Outline

- Limit of ILP on a Wide Issue Processor
- Simultaneous Multithreading
- Comparison of Architectures

Available ILP

```c
for(i=0;i<100;i++){
    A[i]=B[i]+C[i]
}
```

- Available ILP: Program dependent
- 1\textsuperscript{st} Case: ILP of 100, Assume A[i], B[i], C[i] in regs or Loop is unrolled 100 time
- 2\textsuperscript{nd} Case, ILP is limited..you can’t do in efficiently..(inherent data dependency )

Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. Integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
  - Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  - Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  - Motorola Altivec: 128 bit ints and FPs
  - Superpipe Multimedia ops, etc.

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

Ideal (Initial) HW Model and Ideal compilers.
Assumptions for ideal/perfect machine to start:
- Window size
  - infinite (compiler can see whole program for analysis ), infinite issue count
  - Unlimited Resource: 1 cycle latency for all instructions (FP +,,)
    unlimited instructions issued/clock cycle
- Register renaming – infinite virtual registers
  => all register WAW & WAR hazards are avoided
- Branch prediction – perfect; no miss predictions, no control dependencies; perfect speculation & an unbounded buffer of instructions available
- Memory-address alias analysis – all memory access addresses known & a load can be moved before a store provided addresses not equal
- Perfect caches: All memory access takes 1 cycle
### Limits to ILP HW Model comparison

<table>
<thead>
<tr>
<th>Model</th>
<th>Power 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions Issued per clock</td>
<td>Infinite</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>Infinite</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>Infinite</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Perfect</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
</tr>
<tr>
<td>Memory Alias Analysis</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

### Issue Window

- Suppose Issue window size is 4
- All are register instructions
- Number of possible dependency
  - \( R_{d1}(9) + R_{s1}(9) + R_{s2}(9) \) \( \vdash 9 \) test
  - \( +R_{d6}(6) + R_{s1}(6) + R_{s2}(6) \) \( \vdash 6 \) test
  - \( +R_{d3}(3) + R_{s1}(3) + R_{s2}(3) \) \( \vdash 3 \) test
  - \( \text{Total} = (4-1)^3 + (4-2)^3 + (4-3)^3 = 18 \text{ test} \)
- Window size \( N \) require
  - \( (N-1)^3 + (N-2)^3 + \ldots + 3 = 3(1+2+\ldots+N-1) = 3(N^2-N)/2 \)

### More Realistic HW: Window Impact

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

### 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>4</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>Infinite, 2K, 512, 128, 32</td>
<td>200</td>
</tr>
<tr>
<td>Renaming 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</td>
<td>Perfect</td>
<td>?</td>
</tr>
</tbody>
</table>

### Upper Limit to ILP: Ideal Machine

- Integer: 18 - 60
- FP: 75 - 150

- gcc espresso li fpppp doduc tomcatv
More Realistic HW: Branch Impact

Change from Infinite window to 2048, and maximum issue of 64 instructions per clock cycle

Integer: 6 - 12

More Realistic HW: Misprediction Rates

Change to 2048 instr window, 64 instr issue, 8K 2 level Prediction

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>Instruction</td>
<td>2048</td>
<td>Infinite</td>
<td>200</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>Infinite v. 256, 128, 64, 32, none</td>
<td>Infinite</td>
<td>48 integer + 40 FL Pt.</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, 32KID, 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)

Change to 2048 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers

More Realistic HW: Memory Address Alias Impact

Change 2048 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers

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>Instruction</td>
<td>2048</td>
<td>Infinite</td>
<td>200</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>256 int + 256 FP</td>
<td>Infinite</td>
<td>48 integer + 40 FP Pt.</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>8K 2-bit</td>
<td>Perfect</td>
<td>Tournament</td>
</tr>
<tr>
<td>Cache</td>
<td>Perfect</td>
<td>Perfect</td>
<td>64KI, 32KID, 1.92MB L2, 36 MB L3</td>
</tr>
<tr>
<td>Memory Alias</td>
<td>Perfect v. Stack v. Inspect v. none</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

FP: 4 - 45 (Fortran, no heap)
**Simultaneous Multithreading**

- **Key idea**
  - Issue multiple instructions from multiple threads each cycle
- **Features**
  - Fully exploit thread-level parallelism and instruction-level parallelism.
  - Better Performance
    - Mix of independent programs
    - Programs that are parallelizable
    - Single threaded program

**Multithreaded Categories**

- **Superscalar (SS)**
- **Fine-Grained Multithreading (FGMT)**
- **SMT**

**Multiprocessor vs. SMT**

- **Multiprocessor (MP2)**
- **SMT**

**Changes for SMT**

- **Basic pipeline** – unchanged
- **Replicated resources**
  - Program counters, Register maps
- **Shared resources**
  - Register file (size increased)
  - Instruction queue
  - First and second level caches
  - Translation buffers, Branch predictor

**SMT Architecture**

- **Base Processor**: like out-of-order superscalar processor, [MIPS R10000]
- **Changes**: With N simultaneous running threads, need N PC and N subroutine return stacks and more than N*32 physical registers for register renaming in total.
SMT Architecture

- Need large register files, longer register access time, pipeline stages are added. (Register reads and writes each take 2 stages.)

- Share the cache hierarchy and branch prediction hardware.
- Each cycle: select up to 2 threads and each fetch up to 4 instructions. (2.4 scheme)

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Rename</th>
<th>Queue</th>
<th>Reg</th>
<th>Read</th>
<th>Reg</th>
<th>Read</th>
<th>Exec</th>
<th>Reg</th>
<th>Write</th>
<th>Comm</th>
<th>Retire</th>
</tr>
</thead>
</table>

Basic Out-of-order Pipeline

Effectively Using Parallelism on a SMT Processor

Parallel workload

<table>
<thead>
<tr>
<th>threads</th>
<th>Super Scalar</th>
<th>Multi Proc2</th>
<th>Multi Proc4</th>
<th>FGMT</th>
<th>SMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.3</td>
<td>2.4</td>
<td>1.5</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>2</td>
<td>--</td>
<td>4.3</td>
<td>2.6</td>
<td>4.1</td>
<td>4.7</td>
</tr>
<tr>
<td>4</td>
<td>--</td>
<td>--</td>
<td>4.2</td>
<td>4.2</td>
<td>5.6</td>
</tr>
<tr>
<td>8</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>3.5</td>
<td>6.1</td>
</tr>
</tbody>
</table>

Instruction Throughput executing a parallel workload

Effects of Thread Interference in Shared Structures

- Inter-thread Cache Interference
- Increased Memory Requirements
- Interference in Branch Prediction Hardware

Architectural Abstraction

- 1 CPU with 4 Thread Processing Units (TPUs)
- Shared hardware resources
**Inter-thread Cache Interference**

- Because the share the cache, so more threads, lower hit-rate.
- Two reasons why this is not a significant problem
  - The L1 Cache miss can almost be entirely covered by the 4-way set associative L2 cache.
  - Out-of-order execution, write buffering and the use of multiple threads allow SMT to hide the small increases of additional memory latency. 0.1% speed up without interthread cache miss.

**Increased Memory Requirements**

- More threads are used, more memory references per cycle.
- Bank conflicts in L1 cache account for the most part of the memory accesses.
- It is ignorable
  - For longer cache line: gains due to better spatial locality out weighted the costs of L1 bank contention
  - 3.4% speedup if no inter-thread contentions.

**Interference in Branch Prediction Hardware**

- Since all threads share the prediction hardware, it will experience inter-thread interference.
- This effect is negligible since:
  - The speedup outweighed the additional latencies
  - From 1 to 8 threads, branch and jump misprediction rates range from 2.0%-2.8% (branch) 0.0%-0.1% (jump)

**Multiprogrammed workload**

**Decomposed SPEC95 Applications**

**Multithreaded Applications**