Pipelining to Superscalar

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by John P. Shen
Updated by Mikko Lipasti
Pipelining to Superscalar

- Forecast
  - Limits of pipelining
  - The case for superscalar
  - Instruction-level parallel machines
  - Superscalar pipeline organization
  - Superscalar pipeline design
Generic Instruction Processing Pipeline

**Pipeline Stages:**

1. **IF:**
   - **ALU:** I-CACHE PC
   - **LOAD:** I-CACHE PC
   - **STORE:** I-CACHE PC
   - **BRANCH:** I-CACHE PC

2. **ID:**
   - **ALU:** DECODE
   - **LOAD:** DECODE
   - **STORE:** DECODE
   - **BRANCH:** DECODE

3. **OF:**
   - **ALU:** RD. REG
   - **LOAD:** RD. REG
   - **STORE:** RD. REG
   - **BRANCH:** RD. REG

4. **EX:**
   - **ALU:** ALU OP.
   - **LOAD:** RD. MEM.

5. **OS:**
   - **ALU:** ADDR. GEN.
   - **LOAD:** ADDR. GEN.
   - **STORE:** ADDR. GEN.
   - **BRANCH:** ADDR. GEN.

6. **WB:**
   - **ALU:** WR. REG
   - **LOAD:** WR. REG
   - **STORE:** WR. REG
   - **BRANCH:** WR. REG
   - **MEM:** WR. MEM
   - **WB:** WR. PC
   - **PC:** WR. PC
IBM RISC Experience  [Agerwala and Cocke 1987]

• Internal IBM study: Limits of a scalar pipeline?
• Memory Bandwidth
  • Fetch 1 instr/cycle from I-cache
  • 40% of instructions are load/store (D-cache)
• Code characteristics (dynamic)
  • Loads – 25%
  • Stores 15%
  • ALU/RR – 40%
  • Branches & jumps – 20%
    • 1/3 unconditional (always taken)
    • 1/3 conditional taken, 1/3 conditional not taken
IBM Experience

• Cache Performance
  • Assume 100% hit ratio (upper bound)
  • Cache latency: I = D = 1 cycle default

• Load and branch scheduling
  • Loads
    • 25% cannot be scheduled (delay slot empty)
    • 65% can be moved back 1 or 2 instructions
    • 10% can be moved back 1 instruction
  • Branches & jumps
    • Unconditional – 100% schedulable (fill one delay slot)
    • Conditional – 50% schedulable (fill one delay slot)
CPI Optimizations

• Goal and impediments
  • CPI = 1, prevented by pipeline stalls

• No cache bypass of RF, no load/branch scheduling
  • Load penalty: 2 cycles: 0.25 x 2 = 0.5 CPI
  • Branch penalty: 2 cycles: 0.2 x 2/3 x 2 = 0.27 CPI
  • Total CPI: 1 + 0.5 + 0.27 = 1.77 CPI

• Bypass, no load/branch scheduling
  • Load penalty: 1 cycle: 0.25 x 1 = 0.25 CPI
  • Total CPI: 1 + 0.25 + 0.27 = 1.52 CPI
More CPI Optimizations

• Bypass, scheduling of loads/branches
  • Load penalty:
    • 65% + 10% = 75% moved back, no penalty
    • 25% => 1 cycle penalty
    • 0.25 x 0.25 x 1 = 0.0625 CPI
  • Branch Penalty
    • 1/3 unconditional 100% schedulable => 1 cycle
    • 1/3 cond. not-taken, => no penalty (predict not-taken)
    • 1/3 cond. Taken, 50% schedulable => 1 cycle
    • 1/3 cond. Taken, 50% unschedulable => 2 cycles
    • 0.20 x [1/3 x 1 + 1/3 x 0.5 x 1 + 1/3 x 0.5 x 2] = 0.167
  • Total CPI: 1 + 0.063 + 0.167 = 1.23 CPI
Simplify Branches

• Assume 90% can be PC-relative
  • No register indirect, no register access
  • Separate adder (like MIPS R3000)
  • Branch penalty reduced

• Total CPI: $1 + 0.063 + 0.085 = 1.15$ CPI = 0.87 IPC

<table>
<thead>
<tr>
<th>PC-relative</th>
<th>Schedulable</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes (90%)</td>
<td>Yes (50%)</td>
<td>0 cycle</td>
</tr>
<tr>
<td>Yes (90%)</td>
<td>No (50%)</td>
<td>1 cycle</td>
</tr>
<tr>
<td>No (10%)</td>
<td>Yes (50%)</td>
<td>1 cycle</td>
</tr>
<tr>
<td>No (10%)</td>
<td>No (50%)</td>
<td>2 cycles</td>
</tr>
</tbody>
</table>

15% Overhead from program dependences
Limits of Pipelining

• IBM RISC Experience
  • Control and data dependences add 15%
  • Best case CPI of 1.15, IPC of 0.87
  • Deeper pipelines (higher frequency) magnify dependence penalties

• This analysis assumes 100% cache hit rates
  • Hit rates approach 100% for some programs
  • Many important programs have much worse hit rates
    • Later!
Processor Performance

Processor Performance = \frac{\text{Time}}{\text{Program}}

= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}

\text{(code size)} \quad \text{(CPI)} \quad \text{(cycle time)}

- In the 1980’s (decade of pipelining):
  - CPI: 5.0 => 1.15

- In the 1990’s (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)

- In the 2000’s (decade of multicore):
  - Marginal CPI improvement
Amdahl’s Law

- \( h \) = fraction of time in serial code
- \( f \) = fraction that is vectorizable
- \( v \) = speedup for \( f \)
- Overall speedup:

\[
Speedup = \frac{1}{1 - f + \frac{f}{v}}
\]
Revisit Amdahl’s Law

• Sequential bottleneck
• Even if v is infinite
  • Performance limited by nonvectorizable portion

\[
\lim_{v \to \infty} \frac{1}{1 - f + \frac{f}{v}} = \frac{1}{1 - f}
\]

No. of Processors

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>1</th>
<th>h</th>
<th>1 - h</th>
<th>1 - f</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pipelined Performance Model

- $g$ = fraction of time pipeline is filled
- $1-g$ = fraction of time pipeline is not filled (stalled)

![Diagram showing pipeline depth and time spent filled or not filled]

- $N$ represents the number of stages in the pipeline.
Pipelined Performance Model

- $g = \text{fraction of time pipeline is filled}$
- $1-g = \text{fraction of time pipeline is not filled (stalled)}$
Pipelined Performance Model

• Tyranny of Amdahl’s Law [Bob Colwell]
  • When $g$ is even slightly below 100%, a big performance hit will result
  • Stalled cycles are the key adversary and must be minimized as much as possible
Motivation for Superscalar
[Agerwala and Cocke]

Speedup jumps from 3 to 4.3 for \( N=6, f=0.8 \), but \( s=2 \) instead of \( s=1 \) (scalar)

Typical Range
Superscalar Proposal

• Moderate tyranny of Amdahl’s Law
  • Ease sequential bottleneck
  • More generally applicable
  • Robust (less sensitive to f)
  • Revised Amdahl’s Law:

\[
\text{Speedup} = \frac{1}{\left(1 - f\right) + \frac{f}{S + v}}
\]
Limits on Instruction Level Parallelism (ILP)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weiss and Smith [1984]</td>
<td>1.58</td>
</tr>
<tr>
<td>Sohi and Vajapeyam [1987]</td>
<td>1.81</td>
</tr>
<tr>
<td>Tjaden and Flynn [1970]</td>
<td>1.86 (Flynn’s bottleneck)</td>
</tr>
<tr>
<td>Tjaden and Flynn [1973]</td>
<td>1.96</td>
</tr>
<tr>
<td>Uht [1986]</td>
<td>2.00</td>
</tr>
<tr>
<td>Smith et al. [1989]</td>
<td>2.00</td>
</tr>
<tr>
<td>Jouppi and Wall [1988]</td>
<td>2.40</td>
</tr>
<tr>
<td>Johnson [1991]</td>
<td>2.50</td>
</tr>
<tr>
<td>Acosta et al. [1986]</td>
<td>2.79</td>
</tr>
<tr>
<td>Wedig [1982]</td>
<td>3.00</td>
</tr>
<tr>
<td>Butler et al. [1991]</td>
<td>5.8</td>
</tr>
<tr>
<td>Melvin and Patt [1991]</td>
<td>6</td>
</tr>
<tr>
<td>Wall [1991]</td>
<td>7 (Jouppi disagreed)</td>
</tr>
<tr>
<td>Kuck et al. [1972]</td>
<td>8</td>
</tr>
<tr>
<td>Riseman and Foster [1972]</td>
<td>51 (no control dependences)</td>
</tr>
<tr>
<td>Nicolau and Fisher [1984]</td>
<td>90 (Fisher’s optimism)</td>
</tr>
</tbody>
</table>
Superscalar Proposal

• Go beyond single instruction pipeline, achieve IPC > 1
• Dispatch multiple instructions per cycle
• Provide more generally applicable form of concurrency (not just vectors)
• Geared for sequential code that is hard to parallelize otherwise
• Exploit fine-grained or instruction-level parallelism (ILP)
Classifying ILP Machines

[Jouppi, DECWRL 1991]

- Baseline scalar RISC
  - Issue parallelism = IP = 1
  - Operation latency = OP = 1
  - Peak IPC = 1
Classifying ILP Machines

[Jouppi, DECWRL 1991]

- Superpipelined: cycle time = 1/m of baseline
  - Issue parallelism = IP = 1 inst / minor cycle
  - Operation latency = OP = m minor cycles
  - Peak IPC = m instr / major cycle (m x speedup?)

```
  IF  DE  EX  WB
  1  2  3  4  5  6
```

Classifying ILP Machines

[Jouppi, DECWRL 1991]

• Superscalar:
  • Issue parallelism = IP = \( n \) inst / cycle
  • Operation latency = OP = 1 cycle
  • Peak IPC = \( n \) instr / cycle (n x speedup?)

Diagram:

- IF
- DE
- EX
- WB

1 2 3 4 5 6 7 8 9
Classifying ILP Machines

[Jouppi, DECWRL 1991]
• VLIW: Very Long Instruction Word
  • Issue parallelism = IP = n inst / cycle
  • Operation latency = OP = 1 cycle
  • Peak IPC = n instr / cycle = 1 VLIW / cycle
Classifying ILP Machines

[Jouppi, DECWRL 1991]

• Superpipelined-Superscalar
  • Issue parallelism = IP = n inst / minor cycle
  • Operation latency = OP = m minor cycles
  • Peak IPC = n x m instr / major cycle
Superscalar vs. Superpipelined

- Roughly equivalent performance
  - If \( n = m \) then both have about the same IPC
  - Parallelism exposed in space vs. time
Superpipelining - Jouppi, 1989

essentially describes a pipelined execution stage

Jouppi’s base machine

Underpipelined machines cannot issue instructions as fast as they are executed

Underpipelined machine

Note - key characteristic of Superpipelined machines is that results are not available to M-1 successive instructions

Superpipelined machine
Superscalar Challenges

Instruction Flow

Memory Data Flow

Register Data Flow