A highly scalable Met Office NERC Cloud model

EASC 2015

Nick Brown (EPCC), Michele Weiland (EPCC), Adrian Hill (Met Office), Ben Shipway (Met Office) and Chris Maynard (Met Office)

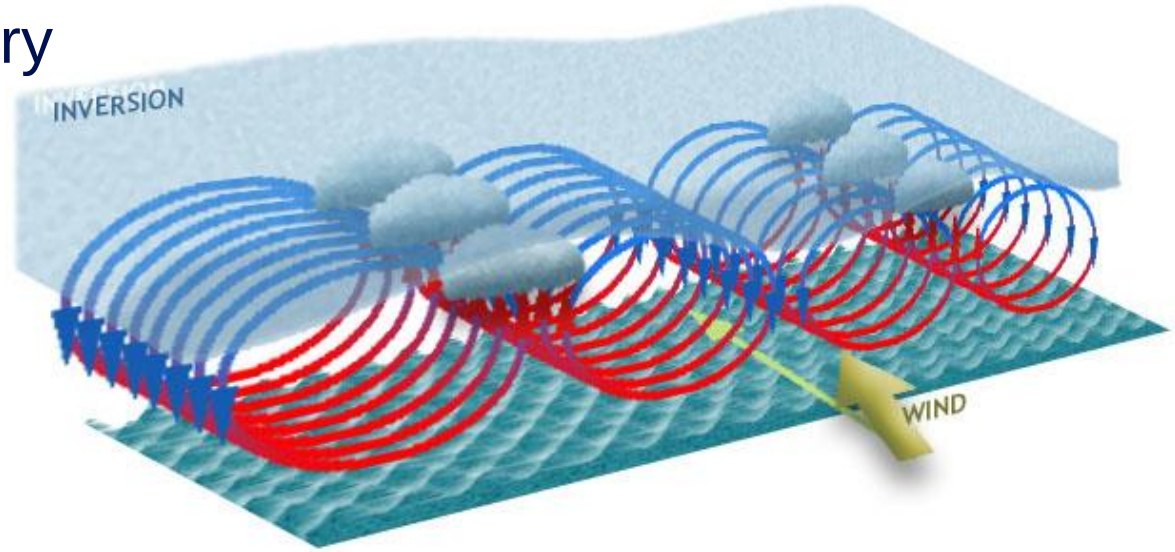
nick.brown@ed.ac.uk

- The existing Large Eddy Model (LEM)
- The replacement Met Office NERC Cloud model (MONC)
- Performance and scalability

Background

- The Met Office's Large Eddy Model (LEM) is used for large eddy simulation and cloud resolving modelling
  - Primarily models clouds and atmospheric flows
  - The results of these simulations inform science in their own right and help develop the parameterisations for the UM.

- The desire is to do very high resolution (<1m) and/or real time modelling



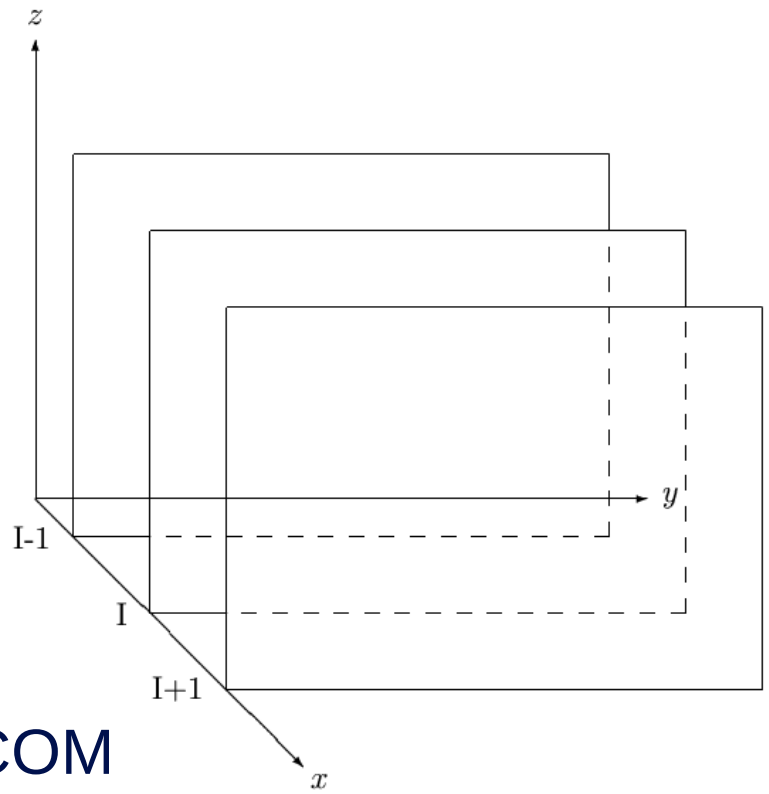
Background

- However the LEM was developed in the late 1980s
  - Designed for scalar machines
  - A mixture of FORTRAN 90, 77 and earlier

- Parallelised in the mid 1990s and initially targeted the T3E (430 GFLOPS.)
  - Some perfective maintenance performed since then to enable use on later generation machines, but still using the same basic assumptions.

Background – scalability issues

- The 3D space is decomposed into 2D slices
  - One of the largest runs has been x=y=384 z=150 (22 million grid points) over 192 processes.



- Parallel calls go to MPI through GCOM
  - Generations of users have miss understood the semantics of these communications (such as blocking) and added in lots of superfluous synchronisation.

Background – code issues

- Uses an archaic system for managing the code
- Global variables
- Gotos
- Equivalences
- Different styles adopted in the same files/procedures
- No unit tests.
- Nobody knows the workings of some areas of the code

MONC

- We elected for a complete rewrite of the code, using modern software engineering and parallelism techniques
  - Written in Fortran 2003 with MPI
  - Using Fruit for unit testing and Doxygen for documentation
  - Designed to be a community model which will be accessible to be changed by non expert HPC programmers and scale/perform well.

- Met Office to get a Cray XC40 machine.
  - This, along with ARCHER is the initial target for the model.

MONC – code architecture

- Architected as plugins called components
  - All independent of each other
  - Follow a specific standard format
  - Can be enabled/disabled at runtime via configuration files
  - Trivial to create new components
  - Managed via a registry
- Components contain optional callbacks
  - At initialisation of MONC
  - Per timestep
  - At finalisation of the model

MONC – Component example

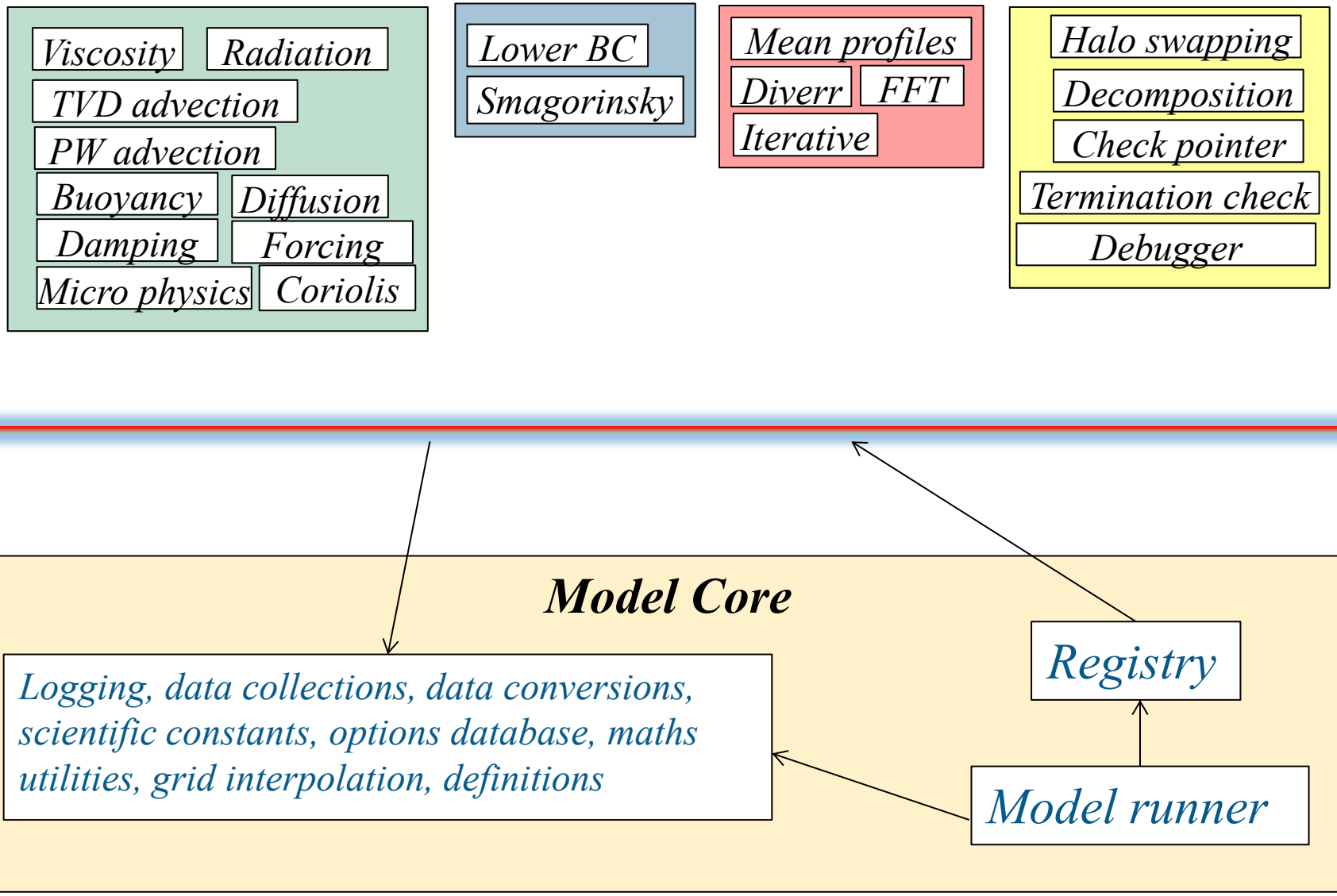
```
type(component_descriptor_type) function test_get_descriptor()
    test_get_descriptor%name="test_component"
    test_get_descriptor%version=0.1
    test_get_descriptor%initialisation=>initialisation_callback
    test_get_descriptor%timestep=>timestep_callback
end function test_get_descriptor
```

```
subroutine initialisation_callback(current_state)
    type(model_state_type), target, intent(inout) :: current_state
    ……
end subroutine initialisation_callback
```

```
subroutine timestep_callback(current_state)
    type(model_state_type), target, intent(inout) :: current_state
    ……
end subroutine timestep_callback
```

```
test_component_enabled=.true.
```

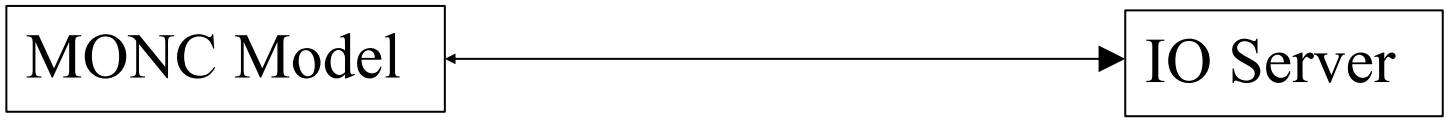
MONC - Components



MONC – IO Server

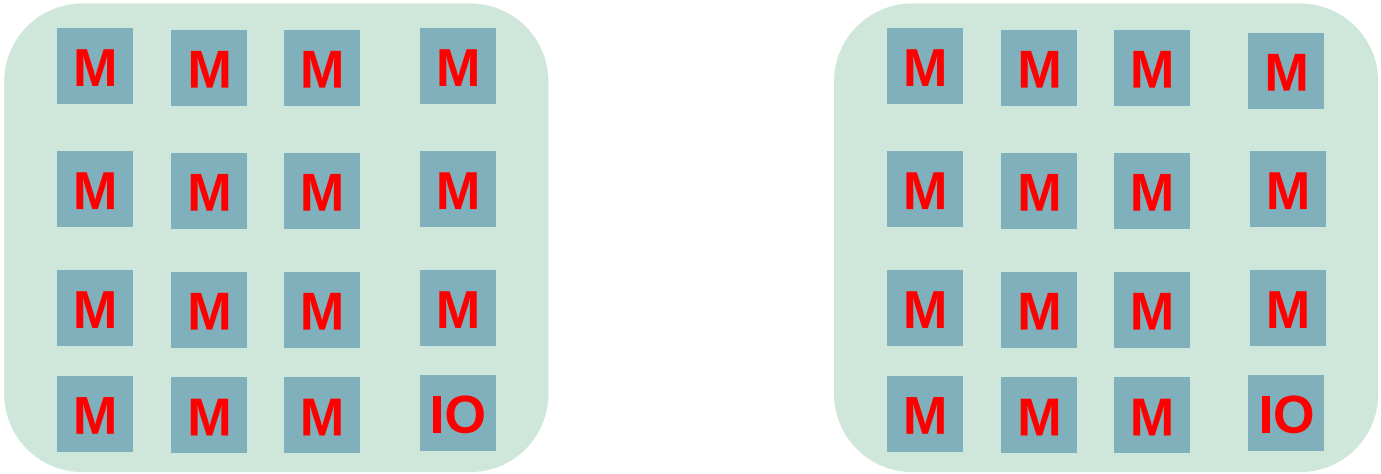
- In addition to the model functionality (working on prognostics), data analysis needs to be done to produce diagnostic data
  - Such as the average temperature at each vertical level
  - In the LEM this is done for each timestep from within the model

- In MONC a separate IO server is used
  - The MONC model can fire and forget required data at any point to the IO server
  - This means that the model can continue to run and not be impacted by IO related latencies.



MONC – IO Server

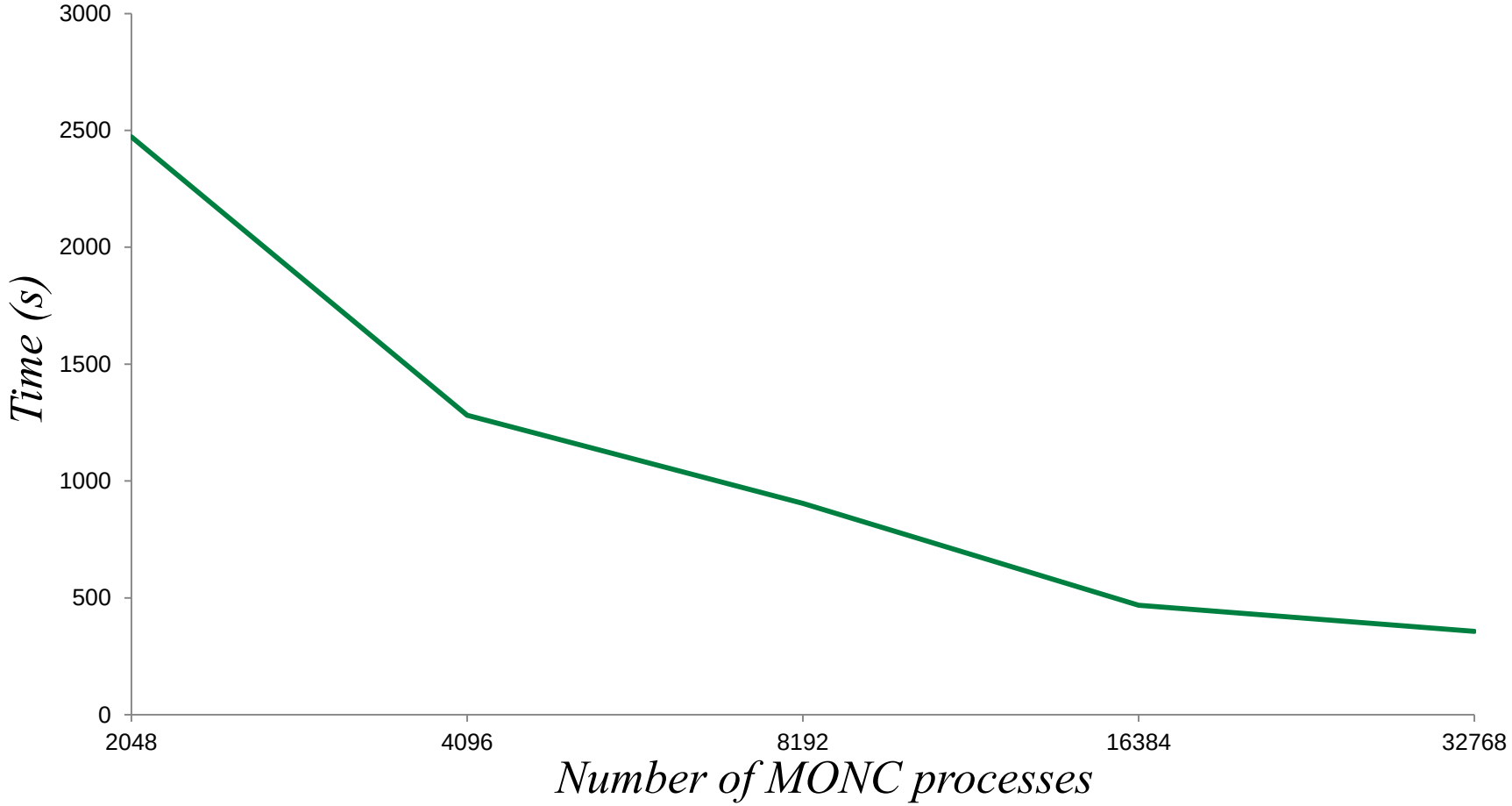
- Have many MONC processes and a number of IO servers
  - Typically one core per processor is dedicated to IO, serving the other cores running the model
  - Our own IO server implementation provides a framework where diagnostics can be configured via XML and/or code.



- Can use any IO server, including XIOS
  - It is just a component in the model which connects to them

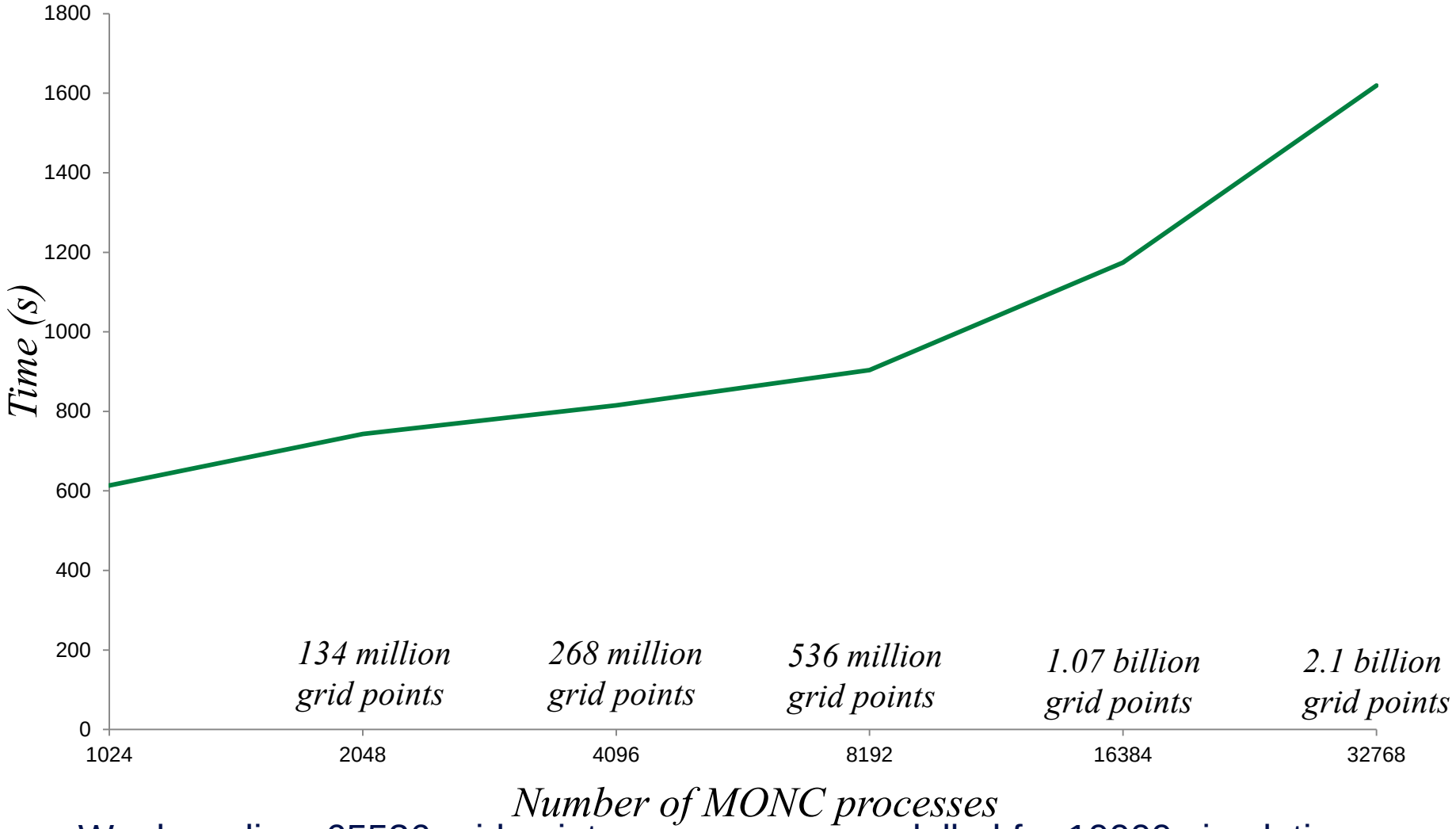
Performance & scalability - strong

- Using the dry boundary layer test case which is wind at a specific level in the vertical



- Strong scaling, 536 million grid points, modelled for 10000 simulation seconds

Performance & scalability - weak



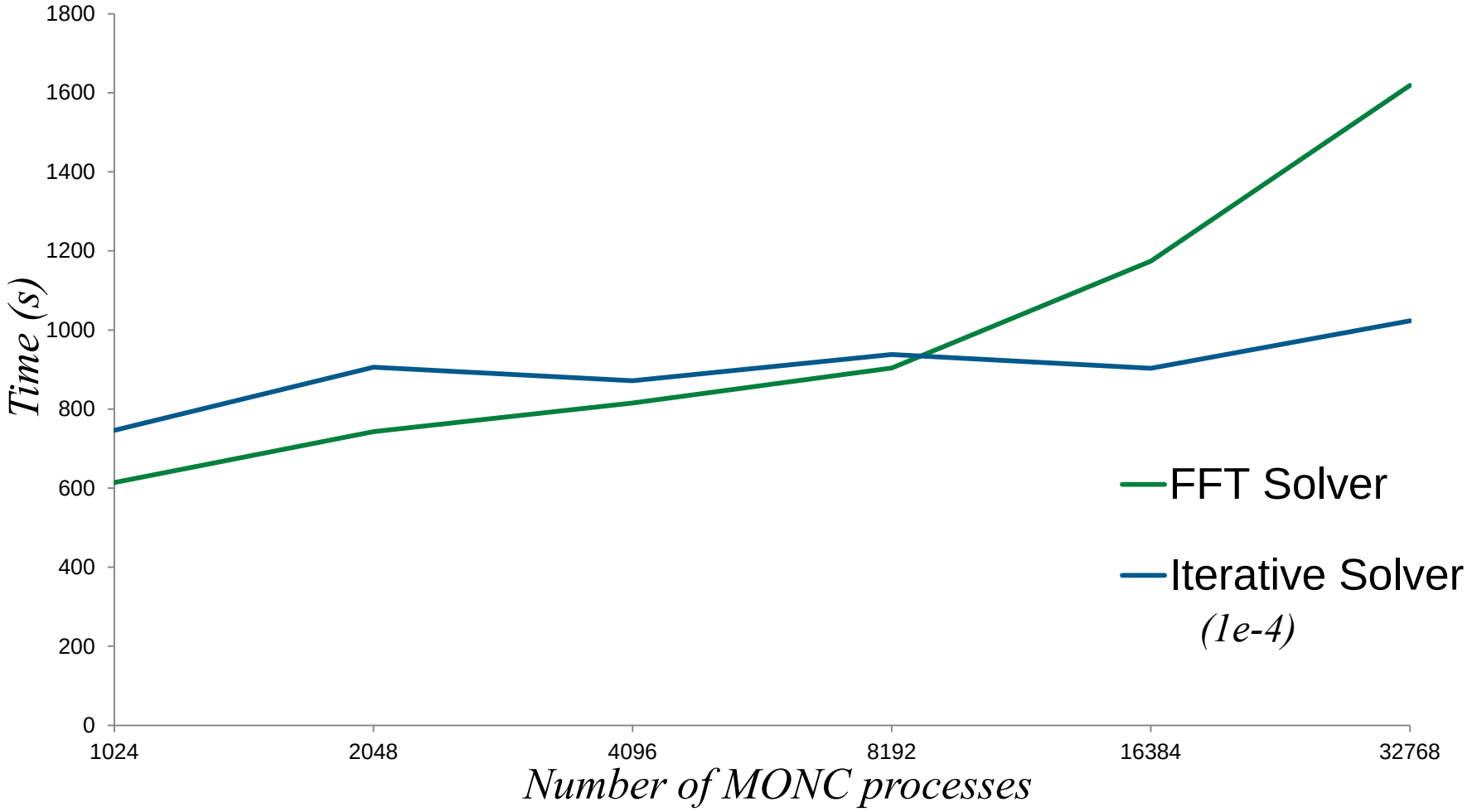
- Weak scaling, 65536 grid points per process, modelled for 10000 simulation seconds

Improving scalability - Iterative solver

- The Poisson equation is solved for pressure terms
  - The LEM uses an FFT method with a tridiagonal solver. Working in Fourier space this solve an ordinary vertical differential equation but requires forwards and backwards global FFTs.
  - A similar version has been implemented in MONC, decomposing in pencil and using FFTW for the actual FFT kernel.
  - Regardless, an FFT based approach requires lots of all to all communications and won't scale.

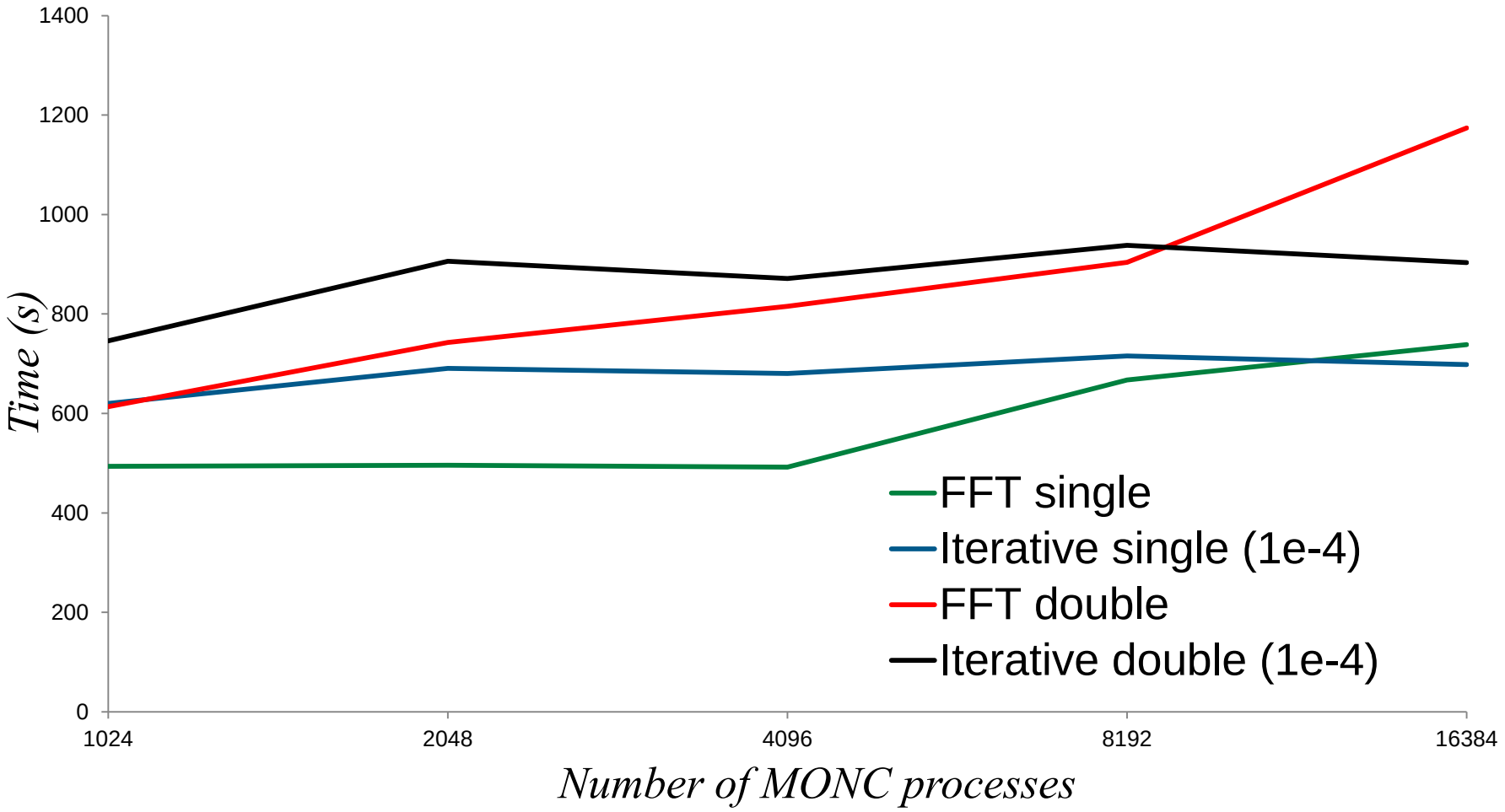
- An iterative solver (component) has been implemented which replaces the FFT solver (component) and should scale better
  - A matrix less implementation of ILU preconditioned BiCGStab
  - CG also provided as an option

Iterative vs FFT solver



- Weak scaling, 65536 grid points per process, modelled for 10000 simulation seconds

Precision - single vs double



- Weak scaling, 65536 grid points per process, modelled for 10000 simulation seconds

Conclusions and further work

- MONC is a highly scalable and configurable community model
- Demonstrated model runs and core counts well beyond what the current model can handle

- GPU version of the advection schemes (to be tested on Piz Daint.)
- The scientific community are starting to use current versions of MONC
- Scalability aspects to be further tuned