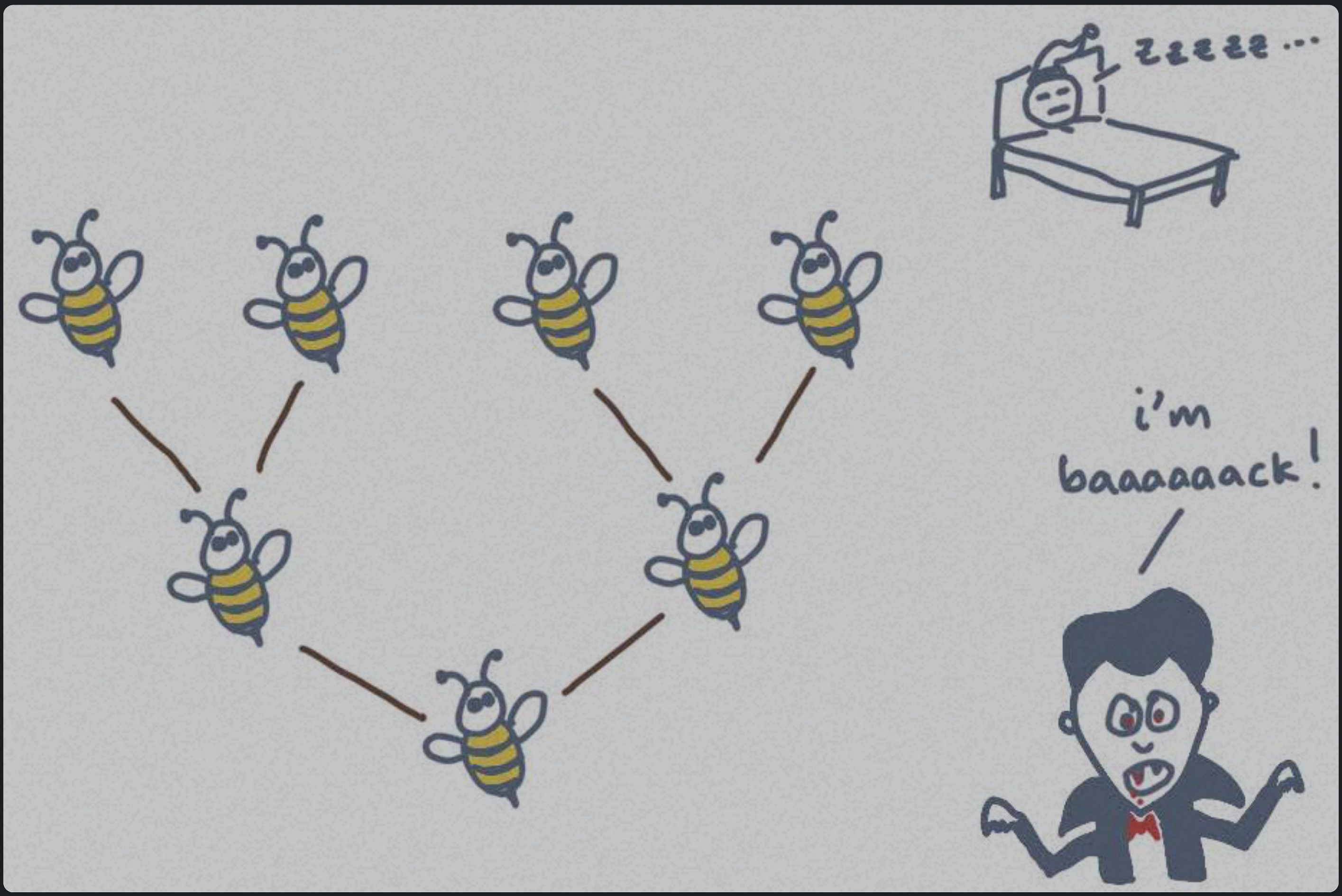
ABOUT



This course is a graduate-level introduction to parallel computing. Its goal is to give you the foundations to develop, analyze, and implement parallel and locality-efficient algorithms and data structures.

ACCESS TO COURSE

MATERIALS

The course appears as "CSE 6220-O01" in OSCAR.

- **Udacity videos**: https://classroom.udacity.com/courses/ud281
- **Piazza discussion forums**: http://piazza.com/gatech/spring2018/cse6220o01
- **T-Square**: https://t-square.gatech.edu/portal/site/gtc-5eba-dfc9-5a69-9060-f81992587653
- **Public reviews of this course**: https://omscentral.com/courses (Login may be required)
- **Schedule**: View schedule here

WHAT YOU'LL LEARN

When we talk about scalability, "*scale*" has two senses: efficiency as the problem size grows and efficiency as the system size—as measured by the numbers of cores or compute nodes—grows.

To really scale your algorithm in both senses, you can start by reducing asymptotic complexity (remember your days as a CS 101 `n00b` ?). But then you *also* need to reduce communication and data movement. This course is about the basic algorithmic techniques you'll need, especially for the latter.

The course videos focus on theory. But that only gets you so far. So, you will need to supplement this algorithmic theory with hands-on labs on parallel shared-memory and distributed-memory machines; otherwise, how will you know if something that works in theory also works well in practice?

The specific techniques you will encounter cover the main algorithm design and analysis ideas for three major classes of machines:

1. multicore and manycore shared memory machines, via the work-span (or “multithreaded DAG”) model;
2. distributed memory machines like clusters and supercomputers, via network models;
3. and sequential or parallel machines with deep memory hierarchies (e.g., caches), via external memory models.

You will see these techniques applied to fundamental problems, like sorting, searching on trees and graphs, and linear algebra, among others. You will implement several of them on real parallel and distributed systems, using practical programming models such as Cilk Plus, OpenMP, MPI, or possibly others.

WHY? (MOTIVATION)

The “standard” undergraduate CS curriculum begins, and usually ends, with the sequential (or serial) random access machine (“RAM”) model. This course helps to fill the gap between algorithm design for serial RAM machines and real machines, which will always have multiple cores, multiple nodes, vector units, and deep memory hierarchies.

HPC vs. HPCA? This course *complements* CS 6290 / d007: High-Performance Computer Architecture. The main difference between this course (“HPC”) and that course (“HPCA”) is that HPC is about *algorithms* and HPCA is about *hardware* (primarily, microprocessors).

So, take both and crush it like nobody’s business!

> *In retrospect, I guess we should have been called these courses “HPC-A” for algorithms and “HPC-H” for hardware (or “HPC-M” for machines). But that would have been less confusing, and where is the fun in that?*

WHAT DO I NEED TO KNOW?

KNOW? (PREREQUISITES)

You should be comfortable designing and analyzing basic sequential algorithms and data structures using “big-Oh-my” notation. That’s the stuff you learned in a first or second course in algorithms and data structures, a la Georgia Tech’s CS 3510 or Udacity’s Intro to Algorithms.

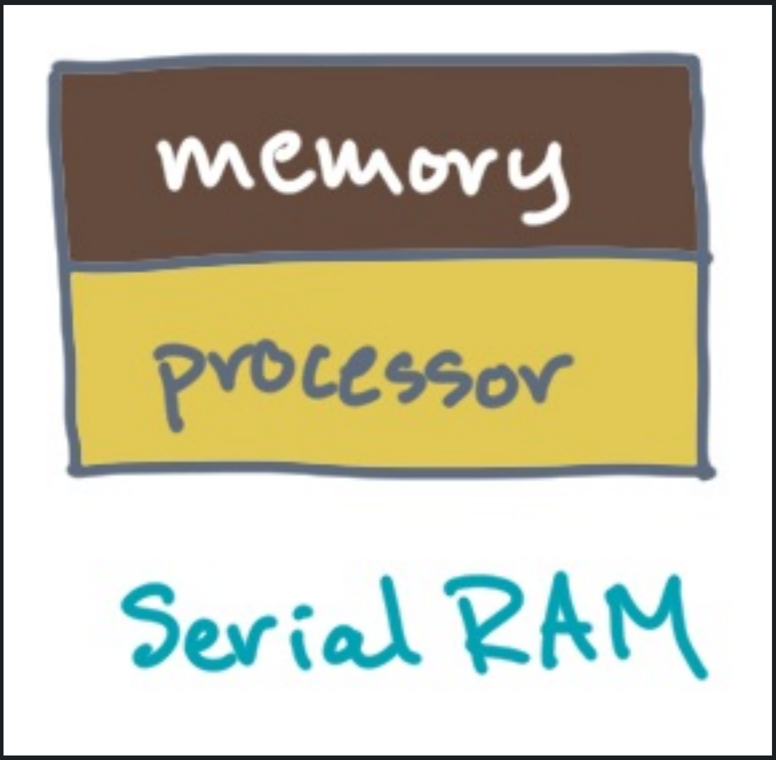
You should also be comfortable programming in C or C++ for the programming assignments. Experience using command-line interfaces in *nix environments (e.g., Unix, Linux) is also helpful, though there are software carpentry materials that may help with that, a la https://software-carpentry.org, for instance.

Please also review the course readiness survey for CSE 6220. It includes some topics in calculus, linear algebra, probability, and algorithms, which are required in many undergraduate CS programs. It’s not so much that you need to remember how to solve *exactly* those sorts of problems without looking things up. Rather, you should feel confident that you *can* make time to brush-up as needed. Being able to do so is strongly correlated with the level of mathematical maturity and independence necessary for this class.

Lastly, check out the reviews of this course, in which alumna tell you what they think you actually need to succeed beyond what I've spelled out here. https://omscentral.com/courses

WHAT ELSE? (TOPICS)

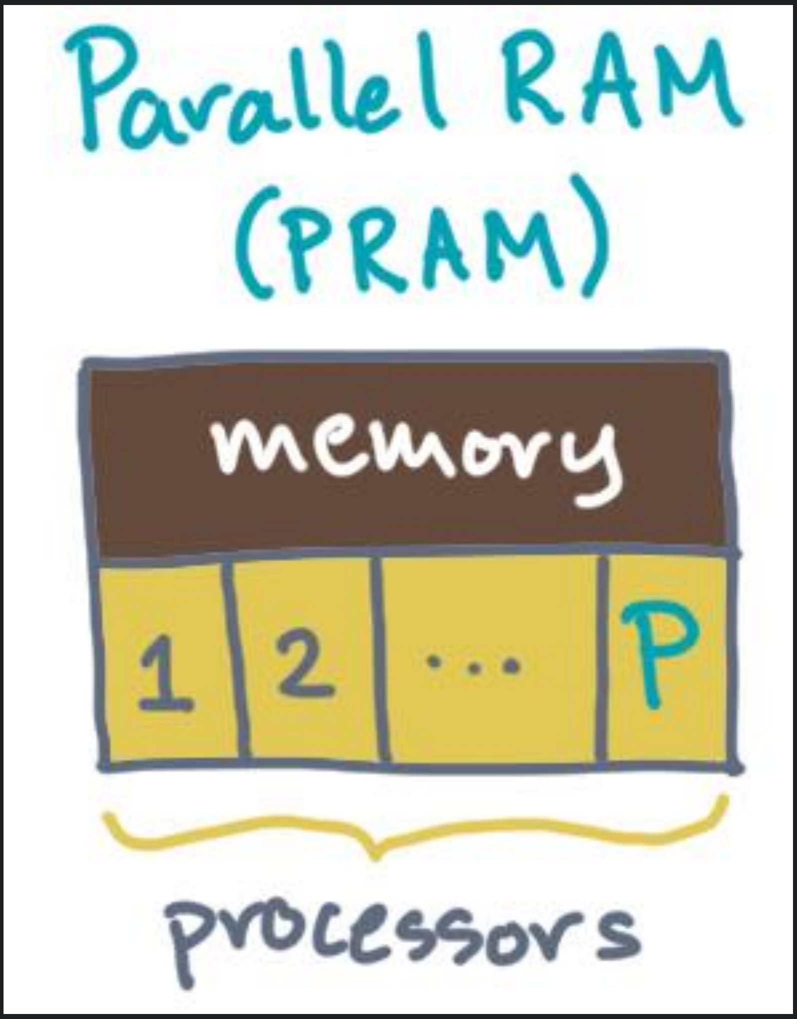
We've organized the course topics around three different ideas or extensions to the usual serial RAM model of CS 101.



Recall that a serial RAM assumes a sequential or serial processor connected to a main memory.

Unit 1: The work-span or dynamic multithreading model for shared-memory machines.

This model assumes that there are multiple processors connected to the main memory. Since they can all "see" the same memory, the processors can coordinate and communicate via reads and writes to that "shared" memory.



Sub-topics include:

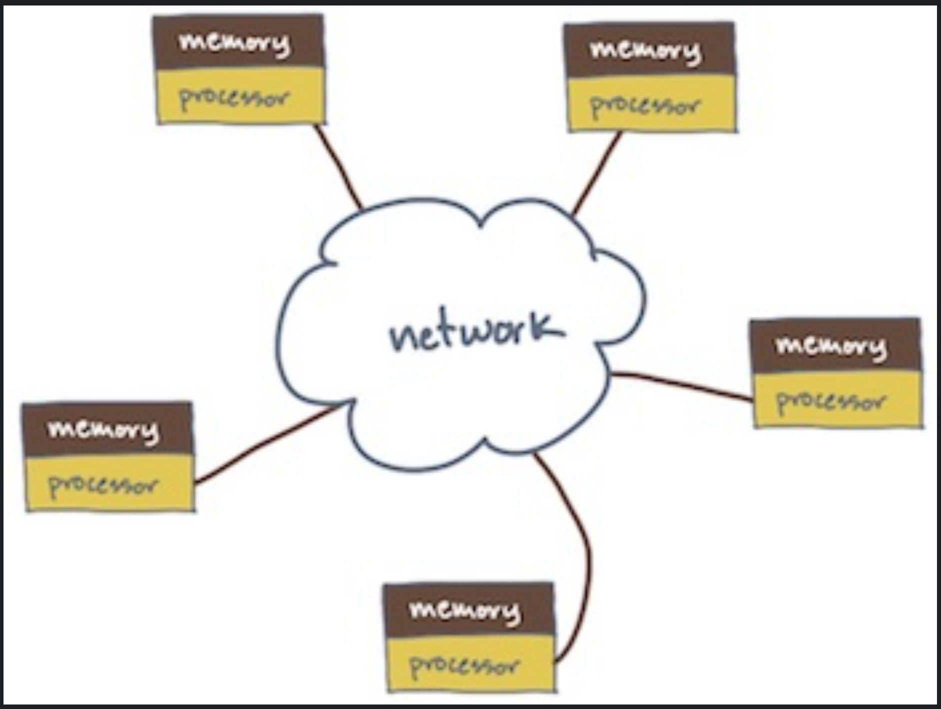
- Intro to the basic algorithmic model
- Intro to OpenMP, a practical programming model
- Comparison-based sorting algorithms
- Scans and linked list algorithms
- Tree algorithms
- Graph algorithms, e.g., breadth-first search

Unit 2: Distributed memory or network models.

This model says there is not one serial RAM, but many serial

This model says there is not one serial RAM, but many serial RAMs connected by a network. Each serial RAM's memory is

private to other RAMs; consequently, the processors must coordinate and communicate by sending and receiving messages. Sort of like passing notes to that other kid you had a crush on in school.

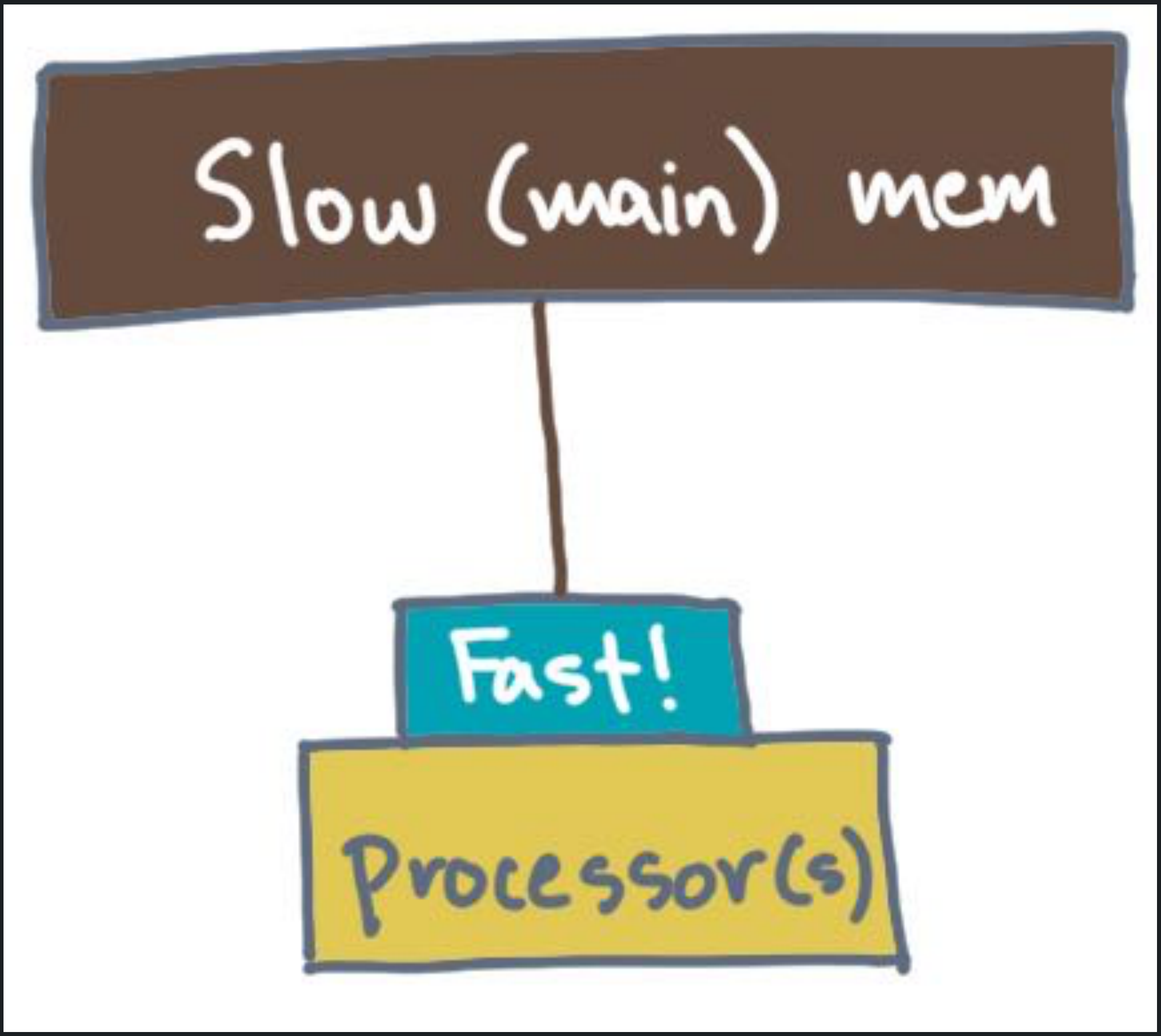


Sub-topics include:

- The basic algorithmic model
- Intro to the Message Passing Interface, a practical programming model
- Reasoning about the effects of network topology
- Dense linear algebra
- Sorting
- Sparse graph algorithms
- Graph partitioning

Unit 3: Two-level memory or I/O models.

This model returns to a serial RAM, but instead of having only a processor connected to a main memory, there is a smaller but faster "scratchpad" memory in between the two. The algorithmic question here is how to use the scratchpad to minimize costly data transfers from main memory.



Sub-topics include:

- Basic models

- Efficiency metrics, including “emerging” metrics like energy and power
- I/O-aware algorithms
- Cache-oblivious algorithms

WHO? (TEACHING STAFF)

- Main instructor: Prof. Richard (Rich) Vuduc
- Teaching assistants: Andrew Becker (OMSCS and CSE 6220-OMS alumnus extraordinaire), Jianan Gao, and Rohan Vora
- Course developers (Udacity): Amanda Deisler

WHERE? (LOGISTICS)

Your first and best source of questions and answers is the class discussion forum on Piazza. We make all critical announcements there, including when assignments go out, reminders about when they are due, clarification of various policies, errata, and hints.

- Link: http://piazza.com/gatech/spring2018/cse6220o01

Prof. Vuduc strongly prefers that, if you need to contact him, you post a private note to the instructors on Piazza rather than you post a private note to the instructors on Piazza rather than sending him email. (You'll get a much faster response from him by doing so.)

For more tips on using Piazza effectively, see these notes.

Office hours. We will hold weekly office hours. Please see Piazza for details.

SCHOOL SUPPLIES

Minimum Technical Requirements.

- Pencil, paper, and brain!
- Browser and connection speed: We strongly recommend an up-to-date version of Chrome or Firefox. We also support Internet Explorer 9 and the desktop versions of Internet Explorer 10 and above (not the metro versions). 2+ Mbps recommended; at minimum 0.768 Mbps download speed
- Operating System: The following are the recommended operating systems for the course. We may also elect to provide virtual machines with a standardized environment, though most of the assignments can be completed by directly logging into the HPC resource we will provide via secure shell (ssh).

- PC: Windows XP or higher with latest updates installed
- Mac: OS X 10.6 or higher with latest updates installed
- Linux: Any recent distribution that has the supported browsers installed.

Textbook and readings. There will be supplemental readings, provided via papers and the following textbook (available electronically to Georgia Tech students for free):

- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. *Introduction to Parallel Computing.* Addison-Wesley, 2003. Available electronically to GT students.

HOW? (ASSIGNMENTS AND EXAMS)

Beyond the videos, there will be supplemental readings and a set of programming assignments. See the schedule for a list of assignments, their due dates, and their associated lessons.

Due date convention. For all labs, the posted due date should be interpreted as "23:59 anywhere on earth" (11:59 pm AOE). For example, if an assignment is due on January 31, as long as there is any place on the planet Earth where it is 11:59 pm or earlier, your submission is on time. (You are responsible for taking signal transmission delays into account, especially if you are connecting from outer space.)

> *For the ultimate in precision timing, here's a handy AOE clock: http://www.timeanddate.com/time/zones/aoe*

Grading. For each programming assignment you submit, we will check your implementation against some test cases for correctness and we will also measure how fast your implementation is. The minimum grade for a correct (but possibly slow) implementation is roughly equivalent to a 'C'; to get a higher grade, your implementation should also be fast, with a 'B' meaning reasonable performance and 'A' for high performance.

But how do you know whether your implementation is slow or fast? We will post guidelines with each assignment.

Late policy. You get two "late passes" – that is, for any two programming assignments, you may submit the assignment up to 48 hours after the official due date without penalty. Any assignment submitted after you run out of passes or after 48 hours (with or without a pass) will get zero credit. We have to enforce a hard limit so that we can grade the assignments and return results within a reasonable timeframe.

> *Late passes* ***may not be used for exams.*** *No late exams will be graded or accepted. You may use no more than one late pass per programming assignment.*

Collaboration policy. All Georgia Tech students are expected to uphold the Georgia Tech Academic Honor Code. Honest and ethical behavior is expected at all times. All incidents of suspected dishonesty will be reported to and handled by the Dean of Students office. Penalties for violating the collaboration policy can be severe; alleged violations are adjudicated by the Dean of Students office and not by the instructor.

Collaboration on assignments is encouraged at the “whiteboard” level. That is, you may share ideas and have technical conversation, which we especially encourage on the Piazza forums, but you must write and submit your own code.

Exams must be done individually but are open-book and “open-internet” (for looking up information only). That is, while taking the exam you are not allowed to use the internet or to actively get help by, for instance, posting questions or messaging with others. We will use the Proctortrack to administer the exams. (The students shown on the Proctortrack website all look so happy, so it must be good, right?)

> *This semester, we will experiment with "stay-home" exams, which are roughly equivalent to "take-home" exams for on-campus students.*

ANYTHING ELSE? (MISCELLANEOUS)

- Set your local timezone in T-Square: link
- Brief orientation videos for the OMS CS program: link