Lazy Spilling for a Time-Predictable Stack Cache: Implementation and Analysis

Sahar Abbaspour\textsuperscript{1} Alexander Jordan\textsuperscript{1} Florian Brandner\textsuperscript{2}

Embedded Systems Engineering Section\textsuperscript{1} Computer Science and System Engineering Department\textsuperscript{2}
DTU Compute ENSTA-ParisTech
Technical University of Denmark
\{sabb, alejo\}@dtu.dk
florian.brandner@ensta-paristech.fr
Introduction

- Memory accesses are crucial for performance
  - Applies to time-predictable systems as well
  - Analyzable memory and cache designs
- Stack Cache
  - Specialized cache dedicated to stack data: Stack Cache
  - No need for precise knowledge about the addresses of accesses
  - Data transfers between the stack cache and the main memory
Stack Cache Background

- Two pointers: stack top (ST) and memory top (MT)
- Reserve: Stack frames are allocated upon entering a function: potential spilling
- Free: Stack frames are freed immediately before returning from a function
- Ensure: Compiler ensures a valid stack cache state: potential filling

- Initial conditions
  - Initial pointers: $ST = MT = 256$ and $|SC| = 8$
  - Stack cache instructions are 2 words (8 bytes)
  - Invariant: $|MT - ST| \leq |SC|$
Stack Cache Example

```c
function bar()
    sres 2
    sws [1] = ...
loop :
    lws ... = [1]
call foo
    sens 2
    cmp ...
    bt loop // jump to beginning of the loop
    sfree // exit function
    ret
```

Invariant: $|MT - ST| \leq |SC|$

Reserve

$ST = 248$
Stack Cache Example

1  function bar()
2    sres 2
3    sws [1] = ...
4  loop :
5      lws ... = [1]
6    call foo
7    sens 2
8      cmp ...
9      bt loop // jump to beginning of the loop
10   sfree // exit function
11  ret

Invariant:  \(|MT - ST| \leq |SC|\)

Call
\(ST = 216\) and \(MT = 248\)
Stack Cache Example

1 function bar()
2 sres 2
3 sws [1] = ...
4 loop :
5 lws ... = [1]
6 call foo
7 sens 2
8 cmp ...
9 bt loop // jump to beginning of the loop
10 sfree // exit function
11 ret

Invariant: $|MT - ST| \leq |SC|$

Ensure
$MT = 256 \quad ST = 216$

$MT - ST = 40$: Occupancy: 10 words
Stack Cache Example

```
1  function bar()
2    sres 2
3    sws [1] = ...
4  loop :
5    lws ... = [1]
6    call foo  
7    sens 2 
8    cmp ...
9    bt loop // jump to beginning of the loop
10   sfree // exit function
11  ret
```

Invariant: $|MT - ST| \leq |SC|$

**Free**

$ST = MT = 256$
Lazy Pointer

- Keeps track of the coherent elements of the stack cache and the main memory
- Invariant: $ST \leq LP \leq MT$
- LP divides the reserved space in the stack cache
  - Between ST and LP: effective occupancy
  - Between LP and MT: coherent data
- The effective occupancy area contains potentially modified data
- Store instruction writes to an address above the LP, some data potentially becomes incoherent
Implementation

Invariant: $ST \leq LP \leq MT$

Reserve

- Spills blocks between ST and LP
- No need to spill already spilled data: LP potentially moves down with MT
- MT and ST updated as in the original reserve

Free

- LP potentially moves down with ST (to satisfy the invariant)
- MT and ST updated as in the original free
Implementation

Ensure

- No change is needed

Store

- Potentially modifies the LP
  - Effective address of the store instruction
  - Current value of LP
  - LP might move up
function bar() 
sres 2 
sws[1] = ... 
loop:
lws ... = [1]
call foo 
sens 2 
cmp ... 
bt loop 
sfree 
ret

▶ Initial pointers:  $LP = ST = MT = 256$ and $|SC| = 8$
▶ Stack cache instructions work on 2 words arguments (8 bytes)

Reserve

$LP = ST = 248$
Example

1 function bar()
2 sres 2
3 sws[1] = ...
4 loop :
5   lws ... = [1]
6   call foo
7   sens 2
8   cmp ...
9   bt loop
10  sfree
11  ret

▶ Initial pointers: \( LP = ST = MT = 256 \) and \( |SC| = 8 \)
▶ Stack cache instructions work on 2 words arguments (8 bytes)

Store

\( LP = 252 \)
Example

```c
function bar()
    sres 2
    sws[1] = ...
loop :
    lws ... = [1]
call foo
    sens 2
cmp ...
btf loop
sfree
ret
```

- Initial pointers: $LP = ST = MT = 256$ and $|SC| = 8$
- Stack cache instructions work on 2 words arguments (8 bytes)

Call

- $MT = 248$
- Replacement of entire cache: $ST = LP = 216$
Example

```c
1 function bar()
2    sres 2
3    sws[1] = ...
4 loop :
5       lws ... = [1]
6    call foo
7       sens 2
8    cmp ...
9    bt loop
10   sfree
11   ret
```

- Initial pointers: \( LP = ST = MT = 256 \) and \( |SC| = 8 \)
- Stack cache instructions work on 2 words arguments (8 bytes)

Ensure

- \( MT = 256 \): Occupancy 2 words
- \( LP = ST = 248 \): Effective Occupancy: 0 words
Two Problems

▶ Worst-case filling of ensure instructions (Ensure Analysis)
  ▶ Filling behavior of ensure instructions is not affected by lazy spilling

▶ Worst-case spilling of reserve instructions (Reserve Analysis)
  ▶ Function-local data-flow analysis
    ▶ Original SC: Utilized stack cache occupancy: $MT - ST$
    ▶ Lazy spilling SC: Smaller region $LP - ST$: smaller effective occupancy
  ▶ Store instructions
  ▶ Inter-procedural analysis on the call graph
Static Analysis

- Simple data-flow analysis
- Computing worst-case effective occupancy before each call
- Propagating maximum effective occupancy among instructions
- Computing context-sensitive bounds of the spilling for each function
Experiment Setup

- Patmos processor
- Mibench benchmarks
- Patmos LLVM compiler with full optimizations (-O3)
- Standard data cache with original stack cache and Lazy spilling stack cache
Experiments: Average Measurement

- Lazy Spilling Cache Performance
- Miss rate is not suitable for measurement
- \[ \frac{\#RD + \#WR}{\#Stalls} \]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SC_128</th>
<th>LP_128</th>
<th>SC_256</th>
<th>LP_256</th>
<th>DC</th>
<th>SC_128</th>
<th>LP_256</th>
<th>DC</th>
<th>SC_256</th>
<th>LP_256</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicmath-tiny</td>
<td>2.3</td>
<td>4.0</td>
<td>26.4</td>
<td>34.0</td>
<td></td>
<td>2.3</td>
<td>4.0</td>
<td>26.4</td>
<td>34.0</td>
<td>34.0</td>
<td></td>
</tr>
<tr>
<td>bitcns</td>
<td>4.6</td>
<td>12.2</td>
<td>17054.7</td>
<td>6154.4</td>
<td>1.0</td>
<td>4.6</td>
<td>12.2</td>
<td>17054.7</td>
<td>6154.4</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>jpeg-small</td>
<td>116.9</td>
<td>148.4</td>
<td>3470.7</td>
<td>6154.4</td>
<td>1.0</td>
<td>116.9</td>
<td>148.4</td>
<td>3470.7</td>
<td>6154.4</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>crc-32</td>
<td>9.0</td>
<td>28.7</td>
<td>18562.0</td>
<td>18562.0</td>
<td></td>
<td>9.0</td>
<td>28.7</td>
<td>18562.0</td>
<td>18562.0</td>
<td></td>
<td>9.0</td>
</tr>
<tr>
<td>csusan-small</td>
<td>11.3</td>
<td>18.6</td>
<td>1218.8</td>
<td>1430.0</td>
<td>2.3</td>
<td>11.3</td>
<td>18.6</td>
<td>1218.8</td>
<td>1430.0</td>
<td>2.3</td>
<td>1.5</td>
</tr>
<tr>
<td>dbf</td>
<td>477.4</td>
<td>623.0</td>
<td>433.7</td>
<td>433.7</td>
<td></td>
<td>477.4</td>
<td>623.0</td>
<td>433.7</td>
<td>433.7</td>
<td>433.7</td>
<td></td>
</tr>
<tr>
<td>dijkstra-small</td>
<td>19.5</td>
<td>32.8</td>
<td>335.2</td>
<td>433.7</td>
<td>1.4</td>
<td>19.5</td>
<td>32.8</td>
<td>335.2</td>
<td>433.7</td>
<td>433.7</td>
<td>1.4</td>
</tr>
<tr>
<td>djjpeg-small</td>
<td>9.0</td>
<td>13.5</td>
<td>293.4</td>
<td>361.5</td>
<td>0.8</td>
<td>9.0</td>
<td>13.5</td>
<td>293.4</td>
<td>361.5</td>
<td>361.5</td>
<td>0.8</td>
</tr>
<tr>
<td>di.jsndael</td>
<td>15.8</td>
<td>28.7</td>
<td>18562.0</td>
<td>18562.0</td>
<td></td>
<td>15.8</td>
<td>28.7</td>
<td>18562.0</td>
<td>18562.0</td>
<td></td>
<td>15.8</td>
</tr>
<tr>
<td>ebf</td>
<td>172.5</td>
<td>224.6</td>
<td>25834.0</td>
<td>25834.0</td>
<td>0.9</td>
<td>172.5</td>
<td>224.6</td>
<td>25834.0</td>
<td>25834.0</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>erjndael</td>
<td>32.6</td>
<td>43.3</td>
<td>25834.0</td>
<td>25834.0</td>
<td>0.9</td>
<td>32.6</td>
<td>43.3</td>
<td>25834.0</td>
<td>25834.0</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>esusan-small</td>
<td>15.9</td>
<td>25.3</td>
<td>70.7</td>
<td>139.5</td>
<td>3.6</td>
<td>15.9</td>
<td>25.3</td>
<td>70.7</td>
<td>139.5</td>
<td>139.5</td>
<td>3.6</td>
</tr>
<tr>
<td>fft-tiny</td>
<td>3.1</td>
<td>5.8</td>
<td>85.0</td>
<td>103.4</td>
<td>1.1</td>
<td>3.1</td>
<td>5.8</td>
<td>85.0</td>
<td>103.4</td>
<td>103.4</td>
<td>1.1</td>
</tr>
<tr>
<td>iff-tiny</td>
<td>3.1</td>
<td>5.9</td>
<td>83.1</td>
<td>101.0</td>
<td>1.1</td>
<td>3.1</td>
<td>5.9</td>
<td>83.1</td>
<td>101.0</td>
<td>101.0</td>
<td>1.1</td>
</tr>
<tr>
<td>patricia</td>
<td>2.5</td>
<td>4.2</td>
<td>26.4</td>
<td>31.9</td>
<td>1.0</td>
<td>2.5</td>
<td>4.2</td>
<td>26.4</td>
<td>31.9</td>
<td>31.9</td>
<td>1.0</td>
</tr>
<tr>
<td>qsort-small</td>
<td>3.1</td>
<td>3.7</td>
<td>7.8</td>
<td>8.6</td>
<td>1.0</td>
<td>3.1</td>
<td>3.7</td>
<td>7.8</td>
<td>8.6</td>
<td>8.6</td>
<td>1.0</td>
</tr>
<tr>
<td>rsynth-tiny</td>
<td>16.0</td>
<td>29.9</td>
<td>1096.1</td>
<td>1539.8</td>
<td>1.9</td>
<td>16.0</td>
<td>29.9</td>
<td>1096.1</td>
<td>1539.8</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>search-large</td>
<td>2.9</td>
<td>3.9</td>
<td>26.3</td>
<td>52.5</td>
<td>0.8</td>
<td>2.9</td>
<td>3.9</td>
<td>26.3</td>
<td>52.5</td>
<td>52.5</td>
<td>0.8</td>
</tr>
<tr>
<td>search-small</td>
<td>2.9</td>
<td>3.7</td>
<td>28.1</td>
<td>54.8</td>
<td>0.8</td>
<td>2.9</td>
<td>3.7</td>
<td>28.1</td>
<td>54.8</td>
<td>54.8</td>
<td>0.8</td>
</tr>
<tr>
<td>sha</td>
<td>8.3</td>
<td>14.1</td>
<td>668.7</td>
<td>700.6</td>
<td>1.6</td>
<td>8.3</td>
<td>14.1</td>
<td>668.7</td>
<td>700.6</td>
<td>700.6</td>
<td>1.6</td>
</tr>
<tr>
<td>ssusan-small</td>
<td>29.2</td>
<td>43.9</td>
<td>4313.5</td>
<td>4678.0</td>
<td>3.3</td>
<td>29.2</td>
<td>43.9</td>
<td>4313.5</td>
<td>4678.0</td>
<td>3.3</td>
<td>3.3</td>
</tr>
</tbody>
</table>
Experiments: Average Measurement

- Best result for LP_{128}: bitcnts with spilling totally avoided
- In the mean for LP_{256}: 30% reduction in spilling
- Worst results: qsort-small with 76% spilling
- Presence of a stack cache improves the performance of the data cache
  - bitcnts and csusan: Bytes per cycle of 1.2 jumps to 192.4 and 196.1 respectively
- Consistent improvements over all benchmarks
### Experiments: Static Analysis

- Static worst-case compared to observations from dynamic execution
- Bytes spilled per stack cache context
- Maximum difference between statically predicted and observed spilling

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>basicmath-tiny</td>
<td>68,128</td>
<td>32,040</td>
<td>2.13×</td>
<td>10,052</td>
<td>8,080</td>
<td>1.24×</td>
</tr>
<tr>
<td>bitcns</td>
<td>892</td>
<td>684</td>
<td>1.30×</td>
<td>768</td>
<td>320</td>
<td>2.40×</td>
</tr>
<tr>
<td>crc-32</td>
<td>844</td>
<td>652</td>
<td>1.29×</td>
<td>684</td>
<td>372</td>
<td>1.84×</td>
</tr>
<tr>
<td>csusan</td>
<td>5,404</td>
<td>2,592</td>
<td>2.08×</td>
<td>2,420</td>
<td>1,196</td>
<td>2.02×</td>
</tr>
<tr>
<td>dbf</td>
<td>684</td>
<td>456</td>
<td>1.50×</td>
<td>564</td>
<td>324</td>
<td>1.74×</td>
</tr>
<tr>
<td>dijkstra-small</td>
<td>10,220</td>
<td>5,796</td>
<td>1.76×</td>
<td>6,676</td>
<td>2,608</td>
<td>2.56×</td>
</tr>
<tr>
<td>drijndael</td>
<td>1,172</td>
<td>664</td>
<td>1.77×</td>
<td>1,024</td>
<td>488</td>
<td>2.10×</td>
</tr>
<tr>
<td>ebf</td>
<td>684</td>
<td>456</td>
<td>1.50×</td>
<td>564</td>
<td>324</td>
<td>1.74×</td>
</tr>
<tr>
<td>erijndael</td>
<td>880</td>
<td>400</td>
<td>2.20×</td>
<td>752</td>
<td>292</td>
<td>2.58×</td>
</tr>
<tr>
<td>esusan</td>
<td>4,724</td>
<td>1,888</td>
<td>2.50×</td>
<td>2,256</td>
<td>1,024</td>
<td>2.20×</td>
</tr>
<tr>
<td>fft-tiny</td>
<td>32,484</td>
<td>9,476</td>
<td>3.43×</td>
<td>5,804</td>
<td>3,712</td>
<td>1.56×</td>
</tr>
<tr>
<td>ifft-tiny</td>
<td>32,224</td>
<td>9,256</td>
<td>3.48×</td>
<td>5,620</td>
<td>3,548</td>
<td>1.58×</td>
</tr>
<tr>
<td>patricia</td>
<td>1,996</td>
<td>1,672</td>
<td>1.19×</td>
<td>1,804</td>
<td>984</td>
<td>1.83×</td>
</tr>
<tr>
<td>qsort-small</td>
<td>3,804</td>
<td>1,492</td>
<td>2.55×</td>
<td>2,432</td>
<td>840</td>
<td>2.90×</td>
</tr>
<tr>
<td>rsynth-tiny</td>
<td>109,864</td>
<td>15,320</td>
<td>7.17×</td>
<td>13,504</td>
<td>3,140</td>
<td>4.30×</td>
</tr>
<tr>
<td>search-large</td>
<td>840</td>
<td>740</td>
<td>1.14×</td>
<td>668</td>
<td>340</td>
<td>1.96×</td>
</tr>
<tr>
<td>search-small</td>
<td>828</td>
<td>728</td>
<td>1.14×</td>
<td>708</td>
<td>312</td>
<td>2.27×</td>
</tr>
<tr>
<td>sha</td>
<td>1,160</td>
<td>660</td>
<td>1.76×</td>
<td>1,032</td>
<td>448</td>
<td>2.30×</td>
</tr>
<tr>
<td>ssusan</td>
<td>6,608</td>
<td>1,824</td>
<td>3.62×</td>
<td>2,452</td>
<td>1,060</td>
<td>2.31×</td>
</tr>
</tbody>
</table>
Experiments: Static Analysis

- Static spill cost is reduced for all programs in the benchmark set
- When LP heavily reduces spilling, it also reduces the gap

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SC_{128} Static</th>
<th>Max-Spilling-Δ Static</th>
<th>LP-SC_{128} Static</th>
<th>Max-Spilling-Δ Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicmath-tiny</td>
<td>68,128</td>
<td>32,040</td>
<td>10,052</td>
<td>8,080</td>
</tr>
<tr>
<td>bitcns</td>
<td>892</td>
<td>684</td>
<td>768</td>
<td>320</td>
</tr>
<tr>
<td>crc-32</td>
<td>844</td>
<td>652</td>
<td>684</td>
<td>372</td>
</tr>
<tr>
<td>csusan</td>
<td>5,404</td>
<td>2,592</td>
<td>2,420</td>
<td>1,196</td>
</tr>
<tr>
<td>dbf</td>
<td>684</td>
<td>456</td>
<td>564</td>
<td>324</td>
</tr>
<tr>
<td>dijkstra-small</td>
<td>10,220</td>
<td>5,796</td>
<td>6,676</td>
<td>2,608</td>
</tr>
<tr>
<td>drijndael</td>
<td>1,172</td>
<td>664</td>
<td>1,024</td>
<td>488</td>
</tr>
<tr>
<td>ebf</td>
<td>684</td>
<td>456</td>
<td>564</td>
<td>324</td>
</tr>
<tr>
<td>erijndael</td>
<td>880</td>
<td>400</td>
<td>752</td>
<td>292</td>
</tr>
<tr>
<td>esusan</td>
<td>4,724</td>
<td>1,888</td>
<td>2,256</td>
<td>1,024</td>
</tr>
<tr>
<td>fft-tiny</td>
<td>32,484</td>
<td>9,476</td>
<td>5,804</td>
<td>3,712</td>
</tr>
<tr>
<td>ifft-tiny</td>
<td>32,224</td>
<td>9,256</td>
<td>5,620</td>
<td>3,548</td>
</tr>
<tr>
<td>patricia</td>
<td>1,996</td>
<td>1,672</td>
<td>1,804</td>
<td>984</td>
</tr>
<tr>
<td>qsort-small</td>
<td>3,804</td>
<td>1,492</td>
<td>2,432</td>
<td>840</td>
</tr>
<tr>
<td>rsynth-tiny</td>
<td>109,864</td>
<td>15,320</td>
<td>13,504</td>
<td>3,140</td>
</tr>
<tr>
<td>search-large</td>
<td>840</td>
<td>740</td>
<td>668</td>
<td>340</td>
</tr>
<tr>
<td>search-small</td>
<td>828</td>
<td>728</td>
<td>708</td>
<td>312</td>
</tr>
<tr>
<td>sha</td>
<td>1,160</td>
<td>660</td>
<td>1,032</td>
<td>448</td>
</tr>
<tr>
<td>ssusan</td>
<td>6,608</td>
<td>1,824</td>
<td>2,452</td>
<td>1,060</td>
</tr>
</tbody>
</table>
Summary

- Analyzable caches for time-predictable systems
- Stack cache
- Stack cache optimization
- Lazy spilling stack cache
- Static analysis of lazy spilling stack cache
- Results