ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.14, No.6, 1–10

Coarse-grained reconfigurable architecture with hierarchical context cache structure and management approach

Chao Wang, Peng Cao, Bo Liu, and Jun Yang[a)]

National ASIC System Engineering Research Center, Southeast University, Nanjing, People's Republic of China, 210096

a) *dragon@seu.edu.cn*

Abstract: This paper proposes a novel coarse-grained reconfigurable array (CGRA) with hierarchical context cache structure and efficient cache management approaches, including time-frequency weighted (TFW) context cache replacement strategy and context multi-casting (CMC) mechanism. By fully exploiting inherent configuration features, the configuration performance is improved by 18.2% with half context memory cost. Our CGRA was implemented under the process of TSMC 65 nm, which can work at the frequency of 200 MHz with the area of 23.2 mm^2. Compared to the previous CGRAs, our work has the advantage of 3.8∼12× performance improvement and 2.3∼15.7× energy efficiency increase.

Keywords: coarse-grained reconfigurable architecture, context cache structure, context management mechanism

Classification: Integrated circuits

References

[1] T. Cervero, *et al.*: "Survey of reconfigurable architectures for multimedia applications," Proc. SPIE **7363** (2009) 736303 (DOI: 10.1117/12.821713).

[2] B. D. Sutter, *et al.*: *Coarse-Grained Reconfigurable Array Architectures* (Springer, 2010) 449 (DOI: 10.1007/978-1-4419-6345-1).

[3] H. Singh, *et al.*: "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Trans. Comput. **49** (2000) 465 (DOI: 10.1109/12.859540).

[4] B. Mei, *et al.*: "Mapping an H.264/AVC decoder onto the ADRES reconfigurable architecture," FPL (2005) 622 (DOI: 10.1109/FPL.2005.1515799).

[5] B. Mei, *et al.*: "Implementation of a coarse-grained reconfigurable media processor for AVC decoder," J. Signal Process. Syst. **51** (2008) 225 (DOI: 10.1007/s11265-007-0152-8).

[6] M. Motomura: "A dynamically reconfigurable processor architecture," Microprocessor Forum (2002).

[7] M. K. A. Ganesan, *et al.*: "H. 264 decoder at HD resolution on a coarse grain dynamically reconfigurable architecture," FPL (2007) 467 (DOI: 10.1109/FPL.

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

2007.4380691).
[8] F. Campi, *et al.*: "RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip," SOC (2009) 110 (DOI: 10.1109/SOCC.2009.5335665).
[9] F. Thoma, *et al.*: "MORPHEUS: Heterogeneous reconfigurable computing," FPL (2007) 409 (DOI: 10.1109/FPL.2007.4380681).
[10] D. Liu, *et al.*: "Polyhedral model based mapping optimization of loop nests for CGRAs," DAC (2013) 19 (DOI: 10.1145/2463209.2488757).
[11] Y. Kim, *et al.*: "Improving performance of nested loops on reconfigurable array processors," ACM Trans. Archit. Code Optim. **8** (2012) 32 (DOI: 10.1145/2086696.2086711).
[12] S. Yin, *et al.*: "Improving nested loop pipelining on coarse-grained reconfigurable architectures," IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **24** (2016) 507 (DOI: 10.1109/TVLSI.2015.2400219).
[13] J. Lira, *et al.*: "LRU-PEA: A smart replacement policy for non-uniform cache architectures on chip multiprocessors," ICCD (2009) 275 (DOI: 10.1109/ICCD.2009.5413142).
[14] B. Ackland, *et al.*: "A single-chip, 1.6-billion, 16-b MAC/s multiprocessor DSP," IEEE J. Solid-State Circuits **35** (2000) 412 (DOI: 10.1109/4.826824).
[15] 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. **22** (2014) 983 (DOI: 10.1109/TVLSI.2013.2263155).

1 Introduction

Coarse-grained reconfigurable architecture (CGRA) has been proven to be a promising solution for computing intensive applications between the extremes of general purpose processor (GPP) and application specific integrated circuit (ASIC), which has higher performance level than GPP and wider applicability than ASIC [1]. CGRA supports dynamic reconfiguration at runtime [2], which not only boosts the performance but also can be reconfigured to adapt different characteristics of applications. However, due to this feature, the performance of CGRA is heavily affected by the efficiency of configuration process. As the scale of reconfigurable array increases, configuration management becomes more critical because more reconfigurable resources are required to be fed with configuration contexts.

To solve this problem, context cache, also called context memory, is usually implemented with the reconfigurable array to provide local storage for recently used configuration contexts, which plays an important role for configuration performance with memory and power overhead. Many kinds of CGRAs have been proposed in recent years by employing context cache, whose structures can be classified into three categories including the centralized, the distributed and the hierarchical. In MorphoSys [3] and ADRES [4, 5], a centralized context cache is shared by all reconfigurable resources. In sprit of the advantage of low hardware complexity for supervision, the configuration performance degrades heavily due to access conflict in the configuration cache. For the distributed context cache as in DRP-1 [6], each PE array access is coupled with an individual cache. Though all PE arrays can access contexts in parallel with high throughput, context redundancy

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

is unavoidable among these context caches, leading to unnecessary hardware overhead in context caches. The hierarchical context cache combines the merits of the centralized and distributed in terms of satisfying configuration performance and low hardware overhead. In CGRAs as XPP [7, 8] and MORPHEUS [9], multiple context caches are implemented in different level of PE arrays, where the main design concern focuses on the sizes and numbers of caches as well as the cache supervision scheme.

In this paper, we propose a novel multi-array CGRA with hierarchical context cache structure for high performance and efficiency. The context cache is structured hierarchically with different bandwidths and sizes in each level to increase the throughput while restrict the hardware cost. Two context management approaches, namely time-frequency weighted (TFW) context cache replacement and context multi-casting (CMC), are employed to improve the cache performance by adapting the features of configuration context.

The rest of this paper is organized as follows. The architecture of the proposed CGRA is described in Section 2 with the hierarchical context cache and its configuration process. The context cache management solutions are presented in Section 3 for the hierarchical context cache by analyzing the features of configuration context. Section 4 demonstrates the improvement of the proposed context cache design compared with the prior work. Finally the conclusion is made in Section 5.

2 Multi-array CGRA with hierarchical context cache structure

In this section, we present a multi-array CGRA with hierarchical context cache structure, whose dynamic configuration process is described as well.

2.1 Multi-array CGRA structure

Similar as the CGRAs of Morphosys [3] and DRP-1 [6], the proposed CGRA is designed hierarchically with multiple reconfigurable arrays to increase the scale of CGRA for higher parallelism. Due to the style of multiple arrays, the CGRA has the advantage of less configuration efficiency degradation when the scale of reconfigurable resources expands.

The block diagram of the CGRA is shown in Fig. 1, which consists of a RISC processor, an external memory interface (EMI) and two homogeneous reconfigurable processing units (RPU). The RISC processor executes control intensive tasks and hosts the whole CGRA whereas the RPUs execute data intensive kernels. The EMI is responsible of the data and configuration contexts transmission from and to the external memory. The RPU incorporates four reconfigurable arrays (RCA). Each RCA contains an 8×8 array of processing elements (PE) with 16-bit data width for arithmetic and logical operations.

2.2 Hierarchical context cache structure

Besides the hardware structure of CGRA, the configuration process plays an increasingly important role in terms of improving performance and reducing power consumption [10, 11, 12]. The proposed CGRA features dynamic configuration,

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

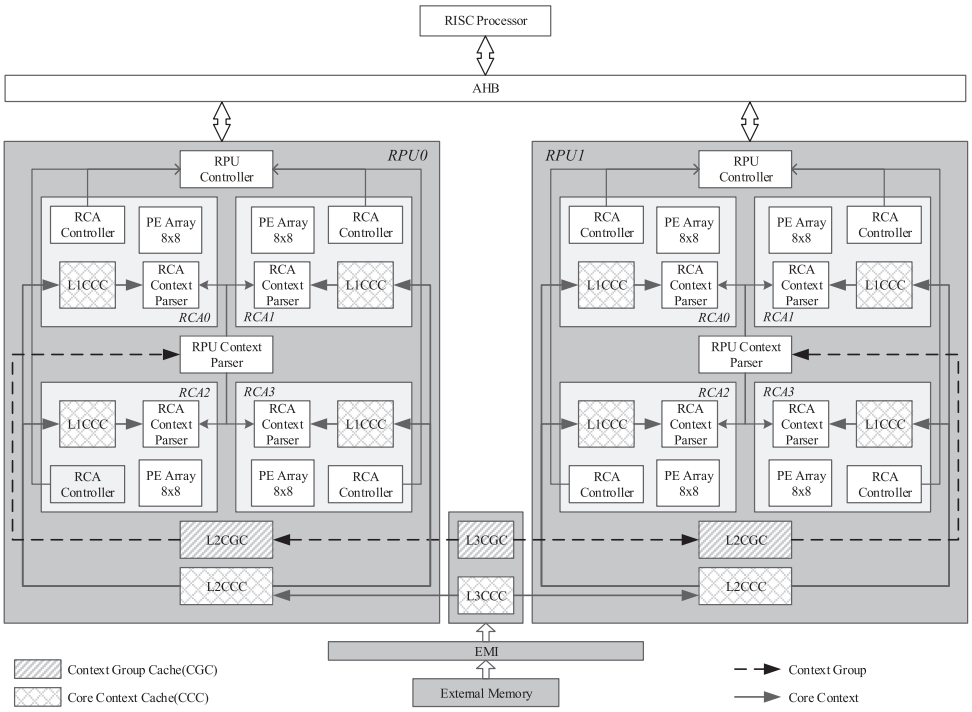

Fig. 1. Diagram of CGRA with hierarchical context cache structure

where no more configuration contexts are updated during its computation. However, the configuration procedure of the following task can be performed in advance along with the computation of the current one in pipeline so as to improve the performance of the whole CGRA.

The configuration contexts are classified into the context group (CG) and the core context (CC). The CC represents the configuration of a iteration mapped to RCA, which occupies 512 bytes for the 8×8 PE array. The CG describes which RCA is mapped to and consists of a sequence of the CC indexes, which is responsible for the configuration of a kernel task. The size of CG varies according to the number of CCs in the sequence, which is 256 bytes at most.

Both configuration contexts are stored in the context cache structure hierarchically as shown in Fig. 1. The cache for CG is composed of L2CGC (level 2 CG cache) and L3CGC (level 3 CG cache), whereas that for CC consists of L1CCC (level 1 CC cache), L2CCC (level 2 CC cache) and L3CCC (level 3 CC cache). As the lowest level of context cache, the L1 context cache (L1CCC) is implemented within RCA and used as a temporal repository of the CCs used recently. The L2 context caches (L2CGC and L2CCC) are attached to RPU and shared by the RCAs inside the RPU. The L3 context caches (L3CGC and L3CCC) are shared by the RPUs.

The configuration procedure is processed in two stages. Firstly, the CG is loaded from the CGC hierarchy and parsed by the RPU context parser to determine the mapped RCA and the required CCs for it. When the required 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. After that, the RCA context parser fetches the CCs in sequence by

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

their indexes to configure the PE operations and interconnections in RCA. The hierarchical CCC is accessed level by level similarly as the CGC.

In order to increase the hardware efficiency of the hierarchical context cache structure, the lower level of cache is designed with higher access bandwidth and smaller cache size. Considering the scale of RCA and the bandwidth of the external memory interface, the bandwidths and sizes of each level are listed in Table I in detail. It can be seen that when the context cache of the lowest level (L2CGC/L1CCC) hits, the CG and CC can be loaded with high throughput in eight and four clock cycles respectively. The total storage capacity of the context cache hierarchy amounts to 144 K byte for the whole CGRA, including 64 CGs and 256 CCs, which has the advantage of low hardware cost as compared in Table II. For each RCA, the number of contexts in cache is roughly 1/8 of that of ADRES. Moreover, the the storage capacity for each PE is 288 bytes in average, which is only 16.7% of ADRES.

Table I. Bandwidths and sizes of context cache hierarchy

	Context Group Cache		Core Context Cache			External Memory
	L2CGC	L3CGC	L1CCC	L2CCC	L3CCC	
Bandwidth (bits/cycle)	256	128	1024	512	256	64
# of contexts	16	32	16	32	64	-
Size (KB)	4	8	8	16	32	-
# of total contexts	64		256			-
Total Size (KB)	16		128			-

Table II. Context cache overhead comparison

	#/Array	Size/PE
ADRES [5]	256	1728 Bytes
Ours	8 CGs/RCA 32 CCs/RCA	288 Bytes

3 Context cache management approaches based on configuration context feature

In this section, two novel management approaches for the hierarchical context cache structure are proposed based on the analysis of the configuration context features.

3.1 Time-frequency weighted (TFW) cache replacement strategy

The cache replacement strategy plays an important role in cache performance improvement. For GPP, various cache replacement algorithms have been proposed such as round-robin, least recently used (LRU) [13], least frequently used (LFU) [14] and so on. However, since the size of configuration context is much larger than that of instruction in GPP, traditional approaches would suffer serious reconfiguration performance degradation when cache misses.

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

The configuration context features the temporal locality and nonuniform access frequency for both CG and CC. The temporal locality shows the objective regularity of configurations when they are performed frequently during their lifetime, which can be demonstrated from the fact the same CG and CC can be invoked frequently in a short time. However, from the global perspective, the access frequencies of configuration contexts vary not only for different tasks but also for different periods. Therefore, the cache performance would deteriorate if keeping the frequency as a constant for a certain context regardless of task switch.

Algorithm 1 TFW cache replacement algorithm

Require: cache holds m valid elements currently

if cache hits ctx at position i **then**
  $ele[i].cnt \Leftarrow ele[i].frq \times tfwf$
  for all $j \neq i$ **and** $j \leq m$ **do**
    $ele[j].cnt \Leftarrow ele[j].cnt + 1$
  end for
else {cache misses}
  if cache is not full **then**
    $ele[m+1].ctx \Leftarrow ctx.dat$ {cache the context at position $m+1$}
    $ele[m+1].addr \Leftarrow ctx.addr$
    $ele[m+1].frq \Leftarrow ctx.frq$
    $ele[m+1].cnt \Leftarrow ctx.frq \times tfwf$
    for all $j \leq m$ **do**
      $ele[j].cnt \Leftarrow ele[j].cnt + 1$
    end for
  else {cache full}
    find x with the maximum of $ele[x].cnt$ {selected as the victim}
    $ele[x].ctx \Leftarrow ctx.dat$ {cache the context at position x}
    $ele[x].addr \Leftarrow ctx.addr$
    $ele[x].frq \Leftarrow ctx.frq$
    $ele[x].cnt \Leftarrow ctx.frq \times tfwf$
    for all $j \neq i$ **do**
      $ele[j].cnt \Leftarrow ele[j].cnt + 1$
    end for
  end if
end if

To adapt with the above configuration context features, a time-frequency weighted (TFW) context cache replacement algorithm is proposed for the hierarchical context cache structure, which combines the merits of LRU and LFU algorithms. Besides the content data of the contexts (*dat*), each element of cache consists of the address (*addr*), the frequency flag (*frq*) and the counter for the context (*cnt*). A time-frequency weighted factor, denoted as *tfwf*, is suggested to trade off between the features of temporal locality and nonuniform access frequency.

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

The pseudo code of the TFW algorithm is shown in Alg. 1, where the currently needed context is noted as *ctx* and the updated element of cache is *ele*. The initialization value of *cnt* is defined as the product of *frq* and *tfwf* instead of zero as in the traditional LRU algorithm, which is to take the average access frequency into account adjusted by *tfwf*.

The presented TFW algorithm was implemented and integrated in the RCA/RPU context parsers as illustrated in Fig. 2, which fetches the required context from the lower level of context cache, $Cache_n$, and updates the stored context with the higher level of context cache, $Cache_{n+1}$, when cache misses. The structure of the TFW replacement controller is composed of three parts. *HitChecker* is responsible for checking if the required context hits in $Cache_n$ by comparing the context address with those stored in $Cache_n$. *VicPosFinder* calculates the position of the victim when cache misses. *CacheUpdater* is used to update the cache element for the missed context by reading it from $Cache_{n+1}$ and increase the counters of all other elements.

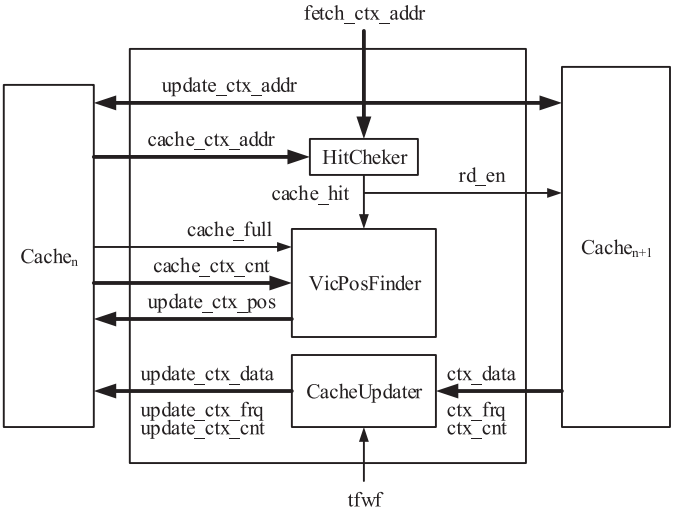

Fig. 2. Hardware structure of TWF replacement controller

3.2 Context multi-casting (CMC) mechanism

For the tasks with larger computation complexity than the scale of RCA, they can be divided into multiple sub-tasks and mapped to multiple RCAs in parallel to break though the resource limitation of RCA. In this case, the configurations are same among different RCAs, which is called the feature of context sharing.

Therefore, in order to reduce the configuration overhead, a context multi-casting (CMC) mechanism is employed during context access. Before the loading of CG, the RPU context parser parses the index of CG and the mapped RCA in advance. If the successive CGs are checked to be the same but mapped to different RCAs, they can be loaded only once, which reduces the transmission time of CG and starts the loading of CCs ahead of time.

The configuration processes with and without CMC mechanism are illustrated and compared in Fig. 3. As an example, the same CG of four sub-tasks is shared by

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

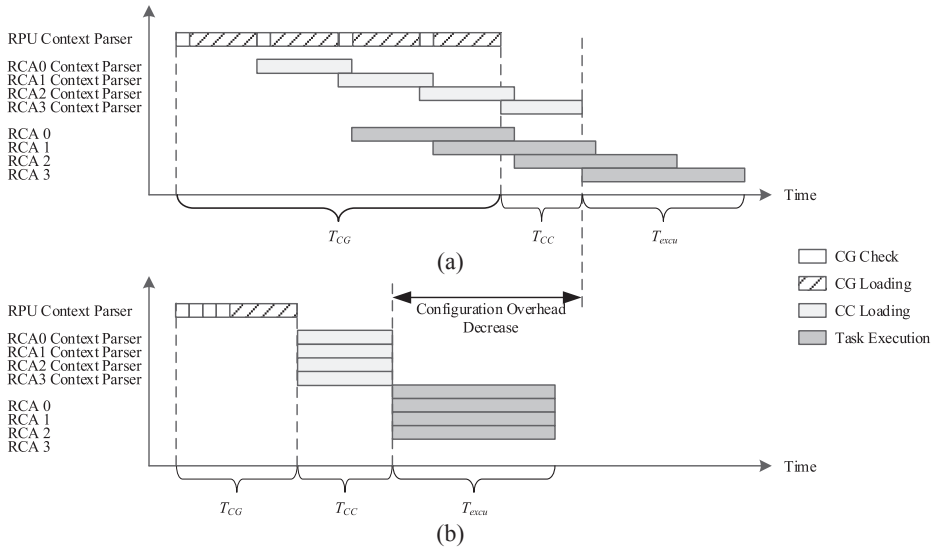

Fig. 3. Process of configuration and execution.
(a) Without CMC mechanism;
(b) With CMC mechanism

and configured to four RCAs. As shown in Fig. 3(a), without regard to the feature of context sharing, the CGs are loaded by the RPU context parser in sequence. Hence, multiple CG loading processes are performed by each RCA. The loading of CC is accordingly suspended waiting for the completion of CG transmission. By adopting the CMC mechanism, the CGs are checked by the RPU context parser before loading. If the successive CGs are with identical content for different RCAs, these CGs are fetched from the CGC by only once and multi-casted to four RCAs simultaneously, as shown in Fig. 3(b). The same CCs in the four RCAs can also be loaded earlier in parallel. Owing to the CMC mechanism, the configuration overhead composed of T_{CG} and T_{CC} is decreased significantly by reducing the overhead of CG loading to 1/4. Though the start time of the first CG loading is delayed, the task divided into four RCAs complete executing in advance along with the configuration process.

4 Experimental results and comparison

The proposed CGRA with hierarchical context cache structure and context management mechanism were described with VerilogHDL language and synthesized under the process of TSMC 65 nm. The timing and area results were reported by Synopsys Design Compiler (DC) using the typical case, which showed that this CGRA can work at the frequency of 200 MHz with the area of 23.2 mm^2. The context caches were implemented by SRAM Macro Cell library with the area size of 2.82 mm^2, which occupied 12.2% of the total area. The RCA/RPU context parsers, which are in charge of the context cache management including TWF and CMC, are area efficient with less than 2% hardware resource overhead.

The configuration overhead with the optimization approaches is compared with [15] in Table III by taking H.264 high-profile decoding as an example. The architecture of the proposed CGRA is similar with that in [15], where two RPUs were implemented and composed of four 8 × 8 RCAs. However, the configuration

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

context in [15] was stored in a centralized style with the total size of 288 KB. Compared with that design, the size of the hierarchical context cache structure is reduced by half. With the TFW context cache replacement strategy, the clock cycles for the configuration of each macro block is reduced by 12.3% in average. By further adopting the CMC mechanism, more 5.9% clock cycles can be saved due to the context sharing among RCAs.

Table III. Configuration overhead comparison (clock cycles/macro block)

	forman	mobile	bluesky	Average	Reduction
Centralized structure [15]	701	672	689	687	-
Hierarchical structure with TFWA	636	573	601	603	12.3%
Hierarchical structure with TFWA and CMC	606	534	548	562	18.2%

The proposed CGRA is compared with two related design targeting at multimedia applications in terms of performance, area and power consumption, as shown in Table IV. Owing to the remarkable configuration performance improvement, our CGRA can realize real-time H.264 high-profile decoding at the speed of 1080p@40 fps with the power consumption of 364 mW. ADRES [5] was designed to perform real-time (30 fps) H.264 baseline profile D1 decoding, which used a centralized context cache called configuration RAM to hold the current active configuration context for certain tasks. The architecture of XPP [7] was based on a scalable array attached with the configuration managers (CM), which was composed of hierarchical context caches including supervising CM and distributed CMs. It can decode H.264 main profile HD stream at the frame rate of 24 fps. It can be seen in Table IV that our CGRA has the advantage of 12× and 3.8× normalized performance compared with ADRES and XPP-III due to the increase of config-

Table IV. Comparison with other CGRAs

	ADRES [5]	XPP-III [7]	Proposed
Context Cache Structure	Centralized	Hierarchical	Hierarchical
Process (nm)	90	90	65
Area (mm^2)	64	75	23.2
Frequency (MHz)	300	450	200
Stream Profile	Baseline Profile	Main Profile	High Profile
Performance	720 × 480@30 fps	1920 × 1080@24 fps	1920 × 1080@40 fps
Normalized Performance (MBs/s/MHz)	135	432	1620
Normalized Performance Comparison	$\frac{1}{12}\times$	$\frac{1}{3.8}\times$	1×
Power (mW)	105.5	3420	364
Energy Efficiency (MBs/s/mW)	393.9	56.8	890.1
Normalized Energy Efficiency Comparison	$\frac{1}{2.3}\times$	$\frac{1}{15.7}\times$	1×

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017

uration efficiency. Moreover, the proposed CGRA offers 2.3× and 15.7× energy efficiency enhancement with pipelined reconfiguration scheme.

5 Conclusion

This paper presents a multi-array CGRA with hierarchical context cache structure to increase configuration throughput and reduce context cache overhead. Two context cache management mechanisms are further implemented to improve the configuration performance. The proposed CGRA were implemented under the process of TSMC 65 nm and can support real-time H.264 decoding application at the speed of 1080p@40 fps. Experimental results show that the proposed context cache management improves the configuration performance by 18.2% with half context cache size. Compared with related works, our CGRA has the advantage of 3.8∼12× performance increase and 2.3∼15.7× energy efficiency improvement.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 61404028, 61574033 and 61550110244), Natural Science Foundation of Jiangsu Province (No. BK20161147) and China Scholarship Council.

DOI: 10.1587/elex.14.20170090
Received February 2, 2017
Accepted February 22, 2017
Publicized March 7, 2017
Copyedited March 25, 2017