OpenMP Programming on Intel Xeon Phi Coprocessors: An Early Performance Comparison

Tim Cramer*, Dirk Schmidl*, Michael Klemm°, Dieter an Mey*
*JARA, RWTH Aachen University, Germany
Email: {cramer, schmidl, anmey}@rz.rwth-aachen.de
°Intel Corporation
Email: michael.klemm@intel.com

Rechen- und Kommunikationszentrum (RZ)
Motivation & Content

- Portability of OpenMP applications for Intel Xeon Phi
  - Effort
  - Programming models
  - Performance

- Synchronization overhead of OpenMP

- Performance Modeling
  - Performance evaluation with simple kernel benchmarks and a sparse Conjugate Gradient kernel
  - Applying the Roofline performance model to investigate the utilization

- Comparison with large SMP production systems

1preproduction system
**The Comparison**

**Intel Xeon Phi Coprocessor vs. Bull’s Coherence Switch (BCS)**

**Intel Xeon Phi Coprocessor**
- 1 x Intel Xeon Phi @ 1090 MHz
- 61 Cores
- ~ 1 TFLOPS DP Peak
- 4 Hardware threads per Core
- 8 GB GDDR5 memory
- 512-bit SIMD vectors
- Plugged into PCI Express bus

**BCS**
- 16 x Intel Xeon X7550 @ 2 GHz
- 128 Cores
- 1 TFLOPS DP Peak
- 1 Hardware thread per Core (hyperthreading deactivated)
- up to 2 TB DDR3
- 128-bit SIMD vectors
- 6 HU in a rack

Source: D. Both, Bull GmbH
David vs. Goliath
Bull’s Coherence Switch (BCS)

High-level overview
• 4 boards with 4 sockets connected over xQPI
• 16 sockets in sum
• 2 levels of NUMAness
• 128 Cores

Source: D. Gutfreund, Bull GmbH
Intel Xeon Phi Programming Models

Host only

Language Extension for Offload ("LEO")

Symmetric

Coprocessor only ("native")

main()
foo()
MPI_*()

main()
foo()
MPI_*()

main()
foo()
MPI_*()

Foo()

#pragma offload

Additional Compiler Flag: -mmic

Host CPU

PCI Express bus

Intel Xeon Phi
Stream Memory Bandwidth

**STREAM Results**

- 2 GB memory footprint
- Balanced thread placement achieves best performance on the coprocessor (60 Threads)
- BCS (16 memory controller) has higher bandwidth than the Intel Xeon Phi
- KNC achieves higher bandwidth than 8 Nehalem-EX

<table>
<thead>
<tr>
<th># Threads</th>
<th>Bandwidth in GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
</tbody>
</table>

- BCS, scatter
- BCS, compact
- Coprocessor, balanced
- Coprocessor, compact

156 GB/s
40 %
OpenMP Synchronization Overhead

- No big distinction between the performance with LEO ("offload") / native
- Faster synchronization on MIC for 120 threads compared with 128 on BCS
- Overhead for "pragma offload" 91.1 $\mu$s (3x higher than a "parallel for" with 240 threads)
- These relatively small overheads will not prevent OpenMP applications to scale on Intel Xeon Phi
Conjugate Gradient Method

- **Implementation Variants**
  - Intel Math Kernel Library (MKL)
  - OpenMP
    - first-touch
    - optimal distribution (no static schedule)

- **Runtime Shares of the Linear Algebra Kernels (OpenMP only)**

<table>
<thead>
<tr>
<th>System</th>
<th>#Threads</th>
<th>Serial Time [s]</th>
<th>Parallel Time [s]</th>
<th>daxpy / dpxay</th>
<th>dot product</th>
<th>SMXV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon Phi</td>
<td>244</td>
<td>2387.40</td>
<td>32.24</td>
<td>3.71 %</td>
<td>1.89 %</td>
<td>94.03 %</td>
</tr>
<tr>
<td>BCS</td>
<td>128</td>
<td>1176.81</td>
<td>18.10</td>
<td>11.41 %</td>
<td>4.45 %</td>
<td>84.01 %</td>
</tr>
</tbody>
</table>

- SMXV more dominant on MIC than on BCS
- daxpy ($\hat{y} = a \ast \hat{x} + \hat{y}$) / dpxay ($\hat{y} = \hat{x} + a \ast \hat{y}$) on BCS much higher

- **Testcase**
  - Fluorem/HV15R
  - N=2,017,169, nnz=283,073,458
  - 3.2 GB Memory footprint
Sparse Matrix-Vector Multiplication (1/4)

- **Roofline Model**
  - Using read memory bandwidth $BW$ and theoretical peak performance $P$

- **Model for SMXV** $\tilde{y} = A \ast \tilde{x}$
  - Assumptions
    - $\tilde{x}, \tilde{y}$ can be kept in the cache (~ 15 MB each)
    - $n \ll nnz$
  - Compressed Row Storage (CRS) Format: One value (double) and one index (int) element have to be loaded (dimension $nnz$) → 12 Bytes

- **Operational intensity** $O = \frac{2 \text{ FLOPS}}{12 \text{ Byte}} = \frac{1}{6} \frac{\text{ FLOPS}}{\text{ Byte}}$ (→ memory-bound)

- **Performance Limit:** $L = \min\{P, O \ast BW\}$
Roofline Model BCS (read memory bandwidth 236.5 GB/s)
Sparse Matrix-Vector Multiplication (3/4)

- Roofline Model Xeon Phi (read memory bandwidth 122.1 GB/s)

![Graph showing operational intensity vs. GFLOPS with peak read bandwidth and peak FP performance indicated.]
Sparse Matrix-Vector Multiplication (4/4)

- Reached absolute performance

  - Best results close to theoretical maximum predicted by Roofline
  - BCS: Performance with 128 threads only slightly better than with 64 (also reflected in STREAM measurements)
  - All results on Xeon Phi reached without special tuning (unmodified code)
  - MIC: hyperthreads really help to reach the performance
Conjugate Gradient Method (Scalability)

- **Speedup for 1000 Iterations**

  - OpenMP, BCS: 65 with compact thread placement
  
  - OpenMP, Xeon Phi: 53 without SMT / 74 with all hardware threads (MKL: 79)
  
  - MKL: absolute performance lower on BCS / similar on Intel Xeon Phi
SIMD Vectorization (for SMXV)

- SIMD Vectorization Results (OpenMP implementation)
  - Only small differences
  - For small amount of threads
    speedup of 2
  - With Roofline model:
    Using the complete bandwidth
    is also possible with not
    vectorizable code

Image:
- X-axis: # Threads
- Y-axis: GFLOPS
- Two lines representing KNC, -no-vec and KNC, vectorized
Conclusion & Future Work

- OpenMP Programming on Intel Xeon Phi
  - Easy cross compiling / porting possible
  - Smaller synchronization overhead on MIC than on big SMP machines

- Bandwidth
  - Impressive bandwidth on Intel Xeon Phi (up to 156 GB/s)
  - Higher than on 8 Nehalem-EX

- Productivity / Performance
  - Good performance result with cross-compiling
  - No code modifications, special tuning necessary

- Future Work
  - Performance investigations for distributed memory paradigms
    like MPI on one or multiple coprocessors
Thank you for your attention.