INTRODUCTION

Jerry Ebalunode HPEDSI

https://hpedsi.uh.edu/ University of Houston

Access Your Account

- UHVPN connection may be required if you are not on campus network

- Make sure that you are added to the classroom cluster access
- If you have confirmed enrollment then you should have access
- Ask the instructor if you have trouble

ssh -l username aerb202.cacds.e.uh.edu -XY

- Log into your accounts
- Username or login = Cougarnet ID
- **Password = Cougarnet password**

Prerequisites

- Participants are expected to have familiarity with a low level programming language such as C/C++, or Fortran, Matlab and working comfortably in a UNIX/Linux environment or completed corresponding tutorials.
- Recommended HPE DSI courses –
  - Introduction to cluster computing,
  - C/C++ or Fortran,
- *To earn credit for the course, participants are expected to have passed the "introduction to cluster computing course" or passed the "placement test" before this course ends.*

OpenACC
Directives for Accelerators

Accessing Tutorial Materials

Will be made available via HPE DSI training website to students who attend the class.
www.hpedsi.uh.edu/training

MODULE OVERVIEW

Topics to be covered

Introduction to parallel programming

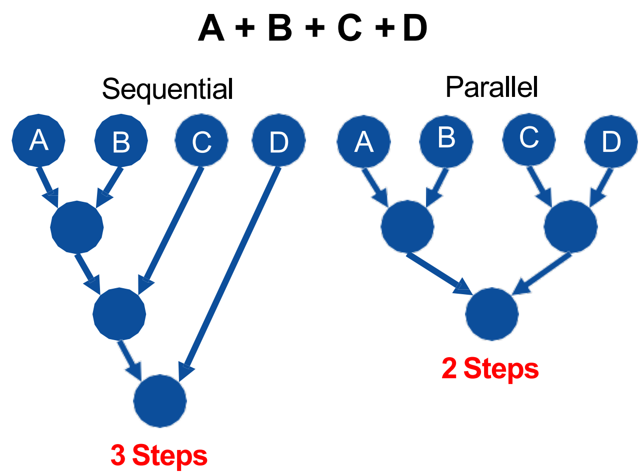
Common difficulties in parallel programming

INTRODUCTION TO PARALLEL PROGRAMMING

WHAT IS PARALLEL PROGRAMMING?

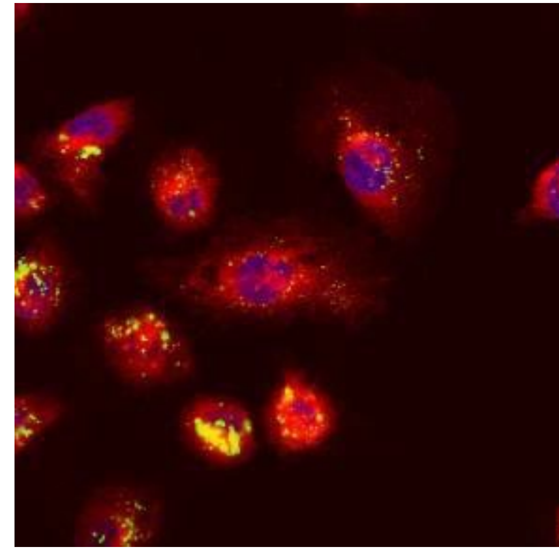
"Performance Programming"

- 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



OpenACC
Directives for Accelerators

A REAL WORLD CASE STUDY



Modern cancer research

- The Russian Academy of Science created a program to simulate light propagation through human tissue
- This program was used to more accurately detect cancerous cells by simulating **billions** of random paths that the light could take through human tissue
- With parallel programming, they were able to run **thousands** of these paths **simultaneously**
- The sequential program took **2.5 hours** to run
- The parallel version took less than **2 minutes**

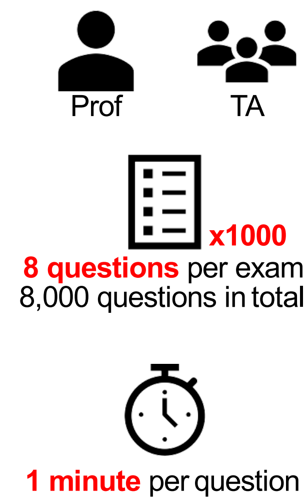
Parallel Computing Illuminating a Path to Early Cancer Detection

OpenACC
Directives for Accelerators

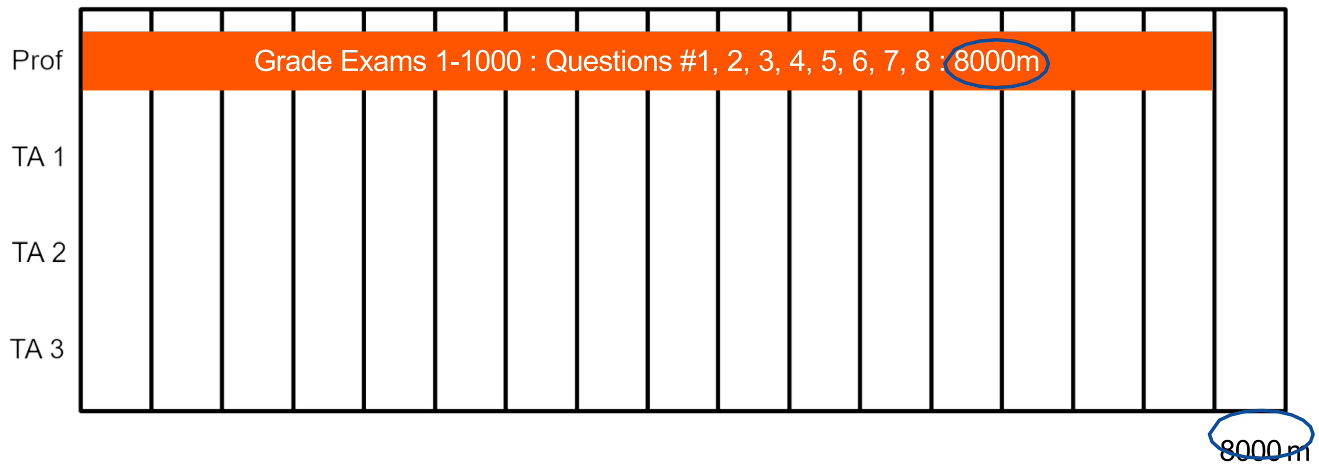
WHAT IS PARALLEL PROGRAMMING?

A real world example

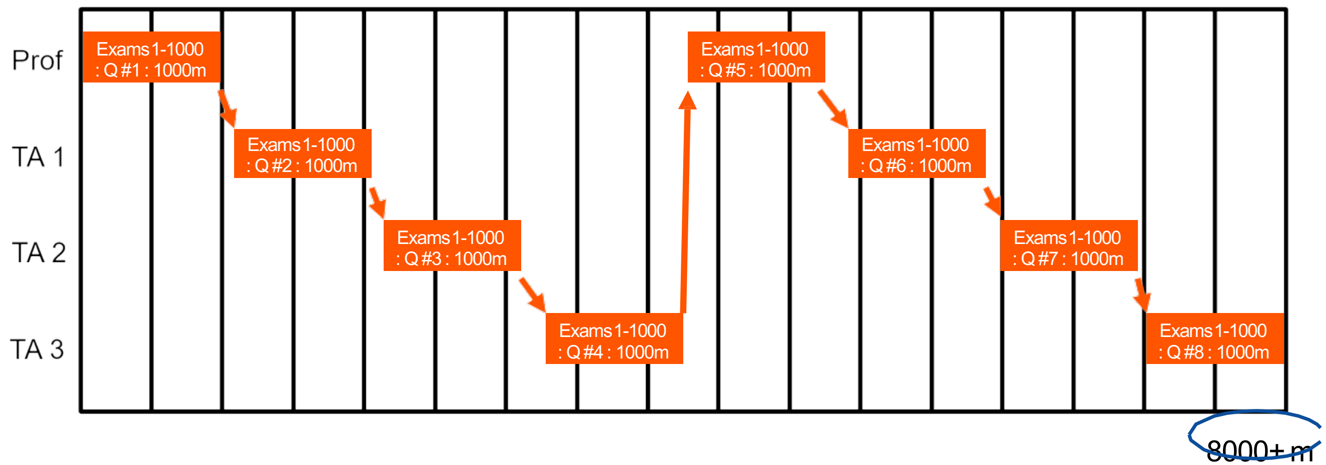
- A professor and his 3 teaching assistants (TA) are grading 1,000 student exams
- This exam has 8 questions on it
- Let’s assume it takes 1 minute to grade 1 question on 1exam
- To maintain fairness, if someone grades a question (for example, question #1) then they must grade that question on all other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



SEQUENTIAL SOLUTION

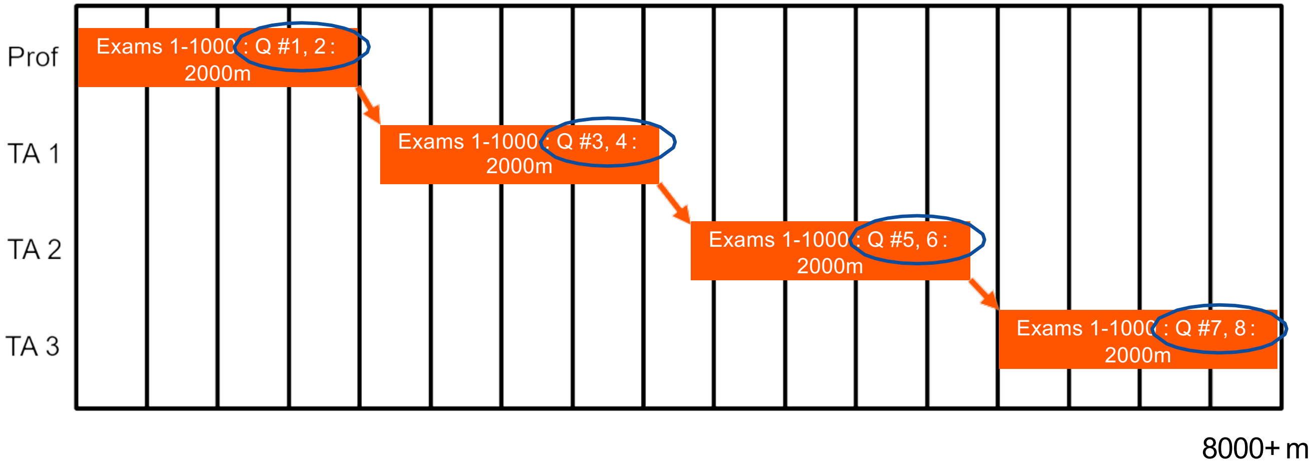


SEQUENTIAL SOLUTION

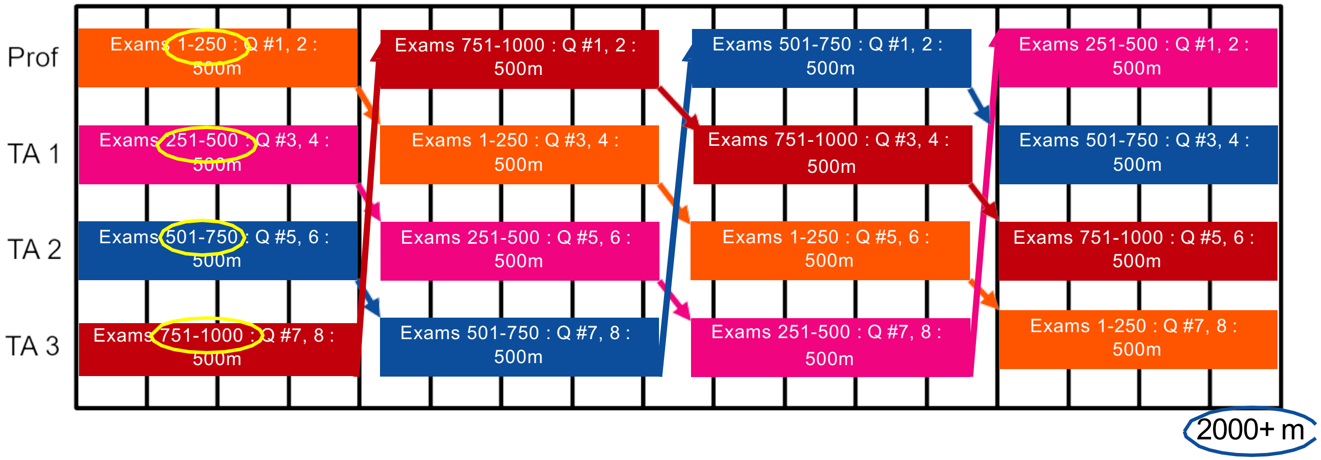


OpenACC
Directives for Accelerators

SEQUENTIAL SOLUTION

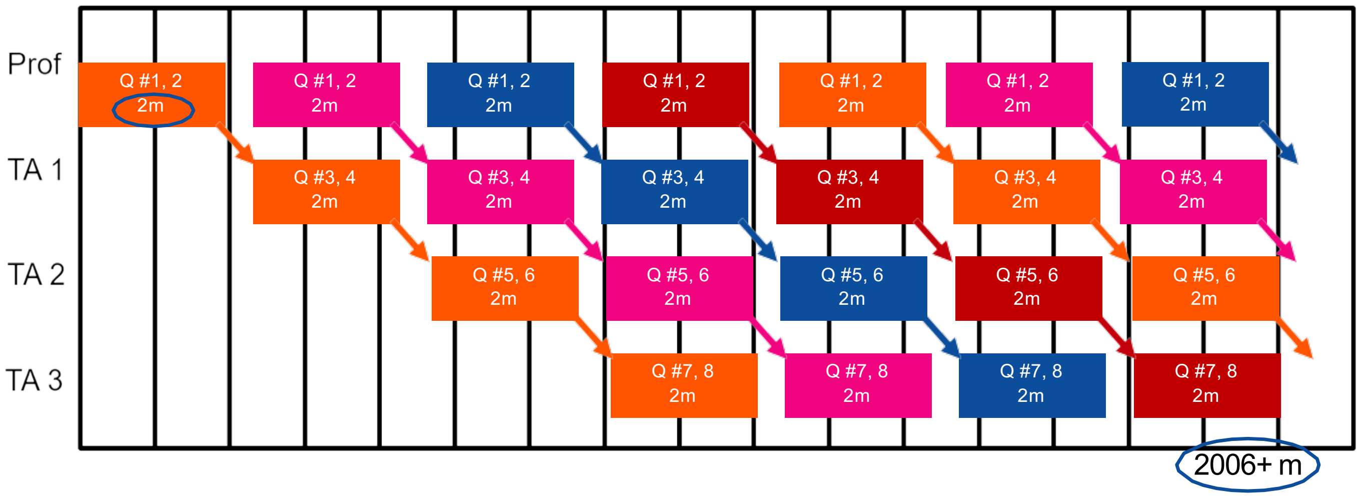


PARALLEL SOLUTION

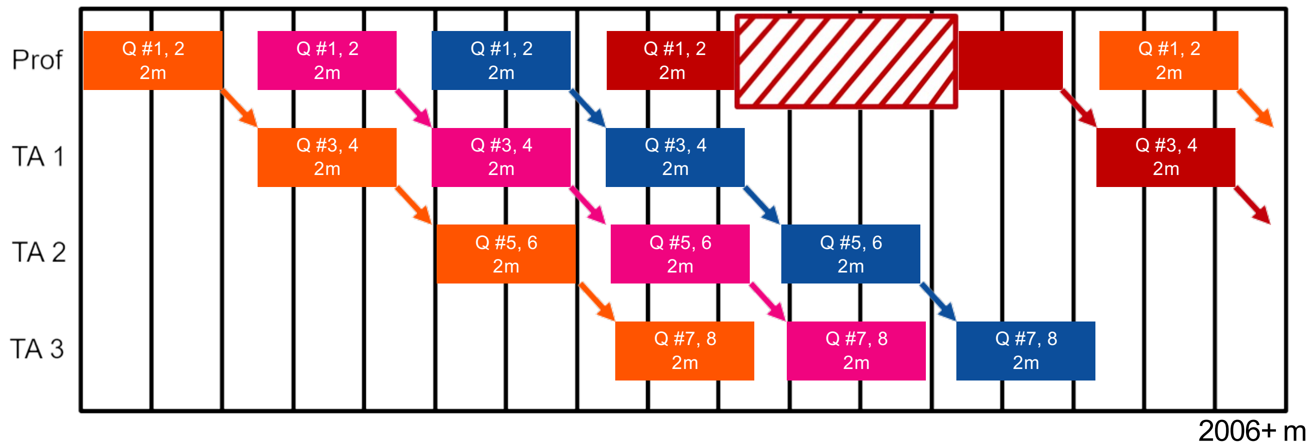


OpenACC Directives for Accelerators

PIPELINE


Prof
Q #1, 2
2m
Q #1, 2
2m
Q #1, 2
2m
Q #1, 2
2m
Q #1, 2
2m
Q #1, 2
2m
Q #1, 2
2m
TA 1
Q #3, 4
2m
Q #3, 4
2m
Q #3, 4
2m
Q #3, 4
2m
Q #3, 4
2m
Q #3, 4
2m
TA 2
Q #5, 6
2m
Q #5, 6
2m
Q #5, 6
2m
Q #5, 6
2m
Q #5, 6
2m
TA 3
Q #7, 8
2m
Q #7, 8
2m
Q #7, 8
2m
Q #7, 8
2m
2006+ m

PIPELINE STALL



GRADING EXAMPLE SUMMARY

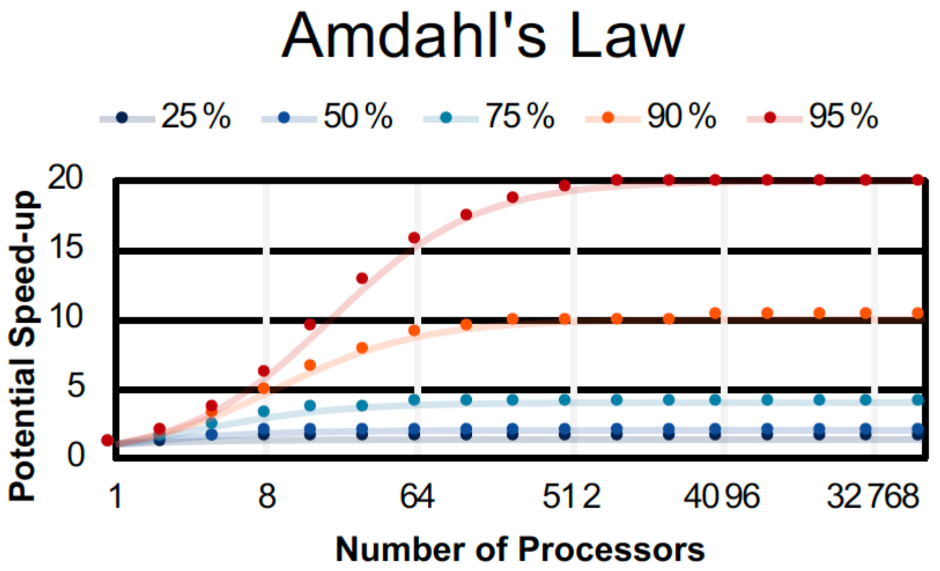
- It's critical to understand the problem before trying to parallelize it
- Can the work be done in an arbitrary order, or must it be done in sequential order?
- Does each task take the same amount of time to complete? If not, it may be necessary to *"load balance."*
- In our example, the only restriction 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 atask 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.

AMDAHL’S LAW

AMDAHL'S LAW

Serialization Limits Performance

- Amdahl's law is an observation that how much speed-up you get from parallelizing the code is limited by the remaining serial part.
- Any remaining serial code will reduce the possible speed-up
- This is why it's important to focus on parallelizing the most time consuming parts, not just the easiest.

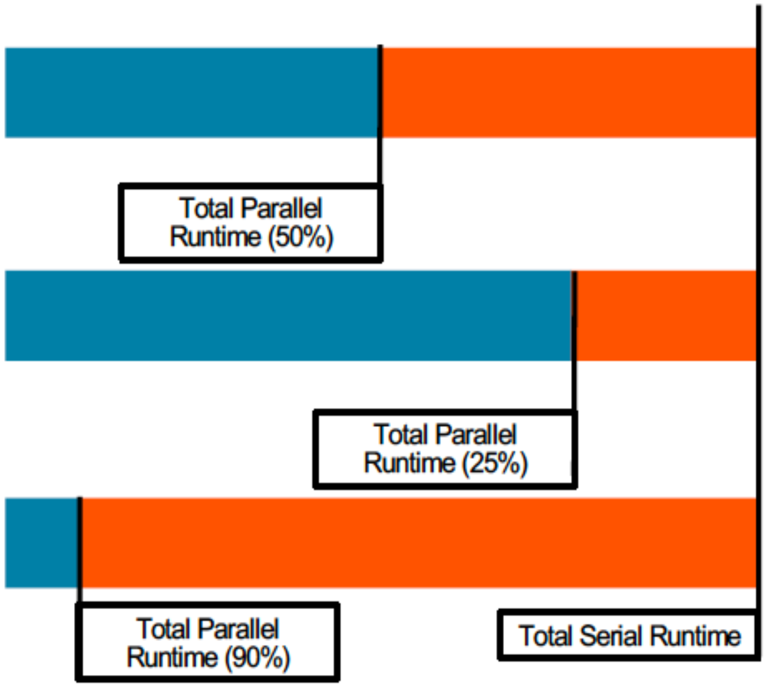


APPLYING AMDAHL'S LAW

Estimating Potential Speed-up

- What's the maximum speed-up that can be obtained by parallelizing 50% of the code?
  - (1 / 100% - 50%) = (1 / 1.0 - 0.50) = 2.0X
- What's the maximum speed-up that can be obtained by parallelizing 25% of the code?
  - (1 / 100% - 25%) = (1 / 1.0 - 0.25) = 1.3X
- What's the maximum speed-up that can be obtained by parallelizing 90% of the code?
  - (1 / 100% - 90%) = (1 / 1.0 - 0.90) = 10.0X

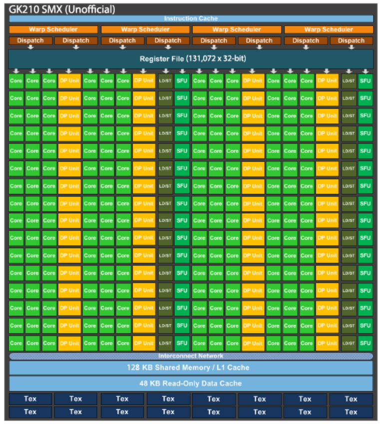
Maximum Parallel Speed-up



GRAPHICAL PROCESSING UNIT (GPU)

GRAPHICAL PROCESSING UNIT (GPU)

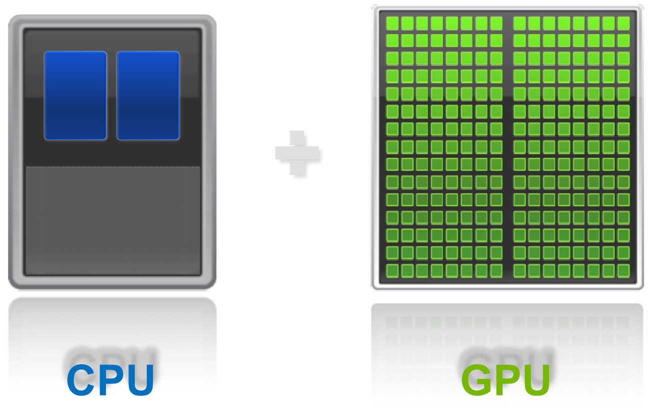
NVIDIA GPU



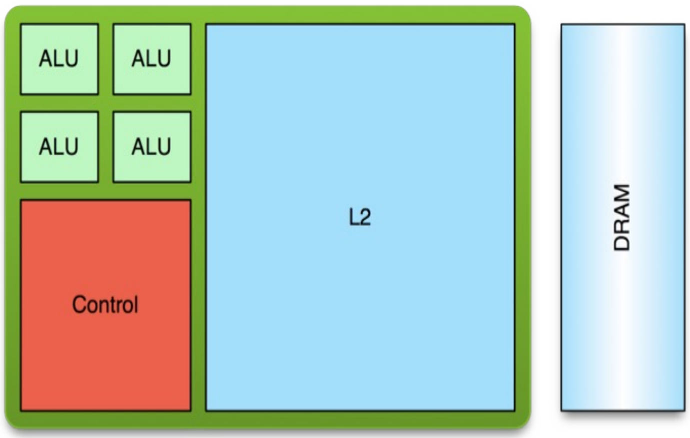
http://www.anandtech.com/show/8729/nvidia-launches-tesla-k80-gk210-gpu

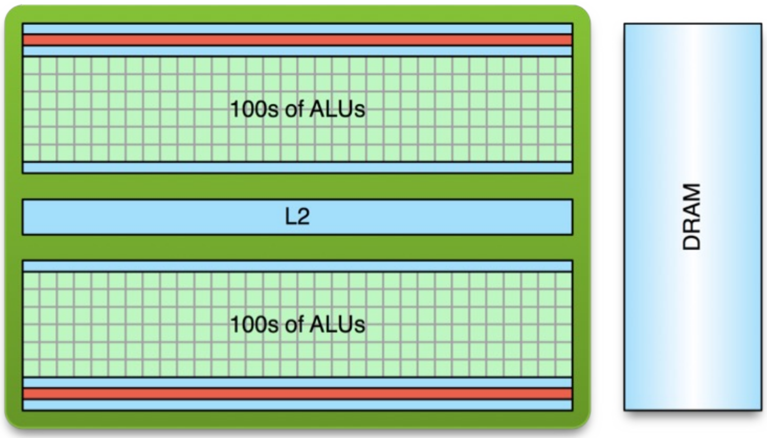
GPGPU Revolutionizes Computing

Latency Processor + Throughput processor



LOW LATENCY OR HIGH THROUGHPUT?





CPU

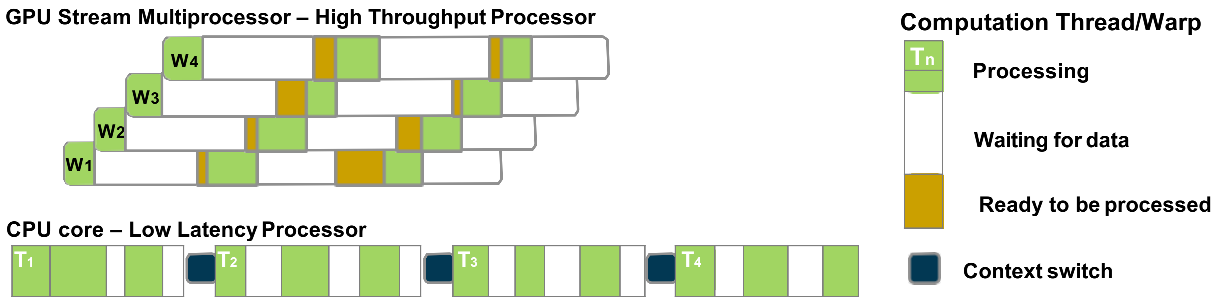
- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution

GPU

- Optimized for data-parallel, throughput computation
- Architecture tolerant of memory latency
- More transistors dedicated to computation

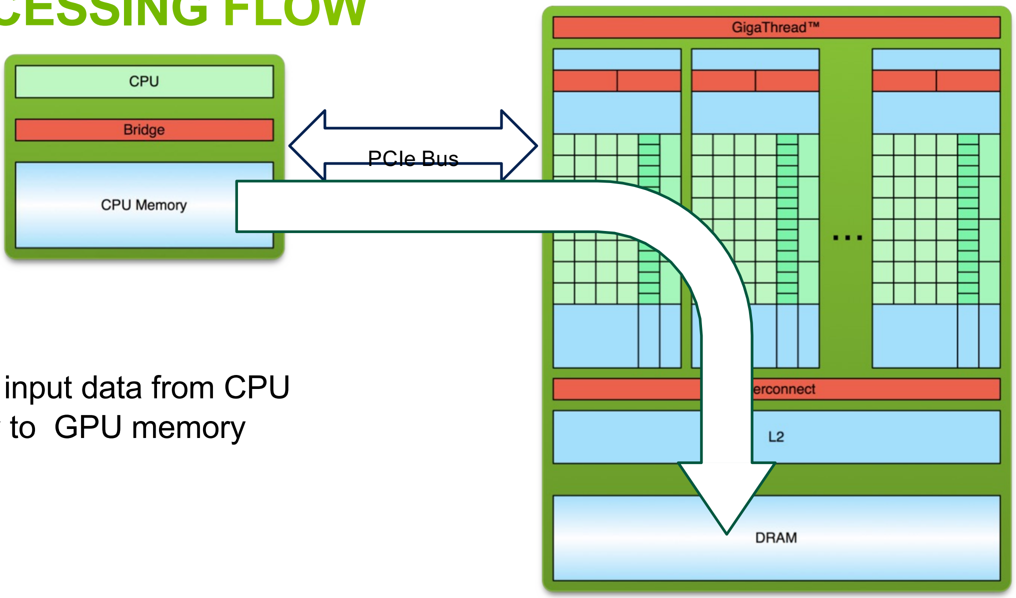
LOW LATENCY OR HIGH THROUGHPUT?

- CPU architecture must minimize latency within each thread
- GPU architecture hides latency with computation from other threads



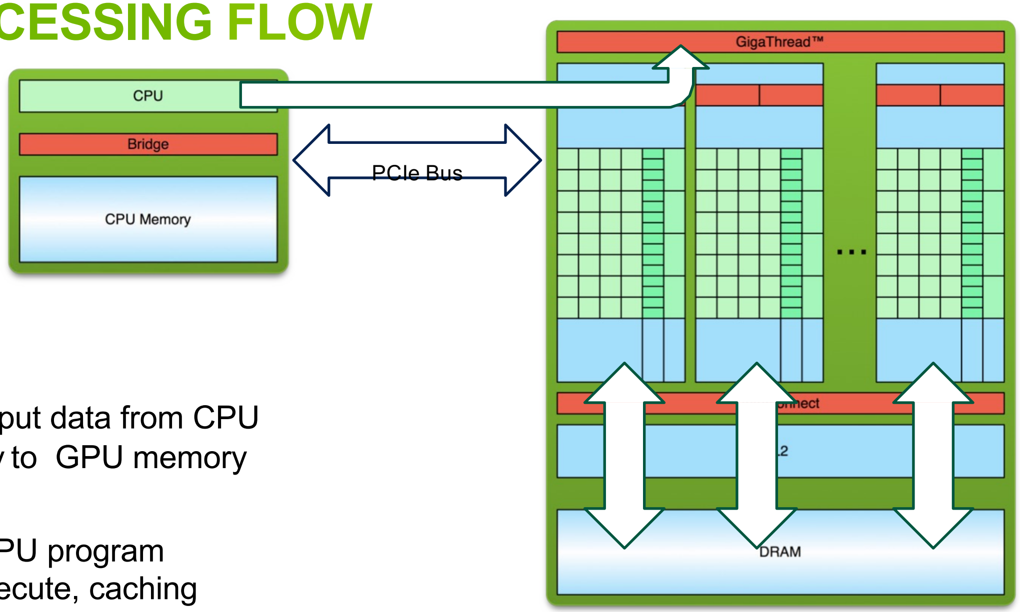
HOW TO PROGRAM A GPU

PROCESSING FLOW



1. Copy input data from CPU memory to GPU memory

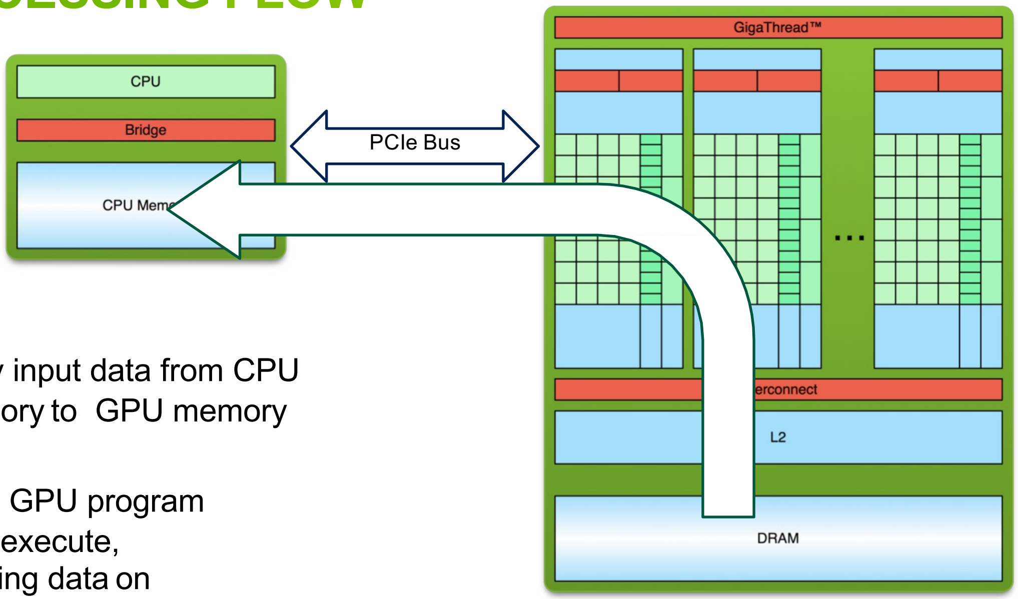
PROCESSING FLOW



Copy input data from CPU memory to GPU memory

Load GPU program and execute, caching data on chip for performance

PROCESSING FLOW

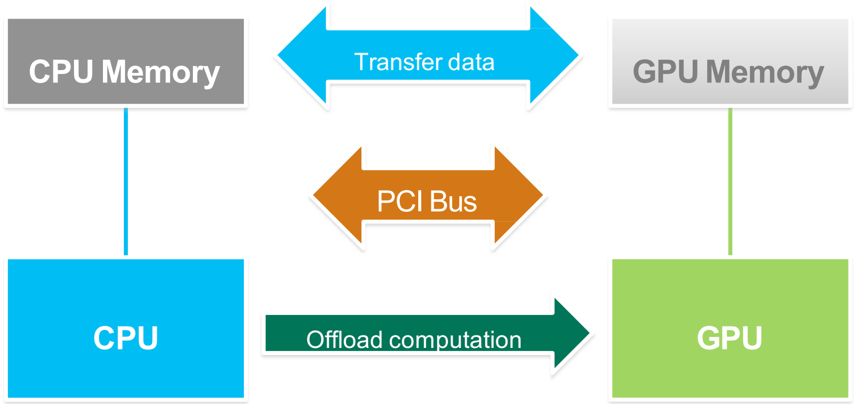


1. Copy input data from CPU memory to GPU memory

2. Load GPU program and execute, caching data on chip for performance

3. Copy results from GPU memory back to CPU memory

BASIC CONCEPTS



For efficiency, decouple data movement and compute off-load

3 Ways to Accelerate Applications

Applications

Libraries	Compiler Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility
Thrust/cuBLAS	OpenACC/OpenMP	CUDA/OpenCL

CLOSING SUMMARY

Introduction

- Parallel programming is the only way to fully utilize modern, parallel hardware
- The key idea parallel programming is split up tasks within a program in a way that they can be run in parallel (at the same time)
- Parallel programming requires the programmer to understand their program and which operations can be performed in parallel and the communicate that to the system
- GPUs can be used for parallel processing

Access Your Account

- UHVPN connection may be required if you are not on campus network

- Make sure that you are added to the classroom cluster access
- If you have confirmed enrollment then you should have access
- Ask the instructor if you have trouble

ssh -XY -l username aerb202.cacds.e.uh.edu

- Log into your accounts
- Username or login = Cougarnet ID
- **Password = Cougarnet password**

OpenACC
Directives for Accelerators