Towards Efficient Code Generation for Exposed Datapath Architectures

Kanishkan Vadivel, Roel Jordans, Sander Stujik, Henk Corporaal
Eindhoven University of Technology
Eindhoven, The Netherlands

Pekka Jääskeläinen, Heikki Kultala
Tampere University
Tampere, Finland

ABSTRACT

Coarse-grained reconfigurable architectures and other exposed datapath architectures such as transport-triggered architectures come with a high energy efficiency promise for accelerating data oriented workloads. Their main drawback results from the push of complexity from the architecture to the programmer; compiler techniques that allow starting from a higher-level programming language and generate code efficiently to such architectures robustly is still an open research area. In this article we survey the known main sources of challenges and outline a generic processor architecture template that covers the most common architecture variations along with a proposal for a common code generation framework for such challenging architectures.

CCS CONCEPTS

• **Software and its engineering** → *Retargetable compilers*; • **Computer systems organization** → Reconfigurable computing.

KEYWORDS

code generation, reconfigurable architectures, CGRA, TTA, scheduling, energy efficiency

ACM Reference Format:
Kanishkan Vadivel, Pekka Jääskeläinen, Roel Jordans, Heikki Kultala, Sander Stujik, and Henk Corporaal. 2019. Towards Efficient Code Generation for Exposed Datapath Architectures. In *22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES '19), May 27–28, 2019, Sankt Goar, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3323439.3323990

1 INTRODUCTION

A conventional way of improving the computational capabilities of programmable processors is by increasing the number of *functional units (FUs)* that can execute independent operations in parallel. Although this method can be very efficient in practice (eg. VLIW or superscalar), its scalability is limited due to RF scaling issues [3] and energy-efficiency challenges [4].

Exposed datapath architectures address the scalability and energy efficiency issues by exposing more of the datapath to the programmer. Fine-grained control over the datapath allows program controlled bypassing of data between FUs at compile-time (software bypassing), and eliminates the need for data-routing and dependency checking circuitry in hardware. More importantly, it slackens the design requirements of the RF and bypass network as the data-bandwidth to RF need not satisfy the worst-case requirements. The combined effect of these two capabilities along with application specific instruction sets can result in energy-efficient programmable-processors capable of reaching near fixed function accelerator performance. Various CGRAs, TTAs, Silicon-Hive's DSPs, Mill, and Explicit-SIMD are examples of architectures that belong to this category [6].

CGRA [12] and TTA [3] style architectures are a favourable candidate for programmable accelerators as they satisfy the modularity, flexibility, scalability, and programmability requirements. Although CGRAs and TTAs have a potential of reaching very high energy-efficiency [13], their true benefits rely solely on the compiler when programmability with a *high-level language (HLL)* is desired. The complexity in designing a compiler for these architectures is increased due to the additional compile-time responsibilities such as operation-to-FU-binding, data routing on partially connected interconnection networks, partitioned RFs, etc. Previously, these issues have been typically solved by exposed datapath architecture-specific compilers which lack retargetability and do not benefit from engineering effort sharing.

In this paper, we outline some of the known shared challenges in code generation for CGRA and TTA architectures, and propose a means of unifying the engineering effort on a generic code generation framework. We suggest a means to achieve this by defining a common processor template that can support the most interesting explicit datapath architectures with the same or similar code-generation techniques, and by extending the existing available toolsets to support it.

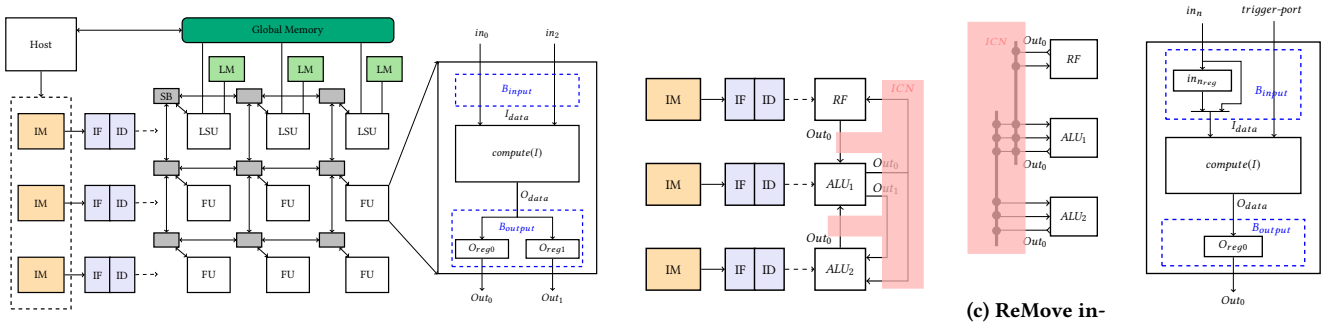
The remainder of the paper is organized as follows, in Section 2 we model the compiler view of the architectures in focus and propose a generic architecture template as a target-model for the generic compiler. The known challenges in code generation for the selected architectures along with research opportunities and required toolchain support is discussed in Section 3. Section 4 concludes the paper by summarizing our findings and proposes research directions.

2 CGRA AND TTA ARCHITECTURES

The class of TTA processors is well-defined in the literature and has been consistent over their generations [2, 3, 5, 9]. However, there exist many informal and conflicting definitions for CGRAs in the literature. Formal guidelines for classifying CGRAs are proposed

SCOPES '19, May 27–28, 2019, Sankt Goar, Germany

ACM ISBN 978-1-4503-6762-2/19/05…$15.00
https://doi.org/10.1145/3323439.3323990



(a) Blocks CGRA and its Architecture Graph of FU. (b) A Blocks CGRA instance. (c) ReMove instance. (d) ReMove FU

Figure 1: Blocks CGRA and ReMove architecture

in [12], and in this paper we use the term CGRA for architectures that have spatial granularity at the FU level and temporal reconfiguration granularity at the region/loop-nest level or above as most CGRAs belong to this category and their programming view is similar to of TTAs. The Blocks CGRA [11] fits well into this category and hence we picked it as a primary target and listed the differences to the other related architectures in comparison to it.

The Blocks CGRA consists of FUs that are arranged in a matrix form with switch box networks for connectivity as shown in Fig. 1a. The FUs can be either compute-storage units (ALU, multiplier, RF, branch, etc.) or memory access units (load-store units). The switch boxes in the design implements a network for operand and result forwarding, and the instructions issued to FUs defines the datapath at the cycle-level. Based on the application requirement, a group of FUs and their connectivity can be selected from the template to form a processor instance that mimics the spatial layout of the application. A sample instance of Blocks CGRA with two ALUs and an RF is shown in Fig. 1b.

Multi-issue TTA is a class of VLIW architectures. Unlike traditional operation-triggered architectures, in TTAs, the instructions represent the data transport and operations are executed as a side effect to the data transports [3]. Similar to CGRAs, a TTA instance can be formed by selecting a group of FUs and defining connectivity between them. Depending on the design of FU and the interconnect network, there exist a few variants of TTA architecture; Move [3], ReMove [9], MovePro [5], and TCE TTA [8] are few examples. ReMove is a reduced connectivity TTA with connections only to the neighbouring FUs which makes them an interesting target for the compiler and hence we selected them as a primary TTA subclass in this paper. A sample instance of a ReMove processor with two ALU and an RF is presented in Fig. 1c.

2.1 Programmer View

Executing an operation in an exposed datapath architecture involves binding of an operation to an FU and routing data (operand and result) through the interconnection network in such a way that it satisfies the timing requirements of the operation without resource and dependency conflicts. From the compiler point of view, this breaks down to describing target machine in the backend of a retargetable compiler and designing an *instruction scheduler* which can map computation on target by obeying a set of constraints imposed by the machine model and the set performance goals.

Supporting new architecture variants in a compiler requires extensions to the target machine model and the scheduler in the framework. By extending the target machine model, it is possible to reuse the associated toolset for newly supported architectures either with a generic enough scheduler or by inclusion of custom schedulers if the framework is flexible enough.

The first two of the following subsections model the programmer view of the exposed datapath architecture and defines a generic processor template. The defined template outlines the requirements of the target-machine model to support the considered architectures, assuming that the compiler and associated toolset has maximum flexibility. The flexibility requirement of the toolset is discussed in Section 3.2.

2.2 Machine Model

A processor instance of the selected architecture consists of a set of FUs and an *interconnection network (ICN)* for communication.

The connectivity of the ICN of the processor instance can be modelled as a flow network with capacitated vertexes termed *Connectivity Graph (CG)* and FUs can be modelled as an *Architecture Graph (AG)* as follows:

$$Connectivity\ Graph, CG = (S, C, D, E) \tag{1}$$

Where $S = \{S_0, S_1, .., S_n\}$ is set of FU result ports which produce data on the ICN, $C = \{C_0, C_1, .., C_n\}$ is set of communication resources in the ICN, $D = \{D_0, D_1, .., D_n\}$ is the set of sink nodes in the ICN (FU input operand ports), and E is a set of directed links between the vertexes. The capacity of the vertexes models the sharing property of the ICN. The CGs of the presented sample Blocks CGRA and ReMove instances are depicted in Fig. 2a and Fig. 2b respectively.

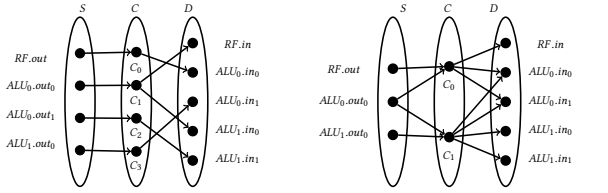
$$Architecture\ Graph, AG = (B_{input}, B_{output}, I, OB_{constrain}) \tag{2}$$

Where B_{input} *and* B_{output} models the input and output interface of FU with their associated buffers. For designs with shared operand and result ports, the effective buffer can be constrained to $B = B_{input} \cup B_{output}$ in *Scheduler*. $I = \{Ins : (I_{data}, InternalState) \rightarrow (O_{data}, InternalState)\}$ is set of supported instructions. The Operand Binding constraints ($OB_{constraints}$) define the constraints on I_{data} and O_{data} with respect to B_{input} and B_{output}. The AG for the sample processor instances are illustrated alongside their block-diagrams in Fig. 1a and Fig. 1d.

2.3 Generic Machine Model

The AG models the computation logic and interface of the FUs, CG models the connectivity between FUs, and together they describe a processor instance. The CG and AG for the selected TTAs, CGRAs, and their generic model is given in Table 1. The VILW is included in the table as a reference for comparison. The hardware bypasses in the processor are not modeled as they are not visible to the programmer.

The ICN in a generic TTA supports bus connections where each communication resource can have more than one source and implements a resource sharing protocol for communication. Depending on the protocol, the bus may also support multicast and broadcast. On the other hand, the Blocks CGRAs implements ICN with point-to-point connections with a dedicated communication resource for each source node. Formally, this property can be inferred from the CG as in-degree and out-degree of the capacitated communication resource C_n. The CG of point-to-point based ICN is a subset of the bus based connectivity in terms of in-degree, out-degree, and capacity model. Therefore supporting bus connectivity will enable the support for modeling the ICN of the all the listed architectures.



(a) Blocks CGRA instance **(b) ReMove instance**

Figure 2: Connectivity Graphs of the Blocks CGRA and ReMove instance given in Fig. 1b and 1c. Blocks-CGRA ICN is based on Point-to-point connectivity and hence the source S has one-to-one mapping to communication resource C. ReMove uses bus connectivity in ICN and therefore the communication resource C is shared by more than one source. The capacity of the C and D nodes enforces the resource sharing in the model.

The FU interface (B_{input} and B_{output}) and operation-binding constraints ($OB_{constrain}$) are architecture specific. In the considered TTAs, the operand ports are associated with single shadow buffer and results are buffered by an arbitrary size output buffer. The CGRAs support unbuffered and as well as buffered operand and result port with arbitrary size buffer. By targetting machine-model of the compiler to be a superset, all the considered architectures can be modeled in the compiler for codegeneration. The superset model of the considered architectures will be a model with buffered operand and result ports with arbitrary buffer size and support for bypassing the buffers (unbuffered operand and result ports).

3 COMPILER FRAMEWORK

The programmable datapath in combination with the static scheduling nature of exposed datapath architectures allows energy efficient mapping of applications in theory. However, in practice, how much this can be benefited from depends on how well the compiler can exploit the programmability. When programmability with an HLL is desired, a good quality compiler is essential for exploiting maximum possible instruction-level parallelism in the user application and utilizing target specific optimizations to achieve high energy efficiency.

Each design choice in the compiler influence the quality of the generated code. In addition to the traditional compiler design challenges, the ability to control the datapath data transfers introduces new challenges and further increases the complexity of the compiler. In the following, we outline the most important identified challenges in code generation for exposed datapath architectures in particular, to provide a glimpse of the complexity required by a generic code generation toolset.

3.1 Challenges in Exposed Datapath Code Generation

The main complexity shift comes from the additional instruction scheduler options; whether to perform software bypassing, dead-move elimination, operand and result sharing, and operand and result data transport scheduling freedom [3, 9].

In case of exposed datapath architectures, the buffers associated with the input and output ports of FUs are an additional level in the data memory hierarchy since they provide a programmer controlled storage. Efficient utilization of these buffers require extensions on both the register allocation and the instruction scheduler.

Instruction scheduling for exposed datapath architectures is an NP-hard problem [8] and most production compilers therefore use heuristics for scheduling.

For example, the compiler of the TCE toolset [7] implements a pre-pass register allocation followed by scheduling, since the complexities of inserting spill code possibly creating a cyclic dependency between register allocation and scheduling stages.

Operand binding and deadlock free scheduling problems are apparent when generating code to the selected architectures.

3.1.1 Operation Binding. Mapping of program operations to FUs is performed by means of the first fit algorithm with some simple heuristics in the current compilers [8]. The first free FU that contains the given operation is selected, and if there are multiple choices, the FU with fewer operations is selected to allow more generic FUs to be used for other operations.

The more complex heuristics take into consideration of the ICN connectivity and program patterns [1]. However, the operation binding can be carried out either before or during the scheduling process, and the effect of this design choice is not yet explored. As the schedule quality can be greatly influenced by this approach due to bad bindings resulting in unnecessary data copies across the network and reduced instruction level parallelism, more focus should be put to the operation binding techniques.

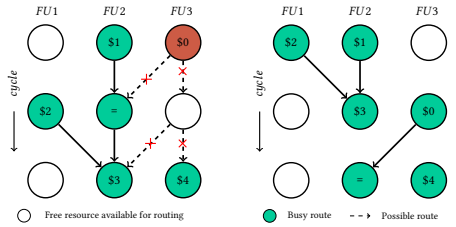
3.1.2 Deadlock Free Scheduling. The considered architectures implement reduced ICN connectivity for efficiency. As a consequence, all the results that are alive, but cannot be written to the RF must be, a) kept alive in the output register of the FU, b) transferred to a different FU by a copy operation until it is consumed or stored in the RF, or d) written back to memory.

Aggressively claiming FUs for operations may result in a deadlock, a state were a live value gets lost, as there is not enough free FUs to copy the value. Most of the existing compilers use an operation based list scheduling algorithm and it is easy to run into deadlock issues with them [5, 9]. An example of a deadlock issue is given in Fig. 3a. Current exposed datapath compilers address this

Table 1: Comparison of the supported exposed datapath architecture classes.

Architecture	ICN Connectivity		FU Model ($\forall fu$)			
	in-degree (C_n)	out-degree (C_n)	B_{input} ($\forall_{in-port}$)	B_{output} ($\forall_{out-port}$)	$OB_{constrain}$ I_{data}	O_{data}
ReMove	N	N	1 (except trigger port)	1	$inPorts \cup inPorts_{reg}$	$outPorts_{reg}$
Move-Pro	N	N	1	N	$inPorts \cup inPorts_{reg}$	$outPorts_{reg}$
Blocks-CGRA	1	N	0	1	$inPorts$	$outPorts_{reg}$
TRIPS [10]	1	N	N	0	$inPorts_{reg}$	$outPorts \cup inPorts_{reg}$
VLIW	1	$\leq \#(RF)$	0	0	$inPorts$	$outPorts$
Generic	N	N	N	N	$inPorts \cup inPorts_{reg}$	$outPorts \cup inPorts_{reg}$

issue either by scheduling the selected operation on a later cycle or by rescheduling the already scheduled instruction to have a solution similar to the one given in Fig. 3b with a trade-off of engineering cost for improved schedule. Constrained programming based scheduling does not have this problem but their scalability is limited. As the scheduling on partially connected architectures is a NP-Hard problem, finding (near)optimal schedule requires improvement to existing algorithms or new approaches.



(a) (b)

Figure 3: Schedule deadlock example. Schedule example is presented in a space-time graph, $0-4 represents the operations and = corresponds to the copy operation. In (a), scheduling operation $0 on FU3 will results in a deadlock as the output of $0 will be overridden at cycle-3. Potential solution would be rescheduling the operation $3 = $1 + $2 to an earlier cycle as shown in (b) to maintain liveness of the output from $0.

3.2 Towards Generic Code Generation Tools

Generic exposed datapath code generation tools should be retargetable to the set of supported architecture variations using a generic enough target machine template model. In addition to that, the selected framework should be modular enough to provide flexibility for research purposes and include supporting toolsets for the simplified user interface.

Most existing compilers of these architectures have C/C++ or an OpenCL C frontend with a target specific custom backend [12], which is not easy to extend. To the best of our knowledge, the TCE toolset [7] seems an ideal fit for this case as it is closely related to our use-case and has a robust compiler (C, C++, and OpenCL) accompanied with a rich set of supporting tools such as simulator, debugger, and GUI based processor design tools. Extending TCE to support the generic machine-model presented in Section 2.3 will benefit the architectures listed in Table 1.

As a proof of concept, we mapped Blocks CGRA to the TTA processor template of TCE. Our findings confirm that such extension is possible with a reasonable framework, and the framework has the flexibility and modularity needed for research and production compiler purposes.

4 SUMMARY AND FUTURE WORK

Towards a vision of a generic toolset for exposed data path architectures which can benefit from closely related codegeneration technique, we selected TTAs and subset of CGRA class architectures and proposed a generic processor template for TCE toolset. To support our claim, we mapped Blocks-CGRA on TCE and verified the functional correctness of the generated code. With minor extensions to the TCE scheduler, we managed to reuse most of the toolset including its cycle-accurate simulator.In our future work, we aim to identify commonalities in the codegeneration of the selected architectures and extend the generic compiler (TCE) to enhance code generation quality.

REFERENCES

[1] M. Bekooij, J. Jess, and J. van Meerbergen. 2001. Phase coupled operation assignment for VLIW processors with distributed register files. In *International Symposium on System Synthesis (IEEE Cat. No.01EX526)*.
[2] A. Cilio, H. Schot, J. Janssen, and P. Jääskeläinen. 2014. Architecture definition file: Processor architecture definition file format for a new tta design framework. (2014). http://tce.cs.tut.fi/specs/ADF.pdf
[3] Henk Corporaal. 1995. *Transport Triggered Architectures: Design and Evaluation.* Ph.D. Dissertation. Delft University of Technology.
[4] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. 2010. Understanding Sources of Inefficiency in General-purpose Chips. *SIGARCH Comput. Archit. News* (2010).
[5] Y. He, D. She, B. Mesman, and H. Corporaal. 2011. MOVE-Pro: A low power and high code density TTA architecture. In *SAMOS*.
[6] Pekka Jääskeläinen, Heikki Kultala, Timo Viitanen, and Jarmo Takala. 2015. Code Density and Energy Efficiency of Exposed Datapath Architectures. *Journal of Signal Processing Systems* (2015).
[7] Pekka Jääskeläinen, Timo Viitanen, Jarmo Takala, and Heikki Berg. 2017. *HW/SW Co-design Toolset for Customization of Exposed Datapath Processors.* Springer International Publishing.
[8] H. O. Kultala, T. T. Viitanen, P. O. Jääskeläinen, and J. H. Takala. 2016. Aggressively bypassing list scheduler for transport triggered architectures. In *SAMOS*.
[9] Steven Roos. 2001. Scheduling for ReMove and Other Partially Connected Architectures. (2001).
[10] K. Sankaralingam, R. Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, D. Burger, S. W. Keckler, and C. R. Moore. 2003. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*
[11] M. Wijtvliet, L.J.W. Waeijen, M. Adriaansen, and H. Corporaal. 2016. Reaching intrinsic compute efficiency requires adaptable micro-architectures. MULTIPROG.
[12] M. Wijtvliet, L. Waeijen, and H. Corporaal. 2016. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification. In *SAMOS*.
[13] J. Žádník and J. Takala. 2019. Low-power Programmable Processor for Fast Fourier Transform Based on Transport Triggered Architecture. In *Accepted to ICASSP 2019.*