Variation Aware Compilation for Improving Energy-Efficiency of Nanometer Processor Caches

Goudarzi, Maziar
System LSI Research Center, Kyushu University

Ishihara, Tohru
System LSI Research Center, Kyushu University

Yasuura, Hiroto
System LSI Research Center, Kyushu University

http://hdl.handle.net/2324/9149
Variation Aware Compilation for Improving Energy-Efficiency of Nanometer Processor Caches

Maziar Goudarzi, Tohru Ishihara, Hiroto Yasuura

System LSI Research Center
Kyushu University
Outline

● Background
  – Process variation in nanometer caches
    ● Delay variation
    ● Leakage variation

● Our previous and current work on SRAM
  – Mitigating effects of ultra-slow SRAM cells
  – Suppressing leakage of ultra-leaky ones
Trends of DSPs

- An improvement of power efficiency highly depends on the process technology
- This trend continues more than 10 years

Chips implementing the same design are no longer the same!

S. Borkar, Parameter variations and impact on circuits and microarchitecture, DAC 2003.
Intra-Die Variations

Large Intra-Die Variation

Current 3-sigma = 13%
Vth 3-sigma = 67mV

Variation is huge in small transistors

\[ \sigma_{Vth} = \frac{q}{C_{ox}} \sqrt{\frac{N_a \cdot W_{dm}}{3 \cdot L \cdot W}} \]

\( L, W \): Effective channel length and width
\( q \): electron charge
\( C_{ox} \): oxide capacitance
\( N_a \): substrate doping concentration
\( W_{dm} \): maximum depletion width

Process Variation at 90nm

\[ I_{\text{Subthreshold}} \propto \frac{W \cdot V_T^2}{T_{ox} \cdot L} \cdot \exp\left(\frac{-V_{th}}{\alpha \cdot V_T}\right) \]

- **\( V_T \)**: Thermal voltage (25mV@room temperature)
- \( \alpha \): Sub-threshold factor (1.40~1.65)
- **\( T_{ox} \)**: Oxide thickness

<table>
<thead>
<tr>
<th>Year</th>
<th>min. ( L ) [nm]</th>
<th>( V_{TH} ) [V]</th>
<th>( 2V_{TH} ) [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>37 (90)</td>
<td>0.32</td>
<td>0.12</td>
</tr>
<tr>
<td>2005</td>
<td>32 (80)</td>
<td>0.33</td>
<td>0.09</td>
</tr>
<tr>
<td>2006</td>
<td>28 (70)</td>
<td>0.34</td>
<td>0.06</td>
</tr>
</tbody>
</table>

1: Low Operating Power Process  
2: MPU process

- 1 transistor out of 512K-bit SRAM
- \( 5\sigma_{V_{th}} = 0.3V \)
- Leakage is 1,400x higher than average!
- 512K-bit SRAM
- \( 100 \) tr.
- \( 330x \)
- \( 1.8x \)
- Delay is 2x of the average

100 tr.
Skipping Slow Cache-Lines

- Use an unused combination of existing flag bits to indicate a slow SRAM cell in a specific cache-line.
- Invalidate and skip a cache-line if it is marked.

4-way set-associative cache memory

Lock and skip these memory sections

Lock-bits
Valid-bits

Cache replacement policy
Cache Miss Reduction

- Using a smaller cache memory does not affect the correct operation of a processor.
- The problem is an increase of a cache miss rate due to a reduced cache size.

Mark sections which contain a leaky or slow bit

Use fault-free sections only

Diagram:
- Processor
- Main Memory
- CPU Core
- Cache
- Program
- DATA
Our Approach

- Modify the order of functions in the address space such that ultra-slow cache-lines are not accessed frequently.
Process Variation at 90nm

\[ I_{\text{Subthreshold}} \propto \frac{W \cdot V_T^2}{T_{ox} \cdot L} \cdot \exp\left(\frac{-V_{th}}{\alpha \cdot V_T}\right) \]

\( V_T \): Thermal voltage (25mV@room temperature)
\( \alpha \): Sub-threshold factor (1.40~1.65)
\( T_{ox} \): Oxide thickness

- Large Delay
- Large Leak
- Mean

<table>
<thead>
<tr>
<th>Year</th>
<th>min. ( L ) [nm]</th>
<th>( V_{TH} ) [V]</th>
<th>( 2V_{TH} ) [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>37 (90)</td>
<td>0.32</td>
<td>0.12</td>
</tr>
<tr>
<td>2005</td>
<td>32 (80)</td>
<td>0.33</td>
<td>0.09</td>
</tr>
<tr>
<td>2006</td>
<td>28 (70)</td>
<td>0.34</td>
<td>0.06</td>
</tr>
</tbody>
</table>

1: Low Operating Power Process  2: MPU process

\( V_{th} = 0.3V \)

1 transistor out of 512K-bit SRAM

\( 5\sigma_{V_{th}} = 0.3V \)

Leakage is 1,400x higher than average!

\( 330x \)

\( 100 \) tr.

Delay is 2x of the average

\( 1.8x \)

\( \pm \sigma: 68.3\% \)
\( \pm 2\sigma: 95.4\% \)
\( \pm 3\sigma: 99.7\% \)
\( \pm 4\sigma: 99.9936\% \)
\( \pm 5\sigma: 99.99994\% \)
Masking Leaky Transistors

Leakage current of a SRAM cell depends on the logic value stored.

If M2, M3, or M5 is ultra-leaky, the SRAM cell is 1-leaky.
If M1, M4, or M6 is ultra-leaky, the SRAM cell is 0-leaky.
Masking Leaky Cache-Lines

1. Skip using leaky cache lines by marking the lines and store “leakage-safe” values to the leaky cache lines.
2. Modify the order of instruction codes considering binary expressions of the codes and locations of 0/1-leaky bits in a cache so that the total leakage current is minimized.

4-way set-associative cache memory

- **1-leaky cells**
- **0-leaky cells**
Leakage-Aware Scheduling

Instruction cache (Direct-map 512-set, 1-byte line size)

Leakage-preference of cache cells:
1: the cell prefers 1 (i.e., leaks less if storing 1)
0: the cell prefers 0 (i.e., leaks less if storing 0)
*: the cell not having preference

Instruction-to-Cell matching table

<table>
<thead>
<tr>
<th>Instruction</th>
<th>488</th>
<th>489</th>
<th>490</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1002</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Leakage improvement

<table>
<thead>
<tr>
<th>Possible schedules</th>
<th># matches</th>
<th>Leak improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># mismatches</td>
<td>%</td>
</tr>
<tr>
<td>1-2-3 (original)</td>
<td>1+0+2=3</td>
<td>2+3+3=8</td>
</tr>
<tr>
<td>1-3-2</td>
<td>1+1+4=6</td>
<td>2+2+1=5</td>
</tr>
<tr>
<td>3-1-2</td>
<td>1+3+4=8</td>
<td>2+0+1=3</td>
</tr>
</tbody>
</table>
The Flow

Fabricated Chip

Target Application

Detect Locations of Bad Cells

Generate a new object code or reuse an object code previously generated

Mark Bad Cache-Lines

Execute Object Code

Testing Phase (BIST)

Compiling Phase

Booting Phase

Running Phase (Flash Microcontroller)
Compiler Optimization Flow

Target Application

Original Object Code

Sample Data

Instruction Trace (Profiling Information)

Locations of Bad Cells

Find instruction scheduling and code placement, for which the total energy consumption can be minimized under a performance constraint

Modified Object Code
New Paradigm

- Use different object codes for different chips

Future work: Reducing the cost involved
Code Placement Results

Target CPU: ARMv4T architecture

Benchmark: MPEG2 encoder
# Inst. Scheduling Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Benchmark</th>
<th>Cache configurations (sets x ways x line size)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>512x1x4</td>
</tr>
<tr>
<td>Exhaustive Search</td>
<td>MPEG2</td>
<td>10.85</td>
</tr>
<tr>
<td></td>
<td>FFT</td>
<td>10.74</td>
</tr>
<tr>
<td></td>
<td>JPEG</td>
<td>8.59</td>
</tr>
<tr>
<td></td>
<td>Compress</td>
<td>10.96</td>
</tr>
<tr>
<td></td>
<td>FIR</td>
<td>12.51</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td><strong>10.73</strong></td>
</tr>
<tr>
<td>List Scheduling</td>
<td>MPEG2</td>
<td>9.05</td>
</tr>
<tr>
<td></td>
<td>FFT</td>
<td>8.80</td>
</tr>
<tr>
<td></td>
<td>JPEG</td>
<td>6.34</td>
</tr>
<tr>
<td></td>
<td>Compress</td>
<td>10.21</td>
</tr>
<tr>
<td></td>
<td>FIR</td>
<td>11.90</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td><strong>9.26</strong></td>
</tr>
</tbody>
</table>

Leakage reduction in a cache memory (%)
## Inst. Scheduling Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Benchmark</th>
<th>Cache configurations (sets x ways x line size)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>512x1x4</td>
<td>256x2x4</td>
<td>128x4x4</td>
<td>64x8x4</td>
</tr>
<tr>
<td>Exhaustive Search</td>
<td>MPEG2</td>
<td>220.10</td>
<td>366.43</td>
<td>672.09</td>
<td>1086.61</td>
</tr>
<tr>
<td></td>
<td>FFT</td>
<td>107.46</td>
<td>129.75</td>
<td>249.39</td>
<td>290.74</td>
</tr>
<tr>
<td></td>
<td>JPEG</td>
<td>150.09</td>
<td>169.89</td>
<td>133.43</td>
<td>132.46</td>
</tr>
<tr>
<td></td>
<td>Compress</td>
<td>91.98</td>
<td>94.84</td>
<td>93.51</td>
<td>187.86</td>
</tr>
<tr>
<td></td>
<td>FIR</td>
<td>9.00</td>
<td>17.66</td>
<td>33.88</td>
<td>67.80</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td><strong>115.73</strong></td>
<td><strong>155.71</strong></td>
<td><strong>272.20</strong></td>
<td><strong>394.40</strong></td>
</tr>
<tr>
<td>List Scheduling</td>
<td>MPEG2</td>
<td>0.06</td>
<td>0.06</td>
<td>0.07</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>FFT</td>
<td>0.03</td>
<td>0.03</td>
<td>0.04</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>JPEG</td>
<td>0.06</td>
<td>0.05</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>Compress</td>
<td>0.03</td>
<td>0.03</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>FIR</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
<td><strong>0.03</strong></td>
<td><strong>0.03</strong></td>
<td><strong>0.04</strong></td>
<td><strong>0.04</strong></td>
</tr>
</tbody>
</table>

* Computational time on 3.8GHz Xeon processor with 3.5GB memory (sec.)
Summary

- Cancel the degradation of cache hit-rate even in presence of 25% slow cache-lines.
- Worst case delay can be reduced by 15%.
- Higher $V_{th}$ (lower leakage) can be used without any performance degradation.
- Leakage power can be reduced by 10%.
- No major HW modification is required.

Future work

- Reduction of test and recompilation costs.
Conclusion

- Chips implementing the same design are no longer the same!
  - Traditional example: Frequency binning
  - Intensifies with each new technology node
- Per-chip customization now reasonable
  - Expensive as a manufacturing post-processing step
- Solution:
  - Let the RTOS customize the application to the chip
  - Variation-Aware Self-Calibrating RTOS


Input Data Dependency

- Compared cache miss rates for 6 different input values.
- The optimized code for Data0 achieves very good results for other input values too.
Previous Work (1/2)

- Vergos et al. proposed a technique using spare cache.

- Sohi proposed a technique using error correcting code.
Shiravani et al. proposed **PADded** cache
- Customize an address decoder so that faulty blocks will not be accessed.

FT bits

<table>
<thead>
<tr>
<th>Cache-Line0 ($a_0a_1=00$)</th>
<th>TAG</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0a_0 + f_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache-Line1 ($a_0a_1=01$)</th>
<th>TAG</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1a_0 + f_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache-Line2 ($a_0a_1=10$)</th>
<th>TAG</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_2a_0 + f_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_2$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache-Line3 ($a_0a_1=11$)</th>
<th>TAG</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_3a_0 + f_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Effect of Technology Scaling

- Savings improve in future technologies
  - Results for Exhaustive-Search (max. BB-len=10) applied to MPEG2 on 512x1x4 2KB cache