The **Stampede** is Coming: A New Petascale Resource for the Open Science Community

Jay Boisseau
Texas Advanced Computing Center
boisseau@tacc.utexas.edu
Stampede: Solicitation

- US National Science Foundation (NSF) competition in 2011 for large HPC system
  - To support open science across all domains of science and engineering
  - To enable simulation-based & data-driven science

- Solicitation requirements & constraints
  - $25M production system, plus up to $5M for ‘innovative component’
  - Up to $6M/year for 4 years of O&M
  - In production by January 2013 (fingers crossed)
Stampede: NSF HPC Context

• System must be integrated into XSEDE
  • Allocations, integrated support, etc.

• Will effectively replace Ranger (TACC) and Kraken (NICS) HPC systems
  – They expire in February 2013 (Ranger), June 2014 (Kraken)
  – So probably had to bid at least 1½ - 2 PF

• Does not come with separate funding for networking, storage, visualization—but all are required!
TACC’s Stampede Project

We wanted a comprehensive science system:

- Tremendous HPC capability, with powerful processors
- Support for HTC, big shared memory apps as well
- Powerful remote viz subsystem: 128 high-end GPUs
- Integration with archive, and (coming soon) TACC global work filesystem and other data systems
- Software complementing hardware to support both simulation-based and data-driven science
- People, training, and documentation to support diverse users, applications, and modes of computational science
Stampede: Background

• TACC evaluation
  – Linux/x86 easiest path for XSEDE users
  – Needed additional performance from innovative component
    • community really needs >> 2PF due to lack of recent large-scale NSF HPC systems
    • so, we considered ‘acceleration’ options for more PFs
  – Evaluated GPU and MIC—decided on MIC
    • Easier porting
    • More programming options (x86)—not just acceleration
    • Leveraging x86 optimization experience
    • More ‘innovative?’ (Didn’t exist yet...)
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?
Intel’s MIC Approach: Coprocessor, Not Just Accelerator

• Since days of RISC vs. CISC, Intel has mastered the art of figuring out what is important about a new processing technology, and saying “why can’t we do this in x86?”

• Intel Many Integrated Core (MIC) architecture is (like GPUs) about large die, simpler circuit, much more parallelism… and in the x86 line
  – Easier to use (even as an accelerator) with standard, familiar tools (e.g. MPI, OpenMP)
  – More flexible programming modes, models
What is MIC?

- Basic Design Ideas
  - Leverage x86 architecture (CPU with many cores)
    - x86 cores that are simpler, but allow for more compute throughput
  - Leverage (Intel’s) existing x86 programming models
  - Dedicate much of the silicon to floating point ops., keep some cache(s)
  - Keep cache-coherency protocol
  - Increase floating-point throughput per core
  - Implement as a separate device
  - Strip expensive features (out-of-order execution, branch prediction, etc.)
  - Widen SIMD registers for more throughput
  - Fast (GDDR5) memory on card
Coprocessor vs. Accelerator

- Architecture: x86 vs. streaming processors
  coherent caches vs. shared memory and caches
- HPC Programming model:
  extension to C++/C/Fortran vs. CUDA/OpenCL
  OpenCL support
- Threading/MPI:
  OpenMP and Multithreading vs. threads in hardware
  MPI on host and/or MIC vs. MPI on host only
- Programming details
  offloaded regions vs. kernels
- Support for any code: serial, scripting, etc.
  Yes  No

• Native mode: Any code may be “offloaded” as a whole to the coprocessor
What We Like about MIC

• Intel’s MIC is based on x86 technology
  – x86 cores w/ caches and cache coherency
  – SIMD instruction set

• Programming for MIC 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 MIC is similar to optimizing for CPUs
  – “Optimize once, run anywhere”
  – Our early MIC porting efforts for codes “in the field” are frequently doubling performance on Sandy Bridge.
TACC’s Stampede: The Big Picture

- Dell, Intel, and Mellanox are vendor partners
- Almost 10 PF peak performance in initial system (2013)
  - 2+ PF of Intel Xeon E5 (6400 dual-socket nodes)
  - 7+ PF of Intel Xeon Phi (MIC) coprocessors (at least 6400)
    - Special edition/release for this project—61 cores
- 14+ PB disk, 150+ GB/s I/O bandwidth
- 272+ TB RAM
- 56 Gb/s Mellanox FDR InfiniBand interconnect
- 16 x 1TB large shared memory nodes
- 128 Nvidia Kepler K20 GPUs for remote viz
- 182 racks, 6 MW power!
New Technologies/Milestones in the Stampede Project

• Density: surpasses 40KW/Cabinet
  – New Dell node designs to support multiple 300W expansion cards in single node ~1U

• Total system power past 5 *megawatts*
  – Thermal storage technology incorporated.

• Breakthrough price/performance and power/performance.
  – Inclusion of Intel MIC; but we must program it.

• Application concurrency past 1 *million* threads per application
Stampede: How Will Users Use It?

- 2+ PF Xeon-only system (MPI, OpenMP)
  - Many users will use it as an extremely powerful Sandy Bridge cluster—and that’s OK!
    - They may also use the shared memory nodes, remote vis
- 7+ PF MIC-only system (MPI, OpenMP)
  - Homogeneous codes can be run entirely on the MICs!
- ~10PF heterogeneous system (MPI, OpenMP)
  - Run separate MPI tasks on Xeon vs. MIC; use OpenMP extensions for offload for hybrid programs
Will My Code Run on MIC?

- Yes

- That’s the wrong question, it’s:
  - Will your code run *best* on MIC?, or
  - Will you get great MIC performance without additional work?
Stampede Programming
Five Possible Models

- Host-only
- Offload
- Symmetric
- Reverse Offload
- MIC Native
Key Questions

Why do we need to use a thread-based programming model?

MPI programming

Where to place the MPI tasks?

How to communicate between host and MIC?
Early MIC Programming Experiences at TACC

• Codes port easily
  – Minutes to days depending mostly on library dependencies

• Performance can require real work
  – While the sw 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 *really important*
  – Getting your code to vectorize *really important*
Programming MIC with Threads

Key aspects of the MIC coprocessor

- A lot of local memory, but even more cores
- 100+ threads or tasks on a MIC

One MPI task per core? — Probably not
- Severe limitation of the available memory per task
- Some GB of Memory & 100 tasks
  ➞ some ten’s of MB per MPI task

➤ Threads (OpenMP, etc.) are a must on MIC
MIC Programming with OpenMP

- MIC specific **pragma** precedes OpenMP pragma
  - Fortran: `!dir$ omp offload target(mic) <...>`
  - C: `#pragma offload target(mic) <...>`

- All data transfer is handled by the compiler *(optional keywords provide user control)*

- Options for:
  - Automatic data management
  - Manual data management
  - I/O from within offloaded region
    - Data can “stream” through the MIC; no need to leave MIC to fetch new data
    - Also very helpful when debugging (print statements)
  - Offloading a subroutine and using MKL
“Offloaded” Execution Model

- MPI task execute on the host
- Directives “offload” OpenMP code sections to the MIC
- Communication between MPI tasks on hosts through MPI
- Communication between host and coprocessor through “offload” semantics
- Code modifications:
  - “Offload” directives inserted before OpenMP parallel regions

One executable (a.out) runs on host and coprocessor
Base program:

```c
#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]; }
}
```

- Example program
- Offload engine is a device
- Objective: Offload foo
- Do OpenMP on device
```c
#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]; } } }
```

**Function offload: Requirements**

- Direct compiler to offload function or block
- “Decorate” function and prototype
- Usual OpenMP directives on device
The Stampede is Coming

• Stampede has a *lot* of new technologies, and as such has a certain element of risk.
• However, as of now, we expect the early user/science period to begin in December.
• Full production by January 7\textsuperscript{th} (Base system at least).
• You might hear some more about this machine at SC12 in November 😊.
• Thank to NSF, Intel, Dell
Datacenter Expansion

Ranger: 3000 ft²

Stampede: 8000 ft²
Power/Physical

- Stampede will physically use 182 48U cabinets.
- Power density (after upgrade in 2015) will exceed 40kW per rack.
- Estimated 2015 peak power is 6.2MW.
Key Datacenter Features

• Thermal energy storage to reduce peak power charges
• Hot aisle containment to boost efficiency (and simply provide enough cooling).
• Total IT power to 9.5MW, total power ~12MW.
• Expand experiments in mineral oil cooling.
Stampede: Powered by Intel Xeon E5 and Xeon Phi

- TACC & Intel have provided open science clusters since 2001
- x86 architecture has become dominant ISA in supercomputing
- Stampede needed to enable discoveries that require petaflops
- Xeon E5 enables more far performance for general applications
- Xeon Phi enables tremendous potential performance for highly parallel applications, with much easier porting than GPUs
- Thus, Stampede will preserve (and reward!) many years of x86 experience and expertise in scientific codes, while offering world-class performance
Stampede Will Enable New Scientific Discoveries Across Domains

1000+ projects, by thousands of researchers