Intel Xeon Phi

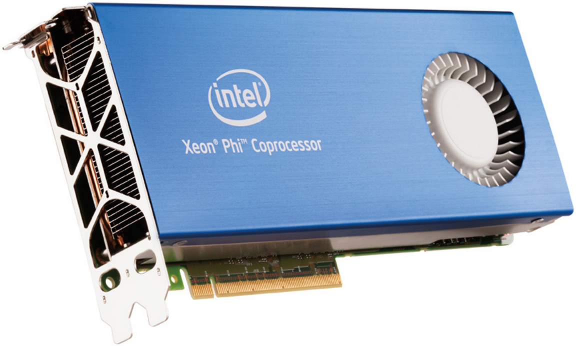
SNUG TechTalk

SciNet
www.scinet.utoronto.ca
University of Toronto
Toronto, Canada

February 12, 2014

Outline

Xeon Phi



http://www.intel.com

Xeon Phi

What is it?

- Intel x86 based Accelerator/Co-processor
- Many Integrated Cores (MIC) Architecture
- Large number of low-powered, but low cost (computational overhead, power, size, monetary cost) processors (pentiums).
- Heterogeneous computing: Host and Phi can work together on the problem.

Xeon Phi

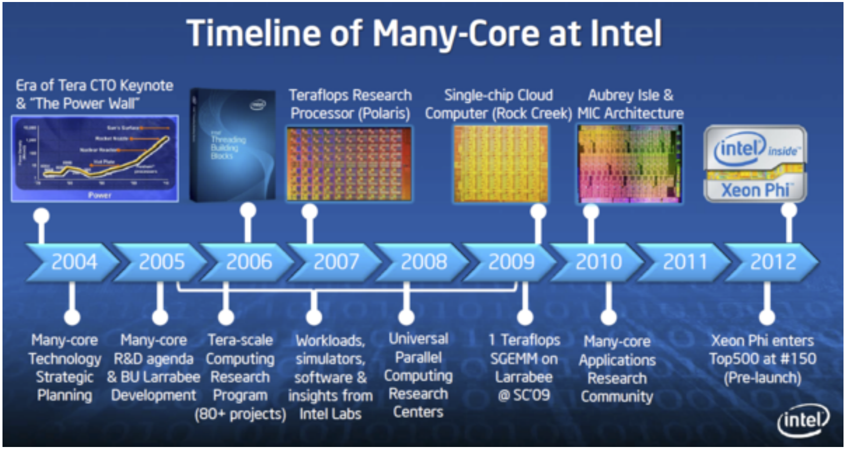
What is it?

- Intel x86 based Accelerator/Co-processor
- Many Integrated Cores (MIC) Architecture
- Large number of low-powered, but low cost (computational overhead, power, size, monetary cost) processors (pentiums).
- Heterogeneous computing: Host and Phi can work together on the problem.

Xeon Phi 5110P

- 60 cores @ 1.053GHz
- 8 GB memory
- 4-way SMT
- PCIe Gen2 bus connectivity
- Runs linux onboard

History


Timeline of Many-Core at Intel

Era of Tera CTO Keynote & "The Power Wall"

Teraflops Research Processor (Polaris)

Single-chip Cloud Computer (Rock Creek)

Aubrey Isle & MIC Architecture

2004 2005 2006 2007 2008 2009 2010 2011 2012

Many-core Technology Strategic Planning

Many-core R&D agenda & BU Larrabee Development

Tera-scale Computing Research Program (80+ projects)

Workloads, simulators, software & insights from Intel Labs

Universal Parallel Computing Research Centers

1 Teraflops SGEMM on Larrabee @ SC'09

Many-core Applications Research Community

Xeon Phi enters Top500 at #150 (Pre-launch)

http://www.intel.com

History

History

- Larrabee (concept video card design)
- Knights ferry (MIC development platform)
  - 32 core, 2GB
  - $\sim$ 750 GFlops SP
- Knights corner (Xeon Phi)
  - 60 core, 8/16GB
  - $\sim$ 1 TFlops DP
- Knights landing (NextGen 2015)
  - 72 core, upto 384GB
  - $\sim$ 3 TFlops DP
  - native processor
  - storm lake interconnect

Compute Canada Xeon Phi Resources

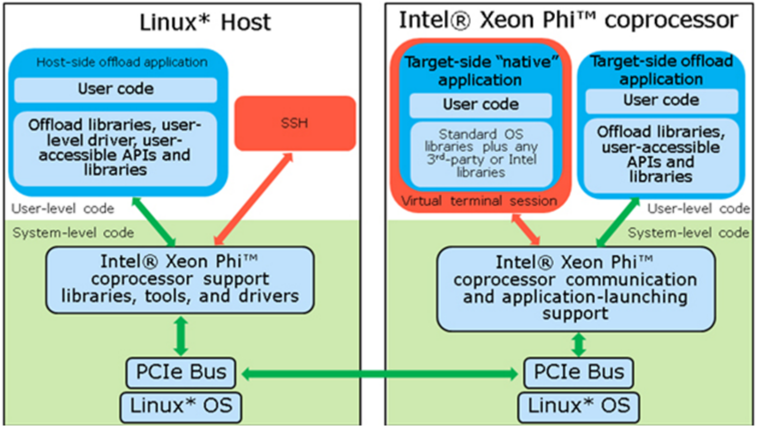
SciNet - ArcX

- 1 node (1 x 8-core Sandybridge Xeon, 32GB)
- 1 x Intel Xeon Phi 3120A (57 1.1 GHz cores and 6GB)
- `qsub -l nodes=1:ppn=8,walltime=2:00:00 -q arcX -I`
- module load intel/14.0.1 intelmpi/4.1.2.040

Calcu Quebec - Guillimin

- 50 nodes (2 x 8-core Intel Sandy Bridge Xeon, 64GB)
- 2 x Intel Xeon Phi 5110P (60 1.053GHz cores and 8GB)

Software Stack



Linux* Host
Host-side offload application
User code
Offload libraries, user-level driver, user-accessible APIs and libraries
SSH
User-level code
System-level code
Intel® Xeon Phi™ coprocessor support libraries, tools, and drivers
PCIe Bus
Linux* OS
Intel® Xeon Phi™ coprocessor
Target-side "native" application
User code
Standard OS libraries plus any 3rd-party or Intel libraries
Virtual terminal session
Target-side offload application
User code
Offload libraries, user-accessible APIs and libraries
User-level code
System-level code
Intel® Xeon Phi™ coprocessor communication and application-launching support
PCIe Bus
Linux* OS

http://www.intel.com

Programming

Languages

- C, C++, Fortran
- MPI, OpenMP 4.0, OpenCL
- TBB, Cilk+

Programming

Languages

- C, C++, Fortran
- MPI, OpenMP 4.0, OpenCL
- TBB, Cilk+

Tools

- Intel Compilers (icc, icpc, ifort)
- Intel MPI
- Intel Tools (VTune, Advisor, Inspector, etc.)

Phi Operating Modes

Single Node Operating Modes

- Native
  - Compile and run native on the phi only
  - Cross compile with -**mmic** option
  - Login and run using **ssh mic0** or using **micnatveloadex**
- Offload
  - Compile and run on Host, with offload to Phi
  - More flexible, but heterogeneous computing
  - Device and Host memory not accessible to each other so data copies required
  - Similar to GPU accelerator model

Phi Operating Modes

MPI Operating Modes

- Native
  - Compile and run native on the phi only
  - Limited to single phi
- Symmetric
  - MPI processes run on both the CPU and the Xeon Phi
  - Load balancing required
- Offload
  - Host to Host with MPI, use Host offload to Phi
  - i.e. Heterogenous/Hybrid code design
  - Most likely design for large salable codes

File System on Phi

Phi and the SciNet Filesystem

- Phi **$HOME** is **/localscratch/$HOME**, mounted from arc09
- Can be useful for development, but performance is in general not very good
- Minimize direct native data transfers from Phi
  (i.e. do your I/O on the host)

Examples: Native OpenMP

- Cross compile a standard OpenMP code on host

```
$module load intel/14.0.1
$icc -o omp.MIC -mmic -openmp omp.c
```

- Run using ssh

```
$cp omp.MIC cd /localscratch/$HOME
$ssh mic0
$export OMP_NUM_THREADS=60
$export LD_LIBRARY_PATH=
/home/scinet/intel/composer_xe_2013_sp1.0.080/compiler/lib/mic
$./omp.MIC
```

- Run using micnativeloadex

```
$micnativeloadex omp.MIC -e "OMP_NUM_THREADS=60"
```

Example: Offload with OpenMP

- Programmer specified parts are 'offloaded' to the device

```
#pragma offload target(mic:0)
```

- Declare device variables

```
#pragma offload_attribute(push, target(mic))
static float *indata, *outdata;
#pragma offload_attribute(pop)
```

- Data movement to/from host/device

```
#pragma offload target(mic:0) \
in(indata:length(size)) \
out(outdata:length(size))
```

Example: Offload with OpenMP

Code

```
#define SIZE=10000

//device memory }
#pragma offload_atribute(push, target(mic))
static float *indata, *outdata;
#pragma offload_attribute(pop)

//device code
__attribute__((target(mic))) void devicework(float *in, float *out)
int i;
#pragma omp parallel for shared(out,in) private(i)
for (i=0; i <SIZE; i++) {
   out[i] = sqrt(in[i]);
}
```

Example: Offload with OpenMP

Code - continued

```
int main(int argc, char* argv[]) {
  indata = (float*)malloc(SIZE*sizeof(float));
  outdata = (float*)malloc(SIZE*sizeof(float));

  for (i = 0; i < SIZE; i++) {
    indata[i] = (float) (i*i);
    outdata[i] = 0.0;
  }

  for(i = 0; i < 100; i++){
   #pragma offload target(mic:0) \
           in(indata:length(SIZE)) \
           out(outdata:length(SIZE))

         devicework(indata,outdata)
  }

  free(indata); free(outdata);
  return 0;
}
```

Example: Offload with OpenMP

- Compile as host code

```
$ icc -o offload -openmp offload.c
```

- Run natively

```
$export MIC_OMP_NUM_THREADS=4
$./offload
```

- For more offload details set

```
$export OFFLOAD_REPORT=3
```

Offload with OpenMP

Memory Persistence

- Data transfers are expensive and should be minimized
- By default variables are allocated at beginning of offload segment, and free'd at the end
- However, data can persist on the device between offload segments

Example: Offload with OpenMP

Code: memory fixed

```
\\after initialized data

#pragma offload_transfer target(mic:0) \
 nocopy(indata, outdata : length(SIZE) alloc_if(1) free_if(0))

for(i = 0; i < 100; i++){
 #pragma offload target(mic:0)  \
         in(indata:length(SIZE) alloc_if(0) free_if(0)) \
         out(outdata:length(SIZE) alloc_if(0) free_if(0))

       devicework(indata,outdata)
}

#pragma offload_transfer target(mic:0) \
 nocopy(indata, outdata : length(SIZE) alloc_if(0) free_if(1))

 free(indata); free(outdata);
```

More Phi Considerations

Vectorization

- 16 wide simd registers (AVX-512)
- `#pragma simd`
- cilk+ C/C++ extensions

Threads

- 4 way SMT (60-240 threads)
- thread affinity
- `KMP_AFFINITY=balanced/compact/scatter`

MKL

- Automatic Offload
- Compiler Assisted Offload
- MKL ScaLAPACK and Cluster FFT (with IntelMPI)

Which Accelerator?

GPU vs. Phi Considerations

- CUDA/OpenCL vs. native C/C++/Fortran & OpenMP/MPI
- 500* vs 60 cores
- GPU more complex programming/memory model
- GPU has higher SP Flops
- DP Flops about par
- More apps & libs for GPUs today*
- Direct access to Xeon Phi (ie ssh mic0)
- Can mount filesystems on Phi (nfs off host)

Summary

Xeon Phi

- High computational intensity (Flops/Watt)
- Interesting compromise between GPU and CPU.
- In many ways analogous to Blue Gene Q
- Promotes heterogeneous offload and Hybrid (MPI & OpenMP) programming
- Top 500 list (#1 Tianhe-2 and #7 Stampede)
- Knights Landing 'native' chip will remove the separate memory bottleneck