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

October 25, 2016

Jim Rosinski
NOAA/ESRL
Intel Contributors

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

• Hardware used for FV3 and NIM testing
• New features in KNL—user implications
• NIM model: thread scaling, performance on multiple generations of Intel hardware
• 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>Hyper threaded?</th>
<th>CPU</th>
<th>Compiler</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stampede 1.0</td>
<td>SNB/KNC</td>
<td>16/61</td>
<td>No/4-way</td>
<td>2.7 GHz/1.1 GHz</td>
<td>Ifort 15.0.2</td>
<td>Impi 5.0.2 (Mellanox FDR IB)</td>
</tr>
<tr>
<td>Stampede 1.5</td>
<td>KNL</td>
<td>68</td>
<td>Yes (4-way)</td>
<td>1.4 GHz</td>
<td>Ifort 17.0.0</td>
<td>Impi 17.0.0 (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)
- Self-booting Linux system (no attached host required)
- Binary compatible with x86
  - Exception handling works just like x86 (-fpe0 works)
- Much lower cost to -fp-model precise
- 512-bit vectors: enable with -xMIC-AVX512
- Out of order instruction execution
- Bump in core count vs. KNC (68 or 72 vs. 61)
- Clock rate increase vs. KNC (1.4 or 1.5 GHz vs. 1.1)
- 1 MPI per core allowed (OMP not mandatory)
  - 10 node 110 km NIM run only 20% slower in pure MPI mode
Cubed-sphere grid (left) and icosahedral-hexagonal grid (right)

Graphic courtesy Peter Lauritzen (NCAR)
NIM, MPAS code structure

• Hybrid MPI/OMP
• Fortran, (k,i) data layout
  – Horizontal index of neighbors (i) addressed via lookup table
  – Overhead of indirect addressing amortized by “k” on the inside
• Shallow water parts of the dynamics easily vectorize over “k”
• Transpose from (k,i) to (i,k) and back for physics
  – Minimal impact on model runtime
NIM performance on 2 generations of Intel hardware

30 km resolution, 80 nodes, runtime (sec)

SNB: 17.459 (Compute: 17.459, Communicate: 1.368, Barrier: 0)
KNC: 13.774 (Compute: 13.774, Communicate: 6.509, Barrier: 0)
SNB+KNC: 8.237 (Compute: 8.237, Communicate: 5.942, Barrier: 2.459)
KNL: 2.169 (Compute: 2.169, Communicate: 1.994, Barrier: 4.931)

Workshop on high performance computing
NIM thread scaling on KNL

NIM thread scaling on a single KNL (G5 resolution--220km)

OMP_PROC_BIND=spread
OMP_PLACES=threads

Inverse runtime

OMP_NUM_THREADS
FV3 Overview

• Selected by NWS as next-generation dynamical core to be used in forecast mode (~10 day forecasts)
• 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
• 2 horizontal resolutions provided to ESRL: c192 (~0.5 degree), c768 (~0.125 degree)
• 127 vertical levels
FV3 code structure

• Hybrid MPI/OpenMP dynamical core
  – Well-suited for many-core architectures
• "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)

np_x=193 -> VLEN=48
NX=4

np_y=193 -> nthreads=48
NY=4
ESRL mods to FV3

• Thread-safe timing library (GPTL)
• Fused a few OMP loops
• Updated compiler flags for HSW
• Turn off manual thread pinning on KNC/KNL
How threading overhead was measured

```c
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
FV3 Low resolution multi-node scaling

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

- Haswell
- KNL (flat)
- KNL (cache)
- KNL (no mcdram)
- KNL (2-way ht)
- KNL (4-way ht OMP_PROC_BIND=close)

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

Workshop on high performance computing
FV3 High resolution multi-node scaling

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

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

nx=ny=4
12 node knl used 64 cores per node not 68
# Workshop on high performance computing

## FV3 comm-time compared to run-time (hi-res)

<table>
<thead>
<tr>
<th># nodes</th>
<th>HSW comm time (sec)</th>
<th>HSW run time (sec)</th>
<th>KNL comm time (sec)</th>
<th>KNL run time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>26.8</td>
<td>1053.7</td>
<td>26.3</td>
<td>523.9</td>
</tr>
<tr>
<td>24</td>
<td>16.9</td>
<td>529.5</td>
<td>16.1</td>
<td>260.5</td>
</tr>
<tr>
<td>48</td>
<td>11.1</td>
<td>267.5</td>
<td>10.6</td>
<td>137.4</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 CPU

• On CPU, 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

Workshop on high performance computing
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
• Cost of –fp-model precise flag (to guarantee bitwise identical results) decreased dramatically on KNL
• Pure MPI (no OMP) is possible, and works well
• 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 benefit
• KNL performance exceeds HSW, particularly at higher resolution
• Attempts to push “k” loop inside will involve significant code restructuring
Where Next?

• 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
Backup slides
NIM: Overlapping communication with computation

30 km resolution, 80 nodes, runtime (sec)

- SNB
- KNC
- SNB+KNC
- KNL

No overlap | Overlap
---|---
SNB
KNC
SNB+KNC
KNL

Workshop on high performance computing