Integrating FPGAs in a heterogeneous and portable parallel programming model

Gabriel Rodriguez-Canal
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
grodriguez-canal@acm.org

Yuri Torres
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
yuri.torres@infor.uva.es

Arturo Gonzalez-Escribano
Departamento de Informática
Universidad de Valladolid
Valladolid, Spain
arturo@infor.uva.es

I. Introduction

FPGAs have attracted a wider scope of HPC researchers in the recent years thanks to the HLS (High Level Synthesis) languages and techniques, such as those introduced for OpenCL. They present a higher abstraction level, reducing the programming effort. However, the user still has to take care of details related to command queue management, kernel launching parameters or data transfers. This leads to cumbersome code that is difficult to develop and maintain and hinders the integration of FPGAs in heterogeneous systems.

More abstract languages are being introduced, like SYCL or DPC++ for Intel oneAPI. They use modern C++ concepts and programming abstractions, and they typically advocate a single source code for both host code and device kernels. A different approach can be found in the Controller [1] model. It is a heterogeneous programming model that enables performance portability across CPU-core sets, GPUs using CUDA or OpenCL and Xeon Phi accelerators. It introduces an event based model which is portable across CPUs, GPUs and FPGAs. It allows the synchronization of tasks avoiding the need to recompile the code targeting each different architecture or lower-level programming model. It is implemented as a library of functions in C99. It is compatible with any C99/C++ compiler and it is interoperable with other libraries and parallel programming models such as OpenMP. It uses an expandable library system that allows to integrate for any kernel both generic common code and specialized versions for different device vendors and families. Each kernel implementation can exploit lower level architecture features using programming models such as CUDA, OpenCL, or OpenMP. For example, different kernel versions can be written for Nvidia Fermi, Volta or Ampere architectures to better exploit their different cache sizes, synchronization primitives, etc. The runtime selects the most appropriate one depending on the target device chosen during the execution. This is specially important to integrate FPGAs in a model supporting devices such as GPUs, as the most appropriate kernels for GPUs and FPGAs may differ even at algorithm level. For example, implementations of stencil programs, such as Rodinia's Hotspot, on GPUs exploit their wide SIMD level and shared-memory capabilities, while efficient FPGA kernels are usually based on a completely different shift-register algorithm with a single SIMD lane. The Controller model also automatically detects and performs data-transfers between host and device when needed, and can use asynchronous techniques to overlap data-transfers with both host and device computations.

This work extends the Controller model with a new back-end for FPGAs that allows the automatic execution of kernels on these devices without modifying the host code of applications already implemented with this model for other types of devices. We present preliminary experimental results that show that our approach highly reduces the development effort comparing with using OpenCL directly, with an almost negligible performance overhead.

II. Support for FPGAs in Controller

To integrate FPGAs in the Controller model, we introduce a new backend module, rebuilding the pre-existing OpenCL back-end for GPUs. We present the following contributions:

- **Extension of the kernel library structure:** We extend the Controller kernel declaration mechanisms. We introduce support for offline compilation of OpenCL kernels with Intel AOC compiler for FPGAs, and dynamic load and execution of kernels. This improves the possibilities for building modular libraries of kernels for different types of devices.
- **FPGAs kernel configuration:** New optional parameters in the FPGA kernel interface allows the programmer a better tuning and improved efficiency of kernel implementations for specific devices. They allow the control of features such as the number of SIMD lanes or the number of replicas of compute units. They are used to generate different binaries at compile time that are added to the kernel implementations library.
- **Support for three execution modes:** The execution mode (either ordinary, emulation or profiling) is also determined in compile-time. Our proposal allows to pre-compile versions of the same kernel with the same parameterization in all the three modes. The execution mode is selected at runtime with an optional parameter during the creation of a controller object associated to an FPGA. The object internally selects the proper version when kernel launch operations are requested during the execution of the portable host code.
- **Support for incongruent grid sizes:** Data structures whose dimensions are not multiple of the work-group sizes were

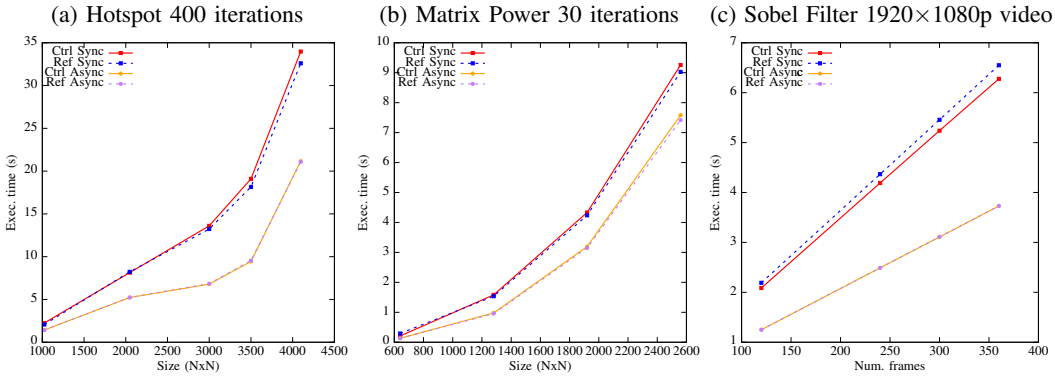

Fig. 1: Performance results for Hostpot and Matrix Pow (matrix size) and Sobel (iterations).

transparently supported in the Controller model through a boundary condition that tested whether a given thread was outside of requested boundaries. However, in FPGAs this would lead to *branch-divergence*, which is not supported by AOC in SIMD kernels. We introduce an alternative and transparent padding mechanism for data-structures that is more appropriate for FPGA kernels.

• **Synchronous and asynchronous execution policies with automatic data-transfers:** FPGAs support of synchronization mechanisms is sometimes limited comparing with GPUs. For example, devices may not allow full-duplex data transfers from-to the device simultaneously. Both the synchronous and asynchronous execution policies of the Controller model are adapted and supported in the new FPGA backend.

III. Experimentation

An experimental study is conducted to assess the performance efficiency of these new abstractions, and to compare the development effort reduction comparing with directly programming the FPGAs with OpenCL. Streaming implementations of three case studies that span a wide range of problems were chosen: Hotspot [2], Matrix Power [3] and Sobel Filter [4] iteratively applied to each frame of a YUV video file. Hotspot is a benchmark with a low computational cost kernel, whilst Matrix Power kernel has a higher order of complexity, leading to a much highly loaded kernel. Data transfers and host processing of partial results are added every fixed set of iterations to introduce opportunities to exploit asynchronous data-transfer and host/device computations overlapping. Fast kernels and data transfers make the Sobel Filter program very demanding in terms of this kind of concurrency exploitation. Every experiment was executed 10 times. We design scenarios that range from more costly data transfers than computations, to more costly computations than data transfers, etc. We explore several matrix sizes for Hotspot and Matrix Power, and different number of iterations for the Sobel Filter.

Figure 1 shows that the performance overhead introduced by Controller is negligible and the asynchronous policy automatically leads to performance improvements. In the Sobel case, the small data sizes lead to very fast kernel computations and data transfers, where the sophisticated and light task and event management of Controller show an advantage even with the synchronous policy. The results indicate an overhead of no more than 1% with a 95% confidence. There are reports showing that the performance obtained with other high-level models such as SYCL/DPC++ are also comparable with the performance obtained with OpenCL programs [5].

Case study	Version	LOC	TOK	CCN	Halstead
Hotspot	Ctrl	230	1772	40	919321
	OpenCL Sync	339	2771	57	1770315
	OpenCL Async	401	3273	53	2332285
Matrix Pow	Ctrl	148	1509	21	525721
	OpenCL Sync	211	1922	30	1243644
	OpenCL Async	271	2348	29	1646456
Sobel filter	Ctrl	137	1231	22	907566
	OpenCL Sync	202	1944	28	1207349
	OpenCL Async	290	2561	38	1689124

TABLE I: Measurements of development effort metrics for the reference and Controller codes.

Table I shows the results of four development effort metrics considered: Lines of codes (LOC); Number of tokens (TOK); McCabe's cyclomatic complexity (CCN) [6]; and Halstead Development Effort [7]. The use of the Controller model shows a high reduction in development effort. This is specially significant in the asynchronous version, due to the manual introduction of more complex mechanisms for kernel and data transfer synchronization in the OpenCL versions.

IV. Conclusions

This work extends the Controller heterogeneous programming model with a new back-end that supports FPGAs. The experimental results show that it achieves similar performance as directly programming with OpenCL, with a high reduction on development effort. Finally, on-going and future work include a direct comparison with SYCL/DPC++ using the same benchmarks, and integration of HDL kernels for a finer programmer tuning and higher performance gains. The software, case study programs and experimental data can be downloaded at http://trasgo.infor.uva.es/controller/.

Acknowledgement

This research has been partially supported by MICINN (Spain), the ERDF program of the European Union and Junta de Castilla y León: PCAS project (TIN2017-88614-R), Scottish Power Masters Scholarship, MECD (Beca de Colaboración, Spain), Salvador de Madariaga/Fulbright Scholar Grant (PRX17/00674) and Universidad de Zaragoza.

REFERENCES

[1] A. Moreton-Fernandez, H. Ortega-Arranz, and A. Gonzalez-Escribano, "Controllers: An abstraction to ease the use of hardware accelerators," *The International Journal of High Performance Computing Applications*, vol. 32, no. 6, pp. 838–853, 2018.

[2] H. R. Zohouri, "High performance computing with FPGAs and OpenCL," *arXiv preprint arXiv:1810.09773*, 2018.

[3] Intel, "Matrix Multiplication Design Example." https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/opencl/matrix-multiplication.html, 2018. [Online; accessed 1-August-2020].

[4] Intel, "Sobel Filter Design Example." https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/opencl/sobel-filter.html, 2018. [Online; accessed 1-August-2020].

[5] T. Deakin and S. McIntosh-Smith, "Evaluating the Performance of HPC-Style SYCL Applications," in *Proceedings of the International Workshop on OpenCL*, IWOCL '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[6] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[7] M. H. Halstead *et al.*, *Elements of software science*, vol. 7. Elsevier New York, 1977.