

SGPRS: Seamless GPU Partitioning Real-Time Scheduler for Periodic Deep Learning Workloads

Amir Fakhim Babaei

Department of Electrical & Computer Engineering,
Virginia Tech
Arlington, Virginia, USA
babaei@vt.edu

Thidapat Chantem

Department of Electrical & Computer Engineering,
Virginia Tech
Arlington, Virginia, USA
tchantem@vt.edu

Abstract—Deep Neural Networks (DNNs) are useful in many applications, including transportation, healthcare, and speech recognition. Despite various efforts to improve accuracy, few works have studied DNN in the context of real-time requirements. Coarse resource allocation and sequential execution in existing frameworks result in underutilization. In this work, we conduct GPU speedup gain analysis and propose SGPRS, the first real-time GPU scheduler considering zero configuration partition switch. The proposed scheduler not only meets more deadlines for parallel tasks but also sustains overall performance beyond the pivot point.

I. INTRODUCTION

The limited processing capability of CPUs is the primary challenge in implementing Deep Learning (DL) techniques, and NVIDIA GPUs are the most popular DNN accelerator. Due to underutilization, the co-location of DNN tasks on GPUs has become inevitable. GPUs can be partitioned spatially or temporally, making spatio-temporal partitioning a state-of-the-art technique to leverage both performance and timeliness [1], [2]. Given the limitations of current works to handle periodic real-time DL tasks, we propose **SGPRS** (*Seamless GPU Partitioning Real-time Scheduler*).

II. TASK MODEL

We define a task set $S = \{\tau_1, \tau_2, \dots, \tau_{|S|}\}$. Each task τ_i represents a DNN with a DAG structure. The nodes within τ_i represent the stages (sub-tasks) of the DNN, denoted as τ_i^j . Assume $CP = cp_1, \dots, cp_{n_p}$ represents a context pool with n_p options (CUDA contexts) and sm SMs for each. Let C_i and C_i^j represent the Worst-Case Execution Times (WCETs), while D_i and D_i^j represent the relative deadlines of the task and sub-task, respectively. D_i should be determined initially, while D_i^j is a virtual deadline explained in [Section IV-A2](#).

III. GPU SPEEDUP ANALYSIS

A linear speedup is not realistic in GPUs due to their architecture [3]. To showcase this, we use NVIDIA RTX 2080 Ti which has 68 SMs. ResNet18 [4] is used as the benchmark DNN to measure the speedup gain of common operations as a function of the number of SMs. The results are illustrated in [Figure 1](#). The *convolution* operation reaches the best speedup gain (32x) followed by *max pooling* (14x). Other operations failed to exceed 7x. The convolution layer dominates the overall speedup behavior of ResNet18 (only 23x) due to the presence of other layers.

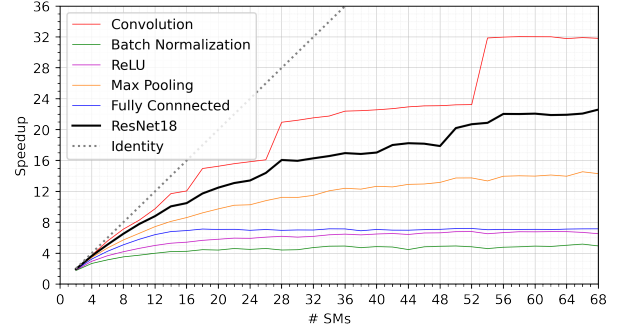


Fig. 1: Speedup gain for different operations when running in isolation

IV. REAL-TIME SCHEDULER DESIGN

Our model philosophy is to consider multi-tenant DNN inference. Popular DNN frameworks such as PyTorch [5], can not efficiently benefit from GPU concurrency mechanisms. Our approach results in enhanced efficiency and performance in scenarios with real-time constraints. We use *LibTorch* (C++ PyTorch) in this work. In SGPRS, we propose dividing a network (task) into multiple stages (sub-tasks) to improve flexibility. SGPRS is divided into two main phases, *offline* and *online* as illustrated in [Figure 2](#).

A. Offline Phase

1) *Two Level Priority Assignment*: We use a two-level offline priority assignment mechanism where the last stage of each task will have a high priority, while the rest of the stages will have a low priority. This helps to meet more deadlines.

2) *WCET and Virtual Deadline Assignment*: The WCETs of each task (C_i) and its stages (C_i^j) are measured offline. The relative deadline of the stages (D_i^j) will be assigned as a portion of the relative deadline of the entire task (D_i), proportional to their relative WCET.

B. Online Phase

1) *Absolute Deadline Assignment*: At each instance, the absolute deadline of each stage (d_i^j) will be assigned based on their relative virtual deadline (D_i^j).

2) *Context Assignment*: Released stages will be assigned to a context based on the following criteria: empty queues first, then the context meeting the deadline with the shortest queue, and if none, the one with the earliest finish time.

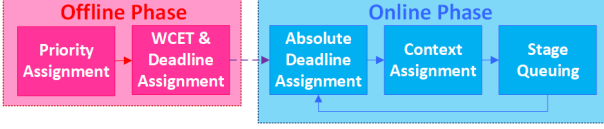


Fig. 2: SGPRS Overview

3) *Stage Queuing*: We use two high and two low-priority CUDA streams in each CUDA context, meaning a maximum of four stages are allowed in parallel in each context. We also add a third priority level (medium), which is assigned to low-priority stages whose preceding stage has missed its deadline. Stages inside each priority level will be scheduled in the *Earliest Deadline First* order.

V. SIMULATION RESULTS

We use the ResNet18 network with a 224x224 input and 30 fps as our benchmark task. We compare the proposed SGPRS alongside a naive approach, which is a simple spatial partitioning scheduler that lacks the context switch and temporal partitioning features. The naive scheduler is the best candidate for comparison because it highlights the benefits of SGPRS over pure spatial partitioning. To reach higher utilization, we use an over-subscribed context pool, meaning, the sum of SMs of all contexts might be larger than the total number of SMs available. Three instances of the proposed scheduler with three over-subscription options are considered. We use *SGPRS_{os}* notation where *os* represents the over-subscription level. Also, two metrics have been used for comparison, total FPS and Deadline Miss Rate (DMR). Finally, two scenarios of *two* and *three* context pool options have been chosen.

Identical periodic tasks (30 fps) with explicit deadlines with each divided into six stages are considered. We refer to the largest number of tasks that the scheduler can handle without deadline misses as the *pivot point*. In the naive scheduler, for both scenarios (Figure 3 and Figure 4), the pivot point happens much earlier than SGPRS. Also, both total FPS and DMR are degraded drastically after the pivot point. This happens due to the lack of temporal partitioning to prevent the domino effect of deadline misses after the pivot point. SGPRS variations, not only can sustain total FPS, but their DMR increases with a moderate slope. This shows that the lack of proper temporal partitioning leads to excessive contention. The total FPS of the naive scheduler drops to 468 fps and 459 fps for Scenario 1 and Scenario 2. This means 38% and 36% FPS drop compared to best-case SGPRS variations.

Even though both scenarios show a similar FPS and DMR pattern, there are some differences. Scenario 2 performs better overall. The best-case pivot point happens at 23 and 24 tasks in Scenario 1 and 2 respectively. Furthermore, in Figure 3a the FPS always increases relative to the over-subscription factor, but as represented in Figure 4a, the highest over-subscription will not lead to the best performance. Higher over-subscription leads to poor predictability and increased resource contention, which is why in Scenario 2 the 1.5x variation reaches higher (741 fps) compared to 2.0x (731 fps). In Scenario 1, this does

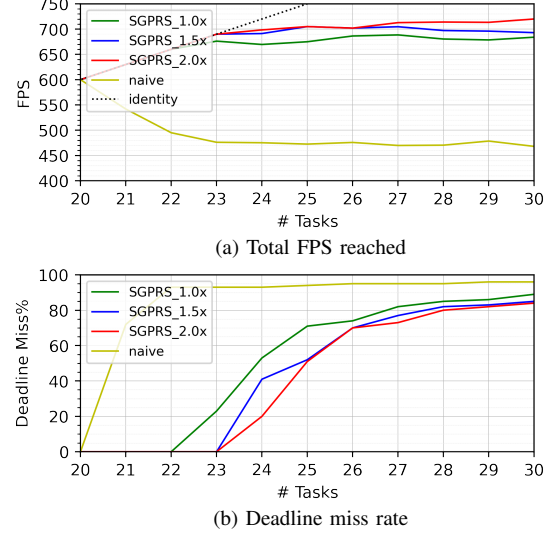


Fig. 3: Result when using 2 contexts (Scenario 1)

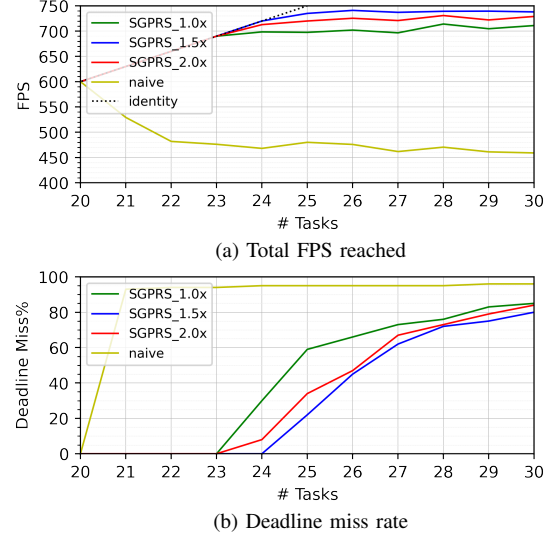


Fig. 4: Result when using 3 contexts (Scenario 2)

not happen because there are too few contexts to leverage the whole GPU and excessive contention is not a problem.

REFERENCES

- [1] L. R. Juracy, R. Garibotti, F. G. Moraes, *et al.*, "From cnn to dnn hardware accelerators: A survey on design, exploration, simulation, and frameworks," *Foundations and Trends® in Electronic Design Automation*, vol. 13, no. 4, pp. 270–344, 2023.
- [2] W. Cui, H. Zhao, Q. Chen, *et al.*, "Enable simultaneous dnn services based on deterministic operator overlap and precise latency prediction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.
- [3] W. Jiang, E. H.-M. Sha, X. Zhang, *et al.*, "Achieving super-linear speedup across multi-fpga for real-time dnn inference," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.
- [4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [5] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.