An Estimation Methodology for Designing Instruction Cache Memory of Embedded Systems

Nikolaos Kroupis, Stylianos Mamagkakis, and Dimitrios Soudris

Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

The project is co-funded by the European Social Fund & National Resources - EPEAEK II - PYTHAGORAS II
Presentation Outline

- Instruction caches & processor performance
- Instruction cache assumptions
- The proposed estimation methodology of the number of executed instructions and the number of cache misses
- The implemented estimation software tool
- Experimental results and comparison study
- Conclusions and future work
Cache and Memory Performance

Average Memory Access Time = Hit time + Miss rate \times Miss penalty

Improving memory hierarchy performance:
- Decrease hit time (IC Technology)
- Decrease miss rate (Application Software)
- Decrease miss penalty (IC Technology)
Embedding Processor Core Instruction Set Simulator is needed

Most Instruction Set simulators do not include Cache simulator

Cache simulation is a time-consuming procedure

Cache miss rate exploration to find the best cache size and parameters needs days of simulation
Loop Size & Cache Miss Rate (Type 1)

Loop Type 1: \( L_s = 3 \) instr.

\[
Num\_misses = \frac{L_s}{B_s}
\]

- \( Num\_misses \): Number of instruction misses
- \( L_s \): Loop Size in number of instructions
- \( B_s \): Instruction cache block size

No Conflict Misses
Loop Type 2: $C_s < L_s < 2 \times C_s$

$L_s = 6$ instr.  $C_s = 4$ instr.

\[
\text{Num\_misses} = \frac{L_s}{B_s} + (N - 1) \times 2 \times \frac{L_s}{B_s} \mod \frac{C_s}{B_s}
\]

- **Num\_misses**: Number of instruction misses
- **$L_s$**: Loop Size in number of instructions
- **$B_s$**: Instruction cache block size
- **$N$**: Number of loop iterations
- **$C_s$**: Cache size
Loop Size & Cache Miss Rate (Type 3)

Loop Type 3: $L_s = 17$ instr.

$$\text{Num\_misses} = N \times \frac{L_s}{B_s}$$

- $\text{Num\_misses}$: Number of instruction misses
- $L_s$: Loop Size in number of instructions
- $B_s$: Instruction cache block size
- $N$: Number of loop iterations

Democritus University of Thrace - VLSI Design and Testing Center
Instruction Cache Miss Rate

\[ \text{Num references} = \frac{L - s}{B - s} \times N \]

\[ \text{Miss rate} = \frac{\text{Num misses}}{\text{Num references}} \]

- \text{Num_misses} : Number of instruction misses
- \text{Num_references} : Number of Memory references
- \text{L_s} : Loop Size in number of instructions
- \text{B_s} : Instruction cache block size
- \text{N} : Number of loop iterations
The Proposed Methodology

Pinpoint the code
Counter insertion
Code execution
# executions of

Pinpoint the assembly
Create the controller
Assignment of code
The Proposed Methodology: 1st Stage

Step 1: Pinpoint the code branches

Step 2: Counter insertion

Step 3: Code executions

Results after code execution

Branch 1:
Type: loop
Counter 1: 10 executions

Branch 2:
Type: if
Counter 2: 3 executions

Input: C code

Counter Insertion

Output
The Proposed Methodology: 2\textsuperscript{nd} Stage

1. **Pinpoint the code branches**

   ```c
   for(i=0;i<10;i++)
   {  if(i<3)  
      a=a-i;
   else
      a=a+i;
   }
   ```

2. **Output**

   - **Step 2**: Code executions
   ```c
   for(i=0;i<10;i++)
   {  counter[1]++;  if(i<3)  
      counter[2]++;
      a=a-i;
   else
      a=a+i;
   }
   ```

3. **Counter Insertion**

   - **Step 1**: Pinpoint the code branches
   ```c
   for(i=0;i<10;i++)
   {  if(i<3)  
      a=a-i;
   else
      a=a+i;
   }
   ```

   - **Step 2**: Create the Control Flow Graph

   - **Step 3**: Code executions

4. **Assembly Code**

   ```asm
   Assembly Code
   ```
The Proposed Methodology: 3rd Stage

**Step 1**: Extract all the unique execution paths of assembly code loops.

**Step 2**: Computation of # of instructions and # iterations of each execution path.

**Output**: Number of instruction cache misses and miss rate

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 3</th>
<th>Node 5</th>
<th>Node 6: 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L2$:</td>
<td>$lw\quad 2,16($fp)</td>
<td>slt $3,$2,10</td>
<td>beq $3,$0,$L5</td>
</tr>
<tr>
<td>$L5$:</td>
<td>$lw\quad 2,16($fp)</td>
<td>slt $3,$2,3</td>
<td>beq $3,$0,$L6</td>
</tr>
<tr>
<td>$L6$:</td>
<td>$lw\quad 2,16($fp)</td>
<td>\quad $3,16($fp)</td>
<td>\quad $2,20($fp)</td>
</tr>
<tr>
<td>$L4$:</td>
<td>$lw\quad 2,16($fp)</td>
<td>\quad $3,16($fp)</td>
<td>\quad \quad $2,20($fp)</td>
</tr>
<tr>
<td>$L1$:</td>
<td>$lw\quad 2,16($fp)</td>
<td>\quad $3,16($fp)</td>
<td>\quad \quad $2,20($fp)</td>
</tr>
</tbody>
</table>

MIPS IV 64 bits 1 instr. = 8 bytes

Unique Path 1:
- Consists of: 15 instr.
- Size: 120 bytes
- Iterations: 3

Unique Path 2:
- Consists of: 16 instr.
- Size: 128 bytes
- Iterations: 7

Output
- Direct Mapped Cache with Block Size 8 bytes:
  - Using Equations (1)-(5) for variable cache sizes:
  - Num References = 157
  - Cache Size: 32 bytes
    - Num Misses = 45
    - Num Misses = 112
    - Miss rate = 100%
  - Cache Size: 64 bytes
    - Num Misses = 42
    - Num Misses = 112
    - Miss rate = 98%
  - Cache Size: 128 bytes
    - Num Misses = 15
    - Num Misses = 16
    - Miss rate = 20%
  - Cache Size: 256 bytes
    - Num Misses = 15
    - Num Misses = 16
    - Miss rate = 20%
FICA Software Tool

Fast Instruction Cache Analyzer (FICA)

- Web based Tool
- PHP Language
- MySQL Database
- MIPS IV Architecture with compiler gcc 2.7.2

- Estimate the number of execution instructions, the number of instruction cache misses and the miss rate
- Extract the critical part of the application which give high miss rate
Number of Executed Instructions

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>C Code Size (bytes)</th>
<th>Simplescalar (#instructions)</th>
<th>FICA (#instructions)</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>1,625</td>
<td>1,138,860,504</td>
<td>1,135,823,110</td>
<td>0.26%</td>
</tr>
<tr>
<td>HS</td>
<td>7,009</td>
<td>37,792,750</td>
<td>36,736,600</td>
<td>2.79%</td>
</tr>
<tr>
<td>3SLOG</td>
<td>3,381</td>
<td>46,717,535</td>
<td>45,865,486</td>
<td>1.82%</td>
</tr>
<tr>
<td>PHODS</td>
<td>3,372</td>
<td>70,865,874</td>
<td>69,375,838</td>
<td>2.10%</td>
</tr>
<tr>
<td>SS</td>
<td>2,587</td>
<td>590,420,249</td>
<td>589,448,105</td>
<td>0.16%</td>
</tr>
<tr>
<td>Wavelet</td>
<td>22,380</td>
<td>39,507,821</td>
<td>39,993,202</td>
<td>1.23%</td>
</tr>
<tr>
<td>Cavity Detector</td>
<td>2,634</td>
<td>18,957,657,996</td>
<td>18,351,605,412</td>
<td>3.19%</td>
</tr>
<tr>
<td>CQ</td>
<td>11,215</td>
<td>6,133,172,759</td>
<td>6,299,874,815</td>
<td>0.26%</td>
</tr>
<tr>
<td>FFT</td>
<td>2,681</td>
<td>687,624</td>
<td>620,154</td>
<td>9.81%</td>
</tr>
</tbody>
</table>
## Number of Instruction Cache Misses

<table>
<thead>
<tr>
<th>Miss Rate</th>
<th>Cache Size (bytes)</th>
<th>Average error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Block size 8 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td>Sim.</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>HS</td>
<td>Sim.</td>
<td>99.9</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>3SLOG</td>
<td>Sim.</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>PHODS</td>
<td>Sim.</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>SS</td>
<td>Sim.</td>
<td>99.9</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>Wavelet</td>
<td>Sim.</td>
<td>98.7</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>Cavity Detector</td>
<td>Sim.</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>CQ</td>
<td>Sim.</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
<tr>
<td>FFT</td>
<td>Sim.</td>
<td>99.8</td>
</tr>
<tr>
<td></td>
<td>FICA</td>
<td>100.0</td>
</tr>
</tbody>
</table>
# Estimation vs. Simulation Time

<table>
<thead>
<tr>
<th>(seconds)</th>
<th>Simpescalar</th>
<th>FICA</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>2.278</td>
<td>0.9</td>
<td>2,531</td>
</tr>
<tr>
<td>3SLOG</td>
<td>93</td>
<td>1.3</td>
<td>75</td>
</tr>
<tr>
<td>PHODS</td>
<td>142</td>
<td>1.7</td>
<td>86</td>
</tr>
<tr>
<td>HS</td>
<td>76</td>
<td>4.5</td>
<td>17</td>
</tr>
<tr>
<td>SS</td>
<td>1.182</td>
<td>1.3</td>
<td>909</td>
</tr>
<tr>
<td>Wavelet</td>
<td>107</td>
<td>1.5</td>
<td>70</td>
</tr>
<tr>
<td>Cavity Detector</td>
<td>37.915</td>
<td>2.9</td>
<td>13,186</td>
</tr>
<tr>
<td>CQ</td>
<td>32.268</td>
<td>13.3</td>
<td>2,426</td>
</tr>
<tr>
<td>FFT</td>
<td>11</td>
<td>0.9</td>
<td>12</td>
</tr>
</tbody>
</table>
Conclusions

- Number of executed instructions and instruction cache misses can be estimated without simulation process.
- Instruction cache memory decisions can be taken from early design steps.
- Find out the critical points of the application which give number of cache miss.
- Instruction code transformations could be applied.
Future Work

- Methodology extension for second level instruction cache (L2)
- Tool extension to accept more embedded processor cores (ARM, TI,…etc)
- Application of the methodology to more complex system architectures (e.g. software control caches)
- High level estimation of processor’s power consumption using instruction level power parameters