

FPGA Framework Improvements for HPC Applications

Antonio Filgueras^{*†}, Miquel Vidal[†], Daniel Jiménez-González^{*†}, Carlos Álvarez^{*†}, Xavier Martorell^{*†}

^{*}Universitat Politècnica de Catalunya

Email: {antonio.filgueras, daniel.jimenez-gonzalez, carlos.alvarez, xavier.martorell}@upc.edu

[†]Barcelona Supercomputing Center

Email: {antonio.filgueras, miquel.vidal, daniel.jimenez, carlos.alvarez, xavier.martorell}@bsc.es

Abstract—In modern FPGA devices, place and route has become an increasingly difficult task due to an increase in resources and device complexity. This results in an exponential increase of implementation possibilities. Such a huge search space causes tools to have a hard time providing a good solution. This is even more challenging in chiplet-based devices due to their topology. In the same way, off-chip memory resources have grown both in size and number of modules. These resources are presented to the user as raw memory interfaces requiring the user to manage how accelerator kernels access off-chip memory to make effective use of the available bandwidth. Efficient usage of memory resources becomes a critical challenge as the more computational resources are added to a design the more pressure they impose on the memory subsystem. This work proposes new features to the OmpSs@FPGA programming model in order to mitigate these issues in a transparent way for programmers.

I. INTRODUCTION

The framework on which this work is built, OmpSs@FPGA [1] [2], is a task-based programming model. In this model, users specify which parts of an application will be implemented as accelerators in the FPGA by labeling functions using pragma directives. Also, task data dependencies are specified in these directives. During compilation, the toolchain compiles and integrates computation kernels into a single hardware design. Then, when running the application, the runtime system will schedule each FPGA task to the accelerators when its data dependencies are satisfied. Features proposed in this work have been implemented in the integration phase of the toolchain, which will add the needed logic to the design with minimal user intervention.

II. NEW FEATURES

1) *Placement*: In order to mitigate the large costs of propagating signals across Super Logic Regions (SLRs), we have added new features that allow users to assign specific computation kernels to different SLRs. This feature serves two main purposes. First, constraining an IP core to a single SLR region prevents the implementation tool from placing it across different SLRs, which often leads to low operating frequencies and quality of results. This also allows to automatically place register slices on accelerator interfaces. These register slices are configured and constrained to specifically perform region crossing. They are placed around the region boundary and pipelined to propagate signals to points that are far away from the boundary without degrading operating frequency. Since

pipelining is applied to latency-insensitive interfaces, impact on performance is negligible.

2) *Priorities*: An adverse effect observed in some applications is memory access conflicts. When multiple accelerators access a single memory interface, transactions are processed in a round-robin fashion to provide all accelerators a fair use of the resources. This, however, prevents transaction pipelining, which decreases throughput as transaction initiation cannot be overlapped with data transfers. By prioritizing accesses, we enable transaction pipelining. Transactions from the accelerator with the highest priority will not be interrupted and therefore will be pipelined, increasing overall throughput. Moreover, this decreases the time accelerators are moving data, which further increases performance.

3) *Interleaving*: To maximize the number of parallel accesses to different memory interfaces, we have developed a transparent way to distribute data accesses among the available memory modules. This is implemented by inserting an interleaver module between any module accessing off-chip memory and the memory interconnection. This interleaver performs a series of bit operations to the memory addresses to distribute nearby accesses between different memory interfaces.

4) *Power measurement*: In order to evaluate the energy efficiency impact of the proposed features, we added power measurement infrastructure into the framework. This allows programmers to automatically query power usage and temperature statistics. This is done by instantiating a Card Management Subsystem (CMS) into the static logic of the design. The CMS module polls power delivery infrastructure and on-board sensors to allow programmers to retrieve power and temperature statistics.

III. RESULTS

The impact of proposed features has been evaluated using different applications. We use matrix multiplication, Cholesky decomposition, n-body, and spectra computation [3]. They are chosen because they are well-known applications that contain computation kernels that are relevant in the HPC context. For matrix multiplication, half, single, and double precision IEEE-754 floating point data is used. Cholesky decomposition and n-body operate using single precision floating point. Spectra uses fixed point computations to output double precision. All

Application	Num acc.	BS	Freq. (MHz)	Perf. (Gops/s)	Power (W)
MM half	8 / 9	256 / 256	300 / 350	548 / 762 ¹	69 / 82
Cholesky	4 / 6	256 / 256	300 / 333	255 / 448 ¹	72 / 98
N-body	8 / 8	2048 / 2048	300 / 333	37 / 42 ²	103 / 111
MM single	5 / 7	384 / 256	300 / 333	353 / 549 ¹	92 / 109
Spectra	10 / 10	2232 / 2232	300 / 333	51 / 57 ²	91 / 97
MM double	6 / 7	128 / 128	250 / 300	106 / 227 ¹	79 / 105

TABLE I: Performance, power and accelerator block size for all analysed applications. Data is presented as *Baseline / Improved*. ¹ Gflops ² Gpair/s

experiments have been run in an AMD Alveo U200 (XCU200-FSGD2104). Designs have been created using OmpSs@FPGA 3.3.1 and built using Vivado 2020.1 custom design flow.

Figure 1 shows a summary of the speedup (bars) and energy efficiency (dots) achieved by using the proposed features. The baseline version (labeled as *baseline*) is the best design we could implement for a given application without using any of the new proposed features, reproducing configurations shown by de Haro et al. [1]. *Placement* shows the improvement when using placement features. *Priorities* show the improvement obtained by enabling memory access priorities feature on top of placement. *Interleave* shows speedup when enabling interleaving on top of placement and priorities. Dots show energy efficiency for baseline and fully improved versions.

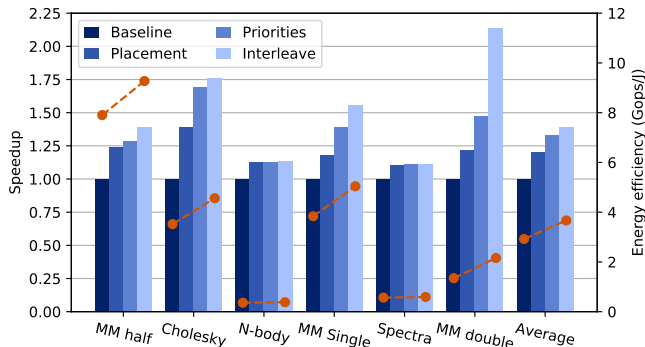


Fig. 1: Speedup (bars) using different features across different applications. Points and dashed lines show energy efficiency.

For matrix multiply and Cholesky, using placement allows designs with more accelerators and greater frequencies. In the case of n-body and spectra, we are able to increase frequency. Interleave and priorities features have a significant effect in cases where the computation-to-data movement ratio is low. This is especially relevant in the case of double-precision floating point matrix multiplication. However, these features have very limited effects on n-body and spectra due to their high computation density. In any case, the proposed features do not negatively impact performance. Energy efficiency is improved by 25% on average. This is mainly caused by a decrease in energy consumed by static power as design architecture has not changed significantly. By shortening run times, static power is consumed during less time.

Table I shows the different design parameters, performance and power for the baseline and fully improved version of the application benchmark (Baseline/Improved). Improved version corresponds to the *Interleave* results in figure 1. *Num acc* shows the number of accelerators in the design. *BS* shows the block size or the amount of data each accelerator processes. *Perf* shows application throughput and *Power* shows average power consumed by accelerators during application execution.

IV. CONCLUSIONS

In this paper, we have extended the OmpSs@FPGA framework to better exploit available resources in FPGA devices, focusing on acceleration of HPC applications. Compared to the already optimized version of FPGA accelerated benchmarks, proposed features help to improve their performance and reach a speedup of 1.37x on average across the analyzed benchmarks. In the case of linear algebra benchmarks, we are able to provide speedups of up to 1.62x in the case of the Cholesky decomposition and over 2x in double precision matrix multiplication. The performance reached using proposed features is competitive with state-of-the-art implementations [1], [3], [4], [5], [6]. Moreover, all of the proposed features are implemented into the compilation toolchain, therefore not requiring any change to the user application. Other high-level frameworks should be able to integrate the proposed features in their designs and provide similar gains.

Acknowledgments: This work is supported by the TEX-TAROSSA project G.A. n.956831, as part of the EuroHPC initiative, by the Spanish Government (Grants PCI2021-121964 - TEXTAROSSA, PID2019-107255GB-C21 MCIN/AEI/10.13039/501100011033, and CEX2021-001148-S), and by Generalitat de Catalunya (2021 SGR 01007)

REFERENCES

- [1] J. M. de Haro, J. Bosch, A. Filgueras, M. Vidal, D. Jiménez-González, C. Álvarez, X. Martorell, E. Ayguadé, and J. Labarta, “Ompss@fpga framework for high performance fpga computing,” *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2029–2042, 2021.
- [2] A. Filgueras, M. Vidal, D. Jiménez-González, C. Álvarez, and X. Martorell, “Improving performance of hpc kernels on fpgas using high-level resource management,” in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2023, pp. 213–213.
- [3] C. González, S. Balocco, J. Bosch, J. M. de Haro, M. Paolini, A. Filgueras, C. Álvarez, and R. Pons, “High performance computing pp-distance algorithms to generate x-ray spectra from 3d models,” *International Journal of Molecular Sciences*, vol. 23, no. 19, 2022. [Online]. Available: <https://www.mdpi.com/1422-0067/23/19/11408>
- [4] J. de Fine Licht, G. Kwasniewski, and T. Hoeffler, “Flexible communication avoiding matrix multiplication on fpga with high-level synthesis,” in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 244–254. [Online]. Available: <https://doi.org/10.1145/3373087.3375296>
- [5] J. Bosch, M. Vidal, A. Filgueras, C. Álvarez, D. Jiménez-González, X. Martorell, and E. Ayguadé, “Breaking master-slave model between host and fpgas,” in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPOPP ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 419–420. [Online]. Available: <https://doi.org/10.1145/3332466.3374545>
- [6] E. Del Sozzo, M. Rabozzi, L. Di Tucci, D. Sciuto, and M. D. Santambrogio, “A scalable fpga design for cloud n-body simulation,” in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–8.