

4-12-2019

# Graphics Processing Units (GPUs) and CUDA

Josiah Brett  
*Hope College*

Josiah Brouwer  
*Hope College*

Follow this and additional works at: [https://digitalcommons.hope.edu/curca\\_18](https://digitalcommons.hope.edu/curca_18)

 Part of the [Systems Architecture Commons](#)

---

## Recommended Citation

**Repository citation:** Brett, Josiah and Brouwer, Josiah, "Graphics Processing Units (GPUs) and CUDA" (2019). *18th Annual Celebration of Undergraduate Research and Creative Activity (2019)*. Paper 29.  
[https://digitalcommons.hope.edu/curca\\_18/29](https://digitalcommons.hope.edu/curca_18/29)  
April 12, 2019. Copyright © 2019 Hope College, Holland, Michigan.

This Poster is brought to you for free and open access by the Celebration of Undergraduate Research and Creative Activity at Hope College Digital Commons. It has been accepted for inclusion in 18th Annual Celebration of Undergraduate Research and Creative Activity (2019) by an authorized administrator of Hope College Digital Commons. For more information, please contact [digitalcommons@hope.edu](mailto:digitalcommons@hope.edu).

# Graphics Processing Units (GPUs) and CUDA

Josiah Brett, Josiah Brouwer, Dr. Charles Cusack (Advisor)

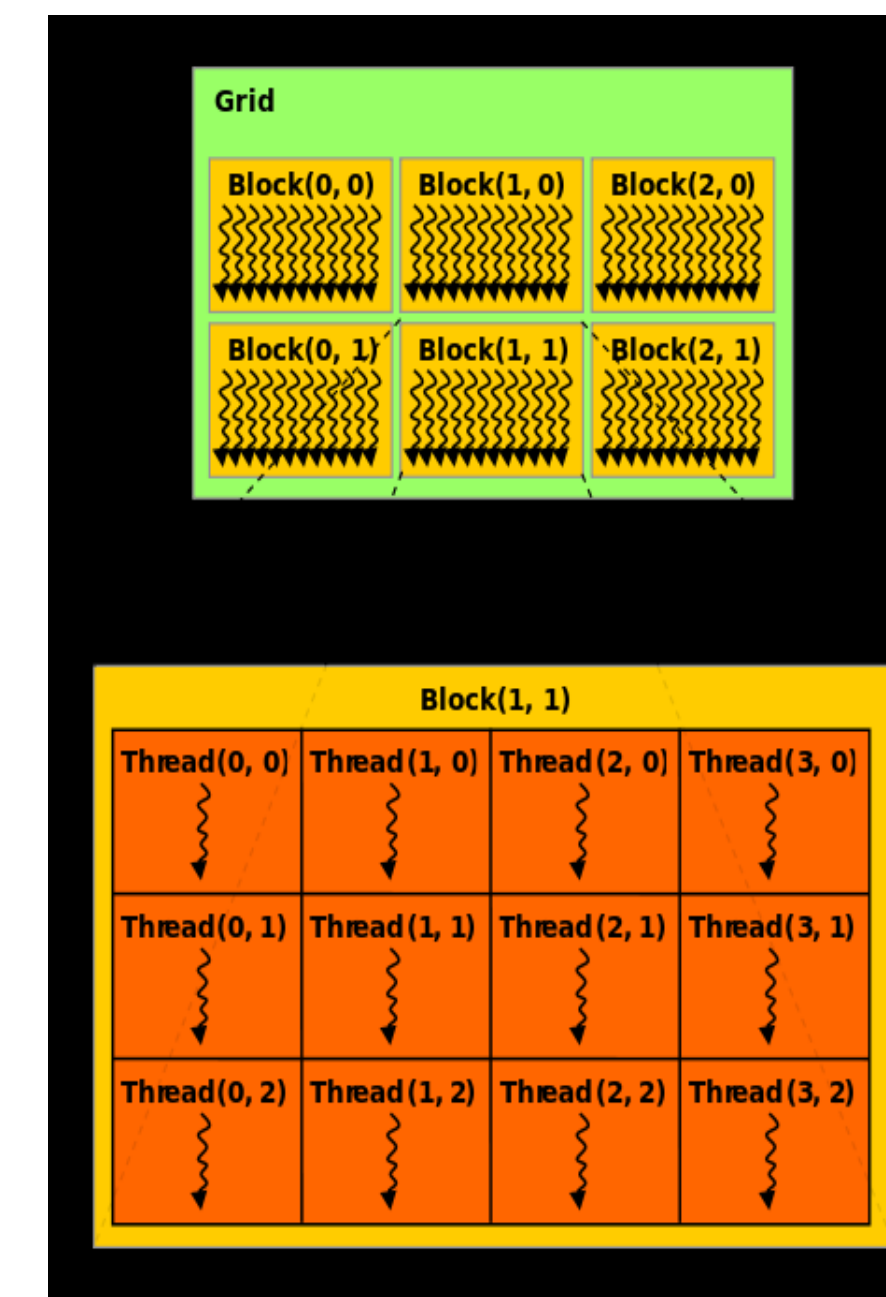
Department of Computer Science, Hope College

## Abstract

Computers almost always contain one or more central processing units (CPU), each of which processes information sequentially. While having multiple CPUs allow a computer to run several tasks in parallel, many computers also have a graphics processing unit (GPU) which contains hundreds to thousands of cores that allow it to execute many computations in parallel. In order to complete a larger task, GPUs run many subtasks concurrently. Each core performs the same instruction on different sets of data, making it useful for performing tasks such as calculating what each individual pixel displays on a screen. The purpose of this research was to learn how GPUs work, how to write CUDA programs to utilize GPUs, and to determine if GPUs could be used to increase the speed of algorithms used to determine the pebbling properties of graphs. In addition, we developed a class module on GPU computing with CUDA for the Advanced Algorithms class in Hope College's Computer Science department.

## Threads, Blocks, and Grids

A thread is the smallest unit of execution on a GPU, and executes a single process. Many threads can be run concurrently. Threads are organized into blocks, which are, in turn, organized into a grid. Blocks and threads are organized using three-dimensional indexing



By NVIDIA - NVIDIA CUDA Programming Guide version 3.0, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=17625645>

## Kernels

Kernels are functions that are executed on the GPU. When a kernel is launched, the number of blocks per thread and the number of blocks are specified. Thread and block index can be determined from within a kernel, allowing different threads to do slightly different tasks despite executing the same function. It is worth noting that kernels can be called from within other kernels.

```
__global__ void add(int *a, int *b, int *c, int numElements) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if(idx < numElements){
        c[idx] = a[idx] + b[idx];
    }
}
```

## Memory

The GPU's memory is independent of the CPU's. In algorithms that use a GPU, data must be transferred from the CPU to the GPU before the GPU can perform computations using it. Afterwards, the result data must be transferred to the CPU. Because of the relatively large amount of time it takes to transfer data, the increase in speed from the GPU's parallel computations must compensate for the time it takes for memory to be transferred if the GPU is to be worth using.

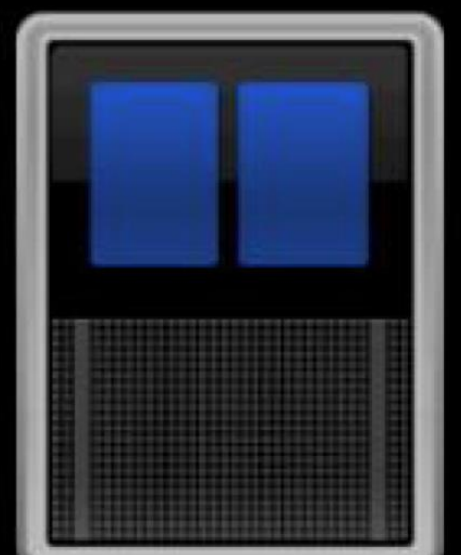
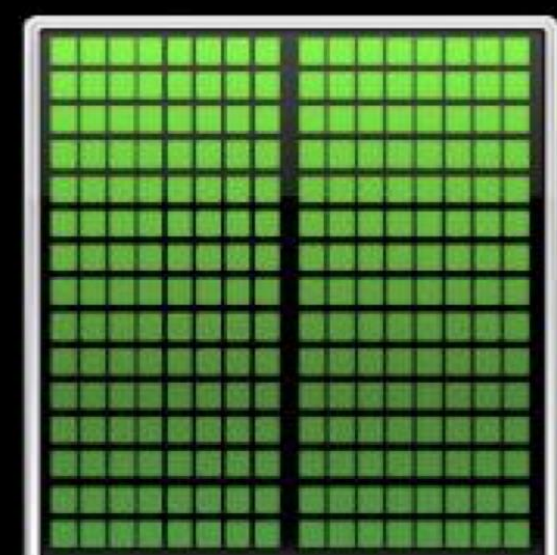
## Conclusions

It was decided that using GPUs to run graph pebbling algorithms would not result in a worthwhile increase in speed. We came to this conclusion because different threads would be running tasks on different vertices, which would make the tasks slightly different. Because threads within the same warp would be completing different tasks, there would likely be a large amount of warp divergence, significantly harming efficiency.

## Warps

Threads within a block are grouped into sets of 32 threads, called warps. Threads within a warp run in lock-step and make synchronized memory accesses. If threads within a warp need to complete different tasks, the warp completes each type of task sequentially. Every thread in a warp that is not completing the type of task that is being completed must wait, creating inefficiency. This is called warp divergence. Conditional statements are one way that different threads in a warp can be made to complete different tasks.

## CPUs vs. GPUs

CPU	GPU
	
contain one to tens of cores	contain hundreds to thousands of cores
low memory transfer overhead	high memory transfer overhead
cores can work independently	cores perform same functions on different data

Images adapted from <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/>

## CUDA C

CUDA C is a platform and programming model for writing code that uses GPUs. Other options for GPU programming exist, which use various languages and types of GPUs.

