Computer Organization

Topics:

- Theme
- Six great realities of computer systems
- How this fits within CS curriculum
Course Theme

- Abstraction is good, but don’t forget reality!

Courses to date emphasize abstraction

- Abstract data types, asymptotic analysis

These abstractions have limits

- Especially in the presence of bugs
- Need to understand underlying implementations

Useful outcomes

- Become more effective programmers
  - Able to find and eliminate bugs efficiently
  - Able to tune program performance
- Prepare for later “systems” classes in CS & ECE
  - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems
Great Reality #1

*Int’s are not Integers, Float’s are not Reals*

**Examples**

- **Is \(x^2 \geq 0\)?**
  - Float’s: Yes!
  - Int’s:
    - \(40000 \times 40000 \rightarrow 16000000000\)
    - \(50000 \times 50000 \rightarrow ??\)

- **Is \((x + y) + z = x + (y + z)\)?**
  - Unsigned & Signed Int’s: Yes!
  - Float’s:
    - \((1e20 + -1e20) + 3.14 \rightarrow 3.14\)
    - \(1e20 + (-1e20 + 3.14) \rightarrow ??\)
Computer Arithmetic

Does not generate random values

- Arithmetic operations have important mathematical properties

Cannot assume “usual” properties

- Due to finiteness of representations
- Integer operations satisfy “ring” properties
  - Commutativity, associativity, distributivity
- Floating point operations satisfy “ordering” properties
  - Monotonicity, values of signs

Observation

- Need to understand which abstractions apply in which contexts
- Important issues for compiler writers and serious application programmers
Great Reality #2

You’ve got to know assembly

Chances are, you’ll never write program in assembly

- Compilers are much better & more patient than you are

Understanding assembly key to machine-level execution model

- Behavior of programs in presence of bugs
  - High-level language model breaks down
- Tuning program performance
  - Understanding sources of program inefficiency
- Implementing system software
  - Compiler has machine code as target
  - Operating systems must manage process state
Assembly Code Example

Sum of Integers
  ■ Finds the sum of the integers from 1 to $n$

C Code:

```c
int find_sum (int n) {
    int i, sum;

    sum = 0;
    for (i=1; i<=n; i++) {
        sum += i;
    }
    return sum;
}
```
Code to Sum Integers

We can use the compiler to translate this code to assembly:

```assembly
find_sum:        movl 4(%esp), %ecx
                 xorl %eax, %eax
                 testl %ecx, %ecx
                 jle .L4
                 movl $1, %edx
                 addl $1, %ecx
.L5:
                 addl %edx, %eax
                 addl $1, %edx
                 cmpl %ecx, %edx
                 jne .L5
.L4:
                 ret
```
Great Reality #3

Memory Matters

Memory is not unbounded
- It must be allocated and managed
- Many applications are memory dominated

Memory referencing bugs especially pernicious
- Effects are distant in both time and space

Memory performance is not uniform
- Cache and virtual memory effects can greatly affect program performance
- Adapting program to characteristics of memory system can lead to major speed improvements
Memory Referencing Bug Example

```c
main ()
{
    long int a[2];
    double d = 3.14;
    a[2] = 1073741824; /* Out of bounds reference */
    printf("d = %.15g\n", d);
    exit(0);
}
```

<table>
<thead>
<tr>
<th>Alpha</th>
<th>MIPS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-g</code> 5.30498947741318e-315</td>
<td>3.1399998664856</td>
<td>3.14</td>
</tr>
<tr>
<td><code>-O</code> 3.14</td>
<td>3.14</td>
<td>3.14</td>
</tr>
</tbody>
</table>

(Linux version gives correct result, but implementing as separate function gives segmentation fault.)
Memory Referencing Errors

C and C++ do not provide any memory protection
- Out of bounds array references
- Invalid pointer values
- Abuses of malloc/free

Can lead to nasty bugs
- Whether or not bug has any effect depends on system and compiler
- Action at a distance
  - Corrupted object logically unrelated to one being accessed
  - Effect of bug may be first observed long after it is generated

How can I deal with this?
- Program in Java, Lisp, or ML
- Understand what possible interactions may occur
- Use or develop tools to detect referencing errors
Memory Performance Example

Implementations of Matrix Multiplication

- Multiple ways to nest loops

```c
/* ijk */
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        sum = 0.0;
        for (k=0; k<n; k++)
            sum += a[i][k] * b[k][j];
        c[i][j] = sum;
    }
}

/* jik */
for (j=0; j<n; j++) {
    for (i=0; i<n; i++) {
        sum = 0.0;
        for (k=0; k<n; k++)
            sum += a[i][k] * b[k][j];
        c[i][j] = sum;
    }
}
```
Matmult Performance (Alpha 21164)

Too big for L1 Cache  Too big for L2 Cache

matrix size (n)
Blocked matmult perf (Alpha 21164)
Great Reality #4

There’s more to performance than asymptotic complexity

Constant factors matter too!

- Easily see 10:1 performance range depending on how code written
- Must optimize at multiple levels: algorithm, data representations, procedures, and loops

Must understand system to optimize performance

- How programs compiled and executed
- How to measure program performance and identify bottlenecks
- How to improve performance without destroying code modularity and generality
Great Reality #5

Computers do more than execute programs

They need to get data in and out
  • I/O system critical to program reliability and performance

They communicate with each other over networks
  • Many system-level issues arise in presence of network
    ● Concurrent operations by autonomous processes
    ● Coping with unreliable media
    ● Cross platform compatibility
    ● Complex performance issues
Great Reality #6

Computers are made from physical devices

A computer is made of logic gates and memories

- How do these devices work?
- How do transistors and capacitors form gates and memories?
- How does Boolean logic help us compute?
Course Perspective

Most Systems Courses are Builder-Centric

- **Computer Architecture**
  - Design pipelined processor in Verilog

- **Operating Systems**
  - Implement large portions of operating system

- **Compilers**
  - Write compiler for simple language

- **Networking**
  - Implement and simulate network protocols
Course Perspective (Cont.)

Our Course is Programmer-Centric

- Purpose is to show how by knowing more about the underlying system, one can be more effective as a programmer
- Enable you to
  - Write programs that are more reliable and efficient
  - Incorporate features that require hooks into OS
    » E.g., concurrency, signal handlers
- Not just a course for dedicated hackers
  - We bring out the hidden hacker in everyone
- Cover material in this course that you won’t see elsewhere