**Video Link: - Maximizing the OS throughput with GPU abstractions**

# Maximizing the OS throughput with GPU abstractions

Prakhar Varshney[1]

[1]Boston University {prakhar}@bu.edu

## Abstract

Throughput has always been a key metric to rate the efficiency of an Operating System. Recent growth in computer hardware technologies has increased the throughput of modern computers by a large margin but is the current Operating system architecture utilizing the capability of the hardware efficiently? This paper focuses on the key factors on which the throughput of an OS depends and how it can be maximized by providing a better abstraction for other hardware components like for example a GPU.

## 1. Introduction

Throughput can be defined as the amount of work done in unit time or it can be further elaborated as the number of processes that have been completed in unit time. The Scheduling algorithm which an OS selects to schedule the processes is highly dependent on achieving the maximum throughput by keeping the minimum latency. A good OS schedules the processes to maximize the throughput. There are other criteria also to select the most efficient scheduling algorithm one of them is the CPU utilization. An efficient OS tries to maximize the CPU utilization however high CPU usage increases the waiting time for new upcoming processes and the computer becomes very slow when it has to perform several concurrent heavy calculations. Recent multicore processors tend to solve this issue but it is yet not the best optimal method to solve it. Modern OS lacks a direct abstraction for switching a process from CPU to GPU. So, in simple words when the CPU is having a tough time in solving complex problems the GPU is sitting idle

doing nothing which is a complete waste of computer hardware resources.

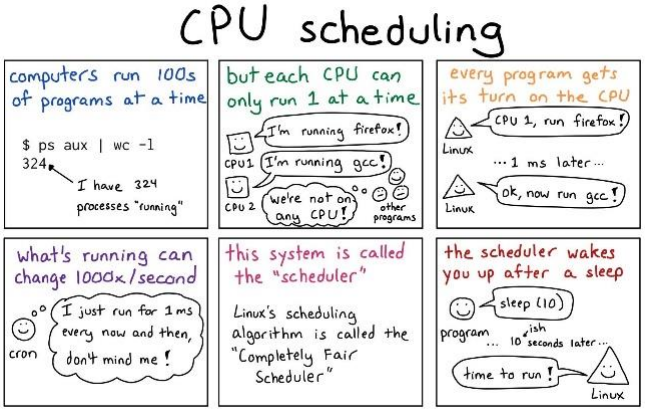


*Figure 1. How the CPU scheduling [1] works*

## 2. The GPU Scheduling

The GPU is capable of solving complex mathematics a lot faster than the CPU and that is why it is highly used for 3-D graphic rendering. To understand why GPU is a lot faster than the CPU we have to understand the basic difference in the architecture of both. The CPU is designed to process tasks concurrently with the ability to switch tasks as fast as possible to reduce the latency while a GPU is designed to provide maximum throughput by pushing as many tasks as possible and parallelly processing them all at once. It is all possible due to the presence of a lot more cores in GPU than in CPU.

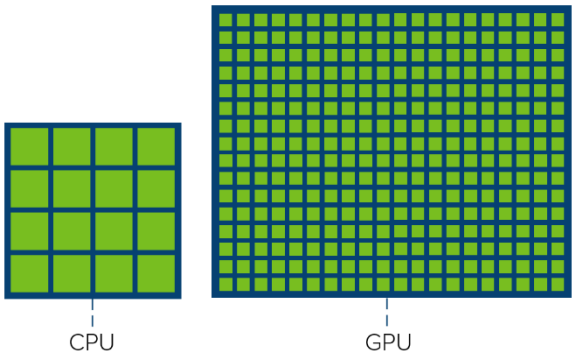


*Figure 2. CPU cores Vs GPU cores*

CPU and GPU have some similarities too. Both CPU and GPU have the same architecture [2] for memory since both use cache memory, memory controller and global memory. Utilizing both CPU and GPU is called High-Performance Computing (HPC). However, HPC is achieved by using a parallel programming model like CUDA [3] or DirectX [4] which could be more efficient if switching the context between GPU and CPU is directly handled by the OS.

After having a rough understanding of architectures and working on both we can now focus on restructuring the modern scheduling algorithms to see if they can improve the overall throughput by increasing the parallelism or are there any roadblocks that we have to work on.

## 2.1 GPU First Come First Serve

First Come First Serve is the simplest of all scheduling algorithms. It is non-preemptive which means a process is held by the CPU until it gets terminated or reaches a waiting state. The GPU FCFS works in the same fashion the only difference is that when the OS detects the CPU to be in a busy state it transfers the process to GPU. Hence, reducing the waiting time for other processes.

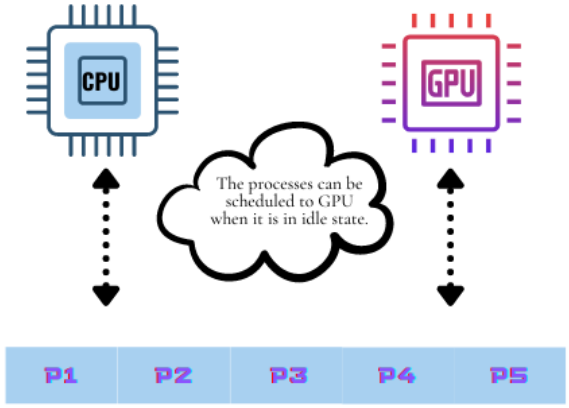


*Figure 3. FCFS with GPU*

## 2.2 Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue is the default scheduling algorithm that is used in Windows operating system. In this algorithm, the processes in the ready queue are divided into different queues according to their priorities such that each queue level has a certain priority level and the priority decreases as we go down in level.

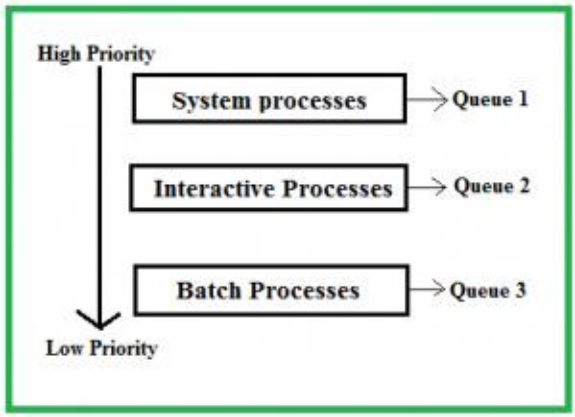


*Figure 4. Multilevel Feedback Queue.*

A GPU driven Multilevel Feedback Queue can work in such a manner that a whole queue can be scheduled to GPU when it is in an idle state. Hence, reducing the workload on the CPU.

Let us look practically at how it is helpful by taking an example problem.

| PROCESS | ARRIVAL TIME | CPU BURST TIME | QUEUE NUMBER |
|---|---|---|---|
| P1 | 0 | 4 | 1 |
| P2 | 0 | 3 | 1 |
| P3 | 0 | 8 | 2 |
| P4 | 10 | 5 | 1 |

*Figure 5. Multilevel Queue Scheduling sample problem.*

Consider four processes with their queue number assigned such as Priority of Q1 is greater than Priority of Q2. Its Gantt chart would be.

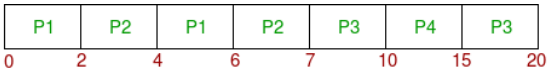


*Figure 6. Gantt Chart for CPU only.*

When the processes arrive both the queues will have processes (P1, P2, P4) in queue 1 and (P3) in queue 2. The processes in queue 1 (P1, P2) will run first due to high priority and they will have the CPU time in a round-robin manner and gets finished in 7 units then, P3 is assigned to the CPU but while it is running P4 arrived at 10 units and is given the CPU time. After completion of P4, P3 is again assigned to the CPU until it gets finished in 20 units.

The same problem can be more efficiently solved by introducing a GPU abstraction. When the four processes arrive P1 and P2 can be assigned to the CPU while P3 can be assigned to GPU to get finished parallelly. So, when P4 arrives at 10 units it is assigned to CPU (since P1 and P2 were finished in 7 units) and it gets finished in 15 units. The following Gantt chart explains it.

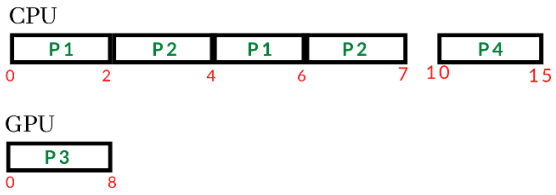


*Figure 7. Gantt Chart for processing with both GPU and CPU*

## 3. Need for OS Abstractions for GPU

In the previous section, we have seen modifications over current scheduling algorithms but for them to work efficiently to maximize the throughput we have to also modify the current OS architecture. Modern OSs treat the GPU as an I/O device rather than a shared computation resource. This is the reason why we need an external GPU driver like CUDA. Still, the application becomes very limited due to a smaller number of operations available. The GPU is designed to provide maximum throughput while the CPU is designed to keep the minimum latency. GPU should be treated as a co-processor and it should have a similar abstraction to CPU so that OS has full control over the load balancing [5].

## 4. Conclusion

This research shows how GPUs can be used as a co-processing unit to maximize the system throughput. The study of the modification of current scheduling algorithms also addresses the problem with

current architecture which needs to be modified so that the GPU resources can be used for computation.

## References


[1] Avi Silberschatz, Greg Gagne, and Peter Baer Galvin, *Operating system concepts, 10th edition,2018.*

[2] VMware, Inc, *Exploring the GPU Architecture*,2020

[3] NVIDIA. *NVIDIA CUDA Programming Guide*, 2011.

[4] D. Blythe. The Direct3D 10 system. *ACM Trans. Graph*., 25(3):724–734, 2006.

[5] Christopher J. Rossbach1, Jon Currey1, and Emmett Witchel. *Operating Systems must support GPU abstractions*, 2011.