Before We Start

• Sign in
• Copy files from the MIC directory in the Shared Folder on the desktop
• Follow the steps on page 1 of the MIC Exercises document
  – See also the Basic Linux Commands handout
Programming for the Intel Xeon Phi Architecture
Advanced Research Computing
March 4, 2014
Outline

• Background
• The Intel Xeon Phi
• Native Mode
• Offloading
  – Automatic
  – Directives
  – Multiple MICs
• MPI
BACKGROUND
The Need for Parallelism

Intel Processor Clock Speed (MHz)

- Pentium 4 Prescott
- Core 2 Extreme
- Pentium III
- Celeron
- Pentium
- 80486
- 80386
- 80286
- 8080

Year:
- 1968
- 1973
- 1979
- 1984
- 1990
- 1995
- 2001
- 2005

Multicore Crisis Is Here!
HPC Trends

Architecture | Code
---|---
Single core | Serial
Multicore | OpenMP
GPU | CUDA
Cluster | MPI
THE INTEL XEON PHI
What is a Xeon Phi?

• “Coprocessor”...something between an accelerator and a node:
  – More cores/threads, slower clock speed
  – Runs Linux (stripped down, based on busybox)
  – x86 architecture

• Terminology
  – “Host”
  – “MIC” vs. “Xeon Phi” vs. “Knights Corner”

• Stampede: Large MIC-based system at Texas Advanced Computing Center (TACC) at UT-Austin
Advantages

• X86 architecture supports familiar programming techniques:
  – C/C++ and Fortran
  – OpenMP and MPI

• Optimizing is similar for both CPU and MIC
  – Vectorize
Considerations

• Binaries are incompatible between CPU and MIC – need to compile separately
• Cores are slow so parallelism is critical
• Vector unit is wider so vectorization is critical
  – Memory alignment
• Memory is limited – MPI programs will not scale on their own
• Ring topology for L2 cache means reduction operations can be very bad
BlueRidge Overview

- 318 compute nodes:
  - Two Intel Sandy Bridge CPUs per node (16 cores/node)
  - Memory: 4 GB/core, 64 GB/node
- Total: 5,088 cores, 20 TB memory
- 130 nodes with 2 MICs each
- Interconnect: Quad-data-rate (QDR) InfiniBand
- Top500 #402 (November 2012)
- Operating system: CentOS 6
- Requires allocation application
# BlueRidge MIC Nodes

<table>
<thead>
<tr>
<th>Specification</th>
<th>CPU (× 2)</th>
<th>MIC (× 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Intel Xeon E5-2670 (Sandy Bridge)</td>
<td>Intel Xeon Phi 5110P</td>
</tr>
<tr>
<td>Cores</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>2.60 GHz</td>
<td>1.05 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>64 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>32 KB (per core)</td>
<td>32 KB (per core)</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>512 KB (per core)</td>
<td>512 KB (per core)</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>20 MB (shared)</td>
<td>N/A</td>
</tr>
<tr>
<td>Vector Unit</td>
<td>256 bit (4 DPFP)</td>
<td>512 bit (8 DPFP)</td>
</tr>
<tr>
<td>Theoretical Peak (DP)</td>
<td>166 Gflops/s</td>
<td>1,011 Gflops/s</td>
</tr>
</tbody>
</table>
BlueRidge MIC Basics

• Compile on the MIC (compute) nodes, not on the login nodes
  – Interactive job
• Linux environment is a little different, e.g. `ps` has reduced functionality
• File system mounts:
  – `/home`: Permanent files (e.g. executables, data)
    • Can be slow
  – `/work`: Fast scratch (temporary)
  – `/opt/apps` is available, however the apps stack has not been changed and may not work on the MIC
Usage Models

• Native: Log into the MIC and run
• Offload: Run on CPU, push portions of code to the MIC
  – Directives: Manually identify code for Phi
  – Automatic: MKL libraries use Phi without changes to code
• MPI: Native and symmetric (shared host/MIC)
NATIVE JOBS
Native Jobs

• Compile (on the host!) with the \texttt{-mmic} flag:
  \begin{verbatim}
  icc -mmic -openmp -O3 helloflops3.c -o helloflops3
  \end{verbatim}

• Running:
  – Log into the MIC (either \texttt{mic0} or \texttt{mic1}):
    \begin{verbatim}
    ssh mic0
    \end{verbatim}
  – Run as you would on a CPU:
    \begin{verbatim}
    OMP_NUM_THREADS=8 ./helloflops3
    \end{verbatim}
Native Jobs: Notes

• MIC OS can wind up skipping every other clock cycle when only one thread is on a core
  – Default is 236 threads (4 per core), but preference may vary
  – Skip core 0, which runs the OS

• Use –vec-report flag to check vectorization

• Thread affinity – use **scatter** or **balanced**
  – **scatter**:  
    | 0 | 4 | 1 | 5 | 2 | 6 | 3 | 7 |
  – **balanced**:  
    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
  – **compact**:  
    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
OFFLOAD JOBS
Offload: Basics

- Program runs on the host (CPU)
- CPU pushes some computations to the MIC:
  - Programmer-specified (manual)
  - MKL-specified (automatic)
- Use `OFFLOAD_REPORT` environment variable to print more information about offload
- Need to manage bandwidth between host and MIC to get performance
- On MIC, offload processes run by `micuser`
Offload: Environment Variables

• Use environment variables to control how the offload portion is run on the MIC

  • \texttt{MIC\_ENV\_PREFIX} sets the prefix for variables that will be used on the MIC
    – Default is \texttt{MIC}

• Then, for example,
  – \texttt{MIC\_OMP\_NUM\_THREADS} sets the number of threads on the MIC
  – \texttt{MIC\_KMP\_AFFINITY} sets the thread affinity on the MIC

• \textbf{Note that} \texttt{OMP\_NUM\_THREADS} controls the number of threads on the \textit{host}
Automatic Offload

• Intel’s Math Kernel Library (MKL) has built-in functionality to offload to the MICs
  – Level 3 BLAS beyond a certain problem size
• `MKL_MIC_ENABLE` environment variable controls whether automatic offloading is enabled
• Run on the CPU:
  \[ \text{MKL_MIC_ENABLE}=0 \quad ./\text{mm_mkl} \quad -s \quad 8192 \]
• Run with offload to the MIC:
  \[ \text{MKL_MIC_ENABLE}=1 \quad ./\text{mm_mkl} \quad -s \quad 8192 \]
Manual Offload: Workflow

• Compiler directives tell the host when to push computation to the MIC:
  #pragma offload target(mic)

Or to target a particular MIC (mic0 in this case):
  #pragma offload target(mic:0)

• Compile as a normal CPU program:
  icc -openmp -O3 omp_hello_offload.c -o omphw.offload

• Load the mkl and mic modules to load libraries:
  module load mkl mic

• Run like a normal CPU program:
  ./omphw.offload
Example: Running Offload

```bash
icc -openmp -O3 omp_hello_offload.c -o omphw.offload

module load mkl mic
export MIC_ENV_PREFIX=MIC

export MIC_OMP_NUM_THREADS=2
./omphw.offload
Hello World from thread = 0
Number of threads = 2
Hello World from thread = 1

export MIC_OMP_NUM_THREADS=4
./omphw.offload
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 1
Hello World from thread = 2
Hello World from thread = 3
```
Asynchronous Offload

- Keep the host working while the offload to the MIC is occurring
- Wait for a previously specified offload:

```c
sig = 25; //signal for offload
//offload to mic
#pragma offload target(mic:0) signal( &sig ) {
    //do some work on the mic
}
//do work on the host here
//pause host until offload ‘sig’ is complete
#pragma offload_wait target(mic:0) wait( &sig )
```
MPI and the MIC

• Several models:
  – To offload to multiple MICs
  – Run directly on the MIC
  – Run across both the MICs and the host(s)

• Currently only available for Intel MPI
  – Coming soon on BlueRidge
  – mvapich2-mic in development (on Stampede)
MPI: Notes

• Need to build separate binaries for the host and the MIC:
  
  ```bash
  mpicc -openmp -o montecarlo.host montecarlo.c
  mpicc -openmp -mmic -o montecarlo.mic montecarlo.c
  ```

• Due to memory limitations, avoid running too many processes on a MIC
  
  – Probably want a few processes, each spawning threads (i.e. hybrid MPI/OpenMP)
Offload to Multiple MICs

Use MPI to start up one process per MIC (two per node), then offload from each to a MIC:

```c
// get the rank of the process on the host node
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

// get the rank of the process on the host node
localrank = rank % 2;

// offload to the MIC #pragma offload
target(mic:localrank)
```
MPI: Execution from the Host

• Run natively on a MIC:
  mpirun -n 3 ./montecarlo.mic

• Run on a MIC from the host:
  mpirun -n 3 -host mic0 ./montecarlo.mic

• Run on multiple MICs:
  mpirun -n 3 -host mic0 ./montecarlo.mic : -n 3 -host mic1 ./montecarlo.mic

• Run in symmetric mode:
  mpirun -n 2 -host br020 ./montecarlo.host : -n 3 -host mic0 ./montecarlo.mic
MPI: Streamlining Execution

• Environment variable for MIC suffix:
  ```
  export I_MPI_MIC_POSTFIX=.mic
  ```

• Define hosts file:
  ```
  br022: 16
  mic0:4
  mic1:4
  ```

• Run using hosts file:
  ```
  mpirun -machinefile hosts_file ./montecarlo
  ```
CONCLUSIONS
Getting Started on BlueRidge

• Request an account (anyone with a VT PID):
  http://www.arc.vt.edu/forms/account_request.php
  – Can also request for external collaborators

• Request a system unit allocation:
  http://www.arc.vt.edu/userinfo/allocations.php
  – MIC nodes are “charged” the same as normal nodes
References

• BlueRidge MIC User Guide:
  http://www.arc.vt.edu/resources/hpc/blueridge_mic

• Main MIC Developers Website:

• User Guide for Stampede:
  https://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide

• Intel Xeon Phi Coprocessor High Performance Programming by Jim Jeffers and James Reinders
  – Some material is out of date
Thank You!