Coarse Grained Reconfigurable Architectures in the Past 25 Years: Overview and Classification

Mark Wijtvliet, Luc Waeijen and Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
{m.wijtvliet, l.j.w.waeijen, h.corporaal}@tue.nl
5612 AZ Eindhoven, The Netherlands

***Abstract*—Reconfigurable architectures become more popular now general purpose compute performance does not increase as rapidly as before. Field programmable gate arrays are slowly moving into the direction of Coarse Grain Reconfigurable Architectures (CGRA) by adding DSP and other coarse grained IP blocks, general purpose processors become more heterogeneous and include sub-word parallelism and even some reconfigurable logic. In the past 25 years, several CGRAs have been published. In this paper an overview and classification of these architectures is presented. This work also provides a clear definition of CGRAs and identifies topics for future research which are key to unlock the full potential of CGRAs.**

***Index Terms*—Coarse Grain Reconfigurable Architecture, CGRA, Classification**

I. Introduction

In recent years, reconfigurable architectures have become increasingly popular due to their ability to adapt the architecture to the application. By doing so, these architectures can gain much higher performance per Watt over a wider range of applications than conventional processors[1].

This trend is reflected in the integration of conventional compute platforms with a reconfigurable fabric, such as the Xilinx Zynq and Altera SoC platforms. Even low-performance micro-controllers such as the Microchip PIC16(L)F150X series [2] now include configurable logic cells to offload some basic logic operations. For all these architectures the goal is to increase performance while keeping energy usage low, a property useful not only for mobile devices, but also large scale data centers [3].

In order to reach higher performance per Watt, field programmable gate arrays (FPGA) increasingly use coarse grain building blocks within their logic fabric. Examples of such building blocks are floating point DSPs [4], external memory interfaces and complete processor cores. These coarse grain building blocks are required to overcome the high cost, both in performance, and energy consumption [5], of the FPGAs interconnect. At the same time general purpose processors move towards a more heterogeneous and fine grained approach with the introduction of sub-word parallelism, narrower floating point units, and even some reconfigurable logic. This enables these architectures to achieve higher compute efficiency [6].

The increased popularity of reconfigurable architectures is also reflected in improved tool support, such as OpenCL support for FPGAs [7]. OpenCL allows application designers to specify a parallel application with much less design effort compared to conventional hardware description languages (HDL), such as Verilog and VHDL. Additionally it requires less in-depth hardware knowledge of the designer, since OpenCL is intended to be portable. However, while OpenCL allows rapid implementation of algorithms, the portability of OpenCL does not allow the description of bit-level optimized hardware. Hence, the FPGAs are used at a much coarser granularity with OpenCL compared to HDL.

Based on the addition of coarse grain blocks and OpenCL support in FPGAs, as well as the addition of finer grained operations in general purpose computing, it can be observed that both FPGAs and general purpose processors are moving in the direction of CGRAs. This is a strong indicator that reconfigurable architectures could be the perfect match between the FPGA and general purpose computing world.

The main contributions of this paper are:

1) Guidelines for CGRA classification.
2) A classification of CGRAs published in the past 25 years, providing an overview of the current CGRA landscape.
3) Guidelines for future CGRA research.

The remainder of this paper is organized as follows, in Section II CGRAs are further defined and this definition is used for the classification model of section III. Section IV presents a large selection of CGRA architectures published in the past 25 years. Each architecture is classified and listed in Table II. For each of these architectures a summary describes the main features. Section V evaluates the classification results, section VI provides guidelines for further research and Section VII concludes this paper.

II. Defining CGRAs

There exist many informal and conflicting definitions of CGRAs. One related work describes CGRAs as architectures that can only process loop bodies [8]. Another describes CGRAs as architectures that perform operations on a multi-bit level inside reconfigurable arrays [9]. However, the first definition excludes CGRAs that can perform full applications, and the second includes FPGAs with DSP blocks. A more generic definition that is used states that reconfigurable architectures use hardware flexibility in order to adapt the data-path at runtime to the application. They essentially provide a spatially programmable architecture [10, 11]. This requires that, before computation can be performed, some sort of configuration has to be loaded into the device. This configuration is typically static for some time, e.g. the duration of an application for an FPGA bitstream. However, virtually every computer architecture has a reconfigurable data-path to some extent. A typical general purpose processor uses cycle-based instructions to reconfigure the data-path to perform the desired operation, and

978-1-5090-3076-7/16/$31.00 ©2016 IEEE

even application specific integrated circuits (ASIC) might have configuration registers that influence the design's operation.

Formally defining the CGRA class of architectures proves complicated, because almost all architectures are reconfigurable at some level. To get a better understanding of this, we observe that a computer architecture can be abstracted to the functional block shown in Figure 1, which takes an input I, a configuration C, and uses these combined with an internal state S to produce an output O. The important part is that this abstraction can be made at various levels. If the data-path of a general purpose CPU is considered, then I is a data stream, C is an instruction stream, and O is the data stream produced by executing the instructions on a cycle-to-cycle basis. However, if the whole CPU is considered, then I would be a dataset, e.g. an image, and C is the application code. To the user the functionality is identical, the granularity is invisible.

It is interesting to note here that the classification of the input into operands/data I or configuration C is merely an artificial classification, which has its origin in the way computer architects think about their designs. Often the definition of what is configuration and what is data is unclear, if not interchangable. A good example is a look up table (LUT) in an FPGA. When programmed to perform a binary operation, the contents of the LUT can be considered a configuration. But when the LUT is used as a small memory, it would be reasonable to define those same contents of the LUT as part of the input data I.

Fig. 1. Abstraction of a computer architecture

Architecture classes reconfigure their hardware at different levels, and in different temporal and spatial granularities (See Figure 2). The size of the hardware blocks that are reconfigured, i.e., the level at which the system is observed according the the model described above, is the spatial domain of the reconfiguration. The granularity in the temporal domain specifies how long a configuration C is kept static before it is updated. A CPU for example reconfigures the operation of an arithmetic logic unit (spatial granularity) at cycle level (temporal granularity), while an FPGA reconfigures LUTs (spatial) at application level (temporal). It is even possible to reconfigure at the transistor level [12] in the spatial domain, but this in this work only gate level reconfiguration and coarser will be considered.

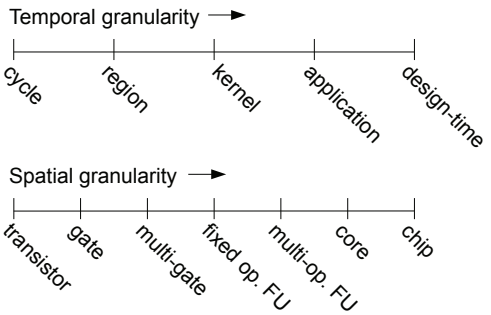

Fig. 2. Reconfiguration granularities in time and space

Considering the granularity of reconfiguration in both the temporal and spatial domain, it seems apt to define coarse grained reconfigurable architectures as those architectures that reconfigure at a 'coarse' level. Based on the CGRAs discussed in this work, we propose to define an architecture to be a CGRA if it posses the following properties:

1) A spatial reconfiguration granularity at fixed functional unit level or above.
2) A temporal reconfiguration granularity at region/loop-nest level or above.

Since there can many temporal and spatial levels of reconfiguration within the same system, as per the model in Figure 1, the focus should be on the dominant level when considering the properties above. Where the dominant level is the level at which (re)configuration has the greatest influence on the functionality of the system. For example, a CGRA can perform cycle-based instructions, but the width of these instructions is often several orders of magnitude below more static configurations at kernel or application granularity. Therefore, the kernel or application configuration is dominant. Figure 3 shows the architectural landscape for the dominant reconfiguration levels in both spatial and temporal dimensions. Reconfiguration can be either performed statically (S) or dynamically (D) and at multiple levels of spatial granularity within the same architecture. The choices made in various spatial levels determine the distribution of configuration bits over the temporal domain, as shown in Figure 4. An FPGA has only coarse temporal granularity while a CPU has mostly fine temporal granularity.

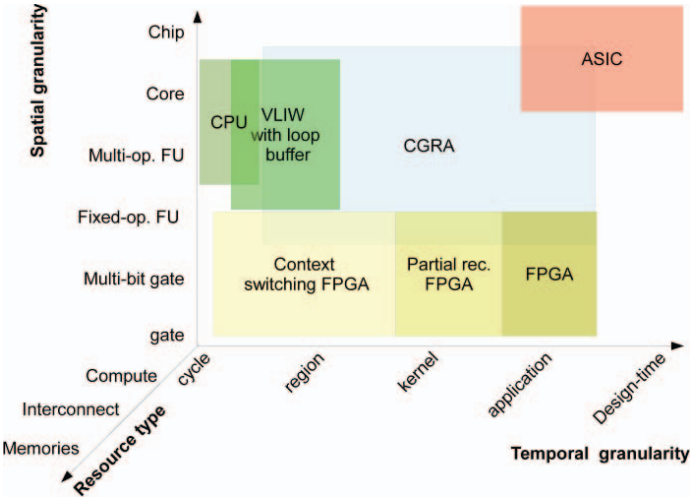

Fig. 3. Spatial-temporal architecture landscape. Horizontal and vertical axes indicate temporal and spatial granularity for reconfiguration.

It is important to note that the CGRA principles can be applied to different subsystems. In many architectures for example, a compute, interconnect, and memory subsystem can be identified. For each of these subsystems, the coarseness of reconfiguration can be chosen differently, which sketches the vast design space of CGRAs (Figure 3). Typically CGRAs differ from other architectures by allowing spatial mapping of applications over the available resources. To enable this,

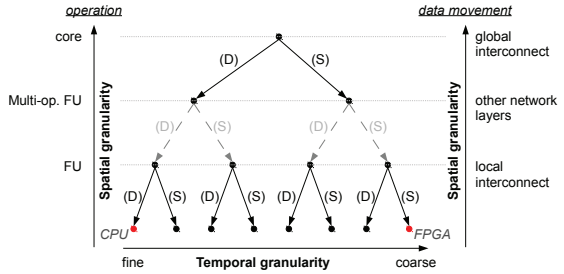

Fig. 4. Temporal distribution of configuration bits dictated by spatial reconfigurability at various levels

the interconnect subsystem has to be reconfigurable, causing many CGRAs to focus their reconfiguration options on the network. This is presumable also why Hartenstein decided that the interconnect is an important feature of CGRAs in his classification already more than a decade ago [9], a classification also adopted by a more recent survey [13].

III. Classification model

This section discusses the classification model that is used in section IV. For each architecture four main categories are evaluated and classified.

Structure - Each CGRA has different design choices for what is considered to be the *structure* of the architecture. The structure describes:

- Various network layers (e.g. topology and routing).
- FU data-path (e.g. FU granularity, FU operation types, register files and custom operation support).
- Memory hierarchy (e.g. scratchpads and caches)

Control - Scheduling (partial) configurations, both in time and space, can be performed either dynamic or static. For example, a compiler could decide the reconfiguration order (static schedule), or leave this up to the device (dynamic scheduling). Similarly, a place and route tool can statically decide which physical resource will perform an operation, or this can mapped dynamically at runtime. This holds for both the configurations of the FUs (including register files and local memories) as well as for network configurations.

Integration - CGRAs can be tightly or loosely coupled to a host processor, or do not need a host processor at all and support stand-alone operation such as TRIPS [14]. Some CGRAs such as ADRES [15] share resources with the host processor, others only feature a communication interface to the host processor.

Tooling - Any architecture's success depends heavily on the available tool support. For CGRAs this is often not only a compiler but also a place and route tool. However, comparing CGRAs just on their *available* tools is not a fair comparison. If tools are not available, an estimation will be made how feasible it is to implement the required tools, based on comparison with other architectures.

Table I shows the categories and properties that will be classified for each of the architectures in section IV.

IV. CGRAs in the past 25 years

This section summarizes and classifies a large selection of architectures that are either described by the authors of the original paper or referenced by other work as coarse grained reconfigurable architectures. The classification is summarized in Table II, while the rest of this section will give a brief summary of each investigated architecture. Finally in section V, these architectures will be matched against the CGRA definition given in section II.

Xputer (1991) [16] The XPuter architecture consists of a reconfigurable array (RA) of homogeneous units that are connected via a 2D mesh. The architecture distinguishes itself from most other CGRAs by the control flow. Instead of being controlled by a program counter based system, it is controlled via a data sequencer and control tags associated to the data stream, unlike a dataflow architecture the data access pattern in memory is explicitly controlled by the sequencer. C-compiler and place-and-route tools for this architecture exist.

PADDI (1992) [17] PADDI is built from several homogeneous FUs that are connected to a crossbar network. Each of these FUs contains an ALU for processing, and an instruction memory for control. The instruction memory is a 53-bit wide control word with eight entries. These entries can be different for each of the FUs, and act as one VLIW instruction. The active instruction is selected by a 3-bit index signal. The crossbar network is statically scheduled on a per-cycle basis. The PADDI toolflow provides a 'C-like assembly' language.

PADDI-2 (1993) [18] PADDI-2 is the evolution of PADDI, and improves on its predecessors shortcomings. In particular the centralized control of PADDI leads to instruction bandwidth problems and limits scalability. To overcome this,

	Property	Abbr.	Description
Structure	Network Type(s)	M	Mesh network
		B	Bus network
		C	Crossbar
		D	Direct connection (e.g. between FUs)
	Granularity	P	Design-time parameter
	Multi granular	Y/N	Yes/No
		S	Sub-word parallelism
	Fixed RF data-path	Y/N	Yes/No
	Memory hierarchy	M	Shared memory
		C	Cache
		S	Scratch-pad
Control	Scheduling	S	Static (compile-time)
		D	Dynamic (run-time)
	Reconfiguration	S	Static (compile-time)
		D	Dynamic (run-time)
	Network	S	Static (compile-time)
		D	Dynamic (run-time)
	Custom operations	Y/N	Yes/No
Integration	Integration	S	Standalone
		A	Accelerator
	Coupling	T	Tightly coupled
		L	Loosely coupled
	Resource sharing	Y/N	Yes/No
Tooling	Compiler	Y/N	Yes/No
		I	Imperative language
		D	Dataflow language
		C	Custom language
	Place and Route & DSE	Y/N	Yes/No

TABLE I
Classification table, and legend for table II

PADDI-2 follows a dataflow model of execution, with localized control. Instead of FUs, nano-processors are constructed which each have their own instruction store, and 3-bit program counter. Execution of an instruction on a nano-processor has no side-effects for other units, which enhances scalability. Furthermore the crossbar of the original PADDI architecture is replaced by a 2-level hierarchical network, implemented by a semi-crossbar with switchable buses.

rDPA/KressArray (1995) [19] The rDPA architecture, later renamed to KressArray, is strongly related to the XPuter architecture. They use the same functional unit design, but differ in architectural organization. rDPA consists of an RA with local, neighbour to neighbour, connections, as well as a global bus that connects all units to each other, the outside world, and a global controller. The global controller performs address generation and controls configuration loading. rDPA includes a C compiler and a custom hardware description language and compiler. Place-and-route is performed via simulated annealing.

Colt (1996) [20] Colt is an array of homogeneous 16-bit FUs connected through a toroidal mesh network, and a 'smart (pruned) crossbar' connects this mesh to the data ports of the chip. Each FU can perform integer operations, and contains a barrel shifter. FUs can be combined to support wider integer or floating point operations. The Colt architecture is statically scheduled, and FUs include delays to align computations. The Colt has three modes of operation for conditional execution, depending on the branch characteristics a specific mode is selected. The idea behind this is to keep the control overhead to a minimum for applications with simple control flow, but still provide support for applications that require runtime reconfiguration coupled with complex looping structures and conditional execution. A compiler is not provided.

MATRIX (1996) [21] The MATRIX architecture consists of a RA of homogeneous units, each with a memory, ALU, control logic, and several network ports. The network contains three levels, each in 2D layout. Each network layer progressively increases the distance spanned per connection from neighbourhood to global. The network is dynamically routed, but can be fixed by the configuration words. No compilation flow for this architecture is described.

RaPID (1996) [22] RaPID is a linear array of heterogeneous processing elements (PE). RaPID mostly resembles a systolic array, but with a reconfigurable interconnect and operation of the PEs. This leads to a highly configurable architecture, but also many control bits. RaPID solves this by defining 'hard' control, which is fixed for the duration of an application (large temporal granularity), and 'soft' control, which can be updated every cycle (fine temporal granularity). The division of bits in hard and soft differs per instance, but approximately 25% of the control is soft, and 75% hard. Furthermore RaPID features hardware support for executing loop-nests in an efficient manner. A compiler and programming language have been developed for the RaPID architecture [23].

Garp (1997) [24] The GARP architecture combines a MIPS processor with an FPGA-like accelerator. The MIPS processor controls the configuration of the RA via extensions to the MIPS instruction set. The RA consists of 2-bit logic elements that can form operations column-wise and can communicate row-wise via a wire network. By doing so the MIPS can offload more complex operations to the RA. Compilation for this architecture is performed via the GCC MIPS compiler, the instruction set extensions are called from assembly sections.

RAW (1997) [25] The RAW architecture is a 2D array of RISC-style pipelined FUs, each with their own instruction and data memory. These FUs can run independently from each other, and can communicate via a mesh network. Each FU has a memory that controls the destination addresses used by the network interface, hence the network access patterns can be statically scheduled. Dynamic routing is also possible. The FUs have byte-granularity and can be combined to form larger data types. A compiler based on a high-level language is available, and place-and-route is TIERS based [26].

PipeRench (1998) [27] The PipeRench architecture consists of several rows (stripes) of PEs. PEs in a row communicate through a neighbourhood network, and between rows a crossbar type of network handles communication. PipeRench exploits 'pipeline stage parallelism'. The stages of a pipelined loop are mapped to the available PEs. If there are more stages than PEs, the PEs are runtime reconfigured to execute different stages. If there are v stages and p PEs, then the throughput for $(v < p)$ is proportional to $\frac{p-1}{v}$, provided there are no dependencies between later stages of a previous execution of the pipeline. A compiler is available which takes a custom language (DIL) as input. A greedy place and route algorithm was developed that has runs much quicker than available commercial P&R tools, but still yields acceptable results due to the coarse grain nature of the architecture.

REMARC (1998) [28] The REMARC RA serves as a co-processor to a MIPS host and communicates via a bus to the host processor. The RA consists of a 64-FU grid of 16-bit FUs that are connected via neighbour to neighbour connections. The RA is controlled by a global control unit that manages data and generates a global program counter. Each FU has an instruction memory indexed by the program counter, within each row or column the FUs can function in SIMD mode. For compilation GCC is used. The application includes a binary file containing the RA configuration. The configuration is generated based on addresses in an assembly language.

CHESS (1999) [29] The appropriately named CHESS architecture defines a grid of 4-bit ALUs and switch boxes in a chequerboard pattern. This layout supports strong local connectivity between ALUs. 4-bit ALUs is small compared to typical CGRAs, but the ALUs of CHESS can be combined to support wider operations. An interesting feature of the switch boxes is that they can be configured to work as local RAMs. This, combined with the ability of ALUs to get instructions from the data network, allows the construction

of ‘mini-CPUs’ that execute a program from local RAM, whenever spatial layout is not efficient to implement (parts of) the application. A compiler does not exist, but is feasible. Complicating factors for compiler construction are dealing with the aforementioned ‘mini-CPUs’, and routing data when multiple ALUs are combined to support wide operations.

Pleiades (2000) [30] Pleiades defines an architecture template for reconfigurable architectures. The architecture is composed of a programmable microprocessor which acts as a host, and heterogeneous compute elements (satellites) which are connected via a reconfigurable interconnect. The satellites follow a data driven execution model, which has support for vector operations. As a proof of concept, the Maia processor was designed according to the Pleiades template. Maia is focussed on voice processing and uses a hierarchical mesh network, and also features an embedded FPGA. Code generation for Maia is not trivial, but has been done to some extent [31].

Chameleon/Montium (2000) [32] The Chameleon architecture is built on several VLIW-like processing units connected with a AMBA-bus. Each of these processors has VLIW-like control and layout. However, connections between the inputs and outputs of compute units and local memories, can be routed via a crossbar network. Configurations are stored in a local instruction memory, and are statically scheduled at compile time. Chameleon is part of FlexaWare by ReCore systems, and has a C-compiler and graphical design support.

DReaM (2000) [33] The DReaM reconfigurable architecture is highly specialized towards mobile wireless protocols. Reconfigurable processing units (RPUs) are grouped in clusters of four. Inside a cluster there is local interconnect, and inter-cluster communication is handled by a configurable SRAM-based switch box network. Both local and global interconnects use a handshaking protocol. Each RPU consists of two 8-bit reconfigurable arithmetic processing units (RAPs), and two dual-ported RAMs that can be used as a look-up-table to aid multiplication and division. Additionally each RPU has a spreading datapath unit for fast execution of CDMA-based spreading operations. A compiler is not provided for the DReaM architecture.

MorphoSys (2000) [34] The MorphoSys RA is a reconfigurable processor connected to a host processor via a system bus. The RA is an 8x8 grid of FUs, each containing an ALU, RF and multiplier. Each unit is configured by a 32-bit configuration word. Multiple contexts are stored in a context memory and can be broadcast row or column-wise to provide SIMD functionality. The network consists of four quadrants that are connected in columns and rows. Inside each quadrant a more complex direct network is present. For the MorphoSys architecture a C compiler is developed, but partitioning is performed manually.

Chimaera (2000) [35] Chimaera pushes the bounds on what can be considered a coarse grain reconfigurable architecture due to the bit-level granularity within the data-path. The architecture defines an RA of logic cells that is integrated into the pipeline of a host processor. The logic cells perform a reconfigurable operation at the bit level, but inside the array they are organized in word-wide rows because the architecture primarily targets word-level operations. A shadow register file allows access of up to nine operands. This allows the definition of custom operations in the host processor. The configuration for a custom operation is stored in the memory of the host, and can be cached locally for fast reconfiguration. Compilation can be done by selecting sequences of operations and mapping these to a single combined operation on the RA.

SmartMemories (2000) [36] The SmartMemories architecture consists of multiple processor tiles on a chip which communicate via a packet switched network. Four tiles are clustered into a ‘quad’ and are connected via a faster local network. The memory in each tile can be connected in various hierarchies (via a crossbar network) to the processor. By doing so the memory hierarchy can be adapted to the application. Each memory has an address generator and can be controlled via instructions. Compiler support is not mentioned, however this could be implemented based on a MIPS compiler with memory mapped interfaces to the crossbar network.

Imagine (2002) [37] The Imagine stream processor contains eight VLIW clusters that operate in SIMD mode and are under control of a microcontroller. Each cluster contains a configurable network that allows direct connection of ALU inputs and outputs, as well as a scratchpad memory and communication unit, much like the Chameleon [32]. All ALU clusters get their inputs from a stream memory. FUs start processing as soon as their input data is ready, thereby indicating a dataflow oriented architecture. Loop support is managed via a control processor. Streaming-C compiler support for this architecture has been developed.

ADRES (2003) [15] The ADRES architecture is a very tightly coupled VLIW and CGRA. The CGRA shares resources (including the register file and VLIW FUs) with the VLIW processor and can therefore run in either VLIW or CGRA mode. In CGRA mode the RA can access the VLIW’s register file, reducing transfer cost. The connections between the FUs in the RA are direct, however ADRES is a architecture template and therefore this can be modified at design time. ADRES features a C compiler for both VLIW and CGRA.

DART (2003) [38] DART defines clusters of ‘reconfigurable datapaths’ (DPRs). A DPR consists of FUs connected via a reconfigurable, entirely connected, multibus network. Both the fully connected network, and the operation of the FUs, can be configured. This results in many control bits for one DPR (122 bits per DPR, excluding bits for the interconnect between DPRs). The goal of DART is to keep the configuration static for as long as possible, but when a change in functionality is required, a technique similar to the ‘hard’ and ‘soft’ configuration bits in RaPID [22] is employed. DART allows a full update of all configuration bits, which takes 4-9 cycles, but also partial updates which can be completed in one cycle. The partial updates don’t affect the network, but only the

operations of the FUs, and the operand sources. A compiler is not available, but the multibus network in a DPR highly resembles a transport triggered architecture TTA[39].

PACT_XPP (2003) [40] PACT_XPP defines two types of processing array elements (PAE), one for computation, and one with local RAM. The PAEs are connected with a packet based network, and computation is event driven. Both the operation of the PAEs and the network are reconfigurable, which results in a large number of configuration bits. The event driven compute model means the control flow is handled in a decentralized fashion such that a configuration can be kept static as long as possible. To support irregular computations that do require to update the configuration, PACT_XPP uses two techniques. Firstly configurations are cached locally to enable fast configuration, and secondly *differential* configurations are supported. Differential configurations only update selected bits, which can keep them small in many cases, optimizing the use of the local configuration cache. The dataflow model of computation plus hardware protocols that guarantee that packets are never lost, ensures that reconfiguration is transparent to the user, making compilation quite easy.

WaveScalar (2003) [41] The WaveScalar architecture is a dataflow based reconfigurable architecture that contains a pool of processing elements. These processing elements are assigned instructions dynamically from a 'work-cache'. Scheduling instructions is dynamic and out-of order, but takes dependencies into account according to a dataflow description of the application. The architecture contains four network layers, from local direct connections between processing elements, to routed connections between processing clusters. The compiler for WaveScalar is able to translate imperative programs (such as C) into a dataflow description, this description is also used as the target code.

TRIPS (2004) [14] TRIPS is different from most CGRAs by it's out of order (OoO) data-driven nature. The architecture contains several OoO grid processors. Each of these processors contains 16 issue slots that can communicate via a routed mesh network. Each processor dynamically schedules frames of operations onto the RA, and a frame cache keeps the most recently used frames for reuse. The TRIPS IIRC Compiler maps basic block Data FLow Graphs (DFG) to the CGRA and is based on the Imagine framework. place-and-route is done within the frame-level and is performed at design time, but placement of the frame itself happens dynamically at runtime.

Kim[1] **(2004) [42]** Kim et al. propose a reconfigurable architecture that is based on MorphoSys [34], but with a notable exception. Where MorphoSys supports only the SIMD model, the proposed architecture also supports loop pipelining similar to PipeRench [27]. As a consequence the configuration of the reconfigurable compute elements (RC) can no longer be broadcast, but has to be stored locally, as each RC can perform a different operation. No compilation flow is available.

STA (2004) [43] The STA architecture consists of several clusters of compute elements. Each of these clusters operate in SIMD mode to provide vector processing capabilities. Within each cluster several processing elements with a fixed operation, such as addition, multiplication, and register files are present. The output of each processing element is connected to the input of all other processing elements. On a cycle basis a VLIW-like instruction activates one or more units, and selects which of the inputs will be used for computation. By doing so, a spatial mapping of the algorithm can be created. The STA architecture is supported by a GCC based back-end that translates MatLab code into executables.

Galanis[1] **(2005) [44]** Galanis et al. define a reconfigurable datapath, which is integrated in a SoC together with a host processor and embedded FPGA. Compute nodes inside the reconfigurable datapath consist of an ALU and a multiplier, and they can take operands from other nodes or a register bank. The interconnect between the nodes is a full crossbar, but the authors note that if scalability is an issue, a fat-tree network can be adopted. The compute nodes and interconnect allow for easy mapping of sub-graphs of an application DFG, making compilation straightforward. The control for the reconfigurable datapath is generated by the embedded FPGA, providing sufficient flexibility to support complex control flow.

Astra (2006) [45] This architecture features an RA of 8-bit FUs that can be combined to function as wider units. These FUs contain multiple static configurations that can be changed at run-time. These contexts are selected via a separate control network that, although not described in the paper, could be controlled by a host processor or by internal logic. The network between FUs is a neighbour to neighbour network, and FU resources can be used to route over longer distances. A compilation flow is not described, but will be similar to place-and-route on an FPGA.

CRC (2007) [46] The configurable reconfigurable core (CRC) is an architecture template for 'processor-like' reconfigurable architectures. In particular NEC's DRP[47] is an instance of this class of architectures. The architecture follows the typical CGRA template with an array of PEs with a reconfigurable interconnect which is not further specified by the CRC architecture. One of the key features of the CRC architecture are context memories inside the PEs which enable fast switching of configurations, similar to those in PACT_XPP and Tabula [48]. This fast switching of contexts enables 'a third dimension for routing by redirecting communication through the time domain'. For compilation the Cyber [49] framework is used, and 'techniques for simultaneous scheduling, placement and routing are developed on the basis of integer linear programming [50]'.

TCPA (2009) [51] TCPA consists of an array of heterogeneous, VLIW-style FUs, connected via a neighbourhood network. The heterogeneity of the FUs is a design-time decision. The interconnect between the FUs is statically configured, and forms direct connections between the FUs. Each FU has a (horizontal and vertical) mask that allows individual reconfiguration of FUs. In this way, SIMD type behaviour can

also be implemented. Unlike conventional VLIW processors, the register files in these FUs are explicitly controlled. Compilation for the architecture is performed based on a description in PARO, a dependency description language, and in later work on the PAULA language.

PPA (2009) [52] The polymorphic pipeline array (PPA) consists of clusters of four PEs called a core. The four PEs inside a core share an instruction cache, loop-buffer, and register file for predicate flags. Each PE has a register file, and a FU for arithmetic integer operations. Furthermore one PE per core can perform a multiplication. Inside a core the FUs are connected through a mesh network. For inter-core communication there exists three options, one of which is a neighbourhood network that links not the FUs, but the register files of adjacent cores. Another distinctive feature is the shared memory per x column of PEs through a memory-bus, where x is configurable to be $1, 2$ or 4. By having the amount of sharing configurable, the load latency can be kept low for applications that do not require sharing between cores. The compiler for PPA uses virtualized modulo scheduling to create schedules that can be virtualized on the available hardware resources.

CGRA express (2009) [53] CGRA Express is an architectural extension to the ADRES architecture. It adds the ability to perform latch-less chaining of functional units to construct complex single-cycle operations. The units are chained together via a bypass network. A modulo scheduling based compiler is extended to find common sub-graphs, and combine operations with sufficient clock slack into a single cycle.

EGRA (2011) [54] The EGRA architecture is constructed as an array of ALU clusters, called RACs. These RACs contain a 2D structure of ALUs that have a switchbox network between ALU rows. This amounts to a CGRA architecture with two network levels, a local network between ALU rows, and a neighbour network between clusters of these ALUs. The ALU clusters contain both local memories, a larger memory, a multiplier, and ALUs. A cluster is therefore heterogeneous, while the cluster array is homogeneous. A compiler is not available, but the paper shows a manual method for application mapping that could be automated.

DySER (2012) [55] The DySER architecture moves reconfigurable computing into the processor pipeline of an OoO processor. With DySER an extra issue slot is added to the execution stage. This issue slot contains a FIFO structure to construct vectors from sequential, and then feeds these to an RA. The DySER RA does not support loops by itself, but loops can be performed via the host processor. The compiler for DySER is a modified Sparc compiler that can perform sub-graph matching to recognize DySER operations, and maps these to the circuit-switched RA.

FPCA (2014) [56] The FPCA architecture has a 2D grid of clusters. Each cluster contains several compute elements, local memories and a crossbar network. Each compute element performs the same operation (Similar to a FPGA DSP). The actual operation is specified by providing constants. These constants and input values are routed between compute elements in the cluster by means of the crossbar network. The network between clusters is a neighbour to neighbour network. A scheduler performs dynamic resource allocation on the cluster array, and tries to replicate an application over the array until no more resources are available. An LLVM based compiler back-end is implemented which also performs placement. Routing is performed at run-time.

HARTMP (2016) [57] HARTMP is a reconfigurable matrix inside a RISC processor. Instructions are dynamically and transparently mapped to the RA. These RA configurations are stored in a configuration cache for future use. When an instruction pattern matches a previously executed RA configuration, it will get invoked instead of the instruction sequence. When the RA is activated the pipeline of the RISC processor will be stalled until processing on the RA is completed. This architecture has some similarities to DySER [55], but adds dynamic configuration generation. Since configuration generation is transparent, no compiler changes are required.

V. Evaluation

Section IV presents an overview and classification for a large selection of CGRAs published in the past 25 years. These architectures were either described by the authors of the original paper or referenced by other work as coarse grained reconfigurable architectures. Section II presented a definition for CGRAs. Since such a definition should align with the general perception of a CGRA, the architectures from section IV are analysed and classified in Figure 5. As can be observed, there is a strong correlation between our initial CGRA classification region in Figure 3 and the evaluated architectures, indicating that these guidelines represent the general consensus.

The following sections summarize the four main classification categories presented in section III.

A. Structure

Direct neighbour-to-neighbour and 2D-Mesh networks are the most popular choices. Both result in a regular grid that has good scaling properties. Crossbar networks are used on architectures with a lower number of FUs, or are used in FU clusters within a larger design. Although most CGRAs operate on a word-level granularity, there are some architectures that are multi-granular. The advantage of a multi-granular architecture is that the data-path of the architecture can adapt to the application at a lower level. Some architectures allow custom operations, either by some bit-level reconfigurable logic within the data-path, or by latch-less operation chaining to form complex single-cycle operations. The memory hierarchy is not very well defined in most architectures. They either have some local scratch-pad memories, or a shared memory/cache, but few present more hierarchical structures.

B. Control

The architectures with a higher level of dynamic control are slightly more popular compared to architectures with more

[1] Architecture has no name, first author's name is shown.

static control. This is especially true for dynamic (partial) reconfiguration, and the interconnect network. Therefore, most CGRA architectures seem to be aimed towards high performance rather than very high energy-efficiency. In order to reduce the required memory bandwidth for configuration loading and switching, some architectures feature configuration caches which reduce memory bandwidth requirements and allow fast configuration switches. Some architectures, such as RaPID [22], allow a variable ratio between hard (more statically configured) and soft (cycle-basis configuration). Doing so can be very beneficial for energy consumption.

C. Integration

CGRAs are both used as a stand-alone device, and as an accelerator. When used as an accelerator they tend to be loosely coupled and communicate via a bus or memory interface. Some architectures, such as ADRES [15], share memories or compute resources. Other architectures are completely integrated into the host processor [55, 57].

D. Tool support

Most architectures provide at least basic tool support such as an assembler, and place and route tools. A substantial number also provides a high-level language compiler. Often the compiler is C based, but domain specific language compilers (e.g. for dataflow descriptions) are also used. Some architectures feature an explicit data-path which makes compilation harder but can yield more energy efficient architectures. Place and route is mostly performed at compile-time, although some architectures perform mapping and routing at runtime. Most architectures do not provide any design space exploration tools, such as simulators and architecture models for power

General		Structure					Control				Integration			Tool support		
		Network	Datapath			Memory										
Architecture	Year of publication	Network type(s)	Granularity	Multi-granular	Register files	Memory hierarchy	Operation scheduling	(Partial) reconfiguration	Network configuration	Custom/Chained operations	Standalone / accelerator	Coupling	Resource sharing	Compiler	Place and Route	Automated DSE tools
Xputer [16]	1991	2D-M	32-bit	N	Y	-	S	S	S	N	S	-	-	D	Y	Y
PADDI [17]	1992	C	16-bit	Y	Y	-	S	D	D	Y	S	-	-	A	Y	Y
PADDI-2 [18]	1993	2D-M,D	16-bit	N	-	-	D	S/D	D	N	A	L	N	D	Y	Y
rDPA [19]	1995	B,D	32-bit	N	N	-	S	S/D	S	N	S	-	-	I	Y	Y
Colt [20]	1996	3D-M,C	16-bit	Y	-	-	S	D	D	Y	S	-	-	N	N	N
MATRIX [21]	1996	D,D,D	8-bit	Y	N	-	S/D	S/D	S/D	Y	S	-	-	N	N	N
RaPID [22]	1996	1D-B	P	Y	-	S	S	S/D	S/D	Y	A	T	N	I	N	N
Garp [24]	1997	D	2-bit	Y	N	M	S	S	S	Y	A	L	Y	I/A	Y	N
RAW [25]	1997	2D-M	8-bit, Bit	Y	Y	S	S	S	D	Y	S	-	-	I	Y	N
PipeRench [27]	1998	C,D	P	Y	Y	-	S	D	D	Y	A	T	N	C	Y	N
REMARC [28]	1998	D	16-bit	N	Y	S	S	S	D	N	A	L	N	A	N	N
CHESS [29]	1999	2D-M	4-bit	Y	Y	S	S	S	S	Y	A	L	N	N	N	N
Pleiades [30]	2000	D,M,M	P	N	-	S	D	D	S	Y	A	L	N	I	Y	N
Chameleon [32]	2000	C	32-bit	N	N	S	S	D	S	N	A	L	N	I	Y	N
DReaM [33]	2000	D,D	8/16-bit	N	-	S	S	D	D	N	S	-	-	N	N	N
MorphoSys [34]	2000	D,D	16-bit	N	N	S	S	D	S	N	A	L	N	I	N	N
Chimaera [35]	2000	D,B	4-bit	Y	Y	S	D	D	D	Y	A	T	N	I	Y	N
SmartMemories [36]	2000	B,D,C	64-bit	N	N	S	D	D	D	N	S	-	-	N	Y	Y
Imagine [37]	2002	C	32-bit	N	N	S	S	D	D	N	S	-	-	N	Y	N
ADRES [15]	2003	D	P	N	N	C	S	D	D	N	A	T	Y	I	Y	N
DART [38]	2003	B	16-bit	N	-	S	D	D	D	Y	S	-	-	N	N	N
PACT_XPP [40]	2003	M	P	N	-	S	D	D	D	N	S	-	-	I/C	N	N
WaveScalar [41]	2003	M,B,B,D	-	-	-	C	D	D	D	N	S	-	-	I	Y	N
TRIPS [14]	2004	2D-M	32-bit	N	Y	C, C	D	D	D	N	S	-	-	I	Y	N
Kim[1] [42]	2004	D,D	16-bit	N	N	S	S	D	S	N	A	L	N	N	N	N
STA [43]	2004	C, C	-	N	-	S	S	S	D	N	S	-	-	I	Y	N
Galanis[1] [44]	2005	C	16-bit	N	Y	C, M	S	D	D	N	A	L	Y	N	N	N
Astra [45]	2006	2D-M, D	8-bit	Y	-	-	S	D	D	Y	S	-	-	N	Y	N
CRC [46]	2007	-	P	N	Y	-	S	D	D	N	S	-	-	I	Y	N
TCPA [51]	2009	D/M/C/B	32-bit	N	N	-	S	S	S/D	N	S/A	L	N	D	Y	Y
PPA [52]	2009	2D-M	32-bit	N	Y	S, M	S	D	D	N	A	L	N	I	Y	N
CGRA express [53]	2009	D,D	N	N	N	C	S	D	D	Y	A	T	Y	I	Y	N
EGRA [54]	2011	D,D	-	N	-	M,S	S	D	D	N	A	L	N	N	N	N
DySER [55]	2012	2D-D	-	N	-	S	S	D	S	N	A	T	Y	I	Y	N
FPCA [56]	2014	2D-D,C	32-bit	N	N	M,S	S	D	D	N	A	L	N	I	Y	N
HARTMP [57]	2016	2D-M,M	-	N	-	N	D	D	D	N	A	T	Y	I	Y	N

TABLE II
ARCHITECTURE CLASSIFICATION, REFER TO TABLE I FOR A LEGEND

and performance estimation. Such tools can prove very useful for decisions between hard/soft configuration and interconnect variations. Only SRP [58], a variation on ADRES, uses a standardized programming model (OpenCL). This is an interesting field for investigation. OpenCL support for CGRAs will improve portability between GPUs, FPGAs, and CGRAs, and might lead to a wider industry acceptance of CGRAs.

The evaluated CGRAs target various application domains, including DSP, computer vision, encryption, and benchmarks such as SPECInt and Parboil.

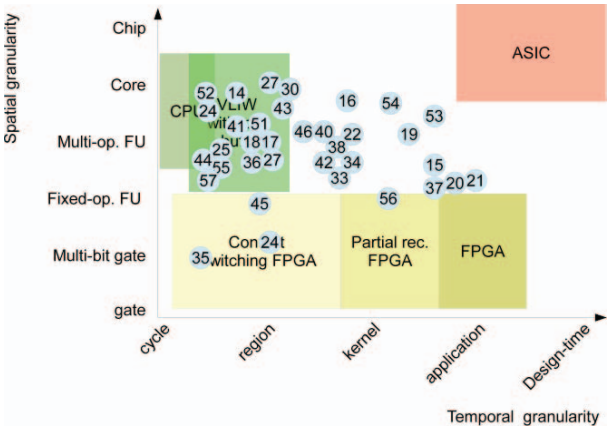

Fig. 5. Evaluated reconfigurable architectures

VI. Guidelines for future research

Based on analysis of existing CGRAs we identify four key topics for future research, which we regard as essential to unlock the full potential of CGRAs:

- Most of the evaluated architectures focus on the compute and interconnect aspect of the CGRA. Few architectures focus on the memory hierarchy despite that a significant part of the energy budget typically is spent on the memory
- CGRAs containing a dynamic configuration for both the compute and the interconnect are the most popular. These architectures can provide better application mappings, but dynamic reconfiguration has a power disadvantage. The trade-off between application mapping and the level of reconfigurability is not well understood.
- Few architectures feature design space exploration tools and models of the architecture, which would allow design-time power and performance estimation of specific architecture configurations. These estimations can guide application developers to target low-energy or high-performance design goals.
- Although many architectures feature some sort of tool-flow, very few are a commercial success. Often, the tool-flow is very architecture specific, thereby limiting industry acceptance. Programming models like OpenCL can help CGRAs compete with GPUs, and FPGAs.

VII. Conclusions

Since general purpose compute performance has stopped increasing as rapidly as before, a shift towards coarse grained reconfigurable logic is observed. Many architectures already contain coarse reconfigurable parts, but to meet the performance and energy demands of the next 25 years in computing, this reconfiguration should be at the heart of the architecture.

This paper provides a clear definition of coarse grained reconfigurable architectures. By analysing a large set of architectures that are referred to in literature as CGRAs, this definition is evaluated. The analysis of these architectures is presented as an overview of work on CGRAs in the past 25 years. Additionally a classification on *structure*, *control*, *integration* and *tooling* is presented. Based on analysis of existing CGRAs we identified four key topics for future research. CGRAs provide a promising compute paradigm, with great potential to reach higher compute performance and energy efficiency than is possible with current mainstream architectures, while maintaining sufficient flexibility to support a wide range of applications. With sufficient development of the field, CGRAs could be the solution to maintain performance and energy scaling beyond Moore's law.

References

[1] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. ACM, 2010.

[2] "Configurable logic cell tips n tricks," http://ww1.microchip.com/downloads/en/DeviceDoc/41631B.pdf, accessed: 2016-03-09.

[3] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Computer Architecture (ISCA), 2014*, June 2014.

[4] "The industrys first floating-point FPGA," https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/po/bg-floating-point-fpga.pdf, accessed: 2016-03-09.

[5] M. Wijtvliet, L. Waeijen, M. Adriaansen, and H. Corporaal, "Position paper: Reaching intrinsic compute efficiency requires adaptable micro-architectures," in *Programmability and Architectures for Heterogeneous Multicores, MULTIPROG 2016, Ninth International Workshop on*, 2016.

[6] J. M. Cebrin, L. Natvig, and J. C. Meyer, "Improving energy efficiency through parallelization and vectorization on intel core i5 and i7 processors," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012.

[7] 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 *Field Programmable Logic and Applications (FPL), International Conference on*, 2012.

[8] B. Sutter, P. Raghavan, and A. Lambrechts, *Handbook of Signal Processing Systems*. Boston, MA: Springer US, 2010, ch. Coarse-Grained Reconfigurable Array Architectures.

[9] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Proceedings of the conference on Design, automation and test in Europe*. IEEE Press, 2001.

[10] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, no. 3, March 2015.

[11] P. Gaillardon, *Reconfigurable Logic: Architecture, Tools, and Applications*, ser. Devices, Circuits, and Systems. CRC Press, 2015.

[12] P. Groeneveld and P. Stravers, *Ocean: the sea-of-gates design system*. Delft University of Technology. Faculty of Electrical Engineering, 1993.

[13] V. Tehre and R. Kshirsagar, "Survey on coarse grained reconfigurable architectures," *International Journal of Computer Applications*, 2012.

[14] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, N. Ranganathan, D. Burger, S. W. Keckler, R. G. McDonald, and C. R. Moore, "Trips: A polymorphous architecture for exploiting ilp, tlp, and dlp," *ACM Transactions on Architecture and Code Optimization*, 2004.

[15] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled VLIW processor and coarse-grained

reconfigurable matrix," in *Field Programmable Logic and Application*. Springer, 2003.
[16] R. W. Hartenstein, A. G. Hirschbiel, M. Riedmuller, K. Schmidt, and M. Weber, "A novel ASIC design approach based on a new machine paradigm," *Solid-State Circuits, IEEE Journal of*, vol. 26, no. 7, 1991.
[17] D. C. Chen and J. M. Rabaey, "A reconfigurable multiprocessor ic for rapid prototyping of algorithmic-specific high-speed DSP data paths," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, Dec 1992.
[18] A. K. Yeung and J. M. Rabaey, "A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput DSP algorithms," in *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, vol. 1. IEEE, 1993.
[19] R. W. Hartenstein and R. Kress, "A datapath synthesis system for the reconfigurable datapath architecture," in *Design Automation Conference, 1995. Proceedings of the ASP-DAC'95/CHDL'95/VLSI'95., IFIP International Conference on Hardware Description Languages. IFIP International Conference on Very Large Scal*. IEEE, 1995.
[20] "Colt: An experiment in wormhole run-time reconfiguration," in *Photonics East Conference on High-Speed Computing, Digital Signal Processing, and Filtering Using FPGAs*, 1996.
[21] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on*. IEEE, 1996.
[22] D. C. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," in *Advanced Research in VLSI, 1999. Proceedings. 20th Anniversary Conference on*. IEEE, 1999.
[23] D. C. Cronquist, P. Franklin, S. G. Berg, and C. Ebeling, "Specifying and compiling applications for RaPiD," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, Apr 1998.
[24] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Field-Programmable Custom Computing Machines, The 5th Annual IEEE Symposium on*, Apr 1997.
[25] E. Waingold, "Baring it all to software: Raw machines," 1997.
[26] C. Selvidge, A. Agarwal, M. Dahl, and J. Babb, "TIERS: Topology independent pipelined routing and scheduling for virtualWire compilation," in *Proceedings of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays*, ser. FPGA '95. ACM, 1995.
[27] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," *Computer*, Apr 2000.
[28] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable multimedia array coprocessor," *IEICE Transactions on information and systems*, 1999.
[29] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*. ACM, 1999.
[30] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, Nov 2000.
[31] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and J. Rabaey, "Design methodology of a low-energy reconfigurable single-chip DSP system," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 28, no. 1-2, 2001.
[32] G. J. M. Smit, A. B. J. Kokkeler, P. T. Wolkotte, P. K. F. Hölzenspies, M. D. van de Burgwal, and P. M. Heysters, "The chameleon architecture for streaming DSP applications," *EURASIP J. Embedded Syst.*, 2007.
[33] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk, "Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems," in *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*. IEEE, 2000.
[34] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, May 2000.
[35] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, June 2000.
[36] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: a modular reconfigurable architecture," in *Computer Architecture, Proceedings of the International Symposium on*, 2000.
[37] U. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany, "The Imagine stream processor," in *Proceedings 2002 IEEE International Conference on Computer Design*, Sep. 2002.
[38] R. David, D. Chillet, S. Pillement, and O. Sentieys, "DART: a dynamically reconfigurable architecture dealing with future mobile telecommunications constraints," *Résumé*, 2003.
[39] H. Corporaal, *Microprocessor Architectures: from VLIW to TTA*. {John Wiley & Sons}, 1997.
[40] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPPA self-reconfigurable data processing architecture," *the Journal of Supercomputing*, vol. 26, no. 2, 2003.
[41] S. Swanson, A. Schwerin, M. Mercaldi, A. Petersen, A. Putnam, K. Michelson, M. Oskin, and S. J. Eggers, "The WaveScalar architecture," *ACM Trans. Comput. Syst.*, vol. 25, no. 2, May 2007.
[42] Y. Kim, C. Park, S. Kang, H. Song, J. Jung, and K. Choi, "Design and evaluation of a coarse-grained reconfigurable architecture," *Proc. of ISOCC04*, 2004.
[43] G. Cichon, P. Robelly, H. Seidel, E. Matúš, M. Bronzel, and G. Fettweis, "Synchronous transfer architecture (STA)," in *Computer Systems: Architectures, Modeling, and Simulation*. Springer, 2004.
[44] M. D. Galanis, G. Theodoridis, S. Tragoudas, and C. E. Goutis, "A reconfigurable coarse-grain data-path for accelerating computational intensive kernels," *Journal of Circuits, Systems, and Computers*, 2005.
[45] A. Danilin, M. Bennebroek, and S. Sawitzki, "Astra: An advanced space-time reconfigurable architecture," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, Aug 2006.
[46] T. Oppold, T. Schweizer, J. Oliveira Filho, S. Eisenhardt, and W. Rosenstiel, "CRC–concepts and evaluation of processor-like reconfigurable architectures (CRC–konzepte und bewertung prozessorartig rekonfigurierbarer architekturen)," *it–Information Technology (vormals it+ ti)*, vol. 49, no. 3, 2007.
[47] H. Amano, T. Inuo, H. Kami, T. Fujii, and M. Suzuki, *Field Programmable Logic and Application: 14th International Conference, FPL 2004, Leuven, Belgium, August 30-September 1, 2004. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, ch. Techniques for Virtual Hardware on a Dynamically Reconfigurable Processor – An Approach to Tough Cases –.
[48] "Going beyond the FPGA with spacetime," http://www.fpl2012.org/Presentations/Keynote_Steve_Teig.pdf, accessed: 2016-04-12.
[49] K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: an ASIC and system vendor perspective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Dec 2000.
[50] J. A. Brenner, J. C. V. D. Veen, S. P. Fekete, J. O. Filho, and W. Rosenstiel, "Optimal simultaneous scheduling, binding and routing for processor-like reconfigurable architectures," in *Field Programmable Logic and Applications, FPL. International Conference on*, 2006.
[51] H. Dutta, D. Kissler, F. Hannig, A. Kupriyanov, J. Teich, and B. Pottier, "A holistic approach for tightly coupled reconfigurable parallel processors," *Microprocess. Microsyst.*, vol. 33, no. 1, Feb. 2009.
[52] H. Park, Y. Park, and S. Mahlke, "Polymorphic pipeline array: A flexible multicore accelerator with virtualized execution for mobile multimedia applications," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2009.
[53] Y. Park, H. Park, and S. Mahlke, "CGRA express: Accelerating execution using dynamic operation fusion," in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '09. ACM, 2009.
[54] G. Ansaloni, P. Bonzini, and L. Pozzi, "EGRA: A coarse grained reconfigurable architectural template," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, June 2011.
[55] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "Dyser: Unifying functionality and parallelism specialization for energy-efficient computing," *Micro*, 2012.
[56] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou, "A fully pipelined and dynamically composable architecture of CGRA," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*. IEEE, 2014.
[57] J. D. Souza, L. C. M. B. Rutzig, and A. C. S. B. Filho1, "A reconfigurable heterogeneous multicore with a homogeneous ISA," *DATE*, 2016.
[58] H. S. Kim, M. Ahn, J. A. Stratton, and W. m. W. Hwu, "Design evaluation of opencl compiler framework for coarse-grained reconfigurable arrays," in *Field-Programmable Technology (FPT)*, Dec 2012.