A Unified Approach to Heterogeneous Architectures Using the Uintah Framework

Qingyu Meng, Alan Humphrey, Martin Berzins

Thanks to: TACC Team for early access to Stampede
John Schmidt and J. Davison de St. Germain, SCI Institute
Justin Luitjens and Steve Parker, Nvidia

DOE for funding the CSAFE project (97-10), DOE NETL, DOE NNSA
NSF for funding via SDCI and PetaApps
Uintah Overview

- Parallel, adaptive multi-physics framework
- Fluid-structure interaction problems
- Patch-based AMR using:
  - particles and mesh-based fluid-solve

- Explosions
- Plume Fires
- Foam Compaction
- Sandstone Compaction
- Industrial Flares
- Carbon Capture Clean Coal Boiler
- Shaped Charges
- Virtual Soldier
Uintah - Scalability

- 256K cores – Jaguar XK6
- 95% weak scaling efficiency & 60% strong scaling efficiency
- Multi-threaded MPI – shared memory model on-node
- Scalable, efficient, lock-free data structures

Uintah Task-Based Approach

- Task Graph
  - Directed Acyclic Graph

- Asynchronous, out of order execution of tasks
  - Multi-stage work queue design

- **Task** – basic unit of work
  - Key idea
  - C++ method with computation

- Allows Uintah to be generalized to support co-processors and accelerators

- No sweeping code changes
Emergence of Heterogeneous Systems

Motivation - Accelerate Uintah Components
Utilize all on-node computational resources
Uintah’s asynchronous task-based approach well suited for Co-processors and Accelerator designs

Natural progression: Accelerator & Co-processor Tasks
Unified Heterogeneous Scheduler & Runtime

GPU support on Keeneland and Titan
The Emergence of the Intel Xeon Phi
Intel Xeon Phi – What is it?

- Co-processor
  - PCI Express card
  - Light weight Linux OS (busy box)
- Dense, simplified processor
  - Many power-hungry operations removed
  - Wider vector unit
  - Wider hardware thread count
- Many Integrated Core architecture, aka MIC
  - Knights Corner (code name)
  - Intel Xeon Phi Co-processor (product name)
Intel Xeon Phi – What is it?

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

George Chrysos, Intel, Hot Chips 24 (2012):
Intel Xeon Phi

4 Hardware threads/core

Knights Corner Core

X86 specific logic < 2% of core + L2 area

George Chrysos, Intel, Hot Chips 24 (2012):
George Chrysos, Intel, Hot Chips 24 (2012):
Programming for the Intel Xeon Phi and TACC Stampede System
Programming Advantages

- 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”
  - Early MIC porting efforts for codes “in the field” are frequently doubling performance on Sandy Bridge.
Xeon Phi Programming Models

• Traditional Cluster
  • Pure MPI and MPI+X
    • X: OpenMP, TBB, Cilk+, OpenCL, …

• Native Phi
  • Use 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

• MPI on Host, Offload to Xeon Phi
  • Targeted offload through OpenMP extensions
  • Automatically offload some library routines with MKL
Xeon Phi Execution Models

(1) Host-only Model

(2) MIC Native Model

(3) Offload Model

(4) Symmetric Model
Xeon Phi Execution Models

**Host Only** – Ignore Xeon Phi cards

**Native**
- Compile with –mmic, MPI/OpenMP/Pthreads
- Single node only, can’t run multi-node jobs
- Need one MPI rank on a host CPU

**Offload**
- MPI inter-node, OpenMP on-node
- Synchronous – no overlapping
- Asynchronous – overlapping with signal/wait

**Symmetric**
- MPI internode, OpenMP / Pthreads on-node
- Use both host and co-processor simultaneously
NSF Stampede System
Texas Advanced Computing Center

- $27.5M acquisition
- 10 petaflops (PF) peak performance
- 2+ PF Linux cluster
  - 6400 Dell DCS C8220X nodes
  - 2.7GHz Intel Xeon E5 (Sandy Bridge)
    - 102,400 total cores
  - 56Gb/s FDR Mellanox InfiniBand
  - 7+ PF Intel Xeon Phi Coprocessor (1.0GHz cores)
  - 500,000+ total cores
    - TACC has a special release: Intel Xeon Phi SE10P
  - 14+ PB disk, 150GB/s
  - 16 1TB shared memory nodes
  - 128 NVIDIA Tesla K20 GPUs

TACC
XSEDE
Extreme Science and Engineering Discovery Environment
## Stampede Processor Specs

<table>
<thead>
<tr>
<th>Arch. Features</th>
<th>Xeon E5 (Sandy Bridge)</th>
<th>Xeon Phi SE10P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>2.7GHz+turbo</td>
<td>1.0GHz +turbo</td>
</tr>
<tr>
<td>Cores</td>
<td>8</td>
<td>61</td>
</tr>
<tr>
<td>HW Threads/core</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Vector Size</td>
<td>256 bits, 4 doubles, 8 singles</td>
<td>512 bits, 8 doubles, 16 singles</td>
</tr>
<tr>
<td>Inst. Pipeline</td>
<td>Out of Order</td>
<td>In order</td>
</tr>
<tr>
<td>Registers</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Caches</td>
<td>L1:32KB, L2:256KB, L3:20MB</td>
<td>L132KB, L2:512KB</td>
</tr>
<tr>
<td>Memory</td>
<td>2 GB/core</td>
<td>128 MB/core</td>
</tr>
<tr>
<td>Sustained Memory BW</td>
<td>75 GB/s</td>
<td>170 GB/s</td>
</tr>
<tr>
<td>Sustain Peak FLOPS</td>
<td>1 thread/core</td>
<td>2 threads/core</td>
</tr>
<tr>
<td>Instruction Set</td>
<td>x86+ AVX</td>
<td>x86+ new vector instructions</td>
</tr>
</tbody>
</table>
Stampede Compute Node

- Two Xeon E5 8-core CPUs
- 16 cores
- 32G RAM

- Xeon Phi Coprocessor
- 61 lightweight cores 8G RAM
- Each core has 4 hardware threads
- Runs micro Linux OS (BusyBox)

Comm layer & Virtual IP Service For MIC

CPUs and MIC appear as separate HOSTS ("symmetric" computing)

PCIe X16
TACC Facilities Footprint (Stampede & Ranger)

Stampede:
- 8000 ft\(^2\)
- 10 PF
- 6.5 MW
- InfiniBand (fat-tree)
- ~75 Miles of InfiniBand Cables

Ranger: (decommissioned)
- 3000 ft\(^2\)
- 0.6 PF
- 3 MW
- InfiniBand (fat-tree)
- ~75 Miles of InfiniBand Cables

Capabilities: 20x
Footprint: 2x

Stampede footprint figure courtesy, Bill Barth, TACC
Preliminary Experiences with the Uintah Framework on Intel Xeon Phi and Stampede
Host-only Model

- Intel MPI issues beyond 2048 cores (seg faults)
- MVAPICH2 required for larger core counts

- Using Hypre with a conjugate gradient solver
- Preconditioned with geometric multi-grid
- Red Black Gauss Seidel relaxation - each patch
Uintah on Xeon Phi
Native Model

- Compile with –mmic (cross compiling)

- Need to build all Uintah required 3p libraries
  - libxml2
  - libz

- Run Uintah natively on Xeon Phi within 1 day

- Single Xeon Phi Card

![Graph showing AMR MPMICE performance](image)

- 2 MPI processes per Phi core, up to 120 processes
- 2 threads per Phi core, up to 120 threads
- Lock-free multi-threaded MPI
Uintah on Xeon Phi Offload Model

- Use compiler directives (#pragma)
  - Offload target: #pragma offload target(mic:0)
  - OpenMP: #pragma omp parallel

- Find copy in/out variables

- Functions called in MIC must be defined with __attribute__((target(mic)))

- Hard for Uintah to use offload mode
  - Rewrite highly templated C++ methods with simple C/C++ so they can be called on the Xeon Phi
  - Have to use OpenMP: Uintah currently use MPI+ Pthreads
  - Less effort than GPU port, but still significant work for complex code such as Uintah
Uintah on Xeon Phi Symmetric Model

- Best fits current Uintah model
- Xeon Phi directly calls MPI
- Two MPI processes per node:
  - Use Pthreads on both host CPU and Xeon Phi:
    - 1 MPI process on host – 16 threads
    - 1 MPI process on MIC – up to 120 threads
- Currently only Intel MPI supported
  - mpiexec.hydra -n 8 ./sus – nthreads 16 : -n 8./sus.mic –nthreads 120
- No major Uintah code changes
Symmetric Model Challenges

- **Native debugging with** *gdb*
  - Built *gdb* for ourselves from source
  - Weren’t aware of *idb* / *idbc* (need idb server/client setup)

- **Different floating point accuracy on host and co-processor**

- **MPI message mismatch issue**
  - Any control logic based on result of FP calculation may choose different execution routes on Xeon Phi and host
  - Size of MPI send/rcv buffers determined by FP result
  - Caused Uintah MPI message mismatch (MSG_TRUNC)

---

\[
p = 0.421874999999999944488848768742172978818416595458984375 \\
\frac{c}{b} = 0.002604166666666665221063770019327421323962655086744384765625 \\
b = p/c = 162.99999999999999 \\
\]
Scaling Results on Xeon Phi Symmetric Model

AMR MPMICE (multi Xeon Phi)

60 threads per MPI process, 2 MPI processes per Xeon Phi card
16 threads per MPI process, 1 MPI processes per host CPU

Scaling results limited by development queue allowances for symmetric model
Current and Future Work

- Load Balancer
  - Cannot treat all MPI ranks uniformly
  - Profile CPU and Xeon Phi separately
  - Need separate forecast model for each.

- Address different cache sizes
  - New regridder to generate large patches for CPU and small patches for Xeon Phi

- Explicitly use long vector

- Asynchronous offload model
  - _Offload_signaled(mic_no, &c)
Questions?

http://www.uintah.utah.edu