

15-213

“The Class That Gives CMU Its Zip!”

Introduction to Computer Systems

**Frank Pfenning
(presented by David O'Hallaron)
January 11, 2005**

Topics:

- **Theme and objective**
- **Five great realities of computer systems**
- **How this fits within CS curriculum**
- **Course mechanics and overview**

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:

 - » $40000 * 40000 \rightarrow 1600000000$

 - » $50000 * 50000 \rightarrow ??$

■ Is $(x + y) + z = x + (y + z)$?

- Unsigned & Signed Int's: Yes!

- Float's:

 - » $(1e20 + -1e20) + 3.14 \rightarrow 3.14$

 - » $1e20 + (-1e20 + 3.14) \rightarrow ??$

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
 - 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

```
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");
}
```

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;
}
```

Measuring Time

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

Great Reality #3

Memory Matters: Random Access Memory is an un-physical 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

```
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
-O	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 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];
}
```

59,393,288 clock cycles

```
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];
}
```

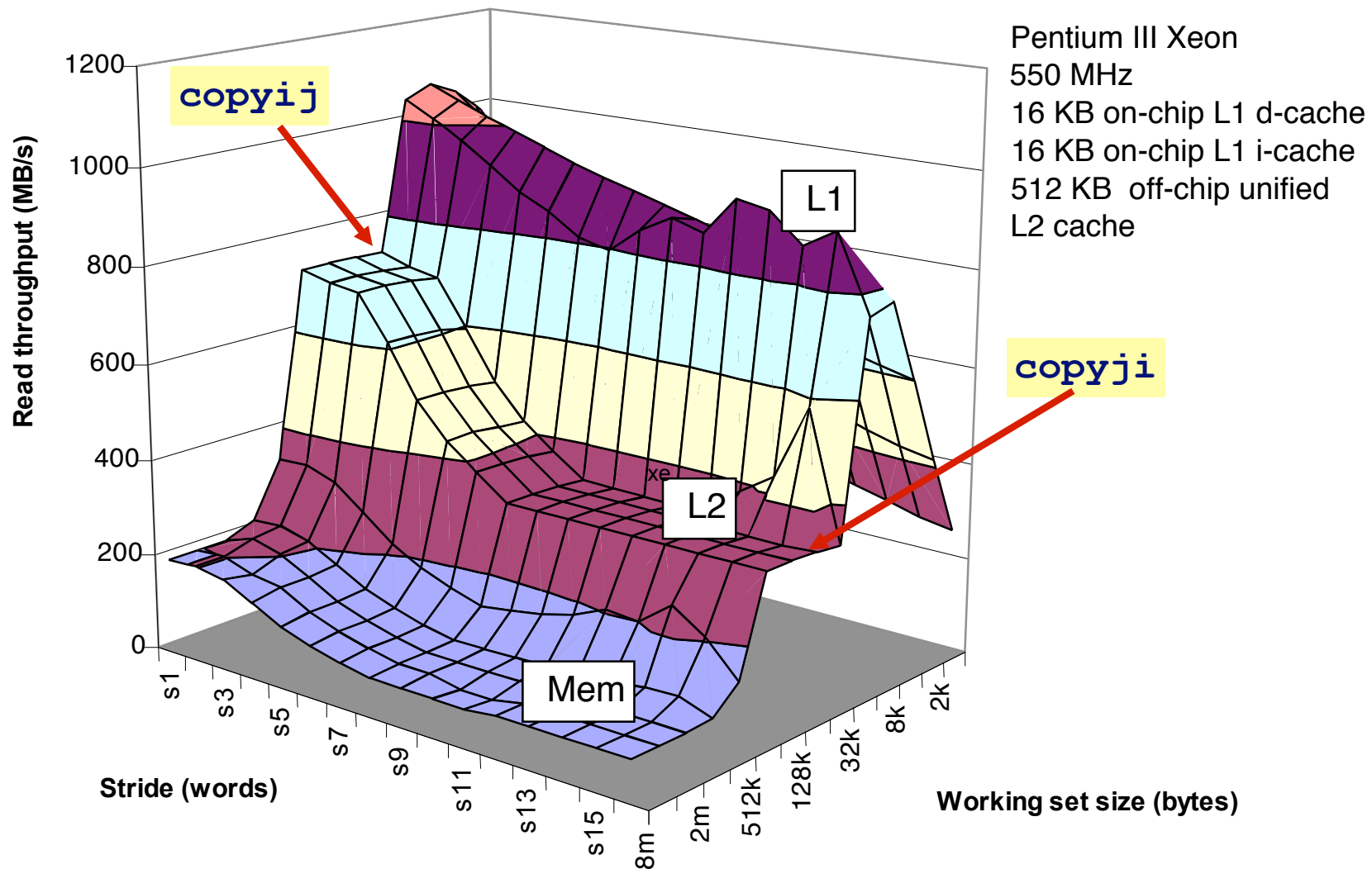
1,277,877,876 clock cycles

21.5 times slower!

(Measured on 2GHz
Intel Pentium 4)

- Hierarchical memory organization
- Performance depends on access patterns
 - Including how step through multi-dimensional array

The Memory Mountain



Memory Performance Example

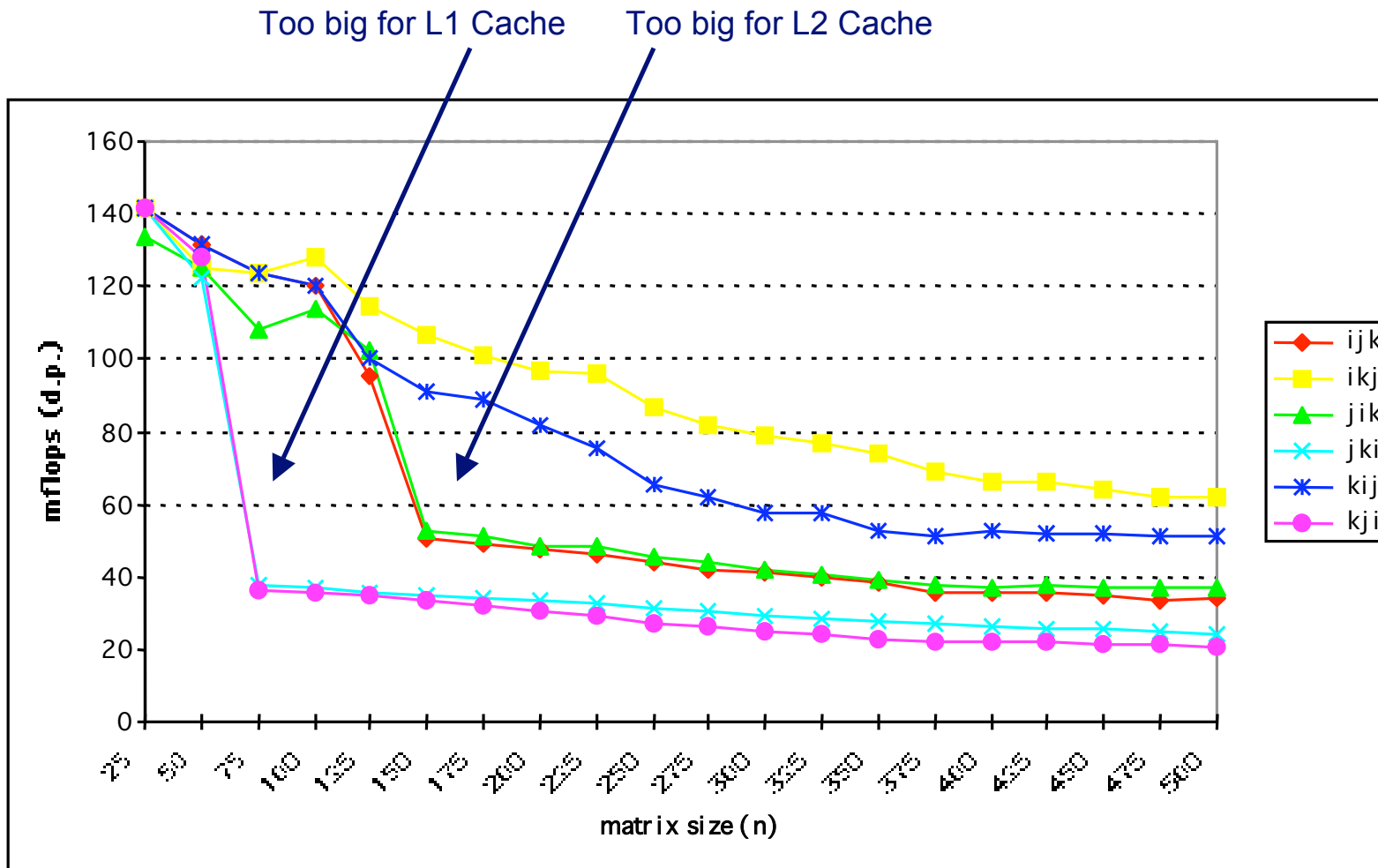
Implementations of Matrix Multiplication

- Multiple ways to nest loops

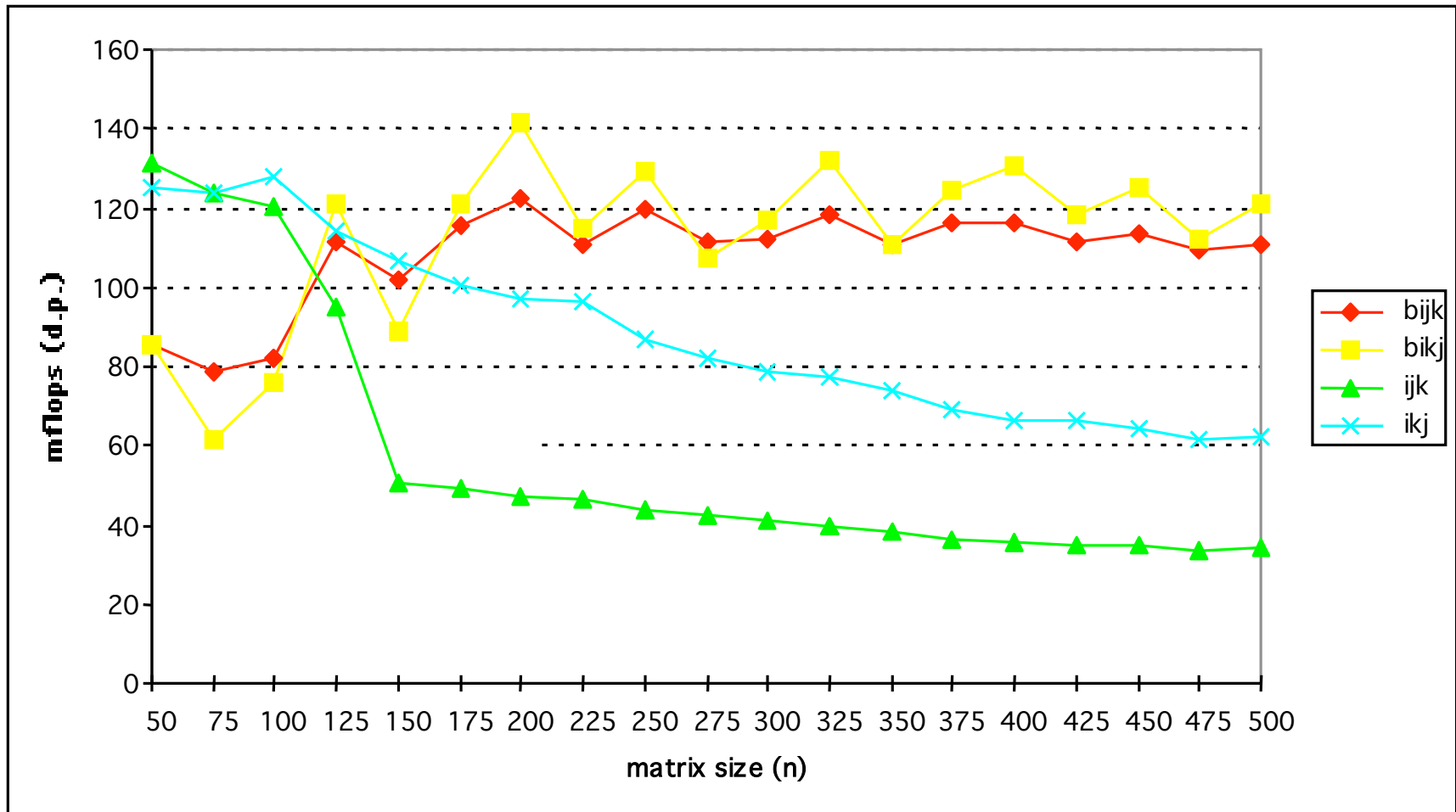
```
/* ijk */
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        sum = 0.0;
        for (k=0; k<n; k++)
            sum += a[i][k] * b[k][j];
        c[i][j] = sum;
    }
}
```

```
/* 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)

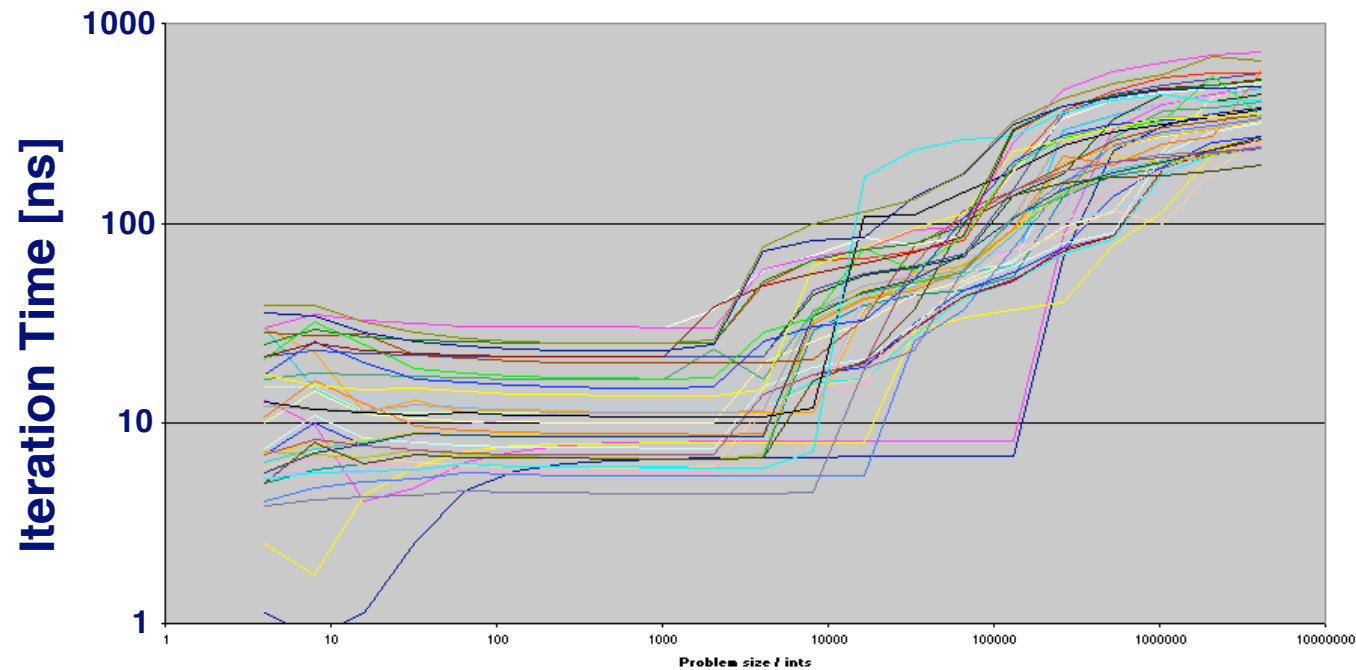
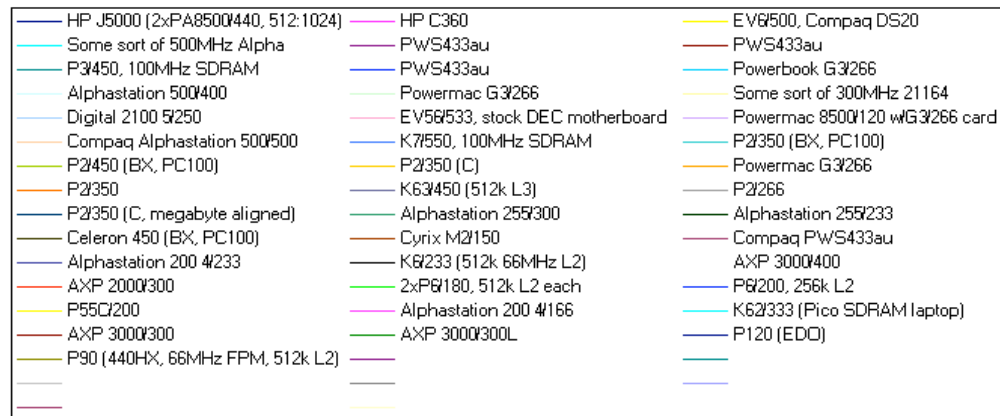


Blocked matmult perf (Alpha 21164)



Real Memory Performance

Pointer-Chase Results



From Tom Womack's
memory latency benchmark

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

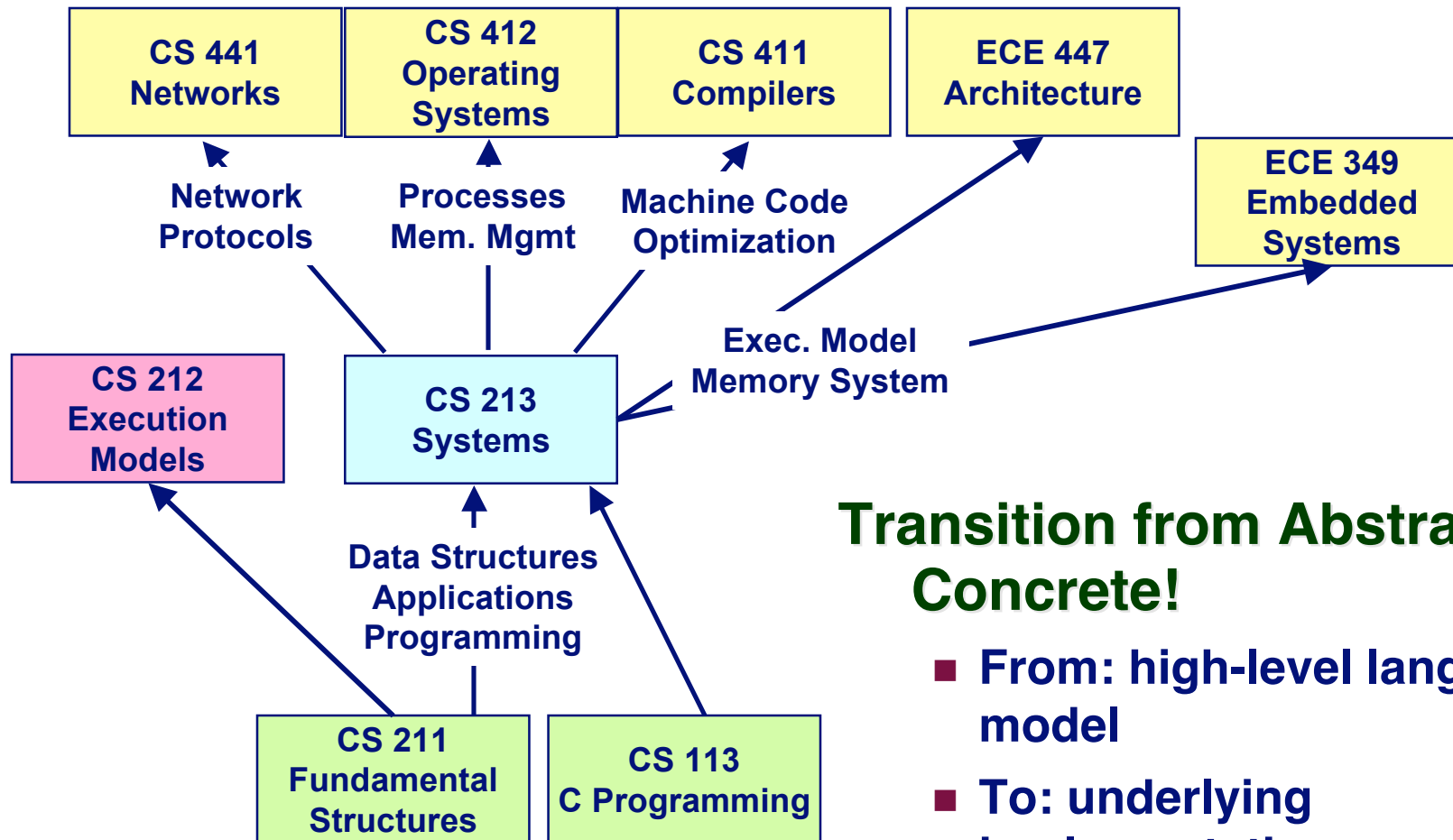
They need to get data in and out

- 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 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

■ Instructors

- Prof. Frank Pfenning (WeH 8117)

■ TA's (with Mon recitations in OSC 203)

- Kun Gao (10:30)
- Boris Sofman (11:30)
- Ben Rister (12:30)
- Kevin Litwack (1:30)
- Uman Kalaria (2:30)
- Naju Mancheril (3:30)

Come talk to us anytime!
(or phone or send email)

■ Course Admin

- Jenn Landefeld (WeH 8120)

Textbooks

Randal E. Bryant and David R. O'Hallaron,

- **“Computer Systems: A Programmer’s Perspective”,
Prentice Hall 2003.**
- **csapp.cs.cmu.edu**

Brian Kernighan and Dennis Ritchie,

- **“The C Programming Language, Second Edition”, Prentice
Hall, 1988**

Course Components

Lectures

- Higher level concepts and context

Recitations

- Applied concepts, important tools and skills for labs, clarification of lectures, review for exams
- Quizzes

Labs

- The heart of the course (600 pts out of 1000 for course)
- 1.5 or 2 weeks
- Provide in-depth understanding of an aspect of systems
- Programming and measurement

Quizzes and Exams

Quizzes

- 20 minutes to end recitation, except before or after exams
- 8 quizzes in total, 15 points each (drop lowest) = 100(+5) pts
- Help you keep up, practice skills for exams

Exams

- 2 exams, 75 points each = 150 pts
- In lecture, open book, open notes, closed computer

Final

- Cumulative, emphasizes last part of course, 150 pts
- Three hour, open book, open notes, closed computer

Getting Help

Web

- <http://www.cs.cmu.edu/~fp/courses/15213-s05/>
- Copies of lectures, assignments, quizzes, exams, solutions
- Clarifications to assignments
- Course schedule

Newsgroup

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

Personal help

- Frank Pfenning: WeH 8117
 - Use office hour (Tue 3:00-4:00) or stop by
- TAs
 - Email, office hour, phone, stop by (see web page)
 - One lead instructor for each lab

Policies: Assignments

Work groups

- You must work alone on all labs

Handins

- Assignments due at 11:59pm on specified due date
- Typically 11:59pm Thursday evening
- Electronic handins only (no exceptions!)
- 5 grace days to use throughout the term
- At most 2 late days for each lab!

Grading

- Autograding plus code review
 - Code must compile and run
 - Code must be readable and intelligible
- Grade only official handins

Cheating

What is cheating?

- Sharing code: either by copying, retyping, looking at, or supplying a copy of a file
- Coaching: helping your friend to write a lab, line by line

What is NOT cheating?

- Helping others use systems, compilers, or tools
- Helping others debug their code

Penalty for cheating:

- Removal from course with failing grade
- Official letter to dean's office at first offense

Detection of cheating:

- Will use MOSS cheating checker which speaks C!

Policies: Grading

Quizzes (10%)

- 8 quizzes at 15 points each, drop lowest

Exams (30%)

- Two in class exams (75 points each)
- Final (150 points)
- All exams are open book / open notes

Labs (60%)

- 7 labs (60-100 points each)

Grading Characteristics

- Lab scores tend to be high
 - Serious handicap if you don't hand a lab in
- Exams typically have a wider range of scores

Facilities

Assignments will use the Intel Computer Systems Cluster (aka “the fish machines”)

- 25 (21) Pentium III Xeon servers donated by Intel for CS 213
- 550 MHz with 256 MB memory.
- Rack mounted in the 3rd floor Wean Hall machine room.
- Your accounts are ready

Getting help with the cluster machines:

- See course Web page for info

Account Initialization

For using the Fish machines:

- Read description on the course web-page carefully
- Run checkin script to set-up Kerberos credentials
- Keep your code in your “213hw” directory on your *Andrew* account
- Do NOT modify anything in the 15-213 directory
- Use
`ssh -1 -l bovic@ANDREW.CMU.EDU xxxx.cmcl.cs.cmu.edu`

For using autolab:

- Give yourself a nickname
- Use a throwaway password
- Provide your preferred e-mail address

Programs and Data (8)

Topics

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

Assignments

- L1 (datalab): Manipulating bits
- L2 (bomblab): Defusing a binary bomb
- L3 (buflab): Hacking a buffer bomb

Performance (2)

Topics

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

Assignments

- L4 (perflab): Optimizing code performance

The Memory Hierarchy (2)

Topics

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

Assignments

- L4 (perflab): Optimizing code performance

Linking and Exceptional Control Flow (3)

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 (tshlab): Writing your own shell with job control

Virtual Memory (4)

Topics

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

Assignments

- L6 (malloclab): Writing your own malloc package

I/O, Networking, and Concurrency (6)

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 (proxylab): Writing your own Web proxy

Lab Rationale

Each lab should have a well-defined goal such as solving a puzzle or winning a contest

Doing a lab should result in new skills and concepts

- **Data Lab: number representations, logic, bit manipulation**
- **Bomb Lab: assembly, using debugger, understanding stack**
- **Buffer Lab: awareness of security issues**
- **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 reasonable 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 Fall 2003, very stable by now

With Autolab you can use your Web browser to:

- Review lab notes, clarifications
- 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

Good Luck and Have Fun!