PAPER

A Data Prefetch and Reuse Strategy for Coarse-Grained Reconfigurable Architectures

Wei GE[†a)], ***Member***, **Zhi QI**[†b)], **Yue DU**[†], **Lu MA**[†], ***and*** **Longxing SHI**[†], ***Nonmembers***

SUMMARY The *Coarse Grained Reconfigurable Architectures* (CGRAs) are proposed as new choices for enhancing the ability of parallel processing. Data transfer throughput between Reconfigurable Cell Array (RCA) and on-chip local memory is usually the main performance bottleneck of CGRAs. In order to release this stress, we propose a novel data transfer strategy that is called Heuristic Data Prefetch and Reuse (HDPR), for the first time in the case of explicit CGRAs. The HDPR strategy provides not only the flexible data access schedule but also the high data throughput needed to realize fast pipelined implementations of various loop kernels. To improve the data utilization efficiency, a dual-bank cache-like data reuse structure is proposed. Furthermore, a heuristic data prefetch is also introduced to decrease the data access latency. Experimental results demonstrate that when compared with conventional explicit data transfer strategies, our work achieves a significant speedup improvement of, on average, 1.73 times at the expense of only 5.86% increase in area.

key words: *coarse-grained reconfigurable architectures, data prefetch, data reuse*

1. Introduction

Coarse-grained reconfigurable architectures (CGRAs), as attractive alternatives, have become more and more popular in both academic research and commercial applications in the past few years [1]–[6]. The combination of the software flexibility and high performance of hardware contributes to the better balance of CGRAs among the key metrics, such as performance, energy efficiency and chip area. Besides, CGRAs also present the advantages of large distributed registers and a rich topology. With the numerous hardware computing resources, the Reconfigurable Cell Array (RCA) has to handle a large amount of high-speed parallel tasks, which results in great demand for a massive data throughput of CGRAs. Therefore, a data transfer strategy efficient both in throughput and speed is particularly needed to satisfy the requirement [7].

The existing data transfer mechanism between RCA and local memory can be classified into types of explicit [8]–[12] or implicit [13]–[15] approaches. Whether the RCAs have the ability of fetching the computing data initiatively is the main difference between these two. Those reconfigurable cells (RCs) [8], [9] or simple programmable processors [11], [12] that explicitly access the memory with a certain flexibility are easy to program. However, their operations of load/store and the calculation of data address consume a considerable number of RCs. Implicit data transfer mechanism, by contrast, does not support explicit load/store instructions. Instead, data has to be prearranged in a specific order [13] in the local memory, before it is broadcasted to the CGRA at a certain delay [14]. This method not only exhibits more challenges to programmers but also is more expensive due to the special data arrangement.

Taking advantage of their easy programming and flexible access, explicit CGRAs have been selected as our research objects. However, dealing with the massive data transfer between RCA and local memory is still a serious problem. Explicit CGRAs of ADRES [9] typically hide the latency of memory accesses by module mapping, but multi-access requirements in a queue may result in stalls of the system. Others that implement multi-bank local memory [8] have to deal with data transfer between different banks respectively. These operations increase the latency and number of memory accesses thus greatly impact the performance of data transfer.

Inspired by the work of implicit CGRAs, *e.g.* a double frame buffer that overlaps the data access with computation in MorphoSys [13] and a block buffer reducing memory access in REMUS-II [15], we consider to solve these limitations by exploiting data prefetch and reuse strategy. The technique of data prefetch proposed by IBM [16] hides the long data transfer latency from the external memory through overlapping the data access procedure with the computational operation. Data reuse efficiently reduces the external memory access and thus not only accelerates the data transfer speed but also reduces the energy overhead of communication.

To the best of our knowledge, data prefetch and reuse strategies have not yet been successfully applied to explicit CGRAs. To keep the merit of data access flexibility, the easy data arrangement with comfortable programming, and to obtain a high efficiency of data transfer, we propose a novel data prefetch and reuse mechanism, which is called Heuristic Data Prefetch and Reuse (HDPR), in our explicit CGRAs. The outstanding features of our work include: in the environment of explicit CGRAs, 1) we overlapped computing phase and data access phase through a data prefetch mechanism; 2) decreased number of accesses to local memory by an effective data reuse mechanism; 3) improved parallel access ability of the RCA at a little hardware cost.

The rest of this paper is organized as follows. Section 2

Manuscript received June 15, 2012.
Manuscript revised October 22, 2012.

[†]The authors are with National ASIC System Engineering Research Center, Southeast University, Nanjing, China.
a) E-mail: duiker@seu.edu.cn
b) E-mail: zhiqi.cim@gmail.com

DOI: 10.1587/transinf.E96.D.616

introduces the related work. Section 3 gives a base architecture. Section 4 explains the motivation of improvement for data access mechanism. Section 5 proposes the detailed design of data prefetch and reuse strategy. Section 6 shows the experimental results with discussions. Finally, Sect. 7 concludes the paper and presents future work.

2. Related Work

Data prefetch is a state-of-art dominant technique to improve the efficiency of data transfer in most of implicit CGRAs. MorphoSys [13] overlapped computation with data transfers by extending extra DMA instructions in TinyRisc. However, the data pre-arrangement under the control of TinyRisc increased the complexity of data management. DREAM [14] retrieved highly parallelized data through adopting a programmable Address Generators (AGs) to generate address with fixed stride after a certain delay due to the required computation of applications. Whereas, the AGs need to be confirmed at the initial step and couldn't change dynamically at run time. REMUS-II [15] designed multi-mode of data prefetch mechanism especially for the multimedia data, which has the character of 2D-access. This data prefetch mechanism named RIM achieved high data throughput and performance. However, without the strong support of compiler, such an adaptable data access mechanism of CGRAs greatly challenged programmers for the expensive manual data arrangement in a very specific order. Since the complicated data preparation and arrangement embarrass the easy and comfortable usage of the technique of data prefetch, data reuse becomes an alternate choice.

Data reuse not only improves the efficiency of data transmission, but it also reduces the energy consumption of the communication through decreasing the memory accesses. G. Dimitroulakos [17] explored the data reuse on the high bandwidth distributed foreground memory. It stored the reused data values in the local RAMs inside the PEs thus successfully avoided the expensive communication with the outside SRAM. CRM [18] proposed a scratchpad memory with a configurable address space. In this way, CRM reduced the data replication and the explicit management, thus promoted the data reuse. However, the restriction in the coprocessor's interface prevented the author from using a cache as the local memory, which also limited the implementation of data prefetch strategy. REMUS-II adopted a special designed frame buffer [19] with the similar function of shared scratchpad memory in CRM to reduce the access of DRAM. Whereas, its lacking of flexibility and its difficulty in programming contained the application.

It is obvious that combining the merits of the data prefetch and reuse strategies should greatly improve the data transfer efficiency. Thus, we attempted for the first time to integrate these two technologies in explicit CGRAs.

3. Motivation

To overcome the resources constrains due to the limited capacity of RCAs, explicit CGRAs typically spread the computation of a loop iteration over multiple configurations, then apply software pipelining to manage operations dynamically by means of modulo scheduling [20].

As is known, when a loop has been modulo-scheduled, the total *Run Cycle* in inner loop of CGRAs is given by Eq. (1) [20], which consists of three phases: the prologue, the loop kernel and the epilogue.

$$RunCycle = Cycle_{prologue} + (II*(IN-1)) + Cycle_{epilogue} \tag{1}$$

$$\min II = \max(\mathrm{Re}\,sMII, \mathrm{Re}\,cMII) \tag{2}$$

$$IPC = IC/RunCycle \tag{3}$$

RunCycle is not the absolute time for application execution, but an important indicator of computational performance within CGRAs. The smaller *RunCycle* represents the higher efficiency of the task kernel. In Eq. (1), the parameter *IN*, i.e., the iteration number of the algorithm, is definitely algorithm-dependent. Furthermore, both cycle numbers of prologue and epilogue stages have little influence on *RunCycle* in a large loop. Thus, the remaining *II*, i.e., Initial Interval, is the key element to reduce the *RunCycle*.

The value of *II* depends on the data dependence recurrences and the resource constraints as given by Eq. (2). The *ResMII* (resource-minimal *II*) represents the computational resource mapping efficiency under the architecture constraints, which is seriously influenced by the interconnection topology, internal register files and modulo scheduling algorithms. Thus the reduction of *ResMII* is irrelevant to the data transfer strategy proposed in this paper. Nevertheless, *RecMII* (recurrence-minimal *II*) reflects the data dependence in the iteration, the reduction of which can be achieved by eliminating the access latency. Since the discussion about the impact of modulo scheduling algorithms on *II* is beyond the scope of this paper. Here, we care about the benefits of raising the efficiency of task kernels if we decrease *II* by optimizing the access latency.

IPC (Instructions per Cycle) in Eq. (3) represents the resource utilization of RCA in loop kernels, which is decided by the *IC* (Instruction Counts) and *RunCycle* directly.

CGRAs favor those application kernels typically with intensive and regular types of data. The data arrangement of these applications has regular pattern stride and consistent address range. Consequently, these applications provide the possibility of efficient data transfer mechanism in explicit CGRAs. In our cases, we exploit data prefetch and reuse strategies to reduce access latency and number for these applications.

We demonstrate the impact of access latency for *II* using an example of a filter benchmark that implements complex multiplications. Figure 1 illustrates the mapping results of data flow of algorithm with conventional modulo scheduling. It is observed from the traditional scheduled results that each context switch needs to wait till the accomplishment of load operation. As indicated in Fig. 1, one of the two cycles of LDR operation is cost by memory access latency. Therefore, the succeeding operations of Mult and STR in cycle number 2 and 5 respectively are both blocked by the multiple LDR operations in different pipelines. In this case, the multi-data requirements of RCA enormously aggravate the read latency. Under the same modulo scheduling approach, the fixed number of contexts for algorithms makes *ResMII* become a constant. According to the definition of *II* in Eq. (2), the exploration of various methods to decrease the influence of data access latency benefits *RecMII*, which decreases *II* correspondingly.

Data prefetch strategy which applied in the loop of complex_multiply, as in Fig. 2 (b), reduces the access latency or ideally makes it even to zero. The prefetch operation starts once the previous data is valid, as indicated by dashed lines. Such an early LDR operation successfully conceals the one cycle spent on the memory access latency. As a result, not only the LDR operation seems to be one cycle shorter than usual, but also the RCA could immediately grab the data for the subsequent computation.

Data reuse strategy in Fig. 2 (c) deals with intensive data in iteration loop. For example, if the operation data of LDR_0, LDR_1, LDR_2 can be loaded in one operation, then we can cache the data and submit it immediately for the next request at the same address range. In such a manner, data reuse strategy not only removes the memory access latency but also dramatically declines the data access times.

Consequently, comprehensive consideration of the strategy of data prefetch and reuse diminishes both data access latency and number, thus generates more timing profits than the traditional method as obviously demonstrated in Fig. 2 (a).

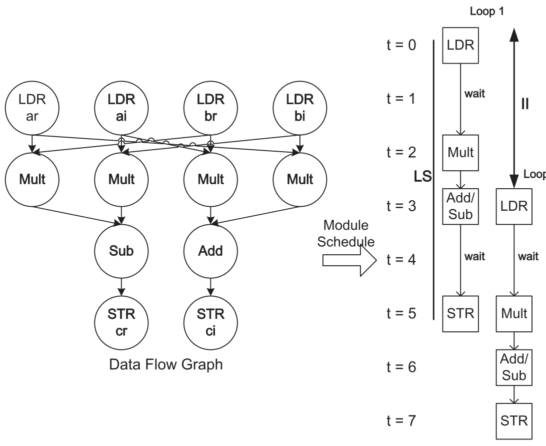


Fig. 1 The conventional modulo scheduling flow. In the figure, the notations LDR, STR represent load and store operation respectively. Mult, Add and Sub indicate the corresponding arithmetical operations. The *ar*, *br*, *cr* and *ai*, *bi*, *ci* are real and imaginary parts of complex respectively.

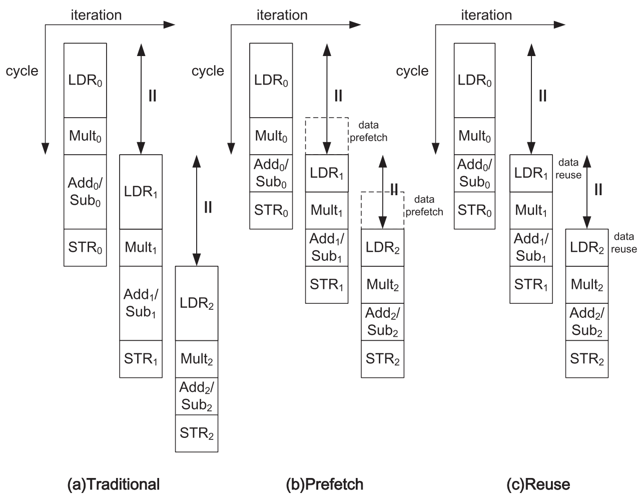


Fig. 2 Demonstrations of different data transfer strategies.

4. Base Architecture

In this section, we introduce base architecture which is used to compare the proposed data transfer mechanism with conventional methods.

The data transfer strategy in base architecture is similar to ADRES [9] and RSPA [10]. The operation between RCA and local memory can be abstracted as all the RCs accessing the local memory initiatively with blocking loads regardless of the various memory architectures. For example, the ADRES adopted a power-efficient, signal-ported, interleaved scratch-pad memory organization [21]. Whereas, the RSPA used multi-bank local memory architecture, in which each bank may be accessed only by the processing elements in the corresponding row. In both cases, the RCs are able to load/store to the local memory. Meanwhile, other RC operations will not be launched until these blocking memory access completed.

The modular design of base architecture reproduces the conventional data access strategy, and can be modified to extend the prefetch and reuse functions. We have implemented the parametric template CGRA as an attached IP connected to system bus in Fig. 3. It is easy to build such a SoC (System on Chip) using a standard processor and a standard reconfigurable array with this kind of coupling style. The proposed prototype system consists of a 32-bit RISC processor with eight pipeline stages, an external memory, an INTC (interrupt controller) and a CGRA. The communication bus is 32 bit AMBA AHB, which couples the RISC processor,

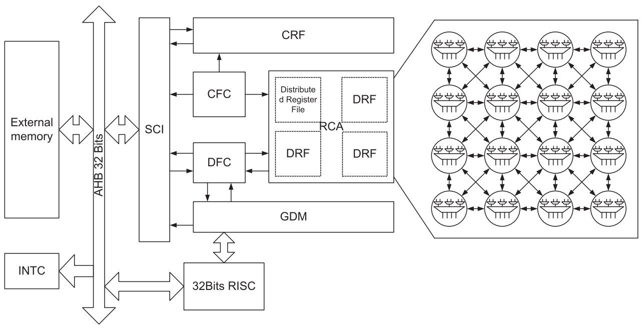


Fig. 3 The top level architecture of base CGRA.

CGRA and other peripheral as INTC.

The considered RCA includes parameterized RC and Distributed Register File (DRF). The parameters of cell number, data width, register depth and interconnection topologies are changeable at various design requirements. Thus, the base architecture is able to reproduce other type of architectures in the experimental part. The System Connect Interface (SCI) connects the CGRA as an attached coprocessor to system bus. Limited by the size, the Context Register File (CRF) only stores several contexts. When the memory size is not enough for all the contexts, extra time has to be paid for loading other contexts from the external configuration memory. The Context Flow Controller (CFC) is responsible for context switch. It receives the configuration switch enable signal from RCA and then updates the contexts correspondingly. The Data Flow Controller (DFC) is in charge of data transfer between Global Data Memory (GDM) and RCA. Finally, the GDM is used for storing both input data that transported from the external memory to RCA and output data generated by the RCA to feed the external memory.

Taking advantage of this base architecture, we are capable to compare the proposed strategy with the existing data transfer strategies.

5. Prefetch and Reuse Strategy

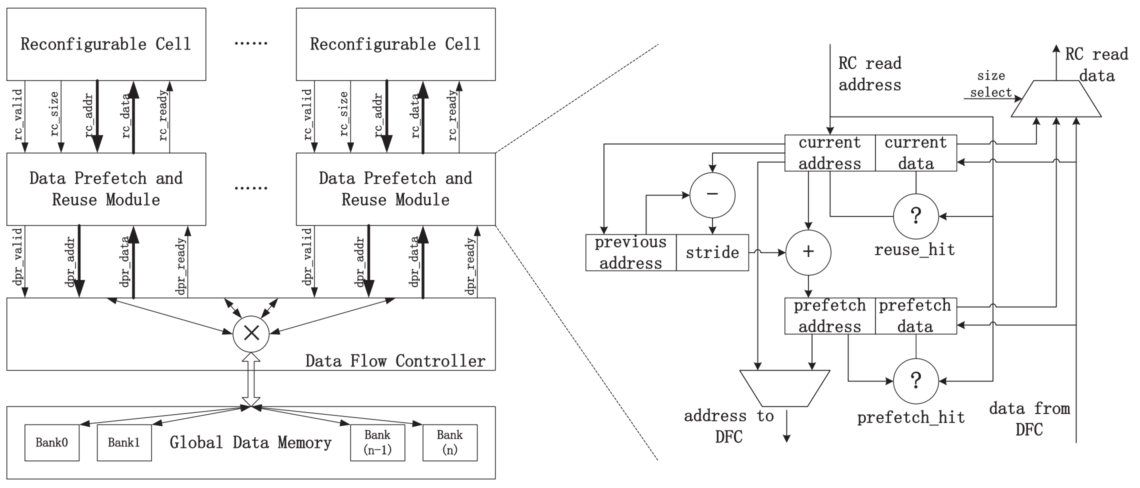
5.1 Design Implementation

In this section, the RC (Reconfigurable Cell) module and the DPR (Data Prefetch and Reuse) module for verifying this HDPR method are described. In Fig. 4 (a), each RC with load/store operation is connected to the DFC (Data Flow Controller) through a DPR. In order to avoid the confliction between multiple non-sequential data requests to DFC, we pipeline these data accesses when they are arranged in a more efficient sequential manner. In our case, all data requirements from the RC have been ordered as a queue to DFC according to priority. The DFC accesses GDM (Global Data Memory) and responses read valid signal to the corresponding DPR.

As the fundamental processing element of the CGRA, a RC is constructed with data multiplexer, ALU (fixed-point operations) and access logic. In addition to complete these basic arithmetic and logical operations, each RC loads or stores data at different data sizes. The communication handshake control signal, the basic data, as well as the address signals can be seen in Fig. 4 (a). The rc_size signal controls the data type of return value. The output signal rc_valid indicates the start of each read operation. And the corresponding input signal rc_ready indicates the completion of read operation. The handshake signals of valid/ready form a closed-loop feedback communication mechanism, which ensures the correctness and robustness of data transmission.

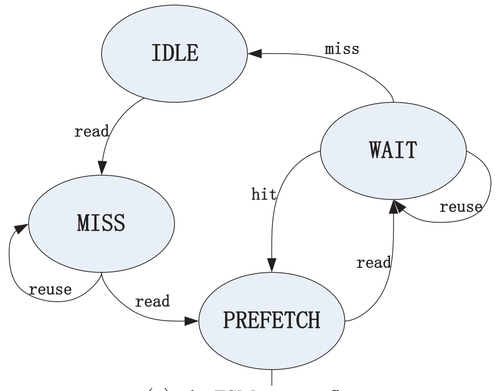
The detail design of DPR is shown in Fig. 4 (b). The offset between the addresses at the current and the previous time instances defines the stride. DPR prepares the data at the distance of a stride unit ahead of the current address. In addition, the stride will not be valid until RC has finished two effective read operations. It means that we need to do normal read operations twice before getting the valid stride value. The data and control signals between DFC and RC are both managed by an internal FSM (Finite State Machine). Considering the data throughput and access latency of multi-port memory in GDM, the buffer width of both current data and prefetch data is 4 times of RC data width. In order to simplify the implementation of hardware design, we set the default parameter of buffer depth to one, which is the minimum value. So the total buffer size of DPR (include both current data and prefetch data) is 8 times of RC data. And the data alignment of both current data and prefetch data is similar to traditional cache line. Rather than the traditional cache that grabs the data in cache line, our DPR heuristic forecasts the possible data in the manner that overlaps the computing and data access efficiently.

The DPR module has four states, i.e., IDLE, MISS, PREFETCH and WAIT as in Fig. 5 (a). The DPR maintains

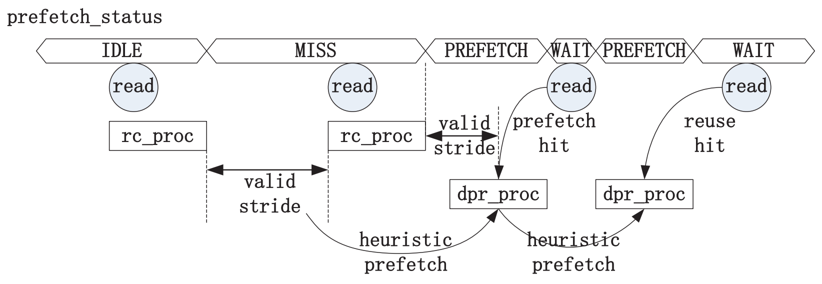


(a) the interconnection of DPR (b) design detail of DPR

Fig. 4 Architecture of data prefetch and reuse module.



(a) the FSM process flow



(b) the illustration of heuristic data prefetch mechanism

Fig. 5 Working mechanisms of DPR.

the IDLE state after the reset process. The MISS state of a DPR will not be triggered until a read operation is valid. In MISS state, the DPR responses to the reuse data under the same address range, but the valid stride is still unknown. After having finished another read access, when the stride is ready, the DPR enters PREFETCH state. Here, the DPR starts the prefetch access by itself, then switches to WAIT state. In WAIT state, there are three possible scenarios for the DPR, 1) when data is being reused, it stays in WAIT state; 2) when the current address hits the prefetch address, it enters PREFETCH state and starts another prefetch operation; 3) when prefetch address fails to match the real data address, it has to return IDLE state.

5.2 Data Transfer Strategy

Our proposed data transfer strategy is conceptually similar to cache. Cache is broadly used in the contemporary GPP (General Purpose Processor), however, its complex design procedure and the large spatial size make it unsuitable for CGRAs, which contains a great quantity of distributing RCs. Considering the features of intensiveness and regularity of data in applications, we figure out a more applicable prefetch and reuse structure for explicit CGRAs.

In our design, the data address for prefetch is calculated by a heuristic procedure, which dynamically changes in the run time and independent of the other RCs. Furthermore, we propose a small dual-bank cache-like data reuse structure, which greatly reduce the size of the traditional cache.

The key technologies to realize the heuristic data prefetch mechanism include the precise predication of the data address distance, i.e., the stride, and the management of data request handshake protocol by DPR. There are two methods to generate a valid stride. Either the DPR spends RC reading operation (rc_proc) in IDLE and MISS states respectively, or the RC hits the prefetch data in WAIT state as shown in Fig. 5 (b).

With the valid stride, DPR initiatively fetches the data (dpr_proc) and saves it in prefetch data register. Next, DPR broadcasts a self-triggered data transfer handshake protocol to inform the RC to preserve the data request signal while DPR is waiting for the end of data prefetch. Once data has been successfully prefetched, the DPR will response the access immediately. Finally, another new request is sent by DPR with a new predicting address. In such a manner, the RCA computing phase can be overlapped with the heuristic data prefetch phase.

This prefetch method especially favors the big stride of data access in media processing, e.g. the current data in position 0x10 and the next data in position 0x40 result in a stride of 0x30, which is much larger than the length of a cache line as 0x8. In addition, the prefetch method is suitable for stream applications, because it efficiently meets the distribution characteristics of stream data, thus reduces the miss rate of dual-bank caches (current data and prefetch data). Furthermore, the data prefetch stride is dynamically changeable at run time without confirming at the initial step.

The data reuse strategy, another method proposed by this paper, is similar to the look-like cache or scratchpad memory. For example, the DPR architecture illustration in Fig. 4 (b) that adopts both the current and prefetch data for data reuse works in the manner similar to a dual-bank cache. In our proposed method, although RCs read data with different sizes, a DPR always provides the enough bank width to guarantee the smooth data transfer. As mentioned above, the buffer size of each DPR is 8 times as big as that of RC data. In the ideal case, the DPR can provide up to 8 different reuse data for a RC requirement. In this way, if the next data is still in the current address range, e.g. the previous address offset 0x1 and the current address offset 0x3 both appear in the cache line range, the DPR responses the previous read data immediately.

In order to maximum the performance of a DPR, we integrate the prefetch and reuse methods synchronously. Considering in the prefetch mode, multi-context increases time slots of every two load operations, thus, the access latency is hidden nicely in the computation phase. Furthermore, the wide data width of a DPR supports more data intensive computation. By these means, the DPR adopts both data transfer strategies or either one of them to handle various application kernels. Thus, we even gain the compositive revenue of both methods.

With the proposed novel data prefetch and reuse strategy, our design successfully overlaps the computation and data access phases. The outstanding performance of the design is demonstrated by the following experiments.

6. Experiments

6.1 Experimental Setup

To evaluate our proposed HDPR strategy, we developed a CGRA prototype at cycle-accurate register transfer level (RTL) as illustrated in Fig. 3. The RCA consists of 16 RCs with an eight neighbor connection embedded in a 4×4 array. In the RCA, every four RCs in a column share one Distributed Register File (DRF) and thus totally 4 DRFs are needed. Under the corresponding parameters, the proposed CGRA is capable to emulate the other architectures, such as an ADRES with 4×4 tiles or a MorphoSys with 4×4 quadrants.

Six algorithms from DSPStone [22] and a kernel operation of image processing are used as the benchmark set mapped manually to the CGRA. *fir* is an 8 point finite impulse response. *convolution*, *complex_multiply*, *n_real_updates* and *n_complxe_updates* constitute the filter benchmark. *sobel* refers to a fundamental operator in image processing. *dot_product* performs a dot product of the form $Z = A[1 \times 2] * B[2 \times 1]$. They are all typical multimedia and telecommunication kernels with abundant inherent parallelism.

6.2 Comparison and Analysis

In the experiments, we demonstrate that our optimal method outperforms the traditional data transfer mechanisms adopted by both ADRES [9] and RSPA [10]. For ease of simulation and verification, the mapping result of instructions is executed using an internal ROM as the CRF. Meanwhile, data has been initialized by the RISC in the GDM.

The experimental results are summarized in Table 1. The column of *RunCycle* directly denotes the number of clocks needed by each algorithms under the traditional and the optimal solutions. With the HDPR strategy, the cycle number of run time dramatically decreases. The column of *Speedup* lists the improvement of performance. The proposed data transfer method achieves averagely 1.73 times speedup over the conventional methods on various benchmark algorithms. Less Initial Iteration represents a better pipeline efficiency, so we deduce the column *II* from the Eq. (1). In Table 1, the value of *II* shows the average iteration restart cycle has been decreased from 13.89 to 7.82 by our method. Furthermore, our raised *IPC* values have proved the HDPR strategy exhibits a powerful computational efficiency.

In Fig. 6 we demonstrate implementing the proposed HDPR strategy generates more performance profits than using the data prefetch or the data reuse method independently, as is mentioned in the section of motivation. However, exceptions happen to the *convolution*, *dot_product* and *n_real_updates* algorithms, where the HDPR solution gets the same performance speedup gain as the data reuse method. Each set of configuration contexts of these algorithms contains the load operation so that the access process can not overlap with the calculation. Consequently, the prefetch method becomes inapplicable to these cases, which illustrates a limitation of the current HDPR strategy.

From the experimental results, we conclude that our date prefetch and reuse strategy results in much less *RunCycle* and lower *II* than the traditional methods. It also improves the average *IPC* of RCA, ranging from 1.43 to 2.56.

Considering the possible influence on clock period and designed area, we compile the design into gate-level circuit. Therefore, we use the Design Compiler to estimate the area of CGRAs. We adopt the tsmc 65 nm 1P8M low power worst case library and the constraint of 200 MHz frequency. As demonstrated in Fig. 7, the DPR occupies about 5.86% of the area of CGRA prototype, while 80% of the area is occupied by GDM and CRF. In addition, the synthesis results show that the DPR is not the critical timing path in the design. So the proposed data transfer strategy does not cause performance degradation in terms of the critical path delay.

Table 1 Experimental results.

Algorithm	RunCycle		Speedup**	II		IPC	
	Conventional*	HDPR		Conventional*	HDPR	Conventional*	HDPR
fir(8taps)(N=64)	1031	476	2.17	16.27	7.46	1.74	3.76
Convolution(N=16)	103	70	1.47	6.6	4.4	1.09	1.6
n_complex_updates(N=16)	328	188	1.74	21.33	12.2	1.28	2.21
Sobel(N=64)	1029	474	2.17	16.25	7.44	1.87	4.05
dot_product(N=16)	199	130	1.53	12.93	8.33	0.96	1.48
complex_multiply(N=16)	165	116	1.42	10.73	7.47	1.65	2.34
n_real_updates(N=16)	119	75	1.59	13.11	7.43	1.42	2.51
Geometric Mean	none	none	1.73	13.89	7.82	1.43	2.56

*: The conventional data transfer of RC access local memory is similar with ADRES and RSPA.

**: The speedup is calculated as Convectional/HDPR.

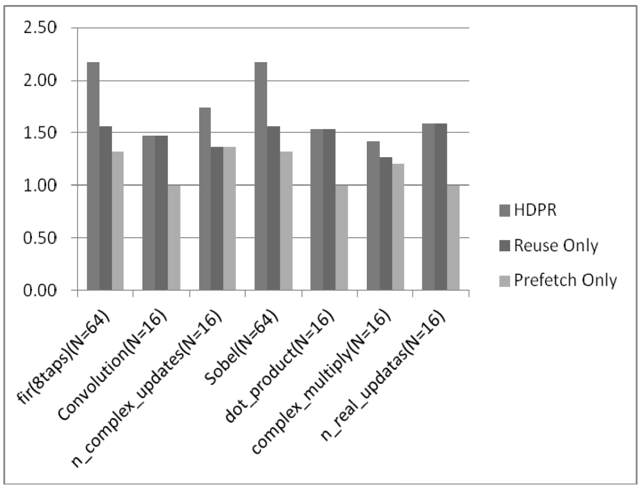


Fig. 6 Performance speedup of different data transfer strategies.

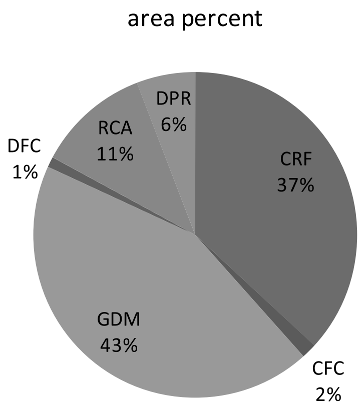


Fig. 7 Area of the prototype of CGRA.
*: The GDM include 38.4 K Bytes, each ram is 32 bits 600 depth, and the CRF is 12.288 K Bytes, each ram is 64 bits 96 depth.

7. Conclusion and Future Work

This paper proposed a novel strategy of data prefetch and reuse for CGRAs. To improve the data utilization efficiency, a dual-bank cache-like data reuse structure is proposed. Besides, a heuristic data prefetch is also introduced to further decrease the data access latency. Due to the HDPR strategy, our design effectively decreases the latency of data access and improves the performance of computation intensive applications. Experiments demonstrates it achieves remarkable low *II* and tremendously high *IPC*, which contribute to a significant speedup in execution time at the expense of only 5.86% increase in area.

Future work will focus on the improvement of external memory throughput and the optimization of data management. Moreover, we will track the key issues of the cost of exchanging data in external memory.

Acknowledgments

We express thanks to Peng Cao, Bo Liu and Jun He for their useful and incisive comments.

We acknowledge the contributions of JiXin Zhang, Ren Chen and Wen Wen, all of whom have been associated with the development of the CGRA architecture.

This work was supported by the National Natural Science Foundation of China (grant no.61204023 and grant no.61203251) and the National High Technology Research and Development Program of China (863 Program) (No. 2012AA012703).

References

[1] D. Novo, W. Moffat, V. Derudder, and B. Bougard, "Mapping a multiple antenna SDM-OFDM receiver on the ADRES coarse-grained reconfigurable processor," 2005 IEEE Workshop on Signal Processing Systems - Design and Implementation (SiPS), pp.473–478, 2005.

[2] K. Choi, "Coarse-grained reconfigurable array: Architecture and application mapping," IPSJ Trans. System LSI Design Methodology, vol.4, pp.31–46, 2011.

[3] W. Wenjie, L. Leibo, Y. Shouyi, Z. Min, W. Yansheng, and W. Shaojun, "H.264 parallel decoder at hd resolution on a coarse-grained reconfigurable multi-media system," 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp.2034–2036, 2010.

[4] M. Hartmann, V. Pantazis, T. Vander Aa, M. Berekovic, and C. Hochberger, "Still image processing on coarse-grained reconfigurable array architectures," J. Signal Processing Systems for Signal Image and Video Technology, vol.60, no.2, pp.225–237, 2010.

[5] C. Liang and X.M. Huang, "Mapping parallel fft algorithm onto smartcell coarse-grained reconfigurable architecture," 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp.231–234, 2009.

[6] M. Bingfeng, F.J. Veredas, and B. Masschelein, "Mapping an h.264/avc decoder onto the adres reconfigurable architecture," 2005 International Conference on Field Programmable Logic and Applications, pp.622–625, 2005.

[7] K. Yongjoo, L. Jongeun, A. Shrivastava, J.W. Yoon, C. Doosan, and P. Yunheung, "High throughput data mapping for coarse-grained reconfigurable architectures," IEEE Trans. Comput.-Aided Des. of Integr. Circuits Syst., vol.30, no.11, pp.1599–1609, 2011.

[8] Y. Kim, J. Lee, A. Shrivastava, J. Yoon, and Y. Paek, "Memory-aware application mapping on coarse-grained reconfigurable arrays," Proc. High Performance Embedded Architectures and Compilers, vol.5952, pp.171–185, 2010.

[9] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," Field-Programmable Logic and Applications, pp.61–70, Springer Berlin/Heidelberg, 2003.

[10] K. Yoonjin, M. Kiemb, C. Park, J. Jinyong, and C. Kiyoung, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," Proc. Design, Automation and Test in Europe, 2005, vol.11, pp.12–17, 2005.

[11] Y. Zhiyi, M.J. Meeuwsen, R.W. Apperson, O. Sattari, M. Lai, J.W. Webb, E.W. Work, D. Truong, T. Mohsenin, and B.M. Baas, "Asap: An asynchronous array of simple processors," IEEE J. Solid-State Circuits, vol.43, no.3, pp.695–705, 2008.

[12] S.R. Chalamalasetti, S. Purohit, M. Margala, and W. Vanderbauwhede, "Mora — An architecture and programming model for a resource efficient coarse grained reconfigurable processor," NASA/ESA Conference on Adaptive Hardware and Systems,

2009 (AHS 2009), pp.389–396, 2009.

[13] H. Singh, L. Ming-Hau, L. Guangming, F.J. Kurdahi, N. Bagherzadeh, and E.M. Chaves Filho, "Morphosys: An integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Trans. Comput., vol.49, no.5, pp.465–481, 2000.

[14] F. Campi, A. Deledda, M. Pizzotti, L. Ciccarelli, P. Rolandi, C. Mucci, A. Lodi, A. Vitkovski, and L. Vanzolini, "A dynamically adaptive dsp for heterogeneous reconfigurable platforms," Design, Automation & Test in Europe Conference & Exhibition, 2007 (DATE '07), pp.1–6, 2007.

[15] Z. Min, L. Leibo, Y. Shouyi, W. Yansheng, W. Wenjie, and W. Shaojun, "A reconfigurable multi-processor soc for media applications," Proc. 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp.2011–2014, 2010.

[16] S.P. Vanderwiel and D.J. Lilja, "Data prefetch mechanisms," ACM Comput. Surv., vol.32, no.2, pp.174–199, 2000.

[17] G. Dimitroulakos, M.D. Galanis, and C.E. Goutis, "Alleviating the data memory bandwidth bottleneck in coarse-grained reconfigurable arrays," 16th IEEE International Conference on Application-Specific Systems, Architecture Processors, 2005 (ASAP 2005), pp.161–168, 2005.

[18] P. Jongkyung, L. Jongeun, and C. Kiyoung, "Crm: Configurable range memory for fast reconfigurable computing," 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp.158–165, 2011.

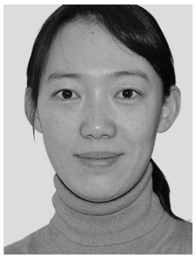
[19] X. Liu, C. Mei, P. Cao, M. Zhu, and L. Shi, "Date flow optimization of dynamically coarse grain reconfigurable architecture for multimedia applications," IEICE Trans. Inf. & Syst., vol.E95-D, no.2, pp.374–382, Feb. 2012.

[20] B.R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," Proc. 27th Annual International Symposium on Microarchitecture, pp.63–74, San Jose, California, United States, 1994.

[21] B. De Sutter, O. Allam, P. Raghavan, R. Vandebriel, H. Cappelle, T. Vander Aa, and B. Mei, "An efficient memory organization for high-ILP inner modem baseband SDR processors," J. Signal Process. Syst., vol.61, no.2, pp.157–179, 2010.

[22] V. Zivojnovic, J. Martinez-Velarde, C. Schlager, and M. Meyr, "Dspstone: A dsp-oriented benchmarking methodology," Proc. International Conference on Signal Processing and Technology (ICSPAT), vol.1, pp.715–720, 1994.

Wei Ge received the B.S. degree in Electronics Engineering from Southeast University in 2006. He is now a Ph.D candidate in Electrical Engineering Department of Southeast University. His research mainly focuses on the SoC design technology and reconfigurable computing and related VLSI design.



Zhi Qi received the B.Eng. degree from Southeast University, Nanjing, China, the M.Eng. degree from the National University of Singapore, Singapore, and the Ph.D. degree from the McGill University, Montreal, QC, Canada. She is currently an Assistant Professor with the Department of Electrical Engineering, Southeast University. Her research interests include computer vision, image processing, and pattern recognition, recently with a focus on reconfigurable computing, embedded computer vision, and network-based human–computer interaction.

Yue Du received the B.S. degree in Electronic Engineering from South China University of Technology in 2010. He is now studying for his master's degree in Electrical Engineering from Southeast University. His research mainly focuses on reconfigurable computing and related VLSI design.

Lu Ma received the B.S. degree in Information Engineering from Southeast University in 2010. She is now studying for her master's degree in Electronic Engineering from Southeast University. Her research mainly focuses on the area of compiling techniques for embedded systems.

Longxing Shi received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1984, 1987, and 1992, respectively, all in electronic engineering. He is currently a Professor and the Dean of Integrated Circuit (IC) College, Southeast University. His research interests include system-on-a-chip design, VLSI design, and power IC design.