Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 108C (2017) 2353–2357
Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Efficient OpenCL-based concurrent tasks offloading on accelerators

A.J. Lázaro-Muñoz[1], J.M. González-Linares[1], J. Gómez-Luna[2], and N. Guil[1]

[1] Dep. of Computer Architecture, University of Málaga , Spain
alazaro,jgl,nguil@uma.es
[2] Dep. of Computer Architecture and Electronics, University of Córdoba, Spain
el1goluj@uco.es

Abstract
Current heterogeneous platforms with CPUs and accelerators have the ability to launch several independent tasks simultaneously, in order to exploit concurrency among them. These tasks typically consist of data transfer commands and kernel computation commands. In this paper we develop a runtime approach to optimize the concurrency between data transfers and kernel computation commands in a multithreaded scenario where each CPU thread offloads tasks to the accelerator. It deploys a heuristic based on a temporal execution model for concurrent tasks. It is able to establish a near-optimal task execution order that significantly reduces the total execution time, including data transfers. Our approach has been evaluated employing five different benchmarks composed of dominant kernel and dominant transfer real tasks. In these experiments our heuristic achieves speedups up to 1.5x in AMD R9 and NVIDIA K20c accelerators and 1.3x in an Intel Xeon Phi (KNC) device.

Keywords: OpenCL, Command Queue, Concurrency, Tasks scheduling, Commands Overlapping

1 Introduction

Some Application Programming Interfaces (API) such as CUDA [3] and OpenCL [1] provide features to overlap communication between CPU (namely the host) and accelerator (the device) with computation by employing CUDA streams or OpenCL command queues (CQ), respectively. Overlapping commands simultaneously increases kernel productivity[1] and accelerator use by reducing idle periods between kernel executions. Given a set of tasks, the achieved productivity depends on the order the tasks are offloaded as the scheduling policy affects the final overlapping degree. This fact is illustrated in Figure 1 where the execution time-line of four offloaded tasks employing two different orders is shown. Notice that in this example transfers from host to device (HtD) and device to host (DtH) can also overlap.

[1]Kernel productivity is defined as the number of kernels executed per time unit.

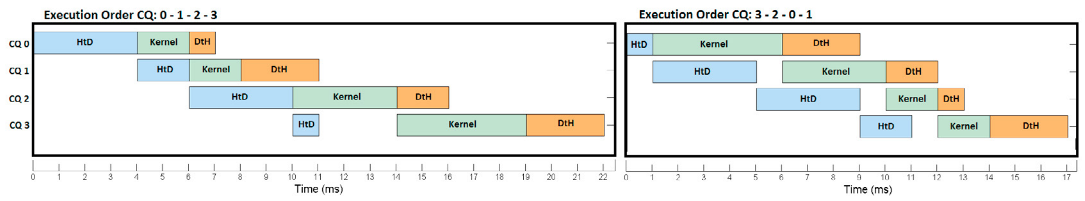

Figure 1: Concurrent execution of the same four tasks on a accelerator using two different orders.

The previous example shows that the order in which tasks are submitted to a device might have an important impact on the total execution time. Assuming several tasks are simultaneously available for offloading, finding the optimal order would require to test all possible orderings to choose the one resulting in the shortest execution time. However, this brute-force approach is not feasible in runtime since testing all possible combinations for N independent tasks involves evaluating $N!$ different orderings. In this paper this issue is addressed by proposing a runtime approach that selects a near-optimal ordering for concurrent task offloading using OpenCL CQs.

2 Asynchronous command execution in OpenCL

A CQ is a software queue used by the host application to submit commands to the device. The proposed scheme to manage CQs is shown in Figure 2 for devices with two DMA engines such as AMD R9 and NVIDIA K20c. This scheme depicts how three tasks are launched using three OpenCL CQs. Two queues are employed for HtD and DtH commands because each DMA engine independently executes the commands. Although in this example only one CQ (CQ2) is employed to submit kernel execution commands, Concurrent Kernel Execution (CKE) could be feasible by using different CQs per kernel command. Notice that in this scheme the host thread submits commands in task order (all the commands of a task sequentially).

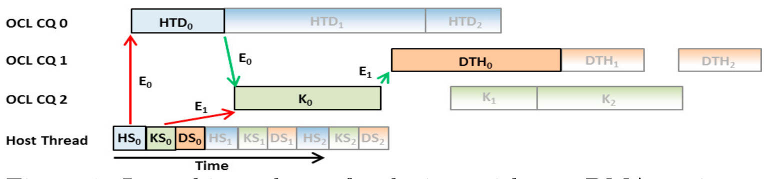

Figure 2: Launching scheme for devices with two DMA engines.

Since the memory transfer and kernel commands belonging to a task are launched to different CQs, inter-task dependencies must be inserted employing OpenCL events. In Figure 2 we show these dependencies. For the sake of clarity, they are only shown for task 0. Thus, when a HtD or K command is submitted by the host thread (indicated as HS_0 or KS_0) an OpenCL event is also associated to this command (indicated as E_0 or E_1 in Figure 2). These events can be used to query the status of the command, that is, if it is queued, submitted, running, or finished. Red and green arrows are drawn to indicate, respectively, the moments when the event is submitted and completed. Hence, the K_0 command execution is delayed until E_0 reaches the completed state. Similarly, the DtH_0 command execution does not start until E_1 is completed.

In case of devices with only one DMA engine, our command mapping scheme uses two CQs. All transfer commands are sent to CQ0 (first HtD commands, and then DtH commands) while

K commands are sent to CQ1.

3 Task reordering

Taking into account the OpenCL task submission schemes already explained, we propose a model with three FIFO software queues to simulate the computation of a group of tasks (TG) that are simultaneously available for offloading. Each queue is devoted to the simulation of a different command type. Figure 3 shows our model for a device with two DMA engines. In this figure, the head of each queue has been highlighted with a blue dotted rectangle. Thus, HtD_2, K_1 and DtH_0 commands are being executed (or ready to be executed). Similarly, white boxes represent commands that have already been executed, while the remaining commands are waiting for the fulfillment of the implicit (FIFO) and explicit dependencies. Since there exist dependencies among the commands belonging to a task, these software queues are not independent. Green arrows between commands from different queues represent dependencies due to ordering inside each task.

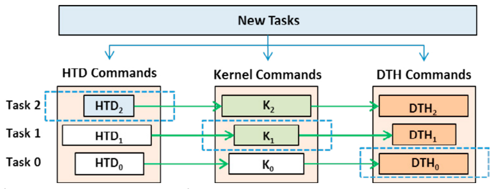

Figure 3: Model for simulation of concurrent task execution on accelerators.

The simulation performed by our model is based on a previous calculation of the execution time of each OpenCL command. Thus, we have improved the transfer model presented by Werkhoven et al. [4] to cope with arbitrary overlapping factors between HtD and DtH transfers. In addition we have used a simple kernel execution model [2] that is suitable for the experiments presented in this paper. If necessary, a more complex framework could be used.

3.1 Task reordering runtime

The search for an optimum order to offload an arbitrary group of tasks must be performed at runtime. Brute-force approaches are not feasible because computation of the execution time for all possible orderings might be very high. Thus, we propose a heuristic based on the estimated execution times calculated by our model. It tries to minimize the inactivity periods of a device when a group of tasks is submitted by looking for the ordering with the highest number of commands that can be concurrently executed according to the device hardware restrictions.

The heuristic starts with the selection of the first task. It is selected among tasks with a short HtD command and a long K command when compared to the remaining tasks in RT (set of remaining task to order). This way, device inactivity is reduced at the beginning of the execution and, in addition, overlapping options for the following tasks are leveraged. The first selected task, T_{ini}, is added to OT (set of ordered tasks) and removed from RT.

While there are tasks available in RT, tasks selection is accomplished by looking for the best fit between the remaining K commands of the previously selected tasks and the HtD command of the new task, and between the remaining DtH commands of the previously selected tasks and the K command of the new task.

Device	Task	MM	BS	FWT	FLW	CONV	VA	MT	DCT
	Dominance	DK	DK	DK/DT	DK	DK	DT	DT	DT
AMD R9	HtD (ms)	0.97-2.57	0.08-1.29	1.29-2.57	0.05-0.07	0.09-0.37	0.65-3.86	2.57-5.15	2.57-5.15
	Kernel (ms)	1.80-9.02	2.98-5.57	2.59-5.47	7.77-10.08	1.51-14.58	0.05-0.30	0.29-3.59	0.95-1.89
	DtH (ms)	0.14-1.18	0.16-2.17	1.18-2.35	0.09-0.16	0.09-0.37	0.30-1.81	2.36-4.70	2.35-4.71

Table 1: Range of execution times of *HtD*, *K* and *DtH* commands for real tasks in AMD R9.

4 Experimental results

Table 1 resumes the selected real tasks for the experiments alongside their classification as dominant kernel tasks (DK) or dominant transfer tasks (DT). These tasks have been selected from NVIDIA and AMD OpenCL SDK and are the following: Matrix Multiplication (MM), Black Scholes (BS), Fast Walsh Transform (FWT), Floyd Warshall (FLW), Separable Convolution (CONV), Vector Addition (VA), Matrix Transposition (MT) and Discrete Cosine Transform (DCT). In order to increase the variability of the benchmarks, each task has been executed using several sets of parameters. In Table 1 the range of execution times for the commands belonging to the real tasks using the different sets of parameters is shown for AMD R9 device. These real tasks have been combined in several benchmarks. Thus, in benchmark $BK0$ every task is transfer dominant, $BK25$ has a 25% of kernel dominant tasks and so on.

In the experiments we consider T sets of independent tasks with T taking values of 4 and 6. In each set, a batch of N dependent tasks is available with N taking values of 1, 2 or 4. The $T \cdot N$ tasks are randomly selected from the corresponding benchmark. Two experimental setups, using two or three CQs as indicated in Section 2, are defined to establish how good is the order calculated by the heuristic compared to all possible tasks orders.

NoReorder setup. A thread asynchronously submits the commands of $T \cdot N$ tasks, taking into account the imposed dependencies between task belonging to the same batch. Each experiment randomly selects the $T \cdot N$ tasks and carries out fifteen executions of all possible tasks permutations ($(T!)^N$). No reordering is applied to these tasks (the standard offloading method is followed).

Heuristic setup. This setup considers T worker threads launching N consecutive tasks per thread. Thus, the maximum number of concurrent tasks in a TG is T. For each experiment, the same tasks selected for the NoReorder setup are employed. Workers write OpenCL API calls corresponding to the tasks launching in a common buffer. Dependencies between the tasks launched by a worker are enforced by imposing that a new task is not written in the buffer until the previous task has completely finished. Host proxy thread reads the common buffer, applies the heuristic to calculate a better tasks order, and submits the commands of reordered tasks. Finally, once the host proxy thread submits the HtD command of the last task belonging to the current TG, it polls again the common buffer and repeats the cycle.

Figures 4.a, 4.b and 4.c depict the achieved results by real benchmarks in AMD R9, NVIDIA K20c and Intel Xeon Phi (KNC) devices respectively. The results show the speedup achieved by the geometric mean (cross symbol) and the minimum (blue rectangle) execution times of the NoReorder setup with respect to the maximum execution time (blue rhombus) of the same setup. This way, we can visualize the range of speedup values achieved for all possibles task orders permutations in the NoReorder setup (vertical segment with blue rectangle and rhombus end points). All possible permutations has been evaluated for the NoReorder setup using four workers (T=4) and N=1, 2 and 4. In case of T=6, all the permutations are run for N=1 but only a subset containing the 5% of all possible permutations are used for N=2. As Xeon Phi has only one DMA engine, experiments have been conducted with all the possible permutations for N=1 (N=2 and N=4 produce the same speedup results). In addition, the speedup achieved

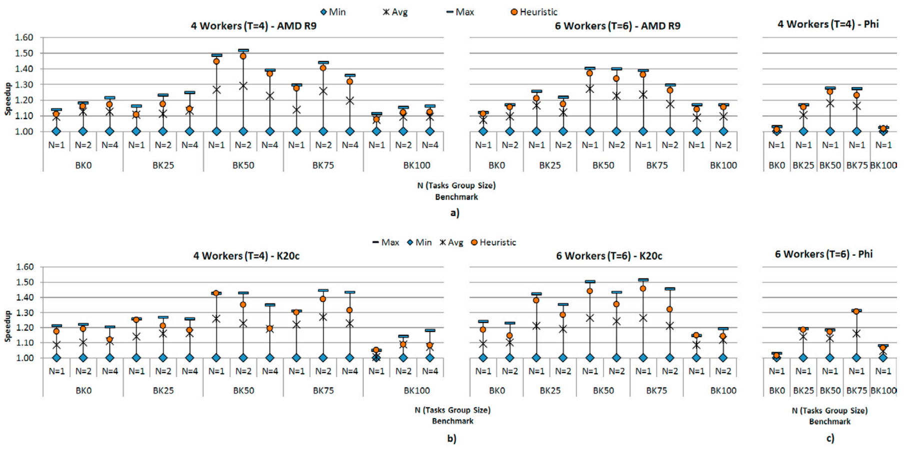

Figure 4: NoReorder setup is compared with the heuristic for a) AMD R9 b) NVIDIA K20c, and c) Xeon Phi (KNC).

by our heuristic with respect to the experiment giving the maximum execution time in the NoReorder setup (red circle) is indicated. The achieved results show that our heuristic predicts orderings very close to the best permutation most of the time, and always better than the mean execution time achieved by the NoReorder setup. It can be observed that our heuristic obtains higher speedups for BK25, BK50, BK75 as these benchmarks contain different types of tasks (kernel and transfer dominant) ergo better opportunities for command overlapping can be found.

5 Conclusions

We have presented a new strategy based on the fact that given a set of independent tasks to be executed in an accelerator, the tasks offloading order can have an important impact on the total execution time. Our approach proposes a runtime heuristic that is able to find a near-optimal order. This order obtains a high degree of overlapping between execution and transfer commands. We have also successfully tested our proposal in a multithreaded scenario where several worker threads are submitting real kernels to Intel, AMD and NVIDIA accelerators.

Acknowledgments

This work has been supported by the Ministry of Education of Spain (TIN2013-42253P) and the Junta de Andalucía of Spain (TIC-1692).

References

[1] Khronos Group. Opencl 2.0 api specification, October 2014.

[2] B. Liu, W. Qiu, L. Jiang, and Z. Gong. Software pipelining for graphic processing unit acceleration: Partition, scheduling and granularity. *International Journal of High Performance Computing Applications*, 2015.

[3] NVIDIA. Cuda programming guide, September 2015.

[4] B. van Werkhoven, J. Maassen, F.J. Seinstra, and H.E. Bal. Performance Models for CPU-GPU Data Transfers. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 11–20. IEEE, May 2014.