Configuration Context Reduction for Coarse-Grained Reconfigurable Architecture

Shouyi YIN†(a), Member, Chongyong YIN†, Leibo LIU†, Min ZHU†, and Shaojun WEI†, Nonmembers

SUMMARY—Coarse-grained reconfigurable architecture (CGRA) combines the performance of application-specific integrated circuits (ASICs) and the flexibility of general-purpose processors (GPPs), which is a promising solution for embedded systems. With the increasing complexity of reconfigurable resources (processing elements, routing cells, I/O blocks, etc.), the reconfiguration cost is becoming the performance bottleneck. The major reconfiguration cost comes from the frequent memory-read/write operations for transferring the configuration context from main memory to context buffer. To improve the overall performance, it is critical to reduce the amount of configuration context. In this paper, we propose a configuration context reduction method for CGRA. The proposed method exploits the structure correlation of computation tasks that are mapped onto CGRA and reduce the redundancies in configuration context. Experimental results show that the proposed method can averagely reduce the configuration context size up to 71% and speed up the execution up to 68%. The proposed method does not depend on any architectural feature and can be applied to CGRA with an arbitrary architecture.

key words: reconfigurable computing, coarse-grained reconfigurable architecture (CGRA), configuration context, structure-correlation elimination

1. Introduction

With the increasing demand for flexible and highly efficient architecture for embedded systems, there is growing interest in the coarse-grained reconfigurable architectures (CGRAs) [1]–[4]. CGRA is a promising approach to achieve the balance between flexibility of GPP (general-purpose processor) and high performance of ASICs (application specific integrated circuits). A typical CGRA consists of a reconfigurable coarse-grained processing elements (PE) array and a tiny GPP. In this hybrid architecture, the reconfigurable PE array is responsible for accelerating computation-intensive algorithms by taking advantage of massive hardware parallelism; the tiny GPP is responsible for controlling and configuring the reconfigurable hardware.

CGRA has a distinct feature of dynamic reconfiguration, which means the functionality of PE array can be dynamically configured in runtime. For dynamic reconfiguration, the configuration context is transferred from main memory to context buffer frequently. The frequent memory-read/write operations cost large reconfiguration time and much power consumption that are becoming the performance bottleneck. Furthermore, large reconfiguration time may wipe out any gains from dynamical reconfiguration. To improve the overall performance, the fundamental way is to reduce the amount of configuration context which can result in less memory-read/write operations.

How to reduce the size of configuration context has been researched during the last decade. Pan et al. [5] proposed a compression algorithm that exploits the redundancies of intra-bitstream and inter-bitstream of configuration context. M. Martina et al. [6] developed an alternative technique to compress configuration context based on the knowledge of the device internal structure. They designed a two-step coder: the configuration bitstream is adaptively filtered to remove data redundancy in the first step, while in the second step an arithmetic coder is used to actually compress the bitstream. Dynamically compressible context architecture for power saving in configuration cache can be found in [7]. This power-efficient design of context architecture works without degrading the performance and flexibility of coarse-grained reconfigurable architecture. In [8], a dictionary-based lossless context compression scheme and its corresponding decompressor hardware unit are proposed.

All above methods have a common feature that they compress context based on statistical characteristics of configuration bitstream just like traditional data compression. In the compilation flow of executable file for CGRA, this kind of methods is usually used after context generation. Thus we call these methods post-context-generation method.

However, when some tasks are mapped onto CGRA, the tasks have some inherent redundancies due to the correlation of operations and data dependency. These redundancies are inherited to configuration context. Since this kind of redundancy is not exactly the statistical redundancy, the post-context-generation method cannot eliminate it efficiently. The only way to tackle this kind of redundancy is eliminating the inherent correlations at the task-level before configuration context generation. This motivates us to design a pre-context-generation method to reduce the context size.

In this paper, we propose a configuration context reduction method which is based on eliminating the structure correlation of the mapped tasks. In the proposed method, the tasks are represented by data-flow graphs (DFGs). The fundamental of this method is generating a series of configuration templates by analyzing and reconstructing the DFGs without changing the task function. Each template represents a set of operations that can be mapped onto the PE array for one time. In other words, each template represents...
one time reconfiguration of the entire PE array. Therefore
the number of templates reflects the reconfiguration cost to
accomplish the given tasks. By eliminating the structure
correlation in DFGs, some unnecessary reconfigurations are
eliminated and some similar reconfigurations are merged.
Hence the number of templates is reduced, which results in
reducing configuration context finally. Some experimental
results show that the context redundancies caused by struc-
ture correlations is much more than statistical redundancies.
Thus the proposed method is more efficient than the post-
context-generation methods for context reduction.

The rest of this paper is organized as follows: In
Sect. 2, we describe the target CGRA architecture of the pro-
posed method. The configuration context reduction method
and its key algorithms are well described in Sect. 3. Experi-
mental results are illustrated in Sect. 4. Finally, we summa-
rize the contribution and suggest directions for future work
in Sect. 5.

2. Target Architecture

2.1 General Architecture of CGRA

The general architecture of CGRA is shown in Fig. 1. Typ-
ically, a CGRA has a tiny GPP and a reconfigurable PE ar-
ray. The tiny GPP controls the execution of the whole sys-
tem. Each PE contains functional units and a few storage
units. The configuration context buffer stores the configura-
tion context and the control unit controls the reconfiguration
and execution of PE array. The data buffer provides operand
data to PE array through a high-bandwidth data bus.

Since the proposed context reduction method does not
depend on any architectural feature, it can be applied to
CGRA with an arbitrary architecture.

2.2 REMUS and Its Context Architecture

In this paper, REMUS processor is used for verification and
evaluation of the proposed method.

The overall structure of REMUS processor is shown in
Fig. 2 (a). REMUS is a reconfigurable multimedia processor
that consists of one RISC processor (ARM11), two 16 × 16
PE arrays (PEAs), a dedicated Entropy Decoder (EnD) and
some assistant modules. For non-multimedia applications,
the EnD can be disabled and bypassed; then REMUS be-
comes a general CGRA.

The 16 × 16 PEA has a 2-level hierarchical architecture
shown in Fig. 2. The 16 × 16 PEA consists of four 8 × 8
PE arrays, Data Buffering Interface (DBI) and Fast Config-
uring Interface (FCI). The data is exchanged through DBI
between PEA and memory. The ARM11 core controls the
reconfiguration by FCI. The 8 × 8 PE array can be config-
ured independently. The schematic of the 8 × 8 PE array is
illustrated in Fig. 2 (c). It consists of a coarse-grained recon-
figurable array of 64 PEs and some assistant modules such
as DBI, FCI and temporary registers.

The architecture of a PE is shown in Fig. 3 (a). Each
PE adopts the common ALU architecture which supports

![Fig. 1](image1.png)  Block diagram of general CGRA.

![Fig. 3](image2.png)  (a) Architecture of PE, and (b) PE’s function.

![Fig. 2](image3.png)  (a) Architecture of REMUS, (b) Architecture of PEA, and (c) Architecture of PE8 × 8.
arithmetic operations and logical operations illustrated in Fig. 3 (b).

Context architecture represents the organization of context word for configuring a PE and it is closely dependent on the hardware architecture. Figure 4 shows the context architecture in REMUS processor. The length of context word is forty bits. It specifies the function of ALU (opcode), the input sources (MUX_A and MUX_B), the output direction of ALU and the temporary register for storing the computation result.

3. The Configuration Context Reduction Method

3.1 Basic Ideas

In reconfigurable computing system, the computation-intensive tasks are usually mapped onto the reconfigurable PE array to execute. The tasks are usually represented by data-flow graph (DFG) that describes the data dependencies between operations. Due to the characteristics of CGRA, the DFGs used in this paper have some features: (1) the DFGs are directed acyclic graphs (DAGs); (2) the types of operator in DFGs are limited by PE’s functions; (3) each DFG node has a level value; (4) DFG nodes in the same level can execute simultaneously (in parallel).

Since the computation tasks have some inherent operation correlations and data dependencies, the corresponding DFGs have some structure correlations. Usually, the dimension of a computation task is larger than that of PE array. Therefore, the corresponding DFG is divided to a series of sub-DFGs (kernels) that satisfy the dimension of PE array. And the kernels are mapped onto the PE array and executed sequentially to accomplish the computation task. In CGRA, the PE array supports loop-execution. If two kernels are congruent and executed consecutively, the PE array can be configured once and loops twice. For the congruent kernels, we can abstract them as a template since they have the same structure. Thus each template represents one time reconfiguration of the PE array. The structure correlations of DFG cause the configuration context redundancies. The context redundancies usually behave as there are some duplicate parts in different kernels. An example of structure correlation is illustrated in Fig. 5. If this task is mapped onto a 4×4 PE array, it will generate two different kernels (subtask 0 and subtask 1) and obtain \((7 \times 40 \text{ bits}) \times 2 = 560 \text{ bits}\) context. In Fig. 5, it can be seen that subtask 0 and subtask 1 have the same structure except the last node. If structure redundancies are eliminated, we can combine these subtasks as one kernel and finally obtain \((6 \times 40 \text{ bits}) + (2 \times 40 \text{ bits}) = 320 \text{ bits}\) context. There are 240 bits \((560 \text{ bits} - 320 \text{ bits})\) can be reduced. Thus the reconfiguration time would be much shorter than original one.

Since this kind of redundancy is caused by structure correlations of DFG, it cannot be identified and eliminated by statistical data compression. It must be tackled at the level of kernels. The fundamental idea of eliminating structure correlations is two-fold. Firstly we generate new kernels by merging some similar kernels just like the case in Fig. 5. Thus some structure correlations are eliminated. Secondly we reorder the sequence of kernel’s execution to reduce reconfiguration times. By reordering the sequence of kernel’s execution, some congruent kernels can be scheduled to execute consecutively. Therefore the PE array can be configured once and executes multi-times, which eliminates the unnecessary reconfiguration.

Based on the above discussion, we propose the configuration context reduction method. Figure 6 illustrates the basic idea and procedure of the proposed method.

For a given computation task, the corresponding DFG
can be broken down to a series independent sub-graphs (also called `atomic-DFGs`). Independent sub-graph means it has no data dependency with other sub-graphs. Then the mapping of DFG to PE array is equivalent to constructing a series of kernels by the `atomic-DFGs`. Since there is no data dependency between the `atomic-DFGs`, the kernels are also independent. Thus the kernels can be scheduled arbitrarily. Considering the congruent kernels can be configured by the same set of context bitstream, the goal of kernels constructing is to construct congruent kernels as much as possible. To construct congruent kernels, the `atomic-DFGs` are classified into several categories by sub-graph isomorphism algorithm. The `atomic-DFGs` in the same category have the identical structure. Then the sub-graph recombination algorithm selects some `atomic-DFGs` from different categories to form several series of congruent kernels. We abstract each series of congruent kernels as a `template` since they have the same structure. For a given computation task, the number of `atomic-DFGs` is inversely proportional to the number of congruent kernels. From the perspective of physical meaning, the number of templates means the overall required reconfiguration times for PE array. And for a template, the number of congruent kernels means the required loop count of PE array’s execution. Finally the configuration context is generated for each template.

In this method, the context reduction is achieved by maximizing the number of congruent kernels and minimizing the number of templates.

The key parts of the proposed method are sub-graph isomorphism algorithm and sub-graph recombination algorithm. All the notations used in the algorithms are shown in Table 1.

### Table 1 Notations used in the algorithms.

| Notation | Explanation |
|---|---|
| $v_i$ | Means node in a data-flow graph |
| $e_{ij} < v_i, v_j >$ | Means directed edge. |
| $g_i$ | Represents an `atomic-DFG`'s independent sub-graph. |
| $L(g_i, v_i)$ | Function: returns the level value of node $v_i$ in $g_i$. |
| $n(g_i)$ | Function: returns the numbers of nodes in $g_i$. |
| $v(g_i)$ | Function: returns the length of critical path in $g_i$. (vertical direction) |
| $h(g_i, k)$ | Function: returns the number of nodes at level $k$ of $g_i$. (horizontal direction) |
| $H(g_i) = \max(h(g_i, 0), h(g_i, 1), ..., h(g_i, k))$ | Function: compares the numbers of nodes in each level of $g_i$ and returns the maximum value of nodes. |
| $PSRet(g_i)$ | Function: returns the parameter set of $g_i$. |
| $AREA(g_i) = H(g_i) \times v(g_i)$ | Function: returns the area of $g_i$. It means that $g_i$ needs $AREA(g_i)$ process elements. |
| $G = (g_1, g_2, ..., g_n)$ | Application represented by a set of `atomic-DFGs`. |
| $g_i \simeq g_j$ | Represents $g_i$ is isomorphic with $g_j$. |
| $g_i \cap g_j = \emptyset$ | Means there is no data dependencies between $g_i$ and $g_j$. |
| $C_i = (g_{i_1}, g_{i_2}, ..., g_{i_k})$ | Represents a category of isomorphic independent sub-graphs. |
| $NUM(C_i)$ | Function: returns the number of `atomic-DFGs` in $C_i$ category. |
| $siteof(C_i)$ | Function: returns the size of $C_i$. |
| $k_i$ | Represents a kernel. $k_i$ consists of a set of `atomic-DFGs` |
| $K_i = (k_{i_1}, k_{i_2}, ..., k_{i_k})$ | Represents a kernel set. |
| $t_i \sim k_i$ | Means a template $t_i$ formed by a kernel set $K_i$. |
| $T = < t_1, t_2, ..., t_n >$ | Means the template set $T$. |
| $r$ | `data` function: Round up function |
| $mod(a, b)$ | `Function`: mod function: $a / b$ |
| $CSet$ | Constraint set of PE array. Limitation of PEA in our implementation platform is: the number of operation nodes in kernel should be less than 512 ($=4 \times 8 \times 8$); and the length of critical path should be less than 16 ($=2^4$). |

### Fig. 7 The particular characteristics of `atomic-DFG`.

| Parameter set | Architecture |
|---|---|
| `atomic-DFG` | `parameter` set |
| `leading node` | `parameter` set |
| `sub-graph parameter set` | `parameter` set |
| `operator` | operator type, inputs, outputs |
| `operator` set | `operator` type, inputs, outputs |

3.2 Sub-Graph Isomorphism Algorithm

It is necessary to classify the original input `atomic-DFGs` into several categories to facilitate the generation of congruent kernels. Sub-graph isomorphism algorithm is introduced to implement classification of input `atomic-DFGs`. Some definitions of sub-graph isomorphism are described as follows:

**Definition 1**: “independent sub-graph”/“`atomic-DFG`”. If $\{g_1 = (V_1, E_1), V_1$: a set of nodes in $g_1; E_1$: a set of directed edge in $g_1\}$ and $\{g_2 = (V_2, E_2), V_2$: a set of nodes in $g_2; E_2$: a set of directed edge in $g_2\}$ satisfy $g_1 \cap g_2 = \emptyset$, $g_1$ and $g_2$ are independent.

**Definition 2**: “leading node”. Assuming a node $v_i$ in a `atomic-DFG` $g_i$, if $L(g_i, v_i) = 0$, and its input parameters come from data input FIFO, $v_i$ is a leading node of $g_i$.

**Definition 3**: “sub-graph parameter set” is a parameter set of `atomic-DFG` which reflects the inner relationship of operations in `atomic-DFG`. For each `atomic-DFG`, there may be multiple equivalent parameter sets; and the number of equivalent parameter sets is equal to the number of its leading nodes. And the multiple equivalent parameter sets are identical in syntax. That is to say, you can reconstruct the corresponding `atomic-DFG` according to an arbitrary parameter set. An example of parameter set of an `atomic-DFG` is shown in Fig. 7. For each operator, a vector (operator type, inputs, outputs) is used to represent the operator’s characteristics. The parameter set is the collection of all operators’
vectors in the atomic-DFG. For example, since the atomic-DFG in Fig. 7 has two leading nodes, it has two parameter sets that start from different leading node. For each parameter set, it includes four operator’s vector.

**Definition 4:** “kernel” is a set of atomic-DFGs. One kernel represents one time execution of the entire PE array. If two kernels are not congruent, the PE array needs to be reconfigured two times to execute them.

**Definition 5:** “template” is abstracted from a set of congruent kernels. Since the congruent kernels have the same structure, the template can represent all characteristics of them.

As we discussed in Sect. 3.1, the reuse of congruent kernels can significantly reduce the configuration context. Therefore the objective of the proposed method is to generate the congruent kernels as much as possible. To obtain the congruent kernels, it is necessary to check the atomic-DFGs whether isomorphic or not and classify the atomic-DFGs into several isomorphic categories according its structure.

The sub-graph isomorphism problem is defined as:

**Given** a set of atomic-DFGs \( G = (g_1, g_2, \ldots, g_m) \), where \( g_i \) represents an atomic-DFG;

**Classify** \( G \) into several isomorphic categories \( \{(C_1 = (g_{s_1}, g_{p_1}, \ldots, g_{t_1})), (C_2 = (g_{s_2}, g_{p_2}, \ldots, g_{t_2})), \ldots, (C_k = (g_{s_k}, g_{p_k}, \ldots, g_{t_k}))\} \).

Since the atomic-DFGs have some unique features, the above classification problem is not a general sub-graph isomorphism problem. We propose a traversal algorithm to resolve this problem. The pseudo-code of this sub-graph isomorphism algorithm is illustrated in Fig. 8.

The basic idea of Algorithm 1 is to classify the atomic-DFGs based on its parameter sets. In the algorithm, the key part aims to find the leading nodes and build the sub-graph parameter sets for each atomic-DFG (Line 4–8). Then by comparing the parameter sets of each atomic-DFG, the atomic-DFGs that have same parameter set are classified into the same category (Line 9–16). If an atomic-DFG is not isomorphic to others, then a new category will be created. When all the atomic-DFGs are processed, the classification is finished and the isomorphic categories are obtained.

### 3.3 Sub-Graph Recombination Algorithm

Since the dimension of PE array is usually much larger than that of atomic-DFG, several atomic-DFGs need to be combined to form a kernel. All input atomic-DFGs will be combined as a set of templates and each template represents a series of congruent kernels. Since each template means one time reconfiguration of PE array, to reduce configuration context, the objective of sub-graph recombination is to generate templates as less as possible.

The sub-graph recombination problem is defined as:

**Given** a set of isomorphic categories \( \{(C_1 = (g_{s_1}, g_{p_1}, \ldots, g_{t_1})), (C_2 = (g_{s_2}, g_{p_2}, \ldots, g_{t_2})), \ldots, (C_m = (g_{s_m}, g_{p_m}, \ldots, g_{t_m}))\} \), where \( C_i \) represents a category of isomorphic atomic-DFGs, and a constraint set \( CSet; \)

**Generate** a template set \( T = \{t_1, t_2, \ldots, t_n\} \) that has the minimal number of templates, where \( t_i \) means the template of congruent kernel set \( (t_1 \sim K_i = (k_{o_i}, k_{p_i}, \ldots, k_{q_i})) \), \( K_i \) represents a congruent kernel sets.

To minimize the number of templates, we should generate the congruent kernels as much as possible. In practices, PE array limits the dimension of kernel. According to the dimensions of DFGs and PE array, we can estimate the number of kernels that should be generated. Once the number of kernels is determined, the straightforward way of generating congruent kernels is to distribute the atomic-DFGs averagely to all kernels, which will result in congruent kernels as much as possible. Based on this idea, we propose a sub-graph recombination algorithm. The pseudo-code of the algorithm is illustrated in Fig. 9.

As illustrated in Algorithm 2, the first step of sub-graph recombination algorithm is to figure out the physical limitation and determine how many kernels that each sub-graph isomorphic category \( (C_i) \) should be distributed to (Line 3–7). This is to ensure the atomic-DFGs in one category to be averagely distributed. Then, atomic-DFGs in each category are averagely distributed to those kernels (Line 8–13). If a kernel’s dimension is less than PE array’s dimension, some other atomic-DFGs will be assigned in that kernel to improve the utilization of PE array (Line 17–25). When all atomic-DFGs are assigned, several series of congruent kernels are generated. For each series of congruent kernels, we abstract the template of it. At this point, the template set is obtained.
3.4 Analyzing Algorithmic Complexity

3.4.1 Complexity of Sub-Graph Isomorphism Algorithm

Assuming the number of input atomic-DFGs is \( n \), and the number of categories is \( m \), the foreach-loop in sub-graph isomorphism algorithm will run \( n \) times. For each atomic-DFG, assuming the maximum number of leading node is \( S \) and the maximum number of nodes is \( N \), the time complexity of calculating leading nodes and building parameter sets is about \( O(n \cdot (N + S \cdot N)) = O(n \cdot (1 + S)) \). The time complexity of searching isomorphic sub-graph is about \( O(n \cdot m \cdot S) \). Therefore, the total computational complexity is about \( O(n \cdot N \cdot (1 + S) + (n \cdot m \cdot S)) \). Considering \( N \) and \( S \) are usually much smaller than \( n \) and \( m \) equals \( n \) in worst case, the time complexity of sub-graph isomorphism algorithm is \( O(n^2) \).

3.4.2 Complexity of Sub-Graph Recombination Algorithm

Assuming the number of categories is \( n \), the maximum number of atomic-DFGs in a category is \( M \), and the maximum number of kernels in a category is \( N \) (\( N \leq M \)), the first foreach-loop in sub-graph recombination algorithm will run \( n \) times. After evaluating the number of kernels for a category, atomic-DFGs in a category are assigned to those kernels. The time complexity of assignment for each category is about \( O(n \cdot M \cdot N) \). Considering \( N \) and \( S \) are usually much smaller than \( n \), the total computational complexity is \( O(n^2 \cdot M \cdot N)) = O(n^2) \).

3.4.3 Overall Algorithmic Complexity

The sub-graph isomorphism algorithm and sub-graph recombination algorithm are executed sequentially, so the time complexity of the proposed method is \( O(n^2) \).

4. Experimental Results

4.1 Implementation of REMUS Compiler

To verify and evaluate the proposed context reduction method, it has been implemented in the REMUS C compiler [9]. The diagram of REMUS C compiler framework is shown in Fig. 10. The applications written in a high-level programming language (such as C language) is partitioned into two sections by the HW/SW partitioning algorithm [10], [12]. The computation-intensive algorithms partitioned for the reconfigurable PE arrays are transformed into a set of atomic-DFGs in front-end processing phase using SUIF2 [13] and MachSUIF [14] tools. The sub-graph isomorphism and sub-graph recombination algorithms constitute the context reduction module. The atomic-DFGs are classified and recombined as a series of kernels. Then the configuration templates are extracted from the kernels before context generation. Finally, the configuration contexts (bitstreams) for each template are generated after mapping and placement phase.
4.2 Performance Evaluation

4.2.1 Experimental Setup

We use REMUS processor and its compiler as the test platform. REMUS processor is implemented in 130 nm SMIC technology and it is capable of real-time decoding of H.264 high-profile streams at 200 MHz [11]. A series of experiments are carried out to evaluate the performance of the proposed pre-context-generation method. First, we use some typical computation-intensive algorithms as test examples to evaluate the performance improvement. Second, we compare the proposed method with some post-context generation methods. Some notations used in performance evaluation are described in Table 2.

4.2.2 Performance Improvement

To measure the performance improvement, we use the context generated by Non-ESC method as comparison baseline. Some key multimedia algorithms of H.264 (Motion estimation, discrete cosine transformation, etc.) are used as test examples. We investigate two kinds of metrics: one is the context characteristics, such as the context size, the number of kernels and templates; the other is the execution performance, such as the reconfiguration cycles and execution cycles. The context characteristics can illustrate the intrinsic features of both methods; the execution performance can demonstrate the breakdown of performance improvement.

The comparison results for context size and the number of kernels and templates are shown in Table 3. As we can see, the numbers of kernels generated by both methods are almost the same. This is because that the kernels are constituted of the atomic-DFGs, the same input atomic-DFGs will lead to little difference in the number of kernels. However, the templates generated by our proposed method are much less than Non-ESC method. This is because that by sub-graph recombination, some kernels are reconstructed as congruent kernels that reduce the number of templates significantly. In contrast, Non-ESC method cannot reconstruct the congruent kernels; so the number of templates equals the number of kernels. Consequently, the size of context is much reduced (averagely reduced about 71%), while the redundant contexts are existed in Non-ESC method.

The breakdown of execution cycles is shown in Table 4. The communication cost (T_comm) of our method is less than that of Non-ESC method (averagely reduced about 19.68%), because there is no excess intermediate data transfer delay (t_inter) caused by frequently reconfiguration in our method. Furthermore, the operating time (T_op) of our method is also much less than that of Non-ESC method (averagely reduced about 78.4%). This is because that the major part of operating time (T_op) is the reconfiguration cycles (tCfg). Our method reduces the context size greatly so that tCfg is much reduced. Finally, the total execution cycles (T_total) of our method are much less than that of Non-ESC method (averagely reduced about 68%). The reduction of execution cycles is the aggregating effect of communication cost and operating time reduction.

The above experimental results shows the effectiveness of reducing reconfiguration cost by eliminating the structure correlation.

We also investigate the compile time of both methods, which is defined as the cost of time for context generation. The results shown in Table 5 are obtained on a 2.4 GHz AMD Althon TM 64 ×2 platform. The experimental results show that the compile time has different characteristics according to the critical-path length of DFG: (1) if the critical-path length is less than hardware dimension (a 8 × 8 PE array in our test), our method consumes a bit more compile time compared with Non-ESC method, since the our proposed method needs to analyze and recombine the sub-graphs; (2) if the critical-path length is greater than hardware dimension (in case of Motion Estimation): the Non-ESC method consumes much more compile time compared with our method, since the inter-kernel data dependency is much more complex which needs a lot of time to analyze in Non-ESC method. However, the data-dependency is clear between atomic-DFGs. Since atomic-DFG is the process-
ing unit in our proposed method, the analysis of inter-kernel data dependency is avoided which results in less compile time. From these results, we can see, although the complexity of our method is \(O(n^2)\), the realistic compile time is acceptable. This shows the practicability of the proposed method.

### 4.2.3 Comparison with Post-Context-Generation Methods

We also compare our proposed method with two post-context-generation methods, DCC [7] and CS3 [8], in terms of the context compression ratio and execution performance. Table 6 shows the comparison results for context size and compression ratio.

As we can see, our proposed method can achieve less number of templates than post-context-generation method. This is due to the principle difference of both kinds of context reduction methods. The proposed method can eliminate the structure correlations in the kernels, which results in less number of templates. While post-context-generation method compresses the context based on statistical redundancies of configuration bitstreams, it cannot reduce the number of templates. Although post-context-generation methods can make improvement in context size (averagely compressed about 38.35%), the redundancies between kernels are not eliminated. In contrast, eliminating the structure correlations can result in much better compression ratio (averagely compressed about 61.9%). These results illustrate that the structure correlation in kernels is the principal reason caused context redundancies. Since the proposed method deals with this issue effectively, it can achieve better compression ratio.

Table 7 shows the compression ratio comparison with both post-context-generation methods. As can be seen, the proposed method outperforms both dictionary-based compression [8] and dynamic compression [7]. Considering both of them need extra hardware support, the proposed method has more obvious advantages.

Furthermore, we use a series of test examples to investigate the execution performance. Table 8 shows the comparison results for execution performance with post-context-generation methods.
As shown in Table 8, the compression ratio of DCC methods is a stable value (about 38%). This is because that DCC is based on the statistical characteristics of configuration bit-streams, which results in a stable compression ability. In contrast, the CR value of our method is determined by the structure correlations in DFGs. Since different test examples have varied algorithm structures, the CR value of our method is in a wide variation range. Table 8 shows that our proposed method reduces the context size more efficiently than DCC. This advantage leads to the better execution performance. The operating cycles ($T_{op}$) of our method shown in Table 8 are less than that of DCC (averagely improved about 25.6%). This improvement comes from the less reconfiguration time ($t_{cfg}$) due to smaller context size reduced by our proposed method.

The above comparison shows that the pre-context-generation method based on structure correlation elimination can achieve better performance than post-context-generation methods in terms of both compression ratio and execution performance.

5. Conclusion

CGRA is evolving rapidly in computational-intensive applications due to flexibility and high performance. As the number of PEs and the complexity of the routing resources are increasing, reconfiguration cost is becoming the performance bottleneck. In this paper, we have proposed a structure-correlation elimination method for reducing configuration context. Experiment results show that, compared with Non-ESC method, the proposed method can averagely decrease the execution time and context size up to 68% and 71%, respectively. And it has obvious advantages compared with other post-context-generation methods.

In the future work, we will revise the sub-graph isomorphism and recombination algorithms to reduce the time complexity of the proposed method.

Acknowledgments

This work was supported by NNSF of China grant 60803018 and the National High-Tech Research and Development Program (863 Program) of China grant 2009AA011702.

References

[1] K. Compton and S. Hauck, “Reconfigurable computing: A survey of systems and software,” ACM Comput. Surv., vol.34, no.2, pp.171–210, June 2002.
[2] R. Hartenstein, “A decade of reconfigurable computing: A visionary retrospective,” Proc. DATE, pp.642–649, 2001.
[3] J.R. Hauser, “Augmenting a microprocessor with reconfigurable hardware,” M.S. thesis, Compt. Sci. Dept., Univ. California, Berkeley, 2000.
[4] S. Khawam, I. Nousias, M. Milward, Y. Ying, M. Muir, and T. Arslan, “The reconfigurable instruction cell array,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.16, no.1, pp.75–85, Jan. 2008.
[5] J. Pan, T. Mitra, and W. Wong, “Configuration bistream compression for dynamically reconfigurable FPGAs,” Proc. ICCAD, pp.766–773, 2004.
[6] M. Martina, G. Masera, A. Molino, F. Vacca, L. Sterpone, and M. Violante, “A new approach to compress the configuration information of programmable devices,” Proc. DATE, pp.48–51, 2006.
[7] Y. Kim and R.N. Mahapatra, “Dynamic context compression for low-power coarse-grained reconfigurable architecture,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.18, no.1, pp.15–28, Jan. 2010.
[8] N. Aslam, M. Milward, A. Erdogan, and T. Arslan, “Code compression and decompression for coarse-grain reconfigurable architectures,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.16, no.12, pp.1596–1608, Dec. 2008.
[9] C. Yin, S. Yin, L. Liu, and S. Wei, “Compiler framework for reconfigurable computing architecture,” IEICE Trans. Electron., vol.E92-C, no.10, pp.1284–1290, Oct. 2009.
[10] F. Hanning, H. Dutta, and J. Teich, “Regular mapping for coarse-grained reconfigurable architectures,” Proc. IEEE Int. Conf. Acoust., Speech, Signal Process., pp.57–60, May 2004.
[11] M. Zhu, L. Liu, S. Yin, Y. Wang, W. Wang, and S. Wei, “A reconfigurable multi-processor SoC for media applications,” Proc. ISCAS 2010, pp.2011–2014, 2010.
[12] J. Henkel, “A low power hardware/software partitioning approach for core-based embedded systems,” Proc. DAC, pp.122–127, 1999.
[13] G. Aigner, A. Diwan, D.L. Heine, M.S. Lan, D.L. Moore, B.R. Murphy, and C. Sapuntzakis, “An overview of the SUIF2 compiler infrastructure,” Stanford University, Portland Group, 1999.
[14] M.D. Smith and G. Holloway, “An introduction to MachineSUIF and its portable libraries for analysis and optimization,” The MachineSUIF documentation set, Harvard University, 2002.

Shouyi Yin received the B.S., M.S. and Ph.D. degree in Electronic Engineering from Tsinghua University, China, in 2000, 2002 and 2005 respectively. He has worked in Imperial College London as a research associate. Currently, he is with Institute of Microelectronics at Tsinghua University as an associate professor. His research interests include mobile computing, wireless communications and SoC design.

Chongyang Yin was born in 1984. He received the B.S. degree from the Department of Microelectronics, Xi’dian University, Shanxi, China, in 2006. Currently, he is working toward the Ph.D. degree in the Institute of Microelectronics, Tsinghua University, Beijing, China. His research interests include reconfigurable computing and its compiler design.
Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree in Institute of Microelectronics, Tsinghua University, in 2004. He now serves as an associate professor in Institute of Microelectronics, Tsinghua University. His research interests include reconfigurable computing, mobile computing and VLSI DSP.

Min Zhu received the B.S. degree from the Department of Micro & Nano Electronics, Tsinghua University, Beijing, China, in 2006, where he is currently working toward the Ph.D. degree in the Institute of Microelectronics. His research interests include reconfigurable computing and multimedia processing.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Fac-ulte Polytechnique de Mons, Belgium, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communicationASIC design.