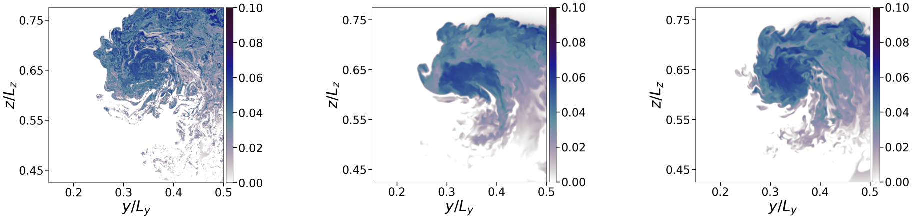
Webinar: Parallel Moist Parcel-In-Cell code

Steef Böing, Gordon Gibb, David Dritschel, Nick Brown,
Michèle Weiland, Doug Parker & Alan Blyth

University of Leeds, University of St Andrews, EPCC

November 13, 2019

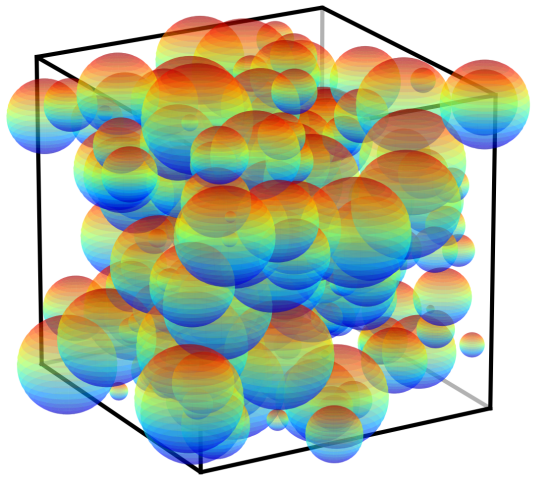


eCSE project 12-10

A fully Lagrangian dynamical core for the Met Office NERC Cloud Model

St Andrews, Leeds, EPCC

- Most fluid dynamics codes are either fully Eulerian (grid-based), or semi-Lagrangian (advection using *departure points* and regridding)
- Here: *essentially Lagrangian* (prognostics on parcels, solver uses grid)
- Atmospheric Large-Eddy Simulation: e.g. Met Office/NERC Cloud model (MONC).
- Evaporation and condensation in clouds: discontinuity in underlying equations.



Essentially Lagrangian modelling

The basic conservation principles of fluid dynamics are naturally expressed in a Lagrangian way: e.g. mass is conserved *following* fluid "parcels".

However, certain fields are more naturally Eulerian in character, e.g. pressure. Here, one needs to solve for the entire field through "inversion".

Conservation is Lagrangian. Inversion is Eulerian.

Can we exploit this for simulation?

Moist Parcel-In-Cell (MPIC)

The new "Moist Parcel-In-Cell" (MPIC) algorithm represents the continuum by discrete "cloud (or environment) parcels".

We use freely-moving parcels carrying *any number* of attributes (e.g. a conserved temperature b_ℓ, specific humidity q, etc...)

Prototype model for 3D incompressible flow (Boussinesq, no rotation, no precipitation, non-dimensional):

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = -\frac{\boldsymbol{\nabla} p}{\rho_0} + b\hat{\mathbf{z}} \qquad \text{momentum}$$

$$\frac{\mathrm{D}b_\ell}{\mathrm{D}t} = 0 \qquad \text{conserved temperature (pressure/phase changes)}$$

$$\frac{\mathrm{D}q}{\mathrm{D}t} = 0 \qquad \text{specific humidity, total amount of water}$$

$$\boldsymbol{\nabla} \cdot \mathbf{u} = 0 \qquad \text{incompressibility}$$

Phase transitions

The total buoyancy b is approximated by

$$b = b_\ell + \alpha q_\ell$$

$$q_\ell = q - q_s(z) \text{ if } q > q_s(z), \text{ otherwise } 0.$$

q_ℓ is the liquid water content.
q_s is the saturation humidity, which decreases with height.
α is a scale factor related to the latent heat of condensation.

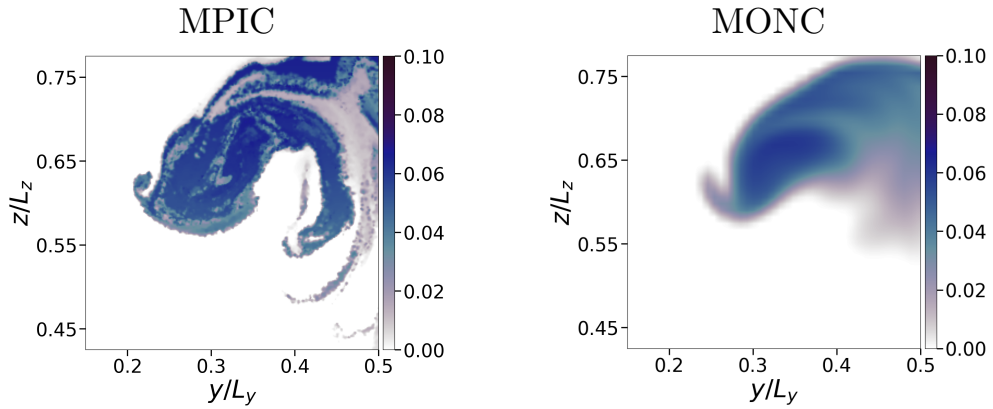


Figure : Zoom of the condensed liquid water distribution in a rising thermal. MPIC uses an underlying grid with the same resolution as MONC.

Numerical implementation

We evolve vorticity (instead of momentum) and parcel position using RK4

$$\frac{\mathrm{D}\boldsymbol{\omega}}{\mathrm{D}t} = (\nabla \cdot \mathbf{F}, \nabla \cdot \mathbf{G}, \nabla \cdot \mathbf{H}),$$

Where $\mathbf{F} = \omega u + b\hat{\mathbf{y}}$, $\mathbf{G} = \omega v - b\hat{\mathbf{x}}$, $\mathbf{H} = \omega w$.

Needs velocity field (from grid).

Vector Poisson solver (finite difference, grid-based) to find velocity potential $\mathbf{A}$ and velocity $\mathbf{u} = -\nabla \times \mathbf{A}$.

$$\boldsymbol{\omega} = \nabla^2 \mathbf{A}.$$

Some further subtleties to ensure the vorticity is divergence free.

Parcel splitting and mixing

- Parcels stretch and can split into 2 smaller parcels, depending on vorticity.
- Splitting: creates new parcel. Old and new parcel change position.
- When parcel becomes too small: merged into surrounding parcels using conservative operation via grid.

Parallelism

Initially, MPIC was developed using shared-memory parallelism (OpenMP), which limits problem sizes to be addressed.

To implement hybrid (MPI+OpenMP) parallelism, we need to consider:

- Change of parcel position: advection, splitting (local)
- Parcel merging (local)
- Vector Poisson solver (global)

Parallelism

- Vector Poisson solver: requires global communication. Efficient algorithms exist.
- First implementation OpenMP. Inherent limitation of problem size. HPC trend to large distributed memory systems.
- Much more parcel data than grid data.
- Parcel data: local communication.

MONC history

Very high resolution ($\sim$ 2 to 50 m), flexible, portable cloud modelling framework, developed through collaboration between NCAS, the Met Office, EPCC and several UK universities.

- Based on Met Office Large Eddy Model (limited to $\sim$ 512 cores)
- Fortran 2003
- Modular structure
- Funded though NERC/JWCRP/eCSE
- Leap-frog time integration

MONC parallelism

MONC supports decomposition in both the x and y dimensions with at least one column per process

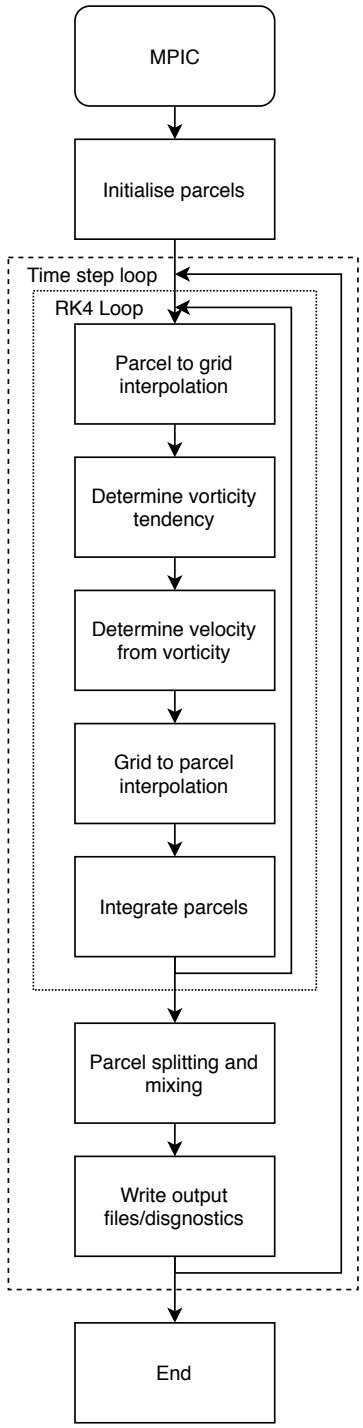
- Number of columns can be distributed unevenly
- Improved decomposition means more parallelism to be exploited
- Asynchronous MPI
- Includes FFT solver, based on FFTW (other solvers not tested)
- IO-server (not used here)
- Scales well on up to 32,768 cores

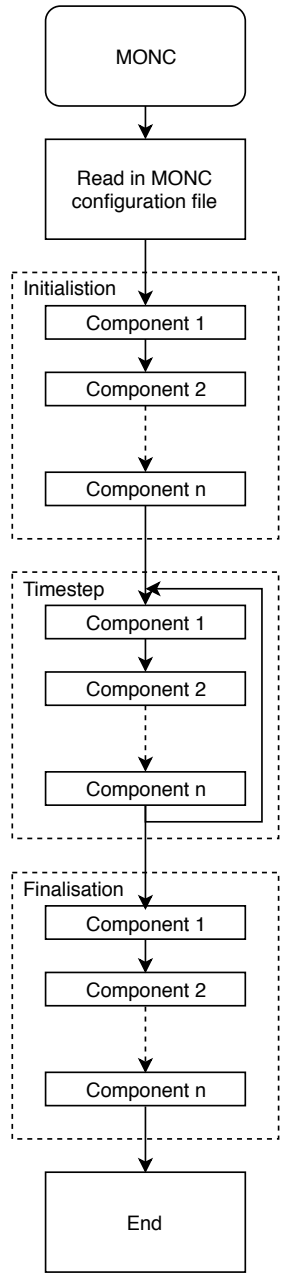
Model core and components

The model core contains the MONC entry point, registry functionality and some utility modules

Plugins called components

- Independent of each other
- Standardized format
- Enabled/disabled at runtime through configuration files
- Easy to create new components for testing
- Managed via a registry
- Each can be called at different times, e.g.
  1. Initialisation
  2. Each time step
  3. Finalisation


MPIC
Initialise parcels
Time step loop
RK4 Loop
Parcel to grid interpolation
Determine vorticity tendency
Determine velocity from vorticity
Grid to parcel interpolation
Integrate parcels
Parcel splitting and mixing
Write output files/disgnostics
End


MONC
Read in MONC configuration file
Initialistion
Component 1
Component 2
Component n
Timestep
Component 1
Component 2
Component n
Finalisation
Component 1
Component 2
Component n
End

eCSE project

St Andrews, Leeds, EPCC (Michèle Weiland, Nick Brown, Gordon Gibb)

Ideas:

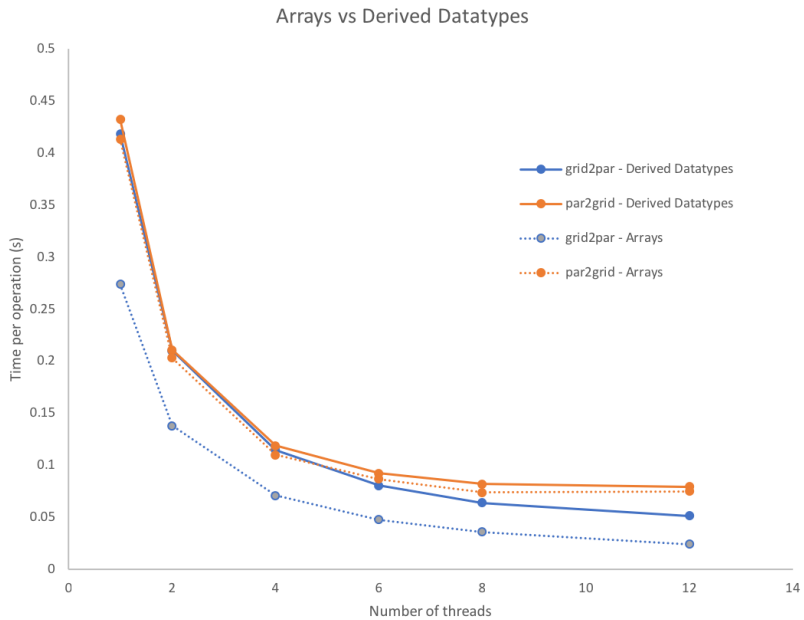
- Harness MONC's parallelism: hybrid OpenMP+MPI.
- Poisson solver available.
- Approach: domain decomposition, number of parcels per subdomain will vary (simplicity versus optimal load balancing).
- Lagrangian diagnostics can feed back into standard MONC.
- Component testing using simplified code.

PMPIC

- Implementation of MPIC in MONC's framework.
- Based on stripped version of MONC core.
- Not compatible with other MONC components (parcels in model state, RK4 time step, different equations, non-staggered grid).
- But uses (FFTs, grids) and extends (parcel parallelism) MONC infrastructure.
- GIT repository + makefile.

Design choices

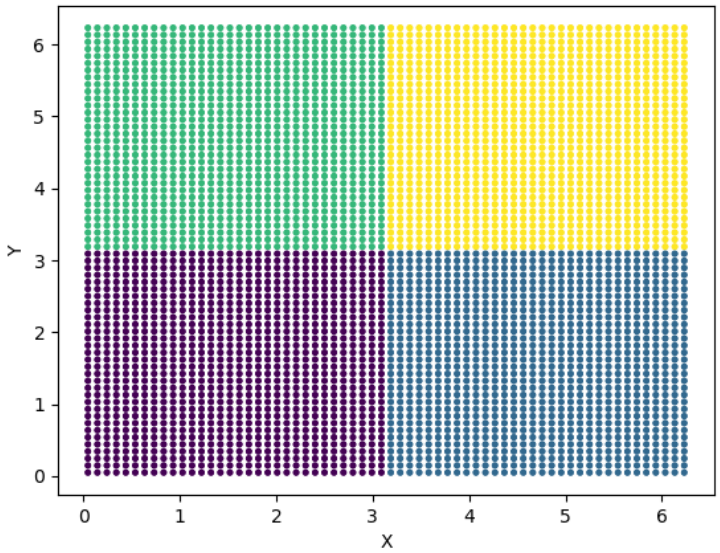
- Data held in arrays, rather than parcel-types (more difficult halo-swap, but efficient).

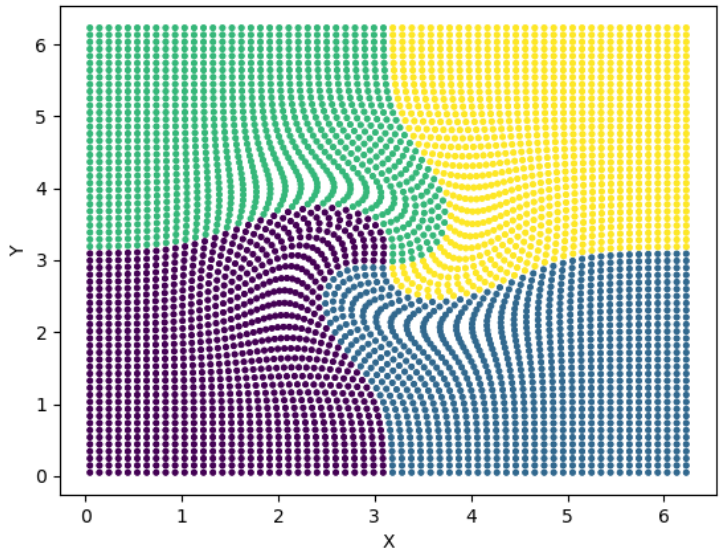


- New group type: RK4 (not all operations substepped).
  Higher memory footprint: work in progress on low-storage variant.
- Binary dumps (parcels/grids) and NetCDF (optional, grids only so far), instead of IO-server. Memory requirements of main code.

Halo-swapping

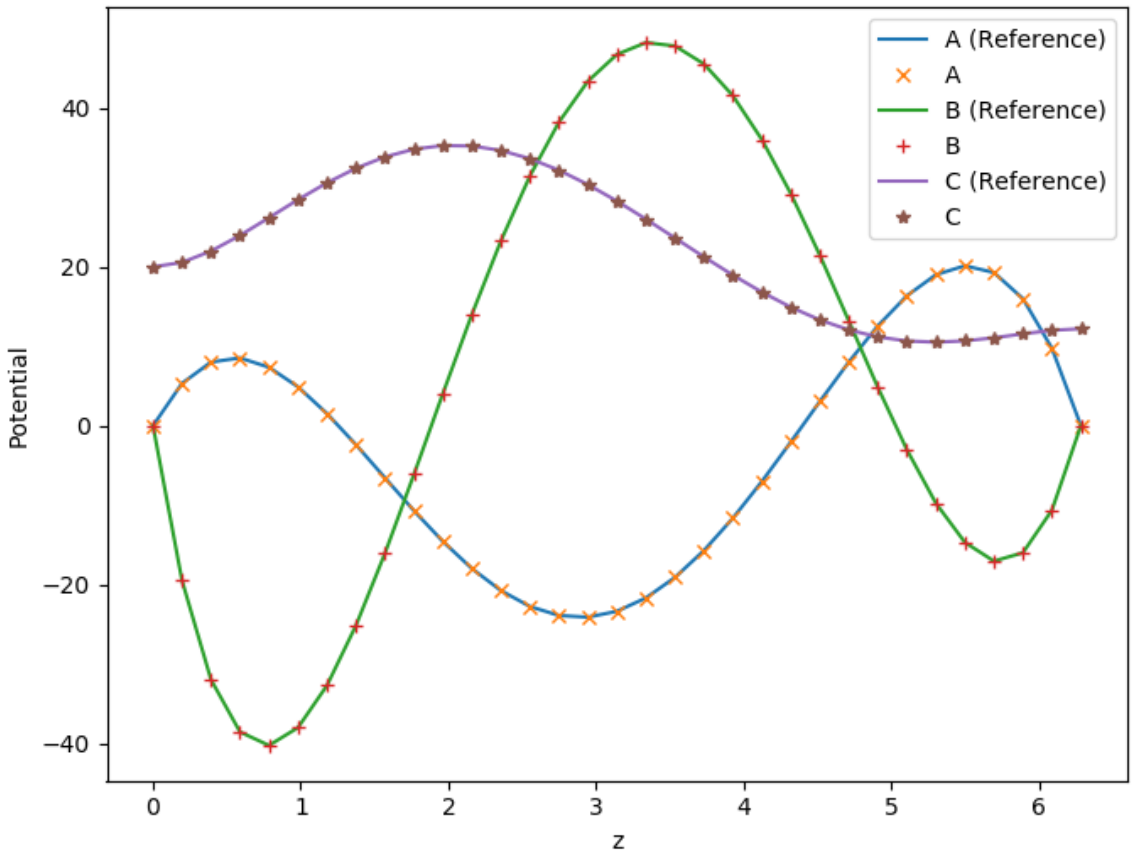
- Parcel halo-swap needed testing.
  In particular: backfill (parcel deletion/creation).
- Modified grid halo-swap in solver. Decision to write new simple halo-swapper for grids.
- Halo-swapping also comes into parcel mixing. Systematic parcel creation/removal tests.





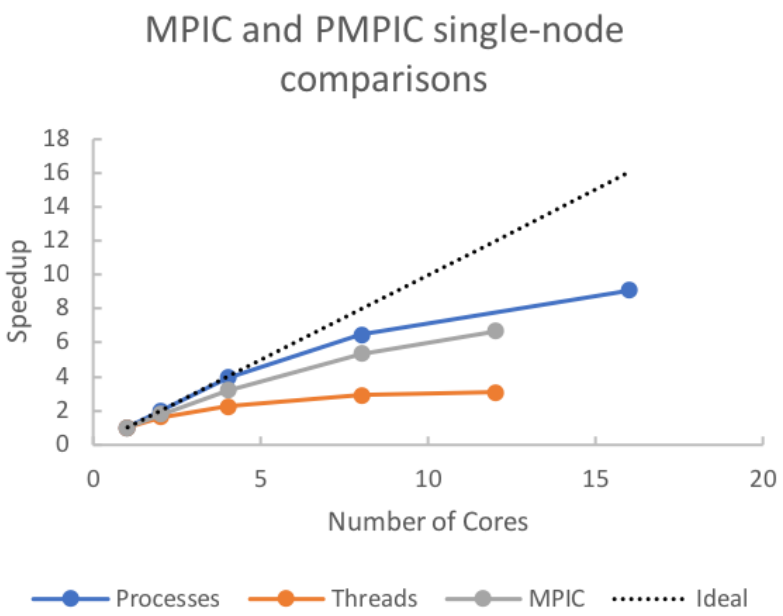
Numerics

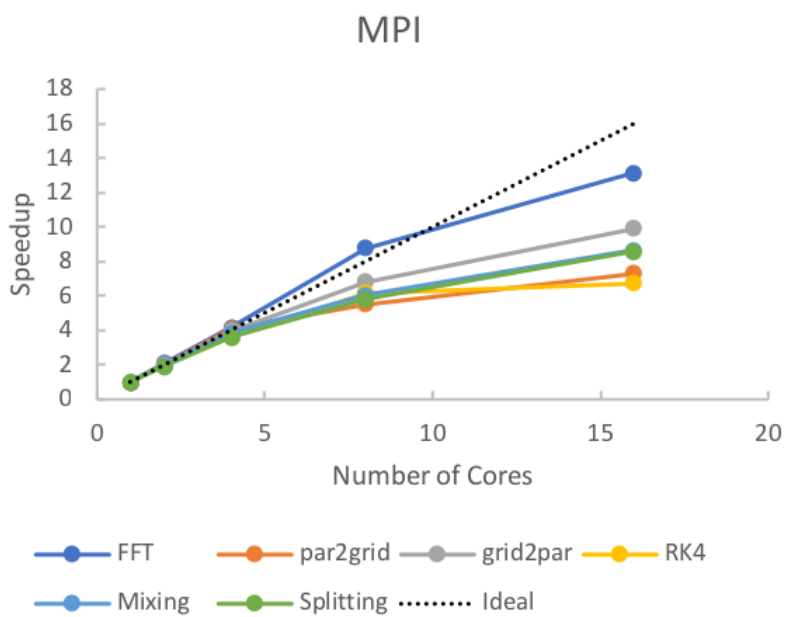
- 4th order compact central differencing in tridiagonal solver (David Dritschel).



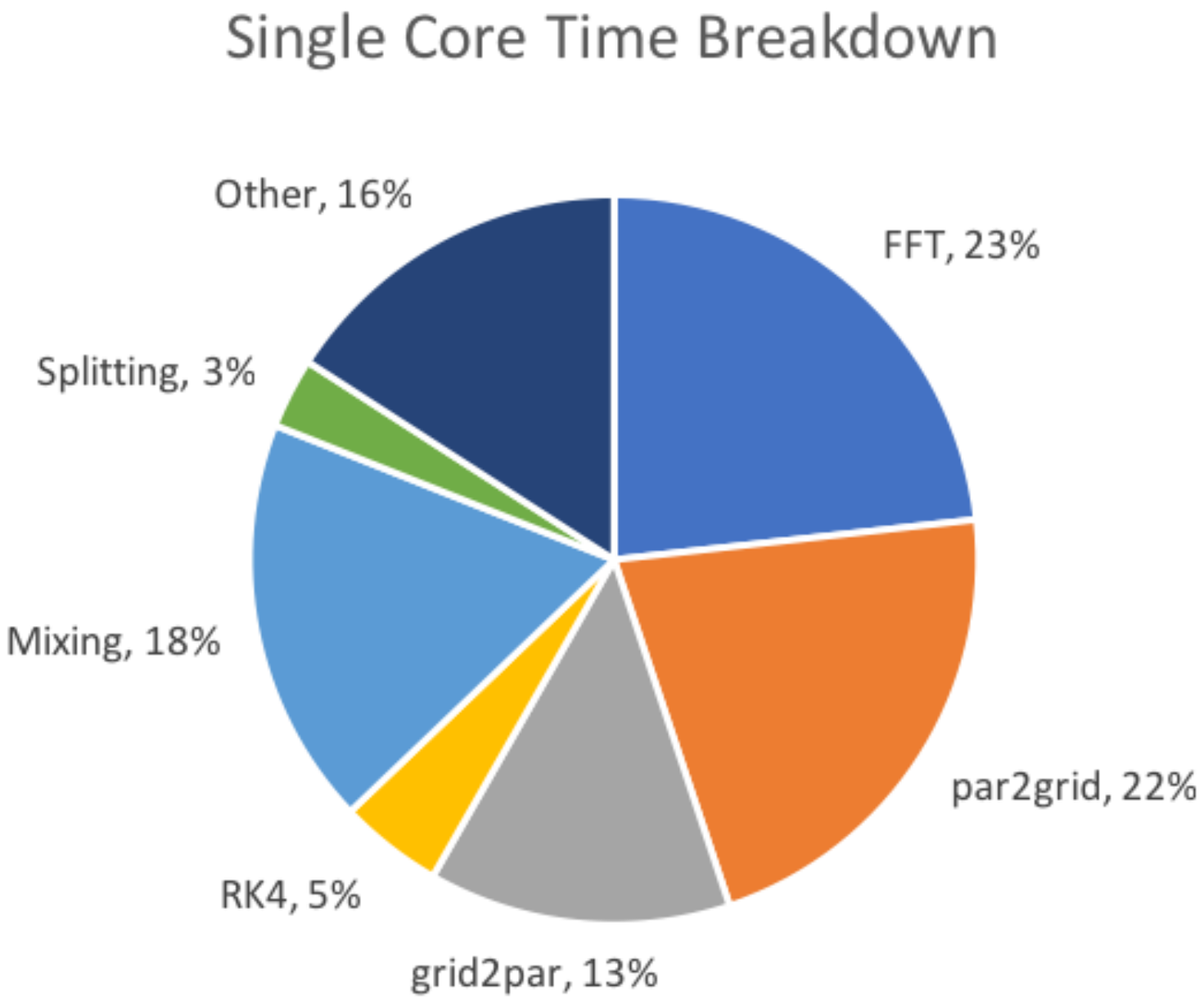
Performance: single node

- Overall performance of new code on single core: 1.6 times faster.
- MPI scaling hindred by load imbalance.
- OpenMP not scaling well (tune chunk size/try static arrays?).

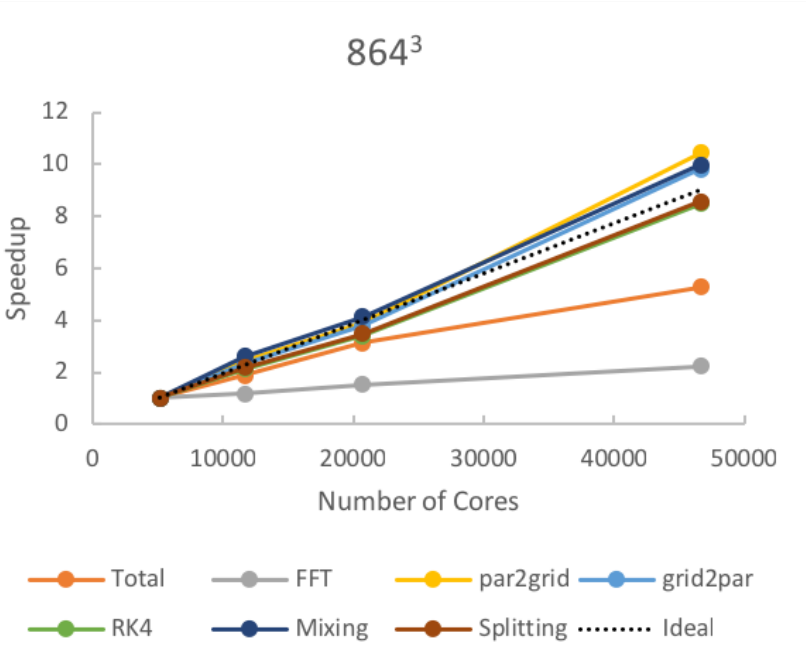


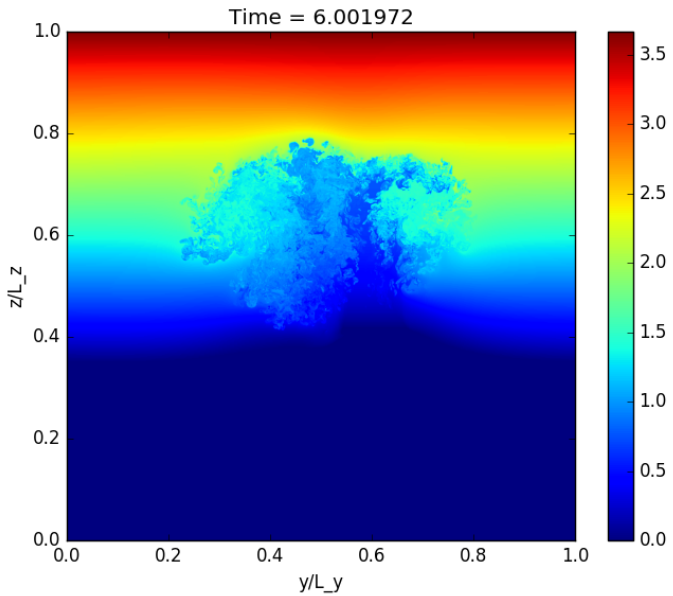


Performance



Performance: large simulations





Benefits

- First use of this type of model in atmospheric community.
- Massively parallel MPIC will make it more attractive for other problems, e.g. ocean mixed layer, density-laden flows.
- Alternative approach for MONC community.
- Could provide basis for Lagrangian diagnostics, currently lacking in MONC.
- BSD license.

PMPIC code repository

Example usage:

```
mpiexec -n 2 monc --config=config.mcf
```

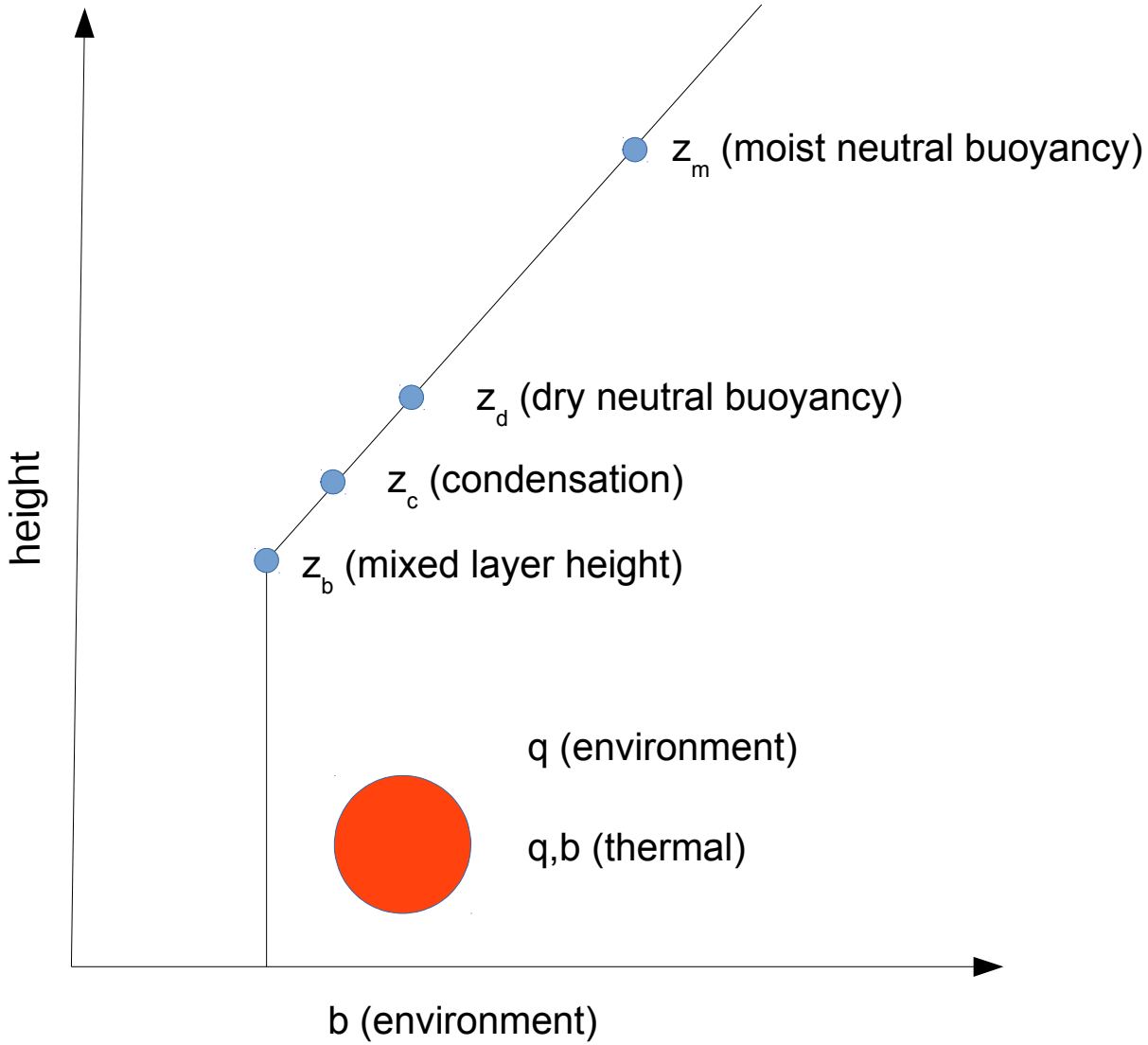
Dependencies:

- MPI
- FFTW
- NetCDF (optional)

Compilers tested:

- GNU (on laptop and ARCHER)
- Cray (on ARCHER)

Default test case is a spherical moist and warm thermal in a neutrally-stable boundary layer overlaid with a stably-stratified atmosphere.



Running PMPIC

Running pmpic should be as simple as executing:

```
[mpiexec/aprun ...] /path/to/monc --config=[config file]
```

The config file controls which components are run (and in what order) and also runtime parameters. There is also a global_config file which contains basic settings required for MONC to operate. This shouldn't be edited unless you know what you're doing.

The config file and global_config should be placed in the working directory.

The default case is small and should be able to run on a laptop in around a minute or less.

Further information

When the code is slow, please first try running with

```
export OMP_NUM_THREADS=1
```

as by default OpenMP uses all cores on your system, so you may end out running many more threads+processes than you have cores.

A list of components is provided on the PMPIC wiki

Subgrid visualisation

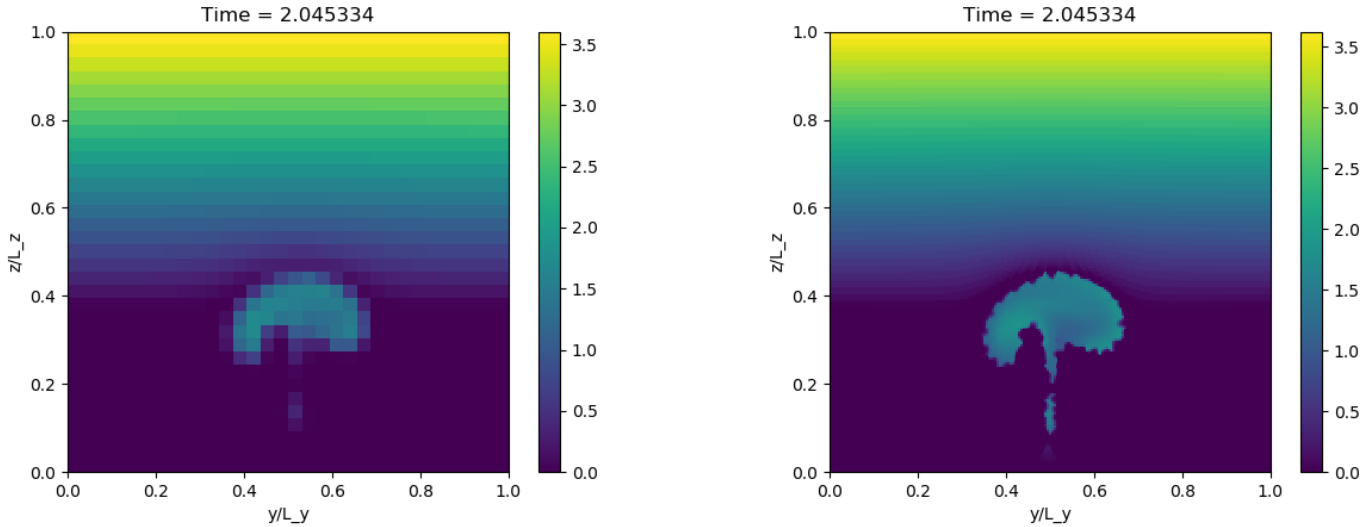


Figure : Comparison of the buoyancy in the $y - z$ plane for a 32^3 grid cell simulation constructed from the gridded values (left) and from the parcels (right). It can be seen that images constructed from the parcels have considerably more detail as they are able to resolve sub-gridcell structure.

Included scripts

display_grid.py: displays a field (buoyancy by default)
display_parcels.py: renders a field based on a Gaussian kernel
planner.py: calculates the approximate memory footprint of PMPIC (total and per process)
timing.py: reads previously created timing data (need to specify this beforehand in MONC configuration file)
visualise.py: older visualisation routine

Writing your own initial condition component

PMPIC comes with two initial condition components

- `basic_parcelsetup` component (which places parcels uniformly in space but does not assign any values to them)
- `plume_parcelsetup` component which sets up the initial condition used here.

To write your own initial condition component, it is easiest to create a copy of the `basic_parcelsetup` directory in `components/` and call it something else (let's say `my_parcelsetup`). Change into this directory and alter the name of `src/basic_parcelsetup.F90` to `src/my_parcelsetup.F90`. Also remember to modify the makefile in this directory to point to the newly renamed file.

Now to edit `my_parcelsetup.F90`. First of all, change the module name to `my_parcelsetup_mod`. Change the `basic_parcelsetup_get_descriptor` function to (next slide)

```
type(component_descriptor_type) function my_parcelsetup_get_descriptor()
    my_parcelsetup_get_descriptor%name="my_parcelsetup"
    my_parcelsetup_get_descriptor%version=0.1
    my_parcelsetup_get_descriptor%initialisation=>initialisation_callback

end function my_parcelsetup_get_descriptor
```

Now you can go and edit the subroutine `initialisation_callback` to put in the initial conditions you want.

To enable this component, alter your config file to have the line `my_parcelsetup_enabled=.true.` (ensuring that the other parcelsetup components are disabled).

After recompiling you should now be able to use your new component!

NetCDF libraries

We have made output to NetCDF available in PMPIC. This requires a version of NetCDF with parallel NetCDF support

These libraries are available under ARCHER as the cray-hdf5-parallel and cray-netcdf-hdf5parallel modules. Note that if you change compiler environment (compilation has been tested with the GNU compiler on ARCHER) you may need to unload and then reload the modules.

See PMPIC wiki for details on local compilation.

Conclusions

- eCSE outcome: MPIC model that scales on thousands of cores.
- Improving scalability: analyse issues with OpenMP, replace FFT-derivatives by compact finite differencing where possible (currently 18 FFTs per time step).

Future plans for MPIC

- Developing a full cloud model: realistic thermodynamics and precipitation, radiation.
- More flexible boundary conditions, in particular for momentum.
- Work on marginally resolved and subgrid-scale dynamics to improve mixing representation (convergence).
- Exploitation of vorticity diagnostics and Lagrangian analysis.

Image: NASA Johnson Space Center (public domain).