Blocks: Challenging SIMDs and VLIWs With a Reconfigurable Architecture

M. Wijtvliet, A. Kumar, *Senior Member, IEEE*, and H. Corporaal

***Abstract*—Demand for coarse grain reconfigurable architectures (CGRAs) has significantly increased in recent years as architectures need to be both energy efficient and flexible. However, most CGRAs are optimized for performance instead of energy efficiency. In this work, a novel paradigm for reconfigurable architectures, Blocks, is presented. Blocks uses two separate circuit-switched networks, one for control and one for the data path. This enables the runtime construction of energy-efficient application-specific VLIW-SIMD processors on a reconfigurable fabric. Its energy efficiency is demonstrated by comparing Blocks to four reference architectures, a VLIW, an SIMD, a commercial low-power microprocessor, and a traditional CGRA. All comparisons are based on commercial low-power 40-nm CMOS layout, including memories. Results show that Blocks can achieve a mean total energy reduction of 2.05×, 1.84×, 8.01×, and 1.22× over a VLIW, an SIMD, an energy-efficient microprocessor and a traditional CGRA, respectively. At the same time, Blocks delivers equal or higher performance per area due to its ability to adapt to applications by reconfiguration.**

***Index Terms*—Coarse grain reconfigurable architecture (CGRA), energy efficiency, reconfigurable architecture.**

I. Introduction

DEVICES containing reconfigurable logic have become increasingly popular in recent years. In part, this can be attributed to increased performance and cost reduction. Post production updates and rapid development of new algorithms and standards are another reason. Whereas, ASICs would require silicon updates and a new production run, devices like field-programmable gate arrays (FPGAs) can be reconfigured to correct an error or implement new functionality.

Reconfigurability comes, of course, at a cost. Taking FPGAs as an example: the reconfigurable interconnect causes longer wires between gates than those in ASICs, reducing the maximum frequency and performance, and increasing area and power. The large configuration memory controlling the interconnect and operation of the configurable logic causes high static power dissipation [1], [2], reducing energy efficiency. FPGA manufacturers have introduced more coarse-grained building blocks, such as digital signal processing (DSP)-blocks [3], [4], RAM-blocks, and dedicated interfaces to reduce area for typical DSP applications and improve performance. These coarse-grained blocks focus at higher bit-width operations, doing so allows FPGAs to become more energy and area efficient.

However, easy programming of FPGAs is still a concern. High-level languages, such as OpenCL [5], improve FPGA programmability by allowing FPGAs to be programmed in a similar way as graphical processors, enabling easier transition toward reconfigurable devices. However, OpenCL describes the structure of the application on a much more coarse-grained level. This implies that, when used for FPGAs, the gate-level reconfigurable logic is used at a more coarse-grained level, thereby not using the FPGA at maximum efficiency. Both of these trends, the introduction of coarse-grained building blocks and high-level languages, indicate a need for more coarse-grained architectures [6].

On the other side of the spectrum, combinations of general-purpose processors and hardware accelerators are being used more extensively. Accelerators provide the required performance for certain applications at much higher energy efficiencies than general-purpose processors [7]. To optimize area efficiency and to increase flexibility of the processor, devices such as the Xilinx Zynq [8] and Intel SoC [9] have emerged. These devices tightly couple high performance general-purpose processors with an FPGA fabric, allowing accelerators to be configured at runtime. Many accelerators used in mobile devices, where energy efficiency is the key, perform DSP applications and generally do not require the bit-level reconfigurability of the FPGA.

Coarse-grained reconfigurable architectures (CGRAs) allow reconfiguration at a granularity (far) above the gate level, such as functional units (FUs) or lightweight processors that can be connected via a static or dynamically reconfigurable network. In DSP applications, where it is much more common to use multibit arithmetic rather than bit-level operations, CGRA style FUs are more efficient in performance, area, and energy compared to FPGAs. Despite the possible gains in energy efficiency, most CGRAs in the past decades focused on performance [10]. With increasing demands on energy efficiency, combined with flexibility, CGRAs will have a strong advantage over FPGAs [6]. The question is whether the penalty of flexibility is not too high when compared to energy efficiency of SIMD, VLIW, or more dedicated architectures.

In this article, an energy-efficient CGRA, called Blocks, is presented. Blocks is unique due to its separation of the control path and data path. The architecture allows lightweight

Manuscript received April 28, 2021; revised July 29, 2021 and October 7, 2021; accepted October 13, 2021. This article was recommended by Associate Editor W. Zhang. *(Corresponding author: M. Wijtvliet.)*

M. Wijtvliet and H. Corporaal are with the Electronic Systems Group, Eindhoven University of Technology Eindhoven, 5612 AZ Eindhoven, The Netherlands (e-mail: m.wijtvliet@tue.nl; h.corporaal@tue.nl).

A. Kumar is with CFAED, Dresden University of Technology, 01069 Dresden, Germany (e-mail: akash.kumar@tu-dresden.de).

Digital Object Identifier 10.1109/TCAD.2021.3120541

instruction fetcher and instruction decoder units (IFIDs) to be arbitrarily connected to one or more FUs over a statically configured interconnect. By doing so, the designer can instantiate a processor that matches the parallelism properties of the applications. For example, one IFID can be connected to more than one FU to construct a vector operation.

The main contributions of this article are as follows.

1) An introduction into how Blocks can be used to construct SIMD and VLIW processors, as well as application-specific structures.
2) A comparison between Blocks, an SIMD, VLIW, and microprocessor. To the best of our knowledge, this is the first post place-and-route comparison on a commercial (40-nm low-power) technology of a CGRA and reference architectures. The evaluation shows that Blocks can be more energy efficient than fixed architectures and provide better performance at the same time.
3) A flexibility analysis of Blocks with respect to a VLIW, SIMD, ARM Cortex M0, and a traditional CGRA.

Some of these contributions appear in our earlier work [11]. The goal of this work was to show the advantage of separating control and data for CGRAs. For this reason, a more traditional CGRA was compared to Blocks. The goal of this work is different and extends our earlier work by showing that Blocks can approach, and often beat, the energy efficiency of fixed processor architectures. To achieve this, Blocks is compared against three fixed reference architectures: 1) a VLIW; 2) an SIMD; and 3) a microprocessor. Furthermore, a flexibility analysis is provided.

The remainder of this article is organized as follows, Section II describes applicable related works, Section III describes architectural details of Blocks, and Section IV describes how various processor structures can be instantiated on Blocks. Section V describes the experimental setup and reference architectures, and Section VI evaluates the obtained results. Finally, Section VII concludes this work and describes future work.

II. Related Work

FPGAs typically provide bit-level architectural flexibility; the overhead from the interconnect and look-up tables (LUTs) can be quite significant. Modern FPGAs reduce the overhead by introducing specialized, more coarse grained, building blocks. Some FPGAs even support floating point operations in hardware [3]. Research architectures have explored other angles for power reduction; examples thereof are the Astra [12] architecture which essentially is a more coarse-grained FPGA consisting of 8-bit building blocks, and [13] which is specialized for FIR filter processing. An Astra building block supports multiple configurations that can be selected at runtime, placing Astra on the border between FPGAs and CGRAs. Blocks uses an FPGA-style circuit switched network to interconnect the data path of FUs and extends this to the control path as well. In contrast to other reconfigurable architectures, this allows the control to be reused for multiple FUs and thus allow SIMD implementations. Blocks is also much more coarse-grained than multibit FPGA architectures, a functional unit in Blocks typically is an arithmetic and logic unit (ALU) or a load-store unit. FPGA designers have also explored other angles to achieve a reduction in reconfiguration overhead. An example thereof is [14], which presents an energy efficient FPGA that supports partial reconfiguration. This FPGA partially integrates into the processing pipeline of a Internet of Things sensor node.

In the past decades, several CGRA architectures have been proposed; there are several survey works that provide an extensive overview of these architectures [15], [16]. Some of the best-known CGRA architectures are probably Xputer [17], TRIPS [18], ADRES [19], and RAW [20]. Wijtvliet *et al.* [10] evaluated 36 CGRAs and conclude that most are not aimed at energy efficiency. TRIPS, for example, performs power-hungry out of order operations. In fact, only four of these 36 works present energy or power numbers, one of which compares power to signal processors (in 130-nm technology). While ADRES focusses on energy efficiency it has local register files per functional unit and cannot perform SIMD operations. The architecture proposed in this work is intended for low-energy processing and achieves this goal in a unique way: by separating control and data and allowing flexible construction of arbitrary VLIW-like processors with true SIMD support at runtime. The approach that Blocks uses has parallels with application-specific instruction-set processor (ASIP) design [21], [22]. ASIPs are, typically, not reconfigurable at runtime but provide optimized architectures for (a small set of) applications or an application domain. These processors are often VLIW processors with optimized instructions and bypasses between issue slots. Although much more energy efficient, these architecture instances lack the flexibility to be generally applicable. Blocks, however, allows to define the structure of these processors at runtime, especially with respect to instruction-level and data-level parallelism. An architecture that performs partial separation of control is KAHRISMA [23], which has instruction fetchers that can translate several traditional ISAs into an ISA supported by the architecture. However, in contrast to Blocks these translated instructions are then forwarded to each individual FU and stored locally before execution starts. Each of the KAHRISMA FUs is effectively an independent processor after it receives the translated kernel and aimed at performance and flexibility rather than energy efficiency.

Other interesting CGRAs that aim at energy reduction are HARTMP [24], X-CGRA [25], and Versat [26]. All three CGRAs approach energy efficiency improvements in very different ways. HARTMP implements a CGRA in the pipeline of a RISC processor. Repeated instruction patterns are automatically recognized and dynamically scheduled on the accelerator. X-CGRA implements a configurable level of approximation within its function units. This results in an approximate CGRA that can dynamically adjust to quality requirements. Versat supports fast partial reconfiguration of the array. This allows on the fly switching between configurations, thus adapting the hardware to (part of) the application. Blocks is intended to work next to a processor as an accelerator, or a stand-alone device. In the future it is possible to add X-CGRA like approximation to

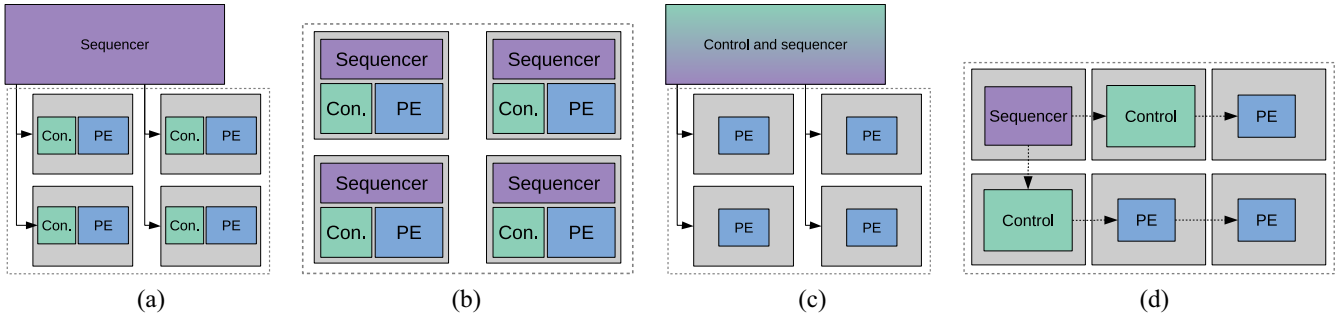

Fig. 1. Various control schemes used in existing CGRAs. The sequencer is responsible for controlling the program flow. Control (Con.) is responsible for loading the configuration or instructions into the PEs. (a) Global sequencing, local decoding. (b) Local decoding and sequencing. (c) Global context switching. (d) Blocks control scheme.

Blocks as a independent, complementary, energy reduction technique.

Many-core architectures are another flexible solution [27]–[29], they typically consist of several independent processors as opposed to a CGRA where a processing element is usually not a complete processor. The processor nodes are connected via a network, which in contrast to FPGAs is not static but a dynamically routed network. These networks are usually packet switched and not circuit switched. The design of many-cores gives a high amount of flexibility but at high power consumption. An interesting architecture is Transmuter [30], this reconfigurable many-core provides a reconfigurable cross-bar between processor tiles and between processing elements (PEs) in these tiles. Energy efficiency is achieved by allowing these PEs to operate as reconfigurable systolic arrays. Blocks supports only statically routed circuit switched networks for energy efficiency considerations. However, the Blocks FUs allow a connected network source to be selected at every cycle. Blocks also supports multiple independent PC generators inside its fabric, allowing construction of multiple independent application-specific VLIW-SIMD processors. According to [7], in which the authors performed an extensive evaluation of overhead in general-purpose and signal processors, specialized data paths exploiting instruction-level and data-level parallelism through adaptation to an application can obtain significant energy efficiency gains. The goal of Blocks is to allow this kind of adaptation, but by using reconfiguration instead of design time optimization.

III. Blocks Architecture

This section describes the motivation behind control path and data path separation found in Blocks and gives an introduction into the architectural details, such as the functional units, the memory hierarchy, the ISA, and tool-flow. Fig. 2 shows an example Blocks instance with different types of functional units, as well as the memory hierarchy around the architecture. This specific Blocks instance is shown for illustrative purposes only, and is not used for the benchmark results in this work.

A. Separation of Control Path and Data Path

In most processor architectures, instruction fetching and decoding significantly contributes to the total power [7]. This is certainly the case for CGRAs, where FUs perform local decoding [Fig. 1(a) and (b)] or are connected to a global configuration context [Fig. 1(c)]. In Blocks, the sequencer and control units are part of the reconfigurable fabric as individual building blocks, as shown in Fig. 1(d). By doing so, Blocks enables reconfiguration of the control structure as well as the data path. The reconfigurable control structure of Blocks allows construction of processors that support VLIW and SIMD programming models as well as combinations of these. These processors behave similar to VLIW-SIMD processors, but with extensive explicit bypassing. This means that intermediate results are directly passed from one function unit to another, instead of via register files. Doing so reduces register file (RF) reads and writes, thus reducing energy. The processors configured onto the Blocks fabric can include application-specific structures, such as reduction trees or (parts of) filters in the spatial layout, which can be controlled in an SIMD manner. This allows the architecture to take advantage of spatial layout of applications and enable software pipelining, reducing RF pressure and memory accesses, and increasing energy efficiency.

The Blocks fabric is realized by two FPGA-like networks operating on data buses instead of individual wires, both are configured once per application and are static during program execution. The data network (shown in blue in Fig. 2) allows the inputs and outputs of FUs to be connected to allow direct data transfer between FUs. The second network is the control-network (shown in red in Fig. 2), this network allows IFIDs to be connected to one or more FUs to create SIMD processors. By connecting multiple IFIDs to the same PC, generated by the accumulate and branch unit (ABU), VLIW-SIMD processors can be instantiated. If there is more than one ABU present within a Blocks instance, there can be multiple independently operating processors on the Blocks fabric.

B. Functional Units

The Blocks architecture features several types of functional units, each responsible for providing different functionality to the configured processor. Most FUs have multiple input ports and output registers connected to the data network, which are selected by the instruction, see Fig. 3. The instruction is provided by the IFID connected to the FU over the control network. Table I shows an overview of the available FUs and their function in Blocks.

TABLE I
SUMMARY OF FUNCTIONAL UNIT PROPERTIES. THERE ARE SIX TYPES OF FUNCTION UNITS: THE ARITHMETIC AND LOGIC UNIT (ALU), THE ACCUMULATE AND BRANCH UNIT (ABU), THE LOAD-STORE UNIT (LSU), THE MULTIPLIER UNIT (MUL), THE REGISTER FILE (RF), AND THE IMMEDIATE UNIT (IU)

FU type	Description	Supported operations	Inputs/outputs
ALU	Provides typical RISC operations. One of the ALU outputs can be used unbuffered, allowing operation chaining.	Arithmetic, logic, comparison, shifting	4 / 2
ABU	Can be configured in two modes, based on the bitstream. Accumulate mode uses the ABU as a 16-register accumulator. Branch mode generates program counter values.	(Conditional) branching, accumulation	4 / 2
LSU	Performs memory operations to global data memory and local memory, including configurable address generation.	Global memory load and store, local memory load and store	4 / 2
MUL	Performs multiply operation which can be combined with arithmetic shift-right operations for efficient computation of fixed point results.	Multiplication, shifting	4 / 2
RF	Provides 16-entry register file support to Blocks. RF operations in Blocks are explicit .	Reading/writing registers	4 / 2
IU	Generate constant values onto the data network	Immediate generation	0 / 1

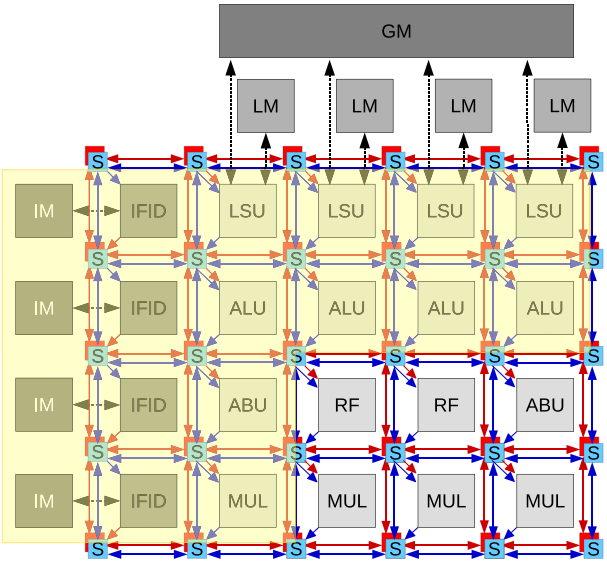

Fig. 2. Example blocks instance showing a VLIW-SIMD processor implemented on the fabric. The networks, data, and control are shown in blue (front) and red (back), respectively.

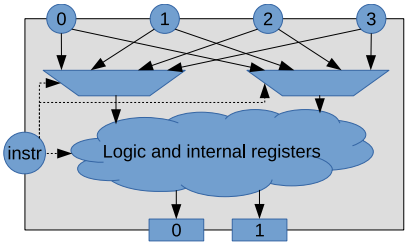

Fig. 3. Generalization of the functional unit structure with four data inputs and two data outputs, the control network connection is marked with "instr."

C. Memory Hierarchy

Blocks contains a large global 32-bit wide global data memory (GM), all load-store units (LSUs) are connected to this memory via a global memory arbiter. The arbiter serves memory requests on a work-preserving round robin basis and detects coalesced memory accesses. For read operations, this means that if requests are made that access the same memory row, these requests are coalesced and read at once. Similarly, if write operations to the same memory row do not conflict, the requests are grouped and written at once. Conflicting accesses are performed sequentially. When not all memory requests can be performed within a single access the processor is stalled until all requests are completed. Besides the GM, every LSU has a local memory (LM), due to the small size of these memories they are implemented as a low-power RF macro from a commercial vendor which internally consists of flip-flops. The local memories are private to an LSU and therefore never cause processor stalls. When data reuse is possible inside a kernel, the local memories will be used to store intermediate results if they cannot be kept inside the processing pipeline.

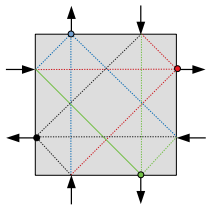

Fig. 4. Generalization of a switch-box, as used in both the data and the control network. The outputs each have a multiplexer which selects from the available inputs (indicated with a dot). The solid green line represents a selected connection.

D. Interconnect Network

Since the goal of Blocks is to reduce energy, a statically configured data network is used [31]. The network configuration is part of the bit-stream that is used to configure the Blocks platform at runtime. Typically, the network is configured just before the start of an application, or a loop-nest within an application. This configuration contains specifications like: connect the left port of the switch-box to the bottom port of the switch-box, as shown in Fig. 4 with a solid green line. This figure shows a generalization of a switch-box as currently used in Blocks. The dots on the output ports are multiplexers which are configured to select one of the available inputs.

The number of connections (channels) on each side of the switch-box is a design parameter. It is also possible for switch-boxes to have no inputs or outputs on one or more sides, this is generally the case for switch-boxes on the outside of the Blocks fabric. Switch-boxes that are close to function units can

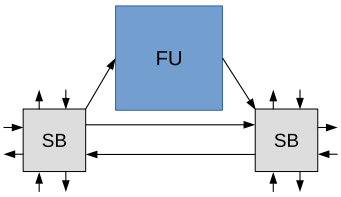

Fig. 5. Connection of function units to the Blocks reconfigurable network. The outputs of function units can be selected in the switch-box configuration, just like any other input on the switch-box. Inputs to a function unit are considered an output on a switch-box, allowing it to select any of the inputs as a source.

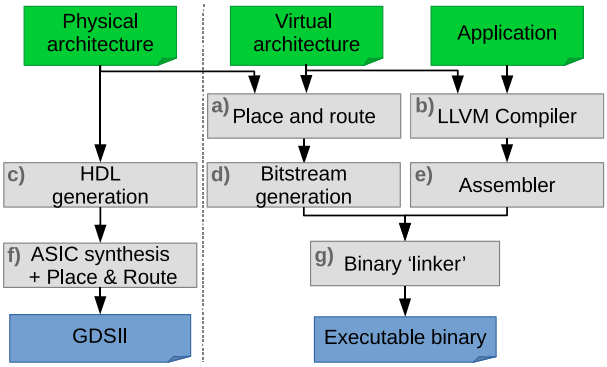

Fig. 6. Blocks tool flow.

have inputs or outputs to these function units. In the Blocks framework, this is typically represented by a diagonal connection, as shown in Fig. 5. The left switch-box (gray) serves as an input to the function unit. To do so, it has an extra output. The right switch-box serves as a sink for the outputs of the function unit.

E. Instruction Set Architecture

The operations that the Blocks FUs support are quite similar to RISC operations. The main difference is that Blocks has its own set of operations for each type of functional unit, allowing opcodes to be reused. A second difference is that unlike most RISC architectures the arguments of the operation do not specify register numbers, instead an input source number and a destination output port are specified. This is because the data available to each FU is dictated by the configuration of the data network; there is no knowledge of these connections in the operation itself. This helps to keep the instruction width very small [12-bit for all units except the Immediate Unit (IU)] and allows construction of VLIW-like processors without much instruction fetch overhead. A typical Blocks instruction has the following form:

```
mnemonic destination, inputA, inputB
```

where `destination` is one of the output registers and `inputA` and `inputB` are inputs selected by multiplexers. These function units can have more inputs than operands that are required for the operation. Throughout this article, most function units are configured to have four inputs (sources from the network).

The Blocks architecture is, as long as at least one ABU and one ALU are present in the virtual architecture, Turing complete. The ABU allows branches to take place, and the ALU allows for comparisons with data to control whether the branches are taken or fall-through.

F. Tool-Flow

Fig. 6 shows the Blocks tool-flow. A physical architecture description details the FUs present in the design and control and data network properties. It is used to generate Verilog that will be synthesized (c), placed-and-routed (f), and eventually exported as GDSII. This tool-flow path has to be executed only once for a specific Blocks instance. Only when the designs resources change, this path has to be rerun.

A virtual architecture describes the architecture that will be instantiated onto the physical architecture and contains the connections between FUs and between IFIDs and FUs. The virtual architecture is used to perform automated placement and routing on the Blocks fabric (a). Placement is performed using simulated annealing and routing is performed using the pathfinder algorithm. The placed-and-routed design is translated into a bit-file that is used to configure the Blocks fabric (d). An LLVM compiler back-end (b) generates functional code, but for the benchmarks in this work, the assembly is hand optimized. Finally, the assembled program code and bitstream are merged into a binary format (g) supported by the Blocks hardware bootstrapper.

IV. SIMD and VLIW Support in Blocks

With its flexibility Blocks allows various architecture types to be specified. These structures can be very similar to existing processors, such as VLIWs and SIMDs. However, they can also be completely adapted to the application and provide spatial layout for (part of) the application.

A. Constructing VLIW Processors

VLIW processors execute instructions which specify multiple operations. These operations control multiple issue-slots at the same clock cycle.

Instead of using a single instruction fetch and decode unit that loads a very wide instruction, Blocks uses multiple instruction fetch and decode units that are connected to the same PC. To do so, each IFID has a connection to the data network to connect to the output of an ABU, which produces a PC value. Fig. 7 shows an example of such a structure for two IFIDs. By doing so, it comes under the control of the sequencing provided by the ABU. As the PC is the same for every connected instruction decoder, all decoders will run in the lock step. Each instruction decoder is connected to its own instruction memory.

The advantage of the Blocks method is that the width of the VLIW processor (i.e., the number of issue-slots) can be easily matched to the requirements of the applications. The disadvantage of the flexible processor construction that Blocks uses is that some VLIW optimizations cannot easily be applied. For example, VLIW processors can reduce some of their power by applying instruction compression. Blocks stores instructions for each instruction decoder in separate

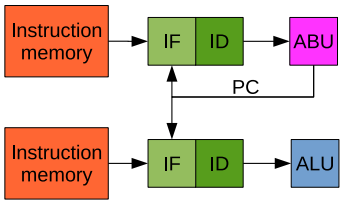

Fig. 7. Construction of a 2-issue VLIW processor using the Blocks fabric. The PC is distributed to multiple instruction decoders to make these operate in lock-step and form a VLIW processor.

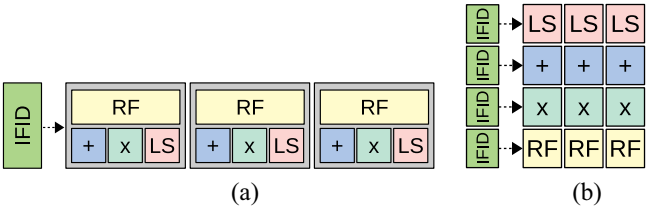

Fig. 8. SIMD processors, (a) typical approach and (b) Blocks approach.

instruction memories. Furthermore, which instruction decoders are going to be used in a specific configuration is unknown at design time. Therefore, it is not possible to compress over multiple instruction decoders. Predefined clusters of instruction decoders could be a solution for this, at the expense of some flexibility.

Since the flexibility of Blocks allows the architecture to adapt to the application it is often possible to apply software pipelining. The idea behind software pipelining is to overlap different iterations of loop bodies in time; i.e., the next iteration starts while the current iteration still has to finish. This shortens the effective iteration time. In the extreme case, the iteration interval becomes a single cycle. Blocks can, as long as there are sufficient resources, efficiently realize this by instantiating a VLIW processor with the right number of issue slot and bypass connections to allow spatial mapping of a kernel.

B. Constructing SIMD Processors

The SIMD processors constructed on the Blocks fabric differ from typical SIMD processors [shown in Fig. 8(a)] where only one of the function units (other than the RF) is active at one time. An example architecture instance for Blocks is given in Fig. 8(b). Since the function units are specialized and controlled by their own instruction decoder the Blocks approach results in a VLIW-SIMD processor. This means that the vector lanes for each function unit type effectively form an issue-slot of a VLIW processor. Although the total number of instruction bits is wider than that in the typical SIMD processor (in this example 48 bit for Blocks versus typically 32 bit in the SIMD), the instruction for the example application can be software pipelined with an iteration interval of a single cycle. This reduces power in the instruction decoders and their attached instruction memories due to reduced toggling of the address and data lines of the instruction memories as well as the control signals for the function units. Additionally, the

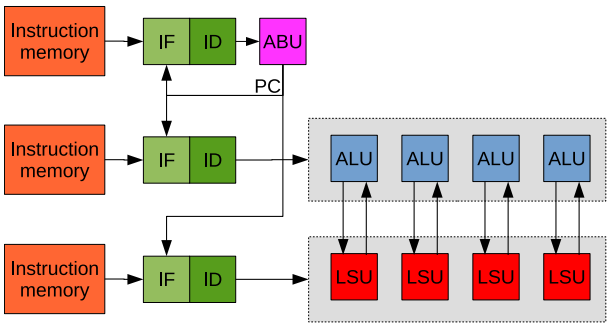

Fig. 9. Construction of an SIMD processor using the Blocks fabric. The ALU and LSU function units are controlled as a vector lane of four elements wide.

Blocks approach will provide a higher throughput since the operations can be pipelined.

A more detailed example of supporting SIMD lanes by Blocks is shown in Fig. 9. SIMD significantly reduces the instruction overhead compared to the local or global decoding used by traditional CGRAs. The construction of vector lanes is very similar to constructing a VLIW issue-slot as described in Section IV-A. An ABU is used to control the sequencing of an instruction decoder. This instruction decoder then loads and decodes the corresponding instructions. The decoded instructions are made available on the control network and routed to multiple function units.

C. Application-Specific Data Paths

Besides the typical processor structures, it is possible to create very application-specific designs. Doing so can significantly improve the performance or energy efficiency of the application as this allows Blocks to create a (near) spatial layout of applications. In some cases, part, or whole, of these special structures can be controlled with a single instruction decoder. For example, when multiple values have to be added together an adder tree can be constructed.

V. Experimental Setup

All architectures are placed-and-routed using commercial ASIC design tools and a 40-nm technology library. Power and energy results are based on real toggling rates on the full layout. Four reference architectures are used: 1) a CGRA; 2) an 8-lane SIMD; 3) an 8-issue VLIW; and 4) an ARM Cortex-M0 microprocessor. Additionally, application-specific processors (ASPs) are instantiated per benchmark application. The SIMD, VLIW, and ASP architectures are all easily realizable with the Blocks framework but use fixed connections instead of a reconfigurable fabric.

All architectures contain a two-ported (one read and one write port) GM of 32 kB, sufficient to hold the initial and resulting data for all benchmarks. The ARM-M0 uses a single ported memory due to AHB-bus limitations. All LSUs, with exception of the ARM Cortex-M0, contain a 1 kB two-ported LM. This memory is private to an LSU and, therefore, does not need arbitration. Its size is chosen such that it allows some

TABLE II
SUMMARY OF ARCHITECTURE PROPERTIES. * EIGHT OF THESE ALUS ARE ONLY USED AS 2-REGISTER DATA BUFFER

Architecture	GM [KB]	LM [KB]	IM [KB]	#ALU	#MUL	#LSU	#RF	#IM	Compute area [mm^2]
VLIW	32	4	14.8	8	8	4	8	4	0.18
SIMD	32	9	4.4	9	9	9	9	1	0.23
ARM M0	32	0	32	1	0	1	1	1	0.02
Traditional CGRA	32	9	15.5	17*	9	9	1	2	0.26
Blocks	32	9	5	17*	9	9	1	2	0.26

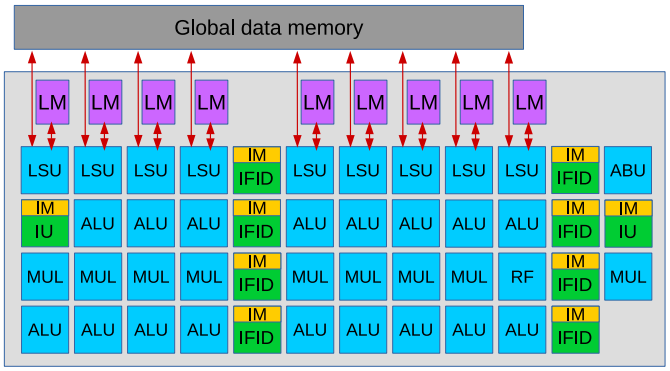

Fig. 10. Blocks architecture instance used for the benchmark kernels. For clarity, the networks are omitted.

local data reuse without allowing the whole input dataset to be available locally.

The overall goal is to ensure fair comparisons, as much as possible. For this reason, the compute area (excluding area for memories and switch-boxes) is designed to be as similar as possible between Blocks, the traditional CGRA, VLIW, and SIMD. Any remaining area differences are corrected for by using the performance per area metric.

A. Blocks Architecture Instance

The instantiated Blocks architecture is reused for all kernels and, therefore, contains the superset (union) of FUs in the ASPs. It includes 9 LSUs, 17 ALUs, 9 multipliers, 2 IUs, 1 RF, and 1 ABU, as shown in Fig. 10. When an FU is not used, the inputs are tied to a fixed value by configuration to reduce power. These compute resources are very similar to the 8-lane reference SIMD with control processor but the RFs are replaced by ALUs which are used as two-element register files in the "FIR," "IIR," and "2-D convolution" benchmarks and unused in other kernels. These benchmarks do not require many registers due to spatial layout and ALUs are cheaper in area and power. Blocks also contains eight IFIDs to control groups of these functional units. The two networks (control and data) have different widths and number of connections per switch-box. The data network in the Blocks instances in this work is 32-bit wide (but can be configured at design time to be 8, 16, 32, or 64-bit wide). The control network has a fixed width of 16 bit and transports the decoded instructions. The interconnect network is set up for three vertical and two horizontal connections (on each side). This number of connections was determined using our automated routing tool. First, only one horizontal and one vertical channel were allotted to each of the switch-boxes, which causes routing to fail due to insufficient resources. For the next iteration, the number of horizontal channels was incremented by one, followed by the number of vertical channels. This procedure was repeated until routing became feasible for all benchmark applications. Although, this does not necessarily result in the minimum required number of channels it should be quite close. In this work, the interconnect pattern is a full mesh. However, the tool flow is designed to allow other connection schemes as well. Currently, only full-mesh is implemented.

B. Reference Architectures

A reference CGRA is used to compare the traditional way of controlling a CGRA with the Blocks method. The available functional units, their capabilities, and the data network are identical to Blocks to ensure a fair comparison. Blocks-based ASPs with fixed connections and FUs are used to evaluate the overhead of Blocks. The VLIW and SIMD processors are used to compare the performance and energy efficiency of Blocks with processor types typically used for DSP applications. The goal is to show that Blocks can achieve similar performance per area and energy efficiency as the best suited general-purpose processor for each benchmark kernel. The ARM-Cortex-M0 is used to compare the performance per area and energy with a general-purpose low-power microprocessor. Blocks is expected to show better energy efficiency by adapting to the application, despite the very low power of the microprocessor. Table II summarizes the architecture properties for all architectures.

1) ARM Cortex-M0: This architecture was chosen for its popularity in commercial applications and reputation as a very low-power architecture. It uses a Cortex-M0 logic core combined with an instruction and data memory. The largest binary for this architecture is approximately 12 kB but since the same memory is also used as RAM 32-kB memory is required. The instruction memory is single ported.

2) 8-Lane SIMD With Control Processor: The SIMD reference processor is based on the architecture presented in [32]. It contains a control processor and 8-lanes with bypass support between ALU, Multiplier Unit (MUL), and LSU. The control processor controls branches and can broadcast values to the lanes in the vector processor, as shown in Fig. 11. All lanes contain a register file, ALU, multiplier, and LSU. The lanes and control processor have a neighborhood network, allowing results to be communicated between lanes. The instruction memory of the SIMD holds 256 instructions, the next power of 2 needed to fit the largest kernel. Only one of the FUs inside each lane (ALU, MUL, LSU) can be active at a clock cycle.

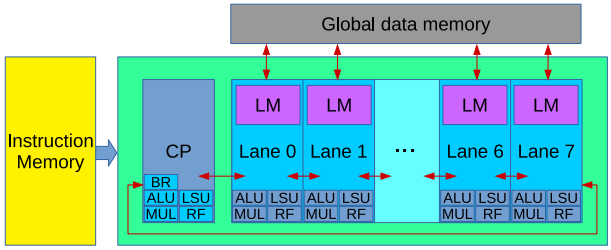

Fig. 11. 8-lane SIMD reference architecture.

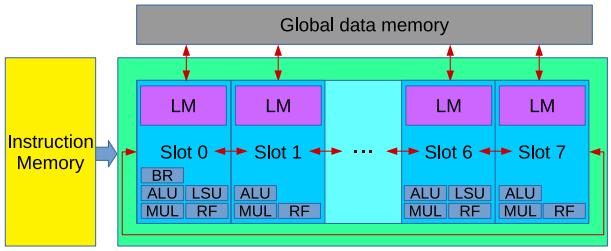

Fig. 12. 8-issue slot VLIW reference architecture.

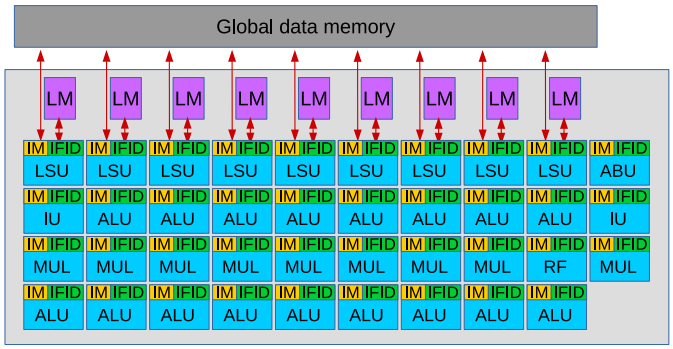

Fig. 13. Traditional CGRA. The instruction memories are marked with "IM." For clarity, the data network is omitted.

3) 8-Issue Slot VLIW: The VLIW reference architecture, shown in Fig. 12, has eight issue slots with private register files, neighborhood communication, and bypass support. If a VLIW with a shared RF would have been used the power results would have been worse due to the many required register ports. One issue slot supports branching. The instruction memory holds 256 instructions, although these instructions are significantly wider than for the SIMD. Half of the issue slots contain an LSU, which is a higher than average percentage for a VLIW. Memory operations can be spread out over time, which prevents memory congestion and therefore stall cycles.

4) Traditional CGRA: The traditional CGRA is a special version of Blocks without separation in the control and data paths, it is referenced throughout this work as "traditional." It contains the same specialized FUs as Blocks, but with local instruction memories for *each* unit, as shown in Fig. 13. Since each functional unit performs local instruction decoding, the control network is no longer required and is removed. This CGRA is used as a reference since comparisons with the existing CGRAs are often inaccurate. Either no power/energy numbers are given or are for a very different technology. Furthermore, what is included in the energy numbers is often unclear (e.g., if it includes memory accesses). In our opinion, using a Blocks-based reference, CGRA provides the fairest comparison to show the benefits of separating the control and datapath.

5) Application-Specific Blocks Instances (Blocks-ASP): The application-specific Blocks instances contain fixed wiring between FUs instead of the reconfigurable data and control network. For each application, the unused FUs are removed. This results in a dedicated processor, which contains only the hardware (connections, functional units, and memories) that is required for executing a specific kernel, called Blocks-ASP.

C. Synthesis and Place-and-Route

Synthesis and place-and-route of the reference architectures is performed using the Cadence ASIC tools on a commercial 40-nm low-power library. To obtain accurate results, all memories are implemented using commercial 40-nm memory macros that can be taped out. The resulting netlist also includes the clock tree and I/O buffers. In this work, all designs are synthesized for 100 MHz in order to be able to make a fair comparison. At this frequency, there are not many effects from the synthesis tools selecting larger gates to reach the desired clock frequency. In practice, Blocks has been successfully placed-and-routed at 300 MHz (worst case corner at 125 °C and 0.99 V). The reference architectures can be synthesized up to 300, 470, 300, and 450 MHz for the traditional CGRA, VLIW, SIMD, and ARM Cortex-M0, respectively. For Blocks, VLIW, and SIMD, the memory arbitration unit eventually becomes the critical path of the design. The maximum frequency is higher for the VLIW as it only contains four LSUs while the SIMD contains nine. Each switch-box introduces a delay of 0.2 ns, and the worst case functional unit, the multiplier, introduces a delay of approximately 2 ns. This means that even when the multiplier is used, it is still possible to route the signal over six switch-boxes before the maximum frequency has to be reduced. A higher achievable frequency allows voltage scaling and further energy reduction. However, Section VI shows that Blocks obtains a significantly higher performance at the same frequency compared to the reference architectures. Therefore, Blocks can achieve similar performance while still scale to lower voltages.

The performance and power results are obtained by simulating the placed-and-routed net list at the typical corners, including delay annotation and capacitances. This has the advantage that functionality can be verified after the whole design flow, as well as obtaining an accurate activity file (TCF). The net list combined with the activity file is used by the Cadence flow to perform power analysis. Since this is performed on a post-place-and-route design, wire and cell capacitances are included. The same technology library and synthesis settings are used for all architectures and all benchmarks.

D. Benchmarks

Eight benchmarks kernels are carefully selected to ensure a realistic comparison between the traditional CGRA and Blocks. Additionally, the benchmark set is selected such that

TABLE III
OVERVIEW OF BENCHMARK KERNELS

Benchmark	Description	Type	Data size
Binarization	Thresholding on greyscale image	scalar to scalar	128*64 pixels (8 bit)
Erosion	Noise removal by AND-ing neighbouring pixels	3*3 window to scalar	128*64 pixels (8 bit)
Projection	Sums each horizontal and vertical row/column	vector to scalar	128*64 pixels (8 bit)
FIR	8-tap low-pass FIR filter on input signal	8*1 convolution	2200 samples (32-bit)
IIR	3rd order low-pass filter on input signal	2*1 + 2*1 convolution	2200 samples (32-bit)
FFT	8-point complex FFT on audio signal	8*1 to 8*1 vector (complex)	2200 samples (32-bit)
2D convolution	Gaussian blur on image	3*3 window to scalar	128*64 pixels (8 bit)
FFoS	Industrial vision application	image to 2 8*1 vectors	128*64 pixels (8 bit)

some applications perform well on SIMD processors and others on VLIW processors. The benchmark kernels include typical operations that can be found in many signal processing applications. For example, "Binarization" includes thresholding, "Erosion," and "Convolution" include "2-D convolution", FIR includes 1-D convolution, "FFT" includes butterfly operations on complex numbers, and IIR includes more irregular patterns. Table III gives an overview of the benchmarks used for evaluation. All kernels are written in assembly (except for the ARM Cortex-M0) and optimized per architecture. The benchmarks for the M0 are written in C and compiled with the "-O3" flag using GCC, version 4.9.3. For all benchmarks and architectures, the initial data are assumed to be in the GM, the local memories are considered to be uninitialized.

The benchmark kernels Binarization, Erosion, "Projection," and "FFoS" are expected to perform well on an SIMD processor since they can be relatively easily vectorized, even though the Erosion benchmark does require some neighborhood communication. Applications that require more irregular communication (not strictly neighbor to neighbor) such as FFT and IIR are expected to perform better on the VLIW.

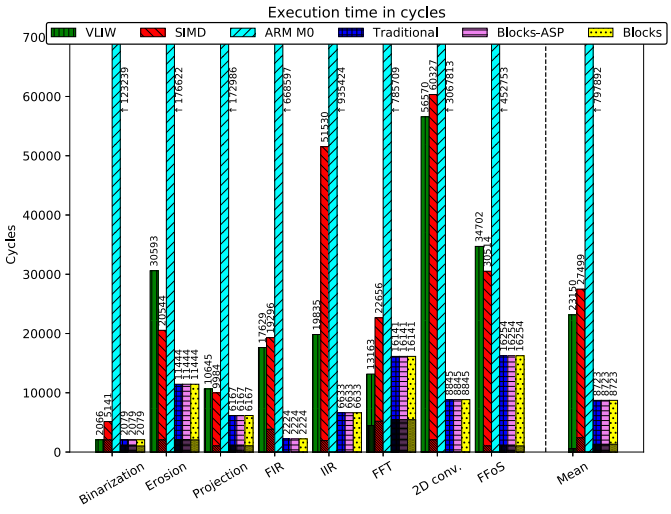

Fig. 14. Number of cycles for the benchmark kernels. Shaded are stalled cycles due to global memory accesses.

VI. EVALUATION

This section describes the performance, power, energy, and area results for Blocks and the reference architectures. The evaluation is split into two parts. First, a comparison is made between Blocks, the reference CGRA, and the dedicated architectures to evaluate overhead. Second, Blocks is compared with the nonreconfigurable reference architectures.

A. Reconfiguration Overhead of Blocks

This section is a summary of the extensive overhead evaluation presented in [11]. The goal of Blocks is to reduce reconfiguration overhead in reconfigurable architectures. To evaluate the effectiveness of Blocks, ASPs for each benchmark are compared with Blocks and the traditional CGRA in terms of energy and area. Performance is the same for these architectures since the underlying virtual architecture and the scheduled instructions on it are the same, see Fig. 14. The more densely shaded areas in the bottom of each bar represent the cycles stalled due to memory arbitration. In Blocks, data-level-parallelism results in one instruction decoder controlling multiple FUs whereas in the traditional CGRA, the instructions are replicated over functional units.

1) Energy: Fig. 15 shows that the energy of Blocks is lower than the traditional CGRA for all benchmark applications. Since the virtual architectures mapped on the reconfigurable Blocks are the same as the Blocks-ASP architectures, the reconfiguration overhead of the traditional CGRA and Blocks can be analyzed. The overhead consists of both the reconfigurable networks and the unused FUs in the reconfigurable version of Blocks. The energy overhead reduction of Blocks compared to the traditional CGRA is between 1.46× and 1.76×. The system-level energy reduction of Blocks compared to the traditional CGRA is between 9% and 29% (average 22%).

2) Area: The architectures for Blocks and the traditional CGRA are kept the same for all benchmarks. Only the Blocks-ASP instances have different areas as they are benchmark specific. The area of Blocks is 19.1% smaller than the traditional CGRA. The area occupied by switch-boxes in Blocks is 49.2%, see Fig. 16. The traditional CGRA has 46% of area dedicated to switch-boxes but 14.4% to instruction memories, while in Blocks this is only 5.2%. Again, the overhead reduction that can be achieved by SIMD support, and therefore a reduction in instruction memories, is what is largely responsible for this difference. The Blocks-ASP instances do not include reconfigurable networks and are therefore much smaller than Blocks and the traditional CGRA.

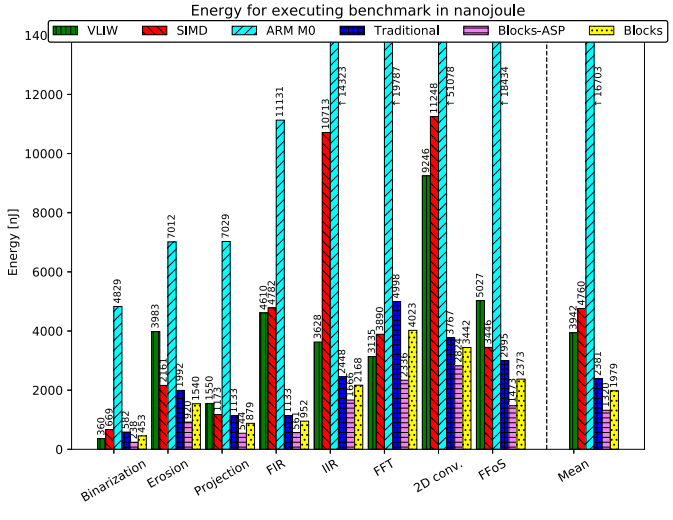

Fig. 15. Energy per architecture per benchmark.

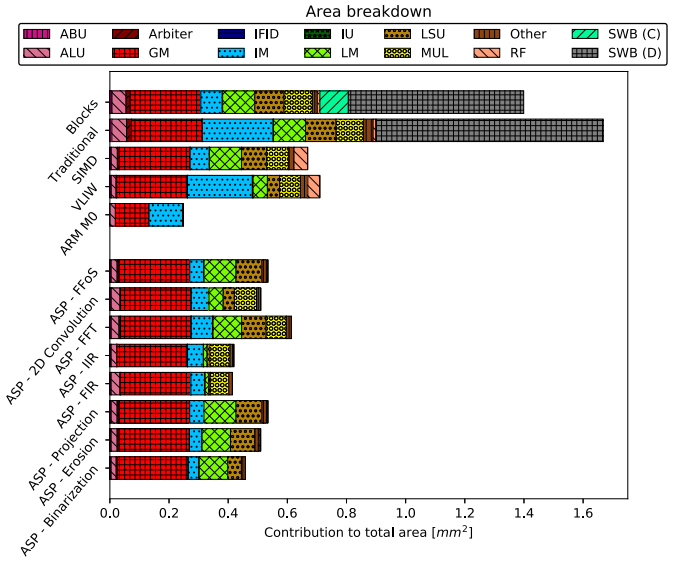

Fig. 16. Area breakdown for all architectures. The lower eight are Blocks-ASP instances for their respective benchmarks. The categories marked SWB(C) and SWB(D) indicate the switchboxes in the control network and data network, respectively.

B. Comparison With Fixed SIMD and VLIW Architectures

The real benefit of a reconfigurable architecture is that it can adapt itself to the type(s) of parallelism available in the application it is going to execute. Fixed architectures, such as VLIW and SIMD processors do not have any reconfiguration overhead. On the other hand, their supported parallelism mix is determined at design time. Blocks would show its value when it can perform similar, performance and energy wise, compared to a VLIW and SIMD over a range of applications. This section will show that Blocks is able to do so.

1) Performance: Although each benchmark kernel has been optimized for each architecture, Blocks achieves similar or better performance for almost all benchmark kernels. For Erosion, Projection, and FFoS, Blocks is instantiated with bypass channels between the processing elements. Compared to the SIMD and VLIW, where the neighborhood network is fixed, this saves cycles needed to move data between lanes. The benchmarks FIR, IIR, and 2-D convolution are implemented spatially. This reduces the number of instructions in the loop-nest significantly, for FIR and 2-D convolution, the loop-nest becomes a single cycle. For Binarization and FFT, Blocks requires more cycles than the VLIW. In Binarization, this is caused by LSU read and write address initialization (13 cycles). For FFT, Blocks uses resources to construct two spatial butterfly units. The VLIW and SIMD process one butterfly per lane since further-than-neighbor communication is expensive. For FFT, Blocks uses only a single register file, since more RFs are not required by the other kernels. This causes a small performance penalty for FFT on Blocks, but does not outweigh the energy and area penalty for the other kernels. Overall, adapting the data path to the application Blocks provides an improvement of a factor 2.7× and 3.2× over the SIMD and VLIW, respectively.

Kernels containing data-level parallelism perform well on the SIMD. An exception to this is Binarization because the SIMD always loads data with eight LSUs, causing memory stalls due to memory bus width. The VLIW reduces the loop-nest of this kernel to a single cycle by performing a load, store, and arithmetic operation in parallel. The VLIW performs worse on kernels where the local memories are used extensively, such as erosion and FFoS, since only half of the issue-slots contain LSUs. Despite the SIMD incurs stall cycles for accessing 8 bytes in parallel over an 4-byte memory bus, the VLIW needs to move loaded data to neighboring issue slots before the next memory access can be made. The SIMD performs worse on kernels with further-than-neighbor communication. IIR, for example, requires intermediate results from other lanes causing communication over the neighborhood network. To prevent this, the IIR filter is unrolled which leads to recomputation, but performs better on the SIMD. The mean performance difference between the SIMD and VLIW is small, showing that the reference architectures and benchmark applications are representative.

The ARM Cortex-M0 takes significantly more cycles, there are several reasons to this: it cannot perform parallel computations, control flow instructions are interleaved with computation and there is no hardware multiplier. However, the ARM Cortex-M0 is mainly included to compare energy numbers. To predict how much difference a more powerful microprocessor makes, the cycle accurate simulator OVPsim is used to predict speed-up using an ARM Cortex-M4 and Cortex-A9. The highest speed-up is 2.47× and 2.59× for FFT on the Cortex-M4 and Cortex-A9, respectively. Similar speed-up is achieved for FIR, IIR, and "2-D convolution," all containing multiplications. There is almost no speed-up for the other kernels. However, core power for the Cortex-M4 increases by 2.3× according to [33], [34], canceling out any energy gains.

2) Power and Energy: Blocks has a higher power draw due to reconfigurability overheads, a geometric mean power increase of 1.32× and 1.51× compared to SIMD and VLIW, respectively. However, the performance increase due to architectural flexibility of Blocks outweighs this power increase in terms of energy. Blocks shows an energy reduction of

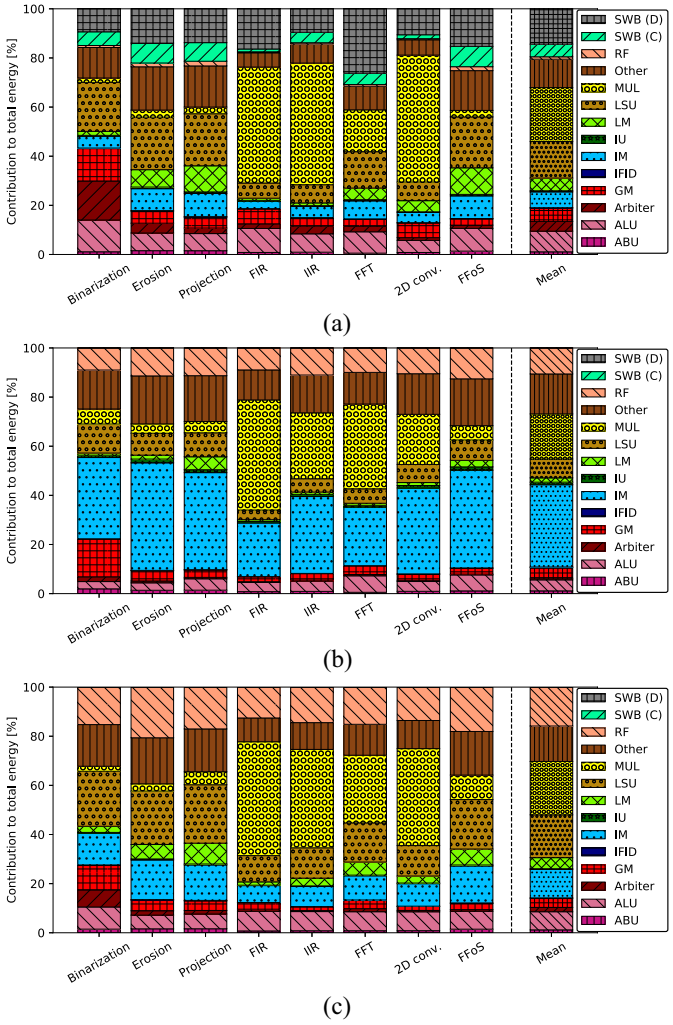

Fig. 17. Power breakdown for SIMD and VLIW architectures. The categories marked SWB(C) and SWB(D) indicate the switchboxes in the control network and data network, respectively. (a) Energy breakdown for Blocks. (b) Energy breakdown for VLIW. (c) Energy breakdown for SIMD.

2.05× and 1.84× when compared to the SIMD and VLIW, see Fig. 15. The cause of this can be seen when Fig. 17(c) and (b) are compared with Fig. 17(a). Blocks shows a much better utilization of the multipliers and ALUs, leading to the energy reduction. For Binarization, Blocks performs slightly worse in energy than the VLIW because there is virtually no possibility for data reuse in this kernel. For FFT, the number of cycles cannot be significantly reduced due to the limited number of register files. This leads to a somewhat higher energy for the FFT benchmark, although roughly on par with the SIMD architecture. Power draw for Blocks-ASP is on par with the SIMD and VLIW architectures while outperforming these architectures in execution time by approximately 3×, leading to a significant energy reduction. This shows the gains that are possible when adapting architecture to the algorithm. The ARM Cortex-M0 has a very low-power footprint. Interesting for this architecture is to observe that kernels containing many multiplications result in a lower power draw, this is caused by the large number of cycles required to perform multiplications with respect to the low number of memory accesses.

3) Area: The area of Blocks without switch-box area in Fig. 16 is very similar compared to the area of the VLIW and SIMD. This shows that the reference architectures are well balanced with respect to the Blocks architecture, and that the extra area is caused by the Blocks reconfigurable fabric. The area distribution of Blocks is very similar to the SIMD. As expected, the VLIW uses more area for instruction memory while the SIMD has more area in the FUs. The ARM Cortex-M0 processor is small in comparison with its memories, as can be observed in Fig. 16, even though the ARM Cortex-M0 memories are single-ported as opposed to the two-ported memories in the other architectures, reducing their relative size.

4) Flexibility: Some methods to classify flexibility have been proposed, these either base their metric on the ability of a design unit to connect to any other design unit [35], or its insensitivity to performance differences [36].

According to the metric described in [35], Blocks would always be able to connect every design unit to every other design unit, thus making it the most flexible architecture available for the four architectures evaluated in this section.

The second metric defines "versatility" as the geometric mean of all speed-ups relative to the best performing processor for each individual benchmark kernel. As the "versatility" metric does not take performance differences into account, the ARM Cortex-M0 gets a very poor versatility rating even though it is, instinctively, very flexible (Section IV). The same holds, albeit to a lesser extent, for the SIMD and VLIW processors.

Since Blocks aims at energy efficiency it seems apt to classify flexibility based on the energy spent to perform the work required to compute a benchmark kernel. The larger the spread in the amount of energy spent between benchmark kernels, the lower the flexibility. An ASIC, for example, will spend very little energy to perform a kernel that it is designed for but may take lots of energy to perform one that does not match its architecture (if it is capable to perform it at all). This means that the variation will be high. For a very flexible architecture, such as a microprocessor, the variation would be much smaller as it is able to perform all applications with reasonable efficiency. However, the magnitude of the spread depends on the absolute differences between benchmark kernels. For this reason, the energy numbers should be normalized per architecture. This method is not perfect either as it will consider architectures that have a relatively high energy overhead compared to their active energy to be more flexible. For each architecture, the standard deviation of the normalized energy numbers is taken, as shown in (1), subsequently, the results for all architectures are scaled such that the maximum is at one [(2) and (3)]. This means that this flexibility metric is relative between the evaluated architectures; it cannot be used to compare with architectures outside of the evaluation set without recomputing (2) and (3). In these equations, $E_{\text{architecture}}$ denotes the energy metrics for all benchmarks for a specific architecture. The results of this metric are shown in Table IV as "energy flexibility." It can be observed that both the ARM Cortex-M0 now scores much better in terms of flexibility compared to the "versatility" metric. The energy

TABLE IV
VERSATILITY OF THE REFERENCE ARCHITECTURES COMPARED TO BLOCKS

Metric	ARM M0	SIMD	VLIW	Blocks
'Versatility'	0.02	0.31	0.40	0.97
'Energy flexibility'	0.82	1.0	0.79	0.92

TABLE V
IMPROVEMENT OF BLOCKS COMPARED TO REFERENCE ARCHITECTURES

Metric	VLIW	SIMD	Traditional CGRA	ARM M0
Performance	2.4x	3.1x	1.0x	68.9x
Power	0.76x	0.66x	1.22x	0.12x
Energy	1.84x	2.05x	1.22x	8.01x
Area	0.51x	0.48x	1.19x	0.18x
EDAP	2.25x	3.05x	1.45x	93.82x

efficiency for Blocks is also higher than for the ARM and VLIW, as could be expected. However, the SIMD processor scores best in this metric. This is due to the relatively large static energy consumption, which "dampens out" the variations between benchmarks. However, since Blocks can be used to implement processors that are similar to the SIMD, this intuitively suggests a more flexible architecture. We can conclude that although this metric is not perfect, it is less sensitive to speed-up relative to the fastest performing processor in the set. In both metrics, however, Blocks scores well with respect to flexibility

$$F_{\text{architecture}} = \sigma\left(\frac{E_{\text{architecture}} - \min(E_{\text{architecture}})}{\max(E_{\text{architecture}}) - \min(E_{\text{architecture}})}\right) \quad (1)$$

$$F_{\text{architectures}} = \left[F^1_{\text{architecture}} F^2_{\text{architecture}} \cdots F^N_{\text{architecture}}\right] \quad (2)$$

$$\text{Flexibility} = \frac{F_{\text{architectures}}}{(\max(F_{\text{architectures}})}. \quad (3)$$

C. Summary of Results

Blocks performs well on performance and energy by providing speed-ups over fixed architectures, as shown in Table V. Like all reconfigurable architectures Blocks trades flexibility for area and consequently has a larger area than fixed architectures. The Energy-Delay-Area-Product (EDAP) of Blocks is better than all nonapplication-specific reference architectures.

When performance per area is considered Blocks performs similarly or better as the VLIW, SIMD for most benchmarks, and always better than the traditional CGRA and the ARM Cortex-M0, as shown in Fig. 18. The dedicated architectures have a better performance per area since performance is identical to Blocks but all overhead is removed. The benchmarks where the VLIW or SIMD outperform Blocks are Binarization and FFT where Blocks has similar execution time.

The Blocks instance shown in Fig. 10 can achieve 26 operations per clock cycle if only the multipliers and ALUs are taken into account. When the load-store operations performed by the LSUs are also taken into account this results in a total of 44 operations per cycle (the LSUs can perform a load and store simultaneously). The energy consumption for Blocks in this case is approximately 299 pJ per cycle, including switch-boxes, arbiter, register operations, etc. Furthermore, it is assumed that all function units (including function units

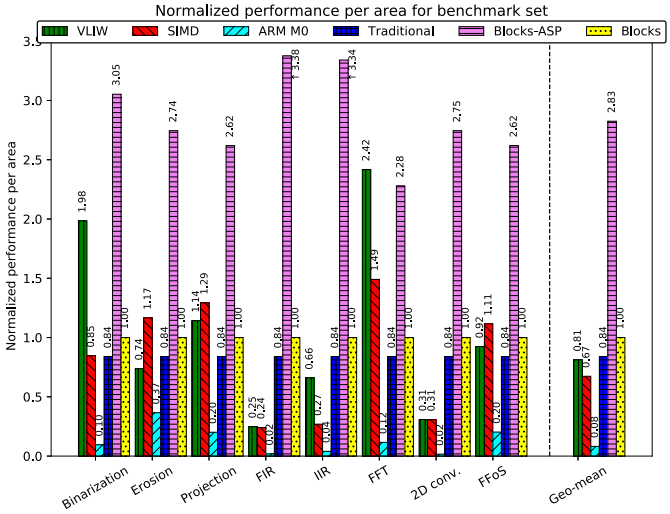

Fig. 18. Normalized performance per area, all results are normalized to the performance per area of Blocks.

that do not count as useful operations, such as RF, IU, and ABU) are fully occupied and performing 32-bit operations. Memory energy is not included in this number. This results in a performance of 11.5 pJ per operation when only multiplications and additions are considered, and 6.8 pJ per operation when the LSU operations are also taken into account. When compared with peak energy efficiency for various architectures in the literature this is quite acceptable. For example, a TTA-like VLIW architecture, optimized for machine learning, achieves 5.3 pJ/operation (28 nm FDSOI at 0.35 V) [37] while Dally *et al.* [38] stated that a typical VLIW achieves an energy efficiency of around 10 pJ/operation. The symbiote [39] VLIW coprocessor achieves 28 pJ/operation. Similarly, a highly optimized SIMD for computer vision achieves 1.9 pJ/operation (180-nm CMOS) [40] while another, more generic, SIMD achieves 4.8 pJ/operation (40-nm CMOS) [41]. Systolic CGRA architectures generally perform somewhat better but are less flexible than Blocks. Examples of such architectures are PHVArray [42] (20 pJ/op, 550 nm), REMUS_HPP [44] (5.5 pJ/op, 65 nm), and REMUS_LPP [44] (1.3 pJ/op, 65 nm). The latter two are video decoding optimized systolic arrays with many function units (256), which helps to amortize overhead. These energy efficiencies are estimates since certain parameters are assumptions, for example: 100% utilization of FUs. The individual benchmarks results show that Blocks can get close to the peak energy efficiency while maintaining flexibility.

VII. CONCLUSION

The results presented in this work showed that Blocks, with the separation of data path and control path, significantly reduces reconfiguration overhead even when compared to an already optimized but more traditional CGRA. All evaluations have been performed on post-place-and-route results on fully working RTL implementations of all architectures. Compared to a traditional CGRA, Blocks reduces reconfiguration energy overhead between 46% and 76% (average 60%), depending

on the benchmark, without performance penalty. The system-level energy reduction is between 9% and 29% (average 22%). This showed that separation of control and data for energy-efficient CGRAs makes sense. In many applications, especially in applications where data-level parallelism is available, Blocks will achieve better energy efficiency. Furthermore, Blocks enables area reduction for applications, where data-level parallelism can be exploited. A first version of Blocks has been taped out and shown to be fully functionally correct.

Blocks is also compared against fixed architectures, an 8-lane SIMD with control processor and a 8-issue slot VLIW. The geometric mean of the performance improvement of Blocks is 3.1× and 2.4× over the SIMD and VLIW, respectively. Despite reconfiguration overhead, energy consumption is 2.1× and 1.8× lower for Blocks. Furthermore, Blocks energy is over 8× lower compared to an ARM Cortex-M0, while achieving a speed-up of 68.9×. The results show that although there is a price to pay for flexibility, it might be lower than expected [45]. To the best of our knowledge, this work contains the first architecture comparison on post place-and-route layouts of full processor designs on a 40-nm commercial library.

When performance per area is considered, Blocks even outperforms the fixed architectures and reference CGRA. In the case of the reference architectures, this is achieved by higher performance, while in the case of the reference CGRA, this is achieved by an area reduction.

A. Future Work

Current architectural improvements that are investigated are latch-based memory designs to reduce the area of the instruction memories and local memories, in multigranular FUs that can adapt to the required data-width, and in instantiation of multiprocessors on the Blocks fabric by using more than one ABU. For any architecture to be widely accepted, tool support is crucial; therefore, our research includes improving compiler support for Blocks. An initial LLVM backend is developed but does not yet take full advantage of all hardware possibilities.

REFERENCES

[1] F. Li, D. Chen, L. He, and J. Cong, "Architecture evaluation for power-efficient FPGAs," in *Proc. ACM/SIGDA 11th Int. Symp. Field Program. Gate Arrays*, 2003, pp. 175–184.

[2] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of FPGA implementations," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Mar./Apr. 2008, pp. 1417–1420.

[3] "The industry's first floating-point FPGA." [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/po/bg-floating-point-fpga.pdf (accessed Oct. 9, 2020).

[4] "Ultrascale architecture DSP slice." [Online]. Available: www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf (accessed Oct. 21, 2020).

[5] T. S. Czajkowski *et al.*, "From OpenCL to high-performance hardware on FPGAS," in *Proc. 22nd Int. Conf. Field Programm. Logic Appl. (FPL)*, 2012, pp. 531–534.

[6] M. Wijtvliet, L. Waeijen, M. Adriaansen, and H. Corporaal, "Reaching intrinsic compute efficiency requires adaptable micro-architectures," in *Proc. 9th Int. Workshop Program. Archit. Heterogeneous Multicores (MULTIPROG)*, 2016, pp. 1–7.

[7] R. Hameed *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, 2010, pp. 37–47.

[8] "Zynq-7000 SoC." [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html (accessed Oct. 21, 2020).

[9] "Intel SoC FPGAs programmable devices." [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/soc.html (accessed Oct. 21, 2020).

[10] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *Proc. Int. Conf. Embedded Comput. Syst. Archit. Model. Simul. (SAMOS)*, Jul. 2016, pp. 235–244.

[11] M. Wijtvliet, J. Huisken, L. Waeijen, and H. Corporaal, "Blocks: Redesigning coarse grained reconfigurable architectures for energy efficiency," in *Proc. 29th Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2019, pp. 17–23.

[12] A. Danilin, M. Bennebroek, and S. Sawitzki, "Astra: An advanced space-time reconfigurable architecture," in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, Aug. 2006, pp. 1–4.

[13] J. R. Anderson, S. Sheth, and K. Roy, "A coarse-grained FPGA architecture for high-performance FIR filtering," in *Proc. ACM/SIGDA 6th Int. Symp. Field Program. Gate Arrays*, 1998, pp. 234–244. [Online]. Available: http://doi.acm.org/10.1145/275107.275143

[14] W.-P. Kiat, K.-M. Mok, W.-K. Lee, H.-G. Goh, and R. Achar, "An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications," *Microprocess. Microsyst.*, vol. 73, Mar. 2020, Art. no. 102966. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933118304691

[15] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in *Proc. Conf. Design Autom. Test Eur.*, 2001, pp. 642–649.

[16] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of Signal Processing Systems*. Boston, MA, USA: Springer, 2010, pp. 449–484.

[17] R. W. Hartenstein, A. G. Hirschbiel, M. Riedmuller, K. Schmidt, and M. Weber, "A novel ASIC design approach based on a new machine paradigm," *IEEE J. Solid-State Circuits*, vol. 26, no. 7, pp. 975–989, Jul. 1991.

[18] K. Sankaralingam *et al.*, "TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 62–93, 2004.

[19] 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*. Heidelberg, Germany: Springer, 2003, pp. 61–70.

[20] E. Waingold *et al.*, "Baring it all to software: Raw machines," *Computer*, vol. 30, no. 9, pp. 86–93, Sep. 1997.

[21] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proc. Int. Conf. Compilers Archit. Synth. Embedded Syst.*, 2003, pp. 137–147. [Online]. Available: http://doi.acm.org/10.1145/951710.951730

[22] R. Jordans, "Instruction-set architecture synthesis for VLIW processors," Elect. Eng., Embedded Syst. Group, Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2015.

[23] R. Koenig *et al.*, "KAHRISMA: A novel hypermorphic reconfigurable-instruction-set multi-grained-array architecture," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2010, pp. 819–824.

[24] J. D. Souza, L. Carro, M. B. Rutzig, and A. C. S. Beck, "A reconfigurable heterogeneous multicore with a homogeneous ISA," in *Proc. DATE*, 2016, pp. 1598–1603.

[25] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique, "X-CGRA: An energy-efficient approximate coarse-grained reconfigurable architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2558–2571, Oct. 2020.

[26] J. D. Lopes, M. P. Véstias, R. P. Duarte, H. C. Neto, and J. T. de Sousa, "Coarse-grained reconfigurable computing with the versat architecture," *Electronics*, vol. 10, no. 6, p. 669, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/6/669

[27] L. Seiler *et al.*, "Larrabee: A many-core x86 architecture for visual computing," in *Proc. ACM SIGGRAPH Papers*, 2008, pp. 1–15. [Online]. Available: http://doi.acm.org/10.1145/1399504.1360617

[28] I. Lebedev *et al.*, "MARC: A many-core approach to reconfigurable computing," in *Proc. Int. Conf. Reconfig. Comput. FPGAs*, Dec. 2010, pp. 7–12.

[29] S. Lee, J. Oh, J. Park, J. Kwon, M. Kim, and H.-J. Yoo, "A 345 mW heterogeneous many-core processor with an intelligent inference engine for robust object recognition," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 42–51, Jan. 2011.

[30] S. Pal *et al.*, "Transmuter: Bridging the efficiency gap using memory and dataflow reconfiguration," in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn.*, 2020, pp. 175–190.

[31] K.-C. Chang, J.-S. Shen, and T.-F. Chen, "Evaluation and design trade-offs between circuit-switched and packet-switched NOCs for application-specific SOCs," in *Proc. 43rd ACM/IEEE Design Autom. Conf.*, 2006, pp. 143–148.

[32] Y. He *et al.*, "A configurable SIMD architecture with explicit datapath for intelligent learning," in *Proc. Int. Conf. Embedded Comput. Syst. Archit. Model. Simul. (SAMOS)*, Jul. 2016, pp. 156–163.

[33] "Cortex-m0—Arm developer." [Online]. Available: https://developer.arm.com/products/processors/cortex-m/cortex-m0 (accessed Jun. 14, 2019).

[34] "Cortex-m4—Arm developer." [Online]. Available: https://developer.arm.com/products/processors/cortex-m/cortex-m4 (accessed Jun. 14, 2019).

[35] P. D.'Stigall and Ö. Tasar, "A measure of computer flexibility," *Comput. Elect. Eng.*, vol. 2, nos. 2–3, pp. 245–253, 1975. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0045790675900117

[36] R. Rabbah, I. Bratt, K. Asanovic, and A. Agarwal, *Versatility and VersaBench: A New Metric and Benchmark Suite for Flexible Architectures*, MIT, Cambridge, MA, USA, Dec. 2005.

[37] J. Teittinen *et al.*, "A 5.3 pj/op approximate TTA VLIW tailored for machine learning," *Microelectron. J.*, vol. 61, pp. 106–113, Mar. 2017.

[38] W. J. Dally, U. J. Kapasi, B. Khailany, J. H. Ahn, and A. Das, "Stream processors: Progammability and efficiency: Will this new kid on the block muscle out ASIC and DSP?" *Queue*, vol. 2, no. 1, pp. 52–62, 2004.

[39] P. S. Vaidya, J. J. Lee, V. S. Pai, M. Lee, and S. Hur, "Symbiote coprocessor unit—A streaming coprocessor for data stream acceleration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 3, pp. 813–826, Mar. 2016.

[40] S. J. Carey, A. Lopich, D. R. W. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535GOPS/W 256× 256 SIMD processor array," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, 2013, pp. C182–C183.

[41] T. Geng, L. Waeijen, M. Peemen, H. Corporaal, and Y. He, "Macsim: A MAC-enabled high-performance low-power SIMD architecture," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Limassol, Cyprus, 2016, pp. 160–167.

[42] Y. Du, W. Li, Z. Dai, and L. Nan, "PVHArray: An energy-efficient reconfigurable cryptographic logic array with intelligent mapping," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 5, pp. 1302–1315, May 2020.

[43] L. Liu *et al.*, "An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding," *IEEE Trans. Multimedia*, vol. 17, no. 10, pp. 1706–1720, Oct. 2015.

[44] Y. Wang *et al.*, "On-chip memory hierarchy in one coarse-grained reconfigurable architecture to compress memory space and to reduce reconfiguration time and data-reference time," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 983–994, May 2014.

[45] J. A. Fisher, P. Faraboschi, and G. Desoli, "Custom-fit processors: Letting applications define architectures," in *Proc. 29th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Dec. 1996, pp. 324–335.