REVIEWS

Journal of Semiconductors
(2020) 41, 021401
doi: 10.1088/1674-4926/41/2/021401

Architecture, challenges and applications of dynamic reconfigurable computing

Yanan Lu, Leibo Liu†, Jianfeng Zhu, Shouyi Yin, and Shaojun Wei

Institute of Microelectronics, Tsinghua University, Beijing 100084, China

Abstract: As a computing paradigm that combines temporal and spatial computations, dynamic reconfigurable computing provides superiorities of flexibility, energy efficiency and area efficiency, attracting interest from both academia and industry. However, dynamic reconfigurable computing is not yet mature because of several unsolved problems. This work introduces the concept, architecture, and compilation techniques of dynamic reconfigurable computing. It also discusses the existing major challenges and points out its potential applications.

Key words: reconfigurable computing; architecture; challenge; application

Citation: Y N Lu, L B Liu, J F Zhu, S Y Yin, and S J Wei, Architecture, challenges and applications of dynamic reconfigurable computing[J]. *J. Semicond.*, 2020, 41(2), 021401. http://doi.org/10.1088/1674-4926/41/2/021401

1. Introduction

Reconfigurable computing has drawn wide attention in both academia and industry in the past decade, and respective commercial products are quickly emerging[1–10]. The popularity of reconfigurable computing is mainly due to the following reasons. First, energy efficiency has become a more important criterion than performance. Therefore, computing infrastructures need to reduce power consumption while pursuing high performance and reconfigurable computing architectures provide higher energy efficiency compared to general-purpose processors[11]. Second, it is essential for computing architectures to keep high flexibility while improving performance[12, 13]. ASIC designs can achieve optimal performance, and thus custom function modules are increasingly integrated into a system on chip (SoC) to form a heterogeneous computing architecture. However, due to the weak flexibility of ASIC designs, resources cannot be reused and they can only perform a specific task. These modules probably cannot run at the same time, leading to low resource utilization and low area efficiency from the view of the SoC. Meanwhile, reconfigurable architectures take advantage of both spatial domain and time domain computing, which improves area efficiency while achieving comparable performance with ASIC designs. Finally, economy is another important factor. As CMOS technology progresses to below 22 nm, the non-recurring engineering cost of chip production becomes ever more expensive and small-quantity dedicated circuits are difficult to recover the cost. Therefore, it is essential to replace these dedicated designs with programmable general-purpose processing architectures[14, 15]. Fig. 1 compares different chips in terms of software and hardware programmability. A dynamic reconfigurable computing chip is a promising alternative due to its strong software and hardware programmability.

Reconfigurable computing is not a new concept. As early as the 1960s, Prof. Gerald Estrin of UCLA proposed that computers can be composed of a main processor and an array of reconfigurable hardware[16]. The main processor is responsible for controlling the behavior of the reconfigurable hardware. The latter accelerates the execution of specific tasks by tailoring and reconfiguration according to the computing characteristics of the target applications[17, 18]. However, this innovative concept was limited by the semiconductor technology at that time and did not receive much attention. In the next few decades, the computing architectures evolved in two primary groups, ASICs and general-purpose processors (GPPs). The advantages and disadvantages of these two groups are prominent. The reconfigurable architectures take advantage of both ASICs and GPPs, and deliver a reasonable tradeoff between performance, power, and flexibility. In the 1990s, reconfigurable computing gradually attracted more interest and was widely studied by researchers. In 1999, the Reconfigurable Technology Research Center of the University of California, Berkeley, proposed a more general definition of reconfigurable architectures, as follows: 1) the functional units on the chips should have post-fabrication programmability (i.e, the function of the hardware units can be reconfigured after silicon implementation); and 2) it should be able to achieve the spatial mapping of algorithms to computational engines. Computing methods with these two characteristics can be classified as reconfigurable computing[19]. This definition highlights two major features that distinguish reconfigurable architectures from other computing architectures.

From the perspective of implementation, reconfigurable architectures mainly include FPGAs and coarse-grained reconfigurable arrays (CGRAs). FPGA is an early form of reconfigurable computing whose development continues today. Xilinx developed the world's first FPGA in 1986[20], and has since continuously improved the structure, technology, and scale of the chip. FPGAs were primarily utilized for functional verification of a system design or as an alternative for ASICs to implement some functions in a system. Thanks to the increas-

Correspondence to: L B Liu, liulb@tsinghua.edu.cn
Received 17 SEPTEMBER 2019; Revised 14 OCTOBER 2019.

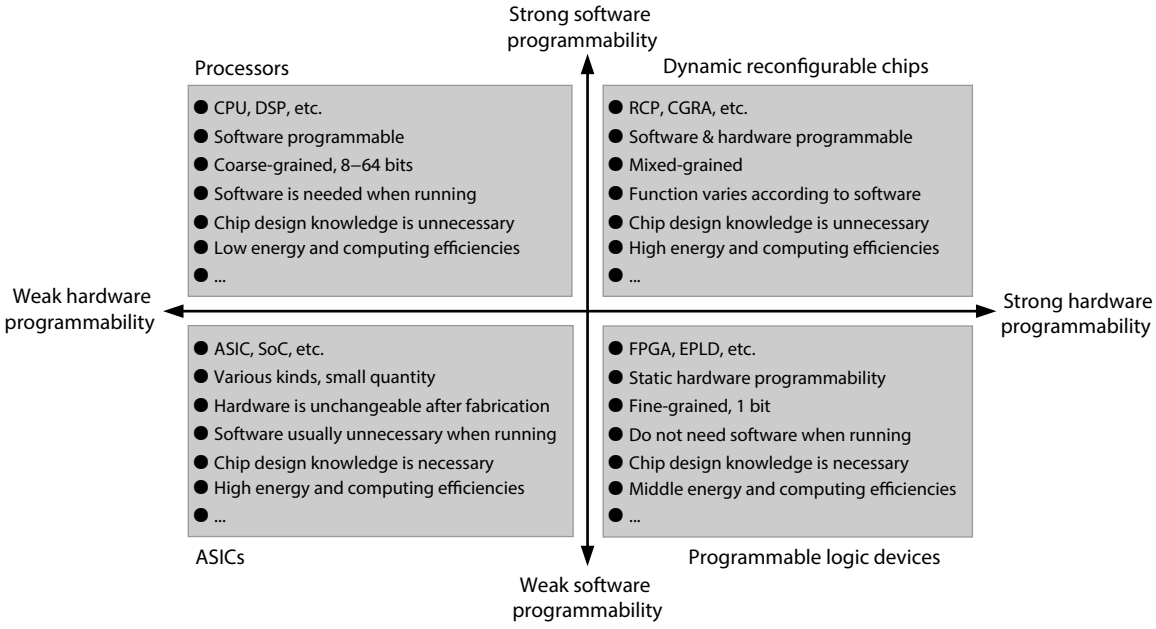

Fig. 1. Programmability comparisons among different chips.

ingly abundant resources on a chip and the fast development of CAD tools based on high-level programming language, FPGAs are now widely employed as accelerators in the mainstream computing infrastructures. For instance, Intel acquired Altera and integrated FPGA using a Xeon CPU as an accelerator[21]; IBM released SuperVessel cloud server based on GPU and FPGA[22]; Microsoft launched a FPGA-based could server Azure[23]. However, due to the fine-grained logic cell and static reconstruction, FPGAs have the drawbacks of low area efficiency, high power consumption, large configuration bit-stream and long reconfiguration time. To mitigate these inefficiencies, recent FPGAs have integrated various hard IPs and employed techniques such as block-based partial reconfiguration[24]. Some researchers even proposed to implement virtual CGRAs in FPGAs[25]. In contrast, CGRAs provide coarse-grained computational granularity and structure that better match the need of applications. Compared with FPGAs, CGRAs have great advantages in area efficiency, power efficiency and reconfiguration time. For example, the typical reconfiguration time of FPGAs ranges from several hundred milliseconds to several seconds, while for CGRAs, reconfiguration only takes a few nanoseconds to several hundred nanoseconds. Consequently, CGRAs are also called dynamic reconfigurable architectures. Since the 1990s, a number of influential CGRAs—such as Morphosys[26], ADRES[27], PACT XPP[28], and REMUS[29]—have been developed, targeting the applications of signal processing, multimedia, and so on. In recent years, researchers have continued to study the design of CGRAs and have proposed many latest implementations, such as Plasticine[1], CGRA-ME[30], PX-CGRA[31], i-DPs CGRA[32], dMT-CGRA[33]. However, CGRAs are not yet widely used in industry due to their inconsistent structures, and immature programming and compilation tools, which will be explained in detail later on.

There are many previous surveys on reconfigurable computing[12, 13, 34–37]. However, most of them focused on FPGA technology, with little to do with CGRAs. Given that FPGAs are relatively mature, while CGRAs still have many unsolved problems and are far from large-scale commercial utilization, this paper focus on CGRAs, which employ a dynamically reconfigurable computing architecture. The rest of this paper is organized as follows: Section 2 and Section 3 introduce the architecture and compilation techniques of CGRAs, Section 4 discusses the main challenges associated with CGRAs and possible solutions, Section 5 explores the future applications of CGRAs, and Section 6 concludes this paper.

2. Architecture

As mentioned earlier, the implementation forms of reconfigurable computing mainly include FPGAs and CGRAs. Since FPGAs are relatively mature and their architecture is well known, this section focuses on the architecture of CGRAs.

2.1. Architecture model

The basic architecture model of CGRAs is shown in Fig. 2. It consists mainly of two parts: a reconfigurable controller (RCC) and a reconfigurable datapath (RCD). Both RCC and RCD contain memory for storing configuration and data, respectively. It can be seen that this architecture is a variant of the von Neumann computing architecture. The main difference from an instruction processor is that RCC controls the behavior of RCD through configuration rather than instructions. RCD can be reconfigured because it integrates abundant basic arithmetic units (such as adders, multipliers, etc.) and logical units (such as AND, OR, NAND, XOR, etc.), and RCC can select and organize these computing units to achieve specific structures and functions according to configuration. The hardware structures of RCC and RCD are introduced next.

2.2. Reconfigurable controller

The hardware structure of RCC consists of three parts: configuration management unit, memory module and configuration interface (as shown in Fig. 3). The configuration management unit receives configuration context from the outside and parses it to get the internal control signals and configuration context. The internal configuration context is stored in

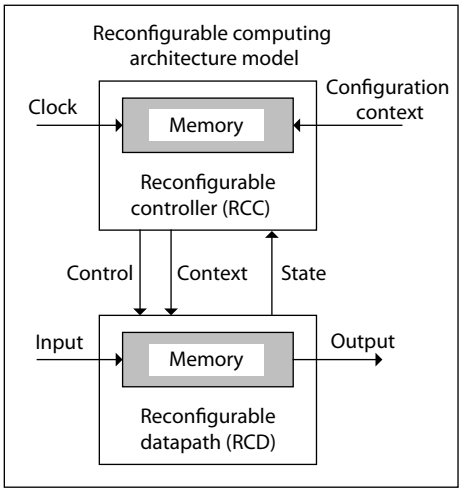

Fig. 2. An architecture model of reconfigurable computing.

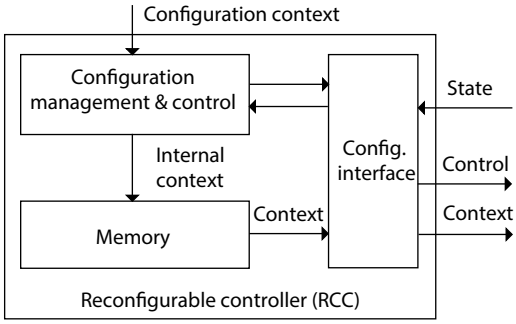

Fig. 3. The structure of RCC.

the memory module and transferred to RCD by the configuration interface as needed. The configuration interface is used to send configuration context and control signals to RCD.

The RCC is responsible for the organization and management of the configuration of RCD. Controllers in traditional single-core processors focus on timing scheduling in single node. Since the instruction stream is repeatedly executed on a single node, many parallelization techniques such as pipelining are employed, thus the timing requirement of the controller is high. In contrast, reconfigurable computing processors are mostly implemented in the form of arrays, which are oriented to computing resource scheduling of multi-nodes. processing elements (PEs) are usually not as complex as a single-core processor, and the node control timing of the controller is relatively simple. The overall efficiency of spatial and temporal utilizations is more important than node scheduling, which presents new design requirements for the controller. In the case of a large amount of configuration, it is conceivable to add a customized accelerator or even a control unit array into RCC.

2.3. Reconfigurable datapath

The RCD generally includes four parts: a processing element array (PEA), a memory, a data interface, and a configuration interface (as shown in Fig. 4). The configuration interface obtains control signals and configuration context from RCC, while it sends out states. The configuration interface then parses the configuration context, configures the function of the PEA, and schedules the execution order of tasks on PEA. After the PEA is configured, it starts to execute in a set time, driven by dataflow, just like an ASIC. The input data

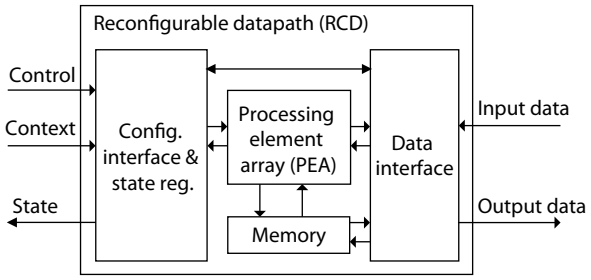

Fig. 4. The structure of RCD.

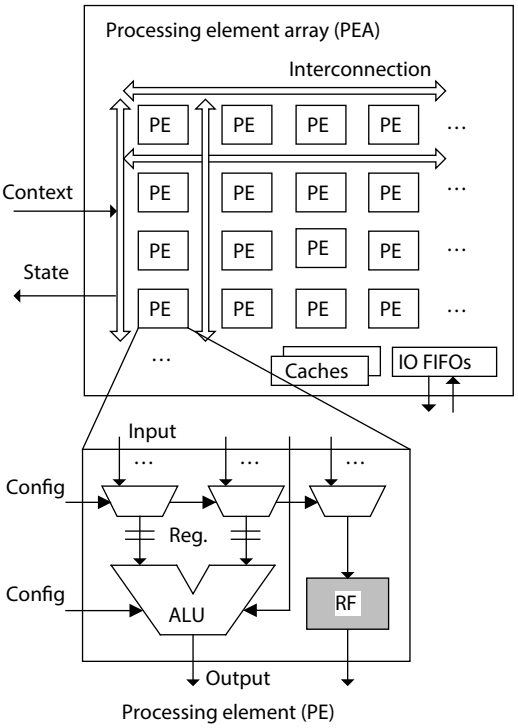

Fig. 5. The structures of PEA and a PE.

of the PEA is obtained from the data interface, and the intermediate data is buffered in the memory. In addition to completing the access and write back of external data, the data interface can also accept signals from the configuration interface to shape and transform (such as transposition, splicing operation, etc.) the data to cooperate with the execution of the PEA.

The basic structure of the PEA is shown in Fig. 5. A large number of PEs are combined together under a certain connection for parallel computing. A PE is generally composed of an arithmetic logic unit (ALU) and a group of registers. For parallel computing, the main bottleneck limiting the performance lies in the external memory interface when computing resources are sufficient, which is referred to as throughput computing. Therefore, the caching and prefetching of data is very important, which can effectively reduce the dependence on external memory. In a PEA, a hierarchical and distributed memory structure is usually employed. Except the multi-layer design of the memory module in Fig. 4, a large number of distributed memories—such as an interface buffer, an array-level cache, an internal PE memory—are also required inside the PEA.

The PEA can be classified into coarse-grained, medium-grained, fine-grained, mixed-grained reconfigurable arrays according to the granularity of the PEs. Higher computation-

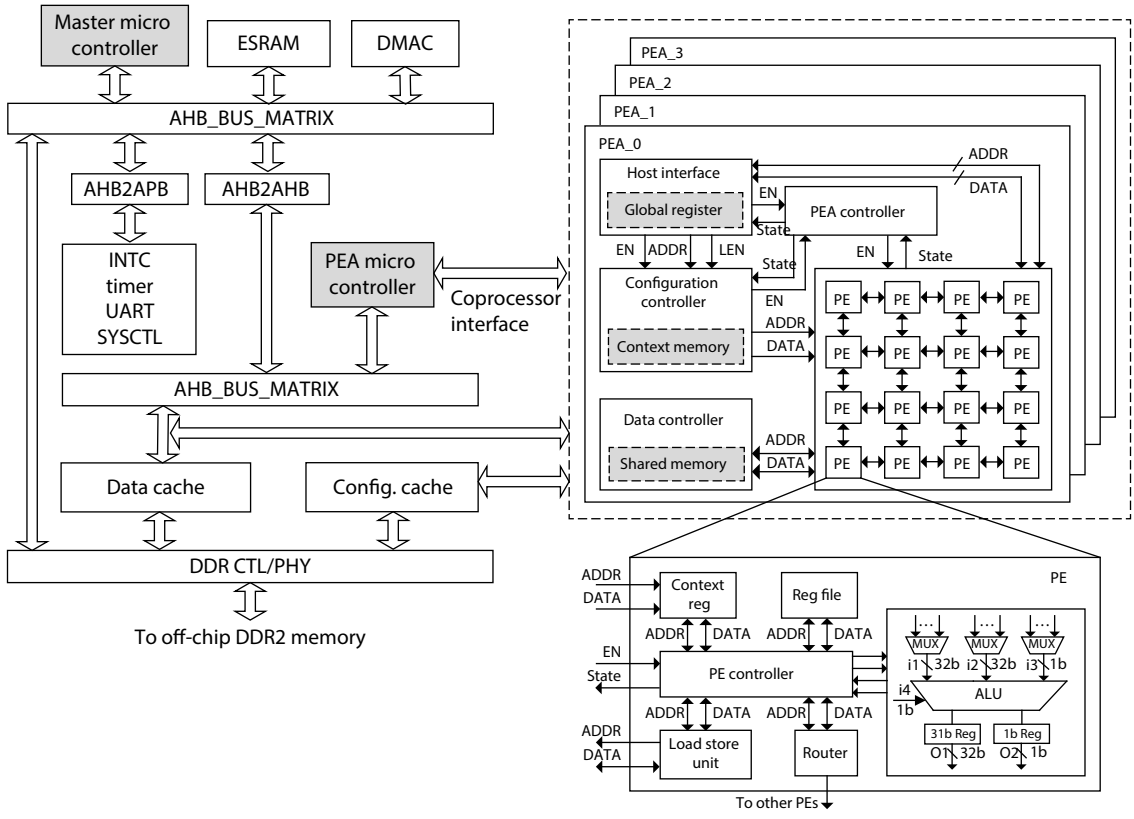

Fig. 6. The architecture of HReA.

al efficiency can be achieved when the granularity matches the data width of the applications. For example, the fine-grained PEA is suitable for bit operations-based applications; the coarse-grained PEA which may include larger functional modules such as addition and multiplication units, performs better for graphic and image processing, as well as digital baseband operations. The mixed-grained PEA combines multiple granularities and is more flexible, making it suitable for various data widths.

2.4. Configuration

The configuration of a dynamic reconfigurable processor includes operator configuration, interconnection configuration and data transmission configuration[38]. Operator configuration is used to configure the ALU functions in PEs. ALU functions are configured as arithmetic logic operations or customized operations related to application fields, which are similar to the arithmetic logic operation instructions in instruction set architecture (ISA). However, since the functions and structures of PEs in different dynamic reconfigurable processors may be significantly different, there is no uniform instruction set and instruction format. The interconnection configuration is used to configure the interconnection structure between the PEs in the array to transmit the intermediate data between the data registers in each PE. Therefore, the interconnection configuration is similar to the MOV instruction in ISA for data transmission between registers. Data transmission configuration is used to configure data transmission between the PE array and the data memory, as well as the transmission between data memories, which is similar to the MOV instruction for data transmission between registers and local memory and the MOVX instruction for data transmission between local memory and global memory in ISA.

The design of a configuration system for a dynamic reconfigurable processor is similar to the design of ISA in GPPs. It includes the design of the organizational structure of configuration information, the configuration storage scheme, and the configuration management scheme. Consequently, it belongs to the category of architecture. In the design of organizational structure, the configuration information is allocated to different layers but organized as a whole. In the design of configuration storage scheme, corresponding storage schemes are designed for the layer configuration and the information in each layer. The configuration information is stored in the configuration memory. The configuration management scheme is designed based on the previous two steps. Unlike the static organizational structure, configuration management refers to the dynamic configuration flow, which reads out the various kinds of configuration information from configuration memory and writes into the corresponding hardware modules to complete the configuration.

2.5. Implementation instance: HReA

To explain the architecture of a dynamic reconfigurable processor clearly, this section will introduce an implementation instance: HReA[39]. As shown in Fig. 6, the HReA architecture comprises three main functional parts: master micro-controller, PEA micro-controller and PEAs. Master micro-controller and PEA micro-controller comprise the RCC of HReA, while PEAs comprise the RCD. There is also direct memory access controller (DMAC), embedded SRAM (ESRAM) and other common peripherals, such as interrupt controller (INTC), timer, UART, and system controller (SYSCTL). On-chip caching (i.e, 128 kB configuration cache and 256 kB data cache) is used to

reduce the required off-chip memory bandwidth. A dedicated on-chip memory controller (i.e, DDR CTL/PHY) is designed to connect off-chip DDR2 memory with on-chip caches.

Master micro-controller is the master-control unit and started up under the control of SYSCTL. It is responsible for configuring DMAC to transfer program package from DDR into the ESRAM. PEA micro-controller is dedicated to control the configuration and data for PEAs. It assigns tasks for PEAs via coprocessor interface. There are four PEAs (i.e, PEA_0, PEA_1, PEA_2, and PEA_3) and they are the key components to implement task acceleration. They can be dynamically combined according to the requirements of calculation so as to achieve algorithm-level parallelism and can also be turned off individually to save power. When completing tasks, PEAs notify PEA micro-controller via INTC.

The main functionality of a PEA is to fetch, process, store and export data driven by control and configuration flows. The core part of a PEA is the 4×4 hybrid-grained PEs which are organized in a nearby manner. Based on configuration context, the interconnections between PEs can be dynamically reconfigured via configuring router connection. Each PEA also contains auxiliary components, including host interface, PEA controller, configuration controller and data controller, to prepare control signal, configuration, and operand data for the PE array. The host interface receives coprocessor instructions from PEA micro-controller and reserves data exchanged between PEA micro-controller and PE array in global register. The PEA controller enables calculation on PE array under the control of the host interface. The configuration controller, containing a context memory for configuration contexts, is responsible for scheduling the execution sequence. The data controller provides operand data to the PE array, with a shared memory for buffering input data, intermediate results, and final outputs.

PEs can be dynamically configured to execute arithmetic and logic operations under the control of configuration context. Each PE in HReA combines a 32-bit data path with a 1-bit data path to accommodate multiple computing granularities, providing up to 15 different operations—including logical operations, such as AND, OR, XOR, and so on—and arithmetic operations—such as adder, subtracter, multiplier, leading-zero detector, shifter, multiplexer, absolute, and so on. Based on configuration context stored in the context register, the PE controller is responsible for selecting operand data (i.e, ALU_input) and generating operation code (i.e, ALU_op) for the ALU. The calculation results of the ALU can be kept in the inner register file for short-term storage or can be sent to shared memory via load store unit (LSU) for long-term storage.

Based on the hybrid-grained PE structure, HReA can efficiently deal with both computing-intensive kernels and control-intensive kernels which involve various branches, loops, and sequential codes. Measured results on kernels from the 13-Dwarfs[40] show that HReA has great improvements in energy efficiency compared with instruction-driven processors, while maintaining high-enough functional flexibility.

3. Compilation

Unlike GPPs and FPGAs, which compute temporally and spatially respectively, dynamic reconfigurable processors are both temporal and spatial computing fabrics. The compilation of a dynamic reconfigurable processor is very important and has a direct impact on performance. This section describes the compiler framework and presents the key compiling techniques for dynamic reconfigurable processors.

3.1. The compiler's framework

To process the computing tasks of various applications, a corresponding target program must be generated by a compiler for the component units (i.e, RCC and RCD) of the reconfigurable processor. The compiler generates control codes for RCC and configurations for RCD via the processes of code transformation, task partition, task scheduling, mapping, and configuration generation.

Since the hardware structure of a reconfigurable processor is significantly different from that of a conventional GPP, the compilation flow and functions of a reconfigurable processor compiler are different from those of traditional compilers (such as GNU gcc compiler). A conventional compiler compiles input application codes to generate assembly language codes and corresponding machine codes for a target processor. However, a reconfigurable processor compiler performs code analysis on the input application, divides the application into software and hardware codes by using the software and hardware co-design method, and then respectively compiles the two kinds of codes to generate control codes for RCC and configurations for RCD.

Fig. 7 shows an example of dividing and executing a kernel on HReA. The two loops in the kernel consume most of the execution time and can be accelerated on PEAs, while the Pre-loop/Inter-loop/Post-loop codes are executed on PEA micro-controller. In a dynamic reconfigurable processor, multiple PEs in the array can achieve parallel processing or pipelined sequential processing. For the first loop which is iteration independent in Fig. 7, it can be fully unrolled. Thus, iteration 0, 1, 2, 3 can be executed in parallel on different PEs. In a spatial mapping, Stage 0-1, Stage 1-1, Stage 2-1, Stage 3-1 are mapped onto PE0, PE1, PE2, PE3, respectively, and Stage 0-2, Stage 1-2, Stage 2-2, Stage 3-2 are mapped onto PE4, PE5, PE6, PE7, respectively (PE4, PE5, PE6, PE7 are on the second row in PEA.). However, in a temporal mapping, Stage 0-2, Stage 1-2, Stage 2-2, Stage 3-2 are also mapped onto PE0, PE1, PE2, PE3 respectively. The second loop in Fig. 7 is iteration dependent. Assumed that the initiation interval is 1. Stage 1-4 mapped onto PE1 should be executed one cycle after Stage 0-4 mapped onto PE0.

The compiler framework of a reconfigurable processor is shown in Fig. 8. First, the compiler needs to transform and optimize the code of an application to get the data flow graph (DFG). The DFG is then mapped to the reconfigurable processor. Owing to limited hardware resources, the DFG usually needs to be partitioned and divided into a series of interdependent subgraphs. These subgraphs will be scheduled by RCC and mapped to RCD for execution after task mapping and configuration generation.

The task mapping process includes register allocation, operator mapping and memory mapping. In a reconfigurable processor, registers and internal memory are designed for data interaction and transfer between subtasks. Therefore, necessary register and memory allocation besides operator mapping is required in compilation. The last process is configuration generation and optimization, which generates con-

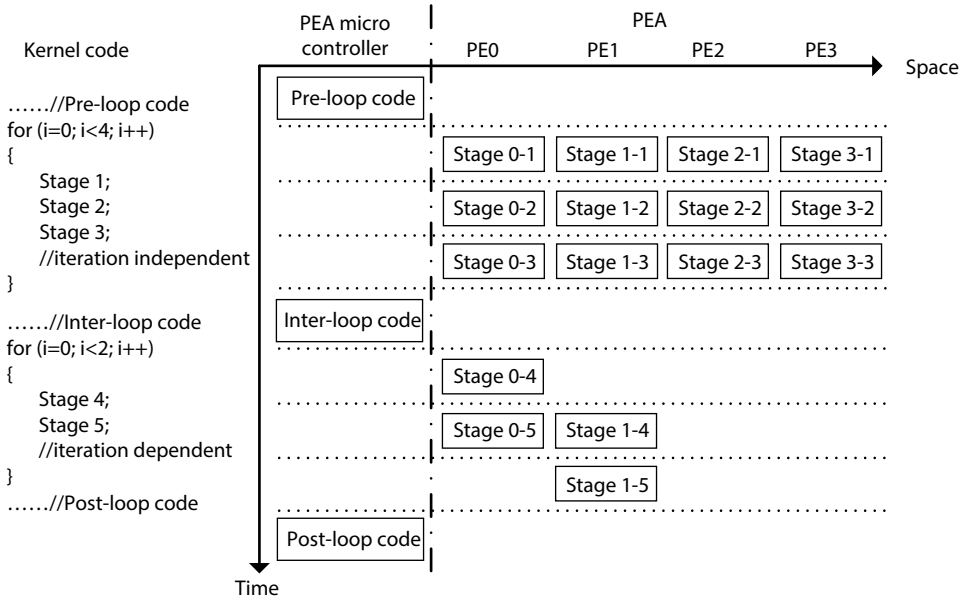

Fig. 7. Example of dividing and executing a kernel.

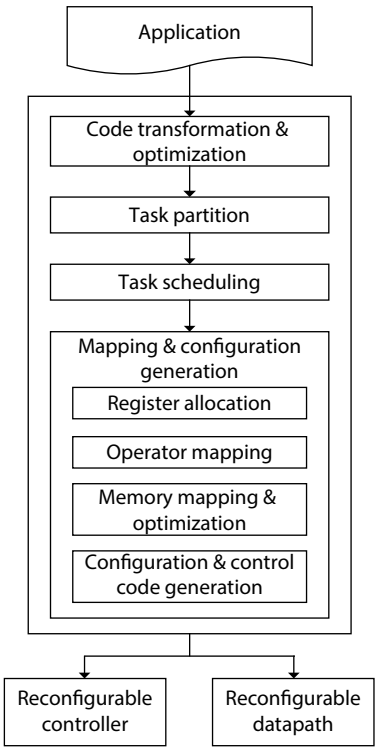

Fig. 8. The compiler framework of a reconfigurable processor.

trol codes and configuration information for RCC and RCD respectively. To improve the overall performance, the configuration information needs to be reasonably optimized by eliminating redundant information and compression.

3.2. Key techniques for compiling

There are several key techniques in the compilation of a dynamic reconfigurable processor, such as code transformation and optimization, temporal task partition, internal memory management, and configuration optimization. This section discusses these techniques.

3.2.1. Code transformation and optimization

For most reconfigurable processors, the application's program codes are written in a high-level programming language (e.g, C), which is mostly procedure oriented and has few parallelizable code segments. The parallelism in code segments are not expressed explicitly in the program. To effectively improve the performance of an application, it is necessary to fully exploit the code blocks that have high parallelism in the program[41]. Relevant research has shown that the kernel loops in applications take up most of the execution time[42]. Since data dependencies may exist between loop iterations, it is necessary to expand the loop body to further explore the potential of parallelism. Some code transformation and optimization techniques have been proposed, such as loop unrolling, scalar substitution[43], affine transformation[44], and so on.

3.2.2. Temporal task partition

Dynamic reconfigurable computing architectures support changing their hardware functions by dynamically switching the configurations. When a task executed on the reconfigurable computing processor exceeds the hardware resources, it is usually divided into a series of small tasks (subtasks), which are scheduled and sequentially executed on the hardware through multiple times of configuration. Therefore, the same hardware can be configured multiple times and perform repeated execution[45].

The temporal task partition technique divides a task into a series of subtasks that are related to each other in the time domain. To execute tasks beyond the computing resources on the limited hardware, large tasks are divided into several subtasks and time-multiplexing the hardware resources are adopted. Fig. 9 shows an example of temporal task partition[46], where a large task is divided into three subtasks whose computational scale satisfies the hardware constraints. These three subtasks temporally reuse the same hardware resource. The configurations of the subtasks are sequentially sent to the computing array to implement the respective functions. The function of the large task is equivalently achieved.

3.2.3. Internal memory management

When multiple subtasks are executed on the same re-

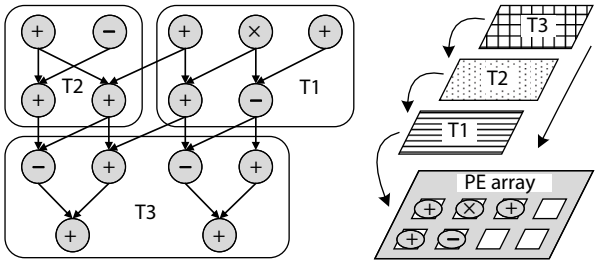

Fig. 9. Temporal partition of task graph.

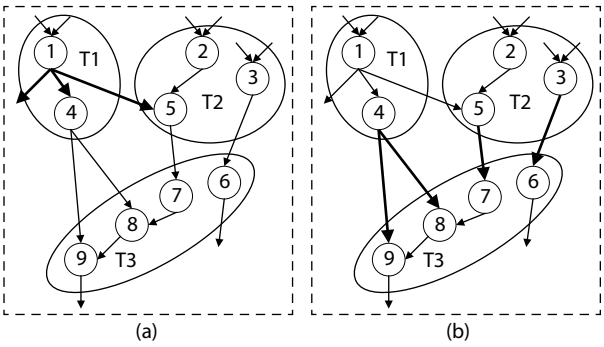

Fig. 10. Illustration of internal memory management. (a) Multiple output targets. (b) Communication between subtasks.

configurable hardware in time-multiplexing manner, there are possibly data dependencies between these subtasks. Therefore, it is necessary to consider data interaction between subtasks through internal memory in the process of mapping and configuration generation.

Figs. 10(a) and 10(b) show the two main problems of internal data interaction that need to be addressed. The first problem comes from data interaction when an operator has multiple output targets, as shown in Fig. 10(a). A dynamic reconfigurable processor usually provides limited data manipulations due to hardware complexity. When there is more than one output targets (external output and internal data transfer between subtasks) for an operator in a subtask, the storage resources need to be effectively managed to reduce the occupancy rate and to ensure the correctness. Techniques such as variable life cycle analysis and operator reordering can be used to reduce the occupancy rate of storage resource and improve the computing performance. The second problem comes from the data interaction between subtasks. When a subtask requires much intermediate data as input and the storage locations of these data are scattered, it is necessary to rearrange these data for block operations. The rearrangement can extract the operation data required by the current subtask and improve the efficiency of memory access. Techniques such as subtask correlation analysis and data splicing can be used to improve the efficiency of memory access.

3.2.4. Configuration optimization

As mentioned in the previous sections, the hardware function of a dynamic reconfigurable processor is changed by dynamically switching the configuration. For the configurations of multiple subtasks, eliminating the redundancy in configurations and compressing the configuration volume will greatly shorten the configuration loading time. The reason for the redundancy is that the operator connection graphs in different subtasks have similar structures. Two techniques can be used to eliminate the redundancy in configurations. First, by analyzing the statistical correlation in the bit rate of redundant information, traditional data compression methods can be utilized to reduce the configuration volume[47]. Second, direct analysis of the correlation of subtask DFGs can reduce the generated configuration[48].

4. Challenges

Although there have been many successful CGRA designs, which are superior in terms of energy efficiency and flexibility, CGRA is still immature and far away from large-scale commercial utilization because there are still some key technologies and bottlenecks that have not been well resolved. Some of the existing technical challenges and proposed solutions follow.

4.1. Cooperation of temporal and spatial mapping

Mapping an application written in a high-level programming language to a reconfigurable chip is a complex issue. A variety of techniques can be used together in the mapping process. Ref. [49] proposed an aggressive pipelining method for irregular applications on reconfigurable hardware. For control flows in irregular applications that could not be predict by static analysis, the abundant spatial computing resources are used at runtime to aggressively execute tasks concurrently. Therefore, fine-grained parallelism in applications can be efficiently developed. After utilizing a combination of methods, the computing performance can be increased by an order of magnitude. A polyhedral model that is based mapping technology can also be adopted for performance optimization. Taking into account parameters such as dynamic reconfiguration, array calculation and cache access, and using a joint optimization method of affine transformation and loop tiling to establish a performance model and a power consumption model, the execution time of a task can be reduced by about 20%[50].

4.2. Control-intensive task parallelization

Reconfigurable computing architectures are effective for compute-intensive tasks, but how to perform control-intensive tasks is a difficult problem. Exploring the parallelization of control-intensive tasks on a centralized-controlled computing array is necessary. By giving a common mapping process and utilizing techniques such as merging branches and condition computation, configuration fusion, and configuration branch optimization, the configuration and execution time of control tasks can be reduced and the performance is improved by approximately 40%[51]. For the distributed-controlled systems, the parallelization methods of the control-intensive tasks on a distributed-controlled computing array is adopted. Ref. [52] proposed a PE that supports triggered configuration. The PE employs a structure that combines trigger mechanism and composite configuration. Thus, the instruction-level parallelism of complex control flows is efficiently achieved, reducing the latency and execution cost caused by control flow. Finally, the performance of processing control-intensive tasks is improved by 20% to 140%.

4.3. Optimization of configuration organization

Reconfigurable hardware needs continuous configuration to change the structure and function. It is important to consider the size of the configuration. Generally, the amount

of a FPGA configuration is about a dozen megabytes or tens of megabytes, and the configuration time is several hundred milliseconds to a few seconds, which is too long for a dynamic reconfigurable CGRA. To achieve reconfiguration in a short time, the first thing is to reduce the amount of configuration information. Through analysis of the computational flow graph, Ref. [53] proposed a hierarchical configuration generation technology based on isomorphic similarity matching of subgraphs. The commonality of the DFG is extracted according to the similarity matching and cross index between subgraphs. The total amount of configuration information can be reduced by more than 70% and an optimized hierarchical organization of configuration is formed.

4.4. Dynamically loading configuration

Although the amount of configuration is reduced, it still takes time to load the configuration onto the datapath. It is found that it is unnecessary to send configuration all the time, and some configuration may be resident in the memory. Ref. [54] proposed a correlation-aware caching strategy for configuration flow. An on-chip cache structure and the prefetching method have been designed for grouping the configuration according to the computing tasks. Redundant transmission of configuration flow in each layer is eliminated. The configuration sets are converged downwards by layer. The gap of configuration flow is optimized by using pipeline equalization. Consequently, the configuration amount is reduced and the configuration speed is increased, which results in a decrease in the configuration time.

These technologies relieve the problems of optimal generation, storage and loading of configuration information in dynamic reconfigurable chips. Through the maximum parallelization of configuration and execution, nanosecond-level function reconfiguration is realized, providing the foundation of both energy efficiency and flexibility for dynamic reconfigurable chips.

5. Applications

From the current successful application of CGRAs, it can be seen that they are more suitable for compute-intensive and data-intensive applications. The following classifications describe the current main applications of CGRAs.

5.1. Neural network

Since 2010, advances in neural networking technology have driven the development of artificial intelligence. Deep neural network (DNN), which is a basic supporting technology, requires complex calculations of large amounts of data with frequent inter-layer communication. Research shows that CGRA is a superior implementation of DNN because of its high throughput computing and on-chip communication capabilities. For example, Eyeriss[55] is based on the CGRA structure to minimize the energy consumption of data movement by maximizing reuse of input data. Ref. [56] proposed a runtime reconfigurable two-dimensional dataflow computing engine, which can implement a variety of convolutional NN operations in a systolic manner.

Thinker[57] is a reconfigurable hybrid-neural network processor, which was proposed by our research team. Its energy efficiency could be as high as 5.09-TOPS/W in 65 nm technology. It has two 16 × 16 reconfigurable heterogeneous PE arrays. To accelerate hybrid-NNs, the PE arrays are designed to support on demand partitioning and reconfiguration for processing different NNs in parallel. Each PE in the array supports bit-width adaption to meet variant bit-width of neural layers. Furthermore, a fused data pattern-based multi-bank memory system is designed to exploit data reuse and guarantee parallel data access. These design techniques improve the PE utilization and computing throughput, as well as energy efficiency.

5.2. Cryptography

Cryptographic processing is also a computing-intensive application, which is especially suited for CGRA-based implementations. Scholars have proposed many reconfigurable cryptographic processors based on CGRA structure. For example, Celator[58] is a reconfigurable coprocessor that implements block ciphers (AES and DES) and HASH function (SHA). Cryptoraptor[59] is a reconfigurable cryptographic processor that implements multiple symmetric cryptographic algorithms with a peak throughput of up to 128 Gbps for AES-128. In addition to implementing encryption and decryption operations, CGRA also has some resistance to physical attacks. CGRA uses its hardware resource redundancy and dynamic real-time reconfigurable features to achieve randomization of cryptographic operations in space and time, which greatly increases the difficulty for attackers who wish to perform physical attacks, such as electromagnetic attacks and fault attacks. This is especially important for the security of cryptographic algorithms, and it also reflects the superiority of CGRA software-defined hardware.

5.3. Multimedia

Multimedia (e.g, voice, image and video) usually need to code or decode abundant data. They are typical stream processing applications that deal with different data in the same way. These applications contain plenty of parallel calculations on macro blocks. CGRA performs well in stream processing because of its "switching configurations to adapt the application" and "one-time configuration, multi-time execution" features. There are a large number of CGRA structures for this type of application. For example, the classic ADRES has been applied to video processing (H.264/AVC decoding[60]), image processing[61]. XPP-III and REMUS are also applied to video processing (MPEG4 and H.264/AVC decoding[62]). Samsung applied a CGRA video processing platform for 8K Ultra HD TV[63].

5.4. Signal processing

CGRAs can also be used in the field of signal processing where the most important algorithms are fast Fourier transform (FFT) and inverse FFT. For instance, ADRES has been applied to software-defined radio (SDR) signal processing (SDM-OFDM) receivers[64] and MIMO SDM-OFDM baseband processing[65]. FLEXDET, proposed in Ref. [66], is a multi-mode MIMO detector based on reconfigurable processing. It is reported that CGRAs can get a higher throughput than conventional digital signal processors (DSPs) because of more powerful computing resources and larger bandwidth interface.

6. Conclusion

CGRA is the main form of dynamic reconfigurable comput-

ing fabric. This paper surveys the important aspects of CGRA, including the concept, architecture, compilation, existing challenges, and prospective applications. However, CGRA is not as mature as FPGA and still has some challenges to overcome. Since CGRA is superior in energy efficiency, area efficiency and flexibility, and does well in several important application domains, it is predicted that CGRA will become an alternative to some existing computing architectures.

Acknowledgments

This work is supported in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China (Grant No. 2018ZX01028201), and in part by the National Natural Science Foundation of China (Grant No. 61672317, No. 61834002), and in part by the National Key R&D Program of China (Grant No. 2018YFB2202101).

References

[1] Prabhakar R, Zhang Y, Koeplinger D, et al. Plasticine: a reconfigurable architecture for parallel paterns. ACM/IEEE International Symposium on Computer Architecture, 2017, 389
[2] Nowatzki T, Gangadhar V, Ardalani N, et al. Stream-dataflow acceleration. ACM/IEEE International Symposium on Computer Architecture, 2017, 416
[3] Nicol C. A coarse grain reconfigurable array (CGRA) for statically scheduled data flow computing. Wave Computing White Paper, 2017
[4] https://www.darpa.mil/
[5] Kim S, Park Y H, Kim J, et al. Flexible video processing platform for 8K UHD TV. Hot Chips 27 Symposium, 2016, 1
[6] http://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5430/
[7] PACT. http://www.pactxpp.com/
[8] https://newsroom.intel.com/news-releases/intel-tsinghua-university-and-montage-technology-collaborate-to-bring-indigenous-data-center-solutions-to-china/
[9] Suzuki M, Hasegawa Y, Yamada Y, et al. Stream applications on the dynamically reconfigurable processor. IEEE International Conference on Field-Programmable Technology, 2004, 137
[10] Sato T, Watanabe H, Shiba K. Implementation of dynamically reconfigurable processor DAPDNA-2. IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT), 2005, 323
[11] Horowitz M. Computing's energy problem (and what we can do about it). IEEE International Solid-state Circuits Conference (ISSCC), 2014, 10
[12] Tessier R, Pocek K L, Dehon A. Reconfigurable computing architectures. Proc IEEE, 2015, 103(3), 332
[13] Wijtvliet M, Waeijen L, Corporaal H. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification. 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016, 235
[14] Nowatzki T, Gangadhar V, Sankaralingam K, et al. Pushing the limits of accelerator efficiency while retaining programmability. IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016, 27
[15] Duranton M, Bosschere K D, Gamrat C, et al. The HiPEAC vision. European Network of Excellence on High Performance and Embedded Architecture and Compilation, 2017, 12
[16] Estrin G. Organization of computer systems—the fixed plus variable structure computer. Proceeding of Western Joint Computer Conference, 1960, 33
[17] Hartenstein R W, Hirschbiel A G, Riedmuller M, et al. A novel ASIC design approach based on a new machine paradigm. IEEE J Solid-State Circuits, 1991, 26(7), 975
[18] Chen D C, Rabaey J M. A reconfigurable multiprocessor IC for rapid prototyping of algorithmic-specific high-speed DSP data paths. IEEE J Solid-State Circuits, 1994, 27(12), 1895
[19] DeHon A, Wawrzynek J. Reconfigurable computing: what, why and implications for design automation. Proceeding of 36th ACM/IEEE Conference on Design Automation, 1999, 610
[20] Xilinx Inc. All Programmable FPGAs [EB/OL]. http://www.xilinx.com/products/silicondevice/fpga/index.html [2014-7-7]
[21] http://sigarch.hosting.acm.org/2015/01/17/call-for-proposals-intel-altera-heterogeneous-architecture-research-platform-program/
[22] Wingbermuehle J G, Cytron R K, Chamberlain R D. Superoptimized memory subsystems for streaming applications. International Symposium on Field-Programmable Gate Arrays, 2015
[23] Putnam A, Jan G, Michael G, et al. A reconfigurable fabric for accelerating large-scale datacenter services. IEEE Micro, 2015, 35(3), 10
[24] Goren S, Turk Y, Ozkurt O, et al. Achieving modular dynamic partial reconfiguration with a difference- based flow. Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2013, 270
[25] Coole J, G Stitt G. Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing. The eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2010
[26] Singh H, Lee M, Lu G, et al. MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications. IEEE Trans Comput, 2000, 49(5), 465
[27] Mei B, Vernalde S, Verkest D, et al. ADRES: an architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. International Conference on Field Programmable Logic and Application (FPL), 2003, 61
[28] Baumgarte V, Ehlers G, May F, et al. PACT XPP—A self-reconfigurable data processing architecture. J Supercomput, 2003, 26(2), 167
[29] Liu L, Deng C, Wang D, et al. An energy-efficient coarse-grained dynamically reconfigurable fabric for multiple-standard video decoding applications. IEEE Custom Integrated Circuits Conference, 2013, 1
[30] Chin S A, Sakamoto N, Rui A, et al. CGRA-ME: A unified framework for CGRA modelling and exploration. IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2017, 184
[31] Akbari O, Kamal M, Afzali-Kusha A, et al. PX-CGRA: Polymorphic approximate coarse-grained reconfigurable architecture. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, 413
[32] Duch L, Basu S, Pe O M, et al. i-DPs CGRA: an interleaved-datapaths reconfigurable accelerator for embedded bio-signal processing. IEEE Embed Syst Lett, 2019, 11, 50
[33] Voitsechov D, Port O, Etsion Y. Inter-thread communication in multithreaded, reconfigurable coarse-grain arrays. IEEE/ACM International Symposium on Microarchitecture (MICRO), 2018, 42
[34] Amano H. A survey on dynamically reconfigurable processors. IEICE Trans Commun, 2006, 89(12), 3179
[35] Zain-ul-Abdin, Svensson B. Evolution in architectures and programming methodologies of coarse-grained reconfigurable computing. Microprocess Microsyst, 2009, 22(3), 161
[36] Dehon A. Fundamental underpinnings of reconfigurable computing architectures. Proc IEEE, 2015, 103(3), 355
[37] Chattopadhyay A. Ingredients of adaptability: a survey of reconfigurable processors. VLSI Design, 2013, 10
[38] Wang Y, Liu L, Yin S, et al. Hierarchical representation of on-chip context to reduce reconfiguration time and implementation area

for coarse-grained reconfigurable architecture. Sci Chin Inform Sci, 2013, 56(11), 1

[39] Liu L, Li Z, Yang C, et al. HReA: an energy-efficient embedded dynamically reconfigurable fabric for 13-dwarfs processing. IEEE Trans Circuits Syst II, 2017, 65(3), 381

[40] Asanovic K, Bodik R, Catanzaro B C, et al. The landscape of parallel computing research: A view from Berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006

[41] Yin C Y, Yin S Y, Liu L B, et al. Front end design of task compiler for reconfigurable multimedia processor. J Beijing Univ Posts Telecommun, 2011, 34, 108

[42] Li Y, Callahan T, Darnell E, et al. Hardware-software co-design of embedded reconfigurable architectures. Proceedings of Design Automation Conference, 2000, 507

[43] So B, Hall M W. Increasing the applicability of scalar replacement. Proceedings of the ACM Symposium on Compiler Construction, 2004, 185

[44] Beletska A, Bielecki W, Cohen A, et al. Coarse-grained loop parallelization: iteration space slicing vs affine transformations. Paral Comput, 2011, 37, 479

[45] Jiang Y C, Wang J F. Temporal Partitioning data flow graph for dynamically reconfigurable computing. IEEE Trans VLSI Syst, 2007, 15, 1351

[46] Yin C, Yin S, Liu L, et al. Temporal partitioning algorithm for a coarse-grained reconfigurable computing architecture. International Symposium of Integrated Circuit, 2009, 659

[47] Aslam N, Milward M, Erdogan A, et al. Code compression and decompression for coarse-grain reconfigurable architectures. IEEE Trans VLSI Syst, 2008, 16, 1596

[48] Yin S, Yin C, Liu L, et al. Configuration context reduction for coarse-grained reconfigurable architecture. IEICE Trans Inform Syst, 2012, E95-D, 335

[49] Li Z, Liu L, Deng Y, et al. Aggressive pipelining of irregular applications on reconfigurable hardware. ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, 575

[50] Liu D, Yin S, Liu L, et al. Polyhedral model based mapping optimization of loop nests for CGRAs. 50th ACM/EDAC/IEEE Design Automation Conference, 2013, 1

[51] Zhu J, Liu L, Yin S, et al. A hybrid reconfigurable architecture and design methods aiming at control-intensive kernels. IEEE Trans VLSI Syst, 2015, 23(9), 1700

[52] Liu L, Wang J, Zhu J, et al. TLIA: Efficient reconfigurable architecture for control-intensive kernels with triggered-long-instructions. IEEE Trans Paral Distrib Syst, 2016, 27(7), 1

[53] Wang Y, Liu L, Yin S, 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 VLSI Syst, 2014, 22(5), 983

[54] Yang C, Liu L, Luo K, et al. CIACP: a correlation-and iteration-aware cache partitioning mechanism to improve performance of multiple coarse-grained reconfigurable arrays. IEEE Trans Paral Distrib Syst, 2016, 27(99), 1

[55] Chen Y H, Krishna T, Emer J S, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE J Solid-State Circuits, 2017, 52(1), 127

[56] Farabet C, Martini B, Corda B, et al. NeuFlow: A runtime reconfigurable dataflow processor for vision. Computer Vision and Pattern Recognition Workshops, 2011, 109

[57] Yin S, Ouyang P, Tang S, et al. 0.6-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications. Symposium on VLSI Circuits, 2017, C26

[58] Fronte D, Perez A, Payrat E. Celator: a multi-algorithm cryptographic Co-processor. International Conference on Reconfigurable Computing and FPGAs, 2008, 438

[59] Sayilar G, Chiou D. Cryptoraptor: High throughput reconfigurable cryptographic processor. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2014, 155

[60] Mei B, VeredaS F J, Masschelein B. Mapping an H.264/AVC decoder onto the ADRES reconfigurable architecture. International Conference on Field Programmable Logic and Applications, 2005, 622

[61] Hartmann M, Pantazis V, Aa T V, et al. Still image processing on coarse-grained reconfigurable array architectures. J Sign Proces Syst, 2010, 60(2), 225

[62] Ganesan M K A, Singh S, May F, et al. H.264 decoder at HD resolution on a coarse grain dynamically reconfigurable architecture. International Conference on Field Programmable Logic and Applications, 2007, 467

[63] Kim S, Park Y H, Kim J, et al. Flexible video processing platform for 8K UHD TV. Hot Chips 27 Symposium, 2016, 1-1

[64] Novo D, Moffat W, Derudder V, et al. Mapping a multiple antenna SDM-OFDM receiver on the ADRES coarse-grained reconfigurable processor. IEEE Workshop on Signal Processing Systems Design and Implementation, 2005, 473

[65] Palkovic M, Cappelle H, Glassee M, et al. Mapping of 40 MHz MIMO SDM-OFDM baseband processing on multi-processor SDR platform. IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2008, 1

[66] Chen X, Minwegen A, Hassan Y, et al. FLEXDET: flexible, efficient multi-mode mimo detection using reconfigurable ASIP. IEEE International Symposium on Field-Programmable Custom Computing Machines, 2012, 69