CS184b: Computer Architecture (Abstractions and Optimizations)

Day 1: March 31, 2003
Architecture Intro

Today

- This Quarter
- What is Architecture?
  - Why?
- Project Discussion

CS184 Sequence

- A - structure and organization
  - raw components, building blocks
  - design space
- B – architectural abstractions and optimization
  - emphasis on abstractions and optimizations including quantification
  - single and multiple threads

Topics this Quarter (1 of 2)

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

Topics (2 of 2)

- Dataflow
- Multithreaded
- Message Passing
- Shared Memory
- Vector/SIMD
- Multiprocessor Interface/Interconnect
- Defect and Fault Tolerance

Material

- Lots of material – will go fast
- …probably going to hit exposure over details

Lectures

- Same scheme as last term
  - Schedule MWF
  - Accommodate holes as necessary
  - Currently have 21 lectures on queue
  - 2 slots held in reserve

Reading

- Will rely on much more than last term
- Will use textbook (Hennessy and Patterson)
  - chapters 1-6 this term
- Lectures more to complement text than completely overlap
  - going to cover some pretty rich topics …can't do it in 1.5--3 hours of lecture
- Classic papers

Assignments

- Will pull some from text
- Emphasize experiments and measurement
  - mostly using simplescalar
    - MIPS-like architectural simulator
  - some with real machines
- Running application(s) to analyze

Logistics

- Four assignments on single threaded architectures
  - Due Monday 9am (out prev. Mon. class)
  - Still want electronic
    - no handwriting/hand drawing
- Last half: project
  - Try some weekly assignment targets
  - (more at end of class)

Themes for Quarter

- Recurring
  - "cached" answers and change
  - merit analysis (cost/performance)
  - dominant/bottleneck resource requirements
  - structure/common case

Themes for Quarter

- 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, Pentium-II-700, P6
- 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

Software Crisis

- Mid 1960's
  - Could build new machines at reasonable pace
  - Could not develop software for new machines fast enough

Historical Anecdotes

- Zuse from *The Computer, My Life*
- Brooks from *Software Pioneers*

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

Architecture Benefits

- ISA addressed the "software crisis"
  - Bottleneck to exploiting new machines was the need to write new software suites for them
- Preserve investment in software
  - Programmer education
- Permitted innovation in hardware
  - Use more/less hardware
  - Allow customers buy as much machine as they need
  - New substrates: TTL, ECL, NMOS, CMOS…

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

- Architecture requires we "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?

- VM/JIT compilation
- Binary Translation
- More advanced compiler technology and algorithms
- Architectural convergence?
  - Single Threaded ISA Maturity?

Conventional, Single-Threaded Abstraction

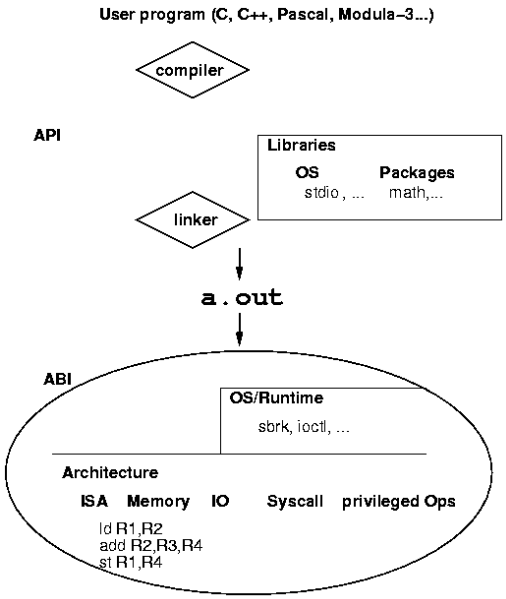
- Single, large, flat memory
- sequential, control-flow execution
- instruction-by-instruction sequential execution
- atomic instructions
- single-thread “owns” entire machine
  - isolation
- byte addressability
- unbounded memory, call depth

Embodiment

- C+OS-API
  - C+unix-API, C+Windows-API
- Compile to:
  - ISA+OS-ABI
    - *e.g.* x86+linux-ABI
- Wrap up in standard, executable definition
  - *e.g.* `a.out`

Abstractions

- Model for fist half of course
- How support?
- How optimize?
- Remarkable
  - How far implementation can diverge



Project Thoughts

Project

- 2nd half not lend as readily to canned simulator
- …and, want to get experience attacking a problem – with measurement, feedback on tradeoffs
- Go for more depth in one area to complement whirlwind tour in class

Idea

- Take architecture concept
- Measure conventional systems
- Modify Simulator
- Measure impact

- Again – small class – work as team on single project

Proposal

- Look at native streaming support added to conv. ISA
  - Adapt simple benchmark for streaming communication
  - Optimize/measure communication costs w/out (using conv. architecture as is)
  - Expand simulator to support
  - Measure/compare results

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