Evaluation of the Complexity-Efficiency of Superscalar Processors

"Complexity-Effective Superscalar Processors"
S. Palacharla, N.P. Jouppi and J.E. Smith
(ISCA-27)

Miquel Pericàs Gleim
mpericas@ac.upc.es
Goals

- Characterizing the complexity of critical structures in wide-issue superscalar processors. Studying the trade-off between hardware complexity and clock speed.

- Proposing an alternative microarchitecture that has lesser complexity (and smaller delays) and still maintains high values of ILP.
Study of Complexity Sources

Methodology

1. Choose specific structures for analysis
2. Selection of a representative CMOS circuit
3. Implementation of the circuit and optimization for speed. Simulation has been performed using HSPICE with three feature sizes: 0.8μm, 0.35μm and 0.18μm

Goal is to analyze how the delays of the structures depend on the issue width and the instruction window
Baseline Microarchitecture

- Alpha 21264
- MIPS R10000
Register Rename & Bypass

Register Rename Logic
- Access to RAM structure
- Delay grows \textit{linearly} with issue width

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Issue & Wire & Delay \\
width & length ($\lambda$) & (ps) \\
\hline
4 & 20500 & 184.9 \\
8 & 49000 & 1056.4 \\
\hline
\end{tabular}
\end{table}

Bypass Logic
- Long wires
- Grows \textit{quadratically} with issue width
Wake up & Select

Wakeup Logic
- Access to CAM array
- Grows quadratically with window size and with issue width!

Selection Logic
- Tree of Arbiters
- Increases logarithmically with window size
Summary of Delays

<table>
<thead>
<tr>
<th>Issue width</th>
<th>Window size</th>
<th>Rename delay (ps)</th>
<th>Wakeup+Select delay (ps)</th>
<th>Bypass delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.8 ( \mu )m technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1577.9</td>
<td>2903.7</td>
<td>184.9</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>1710.5</td>
<td>3369.4</td>
<td>1056.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.35 ( \mu )m technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>627.2</td>
<td>1248.4</td>
<td>184.9</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>726.6</td>
<td>1484.8</td>
<td>1056.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.18 ( \mu )m technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>351.0</td>
<td>578.0</td>
<td>184.9</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>427.9</td>
<td>724.0</td>
<td>1056.4</td>
</tr>
</tbody>
</table>

Window Logic and Bypasses seem to pose the largest problems.

Window Logic and Bypasses are difficult to pipeline.

Large wires impossible to enable single-cycle data moves.
A Complexity Effective Microarchitecture

- Issue window logic is a primary contributor to complexity

- A dependence-based microarchitecture that replaces the issue window is proposed
Instruction Steering

- The *dependence-based* microarchitecture exploits natural dependences among instructions.
- Issue window is replaced by a set of FIFOs that can only issue in-order.
- Dependent instructions are steered to the same FIFO.
Performance of Dependence-Based Microarchitecture

Comparison of 8-FIFO/8-entry DBM with wide-8 OOO superscalar with a 64 inst. window

For SPEC95 DBM loses 8% in the worst case
Complexity Analysis

Wakeup logic does not involve broadcasting the result tags to all waiting instructions

Implemented via a reservation table that contains a single bit per physical register that indicates whether the register is waiting for its data.

Delay of wakeup logic is determined by the delay of accessing the reservation table.

<table>
<thead>
<tr>
<th>Issue width</th>
<th>No. physical registers</th>
<th>No. table entries</th>
<th>Bits per entry</th>
<th>Total delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>80</td>
<td>10</td>
<td>8</td>
<td>192.1</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>16</td>
<td>8</td>
<td>251.7</td>
</tr>
</tbody>
</table>

Instruction steering is done in parallel with register renaming

The delay is determined by the register renaming as the FIFO tables are much smaller
Clusterizing the Dependence-Based Microarchitecture

Clustering is a typical solution to simplify bypass and issue logic
Performance of clustered microarchitecture

IPC for the 2x4 way dependence-based microarchitecture against the centralized wide-8 microarchitecture.
Adjusting speeds

The simpler microarchitecture of the clustered scheme allows a clock speed increase of about 25%

Taking this into account the performance improvements vary from 10% to 22%
Evaluation of rename steering

Rename steering and inter-cluster communication are evaluated using 5 different configurations:

1. 1-cluster.1window (baseline)
2. 2-cluster.FIFOs.dispatch_steer
3. 2-cluster.windows.dispatch_steer
4. 2-cluster.1window.exec_steer
5. 2-cluster.windows.random_steer
Performance Evaluation
Communication Overhead
Conclusions: Hardware Analysis

Few microarchitecture research projects make detailed analysis of underlying hardware.

Circuits have been modelled following microprocessor-published models and with help from DEC.

Analysis is serious and results show high confidence.

Only constant bypass delay is hard to believe. Modelling wire delays in SPICE is complex. The authors choose distributed RC lines which are not too accurate.
Conclusions: Microarchitecture

- The proposal to use the dependence-based microarchitecture is interesting but relies on a flaw.
- The whole proposal bases on the fact that pipelining issue logic makes execution of back-to-back instructions impossible. This turns out not to be true.
- Clustering is an effective technique to simplify issue logic and wire delays as shown by many current designs.
Relevance of the work

The topic of complexity-efficient processor design continues to be relevant today, but microprocessor designers are well trained in it and it is difficult to appreciate big differences today.

In 1997, this study was more interesting as the mhz gap and transistor counts were wider.

- Pentium II: 300MHz, 7.5 Million Transistors
- Alpha 21164: 533MHz, 3.5 Million Transistors

This work has been very relevant in explaining the trade-off between hardware complexity and clock speed.
Thanks for your attention