CS252
Graduate Computer Architecture
Lecture 11

Data Value Prediction
Limits to ILP / Multithreading
February 27th, 2012

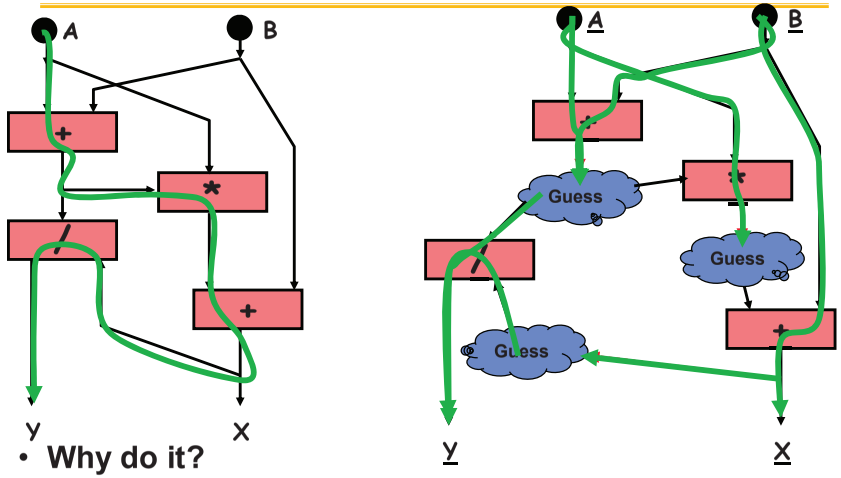
John Kubiatowicz

Electrical Engineering and Computer Sciences

University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

Data Value Prediction



- Why do it?
  - Can "Break the DataFlow Boundary"
  - Before: Critical path = 4 operations (probably worse)
  - After: Critical path = 1 operation (plus verification)

Data Value Predictability

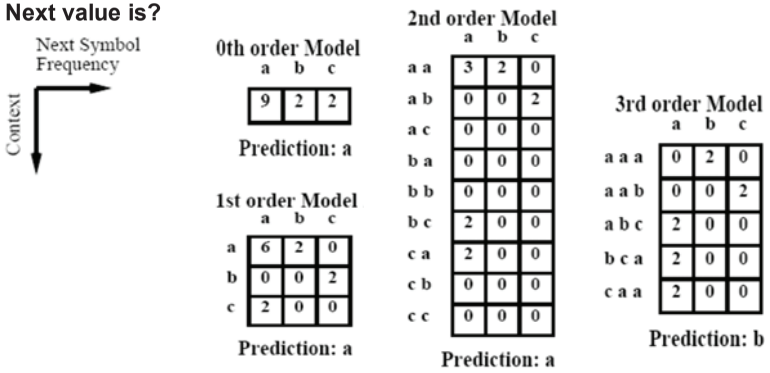
- "The Predictability of Data Values"
  - Yiannakis Sazeides and James Smith, Micro 30, 1997
- Three different types of Patterns:
  - Constant (C): 5 5 5 5 5 5 5 5 5 5 …
  - Stride (S): 1 2 3 4 5 6 7 8 9 …
  - Non-Stride (NS): 28 13 99 107 23 456 …
- Combinations:
  - Repeated Stride (RS): 1 2 3 1 2 3 1 2 3 1 2 3
  - Repeadted Non-Stride (RNS): 1 -13 -99 7 1 -13 -99 7

Computational Predictors

- Last Value Predictors
  - Predict that instruction will produce same value as last time
  - Requires some form of hysteresis. Two subtle alternatives:
    - » Saturating counter incremented/decremented on success/failure replace when the count is below threshold
    - » Keep old value until new value seen frequently enough
  - Second version predicts a constant when appears temporarily constant
- Stride Predictors
  - Predict next value by adding the sum of most recent value to difference of two most recent values:
    - » If v_{n-1} and v_{n-2} are the two most recent values, then predict next value will be: $v_{n-1} + (v_{n-1} - v_{n-2})$
    - » The value $(v_{n-1} - v_{n-2})$ is called the "stride"
  - Important variations in hysteresis:
    - » Change stride only if saturating counter falls below threshold
    - » Or "two-delta" method. Two strides maintained.
      - First (S1) always updated by difference between two most recent values
      - Other (S2) used for computing predictions
      - When S1 seen twice in a row, then S1⇒S2
- More complex predictors:
  - Multiple strides for nested loops
  - Complex computations for complex loops (polynomials, etc!)

Context Based Predictors

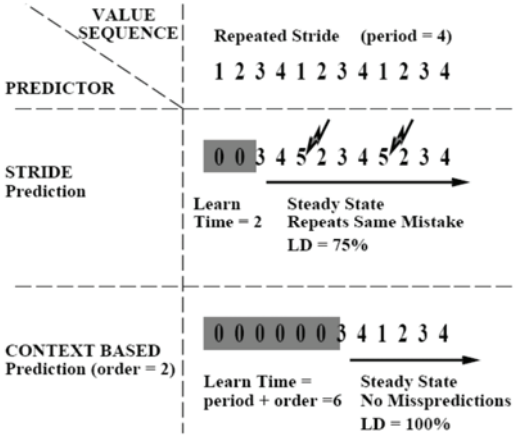
- Context Based Predictor
  - Relies on Tables to do trick
  - Classified according to the order: an "n-th" order model takes last n values and uses this to produce prediction
    - » So – 0th order predictor will be entirely frequency based
- Consider sequence: a a a b c a a a b c a a a
  - Next value is?



- "Blending": Use prediction of highest order available

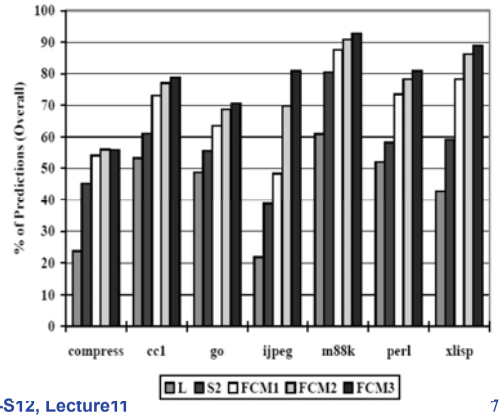
Which is better?

- Stride-based:
  - Learns faster
  - less state
  - Much cheaper in terms of hardware!
  - runs into errors for any pattern that is not an infinite stride
- Context-based:
  - Much longer to train
  - Performs perfectly once trained
  - Much more expensive hardware



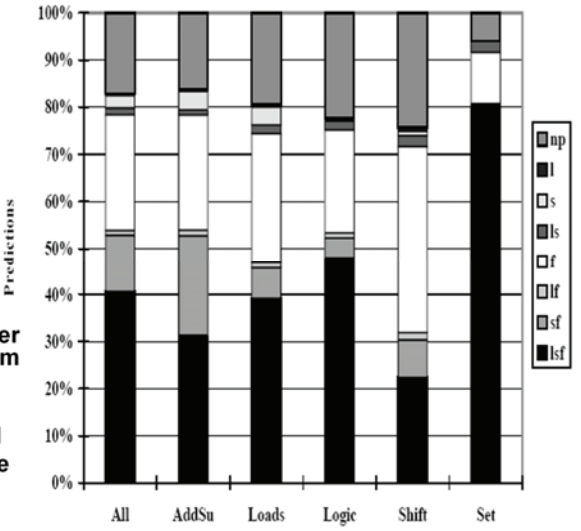
How predictable are data items?

- Assumptions – looking for limits
  - Prediction done with no table aliasing (every instruction has own set of tables/strides/etc.
  - Only instructions that write into registers are measured
    - » Excludes stores, branches, jumps, etc
- Overall Predictability:
  - L = Last Value
  - S = Stride (delta-2)
  - FCMx = Order x context based predictor



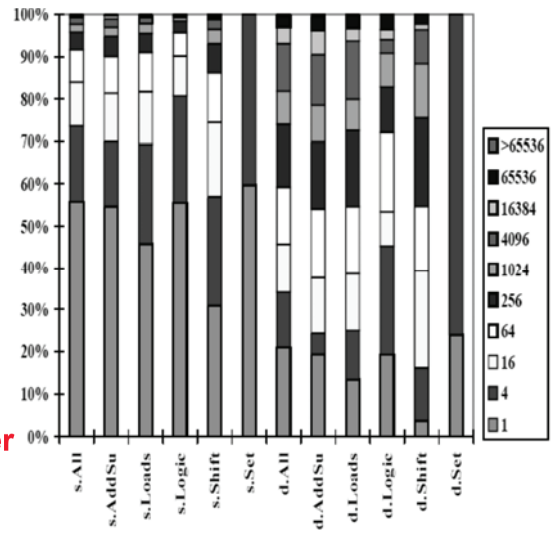
Correlation of Predicted Sets

- Way to interpret:
  - l = last val
  - s = stride
  - f = fcm3
- Combinations:
  - ls = both l and s
  - Etc.
- Conclusion?
  - Only 18% not predicted correctly by any model
  - About 40% captured by all predictors
  - A significant fraction (over 20%) only captured by fcm
  - Stride does well!
    - » Over 60% of correct predictions captured
  - Last-Value seems to have very little added value

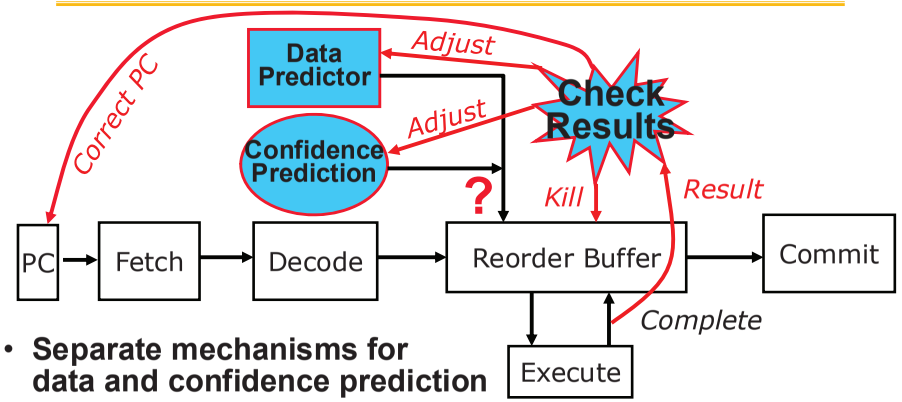


Number of unique values

- Data Observations:
  - Many static instructions (>50%) generate only one value
  - Majority of static instructions (>90%) generate fewer than 64 values
  - Majority of dynamic instructions (>50%) correspond to static insts that generate fewer than 64 values
  - Over 90% of dynamic instructions correspond to static insts that generate fewer than 4096 unique values
- Suggests that a relatively small number of values would be required for actual context prediction



General Idea: Confidence Prediction



- Separate mechanisms for data and confidence prediction
  - Data predictor keeps track of values via multiple mechanisms
  - Confidence predictor tracks history of correctness (good/bad)
- Confidence prediction options:
  - Saturating counter
  - History register (like branch prediction)

Administrivia

- Midterm I: Wednesday 3/21
  Location: 405 Soda Hall
  TIME: 5:00-8:00
  - Can have 1 sheet of 8½x11 *handwritten* notes – both sides
  - No microfiche of the book!
  - Bring DUMB calculator
- Meet at LaVal's afterwards for Pizza and Beverages
  - Great way for me to get to know you better
  - I'll Buy!
- CS252 First Project proposal due by Friday 3/2
  - 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.

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.
  - GPUs?

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 & 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

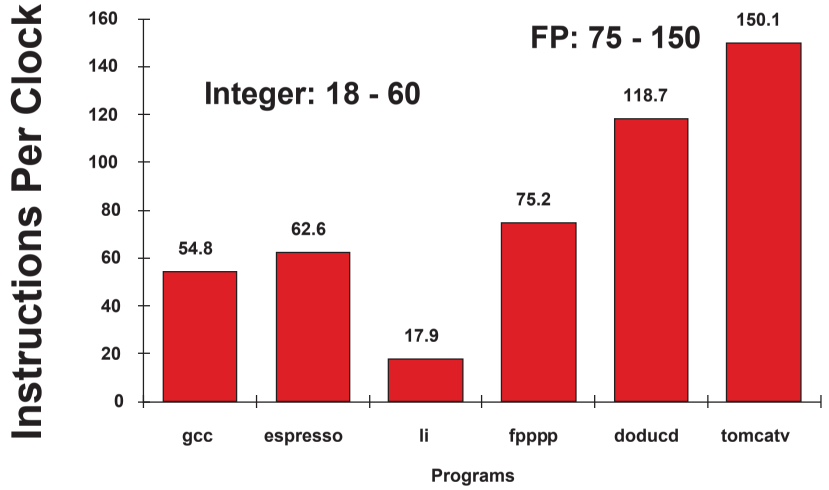
Also: perfect caches; 1 cycle latency for all instructions (FP *,/); unlimited instructions issued/clock cycle;

Limits to ILP HW Model comparison

	Model	Power 5
Instructions Issued per clock	Infinite	4
Instruction Window Size	Infinite	200
Renaming Registers	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias Analysis	Perfect	??

Upper Limit to ILP: Ideal Machine

(Figure 3.1)

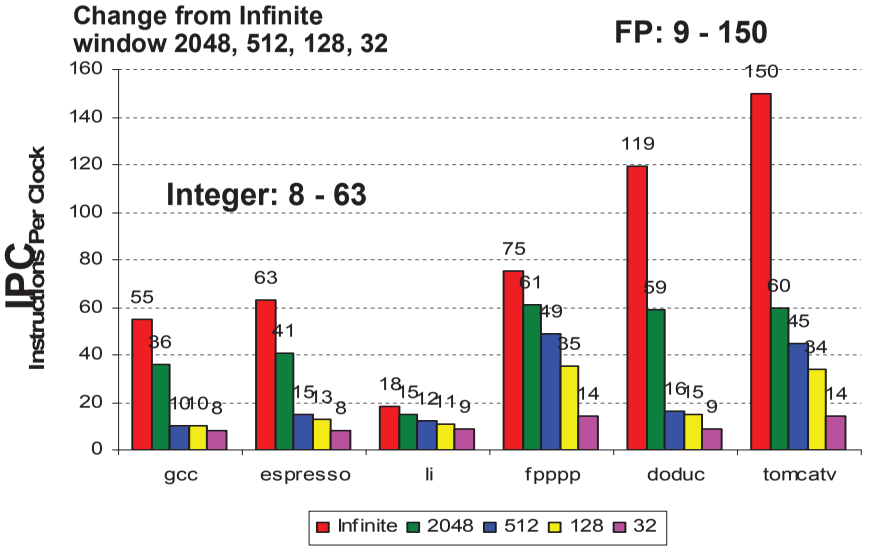


Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	Infinite	Infinite	4
Instruction Window Size	Infinite, 2K, 512, 128, 32	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	??

More Realistic HW: Window Impact

Figure 3.2

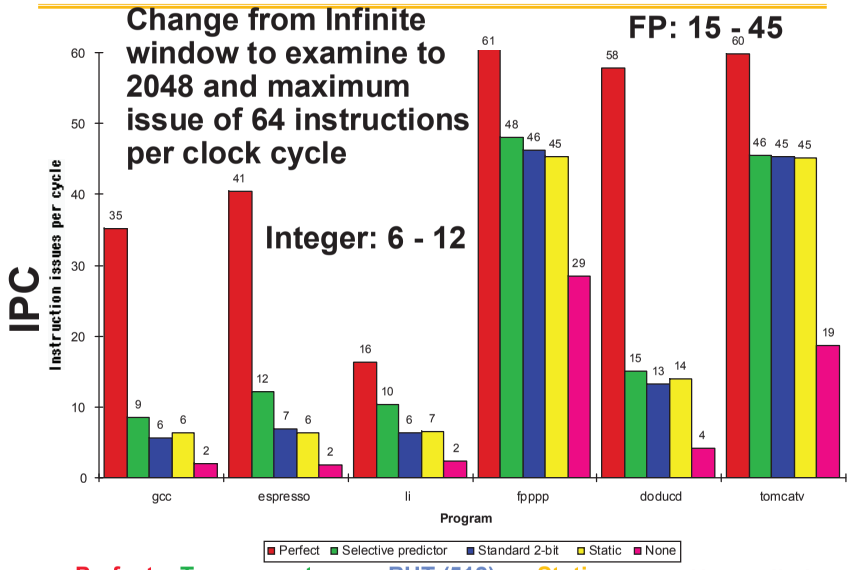


Limits to ILP HW Model comparison

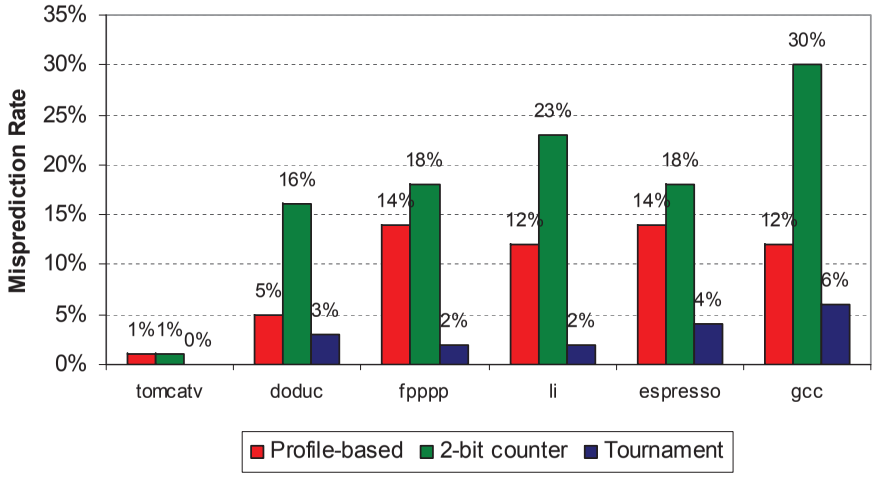
	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect vs. 8K Tournament vs. 512 2-bit vs. static vs. none	Perfect	2% to 6% misprediction (Tournament Branch Predictor)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

More Realistic HW: Branch Impact

Figure 3.3



Misprediction Rates

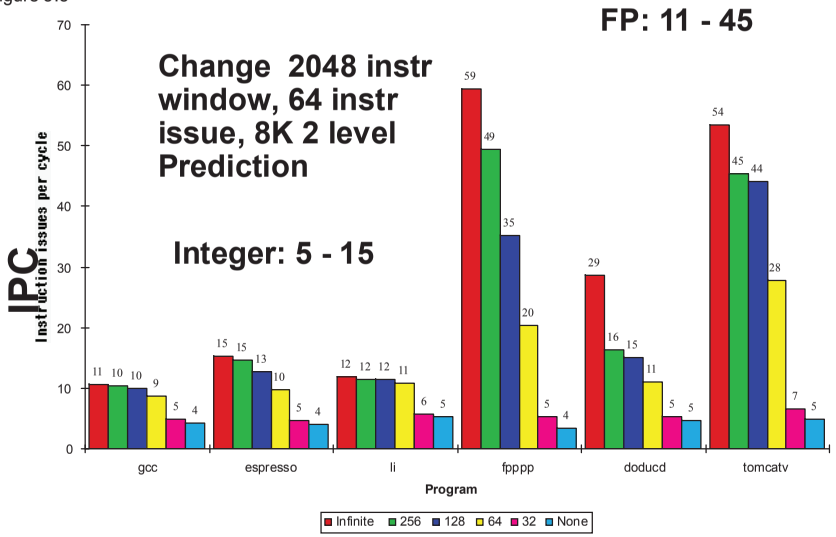


Limits to ILP HW Model comparison

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

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

Figure 3.5

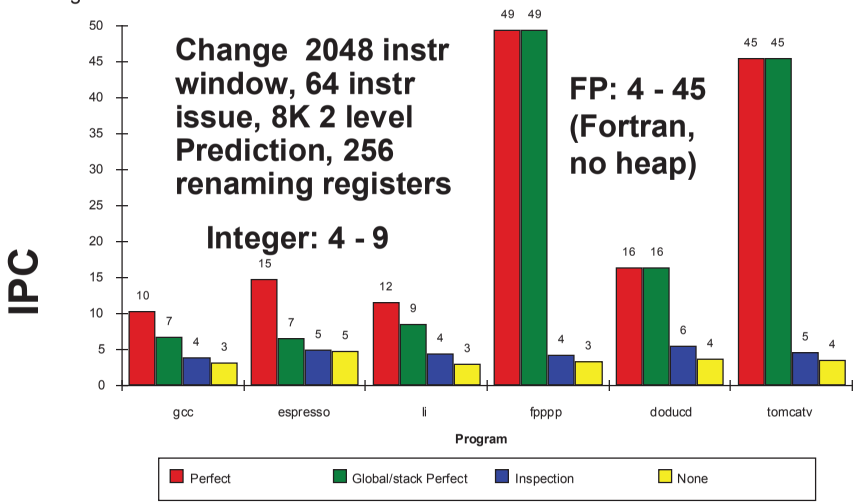


Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	256 Int + 256 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	8K 2-bit	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect v. Stack v. Inspect v. none	Perfect	Perfect

More Realistic HW: Memory Address Alias Impact

Figure 3.6



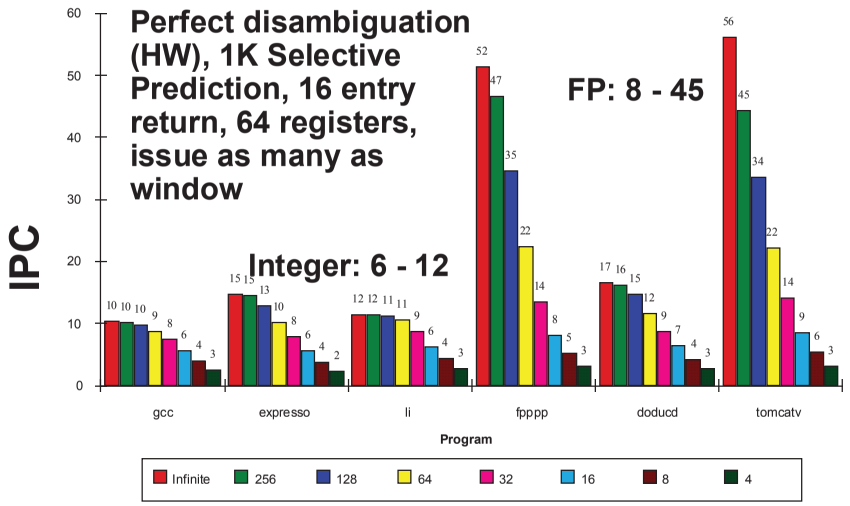
Perfect | Global/Stack perf; heap conflicts | Inspec. Assem. | None

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64 (no restrictions)	Infinite	4
Instruction Window Size	Infinite vs. 256, 128, 64, 32	Infinite	200
Renaming Registers	64 Int + 64 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	1K 2-bit	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	HW disambiguation	Perfect	Perfect

Realistic HW: Window Impact

(Figure 3.7)



Infinite | 256 | 128 | 64 | 32 | 16 | 8 | 4

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

Discussion of papers: Complexity-effective superscalar processors

- "Complexity-effective superscalar processors", Subbarao Palacharla, Norman P. Jouppi and J. E. Smith.
  - 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

Issue width	Window size	Rename delay (ps)	Wakeup+Select delay (ps)	Bypass delay (ps)
0.8μm technology				
4	32	1577.9	2903.7	184.9
8	64	1710.5	3369.4	1056.4
0.35μm technology				
4	32	627.2	1248.4	184.9
8	64	726.6	1484.8	1056.4
0.18μm technology				
4	32	351.0	578.0	184.9
8	64	427.9	724.0	1056.4

Table 2: Overall delay results.

  - 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

Discussion of SPARCLE paper

- Example of close coupling between processor and memory controller (CMMU)
  - All of features mentioned in this paper implemented by combination of processor and memory controller
  - Some functions implemented as special "coprocessor" instructions
  - Others implemented as "Tagged" loads/stores/swaps
- Course Grained Multithreading
  - Using SPARC register windows
  - Automatic synchronous trap on cache miss
  - Fast handling of all other traps/interrupts (great for message interface!)
  - Multithreading half in hardware/half software (hence 14 cycles)
- Fine Grained Synchronization
  - Full-Empty bit/32 bit word (effectively 33 bits)
    - » Groups of 4 words/cache line ⇒ F/E bits put into memory TAG
  - Fast TRAP on bad condition
  - Multiple instructions. Examples:
    - » LDT (load/trap if empty)
    - » LDET (load/set empty/trap if empty)
    - » STF (Store/set full)
    - » STFT (store/set full/trap if full)

Discussion of Papers: Sparcle (Con't)

- Message Interface
  - Closely couple with processor
    - » Interface at speed of first-level cache
  - Atomic message launch:
    - » Describe message (including DMA ops) with simple stio insts
    - » Atomic launch instruction (ipilaunch)
  - Message Reception
    - » Possible interrupt on message receive: use fast context switch
    - » Examine message with simple ldio instructions
    - » Discard in pieces, possibly with DMA
    - » Free message (ipicst, i.e "coherent storeback")
- We will talk about message interface in greater detail

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 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

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)

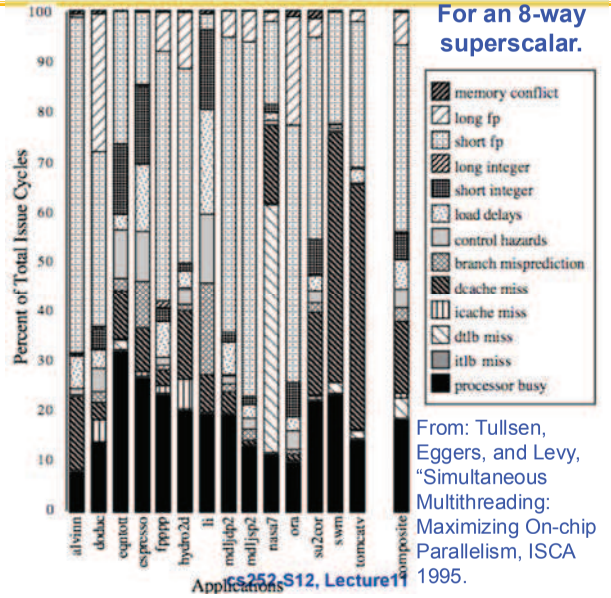
Fine-Grained Multithreading

- 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, Alewife

For most apps: most execution units lie idle

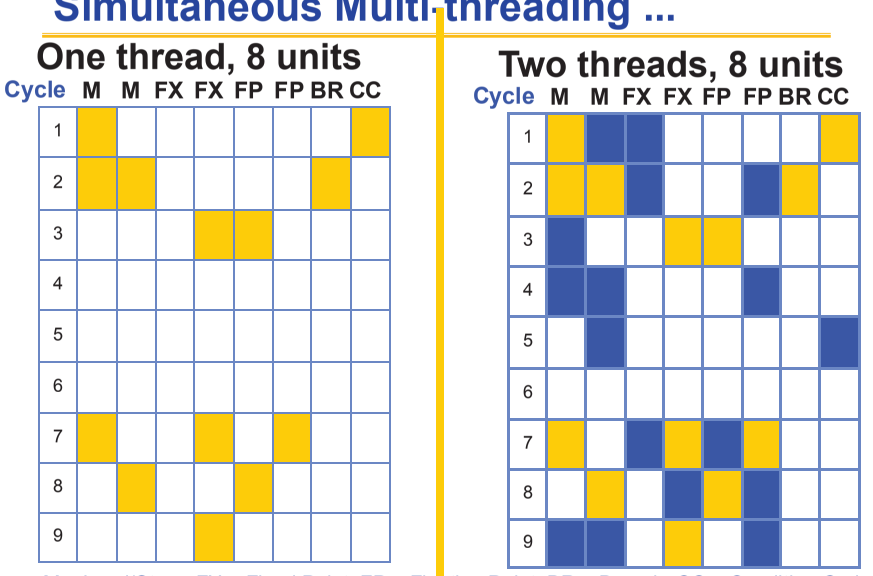


From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

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 Multi-threading ...



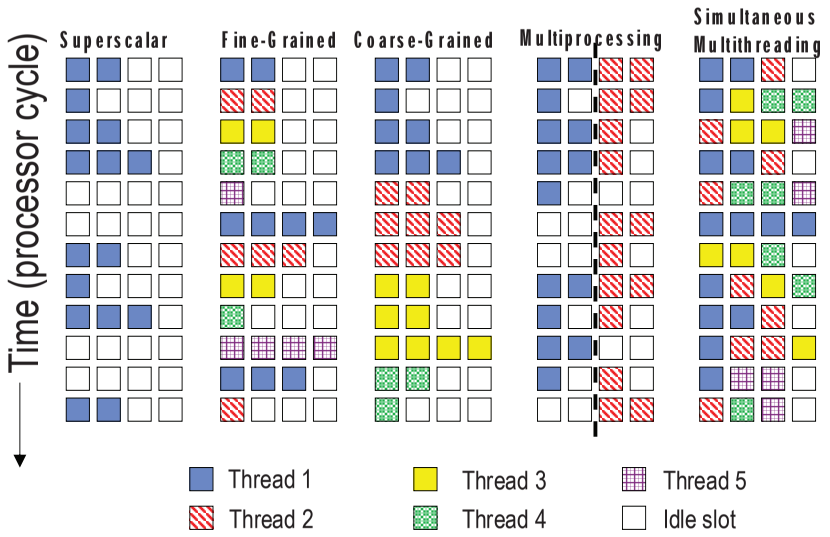
M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition Codes

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

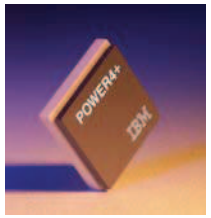


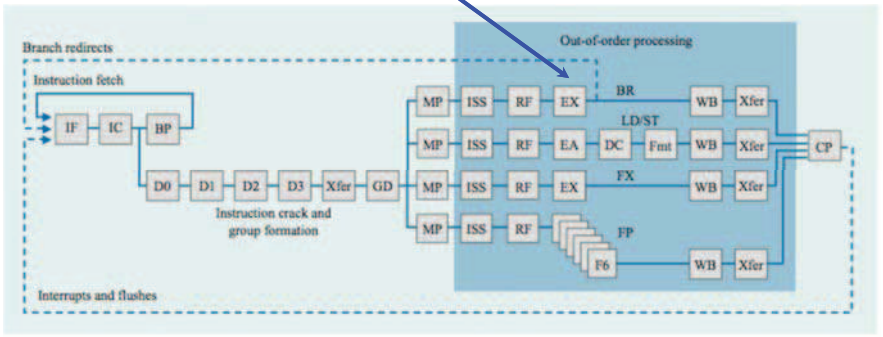
Design Challenges in SMT

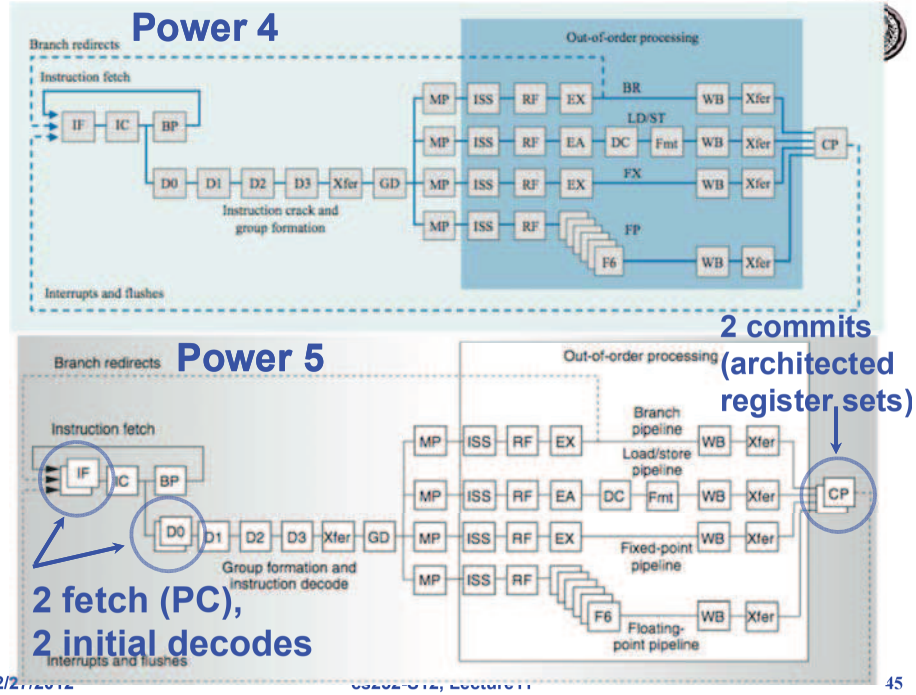
- Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance?
  - A preferred thread approach sacrifices neither throughput nor single-thread performance?
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice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

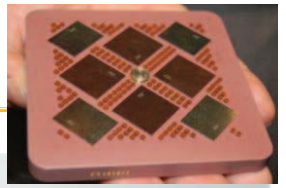
Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.

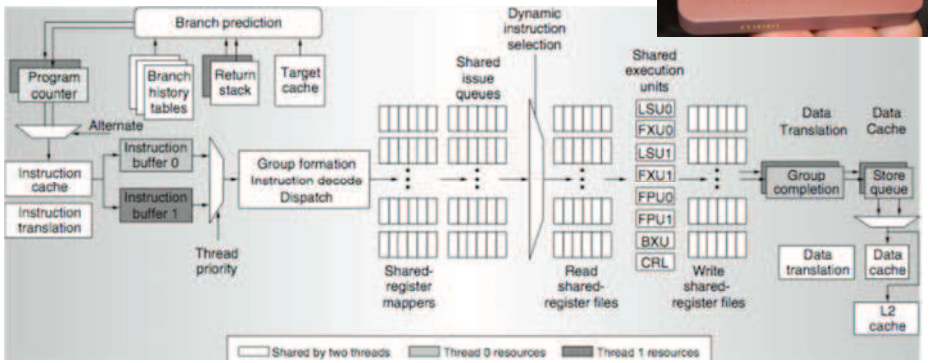






Power 5 data flow ...



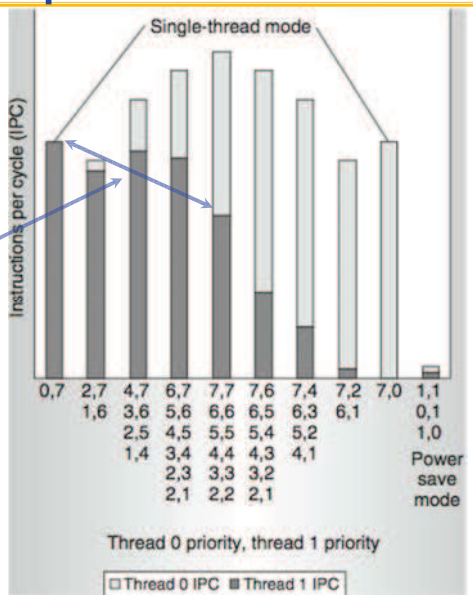


Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

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

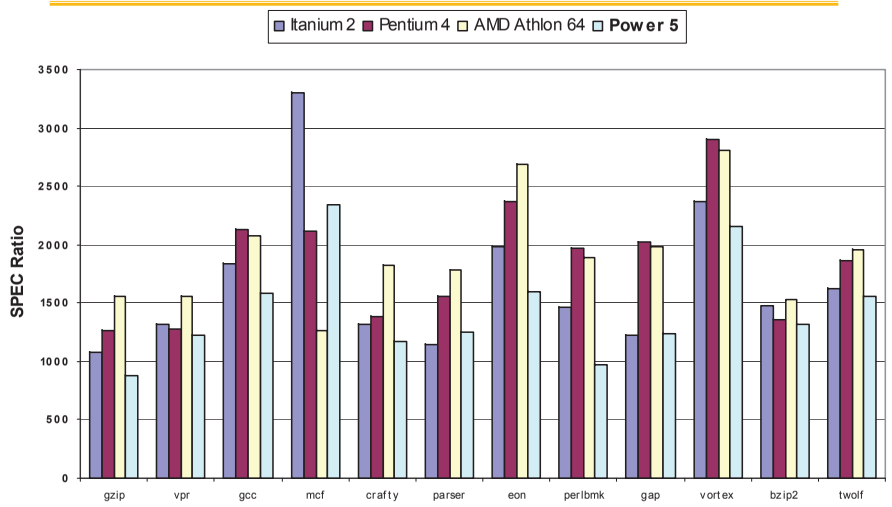
Initial Performance of SMT

- 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

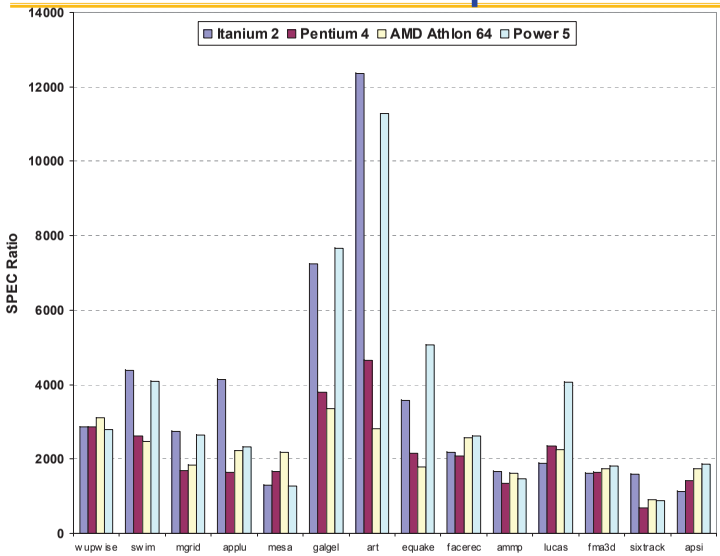
Head to Head ILP competition

Processor	Micro architecture	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis-tors Die size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM Power5 (1 CPU only)	Speculative dynamically scheduled; SMT; 2 CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm^2 (est.)	80W (est.)
Intel Itanium 2	Statically scheduled VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

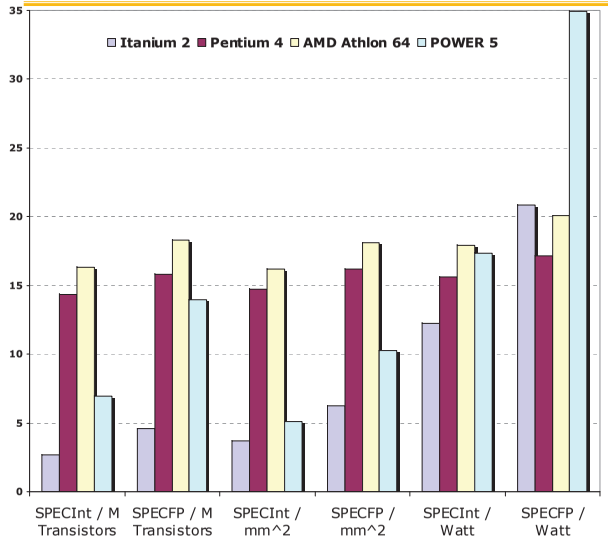
Performance on SPECint2000



Performance on SPECfp2000



Normalized Performance: Efficiency



Rank	Itanium 2	Pentium 4	Athlon	Power 5
Int/Trans	4	2	1	3
FP/Trans	4	2	1	3
Int/area	4	2	1	3
FP/area	4	2	1	3
Int/Watt	4	3	1	2
FP/Watt	2	4	3	1

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

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 multihreading
  - 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