Experiences Porting NOAA Weather Model FIM to Intel MIC

September 12, 2012

Jim Rosinski

2012 NCAR workshop on heterogeneous multi-core platforms
Outline

• Porting methodology and validation
• MIC features and programming modes
• Status of FIM model on MIC
• Techniques to speed up FIM on MIC (OpenMP and single-core)
• NO discussion of absolute performance on MIC (non-disclosure agreement with Intel)
Why FIM instead of NIM?

• OpenMP threading is the best way to get good performance on MIC
• FIM was already threaded with OpenMP
Porting Methodology

• Extract 1 time step from full model run on CPU
  – Save required IC info for each kernel
  – Save end-of-timestep info for verification

• Extract model code for kernel of interest
  – Build driver to read in IC info and pass to kernel
  – Create subroutine to compare verification data to kernel results
Porting Methodology (cont’d)

• Gather kernel timing info
  – gptl_lite

• Modify kernel code (e.g. add directives) for new hardware

• Run and compare results and timing
  – Gather stats about max absolute and relative differences
Validation

• Bit-for-bit is best, but roundoff differences are likely
• How do you know you’re getting the right answer?
  – Perturb the initial conditions on “trusted” hardware and compare results, or
  – Use a different (but still trusted) compiler to produce “trusted” differences
  – Compare “trusted” diffs vs. “test” diffs
CPU performance comparison

![Graph showing CPU performance comparison]

- **nehalem 2.6 GHz**
- **westmere 3.5 GHz**
- **sandybridge 2.6 GHz**
- **interlagos_intel 2.2 GHz**
- **interlagos_pgi 2.2 GHz**
Full dycore scaling on a node
## MIC Features (Public)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brand Name</strong></td>
<td>Intel® Xeon Phi™ coprocessor (codenamed Knights Corner)</td>
</tr>
<tr>
<td><strong>Product Available</strong></td>
<td>Shipping production 22nm in 2012</td>
</tr>
<tr>
<td><strong>Physical Core Count</strong></td>
<td>More than 50</td>
</tr>
<tr>
<td><strong>Logical Cores per Physical Core</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>Vector register size</strong></td>
<td>512 bits</td>
</tr>
<tr>
<td><strong>IO Bus</strong></td>
<td>PCIe</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>8 GB GDDR5</td>
</tr>
<tr>
<td><strong>Peak FLOPS</strong></td>
<td>Greater than 1 TFLOP (DP)</td>
</tr>
<tr>
<td><strong>Programming</strong></td>
<td>Linux OS. IP addressable. Intel Developer tools. Common source code with CPU</td>
</tr>
</tbody>
</table>

---

Multi-core Workshop
MIC programming modes

• Offload
  – Host offloads part of calculation to coprocessor
  – Compiler directives describe how to move data

• Native
  – Everything runs on the coprocessor
  – Use existing OpenMP directives
  – NO code mods required to get it running
  – Can use multiple cores via OpenMP and/or MPI
Offload Mode

Node 0
- Host Code
- Offload Code (probably OMP threads)

Node 1
- Host Code
- Offload Code (probably OMP threads)

Host

MPI

IN  OUT

PCIe

Coprocessor

Offload Code (probably OMP threads)
Native Mode

Node 0

Host

Coprocessor

Native MIC Code
MPI ranks + OMP threads

Node 1

Host

Coprocessor

Native MIC Code
MPI ranks + OMP threads

MPI

PCIe

Multi-core Workshop 12
Current status of FIM on MIC

• 4 individual kernels from FIM dynamics extracted, running, and validated in native mode
• Full FIM dynamics also running as a standalone kernel, running, and validated in native mode
• 1 kernel has been extracted, running and validated in offload mode-thanks to help from Intel
• Getting FIM running in native mode required zero mods to source code
Effect of mods to speedup cnuity kernel on MIC

![Graph showing the effect of compiler flags and code mods on cnuity on MIC. The x-axis represents OMP_NUM_THREADS, and the y-axis represents inverse run time (1/s). The graph includes lines for precise_nofma, nofma, fast, firsttouch_scatter, and vector_scatter.]
Vectorization speedup on SandyBridge (X, Y axes differ from previous slide)
Vectorization example (orig)

```fortran
!$OMP PARALLEL DO PRIVATE (k,edg) SCHEDULE (runtime)
   do ipn=ips,ipe
     do k=1,nvl
       anti_tndcy(k,ipn) = 0.

     do edg=1,nprox(ipn)  <-- Compiler only considers inner loop for vectorization
       anti_tndcy(k,ipn) = anti_tndcy(k,ipn) + antifx(k,edg,ipn)
     end do

     anti_tndcy(k,ipn) = -anti_tndcy(k,ipn)*rarea(ipn)
     dp_tndcy(k,ipn,nf) = dplo_tndcy(k,ipn,nf) + anti_tndcy(k,ipn)
     delp(k,ipn) = delp(k,ipn) + adbash1*dp_tndcy(k,ipn,nf) + &
                   adbash2*dp_tndcy(k,ipn,of) + &
                   adbash3*dp_tndcy(k,ipn,vof)
   end do
end do
```
Vectorization example (fixed)

 !$OMP PARALLEL DO PRIVATE (k,edg) SCHEDULE (runtime)
  do ipn=ips,ipe
    do k=1,nvl
      anti_tndcy(k,ipn) = 0.
    end do
  end do
  do edg=1,nprox(ipn)
    do k=1,nvl
      anti_tndcy(k,ipn) = anti_tndcy(k,ipn) + antifx(k,edg,ipn)
    end do
  end do
  do k=1,nvl
    anti_tndcy(k,ipn) = -anti_tndcy(k,ipn)*rarea(ipn)
    dp_tndcy(k,ipn,nf) = dplo_tndcy(k,ipn,nf) + anti_tndcy(k,ipn)
    delp(k,ipn) = delp(k,ipn) + adbash1*dp_tndcy(k,ipn,nf) + &
                 adbash2*dp_tndcy(k,ipn,of) + &
                 adbash3*dp_tndcy(k,ipn,vof)
  end do
end do
Notes on Vectorization

• Only inner loops vectorize
• MIC vector length exceeds even SandyBridge
• \(a^{**}b\) does not vectorize
• Use –vec-report3
• “if” tests can cause problems
  – “condition may protect exception”
  – Fix with “!DIR$ VECTOR ALWAYS”
How to make MIC code run well

• Vectorize
  – 512 bit vector register

• Use lots of OpenMP threads
  – Up to 4X the number of physical cores

• Memory affinity
  – Add code to apply “first touch”
  – Works best with “schedule(static)”
How to make MIC code run well (cont’d)

• Minimize PCIe transfers
• Minimize I/O issued from MIC
• Don’t use –fp-model precise
  – ~2X performance hit using this flag with FIM on MIC
Notes on OpenMP

• Experiment with $\text{KMP\_AFFINITY}$
  – “compact”, “scatter”, “balanced”

• Experiment with $\text{OMP\_SCHEDULE}$
  – Only takes effect with when the attribute
    “schedule(runtime)” is specified in threaded loops
  – Default is “static”
  – Some success with “guided”
Where Next?

- Multiple time steps
- Multiple KNC cards
- OpenMP in physics
- I/O
Summary

• OpenMP is the best way to get performance on MIC
• Whether MIC or GPU, it matters which CPU architecture is being compared to when assessing speedup
• 2 methods to run code on MIC: “offload” and “native”
• FIM benefits greatly from vectorization on MIC
  - helps CPU also