A Position on Transparent Reconfigurable Systems

Luís M. Sousa
lm.sousa@fe.up.pt
Faculty of Engineering, University of Porto
Porto, Portugal

Nuno Paulino
nmcp@fe.up.pt
INESC-TEC and Faculty of Engineering, University of Porto
Porto, Portugal

João Canas Ferreira
jcf@fe.up.pt
INESC-TEC and Faculty of Engineering, University of Porto
Porto, Portugal

ABSTRACT

With the ever more pressing issue arising from the phenomenon known as the *death* or *slowdown* of Moore's Law and the Dennard Scaling, compute performance has not been increasing at the rate the industry had been accustomed to over the decades [7]. This has prompted a shift from mostly homogeneous compute architectures to increasingly heterogeneous ones [1]. As these systems become increasingly complex, manual tuning and management of these heterogeneous resources becomes unfeasible. In this paper, we propose a runtime mechanism for automatic reconfigurable resource management that will enable a hypothetical flow for combined hardware and software compilation.

1 INTRODUCTION

An application's performance can be maximised if it is executed on specialised hardware, both from a time and power consumption perspective. As customers demand higher performance and functionality, complexity and development time tend to increase. Furthermore, since silicon area is expensive [2], the use of Hardware Accelerators (HwAs) is only justified under certain conditions. Firstly, that they are used frequently to justify the design time and area expended. Secondly, that their workload is both well defined and amenable for parallelization, so that performance benefits can be maximised.

A task is only worth using an accelerator if the time it takes to be completed on such an accelerator is lower than the time it would take on general-purpose hardware. The time to transfer the data to the accelerator must be taken into account. Consider an embedded application with N functions for which accelerator circuits have been created. At run-time, the functions are called with arbitrary arguments, potentially in an unknown order. If the device harbouring the HwAs is incapable of implementing all N accelerators concurrently, some sort of management must be made in order to make available the most valuable subset of HwAs at any given time.

Traditionally, the developer of a heterogeneous system must understand the algorithmic component of the application, design the appropriate hardware to accelerate candidate regions (e.g. bottlenecks or good parallelization opportunities), and then must schedule workloads onto the different components, and synchronise their behaviour [12, 15].

LATTE '21, April 15, 2021, Virtual, Earth

The hardware design aspect has been addressed, in part by new tools such as High-Level Synthesis (HLS) which can be used to automatically generate circuit descriptions from functions written in C/C++ code [8, 14]. This enables simpler and lower cost development workflows. For certain execution models such as Field-Programmable-Gate-Array (FPGA) based server boards, HLS also provides Application Programming Interface (API) level integration of the resulting components.

Although FPGA devices lack the advantages of full-custom, i.e. Application Specific Integrated Circuit (ASIC) implementations [9], they introduce new paradigms to these heterogeneous systems by allowing the application to define custom circuitry to be implemented.

This capability for reconfiguration allows for a single underlying chip to be used to implement the HwAs required by a specific application. Additionally, algorithms are subject to change due to performance reasons, different application needs, or bug fixes. As FPGAs are reconfigurable, the cost-cutting is multiplied as no new devices need to be fabricated and deployed.

Another advantage of FPGAs for application-specific HwA design is the unique ability of *hot-swapping* [6] accelerator circuits at run-time, which is referred to as Dynamic Partial Reconfiguration (DPR) [13]. This feature, which has not seen widespread adoption in real-world applications, allows for a targeted FPGA area to be reconfigured while the device is in operation, allowing for the same resources to implement multiple functions in a time-multiplexed manner. (Figure 1) Allowing an application to hot-swap accelerators instead of solely dealing with a fixed, predefined set, implemented at boot-time, greatly expands on the possibilities of using FPGA devices as platforms for implementing accelerators. Those that are not used concurrently can be swapped out as needed freeing up space for others, reducing the total area of re-configurable fabric required to implement the complete set of HwAs of a specific application. By allowing time-multiplexed use of limited silicon resources, smaller and less expensive devices can be utilised.

Given this context, developers that wish to exploit heterogeneity in the context of FPGAs for the embedded domain, while also relying on their DPR capability for better silicon area usage, must therefore manually perform hardware/software partitioning, hardware design and testing, and runtime management of reconfigurable slots, both spatially and temporally.

2 PROPOSED SOLUTION

Our work focuses on the implementation of a runtime resource management mechanism based on decision trees. We envision this as part of compiler-based hardware/software partitioning approaches

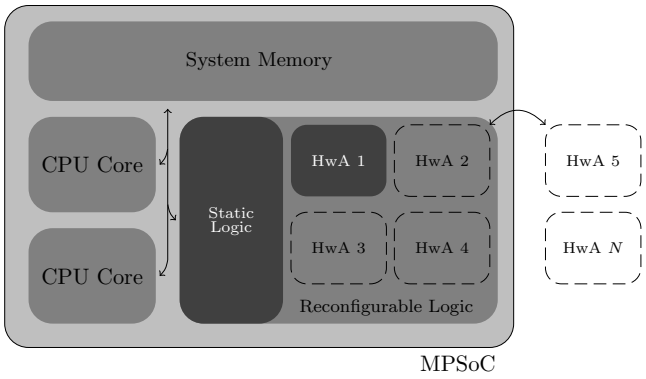

Figure 1: Proposed Architecture

previously proposed in the state-of-the-art, which can be summarised as 1) identifying segmented regions of code amenable for acceleration, 2) generating circuits for those regions without developer intervention, and 3) inserting invocations to the custom hardware. We aim to augment the accelerator invocation runtime management capabilities. Namely, 1) determining if a given accelerated function can indeed benefit from hardware invocation, versus its software counterpart, based on runtime function input parameters, and 2) falling back to software execution based on available accelerator slots.

This would determine if it is advantageous to implement an accelerator for a specific task execution and offload it to a DPR-based slot or execute that task on the general-purpose hardware.

Previous work has been done with this concept. However, the approaches used require time-consuming offline profiling and training of the used models [3, 4, 11] not consistent with compiler flows. These can provide very good offloading decisions on a system ensuring great performance but at the sacrifice of portability. The models developed for one particular system are not applicable if any of the components, or their operating frequency are changed, even when maintaining the same Instruction Set Architecture (ISA). Any change in system configuration will require profiling of the new system and retraining.

Our proposal, intended to automate the use of HLS and DPR transparently, requires no offline profiling and therefore no expended time dealing with changes to the code base. A custom decision engine can be trained on a per-device level that provides decisions tailored to the application running on such a device by employing online Decision Tree (DT) learning [5] fed by data such as the values of the arguments of the functions that are being called, the size of the data they will operate on, an estimate of the arithmetic intensity obtained through static analysis, the order in which the functions are called, whether or not the corresponding HwA is already loaded into the FPGA and the temporal overhead of loading the HwA via DPR (Algorithm 1).

The process starts by using the DT to decide if either software or hardware execution should be used to target each individual function call. If the DT's confidence level in its decision is above a user-defined threshold, the chosen component will execute the function and return the result. On the other hand, if the confidence level in the decision is low, the function is executed concurrently on both Central Processing Unit (CPU) and HwA. Execution is halted once either concurrent execution (i.e., software and hardware) terminates. The result is used to train the DT, increasing its confidence level in that region of the problem space.

Algorithm 1: Runtime Decision Engine

```
Initialisation;
Use the parameters of the function call in DT.
Let σ be the confidence in the choice made by the DT.
if σ > confidence threshold then
    Execute function on the preferred platform.
else
    Start execution on CPU.
    Start timer.
    Use DPR to load the HwA (if needed).
    Copy input data to HwA.
    Start execution on HwA.
end
if CPU or HwA is finished then
    Halt execution of slower platform.
    Copy result from HwA (if available).
    Stop timer.
    Train DT using the winner.
else
end
```

The Hoeffding Tree algorithm [5] is the method we propose to use in this Engine. It guarantees an asymptotically identical result when compared to batch learners, assuming that the data distribution does not change over time. Furthermore, several improvements to the original algorithm have also been proposed to make it more efficient when executed on an FPGA architecture [10].

By compiling the information of all the leaf nodes on the DT, a list of the most used HwAs can be obtained. These can be kept preloaded in the FPGA fabric to minimise DPR overhead. Take a pair of identical devices (A & B) deployed in different conditions. Device A may use HwA 1 more often than device B that prefers HwA 2 due to the conditions surrounding the devices. By keeping HwA 1 preloaded on device A better performance can be achieved by cutting the reconfiguration time. Device A may also have a better CPU than device B. This can mean that, for particular scenarios, device A may prefer CPU execution where device B will use HwA 3.

We propose to explore the implementation of such decision algorithms, firstly through software implementations on-chip running in an auxiliary processor, and then via hardware implementations of the DT. Leveraging technologies such as HLS and DPR, we thus aim to increase the abstraction level given to programmers for the use of re-configurable resources by shifting these development concerns towards the compiler.

REFERENCES

[1] Abderazak Ben Abdallah. 2017. Heterogeneous Computing: An Emerging Paradigm of Embedded Systems Design. In *Computational Frameworks: Systems, Models and Applications*. Elsevier, 61–93. https://doi.org/10.1016/B978-1-78548-256-4.50003-X

[2] Mark T. Bohr and Ian A Young. 2017. CMOS Scaling Trends and Beyond. *IEEE Micro* 37, 6 (2017), 20–29. https://doi.org/10.1109/MM.2017.4241347

[3] Usman Dastgeer, Lu Li, and Christoph Kessler. 2013. Adaptive implementation selection in the SkePU skeleton programming library. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8299 LNCS. Springer, Berlin, Heidelberg, 170–183. https://doi.org/10.1007/978-3-642-45293-2{_}13

[4] D. del Rio Astorga, Manuel F. Dolz, Javier Fernandez, and Javier Garcia Blas. 2019. Hybrid static–dynamic selection of implementation alternatives in heterogeneous environments. *Journal of Supercomputing* 75, 8 (8 2019), 4098–4113. https://doi.org/10.1007/s11227-017-2147-y

[5] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery (ACM), New York, New York, USA, 71–80. https://doi.org/10.1145/347090.347107

[6] John L. Hennessy and David A. Patterson. 2019. *Computer architecture: a quantitative approach* (sixth ed.). Morgan Kaufmann.

[7] Mark Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, Vol. 57. 10–14. https://doi.org/10.1109/ISSCC.2014.6757323

[8] Intel. 2020. High-Level Synthesis Compiler. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html

[9] Ian Kuon and Jonathan Rose. 2007. Measuring the gap between FPGAs and ASICs. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26. 203–215. https://doi.org/10.1109/TCAD.2006.884574

[10] Zhe Lin, Sharad Sinha, and Wei Zhang. 2019. Towards efficient and scalable acceleration of online decision tree learning on FPGA. In *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*. Institute of Electrical and Electronics Engineers Inc., 172–180. https://doi.org/10.1109/FCCM.2019.00032

[11] William F. Ogilvie, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 2015. Fast automatic heuristic construction using active learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8967. Springer Verlag, 146–160. https://doi.org/10.1007/978-3-319-17473-0{_}10

[12] Gavin Vaz, Heinrich Riebler, Tobias Kenter, and Christian Plessl. 2014. Deferring accelerator offloading decisions to application runtime. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. IEEE, Cancun, Mexico, 1–8. https://doi.org/10.1109/ReConFig.2014.7032509

[13] Xilinx Inc. 2020. *Dynamic Function eXchange - Vivado Design Suite User Guide*. Technical Report. Online. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug909-vivado-partial-reconfiguration.pdf

[14] Xilinx Inc. 2020. *Vivado High-Level Synthesis*. Technical Report. Online. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[15] Gina Yuan, Shoumik Palkar, Deepak Narayanan, and Matei Zaharia. 2020. Offload annotations: Bringing heterogeneous computing to existing libraries and workloads. In *Proceedings of the 2020 USENIX Annual Technical Conference, ATC 2020*. 293–306. https://www.usenix.org/conference/atc20/presentation/yuan