COMP 322: Parallel and Concurrent Programming

Lecture 1: Welcome to COMP 322!

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

Acknowledgments

Vivek Sarkar
Zoran Budimlić
Dan Wallach
Shams Imam

Your Teaching Staff!

- TAs
  - Stefan Boskovic, Haotian Dang, Andrew Ondara, Huzaifa Ali, Raahim Absar
- Instructor
  - Mack Joyner (mjoyner@rice.edu)

Office Hours

- Regular office hour schedule can be found at Office Hours link on course web site
- Send email to instructor (mjoyner@rice.edu) or TAs if you need to meet some other time
- Use Piazza to post and answer questions

Course Syllabus

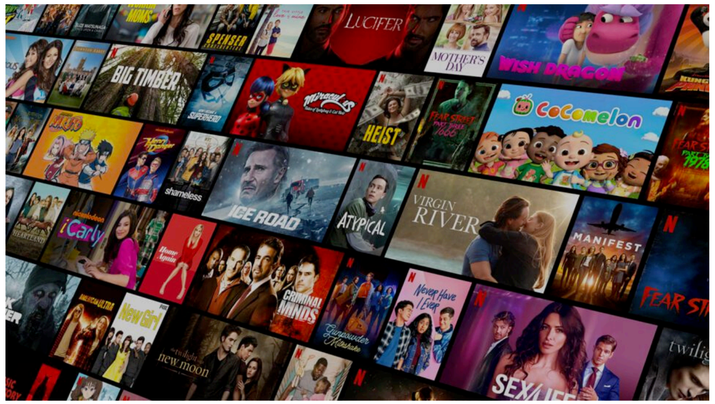
- Fundamentals of Parallel and Concurrent Programming taught in six topics
  1. Functional Programming Fundamentals (2 weeks)
  2. Creating Parallel and Concurrent Programs using Functional Concepts (4 weeks)
  3. Fundamentals of Shared-Memory Concurrent Programming (3 weeks)
  4. Data Parallel Programming Model and Loop Parallelism and Concurrency (2 ⅔ weeks)
  5. Message-Passing Programming Model (1 ⅓ week)
  6. Miscellaneous Topics (1 week)
- Labs and programming assignments will be in Java 11
- Habanero-Java (HJ) library developed at Rice as a pedagogic parallel programming model
  - Only used as a uniform way to illustrate the concepts
  - We will relate concepts to other languages and frameworks

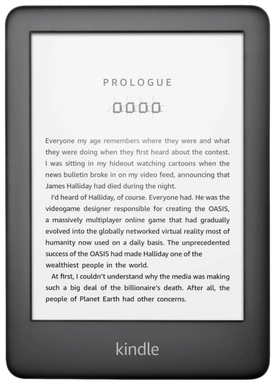
Grade Policies

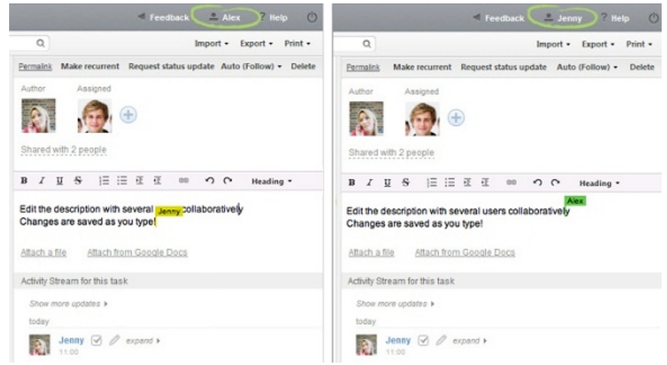
Course Rubric

- Homework (5) 40% (written + programming components)
  - Weightage proportional to # weeks for homework
- Exams (2) 40% (scheduled midterm + scheduled final)
- Weekly Labs 10% (labs need to be submitted by the following Monday)
- Quizzes 5% (on-line quizzes on Canvas)
- Class Participation 5% (in-class worksheets)

Parallelism and Concurrency are Everywhere



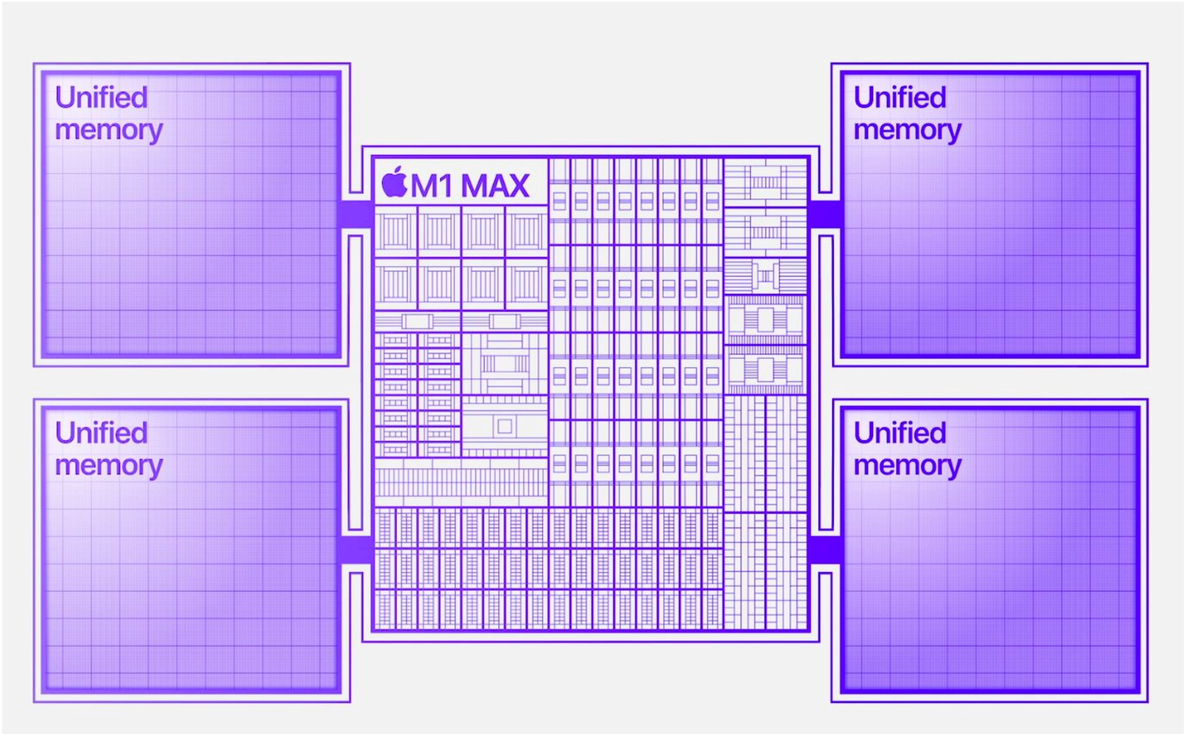




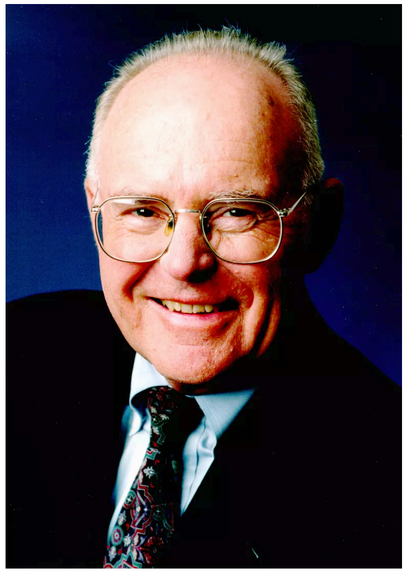
What is Parallel Computing

- Parallel computing: using multiple processors in parallel to solve problems quicker than with a single processor and/or with less energy

- Example of a parallel computer: Apple M1 MAX
  - 10-core Multi-Processor
    - 8 performance cores
    - 2 high-efficiency cores
  - 32 GPU cores
  - 16 neural engine cores
  - Unified memory on-chip (up to 64GB)
  - 57 billion transistors

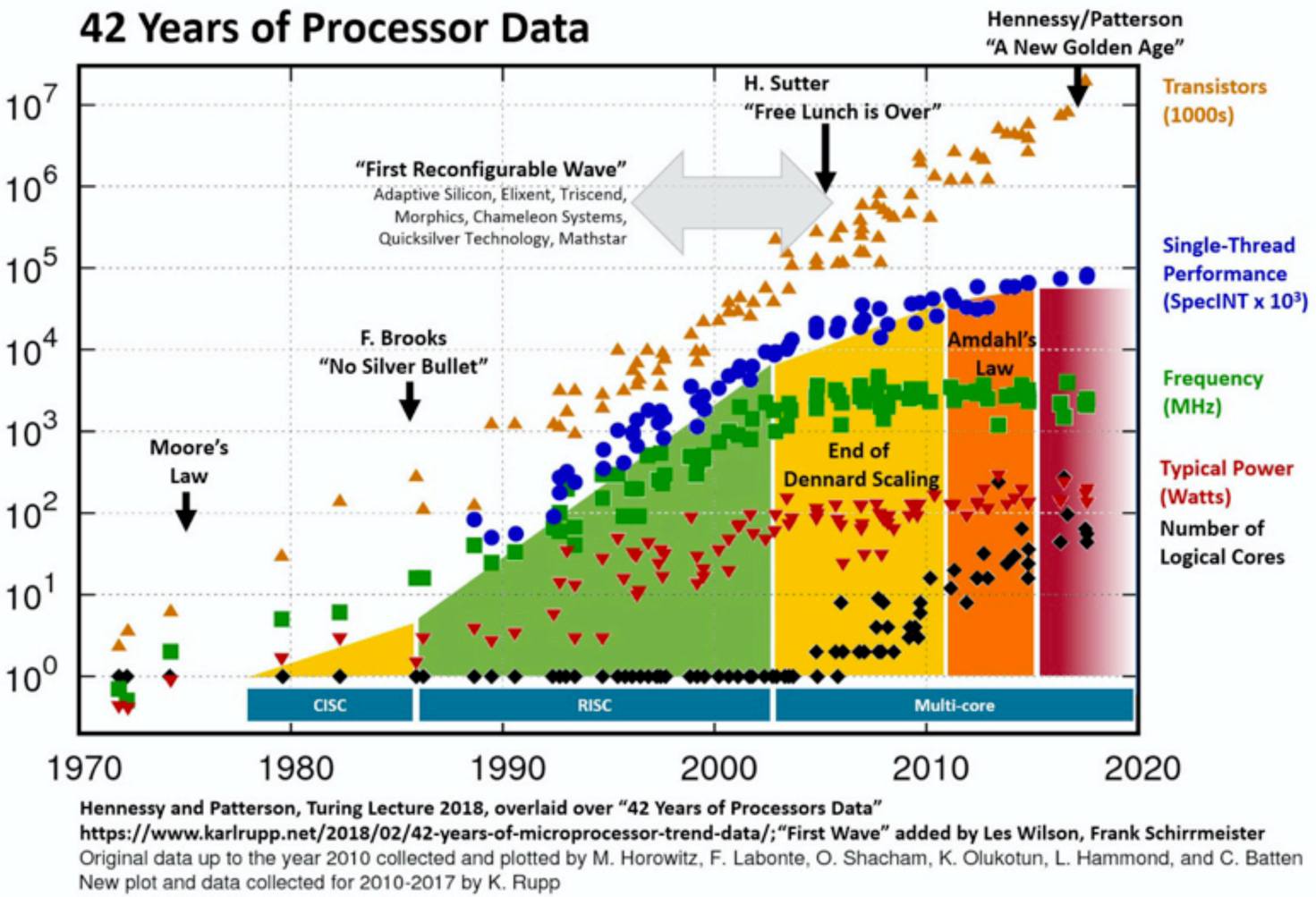


Why?



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips wou double roughly every 1-2 years (Moore's Law)

- area of a transistor halves every 1-2 years
- feature size reduces by √2 every 1-2 years



Hennessy and Patterson, Turing Lecture 2018, overlaid over "42 Years of Processors Data"
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/;"First Wave" added by Les Wilson, Frank Schirrmeister
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Parallelism Saves Power (Simplified Analysis)

Maximum Frequency is capped by Voltage

➔ Power is proportional to (Frequency)^3

Baseline example: single 1GHz core with power P

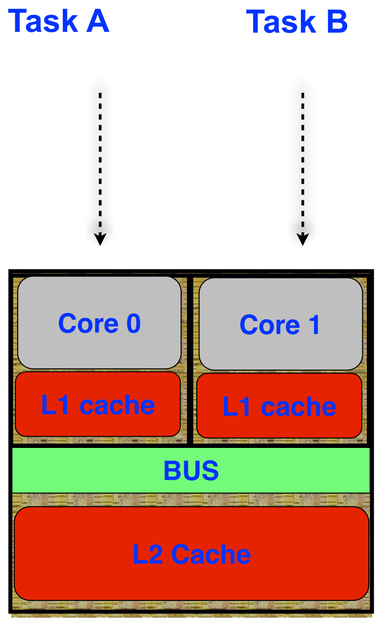
Option A: Increase clock frequency to 2GHz -> Power = 8P

Option B: Use 2 cores at 1 GHz each -> Power = 2P

- Option B could deliver the same performance as Option A with 4x less power … provided your software runs in parallel!

What is Parallel Programming

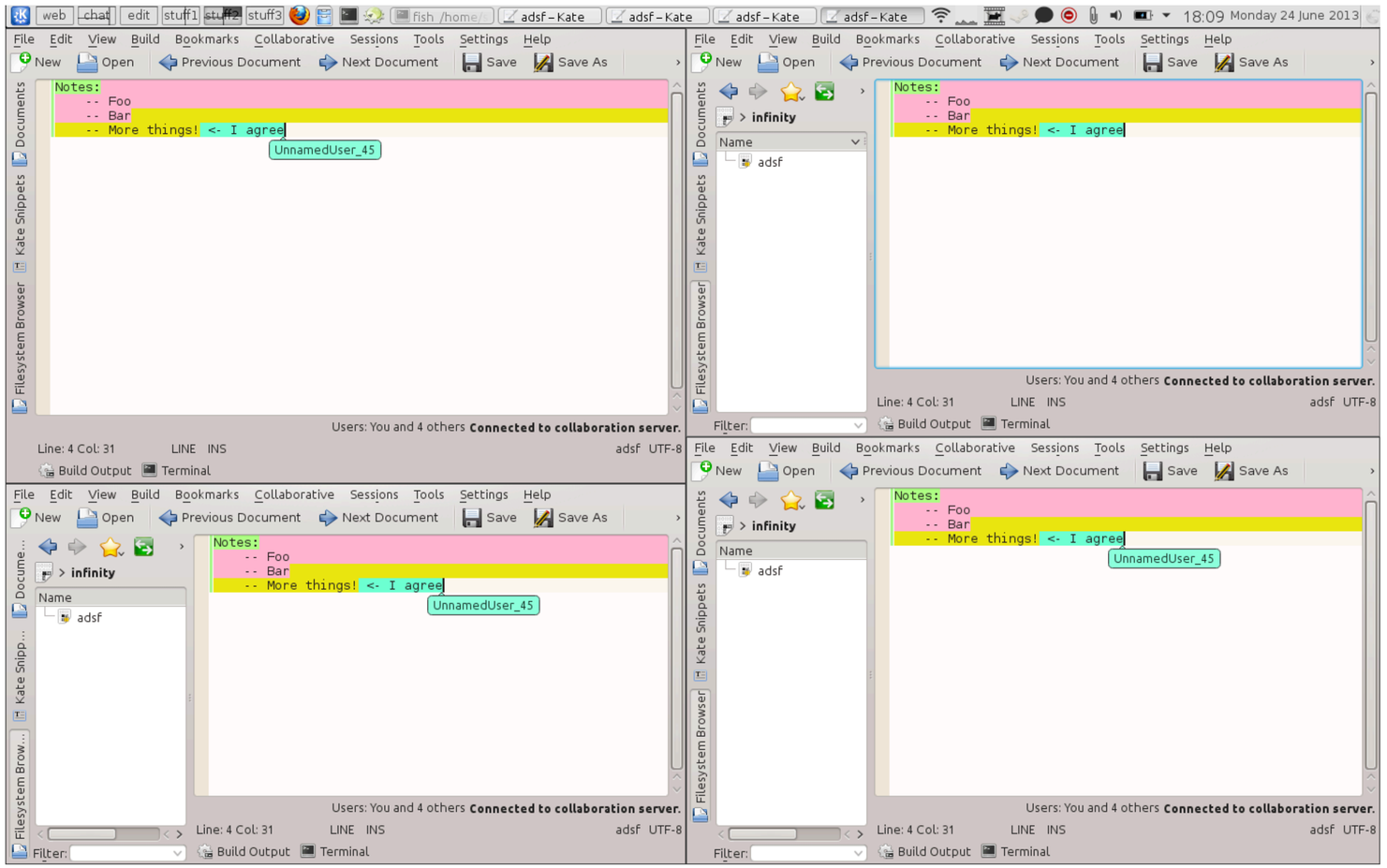
- Specification of operations that can be executed in parallel
- A parallel program is decomposed into sequential subcomputations
- Threads, Jobs, Processes, *Tasks*
- Parallel programming constructs define the semantics of task creation, termination, location and interaction



Schematic of a dual-core Processor

What is Concurrent Programming

- Programming technique
- An application making progress on two or more tasks at the same time
- Execute and complete in an overlapping time period
- Manage access to ***shared resources***



Concurrent vs. Parallel Programming

- "Things happening at the same time"
- *Parallel Programming*: Improve performance
- *Concurrent Programming*: Solve a problem by using concurrently executing tasks and managing shared resources
- Example: 200 COMP 322 students, one final exam with 10 questions
- Parallelism without Concurrency
  - 10 TAs, each TA grades 20 complete exams
- Concurrency without Parallelism
  - 1 TA, the TA grades question 1 first for all exams, then question 2, and so on …
- Concurrency and Parallelism
  - 10 TAs, each TA grades a separate question

Example Problem: Computing the sum of array elements

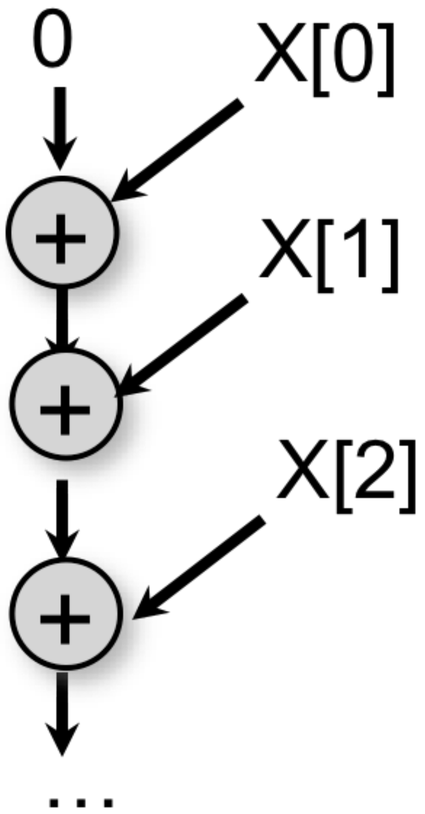
Algorithm 1: Sequential ArraySum

Input: Array of numbers, X.
Output: sum = sum of elements in array X.

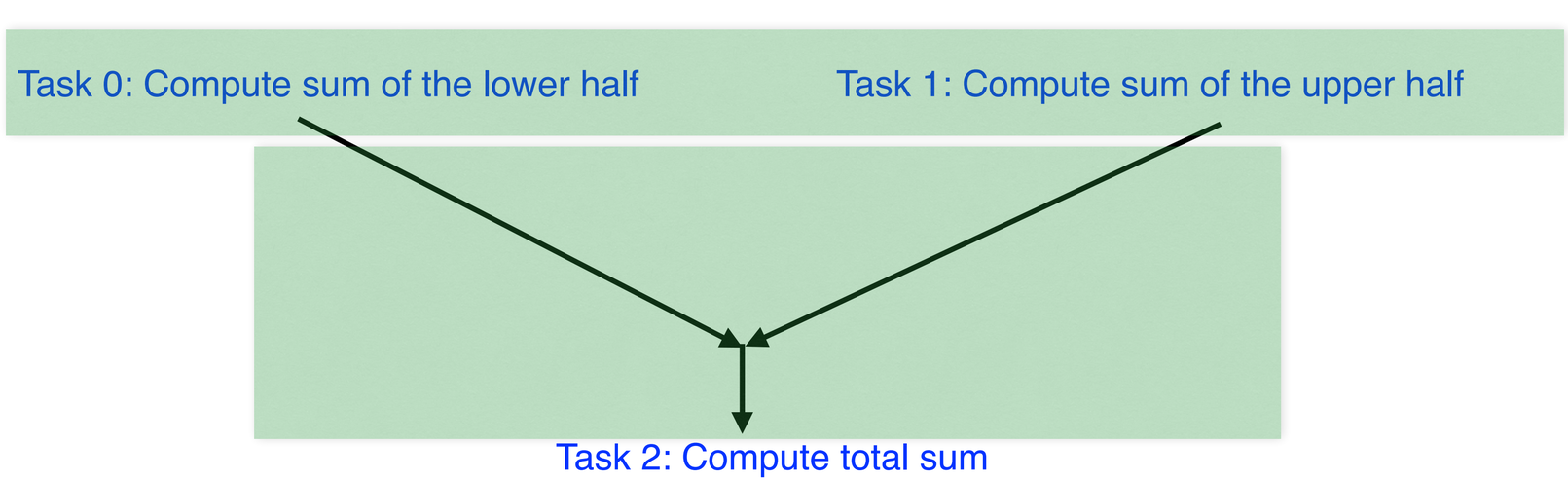
```
sum ← 0;
for i ← 0 to X.length − 1 do
    sum ← sum + X[i];
return sum;
```

Observations:

- The decision to sum up the elements from left to right was arbitrary
- The computation graph shows that all operations must be executed sequentially



Two-way Parallel Array Sum



Parallelism!

Concurrency!

Basic idea:

- Decompose problem into two tasks for partial sums (parallelism)
- Combine results to obtain final answer ***after partial sums are done*** (concurrency)
- This is a parallel divide-and-conquer pattern