COMP 322: Fundamentals of Parallel Programming

Lecture 1: The What and Why of Parallel Programming; Task Creation & Termination (async, finish)

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Acknowledgments

—CS 194 course on "Parallel Programming for Multicore" taught by Prof. Kathy Yelick, UC Berkeley, Fall 2007

- http://www.cs.berkeley.edu/~yelick/cs194f07/

—"Principles of Parallel Programming", Calvin Lin & Lawrence Snyder, Addison-Wesley 2009

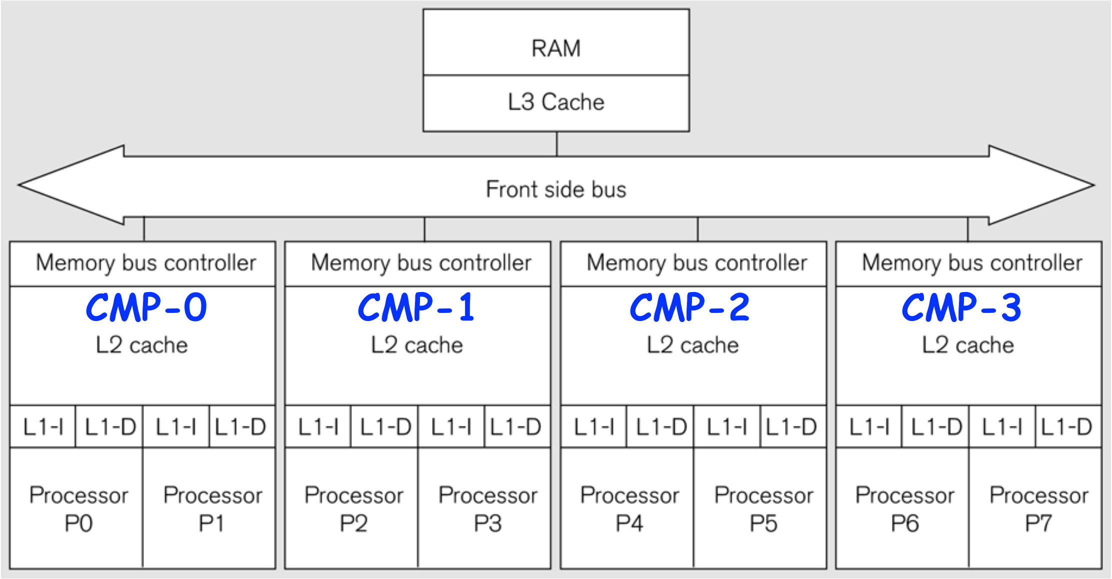
—COMP 322 Module 1 handout, Sections 0.1, 0.2, 1.1

—edX lecture and demonstration videos for Module 1, topic 1.1

What is Parallel Computing?

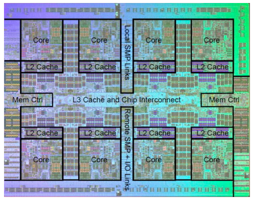
- **Parallel computing:** using multiple processors in parallel to solve problems more quickly than with a single processor and/or with less energy

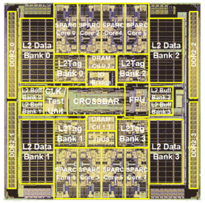
- Example of a parallel computer
    - —An 8-core Symmetric Multi-Processor (SMP) consisting of four dual-core chip microprocessors (CMPs)

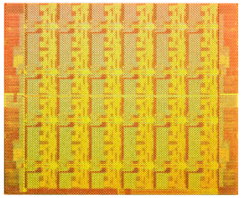


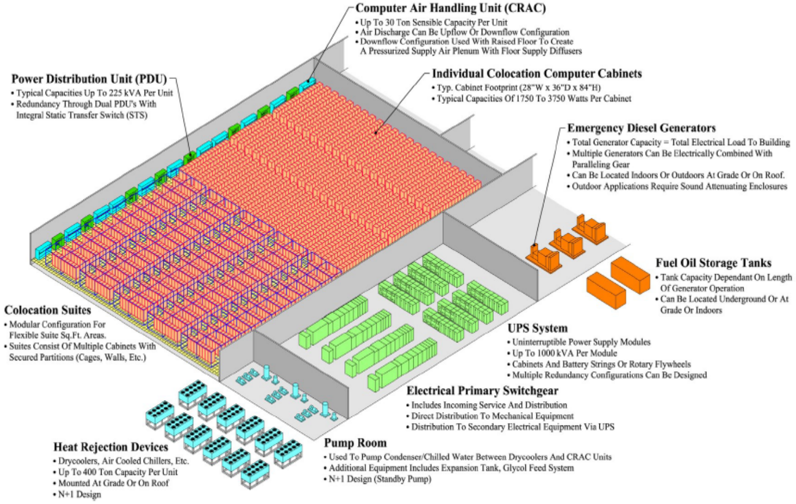
Source: Figure 1.5 of Lin & Snyder book, Addison-Wesley, 2009

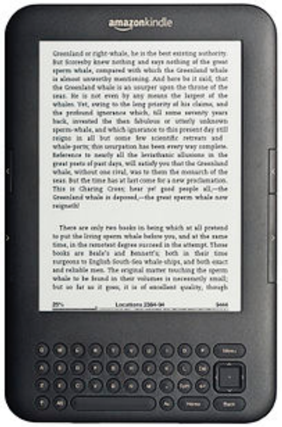
All Computers are Parallel Computers --- Why?



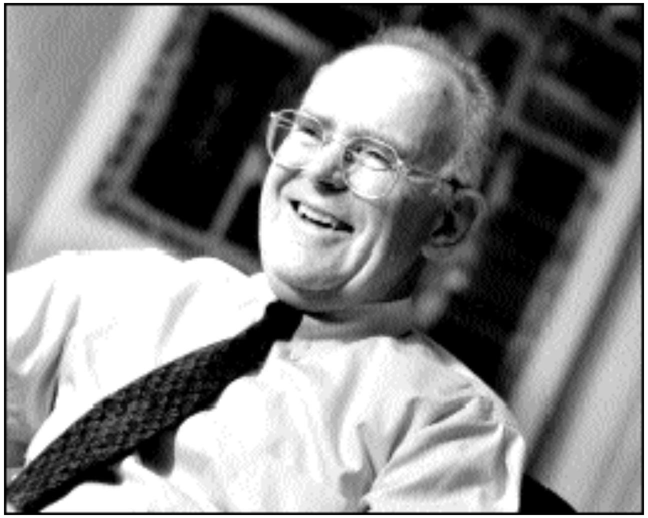


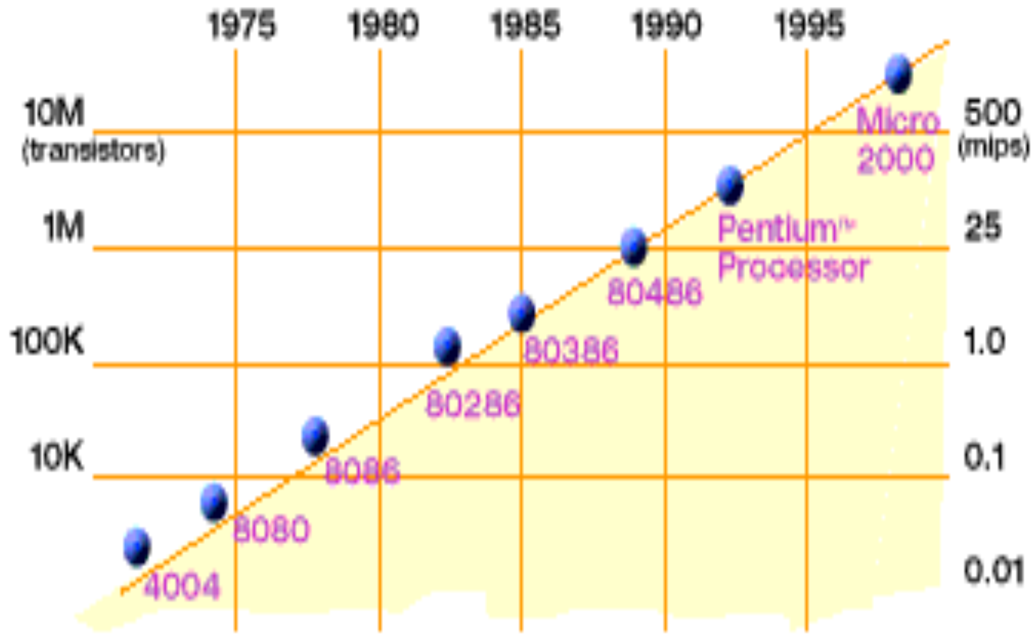






Moore's Law and Dennard Scaling





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

⇒ area of transistor halves every 1-2 years

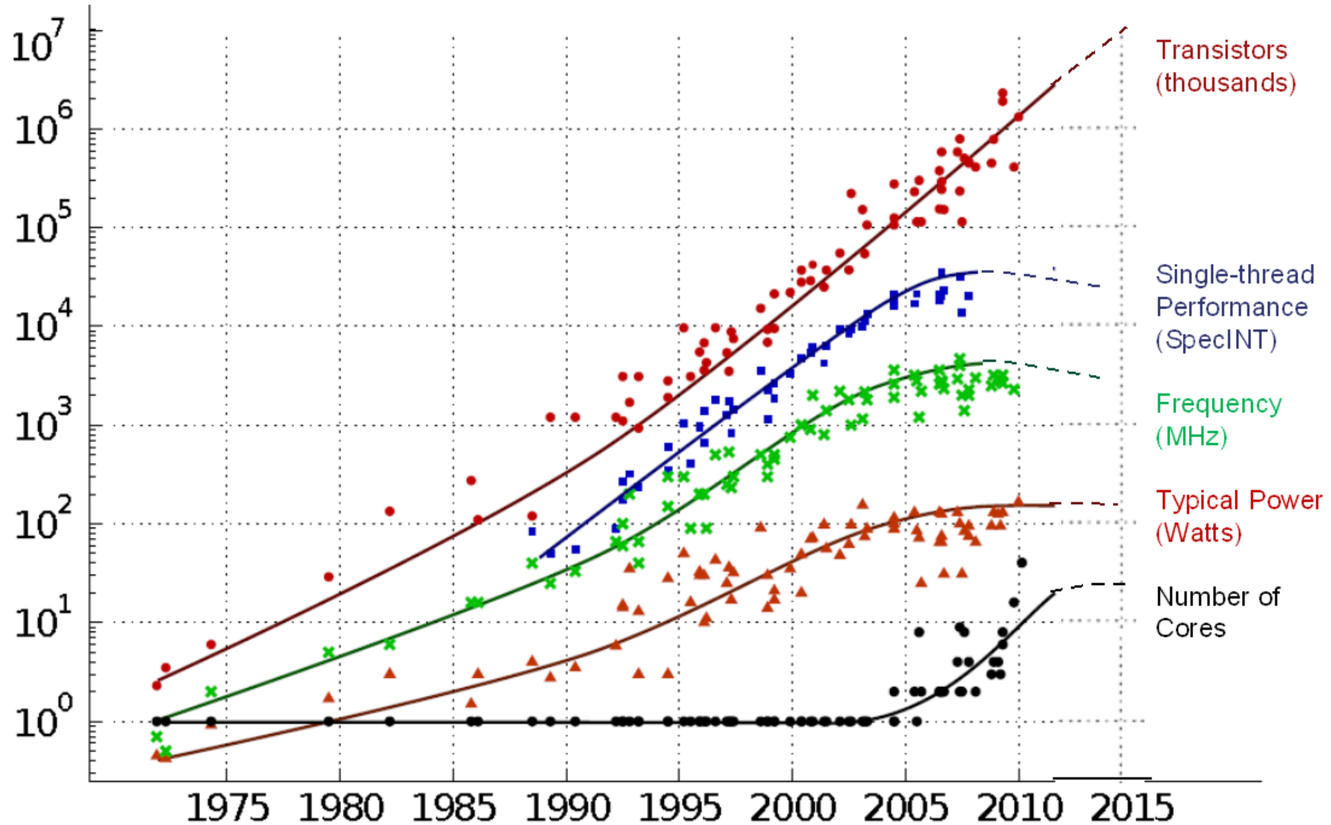
⇒ feature size reduces by √2 every 1-2 years

Dennard Scaling states that clock frequency can be increased as transistor size decreases

Current Technology Trends —
Moore's Law continues, but Dennard Scaling ends

- **Chip density is continuing to increase ~2x every 2 years**
  - — **Clock speed is not!**
  - — **Number of processors is doubling instead**
- **Parallelism must be managed by software**

35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Parallelism Saves Power (Simplified Analysis)

Nowadays, Power ~ (Capacitance) * $(\text{Voltage})^2$ * (Frequency)

and maximum Frequency is capped by Voltage

➔ Power is proportional to $(\text{Frequency})^3$

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 delivers same performance as Option A with 4x less power … provided software can be decomposed to run in parallel!

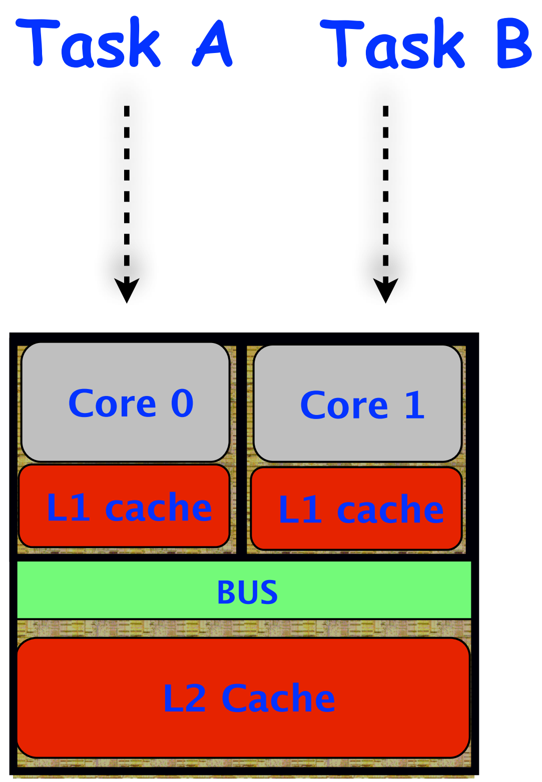
A Real World Example

- Fermi vs. Kepler GPU chips from NVIDIA's GeForce 600 Series
  - —Source: http://www.theregister.co.uk/2012/05/15/nvidia_kepler_tesla_gpu_revealed/

	Fermi chip (released in 2010)	Kepler chip (released in 2012)
Number of cores	512	1,536
Clock frequency	1.3 GHz	1.0 GHz
Power	250 Watts	195 Watts
Peak double precision floating point performance	665 Gigaflops	1310 Gigaflops (1.31 Teraflops)

What is Parallel Programming?

- Specification of operations that can be executed in parallel
- A parallel program is decomposed into sequential subcomputations called ***<u>tasks</u>***
- Parallel programming constructs define task creation, termination, and interaction



Schematic of a dual-core Processor

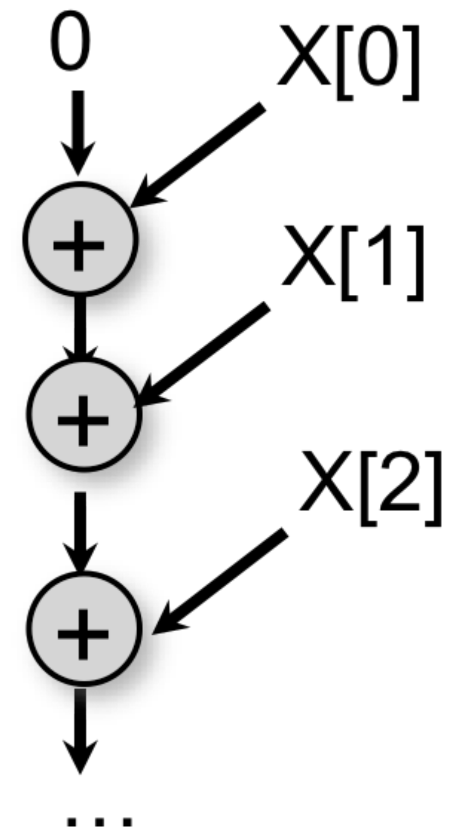
Example of a Sequential Program: Computing the sum of array elements

Algorithm 1: Sequential ArraySum

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

$sum \leftarrow 0$;
for $i \leftarrow 0$ **to** $X.length - 1$ **do**
  $sum \leftarrow sum + X[i]$;
return sum;

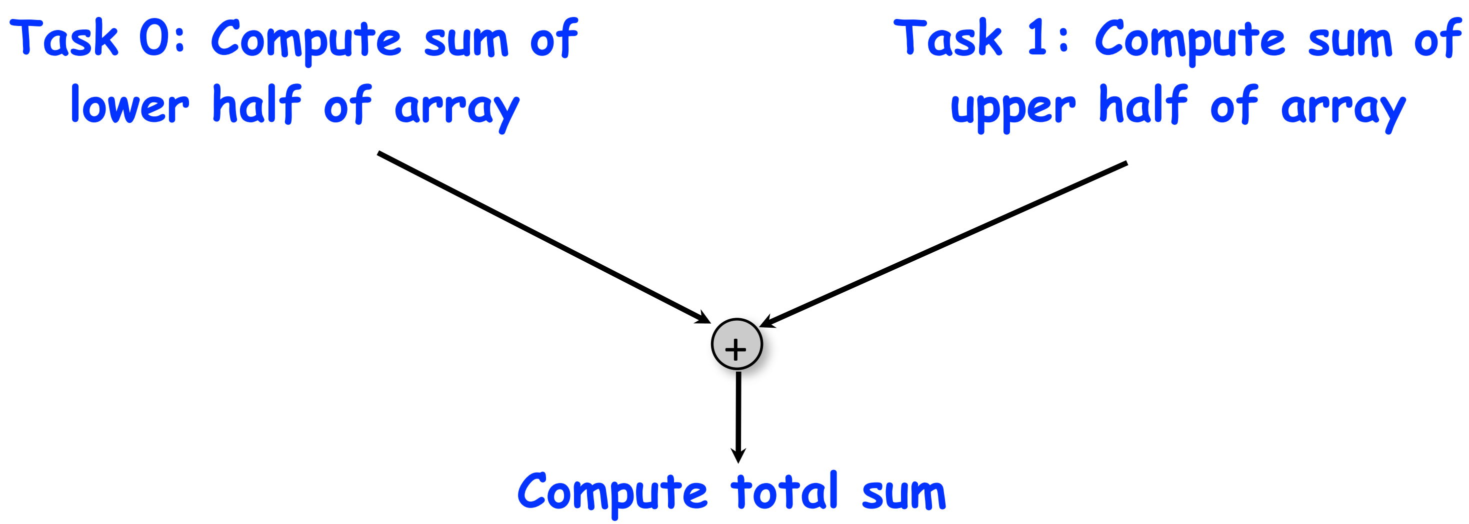
Computation Graph



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

Parallelization Strategy for two cores (Two-way Parallel Array Sum)



Basic idea:

- Decompose problem into two tasks for partial sums
- Combine results to obtain final answer
- Parallel divide-and-conquer pattern

Async and Finish Statements for Task Creation and Termination (Pseudocode)

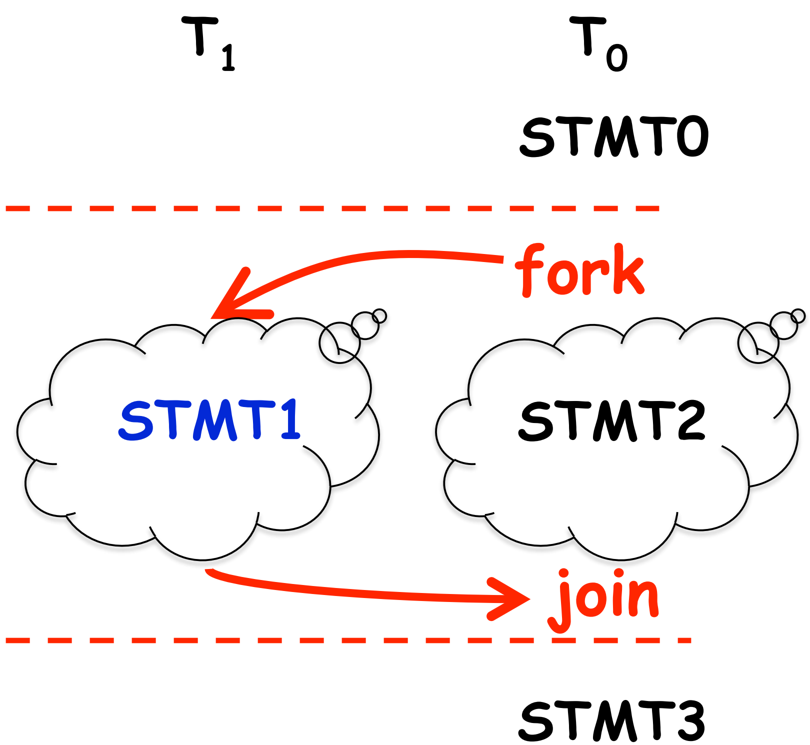
async S

- Creates a new child task that executes statement S

finish S

- Execute S, but wait until *all* asyncs in S's scope have terminated.

```
// T0(Parent task)
STMT0;
finish {     //Begin finish
  async {
    STMT1; //T1(Child task)
  }
  STMT2;     //Continue in T0
             //Wait for T1
}            //End finish
STMT3;       //Continue in T0
```



Two-way Parallel Array Sum using async & finish constructs

Algorithm 2: Two-way Parallel ArraySum

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

```
// Start of Task T1 (main program)
sum1 ← 0; sum2 ← 0;
// Compute sum1 (lower half) and sum2 (upper half) in parallel.
finish{
    async{
        // Task T2
        for i ← 0 to X.length/2 − 1 do
            sum1 ← sum1 + X[i];
    };
    async{
        // Task T3
        for i ← X.length/2 to X.length − 1 do
            sum2 ← sum2 + X[i];
    };
};
// Task T1 waits for Tasks T2 and T3 to complete
// Continuation of Task T1
sum ← sum1 + sum2;
return sum;
```

Course Syllabus

- Fundamentals of Parallel Programming taught in three modules
  1. Parallelism
  2. Concurrency
  3. Locality & Distribution
- Each module is subdivided into units, and each unit into topics
- Lecture and lecture handouts will introduce concepts using algorithmic and pseudocode notations
- Labs and programming assignments will be in Java 8
  - Initially, we will use the Habanero-Java (HJ) library developed at Rice as a pedagogic parallel programming model
    - HJ-lib is a pure Java 8 library (no special compiler support needed)
    - HJ-lib contains many features that are easier to use than standard Java threads, and are also expected in future parallel programming models
  - Later, we will learn parallel programming in standard Java

Grade Policies

Course Rubric

- Homeworks (6) 40% (written + programming components)
- Exams (2) 40% (take-home midterm + scheduled final)
- Quizzes & Labs 10% (quizzes on edX, labs graded as in COMP 215))
- Class Participation 10% (classroom Q&A, Piazza discussions, in-class worksheets)

Grading curve (we reserve the right to give higher grades than indicated below!)

>= 90% ⇒ A or A+

>= 80% ⇒ B, B+, or A-

>= 70% ⇒ C, C+ or B-

others ⇒ C- or below

Next Steps

- **IMPORTANT:**
  - Send email to comp322-staff@mailman.rice.edu if you did NOT receive a welcome email from us
  - Bring your laptop to this week's lab at 7pm on Wednesday (Section A01: DH 1064, Section A02: DH 1070)
  - Watch videos for topics 1.2 & 1.3 for next lecture on Wednesday
- Complete each week's assigned quizzes on edX by 11:59pm that Friday. This week, you should submit quizzes for lecture & demonstration videos for topics 1.1, 1.2, 1.3, 1.4
- HW1 will be assigned on Jan 16th and be due on Jan 28th
- See course web site for work assignments and due dates
  - https://wiki.rice.edu/confluence/display/PARPROG/COMP322