Configuration splitting to extend the processing capabilities of embedded reconfigurable systems

Christopher Cichiwskyj
University of Duisburg-Essen
Duisburg, Germany
christopher.cichiwskyj@uni-due.de

ABSTRACT

The Elastic node is an embedded hardware platform using an 8-bit MCU and an low-power FPGA, allowing to create IoT applications with more local processing power. Due to the limited amount of resources on low-power FPGAs, very large hardware designs cannot be instantiate, thus limiting the range of algorithms for this platform. To circumvent this issue large designs can be split, using the FPGAs reconfigurablity to switch between the smaller parts. This paper outlines the open question regarding such split logic and provides possible solutions.

1 INTRODUCTION

The Internet of Things is a rapidly growing network of everyday objects that are extended with embedded computing devices. With their integrated sensors and actuators they can interact with the physical world and through the internet communicate with other devices and services. Due to their embedded nature these devices typically have to be battery powered and energy efficient to last months or even years. The trend in IoT applications however is that more complex algorithms are required. Machine learning, computer vision, or self-organisation are only some of the concepts, that embedded devices are often not capable of performing locally with their limited resources. A solution to this is offloading these complexe tasks e.g. to the cloud. Depending on the type of calculation and application scenario this is not always viable, due to latency, security or privacy concerns.

2 THE ELASTIC NODE

To be able to increase the local performance without requiring larger, more energy intense processors we propose the Elastic Node. The Elastic node [1][2] is an IoT hardware platform, designed to dynamically adapt its hardware capabilities according to the required workload, thus allowing more tasks to be calculated locally. The idea is that it only "grows" as much as is required to be able to solve a given task, and returns back, or "shrinks", to its low performance state once the complex task is finished.

It is implemented with a low-power 8-bit microcontroller (MCU) and a embedded field-programmable gate array (FPGA), a piece of reconfigurable hardware capable of instantiating arbitrary digital circuits. An application consists of multiple tasks. Simple tasks are executed efficiently on the MCU, complex tasks that require more processing power are offloaded to the FPGA as so called Hardware Functions (HWF). To do so the FPGA is powered up, reconfigured to instantiate the appropriate circuit, performs the necessary calculations, and once finished returns to a sleep state. Preliminary results show that the Elastic Node can perform these tasks more energy efficiently, processing them locally, than when offloading the same tasks through a wireless channel [2].

3 PROBLEM STATEMENT

Unlike the Harvard or Von-Neumann computing architectures an FPGA does not use a classical computer program. Instead an FPGA developer designs the required circuitry using Hardware Description Languages and synthesises them into configuration files. A configuration file describes to the FPGA how its hardware resources on the chip should be interconnected to form the desired circuitry.

Larger FPGAs typically have the capabilities to instantiate all logic corresponding to a single HWF in one configuration file. Offering this functionality, however, usually requires operating system (OS) abstraction [3][4][5]. Embedded FPGAs do not have enough resources to instantiate such large logic and embedded MCUs often cannot provide full OS abstractions. This limits what HWFs can be implemented on the Elastic Node. A solution to this is to split the larger HWF into smaller subtasks, each small enough to fit into an embedded FPGA configuration. The HWF is then calculated by reconfiguring the FPGA to each subtask and pass the intermediate data between them.

The focus of this research is to determine, if it is possible to execute such large HWFs on an embedded FPGA while remaining more or as efficient as offloading these tasks e.g. to the cloud. Certain key challenges need to be addressed.

Coordinating split HWFs: When certain HWFs are not self-contained logical units, but instead have predefined internal subtask data dependencies, their execution order is not purely defined by the application anymore. Instead certain FPGA configurations need to be loaded and executed in a certain order to ensure the correct overall HWF result.

Reducing reconfiuguration overhead: Splitting HWFs into multiple configurations means that more reconfigurations have to be performed to achieve the same result. This means added overhead to the overall execution time of a single HWF. The question is whether this overhead can be reduced.

Intermediate data passing management: As FPGAs are not capable of keeping any data internally throughout reconfigurations, data that has to be passed between subtasks have to buffered and

Middleware Doctoral Symposium '18, December 10-14, 2018, Rennes, France
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

$$\begin{tikzcd} A_1 \arrow[r] & B_1 \arrow[r] & C_1 \arrow[r] & D_1 \arrow[r] & C_2 \arrow[r] & E_2 \arrow[r] & B_2 \arrow[r] & D_2 \end{tikzcd}$$

$\rightarrow$: FPGA Reconfiguration

(a) Sequential execution of two HWFs leads to a large number of reconfigurations

$$\begin{tikzcd} A_1 \arrow[r] & B_1 \arrow[r] & C_1C_2 \arrow[r] & E_2 \arrow[r] & B_2 \arrow[r] & D_1D_2 \end{tikzcd}$$

$\rightarrow$: FPGA Reconfiguration

(b) Grouping subtasks using the same configuration reduces the number of unnecessary reconfigurations

Figure 1: (a) A naive scheduling vs. (b) grouped HWF scheduling

managed externally. Additionally to the added memory overhead, managing this data adds overhead to both memory and execution time as well. Due to scarce resources, it is crucial that this management can be done efficiently.

4 APPROACH

Scheduling split HWFs is dependent on the data dependencies between the tasks, which can be modelled using task graphs. Scheduling task graphs in the general case is an NP-complete problem [6]. However as placement of tasks onto processing units is limited to the Elastic Node's single FPGA, scheduling of the subtasks is reduced to finding the correct linearisation of a given task graph. Due to the task graphs being static, the scheduling decisions can be done at design time.

Calling multiple HWFs requires a scheduling. A naive approach is to queue them sequentially, as seen in fig. 1a. This schedule requires seven reconfigurations. Each reconfiguration, however, means significant overhead, both in time and energy consumption. While the actual execution time of HWFs take nanoseconds [2], the reconfiguration requires multiple milliseconds [7]. Assuming that FPGA subtask configurations are reused across different HWFs, or when the same HWF is called repeatedly, reconfigurations can be reduced. Grouping subtasks, that use identical configurations, allows reducing reconfigurations. As seen in fig. 1b grouping the same two HWF sequences reduced the number of reconfigurations by two. A grouping approach however needs to ensure that the execution order dictated by the task graph is not violated.

Similar approaches using task graphs have been proposed [8][9], however these focus on splitting the available FPGA area into multiple processing units to distribute the tasks instead of reconfiguring the complete FPGA.

A promising approach is to map the grouping problem onto DNA sequence alignment algorithms such as [10]. Each HWF sequence is mapped to a string, where each character represents an FPGA configuration. The goal is to minimise the transitions between characters when aligning both sequences.

As mentioned in section 3 the FPGA cannot store intermediate data across reconfigurations, thus requiring the MCU to perform the data management of such intermediate data. As memory on the MCU is scarce any intermediate data has to be freed as soon as it is obsolete. Doing so in a static fashion however is not possible, as the time when to free data does not only depend on the data dependency described in the task graph but also the runtime schedule of grouping HWFs.

5 EVALUATION

Due to the remaining open questions regarding possible implementation details the evaluation is not completely defined yet. The main criteria is whether splitting HWFs is viable with regards to latency and energy consumption. To this regard it will be compared to alternative approaches, such as offloading it wirelessly as well as performing all calculation solely on MCUs with different processing capabilities. The reconfiguration reduction heavily depends on the similarity of the HWF sequences that are aligned. Varying the HWFs, the similarity of sequences as well as their number will give an insight into what types of applications benefit from this approach. When splitting HWFs into smaller subtasks the reconfiguration overhead is a significant part of the overall performance. This will answer at what point it becomes more viable to use a larger FPGA, regarding both latency and energy consumption, instead of continously reconfiguring on a smaller FPGA.

6 CONCLUSION

Extending embedded devices with reconfigurable hardware has the potential to lessen the dependence on offloading complex calculations. By answering the above mentioned questions the insights can be used to create a middleware abstraction to provides developers a simplified way to implement and use HWFs, as well as provide a better understanding as how to design application on reconfigurable systems and what application scenarios can benefit from this type of hardware.

REFERENCES

[1] A. Burger, C. Cichiwskyj, and G. Schiele. Elastic nodes for the internet of things: A middleware-based approach. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 73–74, July 2017.

[2] A. Burger and G. Schiele. Demo abstract: Deep learning on an elastic node for the internet of things. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Mar 2018.

[3] M. D. Santambrogio, V. Rana, I. Beretta, and D. Sciuto. Operating system runtime management of partially dynamically reconfigurable embedded systems. In *2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia*, pages 1–10, Oct 2010.

[4] A. Agne, M. Happe, A. Keller, E. Lübbers, B. Plattner, M. Platzner, and C. Plessl. Reconos: An operating system approach for reconfigurable computing. *IEEE Micro*, 34(1):60–71, Jan 2014.

[5] X. S. Le, J. L. Lann, L. Lagadec, L. Fabresse, N. Bouraqadi, and J. Laval. Cardin: An agile environment for edge computing on reconfigurable sensor networks. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 168–173, Dec 2016.

[6] Michael R Garey and David S Johnson. Computers and intractability: A guide to the theory of np-completeness. *Computers and Intractability*, 340, 1979.

[7] Khurram Shahzad and Bengt Oelmann. Investigating energy consumption of an sram-based fpga for duty-cycle applications. In *International Conference on Parallel Computing-ParCo 2013, 10-13 Sept, Munich*, pages 548–559, 2014.

[8] Javier Resano, Daniel Mozos, and Francky Catthoor. A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '05, pages 106–111, 2005.

[9] Juan Antonio Clemente, Javier Resano, Carlos González, and Daniel Mozos. A hardware implementation of a run-time scheduler for reconfigurable systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(7):1263–1276, 2011.

[10] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.