Cornell University
Center for Advanced Computing

Parallel Programming on Ranger and Stampede

Steve Lantz
Senior Research Associate
Cornell CAC

Parallel Computing at TACC: Ranger to Stampede Transition
December 11, 2012

What is Stampede?

- NSF-funded XSEDE project at TACC
- New HPC system with two main components
  - 2+ petaflop/s Intel Xeon E5 based Dell cluster to be the new workhorse for the NSF open science community (>3x Ranger)
  - 7+ additional petaflop/s of Intel Xeon Phi™ SE10P coprocessors to change the power/performance curves of supercomputing
- Complete ecosystem for advanced computing
  - HPC cluster, storage, interconnect
  - Visualization subsystem (144 high-end GPUs)
  - Large memory support (16 1TB nodes)
  - Global work file system, archive, other data systems
  - People, training, documentation to support computational science

Construction Under Way at TACC, May 2012

Left: water chiller plant; right: addition to main facility

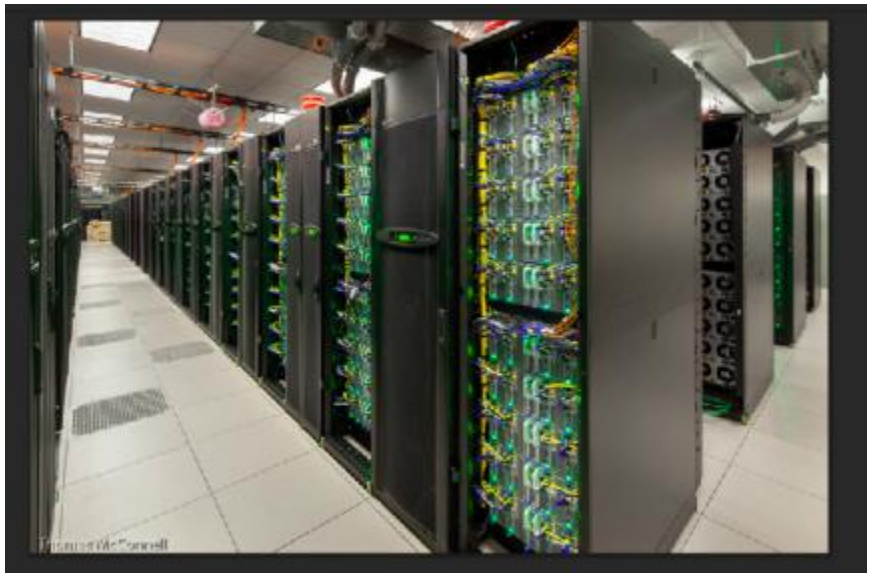
Stampede Specs from News Releases

- 6400 Dell C8220X nodes in initial system
  - 16 Xeon E5 “Sandy Bridge” cores per node, 102400 total
  - 32GB memory per node, 200TB total
- At least 6400 Xeon Phi™ SE10P coprocessor cards
  - 61 core, 4 hardware threads per core
  - 8 GB additional memory per card
- 14+ PB storage
  - Lustre parallel filesystem
  - 4864 3TB drives

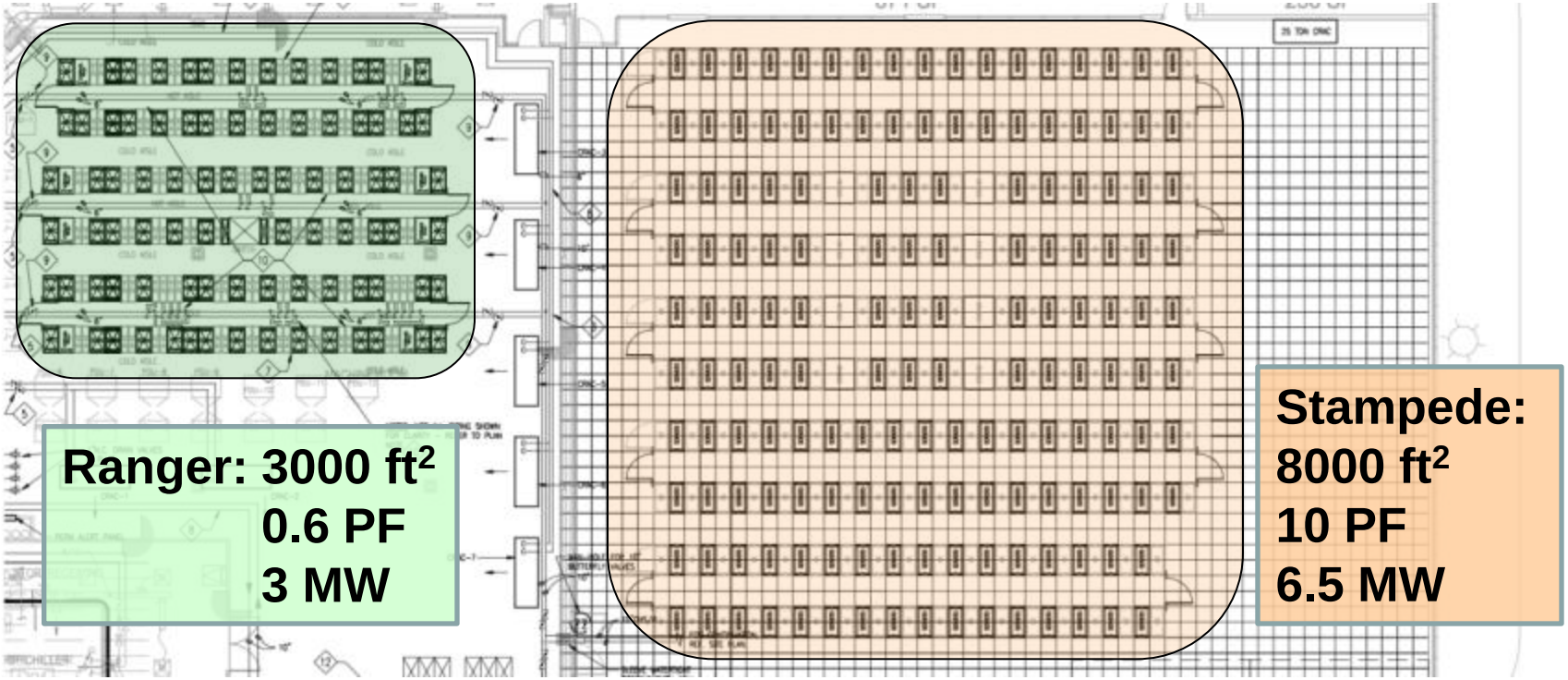

Photo by TACC, June 2012

Stampede Specs from News Releases

- 56Gb/s InfiniBand, fat-tree interconnect
  - ~75 miles of cables
  - < 1.2 µS latency
  - Compare Lonestar: 32Gb/s (eff.)
- 15PF+ after upgrade in 2015
- Nearly 200 racks
- SIX MEGAWATTS total power
  - Thermal energy storage cuts costs
- Datacenter expansion of 10,000 sq. ft.



Stampede Footprint vs. Ranger

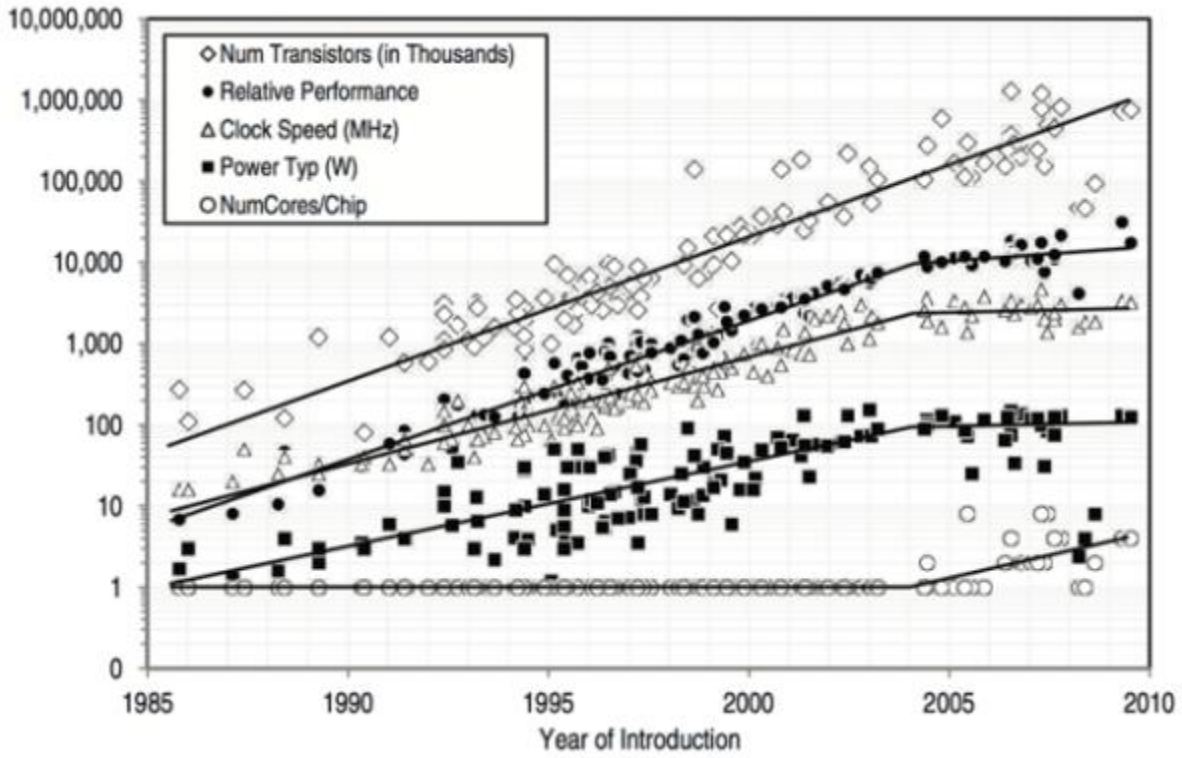


- Capabilities are 17x; footprint is 2.7x; power draw is 2.1x

How Does Stampede Reach Petaflop/s?

- Hardware trend since around 2004: processors gain more *cores* (execution engines) rather than greater clock speed
  - IBM POWER4 (2001) became the first chip with 2 cores, 1.1–1.9 GHz; meanwhile, Intel's single-core Pentium 4 was a bust at >3.8 GHz
  - Top server and workstation chips in 2012 (Intel Xeon, AMD Opteron) now have 4, 8, even 16 cores, running at 1.6–3.2 GHz
- Does it mean Moore's Law is dead? No!
  - Transistor densities are still doubling every 2 years
  - Clock rates have stalled at < 4 GHz due to power consumption
  - Only way to increase flop/s/watt is through greater on-die parallelism…

CPU Speed and Complexity Trends



Committee on Sustaining Growth in Computing Performance, National Research Council.
"What Is Computer Performance?"
In *The Future of Computing Performance: Game Over or Next Level?*
Washington, DC: The National Academies Press, 2011.

Implications for Petaflop/s Machines

- Only way to increase flop/s/watt is through greater on-die parallelism!
- These trends hold true for non-CPU devices too
  - Processors for mobile devices, e.g.
- *If 1 chip holds 10s of the best cores, why not 100s of weaker ones?*
  - Around 2007–8, "Cell" chips had 1 main and 8 synergistic processors
  - But then something else was recognized…

GPUs: Highly Parallel Hardware is in PCs Already

- High-end graphics processing units (GPUs) contain 100 or 1000s of thread processors and enough RAM to rival CPUs in compute capability
- GPUs have been further tailored for HPC
- Stampede example: NVIDIA Tesla K20
  - 2496 CUDA cores @ 732 MHz
  - 5GB dedicated memory
  - 1.17 Tflop/s peak DP rate
  - 225W power consumption

Tesla GPU

- Initially there were hardware obstacles to using GPUs for general calculations, but these have been overcome
  - ECC memory, double precision, IEEE-compliant arithmetic are built in
  - What about software…?

General Purpose Computing on GPUs (GPGPU)

- NVIDIA CUDA (2006) has been the forerunner in this area
  - SDK + API that permits programmers to use the C language to code algorithms for execution on NVIDIA GPUs (must be compiled with nvcc)
  - Stream processing: GPU executes a code "kernel" on a stream of inputs
  - Works well if kernel is multithreaded, vectorized (SIMD), pipelined
- OpenCL (2008) is a more recent, open standard originated by Apple
  - C99-based language + API that enables data-parallel computation on GPUs as well as CPUs (e.g., ARM)
- *Nontrivial (re)coding may be needed, based on a specialized API*
  - Good performance depends on very specific tuning to cache sizes, etc.
  - Hard to keep thread processors busy over slow PCIe interconnect
  - Resulting code is far less portable due to the API and special tuning

The Intel Approach: MIC

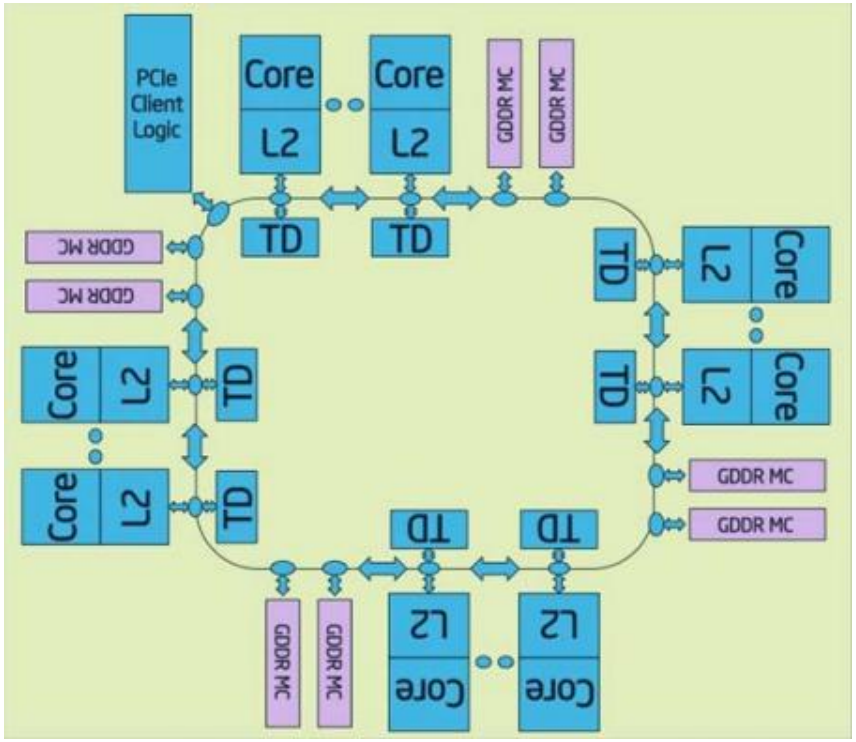
- MIC = Many Integrated Cores = a "coprocessor" on a PCIe card that features >50 cores: released as Xeon Phi™, used in Stampede
  - Represents Intel's response to GPGPU, especially NVIDIA's CUDA
  - Incorporates lessons learned from several internal development efforts: "Larrabee", 80-core Terascale chip, Single-Chip Cloud (SCC)
  - Answers the question: if 8 modern Xeon cores fit on a die, how many early Pentiums would fit?
- Addresses the API problem: standard x86 instructions are supported
  - Includes 64-bit addressing
  - Other recent x86 extensions may not be available
  - Special instructions are added for an extra-wide (512-bit) vector register
- MIC supports general-purpose executables built using familiar Intel compilers, libraries, and analysis tools

Coprocessor vs Accelerator

	Coprocessor	Accelerator
Architecture	x86, multi-core, vector	Streaming processors
	Coherent cache	Shared memory, caches
Programming Model	Extensions to C/Fortran	CUDA, OpenCL
Threading	OpenMP, explicit threading	Hardware threads, little user control
	May use communication, synchronization	Mostly independent
MPI	Host ↔ Host Host ↔ MIC	Host ↔ Host
Programming Constructs	Serial, Thread, Vector	Stream

MIC Architecture

- SE10P is first production version used in Stampede
  - Chip, memory on PCIe card
  - 61 cores, each containing:
    - 64 KB L1 cache
    - 512 KB L2 cache
    - 512 byte vector unit
    - 4 hardware threads
    - In-order instruction pipeline
  - 31.5 MB total coherent L2 cache, connected by ring bus
  - 8 GB GDDR5 memory
    - Very fast, 352 GB/s vs 50 GB/s/socket for E5



Courtesy Intel

MIC vs. CPU

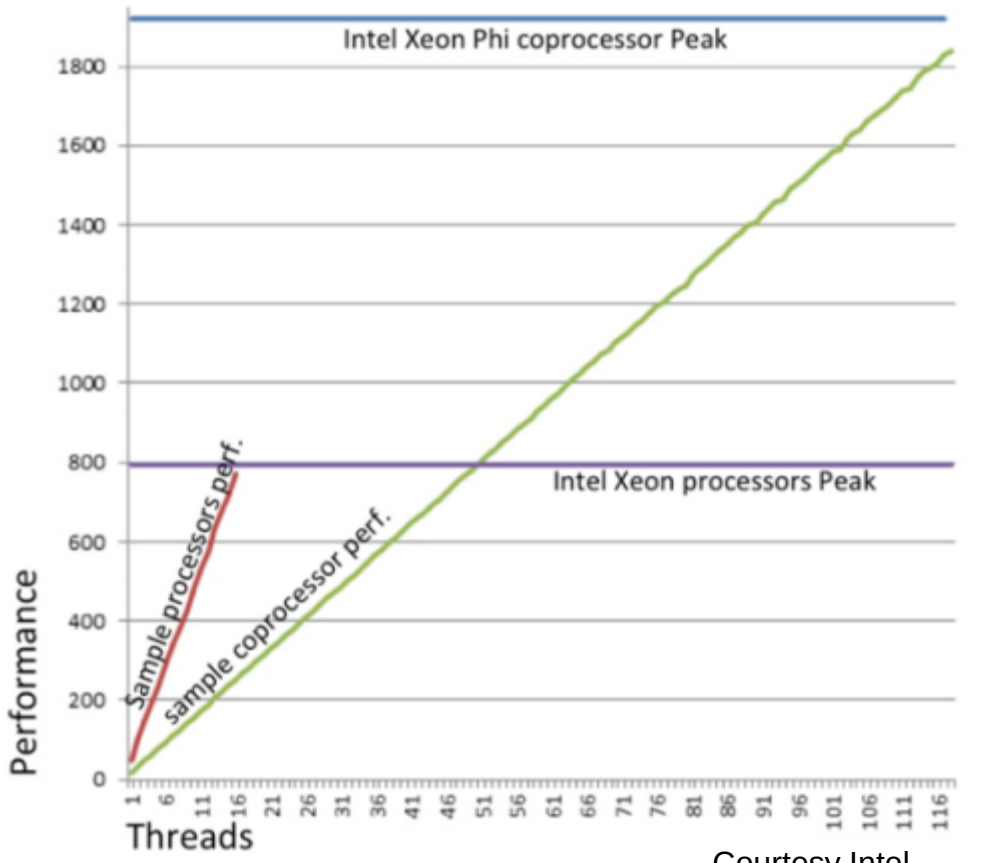
	MIC (SE10P)	CPU (E5)	MIC is…
• Number of cores	61	8	much higher
• Clock Speed (GHz)	1.01	2.7	lower
• SIMD width (bit)	512	256	higher
• DP GFLOPS/core	16+	21+	lower
• HW threads/core	4	1*	higher

- CPUs designed for all workloads, high single-thread performance
- MIC also general purpose, though optimized for number crunching
  - Focus on high aggregate throughput via lots of weaker threads

Two Types of CPU/MIC Parallelism

- Threading (work-level parallelism)
  - OpenMP, Cilk Plus, TBB, Pthreads, etc
  - It's all about sharing work and scheduling
- Vectorization (data-level parallelism)
  - "Lock step" Instruction Level Parallelization (SIMD)
  - Requires management of synchronized instruction execution
  - It's all about finding simultaneous operations
- To fully utilize MIC, both types of parallelism need to be identified and exploited
  - Need at 2-4 threads to keep a MIC core busy (in-order execution stalls)
  - Vectorized loops gain 8x performance on MIC!
  - Important for CPUs as well: gain of 4x on Sandy Bridge

Threading

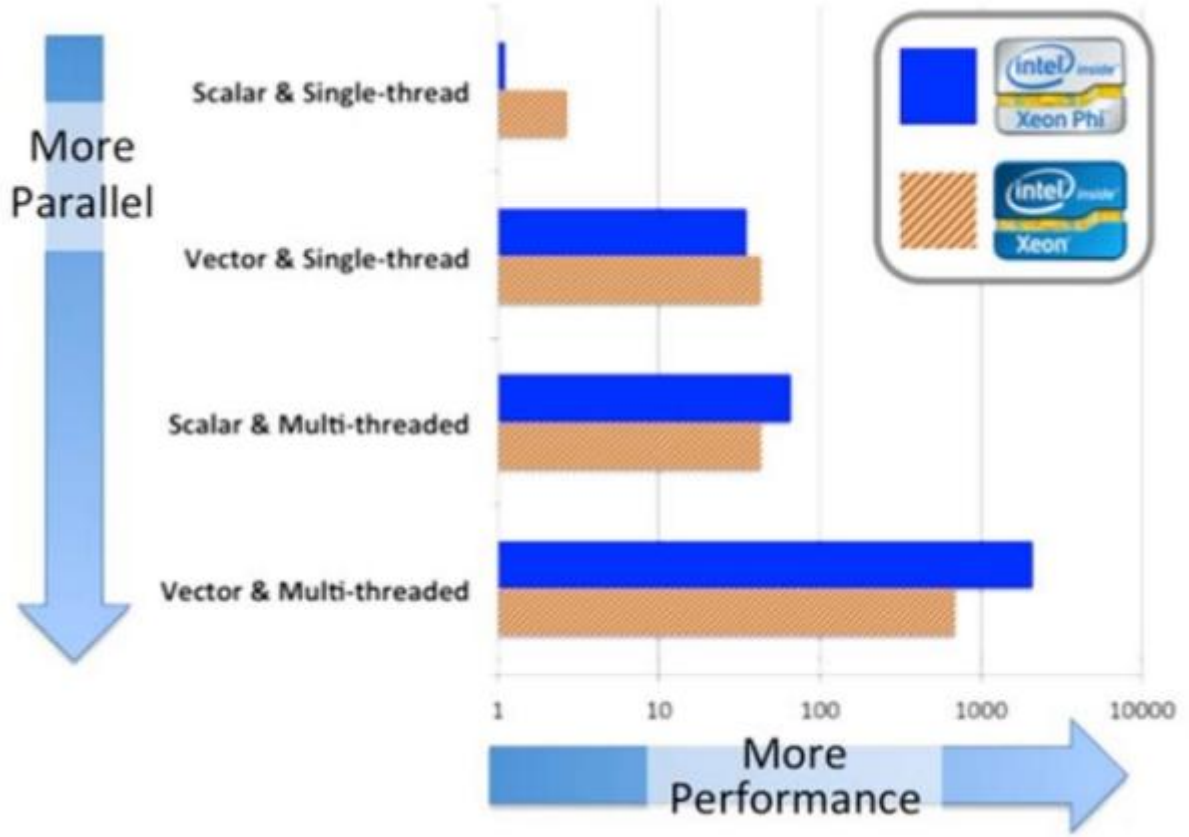


Courtesy Intel

Vectorization

- Instruction set: Streaming SIMD instructions (SSE, AVX)
- Single Instruction Multiple Data (SIMD)
- Wide vector registers hold multiple SP, DP or integer values
- One operation produces, 2, 4, 8, results or more.
- Compilers are good at vectorizing inner loops automatically as an optimization
  - … but they need help
  - Make sure each iteration is independent
  - Align data to “good” boundaries of registers and cache
- Xeon Phi vector width twice as wide as E5 CPU
  - Twice as many calculations vector operation

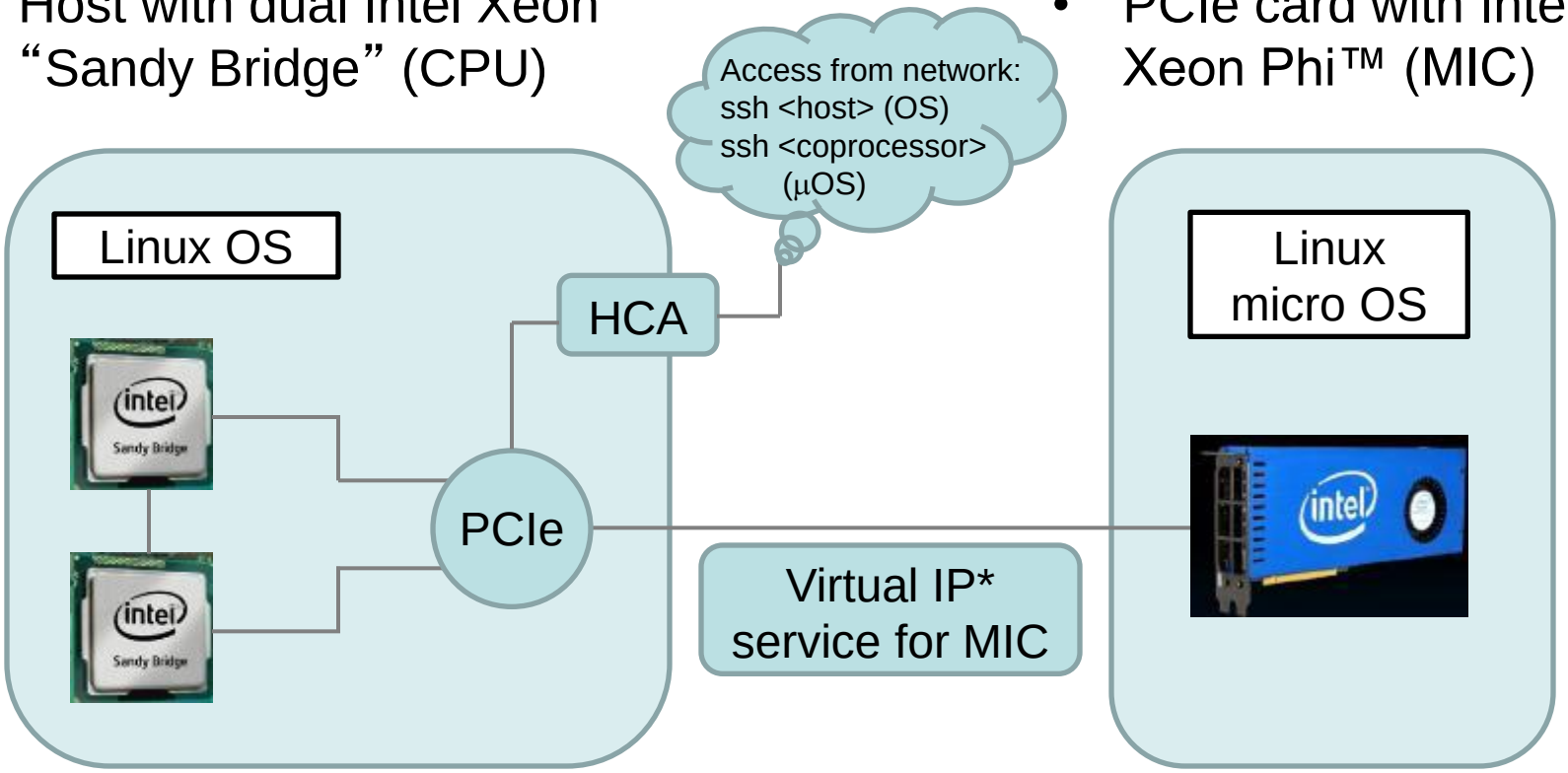
Parallelism and Performance on MIC and CPU



Courtesy Intel

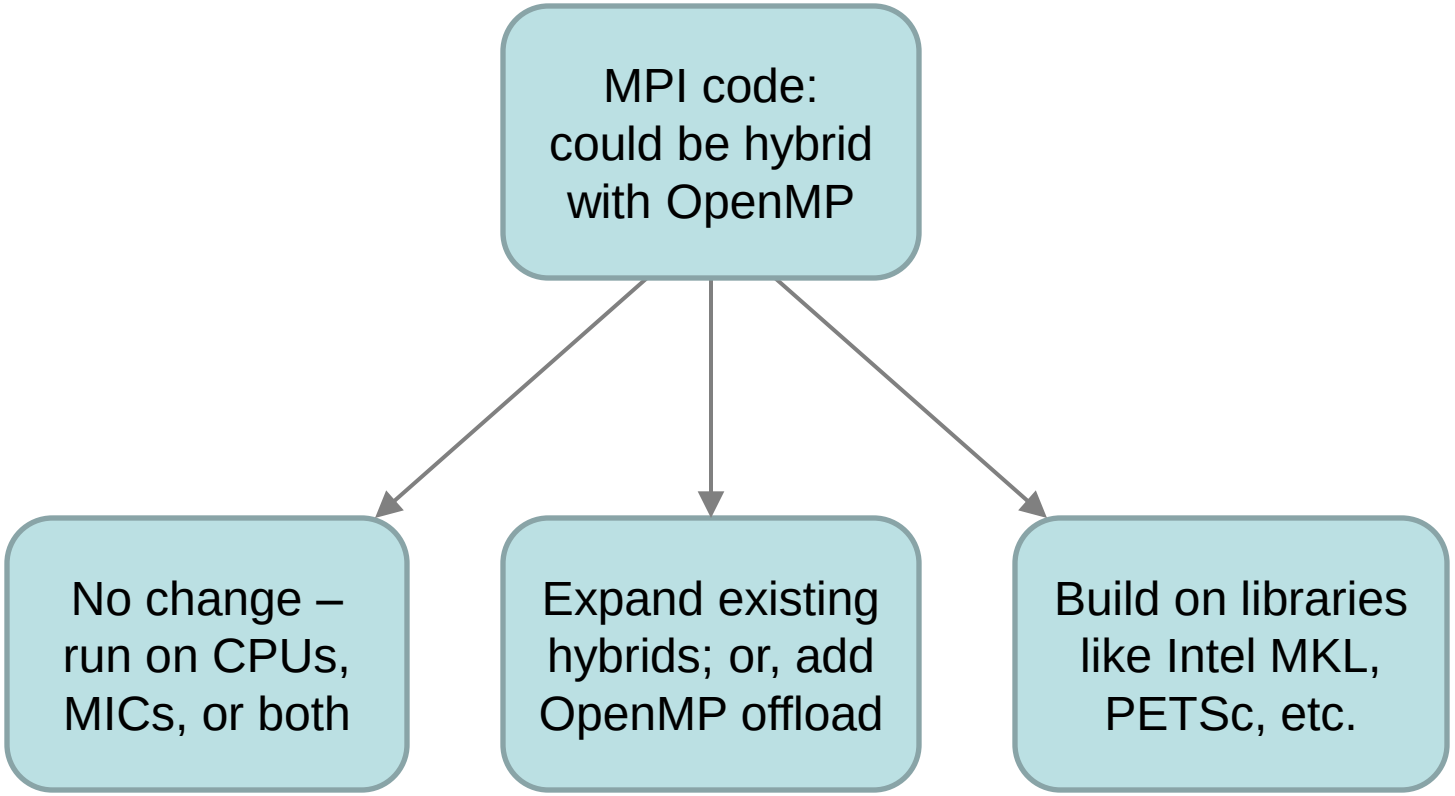
Typical Configuration of a Future Stampede Node

- Host with dual Intel Xeon "Sandy Bridge" (CPU)
- PCIe card with Intel Xeon Phi™ (MIC)



* can't do this with a Lonestar GPU node, e.g., which is otherwise similar

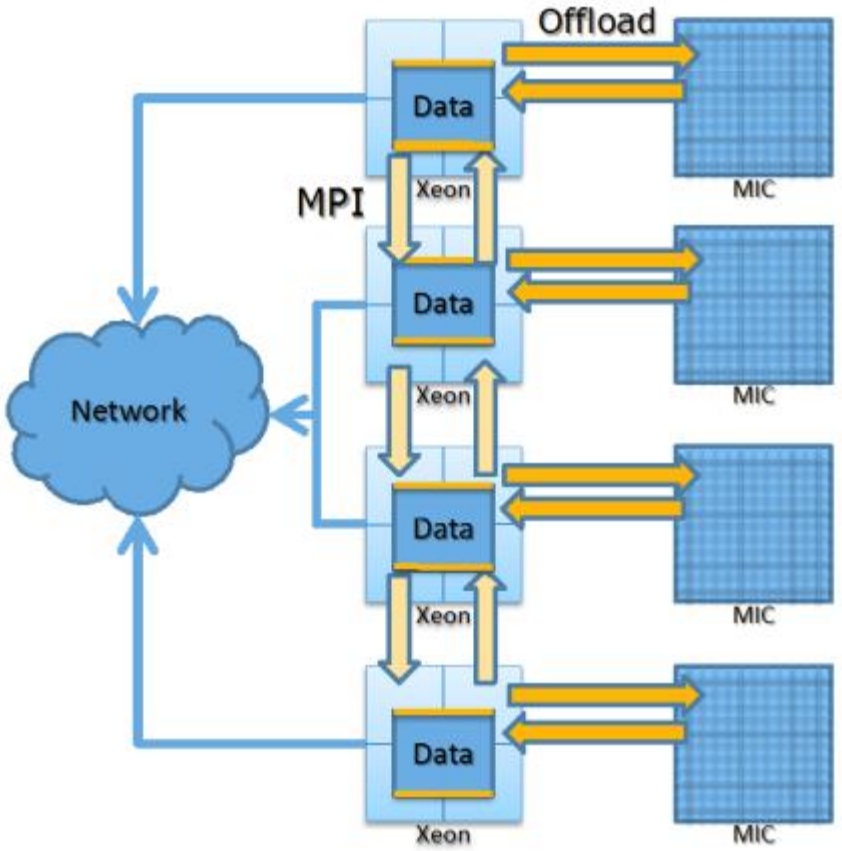
Strategies for HPC Codes



Programming Models for Stampede – 1

Offload Execution

- Directives indicate data and functions to send from CPU to MIC for execution
- Unified source code
- Code modifications required
- Compile once with offload flags
  - Single executable includes instructions for MIC and CPU
- Run in parallel using MPI and/or scripting, if desired

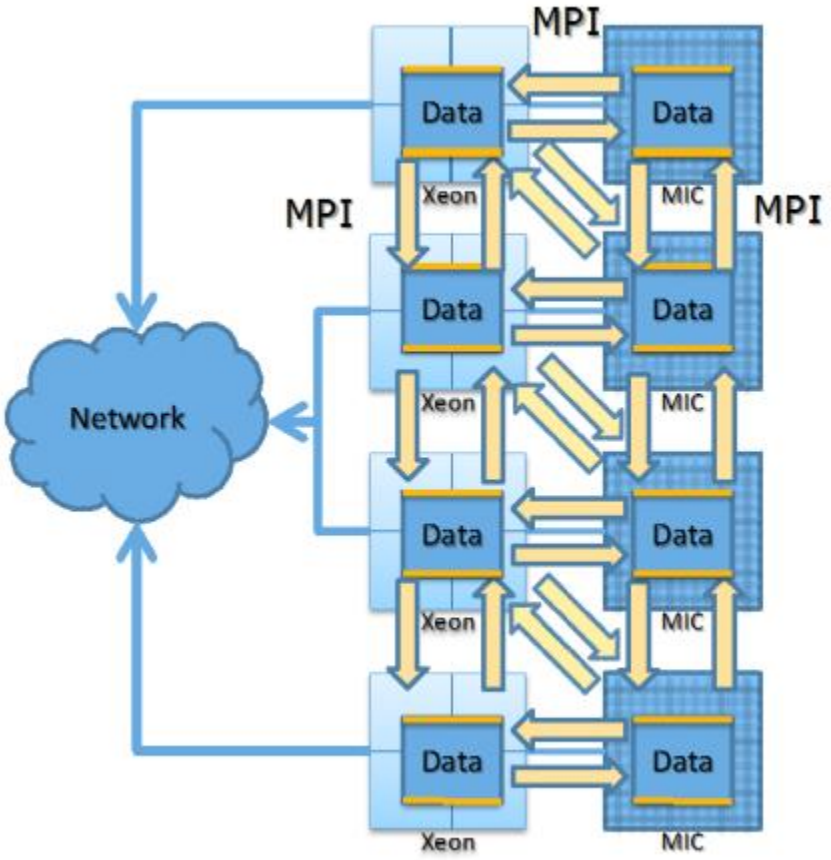


Courtesy Scott McMillan, Intel

Programming Models for Stampede – 2

"Symmetric" Execution

- Message passing (MPI) on CPUs and MICs alike
- Unified source code
- Code modifications optional
  - Assign different work to CPUs vs. MICs
  - Multithread with OpenMP for CPUs, MICs, or both
- Compile twice, 2 executables
  - One for MIC, one for host
- Run in parallel using MPI



Courtesy Scott McMillan, Intel

Pros and Cons of MIC Programming Models

- Offload engine: MIC acts as coprocessor for the host
  - *Pros:* distinct hardware gets distinct role; programmable via simple calls to a library such as MKL, or via directives (we'll go into depth on this)
  - *Cons:* **PCIe** is the path for most work; difficult to retain data on card
- "Symmetric" #1: Everything is just an MPI core
  - *Pros:* MPI works for all cores (though 1 MIC core < 1 server core)
  - *Cons:* **memory** may be insufficient to support a μOS plus lots of data; fails to take good advantage of shared memory; PCIe is a bottleneck
- "Symmetric" #2: Both MIC and host are just SMPs
  - *Pros:* MPI/OpenMP works for both host and MIC; more efficient use of limited PCIe bandwidth and limited MIC memory
  - *Cons:* **hybrid** programming is already tough on homogeneous SMPs; not much experience with OpenMP-based hybrids scaling to 50+ cores

Using Compiler Directives to Offload Work

- OpenMP's directives provide a natural model
  - 2010: OpenMP working group starts to consider **accelerator** extensions
  - Related efforts are launched to target specific types of accelerators…
- LEO, Language Extensions for Offload
  - Intel moves forward to support processors and **coprocessors**, initially
- OpenACC
  - PGI moves forward to support GPUs, initially
- Will OpenMP 4.0 produce a compromise among all the above?
  - Clearly desirable, but it's difficult
  - Other devices exist: network controllers, antenna A/D, cameras…
  - Exactly what falls in the "accelerator" class? How diverse is it?
  - Are "coprocessors" a distinct class?

OpenMP Offload Constructs: Base Program

```
#include <omp.h>
#define N 10000

void foo(double *, double *, double *, int );
int main(){
    int i; double a[N], b[N], c[N];
    for(i=0;i<N;i++){ a[i]=i; b[i]=N-1-i;}

    ...

    foo(a,b,c,N);
}

void foo(double *a, double *b, double *c, int n){
    int i;

    for(i=0;i<n;i++) { c[i]=a[i]*2.0e0 + b[i]; } }
```

- Objective: offload foo to a device
- Use OpenMP to do the offload

OpenMP Offload Constructs: Requirements

```
#include <omp.h>
#define N 10000
#pragma omp <offload_function_spec>
void foo(double *, double *, double *, int );
int main(){
    int i; double a[N], b[N], c[N];
    for(i=0;i<N;i++){ a[i]=i; b[i]=N-1-i;}

    ...
    #pragma omp <offload_this>
    foo(a,b,c,N);
}
#pragma omp <offload_function_spec>
void foo(double *a, double *b, double *c, int n){
    int i;
    #pragma omp parallel for
    for(i=0;i<n;i++) { c[i]=a[i]*2.0e0 + b[i]; } }
```

- Direct compiler to offload function or block
- "Decorate" function and prototype
- Ideally, familiar-looking OpenMP directives work on device

Early MIC Programming Experiences

- Codes port easily
  - Minutes to days depending mostly on library dependencies
- Performance requires real work
  - Really need to put in the effort to get what you expect
- Optimizing for MIC is similar to optimizing for CPUs
  - "Optimize once, run anywhere"
  - Early MIC ports of real codes can show 2x Sandy Bridge performance
- Scalability is pretty good
  - Forking *multiple* threads per core is *really important*
  - Getting your code to vectorize is also *really important*

Roadmap: What Comes Next?

- Expect many of the upcoming large systems to be accelerated
- MPI + OpenMP will be the main HPC programming model
  - If you are not using Intel TBBs or Cilk
  - If you are not spending all your time in libraries (MKL, etc.)
- Many HPC applications are pure-MPI codes
  - Start thinking about upgrading to a hybrid scheme
  - Adding OpenMP is a larger effort than adding MIC directives
- Special MIC/OpenMP considerations
  - Many more threads will be needed:
    60+ cores on production Xeon Phi™ → 60+/120+/240+ threads
  - Good OpenMP scaling (and vectorization) are much more important
- Stampede to deploy January 2013

Reference

- Much of the information in this talk was gathered from presentations at the TACC–Intel Highly Parallel Computing Symposium, Austin, Texas, April 10–11, 2012: http://www.tacc.utexas.edu/ti-hpcs12.
- Early draft Stampede User Guide https://portal.tacc.utexas.edu/user-guides/stampede
- Intel press materials http://newsroom.intel.com/docs/DOC-3126
- Intel MIC developer information http://software.intel.com/mic-developer