COMP 322 / ELEC 323: Fundamentals of Parallel Programming
Lecture 1: Task Creation & Termination (async, finish)

Instructors: Mack Joyner, Zoran Budimlić
Department of Computer Science, Rice University
{mjoyner, zoran}@rice.edu

http://comp322.rice.edu
Special thanks to Vivek Sarkar!
Your Teaching Staff!

• Undergraduate TAs
  — Abbey Baker, Ashok Sankaran, Austin Bae, Avery Whitaker, Aydin Zanager, Eduard Danalache, Frank Chen, Hamza Nauman, Harrison Brown, Jahid Adam, Jeemin Sim, Kitty Cai, Madison Lewis, Ryan Han, Teju Manchenella, Victor Gonzalez, Victoria Nazari

• Graduate TAs
  — Jonathan Sharman, Srdjan Milakovic

• Instructors
  — Mack Joyner, Zoran Budimlić
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)

  ![Diagram of a parallel computer](source: Figure 1.5 of Lin & Snyder book, Addison-Wesley, 2009)
All Computers are Parallel Computers --- Why?

[Images of various computer components and devices, including laptops, tablets, smartphones, and computer hardware.

Text: "Computer Air Handling Unit (CAU)"
- Up to 300°F/93°C in the Data Center
- Emergency Diesel Generator
- Total Capacity = Total Electrical Load + Building
- Multiple Generators Can Be Electrically Coupled for Fil Purification
- Can Be Located Inside or Outside of Grade Or On Roof
- Outdoor Applications Require Sound Isolating Enclosures

"UPS System"
- Consistently Power Supply Module
- Up to 400VAC Per Module
- Can Be Used for Specialized Or Rotary Flywheels
- Multiple Redundancy Configuration Can Be Provided

"Electrical Primary Switchgear"
- Includes Inching Switches And Distribution
- Direct Distribution To Mechanical Equipment
- Distribution To Secondary Electrical Equipment Via UPS

"Heat Rejection Devices"
- Air-Cooled Chillers, Etc.
- To Withstand Capacity Per Unit"
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 power for a fixed chip area remains constant as transistors grow smaller

Slide source: Jack Dongarra
Recent Technology Trends

- **Chip density** (transistors) is increasing ~2x every 2 years
  - \( \Rightarrow \) number of processors doubles every 2 years as well
- **Clock speed** is plateauing below 10 GHz so that **chip power** stays below 100W
- **Instruction-level parallelism** (ILP) in hardware has also plateaued below 10 instructions/cycle
  - \( \Rightarrow \) **Parallelism must be managed by software!**

Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)
Parallelism Saves Power (Simplified Analysis)

Nowadays (post Dennard Scaling), Power \(\sim (\text{Capacitance}) \times (\text{Voltage})^2 \times (\text{Frequency})\) and maximum Frequency is capped by Voltage

\[\Rightarrow \text{Power is proportional to } (\text{Frequency})^3\]

Baseline example: single 1GHz core with power P

Option A: Increase clock frequency to 2GHz \(\Rightarrow\) Power = 8P

Option B: Use 2 cores at 1 GHz each \(\Rightarrow\) Power = 2P

- Option B delivers same performance as Option A with 4x less power ... provided software can be decomposed to run in parallel!
What is Parallel Programming?

• Specification of operations that can be executed in parallel

• A parallel program is decomposed into sequential subcomputations called tasks

• Parallel programming constructs define task creation, termination, and interaction
Example of a Sequential Program: Computing the sum of array elements

Algorithm 1: Sequential ArraySum

Input: Array of numbers, \( X \).
Output: \( \text{sum} = \text{sum of elements in array } X \).

\[
\begin{align*}
\text{sum} &\leftarrow 0; \\
\text{for } i &\leftarrow 0 \text{ to } X \text{.length } - 1 \text{ do} \\
& \quad \text{sum} \leftarrow \text{sum } + X[i]; \\
\text{return } \text{sum};
\end{align*}
\]

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

Task 0: Compute sum of lower half of array

Task 1: Compute sum of upper half of array

Compute total sum
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:** $sum = \text{sum of elements in array } X$.

// Start of Task T1 (main program)

$sum1 \leftarrow 0; sum2 \leftarrow 0; $

// Compute sum1 (lower half) and sum2 (upper half) in parallel.

finish{
    async{
        // Task T2
        for $i \leftarrow 0 \text{ to } X.length/2 - 1$ do
            $sum1 \leftarrow sum1 + X[i];$
    }
    async{
        // Task T3
        for $i \leftarrow X.length/2 \text{ to } X.length - 1$ do
            $sum2 \leftarrow sum2 + X[i];$
    }
}

// Task T1 waits for Tasks T2 and T3 to complete

// Continuation of Task T1

$sum \leftarrow sum1 + sum2;$

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 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 Java 8 library (no special compiler support needed)
    - HJ-lib contains many features that are easier to use than standard Java threads/tasks, and are also being added to future parallel programming models
  — Later, we will learn parallel programming using standard Java libraries, and combinations of Java libs + HJ-lib
Grade Policies

Course Rubric

- Homeworks (5) 40% (written + programming components)
  - Weightage proportional to # weeks for homework
- Exams (2) 40% (scheduled midterm + scheduled final)
- Labs 10% (labs need to be checked off by Monday)
- Quizzes 5% (on-line quizzes on Canvas)
- Class Participation 5% (in-class worksheets)
Next Steps

• IMPORTANT:
  — Bring your laptop to this week’s lab at 4pm on Thursday (SEW 301)
  — Watch videos for topics 1.2 & 1.3 for next lecture on Wednesday

• HW1 will be assigned on Jan 10th and be due on Jan 24th. (Homework is normally due on Wednesdays.)

• Each quiz (to be taken online on Canvas) will be due on the Friday after the unit is covered in class. The first quiz for Unit 1 (topics 1.1 - 1.5) is due by Jan 26.

• See course web site for syllabus, work assignments, due dates, ...
  • http://comp322.rice.edu
OFFICE HOURS

• Regular office hour schedule can be found at Office Hours link on course web site

• This week’s office hours are Tu/W 4pm - 5pm, Duncan Hall 2071

• Send email to instructors (mjoyner@rice.edu, zoran@rice.edu) if you need to meet some other time this week

• And remember to post questions on Piazza!