CS184b: Computer Architecture [Single Threaded Architecture: abstractions, quantification, and optimizations]

Day1: January 3, 2000

"Architecture" and overview

Today

- This Quarter
- What is Architecture?
  - Why?
- Next Week

CS184 Sequence

- A - structure and organization
  - raw components, building blocks
  - design space
- B - single threaded architecture
  - emphasis on abstractions and optimizations including quantification
- C - multithreaded architecture

Topics this Quarter

- "Architecture"
- Instruction-Set Architecture (ISA)
  - including pipeline parallelism
- Exceptions
- Instruction-Level Parallelism (ILP)
- Memory Architecture and Optimization
  - Caching and Virtual Memory
- Binary Translation

Next Quarter

- Multithreaded Abstractions, Optimization, and Structures
  - dataflow
  - multithreaded
  - message passing
  - shared memory
  - vector/SIMD (could be single threaded)
  - multiprocessor interconnect
  - defect and fault tolerance (also single thread)

Reading

- Will rely on much more than last term
- Will use textbook (Hennessy and Patterson)
  - chapters 1-5 this term
  - maybe some later chapters next quarter
- Lectures more to complement text than completely overlap
  - going to cover some pretty rich topics every week or two…can't do it in 3-6 hours of lecture
- Reader with classic papers

Assignments

- Will pull some from text
- Will try to emphasize experiments and measurement
  - (weren't able to do so much last term; more ready-made apps and tools to leverage this term)
  - mostly using simplescalar
    - MIPS-like architectural simulator
- Still like the idea of themes / running application(s) to analyze

Logistics

- Due Tuesday 5pm (out prev. Wed. class)
- Still want electronic
  - no handwriting/hand drawing
- Office/Lab hours:
  - Monday 2-4pm
  - (this Friday 2-4pm; will be away Monday)

Themes for Quarter

- Recurring
  - "cached" answers and change
  - merit analysis (cost/performance)
  - dominant/bottleneck resource requirements
  - structure/common case
- New/new focus
  - measurement
  - abstractions/semantics
  - abstractions 0, 1, infinity
  - dynamic data/event handling (vs. static)
  - predictability (avg. vs. worst case)

"Architecture"

What? Why?

"Architecture"

- "attributes of a system as seen by the programmer"
- "conceptual structure and functional behavior"
- Defines the visible interface between the hardware and software
- Defines the semantics of the program (machine code)

Architecture distinguished from Implementation

- IA32 architecture vs.
  - 80486DX2, AMD K5, Intel Pentium-II-700
- VAX architectures vs.
  - 11/750, 11/780, uVax-II
- PowerPC vs.
  - PPC 601, 604, 630 …
- Alpha vs.
  - EV4, 21164, 21264, …
- Admits to many different implementations of single architecture

Example Distinction: Memory Implementation

- **Abstraction**: large-flat memory
- **Implementation**:
  - multiple-levels of caches, varying sizes
  - virtual memory, with data residing on disk
  - relocation of physical memory placement
- One simple abstraction
  - hides details of implementation/timing
- Many implementations
  - varying costs, performance, technology

Why ?

- What's the value of this distinction?
- Why do we have it?

- What does it cost?

Value?

- Effort
- Economics
- Software Distribution

Value: Effort

- Reduce/minimize effort necessary to exploit new/different technology
- Number of programmers is small
- Rate of new machine/technology advance is large
- Key enabler to riding the technology curve

Value: Economics

- Preserve software investment
  - both uniquely developed and commercial
- Lower barrier to acceptance of new machine
  - all your old code runs…just faster!
- Offer range of scaling:
  - need more power --> buy different/better/newer machine
  - have less money --> buy the cheaper machine
  - little/no software effort to support

Value: Software Distribution

- Vendor not want to sell source
  - "give away" their techniques/technology/IP in a way which can be co-opted/reused
  - [pragmatic argument, not fundamental]

Pragmatic: Binary vs. Source Compatibility

- For various software engineering reasons (failures?)
  - source notoriously bad/problematic to port to new machine
  - entire application not all packaged up in one place
    - must find compatible libraries, compiler, compiler options, header files…
    - different (newer) compilers give different results

Pragmatic: Binary vs. Source Compatibility

- For various software engineering reasons (failures?)
- People generally more comfortable with binary compatibility
- ABI/Binary architectural definition smaller/tighter and more well defined?
- André: Shouldn't have to be this way…but that's where we are today

Fixed Points

- Must “fix” the interface
- Trick is picking what to expose in the interface and fix, and what to hide
- What are the “fixed points?”
  - how you describe the computation
  - primitive operations the machine understands
  - primitive data types
  - interface to memory, I/O
  - interface to system routines?

Abstract Away?

- Specific sizes
  - what fits in on-chip memory
  - available memory (to some extent)
  - number of peripherals
  - where 0, 1, infinity comes in
- Timing
  - individual operations
  - resources (e.g. memory)

Architectural Scalability

- Depends on robustness of fixed-points
  - address space
  - number of registers?
  - operations available
    - right level of abstraction?
  - Adequate primitives
    - e.g. atomic ops
  - sequential assumptions
  - single memory?
  - timing assumptions
    - e.g. branch delay, architectural cycles per op?

Change: Future like the past?

- JIT compilation
- Binary Translation
- More advanced compiler technology and algorithms
- Architectural convergence?

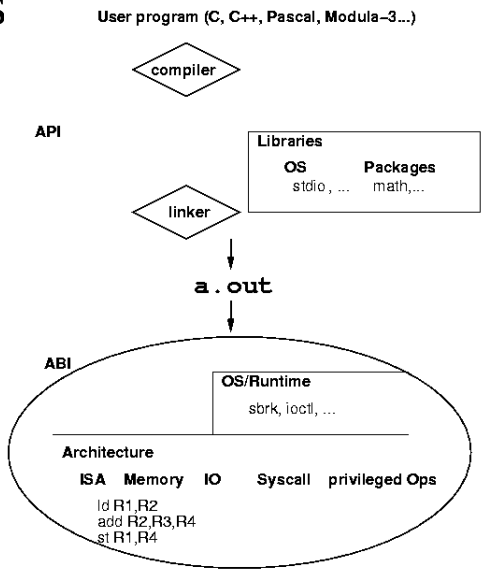
Conventional, Single-Threaded Abstraction

- Single, large, flat memory
- sequential, control-flow execution
- instruction-by-instruction sequential execution
- atomic instructions
- single-thread "owns" entire machine
  - isolation
- byte addressability
- unbounded memory, call depth

Embodiment

- C+OS-API
  - C+unix-API, C+Windows-API
- Compile to:
  - ISA+OS-ABI
    - *e.g.* MIPS+linux-ABI
- Wrap up in standard, executable definition
  - *e.g.* a.out

Abstractions



Next Week

- No Class (I'm away)
- Read about ISA
  - Discuss following week when back
  - Want to get through in these 2 weeks
- Assignment likely to use simplescalar to measure instruction effects

Big Ideas

- Architectural abstraction
  - define the fixed points
  - stable abstraction to programmer
  - admit to variety of implementation
  - ease adoption/exploitation of new hardware
  - reduce human effort