Towards the inclusion of FPGAs on Commodity Heterogeneous Systems

María Angélica Dávila Guzmán, Rubén Gran Tejero, María Villarroya Gaudó & Darío Suárez Gracia
Instituto de Investigación en Ingeniería de Aragón (I3A)
Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, SPAIN
angelicadg@unizar.es, rgran@unizar.es, mvg@unizar.es, dario@unizar.es

INVITED PAPER

***Abstract*— Nowadays, most commodity heterogeneous systems consist in either a CPU+GPU or CPU+FPGA. An attractive alternative consists in merging both in a new class of heterogeneous system: CPU+GPU+FPGA in order to combine the advantages of all of them into a single platform.**

In a three-device heterogeneous system, similarly to a two-device system, we have to face problems like: programmability versus device performance, data-buffers management and workload distribution. In this work, we present the first steps into a runtime which deals with theses problems for a system with a CPU+GPU+FPGA.

Index Terms*— *Heterogeneous System; CPU; GPU; FPGA; Runtime; Load Balance

I. Introduction

Until a few years ago, each new technological node allowed manufacturing computing devices with more transistors that were faster and consumed less energy than the previous generation. Dennard scaling has been supporting this effect for CMOS technologies: the power density, power dissipated per unit area, have remained constant generation after generation. When changing node technology, the supply voltage was reduced, and a greater number of transistors could be integrated per unit of area [1]. Nevertheless, in the recent nanometric CMOS technologies, the requirements for a reliable operation of the transistors have forced to maintain their supply voltage, so the power density has increased. That is, in a constant area there are more and more transistors consuming energy, and, therefore, dissipating power becomes almost impossible, since we are talking about 100 watts per cm2, greater than the surface of a frying pan. Because of this, many transistor can not be active at the same time, and, the dark silicon effect emerges. Nowadays, novel approaches are required in order to sustain the growing computation requirements of future systems.

A viable candidate to tackle this problem are heterogeneous systems. Heterogeneous systems group different types of processors architectures into the system in order to provide an specialized hardware for each different task. Many heterogeneous systems rely on a general purpose CPU and either a discrete or on-chip GPU (Intel Core, AMD Ryzen). An example of the broad adoption of this heterogeneous approach is that the first positions in the list of TOP500 supercomputers rely on a CPU+GPU tandem.

Beyond solutions in the field of digital design, FPGAs have shown great potential in solving problems very efficiently, especially from the energy consumption point of view [2]. Unfortunately, application development on a FPGA requires knowledge of digital design, which has been the main obstacle preventing their broad adoption by programmers. To mitigate this problem, High-Level Synthesis frameworks with languages like C, C++, or OpenCL improve programmer's productivity [10]. In consequence, FPGA's are also a solid proposal for heterogeneous systems and then industry not only offers discrete FPGA's but also on-chip CPU+FPGA (f.e. Altera DE-1, Xilinx Zynq-7000).

In this work, we focus on the combination of these three different devices: CPU, GPU, and FPGA in a single runtime platform. Our aim is the complete integrations of these three different architectures in a cooperative and coordinated work for speeding-up single applications.

II. The Runtime

Our runtime is based on EngineCL [12], it acts as an OpenCL C++ wrapper to simplify the programming of heterogeneous devices and squeeze the performance out of them. This runtime support the input/output data-buffer management besides we add a work scheduler in charge to exploit parallelism among accelerator devices: CPU, GPU and FPGA [3].

The scope of our runtime are data-parallel applications which are divided into slices that are later delivered to available accelerators. Therefore, an important element of the runtime is the scheduler, that now, it is in charge of dividing the work among three accelerators. We have considered two different scheduling policies until now:

- Static. This algorithm requires and offline exploration that sets the distribution of the data-set and work in as many packages as devices that are in the system. After that, a single work package is delivered to each device.
- Dynamic. It divides the data-set in packages of equal size (*chunk*). The number of packages is well above the number of devices in the heterogeneous system. During the execution of the kernel, a master thread in the host assign packages to the different devices.

A. OpenCL Portability

One of the characteristics of OpenCL is the code portability for the different accelerators that provides an OpenCL driver. However, this functional portability does not translate into performance portability, specially in case of FPGA. For FPGAs, performance and energy efficiency depends mainly on three factors: initiation interval, clock frequency, and resource utilization. These three factors deeply depend on the OpenCL source code and the idioms that the compiler is able to recognize in order generate an FPGA optimized hardware design.

Due to the easy for FPGA of exploiting naturally pipeline parallelism, a task based kernel fits better than a NDRange; e.g., a task-based kernel in an application is near to 113000 $\times$ faster than the NDRange version. In contrary GPU behaves 523 $\times$ slower in case of a task based kernel in comparison to NDRange.

B. Host-Device Communications & Overlapping Computation

In a heterogeneous system, input/output data must be transfered among the host thread and the three different devices. In OpenCL, each device requires a *command_queue* in order to forward them orders like: kernel launch and send/receive buffer (Read/Write). Therefore, previously to computation kernel-launch input-data must be sent to device memory and after the computation output-data must come back to the host.

Communication among the host memory and device memory can be significant specially for discrete systems. So that, it is very important to take benefit of overlapping opportunities among the computation and communication. Moreover, each OpenCL driver can provide different PCI-express management policies to the point to not to allow concurrency among independent transfer operations.

C. OpenCL Driver and Synchronization

Synchronization mechanisms are critical for the operation of a runtime, included ours. OpenCL provide a synchronization primitive based on callback functions. Callbacks are asynchronous functions that are invoked on behalf of the application in response to a certain event; i.e., kernel completion. The execution model of these callback functions is inherently concurrent to the execution of the host application, and therefore, it could leverage parallelism. However, this is not possible if the OpenCL driver is not tolerant to several threads running in parallel driver code (thread-safe). In such a case, a conservative approach consists in limiting the number of OpenCL-driver threads to one which impedes any parallelism for callbacks functions.

III. Preliminary Results

This section analyzes the behavior of the two load balancing policies: Static and Dynamic on an heterogeneous system composed by a CPU, GPU, and FPGA.

Fig 1 shows the normalized execution time for single devices without overlapping communication and computations, labeled as CPU, GPU, and FPGA. We also include normalized execution time for all 3 devices in cooperation with *static* and *dynamic* schedulers. Our heterogeneous system is composed by an Intel core i7-6700k CPU, a NVIDIA GeForce GTX TITAN X GPU, and an Altera DE5NET Stratix V GX FPGA, and each device run OpenCL 2.0 LINUX, OpenCL 1.2 CUDA 9.1.83, OpenCL 1.0 FPGA Version 16.1.2, respectively. We use four benchmarks: Matrix Multiplication, Mersenne Twister, Sobel Filter and Watermarking. In Fig 1, considering single accelerator configuration, there is not a clear winner, and depending on the benchmark, some devices behave better than the rest.

When considering the heterogeneous system, schedulers are able to orchestrate both three accelerators and perform slightly better than the best single accelerator option. However, neither the static nor dynamic scheduler is better than the other. Therefore it is still needed an alternative able to find out always the best distribution.

a) Static: With static scheduling, the user has to choose the amount of work each device performs, and if percentages are not right, performance degrades considerably. Therefore, Static scheduler requires from a exploration phase in order to set the proper proportion of work to be delivered to each device. Moreover, this exploration phase is dependent on the problem, the input data and the devices themselves and it must be reevaluated in case of change of any of them. This scheduler minimizes the number of synchronization points, therefore, it performs well when facing regular loads (Matrix Multiplication, Mersenne Twister, Watermarking) with known computing powers that are stable throughout the data-set. Sobel Filter is also a regular load, however, the exploration phase does not find the optimal workload distribution in the given number of exploration experiments (100 exploration experiments per benchmark). In this case, it is required a fine grain exploration.

b) Dynamic: With this scheduler, each device fetches and executes chunks of work (equally-sized for all devices) until there is no work left; therefore, chunk size has a large impact on performance. Also, as chunk size shrinks, there are more opportunities to overlap computation and communication with dedicated command queues as described before. Similarly to *Static* scheduler it is needed an exploration phase in order to determine the optimal chunk size for each device. This algorithm adapts to the irregular behavior of some applications. However, each completed package represents a synchronization point between the device and the host, where data is exchanged and a new package is launched. This overhead has a noticeable impact on performance.

IV. Related Work

Scheduling among several devices has been widely addressed. For example, Navarro *et al.* proposed LogFit, an adaptive partitioning strategy to find the optimal chunk size of the GPU [11], [18], [17]. Pandit and Govindarajan presented FluidiCL where a CPU and GPU work on a shared iteration space, but each device starts from an end of the iteration space [16]. In the same sense, there are studies that focus on systems with a CPU+FPGA integrated on the same chip [13]. For an ample overview of load balancing techniques, please refer to Mittal and Vetter [9].

Luk *et al.* proposed Qilin, a framework providing adaptive mapping to distribute work between CPU and GPU devices running on top of TBB and CUDA, [8]. MKMD maps multiple kernel into multiple devices in a two-phase approach, the first phase assign kernels to devices and the second enables work-group level partitioning to keep all devices busy [7]. There are also open standards that annotate sequential code to be run on an accelerator device [15], [14], [4]. Industry solutions include Intel TBB, supporting GPU offloading with OpenCL [6] or Qualcomm Symphony SDK, supporting GPU and DSP offloading [5]. On the contrary, this work targets systems with three devices: CPU, GPU and FPGA.

V. Conclusions

In the horizon of the heterogeneous systems, a greater number of devices and the management of very different architectures are glimpsed. In order to provide a friendly framework to the programmer a runtime must be provided. This runtime must adapt to the specifics of each device and the driver that manufacturer provides. Beyond this, the schedulers tested are able to properly distribute work and leverage the cooperative parallel work of them. However, the still rely on off-line exploration phases which can be not affordable in some scenarios, besides there is not a best scheduler option according to our experiments.

Acknowledgment

The authors would like to thank Altera and Nvidia for their generous hardware donations through their respective University Programs. This work was supported in part by grants TIN2016-76635-C2-1-R (AEI/FEDER, UE) gaZ: T48 research group (Aragón Gov. and European ESF), and HiPEAC4 (European H2020/687698).

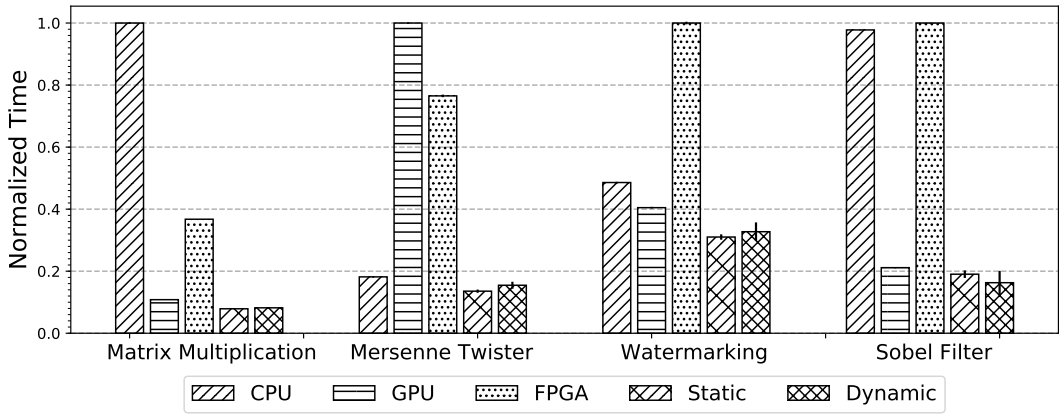

Fig. 1. Normalized execution time with respect to the worst single device version

REFERENCES

[1] M. Bohr. A 30 year retrospective on dennard mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, January 2007.

[2] D. Chiou. The microsoft catapult project. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 124–124, Oct 2017.

[3] Maria A. Dávila-Guzmán, Rubén Gran Tejero, Maria Villaroya Gaudó, and Darío Suárez Gracia. "first steps towards cpu, gpu, and fpga parallel execution with enginecl". In *Proceedings of the 18th International Conference on Mathematical Methods in Science and Engineering (CMMSE)*, Jul 2018.

[4] Swapnil Ghike, Rubén Gran, María J. Garzarán, and David Padua. Directive-based compilers for gpus. In James Brodman and Peng Tu, editors, *Languages and Compilers for Parallel Computing*, pages 19–35, Cham, 2015. Springer International Publishing.

[5] Qualcomm Inc. Symphony System Manager SDK, 2018.

[6] Alexei Katranov and Alexey Kukanov. Intel threading building block (tbb) flow graph as a software infrastructure layer for opencl-based computations. In *Proceedings of the 4th International Workshop on OpenCL*, IWOCL '16, pages 9:1–9:3, New York, NY, USA, 2016. ACM.

[7] Janghaeng Lee, Mehrzad Samadi, and Scott Mahlke. Orchestrating Multiple Data-Parallel Kernels on Multiple Devices. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2016-March:355–366, 2016.

[8] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. *Proceedings of the 42nd Annual IEEE/ACM Intr. Symp. on Microarchitecture - Micro-42*, page 45, 2009.

[9] Sparsh Mittal and Jeffrey S. Vetter. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Computing Surveys*, 47(4):1–35, 2015.

[10] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno. Efficient fpga implementation of opencl high-performance computing applications via high-level synthesis. *IEEE Access*, 5, 2017.

[11] Angeles Navarro, Francisco Corbera, Andres Rodriguez, Antonio Vilches, and Rafael Asenjo. Heterogeneous parallel_for template for cpu–gpu chips. *International Journal of Parallel Programming*, Jan 2018.

[12] Raúl Nozal, Jose Luis Bosque, and Ramón Beivide. EngineCL: Usability and Performance in Heterogeneous Computing. *arXiv*, May 2018.

[13] Jose Nunez-Yanez, Sam Amiri, Mohammad Hosseinabady, Andrés Rodríguez, Rafael Asenjo, Angeles Navarro, Dario Suarez, and Ruben Gran. Simultaneous multiprocessing in a software-defined heterogeneous fpga. *The Journal of Supercomputing*, Apr 2018.

[14] OpenACC Architecture Review Board. OpenACC application program interface version 2.6, November 2017.

[15] OpenMP Architecture Review Board. OpenMP application program interface version 4.5, November 2015.

[16] Prasanna Pandit and R Govindarajan. Fluidic kernels: Cooperative execution of OpenCL programs on multiple heterogeneous devices. *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, 2014.

[17] A. Vilches, A. Navarro, R. Asenjo, F. Corbera, R. Gran, and M. J. Garzarn. Mapping streaming applications on commodity multi-cpu and gpu on-chip processors. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1099–1115, April 2016.

[18] Antonio Vilches, Rafael Asenjo, Angeles Navarro, Francisco Corbera, Ruben Gran, and Mara Garzarn. Adaptive partitioning for irregular applications on heterogeneous cpu-gpu chips. *Procedia Computer Science*, 51:140 – 149, 2015. International Conference On Computational Science, ICCS 2015.