Power-Aware Characterization and Mapping of Workloads on CPU-GPU Processors

Kapil Dev
School of Engineering
Brown University
Providence, RI 02912
Email: kapil_dev@brown.edu

Xin Zhan
School of Engineering
Brown University
Providence, RI 02912
Email: xin_zhan@brown.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI 02912
Email: sherief_reda@brown.edu

***Abstract*—Modern CPU-GPU processors enable workloads to run on both CPU and GPU devices. Current scheduling practices mainly use the characteristics of kernel workloads to decide the CPU/GPU mapping. We observe that runtime conditions such as power and CPU load also affect the mapping decision. Consequently, in this paper, we propose techniques to characterize the OpenCL kernel workloads during run-time and map them on appropriate device under time-varying physical (i.e., chip power limit) and CPU load conditions, in particular the number of available CPU cores for the OpenCL kernel. We implement our Power-Aware Scheduler (PAS) on a real CPU-GPU processor and evaluate it using various OpenCL benchmarks. Compared to the state-of-the-art kernel-level scheduling method, the proposed scheduler provides average improvements of 31% and 4% in runtime and energy, respectively.**

I. Introduction & Motivation

Heterogeneous processors, with integrated CPU and GPU devices, offer great balance between performance and power efficiency for a wide range of applications [2]. Workloads can be scheduled on CPU and GPU devices simultaneously on these processors We observe that the existing schemes (e.g., as proposed by Wen *et al.* [3] and Choi *et al.* [1]) adaptively change the device decisions based on kernel characteristics alone and do not take the time-varying system conditions into account during the scheduling process. Therefore, they may lead to potentially inefficient scheduling under dynamic system conditions as shown in this work.

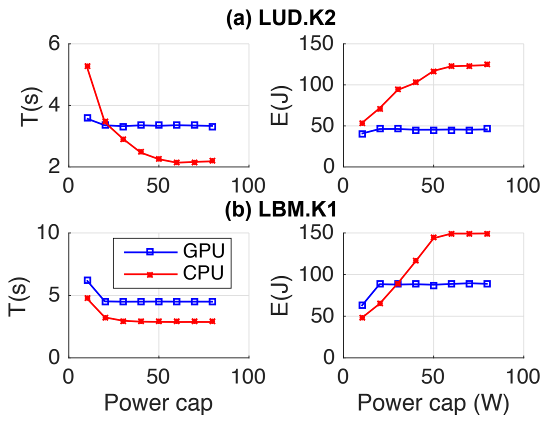

Fig. 1. Runtime and energy versus package power limit for two benchmarks: (a) `LUD` and (b) `LBM` on GPU and CPU devices of an Intel Haswell processor.

Need for Power-Aware Scheduling: The power cap of a processor can change dynamically for multiple reasons such as saving battery life, adapting to the user behavior in mobile system, or to meet different power budgets in server processors. We show the impact of the power cap on the scheduling decisions for runtime and energy for two example kernels in Fig. 1. We observe that for the `LUD.K2` kernel, GPU provides lower energy at all power caps, but the best device for runtime changes from GPU to CPU as the power cap is increased from 20 W to 80 W. Similarly, for the `LBM.K1` kernel, the best device for energy is a function of the power cap. These trends arise because of the differences in maximum operating frequencies and architectures of CPU and GPU devices and the relative differences in performance and power scaling of kernels on the two devices. A static approach based on fixed power cap of 80 W, compared to power-aware scheduler (`PAS`), incurs up to 1.5× higher runtime and 1.3× higher energy for `LUD.K2` and `LBM.K1` kernels, respectively.

Need for CPU Load-Aware Scheduling: Table. I gives the effect of number of available CPU cores (1C to 4C) on the runtime of two OpenCL kernels at three power caps (20, 40, and 80 W). Here, the other CPU cores are busy with running other workloads. The entries in the table denote the ratio of runtime on GPU over CPU device (so, > 1 means CPU faster than GPU and vice-versa). From the table, we notice that, at a fixed power cap (say 80 W), as the number of CPU cores available for the kernels (`LUD.K2` and `LBM.K1`) reduces, the best device for runtime could change from CPU to GPU. Thus, the scheduler should be aware of CPU-load conditions. In particular, a CPU load-unaware scheduler incurs up to 2.3× runtime overhead. Since the existing GPUs do not support running multiple kernels simultaneously, we focus on CPU load-aware scheduling in this work. Further, different

TABLE I
Ratio of GPU runtime over CPU runtime for two kernels (`LUD.K2` and `LBM.K1`) at 3 different power caps and 4 different number of CPU-cores for an Intel Haswell processor.

	LUD.K2				LBM.K1			
Power Cap	4C	3C	2C	1C	4C	3C	2C	1C
80 W	1.55	1.19	0.83	0.44	1.55	1.54	1.22	0.66
40 W	1.32	1.15	0.85	0.44	1.55	1.51	1.25	0.66
20 W	0.96	0.84	0.68	0.41	1.38	1.26	1.03	0.65

TABLE II
NORMALIZED RUNTIME AND ENERGY FROM "PAS" SCHEDULER AGAINST FOUR STATE-OF-THE-ART SCHEDULERS AT TWO POWER LIMITS (20 W AND 80 W) AND TWO CPU-LOAD CONDITIONS (ALL 4 CORES FREE AND ONLY 1 CORE FREE)

		Runtime					Energy				
Power Cap	#Cores available	GPU	CPU	App-level [1]	K-level [3]	PAS	GPU	CPU	App-level [1]	K-level [3]	PAS
80 W	4	1.00	0.55	0.61	0.44	0.44	1.00	1.49	0.90	0.72	0.72
20 W	4	1.00	0.74	0.69	0.54	0.53	1.00	0.89	0.83	0.64	0.62
80 W	1	1.00	1.45	0.92	0.77	0.63	1.00	1.21	0.85	0.62	0.62
20 W	1	1.00	2.06	1.10	0.96	0.66	1.00	2.05	0.89	0.69	0.67

kernels of an application can have different characteristics [3]. Therefore, we implement a kernel-level scheduler instead of simple application-level scheduler [1] to minimize runtime or energy across all kernels.

II. PROPOSED SCHEDULING FRAMEWORK & RESULTS

Motivated by the above-mentioned needs, in this section, we present our power-aware scheduler (PAS) for CPU-GPU processors to minimize runtime or energy.

Framework architecture. Our proposed framework has mainly two components: 1) a *monitoring daemon* to track the system conditions by reading model specific registers (i.e., running average power limit, RAPL) for chip power cap and the performance counter values (PMCs); 2) a *scheduler daemon* for making decisions based on the performance/energy goals and the run-time hardware conditions. The scheduler uses *a priori* trained support-vector machine (SVM) classifier to predict the best device that minimizes energy or runtime for the current kernel under given run-time conditions.

Offline SVM classifier. Similar to the previous work [3], we use SVM-based classifiers to predict the optimal device in our scheduler because for scheduling purposes, it is sufficient to predict the relative ratio of energy or runtime of each device (CPU/GPU) for a kernel; i.e., the exact values of runtime and energy on two devices are not needed. Our SVM classifier uses PMCs, chip power cap and the number of available cores as feature vector for predicting the best device. The training data is collected by running multiple OpenCL applications on our experimental system. Further, we build different SVM models for minimizing runtime or energy goals as the device decision for minimizing one is typically different than the other.

Online kernel characterization and mapping algorithm. During run-time, given a kernel, the scheduler first checks for the free CPU cores using the PMC values. If all cores are occupied by other workloads, the scheduler selects GPU as the default device; otherwise, the scheduler invokes the SVM model to predict the device using the current power cap, number of free cores and PMC values. The current device decision is then recorded for using in the future iterations of the kernel when similar dynamic conditions are encountered. This is done to minimize the overhead of the scheduler.

Results. We perform our experiments on an Intel Haswell Core i7-4790K CPU-GPU processor, which has 4 CPU cores and an HD graphics 4600 GPU with 20 execution units, each with 16 cores, integrated on the same die. We enforce the power cap using RAPL registers and use our scheduler to pick the better device mapping. To show the effectiveness of the proposed scheduler, we ran 13 OpenCL benchmarks (with 24 kernels in total) from three popular benchmark suites (Rodinia, Parboil, and Polybench) to cover wide range of workload characteristics. We evaluated our SVM model at different physical and run-time resource availability conditions; the classification error is below 3.1% across all conditions. Further, the overhead of custom APIs, SVM and kernel migration overheads is reasonably low ($< 2\%$). The improvement results, presented below, from our `PAS` method include these overheads.

Table II provides the comparison of runtime and energy from our `PAS` method against four state-of-the art schedulers under two power cap values (20 W and 80 W) and two CPU load conditions (all 4 cores available to OpenCL or 1 core available to OpenCL). The four schedulers are: `GPU`, `CPU`, `App-level` [1], and `K-level` [3]. The first two schedulers launch kernels always on GPU or CPU devices. In the `App-level` (short for Application-level) scheduler, all kernels of an application are mapped to the same device (CPU or GPU) to minimize the total application runtime [1]. Finally, the `K-level` (short for Kernel-level) scheduler uses offline-trained SVM classifier to make the best device decision on per kernel-basis [3], but unlike our `PAS` scheduler, it ignores the dynamic system conditions during scheduling. The table shows the average runtime and energy values for all 24 kernels; all values are normalized to the GPU cases. As we can see from the table, `PAS` scheduler provides lower runtime and energy than all other scheduling methods. Typically, the improvements from `PAS` method are higher at lower power cap and fewer number of CPU cores because the other methods (`App-level` and `K-level`) assume fixed power cap (80 W) and number of available CPU cores (4); they do not take into account the external run-time conditions. In particular, the `PAS` provides up to 31% better performance and 4% lower energy than the `K-level` scheduler.

In summary, the proposed scheduler provides better performance and energy efficiency by considering both run-time physical and CPU load-conditions during scheduling process.

Acknowledgment: This work is partially supported by NSF grants 1305148 and 1438958.

REFERENCES

[1] H. J. Choi et al. An Efficient Scheduling Scheme Using Estimated Execution Time for Heterogeneous Computing Systems. *The Journal of Supercomputing*, pages 886–902, 2013.
[2] S. Mittal and J. S. Vetter. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comp. Surv.*, 47(4):1–35, July 2015.
[3] Y. Wen, Z. Wang, and M. O'Boyle. Smart Multi-Task Scheduling for OpenCL Programs on CPU/GPU Heterogeneous Platforms. In *Intl. Conference on High Performance Computing*, pages 1–10, 2014.