SAR Processing Performance on Cell Processor and Xeon

Mark Backues, SET Corporation
Uttam Majumder, AFRL/RYAS
Summary

- SAR imaging algorithm optimized for both Cell processor and Quad-Core Xeon
  - Cell implementation partially modeled after Richard Linderman work
- Performance between two processors similar
- Cell would generally perform better on a lower complexity problem
  - Illustrated by bilinear interpolation implementation
- Relative performance can be understood from architectural differences
Simple, general purpose SAR imaging implementation
- Order $n^3$ for $n \times n$ pixel image tiles
- Per pulse:
  - 4x oversampled range compression FFT
- Per pulse, per pixel:
  - Single precision range calculation
  - Linear range interpolation
  - Nearest neighbor table lookup for $\frac{4\pi}{c \cdot f_0} R$ phase term

Optimized on both processor types
- SIMD intrinsics for 4x parallelism per processing unit
- Multiple threads
- Loops unrolled to eliminate instruction related stalls
Back-Projection Performance

- Performance on one Intel Quad-Core Xeon 20% faster than on one IBM Cell processor
  - 3.2GHz clock rate
  - Range compression not included in timing analysis
    - Fast compared to projection process
    - Often performed by hardware front-end

- Cell implementation more difficult
  - Explicit DMAs required
  - Use of select, shift, and shuffle intrinsics required for efficient data movement

- Four 3.2GHz Xeon cores equivalent to eight 1.6GHz cores
  - Global memory access not a problem
Back-Projection Range Calculation

- Range calculation accounts for 43% of execution time on Cell processor, and 33% on Xeon
- Square root used in range calculation, for maximum generality
- The square root can be replaced by a much faster approximation
  - $||r|| - ||r-s|| \approx \langle r,s \rangle / ||r||$ when $||s||<<||r||$
  - Other approximations are possible
  - The allowable error is application dependent
- The performance on Cell processor is then closer to the performance on quad-core Xeon
Single Verses Double Precision

- Double precision most important for range calculation
- PS3 Cell double precision instructions very slow
  - 13 cycle latencies with unavoidable 6 cycle stalls
  - Throughput ~6x worse where used
- Double precision comparison would be much less favorable for PS3 Cell
Instruction pipeline differences

- Cell processing element has two pipelines, but only one is for arithmetic instructions
- Xeon has multiple ports and execution units, and can issue two (2-cycle-throughput) instructions per cycle, with data movement often not requiring additional cycles
Why Performance is Not Predicted by Peak GFLOPS Figure (cont.)

- Operation count poorly reflects computational difficulty
  - Transcendental functions are orders of magnitude slower than most other arithmetic operations
  - Efficiency of table lookup depends on instruction set characteristics not reflected by peak performance figure
  - Shuffling of data into registers for efficient SIMD operation can be the slowest part of the process, and is not predicted by operation count

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Time (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D 2345</td>
<td>rotqbyi</td>
<td>$45, $113,8</td>
</tr>
<tr>
<td>0D 3456</td>
<td>shli</td>
<td>$115, $81,4</td>
</tr>
<tr>
<td>1D 345678</td>
<td>lqd</td>
<td>$94,224($sp)</td>
</tr>
<tr>
<td>0D 4567</td>
<td>shli</td>
<td>$113, $55,4</td>
</tr>
<tr>
<td>1D 456789</td>
<td>stqdi</td>
<td>$34,5984($sp)</td>
</tr>
<tr>
<td>0D 5678</td>
<td>shli</td>
<td>$112, $53,4</td>
</tr>
<tr>
<td>1D 567890</td>
<td>lqx</td>
<td>$31, $126, $127</td>
</tr>
<tr>
<td>0D 67</td>
<td>ceqi</td>
<td>$80, $65,0</td>
</tr>
<tr>
<td>1D 678901</td>
<td>lqd</td>
<td>$95,240($sp)</td>
</tr>
<tr>
<td>0D 78</td>
<td>andi</td>
<td>$65, $27,1</td>
</tr>
<tr>
<td>1D 7890</td>
<td>rotqbyi</td>
<td>$15, $105,8</td>
</tr>
<tr>
<td>0D 8901</td>
<td>shli</td>
<td>$105, $45,4</td>
</tr>
<tr>
<td>1D 8901</td>
<td>rotqbyi</td>
<td>$60, $28,8</td>
</tr>
<tr>
<td>0D 90</td>
<td>ceqi</td>
<td>$65, $65,0</td>
</tr>
<tr>
<td>1D 9012</td>
<td>rotqbyi</td>
<td>$52, $19,8</td>
</tr>
</tbody>
</table>

Example Cell processor disassembly and timing analysis
Bilinear Interpolation on Cell and Xeon

- Bilinear affine transformation of 256x256 pixel 8-bit images
  - Vector intrinsics used for both implementations
  - Instruction related stalls eliminated on Cell
  - DMA time still negligible on Cell – no double buffering required
  - Data movement and type conversions required significant optimization on Xeon
  - More difficult programming would be required for Cell to handle images too big to fit in 256KB memory local to each processing unit
  - Order $n^2$ for $n \times n$ pixel images
Xeon Memory Bottleneck vs Cell

Four 3.2GHz Xeon cores have 1.5x the performance of eight 1.6GHz cores
- Front-side bus is 1600MHz vs 1066Mhz
- Main memory access is limiting factor, not computation or cache use

One 3.2GHz IBM Cell processor has 2.4x the performance of one 3.2GHz Intel Quad-Core Xeon
- Data movement much more difficult to program, but much more efficient
Current and Future Work

- **SSE Optimized polar-format**
  - Image warped to fixed coordinates
  - Includes wavefront curvature and other corrections
  - Currently about 7x faster than back-projection implementation, but with limitations

- **Intel Nehalem**
  - On-board memory controller
  - ‘QuickPath’ memory interconnect
  - Up to 8 cores per die

- **Intel Larrabee**
  - 24 x86 cores
  - 4-way multithreading per core
  - SSE
  - 32KB L1 cache, 512KB L2 Cache