Accelerate Your Science:
An Introduction to High Performance Computing

Session 10: An HPC App in Detail

Robert Henschel – henschel@iu.edu
Manager, Scientific Applications and Performance Tuning

Research Technologies, UITS
Indiana University

July 7th, 2014
Contents

• IUMD Code Overview

• Building and Running on Big Red II and Quarry
Goals

• What kind of performance experiments are possible

• Run jobs on Big Red II and Quarry
IUMD

• Collaboration of SciAPT and the Center for Exploration of Energy and Matter
• XSEDE allocation
• Cray user group papers
• SPEC benchmark
Accelerator History
IUMD

• Classical molecular dynamics code
  – Simulates nuclear matter (neutrons and protons)
  – Used for research in supernovas and neutron stars
IUMD
MD Algorithm

• Screened Coulomb force between protons
  – Uses simple particle-particle method: For Np protons, there are Np(Np-1) interactions
  – Cutoff reduces number of interactions

• Short range nuclear force
  – Applies to all particle pairs (nn, np, pp)
  – Uses a cell method with neighbor lists
Three Variants

• OpenMP
  – Fully utilize one node, good for shared memory systems

• MPI+OpenMP
  – OpenMP used by cores within a node
  – MPI for parallelism across nodes

• MPI+OpenMP+CUDA
  – For GPU accelerated, multi-core nodes
  – GPU does Coulomb calculation
  – CPUs do nuclear calculation (OpenMP)
  – Coulomb and nuclear run concurrently
MD Time Steps per second on BigRed-II

- CPU uses all 32 cores on dual CPU nodes
- GPU uses 16 cores plus the GPU on GPU nodes
Speedup on BigRed-II

Over One Dual-CPU Node

- GPU is 5 times faster than CPU+CPU
Comparison OpenACC and CUDA

• GPU is 5 times faster than CPU+CPU
“Cost” Comparison

• Sample experiment: 1,000,000 time steps

• Runtime
  
<table>
<thead>
<tr>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 node</td>
<td>40 vs. 580 h</td>
</tr>
</tbody>
</table>

• Decrease runtime to less than 8 hours
  
<table>
<thead>
<tr>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 nodes</td>
<td>128 vs.</td>
</tr>
</tbody>
</table>

• Overall usage
  
<table>
<thead>
<tr>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>43 vs. 606 node hours</td>
<td></td>
</tr>
</tbody>
</table>
## Find the Fastest Configuration

<table>
<thead>
<tr>
<th>Nodes</th>
<th>proc/node</th>
<th>thrd/proc</th>
<th>Total thds</th>
<th>GPUs</th>
<th>2k</th>
<th>4k</th>
<th>8k</th>
<th>10k</th>
<th>18k</th>
<th>27k</th>
<th>54k</th>
<th>96k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>32</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>16</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>1</td>
<td>32</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>1</td>
<td>32</td>
<td>256</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>8</td>
<td>2</td>
<td>16</td>
<td>256</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>256</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>256</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>16</td>
<td>1</td>
<td>32</td>
<td>512</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Data Set Size**

- 2k
- 4k
- 8k
- 10k
- 18k
- 27k
- 54k
- 96k

**RESEARCH TECHNOLOGIES**

**PERSUASIVE TECHNOLOGY INSTITUTE**
Relation to your Application

• Similar work can be done for any HPC application
• Actually... it SHOULD be done for any application you plan on running a few hundred or thousand times.
And now some Hands-On

• Compile OpenMP, MPI and CUDA code
• Run in an interactive job on BRII
• Run in a batch job on Quarry
Download/Unpack/Compile/Run

- qsub -l -X -q cpu -l nodes=1:ppn=32,walltime=1:00:00
- cd /N/dc2/scratch/<username>
- wget http://pages.iu.edu/~gjuckela/hpc/labs/session10.tar.gz
- tar -xzvf session10.tar.gz
- cd md_6.3.1/src
- module swap PrgEnv-cray PrgEnv-pgi
- make MDEF=BR2-pgi iumd_omp
- cd ../examples/
- export OMP_NUM_THREADS=32
- aprun -d 32 ../src/iumd_omp
Experiment with the OpenMP Version

- Run with 16 and 32 threads
- Test different compilers
  - “make MDEF=BR2-pgi realclean”
  - Use MDEF=BR2-cce for Cray compiler (and the PrgEnv-crays)
  - Use MDEF=BR2-intel for Intel compiler (and the PrgEnv-intel)
- Test different compiler options – find the fastest one
  - mdef/BR2-pgi.def
    - Change line 27: F95OPT = -fastsse -O3
    - Learn about PGI optimization flags with “man pgf95”
Experiment with the OpenMP Version

• Observe “aprun” output:
  – Application 2194001 resources: 32 threads
    • utime ~2232s, 69.8 seconds total runtime
    • stime ~0s,
    • Rss ~185928, 5.8 MB RAM per thread
    • inblocks ~5735,
    • outblocks ~7151
Build Hybrid Version

- `qsub -l -X -q cpu -l nodes=2:ppn=32,walltime=1:00:00`
- `cd /N/dc2/scratch/<username>/md_6.3.1/src`
- `module swap PrgEnv-cray PrgEnv-pgi`
- `make MDEF=BR2-pgi iumd_mpi_omp`
- `cd ../examples/`
- `export OMP_NUM_THREADS=32`
- `aprun -n 2 -N 1 -d 32 ../src/iumd_omp`
Experiment with the Hybrid Version

• Use different number of nodes, number of MPI processes and number of OpenMP threads
• Observe “aprun” output:
  – Application 2194000 resources: 64 threads
    • utime ~2620s, 40.9 seconds total runtime
    • stime ~0s,
    • Rss ~109668, 1.7 MB RAM per thread
    • inblocks ~7370,
    • outblocks ~9589
Build GPU Version

- `qsub -l -X -q gpu -l nodes=1:ppn=16,walltime=1:00:00`
- `cd /N/dc2/scratch/<username>/md_6.3.1/src`
- `module swap PrgEnv-cray PrgEnv-pgi`
- `module swap pgi/14.1 pgi/14.3`
- `module load cudatoolkit`
- `module swap cray-libsci/13.0.0 cray-libsci/12.2.0`
- `module swap cray-mpich/7.0.0 cray-mpich/6.2.2`
- `make MDEF=BR2-pgi-cuda iumd_mpi_omp_cuda`
- `cd ../examples/`
- `export OMP_NUM_THREADS=16`
- `aprun -n 1 - N 1 -d 16 -cc none ../src/iumd_mpi_omp_cuda`
Experiment with the GPU Version

• Use different number of nodes

• Observe “aprun” output:
  – Application 2194033 resources: 16 threads (and 1 GPU)
    • utime ~465s, 29.1 seconds total runtime
    • stime ~1s,
    • Rss ~266096, 17 MB RAM per thread
    • inblocks ~12844,
    • outblocks ~9132
Build on Quarry

- cd /N/dc2/scratch/<username>/md_6.3.1/src
- module load intel
- make MDEF=Quarry iumd_omp
- cd ../Quarry/
- qsub Quarry_t008.pbs
Running in a job on BRII

• Go back to Big Red II and run it in a job instead of an interactive shell
Thank You!

Questions?