the most popular explicit time discretization is the leapfrog. It is a multistep method (three time levels scheme) derived by approximating the time derivative by a centered difference approximation:

$$\frac{\psi_i^{k+1} - \psi_i^{k-1}}{2\Delta t} = \mathscr{L}_h(\psi^k)_i \tag{18}$$

Filters can be applied to remove computational modes from the solution. Efficiency of simple explicit schemes can be improved through the "split-explicit" or "mode splitting" technique: terms responsible for the fastest waves motion are treated separately, using a smaller timestep.

The CFL condition

- The CFL criterion states that a necessary condition for stability is that the domain of dependence of the numerical solution should include the domain of dependence of the original partial differential equation
- CFL condition requires that at each point x_j the fastest that the numerical solution can propagate is one grid point per time step. Time step Δt cannot exceed the limit $\Delta x/u$.
- The CFL condition has a clear geometric interpretation: The numerical domain of dependence must be larger than the physical domain of dependence, and not the other way around. If this wasn't the case, it would be impossible for the numerical solution to converge to the exact solution, since as the grid is refined there will always be relevant physical information that would remain outside the numerical domain of dependence. And, as we have seen, Lax theorem implies that if there is no convergence then the system is unstable.

The 2D Euler equations

The model simulates the 2-D inviscid Euler equations for stratified fluid dynamics, which are defined as follows:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho w\\ \rho\theta\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uw\\ \rho u\theta\end{bmatrix}+\frac{\partial}{\partial z}\begin{bmatrix}\rho w\\ \rho wu\\ \rho w^2+p\\ \rho w\theta\end{bmatrix}=\begin{bmatrix}0\\ 0\\ -\rho g\\ 0\end{bmatrix}$$

$$\rho_H=-\frac{1}{g}\frac{\partial p}{\partial z}$$

The perturbation approximation

Because small violations of this balance lead to significant noise in the vertical momentum, it's best not to try to directly reconstruct this balance but rather to only reconstruct the perturbations. Therefore, hydrostasis is subtracted from the equations to give:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho' \\ \rho u \\ \rho w \\ (\rho\theta)'\end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix}\rho u \\ \rho u^2 + p \\ \rho u w \\ \rho u \theta\end{bmatrix} + \frac{\partial}{\partial z}\begin{bmatrix}\rho w \\ \rho w u \\ \rho w^2 + p' \\ \rho w \theta\end{bmatrix} = \begin{bmatrix}0 \\ 0 \\ -\rho' g \\ 0\end{bmatrix}$$

where a “prime” quantity represents that variable with the hydrostatic background state subtracted off (not a spatial derivative).

The direction splitting

This equation is solved using dimensional splitting for simplicity and speed. The equations are split into x- and z-direction solves that are, respectively:

$$x: \quad \frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} = \mathbf{0}$$

$$z: \quad \frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{h}}{\partial x} = \mathbf{s}$$

Each time step, the order in which the dimensions are solved is reversed, giving second-order accuracy overall.

The Finite volume scheme

A Finite-Volume discretization is used in which the PDE in a given dimension is integrated over a cell domain, $\Omega_i \in \left[x_{i-1/2}, x_{i+1/2}\right]$, where $x_{i\pm1/2} = x_i \pm \Delta x$, x_i is the cell center, and Δx is the width of the cell. The integration is the same in the z-direction. Using the Gauss divergence theorem, this turns the equation into (using the z-direction as an example):

$$\frac{\partial \overline{\mathbf{q}}_{i,k}}{\partial t} = -\frac{\mathbf{h}_{i,k+1/2} - \mathbf{h}_{i,k-1/2}}{\Delta z} + \overline{\mathbf{s}}_{i,k}$$

where $\overline{\mathbf{q}}_{i,k}$ and $\overline{\mathbf{s}}_{i,k}$ are the cell-average of the fluid state and source term over the cell of index `i,k`.

To compute the update one needs the flux vector at the cell interfaces and the cell-averaged source term. To compute the flux vector at interfaces, fourth-order-accurate polynomial interpolation is used using the four cell averages surrounding the cell interface in question.

The time integrator

So far the PDEs have been discretized in space but are still continuous in time. To integrate in time, we use a simple three-stage, linearly third-order-accurate Runge-Kutta integrator. It is solved as follows:

$$\mathbf{q}^{\star} = \mathbf{q}^{n} + \frac{\Delta t}{3} RHS\left(\mathbf{q}^{n}\right)$$

$$\mathbf{q}^{\star\star} = \mathbf{q}^{n} + \frac{\Delta t}{2} RHS\left(\mathbf{q}^{\star}\right)$$

$$\mathbf{q}^{n+1} = \mathbf{q}^{n} + \Delta t RHS\left(\mathbf{q}^{\star\star}\right)$$

When it comes to time step stability, it is assumed a maximum speed of propagation of $450\,\mathrm{m\,s^{-1}}$, which basically means that the maximum wind speed is assumed to be $100\,\mathrm{m\,s^{-1}}$, which is a safe assumption. The CFL value is set to 1.5.

The diffusion stabilizer

The centered fourth-order discretization is unstable for non-linear equations and requires extra dissipation to damp out small-wavelength energy that would otherwise blow up the simulation. This damping is accomplished with a scale-selective fourth-order so-called “hyper”-viscosity that is defined as:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial}{\partial x}\left(-\kappa \frac{\partial^3 \mathbf{q}}{\partial x^3}\right) = \mathbf{0}$$

and this is also solved with the Finite-Volume method just like above. The hyperviscosity constant is defined as:

$$\kappa = -\beta \left(\Delta x\right)^4 2^{-4} \left(\Delta t\right)^{-1}$$

where $\beta \in [0, 1]$ is a user-defined parameter to control the strength of the diffusion, where a higher value gives more diffusion. The parameter β is not sensitive to the grid spacing, and it seems that $\beta = 0.25$ is generally enough to get rid of $2\Delta x$ noise contamination.

The code – 883 code lines + Makefile

```
Makefile
model.f90
module_output.F90
module_parameters.f90
module_physics.F90
module_types.F90
```

Required:

1. Fortran compiler. Hardcoded in the Makefile to be gfortran
2. netcdf-fortran library. Multiple options available on leonardo, can be installed on any Linux distribution (Ubuntu libnetcdff-dev)

To compile executable, GNU make program required. Just type in `make`.

To run the serial model:

```
./model
```

The model output

```
 SIMPLE ATMOSPHERIC MODEL STARTING.
INITIALIZING MODEL STATUS.
nx       :        100
nz       :         50
dx       :   200.00000000000000
dz       :   200.00000000000000
dt       :  0.66666666666666663
final time :    1000.0000000000000
MODEL STATUS INITIALIZED.
TIME PERCENT :  0%
TIME PERCENT : 10%
TIME PERCENT : 20%
TIME PERCENT : 30%
TIME PERCENT : 40%
TIME PERCENT : 50%
TIME PERCENT : 60%
TIME PERCENT : 70%
TIME PERCENT : 80%
TIME PERCENT : 90%
---------------- Atmosphere check ----------------
Fractional Delta Mass  :    2.7944348593189595E-014
Fractional Delta Energy:    1.0051785903535983E-004
------------------------------------------------
SIMPLE ATMOSPHERIC MODEL RUN COMPLETED.
USED CPU TIME:    2.1255869999999999
```

Typical model standard output.

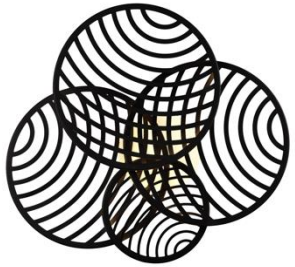
The output created on MacOS with Intel I7 processor.

The netCDF file `output.nc` is also created.

The model.f90

Contains the `PROGRAM` statement.
Use the other modules and contains the call to the initialization and allocation, the output time integration loop, the time advancement, the output control.

The module_parameters.f90



Contains multiple `module` statements:

```
module calculation_types
module legendre_quadrature
module physical_constants
module physical_parameters
module indexing
module iodir
module parallel_parameters
module dimensions
```

Control of the real precision

To put things so that fluxes will be correct

Physical constants

Physical parameters

Index constants

Stdout, stderr

Hints for MPI thingy

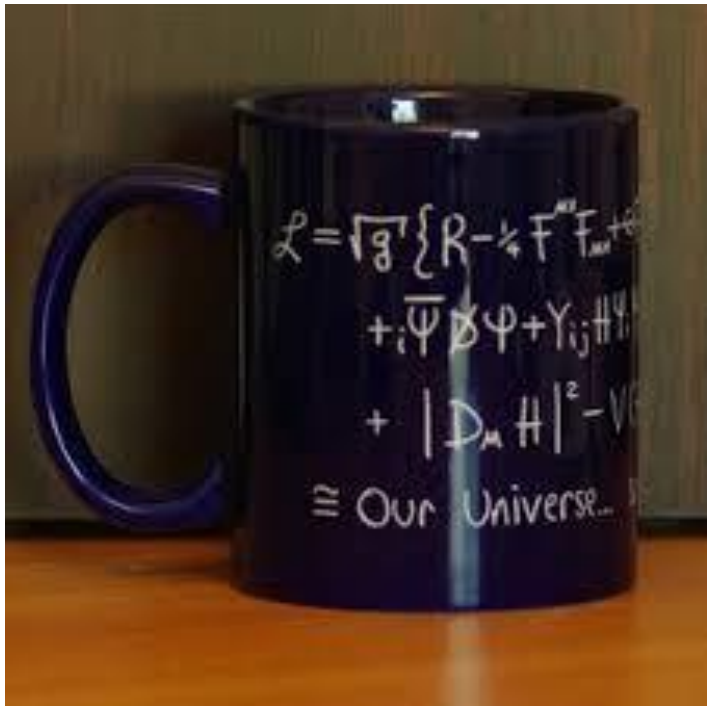
Problem size and run duration

The module_types.F90

Contains some object orientation to satisfy the eye.

- The atmospheric state type (need two for the scheme)
- The flux computed at volume interfaces
- The tendency used to update the state

The module_physics.f90



Contains the physical glue and the initialization. Uses types and their methods.

The module_output.F90

Contains the netCDF library calls part.

https://docs.unidata.ucar.edu/netcdf-fortran/current/f90_The-NetCDF-Fortran-90-Interface-Guide.html

Hint?

In the serial code

```
do l = 1 , 4
  do k = 1 , 50
    do i = 1 , 100
      tend(i,k,l) = &
  -( fl(i+1,k,l) - fl(i,k,l) ) / dx
    end do
  end do
end do
```

In the OMP code

```
!$omp parallel do collapse(3)
do l = 1 , 4
  do k = 1 , 50
    do i = 1 , 100
      tend(i,k,l) = &
  -( fl(i+1,k,l) - fl(i,k,l) ) / dx
    end do
  end do
end do
```

MPI hint?

In the serial code

```
i_beg  = 1
i_end  = nx
```

In the MPI code

```
nxp    = real(nx)/nranks
i_beg = nint( nxp* (my_rank)     )+1
i_end = nint( nxp*((my_rank)+1) )
```

USE async MPI!

Have a look at [MPI_isend, MPI_irecv] or MPI_neighbor_alltoall

Testing

The model prints at the end total fractional mass and energy budgets. **They must not change much.** Mass should stay around 10^-15, energy 10^-5. Any big change means something wrong.

```
----------------- Atmosphere check -----------------
 Fractional Delta Mass  :    2.7944348593189595E-014
 Fractional Delta Energy:    1.0051785903535983E-004
 ------------------------------------------------------
```

Testing

A python script is provided to check three netCDF output files. Suggested is to have:	File 1 : Serial code with NO optimization
	File 2 : Serial code with basic compiler optimization (i.e. Ofast, fast_math, etc)
	File 3 : Your test netCDF output

Check that the script output shows SMALL numbers.

?

Benchmarking

The default problem size (100 points) will not generate big computation load. Use it to develop (fast runtime).

Do increase the problem size for benchmarking!

1. At what problem size you have advantages?
2. At what size the MPI version is worth using?
3. At what size offloading on GPU is worth using?

What is the role of the Project Manager?

Project manager role is to approve merge of pull requests

1. Review the provenance (key verification)
2. Review the test results

She has a diplomatic role

1. Interact with Developers
2. Interact with Scientific Advisors

She has a visibility role with users

1. Tag the final release
2. Update the milestone deadlines

What is the role of the SSD?

A Scientific Software Developer must

1. Ask meetings with the Scientific Advisors to clarify any doubts regarding the computational results
2. Interact respectfully with all the other developers who may have different agendas or skills
3. Interact respectfully with the PM but with the confidence of her own skills and results and strive to reach milestone deadlines
4. Be ready to restart her work oh so many times from scratch
5. Train to become a PM

Resources

1. Chacon Straub – Pro Git (in pdf in the repo)
2. Metcalf - Modern Fortran Explained - 6th edition [https://doi.org/10.1093/oso/9780198876571.001.0001]
3. Markus - Modern Fortran in Practice [https://doi.org/10.1017/CBO9781139084796]
4. Fortran Language best practice [https://fortran-lang.org/learn/best_practices]
5. Fortran performance best practice (in pdf in the repo)
6. Open Catalog of Best Practice https://github.com/codee-com/open-catalog
7. The netCDF Fortran https://docs.unidata.ucar.edu/netcdf-fortran/current