CS 654
Computer Architecture Summary
Peter Kemper
Chapters in Hennessy & Patterson

- Ch 1: Fundamentals
- Ch 2: Instruction Level Parallelism
- Ch 3: Limits on ILP
- Ch 4: Multiprocessors & TLP
- Ap A: Pipelining
- Ap C: Memory Hierarchy
- Ap F: Vector Processors
C1: Fundamentals

• Computer Architecture:
  – Topic:
    • Designing the organization and hardware to meet goals and functional requirements and to succeed with changing technology
    • Not just ISA
  – Technology trends: Bandwidth over latency, scaling of transistors and wires, power in ICs, cost, dependability
  – Measuring, Reporting, Summarizing Performance
  – Quantitative Principles
    • Take advantage of parallelism
    • Principle of locality
    • Focus on common case
    • Amdahl’s law
    • Processor performance equation
C1: Fundamentals

- **Formulas:**
  - CPU time, Amdahl’s law, Power dynamic & static, Average memory access time, Availability, Die yield, Misses per instruction, Cache index size, Means (arithmetic, geometric -> Benchmarks)

- **Rules of Thumb:**
  - Amdahl/Case Rule, 90/10 locality rule, bandwidth rule, 2:1 Cache rule, dependability rule

Check short list inside book cover!
Ap A: Pipelining

• Key idea:
  – Split up work in a sequence of steps, work along stages in a piecemeal manner, start next instruction as soon as previous one proceeded far enough

• RISC, load/store architecture

• Challenges:
  – Hazards: Data (RAW, WAW, WAR), Control, Structural

• Focus:
  – CPI: get average value as small as possible
    • Close to 1
    • Less than 1

• Means to reduce pipeline stalls?
Ap A: Pipelining

Means to reduce pipeline stalls?

• Fetching:
  – Prefetching, Branch prediction, Caches (TLB, BTB)
• Decoding
  – Decode (Trace cache), Issuing (Multi-issue)
• Execution
  – Forwarding
  – Trouble: multicycle instructions (FP)
• Memory
  – Forwarding (trouble: data dep for load & successor)
• Write-back
  – Write first half of cycle (to have reads in 2nd half)

Scheduling: Static vs dynamic
Ap C: Memory

Cache organization
- direct mapped, fully associative, n-way set assoc,
- write through vs write back, write alloc vs no-write alloc
- layered, dimensions, speed, inclusion property,
- size of cache lines, tags, control bits/flags
- Misses: 4 C’s

- Address transformation
  - Virtual memory -> Physical address

- Access in parallel with TLB
  - Virtually indexed, physically tagged

- Average memory access time =
  Hit time + Miss rate * Miss penalty
Formula extends to multiple layers
Does out of order execution help?
Ap C: Memory

6 Basic Cache Optimizations in 3 categories

- Reducing the miss rate:
  larger block size, larger cache size, higher associativity

- Reducing the miss penalty:
  multilevel caches, reads get priority over writes

- Reducing time to hit the cache:
  avoid index translation when indexing the cache

Misses: compulsory, capacity, conflict, coherence
C 2: ILP

Challenge:
- Reorganize execution of instructions to utilize all units as much as possible to speed up calculations

Obstacles:
Hazards: Control, Functional, Data (RAW,WAW,WAR)

Options:
- Compiler techniques: loop unrolling
- Branch prediction, static, dynamic, branch history table, 2-bit prediction scheme, local vs global/correlating predictor, tournament predictor
- Dynamic scheduling, hardware based speculation
  - Tomasulo: reservation station, common data bus, register renaming, issue in order, exec ooo, complete ooo, precise exceptions?
  - Tomasulo + speculation: ROB, commit in order
  - Register renaming
- Multiple Issue
  - Statically/dynamically scheduled super scalar processor, VLIW processors
- Instruction delivery and speculation, BTB
C 3: ILP limits

Simulation study to evaluate design space:
• Register renaming
• Branch prediction, jump prediction
• Memory address alias analysis
• Perfect caches

Spec benchmarks: limited ILP potential
More realistic assumptions reduce potential even further
• Limited window size, maximum issue count
• Realistic branch and jump prediction
• ..

Also: uniform & extremely fast memory access
C 3: ILP limits

Superscalar processors & TLP:

• Coarse-grained, fine-grained and simultaneous multithreading

• Challenges:
  – Larger register file
  – Not affecting clock cycle (issue & commit stages)
  – Cache & TLP conflicts do not degrade performance

• Moderate level of TLP can be supported with little extra HW effort
  – Example Power4 -> Power5 with SMT

• Future trends:
  – superscalar processors to expensive to push further
  – Also wrt power consumption -> Multiproc, multicore
C 4: Multiprocessors & TLP

- Flynn’s taxonomy
- Centralized shared-memory vs distributed memory multiprocessor design
- Cache coherence
  - Snooping protocol vs directory-based protocol
  - 3 state finite state machine / automaton
  - Per cache line, (also memory for directory)
  - Reacts on CPU read/write requests
  - Reacts on bus read miss, write miss, invalidate requests
  - Cache can contain no data, right data, wrong data and be in state invalid, shared, exclusive
  - Coherence traffic increases with #processors, does not decrease with larger size of cache
C 4: Multiprocessors & TLP

- Synchronization
  - Primitives: exchange, test&set, fetch&increment
    Implemented with
  - Pair of instructions: load linked, store conditional
  Implementing locks with primitives
  - Spin locks
    Used to protect access to monitor/lock that synchronizes threads and keeps queue of waiting threads e.g. in Java
Ap F: Vector processors

• ISA includes vector operations & vector registers
  (Also in ordinary processors: SSE and Altivec for short vectors)
• Code:
  – Concise: single instructions carries a lot of work to do
  – No dependencies inside vector operation
  – Stripmining
• Memory access
  – Regular (possible with constant strides) for load & store
• Functional units
  – Accesses same units, allows for lanes to parallelize
• Execution:
  – Vector chaining
  – Gather/scatter with indirect memory access
  – Conditional execution
  – Compress/expand operations