Date of publication DEC-31, 2021, date of current version OCT-27, 2021.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.20.4.2439

Design-Space Exploration of Application-specific Instruction-set Processor Design

M. H. SARGOLZAEI
School of Electrical and Computer Engineering, University of Sistan and Baluchestan, Zahedan, Iran

Corresponding author: M.H. Sargolzaei (e-mail: mh.sargolzaei@ece.usb.ac.ir).

ABSTRACT Application-Specific Instruction-Set Processors (ASIPs) have established their processing power in the embedded systems. Since energy efficiency is one of the most important challenges in this area, coarse-grained reconfigurable arrays (CGRAs) have been used in many different domains. The exclusive program execution model of the CGRAs is the key to their energy efficiency but it has some major costs. The context-switching network (CSN) is responsible for handling this unique program execution model and is also one of the most energy-hungry parts of the CGRAs. In this paper, we have proposed a new method to predict important architectural parameters of the CSN of a CGRA, such as the size of the processing elements (PEs), the topology of the CSN, and the number of configuration registers in each PE. The proposed method is based on the high-level code of the input application, and it is used to prune the design space and increase the energy efficiency of the CGRA. Based on our results, not only the size of the design space of the CSN of the CGRA is reduced to 10%, but also its performance and energy efficiency are increased by about 13% and 73%, respectively. The predicted architecture by the proposed method is over 97% closer to the best architecture of the exhaustive searching for the design space.

KEYWORDS computer architecture; high-performance computing; energy efficient processor design; coarse-grained reconfigurable array; application-specific instruction-set processor.

I. INTRODUCTION

The complexity of software applications grew exponentially in the last decades. Based on the wide range of those application domains, designing an energy-efficient and high-performance general-purpose processor is out of the question. In embedded systems, a lot of different processing architectures have been presented to increase both the processing and the energy efficiency of processing platforms. Application-Specific Instruction-Set Processors (ASIPs) are one of the most efficient processing platforms that are used in many different application domains, such as numerical and scientific computing, digital signal processing, data security, artificial intelligence, etc. [1-4].

In most application source codes of the embedded systems, the biggest part of the execution time is due to execute a small part of the code. ASIP processors, by reducing the energy consumption and execution time of that part, optimize the performance of their processing platforms. Many different architectures were proposed to improve the performance of the process-intensive part of the application, such as custom instructions extraction [5], using field-programmable gate arrays (FPGAs) or coarse-grained reconfigurable arrays (CGRAs) as a coprocessor [1] or using a graphics processing unit (GPU) to parallelize the loops [1]. CGRAs established their power in both high-performance and low-power domains [1, 6].

The efficiency of the ASIP processors is strongly related to the extracted properties of the input applications that have been extracted by application analysis strategies like application profiling [7]. The extracted properties of the input programs lead the ASIP processor to the best architectural parameters. Usually, several parameters have been indicated by those properties and some others not. In those conditions, design-space exploring is the best strategy to find the best ASIP processor architecture [8].

Not only the design-space exploration technique is a very time-consuming process but also its execution time is growing fast by the number of architectural parameters. In the past, a huge number of researchers had presented techniques to reduce the design-space exploration (DSE) time by using application properties [9].

Many different pieces of research to optimize the DSE algorithms of the ASIP processor were presented. Based on the history of the FPGA- or GPU-based ASIP processors, in this paper, we have focused on the CGRA-based ASIPs. In this paper, we propose a graph-based modeling technique (context-switching graph) to extract important properties of the input application source code to reduce the DSE searching time. By using the proposed model, the exploration time needed to find the best CGRA architectural parameters for the input group of applications can be decreased.

Our main contributions in this paper are as follows:

- Our proposed model can describe the behavior of the input application without profiling.
- Our proposed model can estimate several architectural parameters of the CGRA without compilation nor mapping the input high-level language program to the CGRA context.
- Our proposed model can improve the efficiency of the execution time and energy together with less DSE searching time.
- Our proposed model can decrease the design-space exploration time up to 80x.

The rest of the paper is organized as follows. In Section II, a short description of the basic CGRA architecture is given. An overview of the related work is reviewed in Section III. In Section IV, our proposed graph model is presented. Section V describes the proposed method to reduce the design-space exploration time. The experimental results are shown in Section VI. Finally, a conclusion of our proposed method is presented in the last section.

II. PRELIMINARIES

CGRA is an ASIP processor based on the input applications, many different parameters can be changed along the CGRA design process. Since in this section, we have presented both the CGRA architecture and the properties of the chosen benchmarks.

A. CGRA ARCHITECTURE

A lot of different architectures were proposed for the CGRAs in the last decades [6]. Based on the reconfiguration phase of CGRAs, they can be categorized into three different groups. In the first group CGRAs, the CGRA has been configured at the beginning, and then the CGRA goes to the execution phase, such as PACT XPP III [6]. The second group of CGRAs refers to those that have been reconfigured along with the execution phase. In these CGRAs, the mode of the CGRA changes periodically between execution and reconfiguration, such as REMUS [6]. The last group of CGRAs is the hybrid ones who have some statically configured parts and some dynamically reconfigured processing units, such as ADRES [6].

Generally, the area cost of the first category is higher than of the others, but its execution time is less. On the other hand, the flexibility of the last category is much more than the previous ones. Besides their reconfiguration strategy, all CGRAs are constructed by four independent networks that are working together: processing element (PE) network, data-transferring network (DTN), context-switching network (CSN), and controlling-network (CN), as depicted in Figure 1.

Each CGRAs can work only in the reconfiguration mode or on the execution mode. In the reconfiguration mode, the PEs of the CGRA will be reconfigured to execute a different job. To do that, the PEN, CSN, and CN should be worked together. In the execution mode, input data will be proceeded by moving along PEs. In another word, the reconfiguration mode prepares the CGRA to execution of the input algorithm on the input data.

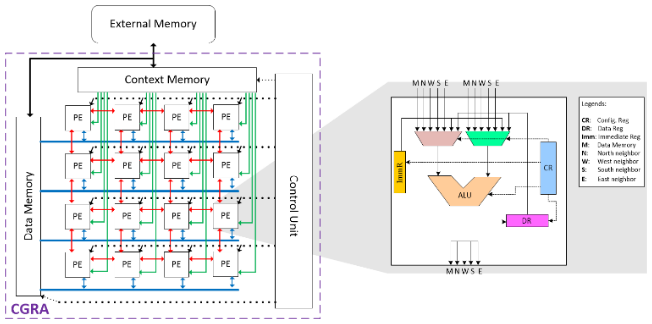


Figure 1. A generic CGRA architecture

A CGRA has a big design space with an enormous number of architectural parameters and considering all those parameters to be optimized in a project is unthinkable. Since the reconfiguration time and energy of a CGRA is strongly related to its reconfiguration strategy [6], we have selected only those who have been working in the reconfiguration mode.

Table 1 shows some important architectural parameters of a CGRA based on implemented ones [6]. The bold options in Table 1 are the fixed parameters that they do not add to the DSE process. But, lines with no bold option present the important parameters that should be added to the design space of the CGRA. Based on Table 1, the size of the PEN, the topology of the CSN, and the number of configuration registers in each PE will be considered in the DSE process of the CGRA.

Table 1. Design-space of a CGRA

	Parameters	Options
PEN	Execution model	**Single-cycle** / Multicycle / Pipelined
	Instruction-Set	**RISC** / CISC
	Granularity	**Atomic** / Clustered
	Similarity	**Homogeneous** / Heterogeneous
	Topology	NoC / **Mesh** / Bus / Crossbar / Fully connected / Hybrid
	Size	4×4 / 5×5 / 6×6 / 8×8 / 4×N / …
DTN	Topology	**Bus** / NoC / Fully connected / Nearest Neighbor
	Granularity	**Central** / Distributed
	Latency	**Single-cycle** / Fixed multicycle / Variable
CSN	Topology	Fully connected / Bus / Nearest Neighbor
	Reconfiguration model	Static / **Dynamic**
	Granularity	**Reg in each PE** / Cache in each cluster / Central memory
	Number of CRs	1, 2, 4, 8
CN	Topology	**Fully connected** / Bus
	Flow control	**Token passing**

B. SELECTED BENCHMARKS

Since the efficiency of the CGRAs is strongly related to the group of applications, we have selected 20 different applications in four groups to test the efficiency of our proposed methods in different situations. Table 2 shows the main properties of the selected benchmarks. The columns of Table 2 are the name of the benchmark suite, its domain of application, the number of selected benchmarks from that suite, the number of the different basic blocks (BBs), the number of memory accesses (MAs) in the innermost loop, and the number of different contexts needed to map the 3AC format [10] of the applications of that group onto the 4×4 CGRA architecture respectively.

Table 2. The selected benchmarks

Name	Domain	Apps	BBs	MAs	Contexts
Livermore Loops	Math.	7	23	33	34
Array Sorting	Memory	5	29	18	31
Image Processing	Matrix	3	9	27	21
BDTi Kernels	DSP	5	13	18	22
		Sum:	74	98	108

The first group is the Livermore Loops benchmark suite [11]. These applications have a small number of branches and memory accesses, but they have a lot of computations. The second group is a set of five well-known sorting algorithms [12]. These applications do not have many mathematical operations, but they have a lot of memory accesses and conditional branches. The third group of applications is three well-known image processing kernels [13]. These applications not only have a lot of mathematical operations but also have many memory accesses. The fourth group of applications is selected from the BDTi kernels [14]. They have a little of all (mathematical operations, memory access, and branches).

III. RELATED WORKS

Based on the big design space of a CGRA, finding the most fitting parameters to its input application is an NP-complete problem. Hence, researchers focused only on one of the CGRA networks. Previous researches on the efficiency of the CGRA can be divided into two main groups: DSE methods and optimizing only one particular component of it.

The design-space exploration works generally are proposed for a particular CGRA for finding the best tradeoff. However, there are few papers ([15-17]) dealing with research of a specific component of the CGRA, and their results can be used for other CGRAs. In [15], authors focused on the CN and they showed that the token-passing protocol is the best flow control protocol for the CGRAs. In [16], they searched the PEN design space to find the best topology for the PEN. They applied different topologies to the ADRES architecture. In [17], they proposed a new ultra-low-power method for near-sensor CGRAs. They also did run an exhaustive search on the size of the PEN to find the best performance and energy of the CGRA. Since the size of the PEN has a major impact on the performance and the energy of the CGRA, finding the best size of the PEN has always been an important concern in the design-space exploration ([16, 18, 19]).

In the second group of research works, optimizing a CGRA component is used as a solution to enhance the energy efficiency and performance of the CGRAs. Some researchers focused on the DTN of the CGRA ([20-22]). In [20], they proposed a hierarchical memory architecture to reduce the energy of the DTN. A reconfigurable DM network is proposed in [21] to decrease the energy. In [22], they used non-unified memory access (NUMA) architecture for the DTN and they proposed a new mapping algorithm to increase the performance of the CGRA by mapping the memory accessing PEs near the data memory unit.

Optimizing the CSN is a popular way to improve the efficiency of CGRAs. Chung and et al ([23]) coded the most frequent patterns in the contexts with a small code and stored it in a register inside the PE (configuration word). By using that technique, they succeeded to reduce the number of data transactions on the CSN. In [24, 25], the configuration words were reordered to make groups of similar configuration words and stored only one of them in the context memory using an address translator to find the reordered configuration words. In [2], the size of the context memory was reduced by eliminating the duplicate configuration words of the contexts. In [19, 26], they proposed a dynamic decompression method to reduce the number of configuration bits that are stored/transferred to/from the context memory.

In [27], they proposed a new method to reduce the energy of the context switching process by decreasing the number of active lines of the context memory and also reducing the number of transition bits on the CSN. In [28], they used the differential loading technique to reduce the number of transition bits on the CSN. But, Kim and Mahapatra [29, 30] neither stored nor transmitted the unnecessary bits of the configuration words to reduce the energy of the CGRA.

In all aforementioned methods, the efficiency of the CGRA was improved by affecting one or two architectural parameters of the CGRA. In this paper, we propose a new method to find a good tradeoff based on three major effective parameters of the CGRA: the granularity of the CSN, the size of the PEN, and the best number of the configuration registers in each PE. The proposed method is based on a graph model of the input application, called context-switching graph.

IV. CONTEXT-SWITCHING GRAPH

The context-switching graph models the behavior of the input application in the CS process based on the standard three-address code (3AC) format of the input program [27]. We have described the context-switching graph based on the control flow graph (CFG) of the program because extracting the CFG is well known. Using the context-switching graph is the key to predict the number of different contexts of the input program based on the context-switching graph of the input program without compiling it.

Suppose a simple program (Figure 2.a) and its 3AC format (Figure 2.b) which it has to be compiled and mapped onto a CGRA. The CFG of the input program can be easily drawn by mapping the basic blocks (BBs) to the nodes of the graph and their relations to the edges. If each BB is compiled into a single context, each edge of the graph indicates the next BB which has to be fetched in the context-switching process. In the context-switching graph, each node contains four different sets information about the BB: ID, pipelined or straight, number of instructions, and number of the memory accesses. The ID field is the unique number of the BB. If the BB contains a whole loop (like BB2), it is supposed to be a pipelined BB and takes “Y” in the “is pipelined?” field otherwise it takes “N”. Both numbers of instructions and memory accesses are calculated based on the 3AC format of the program. The final context-switching graph is shown in Figure 2.c.

As it is shown in Figure 2.c, some nodes have more than one output-edges. In many cases, the probability of using one of those output-edges is higher than others, such as loops, as they have been marked by a star in Figure 2.c. The marked edges are predicting the next fetching context to the CGRA. We have used the predicted next fetching context as the successor context in the next section to optimize the context-switching of the CGRA.

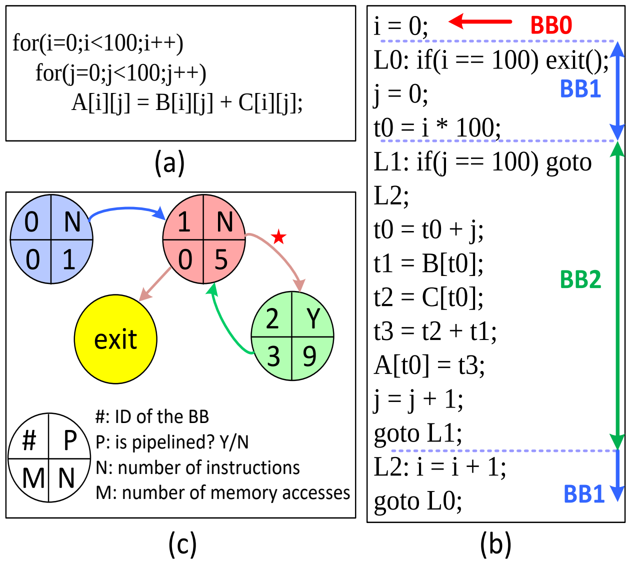
V. PRUNING THE DESIGN-SPACE

To prune the design space of the CGRA, its important architectural parameters should be modeled by some high-level factors that are extracted from the input application program. Predicting the number of different contexts based on the context-switching graph is the first step.

A. PREDICTING THE NUMBER OF DIFFERENT CONTEXTS

The number of different contexts has a strong impact on the number of context-switching processes, and consequently, it has a great effect on both the energy and performance of the CGRA. To the best of our knowledge, there is no method to predict the number of contexts of an input program without going through its compilation.

In the 3AC format of the input program, the BBs are separated with branches, and each branch is equal to a context switching process. So, the bottom limit of the number of contexts of an input program is equal to the number of its BBs. In addition to those, there are three main reasons to break a BB into more than one context: a big number of instructions, too many memory accesses, and complex variable usages.



```
for(i=0;i<100;i++)
   for(j=0;j<100;j++)
      A[i][j] = B[i][j] + C[i][j];
```

(a)

```
i = 0;                      BB0
L0: if(i == 100) exit();
j = 0;                      BB1
t0 = i * 100;
L1: if(j == 100) goto
L2;
t0 = t0 + j;
t1 = B[t0];
t2 = C[t0];                 BB2
t3 = t2 + t1;
A[t0] = t3;
j = j + 1;
goto L1;
L2: i = i + 1;
goto L0;                    BB1
```

(b)

Figure 2. Input high-level program

We have predicted the number of different contexts (C_i) of each basic block (BB_i) based on the size of the PEN (S), the number of the instructions (N_i), and memory accesses (M_i) of that basic block. Finding the exact value of the C_i is generally impossible because it is related to many other parameters which are not in the scope of this paper; for example, the ISA of the CGRA, the efficiency of the compiler, the programming model, etc. However, we can indicate the lower bound of the C_i based on N_i, S, and M_i as it is shown in equations (1)-(3).

Each CGRA has S×S PEs in its PEN. Hence, if the number of instructions of the input basic block (N_i) is more than the number of PEs, it will have to be fragmented into multiple contexts. By fragmenting a BB into multiple contexts, one branching instruction (per context) will be added to the N_i. On the other hand, based on the efficiency of the CGRA compiler, some PEs in the PEN cannot be mapped and they will be idle [28]. Thus, we have added a new parameter for that effect, called α. Equation (1) calculates the number of different contexts for a BB based on its number of instructions

$$\mathrm{C1_i} = \left\lceil \frac{\mathrm{N} + \mathrm{C1_i}}{\mathrm{S} \times \mathrm{S} \times \alpha} \right\rceil. \quad (1)$$

On the other hand, based on the topology of the DTN, if the number of the memory accesses of a BB (M_i) is greater than the size of the PE array (S), the basic block will be broken into more than one context. Equation (2) calculates the minimum number of the required contexts based on the number of the memory accesses in the B:

$$\mathrm{C2_i} = \left\lceil \frac{\mathrm{M_i}}{\mathrm{S}} \right\rceil. \quad (2)$$

The lower bound of the number of different contexts of an input basic block (C_i) should be greater than or equal to the maximum value of the $C1_i$ and $C2_i$. Equation (3) calculates the lower bound of the number of different contexts for a given BB:

$$\mathrm{C_i} = \max\{C1_i, C2_i\}. \quad (3)$$

As mentioned before, there are three reasons to break a BB into more than one context. We have modeled the first two parameters, but the third one, the complex variable usage, cannot be modeled by equations.

To validate the predicted C_i, we have compiled all applications on all CGRA sizes in our design space and found the smallest possible "C_i" manually. However, between 20 chosen applications, only one application could not be mapped to the calculated number of contexts by Equation (3).

B. PREDICTING THE SIZE OF THE PEN

Not only, the number of PEs in the PEN has a strong impact on the number of different contexts of the input program, but also it has a great effect on almost all other architectural parameters of the CGRA. Based on the previously implemented CGRAs, as it is shown in Table 1, we have considered all sizes in the range of 4×4-8×8 for the size of the PEN in the design space (only square PENs).

To reduce the execution time and the energy of the CGRA through optimizing its CSN, we have decreased the number of fetching configuration words through the context-switching processes. Equation (4) calculates the number of the transferred configuration words (CW) based on the predicted number of different contexts for the input program:

$$CW \approx S \times S \times \sum_{i=0}^{B} C_i. \quad (4)$$

By changing the size of the PEN, a new value for the CW is calculated. By increasing S, the number of occurring context-witching processes due to execute input program and consequently the number of the transferring configuration word will be decreased. Hence, the minimum CW indicates the upper bound of the best size of the PEN. By using Equation (4), we have pruned 74% of the CGRA design-space on average.

C. PREDICTING THE NUMBER OF CRS IN EACH PE

The number of configuration registers (CRs) in each PE has a strong effect on the number of the transferred configuration words and consequently on the number of accesses to the context memory. Having enough CRs in each PE to store all configuration words of the program innermost loop can reduce the number of context-switching processes dramatically.

The number of different contexts of the innermost loop depends on the PEN size (Equations (4)). Subsequently, the number of different contexts of the innermost loop and the number of CRs can be calculated by Equation (3). The calculated value of the CRs is the upper bound of its best value in the CGRA design space. Hence, the calculated values for different benchmarks reduce the design space by about 37.3% (on average).

D. PREDICTING THE TOPOLOGY OF THE CSN

As it is shown in Table 1, there are three major topologies for the CSN of a CGRA: fully connected or Point-to-Point (P2P), bus (B), and nearest neighbor (NN). In the P2P topology, there is a dedicated channel between the context memory and each PE, so the reconfiguration phase will be completed in one clock cycle and it has the minimum overhead on the execution time for the input program. In the B topology, there is a shared bus for each row/column of the PEN, consequently, the reconfiguration phase of the B topology takes S clock cycles long. In the NN topology, the configuration words will be shifted through the CRs of the PEs in each row/column. Then, its reconfiguration phased delay is S clock cycles, i.e., the same as the B topology.

The topology of the CSN affects the energy of the CGRA by two factors: the amount of the wiring capacities and the number of the transition bits in those wires. The amount of the wiring capacities can be modeled by the number of required non-local links in the CSN. On the other hand, the number of transition bits can be modeled by the number of different CWs that will be shifted through the CSN due to the context-switching phase. Hence, for each reconfiguration process, the number of transition bits in the P2P topology can be modeled by the number of different CWs of each PE with its predecessor CW. Equation (5.a) models the energy of the CSN for the P2P topology:

$$P2P \approx \sum_{i=1}^{S\times S} D_i \times C, \quad (5.a)$$

where D_i is the number of the different bits of two following configuration words and C is the capacitance of the P2P link between the context memory and the PE. On the other hand, when the topology of the CSN is B or NN, the configuration words should be sequentially shifted through the bus or the PEs. Equation (5.b) models the energy of the CSN for the B and the NN topologies:

$$BNN \approx \sum_{i=1}^{S\times S} i \times D_i \times C, \quad (5.b)$$

where D_i is the number of the different bits of two adjacent PEs in a row/column. In the B topology, the CWs are shifted via a link and C is the capacity of that link, but in the NN topology, C is the capacity of the reconfiguration part of each PE. Predicting the topology of the CSN reduces the size of the design space by about 35% (on average).

E. PRUNING THE DESIGN-SPACE

As discussed before, the delay and the energy of the CGRA are dependent on the architectural parameter of the CGRA. We have predicted the upper/lower bound of those parameters based on the input application programs in the previous subsections. In this subsection, we will reduce the size of the design space by using the pruning method based on the predicted boundaries.

As shown in Table 1, some of the architectural parameters of the CGRA are not fixed: the PEN size (from 4×4 to 8×8), the CSN topology (P2P, B, and NN), and the number of the CRs in each PE (1 to 8). To do an exhaustive search in the design space of the CSN, 120 (5×3×8) different architectures should be simulated for each application.

As discussed in the previous section, the predicted values for the architectural parameters of the CGRA are the upper/lower bounds to the best value. By applying those boundaries to the design space, a big part of it will be pruned. Figure 3 shows the effect of the pruning process on the size of the design space.

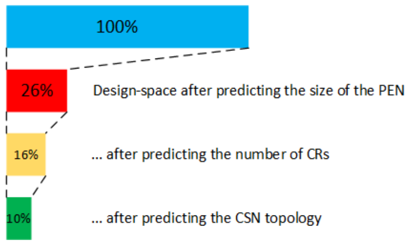


Figure 3. Effect of pruning the size of the design space.

VI. EXPERIMENTAL RESULTS

In this paper, we have not only proposed a new method to predict a near-best (or the best) value of multiple important architectural parameters of the CGRA but also, we have reduced the size of its design space by using the proposed method. In this section, the effect of the proposed method will be discussed in terms of delay and the energy of the CGRA. Then, we will prune its design space to find the best architecture for a given application without searching all the design space. Finally, a comparison in terms of the delay and the energy of the CGRA will be presented in some future works.

In this section, we have used the smallest CGRA (PEN size = 4, the number of CRs = 1, and the CSN topology= B) as the basic CGRA ($CGRA_0$) to compare the efficiency of our proposed method.

A. DELAY ANALYSIS

The execution of an application can be divided into three different phases: initial phase, reconfiguration phase, and running phase. The contexts and data of the input application should be transferred into the context and data memory of the CGRA in the initial phase. Since the initial phase is occurring only once, its run-time can be ignored. The delay of the running phase is related to the number of occupied PEs and their type (pipelined or not). Hence, it can be considered independent of the CGRA architectural parameters. On the other hand, the cost of the reconfiguration phase is the main concern of this paper.

The delay of the reconfiguration phase is strongly related to the predicted parameters. For each reconfiguration process, the delay of the reconfiguration phase is 1 (for the P2P CSN) or S (for the B and NN CSNs). The number of the required reconfiguration processes is related to the number of the different contexts, the number of loops in the CSG, and the efficiency of the compiler. Hence, we have modeled the delay of the reconfiguration phase by Equation (7).

$$Delay \approx D \times \sum_{i=0}^{B}(C_i + \sum_{j=0}^{B} L \times C_i \times C_j), \quad (7)$$

where D is the delay of each reconfiguration process, L is a binary variable for indicating the loop BBs (1 for loops and 0 for other basic blocks). Figure 4 shows the delay reduction ratio of the predicted CGRA compared with the CGRA$_0$.

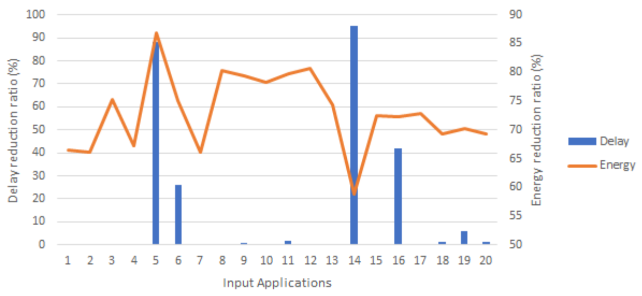


Figure 4. Delay and energy reduction ratio.

As it is shown in Figure 4, the delay in some cases with the predicted architecture is the CGRA$_0$, hence, there is no delay reduction in those cases.

B. ENERGY ANALYSIS

As it is discussed in the previous subsection, there are three different phases of a CGRA application execution process. The energy of the initial and running phases can be ignored as well as their delay. But, the energy of the reconfiguration phase of the CGRA is considered in this paper.

The energy of the reconfiguration phase of the CGRA is related to the size of the PEN (the number of PEs), the CSN topology (the number of channels in the CSN), and the number of CRs in each PE (the number of the needed reconfiguration processes). Hence, the energy of the CGRA can be modeled by Equation (8).

$$Energy \approx S \times S \times E5/CR, \qquad (8)$$

where S is the size of the PEN, E5 is the output of Equation 5 (5.a for the P2P and 5.b for the B/NN topology), and CR is the number of CRs in each PE. Based on the previously calculated parameters (by equations (3) to (5)), the minimum output of Equation (8) is the best architecture. The effect of applying the predicted parameters on the energy of the CGRA for each application is shown in Figure 4.

Based on our results, the energy of the application execution can be reduced by more than 58% (or on average 73%) compared with the CGRA$_0$ architecture.

C. THE ERROR OF THE PREDICTED ARCHITECTURE

The goal of this paper is to design an efficient CSN for the CGRA without exhaustive searching for its design space. We have proposed a model to predict the important parameters of the CSN and use them as an upper/lower bound of those parameters to prune the design space. Using a pruning algorithm to reduce the size of the design space might result in losing the best architecture, but it can find an acceptable one. Thus, the difference between the delay and the energy of the predicted architecture and the best one (who can be found by exhaustive search) should be analyzed.

A semi-best predicted architecture might increase the delay and the energy of the CGRA due to the application execution based on our results, there are only two (out of 20) mismatches between the best architecture of the exhaustive search and our predicted ones. However, the delay and the energy of the predicted CGRA is about 99% (on average, and 97% for the worse case) close to the best CGRA results.

D. MORE OPTIMIZATION WITH THE CUC METHOD

We have proposed a configuration compression method to reduce the energy of the CGRA through decrease in its reconfiguration phase costs in our previous work [27]. To gain more energy efficiency, we have applied the CUC method to the context of the input applications and examined its effect on our new proposed method. Figure 5 shows the effect of our proposed method on the energy and the delay of the CGRA w/ and w/o applying the CUC.

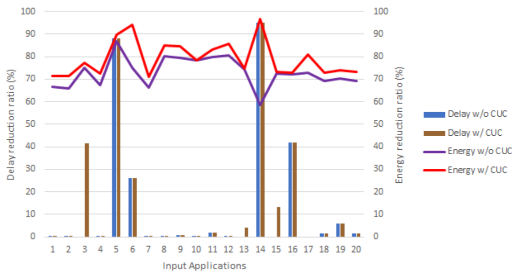


Figure 5. Comparison of the results w/ and w/o using the CUC.

As it is shown in Figure 5, using the CUC method along with our proposed prediction method improves by 3% and 6% the delay and the energy of the CGRA respectively.

E. COMPARISON WITH PREVIOUS WORKS

We have examined the efficiency of our proposed method in two terms individually in the previous subsection. However, we will compare the energy efficiency, area overhead, and performance improvement of our proposed method with some previous works. Table 3 shows the area overhead, the performance improvement, and the energy efficiency of each method. The starred numbers in the last column are for the energy efficiency of the CM module and not for the whole CGRA. The first six selected works in Table 3 are some context compression methods that are only working on a particular architecture of a CGRA. The next four chosen works are some design-space exploration works whose method can be used on the other CGRA architectures just like our proposed method. Finally, the last two lines are the results of our proposed method for the normal and the CUC version of the CGRA, respectively.

Table 3. Comparison of results with previous works.

	Area (%)	Delay (%)	Energy (%)
[29]	*10.35*	*0*	*38.5**
[25]	*2-3*	*0*	*36.85**
[27]	*10.5*	*0*	*19.34 (81*)*
[19]	*6*	*-49.07*	*17.26*
[2]	*8*	*0*	*70**
[26]	*9.7*	*0*	*50**
[30]	*395.21*	*80*	*-*
[17]	*-*	*18.6*	*22.3*
[16]	*-*	*25*	*5-20*
w/o CUC	*1.77*	*13.15*	*73.07*
w/ CUC	*7.68*	*16.1*	*79.11*

As shown in Table 3, our proposed method not only improves the energy efficiency of the CGRA but also shows a better performance. By using our proposed method, the energy consumption and the delay of the CGRA can be reduced about 73% and 13% respectively. On the other hand, by using our design-space pruning method, the design-space exploration time will be reduced more than by 90%.

VI. CONCLUSIONS

In this paper, we proposed a new method to design an energy- and delay-efficient CGRA without running an exhaustive search for the best architectural parameters of the CGRA. We modeled the 3AC format of the input program by a graph, the context-switching graph (CSG). We also predicted the different number of contexts of an application, without performing any compilation or application mapping. The best architecture of the context switching network of the CGRA was predicted based on the proposed CSG and the predicted number of contexts. Based on our proposed method, the best architecture can be found 10× (on average) faster than the exhaustive search on average, with more than 97% of its accuracy in energy prediction. Based on our results, using a context compression method can increase the efficiency of the prediction method by up to 19%. Using the context compression method improves the processing performance and the energy efficiency of the CGRA by 13% and 73% on average.

References

[1] R. Tessier, K. Pocek and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, issue 3, pp. 332-354, 2015. https://doi.org/10.1109/JPROC.2014.2386883.

[2] H. Lee, M. S. Moghaddam, D. Suh, and B. Egger, "Improving energy efficiency of coarse-grain reconfigurable arrays through modulo schedule compression/ decompression," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, issue 1, pp. 1-26, 2018. https://doi.org/10.1145/3162018.

[3] A. Palagin and V. Opanasenko, "The implementation of extended arithmetic's on FPGA-based structures," *Proceedings of the 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, vol. 2, (IDAACS'2017), 21-23 September 2017, Bucharest, Romania, pp. 1014–1019. https://doi.org/10.1109/IDAACS.2017.8095239.

[4] V. Opanasenko, A. Palahin, and S. Zavyalov, "The FPGA-based problem-oriented on-board processor," *Proceedings of the 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, vol. 1, (IDAACS'2019), 18-21 September 2019, Metz, France, pp. 152–157. https://doi.org/10.1109/IDAACS.2019.8924360.

[5] J. Choi, S. Kim and H. Han, "Accelerating loops for coarse grained reconfigurable architectures using instruction extensions," *ACM Symposium on Research in Applied Computation*, New York, USA, 21-24 Mar., 2011, pp. 314-318. https://doi.org/10.1145/2103380.2103445.

[6] H. K. Nguyen, T. V. Le-Van and X. T. Tran, "A survey on reconfigurable system-on-chips," *REV Journal on Electronics and Communications*, vol. 7, pp. 3-4, 2018. https://doi.org/10.21553/rev-jec.147.

[7] J. F. Eusse, C. Williams and R. Leupers, "CoEx: A novel profiling-based algorithm/architecture co-exploration for ASIP design," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, issue 3, pp. 1-16, 2015. https://doi.org/10.1145/2629563.

[8] K. Balasubadra, A. P. Shanthi and V. P. Srinivasan, "Hybrid design space exploration methodology for application specific system design," *International Journal of New Computer Architectures and Their Applications*, vol. 7, issue 3, pp. 102-112, 2017. https://doi.org/10.17781/P002363.

[9] J. Zhang, H. Tabkhi and G. Schirner, "DS-DSE: Domain-specific design space exploration for streaming applications," *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 19-23 March, 2018, pp. 165-170. https://doi.org/10.23919/DATE.2018.8341997.

[10] Three-address code, [Online]. Available at: https://en.wikipedia.org/wiki/Three-address_code.

[11] Livermore Loops Benchmark, [Online]. Available at: http://www.netlib.org/benchmark/livermorec.

[12] joshuakehn. Sorting Algorithms, [Online]. Available at: http://www.joshuakehn.com/2010/10/1/Sorting-Algorithms.html.

[13] Kernel (Image Processing), [Online]. Available at: https://en.wikipedia.org/ wiki/Kernel_(image_processing).

[14] BDTI DSP Kernel Benchmarks, [Online]. Available at: https://www.bdti.com/ Services/Benchmarks/DKB.

[15] H. Park, Y. Park and S. Mahlke, "Reducing control power in cgras with token flow," *Proceedings of the Workshop on Optimizations for DSP and Embedded Systems*, Seattle, USA, 22-25 Mar., 2009.

[16] A. Lambrechts, P. Raghavan, M. Jayapala, B. Mei, F. Catthoor and D. Verkest, "Interconnect exploration for energy versus performance tradeoffs for coarse grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, issue 1, pp. 151-155, 2008. https://doi.org/10.1109/TVLSI.2008.2002993.

[17] S. Das, K. J. Martin, D. Rossi, P. Coussy and L. Benini, "An energy-efficient integrated programmable array accelerator and compilation flow for near-sensor ultralow power processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, issue 6, pp. 1095-1108, 2018. https://doi.org/10.1109/TCAD.2018.2834397.

[18] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe and R. R. Taylor, "PipeRench: A reconfigurable architecture and compiler," *Computer*, vol. 33, issue 4, pp. 70-77, 2000. https://doi.org/10.1109/2.839324.

[19] B. Shehan, R. Jahr, S. Uhrig and T. Ungerer, "Reconfigurable grid alu processor: Optimization and design space exploration," *Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, Lille, France, 1-3 Sep., 2010, pp. 71-79. https://doi.org/10.1109/DSD.2010.28.

[20] Y. Wang, L. Liu, S. Yin, M. Zhu, P. Cao, J. Yang and S. Wei, "On-chip memory hierarchy in one coarse-grained reconfigurable architecture to compress memory space and to reduce reconfiguration time and data-reference time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, issue 5, pp. 983-994, 2013. https://doi.org/10.1109/TVLSI.2013.2263155.

[21] M. A. Tajammul, M. A. Shami, A. Hemani and S. Moorthi, "NoC based distributed partitionable memory system for a coarse grain reconfigurable architecture," *Proceedings of the 24th Internatioal Conference on VLSI Design*, Chennai, India, 2-7 Jan., 2011, pp. 232-237. https://doi.org/10.1109/VLSID.2011.45.

[22] T. Kojima and H. Amano, "A Fine-grained multicasting of configuration data for coarse-grained reconfigurable architectures," *IEICE Transactions on Information and Systems*, vol. 102, issue 7, pp. 1247-1256, 2019. https://doi.org/10.1587/transinf.2018EDP7336.

[23] M. K. Chung, Y. G. Cho and S. Ryu, "Efficient code compression for coarse grained reconfigurable architectures," *Proceedings of the IEEE 30th International Conference on Computer Design (ICCD)*, Montreal, Canada, 30 Sep.-3 Oct., 2012, pp. 488-489. https://doi.org/10.1109/ICCD.2012.6378687.

[24] B. Liu, W. Y. Zhu, Y. Liu and P. Cao, "A configuration compression approach for coarse-grain reconfigurable architecture for radar signal processing," *Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Shanghai, China, 13-15 Oct., 2014, pp. 448-453. https://doi.org/10.1109/CyberC.2014.83.

[25] B. Egger, H. Lee, D. Kang, M.S. Moghaddam, Y. Cho, Y. Lee, S. Kim, S. Ha and K. Choi, "A space-and energy-efficient code compression/ decompression technique for coarse-grained reconfigurable architectures," *Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Texas, USA, 4-8 Feb., 2017, pp. 197-209. https://doi.org/10.1109/CGO.2017.7863740.

[26] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi and J. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," *Proceedings of the 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Washington, USA, 10-12 July, 2017, pp. 184-189. https://doi.org/10.1109/ASAP.2017.7995277.

[27] M. H. Sargolzaei and S. Mohammadi, "Energy efficient configuration unification and compression for CGRAs," *Microprocessors and Microsystems*, vol. 62, pp. 1-11, 2018. https://doi.org/10.1016/j.micpro.2018.06.010.

[28] M. K. Chung, J. K. Kim, Y. G. Cho and S. Ryu, "Adaptive compression for instruction code of coarse grained reconfigurable architectures," *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, Kyoto, Japan, 9-11 Dec., 2013, pp. 394-397. https://doi.org/10.1109/FPT.2013.6718396.

[29] Y. Kim and R. N. Mahapatra, "Dynamic context compression for low-power coarse-grained reconfigurable architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, issue 1, pp. 15-28, 2009. https://doi.org/10.1109/TVLSI.2008.2006846.

[30] Y. Kim, "Power-efficient configuration cache structure for coarse-grained reconfigurable architecture," *Journal of Circuits, Systems and Computers*, vol. 22, issue 3, 1350001, 2013. https://doi.org/10.1142/S0218126613500011.

Mohammad Hossein Sargolzaei received his B.Sc., M.Sc. and Ph.D. degree in computer engineering from Shahid Bahonar University and University of Tehran in 2006, 2009 and 2018, respectively. He is an Assistant Professor in School of Electrical and Computer Engineering, at the University of Sistan and Baluchestan. His research interests include embedded system design, high-performance and low power VLSI design and fault-tolerant system design.