Context Management Scheme Optimization of Coarse-Grained Reconfigurable Architecture for Multimedia Applications

Peng Cao, Bo Liu, Jinjiang Yang, Jun Yang, Meng Zhang, and Longxing Shi

***Abstract*— Due to the combination of flexibility and efficiency, coarse-grained reconfigurable architectures (CGRAs) are suitable for the implementation of computing-intensive applications. However, with the growing performance requirements, the scale of CGRA increases exponentially, which leads to configuration performance degradation and configuration power rise. Based on the analysis of configuration context features, we optimize the context management scheme of CGRA from the aspects of context cache structure and replacement strategy. The context cache is structured hierarchically to reduce the memory overhead without configuration performance degradation and a hybrid context replacement algorithm is proposed to further increase the configuration efficiency with a novel context frequency weight factor. Experimental results show that the proposed context management scheme improves the configuration performance of the base CGRA significantly by 13.6%–20.5% for H.264 decoding and 13.6%–20.5% for MPEG2 decoding with only 43% context cache cost. Compared with other works, the proposed context management scheme shows the advantages of 2.3–6× less normalized context cache size and 2.3–2.7× cache efficiency.**

***Index Terms*— Cache replacement algorithm, coarse-grained reconfigurable architecture (CGRA), context cache, context management, multimedia applications.**

I. Introduction

With the rapid development of information technology and the accelerated change of Internet resources, multimedia processing technologies, as fundamental carriers of the internet and information technology, have been one of the hottest research points during the last decades. The conventional computing architectures can be classified into two types: general purpose processors (GPPs) and application specific integrated circuits (ASICs). Although solutions with GPPs can provide the most flexibility, they cannot fulfill the increasing requirements on performance, cost, and power consumption because of their sequential software execution. ASICs can provide the best performance for specific applications, but they have fixed and limited functions performed by predefined modules, which make them infeasible to suit new design requirements or updates in standards. To overcome their shortcomings, field-programmable gate array (FPGA) was introduced as reconfigurable computing device for several decades. However, the flexibility of FPGAs is achieved at a very high silicon cost to interconnect huge amount of processing primitives since the functionality of the hardware is specified at bit-level. Moreover, FPGAs are reconfigured with intractable and costly reconfiguration process, so that they are not efficient for the implementation of computing-intensive applications.

As a novel architecture to implement computation-intensive and data-parallel applications, coarse-grained reconfigurable architectures (CGRA) can fill the gap between GPP and ASIC solutions, offering potentially much higher performance than GPP, while keeping higher flexibility than ASIC [1]. Due to flexibility and high degree of parallelism, CGRA has been regarded as an alternative to the conventional computing architectures. Various architectures have been proposed to complete computational tasks in different fields [2], in which ADRES [3]–[5], XPP-III [6], [7], ReMAP [8], MORA [9], and MorphoSys [10] are the typical representatives.

Considering that most programs consist of a number of consecutive phases $P = [1,p]$, the performance of CGRA can be defined in terms of the operating frequencies f_p, the instructions executed per cycle IPC_p and the instruction counts IC_p of each phase, and the time overhead involved in switching between the phases$t_{p\to p+1}$ as follows [11]:

$$\frac{1}{\text{performance}} = \text{execution time} = \sum_{p\in P} \frac{\mathrm{IC}_p}{\mathrm{IPC}_p \times f_p} + t_{p\to p+1}. \quad (1)$$

It can be shown in (1) that the performance of CGRA can be improved from three aspects as follows.

1) Increasing the working clock frequency. However, it is restricted by the delay of critical path, and results in higher power consumption, thus affects energy efficiency seriously.
2) Enlarging the array scale of CGRA by adopting more computing resources to obtain higher processing parallelism. When the reconfigurable arrays (RCAs)

Manuscript received December 18, 2016; revised March 14, 2017; accepted April 14, 2017. Date of publication May 5, 2017; date of current version July 24, 2017. This work was supported in part by the National High Technology Research and Development Program of China under Grant 2015AA016601; in part by the National Natural Science Foundation of China under Grant 61404028, Grant 61474022, Grant 61574033, and Grant 61550110244; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20161147; and in part by the China Scholarship Council. *(Corresponding author: Jun Yang.)*

The authors are with the National ASIC System Engineering Research Center, Southeast University, Nanjing 210096, China (e-mail: dragon@seu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2695493

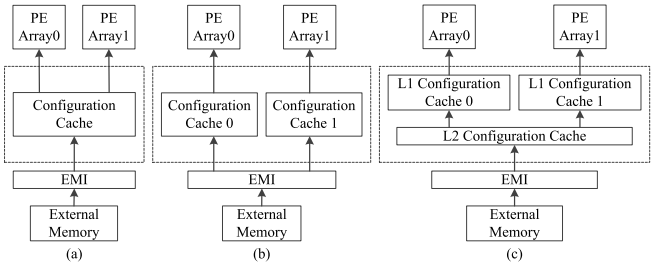


Fig. 1. Context cache Structures. (a) Centralized. (b) Distributed. (c) Hierarchical.

are scaled up, the requirement for reconfiguration efficiency becomes more critical, because there will be more reconfigurable units required to be dynamically feed with configuration data (also called configuration context or context).

3) Reducing the time overhead for configuring tasks. Therefore, for both the second and the third methods, the efficient reconfiguration solution is of essential importance since the growing configuration size and latency may become the major bottleneck in a large-scale CGRA.

Configuration management techniques can be developed from multiple perspectives to attack the bottleneck [12] such as configuration compression [13], [14], configuration prefetching [15], [16], and configuration caching, where configuration caching technique is used to store configuration context with an on-chip memory to reduce the configuration latency. During the process of configuration, the task kernels of application are usually mapped by obeying reconfigurable resource constraints to facilitate loop pipeline, leading to high hardware utilization and less configuration overhead [17], [18].

In this paper, we suggest a novel efficient context management scheme from the aspects of context cache structure and replacement strategy. First, the context cache is structured in a hierarchical fashion to reduce the memory overhead without configuration performance degradation by exploring the bandwidth and capacity of cache in each level, which shows the significant advantage especially for a large-scale CGRA. Second, a hybrid context replacement algorithm is proposed to further increase the configuration efficiency with a novel context frequency weight factor.

The remainder of this paper is organized as follows. The context cache organization scheme is summarized in Section II. Then in Section III, we describe the base architecture of CGRA and its multilayer configuration context structure. After the reconfiguration features of multimedia application are analyzed and illustrated on CGRA in Section IV, we present the context management scheme including hierarchical context cache structure and hybrid context cache replacement algorithm to improve the reconfiguration performance in Section V and carry out design space exploration in Section VI. Experimental results and comparisons are shown in Section VII and conclusions are made in Section VIII as a final.

II. Overview of Context Cache Organization Scheme

Context cache, also called context memory, is usually used to describe the on-chip memory between the reconfigurable array and external memory, which provides the local storage for recently used configuration contexts and plays an important role for high configuration performance and flexibility but suffers from memory and power overhead.

The structure of context cache can be classified to three categories as shown in Fig. 1, namely the centralized, the distributed, and the hierarchical, respectively.

The centralized context cache structure is depicted in Fig. 1(a). It can be seen that only one context cache is implemented in the system and shared by each processing element (PE) array. This cache structure is simple in terms of design and supervision, but is inefficient due to two reasons. The first is the large cache size which results from the context requirements of all PE arrays. The second is the access conflicts during configuration process due to cache access bandwidth, which will limit computing performance of PE arrays seriously. In MorphoSys [10], context is transferred from external memory to a centralized context memory before loading to the reconfigurable PE arrays as needed. Since the content of context memory is organized by a static method, the efficiency and flexibility of such a system will be limited by the specified context memory.

For distributed context cache structure, as depicted in Fig. 1(b), each PE array is attached with an individual context cache, and all the context caches are equally connected to the external memory through external memory interface (EMI). The context access efficiency of this cache structure can be quite high, because each PE array can read contexts from its attached context cache in parallel. However, in this case, contexts should be duplicated in different context caches when they are shared between different PE arrays, which results in heavy cache storage redundancy and memory space occupancy waste. Dynamically reconfigurable processor (DRP)-1 [19] is a commercial CGRA introduced by NEC and used for stream applications. It consists of 4×2 tiles and each tile contains an 8×8 reconfigurable array. Each array of DRP-1 features a dedicated context cache to achieve high configuration performance, but suffers from high memory overhead, especially for large-scales CGRA where more arrays as well as context caches are implemented.

As depicted in Fig. 1(c), the hierarchical context cache structure can be regarded as the heterogeneous combination of centralized and distributed structures. From the outer to the inner level, the memory sizes of context caches progressively decrease and the context access bandwidths progressively increase. Based on the hierarchical cache structure, the system can provide satisfying solutions in terms of both high access performance and small memory occupancy. However, compared to the centralized and distributed approaches, the management mechanism for hierarchical structures is much more complicated with the challenge that storage resources should be allocated reasonably in each level of the hierarchical structure. XPP-III [6], [7] contains several processing array clusters (PACs), which are attached to configuration

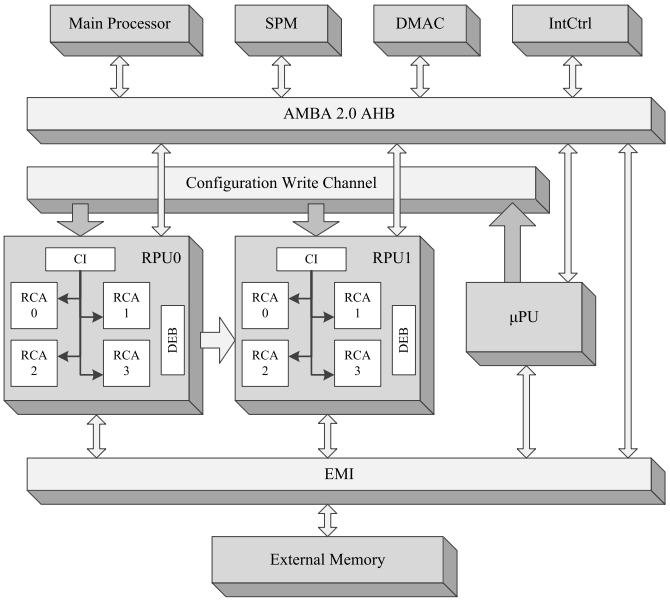


Fig. 2. Block diagram of base architecture of CGRA.

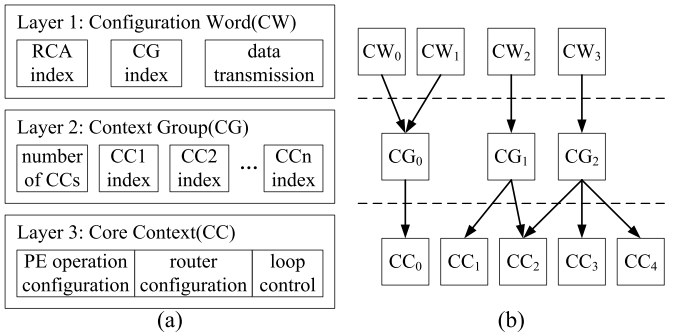


Fig. 3. Three-Layer Context structure. (a) Content of each layer. (b) Example of context structure.

managers (CM). A tiny local memory is included in each CM and used as context cache for the associated PAC. A root CM called the supervising CM (SCM) is responsible for managing the context transfers between each CM and the main memory. The distributed CMs and the root SCM form the hierarchical context cache structure, which makes the reconfiguration process more efficient and flexible. However, due to its centralized management style, when the array scale increases, more CMs need to be supervised by the single SCM, which would be the bottleneck of the system. MORPHEUS [20] also features a typical hierarchical configuration memory structure: level 1 is a specified configuration exchange buffer in each heterogeneous reconfigurable engine (HRE), level 2 is an on-chip context cache accessed by HRE, and level 3 is the external main memory. Though the sizes of configuration memory are configurable at design time for each level, these HREs executed independently with no context sharing among them, which leads to redundancy among the configuration memories. Moreover, it considered little about the trade-off between the configuration memory overhead and performance constraint.

III. Preliminaries

In this section, we describe the base architecture of CGRA and its associated multilayer context structure, which has been published in [21] and [22] and is used for comparison with the proposed context management scheme.

A. Base Architecture of CGRA

The base architecture of CGRA is illustrated in Fig. 2, where multiple CG reconfigurable processing units (RPUs) are connected to an AHB bus as attached IPs as well as a microprocessor unit (μPU) to manage the reconfiguration process of RPU. A main processor works as the host of the whole CGRA system with assistant modules including interrupt controller, direct memory access controller, scratch-pad memory, EMI, and external memory. The RPU incorporates multiple RCAs to execute different tasks while the RCA contains an M $\times$ N PE array for arithmetic and logical operation. The communication interface inside RPU is responsible for the data path of configuration contexts.

Similar as representative reconfigurable architectures such as MorphoSys [10] and DRP-1[19], multipleRCAs are implemented in the base architecture to increase the scale of CGRA instead of extending the array scale. The merit is to limit the size of configuration for the operations and routers in each RCA so as to avoid the configuration efficiency degradation in terms of generation and transmission latency.

The base architecture has been realized at RTL level and fabricated under the process of TSMC65 nm. Two homogeneous RPUs were implemented with four RCAs in each. The RCA contains an 8 $\times$ 8 reconfigurable array of PEs with 16-bit data width. Moreover, since numerous bit-level operations are required to generate the configuration contexts of RPUs, 16 microprocessors are incorporated in μPU to schedule the reconfiguration process in RCAs and RPUs in parallel.

B. Multilayer Context Structure

The base architecture of CGRA features dynamic configuration for computing-intensive applications. For each mapped task, no more configurations are required during its computation. However, the configuration of the following task can be performed along with the computation of the current one in pipeline so as to reduce the execution time of the whole application.

In order to realize configuration pipeline and improve efficiency, the structure of context is organized with three layers, which are called configuration word (CW), context group (CG), and core context (CC), respectively, as shown in Fig. 3(a).

The CW acts as the top layer of context for a certain task mapped to RCA and is generated by μPU at runtime, which indicates the index of the mapped RCA and the index of its corresponding CG, as well as the data transmission command for the RCA. The context size depends on the number of the blocks for data loading and storing.

The second layer context, CG, is invoked by the mapped RCA, which describes the number of CCs for the RCA and the indexes of these CCs in sequence. For different number

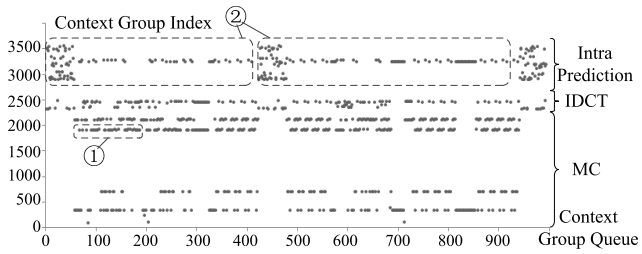


Fig. 4. Temporal Locality of Context.

of CCs included in one CG, the context size varies with the maximum of 64 words.

The third layer context, CC, incorporates the detail configuration for the operations and routers of RCA as well as the loop control command. For the RCA with the scale of 8 × 8, 128 words are occupied for each CC.

With the three-layer context structure, the contexts of different tasks on RCAs can be shared at the layer of CG and CC. As exampled in Fig. 3(b), CW_0 and CW_1 share the same CG_0 whereas CC_2 is used in both CG_1 and CG_2, so that the similarity among the tasks configured with $CW_n (n = 0, 1, 2, 3)$ is well utilized to reduce the on-chip context storage cost.

In each RPU of the base architecture of CGRA, the contexts of CG and CC are stored in a centralized context cache, respectively, namely group cache (CGC) and CC cache (CCC).

IV. Reconfiguration Context Features of Mapping Multimedia Applications Onto Large-Scale CGRA

In this section, three features of the reconfiguration process are demonstrated and analyzed for multimedia applications, including temporal locality, nonuniform access frequency, and nonuniform computation parallelism.

A. Temporal Locality

The temporal locality shows the objective regularity of reconfiguration process that some certain tasks or subtasks of the target application might be performed frequently in a short period of time, which is demonstrated in the aspects of CG and CC.

Fig. 4 depicts the locality analysis of the CGs for the tasks of intra prediction, inverse discrete cosine transform (IDCT), and motion compensation (MC), where the vertical axis represents the indexes of each CG and the horizontal axis denotes the usages and orders of context scheduling process for the period. It can be seen that some CGs would be called by RCAs quite frequently in a short period of time, as shown by "①," and during the whole procedure, the usage of some CGs would repeat regularly, as shown by "②."

The CC also features the temporal locality due to two reasons. First, the CCs inside the CG will also be called by RCAs repeatedly even with a shorter period than CG. Second, since adjacent CGs in sequence may include the same CC, the temporal locality would exist in CC more obviously than CG. For example, the CCs for the operations such as matrix transpose and dot production would be invoked from the adjacent but different CGs for the tasks of intra prediction and MC.

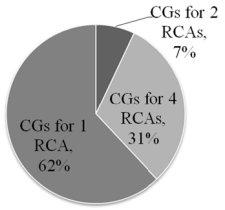


Fig. 5. Usage Percentage of Context Groups for 1, 2, and 4 RCAs.

B. Nonuniform Access Frequency

The nonuniform access frequency feature demonstrates the variation of access frequencies of different configuration contexts within the whole application, which is manifested in the contexts for tasks and subtasks.

During the processing of multimedia application, some kernel tasks among the most 80% frequently used, such as intra prediction, IDCT, and MC, occupy only 10% of the configuration contexts of CG and CC. In addition, the gap between the maximum and minimum usage number of context can be as large as nearly 10,000 times.

This feature can also be found in subtasks. Take the task of intra prediction for example, which contain nine subtasks for a luma 8 × 8 submacro block. Each subtask corresponds to an individual computing mode, whose average access times of contexts vary a lot. The contexts for mode *down_right* makeup nearly 1/4 of the total context access times with only 3% contexts for mode *down_left*.

C. Nonuniform Computation Parallelism

The feature of nonuniform computation parallelism shows that some certain tasks of the target application can be divided into multiple subtasks, which are mapped to multiple RCAs simultaneously to break though the resource limitation of RCA scale and increase performance. Therefore, these subtasks can be processed in parallel by sharing the same contexts.

The computation parallelism feature of each CG for H.264 decoding is depicted in Fig. 5, where those mapped to multiple RCAs constitute 38% of the total CGs so that roughly 26% configuration cost of CG can be reduced by context sharing.

V. Proposed Context Management Mechanism

The proposed context management mechanism includes the hierarchical context storage subsystem and the hybrid context management strategy. The former reduces the total storage overhead and provides the hardware platform of the context storage subsystem for the latter to further increase the configuration transmission efficiency.

A. Hierarchical Context Cache Structure

The proposed hierarchical context cache structure is illustrated in Fig. 6. Compared to the base architecture, the context caches of CG and CC are reconstructed hierarchically for the level of RPU, RCA, and PE array, where the CGCs are composed of L2CGC (level 2 CGC) and L3CGC (level 3 CGC), while the CCCs include L1CCC (level 1 CCC), L2CCC (level 2 CCC), and L3CCC (level 3 CCC).

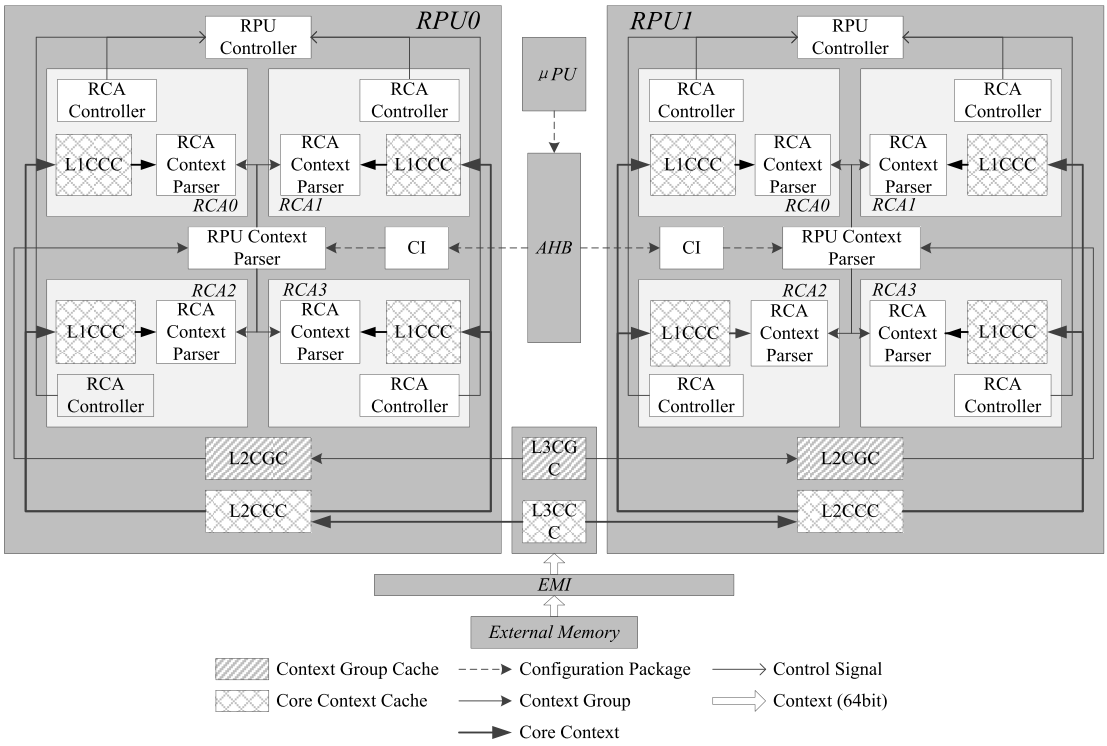


Fig. 6. Hierarchical context storage subsystem structure.

As shown in Fig. 6, the L1 context cache (L1CCC) is tightly coupled with RCA and used as a temporal repository of the CCs used recently. The memory size of L1CCC should be very small, but the access bandwidth is the highest among all levels, because it is responsible for one individual RCA. The L2 context caches (L2CGC and L2CCC) are attached to RPU and shared by the RCAs inside the RPU, whose access bandwidths should be greater than the L3 context caches and less than the L1 context caches, whereas the relation of the memory sizes is the opposite. The L3 context caches (L3CGC and L3CCC) are shared by all RPUs of CGRA, which is similar to the role of the centralized cache structure. Hence, the L3 context caches have the largest memory sizes but the least access bandwidths.

Based on the context storage subsystem, the reconfiguration process is performed by three stages as illustrated in Fig. 7.

Stage 1: CW generation stage. The microprocessors within the μPU parse the *C* code which describes the control flow of the reconfiguration tasks to generate the corresponding CWs, before sending them to the corresponding RCAs.

Stage 2: CG load stage. Once CWs are received and parsed, the CGs required for each RCA in the near future can be completely predicted and loaded in advance. When the CG misses in the lower level of CGC, the RPU context parser checks it from the higher level progressively until the external memory and updates the contexts in the CGC accordingly.

Stage 3: CC load stage. When the RCA gets the required CG, the CC load stage will be performed for the corresponding CCs, which is similar to the CG load stage in Stage 2. The L1CCC, L2CCC, and L3CCC will be checked by the RCA context parser in sequence until all the CCs required are accessed. Moreover, the content of associated CC caches should be updated accordingly.

During the CG/CC load stage, if some CGs/CCs involved are currently not buffered in the associated CG/CC caches, the victim CG/CC should be selected and replaced by the required. Then comes the problem of cache management, which will be discussed in the following.

It should be noted that the proposed context cache hierarchy is different from the context hierarchy mentioned in Section III-B. The context cache hierarchy is applied to store the configuration context in a multilevel cache structure, whereas the context hierarchy organizes the context in a multilayer fashion. In the base CGRA, the configuration context is organized with three layers and each layer of context is stored in a centralized context cache. As for the proposed hierarchical context cache structure, the configuration context is still organized with three layers. However, the context of CG and CC are stored in a hierarchical cache with two levels and three levels, respectively.

B. Hybrid Context Replacement Strategy

In this subsection, a hybrid context replacement strategy is proposed to organize the CG/CC cache and to select the most appropriate victims.

Caching technique is an important approach in general processors to hide memory latency, which includes random, round-robin, least recently used (LRU), least frequently used (LFU), and so on. However, as for CGRA, the replacement algorithm of context cache should be optimized to adapt to the configuration context features due to the following reasons.

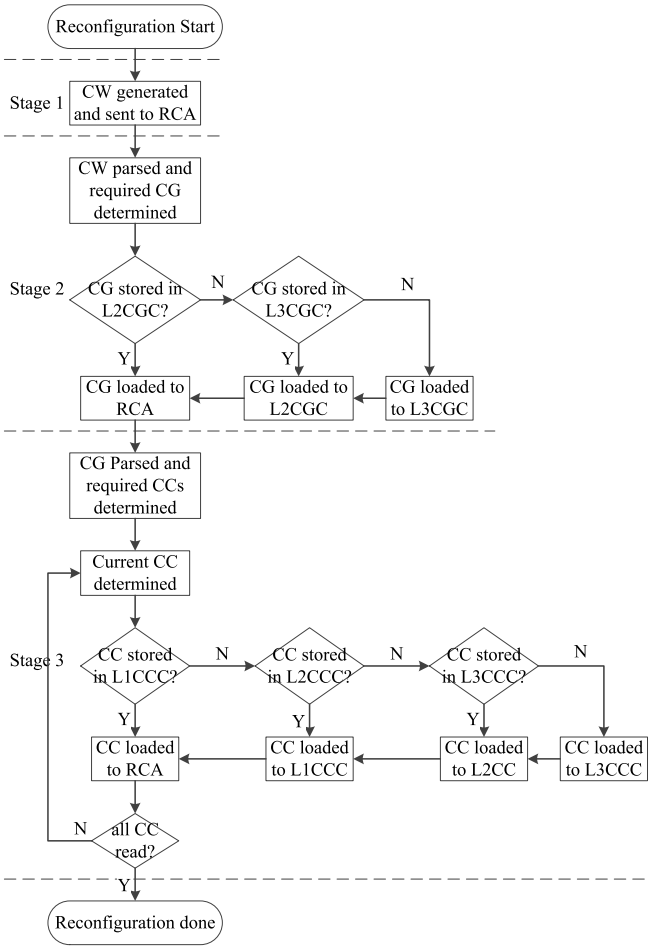


Fig. 7. Flowchart of three-stage reconfiguration process.

```
Algorithm LRU_LFU
1   if(cache hit & tag[i]==CFG_i.addr){
2       cnt[i]=frq[i]*fwf;
3       all other cnt ++;
4   }
5   else if(cache is not full & m CFGs exist in the cache){
6       CFG[m]=CFG_i;
7       tag[m]=CFG_i.addr;
8       frq[m]=CFG_i.frq;
9       cnt[m]=CFG_i.frq* fwf;
10      all other cnt++;
11  }
12  else{
13      CFG[x] with the max counter chosen as victim;
14      CFG[x]=CFG_i;
15      tag[x]=CFG_i.addr;
16      frq[x]=CFG_i.frq;
17      cnt[x]=CFG_i.frq* fwf;
18      all other cnt++;
19  }
```

Fig. 8. Pseudocode of LRU_LFU Algorithm.

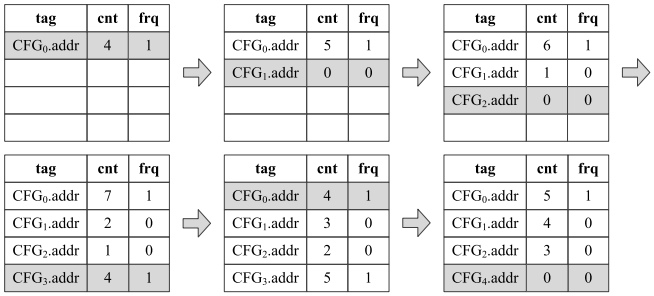


Fig. 9. Replacement sample of LRU_LFU algorithm.

First, the data size of context is usually much larger than that of the instruction, which results in much more latencies for cache prefetching and updating operations, as well as more serious degradation to reconfiguration performance caused by cache miss.

Second, the existing cache replacement algorithms have their limitations for configuration context. The effect of the LRU algorithm would be weakened due to different context lifetimes. In the case of the LFU algorithm, those contexts with high access frequency would be cached with high priority. However, due to significant temporal locality, the cache performance would deteriorate during task switch.

In order to satisfy the specific features of configuration context, a hybrid LRU_LFU cache replacement algorithm is proposed. Besides the traditional parameters including the counter (denoted as *cnt*) and the frequency flag (denoted as *frq*) for the LRU and LFU algorithms, a parameter called the context frequency weight factor (denoted as *fwf*) is suggested.

The pseudocode of LRU_LFU algorithm is shown in Fig. 8, where the current needed context is defined as CFG_i with its address and usage frequency indicated by CFG_i.addr and CFG_i.frq. The LRU_LFU algorithm is different with the traditional LRU in the initialization value of counter, which is set to CFG_i.frq multiplied by *fwf* instead of zero.

Fig. 9 illustrates the procedure of LRU_LFU algorithm assuming that the context cache can hold at most four contexts and the value of *fwf* is 4. Assuming that five contexts, CFG_i (I = 0,1,2,3,4), are read from the context cache in the sequence of CFG_0, CFG_1, CFG_2, CFG_3, CFG_0, and CFG_4, in which the frequencies of CFG_0 and CFG_3 are lower and set to 1 whereas those of CFG_1, CFG_2, and CFG_4 are higher and set to 0. During the first four accesses, the context cache misses and CFG_0, CFG_1, CFG_2, CFG_3 are loaded into the context cache in order. When CFG_0 is needed again, cache hit happens, and the corresponding counter is reset to 4 ($=1 \times 4$). When CFG_4 is needed finally, cache miss happens. Since the current largest counter value is 5, the corresponding context CFG_3 is replaced by CFG_4.

VI. Design Space Exploration and Optimization

A. Design of the Context Cache Structure

Generally, the performance of context cache can be denoted as the clock cycles dependent on the hit/miss ratio and hit/miss overhead, which is given as follows:

$$T_{\mathrm{acc}} = h \times T_{\mathrm{hit}} + m \times T_{\mathrm{miss}} \tag{2}$$

where

T_{acc}	the average access cycles of context cache;
T_{hit}	the access cycles if context cache hits;
T_{miss}	the access cycles if context cache misses;
h, m	the hit and miss ratio of context cache.

Equation (2) can be extended into (3) for the hierarchical context cache structure

$$T_{\text{acc}} = h_{L1} \times T_{\text{hit}L1} + m_{L1} \times (h_{L2} \times T_{\text{hit}L1} + m_{L2} \times (h_{L3} \times T_{\text{hit}L3} + m_{L3} \times T_{\text{ext}})) \tag{3}$$

where

$T_{\text{hit}Ln}$	the access cycles if the level n (n =1,2,3) context cache hits;
T_{ext}	the access cycles from the external memory;
h_{Ln}, m_{Ln}	the hit and miss ratio of the level n ($n = 1, 2, 3$) context cache.

Based on (3), the context access cycles of CGC and CCC in the proposed context cache structure can be deduced in (4) and (5), respectively

$$T_{\text{CGC}} = S_{\text{CG}} \times \left(\frac{h_{L_2\text{CGC}}}{\text{BW}_{L_2\text{CGC}}} + \frac{m_{L_2\text{CGC}} \times h_{L_3\text{CGC}}}{\text{BW}_{L_3\text{CGC}}} + \frac{m_{L_2CGC} \times m_{L_3\text{CGC}}}{\text{BW}_{\text{ext}}} \right) \tag{4}$$

$$T_{\text{CCC}} = S_{\text{CC}} \times \left(\frac{h_{L_1\text{CCC}}}{\text{BW}_{L_1\text{CCC}}} + \frac{m_{L_1\text{CCC}} \times h_{L_2\text{CGC}}}{\text{BW}_{L_2\text{CCC}}} \right) + S_{\text{CC}} \times \frac{m_{L_1\text{CCC}} \times m_{L_2\text{CCC}} \times h_{L_3\text{CCC}}}{\text{BW}_{L_3\text{CCC}}} + S_{\text{CC}} \times \frac{m_{L_1\text{CCC}} \times m_{L_2\text{CCC}} \times m_{L_3\text{CCC}}}{\text{BW}_{\text{ext}}} \tag{5}$$

where

S_{CG}	the total size of CG;
S_{CC}	the total size of CC;
$h_{\text{LmCGC}}/m_{\text{LmCGC}}$	the cache hit/miss ratio of the level $m(m = 2, 3)$ CGC;
$h_{\text{LnCCC}}/m_{\text{LnCCC}}$	the cache hit/miss ratio of the level $n(n = 1, 2, 3)$ CCC;
$\text{BW}_{\text{LmCGC}}/\text{BW}_{\text{LnCCC}}/BW_{\text{ext}}$	the access bandwidth of the context cache of the level $m(m = 2, 3)$ CGC or the context cache of the level $n(n$ =1,2,3) CCC or the external memory.

To evaluate the effect from the context caches in total, the normalized hit ratios, h_{CGC} and h_{CCC}, are introduced to indicate the performance of the entire hierarchical CG and CC context cache, respectively, from the point of view of PE arrays. Therefore, the T_{CGC} and T_{CCC} can also be calculated in (6) and (6), respectively

$$T_{\text{CGC}} = S_{\text{CG}} \times \left(\frac{h_{\text{CGC}}}{\text{BW}_{L_2\text{CGC}}} + \frac{1 - h_{\text{CGC}}}{\text{BW}_{\text{ext}}} \right) \tag{6}$$

$$T_{\text{CCC}} = S_{\text{CC}} \times \left(\frac{h_{\text{CCC}}}{\text{BW}_{L_1\text{CCC}}} + \frac{1 - h_{\text{CCC}}}{\text{BW}_{\text{ext}}} \right). \tag{7}$$

The bandwidth of the outermost level of external memory is designed to be 64 bits/cycle and considering the scale of RPU and RCA, the bandwidths of each level of context cache are designed as shown in Table I.

TABLE I
BANDWIDTH OF CONTEXT CACHE (BITS/CYCLE)

Context Cache	CGC		CCC		
	L2CGC	L3CGC	L1CCC	L2CCC	L3CCC
Bandwidth	256	128	1024	512	256

TABLE II
DESIGNS AND HARDWARE OVERHEAD OF CONTEXT CACHE STRUCTURE WITH DIFFERENT CACHE SIZES

Design	StrucA	StrucB	StrucC
	# of Contexts		
CGC(L2/L3)	8/16	16/32	32/64
CCC(L1/L2/L3)	8/16/32	16/32/64	32/64/128
	Hardware overhead		
CGC(L2/L3/Total) (KB)	2/4/8	4/8/16	8/16/32
CCC(L1/L2/L3/Total) (KB)	4/8/16 /64	8/16/32 /128	16/32/64 /256
Cache Size(KB)	72	144	288
Cache Area(mm^2)	1.44	2.82	5.53
CGRA Area(mm^2)	20.18	21.56	24.27
Area Ratio	7.1%	13.1%	22.8%

With the simultaneous equations of (4) to (6) as well as the bandwidths of each level, the normalized parameter h_{CGC} and h_{CCC} can be deduced as (8) and (10). From the two equations, it can be seen that the inner context caches have more significant effect than the outer, so in order to gain higher hit ratio with low *c*ache overhead, more cache size should be occupied for the inner cache to achieve high reconfiguration performance

$$h_{CGC} = \left(4 - h_{L_2\text{CGC}} - 2 \times m_{L_2\text{CGC}} h_{L_3\text{CGC}}\right)/3 - 4 \times m_{L_2\text{CGC}} h_{L_3\text{CGC}}/3 \tag{8}$$

$$h_{\text{CCC}} = \left(16 - h_{L_1\text{CCC}} - 2 \times m_{L_1\text{CCC}} \times h_{L_2\text{CCC}}\right)/15 - 4 \times m_{L_1\text{CCC}} \times m_{L_2\text{CCC}} \times h_{L_3\text{CCC}}/15 - 16 \times m_{L_1\text{CCC}} \times m_{L_2\text{CCC}} \times m_{L_3\text{CCC}}/15. \tag{9}$$

As a design space exploration on context cache size, three designs of cache structure (marked as StrucA, StrucB, and StrucC) are taken into account for evaluation. Table II shows the context cache sizes of these designs in terms of the number of contexts that can be contained in each level of CGC and CCC as well as their hardware overhead, where the cache size of StrucC is equivalent with the context cache in the base architecture, whereas those of StrucB and StrucA are half and one fourth of it. These three cache structures have been synthesized with TSMC 65-nm technology and integrated into the base architecture, where context caches were implemented by SRAM Macro Cell library. It can be found that the area ratio of context caches varies between 7.1% and 22.8% with respect to the overall CGRA, which indicates that the size of context caches has a significant impact to the hardware overhead of CGRA.

The above designs were simulated with the applications of H.264 and MPEG2 decoding. For H.264, the two RPUs of the whole CGRA were scheduled in pipeline to perform different tasks whereas for MPEG2, they executed in parallel. The normialized context cache hit ratio of the three designs, h_{CGC} and h_{CCC}, are summarized in Tables III and IV along

TABLE III

VARIATION OF NORMALIZED HIT RATIO AND CONTEXT TRANSMISSION TIME WITH DIFFERENT CACHE SIZES FOR H.264 DECODING

	StrucA		StrucB		StrucC	
	RPU0	RPU1	RPU0	RPU1	RPU0	RPU1
h_{CGC}(%)	46.87	16.64	80.93	92.34	96.98	99.92
h_{CCC}(%)	93.07	99.99	97.40	99.99	99.88	99.99
T_{CG}(cycles)	134	140	94	50	70	42
T_{CC}(cycles)	676	265	461	265	337	265
T_{ctx}(cycles)	810	405	555	315	407	307

TABLE IV

VARIATION OF NORMALIZED HIT RATIO AND CONTEXT TRANSMISSION TIME WITH DIFFERENT CACHE SIZES FOR MPEG2 DECODING

	StrucA	StrucB	StrucC
	RPU0/RPU1	RPU0/RPU1	RPU0/RPU1
h_{CGC}(%)	52.56	95.34	99.95
h_{CCC}(%)	95.67	99.99	99.99
T_{CG}(cycles)	94	62	52
T_{CC}(cycles)	343	232	232
T_{ctx}(cycles)	437	294	284

with the configuration overhead, where $T_{\mathrm{CG}}$and T_{CC} denote the context access time per macro block of video stream for CG and CC, whereas T_{ctx} represents the total.

It can be seen that, when the context cache size expands, the normalized cache hit ratio rises up and the context access time falls down. Among the three designs, the trend is the same for the three different workloads, which are the workload of RPU0 in H.264 decoding, that of RPU1 in H.264 decoding and two RPUs in MPEG2. In spite of different computation complexity and configuration complexity, StrucB provides the promising solution for all cases, whose hit ratios and context access time are close to the highest of StrucC with only half cache size.

It can also be found that the hit ratios of CCC are generally higher than those of CGC for all situations, which is tightly corresponding to the higher temporal locality in CCC as noted in Section IV-A. Even though the maximal numbers of CCs in the same level of cache are as twice as those of CGs, they are not sufficient for the corresponding CGs because usually about five to ten CCs are invoked in one CG. Hence the larger size of CCC is not the main reason for the higher hit ratio.

According to the exploration results mentioned above, StrucB was adopted as the implementation of hybrid context cache structure for the following exploration.

B. Design of the Context Frequency Weight Factor

In the proposed hybrid LRU_LFU algorithm, the cache hit ratio and context access time are tightly associated with the context frequency weight factor,*fwf*. For the sake of hardware cost, the value of *fwf* is set as the integer powers of two so that the hardware implementation of the product of *frq* and *fwf* can be simplified as shift operation. Fig. 10 shows the total average context access time per macro blocks using the LRU_LFU replacement algorithm, where several video streams of H.264 and MPEG2 are selected as test cases.

It can be seen that the configuration performance fluctuates with the value of *fwf* for all the streams. The variation trend can be explained as follows. When the value of *fwf* is small,

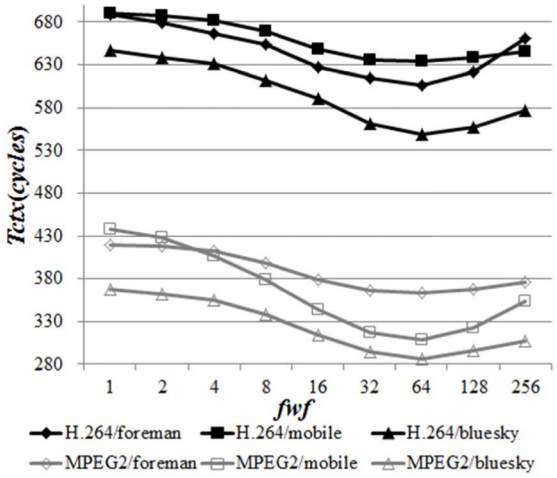


Fig. 10. Variation of context access performance with different values of *fwf*.

TABLE V

RESULTS OF HARDWARE IMPLEMENTATION OF OPTIMIZED CGRA

Technology	TSMC 65nm
PE Array Scale	8×8×8
Working Frequency	200 MHz
Context Cache Size	144KB
Context Cache Area	2.82mm^2
CGRA Area	21.56mm^2

the cache replacement is mainly based on temporal locality. As the value of *fwf* rises, the temporal locality and nonuniform access frequency are both taken into consideration. Therefore, the context cache hit ratio rises with shorter context access time. When the value of *fwf* continues to increase, the weight factor for temporal locality falls down. As a result, the cache memory will be occupied by more contexts with good temporal locality but poor nonuniform access frequency, which will decrease the cache hit ratio. It can be seen that the shortest T_{ctx} can be reached when the value of *fwf* is 64, which indicates the best context cache performance for the proposed hierarchical context cache structure.

VII. EXPERIMENTAL RESULTS AND COMPARISONS

A. Experimental Results

The CGRA with the proposed context management mechanism was described with VerilogHDL language and synthesized under the technology of TSMC 65 nm. The timing results were reported by Synopsys design compiler using the typical case and the dynamic power was estimated by Synopsys PrimeTime-PX (PTPX) under the same condition. The results of hardware implementation are shown in Table V in detail. The CGRA occupies 21.56 mm^2 to implement the PE array with 8 RCAs, each of which contains 8 × 8 PEs, and 144-K bytes context cache with the area of 2.82 mm^2.

The CGRA was verified by H.264 and MPEG2 video streams and the decoding performance was measured in terms of the clock cycles for each macro block, which was illustrated in Table VI. It can be seen that the CGRA can meet the performance requirement of high-resolution (1920 × 1080)

TABLE VI

MEASURED DECODING PERFORMANCE FOR H.264 AND MPEG2 APPLICATIONS (CLOCK CYCLES/MACRO BLOCK)

	foreman	mobile	bluesky
H.264	606	634	548
MPEG2	364	309	287

TABLE VII

COMPARISON OF CONTEXT CACHE HARDWARE COST

	Centralized Structure[21]	Proposed Hierarchical Structure	Reduced
CGC(KB)	32	16	50%
CCC(KB)	256	128	50%
Cache Size(KB)	288	144	50%
Cache Area(mm^2)	4.96	2.82	43%
CGRA Area(mm^2)	23.70	21.56	9%

real-time decoding for H.264 at 40 fps and MPEG2 at 60 fps in average under the working frequency of 200 MHz.

B. Comparison and Analysis

1) Compared With Other Context Management Schemes: Compared with the centralized context cache structure in the base architecture, the proposed hierarchical context cache structure is with only half size and 43% circuit area, leading to 9% deduction in the area of the total CGRA, which can be seen from Table VII.

The configuration performance of the proposed context management scheme is compared with other context cache structures and context cache replacement algorithms as listed in Tables VIII and IX for the scenarios of H.264 and MPEG2 decoding, which is measured with the clock cycles required for each macro block.

It can be seen that in the structure of external storage, no context cache is implemented within CGRA so that all contexts should be fetched from the external memory, where the context access consumes much more time than computation, resulting in serious performance degradation. In the base architecture [21], the context cache is organized in a centralized manner to buffer the recently used CG and CC and shared by all RCAs, which suffers from access conflicts between RCAs and has a negative effect to the configuration performance. Due to high context access bandwidth and cache hit ratio, the proposed context management scheme improves the configuration performance by 13.6%–20.5% for H.264 decoding and 35.2%–42.2% for MPEG2 decoding.

As for different cache replacement algorithms, the traditional LRU and LFU algorithms consider the feature of temporal locality or nonuniform access frequency only. By combining the merits of the two algorithms, the proposed hybrid approach shows the configuration performance gain by roughly 10% over the LRU algorithm and more than 30% over the LFU algorithm. It can also be seen that due to the serious effect on the ignorance of temporal locality, the performance of the hierarchical cache context structure with the LFU algorithm even could not beat that of the centralized one in the base architecture.

When it comes to the flexibility among multimedia applications, it can be found in Tables VIII and IX that the proposed

TABLE VIII

COMPARISON OF CONFIGURATION PERFORMANCE FOR H.264 DECODING (CLOCK CYCLES/MACRO BLOCK)

Context Cache Structure	Cache Replacement Algorithm	foreman	mobile	bluesky
External (no cache)	N/A	11574	10111	10860
Centralized Structure [21]	N/A	701	672	689
Proposed Hierarchical Structure	LRU	677	1085	606
	LFU	1085	793	1009
	Proposed Hybrid	606	634	548
Improvement of the proposed context management scheme				
Compared by the centralized		13.6%	20.5%	20.5%
Compared by the hierarchical with LRU		10.5%	13.4%	8.5%
Compared by the hierarchical with LFU		44.1%	32.7%	45.7%

TABLE IX

COMPARISON OF CONFIGURATION PERFORMANCE FOR MPEG2 DECODING (CLOCK CYCLES/MACRO BLOCK)

Context Cache Structure	Context Replacement Strategy	foreman	mobile	bluesky
External (no cache)	N/A	10522	9479	9092
Centralized Structure [21]	N/A	561	504	496
Proposed Hierarchical Structure	LRU	451	376	337
	LFU	796	636	636
	Proposed Hybrid	364	309	287
Improvement of the proposed context management scheme				
Compared by the centralized		13.6%	35.2%	38.6%
Compared by the hierarchical with LRU		10.5%	19.4%	17.8%
Compared by the hierarchical with LFU		44.1%	54.3%	39.1%

context management scheme has the advantage in general and performs better for MPEG2 decoding than H.264 decoding. The reason is twofold. One is that due to the far less computation complexity and smaller context size than H.264, the hierarchical cache structure has higher cache hit ratio for MPEG2 decoding. The other is that due to parallel scheduling, more contexts could be shared between the two RPUs, which are also beneficial to the cache hit ratio.

2) Compared With Other CGRAs: With the proposed context management scheme, our CGRA is compared with others as shown in Table X, whose context caches are structured in the centralized or hierarchical fashion. For the sake of fair comparison, the performances of all CGRAs are evaluated with the workload of H.264 decoding. It can be seen that our CGRA outperforms others with a 1.3–8× speed up to achieve the decoding frame rate of 40fps at1080 p.

The proposed context management scheme shows significant advantage from two aspects. First, due to the hierarchical context cache structure, the context cache size is decreased considerably in average PE array scale. Since the storage requirement for context cache is closely related to the scale of PE array, the size of context cache for each PE can be considered as an important factor for the context cache structure, which is as large as 2.3–6× in other CGRAs compared with ours. It can be seen that, compared with those relatively small scale CGRAs, the proposed hierarchical context cache structure offers merits especially for large-scale one. Second, the hybrid context replacement strategy improves the performance with the utilized context cache storage. In spite of the difference in the working frequency, the context cache size

TABLE X
COMPARISON WITH OTHER CGRAS

CGRA	ADRES[3]	XPP-III[6]	REMUS-HPP[23]	REMUS-LPP[23]	RP[24]	This work
Context Cache Structure	Centralized	Hierarchical	Centralized	Centralized	Centralized	Hierarchical
PE Array Scale	4×4	5×8	8×8×8	4×8×8	4×8×8	8×8×8
Process(nm)	90	90	65	65	N/A	65
Frequency (MHz)	300	450	200	75	333	200
H.264 Decoding Performance	30fps@D1 (1/8.0×)	24fps@1080p (1/1.7×)	30fps@1080p (1/1.3×)	35fps@D1 (1/6.9×)	30.5fps@1080p (1/1.3×)	40fps@1080p (1×)
Context Cache Size(KB)	27	N/A	288	144	162.5	144
Context Cache Size per PE(KB)	1.69(6.0×)	N/A	0.56(2.0×)	0.56(2.0×)	0.63(2.3×)	0.28(1×)
Cache Efficiency (MBs/s/MHz/KB)*	5.0(1/2.3×)	N/A	4.2(1/2.7×)	4.4(1/2.6×)	4.6(1/2.5×)	11.3(1×)

*Number of macro blocks decoded per second per MHz for each Kbytes of context cache

and the decoding throughput, the proposed work performs 2.3–2.7× better than others in the factor of cache efficiency, which shows the H.264 decoding performance normalized by the working frequency and cache size.

VIII. CONCLUSION

This paper presents a novel context management scheme to improve the performance of context cache and optimize the reconfiguration efficiency of large-scale CGRAs. In order to implement this optimization approach, the analysis of reconfiguration features on large-scale CGRAs is analyzed and illustrated. The proposed context management scheme can optimize the reconfiguration process from two aspects: first, a hierarchical context cache structure is adopted to improve the cache hit ratio and reduce the hardware cost; then, a hybrid context cache replacement strategy is proposed. Comparison results show that, with the proposed context management mechanism, the configuration performance can be improved with lower hardware cost.

REFERENCES

[1] T. Cervero *et al.*, "Survey of reconfigurable architectures for multimedia applications," *Proc. SPIE*, vol. 7363, pp. 736303-1–736303-12, May 2009.

[2] H. Amano, "A survey on dynamically reconfigurable processors," *IEICE Trans. Commun.*, vol. E89-B, no. 12, pp. 3179–3187, 2006.

[3] B. Mei, B. De Sutter, T. Vander Aa, M. Wouters, A. Kanstein, and S. Dupont, "Implementation of a coarse-grained reconfigurable media processor for AVC decoder," *J. Signal Process. Syst.*, vol. 51, no. 3, pp. 225–243, 2008.

[4] B. Mei, F.-J. Veredas, and B. Masschelein, "Mapping an H.264/AVC decoder onto the ADRES reconfigurable architecture," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2005, pp. 622–625.

[5] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architectural exploration of the ADRES coarse-grained reconfigurable array," in *Proc. Int. Workshop Appl. Reconfigurable Comput.*, 2007, pp. 1–13.

[6] M. K. A. Ganesan, S. Singh, F. May, and J. Becker, "H.264 decoder at HD resolution on a coarse grain dynamically reconfigurable architecture," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2007, pp. 467–471.

[7] F. Campi *et al.*, "RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip," in *Proc. Int. Conf. Syst.-Chip*, 2009, pp. 110–113.

[8] P. Dai, X. Wang, and X. Zhang, "Implementation of H.264 algorithm on reconfigurable processor ReMAP," in *Proc. IEEE Asia–Pacific Conf. Postgraduate Res. Microelectron. Electron.*, Jan. 2009, pp. 237–240.

[9] M. Lanuzza, S. Perri, and P. Corsonello, "MORA: A new coarse-grain reconfigurable array for high throughput multimedia processing," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simulation*, 2007, pp. 159–168.

[10] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[11] B. De Sutter, P. Raghavan, and A. Lambrechts, "Coarse-grained reconfigurable array architectures," in *Handbook of Signal Processing Systems*. New York, NY, USA: Springer, 2010.

[12] Z. Li, "Configuration management techniques for reconfigurable computing," Ph.D. dissertation, Dept. Comput. Eng., Northwestern Univ., Evanston, IL, USA, 2002.

[13] S. M. A. H. Jafri, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, "Compression based efficient and agile configuration mechanism for coarse grained reconfigurable architectures," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops PhD Forum*, May 2011, pp. 290–293.

[14] S. Park and K. Choi, "An approach to code compression for CGRA," in *Proc. Asia Symp. Quality Electron. Design*, 2011, pp. 240–245.

[15] B. Wu, L. Yan, Y. Wen, and T. Chen, "Run-time configuration prefetching to reduce the overhead of dynamically reconfiguration," in *Proc. IEEE Int. SOC Conf.*, Sep. 2010, pp. 305–308.

[16] J. Resano, D. Mozos, D. Verkest, and F. Catthoor, "A reconfigurable manager for dynamically reconfigurable hardware," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 452–460, Sep./Oct. 2005.

[17] S. Yin, D. Liu, Y. Peng, L. Liu, and S. Wei, "Improving nested loop pipelining on coarse-grained reconfigurable architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 2, pp. 507–520, Feb. 2016.

[18] S. Yin, X. Yao, D. Liu, L. Liu, and S. Wei, "Memory-aware loop mapping on coarse-grained reconfigurable architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 5, pp. 1895–1908, May 2016.

[19] S. Masayasu *et al.*, "A cost-effective context memory structure for dynamically reconfigurable processors," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2006, pp. 125–132.

[20] F. Thoma *et al.*, "MORPHEUS: Heterogeneous reconfigurable computing," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2007, pp. 409–414.

[21] B. Liu *et al.*, "Reconfiguration process optimization of dynamically coarse grain reconfigurable architecture for multimedia applications," *IEICE Trans. Inf. Syst.*, vol. E95-D, no. 7, pp. 1858–1871, 2012.

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

[23] L. Liu *et al.*, "An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding," *IEEE Trans. Multimedia*, vol. 17, no. 5, pp. 1706–1720, Oct. 2015.

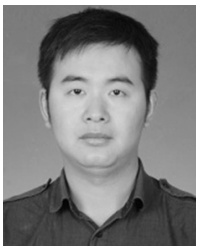
[24] M. Kim, J. H. Song, D.-H. Kim, and S. Lee, "Hybrid partitioned H.264 full high definition decoder on embedded quad-core," *IEEE Trans. Consum. Electron.*, vol. 58, no. 3, pp. 1038–1044, Sep. 2012.

Peng Cao received the B.S. and Ph.D. degrees in microelectronics from Southeast University, Nanjing, China, in 2002 and 2010, respectively.

He is currently an Associate Professor with National ASIC System Engineering Research Center, Southeast University. His current research interests include reconfigurable computing and related VLSI designs.

Bo Liu received the B.S., M.S., and Ph.D. degrees in electronic engineering from Southeast University, Nanjing, China, in 2006, 2008, and 2013, respectively.

His current research interests include digital signal processing, reconfigurable computing, and related VLSI designs.



Jinjiang Yang received the M.S. degree from the School of Electronic Science and Engineering, Southeast University, Nanjing, China, in 2011, where he is currently pursuing the Ph.D. degree with National ASIC System Engineering Research Center.

His current research interests include reconfigurable computing and multimedia processing.

Jun Yang received the B.S., M.S., and Ph.D. degrees in electronic engineering from Southeast University, Nanjing, China, in 1999, 2001, and 2004, respectively.

He is currently a Research Fellow and the Chairman of the SoC Department, National ASIC system Engineering Research Center, Southeast University. His current research interests include chip architecture design and VLSI design.

Meng Zhang received the B.S. degree from the China University of Mining and Technology, Xuzhou, China, in 1986, and the M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1993 and 2014, respectively.

He is currently a Professor with National ASIC System Engineering Research Center, Southeast University. His current research interests include digital signal and image processing, and digital integrated circuit design.

Longxing Shi received the B.S., M.S., and Ph.D. degrees in electronic engineering from Southeast University, Nanjing, China, in 1984, 1987, and 1992, respectively.

He is currently a Professor with Integrated Circuit (IC) College, Southeast University. His current research interests include system-on-a-chip design, VLSI design, and power IC design.