Program Interaction on Shared Cache

Theory and Applications

Chen Ding
Professor
Department of Computer Science
University of Rochester

Illustration: bottlenecks of SPEC2000 on Itanium1

“Nothing travels faster than the speed of light...” Douglas Adams

key problems:
latency/bandwidth
capacity
sharing

Matthew Hertz’s beer
Trishul Chilimbi’s cliff
Chen’s Platform

Cache Performance for SPEC CPU2000 Benchmarks
Version 3.0
May 2003

Jason F. Cantin
Department of Electrical and Computer Engineering
1415 Engineering Drive
University of Wisconsin-Madison
Madison, WI 53706-1691
jcantin@ece.wisc.edu
http://www.jfred.org

Mark D. Hill
Department of Computer Science
1210 West Dayton Street
University of Wisconsin-Madison
Madison, WI 53706-1685
markhill@cs.wisc.edu
http://www.cs.wisc.edu/~markhill


Abstract
The SPEC CPU2000 benchmark suite (http://www.spec.org/osg/cpu2000) is a collection of 26 compute-intensive, non-trivial programs used to evaluate the performance of a computer’s CPU, memory system, and compilers. The benchmarks in this suite were chosen to represent real-world applications, and thus exhibit a wide range of runtime behaviors. On this webpage, we present functional cache miss ratios and related statistics for the SPEC CPU2000 suite. In particular, L1 instruction, L1 data, and L1 unified caches ranging from 1KB to 1MB with 64B blocks and associativities of 1, 2, 4, 8 and full. Prefetch operations were always executed, but results are posted both with and without them counted in the hit ratios. Most of this data was collected at the University of Wisconsin-Madison with the aid of the Simplescalar toolset (http://www.simplescalar.org).
Chen Ding, University of Rochester, PMAM 2014

---

D-cache misses/inst: 1,197,717,058,456 data refs (0.34534--/inst);
782,173,506,477 D-cache 64-Byte block accesses (0.22949--/inst)

<table>
<thead>
<tr>
<th>Size</th>
<th>Direct</th>
<th>2-way LRU</th>
<th>4-way LRU</th>
<th>8-way LRU</th>
<th>Full LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>0.0890418--</td>
<td>0.0762018--</td>
<td>0.0699370--</td>
<td>0.0657938--</td>
<td>0.0652996--</td>
</tr>
<tr>
<td>2KB</td>
<td>0.0651636--</td>
<td>0.0533596--</td>
<td>0.0486152--</td>
<td>0.0462573--</td>
<td>0.0453232--</td>
</tr>
<tr>
<td>4KB</td>
<td>0.0480381--</td>
<td>0.0386862--</td>
<td>0.0353534--</td>
<td>0.0337222--</td>
<td>0.0325938--</td>
</tr>
<tr>
<td>8KB</td>
<td>0.0362358--</td>
<td>0.0290652--</td>
<td>0.0264135--</td>
<td>0.0254564--</td>
<td>0.0245702--</td>
</tr>
<tr>
<td>16KB</td>
<td>0.0277699--</td>
<td>0.0227735--</td>
<td>0.0211365--</td>
<td>0.0204821--</td>
<td>0.0196992--</td>
</tr>
<tr>
<td>32KB</td>
<td>0.0219807--</td>
<td>0.0190920--</td>
<td>0.0181803--</td>
<td>0.0179048--</td>
<td>0.0175964--</td>
</tr>
<tr>
<td>64KB</td>
<td>0.0189635--</td>
<td>0.0166430--</td>
<td>0.0161909--</td>
<td>0.0160494--</td>
<td>0.0159076--</td>
</tr>
<tr>
<td>128KB</td>
<td>0.0158796--</td>
<td>0.0147737--</td>
<td>0.0144648--</td>
<td>0.0143748--</td>
<td>0.0142985--</td>
</tr>
<tr>
<td>256KB</td>
<td>0.0138840--</td>
<td>0.0131826--</td>
<td>0.0130735--</td>
<td>0.0130274--</td>
<td>0.0130001--</td>
</tr>
<tr>
<td>512KB</td>
<td>0.0119997--</td>
<td>0.0115157--</td>
<td>0.0114489--</td>
<td>0.0114018--</td>
<td>0.0113629--</td>
</tr>
<tr>
<td>1MB</td>
<td>0.0101151--</td>
<td>0.0094354--</td>
<td>0.0092640--</td>
<td>0.0093510--</td>
<td>0.0093828--</td>
</tr>
</tbody>
</table>

Compulsory: 0.0000150365--

Benchmarks: 12
Sim Time: 1463.66 days, 4.007 years

---

A Metric and A Tool Box

- Reuse distance
  - independent of coding styles, memory allocation, or hardware
  - possible to correlate between different runs
  - pattern analysis
    - aggregate or temporal
    - cross-program inputs
- Single basis for analysis/optimization
  - to analyze
  - to compose and decompose reuse distance
  - to optimize
    - to shorten long reuse distance

The SLO Tool by Beyls and D’Hollander

- SLO - Suggestions for Locality Optimizations:
  http://slo.sourceforge.net
- An example: 173.APPLU from SPEC 2K

---

Measuring Reuse Distance

- Naive counting, O(N) time per access, O(N) space
  - N is the number of memory accesses
  - M is the number of distinct data elements
  - Too costly
    - N is up to 120 billion, M 25 million

---

Program Locality

Reuse Distance

- Naive counting
  - O(N^2)
- Trace as a stack [IBM'70]
  - O(NM)
- Trace as a vector [IBM'75, Illinois'02]
  - O(NlogN)
- Trace as a tree [LBNL'81], splay tree [Michigan'93], interval tree [Illinois'02]
  - O(NlogM)
- Approximation tree [Rochester'03]
  - O(NloglogM)
- Approx. using time [Rochester'07]
  - O(N)

N is the length of the trace. M is the size of data. C is the size of cache.
Program locality analysis using reuse distance

Full Text: {\urlprefix}To Buy this Article

Authors: Vuyao Zhang \(\text{George Mason University, Fairfax, VA}\)

Aijing Shen \(\text{The College of William and Mary, Williamsburg, VA}\)

Chen Ding \(\text{University of Rochester, Rochester, NY}\)

Published in:

- ACM Transactions on Programming Languages and Systems (TOPLAS)
- TOPLAS Homepage
- Volume 31 Issue 6, August 2009
- ACM New York, NY, USA
- DOI: 10.1145/1552338.1552339

---

Chan Ding, University of Rochester, PMAM 2014

**Analysis Speed**

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>length</th>
<th>data size (64B lines)</th>
<th>unmodified</th>
<th>FP alg</th>
<th>FP alg</th>
<th>RD alg</th>
<th>RD alg</th>
<th>LF alg</th>
<th>LF alg</th>
</tr>
</thead>
<tbody>
<tr>
<td>176.gcc</td>
<td>1.15E+10</td>
<td>3.98E+06</td>
<td>85.1</td>
<td>340</td>
<td>4.1</td>
<td>2.39E2</td>
<td>28.1</td>
<td>5.48E1</td>
<td>65</td>
</tr>
<tr>
<td>181.mct</td>
<td>1.88E+10</td>
<td>2.52E+06</td>
<td>398</td>
<td>1.12E2</td>
<td>2.8</td>
<td>1.05E3</td>
<td>26.4</td>
<td>121.81E</td>
<td>306</td>
</tr>
<tr>
<td>164.gzip</td>
<td>2.00E+10</td>
<td>1.41E+06</td>
<td>150</td>
<td>1.00E2</td>
<td>3.3</td>
<td>5.82E3</td>
<td>38.8</td>
<td>44.37E</td>
<td>296</td>
</tr>
<tr>
<td>252.eon</td>
<td>2.51E+10</td>
<td>1.54E+04</td>
<td>77.4</td>
<td>5.03E2</td>
<td>6.5</td>
<td>5.95E2</td>
<td>76.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256.bzip2</td>
<td>3.20E+10</td>
<td>1.47E+06</td>
<td>173</td>
<td>7.26E2</td>
<td>4.2</td>
<td>7.79E2</td>
<td>45.1</td>
<td>36.42E</td>
<td>211</td>
</tr>
<tr>
<td>175.vpr</td>
<td>3.56E+10</td>
<td>5.08E+04</td>
<td>210</td>
<td>9.64E2</td>
<td>4.6</td>
<td>1.36E5</td>
<td>65.0</td>
<td>51.87E</td>
<td>247</td>
</tr>
<tr>
<td>300.twolf</td>
<td>5.31E+10</td>
<td>3.20E+04</td>
<td>75.5</td>
<td>1.65E3</td>
<td>21.9</td>
<td>1.84E4</td>
<td>249.5</td>
<td>117.47E</td>
<td>1.55E6</td>
</tr>
<tr>
<td>107.parser</td>
<td>1.08E+11</td>
<td>9.47E+04</td>
<td>368</td>
<td>2.07E2</td>
<td>8.1</td>
<td>2.77E5</td>
<td>75.4</td>
<td>155.79E</td>
<td>423</td>
</tr>
<tr>
<td>128.indy</td>
<td>1.21E+11</td>
<td>1.61E+05</td>
<td>230</td>
<td>3.12E2</td>
<td>13.6</td>
<td>3.56E5</td>
<td>104.4</td>
<td>105.62E</td>
<td>462</td>
</tr>
</tbody>
</table>

**Active Sharing (now)**

47 billion accesses
3m16s
3h57m

---

The End of Cache Monopoly

- Multicore
  - desktop, cloud, and handheld
- **Multicore cache**
  - a mixture of private/shared caches
  - Intel Nehalem 256KB private L2, 4MB to 8MB shared L3
  - IBM Power 7 256KB private L2, 32MB shared ERAM L3
  - ERAM to appear on Intel processors
- New problems
  - available cache resource is variable
    - not the full size, not constant size
  - not just performance but also stability
  - not just parallel program but also sequential program

---

The End of Cache Monopoly (by Henry Kautz)

Chen Ding, University of Rochester, PMAM 2014
Footprint

- Example: "abbb"
  - 3 length-2 windows: "ab", "bb", "bb"
  - footprints 2, 1, 1
  - the average fp(2) = (2 + 1 + 1)/3 = 4/3

Footprint Locality

- Private cache locality
  \[ P(\text{capacity miss by me}) = P(\text{my reuse distance} \geq \text{cache size}) \]

- Shared cache locality
  \[ P(\text{capacity miss by me}) = P(\text{my reuse distance} + \text{peer footprint} \geq \text{cache size}) \]

Old Wine in New Bottle?

- Time sharing systems (Multics)
  - memory sharing
  - well studied and solved
  - routine by modern OS

- Cache sharing is more complex
  - hardware managed
  - coffee cup analogy
  - levels, private/shared
  - more frequent access
  - content wiped out in 1ms
  - can’t buy more cache
  - asymmetry/circular feedback

Footprint

\[ \text{footprint} = \text{number of distinct elements accessed in the window} \]
Footprint Measurement 1972 - 2007

- Working set
  - limit value in an infinitely long trace [Denning & Schwartz 1972]
- Direct counting
  - single window size [Thiebaut & Stone TOCS'87]
  - seminal paper on footprints in shared cache
  - same starting point [Agarwal & Hennessy TOCS'88]
- Statistical approximation
  - [Denning & Schwartz 1972; Suh et al. ICS'01; Berg & Hagersten PASS'04; Chandra et al. HPCA'05; Shen et al. POPL'07]
  - level of precision couldn't be directly checked
- No precise definition/solution for all windows
  - can’t be measured for real
  - can’t know the accuracy of an estimate

Footprint Measurement 2008 - 2013

- Footprint distribution
  - all-window enumeration [Ding/Chilimbi PPOPP 2008]
  - max/min/median/percentiles
  - trace compression [Xiang+ PPOPP 11]
  - 70X speedup
  - 4 hours per program
- Average footprint [Xiang+ PACT 11]
  - Xiang formula
  - 22 minutes per program
- Footprint Sampling [Xiang+ ASPLOS 13]
  - shadow profiling
  - 0.5%

Conversion Formulas

The Xiang formula for average footprint [PACT’11]

- rt: reuse time
- m: data size
- n: trace length

\[
fp(x) \approx m - \sum_{k=x+1}^{n-1} (k - x) P(rt = k)
\]

\[
mr(c) = mr(fp(x)) = \frac{fp(x+\Delta x) - fp(x)}{\Delta x}
\]

\[
P(rd = c) = mr(c - 1) - mr(c)
\]
Reality Check

- 20 SPEC 2006 programs
- 190 different pair runs

Modeling
- per program footprint
- composition
- a few hours
- prediction for all cache sizes

Exhaustive parallel testing
- 190 pair runs
- 380 hw counter reads (OFFCORE_DATA_IN, 8MB 16-way L3)
- ~9 days total CPU time

Co-run interference of libquantum;
high miss ratio, zero sensitivity;
measured miss ratio 17.82% to 17.89%,
predicted 17.94% to 17.94%

Co-run interference of gamess;
low miss ratio, high sensitivity
measured miss ratio 0.0002% to 0.04%
predicted 0.000013% to 0.03%

Denning’s Law of Locality

What’s the relation between reuse frequency and footprint?

abc ... abc ...

aaa ... bbb ...

Limit value [Denning and Schwartz, CACM 1972]
Time space [Denning and Slutz, CACM 1978]
All program traces [Rochester, ASPLOS 2013]
An Old Open Question

How quickly can we measure the miss rate for all cache size?

3000+ Cache Sizes In the analysis, the footprint and reuse distance numbers are bin-ed using logarithmic ranges as follows. For each power-of-two range, we sub-divide it into 256 equal-size increments. As a result, we can predict the miss ratio not just for power-of-two cache sizes, but 3073 cache sizes between 16KB and 64MB.

Xiang et al. ASPLOS 13 (Tongxin Bai's tool)

Miss Ratio vs Pressure, 32KB Cache

Miss Ratio vs Pressure, 4MB Cache
An Old Open Question

Is there a machine independent way to compare program behavior in shared cache? How do programs in different domains differ?

Collaborative Rationing

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>a b c a b c a b c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hint Bit</td>
<td>0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>Access Bit</td>
<td>1 0 1 0 1 0 1 1</td>
</tr>
<tr>
<td>Misses</td>
<td>M M M M M M M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread 2</th>
<th>x y z x y z x y z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hint Bit</td>
<td>0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>Access Bit</td>
<td>1 0 1 0 1 0 1 1</td>
</tr>
<tr>
<td>Misses</td>
<td>M M M M M M M</td>
</tr>
</tbody>
</table>

Two threads, each accessing three elements and using two-element cache. Best per thread and overall cache utilization --- 50% miss rate for each program.

On-going Studies

Shared Footprint Analysis

with Hao Luo and Pengcheng Li

An Old Open Question

Does LRU cache produce optimal partition? [Thiebuat and Stone, 1992]

The second type of sharing happens between the instruction and the data of a program. Stone et al. [1992] investigated whether LRU produces the optimal allocation. Assuming that the miss rate functions for instruction and data are continuous and differentiable, the optimal allocation happens at the points “when miss-rate derivatives are equal” [Thiebuat and Stone, 1992]. The miss rate functions, one for instruction and one for data, were modeled instead of measured. The authors showed that LRU is not optimal, but left open a question whether there is a bound on how close LRU allocation is to optimal.

To compute footprints and their overlap, Thiébaut and Stone [1987] assumed a bound on how close LRU allocation is to optimal. The pressure model in Chapter 4 can be used to compute the footprints and thereby answer the open question for any group of programs.

Maximal cache performance?

Answer: LRU-MRU (Gu) distance

Answer: Miss rate in all cache sizes?

[Gu et al. ISMM 2012, Rochester Dissertation 2013]
Recent Developments

• Competitiveness, politeness, sensitivity
  • Jiang et al. [TPDS’11, HiPEAC’10]
• Intensity and sensitivity
  • Zhurovlev et al. [ASPLOS’10]
• Niceness, pressure and sensitivity
  • Mars et al. [CGO’12, Micro’12]
• Interference of cache
  • composable models [Stone+ TOCS’87/TOC’92; Suh+ ICS’01; Chandra+ HPCA’05; Xiang+ PPopp’11/PACT’11/ASPLOS’13]
  • threaded code [Ding/Chilimbi MSR’09, Jiang+ CC’10/TPDS’12, Schuff+ PACT’10, Wu/Yeung PACT’11/ISCA’13]
  • interference model of execution time/speed
  • bubble-up [Mars+ Micro’12, ISCA’13]
  • QoS-aware scheduling [Delimitrou/Kozyrakis ASPLOS’13]

Recent Developments [cont’d]

• Parallel reuse distance measurement
  • cluster [OSU, IPDPS 2012]
  • GPU [ICT and NCSU, IPDPS 2012]
  • sampling
    • footprint shadow sampling [Rochester, ASPLOS 2013]
    • multicores reuse distance [Purdue, PACT 2010]
    • reuse distance sampling [Chang & Zhong, PACT 2008]
• Reuse distance in threaded code
  • multicores reuse distance [Purdue, PACT 2010]
  • CRD/PRD scaling [Maryland, ISCA 2013, to appear]

Recent Developments (cont’d)

• Asymptotic locality effect in parallel algorithms
  • Leslie Valiant, PACT 2011 keynote
  • Guy Blelloch et al. CMU, MIT, Intel Labs Pittsburgh [MSPC 2013]
  • Morris Herlihy and student, [PPOPP 2014]
• Shared footprint [Rochester, WODA 2013]
• Static reuse distance analysis in Matlab [Indiana, ICS 2010]
• Static footprint analysis [Rochester, CGO 2013]
  • peer-aware program optimization [Bao, dissertation’13]
• Collaborative caching
  • practical uses [UT, Ghent, Google etc]
  • optimal collaborative LRU cache [Gu, ISMM’11/12/13, dissertation’13]

Summary

• Program interaction in multicore
  • data sharing in threaded code
  • cache and memory bandwidth sharing by all programs
• Locality theory
  • working set, footprint, shared footprint
  • metrics composition and conversion
  • higher order theory of cache locality (HOTL)
• Recent research
  • locality in parallel algorithms
  • peer-aware program optimization
  • sharing conscious task scheduling
  • collaborative caching