Introduction to Computer Systems

Seth Goldstein & Andreas Nowatzyk
January 13, 2004

Topics:
- Theme
- Five great realities of computer systems
- How this fits within CS curriculum
- Staff, text, and policies
- Lecture topics and assignments
- Lab rationale

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:
    - $40000 \times 40000 \rightarrow 1600000000$
    - $50000 \times 50000 \rightarrow -1794967296$

- Is $(x + y) + z = x + (y + z)$?
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
    - $1e20 + (-1e20 + 3.14) \rightarrow 0$

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
  1. High-level language model breaks down
- Tuning program performance
  1. Understanding sources of program inefficiency
- Implementing system software
  1. Compiler has machine code as target
  1. Operating systems must manage process state

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 required by procedure
  1. In units of clock cycles
  
  ```
  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

```c
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");
}
```

/* 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;
}
Measuring Time

Trickier than it Might Look

Many sources of variation

Example

n

Sum integers from 1 to n

<table>
<thead>
<tr>
<th>n</th>
<th>Cycles</th>
<th>Cycles/n</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>961</td>
<td>9.61</td>
</tr>
<tr>
<td>1,000</td>
<td>8,407</td>
<td>8.41</td>
</tr>
<tr>
<td>1,000</td>
<td>8,426</td>
<td>8.43</td>
</tr>
<tr>
<td>10,000</td>
<td>82,861</td>
<td>8.29</td>
</tr>
<tr>
<td>10,000</td>
<td>82,876</td>
<td>8.29</td>
</tr>
<tr>
<td>100,000</td>
<td>8,419,907</td>
<td>8.42</td>
</tr>
<tr>
<td>1,000,000</td>
<td>8,425,181</td>
<td>8.43</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>8,371,2305,591</td>
<td>8.37</td>
</tr>
</tbody>
</table>

Timing System Performance

int count(int n)
{
    ...
    for (i=0; i<n; i++) {
        start_counter();
        count(n);
        times[i] = get_counter();
    }
    ...
}

int count(int n)
{
    int i;
    int sum = 0;
    for (i=0; i<n; i++) {
        sum += i;
    }
    return sum;
}

main(int argc, char** argv)
{
    ...
    for (i=0; i<t; i++) {
        start_counter();
        count(n);
        times[i] = get_counter();
    }
    ...
}

Timing System Performance

<table>
<thead>
<tr>
<th>Experiment</th>
<th>n</th>
<th>cycles</th>
<th>cycles/n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1649.2</td>
<td>165.9</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>17.2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>24.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>6.1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1657.6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>1a</td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2a</td>
<td>10</td>
<td>16.4</td>
<td></td>
</tr>
<tr>
<td>3a</td>
<td>1000</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>4a</td>
<td>1000</td>
<td>1.6</td>
<td></td>
</tr>
</tbody>
</table>
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;
    }
}

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

Matmul Performance (Alpha 21164)

Memory System

Blocked matmul perf (Alpha 21164)
Real Memory Performance

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);
}

```
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  

-5.30498947741318e-315 3.1399998664856 3.14
-0.3.14 3.14 3.14

(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

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

Role within Curriculum

Transition from Abstract to Concrete!
  - From: high-level language model
  - To: underlying implementation

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 knowing more about the underlying system, leads one to be a more effective 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

Teaching staff

- Instructors
  - Prof. Seth Goldstein (Wed 1–2pm, WeH 7122)
  - Prof. Andreas Nowatzyk (Tue 3–4pm, NSH 4117)
- TA’s
  - Ningning Hu (A, Tue 5–6pm, WeH 8205)
  - Carolyn Au (B, Wed 3–4pm, WeH 3108)
  - David Charlton (C, Fri 11:30–12:30pm, WeH 3108)
  - David Fields (D, Wed 12:30am–1:30pm, WeH 3108)
  - Mike Nollen (E, Thu 3–4pm, WeH 3108)
- Course Admin
  - Norene Mears (WeH 7114)

These are the nominal office hours. Come talk to us anytime!
(Or phone or send email)

Textbooks

Randal E. Bryant and David R. O’Hallaron,
- http://csapp.cs.cmu.edu/

Samuel P. Harbison III and Guy L. Steele Jr.,
- http://careferencemanual.com/

Course Components

- Lectures
  - Higher level concepts
- Recitations
  - Applied concepts, important tools and skills for labs,
    clarification of lectures, exam coverage
- Labs
  - The heart of the course
  - 1, 2, or 3 weeks
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
Getting Help

Web
- www.cs.cmu.edu/~213
- Copies of lectures, assignments, exams, solutions
- Clarifications to assignments

Newsgroup
- cmu.cs.class.cs213
- Clarifications to assignments, general discussion

Personal help
- Professors: door open means come on in (no appt necessary)
- TAs: please mail or zephyr first.

Policies: Assignments

Work groups
- Labs: You must work alone on all labs

Handins
- Assignments due at 11:59pm on specified due date
- Typically 11:59pm Wednesday evening
- Electronic handins only
- Allowed a total of up to 5 late days for the semester

Makeup exams and assignments
- OK, but must make PRIOR arrangements with either Prof. Goldstein or Nowatzyk

Appealing grades
- Within 7 days of due date or exam date
- Assignments: Talk to the lead person on the assignment
- Exams: Talk to either Prof. Goldstein or Nowatzyk

Cheating

What is cheating?
- Sharing code: either by copying, retyping, looking at, or supplying a copy of a file.
- Using solutions or tools other than those from the course book, lectures, or staff.

What is NOT cheating?
- Helping others use systems or tools.
- Helping others with high-level design issues.
- Helping others debug their code.

Usual penalty for cheating:
- Removal from course with failing grade.
- Note in student’s permanent record

Policies: Grading

Exams (40%)
- Two in class exams (10% each)
- Final (20%)
- All exams are open book/open notes.

Labs (60%)
- 7 labs (8-12% each)

Grading Characteristics
- Lab scores tend to be high
  - Serious handicap if you don’t hand a lab in
- Tests typically have a wider range of scores
Facilities

Assignments will use Intel Computer Systems Cluster (aka “the fish machines”)
- 25 Pentium III Xeon servers donated by Intel for CS 213
- 550 MHz with 256 MB memory.
- Rack mounted in the 3rd floor Wean machine room.
- We’ll be setting up your accounts this week.

Getting help with the cluster machines:
- See course Web page for info
- Please direct questions to your TAs

Programs and Data

Topics
- Bits operations, arithmetic, assembly language programs, representation of C control and data structures
- Includes aspects of architecture and compilers
- Learning the tools

Assignments
- L1 Available THUR! (Due 1/25 11:59pm)
  - L1: Manipulating bits
  - L2: Defusing a binary bomb
  - L3: Hacking a buffer bomb

Performance

Topics
- High level processor models, code optimization (control and data), measuring time on a computer
- Includes aspects of architecture, compilers, and OS

Assignments
- L4: Optimizing Code Performance

The Memory Hierarchy

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

Assignments
- L4: Optimizing Code Performance
**Linking and Exceptional Control Flow**

**Topics**
- Object files, static and dynamic linking, libraries, loading
- Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
- Includes aspects of compilers, OS, and architecture

**Assignments**
- L5: Writing your own shell with job control

---

**Virtual memory**

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

**Assignments**
- L6: Writing your own malloc package

---

**I/O, Networking, and Concurrency**

**Topics**
- High level and low-level I/O, network programming, Internet services, Web servers
- Concurrency, concurrent server design, threads, I/O multiplexing with select.
- Includes aspects of networking, OS, and architecture.

**Assignments**
- L7: Writing your own Web proxy

---

**Lab Rationale**

Each lab should have a well-defined goal such as solving a puzzle or winning a contest.
- Defusing a binary bomb.
- Winning a performance contest.

Doing a lab should result in new skills and concepts
- Data Lab: computer arithmetic, digital logic.
- Bomb Labs: assembly language, using a debugger, understanding the stack
- Perf Lab: profiling, measurement, performance debugging.
- Shell Lab: understanding Unix process control and signals
- Malloc Lab: understanding pointers and nasty memory bugs.
- Proxy Lab: network programming, server design

We try to use competition in a fun and healthy way.
- Set a threshold for full credit.
- Post intermediate results (anonymized) on Web page for glory!
Autolab Web Service

Labs are provided by the Autolab system
- Developed in summer 2003 by Dave O'Hallaron
- Apache Web server + Perl CGI programs
- Beta tested in Fall 2003, so of course, bug free now

With Autolab you can use your Web browser to:
- Review lab notes
- Download the lab materials
- Stream autoresults to a class status Web page as you work.
- Upload (handin) your code for autograding by the Autolab server.
- View the complete history of your code handins, autoresult submissions, autograding reports, and instructor evaluations.
- View the class status page

Acknowledgement

15-213 was developed and fine-tuned by Randal E. Bryant and David O’Hallaron. They wrote The Book!

Have a Great Semester!