Process Isolation on Edge GPUs

Kevin Jain
Dept of Computer Science
Athens, USA
kevin.jain@uga.edu

Aditi Patil
Dept of Computer Science
Athens, USA
aditi.patil@uga.edu

***Abstract*—Edge computing is gaining a lot of traction due to its ability to process large amounts of data at the edge of the network thus reducing workloads at the cloud. They also improve response times for AI applications leading to better user experience. Edge devices find many applications today ranging from autonomous vehicles, cloud gaming, content delivery & real time traffic management. However, they are limited in terms of resources (CPU, GPU and memory) and thus maintaining QoS while running multiple AI applications becomes difficult. We investigate latency issues introduced by running multiple applications on edge devices and propose a scheduler that aims to break the workloads into batches of smaller tasks and queue them on the GPUs so that they run in limited space. It aims to isolate the processes from each other and mitigate resource interference. Our scheduler will effectively allow maximizing the throughput of GPUs and also enable guarantees in reponse time.**

***Index Terms*—edge devices, gpu, isolation, scheduler**

I. Introduction

Industries are trying to enhance their services by deploying edge devices in almost every paradigm imaginable. They are used in autonomous trucks, where group of truck travel close behind one another in a convoy, saving fuel costs and decreasing congestion. They are used for cloud gaming, where companies build edge servers as close to gamers as possible in order to reduce latency and provide a fully responsive and immersive gaming experience. They are also used in content delivery, as by caching content – e.g. music, video stream, web pages – at the edge, improvements to content deliver can be greatly improved. Latency can be reduced significantly. Content providers are looking to distribute CDNs even more widely to the edge, thus guaranteeing flexibility and customisation on the network depending on user traffic demands. Edge computing can enable more effective city traffic management. Examples of this include optimising bus frequency given fluctuations in demand, managing the opening and closing of extra lanes, and, in future, managing autonomous car flows [1].

Previously, work has been done in the following directions. EdgeIso [2] aims to isolate processes on the CPU of edge devices using cgroups, however, it does not effectively target GPUs. Fractional GPUs [3] aims to perform compute and memory isolation on higher end GPUs, however it does not support GPUs on Edge devices due to having a different architecture. Other than scheduling applications on GPUs, work has also been done in trying to compress the AI applications to enable them to run in resource constrained devices [4], however these compressions come with reduction in accuracy and sometimes may not be feasible.

We propose a scheduler that will provision AI applications on GPUs as per their requirements and queue them in a way that will improve GPU throughput and minimize latency. We use Nvidia Jetson Nano [5] and Nvidia Xavier [6] to test the performance of our scheduler.

II. Related Works

A. Fractional GPUs

The paper presents Fractional GPUs (FGPUs), a software-only mechanism to partition both compute and memory resources of a GPU to allow parallel execution of GPU workloads with performance isolation. Further, this paper presents a software-based mechanism to allow multiple applications to run in parallel on a GPU while still maintaining isolation by partitioning compute and memory resources among these co-running tasks. As partitioning a single large GPU into smaller fractional GPUs opens up more options for real-time system architects regarding scheduling GPU resources this paper evaluations show that applications using FGPUs have significantly more predictable runtimes irrespective of other applications running in parallel and also present various details about Nvidia GPU which were previously unknown in the public literature. This paper focuses on Nvidia GPUs (and corresponding SDK i.e., CUDA [6]) because they are the leading platforms for high-performance computing.

B. EdgeIso

EdgeIso is a light-weight scheduler that monitors the resource contention of apps and mitigates them to meet the SLOs in an incremental manner. The paper focuses on isolating the performance of tasks while maximizing resource efficiency by profiling the dominant resource contention for tasks, detecting the phase changes of tasks such as load fluctuations and performing isolation techniques incrementally and adaptively. The paper also focuses on CPU, last-level cache (LLC) and memory bandwidth. It controls them by using three isolation techniques - CPU core allocation, CPU cycles throttling & GPU core frequency scaling.

C. A comprehensive survey on model compression and acceleration

This paper presents a survey of various techniques suggested for compressing and accelerating the ML and DL models. The

paper reviews the techniques, methods, algorithms proposed by various researchers to compress and accelerate the ML and DL models and presents a perceptive performance analysis, pros and cons of popular DNN compression and acceleration as well as explored traditional ML model compression techniques. Further in this paper, various aspects of model compression and acceleration techniques varying from DNNs to traditional machine learning algorithms are discussed. The actual power of model compression and acceleration will come from a combination of different techniques with combined research outcome in the form of software, efficient algorithm as well as specially designed hardware. The research explores on the lines of how much a model can be compressed and accelerated subject to given resource-constraints (storage, computational power, and energy) and user-specified performance goals (accuracy, latency).

III. Method

We propose a scheduler that will work as follows. Firstly, it will accept a requirements file from the user which will contain list of processes to run on GPU and their required resource allocations. Then it will map the resource requirements as per GPU availability. Then it will evaluate and break the tasks and push them onto an array of queues (here thread will have its own queue). After that, scheduler will start running and each thread on the GPU will start polling the tasks from the queue and perform the tasks. Finally, the scheduler will sychronize the threads to combine the results. We will also run a separate profiler that will run that will measure the efficacy of the scheduler and report results.

The above described method can allows us to implement functional level parallelism in our applications. The new scheduler isolates kernels with different mathematical functions like add, subtract & multiply. Each function processes one million elements each storing the results in shared variables. The working of our scheduler is also summarized in the psuedo code below. We have focused on the most important function of the scheduler which involves the mapping of resources and pushing the tasks onto a queue.

Algorithm 1 Mapper

Require: Requirements *req

$y \leftarrow 1$
while $i < num\_of\_kernels$ **do**
  while $j < req[i].numBlocks$ **do**
    while $k < req[i].blockSize$ **do**
      $queue[cursor].func \leftarrow req[i].func$
      $queue[cursor].blockSize = req[i].blockSize$
      $queue[cursor].numBlocks = req[i].numBlocks$
      $queue[cursor].threadIdx = k$
      $queue[cursor].blockIdx = j$
      $cursor++$
    end while
  end while
end while

IV. Results

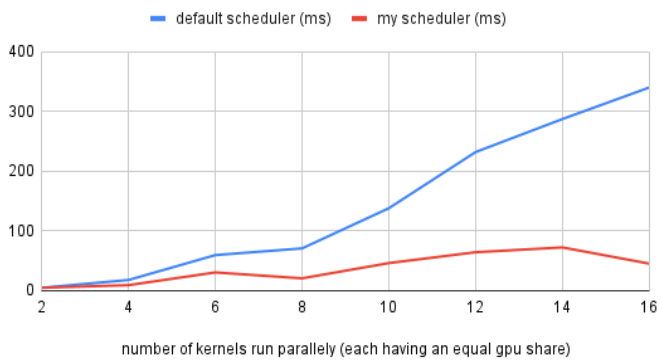

Fig. 1. Comparison of running multiple parallel kernels with the default scheduler and our new proposed scheduler.

"Fig. 1" compares our results with the original GPU scheduler by timing multiple kernels run together in parallel. The original scheduler is invoked by the use of CUDA streams [7], this allows us to run multiple kernels at the same time. We notice that under smaller workloads of two to four kernels being run parallely, our scheduler takes similar time to complete as the original scheduler. However, as we increase the number of kernels, we see that our scheduler out-performs the original scheduler by a huge factor. For example, with 10 kernels the original scheduler takes 137ms to finish however our scheduler takes only 45ms.

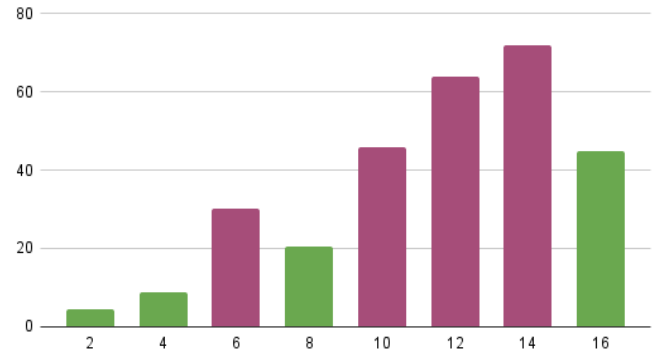

Fig. 2. Effect of running kernels in doubles of two on the speed of execution.

Secondly, we also notice in "Fig. 2" that our scheduler runs much faster in doubles of two. That is because the GPUs manage the threads in batches of 32. This means an extra latency is added if we divide the resources in odd numbers, which explains why it takes more time to run 14 kernels parallely than running 16 kernels.

V. Conclusion

Preliminary work is ongoing in this project. We plan to compare our schedulers efficiency with the original scheduler in GPUs (where we will run the same workloads). We also plan to run real-life workloads like Image Classification, Object

detection & Segmentation and see if our scheduler is able to improve their throughput.

REFERENCES

[1] Edge Computing use cases. https://stlpartners.com/articles/edge-computing/10-edge-computing-use-case-examples/. Accessed 2022-04-29.

[2] Y. Nam, Y. Choi, B. Yoo, H. Eom and Y. Son, "EdgeIso: Effective Performance Isolation for Edge Devices," 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 295-305, doi: 10.1109/IPDPS47924.2020.00039.

[3] S. Jain, I. Baek, S. Wang and R. Rajkumar, "Fractional GPUs: Software-Based Compute and Memory Bandwidth Reservation for GPUs," 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2019, pp. 29-41, doi: 10.1109/RTAS.2019.00011.

[4] Choudhary, T., Mishra, V., Goswami, A. et al. A comprehensive survey on model compression and acceleration. Artif Intell Rev 53, 5113–5155 (2020). https://doi.org/10.1007/s10462-020-09816-7

[5] Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano. Accessed 2022-04-29.

[6] Jetson Xavier NX. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/. Accessed 2022-04-29.

[7] CUDA Streams. https://on-demand.gputechconf.com/gtc/2014/presentations/S4158-cuda-streams-best-practices-common-pitfalls.pdf. Accessed 2022-04-29.