Volume : 2 | Issue : 12 | December 2013 • ISSN No 2277 - 8179

Research Paper

Design and Applications of Coarse-Grained Reconfigurable Architectures

Engineering

KEYWORDS : High level synthesis, Parallelizing compiler, Design automation, Reconfigurable architectures.

Shaik Rizwan Hussain — Department of Electronics & Communication Engineering,M.Tech Scholar of Madina Engg College Kadapa, INDIA

Syed Jahangir Badashah — Assoc Professor in ECE Dept, Madina Engg College Kadapa,INDIA

ABSTRACT

The increasing requirements for more flexibility and higher performance have drawn attention on various coarse-grained reconfigurable architectures. In this paper, an approach to mapping coarse-grained reconfigurable architectures application supports both floating-point arithmetic and integer. It consists of a reconfigurable array of processing elements and their applications have been embedded systems Design, reconfigurable computing is becoming more popular. It has different target domains of applications and tradeoffs between flexibility and performance. Typically, they restricted to domains based on integer arithmetic since typical CGRAs support only integer arithmetic or logical Operations. We introduce how to approach the mapping Applications onto CGRAs supporting both integer and floating-point Arithmetic. After presenting an optimal formulation using Integer linear programming.

I.INTRODUCTION

Coarse-grained reconfigurable architectures (CGRAs) are capable of achieving both goals of high performance and flexibility. Figure 1 CGRAs not only improve performance by exploiting the features of repetitive computations, but also can adapt to diverse computations by dynamically changing configurations of an array of its internal processing elements (PEs) and their interconnections. Many CGRAs have been developed recently as programmable coprocessors, minimizing the overhead of the central processor in many computation-intensive applications. They act as co-processors for accelerating computation-intensive portions of embedded system applications. System designers are attracted to CGRAs because they bridge the gap between Application Specific Integrated Circuits (ASICs) and microprocessors by providing the high performance of ASICs with flexibility of reconfiguration of fine-grained FPGAs. Some of the application areas of these architectures are image processing, Digital Signal Processing, encryption, pattern recognition, and other multimedia applications. This paper surveys the methods of compiling applications to coarse-grained reconfigurable architectures.

The idea driving reconfigurable computing is to avoid the von Neumann bottleneck (the bandwidth limitation between processor and memory) through direct computation mapping into hardware. These systems are also capable of dynamic changing of hardware logic, which it implements. Thus, an application can be partitioned temporarily for execution on the hardware which enables to execute more hardware than the existing gates to fit.

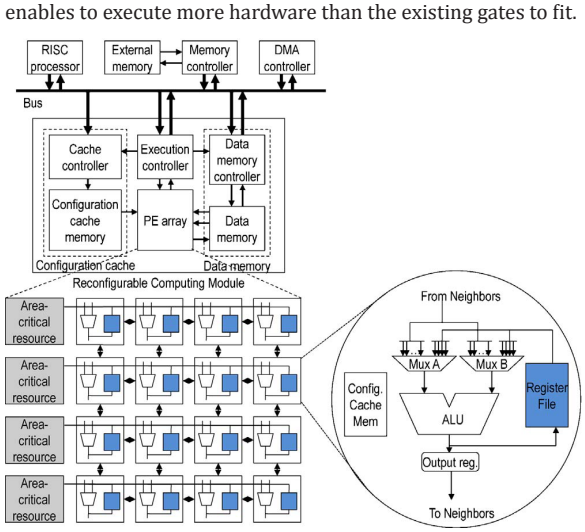

Fig 1.CGRA containing a 4*4 reconfigurable array of PEs

Reconfigurable computing refers to systems

Incorporating some form of hardware programmability customizing how the hardware is used using a number of physical control points. These control points can then be changed periodically in order to execute different applications using the same hardware. Over the last one and a half decade, the rapid growth of computer architecture and microprocessor has brought about a radical growth in both circuit densities and speed of VLSI systems. Some computation-intensive applications, previously feasible only on supercomputers, are presently feasible on workstations and PCs.

The principle attributes of these reconfigurable architectures are the capability of dynamic mapping of a portion of a program to the hardware to exploit the implicit data parallelism in the program. Field Programmable Gate Arrays (FPGAs), which are the most widely used reconfigurable hardware, are more capable of exploiting the inherent parallelism than traditional processors. So naturally for some applications FPGA-based reconfigurable architectures perform much better than processor-based alternatives. However, for applications where the data path is coarse-grained, the performance and power consumptions on FPGAs are handled inefficiently. To overcome these disadvantages, many coarse-grained or ALU-based reconfigurable systems have been proposed as an alternative between FPGA based systems and fixed logic CPUs. Coarse-grained reconfigurable architectures provide massive parallelism, high computational capability and they can be configured dynamically, making them the most attractive in the years to come especially in embedded system design.

II.LITERATURE SURVEY

Hartej Singh, Ming-Hau Lee, Guangming Lu "Morphosys-"A Reconfigurable Architecture for Multimedia Applications" IEEE transactions on computer-aided design of integrated circuits and systems, vol. 49, no. 5. May 2000,[1] The Morphosys reconfigurable system, which combines a reconfigurable array of processor cells with a RISC processor core and a high bandwidth memory interface unit. Introduce the array architecture, its configuration memory, inter-connection network, role of the control processor and related components. Architecture implementation is described in brief and the efficacy of Morphosys is demonstrated through simulation of video compression and target-recognition applications. It is not easy to map an application onto the reconfigurable array because of the high complexity of the problem.

T. J. Callahan and J. Wawrzynek "Instruction-level Parallelism for Reconfigurable computing," IEEE transactions on computer-aided design of integrated circuits and systems, vol. 13, no. 7. Sep 1998[5]. Reconfigurable coprocessors can exploit large degrees of instruction-level parallelism (ILP). In compiling for reconfigurable coprocessors, we have found it convenient to borrow techniques previously developed for exploiting ILP for very long instruction word (VLIW) processors, specifically, hyper-

Volume : 2 | Issue : 12 | December 2013 • ISSN No 2277 - 8179

block formation and scheduling. With some minor adaptations, these techniques are a natural match for automatic compilation to a reconfigurable coprocessor. This paper reviews these techniques in their original context, describes how we have adapted them for reconfigurable computing, and presents some preliminary results on compiling actual application programs written in the ANSI C programming language.It had been failed to fully utilize the resources in CGRAs.

Jonghee W. Yoon, Aviral Shrivastava, Sanghyun Park, Minwook Ahn, and Yunheung Paek "A Graph Drawing Based Spatial Mapping Algorithm for Coarse-Grained Reconfigurable Architectures," IEEE transactions on very large scale integration (vlsi) systems, vol. 17, no. 11, november 2009.[11].Many CGRAs have demonstrated impressive performance improvements, the effectiveness of CGRA platforms ultimately hinges on the compiler. Existing CGRA compilers do not model the details of the CGRA, and thus they are i) unable to map applications, even though a mapping exists, and ii) using too many processing elements (PEs) to map an application. The CGRA details, e.g., irregular CGRA topologies, shared resources and routing PEs in our compiler and develop a graph drawing based approach, Split-Push Kernel Mapping (SPKM), for mapping applications onto CGRAs. The main advantage of spatial mapping is that each PE may not need reconfiguration during execution of a loop because of its fixed functionality. It has a disadvantage that spreading all operations of the loop body over the reconfigurable array may require a very large array size.

T. Toi, N. Nakamura, Y. Kato, T. Awashima, K. Wakabayashi, and L. Jing, "High-Level Synthesis Challenges and Solutions for A Dynamically Reconfigurable Processor," IEEE transactions on very large scale integration (vlsi) systems, vol. 19, no. 9, nov. 2006 [8].A dynamically reconfigurable processor (DRP) is designed to achieve high area efficiency by switching reconfigurable data paths dynamically. Our DRP architecture has a standalone finite state machine and that switches "contexts" consisting of many operational and storage units in processing elements (PEs) and wires between them. Utilizing the resources not only in two spatial dimensions but also vertically (time-multiplexed) under accurate timing and area constraints imposes challenges for a high-level synthesizer for the DRP. This is achieved by including the overhead of selectors in the scheduling algorithm, and considering a wire delay at each PE level. A new technique is introduced to achieve high area efficiency. It works by effectively allocating multiple steps into the context. From the original high-level synthesizer for application-specific integrated circuits, some of the basic rules such as operator and register sharing were completely changed due to the coarse grained and multi-context architecture.They introduced a compiler to exploit the parallelism in the CGRA provided by the abundant resources. However, their approaches use shared registers to solve the mapping problem.

III.PROPOSED SYSTEM

Typically, the approach takes hundreds of minutes whereas our approach takes only a few minutes. Furthermore, all previous mapping/compiling/synthesis tools have been restricted to integer-type application domains figure 2, whereas ours extends the coverage to floating-point-type application domains. As floating-point applications such as 3-D graphics become more prevalent, acceleration of floating-point operations become more important. It has introduced floating-point behavioral synthesis to provide an optimized floating-point functional library giving tradeoffs between performance and area . as shown in figure 3.

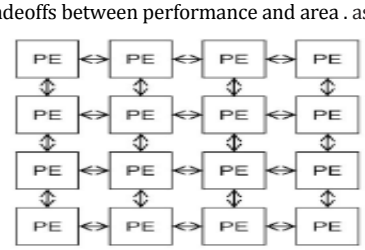

Figure 2: Integer PE Array (4 x 4).

- Our target architecture consists of a reconfigurable computing module (RCM) for executing loop kernel code segments and a general-purpose processor for controlling the RCM, and these units are connected with a shared bus.
- Where the application is represented by a complete binary tree and the CGRA consists of a 2-D grid with just the neighboring connections. ILP-based application mapping yields an optimal solution.
- We consider four PEs in a column that has mesh connectivity. The approach using Steiner tree gives better solution than the one using spanning tree.
- The shared registers can be eliminated if routing resources are considered explicitly during the mapping process.

Memory can be divided into local memory in the form of register files inside each processor element and into memory banks with storage capacities in the range from hundreds to thousands of words.

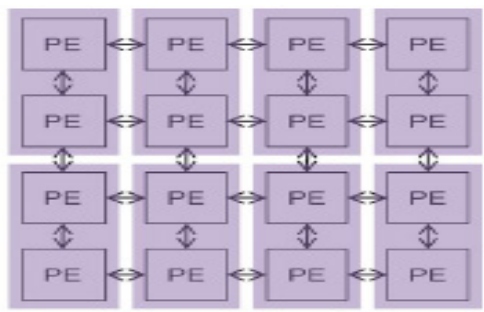

Figure 3. Floating point PE array (4×4).

The alignment and the number of such memory banks are important for the data mapping. Furthermore, knowledge of several memory modes is helpful, e.g., configuration as FIFO. Here, the structure and number of communication channels is of interest. Which type of interconnect is used, buses or point-to-point connections? How are these channels aligned, vertically, horizontally, or in both directions? How long can point-to-point connections be, without delay, or how many cycles have to be taken into account when communicating data from processor element to processor element. Additionally, similar structures are required to handle the control flow.

The maximum bandwidth is defined by the number and width of the I/O-ports. The placement of these I/O-ports is important, since they are responsible for feeding data in and out. Furthermore, it has to be considered if the I/O-port is a streaming port or an interface to external memory. In addition, possibilities of partial and dynamical reconfiguration during the execution have to be considered.

IV. METHODS AND SOLUTIONS.

Methodologies are the principles and explanations of mapping multi-domain applications onto Coarse-grained reconfigurable architectures. We have four modules. Multiplexers, ALU, Registers, Processing Element.

A. MULTIPLEXERS.

A 2n-to-1 multiplexer sends one of 2n input lines to a single output line. A multiplexer has two sets of inputs: 2n data input lines, n select lines, to pick one of the 2n data inputs. The mux output is a single bit, which is one of the 2n data inputs. A 2n-to-1 multiplexer routes one of 2n input lines to a single output line. Muxes can implement arbitrary functions. Smaller muxes can be combined to produce larger ones. It can add active-low or active-high enable inputs. As always, we use truth tables and Boolean algebra to analyze things.

B. ALU - ARITHMETIC LOGIC UNIT

In computing, an arithmetic logic unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit of a computer, and even the simplest microprocessor contains one

for purposes such as maintaining timers.

The Arithmetic Logic Unit (ALU) is essentially the heart of a CPU. This is what allows the computer to add, subtract, and to perform basic logical operations such as AND/OR. Since every computer needs to be able to do these simple functions, they are always included in a CPU.

C. REGISTERS

Actual definition of Register is "a combinational of flip-flops". Flip-flops are used as data storage elements. Such data storage can be used for storage of computer science, and such a circuit is described as sequential logic. It basically consists of several single bits "D-Type Data Latches", one for each bit (0 or 1) connected together in a serial or daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on. The data bits may be fed in or out of the register serially, i.e. one after the other from either the left or the right direction, or in parallel, i.e. all together. The number of individual data latches required to make up a single Shift Register is determined by the number of bits to be stored with the most common being 8-bits wide.

The Shift Register is used for data storage or data movement and are used in calculators or computers to store data such as two binary numbers before they are added together, or to convert the data from either a serial to parallel or parallel to serial format. The individual data latches that make up a single shift register are all driven by a common clock signal making them synchronous devices.

D.PROCESSOR ELEMENT.

Each processor element consists of resources for functional operations. This can be either one or more dedicated functional units or one or more arithmetic logic units (ALUs). In case of an ALU we have to consider if this unit is configured in advance or during a reconfiguration phase, or if this ALU can be programmed with an instruction set. In case of programmability it has to be considered if a local program is only modulo sequentially executed by a sequencer or if the instruction set includes also conditional branches.

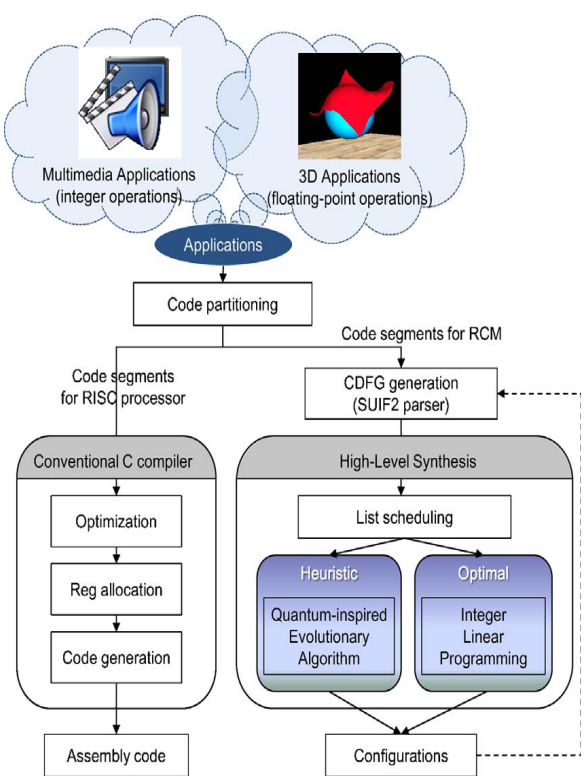

Figure 4. Overall design flow for application mapping onto CGRA

E. Mapping Operations

In mapping flow, the floating-point operations as well as integer operations. Figure.5 shows of mapping a kernel of 3-D physics engine consisting of integer and floating-point operations. Unlike normal operations, floating point operations are multicycle operations executed on a pair of integer PEs. Normally, each PE fetches a new context word from the configuration cache every cycle to execute the operation corresponding to the context word. However, if the fetched context word is for a multicycle operation such as floating-point operation, the control is passed over to the FSM.

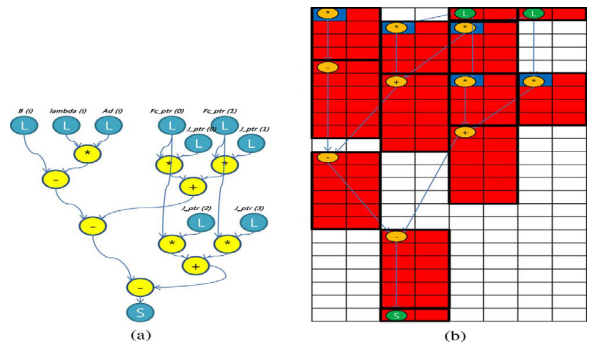

Figure 5a.Kernel part 3-D Physics engine.
Figure 5b.Mapping of the floating point operation onto CGRA.

A context word for such multicycle operation contains information about how long the operation will be executed, so that the cache control unit can wait until the current multicycle operation is finished. During the multicycle operation, the cache control unit does not send the next context words to the PEs but resumes sending the context words right after the multicycle operation is finished. The operations that a PE in our CGRA can execute are classified into the following three groups.

- Arithmetic/logical operations: A PE can execute ALU operations in one clock cycle.
- load/store operations: A PE can execute load/store operations in several clock cycles. These operations are executed by dedicated functional resources located outside the PE array. Since the functional resources are pipelined, they can start a new computation every clock cycle.
- Floating-point operations: A pair of PEs can execute floating-point operations taking several clock cycles. Both operands of a floating-point operation must be of floating-point type since we do not support mixed-type inputs or typecasting in our current CGRA implementation.

F.LOOP UNROLLING

Loop unrolling is the process of reusing the loop code to include more than one iteration of the old code, in a single pass with the new one figure 6 and figure 7. Loop unrolling works by replicating the body of a loop some number of times and scheduling the resulting code as a single basic block. Replicating the loop body has a couple of performance advantages.

- Producing a larger loop body provides a larger block of instructions for the scheduler to work with, which gives the schedule more options when positioning operations.
- Combining multiple iterations allows induction variable computations to be combined.

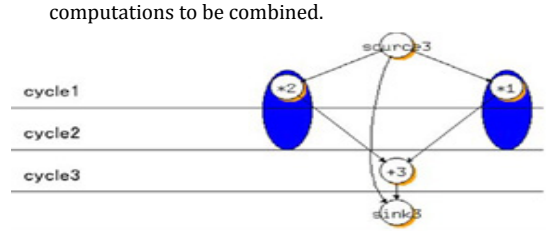

Figure 6.CDFG view.

Volume : 2 | Issue : 12 | December 2013 • ISSN No 2277 - 8179

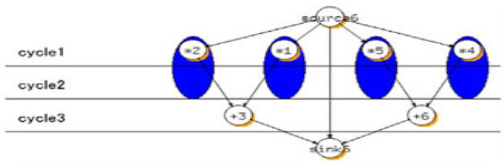

Figure 7. Loop unrolling.

These performance improvements are traded against the potential penalty caused by increased I-cache misses on the larger loop body. Loop unrolling is used to minimize stalls that may be encountered inside loops, and also to get rid of the overhead of running unnecessary branch conditionals. To keep pipeline full, parallelism among instruction must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline.

- Increases program size
- Requires more registers
- To unroll an n-iteration loop by degree k, we will need (n/k) iterations of the larger loop, followed by iterations of the original loop
- The dependences across iterations.
- If unrolling will help possible only if iterations are independent.
- Address offsets for different loads/stores.
- Dependency analysis to schedule code without introducing hazards, eliminate name dependences by using additional registers Determine loop unrolling useful by finding that loop iterations were independent.
- Determine address offsets for different loads/stores.
- Increases program size.
- Use different registers to avoid unnecessary constraints forced by using same registers for different computations Stress on registers.
- Eliminate the extra test and branch instructions and adjust the loop termination and iteration code

V.SIMULATION AND RESULTS.

VERILOG is frequently used for two different goals: simulation of electronic designs and synthesis of such designs. Synthesis is a process where a VERILOG is compiled and mapped into an implementation technology such as an FPGA or an ASIC. Many FPGA vendors have free tools to synthesize VERILOG for use with their chips, where ASIC tools are often very expensive.

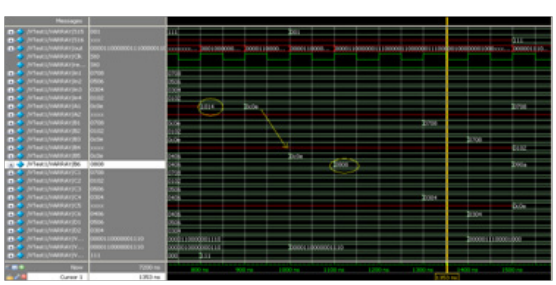

Figure 8. INTEGER PE ARRAY.

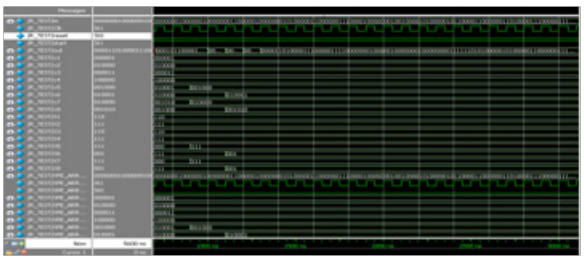

Figure 8. FLOATING PE ARRAY.

The integer PE array and floating PE array are shown in figure 7 and figure 8. For example, most constructs that explicitly deal with timing such as wait for 10 ns; are not synthesizable despite being valid for simulation. While different synthesis tools have different capabilities, there exists a common synthesizable subset of VERILOG that defines what language constructs and idioms map into common hardware for many synthesis tools.

VI.CONCLUSION AND FUTURE ENHANCEMENT.

We presented a slow but optimal approach and fast heuristic approaches to mapping of applications from multiple domains onto a CGRA supporting both integer and floating point operations. In particular, we considered Steiner tree routing since it gives better result than spanning tree routing. After presenting an ILP formulation for an optimal solution, we presented a fast heuristic approach based on HLS techniques that performs loop unrolling and pipelining, which achieves drastic performance improvement. For randomly generated test examples, we showed that the proposed heuristic approach considering Steiner points finds the optimal solutions within a few seconds. There has not been any work so far for mapping applications solely on coarse-grained reconfigurable architectures. We are working in this direction. The application will be first written using the parallel, object oriented language proposed. The source code will then be translated into parallel program graph and conventional control flow graphs. Sequential consistency will be used as the correctness criteria. Then after applying scheduling, allocation and routing on the PPG, the algorithm can be mapped to the target CGRA.

REFERENCE

[1] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E.M. C. Filho, "Morphosys: An integrated reconfigurable system for dataparallel and computation-intensive applications," IEEE Trans. Comput.,vol. 49, no. 5, pp. 465–481, May 2000. | [2] PACT XPP Technologies [Online]. Available: http://www.pactxpp.com | [3] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins,"ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in Proc. FPLA, 2003, pp. 61–70. | [4] Chameleon Systems, Inc. [Online]. Available: http://www.chameleonsystems.com | [5] T. J. Callahan and J. Wawrzynek, "Instruction-level parallelism for reconfigurable computing," in Proc. IWFPL, 1998, pp. 248–257. | [6] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S.P. Amarasinghe, "Space-time scheduling of instruction level parallelism on a RAW machine," in Proc. ASPLOSV, 1998, pp. 46–57. | [7] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins,"DRESC: A retargetable compiler for coarse-grained reconfigurable architectures," in Proc. ICFPT, Dec. 2002, pp. 166–173. | [8] T. Toi, N. Nakamura, Y. Kato, T. Awashima, K. Wakabayashi, and L.Jing, "High-level synthesis challenges and solutions for a dynamicallyreconfigurable processor," in Proc. ICCAD, Nov. 2006, pp. 702–708. | [9] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H. Kim, "Edgecentricmodulo scheduling for coarse-grained reconfigurable architectures,"in Proc. PACT, Oct. 2008, pp. 166–176. | [10] S. Friedman, A. Carroll, B. V. Essen, B. Ylvisaker, C. Ebeling, and S.Hauck, "SPR: An architecture-adaptive CGRA mapping tool," in Proc.FPGA, 2009, pp. 191–200. | [11] J. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, "A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architecture," IEEE Trans. Very Large Scale Integr. Syst., vol. 17, no.11, pp. 1565–1578, Jun. 2008. | [12] Y. Ahn, K. Han, G. Lee, H. Song, J. Yoo, and K. Choi, "SoCDAL:System-on-chip design accelerator," ACM Trans. Des. Automat. Electron.Syst., vol. 13, no. 1, pp. 171–176, Jan. 2008. | [13] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domainspecific optimization," in Proc. DATE, Mar. 2005, pp. 12–17. | [14] Y. Kim, I. Park, K. Choi, and Y. Paek, "Power-conscious configuration cache structure and code mapping for coarse-grained reconfigurable architecture," in Proc. ISLPED, Oct. 2006, pp. 310–315.