Introduction to Xeon Phi

IXPUG 14

Lars Koesterke

Acknowledgements

- Thanks/kudos to:
  - *Sponsor: National Science Foundation*
    - NSF Grant #OCI-1134872 Stampede Award, "Enabling, Enhancing, and Extending Petascale Computing for Science and Engineering"
    - NSF Grant #OCI-0926574 - "Topology-Aware MPI Collectives and Scheduling"
  - *Many, many Dell, Intel, and Mellanox engineers*
  - *Professor D.K. Panda at OSU (MVAPICH2)*

Thanks for Coming!

- In this tutorial, we will teach you what you need to know to start running your code on the Intel Xeon Phi Co-Processor.
- For the hands-on portion, we will user the Stampede system at TACC in Austin, Texas,USA, which has 6,880 Xeon Phi cards

Team Introductions

- Carlos Rosales
- Lars Koesterke
- Kent Milfeld
- Luke Wilson

Today's Session

- **Start End Session**
- 01:00 01:45 Programming the Phi, general (Lars)
- 01:45 02:30 Native Computing (Luke)
- 02:30 03:15 *Break* and Lab: Native (Lars, Luke)
- 03:15 04:00 Symmetric Computing (Carlos)
- 04:00 04:45 Offload Computing (Kent)
- 04:45 05:30 Lab: Symmetric/Offload -- take home optional (Carlos, Kent)

TACC introduction

- The Texas Advanced Computing Center is a leading US provider of cyberinfrastructure.
  - Stampede, the Xeon/Xeon Phi system you will hear about today, is our 10PF flagship system
  - Come to the BOF tomorrow to hear about Wrangler, our next Big Data System
- TACC has about 15,000 users around the world (primarily US), for whom we provide:
  - ~1 billion CPU hours per year.
  - Manage 5 billion files and ~40PB of data
  - 100

Stampede - High Level Overview

- Base Cluster (Dell/Intel/Mellanox):
  - Intel Sandy Bridge processors
  - Dell dual-socket nodes w/32GB RAM (2GB/core)
  - 6,400 nodes
  - 56 Gb/s Mellanox FDR InfiniBand interconnect
  - More than 100,000 cores, 2.2 PF peak performance
- Co-Processors:
  - Intel Xeon Phi "MIC" Many Integrated Core processors
  - Special release of "Knight's Corner" (61 cores)
  - All nodes have at least 1 card
    - 12 racks (480 nodes) have
  - 7+ PF peak performance
- *Max Total Concurrency:*
  - *exceeds 500,000 cores*
    1.8M threads

- ***Entered production operations on January 7, 2013***

Key aspects of acceleration

- We have lots of transistors… Moore's law is holding; this isn't necessarily the problem.
- We don't really need lower power per transistor, we need lower power per *operation*.
- How to do this?
  - nVidia GPU -AMD Fusion
  - FPGA -ARM Cores

Xeon Phi — MIC

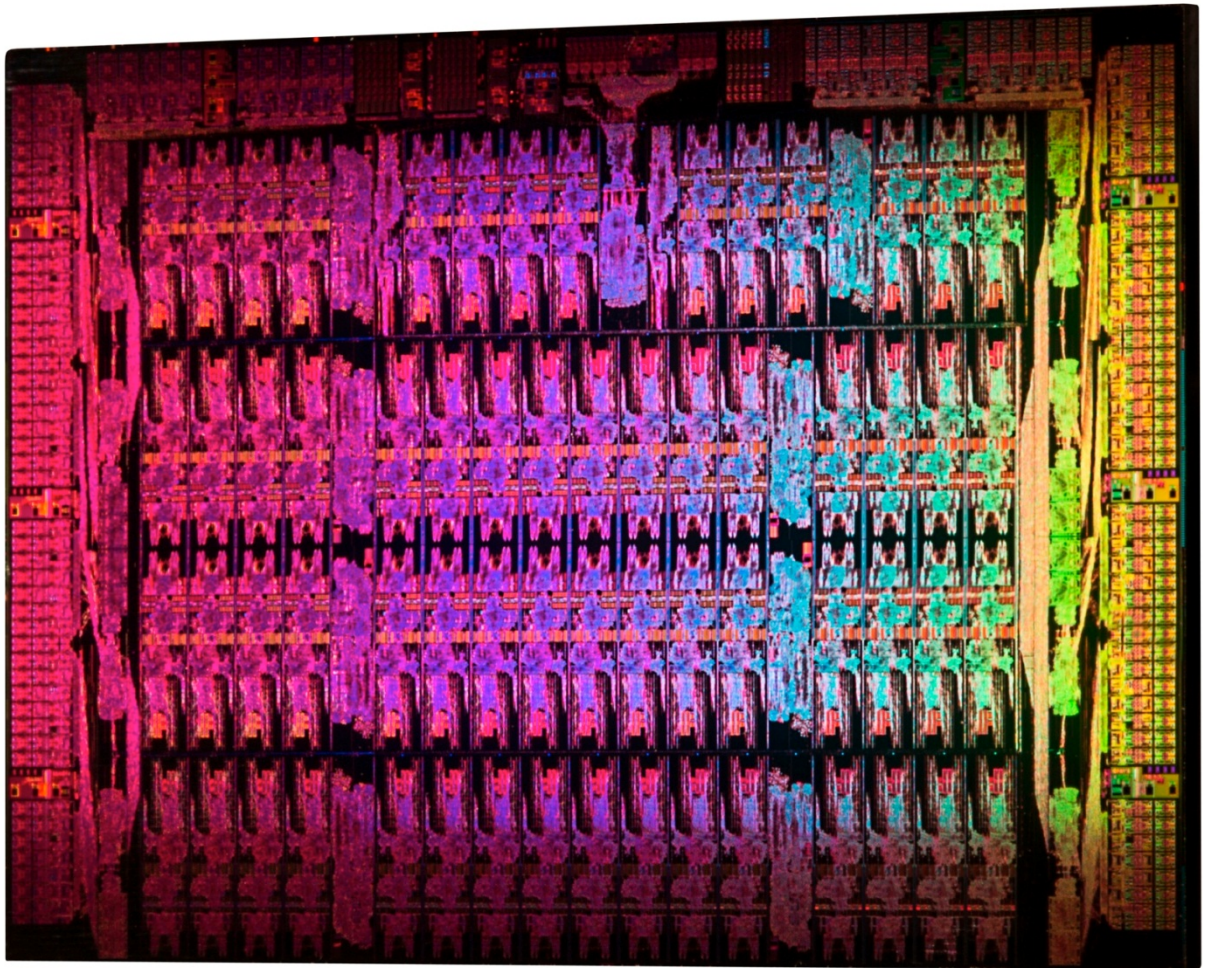
- Xeon Phi = first product of Intel's Many Integrated Core (MIC) architecture
- Co-processor
  - PCI Express card
  - Stripped down Linux operating system
- Dense, simplified processor
  - Many power-hungry operations removed
  - Wider vector unit
  - Wider hardware thread count
- Lots of names
  - Many Integrated Core architecture, aka MIC
  - Knights Corner (code name)
  - Intel Xeon Phi Co-processor SE10P (product name)

Xeon Phi — MIC

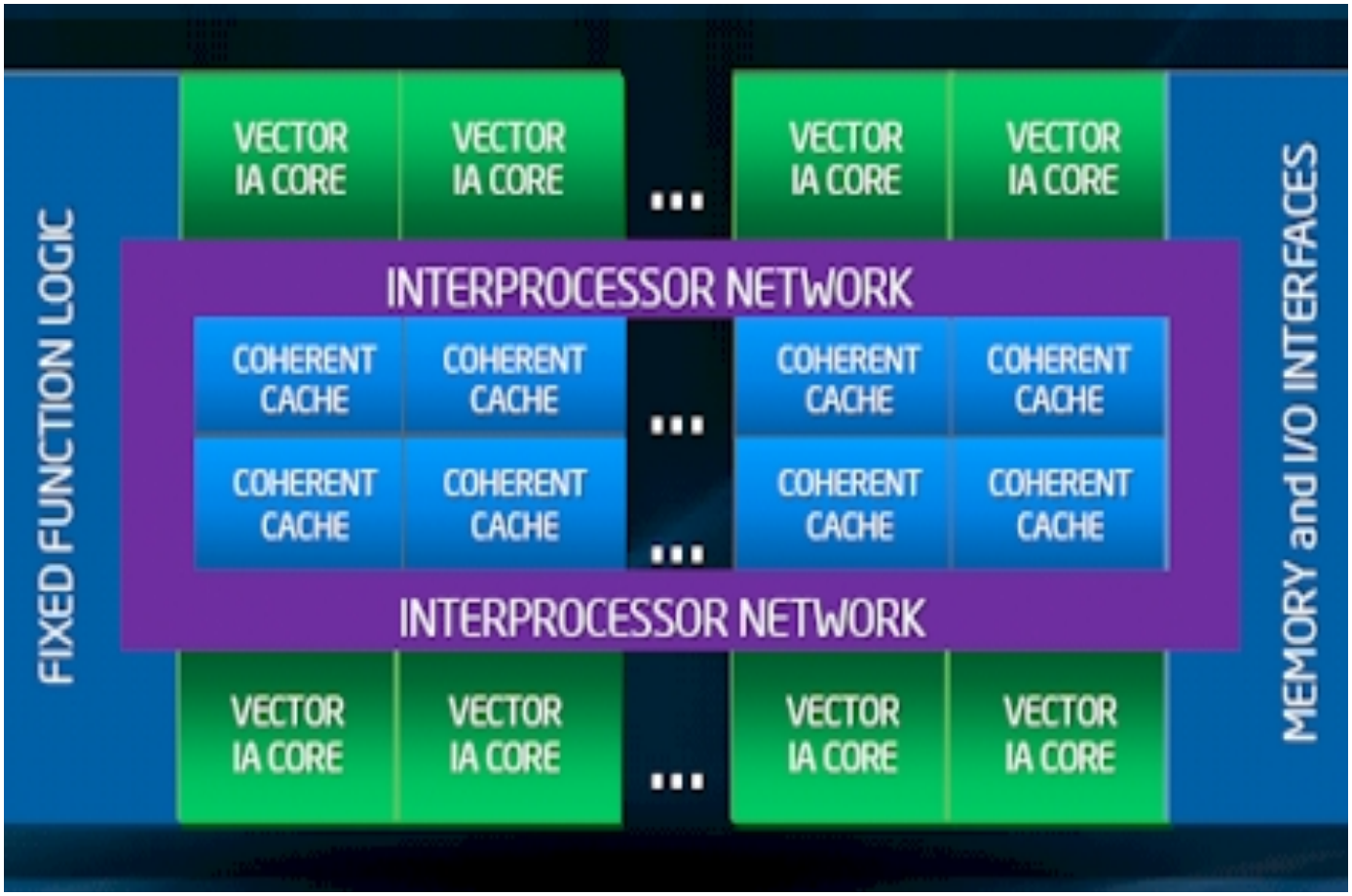
- Leverage x86 architecture (CPU with many cores)
  - x86 cores that are simpler, but allow for more compute throughput
- Leverage existing x86 programming models
- Dedicate much of the silicon to floating point ops
- Cache coherent
- Increase floating-point throughput
- Strip expensive features
  - out-of-order execution
  - branch prediction
- Widen SIMD registers for more throughput
- Fast (GDDR5) memory on card

Intel Xeon Phi Chip

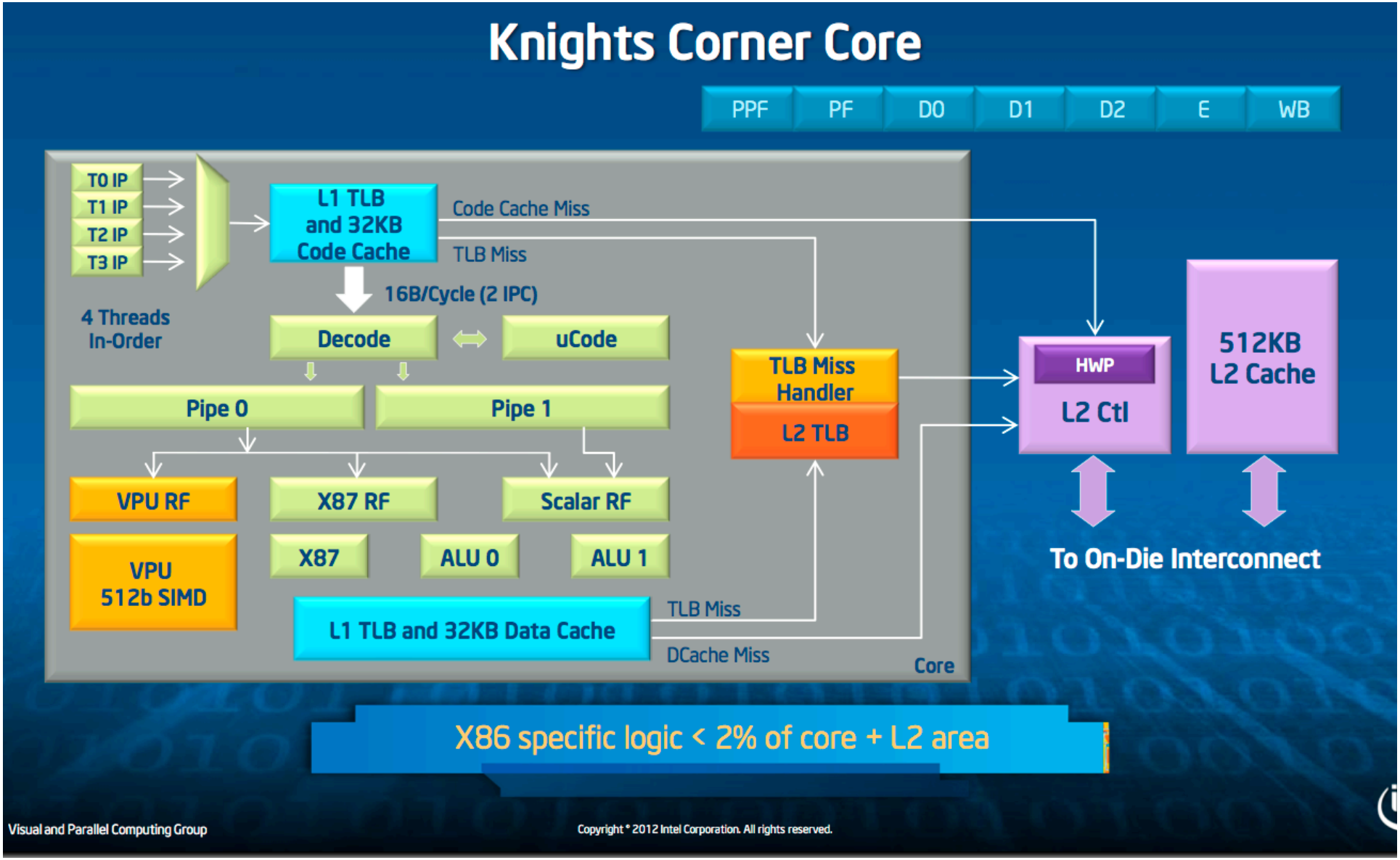
- 22 nm process
- Based on what Intel learned from
  - Larrabee
  - SCC
  - TeraFlops Research Chip



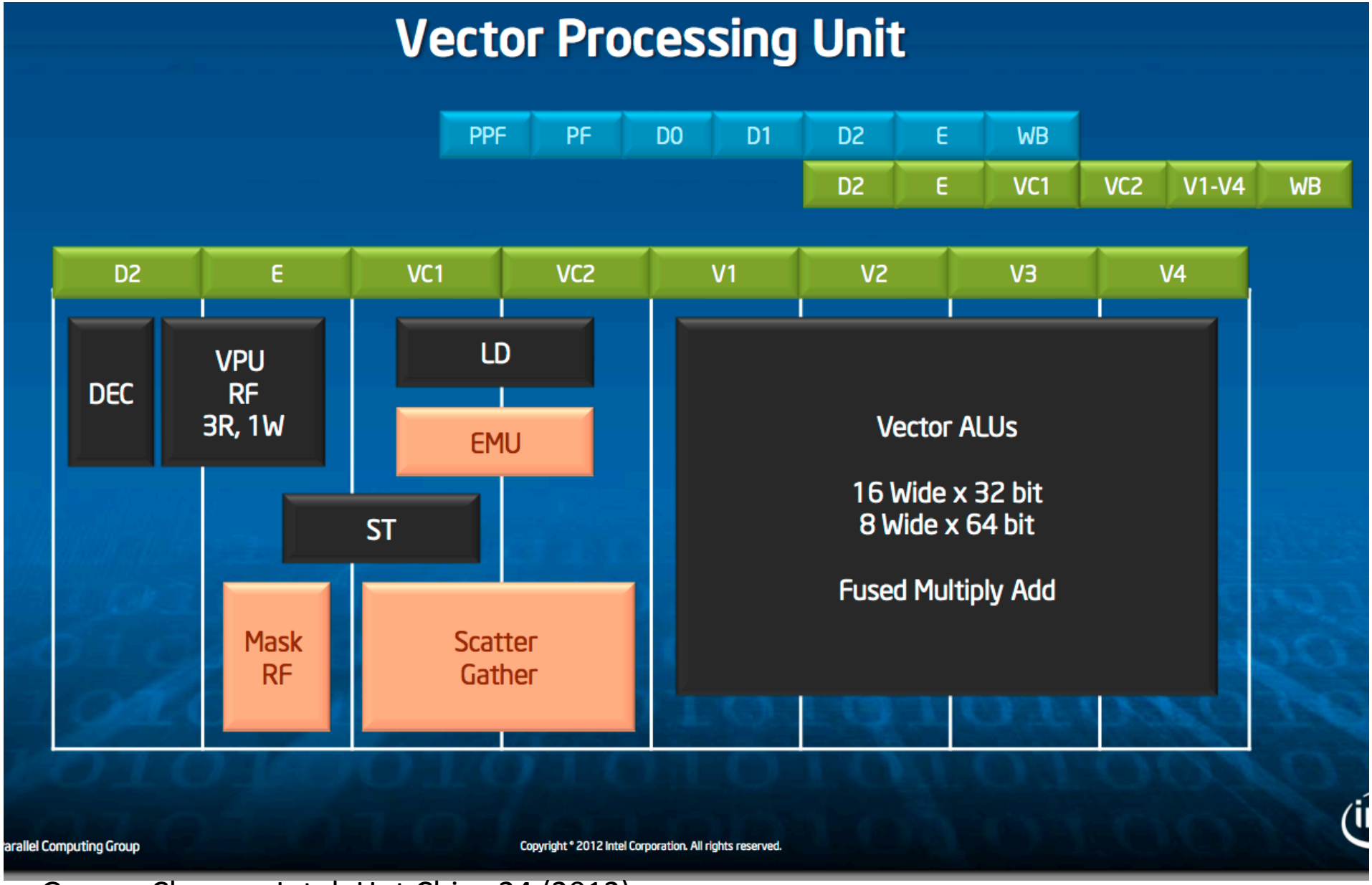
MIC Architecture



- Many cores on the die
- L1 and L2 cache
- Bidirectional ring network for L2
- Memory and PCIe connection



George Chrysos, Intel, Hot Chips 24 (2012):
http://www.slideshare.net/IntelXeon/under-the-armor-of-knights-corner-intel-mic-architecture-at-hotchips-2012



George Chrysos, Intel, Hot Chips 24 (2012):
http://www.slideshare.net/IntelXeon/under-the-armor-of-knights-corner-intel-mic-architecture-at-hotchips-2012

Speeds and Feeds (Intel Xeon Phi SE10P)

- Processor
  - ~1.1 GHz
  - 61 cores
  - 512-bit wide vector unit
  - 1.074 TF peak DP
- Data Cache
  - L1 32KB/core
  - L2 512KB/core, 30.5 MB/chip
- Memory
  - 8GB GDDR5 DRAM
  - 5.5 GT/s, 512-bit*
- PCIe
  - 5.0 GT/s, 16-bit

What we at TACC like about Phi

- Intel's MIC is based on x86 technology
  - x86 cores w/ caches and cache coherency
  - SIMD instruction set
- Programming for Phi is similar to programming for CPUs
  - Familiar languages: C/C++ and Fortran
  - Familiar parallel programming models: OpenMP & MPI
  - MPI on host and on the coprocessor
  - Any code can run on MIC, not just kernels
- Optimizing for Phi is similar to optimizing for CPUs
  - **"Optimize once, run anywhere"**
  - Our early Phi porting efforts for codes "in the field" have doubled performance on Sandy Bridge.

(Current) Xeon Phi Programming Models

- Traditional Cluster
  - Pure MPI and MPI+X
    - X may be OpenMP, TBB, Cilk+, OpenCL, …
- Native Phi
  - Use one Phi and run OpenMP or MPI programs directly
- MPI tasks on Host and Phi
  - Treat the Phi (mostly) like another host
    - Pure MPI and MPI+X (limited memory: using 'X' is almost mandatory)
- MPI on Host, Offload to Xeon Phi
  - Targeted offload through OpenMP extensions
  - Automatically offload some library routines with MKL

Traditional Cluster

- Stampede is 2+ PF of FDR-connected Xeon E5
  - High bandwidth: 56 Gb/s (sustaining >52 Gb/s)
  - Low-latency
    - ~1 µs on leaf switch
    - ~2.5 µs across the system
- Highly scalable for existing MPI codes
- IB multicast and collective offloads for improved collective performance

Native Execution

- Build for Phi with –mmic
- Execute on host (runtime will automatically detect an executable built for Phi)
- … or ssh to mic0 and run on the Phi
- Can safely use all 61 cores
  - But: I recommend use of 60 cores, i.e. 60, 120, 180, or 240 threads
  - Offload programs should **certainly** stay away from the 61st core since the offload daemon runs here

Symmetric MPI

- Host and Phi can operate symmetrically as MPI targets
  - High code reuse
  - MPI and hybrid MPI+X (X = OpenMP, Cilk+, TBB, pthreads)
- Careful to balance workload between big cores and little cores
- Careful to create locality between local host, local Phi, remote hosts, and remote Phis
- Take advantage of topology-aware MPI interface under development in MVAPICH2
  - NSF STCI project with OSU, TACC, and SDSC

Symmetric MPI

- Typical 1-2 GB per task on the host
- Targeting 1-10 MPI tasks per Phi on Stampede
  - With 6+ threads per MPI task

MPI with Offload to Phi

- Existing codes using accelerators have already identified regions where offload works well
- Porting these to OpenMP offload should be straightforward
- Automatic offload where MKL kernel routines can be used
  - xGEMM, etc.

Will My Code Run on Xeon Phi?

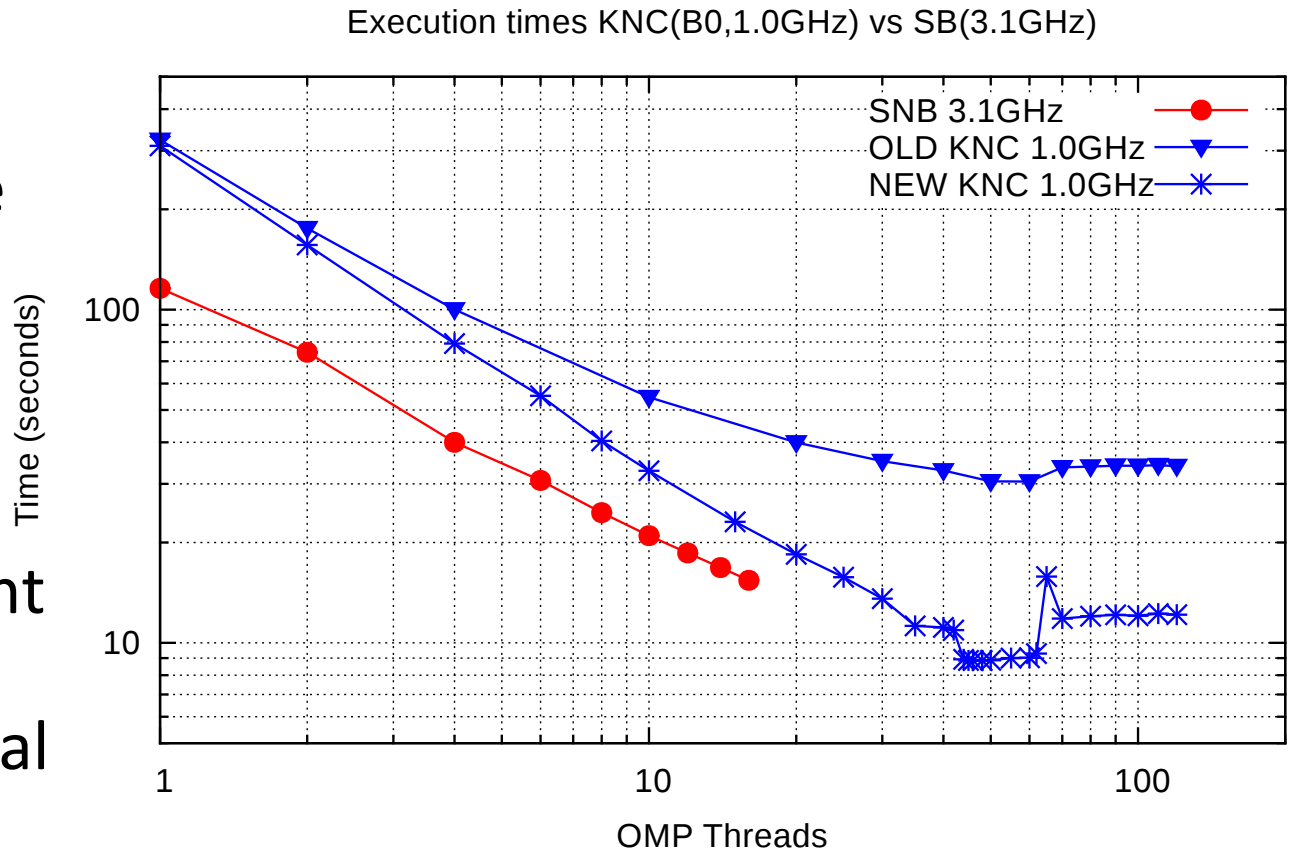
- Yes
- … but that's the wrong question
  - Will your code run *best* on Phi?, or
  - Will you get great Phi performance without additional work? (The answer is most likely **NO**)

Early Phi Programming Experiences at TACC

- Codes port easily
  - Minutes to days depending mostly on library dependencies
- Performance can require real work
  - While the software environment continues to evolve
  - Getting codes to run *at all* is almost too easy; really need to put in the effort to get what you expect
- Scalability is pretty good
  - Multiple threads per core is really important
  - Getting your code to vectorize is really important

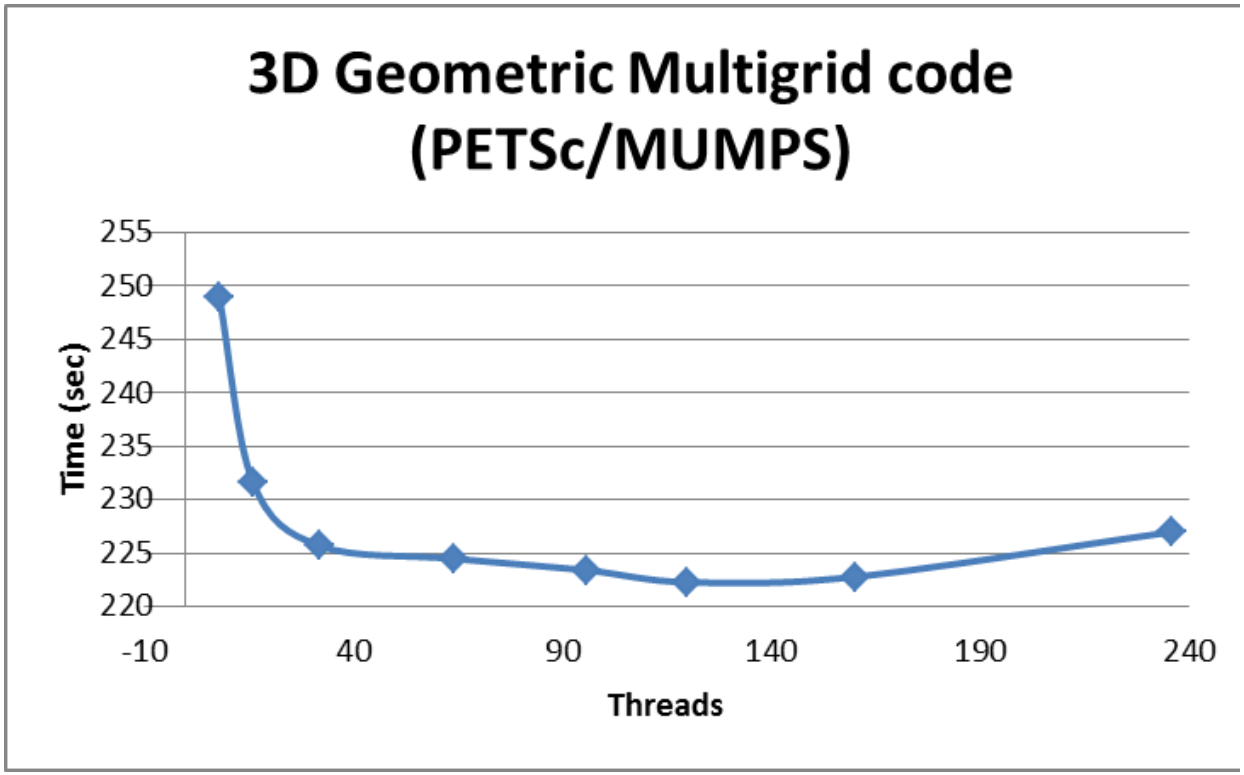
LBM Example

- Lattice Boltzmann Method CFD code
  - Carlos Rosales, TACC
  - MPI code with OpenMP
- Finding all the right routines to parallelize is critical



PETSc/MUMPS with AO

- Hydrostatic ice sheet modeling
- MUMPS solver(called through PETSC)
- BLAS calls automatically offloaded behind the scenes



Phi evolution

- A few details of the next generation of Xeon Phi are announced.
  - "Knight's Landing"
  - 14nm process
  - Available as a **stand alone** processor
    - Think about that as you consider programming model
  - On-package high bandwidth memory
  - AVX-512 instruction set (now publicly available).

Summary

- MIC experiences
  - Early scaling looks good; application porting is fairly straight forward since it can run native C/C ++, and Fortran code
  - Some optimization work is still required to get at all the available raw performance for a wide variety of applications; but working well for some apps
  - **vectorization** on these large many-core devices is key
  - **affinitization** can have a strong impact (positive/negative) on performance
  - **algorithmic threading performance** is also key; if the kernel of interest does not have high scaling efficiency on a standard x86_64 processor (8-16 cores), it will not scale on many-core
  - ***MIC optimization efforts also yield fruit on normal Xeon*** (in fact, you may want to optimize there first.
- Questions?
- Then let's dive in deeper