Experiences Porting NOAA Weather Model FIM to Intel MIC

October 2, 2012

Jim Rosinski

2012 ECMWF workshop on High Performance Computing in Meteorology

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

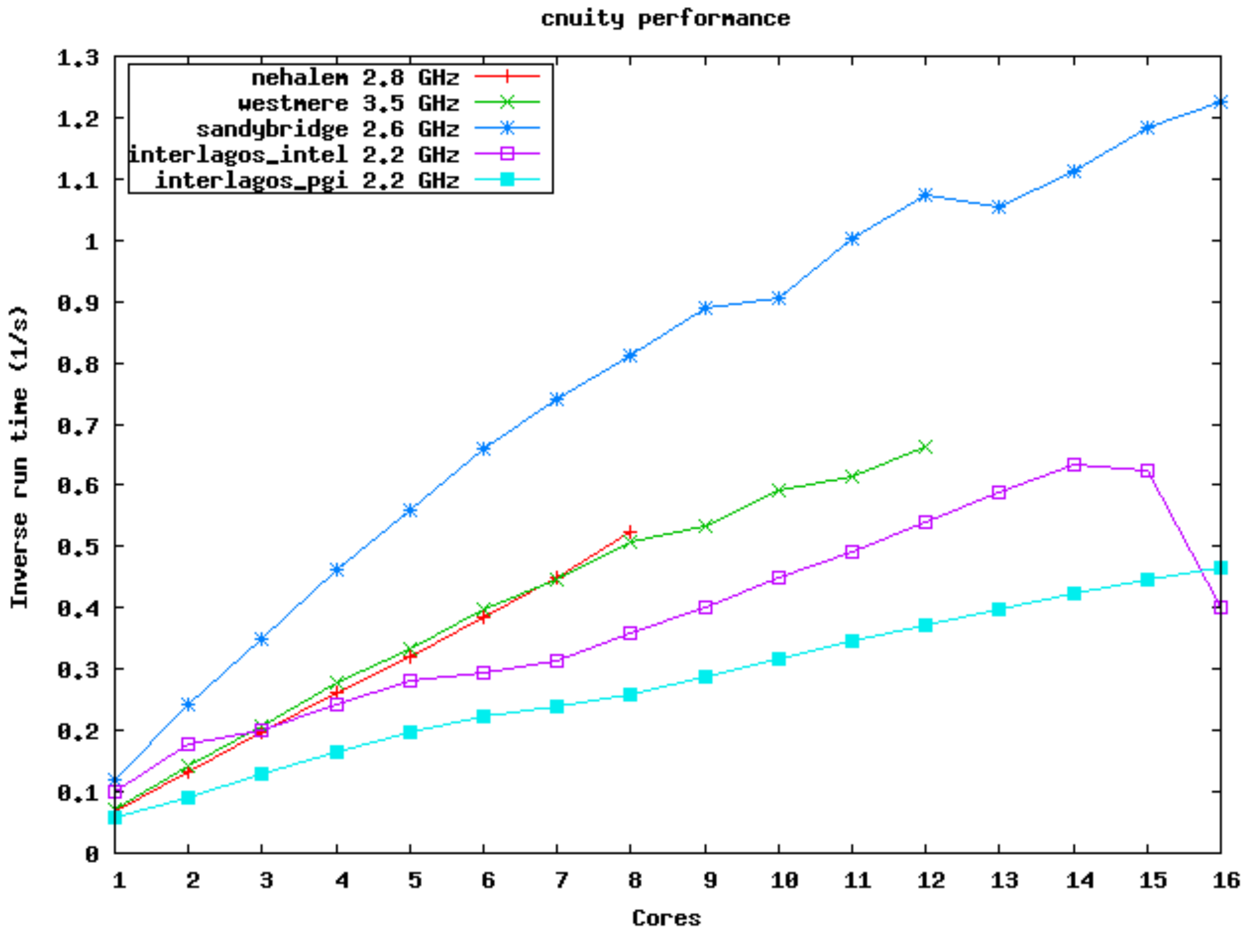
Porting Methodology (cont'd)

- 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

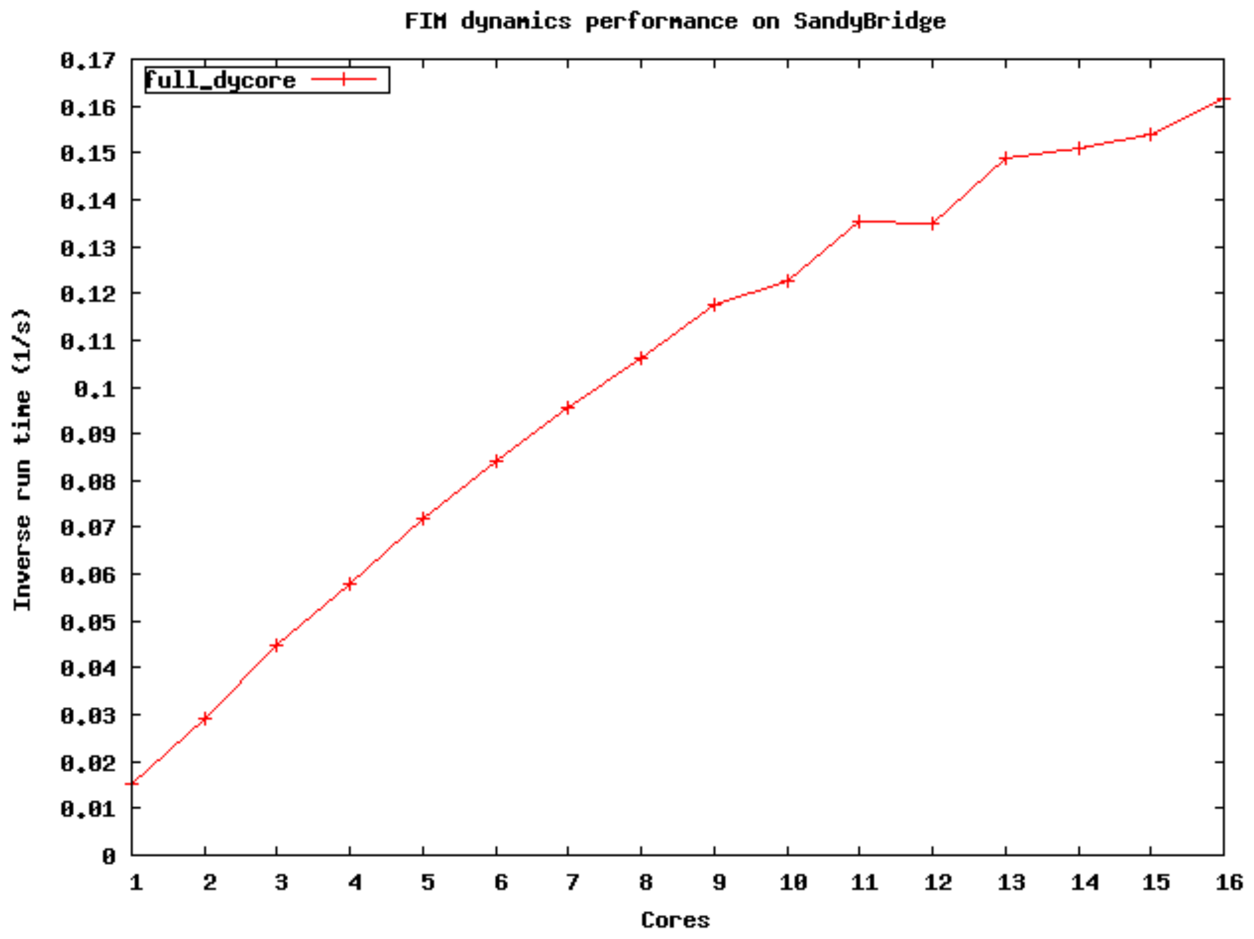
Validation

- 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



Full dycore scaling on a node



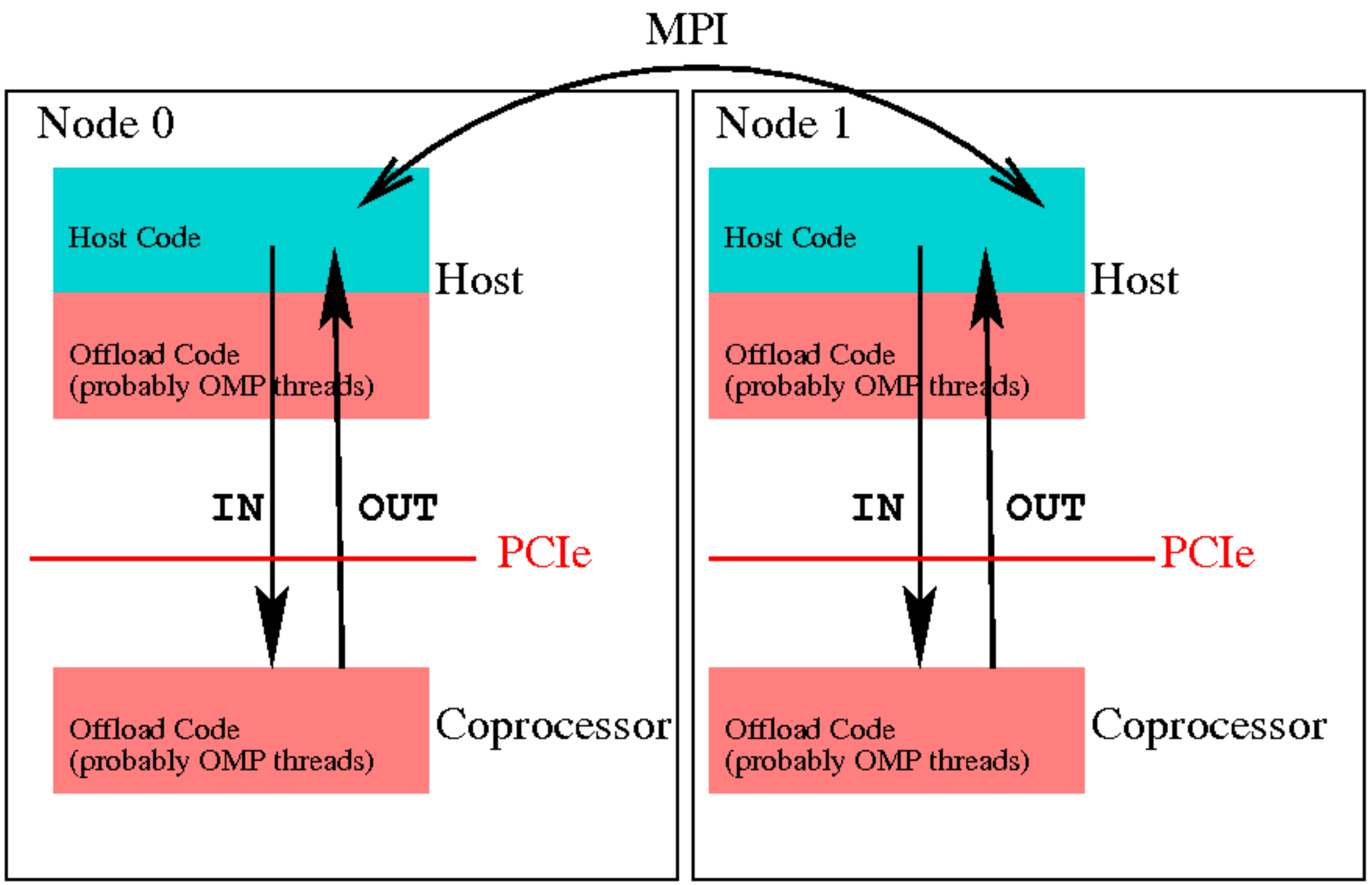
MIC Features (Public)

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

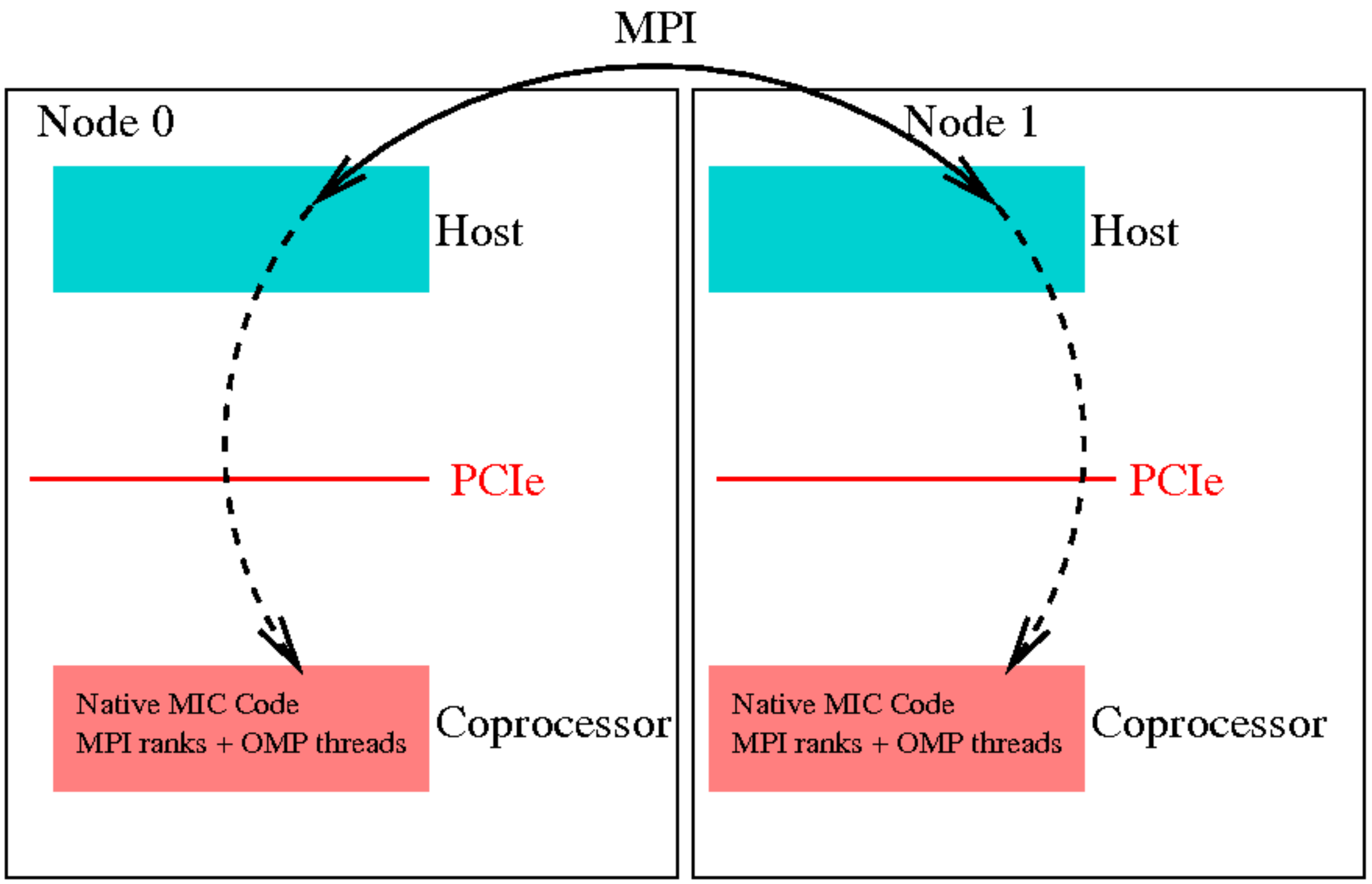
MIC programming modes

- 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



Native Mode


MPI
Node 0
Host
PCIe
Native MIC Code
MPI ranks + OMP threads
Coprocessor
Node 1
Host
PCIe
Native MIC Code
MPI ranks + OMP threads
Coprocessor

Mods required for MIC (native)

- No source code mods required
- Add –mmic compiler flag
- Some compiler flags not supported or generate very slow code

Mods required for MIC (offload)

- Declare data to live in offloaded region:

```
!DIR$ OPTIONS /OFFLOAD_ATTRIBUTE_TARGET=MIC
<data definitions>
!DIR$ END OPTIONS
```

- Declare subroutine to be offloadable:

```
!DIR$ ATTRIBUTES OFFLOAD:MIC :: <routine>
```

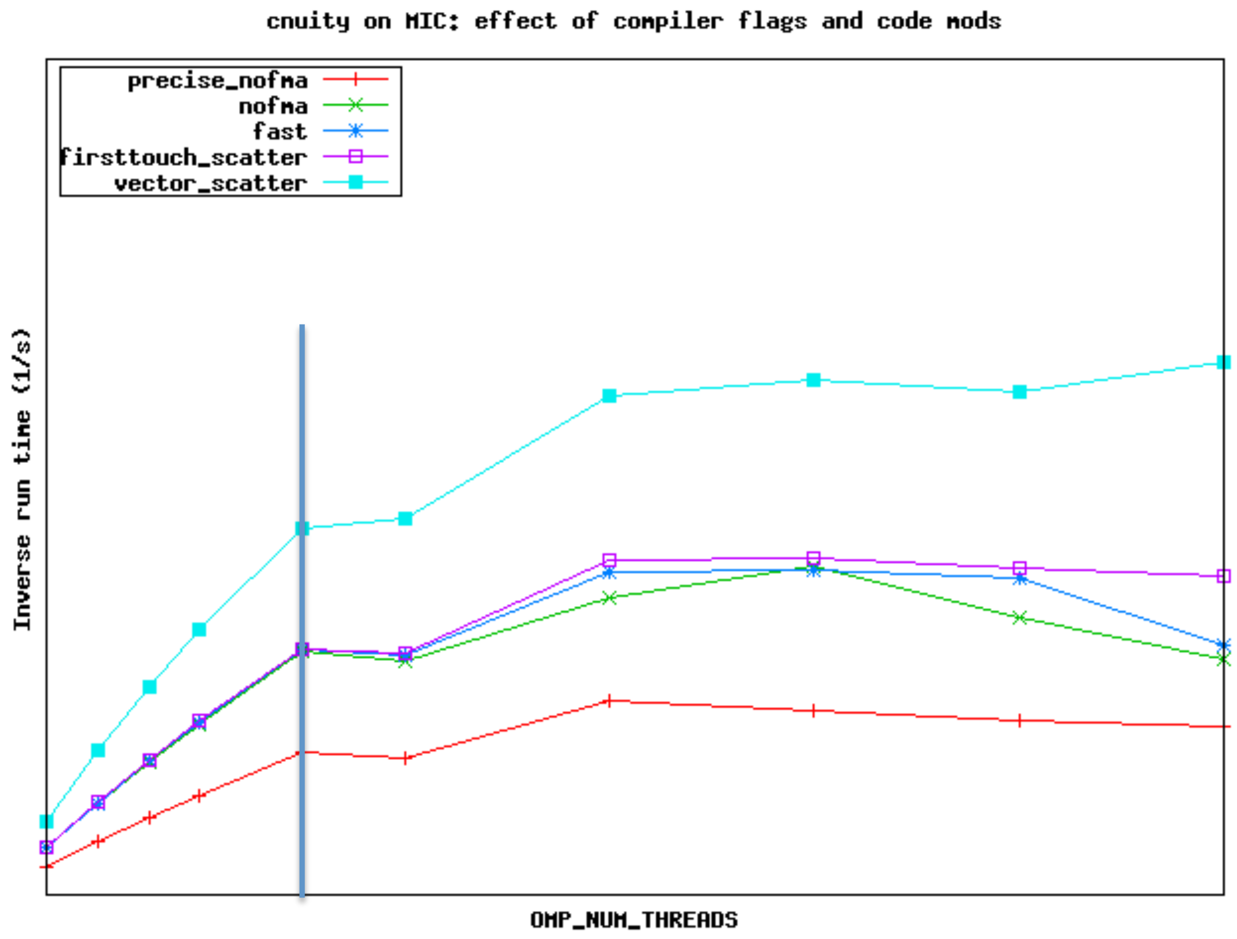
- Run region in offload mode:

```
!DIR$ OFFLOAD BEGIN TARGET(MIC:0) IN(…) OUT(…)
<code to run in offloaded region>
!DIR$ END OFFLOAD
```

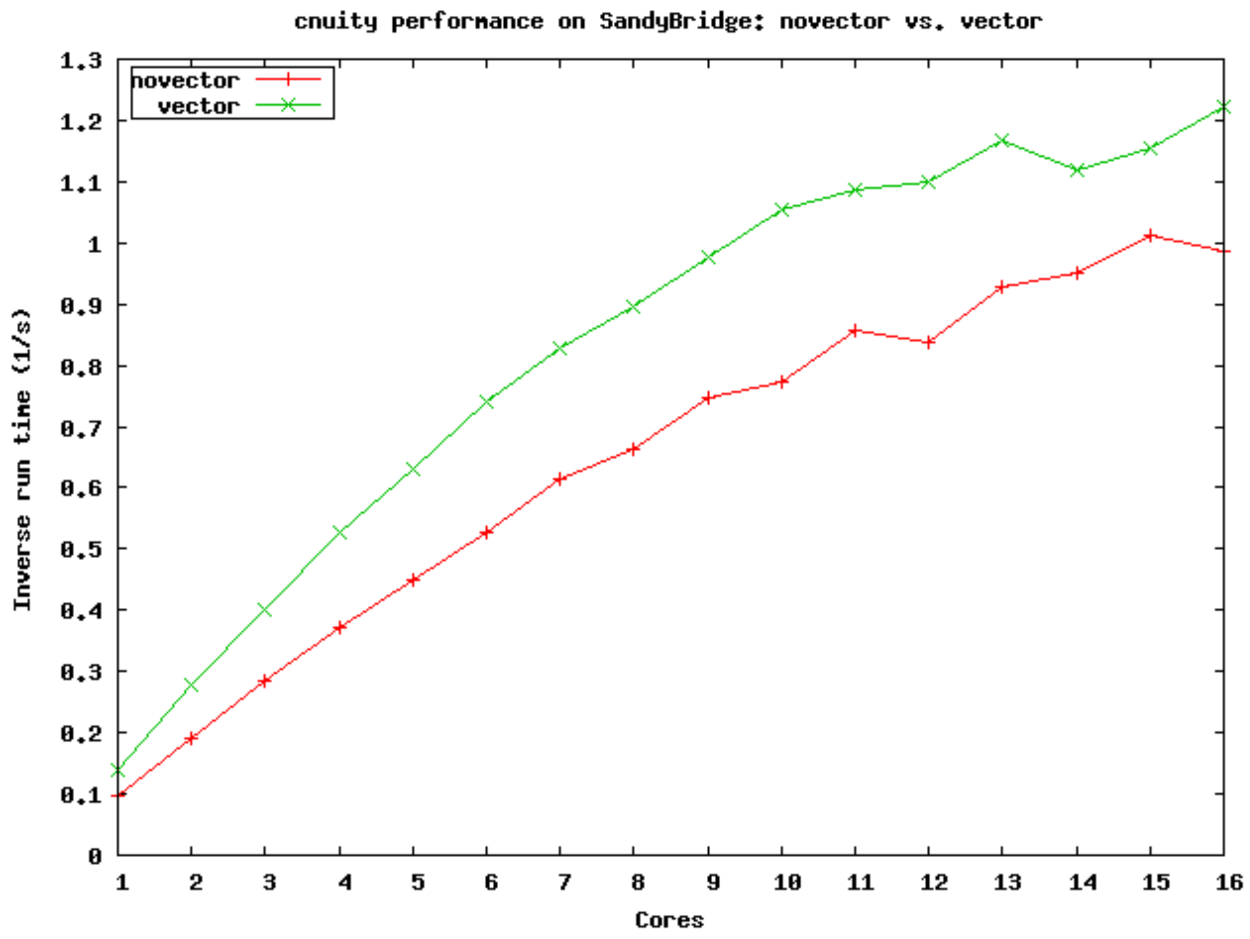
Current status of FIM on MIC

- 4 individual kernels from FIM dynamics extracted, running, and validated in native mode (all but 1 can be made bit-for-bit)
- 1 kernel running and validated in offload mode (bit-for-bit)
- Full FIM dynamics running and validated as a standalone kernel in native and offload mode
- Getting FIM running in native mode required zero mods to source code

Effect of mods to speedup cnuity kernel on MIC



Vectorization speedup on SandyBridge (X, Y axes differ from previous slide)



Vectorization example (orig)

```
!$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
      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 (512-bit) exceeds even SandyBridge (256-bit)
- 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 $KMP_AFFINITY
  - "compact", "scatter", "balanced"
- Experiment with $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?

- Move offload I/O to host
- 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

Contributors

- Mike Greenfield (Intel)
- Ruchira Sasanka (Intel)
- Karthik Raman (Intel)
- Craig Tierney (NOAA)