15-213
"The Class That Gives CMU Its Zip!"

Introduction to Computer Systems

Guy Blelloch
January 16, 2001

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

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
  - Compilers, Operating Systems, Networks, Computer Architecture

Great Reality #1

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

Examples
- Is $x^2 \geq 0$?
  - Float's: Yes!
  - Int's:
    - » `65535 * 65535 --> -131071` (On most machines)
    - » `65535L * 65535 --> 4292836225` (On Alpha)
- Is (x + y) + z = x + (y + z)?
  - Unsigned Int's: Yes!
  - Float's:
    - » `(1e10 + -1e10) + 3.14 --> 3.14`
    - » `1e10 + (-1e10 + 3.14) --> 0.0`

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 (usually)
  - 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 at this 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

Great Reality #3

Memory Matters

Memory is not unbounded
- It must be allocated and managed
- Many applications are memory dominated
- The memory system can be the largest portion of a machine's cost

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	Sun
-g	5.30498947741318e-315	3.1399998664856	3.14
-O	3.14	3.14	3.14

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 system and compiler dependent
- Action at a distance
  - Corrupted object logically unrelated to one being accessed
  - Effect of bug may occur long after it occurs

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
  - E.g., Purify

Memory Performance Example

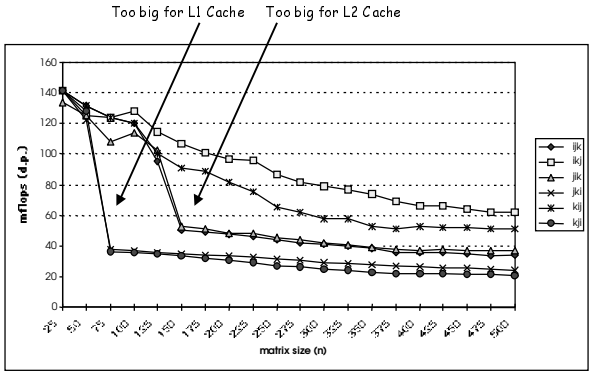
Implementations of Matrix Multiplication

- Multiple ways to nest loops

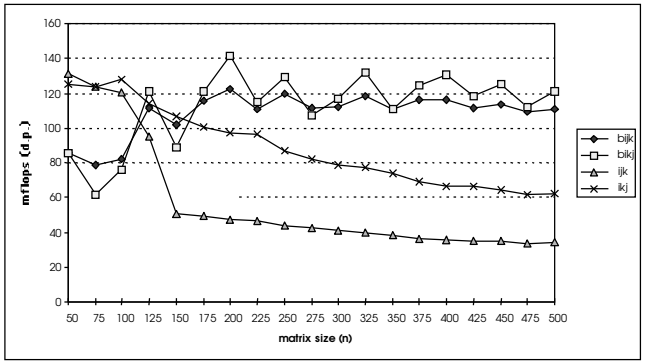
```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

```
/* 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
  }
}
```


Matmult Performance (Alpha 21164)



Blocked matmult perf (Alpha 21164)



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

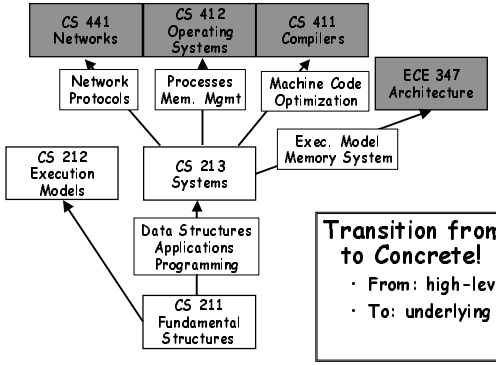
They need to get data in and out

- 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