Outline

- Limits to ILP
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading

Limits to ILP

- Study conclusions conflict on amount available
  - Different benchmarks (vectorized Fortran FP vs. integer C programs)
  - Assumptions are made
    » Hardware sophistication
    » Compiler sophistication
- How much ILP can we expect using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on the processor performance curve?

Overcoming Limits - What do we need??

- 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

- Determine maximum limit on ILP given an ideal/perfect machine. Look at how each effect maximum ILP
  - Assumptions:
    1. Register renaming
       - infinite virtual registers ⇒ all register WAR & WAR hazards are avoided
    2. Branch prediction
       - perfect; no mispredictions
    3. Jump prediction
       - all jumps perfectly predicted (returns, case statements)
    4. Memory-address alias analysis
       - addresses known & a load can be moved before a store provided addresses not equal
    5. Perfect caches
    6. 1 cycle latency for all instructions (FP *,/)
    7. unlimited instructions issued/clock cycle;
    - 2 & 3 ⇒ no control dependencies; perfect speculation & an unbounded buffer of instructions available
    - 1&4 eliminates all but RAW

Upper Limit to ILP: Ideal Machine
(Figure 3.1)

Everything infinite or perfect

<table>
<thead>
<tr>
<th>Instructions Per Clock</th>
<th>Integer: 18 - 60</th>
<th>FP: 75 - 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>54.8</td>
<td>75.2</td>
</tr>
<tr>
<td>espresso</td>
<td>62.6</td>
<td>118.7</td>
</tr>
<tr>
<td>li</td>
<td>17.9</td>
<td>150.1</td>
</tr>
<tr>
<td>fpppp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doducd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomatcv</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Limits to ILP HW Model comparison

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions Issued</td>
<td>Infinite</td>
<td>4</td>
</tr>
<tr>
<td>Instruction Window</td>
<td>Infinite</td>
<td>200</td>
</tr>
<tr>
<td>reneging Registers</td>
<td>Infinite</td>
<td>48 integer + 40 Fl. Pt.</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Perfect</td>
<td>2% to 6% misprediction (Tournament Branch Predictor)</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>64KI, 32KD, 1.92MB L2, 36 MB L3</td>
</tr>
<tr>
<td>Memory Alias Analysis</td>
<td>Perfect</td>
<td>??</td>
</tr>
</tbody>
</table>
More Realistic HW: Window Impact
Figure 3.2

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

Integer: 8 - 63

More Realistic HW: Branch Impact
Figure 3.3

FP: 15 - 45

Integer: 6 - 12

Limits to ILP HW Model comparison

<table>
<thead>
<tr>
<th></th>
<th>New Model</th>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions</td>
<td>64</td>
<td>Infinite</td>
<td>4</td>
</tr>
<tr>
<td>Issued per</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>2048</td>
<td>Infinite</td>
<td>200</td>
</tr>
<tr>
<td>Window Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Renaming</td>
<td>Infinite</td>
<td>Infinite</td>
<td>48 integer +</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40 Fl. Pt.</td>
</tr>
<tr>
<td>(Vary) Branch</td>
<td>Perfect vs. 8K</td>
<td>Perfect</td>
<td>2% to 6% misprediction</td>
</tr>
<tr>
<td>Prediction</td>
<td>Tournament vs. 512</td>
<td>(Tournament Branch Predictor)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2-bit vs. profile vs. none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>Perfect</td>
<td>64KI, 32KD, 1.92MB L2, 36 MB L3</td>
</tr>
<tr>
<td>Memory Alias</td>
<td>Perfect</td>
<td>Perfect</td>
<td>??</td>
</tr>
</tbody>
</table>

Branch Misprediction Rates
### Limits to ILP HW Model comparison

<table>
<thead>
<tr>
<th></th>
<th>New Model</th>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions</td>
<td>64</td>
<td>Infinite</td>
<td>4</td>
</tr>
<tr>
<td>Issued per clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction Window</td>
<td>2048</td>
<td>Infinite</td>
<td>200</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Vary) Renaming</td>
<td>Infinite v. 256, 128, 64, 32, none</td>
<td>Infinite</td>
<td>48 integer + 40 Fl. Pt.</td>
</tr>
<tr>
<td>Registers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>8K 2-bit</td>
<td>Perfect</td>
<td>Tournament Branch Predictor</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>Perfect</td>
<td>64KI, 32KD, 1.92MB L2, 36 MB L3</td>
</tr>
<tr>
<td>Memory Alias</td>
<td>Perfect</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

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

![Graph showing IPC vs Programs]

**FP: 11 - 45**

**Integer: 5 - 15**

#### More Realistic HW: Memory Address Alias Impact

![Graph showing IPC vs Programs]

**FP: 4 - 45**

(Fortran, no heap)

**Integer: 4 - 9**
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 (no restrictions)</td>
<td>Infinite</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>Infinite vs. 256, 128, 64, 32</td>
<td>Infinite</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>64 Int + 64 FP</td>
<td>Infinite</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>1K 2-bit</td>
<td>Perfect</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
<tr>
<td>Memory Alias</td>
<td>HW disambiguation</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

Realistic HW: Window Impact
(Figure 3.7)

Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window

Infinite vs. 256, 128, 64, 32

Infinite

4

100
200
300
400
500
600

10/8/08
17

Outline

• Limits to ILP
• Thread Level Parallelism
• Multithreading
• Simultaneous Multithreading

How to Exceed ILP Limits of this study?

• These were practical limits for modern computers
  – These are not laws of physics
  – Perhaps overcome via research
• Compiler and ISA advances could change results
• Memory aliasing - WAR and WAW hazards through memory
  – eliminated WAW and WAR hazards through register renaming, but not in memory usage
  » Research on predicting address conflicts should help
  – Can get conflicts via allocation of stack frames as a called procedure reuses the memory addresses of a previous frame on the stack

Infinite 256 128 64 32 16 8 4

10/8/08
18

10/8/08
19

10/8/08
20
Which is better for increasing ILP: HW vs. SW

- Memory disambiguation:
  - HW best
  - Compile time pointer analysis is hard
- Speculation:
  - HW best when dynamic branch prediction better than compile time prediction
  - Profiling is not good enough
  - Exceptions easier for HW
  - HW doesn’t need bookkeeping code or compensation code
  - Speculation is very complicated to get right
  - Execution is hard enough to get right without speculation
  - Speculation leads to many special cases
  - Hard to get right
- Scheduling
  - SW can look ahead to schedule better, look beyond current PC
- Advantage for HW based:
  - Compiler independence: does not require new compiler, recompilation to run well

Performance beyond single thread ILP - How do we progress?

- Some applications have high natural parallelism
  - Database, searching
- Explicit Thread Level Parallelism or Data Level Parallelism
  - Thread: process with own instructions and data
  - Part of parallel program (same address space) 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
    - Graphics processing
    - ATI - 130,000+ threads

Thread Level Parallelism (TLP)

- ILP vs. 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
- TLP Goal: Use multiple instruction streams to improve...
  - Throughput of computers that run many programs
  - Execution time of multi-threaded programs
- TLP could be more cost-effective to exploit than ILP

Outline

- Limits to ILP
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading
New Approach: Mulithreaded Execution

• Attempt better performance while reusing a lot of existing hardware
• Mulithreading: multiple threads to share the functional units of 1 processor via overlapping
• What do we need to support it:
  – duplicate independent state of each thread
    » a separate copy of register file
    » a separate PC
    » a separate page table (if separate programs)
  – Memory shared through the virtual memory mechanisms, which already support multiple processes
  – HW for fast thread switch
    » Needs to be faster than full process switch ~ 100s to 1000s of clocks

• When to switch?
  – Fine grained
    » Alternate instruction per thread - switch on each clock cycle
  – Coarse grained
    » When a thread is stalled, perhaps for a cache miss, another thread can be executed
• Both switching methods allow stalls to be hidden by doing work for another thread

Fine-Grained Multithreading

• Switches between threads on each instruction, causing the execution of multiple threads to be interleaved
• CPU must be able to switch threads every clock
• Usually done in a round-robin fashion, skipping any stalled threads
• Advantage
  – can hide both short and long stalls, since instructions from other threads executed when one thread stalls
• Disadvantage
  – 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

Course-Grained Multithreading

• More conservative
  – Switches threads only on costly stalls, such as L2 cache misses
• Advantages
  – Relieves need to have very fast thread-switching
    » Easier to build
  – Doesn’t slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
• Disadvantage
  – hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
    » On switch, pipeline is emptied. Need to refill for new thread
  – Doesn’t switch on short stalls, can’t hide those
• 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
For most apps, most execution units lie idle.


Do both ILP and TLP?

- TLP and ILP exploit two different kinds of parallel structure in a program
- Could a processor oriented for ILP be used 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?

Outline

- Limits to ILP
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading

Simultaneous Multi-threading ...
Simultaneous Multithreading (SMT)

- **Simultaneous multithreading (SMT)**
  - 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 execution allows the threads to execute out of order, and get better utilization of the HW
  - Added support
    - Different threads can be scheduled together on same clock cycle
    - Per thread renaming table
    - Separate PCs
    - Independent commitment can be supported by logically keeping a separate reorder buffer for each thread

Design Challenges in SMT

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

- **Larger 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**

Multithreaded Categories

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 superscalar microarchitecture**
  - Instead of replicating registers, reuse rename registers

- **Balance of ILP and TLP decided in marketplace**