2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-0386-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/RTAS52030.2021.00041

Brief Industry Paper: Workload-Aware GPU Performance Estimation in the Airborne Embedded System

1st Yuan Yao
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
yaoyuan@nwpu.edu.cn

2nd Sikai Wu
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
wusikai@mail.nwpu.edu.cn

3rd Shuangyang Liu
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
liushuangyang@mail.nwpu.edu.cn

4th Qingshuang Sun
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
sunqsh@mail.nwpu.edu.cn

5th Gang Yang
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
yeungg@nwpu.edu.cn

6th Yujiao Hu
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
yujiao_hu@mail.nwpu.edu.cn

7nd Yu Zhang
School of Computer Science
Northwestern Polytechnical University
Xi'an, China
zhangyu@nwpu.edu.cn

8nd Huixia Liu
Department of Command and Control
Xian Aisheng Technology Group Co.,Ltd
Xi'an, China
liuhuixia_nwpu@126.com

9nd Xinyu Tian
Department of Aviation Electronics
Xian Aisheng Technology Group Co.,Ltd
Xi'an, China
windtian2002@163.com

***Abstract*—New generation airborne embedded system has deployed Graphical Processing Units (GPUs) to raise processing capability to meet growing computational demands. Applications in the airborne embedded system have strict real-time constraints. Therefore, it is necessary to accurately predict timing behaviors of those applications. Many previous work propose GPU performance models to estimate the execution time of applications. However, most of those models do not consider the impact of co-execution on the GPU performance. In this paper, we propose a workload-aware GPU performance model to predict the execution time of applications executed concurrently on a single GPU. Experimental results illustrate that the proposed model can achieve a 5.1%-11.6% prediction error in a real airborne embedded hardware platform.**

***Index Terms*—GPU performance model, GPU workload, Airborne embedded system, Self-organization fuzzy neural network**

I. Introduction

With the rapid development of Unmanned Aerial Vehicle (UAV), more and more payloads are equipped on it. This makes a strong demand of computational capacity for airborne embedded system to deal with an increasing number of applications such as radar signal processing, object detection, target tracking and so on. With the widespread use of GPU in diverse industrial fields, most of advanced airborne embedded systems adopt GPU as an accelerator to improve performance.

The airborne embedded system is a typical real-time system which has strict timing constraints for applications. Therefore, to accurately predict timing behavior of an executing application is crucial for such systems. To handle this problem, researchers have established various analytical GPU performance models to estimate the kernel execution time. Nath *et al.* propose the pipeline models to reveal the relationship between computation cycles and memory latency [1]. Song *et al.* establish a counter-based performance model which is faster than cycle-based simulations [2]. Lew *et al.* adopt simulation method that combines performance counters and specific hardware information to estimate the kernel execution time [3]. Wu *et al.* use K-means clustering that builds a performance model of a kernel at a single configuration to predict performance across other configurations [4]. Dao *et al.* use a machine learning based model to improve the proposed linear performance model [5]. Wang *et al.* establish a novel GPU pipeline performance model with GPU memory model to predict kernel execution time under different core and memory frequencies [6]. However, these models only predict the performance considering the situation that a single application runs on the GPU.

The co-execution of multiple applications on a single GPU was first supported by NVIDIA Fermi architecture. Then, NVIDIA's Kepler GPUs adopt Hyper-Q technology that employ 32 hardware queues to avoid false dependencies between computations. They can achieve higher efficiency of concurrent execution than Fermi GPUs. When multiple applications concurrently run on a single GPU, each application suffers significant execution performance degradation caused by the resource contention. To motivate our work, we conduct an experiment to measure the execution time of an application lavaMD[1] which runs concurrently with different number of

[1]lavaMD is a molecular dynamics application in Rodinia benchmark.

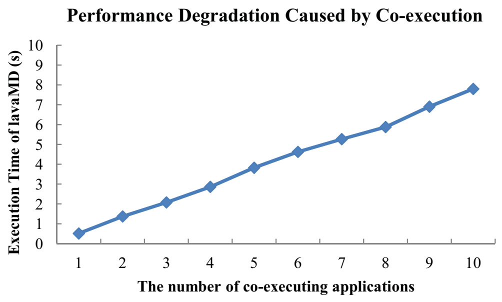

Fig. 1. The impact of co-execution on GPU performance

other applications. As shown in Fig. 1, we observe that execution time of lavaMD is greatly increasing with the growth of co-executing applications. It is challenging to establish an accurate performance model to predict the execution time of co-executing applications since there are various resources (CUDA core, global memory, shared memory, cache) involved in concurrent execution.

In this paper, for a specific application A, we define workload as the other applications concurrently executing with A on a single GPU. We propose a Workload-Aware GPU Performance Model to predict the execution time of an application under different workload circumstances.

The rest of the paper is organized as follows. Section II gives the propose GPU performance model. Section III presents the evaluation results. Section IV concludes the paper.

II. Workload-Aware GPU Performance Model

In this section, we describe the proposed Workload-Aware GPU Performance Model in detail. We first give the overview architecture of the system. Then, we analyze some GPU workload-related indicators. Finally, we put forward a Self-Organizing Fuzzy Neural Network (SOFNN) to predict the execution time of applications.

A. System Overview

The system architecture is shown in Fig. 2. We add the proposed performance model on gVirtuS [7] to support real-time scheduling in the airborne embedded system. The gVirtuS is a GPU visualization framework which is composed of two parts. The *frontend* of gVirtuS implements a wrapper stub CUDA runtime library. The wrapper library intercepts all CUDA requests from applications and collects arguments. All intercepted CUDA requests are packaged with arguments and are sent to gVirtuS *backend* via communicator. The *backend* of gVirtuS will launch a daemon thread for each received CUDA request from the service thread pool and put it into the waiting queue. The scheduler selects appropriate GPUs for threads from the waiting queue according to the specific schedule strategy and the estimated GPU performance based on GPU performance model. Then, those applications execute concurrently on the real hardware platform. We use CUDA Profiling Tool Interface (CUPTI) [8] to get workload-related

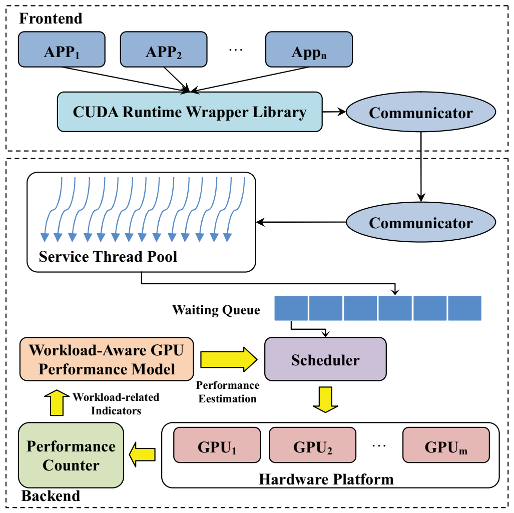

Fig. 2. The overview of the system architecture

hardware performance counters. The proposed performance model predicts the execution time of incoming applications and estimates the remaining time of other co-executing applications. Due to space limitations, we only focus on the workload-aware GPU performance model in this paper. The real-time scheduling of the airborne embedded system is not taken into account.

B. GPU Workload-related Indicators

Co-executing applications on a single GPU will lead resource contention which greatly degrades GPU performance. For a specific application A, we define the workload as the other applications concurrently executing with application A. Thus, the GPU performance model needs to collect the workload-related indicators as inputs. The major GPU resources include stream multi-processor, global memory, L2 cache and shared memory. To understand the resource usage of the workload, we profile each application in the workload via CUPTI. The CUPTI events used to describe resource usage are detailed in Table I.

C. SOFNN based GPU Performance Estimation

We adopt an Self-Organizing Fuzzy Neural Network (SOFNN) [9] to model the GPU performance with different workloads. SOFNN can efficiently model non-linear dependencies among different inputs based on fuzzy inference. In addition, it also has the learning ability of neural network to generate the membership functions and fuzzy rules by itself.

The structure of SOFNN for GPU performance estimation is shown in Fig. 3. Since the airborne embedded system has a fixed application task set, the number of applications which need to use the GPU is deterministic. Thus, besides of the selected events of resources usage, we add the status of running applications as inputs. The status include the number of running applications (denoted by N) and the elapsed

TABLE I
THE EVENTS OF RESOURCE USAGE

Resources	Indicators	Events
Stream Multi-processor	*instructions*	inst_executed
	cycles	active_cycles
	warps	active_warps
	threads	thread_inst_executed
Global Memory	*global_load*	gld_inst_8bit+gld_inst_16bit+gld_inst_32bit+gld_inst_64bit+gld_inst_128bit
	global_store	gst_inst_8bit+gst_inst_16bit+gst_inst_32bit+gst_inst_64bit+gst_inst_128bit
L2 Cache	*l2_read*	l2_subp0_total_read_sector_queries+l2_subp1_total_read_sector_queries
	l2_write	l2_subp0_total_write_sector_queries+l2_subp1_total_write_sector_queries
Shared Memory	*shared_load*	shared_load
	shared_store	shared_store

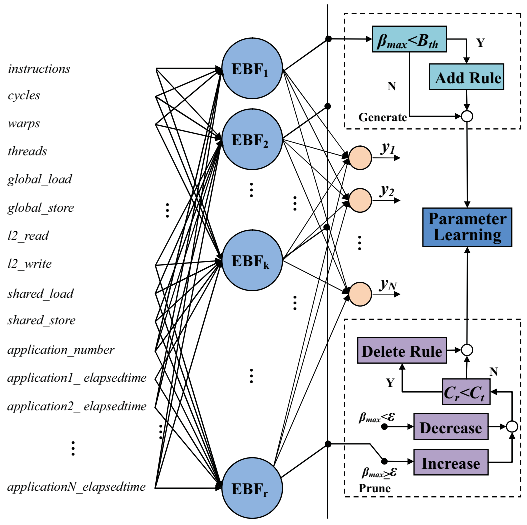

Fig. 3. The structure of SOFNN

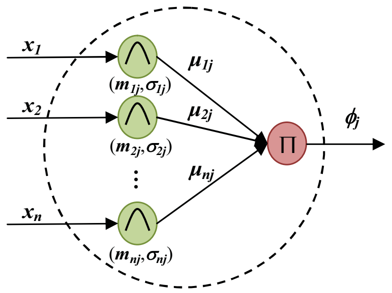

Fig. 4. The j^{th} EBF unit

execution time of these applications (the elapsed time of a non-executed application is set to be 0).

SOFNN is a four-layer network. The first layer is the input layer which contains all workload-related indicators. The second and third layers can be seen as multiple Ellipsoidal Basis Function (EBF) units. As shown in Fig. 4, an EBF unit includes the membership function layer and the fuzzy rule layer. Each EBF unit is a premise part (IF-part) of the fuzzy rule. Nodes in the membership function layer represent the fuzzy linguistic level of the input variables. For an input x_i, the Gaussian membership function is used.

$$\mu_{ij} = exp\left[-\frac{(x_i - m_{ij})^2}{2\sigma_{ij}^2}\right], i = 1, 2 \dots n, j = 1, 2 \dots r, \quad (1)$$

where μ_{ij}, m_{ij} and σ_{ij} are the output, mean and the deviation of the Gaussian membership function of the i^{th} input x_i in the j^{th} EBF unit. r is the number of generated fuzzy rules. The fuzzy rule layer uses the product operation as the fuzzy AND operation to determine the firing strength of each rule. The output ϕ_j of the j^{th} rule is as follows.

$$\phi_j = \prod_{i=1}^{n} \mu_{ij}, j = 1, 2, \dots, r, \quad (2)$$

Nodes in output layer perform defuzzification. The outputs of the network are given as follows.

$$y_i = \frac{\sum_{j=1}^{r} \phi_j w_{ij}}{\sum_{j=1}^{r} \phi_j}, i = 1, 2 \dots N, \quad (3)$$

where w_{ij} is the connect weight between the jth rule and the ith output as a consequent part (THEN-part) of the fuzzy rule. The output of the SOFNN is a $1 \times N$ vector $\{y_1, y_2, ..., y_N\}$. For an incoming application A_i, y_i is the predicted execution time of A_i. For a running application A_i, y_i is the estimated remaining execution time of A_i.

III. EXPERIMENTS

A. Experimental Setup

We evaluate the proposed Workload-aware GPU performance model on a real airborne embedded system as shown in Fig. 5. Table II gives the detail hardware specifications of GPU in our experimental platform.

B. SOFNN Training

We select 8 applications from Rodinia benchmark [10] to simulate a task set of the airborne embedded system. The applications includes b+tree(BT), cfd(CF), Gaussian(GS),

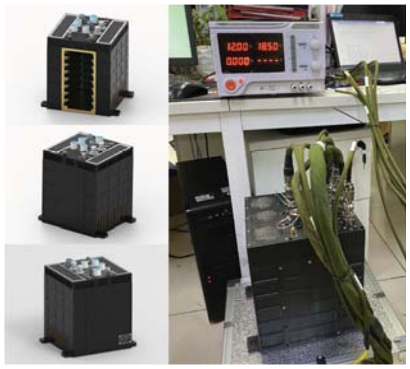

Fig. 5. The airborne embedded system

TABLE II
SPECIFICATIONS OF THE GPU

Parameter	Value	Parameter	Value
GPU	GTX1060	Core frequency	1506MHz
Architecture	Pascal	Memory frequency	8000MHz
CUDA Core	1280	Memory size	6GB
SM number	10	Memory bandwidth	192GB/s

hotspot3d (HS), lavaMD(LM), leukocyte(LC), myocyte(MC) and srad(SR).

In the training phase, we implement a monitor thread to collect all input variables of SOFNN. The workload-related indicators are get via CUPTI and the elapsed time of co-executing applications are computed based on a timer. All applications are scheduled periodically. For each simulation run, we set different initial arrival time for applications. First, the monitor thread samples input variables 100 times per second in format of a tuple $\langle t, x_1, x_2, ..., x_n \rangle$. t is the timestamp of each sampling. Then, we can compute the remaining time of all applications according to the timestamp and the finish time of these applications. Finally, the sampling tuples and the computed remaining time are used as inputs and outputs to train SOFNN.

C. Experimental Results

We evaluate the accuracy of the proposed workload-aware GPU performance model using Mean Absolute Percentage Error (MAPE) defined as follows.

$$MAPE = \frac{1}{M} \sum_{k=1}^{M} \frac{\left| S_m^k - S_e^k \right|}{S_e^k} \times 100\% \qquad (4)$$

where S_m^k is the measured execution time and S_e^k is the estimate execution time.

In the evaluation phase, the monitor thread only collects a sample pair $\langle S_m^k, S_e^k \rangle$ when an incoming application arrivals or an already executed application is finished. We collect over 8000 sample pairs to compute MAPE for 8 applications. The results are shown in Fig. 6. Our workload-aware GPU performance model estimates achieve a low average MAPE (5.1%-11.6%).

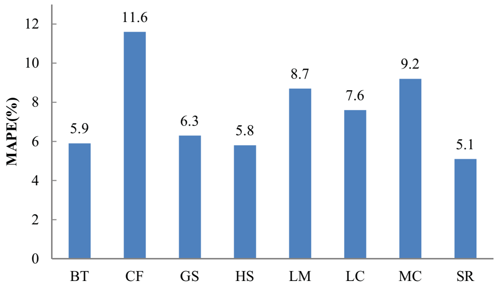

Fig. 6. MAPE value for different applications

IV. Conclusion

In this paper, we present a workload-aware GPU performance model to predict the execution time of applications under co-execution circumstance. The model first collects workload-related indicators via CUPTI. Then, it predicts the execution time of incoming applications and estimates the remaining time of the executed applications based on SOFNN. The experimental results prove that the proposed performance model achieve a low prediction error.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (No. 61876151, 62032018) and the Fundamental Research Funds for the Central Universities (No. 3102019DX1005).

References

[1] R. Nath and D. Tullsen, "The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU," in Proc. 48th Annu. IEEE/ACM Int. Symp. Microarchitecture, 2015, pp. 281–293.

[2] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in Proc. 27th Int. Symp. Parallel Distrib. Process., 2013, pp. 673–686.

[3] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogersand and T. M. Aamodt, "Analyzing machine learning workloads using a detailed GPU simulator," in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw., 2019, pp. 151–152.

[4] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in Proc. IEEE 21st Int. Symp. High Perform. Comput. Architecture, 2015, pp. 564–576.

[5] T. T. Dao, J. Kim, S. Seo, B. Egger, and J. Lee, "A performance model for GPUs with caches," IEEE Trans. Parallel Distrib. Syst., vol. 26, no. 7, pp. 1800–1813, Jul. 2015.

[6] Qiang Wang and Xiaowen Chu, "GPGPU Performance Estimation With Core and Memory Frequency Scaling," IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 12, pp. 2865–2881, Dec. 2020.

[7] Giunta G., Montella R., Agrillo G. and Coviello G., "A GPGPU Transparent Virtualization Component for High Performance Computing Clouds," In Proc. 16th Int. Euro-Par Conference., 2010, pp. 379–391.

[8] NVIDIA, "CUPTI User's Guide," 2020. [Online]. Available: https://docs.nvidia.com/cupti/pdf/Cupti.pdf

[9] Y. Yao, K. Zhang, and X. Zhou, "Application of a self-organizing fuzzy neural network controller with group-based genetic algorithm to greenhouse," in Proc. 7th International Conference on Natural Computation, 2011, pp. 660–667.

[10] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in Proc. of the IEEE Intl Symp. on Workload Characterization (IISWC), 2009, pp. 44–54.