Sparse Matrix-Vector Multiply on the Texas Instruments C6678 Digital Signal Processor

Yang Gao and Dr. Jason D. Bakos

Application-Specific Systems, Architectures, and Processors 2013
## TI C6678 vs. Competing Coprocessors

<table>
<thead>
<tr>
<th>Coprocessor</th>
<th>NVIDIA Tesla K20X GPU</th>
<th>Intel Xeon Phi 5110p</th>
<th>TI C66</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak single precision performance</td>
<td>3.95 Tflops/s</td>
<td>2.12 Tflops/s</td>
<td>128 Gflops/s</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>250 GB/s</td>
<td>320 GB/s</td>
<td>12.8 GB/s</td>
</tr>
<tr>
<td>Power</td>
<td>225 W</td>
<td>225 W</td>
<td>10 W</td>
</tr>
<tr>
<td>Primary programming model</td>
<td>CUDA/OpenCL</td>
<td>OpenMP</td>
<td>None, but OpenMP/OpenCL in development</td>
</tr>
</tbody>
</table>
Why the C6678?

- **Unique architectural features**
  - 8 symmetric VLIW cores with SIMD instructions, up to 16 flops/cycle
  - No shared last level cache
  - 4MB on-chip shared RAM
  - L1D and L2 can be configured as cache, scratchpad, or both
  - DMA engine for parallel loading/flushing scratchpads

- **Power efficiency**
  - At 45 nm, achieves 12.8 ideal SP Gflops/Watt
  - Intel Phi [22 nm] is 9.4 Gflops/Watt
  - NVIDIA K20x [28 nm] is 17.6 Gflops/Watt

- **Fast on-chip interfaces for potential scalability**
  - 4 x Rapid IO(SRIO) 2.1: 20 Gb/s
  - 1 x Ethernet: 1 Gb/s
  - 2 x PCI-E 2.0: 10 Gb/s
  - HyperLink: 50 Gb/s
C66 Platforms
Software Pipelining

- VLIW architecture requires explicit usage of functional units
- C66 compiler uses software pipelining to maximize FU utilization
- Conditional prevents SP and lowers utilization

<table>
<thead>
<tr>
<th>Time</th>
<th>Regular Loop</th>
<th>Software Pipelining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Prolog | Kernel | Epilog
--------|--------|--------
1       | 1      | 1      
2       | 2      | 2      
3       | 3      | 3      

ALU1    | ALU2   | ALU3
Sparse Matrices

- We evaluated the C66 relative using a SpMV kernel
  - GPUs achieve only 0.6% to 6% of their peak performance with CSR SpMV
- Sparse Matrices can be very large but contain few non-zero elements
- Compressed formats are often used, e.g. Compressed Sparse Row (CSR)

\[
\begin{pmatrix}
  1 & -1 & 0 & -3 & 0 \\
-2 & 5 & 0 & 0 & 0 \\
0 & 0 & 4 & 6 & 4 \\
-4 & 0 & 2 & 7 & 0 \\
0 & 8 & 0 & 0 & -5
\end{pmatrix}
\]

\[
\begin{align*}
\text{val} & : (1, -1, -3, -2, 5, 4, 6, 4, -4, 2, 7, 8, -5) \\
\text{col} & : (0, 1, 3, 0, 1, 2, 3, 4, 0, 2, 3, 1, 4) \\
\text{ptr} & : (0, 3, 5, 8, 11, 13)
\end{align*}
\]
Sparse Matrix-Vector Multiply

- Code for $y = A\alpha x + \beta y$

```c
row = 0
for i = 0 to number_of_nonzero_elements do
  if i == ptr[row+1] then row=row+1, y[row]*=beta;
  y[row] = y[row] + alpha * A[i] * x[col[i]]
end
```

- Conditional execution
- Reduction
- Low arithmetic intensity (~3 flops / 20 bytes)
- Memory bound kernel
Naïve Implementation

for i = columns assigned to current core

\[\text{val array}\]
\[\text{col array}\]
\[\text{x array}\]

\[\alpha \ast \text{val} \ast \text{x}\]

\[\text{if ptr[row] == i then}\]
\[\text{row} = \text{row} + 1\]
\[y[\text{row}] \leftarrow y[\text{row}] \ast \beta\]
\[\text{end if}\]
\[y[\text{row}] \leftarrow y[\text{row}] + \text{Acc}\]

\[\text{ptr buffer}\]
\[\text{y buffer}\]
\[\text{y write back}\]

\[\text{product results Acc}\]

0.55 Gflops/s

60.4% of cycles were uncovered memory latency
Double Buffer and DMA

- SDRAM buffer
- DMA

Product Loop
- index
- Val L2 buffer
- Col L2 buffer
- x buffer
- \( \alpha \times \text{val} \times x \)

DSP

0.78 Gflops/s
28.8% of cycles were uncovered memory latency
The accumulate loop is **manually unrolled** by 8.

Predicate instructions are applied to replace the if-statements in assembly.

1.63 Gflops/s
50.1% cycles were uncovered memory latency
Loop Fission

**Product Loop**
- Index
- `i`
- `val` buffer
- `col` buffer
- `x` buffer
- `α * val * x`
- Product results

**Accumulate Loop**
- `i`
- `ptr` buffer
- `y` buffer
- `row`
- `Acc ← Acc + prod[i]`
- `if ptr[row] == i then`
  - `row = row + 1`
  - `y[row] ← y[row] * β`
- `end if`

2.08 Gflops/s
36.6% cycles were uncovered memory latency
Adaptive Row Pointer

Acc ← Acc + prod[i]
Acc ← Acc + prod[i+1]
... 
Acc ← Acc + prod[i+K]

Acc ← Acc + prod[i+1]
Acc ← Acc + prod[i+K]
if ptr[row] == i then
  row = row +1
  y[row] ← y[row] * β
end if
# Test Environment

<table>
<thead>
<tr>
<th></th>
<th>i5 650 MKL</th>
<th>GTX680 CUSPARSE</th>
<th>GTX650Ti CUSPARSE</th>
<th>C66</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Clarkdale</td>
<td>Kepler</td>
<td>Kepler</td>
<td>Shannon</td>
</tr>
<tr>
<td>Process (nm)</td>
<td>32</td>
<td>28</td>
<td>28</td>
<td>45</td>
</tr>
<tr>
<td>Memory throughput (GB/s)</td>
<td>21</td>
<td>192.3</td>
<td>86.4</td>
<td>12.8</td>
</tr>
<tr>
<td>TDP (W)</td>
<td>73</td>
<td>195</td>
<td>110</td>
<td>10</td>
</tr>
<tr>
<td>Single precision</td>
<td>26</td>
<td>3090</td>
<td>1425</td>
<td>128</td>
</tr>
<tr>
<td>performance (Gflops)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Power Analyzer

Power Socket
Provide by WT500

PSU with EVM board

110 V

12 V
Matrix

- Tri-diagonal
  
  $\begin{bmatrix}
  2, & 4, & 0, & 0, & 0, & 0 \\
  5, & 4, & 7, & 0, & 0, & 0 \\
  0, & 6, & 2, & 4, & 0, & 0 \\
  0, & 0, & 3, & 10, & 1, & 0 \\
  0, & 0, & 0, & 4, & 6, & 8 \\
  0, & 0, & 0, & 0, & 2, & 12 \\
  \end{bmatrix}$

- N-diagonal
  
  3 - 501

- University of Florida sparse matrix collection

- Matrix Market
SpMV Performance

- N-diagonal Matrix

- Generally, the C66 achieves ~2/3 CPU performance
SpMV Gflops/Watt

- N-diagonal Matrix

- C66 is equivalent to GPUs when $N > 51$
• C66 power efficiency also scales with density for real-world matrices
## Memory Efficiency

The efficiency index, denoted as $AI$, is calculated using the formula:

$$AI = \frac{9 \times \text{rows} \times n + 8 \times \text{rows} + 2}{12(2 \times \text{rows} \times n + n + 2 \times \text{rows} + 1)} \text{ ops/byte}$$

<table>
<thead>
<tr>
<th></th>
<th>Intel i5 650</th>
<th>Nvidia GTX680</th>
<th>Nvidia GTX650Ti</th>
<th>Texas Instruments C6678</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max memory throughput</td>
<td>25.6 Gbytes/s</td>
<td>192.3 Gbytes/s</td>
<td>86.4 Gbytes/s</td>
<td>12.8 Gbytes/s</td>
</tr>
<tr>
<td>Peak computational throughput</td>
<td>9.59 Gflops</td>
<td>72.1 Gflops</td>
<td>32.4 Gflops</td>
<td>4.80 Gflops</td>
</tr>
<tr>
<td>Actual performance</td>
<td>5.89 Gflops</td>
<td>27.8 Gflops</td>
<td>11.0 Gflops</td>
<td>3.9 Gflops</td>
</tr>
<tr>
<td>Memory efficiency</td>
<td>0.61</td>
<td>0.39</td>
<td>0.34</td>
<td>0.81</td>
</tr>
</tbody>
</table>

$N = 151$

rows = 208326

$AI \times 12.8$
Memory Efficiency

![Graph showing Memory Efficiency with various N values and different hardware specifications. The graph includes lines for i5 650, GTX680, GTX650Ti, and C6678.]
Next Generation

- Keystone-II
  - 28 nm
  - Doubles caches
  - Increases memory bandwidth by 125%

<table>
<thead>
<tr>
<th></th>
<th>C66</th>
<th>66AK2H12 “Keystone-II”</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>n/a</td>
<td>4 x ARM A15</td>
</tr>
<tr>
<td>DSP</td>
<td>8 Cores</td>
<td>8 Cores</td>
</tr>
<tr>
<td>DSP L2</td>
<td>512 KB</td>
<td>1024 KB</td>
</tr>
<tr>
<td>DDR3</td>
<td>64 bit</td>
<td>2 x 72 bit</td>
</tr>
<tr>
<td>Process</td>
<td>45 nm</td>
<td>28 nm</td>
</tr>
<tr>
<td>Power</td>
<td>10 W</td>
<td>?</td>
</tr>
</tbody>
</table>
Conclusions

- TI DSP is a promising coprocessor technology for HPC

**Advantages:**

1. Unique architectural features that facilitate automated parallelization (easier to program?)
2. Inherently power efficient microarchitecture
   - Equivalent to modern GPUs and Phi despite older process technology
3. Has advanced memory system for memory bound kernels
   - Simultaneous DMA and caching to match access pattern of individual arrays
4. Has advanced on-chip interfaces for efficient scalability
   - Large-scale multi-DSP platforms already exist

**Looking forward:**

- Keystone II will:
  1. Improve efficiency and memory performance (cache + b/w)
  2. Has onboard host CPUs to facilitate runtimes for multi-DSP scaling
Arithmetic Intensity

\[ AI = \left(\frac{3 \text{ ops}}{8 + \frac{4 \text{ bytes}}{\text{rows}}} + \frac{1}{n} \cdot \frac{2 \text{ ops}}{12 \text{ byts}}\right)/(1 + \frac{1}{n}) \]