Supporting task creation inside FPGA devices

Jaume Bosch*†, Carlos Álvarez*†, Daniel Jiménez-González*†
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
E-mail: jbosch@bsc.es, {calvarez, djimenez}@ac.upc.edu

Keywords—Heterogeneous computing, Device offloading, Task Based Parallel Programming Models, High-performance computing.

I. Introduction

The most common model to use co-processors/accelerators is the master-slave model where the slaves (co-processors/accelerators) are driven by a general purpose cpu. This simplifies the management of the accelerators because they cannot actively interact with the runtime and they are just passive slaves that operate over the memory under demand. However, the master-slave model limits system possibilities and introduces synchronization overheads that could be avoided.

To overcome those limitations and increase the possibilities of accelerators, we propose extending task based programming models (like OpenMP [1] or OmpSs) to support some runtime APIs inside the FPGA co-processor. As a proof-of-concept, we implemented our proposal over the OmpSs@FPGA environment [2] adding the needed infrastructure in the FPGA bitstream and modifying the existing tools to support creation of children tasks inside a task offloaded to an FPGA accelerator. In addition, we added support to synchronize the children tasks created by a FPGA task regardless they are executed in a SMP host thread or they also target another FPGA accelerator in the same co-processor.

II. Design

The main design goal is to allow interaction of the FPGA with the runtime to make both parts cooperate in the application execution beyond the current offload model. The new interaction capabilities include the creation of tasks inside a device task and child task synchronization.

Listing 1 shows an example code where just replacing the line 6 (with clause `device(smp)`) by line 7 (`device(fpga)`), we will have a FPGA accelerator that creates and then synchronizes tasks inside the FPGA. All interactions are based on queues (memory regions) where the FPGA accelerators write requests to the runtime. Then, the runtime reads these requests and makes the needed actions. Some interactions are optimized and directly read and handled inside the FPGA, avoiding the latency between the host and the device. This way there is runtime support in both parts of the machine and both are coordinated when needed to correctly execute the application.

Queues and new IPs to manage them are placed inside the Task Manager as shown in Fig. 1. The proposed design

```
1  #pragma omp target device(fpga) num_instances(3)
2  #pragma omp task copy_inout([BS]a)
3  void update_fpga(int *a, int val, size_t BS) {
4    for (size_t i=0; i<BS; ++i) a[i] += val;
5  }
6  #pragma omp target device(smp)     //< Task exec. in SMP
7  //#pragma omp target device(fpga)  //< Task exec. in FPGA dev.
8  #pragma omp task copy_inout([LEN]a)
9  void update_blocked(int *a, int val, size_t LEN, size_t BS) {
10   for (size_t i=0; i<LEN; i+=BS)
11     update_fpga(a+i, val, BS);
12   #pragma omp taskwait
13 }
14 int main(...) {
15   int *a = (int *)malloc(NUM_ELEMENTS*sizeof(int));
16   update_blocked(a, 2019, NUM_ELEMENTS, NUM_ELEMENTS_BLOCK);
17   #pragma omp taskwait
18 }
```

Listing 1: OmpSs example of FPGA nested tasks

includes three new memory modules: Internal Ready Queue (Int. Ready Q), Remote New Queue (Remote New Q) and Remote Finished Queue (Remote Fini. Q); four new IPs: Remote Finished Task Manager (Rem. Fini. TM), New Task Manager (New TM), Taskwait Task Manager (Taskwait TM) and Scheduler Task Manager (Scheduler TM); and an additional interconnection network between the accelerators with creation capabilities and the Task Manager. The new Task Manager elements are automatically added to the design by autoVivado when needed by any of the accelerators. Moreover, Mercurium adapts the wrapper around the accelerator to support the runtime API calls from user code. Those calls may be explicitly invoked by the user or automatically inserted by the compiler during the translation phase where the OpenMP directives are translated into API calls.

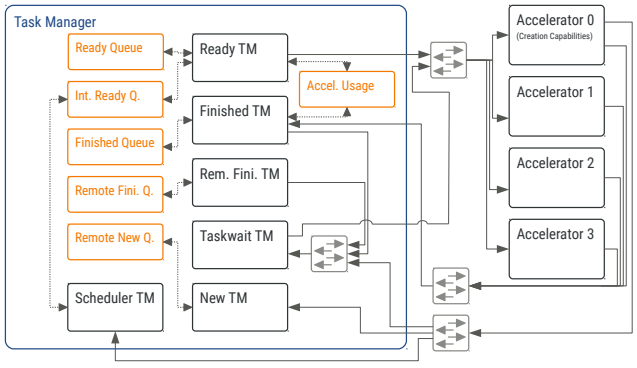


Fig. 1. OmpSs@FPGA bitstream organization with the proposed design

III. Evaluation

To analyze the performance, we used the synthetic benchmark in listing 1 that updates all elements of an array. Therefore, we parameterize the benchmark with the length of the array and the chunk length. The tools used to generate the application bitstream and binary are: Vivado Design Suite 2016.3, GNU C/C++ Compiler 6.2.0, PetaLinux Tools 2016.3 and all modified OmpSs@FPGA tools of release 1.2.1. The applications are run in a Zynq Ultrascale+ MPSoC Chip XCZU9EG-FFVC900 [3].

Fig. 2 shows the average execution time (y-axis) when decreasing the task size (chunk length) and increasing the number of tasks (x-axis). Note that the total amount of work remains constant for all the executions, as the array size equals the chunk length times the number of tasks. The same result is shown for different configurations, all of them executed within the OmpSs@FPGA environment. The label of each configuration defines the location where the tasks are created and the number of accelerators used to execute them.

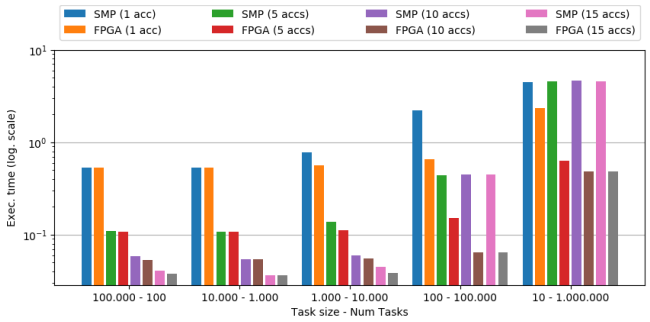
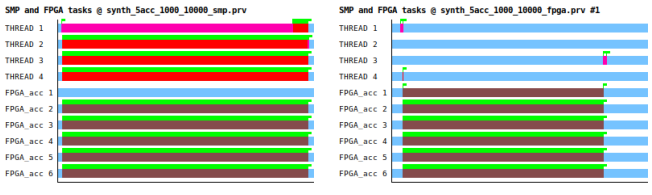


Fig. 2. Synthetic benchmark execution time with different configurations

The results show that the creation and management of FPGA tasks directly from the FPGA is faster than doing it from the host side. For any pair of benchmark arguments, the reduction of the execution time increases with the number of accelerators. This is because of the lower task creation overheads in our proposed implementation in comparison to the already implemented management in the host runtime. Consequently, the FPGA creation is able to discover more parallelism, increasing the accelerators utilization and reducing to overall execution time. In contrast, the host management and the FPGA management take a similar execution time when the task size is large enough to hide the runtime overheads with the tasks execution.



(a) smp (b) fpga

Fig. 3. Execution traces of synthetic benchmark with 5 accelerators, 1.000 task size and 10.000 tasks

The execution traces in Fig. 3 show that the FPGA tasks execution take less time when they are directly created on the FPGA than when they are created and sent by the SMP threads. In addition, the `FPGA_acc 1` accelerator does not execute any task when the FPGA tasks are created by the SMP threads and it runs the `update_blocked` task until all children tasks finish. The FPGA accelerators does not have the possibility to block a task and pick another one for execution, in contrast to SMP threads.

As Fig. 3b shows, the SMP threads are under-utilized when the FPGA creates the tasks directly. In contrast, they are needed in the SMP creation to create the tasks, offload them to the FPGA and retrieve the finalization messages that allow the host synchronize the remote tasks. With the proposed extension, we are able to replace the general purpose cores by an small FPGA accelerator that consumes a small portion of power when compared but does the same work.

IV. Conclusion

This paper presents an extension of the OmpSs@FPGA ecosystem to support task creation and synchronization operations in the FPGA co-processors. This extension enables a new dimension of possibilities for application programmers as they can mix tasks for different devices and nest them without restrictions. Finally, the initial performance results show that the creation of FPGA tasks inside the same FPGA device allows the usage of very fine-grain tasks thanks to the significant overheads reduction.

V. Acknowledgments

This work is under review for proceedings of the International Symposium on Memory Systems (EUROPAR), 2019. This work is partially supported by the European Union H2020 Research and Innovation Action through the EuroEXA project (GA 754337) and HiPEAC (GA 687698), by the Spanish Government (projects SEV-2015-0493 and TIN2015-65316-P, grant BES-2016-078046), and by the Generalitat de Catalunya (contracts 2017-SGR-1414 and 2017-SGR-1328).

References

[1] L. Dagum and R. Menon, "OpenMP: an Industry Standard API for Shared-Memory Programming," *Computational Science Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, Jan 1998.

[2] J. Bosch and et al., "Application Acceleration on FPGAs with OmpSs@FPGAS," *2018 International Conference on Field-Programmable Technology (FPT)*, 2018.

[3] Xilinx, Inc. (2019, April) ZYNQ UltraScale+ MPSoC Overview. [Online]. Available: www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

Jaume Bosch received the B.S. and M.S. degrees in Computer Science from the Technical University of Catalunya (UPC) in 2015 and 2017, respectively. Currently, he is is a PhD student in the department of Computer Architecture of UPC and in the Programming Models Group of Barcelona Supercomputing Center (BSC). His research interest lies in parallel, distributed and heterogeneous runtime systems for High Performance Computing, specially systems with hardware accelerators like FPGAs or many-core architectures.