POSTER: Breaking Master-Slave Model between Host and FPGAs

Jaume Bosch
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
jbosch@bsc.es

Miquel Vidal
Barcelona Supercomputing Center
miquel.vidal@bsc.es

Antonio Filgueras
Barcelona Supercomputing Center
antonio.filgueras@bsc.es

Carlos Álvarez
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
calvarez@ac.upc.edu

Daniel Jiménez-González
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
djimenez@ac.upc.edu

Xavier Martorell
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
xavim@ac.upc.edu

Eduard Ayguadé
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
eduard.ayguade@bsc.es

Abstract

This paper proposes to enhance current task-based programming models by breaking their current master-slave approach between the main processor and its hardware accelerators. As a proof-of-concept, it presents an extension of the OmpSs@FPGA toolchain that allows the tasks offloaded into the FPGA to create and synchronize nested tasks on their own without involving the host. Those FPGA spawned tasks may target the host to execute code not suitable for the FPGA, like system calls or I/O operations; or target other kernel accelerators inside the same FPGA. In addition to the programmability benefits of this new feature, the proposed system presents significant performance improvements and a better productivity over the classical master-slave approach.

CCS Concepts • **Computing methodologies → Parallel programming languages**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

Keywords Task-Based Parallelism, Heterogeneous computing, FPGA, Runtime, OmpSs, OpenMP.

1 Introduction

The OmpSs@FPGA environment extends the OmpSs programming model (a forerunner of OpenMP) to support task offloading to FPGA devices [1]. The OmpSs and OpenMP programming models have a master-slave model where the accelerators are driven by a general-purpose processor acting as a master [2]. This simplifies the management of the accelerators because they cannot actively interact with the runtime and they are just slaves that compute and/or process data when requested by the master. However, the master-slave model limits programmability and potential performance, especially for FPGA based systems.

The fact of breaking the master-slave model implies the possibility of avoiding numerous synchronization overheads, which will lead to significant performance gains. As the results show, the host threads are not always able to adequately feed the task-accelerators in the FPGA device when dealing with fine grained tasks. By creating and executing the tasks directly in the FPGA, the application avoids the host-FPGA latency allowing a faster task creation (10x), more free CPU time to do useful work, and better resource productivity. Moreover, there are pieces of code that cannot be executed in the FPGA (like Operating System code or complex application parts that are executed scarcely and consume several FPGA resources). With the proposed system, the task-accelerator can autonomously spawn a task for the host that executes an OS call, I/O operation, or a complex application part.

2 Design

The main goal is to allow interaction of the FPGA device with the parallel programming runtime to make the FPGA cooperate in the application execution beyond the current offload model. To do so, new queues and IPs to manage the requests from task-accelerators are placed inside the Task Manager, as shown in figure 1. The proposed design includes three new memory modules: Internal Ready Queue (for FPGA tasks ready to execute), Remote New Queue (for tasks reverse

PPoPP '20, February 22–26, 2020, San Diego, CA, USA

ACM ISBN 978-1-4503-6818-6/20/02.
https://doi.org/10.1145/3332466.3374545

offloaded to the host) and Remote Finished Queue (for finalization notification of reverse offloaded tasks); four new IPs: Scheduler Task Manager, New Task Manager, Remote Finished Task Manager and Taskwait Task Manager (it tracks the pending tasks in each context and notifies the waiters). This design is built by the OmpSs@FPGA tools during the bitstream generation and after transforming the task annotated function calls into requests to the Task Manager.

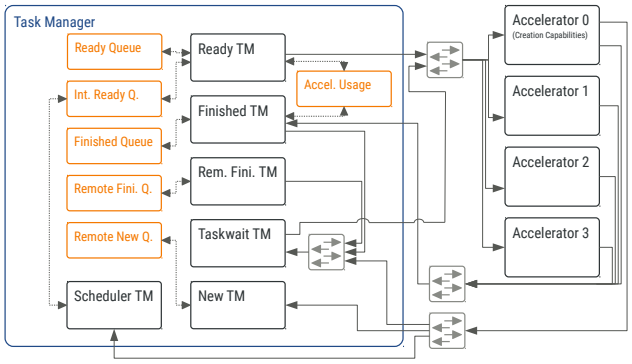


Figure 1. Bitstream organization with the proposed design

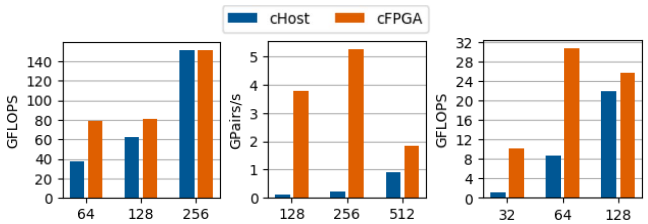
3 Evaluation

The evaluated applications are Matrix Multiply, N-Body, and Cholesky (with and without the added features, and with three block sizes). For Matrix Multiply, all designs run at 300Mhz and contain 7, 3, or 3 task-accelerators (implemented in C) for 64x64, 128x128, and 256x256 block size respectively. For N-Body, all designs run at 250MHz and contain different amounts of task-accelerators that maximize the performance for each block size interacting between them (nested tasks). For Cholesky, all designs run at 250MHz and contain task-accelerators for `trsm`, `gemm` and `syrk`, but not for `potrf` which is always executed in the host threads (showing the potential of reverse offload). The tools used to generate the application bitstreams and binaries are: Vivado Design Suite 2017.3, GNU C/C++ Compiler 6.2.0, PetaLinux Tools 2019.1 and OmpSs@FPGA tools of release 1.3.2. The applications are run in a Xilinx Zynq UltraScale+ MPSoC ZCU102 (4 ARM Cortex-A53 cores, at 1.1 GHz, ZU9EG Xilinx FPGA, 4 GB DDR4 memory).

Table 1. Resources utilization and power estimation in ZCU102

Name	BRAM	DSP	FF	LUT	Power
ZCU102	912	2520	548.160	274.080	-
ARM A53 Core	-	-	-	-	800mW
TM (baseline)	10 (1%)	0 (0%)	1.350 (0,2%)	3.929 (1,4%)	40mW
Ext. TM	25 (2,7%)	0 (0%)	2.967 (0,5%)	8.796 (3,2%)	200mW
Task Acc. Creat.	3 (0,3%)	0 (0%)	2.225 (0,4%)	4.008 (1,4%)	30mW

Table 1 shows a summary of available resources in the FPGA chip of ZCU102 SoC, the power consumption of each A53 core reported by Vivado, and the resources utilization and power consumption of the implemented IPs as reported by Vivado. The resource utilization for the Task Manager with the proposed extensions (*Ext. TM*) shows that the new IP still uses a small portion of total resources. That leaves the bulk of FPGA resources available to implement application kernels.



(a) Matrix Multiply **(b)** N-Body **(c)** Cholesky

Figure 2. Applications performance for different block sizes

Figure 2 shows the performance of applications without and with the added features, *cHost* (creation host) and *cFPGA* (creation FPGA) respectively. Results show that *cFPGA* outperforms the *cHost* results, especially for smaller block sizes. This is due to a faster task creation and the reduction of creator-executor latencies. Finally, the results show that thanks to the added capabilities, the peak performance of the applications is not always achieved with the bigger block (accelerator) sizes. This effect is due to the extra parallelism obtained with smaller accelerators by the fast task management of our proposal.

Acknowledgments

This work has received funding from EPEEC project (European Union's Horizon 2020 Research and Innovation Programme, under grant agreement No 801051), from Spanish Government (projects SEV-2015-0493 and TIN2015-65316-P, grant BES-2016-078046), and from Generalitat de Catalunya (contracts 2017-SGR-1414 and 2017-SGR-1328).

References

[1] Jaume Bosch, Xubin Tan, Antonio Filgueras, Miquel Vidal, Marc Mateu, Daniel Jiménez-González, Carlos Álvarez, Xavier Martorell, Eduard Ayguadé, and Jesús Labarta. 2018. Application Acceleration on FPGAs with OmpSs@FPGA. In *2018 International Conference on Field-Programmable Technology (FPT)*. 70–77. https://doi.org/10.1109/FPT.2018.00021

[2] Lukas Sommer, Jens Korinth, and Andreas Koch. 2017. OpenMP device offloading to FPGA accelerators. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 201–205. https://doi.org/10.1109/ASAP.2017.7995280