Preparing atmospheric modeling codes for the latest generation MIC architecture (KNL)

September 14, 2016

Jim Rosinski
NOAA/ESRL
Intel Contributors

• Mike Greenfield
• Ruchira Sasanka
• Ashish Jha
• Richard Mills
Outline

• Hardware used for FV3 testing
• New features in KNL—user implications
• FV3 overview for software engineers
• Performance issues and non-issues
• Performance results: HSW vs. KNL
• Future directions
## Machine specs

<table>
<thead>
<tr>
<th>Machine</th>
<th>Arch</th>
<th>Cores/node</th>
<th>Hyperthreaded?</th>
<th>CPU</th>
<th>Compiler</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stampede 1.5</td>
<td>KNL</td>
<td>68</td>
<td>Yes (4-way)</td>
<td>1.4 GHz</td>
<td>Ifort 16.0.3</td>
<td>Impi 5.1.3 (Omnipath)</td>
</tr>
<tr>
<td>theia</td>
<td>HSW-EP</td>
<td>24</td>
<td>Yes (2-way)</td>
<td>E5-2690@2.6 GHz</td>
<td>Ifort 15.1.133</td>
<td>Impi 5.0.1.035 (Infiniband)</td>
</tr>
</tbody>
</table>
New features in KNL

- Up to 384 GB DDR4 main memory
- 16 GB High-bandwidth memory (MCDRAM)
- Binary compatible with x86
- 512-bit vectors: enable with –xMIC-AVX512
- Out of order instruction execution
- Self-booting Linux system (no attached host required)
- Bump in core count vs. KNC (68 or 72 vs. 61)
- Clock rate increase vs. KNC (1.4 GHz vs. 1.1)
- Increase in allowed MPI task count per node
Cubed-sphere grid (left) and icosahedral grid (right)

Graphic courtesy Peter Lauritzen (NCAR)
FV3 Overview

• Selected by NWS as next-generation dynamical core to be used in forecast mode (~10 day forecasts) by US national center (NCEP)
• Non-hydrostatic capability required for high-resolution runs (< 10 km)
  – Hydrostatic assumption means force of gravity exactly balances vertical pressure gradient force (dp/dz = -\rho*g)
• Physical parameterizations from current forecast model (GFS) have been enabled in FV3 by Rusty Benson (GFDL)
• 2 horizontal resolutions: c192 (~0.5 degree), c768 (~0.125 degree)
• 127 vertical levels
FV3 code structure

• Hybrid MPI/OpenMP dynamical core
  – Well-suited for KNL
• "Cubed sphere" means 6 separate "faces"
  – Must have at least 1 MPI rank per face
  – Ability to run w/o MPI would be a welcome addition
• (i,j,k) data layout (Fortran ordering)
• Shallow water portions of code thread over “k” and vectorize over “i”
• Vertical dependencies: Remapping portion of code threads over “j” and vectorizes over “i”
• Special handling required for “edge” and “corner” points
FV3 code structure (cont’d)

• Highly vectorized as indicated by compiler opt report
  – Corner calculations excepted (moving ”k” inside would help)
  – Turning off vectorization slowed the code down by ~2X
• MPI communication utilizes FMS “wrapping” infrastructure
  – All messages are “packed” prior to sending, “unpacked” after receiving
  – 2-deep and sometimes 3-deep halos are exchanged
• On average, 17 OMP loops are executed each time step
  – GPTL timers indicate threading overhead is not a problem, even on KNL
  – Many “j” and “i” loops nested inside: Pushing “k” inside will be difficult
Example MPI task layout (4x4) on a single FV3 face (~0.5 degree)
ESRL code mods

• Thread-safe timing library (GPTL)
• Fused a few OMP loops
• Updated compiler flags for HSW
How threading overhead was measured

ret = gptlstart_threadohd_outer (‘c_sw’)

!$OMP PARALLEL DO
do k=1,npz
   ret = gptlstart_threadohd_inner (‘c_sw’)
   call c_sw (. . .)
   ret = gptlstop_threadohd_inner (‘c_sw’)
end do
ret = gptlstop_threadohd_outer (‘c_sw’)

• Each invocation, ”inner” timer notes time taken by slowest thread for each “inner” call
• Each invocation, “outer” timer accumulates difference between its c_sw and slowest “inner” thread
• When timers are printed, overhead is the sum of these differences

Multi-core workshop
Thread scaling (low-res, 12-node)
Low resolution multi-node scaling

FV-DYN-LOOP scaling HSW and KNL (C192 resolution--60km)

- Haswell
- KNL (cache)
- KNL (flat)
- KNL (no mcdram)
- KNL (2-way ht)
- KNL (4-way ht compact)

nx=ny=2
2 node knl: 60 cores per node not 68
3 node knl: 64 cores per node not 68
4 node knl: 66 cores per node not 68
5 node knl: 56 cores per node not 68

5 node hsw: 20 cores per node not 24, 6 nodes

Multi-core workshop
High resolution multi-node scaling

FV-DYN-LOOP scaling Haswell and KNL (C768 resolution - 15km)'

- Haswell
- hsw-linear
- KNL (cache)
- knl-linear

12 node knl used 64 cores per node not 68

Multi-core workshop
### Comm time fraction (hi-res)

<table>
<thead>
<tr>
<th># nodes</th>
<th>HSW comm time (s)</th>
<th>HSW run time (s)</th>
<th>KNL comm time (s)</th>
<th>KNL run time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>27.9</td>
<td>1058.7</td>
<td>26.8</td>
<td>528.5</td>
</tr>
<tr>
<td>24</td>
<td>17.9</td>
<td>533.1</td>
<td>18.0</td>
<td>276.3</td>
</tr>
<tr>
<td>48</td>
<td>12.1</td>
<td>277.3</td>
<td>14.5</td>
<td>153.2</td>
</tr>
</tbody>
</table>
A note about thread pinning...

• As received from GFDL, FV3 code manually pins threads to individual cores inside user code. This was removed for KNL, and modified for theia
• On theia, allowing threads to be assigned to a range of cores incurred little to no penalty, as long as they were not allowed to span sockets.
• On KNL (and KNC), threads are automatically pinned to individual cores
Proposed MPI task layout to address load imbalance from edge/corner points

Multi-core workshop
Summary

- FV3 is well vectorized, and scales well in OpenMP and MPI
  - Corner and edge point calculations contribute some imbalance
  - 17 OMP loops per time step add some load-induced imbalance but little OpenMP overhead
- MCDRAM provides a significant performance benefit
  - Cache and flat modes perform similarly when most or all memory fits in MCDRAM
- Unlike KNC, hyperthreading provides little or no benefit
- KNL performance exceeds HSW
  - Why so much more at high resolution?
- Attempts to push “k” loop inside will involve significant code restructuring
Where Next?

• Further research on why node to node, KNL to HSW performance ratio increases with resolution
• Tweaks to thread and task pinning?
• Adjust number of points owned by MPI tasks to improve load imbalance caused by edge and corner calculations
• Improve MPI performance—can packing be eliminated in some cases?
• Enable FV3 to run without MPI (for testing)
• Can pushing “k” loop inside provide a benefit on KNL (or at least no penalty)?
• Physics optimizations