New-School Machine Structures
(It’s a bit more complicated!)

- Parallel Requests
  - Assigned to computer
  - e.g., Search “Katz”
- Parallel Threads
  - Assigned to core
  - e.g., Lookup, Ads
- Parallel Instructions
  - >1 instruction @ one time
  - e.g., 5 pipelined instructions
- Parallel Data
  - >1 data item @ one time
  - e.g., Add of 4 pairs of words
- Hardware descriptions
  - All gates @ one time

Levels of Representation/Interpretation

<table>
<thead>
<tr>
<th>High Level Language Program (e.g., C)</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language Program (e.g., MIPS)</td>
<td>Assembler</td>
</tr>
<tr>
<td>Machine Language Program (MIPS)</td>
<td>Machine Interpreter</td>
</tr>
</tbody>
</table>

Review

- Everything is a (binary) number in a computer
  - Instructions and data; stored program concept
- Assemblers can enhance machine instruction set to help assembly-language programmer
- Translate from text that easy for programmers to understand into code that machine executes efficiently: Compilers, Assemblers
- Linkers allow separate translation of modules

Agenda

- Compilers, Optimization, Interpreters, Just-In-Time Compiler
- Administrivia
- Dynamic Linking
- Technology Trends Revisited
- Technology Break
- Components of a Computer

What’s a Compiler?

- Compiler: a program that accepts as input a program text in a certain language and produces as output a program text in another language, while preserving the meaning of that text.
- The text must comply with the syntax rules of whichever programming language it is written in.
- A compiler’s complexity depends on the syntax of the language and how much abstraction that programming language provides.
  - A C compiler is much simpler than C++ Compiler
- Compiler executes before compiled program runs
Compiled Languages: 
Edit-Compile-Link-Run

Editor   
Source code   
Compiler   
Object code   
Linker   
Executable program

Compiler Optimization

- gcc compiler options
  - O1: the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time
  - O2: Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to -O, this option increases both compilation time and the performance of the generated code
  - O3: Optimize yet more. All -O2 optimizations and also turns on the -finline-functions, ...

What is Typical Benefit of Compiler Optimization?

- What is a typical program?
- For now, try a toy program: BubbleSort.c

Unoptimized MIPS Code

```mips
# define ARRAY_SIZE 20000
int main() {
    int iarray[ARRAY_SIZE], x, y, holder;
    for(x = 0; x < ARRAY_SIZE; x++)
        for(y = 0; y < ARRAY_SIZE-1; y++)
            if(iarray[y] > iarray[y+1]) {
                holder = iarray[y+1];
                iarray[y+1] = iarray[y];
                iarray[y] = holder;
            }
}
```

-O2 optimized MIPS Code

```mips
... code...
```

What’s an Interpreter?

- It reads and executes source statements executed one at a time
  - No linking
  - No machine code generation, so more portable
  - Start executing quicker, but run much more slowly than compiled code
  - Performing the actions straight from the text allows better error checking and reporting to be done
  - The interpreter stays around during execution
    - Unlike compiler, some work is done after program starts
    - Writing an interpreter is much less work than writing a compiler
Interpreted Languages: Edit-Run

Compiler vs. Interpreter

Advantages

Compilation:
- Faster execution
- Single file to execute
- Compiler can do better diagnosis of syntax and semantic errors, since it has more info than an interpreter (interpreter only sees one line at a time)
- Can find syntax errors before run program
- Compiler can optimize code

Interpreter:
- Easier to debug program
- Faster development time

Disadvantages

Compilation:
- Harder to debug program
- Takes longer to change source code, recompile, and relink

Interpreter:
- Slower execution times
- No optimization
- Need all of source code available
- Source code larger than executable for large systems
- Interpreter must remain installed while the program is interpreted

Java’s Hybrid Approach: Compiler + Interpreter

- A Java compiler converts Java source code into instructions for the Java Virtual Machine (JVM)
- These instructions, called bytecodes, are same for any computer / OS
- A CPU-specific Java interpreter interprets bytecodes on a particular computer

Java’s Compiler + Interpreter

Why Bytecodes?

- Platform-independent
- Load from the Internet faster than source code
- Interpreter is faster and smaller than it would be for Java source
- Source code is not revealed to end users
- Interpreter performs additional security checks, screens out malicious code
JVM uses Stack vs. Registers

\[ a = b + c; \]

\[ \Rightarrow \]

iload b ; push b onto Top Of Stack (TOS)
iload c ; push c onto Top Of Stack (TOS)
iadd ; Next to top Of Stack (NOS) =
; Top Of Stack (TOS) + NOS
istore a ; store TOS into a and pop stack

Java Bytecodes (Stack) vs. MIPS (Reg.)

Starting Java Applications

<table>
<thead>
<tr>
<th>Simple portable instruction set for the JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class files (Java bytecode)</td>
</tr>
<tr>
<td>Java Virtual Machine</td>
</tr>
</tbody>
</table>

Just In Time (JIT) compiler translates bytecode into machine language just before execution

Agenda

- Compilers, Optimization, Interpreters, Just-In-Time Compiler
- Administrivia
- Dynamic Linking
- Technology Trends Revisited
- Technology Break
- Components of a Computer

Projects

- Project 1: MapReduce on EC2
  - No credit if programs didn’t compile
  - 6 that used combiners worked after regrade
  - Can’t share even a little code with your friend; mostly your own is not your own
- Project 2: MIPS ISA simulator in C
  - Add ~ 200 (repetitive) lines of C code to framework
  - Lots of Cut & Past
  - Appendix B describes all MIPS instructions in detail
  - Make your own unit test!

Administrivia

- Lab 5 posted, Project 2 posted
- Homework, Proj 2- Part 1 Due Sunday @ 11:59:59
- Want in from the Wait List?
  - Sign up for Lab 019 (Friday, 7-9 PM – there is space available! 13 on wait list, 8 slots for Friday)
- Midterm is now on the horizon:
  - Exam: Tu, Mar 8, 6-9 PM, 145/155 Dwinelle
  - No discussion during exam week, no lecture March 8
  - TA Review: Su, Mar 6, 2-5 PM, 2050 VLSB
  - Small number of special consideration cases, due to class conflicts, etc.—contact Dave or Randy
EC2 instances over 2 weeks

<table>
<thead>
<tr>
<th>Time</th>
<th>Number of Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sat</td>
<td>50</td>
</tr>
<tr>
<td>Mon</td>
<td>60</td>
</tr>
<tr>
<td>Wed</td>
<td>70</td>
</tr>
<tr>
<td>Fri</td>
<td>80</td>
</tr>
<tr>
<td>Sat</td>
<td>90</td>
</tr>
<tr>
<td>Mon</td>
<td>100</td>
</tr>
<tr>
<td>Wed</td>
<td>110</td>
</tr>
<tr>
<td>Fri</td>
<td>120</td>
</tr>
</tbody>
</table>

EC2 usage: $/student
Avg $40, 25th $30, 75th $55

Histogram of Student EC2 Usage

Computers In The News

- IBM Watson plays Jeopardy! with champions
  - A significant milestone in computing, on par with IBM Deep Blue vs. Kasparov in chess in 1997
- Tonight & Wed 2/16 7-7:30PM KGO Channel 7

Dynamic Linking

- Only link/load library procedure when it is called
  - Automatically picks up new library versions
  - Requires procedure code to be relocatable
  - Avoids image bloat caused by static linking of all (transitively) referenced libraries
- Dynamic linking is default on UNIX and Windows Systems

Dynamic Linking Idea

- 1st time pay extra overhead of DLL (Dynamically Linked Library), subsequent times almost no cost
- Compiler sets up code and data structures to find desired library first time
- Linker fixes up address at runtime so fast call subsequent times
- Note that return from library is fast every time

Dynamic Linkage

1. Call to DLL Library
2. Indirection table that initially points to stub code
3. Stub: Loads routine ID so can find desired library
4. Jump to linker/loader
5. Linker/loader code finds desired library and edits jump address in indirection table, jumps to desired routine
6. Dynamically mapped code executes and returns
Dynamic Linkage

- Call to DLL Library
- Indirection table that initially points to stub code
- Stub: Loads routine ID so can find desired library, jumps to linker/loader
- Linker/loader code finds desired library and edits jump address in indirection table, jumps to desired routine
- Dynamically mapped code executes and returns

Technology Cost over Time: What does Improving Technology Look Like?

![Graph showing technology cost over time]

Moore’s Law

“Moore’s Law: The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.” (from 50 in 1965)

Gordon Moore, “Cramming more components onto integrated circuits,” Electronics, Volume 38, Number 8, April 19, 1965

“Integrated circuits will lead to such wonders as home computers—or at least terminals connected to a central computer—automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today.”

Predicts: 2X Transistors / chip every 2 years

Moore’s Law

[Growth in memory capacity slowing]

Memory Chip Size

- Cost $ vs. Time
- How Can Tech 2 Replace Tech Gen 1?
- Technology Generations:
  - Technology Generation 1
  - Technology Generation 2
  - Technology Generation 3

Gordon Moore, Intel Cofounder
B.S. Cal 1950!
End of Moore’s Law?
- It’s also a law of investment in equipment as well as increasing volume of integrated circuits that need more transistors per chip
- Exponential growth cannot last forever
- More transistors/chip will end during your careers—2020? 2025?
- (When) will something replace it?

Technology Trends: Uniprocessor Performance (SPECint)

Power is proportional to Capacitance * Voltage^2 * Frequency of switching
- What is the effect on power consumption of:
  - “Simpler” implementation (fewer transistors)?
  - Smaller implementation (shrunk down design)?
  - Reduced voltage?
  - Increased clock frequency?

Doing Nothing Well—NOT!
- Traditional processors consume about two thirds as much power at idle (doing nothing) as they do at peak
- Higher performance (server class) processors approaching 300 W at peak
- Implications for battery life?

Computer Technology: Growing, But More Slowly
- Processor
  - Speed 2x / 1.5 years (since ’85-’05) (slowing!)
  - Now >2 cores / 2 years
  - When you graduate: 3-4 GHz, 10-14 Cores
- Memory (DRAM)
  - Capacity: 2x / 2 years (since ’96) (slowing!)
  - Now 2X/3-4 years
  - When you graduate: 8-16 GigaBytes
- Disk
  - Capacity: 2x / 1 year (since ’97)
  - 20X size last decade
  - When you graduate: 6-12 Terabytes
- Network
  - Core: 2x every 2 years
  - Access: 100-1000 mbps from home, 1-10 mbps cellular
Five Components of a Computer

- Control
- Datapath
- Memory
- Input
- Output

The Processor

- Processor (CPU): the active part of the computer, which does all the work (data manipulation and decision-making)
- Datapath: portion of the processor which contains hardware necessary to perform operations required by the processor (the brawn)
- Control: portion of the processor (also in hardware) which tells the datapath what needs to be done (the brain)

Stages of the Datapath: Overview

- Problem: a single, atomic block which "executes an instruction" (performs all necessary operations beginning with fetching the instruction) would be too bulky and inefficient
- Solution: break up the process of "executing an instruction" into stages or phases, and then connect the phases to create the whole datapath
  - Smaller phases are easier to design
  - Easy to optimize (change) one phase without touching the others

Instruction Level Parallelism

Project 2 Warning

- You are going to write a simulator in C for MIPS, implementing these 5 phases of execution
Phases of the Datapath (1/5)

• There is a wide variety of MIPS instructions: so what general steps do they have in common?

• Phase 1: Instruction Fetch
  – No matter what the instruction, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy)
  – Also, this is where we Increment PC (that is, PC = PC + 4, to point to the next instruction: byte addressing so + 4)

• Simulator: Instruction = Memory[PC]; PC+=4;

Phases of the Datapath (2/5)

• Phase 2: Instruction Decode
  – Upon fetching the instruction, we next gather data from the fields (decode all necessary instruction data)
  – First, read the opcode to determine instruction type and field lengths
  – Second, read in data from all necessary registers
    • For add, read two registers
    • For addi, read one register
    • For jal, no reads necessary

Simulator: Instruction = Memory[PC]; PC+=4;

Phases of the Datapath (3/5)

• Phase 3: ALU (Arithmetic-Logic Unit)
  – Real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (slt)
  – What about loads and stores?
    • lw $t0, 40($t1)
    • Address we are accessing in memory = the value in $t1 PLUS the value 40
    • So we do this addition in this stage

• Simulator: Result = Register1 op Register2; Address = Register1 + Addressfield

Phases of the Datapath (4/5)

• Phase 4: Memory Access
  – Actually only the load and store instructions do anything during this phase; the others remain idle during this phase or skip it all together
  – Since these instructions have a unique step, we need this extra phase to account for them
  – As a result of the cache system, this phase is expected to be fast

• Simulator: Register[rtfield] = Memory[Address] or Memory[Address] = Register[rtfield]

Phases of the Datapath (5/5)

• Phase 5: Register Write
  – Most instructions write the result of some computation into a register
  – E.g.: arithmetic, logical, shifts, loads, slt
  – What about stores, branches, jumps?
    • Don’t write anything into a register at the end
    • These remain idle during this fifth phase or skip it all together

• Simulator: Register[rtfield] = Result
Summary

- Key Technology Trends and Limitations
  - Transistor doubling BUT power constraints and latency considerations limit performance improvement
  - (Single Processor) computers are about as fast as they are likely to get, exploit parallelism to go faster

- Five Components of a Computer
  - Processor/Control + Datapath
  - Memory
  - Input/Output: Human interface/KB + Mouse, Display, Storage ... evolving to speech, audio, video

- Architectural Family: One Instruction Set, Many Implementations