Introduction to Xeon Phi

ACES
Austin, TX
Dec. 04 2013

Kent Milfeld, Luke Wilson, John McCalpin, Lars Koesterke
TACC
What is it?

• 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)
What is it?

- 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
Knights Corner Core

George Chrysos, Intel, Hot Chips 24 (2012):
Vector Processing Unit

George Chrysos, Intel, Hot Chips 24 (2012):
Speeds and Feeds

• 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
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”
  – Our early MIC porting efforts for codes “in the field” are frequently doubling performance on Sandy Bridge.
Stampede Programming Models

• Traditional Cluster
  – Pure MPI and MPI+X
    • X: 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

• 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
- ... or ssh to mic0 and run on the Phi
- Can safely use all 61 cores
  - Offload programs should 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
- 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.
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.
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?
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
Lab I

• What you will learn
  – The lab introduces you to Stampede and to the Xeon Phi processors built into Stampede

• What you will do:
  – Compile for Xeon (host) and Xeon Phi (MIC)
  – Submit a job
  – Inspect the queue
  – Submit an interactive job
  – Execute on the host and on the Phi