Facilitating Easier Access to FPGAs in the Heterogeneous Cloud Ecosystems

Umar Ibrahim Minhas, Roger Woods and Georgios Karakonstantis

***Abstract*—With FPGAs being increasingly integrated into existing software-based heterogeneous cloud environments, novel evaluation mechanisms are required to reveal the energy-performance trade-offs of accelerators (FPGAs, GPUs, etc) in high-level heterogeneous programming environments. For FPGAs, this involves also a reconsideration of scheduling policies and reconfiguration methods with an aim of integrating software-based approaches as well as performance optimizations for wider workload sizes. The approaches are evaluated using various reconfiguration methodologies for a number of applications.**

I. Introduction

Cloud computing offers users ubiquitous access to a shared pool of resources, through centralized data centres managed by high level software based ecosystems. In recent times, there has been an increased interest in integrating Field Programmable Gate Arrays (FPGAs) in these ecosystesm [1] but these type of solutions differ in design effort, reconfiguration and scheduling policies when compared to those based on Graphic Processing Units (GPUs). This paper re-evaluates accelerators in the context of a heterogeneous computing environment within cloud ecosystems.

The first challenge is concerned with the evaluation of achievable gains using a state-of-the-art heterogeneous programming model - OpenCL. Although the traditional approach of programming accelerators using platform specific languages (e.g VHDL, CUDA, etc) [2] is performance-efficient, it suffers in heterogeneous environments due to lack of portability.

The second challenge investigates the reconfiguration overhead on FPGAs [3] incurred due to multi-task execution in cloud systems. Initial work has targeted such a challenge via intelligent scheduling targeting techniques such as module reuse [4] and faster reconfiguration memories [5]. However, to our knowledge, no prior work has explored the use of more reconfiguration methods in the scheduling model, particularly the more recent software based approaches. If achievable, this would allow easier access of the FPGA as a heterogeneous resource and permit a trade-off between reconfiguration overhead and performance, helping to improve the execution time of any workload.

To summarize, our contributions are:

- Evaluation of accelerators (FPGA and GPU) using a modern uniform programming model (OpenCL) and similar optimization efforts.
- Inclusion of multiple reconfiguration approaches into the scheduling model for task size aware selection of the reconfiguration method.

II. Methodology

To address the aforementioned challenges, we apply a systematic evaluation and scheduling methodology.

Evaluation of Accelerators: We evaluate FPGAs against GPUs using the unified parallel programming model, OpenCL, as follows:

- *Step-1:* Identify a set of micro-architectural optimizations and design space exploration based on platform-independent parameters of OpenCL that are applicable to both FPGAs and GPUs.
- *Step-2:* In addition to *Step-1*, apply application-specific optimizations on three accelerated computing tasks including matrix-matrix multiply (SGEMM), binomial option pricing (BOP) and finite difference time domain (FDTD), while analyzing the architectural and algorithmic challenges using OpenCL.

We then compare the implementations on state-of-art platforms which use the same technology, namely an Altera FPGA (Nallatech 385 with Stratix V A7 chip) and a NVIDIA GPU (GTX-980). We consider the use of application specific metrics for throughput and energy efficiency while keeping the design effort constant. We also compare the achieved throughput against theoretical peak for each platform.

Although the GPU outperforms FPGA in terms of throughput (due to larger size of device), the results indicate that the Altera FPGA performs better in terms of achieved percentage of device's peak performance (68%) compared to the NVIDIA GPU (20%); also, it achieves better energy efficiency (up to $1.4\times$) for some examples without the need for detailed hardware optimisation.

Scheduling Model: The most common reconfiguration methods for FPGAs involve: single task reconfiguration on the whole FPGA with a reconfiguration time per task, T_r in the order of few seconds; and dynamic partial reconfiguration (DPR), in which multiple tasks are loaded onto defined regions, where each region is a subset of whole FPGA, with T_r in the order of few 100ms. We explore two additional reconfiguration methods:

- *Multi-core reconfiguration (MCR)* comprising 64 lightweight programmable cores which allow for rapid runtime reprogramming in less than 20ms [6].
- *Single bitstream multi-task reconfiguration (SBMT)*, which is similar to DPR. However, instead of dividing FPGA into fixed size regions, the OpenCL synthesizer maps combined source code of all tasks to a single bitstream, allowing for system optimization. The intelligent

resources allocation per task is proportional to workload size (W_S), for all tasks to have a similar execution time.

The order of the the above mentioned methods for highest to lowest T_r and throughput, T, is SBST, SBMT/DPR and MCR. DPR in our case has the same performance as SBMT. However, SBMT may have lower T_r than DPR. This is because for SBMT, bistream size corresponds only to logic used per task while for DPR, it is proportional to size of region onto which the task is loaded which will always be greater than the minimum area required by the task.

The net execution time in a multi-task environment depends on the *reconfiguration time - throughput - task size* trade-off. Using the aforementioned information, our task size-aware scheduler tries to minimize the following sum for *n* tasks:

$$\sum_{i=0}^{n} T_r(i) + T(i) \times W_S(i) \tag{1}$$

III. Results

We consider two real world tasks from financial computation and graph analytics, namely BOP and sparse matrix vector multiplication (SpMV). The W_S for both tasks can vary widely in real-time environment with the number of options for BOP, depending on the stocks, and the number of non-zero elements (NNZ) in SpMV, depending on the network being evaluated.

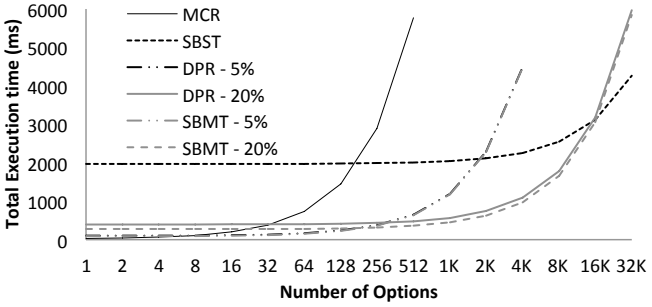

Fig. 1. BOP execution using various reconfiguration methodologies

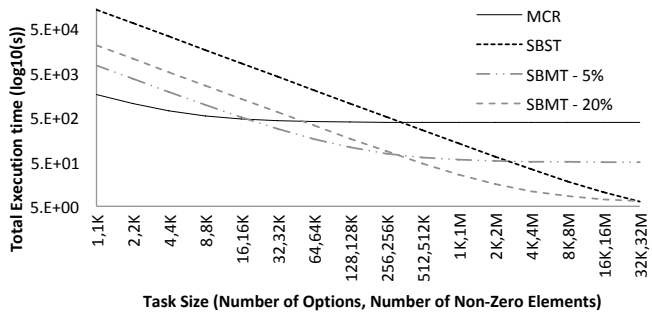

Fig. 2. Multi-task computations using various reconfiguration methodologies

Fig. 1 shows the relationship between total execution time and W_S using various reconfiguration methods for BOP, while SpMV followed a similar trend. For DPR and SBMT, we explore two different configurations that use logic less than or equal to 5% and 20% of total FPGA logic, respectively. The results show that the selection of a reconfiguration method based on W_S can provide up to $86\times$ and $106\times$ improvement over worst case for tasks with a similar execution model as BOP or SpMV, respectively.

Next, we mimic multi-task execution by using multiple instances of BOP and SpMV as independent tasks (requiring reconfiguration each time). To evaluate the effect of individual tasks' W_S, we take a total W_S of 32K options and 32M NNZ for BOP and SpMV, respectively and vary the W_S per instance. The evaluation of reconfiguration methodology against W_S in Fig. 2 shows that even when computing tasks with variable execution models, the proposed approach can provide up to $79\times$ and $60\times$ improvement over worst case on the lowest and highest end of W_S/instance, respectively.

The results suggest a case for selection of reconfiguration method based on the dynamic nature of task queues. For a more dynamic queue requiring frequent reconfiguration for tasks with smaller workload sizes, a space sharing model or run-time programmable model is better. However for bigger tasks, whole FPGA reconfiguration may be used.

IV. Future Work

Our initial evaluation of multiple reconfiguration methods shows that SBMT performs the best on average because,

- SBMT has lower average T_r than DPR as explained in Section II.
- SBMT has the best compute density, i.e. the least total execution time for a set of tasks excluding T_r, as depicted by another set of independent experiments using four computationally different tasks.

This motivates us to further explore SBMT. In our context of SBMT using OpenCL, we explore the design space by changing OpenCL parameters such as work-items, loop unrolling, local memory, etc. To speed up this exploration, we propose a static source code analysis framework that can leverage the correlation between multiple versions of same task to achieve accuracy in performance estimates.

Acknowledgment

The work was supported by the European Union under grant number 6876281 (VINEYARD).

References

[1] N. Tarafdar et al. "Designing for FPGAs in the Cloud", in *IEEE Design & Test*, pp. 23-29, 2018.

[2] U. I. Minhas et al., "GPU vs FPGA: A comparative analysis for non-standard precision", in *Int. Symp. on Appl. Reconfigurable Comput.*. Springer, pp. 298-305, 2014.

[3] A. Purgato et al., "Resource-efficient scheduling for partially-reconfigurable FPGA-based systems", in *IEEE Int. Parallel and Distrib. Process. Symp. Workshops*, pp. 189-197, 2016.

[4] R. Cattaneo et al., "Para-sched: A reconfiguration-aware scheduler for reconfigurable architectures", in *IEEE Int. Parallel & Distrib. Process. Symp. Workshops*, pp. 243-250, 2014.

[5] H. Liang et al., "Parallelizing hardware tasks on multicontext FPGA with efficient placement and scheduling algorithms", *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*, pp. 350-363, 2018.

[6] U. I. Minhas et al, "Nanostreams: A microserver architecture for real-time analytics on fast data streams", *IEEE Trans. Multi-Scale Comput. Syst.* , 2017