Lecture 10: ILP Innovations

- Today: ILP innovations and SMT (Section 3.5)
Improving Performance

• Techniques to increase performance:
  ➢ pipelining
    ▪ improves clock speed
    ▪ increases number of in-flight instructions
  ➢ hazard/stall elimination
    ▪ branch prediction
    ▪ register renaming
    ▪ efficient caching
    ▪ out-of-order execution with large windows
    ▪ memory disambiguation
    ▪ bypassing
  ➢ increased pipeline bandwidth
Deep Pipelining

• Increases the number of in-flight instructions

• Decreases the gap between successive independent instructions

• Increases the gap between dependent instructions

• Depending on the ILP in a program, there is an optimal pipeline depth

• Tough to pipeline some structures; increases the cost of bypassing
Increasing Width

• Difficult to find more than four independent instructions

• Difficult to fetch more than six instructions (else, must predict multiple branches)

• Increases the number of ports per structure
Reducing Stalls in Fetch

• Better branch prediction
• Trace cache
Reducing Stalls in Rename/Regfile

- Larger ROB/register file/issue queue
- Virtual physical registers
- Runahead
- Two-level register files
Stalls in Issue Queue

- Two-level issue queues
- Value prediction
- Memory dependence prediction
- Load hit prediction
Functional Units

• Pipelining

• Clustering
Thread-Level Parallelism

• Motivation:
  ➢ a single thread leaves a processor under-utilized for most of the time
  ➢ by doubling processor area, single thread performance barely improves

• Strategies for thread-level parallelism:
  ➢ multiple threads share the same large processor → reduces under-utilization, efficient resource allocation
    Simultaneous Multi-Threading (SMT)
  ➢ each thread executes on its own mini processor → simple design, low interference between threads
    Chip Multi-Processing (CMP)
Multithreading Within a Processor

- Until now, we have executed multiple threads of an application on different processors – can multiple threads execute concurrently on the same processor?

- Why is this desirable?
  - inexpensive – one CPU, no external interconnects
  - no remote or coherence misses (more capacity misses)

- Why does this make sense?
  - most processors can’t find enough work – peak IPC is 6, average IPC is 1.5!
  - threads can share resources → we can increase threads without a corresponding linear increase in area
How are Resources Shared?

Each box represents an issue slot for a functional unit. Peak thruput is 4 IPC.

- Superscalar processor has high under-utilization – not enough work every cycle, especially when there is a cache miss
- Fine-grained multithreading can only issue instructions from a single thread in a cycle – can not find max work every cycle, but cache misses can be tolerated
- Simultaneous multithreading can issue instructions from any thread every cycle – has the highest probability of finding work for every issue slot
What Resources are Shared?

• Multiple threads are simultaneously active (in other words, a new thread can start without a context switch)

• For correctness, each thread needs its own PC, its own logical regs (and its own mapping from logical to phys regs)

• For performance, each thread could have its own ROB (so that a stall in one thread does not stall commit in other threads), I-cache, branch predictor, D-cache, etc. (for low interference), although note that more sharing → better utilization of resources

• Each additional thread costs a PC, rename table, and ROB – cheap!
Resource Sharing

Thread-1
- R1 ← R1 + R2
- R3 ← R1 + R4
- R5 ← R1 + R3

Instr Fetch

Instr Rename
- P73 ← P1 + P2
- P74 ← P73 + P4
- P75 ← P73 + P74

Thread-2
- R2 ← R1 + R2
- R5 ← R1 + R2
- R3 ← R5 + R3

Instr Fetch

Instr Rename
- P76 ← P33 + P34
- P77 ← P33 + P76
- P78 ← P77 + P35

Issue Queue
- P73 ← P1 + P2
- P74 ← P73 + P4
- P75 ← P73 + P74
- P76 ← P33 + P34
- P77 ← P33 + P76
- P78 ← P77 + P35

Register File

FU

FU

FU

FU
Performance Implications of SMT

- Single thread performance is likely to go down (caches, branch predictors, registers, etc. are shared) – this effect can be mitigated by trying to prioritize one thread

- While fetching instructions, thread priority can dramatically influence total throughput – a widely accepted heuristic (ICOUNT): fetch such that each thread has an equal share of processor resources

- With eight threads in a processor with many resources, SMT yields throughput improvements of roughly 2-4

- Alpha 21464 and Intel Pentium 4 are examples of SMT
Title

• Bullet