GungHo: Designing a next generation dynamical core for weather & climate prediction

Thomas Melvin
Overview

Part I
• Gungho Dynamical Core

Part II
• LFRic Atmosphere model
• Code generation
• Next generation modelling system

Acknowledgments

Nerc: Gungho partner Universities
STFC: PSyClone development
Met Office: LFRic development team
Dynamics Research
UM Physics developers
NGMS programme
What we are trying to do

• Move to quasi-uniform mesh to remove polar singularity
• Maintain ‘good’ aspects of current model
  • No computational modes
  • Accurate wave dispersion
  • Semi-Implicit timestepping
  • Reuse subgrid parametrizations
• Improve inherent conservation
• Improve scalability
Model Components

Gungho: Mixed finite element dynamical core

LFRic: Model infrastructure for next generation modelling

PSyclone: Parallel Systems code generation used in LFRic and Gungho

UM: Current modelling environment (UM parametrisations are being reused in LFRic)
Mixed Finite Elements

Mixed Finite Element method gives

- Compatibility: $\nabla \times \nabla \varphi = 0$, $\nabla \cdot \nabla \times v = 0$
- Accurate balance and adjustment properties
- No orthogonality constraints on the mesh
- Flexibility of choice mesh (quads, triangles) and accuracy (polynomial order)
**Mixed Finite Element Method**

\[ \mathbf{W}_0 \xrightarrow{\nabla} \mathbf{W}_1 \xrightarrow{\nabla \times} \mathbf{W}_2 \xrightarrow{\nabla \cdot} \mathbf{W}_3. \quad \mathbf{W}_\theta = \mathbf{k} \cdot \mathbf{W}_2 \]

<table>
<thead>
<tr>
<th>( \mathbf{W}_0 )</th>
<th>Pointwise scalars</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{W}_1 )</td>
<td>Circulation Vectors</td>
</tr>
<tr>
<td>( \mathbf{W}_2 )</td>
<td>Flux Vectors</td>
</tr>
<tr>
<td>( \mathbf{W}_3 )</td>
<td>Volume integrated Scalars</td>
</tr>
<tr>
<td>( \mathbf{W}_\theta )</td>
<td>Pointwise scalars</td>
</tr>
</tbody>
</table>
New developments

• Switch from vector invariant form to advective form momentum equation
  • Allows use of same explicit upwind advection scheme as for scalars: improves stability & accuracy

• Switch from Krylov (BiCGStab) to Multigrid preconditioner for the pressure solver (improves scalability)

• Move to 2x2 (advection, nonlinear) iterations in the timestep from 4x1 (reduces cost)

• Over relaxation ($\tau_{\rho} = 1$) in continuity equation (improves stability)
  \[
  \rho' + \tau_{\rho}\Delta t \nabla \cdot (\rho^* u') = - \left( \rho^{n+1} - \rho^n + \Delta t \nabla \cdot \mathbf{F} \right)
  \]
Gungho Discretisation

- Inspired by iterative-semi-implicit semi-Lagrangian scheme used in UM
- Transport uses high-order, upwind, explicit Eulerian FV scheme
- Wave dynamics use iterative-semi-implicit, lowest order mixed finite element method (equivalent to C-grid/Charney-Phillips staggering)

\[
\int_D \mathbf{v} \cdot \delta_t \mathbf{u} \, dV = - \int_D \mathbf{v} \cdot \left[ \mathcal{A} \left( \mathbf{u}^p, \mathbf{u}^{1/2} \right) + 2\Omega \times \mathbf{u} + \nabla \Phi + c_p \theta \nabla \Pi^\alpha \right] \, dV,
\]

\[
\int_D \sigma \delta_t \rho \, dV = - \int_D \sigma \nabla \cdot \mathcal{F} \left( \rho^p, \mathbf{u}^{1/2} \right) \, dV,
\]

\[
\int_D w \delta_t \theta \, dV = - \int_D w \mathcal{A} \left( \theta^p, \mathbf{u}^{1/2} \right) \, dV,
\]

\[
\int_D \sigma \left( \Pi^{n+1} \right) \frac{1-\kappa}{\kappa} \, dV = \int_D \sigma \frac{R}{p_0} \rho^{n+1} \theta^{n+1} \, dV,
\]

\[
\overline{F}^\alpha \equiv \alpha F^{n+1} + (1 - \alpha) F^n
\]

\[
\mathcal{A} \left( q, \mathbf{u} \right) \equiv \int_t^{t+\Delta t} \mathbf{u} \cdot \nabla q \, dt,
\]

\[
\mathcal{F} \left( q, \mathbf{u} \right) \equiv \int_t^{t+\Delta t} u q \, dt,
\]

Timestep

- **Compute Slow Physics:** $S^n(q^n)$
- **Compute dynamics forcing:** $F^n(q^n)$
- **Advect fields:** $A(q^p, u) = A(q^n + S^n + F^n, u)$
- **Compute Fast Physics:** $S^{n+1}(q^{n+1})$
- **Compute dynamics forcings:** $F^{n+1}(q^{n+1})$
- **Solve linear system:** $Lq' = R$

Dynamics forcing (pgf, Coriolis)

Fast (convection, bl) and Slow (radiation, gwd) subgrid terms

Transport

$$Lq' = R = F^{n+1} + F^n + S^{n+1} + S^n + A(q^p, u)$$
Transport

- Advection equation solved using a multi-stage scheme
- Field ($\bar{q}$) is reconstructed at each stage using a high order upwind polynomial
- Flux ($\mathcal{F}$) or advective ($\mathcal{A}$) increment used to advect field is diagnosed and used in iterative-implicit scheme
- Wind field is fixed during the advection step $\bar{u}^{1/2} \equiv \frac{1}{2}(u^{(k)} + u^n)$
- Moisture is transported on a shifted vertical mesh using flux form scheme $\rightarrow$ moisture mass is conserved
Linear Solver

Iteratively (GCR) solve mixed system

\[ \mathcal{L}(x^*) [u', \theta', \rho', \Pi']^T = -\mathcal{R}^{(k)}, \]

Form approximate Schur complement

\[ \mathcal{L} \rightarrow H, \quad \mathcal{R} \rightarrow R_H, \]

Solve (Multigrid) Helmholtz system

\[ H\Pi' = R_H, \]

Apply smoother on each level

\[ \tilde{\Pi}' \leftarrow \tilde{\Pi}' + \omega \tilde{H}_z^{-1} (B - H\tilde{\Pi}') \]

Error is determined by convergence of mixed system. Multigrid v-cycle used to speed up convergence.
Multigrid Preconditioner

- Helmholtz system $H \Pi' = R$ solved using a single Geometric-Multi-Grid V-cycle with block-Jacobi smoother

$$H = M_3^{\Pi^*} + \left( P_{3*}^{\Pi^*} M_{\theta}^{-1} P_{\theta^2, \theta}^{\Pi^*} + M_{3*}^{\Pi^*} M_{3}^{-1} D^{\Pi^*} \right) \left( M_{2, C}^{\Pi^*} \right)^{-1} G$$

- Block-Jacobi smoother with small number (2) of iterations on each level
- Smoother: Exact vertical solve: $\tilde{H}_{z}^{-1}$

$$\tilde{\Pi}' \leftarrow \tilde{\Pi}' + \omega \tilde{H}_{z}^{-1} \left( B - H\tilde{\Pi}' \right)$$

- Reduced number of operator (H) applications
Questions?
Results
2D Schar hill
Balance

DCMIP2012 Test 200: Balanced atmosphere over a large mountain

Gungho C96
max: 4.951393e-05, min: -4.412776e-05

Vertical velocity

ENDGame 1degree
W, max = 9.7865e-05, min = -6.2345e-05
Baroclinic Wave: Day 10

Gungho C96

ENDGame (1deg)
Grid Imprinting

Vertical velocity (with zonal mean removed) after 1 day of baroclinic wave simulation

C96, Dt=900s  
C192, Dt=450s  
C384, Dt=225s
Code Generation: PSyclone

- PSyclone reads the code and generates interfaces based upon defined rules
- Parallelisation & optimisation strategies are defined via a simple script


Code generation: User code

Compute: \( u + \Delta t \nabla p \)

```fortran
subroutine gradient_code(nlayers, &
    gradient, pressure, &
    ndf_w2, undf_w2, map_w2, w2_diff_basis, &
    ndf_w3, undf_w3, map_w3, w3_basis, &
    nqp_h, nqp_v, w_h, w_v )

implicit none

! ...

! Loop over layers in the cell
do k = 0, nlayers-1
    ! Loop over quadrature points
    do qpl = 1, nqp_h
        ! Compute the gradient
        grad_p = 0.0 r def
        do df = 1, ndf_w3
            grad_p = grad_p + pressure(map_w2(df)+k)*w3_basis(1,df,qpl,qp2) &
        end do
        ! Multiply by test function and integrate
        do df = 1, ndf_w2
            gradient(map_w2(df)+k) = gradient(map_w2(df)+k) &
            + w_h(qpl)*w_v(qp2)*w2_diff_basis(1,df,qpl,qp2)*grad_p &
        end do
    end do
end do

end subroutine gradient_code
```
Code generation: Auto generation

Psyclone parses:
Algorithm
Kernel metadata
and produces:
New algorithm
& Psy Layer

---

```fortran
! Compute rhs = u + dt*grad(p)
call rhs%initialise(u%get_function_space())
call grad%initialise(u%get_function_space())
call invoke( setval_c(grad, 0.0, r_def), 
    gradient_kernel_type(grad, pressure, qr), & 
    aX_plus_Y(rhs, dt, grad, u) )

*****************************************************************************
!* The type declaration for the kernel. 
!* Contains the metadata needed by the Psy layer.
*****************************************************************************
type, public, extends(kernel_type) :: gradient_kernel_type
private
type(arg_type) :: meta_args(z) = (/ &
    arg_type(GH_FIELD, GH_INC, W2), &
    arg_type(GH_FIELD, GH_READ, W3) &
    /)
type(func_type) :: meta_funcs(z) = (/ &
    func_type(W2, GH_DIFF_BASIS), &
    func_type(W3, GH_BASIS) &
    /)
integer :: iterates_over = CELLS
integer :: gh_shape = GH_QUADRATURE_XYoz
contains
    procedure, nospass :: gradient_code
end type

SUBROUTINE invoke_0(grad, pressure, rhs, dt, u, qr) 
    ...
    !$omp end parallel
    IF (pressure_proxy%dirty(depth=1)) THEN
        CALL pressure_proxy%halo_exchange(depth=1)
    END IF
    DO colour1, colour2
        !$omp parallel default(shared), private(cell)
        !$omp do schedule(static)
        DO cell=cell, mesh%get_last_halo_cell_per_colour(colour,1)
            CALL gradient_code(layers,
                grad_proxy%data, pressure_proxy%data, &
                ndf_w2, undf_w2, map_w2(:, cmap(colour, cell)), &
                diff_basis_w2, &
                ndf_w3, undf_w3, map_w3(:, cmap(colour, cell)), &
                basis_w3, mp_xy, np_z, weights_xy, weights_z)
        END DO
        !$omp end do
    !$omp end parallel
END DO

! Set halos dirty/clean for fields modified in the above loop
CALL grad_proxy%set_dirty()

! Set halos dirty/clean for fields modified in the above loop
CALL rhs_proxy%set_dirty()

$omp end master
END SUBROUTINE invoke_0
```
Code generation: Auto generation

! Compute rhs = u + dt*grad(p)
call rhs%initialize( u%get_function_space() )
call grad%initialize( u%get_function_space() )
call invoke( setval_c(grad, 0.0 r def), & gradient_kernel_type(grad, pressure, qr), & aX_plus_Y(rhs, dt, grad, u) )

SUBROUTINE invoke_0(grad, pressure, rhs, dt, u, qr)

...

$omp parallel
IF (pressure_proxyhsis_dirty(depth=1)) THEN
CALL pressure_proxyhsis_exchange(depth=1)
END IF

$omp end parallel

MPI

Horizontal Looping

DO colour=1,ncolour
  !$omp parallel default(shared), private(cell)
  !$omp do schedule(static)
  DO cell=1,nmeshget_last_halo_cell_per_colour(colour,1)

Kernel Call

CALL gradient_code(layers, 
  grad_proxydata, pressure_proxydata, 
  nbf w2, unb w2, map w2(:,cmap(colour, cell)), 
  diff basis w2, 
  nbf w3, unb w3, map w3(:,cmap(colour, cell)), 
  basis w3, np_xy, np z, weights xy, weights z)

END DO
  !$omp end do
$omp end parallel
END DO

OpenMP

Built-in

DO df=1,rhs_proxyhspaceget_last_df_annexed()
  rhs_proxydata(df) = dt*grad_proxydata(df) + u_proxydata(df)
$omp end do

MPI master
CALL rhs_proxyset_dirty()
$omp end master

$omp end parallel
END SUBROUTINE invoke_0
LFRic Atmosphere

- Gungho dynamical core
- Unified model subgrid physics
- Within the LFRic infrastructure
UM Physics

- LFRic uses existing UM Physics (+ code repository)
- Can run in single column model (SCM) mode to compare output
- LFRic uses k-first indexing while UM uses i-first
- i-first allows physics to work on a large segment of data giving a performance boost. LFRic has to use single columns as a segment
- Investigating transposing data in LFRic

```
LFRic

do ij = 1, ncells
  do k = 0, nlayers - 1
    q(map(1,ij) + k) = rhs
  end do
end do
```

```
UM

do k = 1, nlayers
  do j = 1, nrows
    do i = 1, ncols
      q(i,j,k) = rhs
    end do
  end do
end do
```
Couple UM physics to Gungho dynamical core
- Read in reconfigured UM start dump
- I/O using XIOS
- Parallel code layer automatically generated

Dynamical core + subgrid physics
No land surface or orography
Proscribed surface temperature

Aquaplanet & beyond
Aquaplanet

- Comparison to current ENDGame (SISL) dynamical core over 2yr simulation
- Same subgrid physics, vertical mesh & timestep

Reversed jet in UM, not present in LFRic
LFRic jets extend higher
Aquaplanet

850hPa Cloud Ice

850hPa Vertical velocity

Time averages over 800 days of simulation
"Reformulating and redesigning our complete weather and climate research and operational/production systems, including oceans and the environment, to allow us to fully exploit future generations of supercomputer for the benefits of society"
## NGMS Projects

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gungho Atmospheric Science Project</strong></td>
<td>Develop atmospheric science aspects &amp; deliver model scientifically as good as UM</td>
</tr>
<tr>
<td><strong>LFRic Infrastructure Development</strong></td>
<td>Deliver infrastructure to replace the UM scalable for future platforms</td>
</tr>
</tbody>
</table>
| **LFRic Inputs**       | - Tools to ingest fixed & time-varying fields.  
                          - Include initial conditions, ancillary fields and LBCs |
| **LFRic Diagnostic Infrastructure** | Development of research diagnostics and research workflow capabilities |
| **NG-Marine Systems**  | - Deliver scalable marine systems including ocean, sea-ice & wave models |
| **NG-Coupling**        | - OASIS3-MCT coupled components |
| **NG-DA**              | - NGMS-ready coupled atmos/ocean DA  
                          - JEDI as a DA framework |
| **NG-OPS**             | - Processing of NWP observational data for NG-DA |
| **NG-VAT (Visualisation Analysis Tools)** | - Support for visualisation and evaluation tools used by scientists |
| **FAB Build System**   | - Development of new build systems for NGMS components |
| **NG-R2O**             | - Support transition of NGMS capability from research to NWP operations |
| **NG-Composition**     | - Coordination of aerosol & chemistry development within NGMS |
| **NG-R2C**             | - Support transition of capability from research to climate production |
| **NG-ADAQ**            | - Development of dispersions models (e.g. NAME) for next generation computing |
| **NG-VER**             | - Development of NWP verification capability for NGMS |
Summary

- Gungho uses a mixed finite element + finite volume transport on a cubed sphere
- Shows similar accuracy to ENDGame
- Has been coupled to UM subgrid physics
  - Run in aquaplanet mode
  - Functionality to run in global atmosphere more exists but needs to be tested
- Only one part of NGMS
Questions?
**Linear Solver**

- **Quasi-Newton Method:** \( \mathcal{L} (x^*) x' = -\mathcal{R} (x^{(k)}) \).
- Linearized around reference state (previous timestep state) \( x^* \equiv x^n \)
- Solve for increments on latest state: \( x' \equiv x^{(k+1)} - x^{(k)} \)
- Semi-Implicit system contains terms needed for acoustic and buoyancy terms

\[
\mathcal{L} (x^*_{\text{phys}}) x'_{\text{phys}} = \begin{cases} 
    u' - \mu \left( \frac{n_b \cdot u'}{n_b \cdot z_b} \right) z_b \\
    + \tau_u \Delta t c_p \left( \theta' \nabla \Pi^* + \theta^* \nabla \Pi' \right), \\
    \rho' + \tau_\rho \Delta t \nabla \cdot (\rho^* u'), \\
    \theta' + \tau_\theta \Delta t \nabla \cdot \nabla \theta^*, \\
    \frac{1 - \kappa}{\kappa} \frac{\Pi'}{\Pi^*} - \frac{\rho'}{\rho^*} - \frac{\theta'}{\theta^*}.
\end{cases}
\]

Linear Solver

- Solver Outer system with Iterative (GCR) solver

\[
\begin{pmatrix}
M_2^{\mu,C} & -P_2^{\Pi*} & -G^{\theta*} \\
D^\rho & M_3 & \\
0 & M_\theta & M_3^{\Pi*}
\end{pmatrix}
\begin{pmatrix}
\tilde{u}' \\
\tilde{\rho}' \\
\tilde{\theta}'
\end{pmatrix}
= 
\begin{pmatrix}
-\mathcal{R}_u \\
-\mathcal{R}_\rho \\
-\mathcal{R}_\theta \\
-\mathcal{R}_{\Pi}
\end{pmatrix}
\]

- Contains all couplings
- Preconditioned by approximate Schur complement for the pressure increment
- Velocity and potential temperature mass matrices are lumped
  - Drops non-orthogonal terms (cubed-sphere, terrain following)