P573 Computer Science

Randall Bramley

1104 Luddy

8:00 – 9:15 AM, Monday & Wednesday

P573 Overview

- Today:
  - course goals and mechanics
  - assumed knowledge, abilities: prerequisites
  - conventions in coding, math, and notation
  - the fundamental principle underlying p573
  - … and the analytic tool that it provides

Layer Cake of Scientific Computing

- Scientific computing includes
  - domain science: physics/chem/geo describes phenomena
  - math models: PDEs, network graphs, …
  - numerical methods: solver methods, error analysis
  - computer implementation of algorithms: languages, hardware
  - performance analysis: is an implementation unnecessarily slow?
  - data analysis: visualization, plotting, user presentation
  - validation and verification

Layer Cake of Scientific Computing

- Scientific computing includes
  - domain science: physics/chem/geo describes phenomena
  - math models: PDEs, network graphs, …
  - numerical methods: solver methods, error analysis
  - computer implementation of algorithms: languages, hardware
  - performance analysis: is an implementation unnecessarily slow?
  - data analysis: visualization, plotting, user presentation
  - validation and verification
- What P573 concentrates on

P573 Goals

- Course goals oriented towards CS capabilities
  - performance modeling and analysis
  - mapping implementations to computer architectures
  - *practical* software tools and methods in scientific computing
- Building a software toolkit for exploring computer architecture and algorithms
- Analytic methodology for determining (non)existence of fast implementations for given problem
- Will not prove theorems, derive algorithms, prove convergence, or analyze “rounding errors”
- Will *use* such results … and experimentally evaluate their validity

Course Mechanics

P573 Mechanics

- Check the Web pages for the course: https://www.cs.indiana.edu/classes/p573
- Check the Canvas page often, and always before starting an assignment
- Grading:
  - assignments 50%
  - midterm 20%
  - final 30%
- Both midterm and final may consist entirely or in part of projects

P573 Mechanics

- Course material:
  - you are responsible for everything that is presented or covered in class lectures
  - some derivation and material will be on the whiteboard (so will need notes from class)
  - questions and answers in class are important
  - slides are not definitive
  - web pages may give more detail than in class
  - web pages will have timestamp history for corrections, updates, additions

P573 Mechanics

- Cheating: difficult to do but some insist on doing it anyway. Easy to avoid:
- Report in detail all help received
  - Found on web, asked another student, telepathy, coercion, ...
  - Need not report info exchanged via course Canvas pages
  - Need not report help received from instructor(s)
  - *Both* giver and receiver must report sharing/transfer of ideas and material
  - *All* documents handed in must have your name and the names of any collaborators
  - Citation avoids academic death penalty of plagiarism

P573 Mechanics: Assignments, Handins

- You will write, run, analyze programs for assignments
- Report the full environment necessary to reproduce your results:
  - compiler used, compiler version, flags (like -O3 -msse)
  - OS and its version number (*uname -a* gives this in Unix)
  - hostname of machine used, date/time when program run
  - machine hardware configuration: type of processor, speed of processor, amount of memory (summarize results from */proc/cpuinfo* andc */proc/meminfo*)
  - just build the boilerplate once for a given machine, put into a text file called *TestEnvironment*, then reuse/modify it for other assignments
  - example ...

Vector Ops Test Environment

Host: behemoth.cs.indiana.edu

OS : Linux 4.4.0-24 , 64-bit

CPU: 6-core, hyperthreaded

Model: Intel(R) Core(TM) i7-3960X CPU @ 3.30GHz

CPU max MHz: 5700.0000

CPU min MHz: 1200.0000

L1 data cache: 32K

L2 cache: 256K

L3 cache: 15360K

Memory: 32 Gbytes

Relevant CPU capabilities: mtrr, mmx, sse4_2, cpufreq

Compiler: Intel Fortran ifort, version 13.1.1 20130313

Compiler options: -O3 -opt-prefetch -align all -ccdefault none -ftz -funroll-loops -pad -falign-functions=16 -fp-model fast=2 -fp-speculation=fast -opt-prefetch -xHost

Job Start: Wed 22 Jul 2016, 5:31:42

Job End: Wed 22 Jul 2016, 9:08:12

Conventions for Assignments

- Reporting test environment
  - don't just dump the contents of /proc filesystem into a file and submit that. Summarize only the essentials
  - a 2000 line file about the test environment is good for checking and comparing results years later, but not to hand in for an assignment
  - rough rule: if you don't know what a item is, don't include it in the test environment file (just what does the *mtrr* CPU flag on the preceeding slide mean?)

Conventions for Assignments: Handins

- Create a single tar or zip file with source code(s) and a plain text file with any reportage or notes
  - do *not* include executable, object (.o) files, libraries (.a, .so), backup files, or hidden directories
  - submit it via Canvas
  - do *not* send email with multiple attachments – put it all in a single tar or zip file
  - for any files I provide, don't include them in the hand-in unless you changed them
  - use plain text files for reports. No PDF, Word, ODF, cuneiform, quipu, or other unnecessarily complex formats
- If you hand in multiple versions, only the latest one is “official”. Be sure it's complete and is not just an update for one or two files

Required Knowledge

https://www.cs.indiana.edu/classes/p573/prereq-check/requirements.html

P573 Required Knowledge: Coding

- Programs required to be coded in C or C++ or Fortran
  - You need know *one* of those, not all three
- How to open, read, and write to/from files
- Know how to code to a specified interface
- Using 1 and 2-D arrays
- Algorithms usually stated in pseudo-code; ask in class if they are not clear
- Will use makefiles, with multiple source files, link in external libraries, run codes on Unix-like systems
- The codes are tools; getting them running correctly is only the start
  - Don’t underestimate the time required
  - Generally, will take you 8-11x longer to code than me

P573 Required Knowledge: Numeracy

- Exponential notation: 1.8E12 = 1.8×10^{12}
- … and 1.8026175E12 furlongs/fortnight is …?
- von Neumann: max length of a lecture is 1 microcentury What's that in hours/minutes/weeks?
- How many significant digits does 0.00623 display?
- What's largest value of *n* that allows 3 *n x n* arrays of 8-byte doubles to be held in *G* gigabytes of memory?
- Is 7.3×10^{85} billion operations per second a reasonable computational speed? What about 7.3×10^{-85} billion operations per second?
- What values of *x* blowup *log(x), 1/x, acos(x)*?
  - assuming *x* and the functions are real-valued, and "blowup" means "barfs, overflows, or returns unexpected values"
- I use American "billion" = 10^9, not British 10^{12}

P573 Required Knowledge: Math

P573 Required Knowledge: Math

- Math knowledge: mostly from linear algebra
  - definition of a derivative
  - basic operations in linear algebra, e.g.
    - matrix product $A*B$ (and conformality of sizes required)
    - can write down a triple-nested loop to compute $C = A*B$
    - matrix times a vector
    - dotproduct (AKA inner product) of two vectors
- Definitions of (e.g.)
  - transpose A^T of a matrix A
  - orthogonal matrix Q
  - upper/lower triangular matrix (must that matrix be square?)
  - 2-norm of a vector
  - eigenvalue λ: $A*x = \lambda*x$, with one more stipulation … what?

Example linear algebra problem

- Solve a linear system of equations $A*x = b$
- $A, \ b$ are given
  - A is an $n \times n$ matrix with real-valued entries
  - b is an $n \times 1$ vector with real-valued entries
  - Want to solve for x, so x must be $n \times 1$
- The sizes of A, x, b work out to be correct:

$$
\begin{array}{ccccc}
(n \times n) & * & (n \times 1) & \rightarrow & (n \times 1) \\
A & * & x & \rightarrow & b
\end{array}
$$

Linear algebra gives a clear and useful area in which to apply load/store analysis, the real crux in P573

P573 Conventions

Linear algebra conventions

- Mostly will follow the "Householder convention"
  - scalars: lower case Greek letters like α, γ, σ
  - vectors: lower case Roman letters: x, y, u, v
  - matrices: upper case Roman letters: A, G, C
- Not Householder convention:
  - symmetric matrices: horizontally symmetric letters: *A, H, V*
  - dimensions: lower case Roman: *m, n: "A* is an $m \times n$ matrix*"*
  - indices: lower case letters *i-n,* like *"A(i,j) = 0 for i < j"*
  - block indices: upper case Roman letters: *I, J, K*
- Don't need to memorize this, just a helpful guide

Linear algebra conventions

- All vectors are column vectors by default. Use transpose notation to specify row vectors
- More generally: if A is m x n, then A^T is n x m and $A^T(i,j) = A(j,i)$
- The dotproduct of two vectors x and y is defined as

$$\alpha = \Sigma\, x(i)*y(i),$$

which can also be stated as $\alpha = x^T y$

- Matlab uses a single quote mark to denote transpose:

```
alpha = x'*y
```

and sometimes I'll use that notation

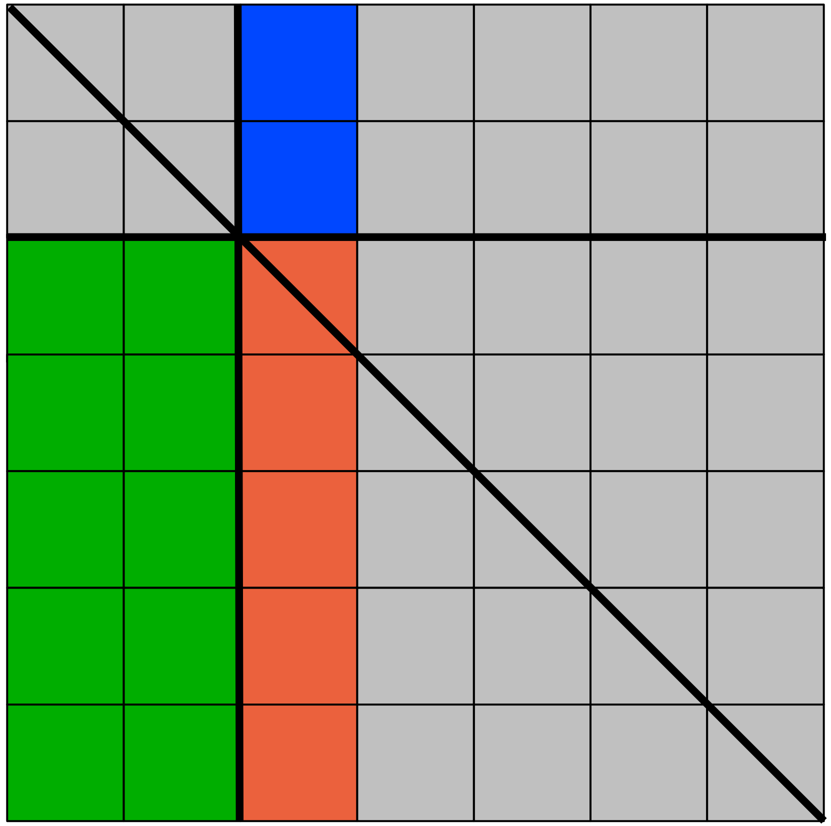
Linear algebra conventions

- A matrix is a mathematical entity; a 2D array is a computer data structure often used to hold a matrix
- Matrices can also be stored in a linked list, a graph node-adjacency list, a pair of 1d arrays, a 3d array, ...
- Matrices are indexed starting from 1; arrays can be indexed starting from 1, 0, -27, or any integer
  - C/C++ require array start index is 0
  - Matlab requires array start index is 1
  - Fortran allows any start index, from -2147483648 to 2147483647 (or from -9223372036854775808 to 9223372036854775807)
- Entries in a matrix are indicated via $A(i,j)$. Entries in an array might be denoted as $A[i][j]$, $A[i*n+j]$, $A(i,j)$

Linear algebra conventions

- Why picky matrix vs. array distinction?
  - storing an $n \times n$ diagonal matrix in an $n \times n$ array is dumb
- Gaussian elimination starts out with an $n \times n$ matrix A in a 2d array A, ends up with two triangular matrices L and U stored in the *same* array A
- At intermediate stages, the array A contains parts of L, U, and intermediate computational byproducts
- At intermediate stages, need to carry out operations on vectors and matrices that are stored in subarrays of A
- An example of this ...

A(3:7, 3) = A(3:7, 3) – A(3:7, 1:2) * A(1:2, 3)



- Cannot copy parts over to temporary arrays; leads to $O(n^3)$ memory copying/allocation/deallocation
- For n = 56k, difference between 14 *minutes* versus 1.5 *years* for the Gaussian elimination algorithm

Other P573 Conventions

- All floating point numbers will be double precision or 8-byte floats
- All doubles must be printed out to 17 significant digits
- A flop is a *floating point operation*. All of +, *, /, -, sqrt, log, exp, sin, max, abs,.... count as one flop
  - Yes, log(x) is probably slower than x + 1.0, but both still count as one floating point operation
- Integer operations are not flops (well, duh)
- Can use Mflop = 10^6 flops and Gflop = 10^9 flops, *or* Mflop = 2^{20} and Gflop = 2^{30}
- You can use either, just state the choice clearly
- Computational rates will be in units of Gflops/second (sometimes confusingly denoted in books as Gflops)

Fundamental Idea

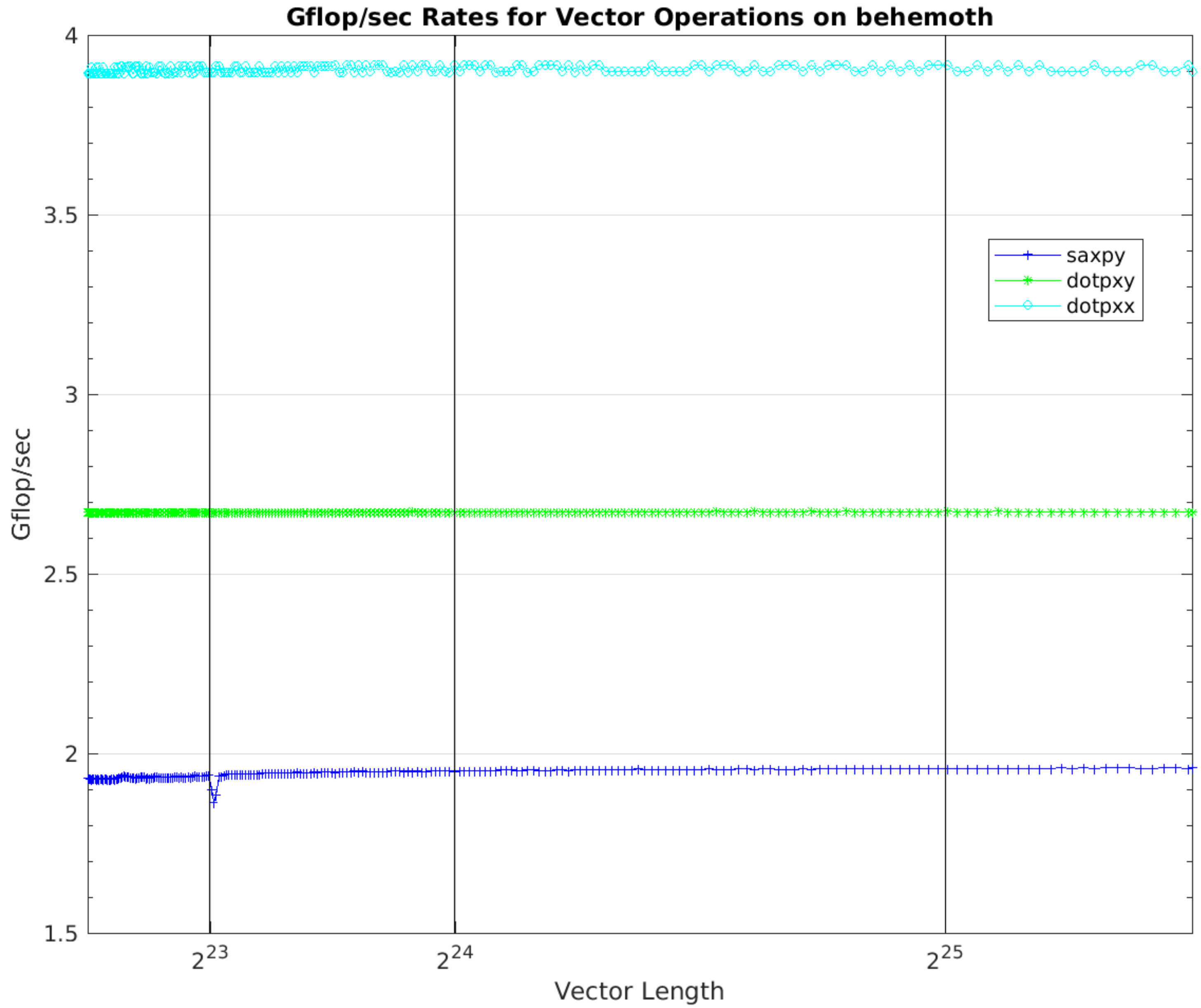
Other Fundamental Ideas

- *Computation* is not the bottleneck in scientific computing: instead, it is *data movement*
  - memory hierarchy and pipelining in architectures
  - data and metadata management in scientific computing
- Interplay between continuum and discrete space
  - use one regime to approximate the other
  - discretizations (time, space) to get finite size problems
  - reducing infinite dimensional problem to finite dimensions certainly reduces the amount of data involved
- Finite precision arithmetic effects
  - IEEE 754 standard and implications for accuracy, computability, reliability, reproducibility, debugging
  - Finite precision reduces amount of data involved compared to unlimited precision computation

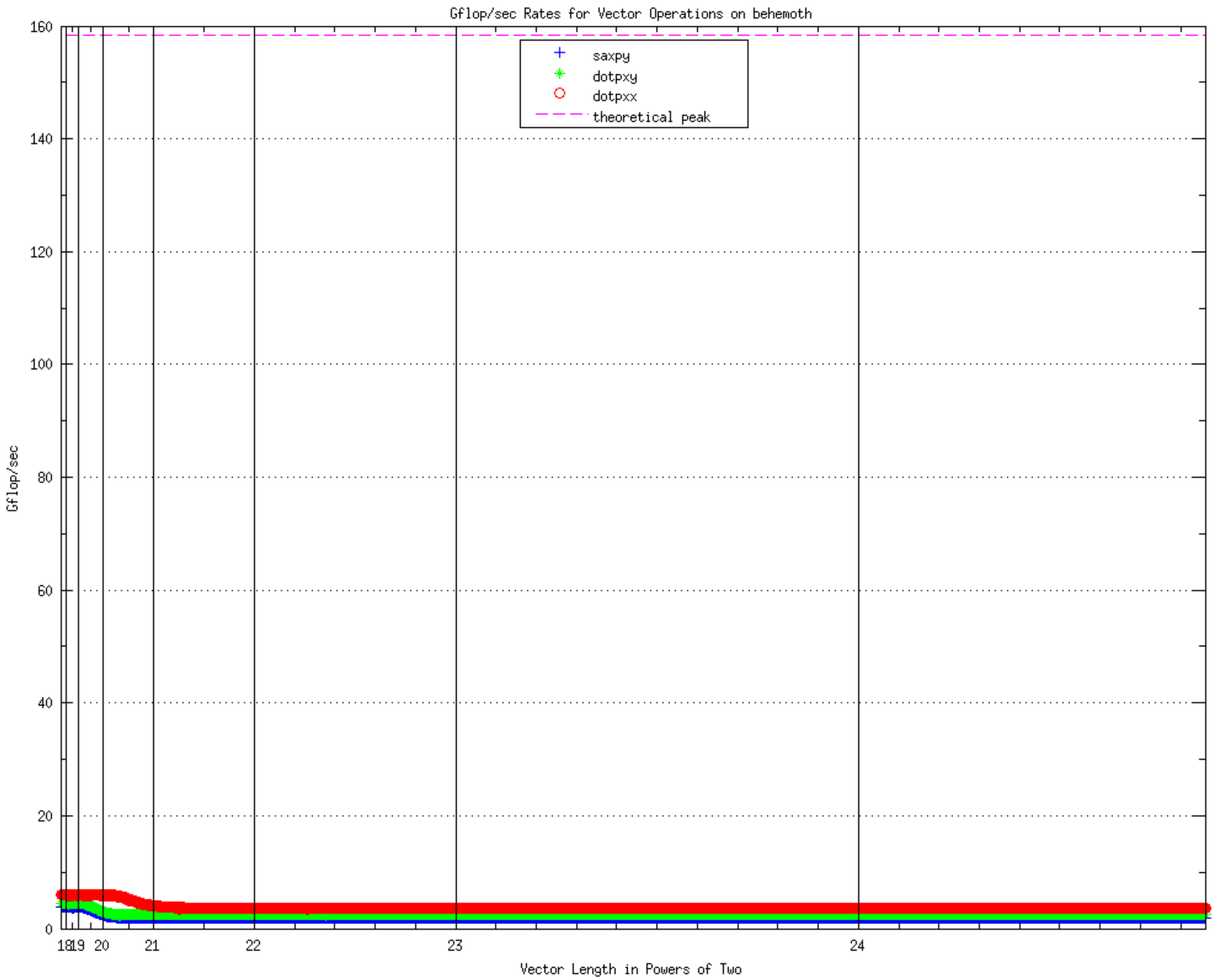
Fundamental Principle

- Analytic tool is *load/store analysis*, and real easy to use: it is the ratio of data required to computation
- Example: Let x and y be n-vectors, α = scalar
  - daxpy = vector update; $y = y + \alpha * x$
  - dotpxy = dotproduct with 2 vectors; $\alpha = x^T y$
  - dotpxx = dotproduct with 1 vector; $\alpha = x^T x$
- Load/store predicts dotpxx > dotpxy > daxpy in speed for large n (it also predicts their relative performance)
- Can implement Gaussian elimination to have either dotpxy or daxpy as the innermost kernel operation; which is better to choose?
- We can (and will) do even better than that, guided by load/store analysis

Load/store example



Fundamental Principle



Fundamental Principle

- Load/store analysis based on ratio of just 2 numbers
- $r = m/f$, where
  - f = number of floating point operations required
  - m = minimum possible number of memory references
- f comes from looking at code fragment
- m comes from looking at mathematical operation, not from any code or machine implementation
- Load/store analysis is the pass/fail knowledge for P573
- As the course proceeds, will define it more fully, look at it in more detail, and apply it in more examples

Next Steps (Interleaved)

- Basics of computer architecture
- How to reliably time things on a system
- Load/store analysis of linear algebra kernels
- Implementing linear solvers, least squares solvers, principal component analysis, etc. using high-perf techniques
- Floating point oddities and usage
- Matlab basics
- Your immediate tasks include
  - make sure you have the required knowledge
  - check the course web page https://www.cs.indiana.edu/classes/p573
- A PDF of these slides will be there