Lecture overview

- Repetition
  - Superscalar processor (our-of-order)
  - Dependencies
  - Loop unrolling
  - VLIW

- Today
  - Instruction fetch optimization
  - Thread level parallelism (TLP)
  - Multithreading
  - CMP design space exploration
    - Scientific paper
    - Vector processors (if time)
INSTRUCTION FETCHING
Instruction fetching

- Want to issue >1 instruction every cycle
- Several problems
  - Bandwidth / Latency
  - Determining which instructions
    - Jumps
    - Branches
- Integrated instruction fetch unit
Branch Target Buffer (BTB)

- To reduce miss-penalty
- PC is checked against entries in 1st column that stores addresses of known branches.
- If match is found there, it is a taken branch, and the predicted PC contains the prediction of the next PC after the branch, and fetching can begin immediately at that address.
Return Address Predictor

- Small buffer of return addresses acts as a stack
- Caches most recent return addresses
- Call ⇒ Push a return address on stack
- Return ⇒ Pop an address off stack & predict as new PC

Zero buffer entries implies that the standard branch prediction is used
Integrated Instruction Fetch (IF) Units

• Recent designs have implemented the fetch stage as a separate, autonomous unit
  – Multiple-issue IF in one simple pipeline stage is too complex

• An integrated fetch unit provides:
  – Branch prediction
  – Instruction prefetch
  – Instruction memory access and buffering
Limits to ILP

- Advances in compiler technology + significantly new and different hardware techniques *may* be able to overcome limitations assumed in studies
- However, unlikely such advances when coupled *with* *realistic hardware* will overcome these limits in near future
- How much ILP is available using *existing* mechanisms with increasing HW budgets?
Ideal HW Model

1. *Register renaming* – infinite virtual registers
   all register WAW & WAR hazards are avoided
2. *Branch prediction* – perfect; no mispredictions
3. *Jump prediction* – all jumps perfectly predicted
   
   $2 \& 3 \implies$ no control dependencies; perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis* – addresses known & a load can be moved before a store provided addresses not equal
   
   $1 \& 4$ eliminates all but the true data dependencies
5. *perfect caches*; 1 cycle latency for all instructions; unlimited instructions issued/clock cycle
Upper Limit to ILP: **Ideal** Machine

(Figure 3.26)

**SPEC92 benchmarks**

<table>
<thead>
<tr>
<th>Programs</th>
<th>Integer (18 - 60)</th>
<th>FP (75 - 150)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>54.8</td>
<td>118.7</td>
</tr>
<tr>
<td>espresso</td>
<td>62.6</td>
<td>150.1</td>
</tr>
<tr>
<td>li</td>
<td>17.9</td>
<td></td>
</tr>
<tr>
<td>fpppp</td>
<td>75.2</td>
<td></td>
</tr>
<tr>
<td>doducd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instruction window

• **Ideal** HW need to know **entire** code
• Obviously not practical
  – Register dependencies scales quadratically
• Window: The set of instructions examined for simultaneous execution
• How does the size of the window affect IPC?
  – Too small window ➔ Can’t see whole loops
  – Too large window ➔ Hard to implement
More Realistic HW: Window Impact

Assume can issue up to 64 instructions in one cycle
Thread Level Parallelism (TLP)

- ILP exploits *implicit* parallel operations within a loop or straight-line code segment
- TLP is *explicitly* represented by the use of multiple threads of execution that are inherently parallel
- Use multiple instruction streams to improve:
  1. Throughput of computers that run many programs
  2. Execution time of a single application implemented as a multi-threaded program (parallel program)
Multi-threaded execution

- Multi-threading: multiple threads share the functional units of 1 processor via overlapping
  - Must duplicate independent state of each thread e.g., a separate register file, PC and page table
  - Memory shared through virtual memory mechanisms
  - HW for fast thread switch; much faster than full process switch \( \approx 100\text{s to 1000s of clocks} \)

- When switch?
  - fine grained vs. coarse grained
Fine-Grained Multithreading

• Switches between threads on each instruction
  – Multiples threads interleaved
• Usually round-robin fashion, skipping stalled threads
• CPU must be able to switch threads every clock
• Hides both short and long stalls
• But slows down execution of individual threads
  – Thread ready to execute without stalls will be delayed by instructions from other threads
• Used on Sun’s Niagara (T1)
Coarse-Grained Multithreading

• Switch threads only on costly stalls (L2 cache miss)

• Advantages
  – No need for very fast thread-switching
  – Doesn’t slow down thread, since switches only when thread encounters a costly stall

• Disadvantage: hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
  – Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
  – New thread must fill pipeline before instructions can complete
Do both ILP and TLP?

- TLP and ILP exploit two different kinds of parallel structure in a program
- Can a high-ILP processor also exploit TLP?
  - Functional units often idle because of stalls or dependences in the code
- Can TLP be a source of independent instructions that might reduce processor stalls?
- Can TLP be used to employ functional units that would otherwise lie idle with insufficient ILP?
- => Simultaneous Multi-threading (SMT)
  - Intel: Hyper-Threading
Simultaneous Multi-threading

### One thread, 8 units

<table>
<thead>
<tr>
<th>Cycle</th>
<th>M</th>
<th>M</th>
<th>FX</th>
<th>FX</th>
<th>FP</th>
<th>FP</th>
<th>BR</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Two threads, 8 units

<table>
<thead>
<tr>
<th>Cycle</th>
<th>M</th>
<th>M</th>
<th>FX</th>
<th>FX</th>
<th>FP</th>
<th>FP</th>
<th>BR</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition
Simultaneous Multi-threading (SMT)

- A dynamically scheduled processor already has many HW mechanisms to support multi-threading
  - Large set of virtual registers
    - Virtual = not all visible at ISA level
    - Register renaming (Tomasulo, course TDT4255)
  - Dynamic scheduling
- Just add a per thread renaming table and keeping separate PCs
  - Independent commitment can be supported by logically keeping a separate reorder buffer for each thread
Multi-threaded categories

<table>
<thead>
<tr>
<th>Time (processor cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Thread 1</td>
</tr>
<tr>
<td>Thread 2</td>
</tr>
</tbody>
</table>

Simultaneous Multithreading
Design Challenges in SMT

- SMT makes sense only with fine-grained implementation
  - How to reduce the impact on single thread performance?
    - Give priority to one or a few preferred threads
- Large register file needed to hold multiple contexts
- Not affecting clock cycle time, especially in
  - Instruction issue - more candidate instructions need to be considered
  - Instruction completion - choosing which instructions to commit may be challenging
- Ensuring that cache and TLB conflicts generated by SMT do not degrade performance
  - “Threads should be good neighbours”
  - QoS and fairness
UltraSPARC T1 ("Niagara")

- Target: Commercial server applications
  - High thread level parallelism (TLP)
    - Large numbers of parallel client requests
  - Low instruction level parallelism (ILP)
    - High cache miss rates
    - Many unpredictable branches
- Approach: Multicore, Fine-grain multithreading, Simple pipeline, Small L1 caches, Shared L2
T1 processor - “logical” overview

1.2 GHz at \(\approx\) 72W typical, 79W peak power consumption
**T1 pipeline / 4 threads**

- Single issue, in-order, 6-deep pipeline: F, S, D, E, M, W
- **Shared** units:
  - L1 cache, L2 cache
  - TLB
  - Exec. units
  - pipe registers
- **Separate** units:
  - PC
  - instruction buffer
  - reg file
  - store buffer

Note: thread select logic & thread state storage
UltraSPARC T1 and T2 available as open-source hardware

http://en.wikipedia.org/wiki/OpenSPARC

OpenSPARC Overview

In March 2006, the complete design of Sun Microsystems' UltraSPARC T1 microprocessor was released in open-source form, it was named OpenSPARC T1. In early 2008, its successor, OpenSPARC T2, was also released in open-source form. These were the first (and still only) 64-bit microprocessors ever open-sourced. They were also the first (and still only) CMT (chip multithreaded) microprocessors ever open-sourced. Both designs are freely available to anyone under open-source licenses. These downloads include not only the processor design source code but also simulation tools, design verification suites, Hypervisor source code, and other helpful tools. Variants that easily synthesize for FPGA
T1 Multithreading Unicore Performance, (fig 3.31)
Not Ready Breakdown (fig 3.32)

- Cache effects responsible for 50% - 75% of waiting
- SPECJBB has a higher amount of pipeline-delay because it has a higher branch frequency
## CPI Breakdown of Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Per Thread CPI</th>
<th>Per core CPI</th>
<th>Effective CPI for 8 cores</th>
<th>Effective IPC for 8 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-C</td>
<td>7.20</td>
<td>1.80</td>
<td>0.23</td>
<td>4.4</td>
</tr>
<tr>
<td>SPECJBB</td>
<td>5.60</td>
<td>1.40</td>
<td>0.18</td>
<td>5.7</td>
</tr>
<tr>
<td>SPECWeb99</td>
<td>6.60</td>
<td>1.65</td>
<td>0.21</td>
<td>4.8</td>
</tr>
</tbody>
</table>

**Figure 3.33**
Exploring the Design Space of Future CMPs


Focus on simulation-based research method
Summary

- **Design space exploration** of chip multiprocessors
  - area and performance tradeoffs
  - how many processing cores?
  - core complexity
    - in-order or out-of-order
  - how big on-chip caches?
  - limited off-chip bandwidth will become an increasing problem
    - need for larger caches to reduce bandwidth demand
  - applications with different access patterns require different CMP designs to maximize throughput

See also: *Computational Computer Architecture Research at NTNU* by Magnus Jahre and Lasse Natvig (ERCIM News 81)
Introduction

• Job throughput in servers
• Future CMPs will have a larger # of cores
  – superscalar paradigm is reaching its limits
    • little use of more than 4 or 5-way issue superscalar processors
    • global wire delays will limit the area of the chip that is useful for a single conventional processing core
• Considers CMOS technology scaled to ultra small (35 nanometer) devices (Note; paper is from 2001)
• Power consumption not considered
Technology model

Chip-Multiprocessor

Processor

Icache  Dcache

L2 cache

Memory Channel
Memory channel

- L2 cache connected to off-chip memory via a set of distributed memory channels
  - limited resource
    - time multiplexing
    - channel contention
Method

• throughput-oriented workloads with no sharing of data among tasks
• technology independent area models
  – found empirically
  – core area and cache area measured in cache byte equivalents (CBE)
• study the relative costs in area versus the associated performance gains --- maximize performance per unit area for future technology generations
## Processor models

<table>
<thead>
<tr>
<th>L2 cache size</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In-order</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128KB</td>
<td>0.20</td>
<td>0.21</td>
<td>0.21</td>
</tr>
<tr>
<td>256KB</td>
<td>0.23</td>
<td>0.24</td>
<td>0.25</td>
</tr>
<tr>
<td>512KB</td>
<td>0.24</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>1MB</td>
<td>0.27</td>
<td>0.28</td>
<td>0.29</td>
</tr>
<tr>
<td><strong>Out-of-order</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128KB</td>
<td>0.26</td>
<td>0.31</td>
<td>0.33</td>
</tr>
<tr>
<td>256KB</td>
<td>0.31</td>
<td>0.38</td>
<td>0.40</td>
</tr>
<tr>
<td>512KB</td>
<td>0.32</td>
<td>0.39</td>
<td>0.41</td>
</tr>
<tr>
<td>1MB</td>
<td>0.38</td>
<td>0.47</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Table 1. Harmonic means of IPCs for six processor models.

Most area-efficient
## Processor model parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>( P_{IN} )</th>
<th>( P_{OUT} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction issue</td>
<td>in-order</td>
<td>out-of-order</td>
</tr>
<tr>
<td>Issue width</td>
<td>dual-issue</td>
<td>quad-issue</td>
</tr>
<tr>
<td>Instruction window (entries)</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>Load/store queue (entries)</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>bimodal (2K)</td>
<td>2 level (16K)</td>
</tr>
<tr>
<td>Number of integer ALUs</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Number of floating-point ALUs</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Estimated core area (CBE)</td>
<td>50 KB</td>
<td>250 KB</td>
</tr>
<tr>
<td>L1 Instruction cache</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>L1 Data cache</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>Total area (core + I/D caches)</td>
<td>114 KB</td>
<td>314 KB</td>
</tr>
</tbody>
</table>

**Table 2. Processor model parameters.**
What is harmonic mean

- Harmonic mean
  - In mathematics, the harmonic mean is one of several methods of calculating an average. Typically, it is appropriate for situations when the average of rates is desired.
  - The harmonic mean \((H)\) of the positive real numbers \(a_1,\ldots,a_n\) is defined to be

\[
H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots + \frac{1}{a_n}}.
\]

[Wikipedia]
Techn. scaling and number of cores

<table>
<thead>
<tr>
<th>Gate length</th>
<th>CBE (Megabytes)</th>
<th>$\lambda^2$ area</th>
<th>$P_{IN}$</th>
<th>$P_{OUT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100nm</td>
<td>7.6</td>
<td>$1.60e+11$</td>
<td>68</td>
<td>24</td>
</tr>
<tr>
<td>70nm</td>
<td>15.5</td>
<td>$3.26e+11$</td>
<td>139</td>
<td>50</td>
</tr>
<tr>
<td>50nm</td>
<td>30.5</td>
<td>$6.40e+11$</td>
<td>273</td>
<td>99</td>
</tr>
<tr>
<td>35nm</td>
<td>61.9</td>
<td>$1.30e+12$</td>
<td>556</td>
<td>201</td>
</tr>
</tbody>
</table>

Table 3. Total Chip Area.

• Assumes
  – a large fixed die of $400 \text{ mm}^2$ (20 x 20 mm)
  – per core:
    • 32 KB L1 I-cache and 32 KB L1 D-cache
    • no on-chip L2 cache
The I/O pin problem

- # I/O signaling
  - pins limited by physical technology
  - speeds have not increased at the same rate as processor clock rates

- Projections
  - from ITRS (International Technology Roadmap for Semiconductors)
Server workload

- maximize throughput
  - assume multiprogramming mode, i.e. independent threads, or several independent tasks to be executed
- $P_i =$ performance of core $i$
- $N_c =$ number of cores

\[ P_{cmp} = \sum_{i=1}^{N_c} P_i \]
Application characteristics

- 10 SPEC-2000 app’s + sphinx
- wide range of memory system behaviour, three groups
  - Processor-bound
  - Cache-sensitive
  - Bandwidth-bound
- applications move among these three domains as the processor, cache, and bandwidth capacities are changed
Experiments

• Simplescalar tool set
• Effect of cache size on cache access latency given by the eCacti tool
• SimpleScalar modification to CMP
  – multiprogrammed SPEC workload
  – sharing of memory channels
  – DRAM as Rambus
  – avoid cold-start effects
    • skip first 5 billion instructions
  – then simulate 200 million instructions in detail for every application