Fast Instruction Cache Performance Evaluation

Using Compile-Time Analysis

by

David B. Whalley

Florida State University
Motivation

- traditional trace-driven cache evaluations are time-consuming

- past attempts to speed up cache evaluations either
  
  — require an initial pass by a filter to reduce the length of trace

  — or use only a sampling of the references which results in a loss of accuracy
Overview

• instruction addresses (unlike many data references) do not change during execution

• use compile-time analysis to avoid simulating instruction references that

  — will be cache hits

  — and will not change the state of the cache

• seven techniques (A-G) are described
Technique A

- call trace routine as enter each basic block

```
...  
L142: pea #145  /* push block number 145 */  
jbsr _traceblknum  /* call trace routine */  
addq1 #4,a7  /* adjust stack pointer */  
...  
jne L67  
pea #146  /* push block number 146 */  
jbsr _traceblknum  /* call trace routine */  
addq1 #4,a7  /* adjust stack pointer */  
...
```

- trace routine invokes cache simulator for each instruction in the block
Technique B

- assembly code same as Technique A
- trace routine treats entire block as one big instruction when a context switch cannot occur during the execution of the block
- number of references equal to number of lines referenced
- remaining references are hits due to spatial locality
Technique C

- trace routine only invoked on a taken branch

<table>
<thead>
<tr>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>jbsr _foo</td>
<td>jbsr _foo</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>jne L74</td>
<td>jne LN10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>jeq L78</td>
<td>jeq LN11</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>jra L67</td>
<td></td>
</tr>
<tr>
<td>pea #17</td>
<td></td>
</tr>
<tr>
<td>jbsr _traceblknum</td>
<td>* call trace routine */</td>
</tr>
<tr>
<td>addql #4,a7</td>
<td>* adjust stack pointer */</td>
</tr>
<tr>
<td>movl #32,_startblk</td>
<td>* reset start block */</td>
</tr>
<tr>
<td>jra L67</td>
<td>jra L74</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>LN10: pea  #15</td>
<td>* push last block in seq */</td>
</tr>
<tr>
<td>jbsr _traceblknum</td>
<td>* call trace routine */</td>
</tr>
<tr>
<td>addql #4,a7</td>
<td>* adjust stack pointer */</td>
</tr>
<tr>
<td>movl #22,_startblk</td>
<td>* reset start block */</td>
</tr>
<tr>
<td>jra L74</td>
<td>jra L78</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>LN11: pea  #16</td>
<td>* push last block in seq */</td>
</tr>
<tr>
<td>jbsr _traceblknum</td>
<td>* call trace routine */</td>
</tr>
<tr>
<td>addql #4,a7</td>
<td>* adjust stack pointer */</td>
</tr>
<tr>
<td>movl #26,_startblk</td>
<td>* reset start block */</td>
</tr>
<tr>
<td>jra L78</td>
<td>jra L78</td>
</tr>
</tbody>
</table>
Requirements for Techniques D-G

- compiler reads in line size and minimum number of cache sets

- if number of sets not decreased and line size stays the same
  
  — can obtain measurements for a different cache configuration with the same executable
Technique D

- if the compiler finds that a loop with no calls fits in cache
  — markers (array elements) associated with loop blocks are cleared in preheader
  — each loop block is checked if currently in cache

```
crl1 _marker+952    /* clear marker[238] */
movl #238,_lowmarker /* first block cleared on switch */
movl #238,_highmarker /* last block cleared on switch */
L405:  cmpl #0,_marker+952 /* check if block 238 is in cache */
       jeq LN191 /* if not then invoke simulator */
       cmpl #9992,_c_switch /* check if switch in 8 cycles */
       jge LN191 /* if so then invoke simulator */
       addql #8,_c_switch /* adjust switch information */
       addql #8,_c_hits /* adjust number of cache hits */
LN192: ...
       jne L405
       movl #-1,_highmarker    /* out of loop so nothing to clear */

LN191: movl #1,_marker+952    /* block 238 is now in cache */
       movl #238,_startblk    /* first block in sequence */
       pea #238                /* last block in sequence */
       jbsr _traceblknum      /* call trace routine */
       addql #4,a7             /* adjust stack pointer */
       jra LN192               /* execute insts in block 238 */
```
Technique E

• similar to Technique D except that
  — a call graph is built from information supplied from first compilation pass
  — if entire function and routines it can invoke fit in cache
    • use Technique D throughout the function
  — markers are cleared before calls to functions that fit in cache
  — for loops with calls that fit in cache
    • markers of blocks in the loop and the routines that can be invoked from the loop are cleared as well in the preheader of the loop
Assembly Code Example for Technique E

```assembly
cr11l _marker+952    /* clear marker[238] */
cr11l _marker+956    /* clear marker[239] */
cr11l _marker+1056   /* clear marker[264] */
cr11l _marker+1060   /* clear marker[265] */
cr11l _marker+1064   /* clear marker[266] */
movl #238,_lowmarker /* first block cleared on switch */
movl #266,_highmarker /* last block cleared on switch */
L405: cmpl #0,_marker+952    /* check if block 238 is in cache */
      jeq   LN191       /* if not then invoke simulator */
      cmpl #9995,_c_switch /* check if switch in 5 cycles */
      jge   LN191       /* if so then invoke simulator */
      addql #5,_c_switch /* adjust context switch info */
      addql #5,_c_hits  /* adjust number of cache hits */
LN192: ...
      jbsr _foo
      cmpl #0,_marker+956 /* check if block 239 is in cache */
      jeq   LN193       /* if not then invoke simulator */
      cmpl #9997,_c_switch /* check if switch in 3 cycles */
      jge   LN193       /* if so then invoke simulator */
      addql #3,_c_switch /* adjust context switch info */
      addql #3,_c_hits  /* adjust number of cache hits */
LN194: ....
      jne  L405
      movl #-1,_highmarker /* out of loop so nothing to clear */
```
Technique F

• avoids references of loops that do not fit in cache if
  — at least one half of blocks in loop and routines invoked from the loop fit in cache
  — and each function invoked directly from the loop and the set of functions they could invoke fit in cache
Assembly Code Example for Technique F

- clear markers of conflicting blocks when a block in a routine invoked from the loop conflicts with
  - a block in the loop
  - or a block from another routine invoked from the loop

L85:

```assembly
...  
cmpl #0,_marker+2120  /* check if block 530 is in cache */  
jeq LN141  /* if not then invoke simulator */  
cmpl #9981,_c_switch  /* check if switch in 19 cycles */  
jge LN141  /* if so then invoke simulator */  
addl #19,_c_switch  /* adjust context switch info */  
addl #19,_c_hits  /* adjust number of cache hits */  
LN142: clrl _marker+7344  /* clear block 1836 in foo */  
...  
clrl _marker+2120  /* clear block 530 in loop */  
jbcr _foo  
...  
jne L85
```
Technique G

- potential temporal locality when a routine is invoked from more than one location

- only clear markers that have lines that could be replaced

```asm
clrl _marker+528  /* clear marker[132] */
clrl _marker+532  /* clear marker[133] */
clrl _marker+536  /* clear marker[134] */
clrl _marker+540  /* clear marker[135] */
movl #132,_lowmarker /* first block cleared on switch */
movl #135,_highmarker /* last block cleared on switch */
jbsr _foo
...
/* insts conflicting with block 132 */
clrl _marker+528 /* clear marker[132] */
jbsr _foo
movl #-1,_highmarker /* out of func so nothing to clear */
```
Test Set

• all cache simulations with
  — direct-mapped organization
  — 16 byte line size
  — periodic context switches

<table>
<thead>
<tr>
<th>Name</th>
<th>Description or Emphasis</th>
<th>Size in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>compact</td>
<td>Huffman Coding Compression</td>
<td>4322</td>
</tr>
<tr>
<td>cpp</td>
<td>C Preprocessor</td>
<td>12678</td>
</tr>
<tr>
<td>diff</td>
<td>Differences between Files</td>
<td>9166</td>
</tr>
<tr>
<td>lex</td>
<td>Lexical Analyzer Generator</td>
<td>26318</td>
</tr>
<tr>
<td>sed</td>
<td>Stream Editor</td>
<td>13946</td>
</tr>
<tr>
<td>sort</td>
<td>Sort or Merge Files</td>
<td>5500</td>
</tr>
<tr>
<td>tbl</td>
<td>Table Formatter</td>
<td>24592</td>
</tr>
<tr>
<td>yacc</td>
<td>Yet Another Compiler-Compiler</td>
<td>22392</td>
</tr>
</tbody>
</table>
Calls to Simulator with Cache Sizes from 2K to 16K

- As cache size increases
  - hit ratio increases
  - calls to simulator for Techniques E, F, and G decrease

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Hit Ratio</th>
<th>Relative to Technique A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>E</td>
</tr>
<tr>
<td>2K</td>
<td>97.88%</td>
<td>6.79%</td>
</tr>
<tr>
<td>4K</td>
<td>98.49%</td>
<td>5.11%</td>
</tr>
<tr>
<td>8K</td>
<td>98.98%</td>
<td>3.53%</td>
</tr>
<tr>
<td>16K</td>
<td>99.16%</td>
<td>1.91%</td>
</tr>
</tbody>
</table>
Execution Time Overhead with Cache Sizes from 2K to 16K

- As cache size increases
  - overheads for Techniques E, F, and G decrease

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Ratio to Execution Time without Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td>2K</td>
<td>19.27</td>
</tr>
<tr>
<td>4K</td>
<td>15.09</td>
</tr>
<tr>
<td>8K</td>
<td>11.59</td>
</tr>
<tr>
<td>16K</td>
<td>9.36</td>
</tr>
</tbody>
</table>
Future Work

- can reduce number of trace instructions executed when assume no context switches
  - block markers now contain number of outstanding hits for that block
  - each marker first set to negated number of references for the block
  - in loops add number of references to marker
  - after loop update cache hits and misses

L405: addql #8,_marker+952 /* adjust number of hits for block 238 */
... 
jne L405
cmp1 #0,_marker+952 /* check if block 238 was executed */
jlt LN191 /* if not then skip over trace code */
movl #238,_startblk /* first block in sequence */
pea #238 /* last block in sequence */
jbsr _traceblknum /* call trace routine */
addql #4,a7 /* adjust stack pointer */
movl _marker+952,d0 /* load remaining hits for block 238 */
addl d0, _c_hits /* adjust number of cache hits */
movl #-8,_marker+952 /* reset hits for block 238 */
LN191: ...
Execution Time Overhead
with Cache Sizes from 2K to 16K
and No Context Switches

- As cache size increases
  - more instructions are in loops that fit in cache
  - and fewer trace instructions are executed

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>2K</td>
<td>12.87</td>
<td>11.53</td>
<td>11.61</td>
</tr>
<tr>
<td>4K</td>
<td>9.78</td>
<td>8.42</td>
<td>8.38</td>
</tr>
<tr>
<td>8K</td>
<td>6.05</td>
<td>5.97</td>
<td>5.97</td>
</tr>
<tr>
<td>16K</td>
<td>2.55</td>
<td>2.19</td>
<td>2.16</td>
</tr>
</tbody>
</table>
Conclusions

- techniques significantly reduce execution overhead with no loss of accuracy

- Technique D is particularly attractive since it
  - is simple to implement
  - still results in a significant improvement
Trace Routine for Technique A

- allows context switches to occur between instructions within a basic block

```c
void traceblknum(blk)
int blk;
{
    register int i;

    /* Invoke the cache simulator for each instruction in the block. */
    for (i = blkinfo[blk].first;
         i < blkinfo[blk].first+blkinfo[blk].numinst; i++)
        cachesim(instsize[i], instaddr[i]);
}
```
Trace Routine for Technique B

- treats entire basic block as one big instruction

```c
void traceblknum(blk)
int blk;
{
    register struct blkinfotype *p;
    int i, j;

    /* If a context switch cannot occur during the execution of the block then invoke the cache simulator once for the entire block. */
    p = &blkinfo[blk];
    if (!(c_switch + p->worst > SWITCHTIME)) {
        i = c_hits + c_misses;
        cachesim(p->size, instaddr[p->first]);
        c_hits += (j = (p->refs - ((c_hits + c_misses) - i)));
        c_switch += j*HITTIME;
    }

    /* Else invoke the cache simulator for each instruction in the block. */
    else
        for (i = p->first; i < p->first+p->numinst; i++)
            cachesim(instsize[i], instaddr[i]);
}
```
Trace Routine for Technique C

```c
void traceblknum(blk)
    int blk;
{
    register int i, refs, size, worst;
    int j;

    /* Sum information about the set of consecutive blocks. */
    refs = size = worst = 0;
    for (i = startblk; i <= blk; i++) {
        refs += blkinfo[i].refs;
        size += blkinfo[i].size;
        worst += blkinfo[i].worst;
    }

    /* If a context switch cannot occur during the execution of the set
    of blocks then invoke the cache simulator once for the entire set. */
    if (!(c_switch + worst > SWITCHTIME)) {
        i = c_hits + c_misses;
        cachesim(size, instaddr[blkinfo[startblk].first]);
        c_hits += (j = (refs - ((c_hits + c_misses) - i)));
        c_switch += j*HITTIME;
    }

    /* Else invoke the cache simulator for each instruction in the set of blocks. */
    else
        for (i = startblk; i <= blk; i++)
            for (j = blkinfo[i].first;
                j < blkinfo[i].first+blkinfo[i].numinst; j++)
                cachesim(instsize[j], instaddr[j]);
}
Trace Routine for Technique D-G

- same trace routine used for Techniques D-G

- similar to trace routine for Technique C except that a set of markers are cleared on a context switch

```c
void traceblknum(blk)
int blk;
{
    ...
    /* Else invoke the cache simulator for each instruction in the set of blocks. */
    else
    for (i = startblk; i <= blk; i++)
        for (j = blkinfo[i].first;
            j < blkinfo[i].first+blkinfo[i].numinst; j++) {
            cachesim(instsize[j], instaddr[j]);

            /* Clear set of block markers if context switch occurred. */
            if (old_context_switch < context_switch) {
                old_context_switch = context_switch;
                for (k = lowmarker; k <= highmarker; k++)
                    blkmarker[k] = 0;
            }
        }
    ...
}
```