

## Introduction to Computer Systems

Dr. Steve Goddard  
[goddard@cse.unl.edu](mailto:goddard@cse.unl.edu)

<http://cse.unl.edu/~goddard/Courses/JDEP284>

## Giving credit where credit is due

- Most of slides for this lecture are based on slides created by Drs. Bryant and O'Hallaron, Carnegie Mellon University.
- Some examples and slides are based on lecture notes created by Dr. Shard Seth, UNL.
- I have modified them and added new slides.

2

## Topics

- Why do we care about this stuff?
- Course theme
- Five great realities of computer systems
- Computer system overview

3

## Why Do We Care...

### Rapidly changing field:

- vacuum tube -> transistor -> IC -> VLSI
- doubling every 1.5 years:
  - memory capacity
  - processor speed *(Due to advances in technology and organization)*

### Things you'll be learning:

- how computers work, a basic foundation
- how to analyze their performance (or how not to!)
- issues affecting modern processors (caches, pipelines)

### Why learn this stuff?

- you want to call yourself a "computer scientist"
- you want to build software people use (need performance)
- you need to make a purchasing decision or offer "expert" advice

4

## 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 & CE
  - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems

5

## Great Reality #1

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

### Examples

- Is  $x^2 \geq 0$ ?
  - Float's: Yes!
  - Int's:
    - »  $40000 * 40000 \rightarrow 1600000000$
    - »  $50000 * 50000 \rightarrow ??$
- Is  $(x + y) + z = x + (y + z)$ ?
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - »  $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
    - »  $1e20 + (-1e20 + 3.14) \rightarrow ??$

6

## 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

7

## Great Reality #2

### You’ve got to know assembly

### Chances are, you’ll never write a program in assembly

- Compilers are much better & more patient than you are

### Understanding assembly is key to understanding the 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

8

## Assembly Code Example

### Time Stamp Counter

- Special 64-bit register in Intel-compatible machines
- Incremented every clock cycle
- Read with rdtscl instruction

### Application

- Measure time required by procedure
  - In units of clock cycles

```
double t;
start_counter();
P();
t = get_counter();
printf("P required %f clock cycles\n", t);
```

9

## Code to Read Counter

- Write small amount of assembly code using GCC’s asm facility
- Inserts assembly code into machine code generated by compiler

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

/* Set *hi and *lo to the high and low order bits
of the cycle counter.
*/
void access_counter(unsigned *hi, unsigned *lo)
{
    asm("rdtscl; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        : "%edx", "%eax");
}
```

10

## Code to Read Counter

```
/* Record the current value of the cycle counter. */
void start_counter()
{
    access_counter(&cyc_hi, &cyc_lo);
}

/* Number of cycles since the last call to start_counter. */
double get_counter()
{
    unsigned ncyc_hi, ncyc_lo;
    unsigned hi, lo, borrow;
    /* Get cycle counter */
    access_counter(&ncyc_hi, &ncyc_lo);
    /* Do double precision subtraction */
    lo = ncyc_lo - cyc_lo;
    borrow = lo > ncyc_lo;
    hi = ncyc_hi - cyc_hi - borrow;
    return (double) hi * (1 << 30) * 4 + lo;
}
```

11

## Measuring Time

### Trickier than it Might Look

- Many sources of variation

### Example

- Sum integers from 1 to n

n	Cycles	Cycles/n
100	961	9.61
1,000	8,407	8.41
1,000	8,426	8.43
10,000	82,861	8.29
10,000	82,876	8.29
1,000,000	8,419,907	8.42
1,000,000	8,425,181	8.43
1,000,000,000	8,371,2305,591	8.37

12

## Great Reality #3

### Memory Matters

#### Memory is not unbounded

- It must be allocated and managed
- Many applications are memory dominated

#### Memory referencing bugs are 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

13

## Memory Referencing Bug Example

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

	Alpha	MIPS	Linux
-g	5.30498947741318e-315	3.1399998664856	3.14
-O	3.14	3.14	3.14

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

14

## 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

15

## Memory Performance Example

### Implementations of Matrix Multiplication

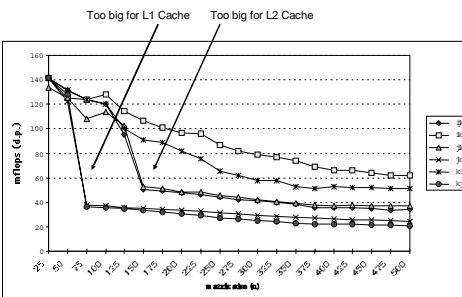
- Multiple ways to nest loops

```
/* 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;
  }
}
```

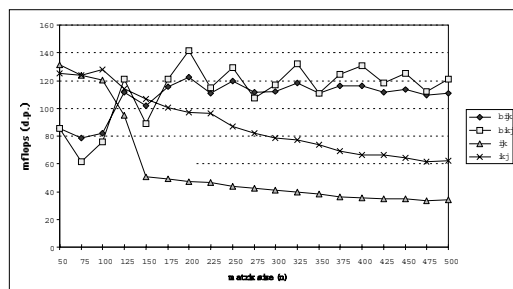
16

## Matmult Performance (Alpha 21164)



17

## Blocked matmult perf (Alpha 21164)



18

## 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 is 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

19

## 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

20

## 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

21

## Course Perspective (Cont.)

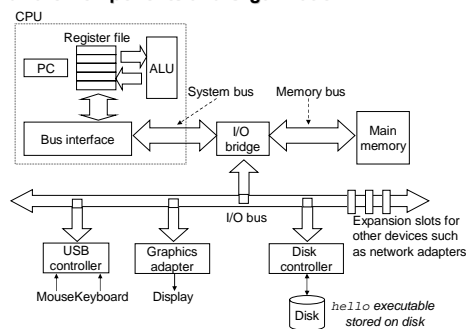
### This Course is Programmer-Centric

- 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

22

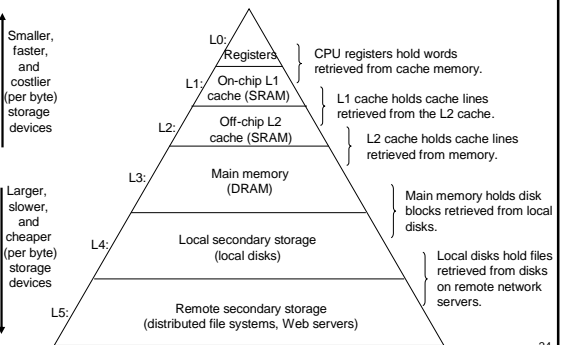
## What is a computer?

### Hardware Components and Organization:



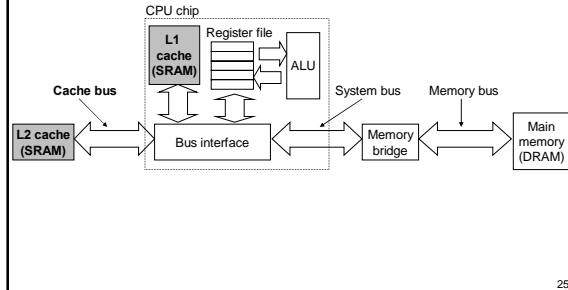
23

## Memory Hierarchy



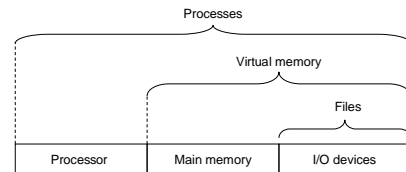
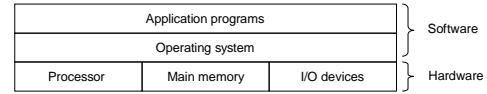
24

## Cache Memories



25

## OS Abstracts HW



26

## Summary

**The Computer system is more than just hardware!**

**We have to understand both the hardware and the system interfaces to properly understand and use a computer.**

**The rest of this semester will be spent studying these concepts in much more detail.**

27