Systems I Introduction to Computer Systems

Don Fussell
Spring 2011
Topics:

- Theme
- Five great realities of computer systems
- How this fits within CS curriculum



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
 - Compilers, Operating Systems, Networks, Computer Architecture, etc.



Great Reality #1

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

Examples

```
Is x^2 \ge 0?

Float's: Yes!

Int's:

40000 * 40000 --> 16000000000

50000 * 50000 --> ??

Is (x + y) + z = x + (y + z)?

Unsigned & Signed Int's: Yes!

Float's:

(1e20 + -1e20) + 3.14 --> 3.14
1e20 + (-1e20 + 3.14) --> ??
```



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



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
 - 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



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
 - 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



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;
```



- 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



Memory Matters

- 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

```
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
-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



Memory Performance Example

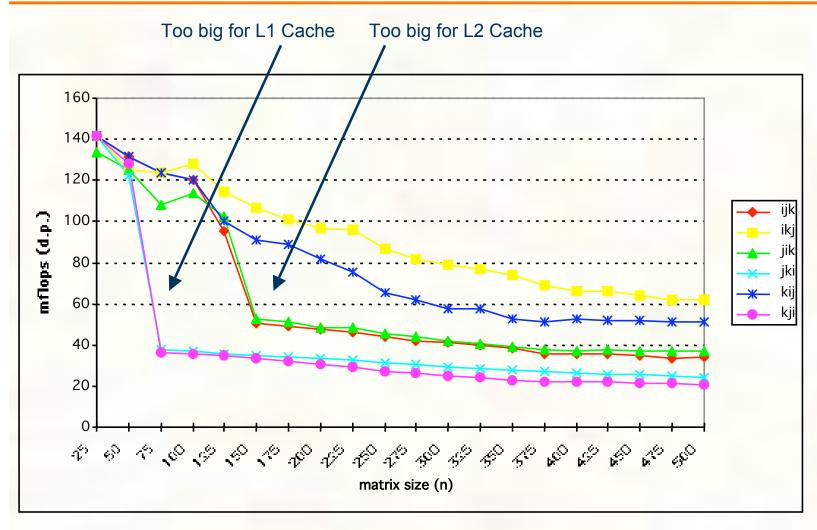
- Implementations of Matrix Multiplication
 - Multiple ways to nest loops

```
/* 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;
}
</pre>
```

```
/* 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
  }
}</pre>
```

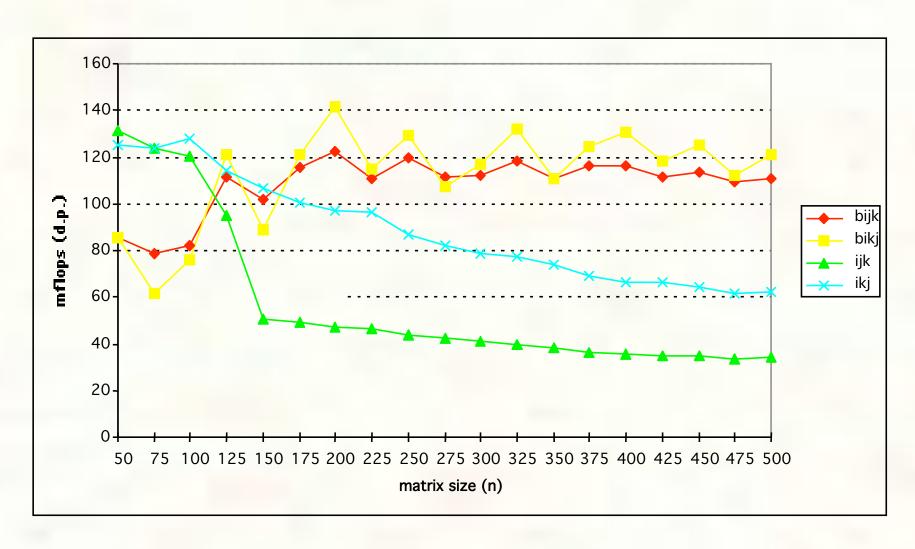


Matmult Performance (Alpha 21164)





Blocked matmult perf (Alpha 21164)





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



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
 - 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 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



Teaching staff

- Instructor Don Fussell
 - Office: ACES 2.120
 - Office Hours: MW 11-12
 - Email: fussell@cs.utexas.edu
 - http://www.cs.utexas.edu/~fussell/
- TA Christian Miller
 - Office: TBD
 - Office Hours: TBD
 - Email: ckm@cs.utexas.edu
 - http://www.cs.utexas.edu/~ckm

Required

- Randal E. Bryant and David R. O'Hallaron,
 - "Computer Systems: A Programmer's Perspective", Prentice Hall 2003.
 - http://csapp.cs.cmu.edu/

Optional

- Brian Kernighan and Dennis Ritchie,
 - "The C Programming Language, Second Edition", Prentice Hall, 1988



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 or 2 weeks
 - Provide in-depth understanding of an aspect of systems
 - Programming and measurement



Getting Help

- Web
 - http://www.cs.utexas.edu/~fussell/courses/cs429h/
 - Copies of lectures, assignments, etc.
 - Clarifications to assignments
- Newsgroup
 - TBD
- Personal help
 - Office hours or by appointment with either instructor or TA



Policies: Assignments

- Work groups
 - You must work alone on all labs
- Handins
 - Assignments due at 11:59pm on specified due date.
 - Electronic handins only.
- Makeup exams and assignments
 - Not normally done, except by prior arrangement with instructor.
- Appealing grades
 - Within 7 days of due date or exam date.
 - Assignments: Talk to the TA
 - Exams: Talk to instructor.



- What is cheating?
 - Sharing code: either by copying, retyping, looking at, or supplying a copy of a file.
- What is NOT cheating?
 - Helping others use systems or tools.
 - Helping others with high-level design issues.
 - Helping others debug their code.
- Penalty for cheating:
 - Removal from course with failing grade.



Policies: Grading

- Exams (50%)
 - Three in class exams (10% each)
 - Final (20%)
 - All exams are open book/open notes.
- Labs (50%)
 - 7 labs (7-8% each)
- Grading Characteristics
 - Lab scores tend to be high
 - Serious handicap if you don't hand a lab in
 - We offer generous redemption programs
 - Tests typically have a wider range of scores



- Assignments will use the erstwhile Taylor Ubuntu lab, now in Trailer Hall
 - You will need a CS account, if you don't have one, see the UTCS webpage for a form and the procedure to apply for a class account.
- Getting help with the cluster machines:
 - See course Web page for info
 - Please direct questions to your TAs



Course Topics

Topics

- Data representation
- Hardware building blocks
- From application programs to machine-level programs
- Processor design
- Pipelining principles
- Memory hierarchies
- Performance programming

Assignments to include

- Learning to program in C (multiple)
- Lab: Manipulating bits
- Lab: Defusing a binary bomb
- Lab: Hacking a buffer bomb
- Lab: Program optimization



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.
 - Etc.



Good Luck!