Microarchitecture Support for Interconnect Power-aware Instruction Permutation

Hui Lin, Md. Sajjad Rahaman and Prof. Masud H Chowdhury
ECE Department, University of Illinois at Chicago
Outline

- Introduction & Background
- Problem Statement
- Detail Design to Reorder Instructions
- Experiments Analysis
Introduction
VLSI technology steps into the deep sub-micrometer (DSM) level
- Resulting in reduction on component spacing
- Resulting in complication of interconnect network

Self and coupling capacitance badly affects power consumption
- Effective values are affected by physical features as well as dynamic bit values
Background Work

- Physical approaches to reduce the power consumption on interconnect bus
  - Shielding line
  - Reformat the physical interconnect lines

- Logic approaches
  - Encoding/decoding: operation can change the resulting bits values as well as the switching activity
  - Compiler design: change the instructions order or register name
Separating Program Behavior in Processor and at Interconnect

- Processor execute instructions from the binary code stored in memory
  - It only retrieves instructions from the Cache.
- Interconnect bus transmits instructions, data values, as well as commands
- Reorder instructions in the memory which reduces the switching activity during transmission
- Operations is performed during every cache miss
Problem Statement
Interconnect Bus Model

- RC interconnect model is used to describe interconnect bus connecting processor and memory.

- Dynamic power is represented as:
  - \[ P = \alpha (C_c + C_g) V_{dd}^2 f \] or
  - \[ P = (1 + N_c \lambda) \alpha C_g V_{dd}^2 f \]
  - \( N_c \) is used to describe dynamic factor as bit values change.
Design Objective

- Reorders program’s binary code stored in memory in the size of cache line into the power efficient format
- Reduce the overhead of index table which is used to reinstate the instruction in the cache
Detail Design to Reorder Instructions
Overview of Permutation Algorithm

- When instructions are stored in the memory, reorder instructions according to cache line size
  - No consideration on data or control dependency
  - Could be done offline
  - Construct 1D index table which is used to recover instructions original order

- How to manage such index table?
  - Its transmission requires extra power consumption. In the same level of transmission of reformatted instruction block
Framework of Power Efficient Instruction Rescheduling

Disk Drive

Application’s executable code

Scheduling to power efficient before loading to the memory

Main Memory

Instruction block #1
Instruction block #2
⋮
Instruction block #n

Transmitting
1. instructions in power efficient format
2. index table for restoring

Cache

Instruction #1
Instruction #2
⋮
Instruction #r
Instruction Permutation

- Bit values within instructions are correlated to each other
  - Divided into several field: opcode, source register, destination register
  - RISC instruction set is adopted
  - Reorder instruction from field to field
Instruction Permutation (cont’d)

- Grouping instructions with same opcode
  - Number of opcode used is very limited
  - Efficient algorithm existed
  - Further algorithm design does not jeopardize the power reduction done before

- Reordering within each group
  - With small size, inefficient algorithm can be used
The index table contains relationship between instructions in its original format and ones in its power efficient format.

Relative location of instructions are recorded.

<table>
<thead>
<tr>
<th>Instruction</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD r0, r1, r2</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>ADD r2, r3, 5</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ADD r2 r0, r2</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>SUB r3, r2, r1</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>SUB r4, r5, r0</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>MUL r2, r2, 4</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>LW r1, 4 (r2)</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>LW r3, 3 (r4)</td>
<td>011</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LW r1, 4 (r2)</td>
<td></td>
</tr>
<tr>
<td>ADD r0, r1, r2</td>
<td></td>
</tr>
<tr>
<td>ADD r2 r0, r2</td>
<td></td>
</tr>
<tr>
<td>LW r3, 3 (r4)</td>
<td></td>
</tr>
<tr>
<td>ADD r2, r3, 5</td>
<td></td>
</tr>
<tr>
<td>SUB r3, r2, r1</td>
<td></td>
</tr>
<tr>
<td>MUL r2, r2, 4</td>
<td></td>
</tr>
<tr>
<td>SUB r4, r5, r0</td>
<td></td>
</tr>
</tbody>
</table>
Index Table Overhead Analysis

- Power consumption
  - Index table is sent before instructions
  - Transmission of Index table itself will cause same level of power consumption as the reordered instructions do
  - Reduce the frequency of transmission: store all of them into processor
  - Storing index table also introduces extra power consumption
    - Not dominating as technology goes to small level
Index Table Overhead Analysis (Cont’d)

- Area overhead
  - Reorder operations is much easier than the encoding/decoding circuit
  - Storing space is small comparing to the cache size

- Performance overhead
  - The performance comes from accessing index table
  - Increase the cache miss penalty
Experiments Analysis
Power Reduction Achieved

- Switching activity is counted for both original instructions format and the power efficient format
- Self and coupling capacitance are calculated separately
- The effect of grouping and intra-group permutation are compared
- Overall complexity can be regarded as $O(M \log_2 N)$
Power Reduction Result

Reduction of switching activity for self-capacitance (%)

- Opcode permutation
- Intra-group permutation

Reduction of switching activity for coupling-capacitance (%)

- Opcode permutation
- Intra-group permutation
Index Table Overhead

- In order to reduce the frequency to transmit index table during the runtime, storing all of them in the processor.
- Implemented as a level-2 cache: 20k byte size can meet most application programs.
- Area, power consumption and access latency are get from the simulation under CACTI 5.0.
### Area, Power & Access Time Overhead

<table>
<thead>
<tr>
<th>Technology node</th>
<th>Overhead criterion</th>
<th>L2 Cache (256k)</th>
<th>Index Table (20k)</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>90nm</td>
<td>Area (mm^2)</td>
<td>7.6317</td>
<td>1.0682</td>
<td>14.0%</td>
</tr>
<tr>
<td></td>
<td>Power (W)</td>
<td>0.62060</td>
<td>0.16960</td>
<td>27.3%</td>
</tr>
<tr>
<td></td>
<td>Access time (ns)</td>
<td>1.6857</td>
<td>0.86934</td>
<td>51.6%</td>
</tr>
<tr>
<td>65nm</td>
<td>Area (mm^2)</td>
<td>3.9857</td>
<td>0.55717</td>
<td>14.0%</td>
</tr>
<tr>
<td></td>
<td>Power (W)</td>
<td>0.47897</td>
<td>0.13750</td>
<td>28.7%</td>
</tr>
<tr>
<td></td>
<td>Access time</td>
<td>1.1932</td>
<td>0.58601</td>
<td>49.1%</td>
</tr>
<tr>
<td>32nm</td>
<td>Area (mm^2)</td>
<td>0.96800</td>
<td>0.13506</td>
<td>14.0%</td>
</tr>
<tr>
<td></td>
<td>Power (W)</td>
<td>0.43274</td>
<td>0.08099</td>
<td>18.7%</td>
</tr>
<tr>
<td></td>
<td>Access time (ns)</td>
<td>0.60690</td>
<td>0.25402</td>
<td>41.9%</td>
</tr>
</tbody>
</table>
Any questions?
Thanks