LAU
الجامعة اللبنانية الأميركية
Lebanese American University

CSC 447: Parallel Programming for Multi-Core and Cluster Systems

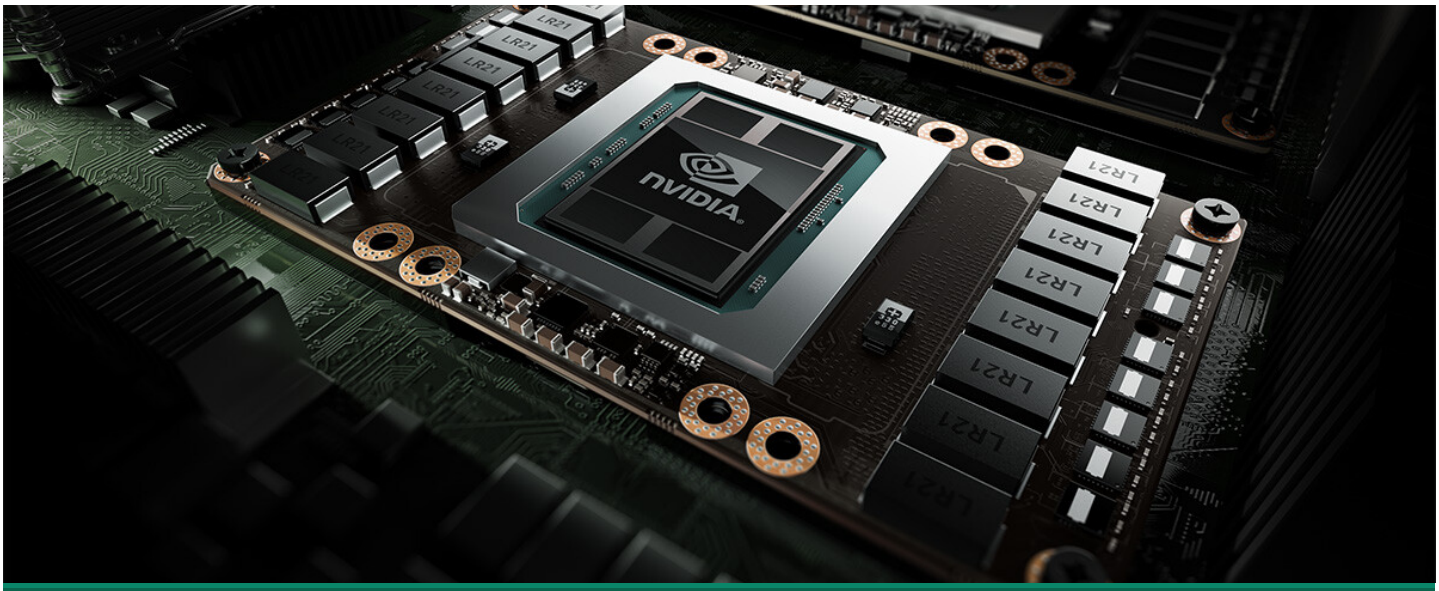
Administrivia and Introduction

Haidar M. Harmanani

Spring 2021

Today's Agenda

- Administrivia
- Course Introduction



Today's Agenda: Administrivia

Course Introduction

- Lectures
  - TTh, 11:00-12:15 from January 18, 2021 until April 29, 2021
  - Prerequisites
    - Competency in a high-level programming language, preferably C
    - CSC 310, Data Structures and Algorithms
    - CSC320, Computer Organization

Grading and Class Policies

- Grading
  - One Midterm: 25%
  - Final: 30%
  - Short Quizzes [4-6]: 10%
  - Short Labs [on your own]: 15%
  - Three milestones projects: 20%
    - OpenMP: 6%
    - OpenACC: 7%
    - CUDA: 7%
- Exams Details
  - Midterm exam will be face to face.
  - Final exam will be online.
- All assignments must be your own original work.
  - Cheating/copying/partnering will not be tolerated

Programming Assignments

- All assignments and handouts will be communicated via `Google Classroom`
  - Make sure you enable your account
- Use `Google Classroom` for questions and inquiries
  - No course questions will be answered via email
  - Use email only for private issues
- All assignments must be submitted via `github`
  - `git` is a distributed version control system
  - `git` or its variations have become a universal standard for developing and sharing code
  - Make sure you get a private repo
    - Apply for a free account: https://education.github.com/discount_requests/new

Policy on Independent Work

- With the exception of laboratories and assignments (projects and HW) that explicitly permit you to work in groups, all homework and projects are to be YOUR work and your work ALONE.
- It is NOT acceptable to copy solutions from other students.
- It is NOT acceptable to copy (or start your) solutions from the Web.
- PARTNER TEAMS MAY NOT WORK WITH OTHER PARTNER TEAMS
- You are encouraged to help teach other to debug. Beyond that, we don't want you sharing approaches or ideas or code or whiteboarding with other students, since sometimes the point of the assignment is the "algorithm" and if you share that, they won't learn what we want them to learn). We expect that what you hand in is yours.
- It is NOT acceptable to leave your code anywhere where an unscrupulous student could find and steal it (e.g., public GITHUBs, walking away while leaving yourself logged on, leaving printouts lying around,etc)
- The first offense is a zero on the assignment and an F in the course the second time
- Both Giver and Receiver are equally culpable and suffer equal penalties

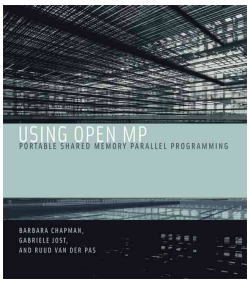
Contact Information

- Haidar M. Harmanani
  - Office: Block A, 810
  - Hours: M 4:00-6:30pm or by appointment.
  - Email: haidar@lau.edu.lb

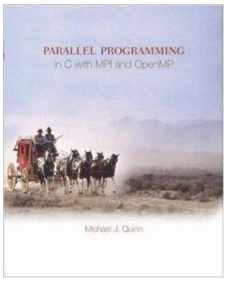
Topics and Tools

- Introduction to parallel programming
- Parallel programming performance
- Programming using `Pthreads`, `OpenMP`, `OpenACC`, and `CUDA`
- Introduction to Neural Networks and Deep Learning using `Python`
- You will be using:
  - The lab or your own machines for `Pthreads` and `OpenMP`
  - The Computer Science Lab for `OpenACC` and `CUDA` Programming, or your own machine if you have a CUDA-capable GPU
  - The *Cloud* for GPU Tutorial Labs

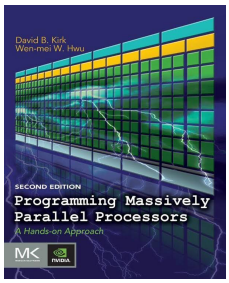
Course Materials



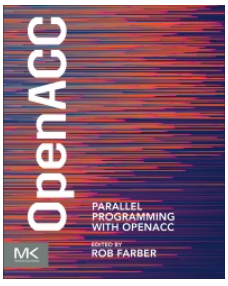
Optional



Optional



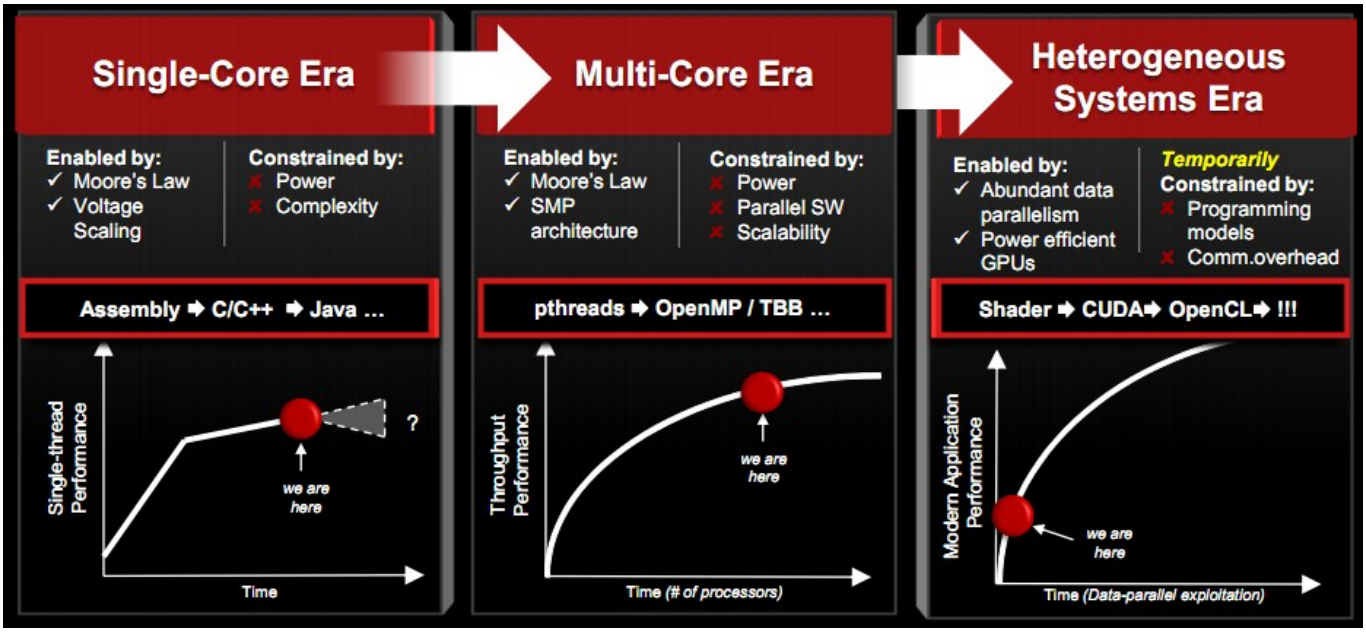
Required



Optional

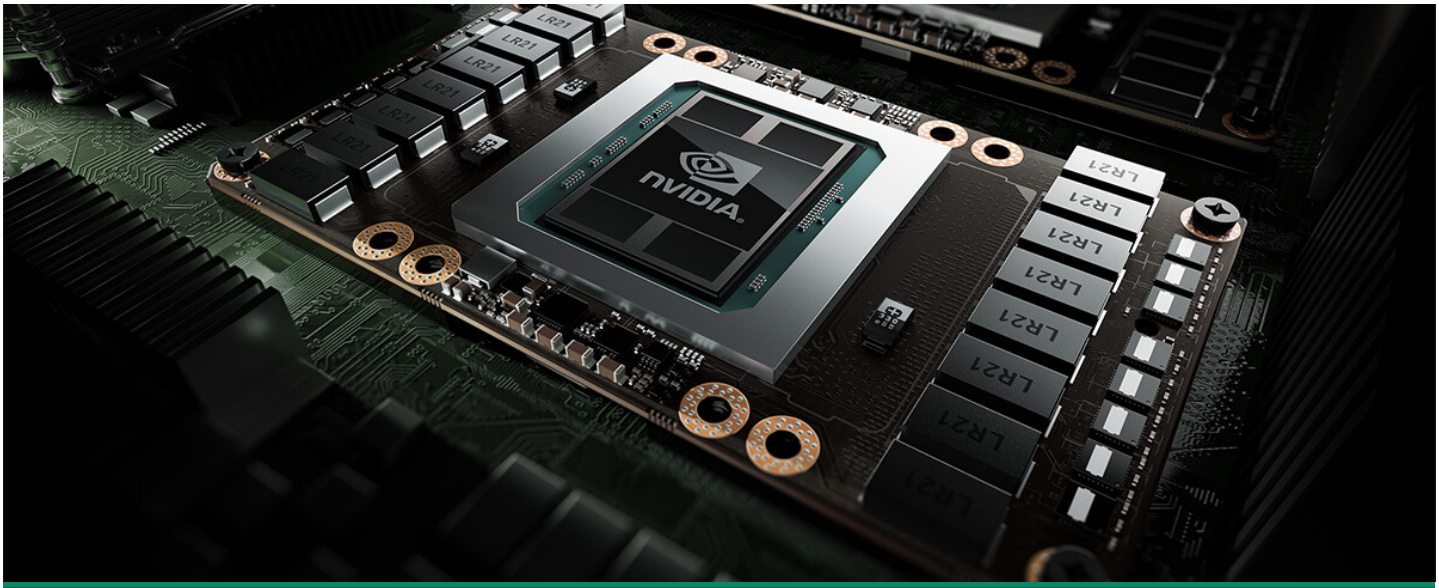
Supercomputing/Parallel Architecture Timeline

- Phase 1 (1950s): sequential instruction execution
- Phase 2 (1960s): sequential instruction issue
  - Pipeline execution, reservations stations
  - Instruction Level Parallelism (ILP)
- Phase 3 (1970s): vector processors
  - Pipelined arithmetic units
  - Registers, multi-bank (parallel) memory systems
- Phase 4 (1980s): SIMD and SMPs
- Phase 5 (1990s): MPPs and clusters
  - Communicating sequential processors
- Phase 6 (>2000): many cores, accelerators, scale, …



LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Administrative Questions?



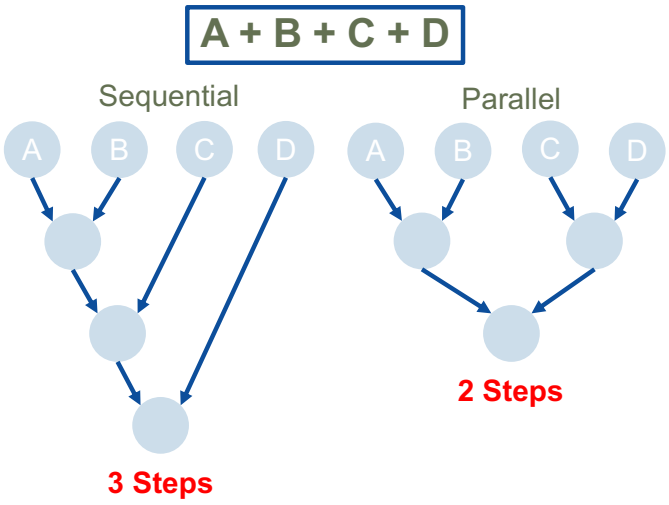
Today's Agenda: Course Introduction

Definitions

- What is parallel?
  - Webster: *"An arrangement or state that permits several operations or tasks to be performed simultaneously rather than consecutively"*
- Parallel programming
  - Programming in a language that supports *explicit* concurrency
  - An evolution of serial programming

Parallel Programming

- "Performance Programming"
- Parallel programming involves exposing an algorithm's ability to execute in parallel
- This may involve breaking a large operation into smaller tasks (task parallelism)
- Or doing the same operation on multiple data elements (data parallelism)
- Parallel execution enables better performance on modern hardware



Parallel Programming

- A parallel computer is a computer system that uses multiple processing elements simultaneously in a cooperative manner to solve a computational problem
- Parallelism is all about performance! Really?

Concurrent Programming

- Concurrency is fundamental to computer science
  - Operating systems, databases, networking, …
- Multiple executing tasks are concurrent with respect to each other if
  - They can execute asynchronously
  - Implies that there are no dependencies between the tasks

Concurrent Programming

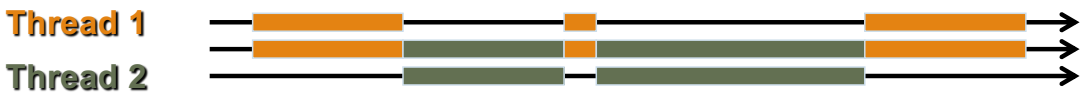
- Two threads can run *asynchronously* on the same core by interleaving instructions
- Dependencies
  - If a task requires results produced by other tasks in order to execute correctly, the task's execution is *dependent*
  - Some form of synchronization must be used to enforce (satisfy) dependencies

Concurrency and Parallelism

- Concurrent is not the same as parallel!
- Parallel execution
  - Concurrent tasks *actually* execute at the same time
  - Multiple (processing) resources have to be available
- Parallelism = concurrency + "parallel" hardware
  - Both are required
  - Find concurrent execution opportunities
  - Develop application to execute in parallel
  - Run application on parallel hardware

Concurrency vs. Parallelism

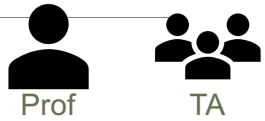
– Concurrency: two or more threads are in progress at the same time:



– Parallelism: two or more threads are executing at the same time

Parallel Programming: Example

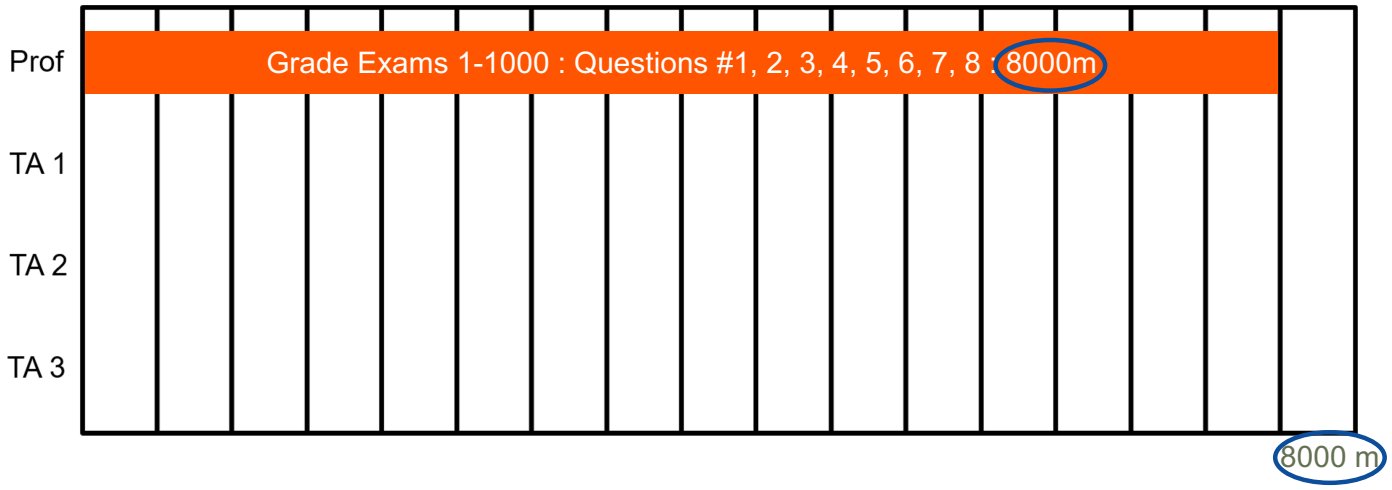
- A professor and his 3 teaching assistants (TA) are grading 1,000 student exams
- This exam has 8 questions on it
- Let's assume it takes 1 minute to grade 1 question on 1 exam
- To maintain fairness, if someone grades a question (for example, question #1) then they must grade that question on all other exams
- The following is a sequential version of exam grading



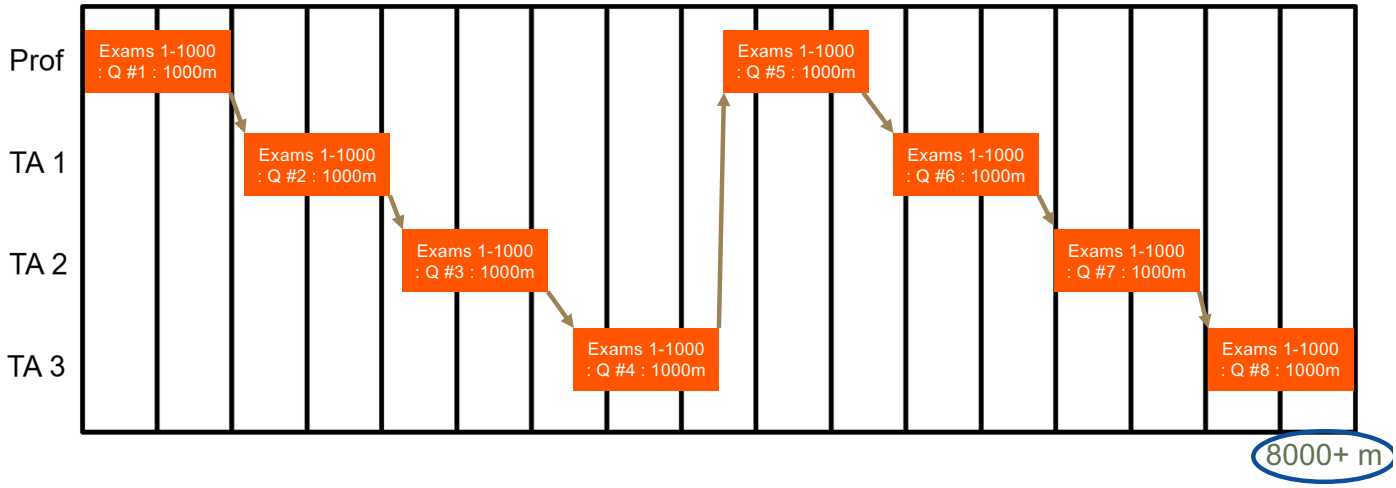
8 questions per exam
8,000 questions in total

1 minute per question

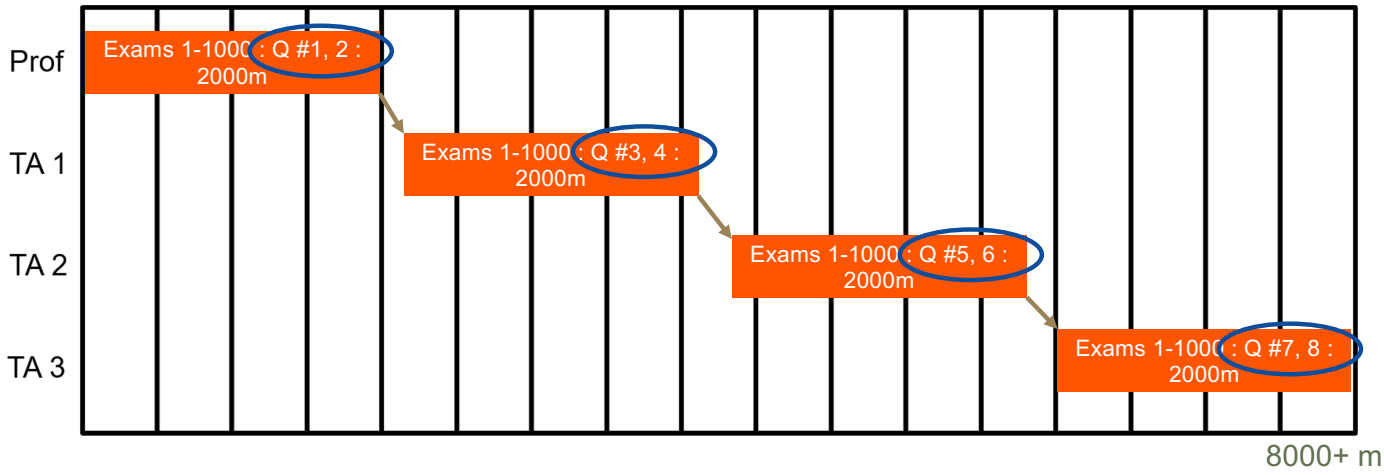
Sequential Solution



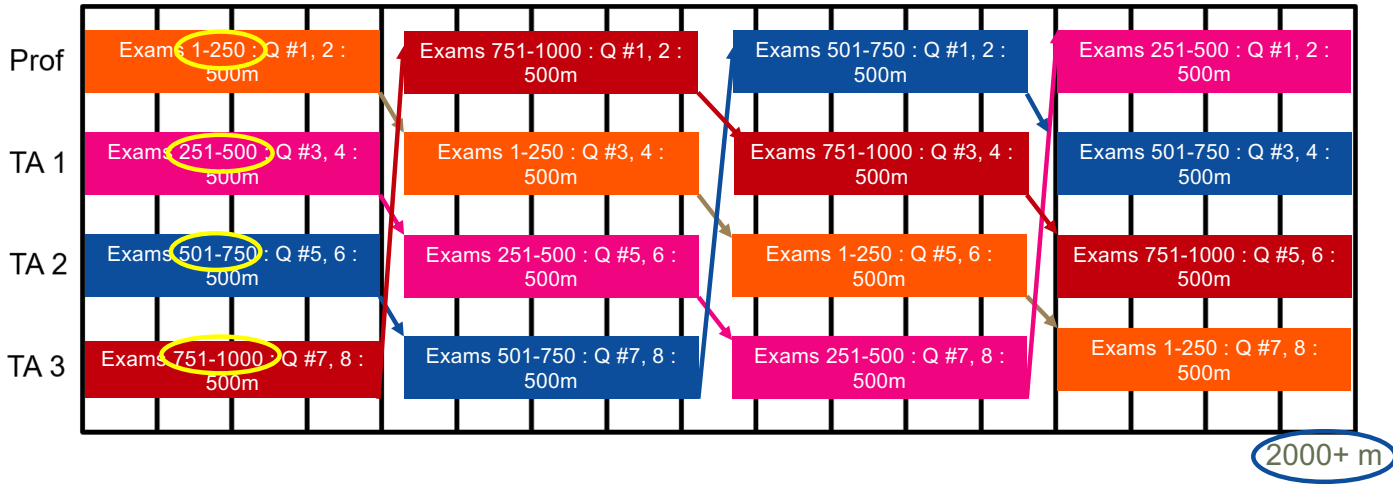
Sequential Solution



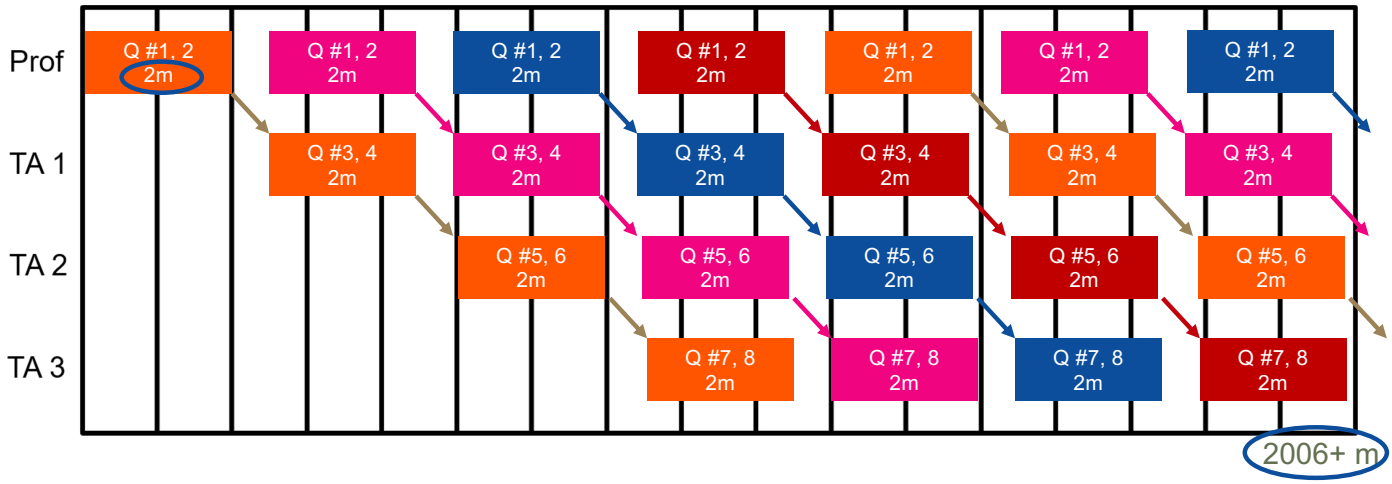
Sequential Solution



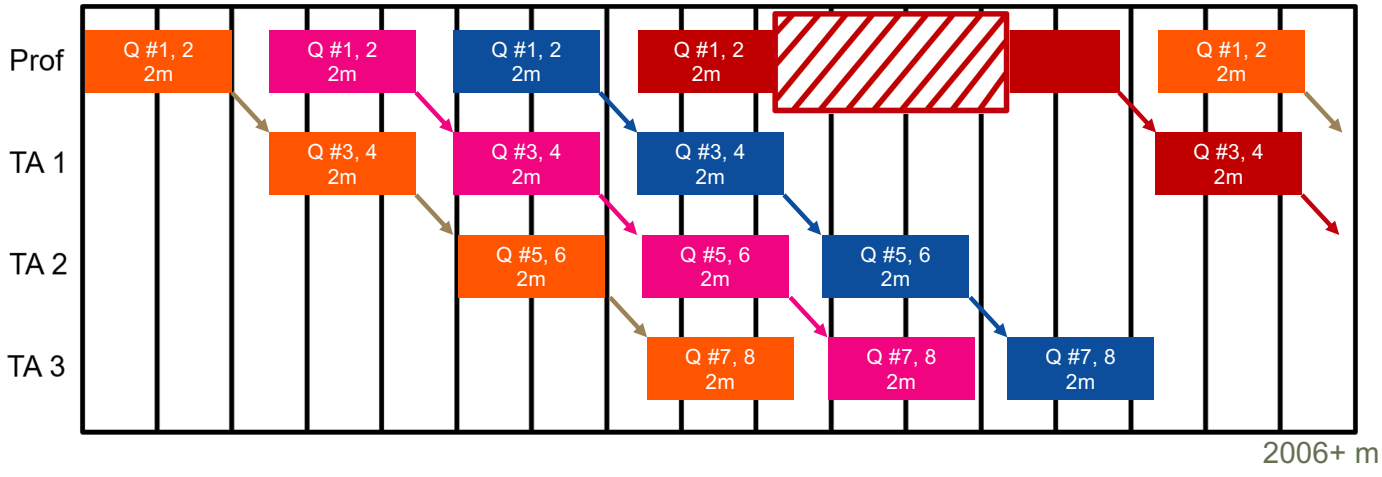
Parallel Solution



Pipelined Solution [CSC 320]

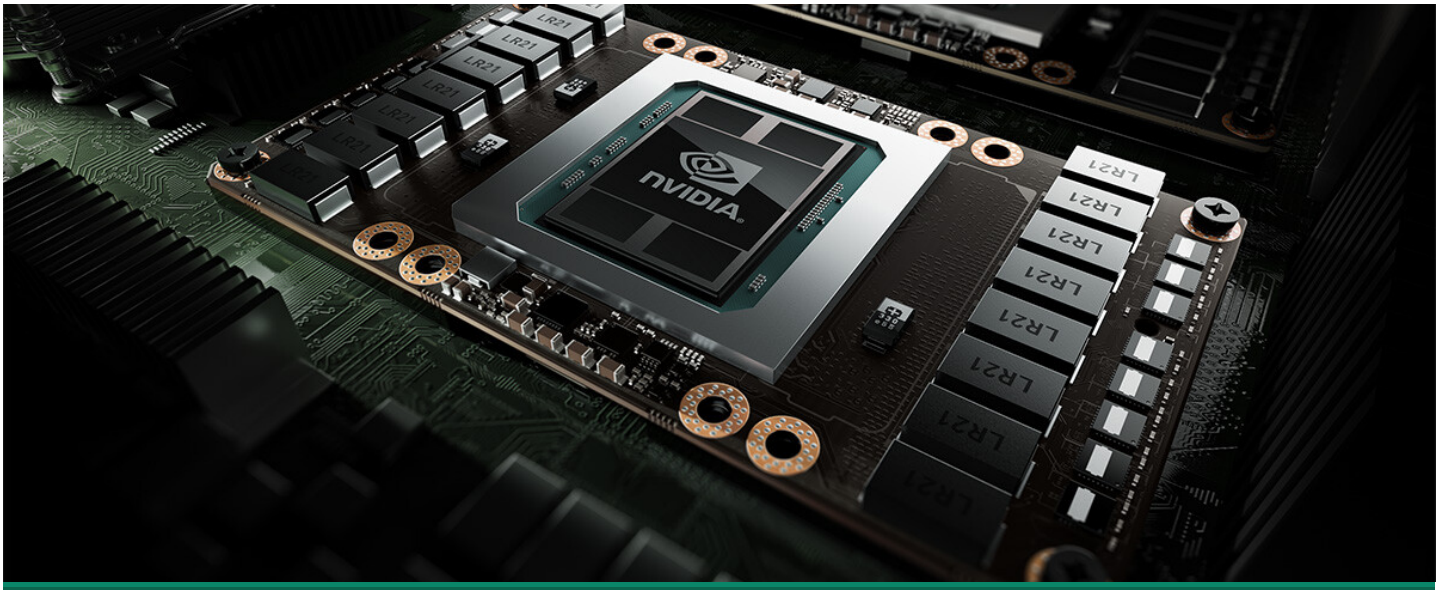


Pipelined Solution: Stall



Grading Example Summary

- It's critical to understand the problem before trying to parallelize it
  - Can the work be done in an arbitrary order, or must it be done in sequential order?
  - Does each task take the same amount of time to complete? If not, it may be necessary to "load balance."
  - In our example, the only restriction is that a single question be graded by a single grader, so we could divide the work easily, but had to communicate periodically.
  - This case study is an example of task-based parallelism. Each grader is assigned a task like "Grade questions 1 & 2 on the first 500 tests"
  - If instead each question could be graded by different graders, then we could have data parallelism: all graders work on Q1 of the following tests, then Q2, etc.



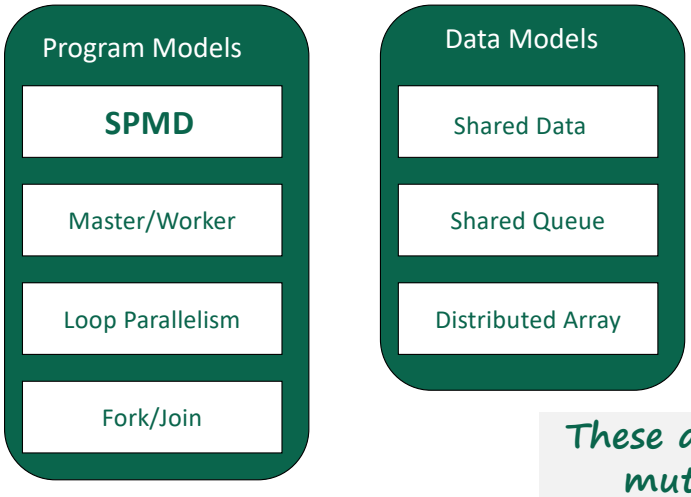
Computational Thinking

Fundamentals of Parallel Computing

- Parallel computing requires that
  - The problem can be decomposed into sub-problems that can be safely solved at the same time
  - The programmer structures the code and data to solve these sub-problems concurrently
- The goals of parallel computing are
  - To solve problems in less time (strong scaling), and/or
  - To solve bigger problems (weak scaling), and/or
  - To achieve better solutions (advancing science)

The problems must be large enough to justify parallel computing and to exhibit exploitable concurrency.

Parallel Programming Coding Styles – Program and Data Models



These are not necessarily mutually exclusive.

Program Models

- SPMD (Single Program, Multiple Data)
  - All PE's (Processor Elements) execute the same program in parallel, but has its own data
  - Each PE uses a unique ID to access its portion of data
  - Different PE can follow different paths through the same code
  - This is essentially the CUDA Grid model (also OpenCL, MPI)
  - SIMD is a special case – WARP used for efficiency
- Master/Worker
- Loop Parallelism
- Fork/Join

Program Models

- SPMD (Single Program, Multiple Data)
- Master/Worker (OpenMP, OpenACC, TBB)
  - A Master thread sets up a pool of worker threads and a bag of tasks
  - Workers execute concurrently, removing tasks until done
- Loop Parallelism (OpenMP, OpenACC, C++AMP)
  - Loop iterations execute in parallel
  - FORTRAN do-all (truly parallel), do-across (with dependence)
- Fork/Join (Posix p-threads)
  - Most general, generic way of creation of threads

More on SPMD

- Dominant coding style of scalable parallel computing
  - MPI code is mostly developed in SPMD style
  - Many OpenMP code is also in SPMD (next to loop parallelism)
  - Particularly suitable for algorithms based on task parallelism and geometric decomposition.
- Main advantage
  - Tasks and their interactions visible in one piece of source code, no need to correlated multiple sources

SPMD is by far the most commonly used pattern for structuring massively parallel programs.

Typical SPMD Program Phases

- Initialize
  - Establish localized data structure and communication channels
- Obtain a unique identifier
  - Each thread acquires a unique identifier, typically range from 0 to N-1, where N is the number of threads.
  - Both OpenMP and CUDA have built-in support for this.
- Distribute Data
  - Decompose global data into chunks and localize them, or
  - Sharing/replicating major data structure using thread ID to associate subset of the data to threads
- Run the core computation
  - More details in next slide…
- Finalize
  - Reconcile global data structure, prepare for the next major iteration