Course Overview and Introduction

CSci 2021: Machine Architecture and Organization
Lecture #1, January 21st, 2015

Your instructor: Stephen McCamant

Based on slides originally by:
Randy Bryant, Dave O'Hallaron, Antonia Zhai

Overview

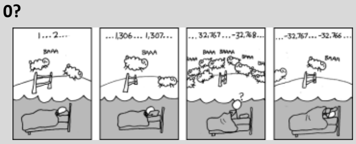
- Course theme
- Four realities
- How the course fits into the CS curriculum
- Logistics

Course Theme: Abstraction Is Good But Don't Forget Reality

- **Most CS courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS & EE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems

Great Reality #1: Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!



  - Int's:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

Code Security Example

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

- **Similar to code found in FreeBSD's implementation of getpeername**
- **There are legions of smart people trying to find vulnerabilities in programs**

Typical Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

Malicious Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

7

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all "usual" mathematical 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

8

Great Reality #2: You've Got to Know Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But, assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the lingua franca

9

Assembly Code Example

- **Time Stamp Counter**
  - Special 64-bit register in Intel-compatible machines
  - Incremented every clock cycle
  - Read with rdtsc instruction
- **Application**
  - Measure time (in clock cycles) required by procedure

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

10

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("rdtsc; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        :
        : "%edx", "%eax");
}
```

11

Great Reality #3: Memory Matters
Random Access Memory Is an Unphysical Abstraction

- **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

12

Memory Referencing Bug Example

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  →   3.14
fun(1)  →   3.14
fun(2)  →   3.1399998664856
fun(3)  →   2.00000061035156
fun(4)  →   3.14, then segmentation fault
```

- **Result is architecture specific**

13

Memory Referencing Bug Example

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  →   3.14
fun(1)  →   3.14
fun(2)  →   3.1399998664856
fun(3)  →   2.00000061035156
fun(4)  →   3.14, then segmentation fault
```

Explanation:

Saved State	4
d7 ... d4	3
d3 ... d0	2
a[1]	1
a[0]	0

Location accessed by `fun(i)`

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, Ruby or ML
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

15

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

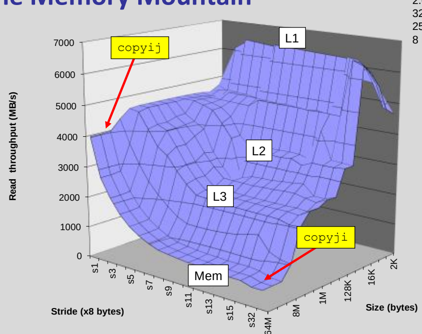
21 times slower (Pentium 4)

- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how step through multi-dimensional array

16

The Memory Mountain

Intel Core i7
2.67 GHz
32 KB L1 d-cache
256 KB L2 cache
8 MB L3 cache



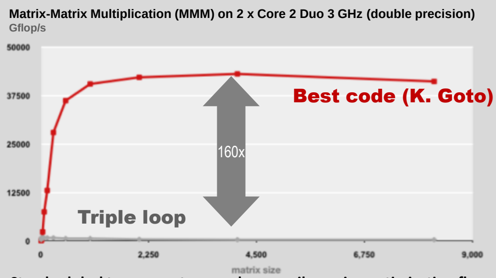
17

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - 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

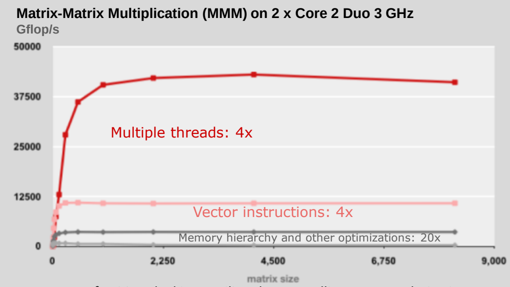
18

Example Matrix Multiplication



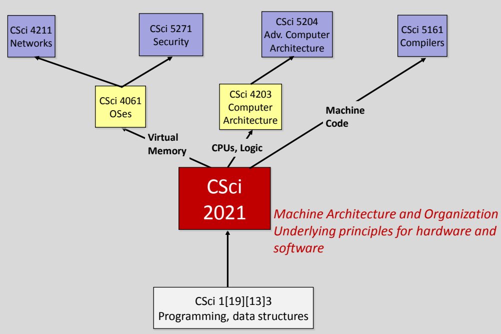
- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have *exactly* the same operations count ($2n^3$)
- What is going on?

MMM Plot: Analysis



- Reason for 20x: Blocking or tiling, loop unrolling, array scalarization, instruction scheduling, search to find best choice
- *Effect: fewer register spills, L1/L2 cache misses, and TLB misses*

Role within Computer Science



Course Perspective

- **Most Systems Courses are Builder-Centric**
  - Computer Architecture (CSci 4203)
    - Design pipelined processor in Verilog
  - Compilers (CSci 5161)
    - Write compiler for simple language
- **2021 is Programmer-Centric**
  - Purpose is to show how by knowing more about the underlying system, one can be more effective as a programmer
  - Including, enable you to write programs that are more reliable and efficient
  - 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

Textbooks

- **Required: Randal E. Bryant and David R. O'Hallaron,**
  - "Computer Systems: A Programmer's Perspective, Second Edition" (CS:APP2e), Prentice Hall, 2011
  - http://csapp.cs.cmu.edu
  - Paper version recommended
    - Tests are open book, open notes, any paper, no electronics
  - Used quite heavily
    - How to solve labs
    - Practice problems typical of exam problems
- **Optional: a book about C**
  - Labs, homework, and tests require reading and writing code in C
  - Some possible suggestions listed on the course home page

Course Components

- **Lectures: Higher level concepts**
- **Discussion (AKA Recitation) Sections**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
- **Labs (5)**
  - The heart of the course, fun but often time-consuming
  - About 2 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Homework Assignments (5)**
  - Practice thinking and writing, similar to tests, partially graded
- **Two Quizzes and One Final Exam**
  - Test your understanding of concepts & mathematical principles

Electronic Resources

- **Class Web Page:**
  - **http://www-users.cs.umn.edu/classes/Spring-2015/csci2021/**
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions
  - Clarifications to assignments
- **Moodle Page**
  - Discussion forums
  - Online turn-in of labs
- **Where to send electronic questions?**
  1. Moodle forum
  2. Responsible TA for a homework or lab
  3. cs2021s15-staff@cs.umn.edu (general mailing list)

25

Policies: Assignments, Labs, And Exams

- **Groups? No.**
  - You must work alone on all assignments
- **Hand-in process**
  - Labs due online, by 11:55pm on a weekday evening
  - Homeworks due on paper, by start of class on course days
- **Conflicts**
  - There will be no makeup quizzes
  - One excused missed quiz will be replaced by more weight on final
- **Appealing grades**
  - Within 7 days of completion of grading
    - Following procedure described in syllabus
  - Note, we will regrade the whole assignment/exam

26

Facilities

- **Do labs using CSELabs Linux machines**
  - Accessible from on-campus labs or remotely (SSH)
  - Get an account if you don't have one already, login with UMN account name and password
  - Some VMs specially for us are in preparation
  - Working on your own machines may sometimes be possible, but is not supported by course staff
  - Grade based on how it runs on our machines, so be sure to test there

27

Timeliness

- **Late labs and homeworks**
  - Lose 15% for each day or fraction late
  - No credit after 3 days
- **Catastrophic events**
  - Major illness, death in family, …, (full list in syllabus)
  - Are an exception, and can be excused
- **Advice**
  - The course is fast-paced
  - Once you start running late, it's really hard to catch up

28

Cheating

- **What is cheating?**
  - Sharing code: by copying, retyping, looking at, or supplying a file
  - Coaching: helping your friend to write a lab, line by line
  - Copying code from previous course or from elsewhere on WWW
    - Only allowed to use code we supply, or from CS:APP website
- **What is NOT cheating?**
  - Explaining how to use systems or tools
  - Helping others with high-level design issues
- **Penalty for cheating:**
  - Minimum: 0 grade on assignment or exam, report to campus OSCAI
- **Detection of cheating:**
  - We do check
  - Our tools for doing this are better than most cheaters think!

29

Policies: Grading

- **Tests (60%): weighted 10%, 10%, 40% (final)**
- **Labs (30%)**
- **Homework Assignments (10%)**

- **Guaranteed:**
  - ≥ 90%: A-
  - ≥ 80%: B-
  - ≥ 70%: C-

- **Curve:**
  - Will likely apply, in your favor only, so that grade distribution is similar to historical averages.

30

Lab 0: Logistics Practice

- Learn how to log into Unix machine, edit and compile a program
- “Hello, world”-style program that just prints a message
- Due on Moodle, Monday by 11:55pm
- Worth only one point (extra credit), but good to practice if you haven’t work with C or Unix much before
- More details covered in tomorrow’s discussion sections

Data Representation

- Topics
  - Bit-level operations
  - Machine-level integers and floating-point
  - C operators and things that can go wrong
- Assignments
  - L1 (Data lab): Manipulating bits

Machine-level Program Representation

- Topics
  - Assembly language programs
  - Representation of C control and data structures
  - E.g., what does a compiler do?
- Assignments
  - L2 (Bomb lab): Defusing a binary bomb
  - L3 (Buffer lab): Hacking a program that has a buffer overflow bug

CPU Architecture

- Topics
  - The parts of a CPU and how they work together
  - Related topic: optimizing to use CPU resources more efficiently
- Assignments
  - L4 (Architecture lab): Modify a simplified CPU and some code that runs on top of it

The Memory Hierarchy

- Topics
  - Memory technology, memory hierarchy, caches, disks, locality
  - How virtual memory works
- Assignments
  - L5 (Cache lab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.

Logic Design

- Topics
  - A level below architecture: how to “program” with gates and wires
  - Lowers abstraction all the way to how hardware works
  - Basis for later courses in computer architecture
- Assignments
  - No time for a lab, covered in final homework assignment

Lab Rationale

- **Each lab has a well-defined goal such as solving a puzzle or winning a contest**
- **Doing the lab should result in new skills and concepts**
- **We try to use competition in a fun and healthy way**
  - Set a reasonable threshold for full credit
  - Post intermediate results (anonymized) on Web page for glory!

Welcome
and Enjoy!