Exposing More ILP

- These techniques were originally motivated by VLIW, which needs tons of ILP to work at all – but useful for superscalar/dynamic/speculative processors, as well.
- Software Techniques
  - Software Pipelining
  - Trace Scheduling
- Hardware/Software Technique
  - Predicated execution

Compiler support for ILP: Software Pipelining

- Observation: if iterations from loops are independent, then can get ILP by taking instructions from different iterations
- Software pipelining: reorganizes loops so that each iteration is made from instructions chosen from different iterations of the original loop

SW Pipelining Example

Unrolled 3 times

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LD F0,0(R1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ADDD F4,F0,F2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LD F0,-8(R1)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADDD F8,F6,F2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>LD F6,-8(R1)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADDD F10,-16(R1)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SD -8(R1),F8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>LD F10,-16(R1)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ADDD F12,F10,F2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SD -16(R1),F12</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>SUBI R1,R1,#24</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>BNEZ R1,LP</td>
<td></td>
</tr>
</tbody>
</table>

Compiler Support for ILP: Trace Scheduling

- Creates long basic blocks by finding long paths in the code
**Trace Scheduling**

- Parallelism across IF branches vs. LOOP branches
- Two steps:
  - **Trace Selection**
    - Find likely sequence of basic blocks (trace) of (statically predicted) long sequence of straight-line code
  - **Trace Compaction**
    - Squeeze trace into few VLIW instructions
    - Need bookkeeping code in case prediction is wrong

**Predication: HW support for More ILP**

- Avoid branch prediction by turning branches into *conditionally executed instructions*: (aka predicated instructions)
  
  ```
  add c, a, b (x) => if (x) then a = b + c else NOP
  ```
  
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instruction (even have multiple predicates)

  ```
  go_on: addi R2, R2, #8
  ```

**Predicated Execution**

- **Drawbacks to conditional instructions**
  - Still takes a clock & alu even if “annulled”
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline
  - Requires more operands! Typically only available as *conditional move*.

- **Advantages**
  - Eliminate prediction, misprediction
  - Longer basic blocks, ...

- Critical technology for VLIW, sw pipelining. Why?

**ILP in real code**

- Based on all kinds of ideal assumptions. Further limited by:
  - Realistic branch prediction
  - Finite renaming registers
  - Imperfect alias analysis for memory operations
### Window Size

- Renaming registers
- Memory aliasing
- Pentium Pro (II, III, etc.) microarchitecture

- **Uncommitted instructions**: 40
- **Unissued instructions**: 20
Pentium 4 microarchitecture

- Front-End BTB (4K Entries)
- Instruction TL R/Prefetcher
- Instruction Decoder
- Trace Cache BTB (512 Entries)
- Trace Cache (12K uops)
- ALU (caches and bypass)
- Memory Scheduler
- Integer/ Floating point ops Queue
- Integer Register File / Bypass Network
- FP Register / Bypass
- L1 Data Cache (8K/byte 4-way)
- 48GB/s

Pentium 4 front-end

- L2 Cache (256K Byte 8-way)
- 20-stage pipeline
- IA32 (x86) ISA translated to RISC-like uops
- Uops stored in trace cache
- Decode / retire 3 uops / cycle
- Execute 6 uops / cycle
- Dynamically scheduled
- Explicit Register Renaming
- Simultaneous Multithreading (hyper-threading)

Pentium 4 Summary

- 126 in-flight instructions (ROB size)
ILP Summary

• Parallelism is absolutely critical to modern computer system performance, but at a very fine level.
• Mechanisms that create, or expose parallelism: loop unrolling, software pipelining, code motion
• Mechanisms that allow the machine to exploit ILP: pipelining, superscalar, dynamic scheduling, speculative execution

Motivation

• Modern processors fail to utilize execution resources well.
• There is no single culprit.
• Attacking the problems one at a time (e.g., specific latency-tolerance solutions) always has limited effectiveness.
• However, a general latency-tolerance solution which can hide all sources of latency can have a large impact on performance.

Hardware Multithreading

Tullsen, Eggers, Emer, Levy, Lo, Stamm, Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, ISCA, 1996
Superscalar Execution

Simultaneous Multithreading

The Potential for SMT
Goals

We had three primary goals for this architecture:

1. Minimize the architectural impact on conventional superscalar design.
3. Achieve significant throughput gains with many threads.

A Conventional Superscalar Architecture

An SMT Architecture

Performance of the Naive Design
Bottlenecks of the Baseline Architecture

- Instruction queue full conditions (12-21% of cycles)
  - Lack of parallelism in the queue.
- Fetch throughput (4.2 instructions per cycle when queue not full)

Improving Fetch Throughput

- The fetch unit in an SMT architecture has two distinct advantages over a conventional architecture.
  1. Can fetch from multiple threads at once.
  2. Can choose which threads to fetch.

Improved Fetch Performance

- Fetching from 2 threads/cycle achieved most of the performance from multiple-thread fetch.
- Fetching from the thread(s) which have the fewest unissued instructions in-flight significantly increases parallelism and throughput.

Improved Performance

![Graph showing improved performance over baseline and unmodified superscalar with increasing number of threads.](image-url)
This SMT Architecture, then:

- Borrows heavily from conventional superscalar design.
- Minimizes the impact on single-thread performance
- Achieves significant throughput gains over the superscalar (2.5X, up to 5.4 IPC).

Commercial SMT

- Alpha 21464 (⃣)
- Intel Pentium 4 “hyper-threading” processor.
- IBM Power 5 – 2 cores, 2 SMT threads/core
- IBM Power 6 – again, 2 cores, 2 SMT threads/core
- Sun Niagara (2006) – 8 cores, 4 threads/core (SMT?)
- Sun Niagara 2 – 8 cores, 8 threads/core
- Intel Nehalem (core i7) 4-8 cores, 2 SMT threads/core