Course Overview
Computer Science 104: Machine Organization and Programming

Instructor:
Alvin R. Lebeck

Slides based on those from Randy Bryant and Dave O’Hallaron

Overview
- Administrivia
- Machine/system overview
- Course theme
- Five realities
Staff Information

- Instructor: Alvin R. Lebeck
  - Office: D308 LSRC
  - Email: alvy@cs.duke.edu
  - Office Hours: Mon 1:30-2:30, Thurs 11:00-noon, or by appointment

- TA: Alex Dutu
  - Office:
  - Email: alexdutu@cs.duke.edu
  - Office Hours: TBD, or by appointment
  - Will run recitations and answer homework questions

- UTAs:
  - More info later

Information

- I AM NOT PERFECT
  - Ask Questions!!

- Course Web Page
  http://www.cs.duke.edu/courses/cps104/fall11/
  - Lecture slides available on web before or shortly after class
  - See schedule link for readings also

- Blackboard (http://courses.duke.edu)
  - Grades
  - Discussion forum (post questions, etc. there)

- You are required to monitor course web page & blackboard
  - Homework will appear on web page
  - If necessary, additional information about homework on forum
  - You must post questions about homework on forum (not email to me or TA). We will respond quickly on forum (others can respond).
Textbooks

- Randal E. Bryant and David R. O'Hallaron,
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- Brian Kernighan and Dennis Ritchie,
  - This is recommended, not required, but a great reference to have.

Course Components

- Lectures
  - Higher level concepts

- Recitations
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage

- Homework (7-8): Programming + written problems
  - The heart of the course
  - 1-2 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement

- Exams (2 midterms + final)
  - Test your understanding of concepts & principles
Grading

- **Grade breakdown**
  - Midterm Exams: 30% (15% each)
  - Final Exam: 25%
  - Homework/labs: 45%
    - Combination of written and programming
    - 10 point scale for final grade (>=90 A- to A++; >=80 Bs, etc.) can slide to lower cutoff; not up to higher cutoff.

- **Late homework policy**
  - 10 point reduction for each day late
  - No credit after the homework is graded and handed back.
  - Feedback => return results quickly => grade almost immediately => late homework is a hassle

- **This course takes time, start assignments early.**
  - Average 3-5 hrs/week from previous course evaluations. New labs may be longer.

---

Course Problems

- **Academic Conduct**
  - Duke Community Standard
  - Studying together in groups is encouraged
  - All written/programming work must be your own, unless otherwise stated. Programs that are substantially the same as others will receive a grade of 0

- **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 Internet
    - 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
Course Problems (Continued)

- Can’t make midterms / final, other conflicts
  - Tell us early and we will schedule alternate time
- If you are having problems
  - See me
  - See DUS
  - See Academic Dean (very good resource)

Theme: Abstraction Is Good But Don’t Forget Reality

- Most CS courses emphasize abstraction
  - Abstract data types
  - Asymptotic analysis (e.g., O(n))
  - Helps manage complexity (1B transistors, the internet, etc.)
- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
  - Prof. Astrachan “from the abstract to the ridiculous”
- 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
    - Operating Systems, Networks, Databases, Computer Architecture, Embedded Systems, Compilers, etc.
The Big Picture

- What is inside a computer?
- How does it execute a program?

System Organization

The diagram shows a computer system with various components:

- Processor
- Cache
- Memory Bus
- Core Chip Set
- I/O Bridge
- I/O Bus
- Main Memory
- Disk Controller
- Graphics Controller
- Network Interface
- Network
- Graphics
- Disk
- Disk
- Disk
How does a Java program execute?

- Compile Java Source to Java Byte codes
- Java Virtual Machine (JVM) interprets/ translates Byte codes
- JVM is a program executing on the hardware...
- Java has lots of things that make it easier to program without making mistakes
- JVM handles memory for you
  - What do you do when you remove an entry from a hash table, tree, etc.?

The C Programming Language

- No virtual machine
- Compile source file directly to machine (this is a little simplified).
- Closer to hardware -> easier to make mistakes
  - E.g., Programmer must manage memory
Levels of Representation (Abstraction Good!)

High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Control Signal Specification

- temp = v[k];
- v[k] = v[k+1];
- v[k+1] = temp;

movl 0(%eax), %ebx
movl 4(%eax), %edx
movl %edx, 0(%eax)
movl %ebx, 4(%eax)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

Transistors turning on and off

Great Reality #1: Ints != 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 Int's: Yes!
  - Float's:
    - \((1e20 + -1e20) + 3.14\) -> 3.14
    - \(1e20 + (-1e20 + 3.14)\) -> ??
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
- Understanding 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 language targeted/used.

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
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

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:

<table>
<thead>
<tr>
<th>Saved State</th>
<th>Location accessed by fun(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>d7 ... d4</td>
<td></td>
</tr>
<tr>
<td>d3 ... d0</td>
<td></td>
</tr>
<tr>
<td>a[1]</td>
<td></td>
</tr>
<tr>
<td>a[0]</td>
<td></td>
</tr>
</tbody>
</table>

Compsci 104
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)

Memory System Performance Example

```c
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];
}
```

```c
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 you step through multi-dimensional array
**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

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (double precision)

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

MMM Plot: Analysis

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz

- 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
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

Course Perspective

- Most Systems Courses are Builder-Centric
  - Computer Architecture
    - Design pipelined processor in Verilog/VHDL
  - Operating Systems
    - Implement large portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols
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

Programs and Data

- Topics
  - Bits operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

- Assignments
  - L1 (datalab): C Programming, memory, & manipulating bits
  - L2 (bomblab): Defusing a binary bomb
  - L3 (buflab): Hacking a buffer bug (security vulnerability)
Computer Organization

- Topics
  - Instruction Set Architectures
  - Logic Design
  - Processor (CPU) Implementation

- Assignments
  - Logic design:
    - L4 (archlab): Assembly Programming, Modify/Implement Processor

The Memory Hierarchy

- Topics
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- Assignments
  - L5 (cachelab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.
Performance (if time)

- Topics
  - Co-optimization (control and data), measuring time on a computer
  - Includes aspects of architecture, compilers, and OS

Exceptional Control Flow

- Topics
  - Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
  - User level vs. Kernel
  - Includes aspects of compilers, OS, and architecture

- Assignments
  - L6 (exceptionlab): write exception handlers
    - A first introduction to concurrency
Virtual Memory

- Topics
  - Virtual memory, address translation, dynamic storage allocation
  - Includes aspects of architecture and OS

- Assignments
  - L7 (malloclab): Writing your own malloc package
    - Get a real feel for systems programming

Concurrency

- Topics
  - Multicore processors, synchronization, atomic primitives
  - Includes aspects of architecture and OS

- Assignments
  - L8 (synclab): Writing a multithreaded program
    - Get a real feel for challenges of concurrent programming
Representations (Group task)

- Form partners
- Using only the three symbols @ # $ specify a representation for the following:
  - All integers from 0 to 10
  - Commands to 1) walk, 2) turn, 3) sit, 4) raise right arm, 5) raise left arm

- Using only your representation write down series of commands & integers (if appropriate, e.g., raise left arm-3, turn-2)
  - Must have at least 5 commands

What You Know Today

```
C
...
int result;
double score;

double curve(double score) {
    return(score * 0.22124);
}
int main()
{
    int x;
    ...
    result = x + result;
    printf("Score is %d\n", curve(80));
    ...
}
```

```
JAVA
...
System.out.println("Please Enter In Your First Name: ");
String firstName = bufRead.readLine();
System.out.println("Please Enter In The Year You Were Born: ");
String bornYear = bufRead.readLine();
System.out.println("Please Enter In The Current Year: ");
String thisYear = bufRead.readLine();
int bYear = Integer.parseInt(bornYear);
int tYear = Integer.parseInt(thisYear);
int age = tYear - bYear;
System.out.println("Hello " + firstName + ". You are " + age + " years old");
```
High Level to Assembly

High Level Language (C, C++, Fortran, Java, etc.)
- Statements
- Variables
- Operators
- Methods, functions, procedures

Assembly Language
- Instructions
- Registers
- Memory

Welcome and Enjoy!