Compiler-Based Adaptive Fetch Throttling for Energy-Efficiency

Huaping Wang, Yao Guo, Israel Koren and C. Mani Krishna

ECE Dept, UMass at Amherst
Introduction

- Power consumption increases significantly in modern computer architecture.
- Fetch throttling can reduce executions of miss-fetched instructions and number of Icache accesses.
Fetch throttling techniques

- **Hardware-based runtime techniques**
  - Use past behavior to predict future behavior.
  - Can not catch irregular situations such as abrupt program phase changes.
  - Cause substantial performance degradation.

- **Software-based static techniques**
  - Estimate Instruction Level Parallelism (ILP) based on compile-time program analysis.
  - Can not capture dynamic effects, such as cache misses.
  - Use fixed low IPC threshold for throttling - to avoid high performance loss.
  - Energy savings is small if IPC threshold is low.
Potential problems of fixed low IPC threshold

- Limits throttling opportunities at high IPC values:
  - If estimated IPC (e.g., 3) is less than number of instructions left unexecuted in previous cycle (e.g., 5), we can throttle fetch even at a high IPC value.

- May throttle at an inappropriate time resulting in a performance loss:
  - If estimated IPC is low (e.g., 2) but no instructions left in the issue queue (from previous cycle), throttling results in performance loss.
Compiler-based Adaptive Fetch Throttling (CAFT)

- IPC estimate using compile-time analysis.
- A large Decode/Issue Difference (DID) means that many instructions were left unexecuted.
- DID value can be used as recent history information to change the IPC threshold adaptively

\[
\text{IF } \text{Estimated\_IPC} \leq \text{DID} \\
\text{THEN throttle for one cycle}
\]
Compiler-level implementation

- Used SUIF/MachSUIF as our compiler framework
- Added new passes to both SUIF and MachSUIF to annotate and propagate the static IPC-estimation

Compiler-based IPC estimate

- Consider only true data dependencies.
- Identify data dependencies for both registers and memory accesses.
- Use approximate and speculative alias analysis for memory accesses.
Experiments

- Setup
  - SimpleScalar/Wattch
  - SPEC2000 and Mediabench benchmarks

- Examined several existing throttling techniques
  - Hardware dependence-based (DEP)
  - Just-In-Time instruction delivery (JIT)
  - Compiler-based fixed IPC threshold (CFT)

- Compared CAFT to above techniques
  - Throttling cycles and IPC threshold distribution
  - Execution Time and Energy
  - Energy Delay Product (EDP)
Number of throttling cycles and IPC distribution

- Number of throttling cycles increases significantly compared to fixed low IPC-threshold
- Percent of throttling cycles above IPC-threshold of 2 is larger than 50% in most benchmarks
CAFT keeps the advantage of low performance decrease of CFT, and has a good energy savings as hardware-based techniques.
Energy Delay Product (EDP)

- Compared to fixed threshold technique (CFT), CAFT achieves a 3.7% additional EDP saving and 6.7% overall EDP reduction.
- Compared to DEP, CAFT achieves a 3.2% additional EDP reduction.
Conclusion

- CAFT has a better EDP savings than software- or hardware-only fetch throttling techniques.
Experiment setup (Backup)

- Skip the initialization stage and simulate next 500M instructions for SPEC; run Mediabench to completion.

<table>
<thead>
<tr>
<th>Processor Speed</th>
<th>5GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Parameters</td>
<td>0.18µm, 2V</td>
</tr>
<tr>
<td>Issue</td>
<td>Out-Of-Order</td>
</tr>
<tr>
<td>Fetch,Issue,Decoded,Commit</td>
<td>8-way</td>
</tr>
<tr>
<td>Fetch Queue Size</td>
<td>32</td>
</tr>
<tr>
<td>Instruction Queue Size</td>
<td>128</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>2K entry bimodal</td>
</tr>
<tr>
<td>Int.Functional Units</td>
<td>4 ALUs, 1Mult./Div.</td>
</tr>
<tr>
<td>FP Functional Units</td>
<td>4 ALUs, 1 Mult./Div.</td>
</tr>
<tr>
<td>L1 D-cache</td>
<td>128Kb, 4-way, writeback</td>
</tr>
<tr>
<td>L1 I-cache</td>
<td>128Kb, 4-way, writeback</td>
</tr>
<tr>
<td>Combined L2 cache</td>
<td>1Mb, 4-way associative</td>
</tr>
<tr>
<td>L2 Cache hit time</td>
<td>20 cycles</td>
</tr>
<tr>
<td>Main memory hit time</td>
<td>100 cycles</td>
</tr>
</tbody>
</table>