Comparative Architectures

Source

- IB Architecture / Computer Design
- IB Compiler Construction
- II Advanced Computer Architecture / Comparative Architectures
- Computer architecture: a quantitative approach
  - Hennessy, J.L. & Patterson, D.A (2011)

Analogue and digital

- What are the advantages and disadvantages of analogue computers over their digital counterparts?

	analogue (oscilloscope)	**digital computer**
feature	continuous values / physical data	discrete values / binary system
speed	slow	fast
memory capacity	low or limited	large
reliable / accurate	no (checksum)	yes
usage, arch	complicated	easy
result	voltage signals	computer screen
energy	current -- power-hungry	lower power
reprogram	wirable	reconfigurable
communication	radio signal (speed)	bus, wire (1/10 speed of light)

Relationship: Analog = Quantize [0,255] saturated by Max $\Longrightarrow$ Digital

Digital computer system comes from analog and the conversion has a cost.

Digital one has repeatable complex components, Inductor.

Modern Compiler

Key takeaway from translator (interpreter) shown in Compiler Lecture,

- Divide from single into two stages
  - Compile (inspect)

  - Interpret (compute)
- Divide from single into two stacks (memories)
  - instruction stack / IM

```
PUSH, POP, MK_PAIR
```

  - data value stack / DM
- Separation of the two memories (Instruction and Data)
  - allows for simultaneous access
    - an instruction can be read while a data memory is read or written in the same cycle.
    - Motivation for pipeline and multi-issue *superscalar* (Instruction level parallelism)
    - more difficult with a unified cache/memory.
  - instruction memory is read-only and has less circuitry
    - has no dirty bits, no write back, etc
    - the IM and DM can have different associativity
  - Downside: von Neumann bottleneck
    - common bus (address, data, control)

Aside: Turing tax (universal computing machine) vs special purpose processor.

Flynn's Taxonomy

Based on parallelism on instruction and data streams, the first four kinds listed below

SISD

- A simple processor

MISD

- Used for redundancy
  - Fligh control system, error-detection

SIMD

- Vector processing
  - Vector registers each hold several data items
    - hardware: Regs = Reg $\times n$
  - Vector operations (add, multiple)
    - hardware: ALU $\times n$
- Energy-efficient, data level parallelism

MIMD

- Multicore, standard general purpose CPUs

Extra: SIMT

- each thread has *separate state* (registers and memory)
    - e.g. stack pointer (sp)

- data level parallelism

Processor					Note
instruction	---S---	Fetch	Decode	---S---	*Shared*
memory	---S---	Shared	Memory	---S---	*Shared*
processing	ALU0	… …	… …	ALU31	*single 32-value* *vector operation*
thread states	Regs 0 … …	Regs 1 … …	Regs 2 … …	Regs 3 … …	*each vector register* *contain 32 floats*
	… … Regs 12	… … Regs 13	… … Regs 14	… … Regs 15	*hide latency* *when stall*
thread context	T0	T1	T2	T3	only**one** run *save context switch*

Architectures comparison

Source: Classifying Instruction Set Architectures (Textbook **A.2**)

Architecture	Accumulator	Stack	Register File
operands: from memory and	acc + 3 = 4	top of the stack	rs1, rs2 (disjoint), rd orthogonal needs less
instruction density	shortest less mem space	concise (short instr)	longer
von Neumann bottleneck (Mem bus)	worse for mem (RTT) mem bus 2x CPU $\Longleftrightarrow$ 2x frequency	store imm in stack (near) If stack is full, memory	store in cache (nearer) fast mem access $\Longleftrightarrow$ higher frequency
caching	hard to predict	predictable	in the middle
power consumption	less few memory accesses	less for control few memory accesses	most multi-issue
multi-issue	0	0	Yes
performance	Calculator ENIAC	razer printer, compiler(JVM) Hard for queue, list, swap	modern CPU IC best

	superscalar	compiler VLIW	SIMD	multi-core	DSA
parallelism	static or dynamic ILP, MLP	static ILP, MLP	DLP	TLP	custom
features	instruction fetch, dynamic prefetch, memory addr. alias, physical regs. rename	scheduling; sw. speculate; var. /fixed bundle	N ops., independent, same FU, disjoint regs., known mem access,	fine/coarse-grained vs. SMT	specialized
instr. count (IF/DE)	↑ out-of-order	one VLI	↓	var.	custom
branch handling	dynamic branch pred.	limited	poor (predicted)	per-core	custom
limitations	fabrication, and below	tailor to a pipeline	data-level tasks	Amdahl's Law	inflexible
hardware cost/area	↑	↓	vector regs. / FUs	pipeline regs.	custom
interconnect	↑ (in single core)	↓	wide data bus, lane	mesh/cache coherence	scratchpad
energy cost	↑	↓	↓	var.	↓ ✰
binary compatibility	✓	×	☐ (✓ VLA)	✓	×
use cases	CPU, general-purpose	embedded, GPU	ML, graphics	CPU, server, SoC	TPU, DSP

Trends

Design goals and constraints

- target markets
  - cost, size, time-to-market goals, power consumption, performance and
  - the types of programs (parallelism forms, dataset and program characteristics, e.g. cache).
  - predictable execution times, fault tolerance, security, compatibility requirements.
  - more: fabrication and packaging technology (mask costs), i.e. transistors size and speed, power consumption limits, interconnect speed , number of metal layers, I/Os speed and number (e.g. limits of off-chip/DRAM bandwidth); PCB design considerations, the size and cost of the overall product etc.

Early computers exploit *Bit*-Level Parallelism.

- multi-processing important as memory falls behind
- OS and compiler support for parallelism, limitation

Limitations of Moore's law

- power wall, Amdahl's law (sequential dominates), parallel programming challenges (TLP, correctness), thread scheduling.
- Off-chip memory bandwidth, package pins, shared-memory communication models (bus vs. directory).
- On-chip interconnects (e.g. mesh, ring, crossbar) and their limitations.
- Process variations, temperature variations, aging and reliability.

Die stacking

Fundamentals of Computer Design

Common case first

Energy, power vs. parallelism, performance

- parallelism
- specialization, superscalar vs. multi-core, vector

Instruction Set Architecture (ISA)

- pipelined processor
- both 16-bit and 32-bit ISA
- RISC vs. CISC; number of registers.

Predicated operations

- relation with out-of-order execution
- complicates out-of-order execution
- See more in VLIW section.

Scalar pipeline

Multiple / Diversified pipelines

- allow different instructions without data dependencies to execute in parallel, to hide the latency of the long-cycle instructions (e.g. load/store).
- exploits ILP and improves IPC.

Deep pipeline and optimal pipeline depth

- pros: allow processors to be clocked at higher frequencies, increasing the number of instructions that could be executed in any period of time.
- cons: extra logic (area) required to support deeper pipelines and extract ILP consume large amounts of power.
  - minimise the critical path by balancing workload among the pipeline stages.
  - CPI increases because stall penalties or pipeline interruptions, i.e. operations can't be done in a single cycle.
    - branch predictors to keep the pipeline full (control hazards).
    - memory access (e.g. L1 cache), which negates the benefits achieved.
  - pipeline registers overhead, clock skew.
  - limited ILP, atomic operations.
  - branch predictor limitations for deep pipelines.

Branch prediction

benefits

how to avoid pipeline bubble for complex predictors?

- have a valid fetch address generated every cycle.
  - complex + less accurate predictors together, refetch if differs.
  - add next line and way info to each four-instruction fetch block within the instruction cache.

Static

- compiler support for branch prediction, i.e. static heuristic and profile info.
- e.g. forward-not-taken and backwards-taken.

Dynamic

One-level predictor: saturating counter

Two-level predictor: local/global history

- failure cases

Tournament predictor

Branch Target Buffer (BTB)

- procedure return address stack (PRAS), indirect jump.

Precise exceptions

- all instructions before E (the instruction that caused the exception) must be completed, and
- all instructions after E must not have completed, including not modifying the architectural state.
- whether E should complete or not depends on the exception type.
    - buffer pre-executed results allowing the effects of the second partially executed instruction to be undone,
        - or to buffer results until we know all earlier operations have completed;
    - another option, record which parts of the second instruction have executed and selective replay,
        - now upon restarting execution after the exception handler runs,
        - the processor knows which operations (“beats” in Arm terminology) not to re-execute in the case of the second instruction.
        - supported by Arm’s MVE/Helium ISA extension.

Improvements for scalar pipeline

- micro-architectural techniques or elements:
    - I/D-caches,
    - branch prediction,
    - multiple / diversified pipelines,
    - skid buffer for stalling and replaying.

Superscalar pipeline

Exploits *Instruction*- and *Memory*-Level Parallelism.

- Instruction Fetch, DEcode, [Rename], [Dispatch], [Issue: static/dynamic scheduling], EXecute, Memory, Write Back.
- vs. In-order
    - dispatch: stall for data hazards (false dependencies);
    - issue: stall for structural hazards.

OOO processors benefit from,

- after instructions fetched, decoded and renamed, dispatch into the issue queue, which can be scheduled out-of-order.
- issue: create a window into the dynamic instruction stream, from different basic blocks of the original program.
- pros: schedule instructions dynamically aided by speculation,
    - constrained by little more than true data dependencies and functional unit availability.
    - different instruction schedules depending on run-time info. and the actual state of the processor,
        - branch prediction, react to data cache misses,
        - exploit knowledge of load/store addresses, i.e. disambiguate memory addresses.
    - avoids the need to stall when the result of a cache miss is needed.
        - improved memory-level parallelism, tolerate longer cache access latencies.

    - simplify the compiler (register allocator), without increasing static code size.
    - allows code compiled for another pipeline to run efficiency.
- cons: more hardware cost,
    - complexity of instruction fetch, memory speculation, and register renaming, cache access.
    - large area, high power consumption, heat dissipation, and fabrication complexity.
- limitation: task with low performance targets (ILP, Amdahl's law), competing for FUs.
    - exploiting only ILP, without TLP or DLP.
    - interconnect, limiting on state reachable per cycle and centralised memory-like structures scale.

Instruction fetch

- instruction cache and misalignment (spanning multiple cache lines), branch prediction.
- multiple basic blocks (path prediction and branch address cache, trace cache)

Register rename

- arch (or logical) register $\rightarrow$ **physical** of the last destination targeted.
    - Register Map Table (RMT), Free **Physical** Register List (FPRL);

- to remove false (or name) dependencies (anti-:WaR, output:WaW).
    - VLIW: Rotating Register File (RRF).
- increase the maximum number of instructions that could be in-flight simultaneously.
- vs. compile time,
    - latter is difficult due to the presence of short loops, control dependencies, or
    - when the ISA only defines a very limited number of architectural registers.
- support speculative execution and precise exceptions.
    - speculative execution benefits from the presence of a large number of physical registers to hold live variables from speculative execution paths.
    - quickly return the processor to a particular state, either to implement precise interrupts or to recover from a mis-predicted branch.

Clustered data forwarding network

- partitioned issue windows;
- issue: inter-cluster communication.
- for VLIW, the same applies.

Hardware-based dynamic memory speculation

- memory-carried dependencies (aliases)

    - not to issue a load instruction before a pending store that is writing to the same memory location has executed.
    - or store-to-load forwarding; otherwise, the load can bypass the store, via load queue.

- irreversible store

    - store instructions are executed in program order, via store queue,
    - never speculatively before earlier branches have been resolved.
        - ensures any exceptions caused by earlier instructions are handled.

- improve performance by avoiding repeated false speculated loads,
    - Load Wait Table: PC -> one bit.
- skip speculative loads, if predicts L1 D-cache miss.

ReOrder Buffer (ROB)

- holds both the instructions and data for in-flight instructions,
- for [precise exceptions], ensuring results are committed in order to prevent data hazards.

To search for operands between register file and the reorder buffer,

- If the operand is located in the reorder buffer, it is represents the most recent value.
    - The reorder buffer is searched and accessed in parallel with the register file.
- A second approach is to maintain a register mapping table,
    - this records whether each register should be read from the reorder buffer or the register file.
    - It also records the entry in the reorder buffer where the register can be found.

Unified **Physical** RF

- with two register mapping tables, and a simplified ROB (in-order instruction queue),
    - the front-end future map represents the current potentially speculative state,
    - the architectural register map maintains the user visible state (checkpoint).
    - enhancement: one arch RM per branch prediction (MIPS 10K).
- upon detecting a mis-predicted branch,
    - the architectural register map is copied to the future register map and
    - any instructions along the mis-predicted path are discarded.
    - execution can then continue along the correct path with the correct register state.

ROB vs Unified **Physical** RF.

General

- give an assembly language program benefiting from superscalar techniques.
- switching between two configurations.

Software ILP (VLIW)

Exploits *Instruction*- and *Memory*-Level Parallelism via *static scheduling*.

vs. superscalar

- implementations.

Clustered architecture: see superscalar pipeline.

Local scheduling

Loop unrolling

Software pipelining with Rotating Register File (RRF)

Global scheduling

- trace vs superblock scheduling

Conditional / Predicated operations

- vs. branch prediction and limitations

Memory reference speculation

- compiler + advanced load address table (ALAT).

Variable-length bundles of independent instructions

VLIW	**fixed-width bundle**	**var-len bundle**
code density, i-cache size, instr fetch	↑ no-ops	↓ only st., end
i-cache hit rate	↓	↑
fixed SLOT-to-FU	✓	×
hardware	simpler	+ decoding; check before issue; + interconnect, muxes for op, operands to FU.

Binary compatibility

Multi-threaded processors

Exploits *Thread*-Level Parallelism.

Coarse-grained MT

Fine-grained MT

- VLIW vs fine-grained multi-threaded
- round-robin thread schedule, functional units

SMT

- threads characteristics
- SMT vs Multi-core
- (store) multi-copy atomic

Memory hierarchy

Direct-mapped vs. set/fully associative cache

- hit rate
- reduce conflict misses for direct-mapped cache

Block replacement policy

- LRU

Hit time calculation

Virtual addressing and caching

- VIPT

Cache: optimizing performance

- list all the techniques
- trade-off between memory vs. computation.
- memory layout and access order for better locality
- loop interchange, loop fusion, array merging, array padding, cache blocking and alignment of data structures.
- multi-banked cache
- merging or coalescing write-buffer violates TSO?
- load/store buffer
- (stride) prefetching hardware
- non-blocking cache (out-of-order)
- avoid false sharing in multiprocessor systems
- private variables in the same cache line accessed by different threads/cores

Multi-level cache hierarchy

- why multi-level ? hit time vs. number of hits
- L1, L2 examples

Inclusive vs. exclusive caches vs. NINE

- private vs. shared L2 caches
- non-inclusive

Addressing and cache

Reference: Memory address calculation.

Vector processors / SIMD

Exploits *Data*-Level Parallelism.

- VLIW vs SIMD

Potential advantages

- energy efficiency

Vector chaining (RaW) and tailgating (WaR)

Precise exceptions: please refer to the scalar pipeline section.

ISA: vector length

Multi-core processors

Exploits *Chip* Multiprocessing.

vs. superscalar

- less power but equal performance
- modern architecture: multi-core > single multi-threaded

Multi-banked caches

Cache partitioning

Cache coherence

invalidate vs. update

snoopy protocol

- bus's feature here, GPU
- MESI
- inclusion policy benefit for snoopy protocol
- ring interconnect
- via multiple buses for a greater number of processors.

directory protocol

- for each cache line, store a list of sharers and the exclusive individual processor in the directory.
- inclusive directory for non-inclusive caches
- hierarchy of on-chip caches, clustered cache

Memory consistency

Sequential Consistency vs. Total Store Order

- SC vs. TSO
- coalescing write buffer violates TSO?

Store atomicity

- SMT, (store) multi-copy atomic

False sharing: cache line/block granularity

On-chip interconnection network

- virtual channels
- mesh network, H-tree
- large scale networks: challenges and constraints

Specialised processors

Exploits *Accelerator*-Level Parallelism.

Heterogeneous/asymmetric vs homogeneous/symmetric

GPU

Domain-Specific Accelerators (DSA)