Efficient FPGA Cost-Performance Space Exploration Using Type-driven Program Transformations

Cristian Urlea
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
c.urlea.1@research.gla.ac.uk

Wim Vanderbauwhede
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
wim.vanderbauwhede@glasgow.ac.uk

Syed Waqar Nabi
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
syed.nabi@glasgow.ac.uk

***Abstract*—Many numerical simulation applications from the scientific, financial and machine-learning domains require large amounts of compute capacity. They can often be implemented with a streaming data-flow architecture. Field Programmable Gate Arrays (FPGA) are particularly power-efficient hardware architectures suitable for streaming data-flow applications. Although numerous programming languages and frameworks target FPGAs, expert knowledge is still required to optimise the throughput of such applications for each target FPGA device.**

The process of selecting which optimising transformations to apply, and where to apply them is dubbed Design Space Exploration (DSE). We contribute an elegant and efficient compiler based DSE strategy for FPGAs by merging information sourced from the compiled application's semantic structure, an accurate cost-performance model and a description of hardware resource limits for particular FPGAs. Our work leverages developments in functional programming and dependent type theory to bring performance portability to the realm of High-Level Synthesis (HLS) tools targeting FPGAs. We showcase our approach by presenting achievable speedups for three example applications. Results indicate considerable improvements in throughput of up to $58\times$ in one example. These results are obtained by traversing a minute fraction of the total Design Space.

***Index Terms*—fpga, compiler, functional programming, design space exploration, high level synthesis, cost model**

I. Introduction and Background

The High-Performance Computing (HPC) arena has seen an increase in the adoption rate of FPGAs as a target hardware platform. This is primarily driven by the increased need for energy efficient computation at scale.

HLS tools allow programmers to quickly develop applications targeting FPGAs. Instead of describing computation as a circuit, as is the case with Hardware Description Languages (HDL), developers using HLS solutions specify the algorithms and functionality of the application leaving the task of synthesising an equivalent circuit to the compiler.

In the context of HPC applications, we argue that even this abstraction level too low. Both HDL and HLS solutions require expert programmers to guide the process of finding an efficient parallel implementation with explicit annotations.

The TyTra project is funded by the EPSRC Grant EP/L00058X/1 Exploiting Parallelism through Type Transformations for Hybrid Manycore Systems

We address this situation by contributing an efficient DSE strategy. Our compiler computes the optimal parallel program structure from the provided sequential application, a cost-performance model and the target hardware device's resource bounds. This search strategy builds upon our earlier work published in [1] by avoiding regions of the design space that would lead to inefficient hardware resource utilisation. This is done automatically, without instruction from the application developer.

We contrast our approach to the industry standards such as OpenCL implementations and other HLS tools. OpenCL [2] is a *heterogeneous* framework for general purpose parallel programming on CPUs, GPUs and even FPGAs [3]. FPGA vendors provide OpenCL programming flows, as well as other HLS solutions. Xilinx's Vivado [6] and Intel's HLS Compiler [8] for example both require specialist programmers to annotate their applications with explicit pragma directives to generate efficient parallel solutions.

II. Design Space Exploration

Design Space Exploration can be viewed as the application of two processes: a program variant *generator* and a program variant *filter*. The *generator* produces program variants through the application of term-level *program transformations* to each node in the Abstract Syntax Tree (AST). Term-level transformations are generated from type-level transformations as detailed in print [1]. The *filter* process removes program variants from the design space by comparing their relative performance and expected resource cost utilisation.

Within our formalism, the leaf nodes of the AST represent *opaque functions*, computations that work on scalar-inputs and produce scalar outputs. Branch nodes represent *higher-order functions* such as *map, fold, zip and unzip* that take *functions as inputs* and produce *functions as outputs*. A *map* node is the functional equivalent of a for loop in an imperative language.

In the naïve approach, the generator produces, for each leaf node, a number of program variants. Each corresponds to a particular degree of expressed parallelism. For branch nodes program variants are generated by selecting items from

TABLE I
BEST PERFORMING BOARD/EXAMPLE

Example	FPGA	Throughput Ratio	Registers	Block Ram	DSP	LUT	Program Variants	DSE Ratio	Relative Speedup
SOR	*baseline*	0.017	2432	3	40	4631	0	0	1x
SOR	XC6SLX4	0.031	3136	6	44	5473	8	0.001	1.8x
SOR	XC6SLX9	0.218	21248	39	304	37469	19	0.01	12x
SOR	XC6SLX16	0.375	36224	66	520	63992	33	0.02	22x
SOR	XC6SLX25	0.625	59904	108	864	106092	55	0.03	36.7x
SOR	**XC6SLX45**	1.0	95424	171	1380	169242	88	0.05	58.0x
1DS	*baseline*	0.043	64	0	4	0	0	0	1x
1DS	**XC6SLX4**	1.0	1472	0	92	0	23	1.0	23x
Synth	*baseline*	0.037	1388	7	13	3380	9	0	1x
Synth	XC6SLX4	0.26	8520	42	82	21122	13	0.02	7x
Synth	XC6SLX9	0.69	22496	112	214	55343	34	0.05	18x
Synth	**XC6SLX16**	1.0	32308	161	307	79424	49	0.07	27x

the Cartesian product of program variants associated to their respective sub-expressions. The *filter* process enumerates and selects program variants based on the performance-cost model. The search space is thus a rapidly growing function of the number of AST nodes and their expected latency.

III. APPROACH

We contribute a DSE strategy implemented in the Haskell programming language. We compose the *generator* and *filter* processes by expressing the former as an *Anamorphism* and the latter as a *Catamorphism* over the base Functor of our AST representation. The resulting Hylomorphism is a fused and optimised version of the two. This formulation traverses the search space according to implicit data dependencies allowing for a single bottom-up traversal of the AST.

We begin by conceptually assuming infinite computational resources. For each leaf node we produce program variants expressing the maximum level of parallelism leading to a balanced pipeline implementation with an aggregate throughput of 1 work item per clock cycle. The maximum degree of parallelism corresponds to the opaque function's input-to-output latency as provided by the cost-performance model, details of which may be found in print [7]. Assuming infinite computational resources allows our search-strategy to start from a globally optimal solution that may or may not fit on the target device. We thus avoid the problem of getting stuck in a local-maximum, as may happen with a gradient-decent strategy with random starts.

For branch nodes we make the observation that their throughput can only be as high as its least performing sub-expression. We thus generate branch node variants that express the same degree of parallelism as the node's sub-expression having the smallest number of variants. For each of these, all other sub-expressions only retain the minimum resource-cost variants that expresses that degree of parallelism.

This observation allows us to prune away entire regions of the design space at each step in our bottom-up AST Traversal. We account for the target FPGA device's resource constraints by immediately discarding program variants that exceed those bounds at each step. Once we have reached the root node, representing the application's output, only the Pareto-optimal frontier of program variants remains.

IV. RESULTS

We applied our DSE strategy to the compilation of three example applications with results visible in Table I. The first two of these are derived from real-world applications: a large-eddy weather simulator [4] and an ocean model [5]. The last example is synthetically derived from second. During DSE we supplied hardware resource bounds for a number of Xilinx Spartan 6 FPGAs namely: XC6SLX4, XC6SLX9, XC6SLX16, XC6SLX25, XC6SLX45 and XC6SLX150T.

The first columns reflect the example name, FPGA device targeted and the relative throughput compared to a fully balanced pipeline. Following that are the number of registers, block ram, DSP blocks and look-up tables slices required by the best performing variant found. The *Program Variants* column is the total number of variants generated by our DSE strategy. *DSE Ration* shows the proportion of considered variants compared to the total *naïve* search space. Finally the *Relative Speedup* is the expected gain in throughput, compared to that of the sequential input application.

REFERENCES

[1] W. Vanderbauwhede, S. W. Nabi, and C. Urlea, Type-driven automated program transformations and cost modelling for optimising streaming programs on fpgas, International Journal of Parallel Programming , pp. 123, 2019

[2] Khronos OpenCL Working Group, The opencl specification, https://www.khronos.org/registry/OpenCL/specs/opencl-1.0.pdf, 2009

[3] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh, From opencl to high-performance hardware on fpgas, in 22nd international conference on field programmable logic and applications (FPL) . IEEE, 2012, pp. 531534.

[4] H. Nakayama, T. Takemi, and H. Nagai, Large-eddy simulation of urban boundary-layer flows by generating turbulent inflows from mesoscale meteorological simulations, Atmospheric Science Letters, vol. 13, no. 3, pp. 180186, 20

[5] J. Kämpf, Ocean modelling for beginners: using open-source software. Springer Science & Business Media, 200

[6] T. Feist, Vivado design suite, White Paper , vol. 5, p. 30, 2012.

[7] S. W. Nabi and W. Vanderbauwhede, FPGA design space exploration for scientific HPC applications using a fast and accurate cost model based on roofline analysis, Journal of Parallel and Distributed Computing, 2017

[8] M. Sussmann, T.Hill Intel HLS Compiler: Fast Design, Coding, and Hardware White paper , 2017