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?

• Do we need to invent new HW/SW mechanisms to keep on processor performance curve?
  – Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  – Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  – Motorola AltaVec: 128 bit ints and FPs
  – Supersparc 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

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 & 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 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></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</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 Analysis</td>
<td>Perfect</td>
<td>??</td>
</tr>
</tbody>
</table>

Upper Limit to ILP: Ideal Machine

(Figure 3.1)

<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>150.1</td>
</tr>
<tr>
<td>espresso</td>
<td>62.6</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>17.9</td>
<td></td>
</tr>
<tr>
<td>fppp</td>
<td>75.2</td>
<td></td>
</tr>
<tr>
<td>doduc</td>
<td>118.7</td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>15.1</td>
<td></td>
</tr>
</tbody>
</table>

More Realistic HW: Window Impact

(Figure 3.2)

<table>
<thead>
<tr>
<th>Programs</th>
<th>Integer: 8 - 63</th>
<th>FP: 9 - 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>55</td>
<td>160</td>
</tr>
<tr>
<td>espresso</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>fppp</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>doduc</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>74</td>
<td></td>
</tr>
</tbody>
</table>
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 Issued per clock</td>
<td>64</td>
<td>Infinite</td>
<td>4</td>
</tr>
<tr>
<td>Instruction Window Size</td>
<td>2048</td>
<td>Infinite</td>
<td>200</td>
</tr>
<tr>
<td>Renaming Registers</td>
<td>Infinite</td>
<td>Infinite</td>
<td>48 integer + 40 Fl. Pt.</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Perfect vs. 8K Tournament vs. 512 2-bit vs. profile vs. none</td>
<td>Perfect</td>
<td>2% to 6% misprediction (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>??</td>
</tr>
</tbody>
</table>

More Realistic HW: Branch Impact

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

Program | Integer: 6 - 12
---|---
gcc, espresso, li, fpppp, doduc, tomcatv

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 Issued per clock</td>
<td>64</td>
<td>Infinite</td>
<td>4</td>
</tr>
<tr>
<td>Instruction Window Size</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, 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)

Figure 3.5

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

FP: 11 - 45

Integer: 5 - 15

Program

Infinite 256 128 64 32 None

More Realistic HW: Memory Address Alias Impact

Figure 3.6

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

FP: 4 - 45 (Fortran, no heap)

Integer: 4 - 9

Program

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

FP: 8 - 45

Integer: 6 - 12

IPC

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

Administrivia

- Exam: Wednesday 3/17
  Location: 310 Soda
  TIME: 6:00-9:00
  - This info is on the Lecture page (has been)
  - Get on 8 ½ by 11 sheet of notes (both sides)
  - Meet at LaVal’s afterwards for Pizza and Beverages
- CS252 First Project proposal due by Friday 3/5
  - Need two people/project (although can justify three for right project)
  - Complete Research project in 9 weeks
    » Typically investigate hypothesis by building an artifact and measuring it against a “base case”
    » Generate conference-length paper/give oral presentation
    » Often, can lead to an actual publication.
- Feel free to use the email list to search for a partner
  - cs252@kubi.cs.berkeley.edu
Discussion of papers:
Complexity-effective superscalar processors

  - Several data structures analyzed for complexity WRT issue width
    » Rename: Roughly Linear in IW, steeper slope for smaller feature size
    » Wakeup: Roughly Linear in IW, but quadratic in window size
    » Bypass: Strongly quadratic in IW

  - Overall results:
    » Bypass significant at high window size/issue width
    » Wakeup+Select delay dominates otherwise

- Proposed Complexity-effective design:
  » Replace issue window with FIFOs/steer dependent Insts to same FIFO

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: instruction stream with own PC and data
  » thread may be a 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

Another Approach:
Multithreaded 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 multiples threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads 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

**For most apps: most execution units lie idle**

For an 8-way superscalar.

- memory conflict
- long fp
- short fp
- long integer
- short integer
- load delays
- control hazards
- branch misprediction
- data cache miss
- L1 cache miss
- L2 cache miss
- L3 cache miss
- processor busy


**Do both ILP and TLP?**

- TLP and ILP exploit two 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

Source: Microprocessor Report, December 6, 1999
"Compaq Chooses SMT for Alpha"

Multithreaded Categories

- Superscalar
- Fine-Grained
- Coarse-Grained
- Multiprocessing
- Simultaneous Multithreading

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
- 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
Power 4

Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.

Power 5 data flow ...

Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Power 5 thread performance ...

Relative priority of each thread controllable in hardware.

For balanced operation, both threads run slower than if they "owned" the machine.
Changes in Power 5 to support SMT

- Increased associativity of L1 instruction cache and the instruction address translation buffers
- Added per thread load and store queues
- Increased size of the L2 (1.92 vs. 1.44 MB) and L3 caches
- Added separate instruction prefetch and buffering per thread
- Increased the number of virtual registers from 152 to 240
- Increased the size of several issue queues
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support

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
- Power 5, 8 processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- Power 5 running 2 copies of each app speedup between 0.89 and 1.41
  - Most gained some
  - FL.Pt. apps had most cache conflicts and least gains

Head to Head ILP competition

<table>
<thead>
<tr>
<th>Processor</th>
<th>Micro architecture</th>
<th>Fetch / Issue / Execute</th>
<th>FU</th>
<th>Clock Rate (GHz)</th>
<th>Transistor Die size</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium 4 Extreme</td>
<td>Speculatively dynamically scheduled; 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>Speculatively 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>Speculatively 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 423 mm^2</td>
<td>130 W</td>
</tr>
</tbody>
</table>

Performance on SPECint2000

![Graph showing performance on SPECint2000]
No Silver Bullet for ILP

- No obvious overall 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 FP, 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 \(\Rightarrow\) 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 uniprocessors.
- 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 OOO superscalar microarchitecture
  - Instead of replicating registers, reuse rename registers