# CMSC 411
# Computer Systems Architecture
# Lecture 13
# Instruction Level Parallelism 7
# (Limits to ILP & Threading)

## Limits to ILP

- Conflicting studies of amount of ILP
  - 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 FP per clock
  - Motorola AltiVec: 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



## 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;

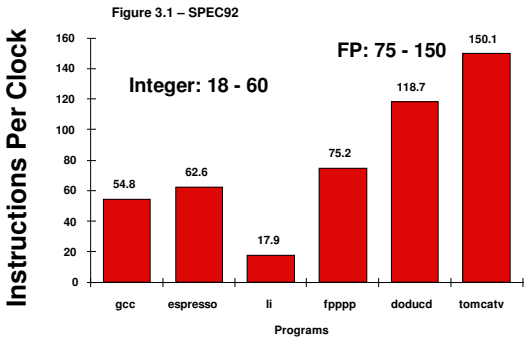


## Limits to ILP HW Model comparison

| | 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

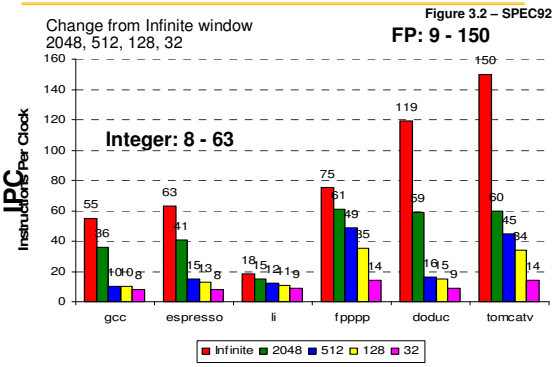


## Limits to ILP HW Model comparison

| | 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 – SPEC92

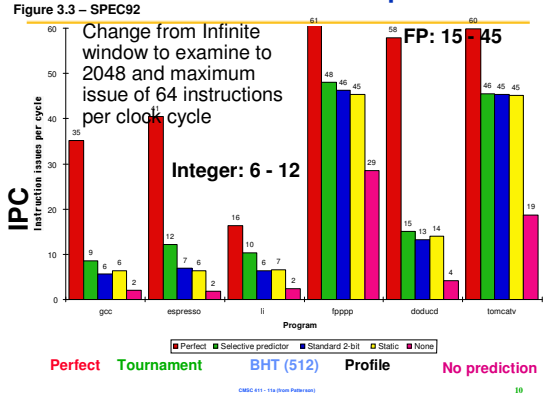



## Limits to ILP HW Model comparison

| | 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. profile 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 | ?? |

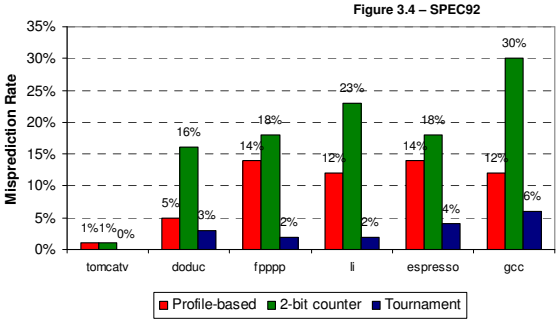
## More Realistic HW: Branch Impact

Figure 3.3 – SPEC92




## Misprediction Rates

Figure 3.4 – SPEC92

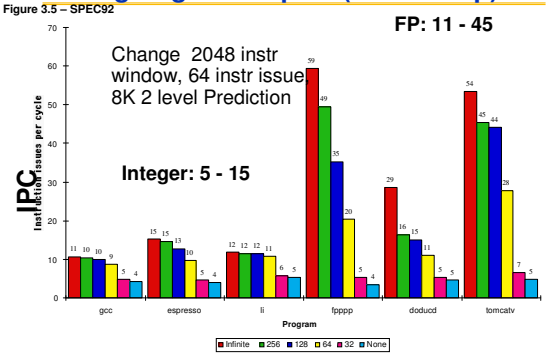



## Limits to ILP HW Model comparison

| | 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 – SPEC92




Infinite 256 128 64 32 None



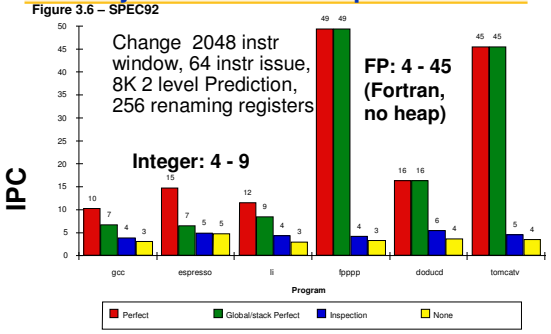
## Limits to ILP HW Model comparison

| | 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 – SPEC92




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



## 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



## 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: 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
- 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



## New 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 to 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)



## Coarse-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses
- Used in IBM AS/400
- 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

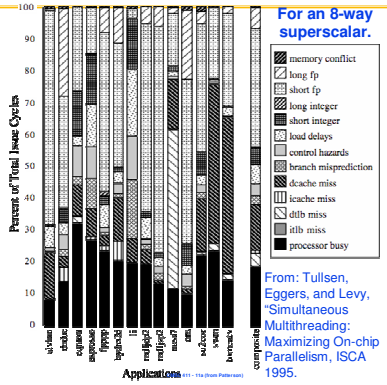


## Coarse-Grained Multithreading

- 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 start-up overhead, coarse-grained multithreading better at reducing penalty of high cost stalls, where pipeline refill << stall time



## For most apps, most execution units lie idle



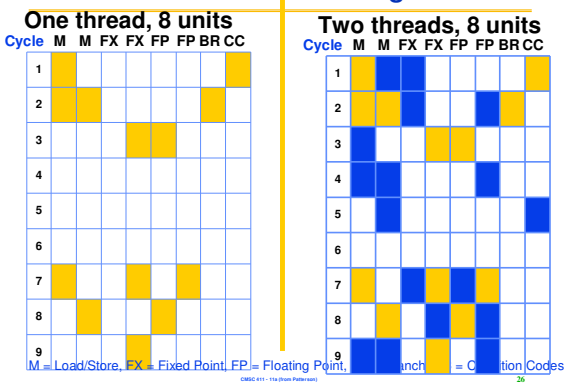

From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

## Do both ILP and TLP?

- TLP and ILP exploit two different kinds of parallel structure in a program
- Could a processor oriented at ILP 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 ... Figure 3.8 (almost)






## 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 add a per thread renaming table and keep separate PCs
  - Independent commitment can be supported by logically keeping a separate reorder buffer for each thread



## 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



## Design Challenges in SMT (con't)

- 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