Resource-Constrained Spatial Multi-Tasking for Embedded GPU

Woohyun Joo[1], and Dongkun Shin[2], *Member, IEEE*
[1] Digital Media and Communication R&D Center, Samsung Electronics, Suwon, Korea
Email: wh0225.joo@samsung.com
[2] School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea
Email: dongkun@skku.edu

Abstract-- **For recent smart devices, the embedded GPU is an indispensable unit and multi-task scheduling on the GPU becomes an important issue. To prevent the performance degradation on foreground GPU task by competing background GPU tasks, we propose a novel spatial multi-tasking which limits the number of available GPU cores for each task based on its priority. The proposed algorithm improved the performance of high priority task by up to 14% over the temporal budget based multi-tasking.**

I. INTRODUCTION

For recent smart consumer devices such as smartphones, smart TVs, and tablets, the embedded graphics processing unit (GPU) is an essential hardware renderer for rich GUI and fast responsiveness on user experience. Generally, the GPU has multiple processing cores for parallel executions. For example, the latest embedded GPU models have 8 shader processors.

Recently, multi-tasking support is a key requirement for such embedded GPUs. For example, users could launch multiple GPU applications at the home screen, such as live wallpaper, widget, and popup browser. Furthermore, GPUs with unified shaders support general purpose GPU (GPGPU) operations for compute-intensive workloads such as real-time motion estimation. As more applications rely on GPU for their computations, multi-task scheduling on GPU becomes an important issue to support priority-driven scheduling, QoS guarantee, and performance isolation, etc. Especially, the time-sensitive foreground process should be prioritized over the background processes such as live wallpaper application. However, current GPUs use the first-come-first-service (FCFS) scheduler or provide fairness among applications.

Recently, a time-driven GPU scheduling algorithm is proposed [1], which allocates different time budgets to GPU tasks based on their priorities. However, the time-driven approach cannot control the time utilization of each task completely due to the non-preemptive nature of GPU tasks. In addition, the interrupt-handling overhead for time-driven scheduling will not be negligible.

In this paper, a novel spatial multi-tasking technique is proposed for arbitrarily arriving non-preemptive GPU tasks. It allows multiple applications to execute simultaneously on different GPU cores. Especially, it assigns each GPU application with a different number of processing cores based on the priority of the application. Since the spatial multi-tasking is not a form of temporal multi-tasking, it can effectively handle the non-preemptive nature of GPU processing as well as remove the overhead of timer interrupt handling.

This research was partly supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2010-0020724).

II. RELATED WORKS

A. Temporal Budget Reservation

TimeGraph [1] provides the temporal budget reservation (TBR) techniques, where different time budgets are assigned to different priority GPU tasks. The TBR scheduler allows each task to occupy GPU resources only when the task has a remaining budget. In order to manage the temporal budget, TBR scheduler requires timer interrupt service, which increases the number of context switches.

In addition, TBR cannot prevent the overrun of low-priority tasks due to the non-preemptive nature of GPU processing. To handle such a problem, TBR scheduler employed the Apriori Enforcement (AE) mechanism, which predicts the execution times of GPU tasks by profiling, and prevents a GPU task from being scheduled if the remaining temporal budget is smaller than the expected execution time. However, the prediction-based scheduling requires a considerable CPU overhead. Moreover, since it is hard to predict the execution time precisely, TBR cannot completely remove the overrun situation.

B. Spatial Multi-Tasking on GPU

While the temporal GPU scheduling does not allow multiple different GPU applications to access GPU resources simultaneously, the spatial multi-tasking allows multiple applications to execute simultaneously on different cores of GPU. Spatial multi-tasking divides GPU resources, rather than GPU time, among competing applications. Adriaens et al. [2] showed there is an average speedup of up to 1.19 over temporal multi-tasking. However, they examined only simple heuristics such as even-split without considering the priorities of competing tasks. Recent embedded GPUs also adopt the spatial multi-tasking. Different GPU tasks can be executed simultaneously at different processing cores. However, it does not control the number of allocated GPU cores for each task.

III. RESOURCE-CONSTRAINED SPATIAL MULTI-TASKING

We propose the resource-constrained spatial multi-tasking (RCSM) technique, which constrains low-priority tasks from occupying some resources even though they are available in order to reserve the resources for high priority tasks. The proposed RCSM scheduler is based on job arrival and completion events and does not require timer interrupts.

Fig. 1 shows an example of GPU resource scheduling for

three different priorities of GPU tasks, T_1, T_2, and T_3. The low-priority task T_3 and the mid-priority task T_2 are allocated at maximum one and two processing cores, respectively. When two jobs of T_3 arrive at the time of 10, the task T_3 can use only one processing core even though there are four available cores. When three jobs of T_2 arrive at the time of 20, the task T_2 can use only two processing cores while reserving one processing core for the high-priority task T_1. When the jobs of T_2 arrive at the time of 30, they cannot be serviced even though one processing core is idle. As a result, when the job of T_1 arrives at the time of 40, it can be executed immediately without waiting delay. To improve the resource utilization, a higher-priority task may use more resources than its budget only when lower-priority tasks have not occupied their budgets. When the job of T_3 completes at the time of 40, T_2 can use three processing cores since there is no resource allocated for T_3.

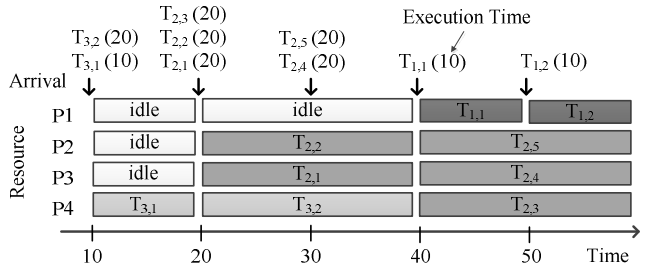


Fig. 1. An example of resource distribution on RCSM. The budgets of T_1, T_2, and T_3 are 1, 2, and 1, respectively.

To know the required resource budget for a task T_i, we should know its execution time and deadline. Then, we can calculate the required minimum budget $r_{min}(T_i)$. With the required budget and the available budget $r_{avail}(T_i)$, the allocated budget, $r_{alloc}(T_i)$, is determined as shown in (1) and (2). R represents the total number of resources. For low-priority tasks, r_{alloc} could be smaller than r_{min}. However, our goal is to guarantee the performance of high-priority tasks even while sacrificing the performance of low-priority tasks. This is a reasonable approach since the performance of user-interactive foreground job is more important.

$$r_{alloc}(T_i) = min(r_{min}(T_i), r_{avail}(T_i)) \quad (1)$$

$$r_{avail}(T_i) = R - \sum_{j=1}^{i-1} r_{alloc}(T_j) \quad (2)$$

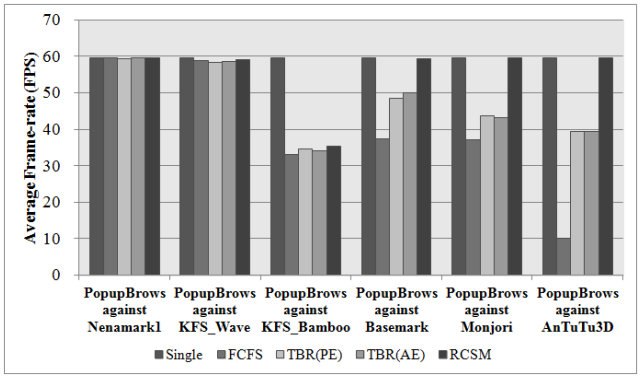
One weakness of RCSM is that its scheduling granularity depends on the total number of resources. Especially, if the number of concurrent tasks is larger than the number of resources, the low-priority tasks cannot be scheduled. However, only two or three GPU applications are executed concurrently in a single user mobile device, while the number of processing cores in GPU is increasing.

The RCSM scheduler can be integrated with power management technique. When there are only low-priority tasks, several GPU processing cores will be idle by the reservation mechanism of RCSM. For low power consumption, the idle devices can be changed into low-power mode.

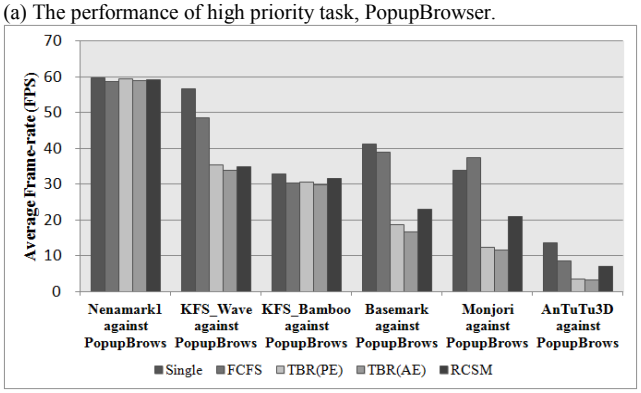
IV. Experimental Results

We evaluated the performance of RCSM at a smartphone which has an embedded GPU with four fragment processors. We implemented RCSM within the GPU device driver in Linux 3.0.15. For evaluation scenarios, we executed the popup browser as a primary task while executing several GPU benchmark programs as background GPU tasks. We also implemented FCFS and TBR schedulers for comparison.

As shown in Fig. 2(a), the RCSM scheduler improves the performance of primary task significantly. In most of cases, the performance of primary task is close to the performance at single mode, where only one task is executed without any background tasks. In addition, the RCSM improves the performance of background task compared with the TBR as shown in Fig. 2(b), since RCSM does not wait the budget replenishment.



(a) The performance of high priority task, PopupBrowser.



(b) The performance of low priority task, GPU benchmark.

Fig. 2. Performance under different GPU scheduling algorithms.

V. Conclusions

We proposed a novel spatial multi-tasking technique. It can improve the performance of high-priority GPU task by reserving the GPU processing cores and can reduce the power consumption without affecting user responsiveness.

References

[1] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "TimeGraph: GPU scheduling for real-time multi-tasking environments," Proceedings of the USENIX Annual Technical Conference, 2011.

[2] J. T. Adriaens, K. Compton, N. Kim, and M. J. Schulte, "The case for GPGPU spatial multitasking," Proceedings of High Performance Computer Architecture (HPCA), pp.1-12, Feb. 2012.