Approaching the limits of ILP

- Limitations due to assumption of uniprocessor programming model
- Techniques for ILP are conceptually simple, but design problems are amazingly complex in practice.

- Not much change in processors of last 5 years
  - Pentium 4, IBM Power 5, AMD Opteron have the same basic structure and similar sustained issue rates (2 to 4 instructions per clock) as the first 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

Limits to ILP

- Conflicting studies of amount possible ILP
  - Inherent limits in program code
  - Highly parallel numeric pgms: ILP as high as 1000

- Hardware sophistication vs. 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 AltaVec: 128 bit ints and FPs
  - Supersparc Multimedia ops, etc.

Limits to ILP: based on a study by David Wall (1993)

Wall’s study: 350 HW configurations, 18 benchmarks

Start with ideal HW assumptions:

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)
4. Memory-address alias analysis – perfect
5. Perfect caches; 1 cycle latency for all instructions
6. Unlimited instructionsissued/clock cycle
7. Unlimited functional units

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>48 Integer + 40 Fl. Pt.</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Perfect</td>
</tr>
<tr>
<td>Cache</td>
<td>64K, 32K, 1.5EB L2, 36 MB L3</td>
</tr>
<tr>
<td>Memory Alias Analysis</td>
<td>Perfect</td>
</tr>
</tbody>
</table>

Concepts:

- **issue packet size**: max. number of instructions issued simultaneously in one cycle
- **window size**: max. set of instructions examined for simultaneous execution
- **cycle width**: max. number of instructions executed simultaneously in one cycle
- **memory-address alias analysis**: ability to detect whether memory addresses refer to same data (for elimination of data hazards and code reordering)
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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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>Infinite</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>8K 2-bit</td>
<td>Perfect</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>

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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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>Infinite</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>8K 2-bit</td>
<td>Perfect</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>

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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issued per clock</td>
<td>64 (no restrictions)</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 + 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>

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

<table>
<thead>
<tr>
<th>Change 2048 instr window, 64 instr issue, 8K 2 level Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer: 5 - 15</td>
</tr>
</tbody>
</table>

More Realistic HW: Memory Address Alias Impact

<table>
<thead>
<tr>
<th>Change 2048 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer: 4 - 9</td>
</tr>
</tbody>
</table>

Realistic HW: Window Impact

<table>
<thead>
<tr>
<th>Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer: 6 - 12</td>
</tr>
</tbody>
</table>

NOW Handout Page
Conclusions from David Wall Study (1993)

Under a lot of idealized assumptions:
- Branch prediction most effective
  - Eliminating BP decreases median parallelism from 30.6 to 2.2
- Alias analysis and register renaming very
  - Elimination decreased parallelism to 3.4, 4.8 resp.
- Jump prediction not as critical
  - Elimination decreased parallelism to 21.3
- Branch prediction + speculation very promising
- Wall was cautiously optimistic about uniprocessor ILP.

Today's view of Overcoming ILP 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
- Market forces are an unknown factor

Performance beyond single thread ILP

- There is much higher level natural parallelism in many applications (e.g., Database or Scientific codes)
  - Thread Level Parallelism: 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
- TLP could be more cost-effective to exploit than ILP

New Approach: Multithreaded Execution

- Multithreading: multiple threads to share the functional units of ONE 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

NOW Handout Page 4
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 two different kinds of parallel structure in a program
- Could a processor oriented at ILP also 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

- Superscalar
- Fine-Grained
- Coarse-Grained
- Multiprocessing

For most apps, most execution units lie idle

For an 8-way superscalar.

Do both ILP and TLP?

- TLP and ILP exploit two different kinds of parallel structure in a program
- Could a processor oriented at ILP also 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

- Superscalar
- Fine-Grained
- Coarse-Grained
- Multiprocessing

For most apps, most execution units lie idle

For an 8-way superscalar.
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
- 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

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

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

NOW Handout Page 6
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 (26² 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
  - FLIP apps had most cache conflicts and least gains

Power5 running 2 copies of each app speedup between 0.89 and 1.41
• Most gained some
• FLIP apps had most cache conflicts and least gains

Performance on SPECint2000

Performance on SPECfp2000

Normalized Performance: Efficiency

No Silver Bullet for ILP

• No obvious over all 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 FL 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

NOW Handout Page 7
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 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
  • Itanium/EPIC/VLIW is not a breakthrough in ILP
  • Balance of ILP and TLP decided in marketplace