RECU: Rochester Elastic Cache Utility

Chen Ding

Professor
Department of Computer Science
University of Rochester

Joint work with Jacob Brock and Chencheng Ye. Publication in NPC 2015 and upcoming in IJPP.
Computer Rentals

• IBM Bluemix
  • instant access, launch quickly, scale with demand

• Amazon AWS
  • burstable (T2), balanced (M3), computer optimized (C4)
    • a user may rent 1, 2, 4 or more vCPUs
  • an on-demand instance
    • fixed price, immediately available
  • a spot instance
    • dynamically assigned to the highest bidder

• Jelastic
  • a unit is a Cloudlet
  • CPU usage measured by the number of CPUs per hour
Cache Memory

- Memory on modern multicores
  - most expensive operations
  - most transistors on-chip

- Multicore cache
  - a mixture of private/shared caches
    - IBM Power 8 512KB private L2, 96MB shared ERAM L3
    - Intel Haswell 256KB private L2, up to 20MB shared L3
  - cache is fast memory “operating system”

- Dilemma of sharing
  - maximize or equalize
  - consistency needed for
    - performance tuning and optimization
### Summary Table

<table>
<thead>
<tr>
<th></th>
<th>G80</th>
<th>GT200</th>
<th>Fermi</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transistors</strong></td>
<td>681 million</td>
<td>1.4 billion</td>
<td>3.0 billion</td>
</tr>
<tr>
<td><strong>CUDA Cores</strong></td>
<td>128</td>
<td>240</td>
<td>512</td>
</tr>
<tr>
<td><strong>Double Precision Floating Point Capability</strong></td>
<td>None</td>
<td>30 FMA ops/clock</td>
<td>256 FMA ops/clock</td>
</tr>
<tr>
<td><strong>Single Precision Floating Point Capability</strong></td>
<td>128 MAD ops/clock</td>
<td>240 MAD ops/clock</td>
<td>512 FMA ops/clock</td>
</tr>
<tr>
<td><strong>Special Function Units (SFUs) / SM</strong></td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td><strong>Warp schedulers (per SM)</strong></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Shared Memory (per SM)</strong></td>
<td>16 KB</td>
<td>16 KB</td>
<td>Configurable 48 KB or 16 KB</td>
</tr>
<tr>
<td><strong>L1 Cache (per SM)</strong></td>
<td>None</td>
<td>None</td>
<td>Configurable 16 KB or 48 KB</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td>None</td>
<td>None</td>
<td>768 KB</td>
</tr>
<tr>
<td><strong>ECC Memory Support</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Concurrent Kernels</strong></td>
<td>No</td>
<td>No</td>
<td>Up to 16</td>
</tr>
<tr>
<td><strong>Load/Store Address Width</strong></td>
<td>32-bit</td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
</tbody>
</table>

---

**Whitepaper**

NVIDIA’s Next Generation CUDA™ Compute Architecture:

**Fermi™**

Chen Ding, University of Rochester, CDP 2015
- An access
  - shorter reuse distance $\rightarrow$ better locality
- An execution window
  - smaller WSS $\rightarrow$ better locality
- A timescale (window length)
  - smaller footprint $\rightarrow$ better locality

$$\begin{align*}
&\infty \quad \infty \quad \infty \quad 3 \quad 3 \quad 3 \quad 3 \\
&\text{a b c a b c c}\text{ a}
\end{align*}$$

$$\begin{align*}
&\infty \quad \infty \quad \infty \quad 1 \quad 2 \quad 3 \\
&\text{a b c c b a a}
\end{align*}$$

$$\begin{align*}
&\begin{array}{c}
3 \\
3 \\
3
\end{array} \\
&\begin{array}{c}
3 \\
3 \\
3
\end{array} \\
&\begin{array}{c}
3 \\
2 \\
2
\end{array}
\end{align*}$$

$$\begin{align*}
\text{fp}(3) &= 3 \\
\text{fp}(3) &= 10/4 = 2.5
\end{align*}$$
High-order Theory of Locality (HOTL)

- Footprint [PPOPP 2008/2011, PACT 2011]
  - average working-set size in time $w$
- Shared cache
  - composable analysis
    - $fp_{p1,p2}(w) = fp_{p1}(w) + fp_{p2}(w)$
- Miss ratio curve [Xiang+ ASPLOS 2013]
  - derivative of the footprint
- Xiaoya was a CAS student while doing this research

\[ fp \]
Optimal Cache Sharing

- Social choice [Xie and Loh, 2008]
  - communist (equal partitioning), capitalist (free-for-all), utilitarian

- Economics/game theory [Zahedi and Lee, 2014]
  - sharing incentive, envy free, Pareto optimal

<table>
<thead>
<tr>
<th>Sharing Only</th>
<th>Partition-Sharing</th>
<th>Partitioning Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Caches</td>
<td>Single Cache</td>
<td>Only</td>
</tr>
<tr>
<td>1. 1 3 2 4</td>
<td>2. 1 4 3 2</td>
<td>3. 1 2 3 4</td>
</tr>
</tbody>
</table>

Chen Ding, University of Rochester, CDP 2015
Optimal Partition Sharing [Brock+ ICPP 2015]

• Footprint theory
  • cache sharing is equivalent to natural cache partitioning

• Optimal partition
  • implies optimal partition sharing

• Dynamic programming
  • to find the optimal partition
  • generalizes previous work
    • Stone et al. 1992, convex miss ratios
    • Suh et al. 2004, piecewise convex
  • supports baseline optimization
Baseline Optimization

- Rochester elastic cache utility (RECU)
  - a way to combine fairness and synergy
- Equal baseline
  - no higher miss ratio than equal partition
- Natural baseline
  - no higher miss ratio than free-for-all sharing
- Elastic baseline
  - no more than x% higher
- RECU
  - equal or natural baseline
  - miss ratio or cache space
Cache Space based Elasticity [NPC’15, IJPP]

- No elasticity
  - equal partition
- 100% elasticity
  - optimal partition
- Intermediate baselines
  - not as effective as miss ratio elasticity
Figure 6: The group miss ratio of the five partitioning methods.

Figure 7: The group miss ratio of Optimal and STTW.

Random Phase Interaction:
We assume that programs interact in their average behavior. Programs may have phases, but their phases align randomly rather than deterministically. An example of synchronized phase interaction is shown at the beginning of the paper in Figure 1. The natural partition does not exist since no cache partition can give the performance of cache sharing. However, synchronized phase interaction is unlikely for independent applications. It is unlikely that they have the same-length phases, and one program always finishes a phase just when a peer program starts another phase. We assume that the phase interaction is random. This assumption is implicit and implicitly validated in Xiang et al., who tested 20 SPEC programs and found that HOTL is acceptable in 99.5% of cases of possible paired running.

Locality-performance Correlation:
A recent study by Wang et al. shows that the HOTL-based miss ratio prediction has a linear relationship between execution time, with a coefficient of 0.938. They measure execution times and miss ratios of all 1820 4-programs co-run groups from a set of 16 SPEC programs. Thus, reducing execution time can be achieved through reducing a portion of miss ratio.

Practicality:
The profiled metrics are average footprint, total number of memory accesses, and solo-run time for each program. Xiang et al. reported on average 23 times slowdown from the full-trace footprint analysis. Wang et al. developed a sampling method called adaptive bursty footprint (ABF) profiling, which takes on average 0.09 second per program. To have reproducible results, our implementation uses the full-trace footprint. We assume that in practice, the data can be collected in real time. While these assumptions do not always hold, they have been carefully studied and validated through experiments on actual systems. Section VII-C gives more details on some of these studies. In this paper, we use these assumptions to develop optimal partition sharing for general programs on any size cache. For hardware cache design, these assumptions may not be adequate, and careful simulation may be necessary. However, our objective in this paper is much narrower and more specific, which is a machine...

- 16 prog. in SPEC 2006, 1820 4-prog groups
- 0.9 sec for sampling per program
- 8MB shared cache, 1024 8KB units
- over 45 billion ways to partition for each group
- 0.2 sec for optimization

Chen Ding, University of Rochester, CDP 2015
### Table 1: Comparison of RECU methods, optimal caching, and free-for-all cache sharing

<table>
<thead>
<tr>
<th>Methods of cache allocation</th>
<th>Overall performance improvement</th>
<th>Individual performance loss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg</td>
<td>Median</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECU with elastic miss ratio baseline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>strict (0%)</td>
<td>6.01%</td>
<td>0%</td>
</tr>
<tr>
<td>5%</td>
<td>52.7%</td>
<td>6.63%</td>
</tr>
<tr>
<td>10%</td>
<td>74.4%</td>
<td>9.32%</td>
</tr>
<tr>
<td>20%</td>
<td>94.2%</td>
<td>12.5%</td>
</tr>
<tr>
<td>50%</td>
<td>108%</td>
<td>16.0%</td>
</tr>
<tr>
<td>100%</td>
<td>115%</td>
<td>22.5%</td>
</tr>
<tr>
<td>alternatives to RECU (2 types of losses: miss ratio and cache space)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>optimal caching</td>
<td>131%</td>
<td>43.7%</td>
</tr>
<tr>
<td>free-for-all sharing</td>
<td>102%</td>
<td>20.5%</td>
</tr>
</tbody>
</table>

#### efficiency

From a provider's perspective, RECU is designed to improve efficiency while guaranteeing a baseline performance. The baseline is the lower bound performance specified by the upper bound on the worst-case degradation compared to equal partitioning. Table 1 shows the overall and individual program performance when using different baselines.

#### quality of service guarantee

Two RECU methods are implicit: 0% cache space baseline is the same as strict, and 100% is the same as optimal.
(a) Miss ratio reduction (positive) vs. increase (negative)

(b) Cache space gain vs. loss
RECU Benefits

• **Cloud provider**
  • baseline optimization
    • 6%, 53%, 108% improvements for 0%, 5%, 50% concession
  • at 50% elasticity
    • half the misses overall
    • close to optimal throughput (108% vs. 131%)
  • at most 50% increase individually
  • free-for-all sharing
    • similar throughput but as high as 32X miss-ratio increase for individual tasks

• **Cloud user**
  • cost saving from greater cloud efficiency
  • bounded increase in miss count