PSyclone - Portability and scalability for a large KNL system.

Sergi Siso (sergi.siso@stfc.ac.uk)
High-Performance Software Engineer
IPCC @ Hartree, STFC Daresbury Laboratory, United Kingdom
Work done with: Rupert Ford, Andrew Porter, Christopher Maynard
1. Background: LFRic and GungHo
2. The PSyKAI model and PSyclone
3. PSyclone Code Example
4. Scafell Pike: Intel Skylake + KNLs
   1. Vectorization and serial performance
   2. Parallelisation strategies
5. Conclusions and Future work
BACKGROUND: LFRIC AND GUNGHO
Background: LFRic and GungHo projects

- **LFRic project**
  - Met Office (UK) project to develop a replacement for the Unified Model (2020 timeframe)
  - Named in honour of Lewis Fry Richardson (first numerical weather ‘prediction’)
  - UM is a large software (~1M loc, hundreds staff working)
  - Aims to be at least as ‘good’, scientifically as the UM
  - Achieve good performance on current and future supercomputers.

- **GungHo project**
  - Initial investigation project
  - Produced a set or recommendations for the LFRic implementation.
GungHo Project

Need a more scalable dynamical core

Need to make porting the codes from machine to machine easier:
Flexible Implementation
GungHo Science Recommendations

- Cubed Sphere Mesh
- Extruded in vertical dimension
- Mixed finite-element scheme
- Multigrid (under development)
The UM has ~1M lines of code with a relatively flat performance profile.
- Optimising UM is very labour intensive

UM has outlasted many, many generations of supercomputer
- Porting it to a new architecture that may only be around for a few years is not feasible

Separation of concerns
Separation of concerns

**Natural Science Knowledge**
- Finite element/volume/difference-specific
- Time-stepping
- Operations over a mesh
- Typically same operation at each element/volume/point

**Computational Science Knowledge**
- Parallelism
- Data decomposition
- Data locality
- Architecture/platform dependent optimizations

---

**Domain Scientists**

**HPC Experts**
THE PSYKAL MODEL AND PSYCLONE
PSyKAI model

- Operates on full fields
- Operates on local elements or columns
Given domain-specific knowledge and information about the Algorithm and Kernels, the Parallel System layer can be generated by ...
PSyclone

- A domain-specific source-to-source (Fortran -> Fortran) compiler for embedded DSL(s)
  - Supports distributed- and shared-memory parallelism
  - Reduces programmer errors (both correctness and performance) in implementing parallel code
  - Optimisations encoded as a ‘recipe’ rather than baked into the scientific source code
  - Different recipes for different architectures
PSyclone architecture

- Alg Code
- Parser
- Kernel Codes
- Transformation
- Algorithm Generator
- PSy Generator
- Alg Code
- PSy Code

Arrows indicate the flow of information:
- AST → Algorithm Generator
- info → PSy Generator
- schedule
- f90
- psy
- f90
PSYCLONE CODE EXAMPLE
Algorithm Layer example

call invoke(
    continuity(ssha_t, sshn_t, sshn_u, sshn_v, hu, hv, un, vn, rdt),
    momentum_u(ua, un, vn, hu, hv, ht, ssha_u, sshn_t, sshn_u, sshn_v),
    momentum_v(va, un, vn, hu, hv, ht, ssha_v, sshn_t, sshn_u, sshn_v),
    bc_ssh(istp, ssha_t),
    bc_solid_u(ua),
    bc_solid_v(va),
    bc_flather_u(ua, hu, sshn_u),
    bc_flather_v(va, hv, sshn_v),
    copy(un, ua),
    copy(vn, va),
    copy(sshn_t, ssha_t),
    next_sshu(sshn_u, sshn_t),
    next_sshv(sshn_v, sshn_t)
)

subroutine continuity_code(ji, jj, 
    ssha, sshn, sshn_u, sshn_v, 
    hu, hv, un, vn, rdt, e12t)

    implicit none
    integer, intent(in) :: ji, jj
    real(wp), intent(in) :: rdt
    real(wp), dimension(:,:), intent(in) :: e12t
    real(wp), dimension(:,:), intent(out) :: ssha
    real(wp), dimension(:,:), intent(in) :: sshn, sshn_u, sshn_v, 
        hu, hv, un, vn

! Locals
real(wp) :: rtmp1, rtmp2, rtmp3, rtmp4

rtmp1 = (sshn_u(ji ,jj ) + hu(ji ,jj )) * un(ji ,jj )
rtmp2 = (sshn_u(ji-1,jj ) + hu(ji-1,jj )) * un(ji-1,jj )
rtmp3 = (sshn_v(ji ,jj ) + hv(ji ,jj )) * vn(ji ,jj )
rtmp4 = (sshn_v(ji ,jj-1) + hv(ji ,jj-1)) * vn(ji ,jj-1)

ssha(ji,jj) = sshn(ji,jj) + (rtmp2 - rtmp1 + rtmp4 - rtmp3) * &
    rdt / e12t(ji,jj)

end subroutine continuity_code
do jj = ssha%internal%ystart, ssha%internal%ystop, 1
  do ji = ssha%internal%xstart, ssha%internal%xstop, 1

    call continuity_code(ji, jj,
      ssha%data, sshn_t%data, &
      sshn_u%data, sshn_v%data, &
      hu%data, hv%data, un%data, vn%data, &
      rdt, sshn_t%grid%area_t)
  
  end do
end do

do jj = ua%internal%ystart, ua%internal%ystop, 1
  do ji = ua%internal%xstart, ua%internal%xstop, 1

    call momentum_u_code(ji, jj, &
      ua%data, un%data, vn%data, &
      hu%data, hv%data, ht%data, &
      ssha_u%data, sshn_t%data, &
      sshn_u%data, sshn_v%data, &
      un%grid%tmask, &
      un%grid%dx_u, &
      un%grid%dx_v, &
      un%grid%dx_t, &
      un%grid%dy_u, &
      un%grid%dy_t, &
      un%grid%area_u, &
      un%grid%gphiu)
  
  end do
end do
PSyclone Internal representation

GOSchedule[invoke='invoke_0', Constant loop bounds=True]

Loop[type='outer', field_space='ct', it_space='internal_pts']
  Loop[type='inner', field_space='ct', it_space='internal_pts']
    KernCall continuity(ssha_t, sshn_t, sshn_u, sshn_v, hu, hv, un, vn, rdt, area_t) [mod_inline=False]

Loop[type='outer', field_space='cu', it_space='internal_pts']
  Loop[type='inner', field_space='cu', it_space='internal_pts']
    KernCall momentum_u(ua, un, vn, hu, hv, ht, ssha_u, sshn_t, sshn_u, sshn_v, tmask, dx_u, dx_v, dx_t, dy_u, dy_t, area_u, gphiu) [mod_inline=False]

...
```python
def trans(psy):
    ''' Python script intended to be passed to PSyclone's generate() function via the -s option. Applies OpenMP to every loop before enclosing them all within a single OpenMP PARALLEL region. '''

    from psyGen import TransInfo
    tinfo = TransInfo()
    ltrans = tinfo.get_trans_name('GOceanOMPLoopTrans')
    rtrans = tinfo.get_trans_name('OMPParallelTrans')

    schedule = psy.invokes.get('invoke_0').schedule
    schedule.view()

    # Apply the OpenMP Loop transformation to *every* loop
    # in the schedule
    for child in schedule.children:
        newschedule, _ = ltrans.apply(child)
        schedule = newschedule

    # Enclose all of these loops within a single OpenMP
    # PARALLEL region
    newschedule, _ = rtrans.apply(schedule.children)

    newschedule.view()
    psy.invokes.get('invoke_0').schedule = newschedule
    return psy
```
PSyclone Internal representation

GOSchedule[invoke='invoke_0', Constant loop bounds=True]

Loop[type='outer', field_space='ct', it_space='internal_pts']
  Loop[type='inner', field_space='ct', it_space='internal_pts']
    KernCall continuity(ssha_t, sshn_t, sshn_u, sshn_v, hu, hv, un, vn, rdt, area_t) [mod_inline=False]

Loop[type='outer', field_space='cu', it_space='internal_pts']
  Loop[type='inner', field_space='cu', it_space='internal_pts']
    KernCall momentum_u(ua, un, vn, hu, hv, ht, ssha_u, sshn_t, sshn_u, sshn_v, tmask, dx_u, dx_v, dx_t, dy_u, dy_t, area_u, gphiu) [mod_inline=False]

...
SCAFELL PIKE: INTEL SKYLAKE + KNLS
Hartree Centre Systems

Napier (and Iden extension)
- 440 – 2 x Intel Xeon IvyBridge E5-2697v2 12 cores up to 3.5 GHz
- 42 Intel Xeon Phi KNL 5110P
- Mellanox FDR Infiniband

Scafell Pike
- 840 – 2 x Intel Xeon Gold E5-6142 v5 (formerly Skylake) 16 core up to 3.7GHz
- 840 – self-hosted Intel Xeon Phi KNL 7210 nodes 64 cores up to 1.5 GHz
- 27000 + 54144 cores
- Mellanox EDR Infiniband
MPI Scalability (LFRic compute)

seconds

nodes (IvB x24, KnL x60 cores)
MPI Analysis

Communication Pattern

MPI Time distribution

- MPI MPI_Isend: 2%
- MPI MPI_Irecv: 3%
- MPI MPI_Wait: 2%
- MPI MPI_Barrier: 1%
- MPI MPI_Allreduce: 2%
- MPI MPI_AlltoAll: 0%
- MPI MPI_Allgather: 1%
- MPI MPI_Isend: 2%

Compute: 89%
MPI Timeline
KNL Clustering and Memory Modes

KNL configuration MPI only
(60 MPI - Test: 6x24x24 30 layers 1it)
## Most expensive routines on the solver

<table>
<thead>
<tr>
<th>Routine</th>
<th>Xeon %</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix_vector_code (matrix_vector_kernel_mod.f90)</td>
<td>6.83%</td>
</tr>
<tr>
<td>dg_matrix_vector_code (dg_matrix_vector_kernel_mod.f90)</td>
<td>5.21%</td>
</tr>
<tr>
<td>__libm_pow_e7 (called from pret_pressure_gradient_bd_kernel_mod.f90)</td>
<td>5.14%</td>
</tr>
<tr>
<td>transpose_matrix_vector_code (transpose_matrix_vector_kernel_mod.f90)</td>
<td>3.01%</td>
</tr>
<tr>
<td>pert_pressure_gradient_bd_code (pert_pressure_gradient_bd_kernel_mod.f90)</td>
<td>2.42%</td>
</tr>
<tr>
<td>coordinate_jacobian (coordinate_jacobian_mod.f90)</td>
<td>1.69%</td>
</tr>
</tbody>
</table>

All in the Kernel layer – good for manual optimization
**LFRic KNL VTune Analysis**

### Function / Call Stack

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>Clockticks</th>
<th>Back-End Bound</th>
<th>Memory Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>L1 Hit Rate</td>
<td>L2 Hit Rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L1 Miss Bound</td>
<td>L2 Miss Bound</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SIMD Co...</td>
<td>SIMD Co...</td>
</tr>
<tr>
<td>opt_apply_variable_hx_code</td>
<td>16,828,500,000</td>
<td>97.5%</td>
<td>65.1%</td>
</tr>
<tr>
<td>opt_scaled_matrix_vector_code</td>
<td>9,744,800,000</td>
<td>95.7%</td>
<td>78.3%</td>
</tr>
<tr>
<td>matrix_vector_code</td>
<td>8,396,700,000</td>
<td>99.1%</td>
<td>93.3%</td>
</tr>
<tr>
<td>tri_solve_code</td>
<td>7,783,100,000</td>
<td>95.4%</td>
<td>75.0%</td>
</tr>
<tr>
<td>coordinate_jacobian</td>
<td>6,242,600,000</td>
<td>99.9%</td>
<td>100.0%</td>
</tr>
<tr>
<td>gp_vector_rhs_code</td>
<td>2,626,000,000</td>
<td>99.7%</td>
<td>100.0%</td>
</tr>
<tr>
<td>compute_curl_operator_code</td>
<td>2,249,000,000</td>
<td>100.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>_libm_pow_l9</td>
<td>2,237,300,000</td>
<td>100.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

- Poor L2 usage
- Poor Simd-to-L1/L2 ratios
VectorMatrix Optimization

Refactor arrays and kernel code
(fuse loops, avoid gather/scatters, …)

We also tried MKL and libxsmm but the matrices were too small!
(we will revisit this when Column Matrix Assembly operations available)
Array traversing

- Stencil access 20% size (worth conserving 2D neighbours in cache)
- Local access 80% size (but still benefits from 1D data locality due to cache blocking and prefetching)

<table>
<thead>
<tr>
<th>Colouring (original)</th>
<th>Linear</th>
<th>HFC - Hilbert</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 17 2 18 3 19 4 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 49 34 50 35 51 36 52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 21 6 22 7 23 8 24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37 53 38 54 39 55 40 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 25 10 26 11 27 12 28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41 57 42 58 43 59 44 60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 29 14 30 15 31 16 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45 61 46 62 46 63 48 64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VectorMatrix Optimization

serial gungho-mv

MIC-AVX512  No SIMD

time (s)

Different Array traversal order(serial case)
VectorMatrix Optimization

Test increasing the number of operations inside loop body. Fusing kernels could be introduced to Psyclone in the future.
OpenMP Implementations

- Implemented OpenMP Parallelisation Strategies
  - Atomics
  - Arrays of locks
  - 4-colours
  - Block-colouring (e.g. 2x1)
  - Row-colouring

Par. elements / omp region
Data locality
locks overhead & contention

fewer parallel elements / omp region
Better data locality
OpenMP Implementations

- gungho-mv nlayers first
- Problem size: 32x32 (x16)
- SMT x 1
- OMP Affinity: Scattered
- KNL: cache-quadrant

Locking wins but loses its advantage on large #cores
Static schedule wins but loses its advantage on large #cores
OpenMP Implementations

SMTx4 helps across all implementations
Colouring beats Locking when +10 (4smt) threads

gungho-mv nlayers first
Problem size: 32x32 (x16)
OMP Schedule: Static
OMP Affinity: Scattered
KNL: cache-quadrant
Super-Scalability and Problem Size

We observed a super-scalability effect dependent on the problem size and number of cores.

gungho-mv nlayers first
OMP Strategy: colouring
OMP Schedule: Static
OMP Affinity: Scattered
KNL: cache-quadrant
CONCLUSIONS AND FUTURE WORK
Hartree developed **PSyclone**, a code-generation tool to abstract the parallel implementation and performance tuning from the science code.

**HC IPCC** explored and provided expertise for **KNL** the platform.

**Future work:**
- Continue exploring KNL platform: hybrid MPI+OpenMP, ...
- Bring useful optimizations to PSyclone (automatic expansion to full LFRic codebase)
Thanks for your attention

Sergi Siso : sergi.siso@stfc.ac.uk

http://www.hartree.stfc.ac.uk