Injecting PDC into the CS Curriculum

Joel Adams
Department of Computer Science
Calvin College

CALVIN

What Are The Key PDC Topics?

PDC != *concurrency*:

- ***Parallel* emphasizes:**
  - **Throughput / Performance (and timing)**
  - **Scalability (performance scales with # of threads)**
  - **Topics like *speedup*, *Amdahl's Law***
- ***Distributed* emphasizes:**
  - **Multiprocessing (no shared memory)**
    - **MPI, MapReduce/Hadoop/Spark, BOINC, …**
  - ***Cloud computing***
  - **Mobile apps accessing web services**

Software: Communication Options

- Communicate via the ***shared-memory***
  - Languages: Java, C++11, …
  - Libraries: POSIX threads, **OpenMP**
- Communicate via ***message passing***
  - Message-passing languages: Erlang, Scala, …
  - Libraries: the Message Passing Interface (**MPI**)

ACM / IEEE CS2013 Curriculum

- **The CS2013 core curriculum includes 15 hours of parallel & distr. computing (PDC) topics:**
  - **5 hours in core Tier 1**
  - **10 hours in core Tier 2**
  - **More parallel topics in *System Fundamentals***
- **ABET criteria 5.a.3 requires *“Exposure to … parallel and distributed [computing].”***
- **How/where do we cover these topics in the CS curriculum?**

Model 1: Add a New Course

Add a new course to the CS curriculum that covers the core PDC topics:

- \+ If someone else has to teach this new course, then PDC is their problem, not mine!
  - ○ But what happens if that person leaves?
- – Curriculum is already full!
  - ○ What course do we drop to make room?
- – Students don't see PDC applied consistently
  - ○ If early, they'll likely forget much of it
  - ○ If late, the cognitive adjustment is much harder

Model 2: Across the Curriculum

Spread at least 15 hours (3 weeks) of PDC content across select core CS courses:

- + Explore PDC in context of data structures, algorithms, prog. lang., OS, …
- + Easier to add 1 week to a few courses than jettison an entire course.
- + Spreads the effort across multiple faculty
- – All those faculty have to be "on board"
  - Getting faculty buy-in is the biggest challenge
    - – Use TCPP Early Adopter funding to provide "carrots"

Model 2: Where to Start?

- CS1 would be ideal
  - + Supports "early and often"
  - – What do we eliminate to make room for PDC?
  - – Instructor buy-in a bigger challenge
    - o Many sections == many instructors
  - – Many students struggle with sequential CS1 concepts and can't see past syntax
    - o How will they master abstract PDC concepts?
  - ? Perhaps limit to "unplugged" PDC activities?

Calvin CS Curriculum

Year	Fall Semester	Spring Semester
1	*Intro to Computing* *Calculus I*	*Data Structures* *Calculus II*
2	*Algorithms & DS* *Intro. Comp. Arch.* *Discrete Math I*	*Programming Lang.* *Discrete Math II*
3	*Software Engr.* *Adv. Elective*	*OS & Networking* *Adv. Elective* *Statistics*
4	*Adv. Elective: HPC* *Sr. Practicum I*	*Adv. Elective* *Sr. Practicum II* *Perspectives on Comp.*

Why Introduce Parallelism in CS2?

- *Performance* (Big-Oh) is a topic that’s first addressed in CS2

- Data structures let us store *large data sets*
  – Slow sequential processing of these sets provides a *natural motivation for parallelism*

Parallel Topics in CS2

- Lecture topics:
  - Threads: Single threading vs. multithreading
  - The single-program-multiple-data (SPMD), fork-join, parallel loop, and reduction patterns
  - Speedup, asymptotic performance analysis
  - Race conditions: non-thread-safe structures
  - Live-coding demos of these using the patternlets

CS2 Lab Exercise Possibilities

Using OpenMP:

- Compare times of sequential vs. parallel operations on large matrix objects (e.g., addition, transpose)

OR

- Compare times of sequential vs parallel image-processing operations (e.g., image inversion, gray-scale, blur) using TSGL

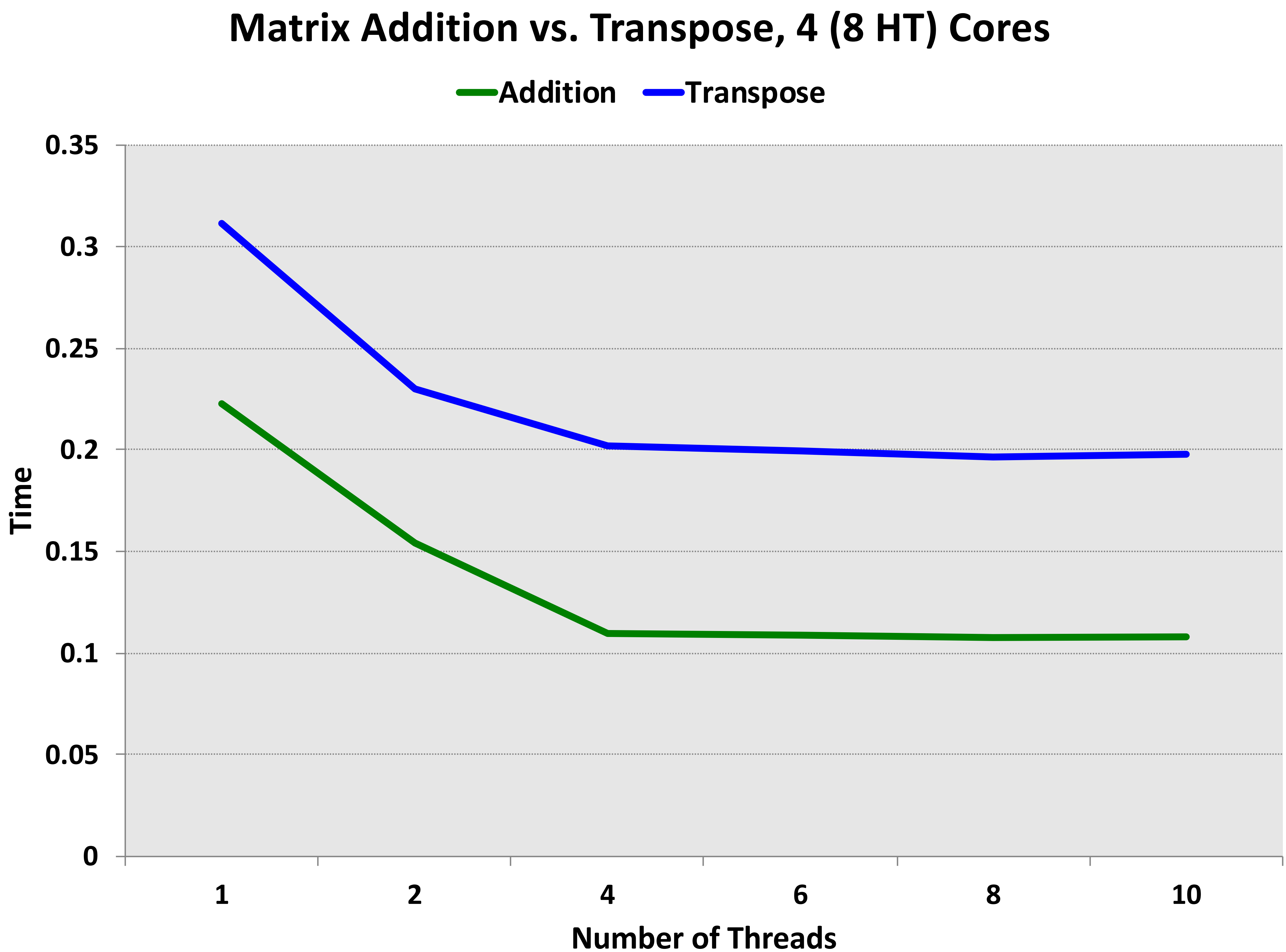
Lab Exercise: Matrix Operations

Given a Matrix class, the students:

- **Measure the time to perform sequential addition and transpose methods**

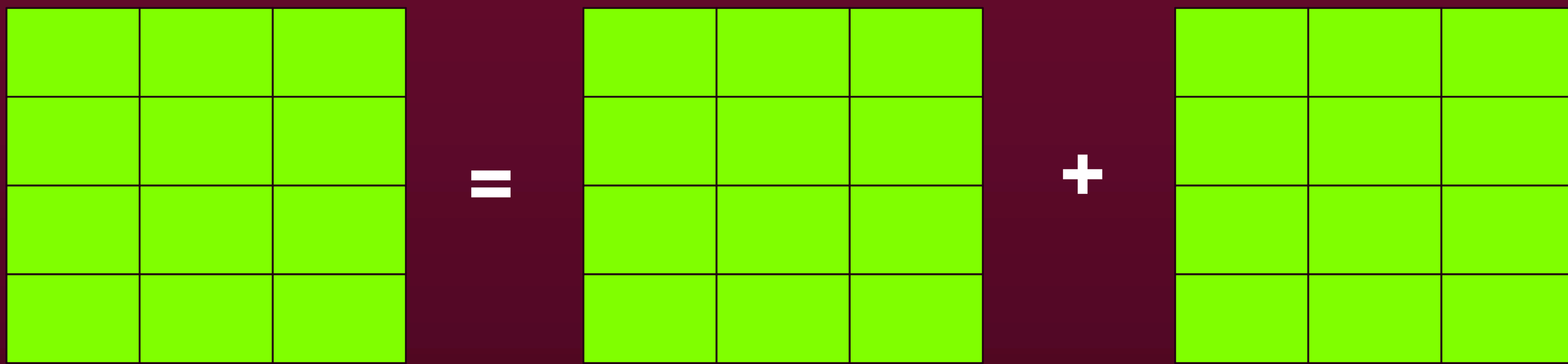
- **For each of three different approaches:**
  - **Use the approach to parallelize those methods**
  - **Record execution times in a spreadsheet**
  - **Create a chart showing time vs # of threads**

Students *directly experience* the benefits…

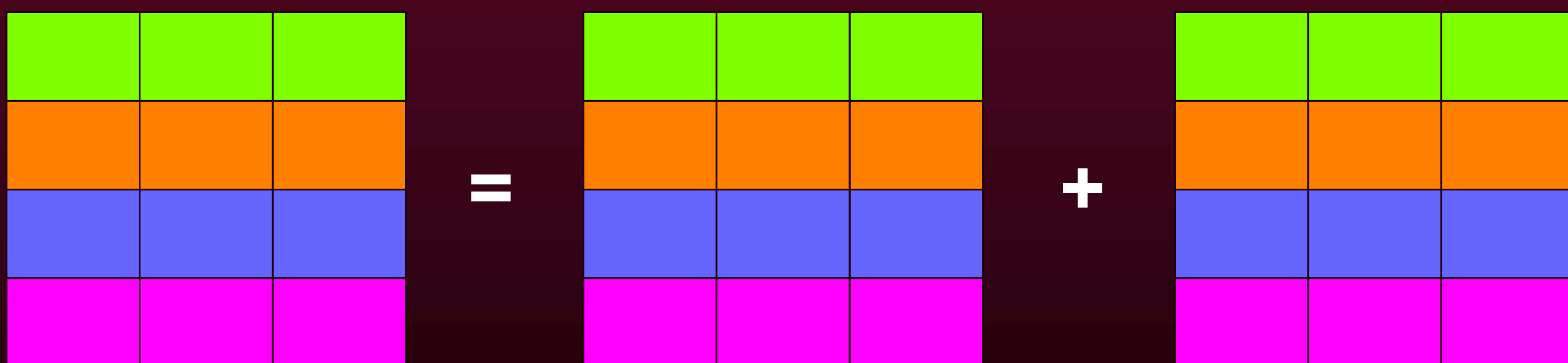

Matrix Addition vs. Transpose, 4 (8 HT) Cores
Addition
Transpose
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Time
1
2
4
6
8
10
Number of Threads

Addition: m3 = m1 + m2

Single-threaded:

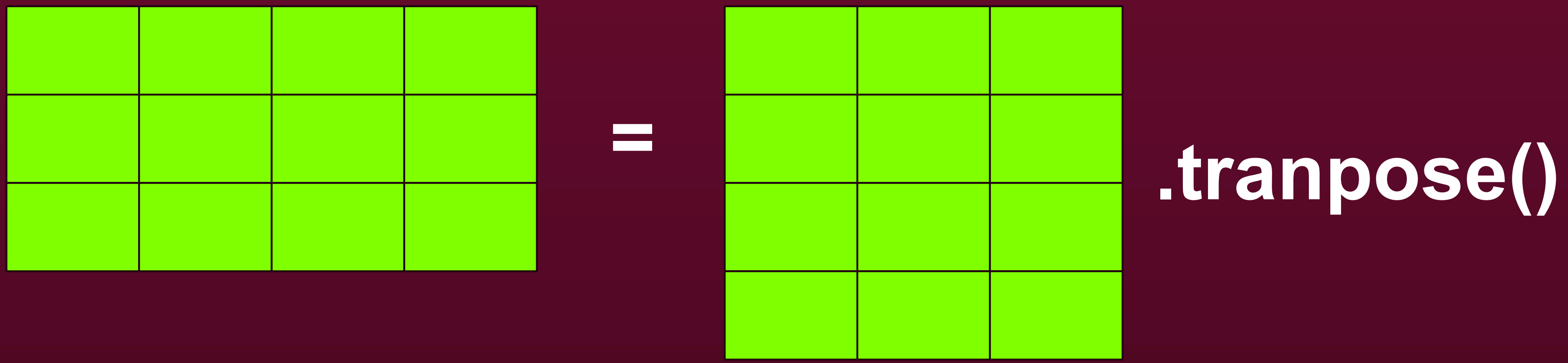


Multi-threaded (4 threads):

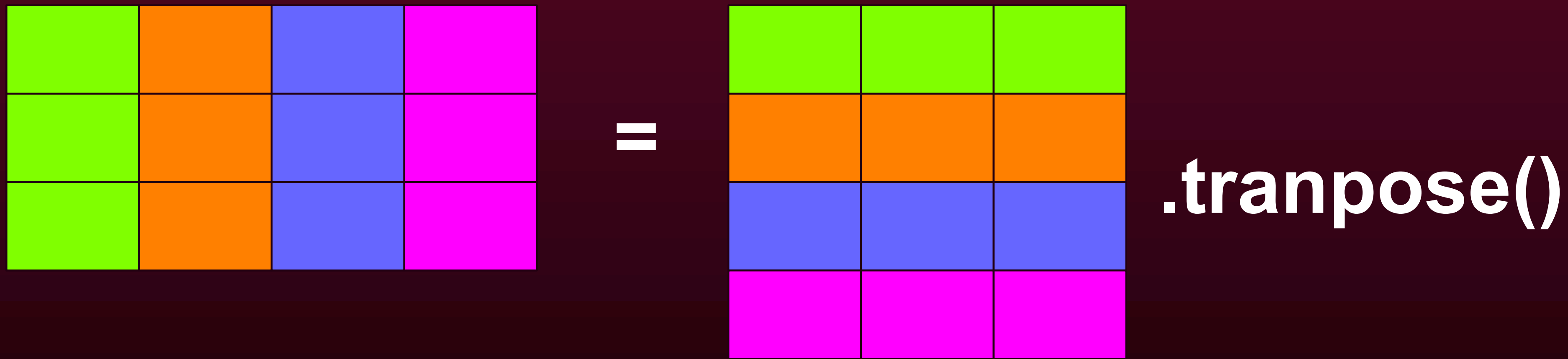


Tranpose: m2 = m1.transpose()

Single-threaded:



Multi-threaded (4 threads):



Programming Project

- Parallelize other Matrix operations
  - Multiplication
  - Assignment
  - Constructors
  - Equality
- Some operations (file I/O) are inherently sequential, providing a useful lesson…

Assessment

All students complete end-of-course evaluations with open-ended feedback:

- **They *really* like the week on parallelism**
  - **Covering material that is not in the textbook makes CS2 seem *fresh* and *cutting edge***
  - **Students really like learning how they can use *all their cores* instead of just one**
  - **Having students *experience speedup* is key (and even better if they can *see* it)**

PDC in CS3 (*Algorithms*)

Parallel Algorithms:

- Parallel Searching
- Parallel Sorting
- Distributed Graph Algorithms
- Parallel features in C# (.NET)
- …

PDC in *Programming Languages*

- Shared Memory Communication
  - Race conditions
  - Synchronization mechanisms: *Using*…
    - Semaphores, Locks, Condition Variables, Monitors,
- Distributed Memory Communication
  - Send-Receive in different languages
  - Blocking vs non-blocking behavior
- Lab exercise: Compare multithreading performance in Ada, Clojure, Java, Ruby

PDC in *Software Engineering*

Distributed computing via the cloud…

- Accessing cloud services via APIs
- Group Project: Client-server system
  - Front-side mobile app
  - Server-side in the cloud

PDC in *OS & Networking*

- Shared Memory Communication
  - Race conditions
  - Synchronization mechanisms: *Building*…
    - Semaphores, Locks, Condition Variables, Monitors,
- Distributed Memory Communication
  - Sockets, RPC, Send-Receive behavior, …
- Final Project: Multithreaded Client-Server

Digging Deeper

- Covering PDC in core courses ensures that every major receives basic exposure
- For students who want more, we have CS 374: *High Performance Computing*
  - 5 weeks of MPI
  - 1 week of Pthreads
  - 2 weeks of OpenMP
  - 1 week of MPI+OpenMP
  - 2 weeks of CUDA
  - 1 week of Hadoop (needs to morph to Spark)

Summary

- Every CS major needs "exposure" to PDC
  - One course model
  - Multiple course model
    - CS2 is a possible place to introduce parallelism
    - Gradual progression:
      a. 'Embarrassingly parallel' problems to avoid race conditions
      b. *Using* synchronization mechanisms
      c. *Implementing* synchronization mechanisms
      d. Leave deeper study as an advanced elective
- What will work at your institution?