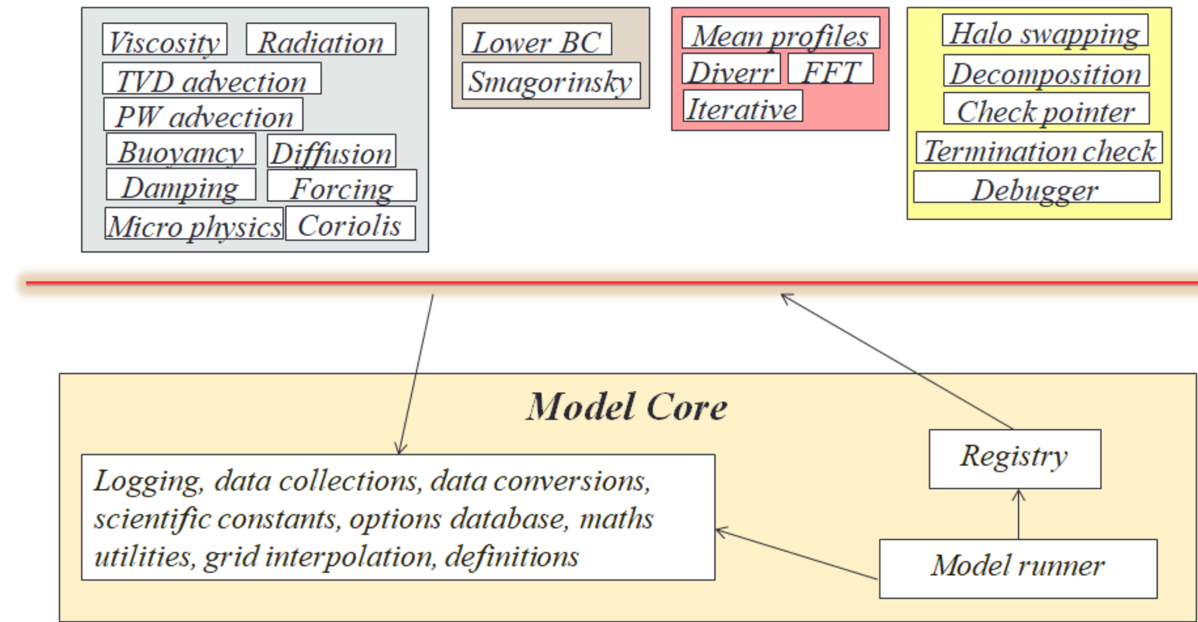
# MONC: GAINING GREATER INSIGHTS INTO CLOUDS AND TURBULENT PROCESSES VIA GPUS


**Nick Brown, EPCC at University of Edinburgh**
Christopher Day, EPCC at University of Edinburgh
Steven Boeing & Mark Richardson, University of Leeds
Todd Jones, University of Reading

# Met Office NERC Cloud model (MONC)

- A redevelopment of a model from the early 1990s called the LEM
- Been operational for around 11 years
- Used to model atmosphere at high resolution
  - Turbulence parameterisation, boundary layer flows, convection, fog, driver for sub models, developing parameterisation schemes, response of ice clouds to aircraft emissions



- Architected as independent components which plug in and out, with a central core that provides marshalling and control, as well we utility functionality
  - A specific run is based around a set of components being enabled
  - Community has grown up around this

epcc

# Redesign delivered an important capability (on original ARCHER)

Time (seconds)

1200
1000
800
600
400
200
0

67 million grid points
134 million grid points
268 million grid points
536 million grid points
1.07 billion grid points
2.1 billion grid points

1024 2048 4096 8192 16384 32768

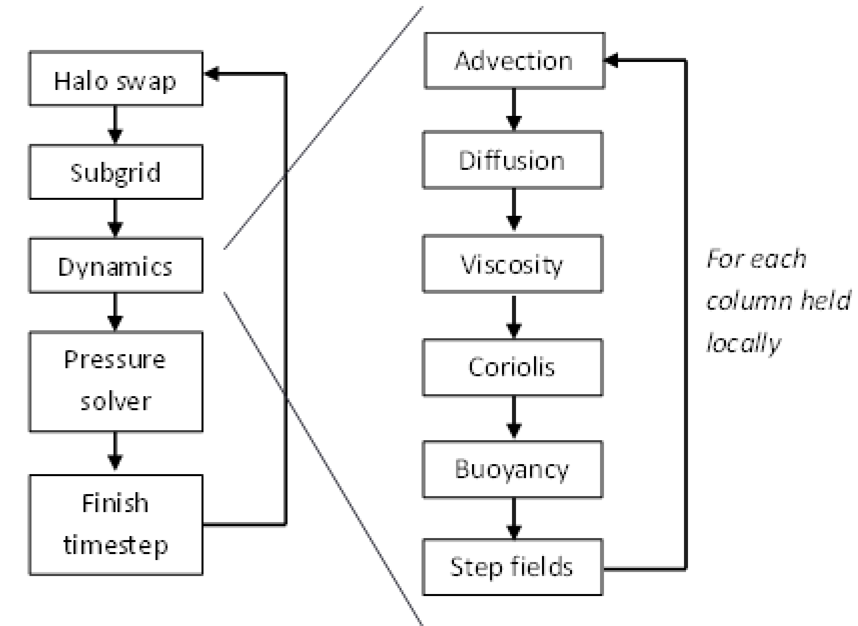
Processes

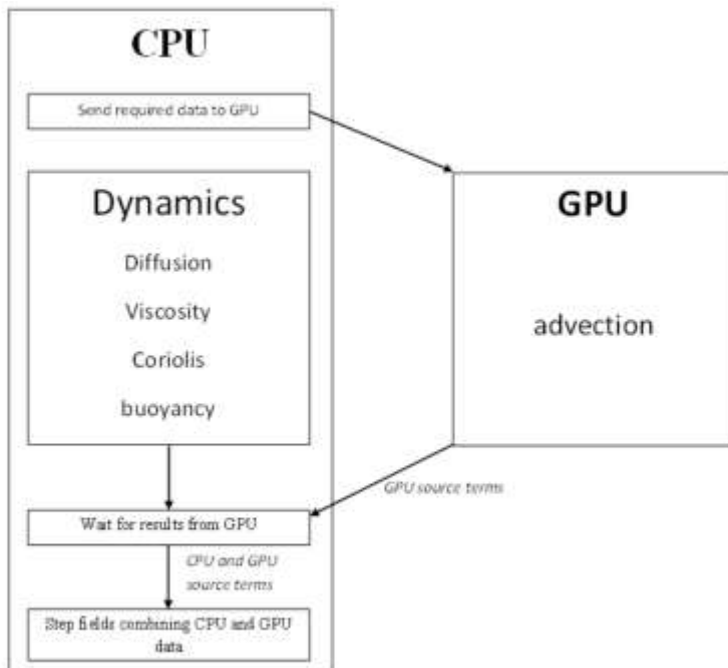
- Weak scaling on ARCHER, stratus cloud test-case 65536 grid points per process, modelled for 10000 simulation seconds

epcc

# Back to 2015……

- A couple of EPCC MSc students explored GPU acceleration of MONC on Piz Daint
  - K20X GPUs using OpenACC to decorate the parts
  - Battling Cray compiler maturity



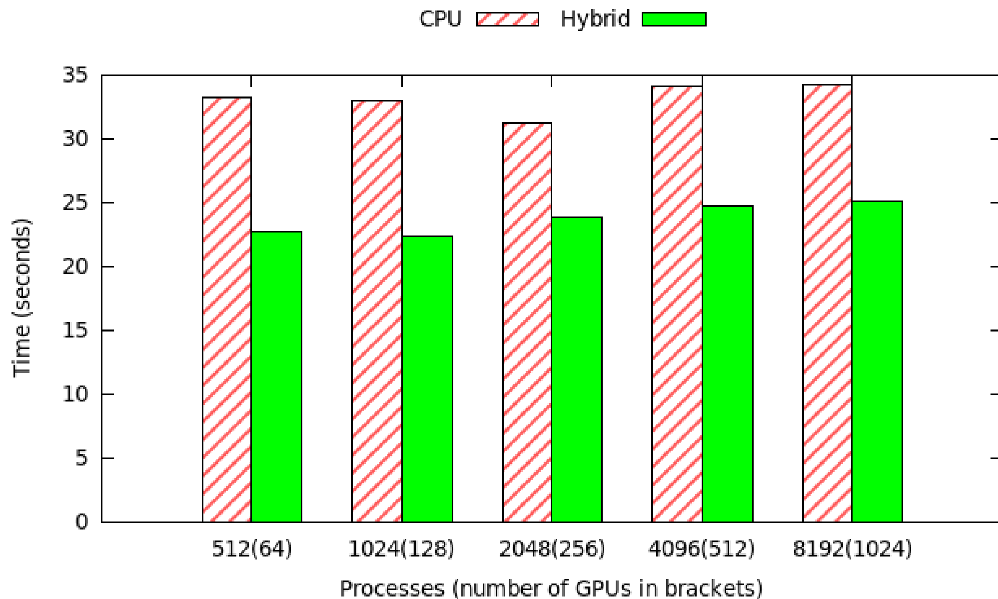




- The key is that dynamical core activities can run in any order and results (source terms) are simply added with a running total.
  - Therefore, can have some components on CPU and some on GPU that are all running concurrently
- A whole bunch of optimisations explored and adopted
  - Such as data transfer

epcc

# Driving with OpenACC

- The compiler support etc all felt very early at the time
  - Lots of errors with the Cray compiler, especially challenges around derived types




*Weak scaling on K20X GPUs, 262144 local grid points for tank experiment test-case*

```
!$acc update device(u, v, w) async(1)
………
!$acc update device(theta) async(2)
………

!$acc parallel loop collapse(3) wait(1,2) async(30) default(none) present(u, v, w,
theta, stheta, start_index, end_index, tcx, tcy, tzc1, tzc2, tzd1, tzd2)

do x = start_index(X_INDEX), end_index(X_INDEX)
    do y = start_index(Y_INDEX), end_index(Y_INDEX)
        do k = start_index(Z_INDEX), end_index(Z_INDEX)
                ……
        end do
    end do
end do

!$acc update host(stheta) wait(30) async(31)
………
!$acc wait(31)
```

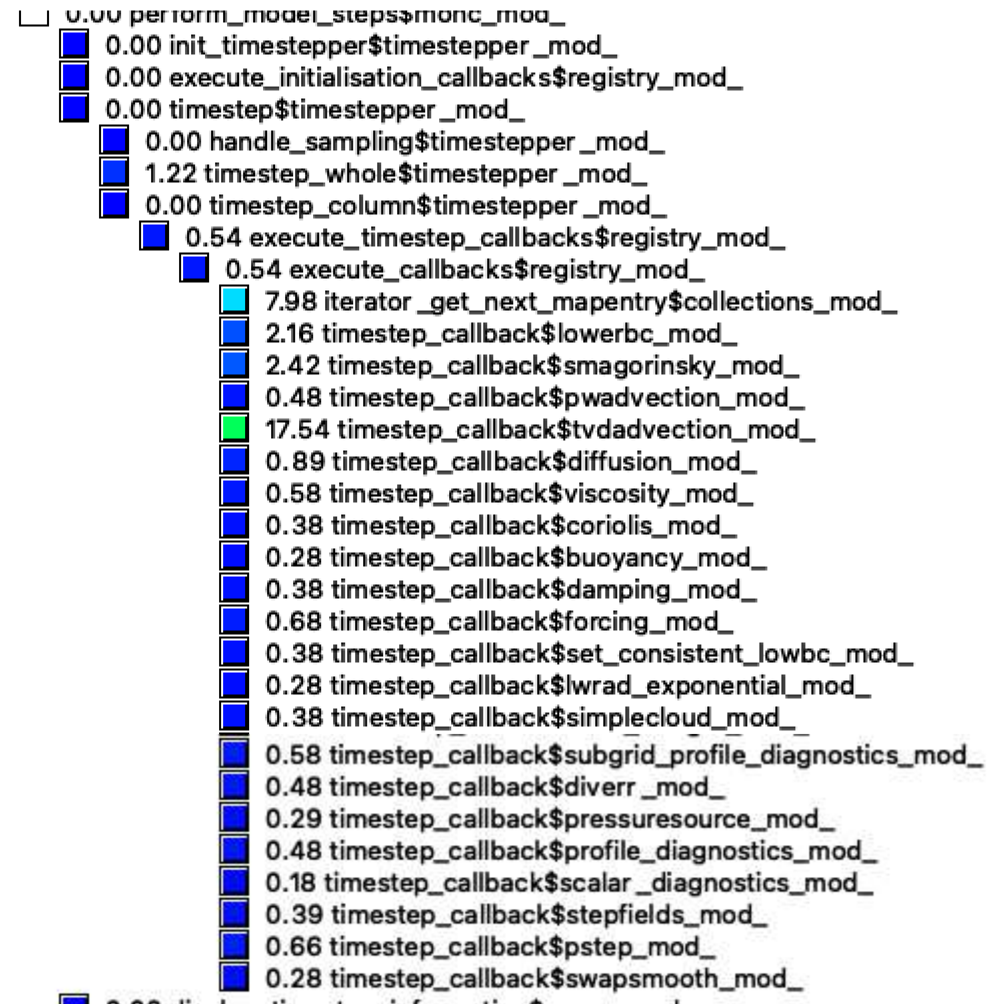
epcc

# Where is the code overhead?

- The previous GPU effort cheated a little as focussed on PW advection
  - The simplest advection from a code perspective but also quite low overall runtime

- Advection is the bottleneck, but TVD advection which is a far more complex scheme
  - This is a high-order flux-limited advection scheme.
  - Around 300 FLOPs per grid point per field

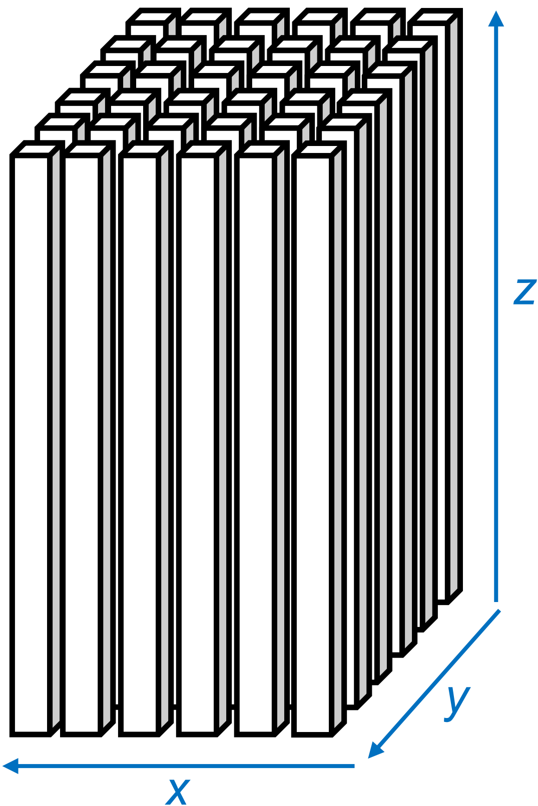
| Component | Category | % runtime |
|---|---|---|
| PW advection | Dynamics | 4% |
| TVD advection | Dynamics | 37% |
| Profile diagnostics | Dynamics | 9% |
| Swap smooth | Dynamics | 3% |
| Mean profiles | Dynamics | 10% |
| FFT solver | Pressure solver | 28% |




epcc

# Adopting OpenMP

- MONC is used on a range of HPC machines, so portability is very important
  - In this work we are focussing on using OpenMP *target* to drive the GPU offload




```
!$OMP target enter data map(alloc: su, sv, sw, sth, sq)
……
!$OMP target enter data map(to: u, v, w, th, q)
……
!$OMP target teams distribute parallel do collapse(2) nowait
do x = local_domain_start_index(X_INDEX), local_domain_end_index(X_INDEX)
    do y = local_domain_start_index(Y_INDEX), local_domain_end_index(Y_INDEX)
          call advect_scalar_field(y, x, dtm, u,&
            v, w, th, sth, global_grid,&
            local_grid, parallel, halo_column, field_stepping)
    end do
end do
!$OMP end target teams distribute parallel do
……
#pragma omp taskwait

!$OMP target exit data map(from: su, sv, sw, sth, sq)
```

- Beneath this there are 11 Fortran subroutines
  - Copy static data on once, avoidance of global/shared variables, pulling out MPI communications, expanding dimensions of flux arrays so each GPU thread is writing to its own memory

epcc

# AMD Instinct MI210 GPUs are…. a bit slow

| | ARCHER2 CPU (s) | ARCHER2 GPU no USM (s) | ARCHER2 GPU with USM (s) |
|---|---|---|---|
| First transfer | - | 0.233 | - |
| First compute | - | 0.053 | 0.355 |
| **First total** | - | **0.286** | **0.355** |
| Subsequent transfer | - | 0.00002 | - |
| Subsequent compute | 0.006 | 0.042 | 0.103 |
| **Subsequent total** | **0.006** | **0.042** | **0.103** |

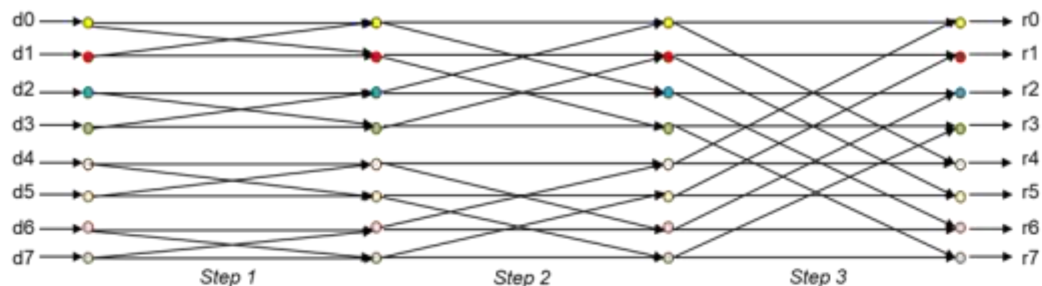
- Full status cloud testcase over three MPI processes, one process per GPU (so three GPUs in total). Time is for a TVD advection over *th* for a timestep.
- The AMD toolchain on ARCHER2 is also a pain as it doesn't handle asynchronous data transfers for derived types, so these need to be manually unpacked/repacked which adds to boilerplate code

# Nvidia Grace Hopper (GH200) is much faster

| | GH200 CPU (s) | GH200 GPU separate memory (s) | GH200 GPU unified memory (s) |
|---|---|---|---|
| First transfer | - | 0.336 | - |
| First compute | - | 0.006 | 0.017 |
| **First total** | - | **0.286** | **0.017** |
| Subsequent transfer | - | 0.0008 | - |
| Subsequent compute | 0.0020 | 0.0005 | 0.0004 |
| **Subsequent total** | **0.0020** | **0.0013** | **0.0004** |

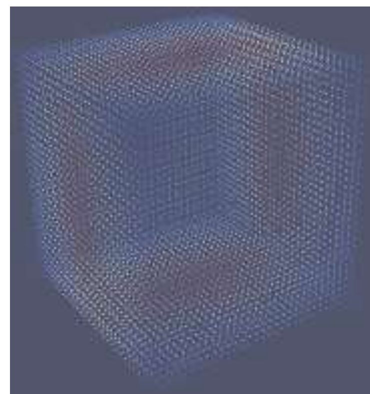
- Full status cloud testcase over three MPI processes, three process per GPU (so one GPU in total). Time is for a TVD advection over *th* for a timestep.
- Using nvfortran 16.1, toolchain is newer and more powerful (we don't have the same issues as with the AMD toolchain)

# Next steps….

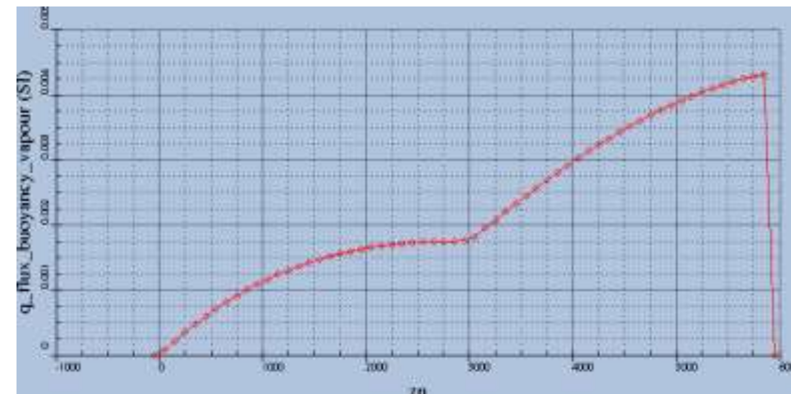



- Have started looking at FFTs
  - FFT GPU libraries, so in theory can work with off the shelf components, but this forces us to place more emphasis on the communication aspect (TVD advection has some halo-swapping but we currently drive this via MPI from the host)

- We need to make IO GPU aware
  - The other piece of the puzzle is transforming raw computational data into higher level diagnostics
  - TVD advection calculates diagnostic values which are often useful for reporting



*Prognostics*



*Diagnostics*

  - Are currently ignoring these, we can copy back to the host and handle as usual, but think that we might be able to optimise this and are looking at integrating ADIOS2 here are part of that

epcc