Hybrid access-specific software cache techniques for the Cell BE architecture

April 16, 2015
Cell BE architecture

Power Processor Element (PPE)
(Simplified 64 bit PowerPC with VMX)

I/O Controller

I/O Controller

Memory Controller

Memory Controller

RAM

RAM

SPE 1

SPE 2

SPE 3

SPE 4

SPE 5

SPE 6

SPE 7

SPE 8

Dual "configurable" High speed I/O channels
(76.8 GBytes per second in total)

Dual 12.8 GByte per second memory busses give Cell huge memory bandwidth. (25.6 GBytes per second in total)

EIB (Element Interconnect Bus) is the internal communication system.
Traditional cache approach

```
fct1(v1[], v2[], N)
{
    for (i = 0; i < N; i++)
        tmp = v1[i];
        v1[i] = -1;
        v2[tmp]++;
}
```

```
fct1(v1[], v2[], N)
{
    for (i = 0; i < N; i++)
        if (!HIT(h1, &v1[i])
            MAP(h1, &v1[i]);
            REF(h1, &v1[i]) = -1;
            CONSISTENCY(h1, &v1[i]);
        
        if (!HIT(h2, &v2[tmp]))
            MAP(h2, &v2[tmp]);
            REF(h2, &v2[tmp]) =
            REF(h2, &v2[tmp])++;
            CONSISTENCY(h2, &v2[i]);
}
```
Software cache
High locality cache
Transactional cache
C code transformation

a) Label references.

```
for (i = 0; i<N; i++)
  tmp = v1[i];
  v1[i] = -1;
  r1
```

b) High-locality cache transform.

```
i=0;
while (i<N)
  n = N;
  if (!AVAIL(h1, &v1[i]))
    HMAP(h1, &v1[i]);
    n = min(n, i+AVAIL(h1, &v1[i]));
  HCONSISTENCY(n, h1)
  HSYNC(h1);
  for(i; i<n; i++)
    tmp = REF(h1, &v1[i]);
    REF(h1, &v1[i]) = -1;
    v2[tmp]++;
  r1
```

c) Transactional cache transform.

```
for(i; i<2[n/2]; i+=2):
  TINIT();
  tmp = REF(h1, &v1[i]);
  REF(h1, &v1[i]) = -1;
  r1
  GET(h2, &v2[tmp]);
  tmp' = REF(h1, &v1[i+1]);
  REF(h1, &v1[i+1]) = -1;
  r2
  GET(h2', &v2[tmp']);
  TSYNC(h2, h2');
  r1
  tmp = REF(h1, &v1[i]);
  REF(h1, &v1[i]) = -1;
  PUT(h2, &v2[tmp]);
  r2
  GET(h2', &v2[tmp']);
  PUT(h2', &v2[tmp']);
```
Cell benchmark
Cache overhead
Cell vs POWER5 benchmark
Scalability of Cell

Scalability on Cell BE

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>1 SPE</th>
<th>2 SPEs</th>
<th>4 SPEs</th>
<th>8 SPEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MG-A</td>
<td>23.99</td>
<td>12.28</td>
<td>6.42</td>
<td>3.5</td>
</tr>
<tr>
<td>FT-A</td>
<td>72.48</td>
<td>37.88</td>
<td>20.46</td>
<td>10.96</td>
</tr>
<tr>
<td>CG-B</td>
<td>73.74</td>
<td>37.75</td>
<td>20.17</td>
<td>12.25</td>
</tr>
<tr>
<td>IS-B</td>
<td>45.59</td>
<td>24.21</td>
<td>14.11</td>
<td>10.24</td>
</tr>
</tbody>
</table>
Scalability of POWER5

Scalability on Power 5

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>MG-A</th>
<th>FT-A</th>
<th>CG-B</th>
<th>IS-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.86</td>
<td>11.64</td>
<td>24.86</td>
<td>10.25</td>
</tr>
<tr>
<td>2</td>
<td>3.79</td>
<td>6.94</td>
<td>13.20</td>
<td>9.83</td>
</tr>
<tr>
<td>4</td>
<td>3.12</td>
<td>5.61</td>
<td>10.76</td>
<td>8.25</td>
</tr>
</tbody>
</table>
Efficient computation of sum-products on GPUs through software-managed cache

April 16, 2015
Marginalization of a product of functions

1. **Tensor product** $f \otimes g$ is a function $\alpha_{w,x,y,z} \triangleq f_{x,y,z} \times g_{w,x}$.

2. **Marginalization (summation)** over a variable $x$ is a function $\beta_{y,z} \triangleq \sum_{x \in X} f_{x,y,z}$. 
MPF bucketization

\[ \sum_{w,y} f(x, y, z) \otimes g(w, x) \otimes h(w, y) \]

\[
\left( \sum_{y} f(x, y, z) \otimes \left( \sum_{w} g(w, x) \otimes h(w, y) \right) \right)
\]
Computing MPF

\[
\begin{align*}
\text{xyz} & \quad f(x, y, z) \\
000 & \quad f_{000} \\
\ldots... & \quad \ldots \\
112 & \quad f_{112} \\
\hline
\text{wx} & \quad g(w, x) \\
00 & \quad g_{00} \\
\ldots... & \quad \ldots \\
11 & \quad g_{11} \\
\hline
\text{wy} & \quad h(w, y) \\
00 & \quad h_{00} \\
\ldots... & \quad \ldots \\
11 & \quad h_{11} \\
\end{align*}
\]

\[
\begin{align*}
\alpha(x, y, z, w) = & \quad f(x, y, z) \times g(w, x) \times h(w, y) \\
\alpha_{0000} = & \quad f_{000} \times g_{00} \times h_{00} \\
\alpha_{0001} = & \quad f_{000} \times g_{10} \times h_{10} \\
\alpha_{0010} = & \quad f_{001} \times g_{00} \times h_{00} \\
\alpha_{1121} = & \quad f_{112} \times g_{11} \times h_{11} \\
\hline
\text{xz} & \quad k(x, z) = \sum_{w, y} \alpha(x, y, z, w) \\
00 & \quad \alpha_{0000} + \alpha_{0100} + \alpha_{0001} + \alpha_{0101} \\
01 & \quad \alpha_{0010} + \alpha_{0110} + \alpha_{0011} + \alpha_{0111} \\
02 & \quad \alpha_{0020} + \alpha_{0120} + \alpha_{0021} + \alpha_{0121} \\
\ldots... & \quad \ldots \\
12 & \quad \alpha_{1020} + \alpha_{1120} + \alpha_{1021} + \alpha_{1121}
\end{align*}
\]
### MPF access patterns

#### Table 1

<table>
<thead>
<tr>
<th>xyz</th>
<th>( f(x, y, z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Ⓧ Ⓧ</td>
</tr>
<tr>
<td>001</td>
<td>▲ ▲</td>
</tr>
<tr>
<td>002</td>
<td>⭐ ⭐</td>
</tr>
<tr>
<td>010</td>
<td>Ⓧ Ⓧ</td>
</tr>
<tr>
<td>011</td>
<td>▲ ▲</td>
</tr>
<tr>
<td>012</td>
<td>⭐ ⭐</td>
</tr>
<tr>
<td>100</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>110</td>
<td>⚢ ⚢</td>
</tr>
</tbody>
</table>

#### Table 2

<table>
<thead>
<tr>
<th>wx</th>
<th>( g(w, x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>⚢ ▲</td>
</tr>
<tr>
<td>01</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>10</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>11</td>
<td>⚢ ⚢</td>
</tr>
</tbody>
</table>

#### Table 3

<table>
<thead>
<tr>
<th>wy</th>
<th>( h(w, y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>01</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>10</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>11</td>
<td>⚢ ⚢</td>
</tr>
</tbody>
</table>

#### Table 4

<table>
<thead>
<tr>
<th>xz</th>
<th>( k(x, z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>⚢</td>
</tr>
<tr>
<td>01</td>
<td>▲</td>
</tr>
<tr>
<td>02</td>
<td>⚢</td>
</tr>
<tr>
<td>10</td>
<td>⚢</td>
</tr>
</tbody>
</table>

#### Table 5

<table>
<thead>
<tr>
<th>x</th>
<th>z y</th>
<th>( f(x, z, y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>⚢ ⚢</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>▲ ▲</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>▲ ▲</td>
</tr>
</tbody>
</table>

#### Table 6

<table>
<thead>
<tr>
<th>xW</th>
<th>( g(x, w) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>⚢ ⚢ ▲ ▲ ● ● ● ●</td>
</tr>
<tr>
<td>01</td>
<td>⚢ ⚢ ▲ ▲ ● ● ● ●</td>
</tr>
</tbody>
</table>
\[ \sum_{M} \bigotimes_{i} f^i(X^i), \quad M \subseteq \bigcup_{i} X^i, f^i \in F \]

\[ \Psi(O) = \sum_{M} f^1 \bigotimes \cdots \bigotimes f^n \]
Function SumProductKernel
Input: Set of functions $F$, union of functions’ scopes $V$, set of marginalization variables $M \subseteq V$
Output: Function $\Psi$ with scope $O = V \setminus M$

for all configurations $p$ of $O$
    $sum \leftarrow 0$
    for all configurations $m$ of $M$
        $product \leftarrow 1$
        for all functions $f \in F$
            $product \leftarrow product \times f(p, m)$
        end for
        $sum \leftarrow sum + product$
    end for
    $\Psi(p) \leftarrow sum$
end for
return $\Psi$
Arithmetic intensity

\[ A = \frac{\text{compute operations}}{\text{memory operations}} \]
Arithmetic intensity (example 1)

\[ k(x) = f(x) \otimes g(x) \]

\[ A = \frac{1}{3} \]
Arithmetic intensity (example 2)

Matrices $M \times N$ and $N \times K$

\[ A = \frac{2N-1}{2N+1} \]
Arithmetic intensity (example 2 with cache)

Matrices $M \times N$ and $N \times K$

$$A = \frac{2N-1}{N(\frac{1}{M}+\frac{1}{K})+1} = \frac{2-\frac{1}{N}}{\frac{1}{M}+\frac{1}{N}+\frac{1}{K}}$$
Speed

\[ Speed = \min [P, M \times A] \]
Cached arithmetic intensity

\[ A_{\text{cache}} = \frac{\#m - \frac{1}{N}}{\sum_{i}^{m} c_i + \frac{1}{N}} \]
Index vector

\[ < x, z, w, y > \]
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>GPU speedup over CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random benchmark (log-domain)</td>
<td>2708</td>
</tr>
<tr>
<td>Random benchmark (linear-domain)</td>
<td>52</td>
</tr>
<tr>
<td>Bayesian networks (log-domain)</td>
<td>274</td>
</tr>
<tr>
<td>Bayesian networks (linear-domain)</td>
<td>24</td>
</tr>
</tbody>
</table>
Random data performance

![Graph showing performance metrics for GPU and CPU with complexity on the x-axis and performance on the y-axis. The graph illustrates a comparison between GPU and CPU performance with a peak of 53.6 GFLOP/s and ~212 GB/s.]
Random data speedup
Performance by cache size

- 500 MFLOP
- 4000 MFLOP
- 11000 MFLOP
- 19000 MFLOP
- Theoretical 11000 MFLOP

Maximum GPU performance

Cache Size (bytes)

Performance (GFLOP/s)
Performance by # cache pages per thread block
Loop unrolling
Texture cache
Speedup overhead
Speedup with/without overhead

![Graph showing speedup with/without overheads](image)

- Solid line: Speedup not including overheads
- Dotted line: Speedup with overheads included

**X-axis:** Complexity (MFLOP)

**Y-axis:** Speedup (GPU time/CPU time)