Csci 211 Computer System Architecture

Limits on ILP and Simultaneous Multithreading

Xiuzhen Cheng
Department of Computer Sciences
The George Washington University

Adapted from the slides by Dr. David Patterson @ UC Berkeley

Review from Last Time

- Interest in multiple-issue because wanted to improve performance without affecting uniprocessor programming model
- Taking advantage of ILP is conceptually simple, but design problems are amazingly complex in practice
- Conservative in ideas, just faster clock and bigger
- Processors of last 5 years (Pentium 4, IBM Power 5, AMD Opteron) have the same basic structure and similar sustained issue rates (3 to 4 instructions per clock) as the 1st dynamically scheduled, multiple-issue processors announced in 1995
  - Clocks 10 to 20X faster, caches 4 to 8X bigger, 2 to 4X as many renaming registers, and 2X as many load-store units
  - Performance 8 to 16X
- Peak v. delivered performance gap increasing

Outline

- Review
- Limits to ILP (another perspective)
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading
- Head to Head: VLIW vs. Superscalar vs. SMT
- Commentary
- Conclusion

Limits to ILP

- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?

Overcoming Limits

- 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

Limits to ILP

Initial HW Model here; MIPS compilers.

Assumptions for ideal/perfect machine to start:

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 (returns, case statements)
2 & 3 => no control dependencies; perfect speculation
4. Memory-address alias analysis – addresses known & a load can be moved before a store provided addresses not equal; 1&4 eliminates all but RAW
Also: perfect caches; 1 cycle latency for all instructions (FP*,*); unlimited instructions issued/clock cycle;
Limits to ILP HW Model comparison

<table>
<thead>
<tr>
<th>New Model</th>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions Issued per clock</td>
<td>Infinite</td>
<td>Infinite</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>Infinite, 2K, 512, 128, 32</td>
<td>Infinite</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>Infinite</td>
<td>Infinite</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
<tr>
<td>Memory Alias Analysis</td>
<td>Perfect</td>
<td>??</td>
</tr>
</tbody>
</table>

More Realistic HW: Window Impact

Change from Infinite Window to 2048, 512, 128, 32 FP: 9 - 150

Upper Limit to ILP: Ideal Machine

Programs

Perfect
Selective predictor
Standard 2-bit
Static
Profile BHT (512)
Tournament
Perfect
No prediction

More Realistic HW: Branch Impact

Change from Infinite Window to a window size of 2048 and maximum issue of 64 instructions per clock cycle

Branch prediction is critical!
**Misprediction Rates**

For conditional branches in the SPEC92 subset:
1. Branch prediction is critical!

<table>
<thead>
<tr>
<th>Misprediction Rate</th>
<th>tomcat</th>
<th>doduc</th>
<th>fpppp</th>
<th>li</th>
<th>espresso</th>
<th>gcc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>5%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>10%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>15%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>20%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>25%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
<tr>
<td>30%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>18%</td>
<td>18%</td>
</tr>
</tbody>
</table>

**Limits to ILP HW Model comparison**

<table>
<thead>
<tr>
<th>New Model</th>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions Issued per clock</td>
<td>64</td>
<td>Infinite</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>2048</td>
<td>Infinite</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>Infinite v. 256, 128, 64, 32, none</td>
<td>88 integer + 88 Fl. Pt.</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>8K two level tournament</td>
<td>Perfect Tournament Branch Predictor</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
<tr>
<td>Memory Alias</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

**More Realistic HW: Renaming Register Impact (N int + N fp)**

<table>
<thead>
<tr>
<th>IPC</th>
<th>gcc</th>
<th>espresso</th>
<th>x</th>
<th>Program</th>
<th>Change 2048 instr window, 64 instr issue, 8K 2 level Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>Perfect</td>
</tr>
<tr>
<td>5%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>Global/Stack perf; Inspec.</td>
</tr>
<tr>
<td>10%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>None</td>
</tr>
<tr>
<td>15%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>None</td>
</tr>
<tr>
<td>20%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>None</td>
</tr>
<tr>
<td>25%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>None</td>
</tr>
<tr>
<td>30%</td>
<td>5%</td>
<td>14%</td>
<td>23%</td>
<td>18%</td>
<td>None</td>
</tr>
</tbody>
</table>

**Limits to ILP HW Model comparison**

<table>
<thead>
<tr>
<th>New Model</th>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions Issued per clock</td>
<td>64</td>
<td>Infinite</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>2048</td>
<td>Infinite</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>256 Int + 256 FP</td>
<td>88 integer + 88 Fl. Pt.</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>8K two level tournament</td>
<td>Perfect Tournament Branch Predictor</td>
</tr>
<tr>
<td>Cache</td>
<td>64K, 32KD, 1.92MB L2, 36 MB L3</td>
<td>Perfect</td>
</tr>
<tr>
<td>Memory Alias</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

**Outline**

- Review
- Limits to ILP (another perspective)
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading
- Head to Head: VLIW vs. Superscalar vs. SMT
- Commentary
- Conclusion
How to Exceed ILP Limits of this study?

- These are not laws of physics; just practical limits for today, and perhaps overcome via research
- Compiler and ISA advances could change results
- WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage
  - Can get conflicts via allocation of stack frames as a called procedure reuses the memory addresses of a previous frame on the stack

HW v. SW to increase ILP

- Memory disambiguation: HW best
- Speculation:
  - HW best when dynamic branch prediction better than compile time prediction
  - Exceptions easier for HW
  - HW doesn't need bookkeeping code or compensation code
  - Very complicated to get right
- Scheduling: SW can look ahead to schedule better
- Compiler independence: does not require new compiler, recompilation to run well

Performance beyond single thread ILP

- There can be much higher natural parallelism in some applications (e.g., Database or Scientific codes)
- Explicit Thread Level Parallelism or Data Level Parallelism
- Thread: process with own instructions and data
  - thread may be a process part of a parallel program of multiple processes, or it may be an independent program
  - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
- Data Level Parallelism: Perform identical operations on data, and lots of data

Thread Level Parallelism (TLP)

- ILP exploits implicit parallel operations within a loop or straight-line code segment
- TLP explicitly represented by the use of multiple threads of execution that are inherently parallel
- Goal: Use multiple instruction streams to improve 1. Throughput of computers that run many programs 2. Execution time of multi-threaded programs
- TLP could be more cost-effective to exploit than ILP

New Approach: Mulithreaded Execution

- Multithreading: multiple threads to share the functional units of 1 processor via overlapping
  - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch ~ 100s to 1000s of clocks
- When switch?
  - Alternate instruction per thread (fine grain)
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

Fine-Grained Multithreading

- Switches between threads on each instruction, causing the execution of multiple threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads at every clock
- Advantage is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used on Sun’s Niagara (will see later)
Course-Grained Multithreading

• Switches threads only on costly stalls, such as L2 cache misses
• Advantages
  – Relieves need to have very fast thread-switching
  – Doesn’t slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
• Disadvantage is 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
• Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill << stall time
• Used in IBM AS/400

Do both ILP and TLP?

• TLP and ILP exploit different kinds of parallel structure in a program
• Could a processor oriented at ILP to exploit TLP?
  – Functional units are often idle in data path designed for ILP because of either stalls or dependences in the code
• Could the TLP be used as a source of independent instructions that might keep the processor busy during stalls?
• Could TLP be used to employ the functional units that would otherwise lie idle when insufficient ILP exists?

Simultaneous Multithreading (SMT)

• Simultaneous multithreading (SMT): insight that dynamically scheduled processor already has many HW mechanisms to support multithreading
  – Large set of virtual registers that can be used to hold the register sets of independent threads
  – Register renaming provides unique register identifiers, so instructions from multiple threads can be mixed in datapath without confusing sources and destinations across threads
  – Out-of-order completion allows the threads to execute out of order, and get better utilization of the HW
• Just adding a per thread renaming table and keeping separate PCS
  – Independent commitment can be supported by logically keeping a separate reorder buffer for each thread

Multithreaded Categories
Design Challenges in SMT

- Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance?
  - A preferred thread approach sacrifices neither throughput nor single-thread performance?
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput, when preferred thread stalls

- Larger register file needed to hold multiple contexts
  - 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

Initial Performance of SMT

- Pentium 4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark

- Running on Pentium 4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average was 1.20
  - Most gained some
  - FP/Int. apps had most cache conflicts and least gains

Performance on SPECint2000

<table>
<thead>
<tr>
<th>Processor</th>
<th>Micro architecture</th>
<th>Fetch / Issue / Execute</th>
<th>PU</th>
<th>Clock Rate (GHz)</th>
<th>Transistors /mm^2</th>
<th>Power /Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium 4 Extreme</td>
<td>Speculative dynamically scheduled; deeply pipelined; SMT</td>
<td>3/3/4</td>
<td>7 int. 1 FP</td>
<td>3.8</td>
<td>125 M 122 mm^2</td>
<td>115 W</td>
</tr>
<tr>
<td>AMD Athlon 64 FX-57</td>
<td>Speculative dynamically scheduled</td>
<td>3/3/4</td>
<td>6 int. 3 FP</td>
<td>2.8</td>
<td>114 M 115 mm^2</td>
<td>104 W</td>
</tr>
<tr>
<td>IBM Power5 (1 CPU only)</td>
<td>Speculative dynamically scheduled; SMT; 2 CPU cores/chip</td>
<td>8/4/8</td>
<td>6 int. 2 FP</td>
<td>1.9</td>
<td>200 M 300 mm^2 (est.)</td>
<td>80W (est.)</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>Statically scheduled VLIW-style</td>
<td>6/5/11</td>
<td>9 int. 2 FP</td>
<td>1.6</td>
<td>592 M 453 mm^2</td>
<td>130 W</td>
</tr>
</tbody>
</table>

Performance on SPECfp2000

Normalized Performance: Efficiency
No Silver Bullet for ILP

• No obvious leader in performance
• The AMD Athlon leads on SPECInt performance followed by the Pentium 4, Itanium 2, and Power5
• Itanium 2 and Power5, which perform similarly on SPECFP, clearly dominate the Athlon and Pentium 4 on SPECFP
• Itanium 2 is the most inefficient processor both for FI, PT, and integer code for all but one efficiency measure (SPECFP/Watt)
• Athlon and Pentium 4 both make good use of transistors and area in terms of efficiency,
• IBM Power5 is the most effective user of energy on SPECFP and essentially tied on SPECINT

Limits to ILP

• Doubling issue rates above today’s 3-6 instructions per clock, say to 6 to 12 instructions, probably requires a processor to
  – issue 3 or 4 data memory accesses per cycle,
  – resolve 2 or 3 branches per cycle,
  – rename and access more than 20 registers per cycle, and
  – fetch 12 to 24 instructions per cycle.
• The complexities of implementing these capabilities is likely to mean sacrifices in the maximum clock rate
  – E.g. widest issue processor is the Itanium 2, but it also has the slowest clock rate, despite the fact that it consumes the most power!

Limits to ILP

• Most techniques for increasing performance increase power consumption
• The key question is whether a technique is energy efficient: does it increase power consumption faster than it increases performance?
• Multiple issue processors techniques all are energy inefficient:
  1. Issuing multiple instructions incurs some overhead in logic that grows faster than the issue rate grows
  2. Growing gap between peak issue rates and sustained performance
• Number of transistors switching = f(peak issue rate), and performance = f(sustained rate),
  growing gap between peak and sustained performance ➔ increasing energy per unit of performance

Commentary

• Itanium architecture does not represent a significant breakthrough in scaling ILP or in avoiding the problems of complexity and power consumption
• Instead of pursuing more ILP, architects are increasingly focusing on TLP implemented with single-chip multiprocessors
• In 2000, IBM announced the 1st commercial single-chip, general-purpose multiprocessor, the Power4, which contains 2 Power3 processors and an integrated L2 cache
  – Since then, Sun Microsystems, AMD, and Intel have switch to a focus on single-chip multiprocessors rather than more aggressive unprocessors.
• Right balance of ILP and TLP is unclear today
  – Perhaps right choice for server market, which can exploit more TLP, may differ from desktop, where single-thread performance may continue to be a primary requirement

And in conclusion ...

• Limits to ILP (power efficiency, compilers, dependencies …) seem to limit to 3 to 6 issue for practical options
• Explicitly parallel (Data level parallelism or Thread level parallelism) is next step to performance
• Coarse grain vs. Fine grained multithreading
  – Only on big stall vs. every clock cycle
• Simultaneous Multithreading if fine grained multithreading based on Out-Of-Order superscalar microarchitecture
  – Instead of replicating registers, reuse rename registers
• Itanium/EPIC/VLIW is not a breakthrough in ILP
• Balance of ILP and TLP decided in marketplace