Work-In-Progress: Understanding the Effect of Kernel Scheduling on GPU Energy Consumption

Yidi Wang and Hyoseung Kim
University of California, Riverside
ywang665@ucr.edu, hyoseung@ucr.edu

***Abstract*—General-purpose graphics processing units (GPUs) made available on embedded platforms have gained much interest in real-time cyber-physical systems. Despite the fact that GPUs generally outperform CPUs on many compute-intensive tasks in a multitasking environment, higher power consumption remains a challenging problem. This paper presents our study on the energy consumption characteristics of an NVIDIA AGX Xavier GPU, the latest commercially available embedded hardware, under different concurrency levels and kernel scheduling orders. Our findings pave the way for designing an energy efficient scheduler for GPUs with real-time guarantees.**

I. Introduction

Nowadays, GPUs are becoming popular due to their outstanding performance. Offloading tasks that require a massive amount of computation and parallelism to the GPUs brings a significant performance improvement for cyber-physical and autonomous applications. Real-time multitasking is an essential prerequisite for developing such GPU-accelerated applications. For example, users can create multiple streams and assign independent kernels to those streams for concurrent kernel execution, in order to achieve speed-up and improve GPU resource efficiency.

Power management is one of the major factors for the efficient use of GPUs in an embedded environment. According to [1], GPU power management can bring multiple benefits, such as reducing the energy waste caused by kernel synchronization and resource utilization, improving scalability and reliability through reduced component temperature, and preventing the need for extra cooling.

Kernel scheduling order also has an important role in the execution efficiency of GPUs. A kernel contains a set of thread blocks (TBs), each of which has multiple threads. The number of TBs (grid size) and the number of threads in each TB (TB size) are given when the kernel is launched. Then, TBs are distributed to Streaming Multiprocessors (SMs) for execution in a near round-robin manner [2]. The amount of resources available on each SM varies depending on the hardware architecture. The actual start time of each TB on its assigned SM is determined by various SM resources, such as shared memory, number of threads, and number of registers. It has been shown in prior work [3] that when multiple kernels are launched concurrently, the hardware behavior and performance may vary depending on their scheduling order.

This paper presents our experimental findings on the effect of kernel scheduling on GPU energy consumption. Based on the NVIDIA GPU scheduling policy that has been well illustrated in [2], we have constructed various controlled environments with different TB sizes of synthetic kernels on the latest Nvidia embedded GPU platform, AGX Xavier. We have found that there is a strong relationship between the power consumption and kernel scheduling decisions, i.e., sequential vs. concurrent and execution order, and using such information has the potential to build an energy-efficient real-time GPU scheduler.

The rest of the paper is organized as follows: Section II gives a brief summary of our experimental setup, Sections III and IV discuss the GPU energy consumption under different concurrency levels and kernel execution orders, and Section V concludes the paper.

II. Experimental Environment

The experiments are done on a Jetson AGX Xavier Developer Kit using CUDA 10.0 SDK. The Xavier platform has an integrated 512-core GPU, sharing 16GB memory with the CPU. The GPU has 8 SMs, each containing 64 CUDA cores. The maximum number of active threads per SM is 2048.

GPU power consumption is measured using `tegrastats` [4], and the energy consumption is obtained by integrating the power consumption records over the duration of kernel execution. To minimize measurement inaccuracies, we disabled the dynamic voltage/frequency scaling (DVFS) of GPU and used a set of fixed clock frequencies during experiments. By default, the platform is running under 15W mode, in which the GPU clock frequency is set to 670MHz and 4 CPU cores are enabled. The GPU can be configured to run at other frequencies, e.g., 520, 900, and 1377MHz. However, since the energy consumption behavior under different frequencies has similar patterns in our experiments, we report only the results obtained at 670MHz for simplicity.

III. Effect of Kernel Configuration and Concurrency

To explore kernel parameters and execution modes, we have created a set of compute-intensive synthetic kernels, each of which has a fixed amount of workloads for each thread used. Memory operations are not involved in these kernels in order to limit our focus to processing elements. The number of TBs for each kernel (grid size) is set to 8, which is equal to the total number of SMs on the target GPU and allows all TBs to be evenly distributed across all SMs by the hardware thread-block scheduler [2]. For the ease of experiments, we have developed

TABLE I: Execution time of synthetic kernels (grid size = 8)

TB size	Execution time (s)
1024	10.63
512	7.57
256	6.43
128	5.97
64	5.95
32	5.95
16	5.95
8	5.95

a tool that allows the user to configure the number of TBs and the TB size of individual kernels, and to define the number of kernels and concurrency levels in the experiment.

Observation 1. *The execution time of a kernel does not decrease linearly with the TB size.*

We first measured the execution time of individual kernels with different TB size (# of threads per TB) since the execution time is assumed to be directly related to the energy consumption. Table I shows the results. It is worth noting that in this table, the TB size value represents the relative amount of computational workloads of the corresponding kernel because each thread performs the same operations and all other parameters are the same across the kernels. The execution time reduces with the TB size and plateaus when the TB size gets smaller than 128. This implies that the GPU cannot parallelize well when the number of threads is too small, thereby likely leading to resource underutilization.

To understand the relation between concurrency level and energy consumption, we run a set of synthetic kernels under two execution modes: sequential and concurrent. In sequential mode, the next kernel in the set is launched only when the previous kernel has completed. On the other hand, in concurrent mode, the launch requests of all kernels in the set are submitted to the GPU at once, each with a separate CUDA stream.

TABLE II: Completion time of 16 kernels (grid size = 8)

TB size	Seq. (s)	Con. (s)	Con./Seq. (%)
1024	170.59	157.62	92.40
512	121.28	78.83	65.00
256	103.23	39.67	38.42
128	95.36	19.66	20.61
64	95.37	10.20	10.69
32	95.36	7.28	7.63
16	95.41	6.64	6.96
8	95.43	6.65	6.97

TABLE III: Energy consumption of 16 kernels (grid size = 8)

TB size	Seq. (J)	Con. (J)	Con./Seq.(%)
1024	636.53	641.70	100.81
512	346.27	324.88	93.82
256	231.49	162.96	70.40
128	175.19	78.59	44.86
64	146.35	40.44	27.63
32	131.92	22.45	17.02
16	131.48	18.71	14.23
8	117.25	16.57	14.13

Observation 2. *For a given number of kernels, the concurrent execution mode improves the execution time and energy consumption compared to the sequential mode, but the amount of the improvement varies depending on the TB size.*

Tables II and III compare the completion time and energy consumption of a set of 16 kernels under the sequential and concurrent execution modes. All kernels have the same grid size (# of TBs per kernel). In general, the concurrent mode gives a large improvement when the TB size is small because many of such small TBs can execute concurrently on the same SM. Reducing the TB size by half, however, does not lead to half the execution time or energy consumption, e.g., the TB size from 512 to 256 under concurrent execution. Also, the ratio of improvement in time and energy differs.

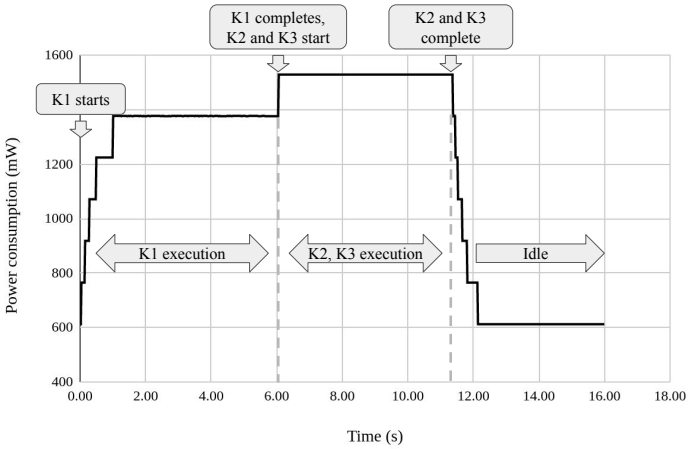


Fig. 1: Power consumption curve of three identical kernels

Observation 3. *GPU power consumption is affected by the concurrency level, but not proportional to the amount of workloads.*

Fig. 1 shows an example power consumption curve when we launch three identical kernels, K1, K2, and K3. Each kernel has a TB size of 32 and a grid size of 8. We first launch K1 at time 0 (Phase 1), and right after K1 finishes, launch K2 and K3 concurrently (Phase 2). The curve represents the instant power consumption obtained from the `tegrastats` utility, and the dashed line indicates the time when each phase ends.

As shown in Fig. 1, at the beginning, when the kernel instance is about to be launched, power consumption is as low as about 600 mW. Note that this is due to static power consumption and cannot be zero. As soon as the first kernel starts, the power consumption increases to a relatively high level. Then, when the two concurrent kernels start, the power consumption goes even higher because more GPU resources are utilized. Finally, after the completion of all the kernels, the power consumption is returned to the initial level. It is interesting to note that, the power consumption difference between Phase 1 and Phase 2 is merely about 150 mW, but the difference between the initial state and Phase 1 is significantly higher, about 750 mW.

TABLE IV: The results of sequential execution (TB size = 32)

Number of kernels	Duration (s)	Power (mW)	Energy (J)
1	5.95	1373	8.49
2	11.92	1377	16.79
4	23.85	1377	33.21
8	47.70	1377	66.02
16	95.36	1377	131.92
32	190.87	1377	263.23
64	381.55	1377	525.78
128	762.98	1377	1051.14

TABLE V: The results of concurrent execution (TB size = 32)

Number of kernels	Duration (s)	Power (mW)	Energy (J)
1	5.95	1373	8.49
2	5.96	1530	9.54
4	5.97	1836	11.34
8	6.28	2294	14.60
16	7.28	3212	22.45
32	14.45	3212	45.32
64	29.17	3212	90.06
128	58.32	3212	178.93

Observation 4. *For a set of identical kernels, the total completion time is not linear to the number of kernels under the concurrent execution mode.*

We also discuss the impact of the number of kernels when all the kernels are identical. Tables IV and V show the completion time (duration), power, and energy consumption of a given number of kernels under different execution modes. Under the sequential mode (Table IV), power consumption remains always the same regardless of the number of kernels. This is obvious since this mode executes only one kernel at a time. Energy consumption is given as a linear function of the execution time. However, a completely different observation is made under the concurrent mode. The power consumption increases with the number of kernels sub-linearly until the number becomes 16. In case of the energy consumption, it increases only marginally until 16 (e.g., the increase in the number of kernels from 1 to 8 causes only about 70% energy increase), and then increases linearly with the number of kernels. In other words, under the concurrent execution mode, power and energy consumption do not necessarily grow proportionally to the number of kernels.

Observation 5. *Concurrent execution mode generally outperforms sequential mode in terms of energy consumption.*

This observation is supported by all the above results comparing the energy consumption of the two modes. Except for one case (TB size = 1024 in Table III), a considerable amount of energy is saved under the concurrent execution mode. This effect is significant particularly when the TB size gets smaller and the number of kernels is large, up to 83% energy is saved in our experimental results.

IV. Effect of Kernel Scheduling

Based on the aforementioned results, further experiments are done to show that the energy consumption also differs when the kernel scheduling order is changed. As listed in Table VI, we use 40 independent kernels, K1, K2, ..., K40,

TABLE VI: Info of kernels used in the example

Group	Kernel ID	TB size	Exec. time (s)
1	1, 6, 11, 16, 21, 26, 31, 36 2, 7, 12, 17, 22, 27, 32, 37	128	5.95
2	3, 8, 13, 18, 23, 28, 33, 38	256	6.42
3	4, 9, 14, 19, 24, 29, 34, 39 5, 10, 15, 20, 25, 30, 35, 40	1024	10.62

which fall into 3 groups based on their TB size, 128, 256, and 1024. The grid size of each kernel is chosen to be 8, which is the number of SMs on the Xavier GPU as mentioned earlier. This allows us to easily keep track of the TB execution on each SM. Each kernel is assigned a separate CUDA stream.

For this configuration, the following three kernel scheduling approaches are considered: (i) sequential, (ii) concurrent, and (iii) partially concurrent scheduling. First, sequential scheduling serializes all kernel execution and launches the kernels in their ID order. Secondly, concurrent scheduling reorders kernels to maximize the concurrency and utilization of the GPU. Lastly, partially concurrent scheduling allows concurrent kernel execution but limits the number of concurrent kernels to two at any time.

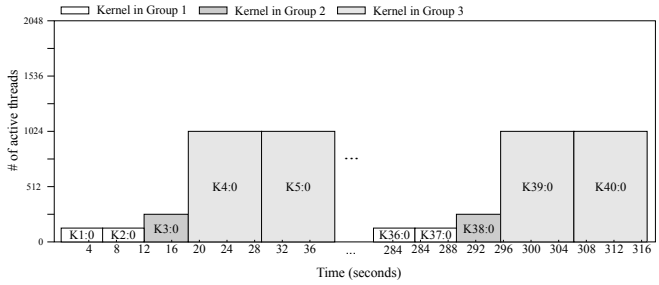
Fig. 2 illustrates the execution timeline of TBs of all kernels under the three scheduling approaches. For simplicity, we only show the results from the first SM of the GPU, SM 0, but the other SMs have the same results since all SMs are assigned the same number of TBs for each kernel. In this figure, $K_{i:j}$ denotes the j-th TB of a kernel K_i. The x-axis is time, and the y-axis represents the number of active threads on that SM. Note that the maximum number of active threads per SM supported by the Xavier GPU is 2048.

As can be seen, sequential scheduling (Fig. 2a) executes only one kernel at a time and yields the longest completion time. In contrast, concurrent scheduling (Fig. 2c) gives the shortest completion time by launching the kernels in the order of groups 3, 2, and 1. For kernels in Group 3, each kernel has 1 TBs with 1024 threads, and two of them thus fully occupy all the available threads of an SM. Then, 8 kernels in Group 2 execute simultaneously, and all 16 kernels of Group 1 do the same. Partially concurrent scheduling (Fig. 2b) shows mixed results. Its completion time is longer than the concurrent but shorter than the sequential scheduling approach due to its limited concurrency level.

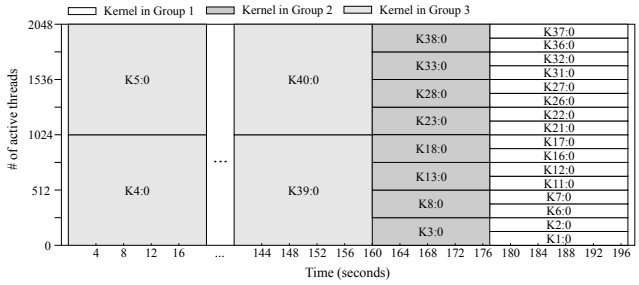
TABLE VII: Results of kernel scheduling approaches

Scheduling	Duration (s)	Idling time (s)	SM util.	Energy (J)
Sequential	316.52	3.48	30.76%	931.14
Concurrent	196.91	123.09	90.50%	867.98
Partially concurrent	211.67	108.33	45.98%	851.36

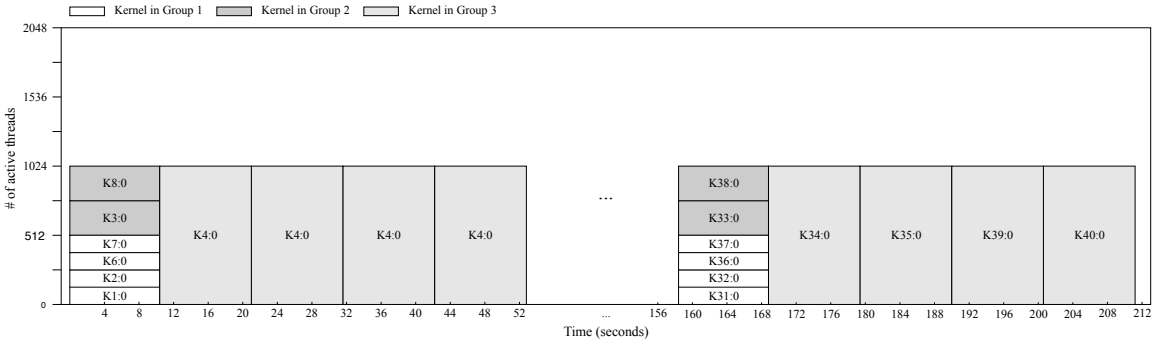
Table VII summarizes the experimental results under the three scheduling approaches. We choose an observation window of 320s, which is slightly larger than the longest completion time among three scheduling method, in order to take into account the impact of idle time in the energy consumption of given workloads. The duration in the table means the actual execution time of kernels, and the idling time is the remaining



(a) Sequential scheduling



(b) Concurrent scheduling



(c) Partial concurrent scheduling

Fig. 2: Kernel execution timeline on Xavier GPU SM 0

time during which no active thread is running on the GPU and only static power consumption exists. SM utilization means the average thread occupancy during the execution of all kernels, and is calculated by:

$$\frac{\sum (t_i \times TB_i \times n_i)}{S_{thread} \times t_{total}}$$

where t_{total} is the total duration of kernel execution, t_i is the execution time of an individual kernel K_i, TB_i is the TB size of K_i, n_i is the number of TBs per SM for K_i, and S_{thread} is the maximum number of active threads per SM supported by the GPU ($= 2048$ on Xavier).

While the partially concurrent scheduling does not give the shortest completion time, interestingly, it yields the lowest energy consumption among all three approaches in this experiment (1.91% reduction from concurrent scheduling, and 8.57% reduction from sequential scheduling). The trend may vary depending on kernel configurations and concurrent scheduling can outperform the other two in other cases. Nevertheless, our experimental results encourage the development of a new scheduler design for optimal GPU energy consumption.

V. Conclusion

This paper presents the GPU energy consumption behavior with different concurrency levels and kernel scheduling orders. We have demonstrated that concurrent kernel scheduling has the potential to improve execution time and energy consumption, but maximizing the concurrency level does not lead to the most energy-efficient schedule. Therefore, it is worth comparing different kernel schedules with respect to energy consumption, as long as they meet real-time requirements.

It should be, however, noted that we used independent CUDA kernels and no memory operations were involved in our experiments. Although we have not verified experimentally, it is expected that transferring data and accessing memory will create indirect dependency among logically-independent kernels due to shared memory resources (e.g., copy engines, buses, and caches) and may lead to different observations.

For future work, we plan to investigate the energy consumption of kernels with memory operations, take into account temporal dependency among kernels due to shared resources, and develop an energy-efficient real-time GPU scheduler. We believe our findings in this paper serve as a good starting point for these directions.

References

[1] S. Mittal and J. Vetter, "A Survey of Methods For Analyzing and Improving GPU Energy Efficiency," *ACM Computing Surveys*, vol. 47, 04 2014.

[2] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2017, pp. 104–115.

[3] R. A. Cruz, C. Bentes, B. Breder, E. Vasconcellos, E. Clua, P. M. de Carvalho, and L. M. Drummond, "Maximizing the GPU resource usage by reordering concurrent kernels submission," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 18, p. e4409, 2019, e4409 cpe.4409. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4409

[4] "NVIDIA Xavier - JetPack 4.1 - Performance Tuning - Evaluating Performance kernel description," https://developer.ridgerun.com/wiki/index.php?title=Xavier/JetPack_4.1/Performance_Tuning/Evaluating_Performance.