Course Overview and Introduction

CSci 2021: Machine Architecture and Organization
Lecture #1, January 22nd, 2020

Your instructor: Stephen McCamant

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

Overview

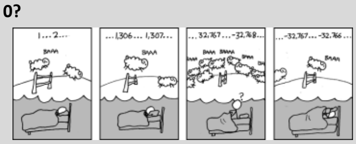
- Course themes
- Four realities
- How the course fits into the CS curriculum
- Logistics

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 $x^2 \geq 0$?**
  - Floats: Yes!



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

Cartoon 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);
    . . .
}
```


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

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

- **Chances are, you'll never write full programs in assembly**
  - Compilers are much better & more patient than you are
- **But, assembly is key to the machine-level execution model**
  - Behavior of programs in the presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done or 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

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);
```


Code to Read Counter

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

```
/* Return the cycle count as a 64-bit integer */

unsigned long access_counter(void)
{
    unsigned long high, low;
    asm("rdtsc"
        : "=d" (high), "=a" (low));
    return (high << 32) | low;
}
```


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 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

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0) →   3.14
fun(1) →   3.14
fun(2) →   3.1399998664856
fun(3) →   2.00000061035156
fun(4) →   3.14
fun(6) →   Segmentation fault
```

- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

Explanation:

	Critical State	6	Location accessed by `fun(i)`
	?	5	
	?	4	
`struct_t`	`d7 ... d4`	3	
`struct_t`	`d3 ... d0`	2	
`struct_t`	`a[1]`	1	
`struct_t`	`a[0]`	0	

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

Memory System Performance Example

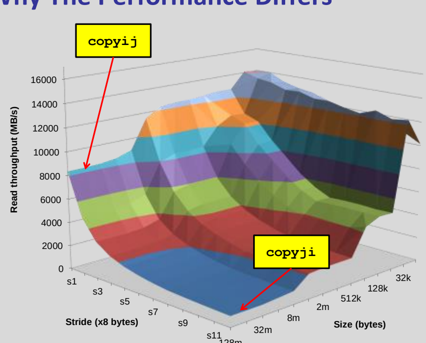
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
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

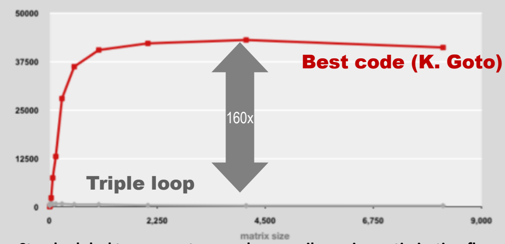
Why The Performance Differs



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

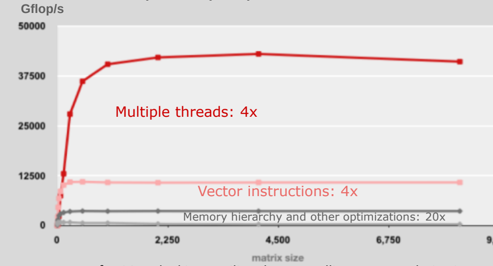
- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Example Matrix Multiplication



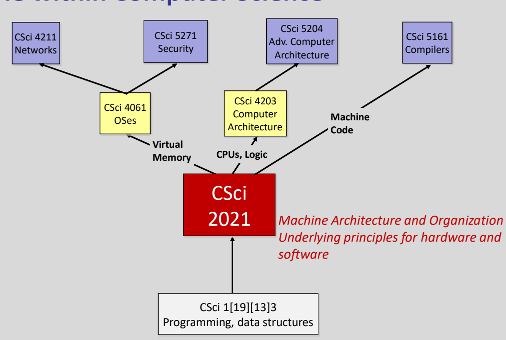
- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have exactly the same operations count ($2n^3$)
- What is going on?

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

Things That Are Different This Semester

- **Not committing to talk through every slide in lecture**
  - Will still cover the most important points, but leave time for more Q&A, interaction, and demonstrations
  - Slides posted on web site will include points skipped in class
  - Also makes it more important to read the textbook
  - If you like listening to lectures, multiple versions of lectures based on this same textbook are available on YouTube
- **Increased emphasis on Piazza for Q&A**
  - More powerful and easier to use than Canvas (or old Moodle) forums
  - Works best if students participate a lot too
- **Reduce number of major projects from 5 to 4**
  - Gives a little more time to work on each one
  - But, each one is also more important to your grade

Textbooks

- **Required: Randal E. Bryant and David R. O'Hallaron,**
  - "Computer Systems: A Programmer's Perspective, Third Edition" (CS:APP3e), Prentice Hall, 2016
  - http://csapp.cs.cmu.edu
  - Paper version recommended
    - Tests are open book, open notes, any paper, no electronics
  - Used quite heavily
    - How to solve assignments
    - Practice problems with similar style as exam problems
- **Supplemental: a book about C**
  - Labs, homework, and tests require reading and writing code in C
  - One free tutorial is recommended on the course site
  - Other tutorial/reference books can also substitute

Course Components

- **Lectures: Higher level concepts**
- **Lab sections**
  - Wednesdays in 1st floor of Keller. Try new ideas out in a supportive environment, graded only on attendance.
- **Projects (4)**
  - The heart of the course, fun but often time-consuming
  - About 2-3 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Written Problem Sets (5)**
  - Practice thinking and writing, similar to tests, on paper
- **Two midterms and a comprehensive final exam**
  - Test your understanding of concepts & mathematical principles

26

Electronic Resources

- **Class Web Page:**
  - **http://www-users.cs.umn.edu/classes/Spring-2020/csci2021/**
  - Complete schedule of lectures, exams, and assignments (coming)
  - Lecture slides, assignments, practice exams, solutions (coming)
  - Watch for announcements
- **Canvas Page**
  - Online turn-in of hands-on assignments
  - Grade information
- **Where to send electronic questions?**
  1. Piazza forum
  2. cs2021s20-010-staff@umn.edu (mailing list, for non-public Qs)
  3. Individual staff members have higher latency

27

Policies: Assignments and Exams

- **Groups? No.**
  - All homework assignments are individual work
- **Hand-in process**
  - Project assignments due online, by 11:55pm on a weekday evening
  - Problem sets due on paper, by start of class on Wednesdays
- **Conflicts**
  - There will be no makeup midterms
  - One excused missed midterm will be replaced by more weight on final
- **Appealing grades**
  - Within 7 days of completion of grading
    - Following procedure described in syllabus and Piazza
  - Note, we will regrade the whole assignment/exam

28

Facilities

- **Do labs using CSELabs Linux machines**
  - Accessible from on-campus labs, or remotely (VOLE, SSH)
  - Get an account if you don't have one already, login with UMN account name and password
- **Can I use my own machine?**
  - Working on your own machines may sometimes be possible, but is not a priority for support by course staff
  - Grade based on how it runs on our machines, so at least be sure to test there
  - Ubuntu 18.04 Linux (maybe in a VM) will be closest to lab experience
  - For Mac users, install GCC instead of Clang wrapper

29

Timeliness

- **Late exercises and hands-on assignments**
  - Late period is 24 hours from due date, 85% credit
  - For written assignments after class, bring to instructor's office (4-225E Keller)
  - No credit after 24 hours
- **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

30

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/text from previous course or from elsewhere on WWW
- **What is NOT cheating?**
  - Explaining how to use systems or tools
  - Helping others with high-level design issues
  - Getting ideas from public books or web sites, if you give credit
- **Penalty for cheating:**
  - Minimum: 0 grade on assignment or exam, report to campus OCS
- **Detection of cheating:**
  - We check with both human and automated efforts
  - Avoid surprises that would be unpleasant for all of us

31

E.g.: what if you find an answer online?

- **"When I was feeling stumped on a problem set question, I did some related web searches and accidentally discovered that it had been answered on StackOverflow"**
  - (Note: not posted by a 2021 student or in response to a 2021 student question)
- **Don't:**
  - Copy the answer from StackOverflow verbatim
  - Reword the StackOverflow answer without acknowledgment
- **Acceptable:**
  - Write your own answer to the question, based on what you learned on StackOverflow, and credit the web resource
- **Ethically preferable:**
  - Tell the staff or post on Piazza about the source

Policies: Grading

- **Exams (60%): weighted 15%, 15%, 30% (final)**
- **Projects (20%)**
- **Written Problem Sets (15%)**
- **Attending at least 11 out of 14 lab sections (5%)**

- **Guaranteed:**
  - ≥ 85%: at least A-
  - ≥ 72%: at least B-
  - ≥ 60%: at least C-

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

Exams schedule

- **A schedule of readings, lecture topics, assignments, and exams is now available on the course web site**
- **Put these exams in your calendar:**
  - Midterm 1: Monday February 24th, in class
  - Midterm 2: Friday, April 10th, in class
  - Final exam: Wednesday, May 13th, 8:00-10:00am

C Language Basics

- **Topics**
  - Variables and operations, control flow and functions, data structures
  - Differences from Java and high-level C++
  - Just enough to get you started: various topics return in more depth later
- **Assignments**
  - Proj1: Write a modest 19x3-style program, but in pure C

Data Representation

- **Topics**
  - Bit-level operations
  - Machine-level integers and floating-point
  - C operators and things that can go wrong
- **Assignments**
  - Proj2 (formerly "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?
  - How dynamic memory allocation works

- **Assignments**
  - Proj3 (formerly "Bomb lab"): Defusing a binary bomb with a debugger
  - Proj4 (formerly "Malloc lab"): Implement your own memory allocator

CPU Architecture

- **Topics**
  - The parts of a CPU and how they work together
  - How CPUs save time by doing multiple things at once (pipelining)
- **Lab activities**
  - Work with a CPU simulator
  - Implement your own instruction

The Memory Hierarchy

- **Topics**
  - Memory technology, memory hierarchy, caches, disks, locality
  - How virtual memory works
- **Lab activities**
  - Simulate and optimize cache behavior

Shorter Topics

- **Optimization**
  - Some code features that are good or bad for performance
  - Profiling code to know what parts are slow
- **Linking**
  - How compilers put code and data together into a final program
  - How code from libraries can be loaded as a program runs

Welcome
and Enjoy!