PROCEEDINGS OF SCIENCE

Memory Partitioning Algorithm for Modulo Scheduling on Coarse-Grained Reconfigurable Architectures

Yuli Dai[1]
Mobile Computing Center of Institute of Microelectronics, Tsinghua University
Beijing, China 100084
E-mail: daiyuli1988@126.com

Coarse–Grained Reconfigurable Architectures (CGRAs) have become increasingly popular because of their flexibility, performance and power efficiency. CGRAs are often used to accelerate the computation–intensive applications with their high degree of parallelism; however, the performance of CGRA is limited by the bottleneck of memory bandwidth. Scratchpad memory with multiple banks architecture can provide adequate memory bandwidth. As a result, the compiler must assign data to different memory banks. In this paper, we make two contributions: i) we propose a modulo scheduling algorithm based on an automatic memory partitioning method; ii) we propose a data routing cost model to reduce the search space. Experimental results show that the overall execution time is reduced by 23% when compared with the state-of-the-art modulo scheduling method.

CENet2015
12-13 September 2015
Shanghai, China

[1]Speaker

PoS(CENet2015)072

1.Introduction

Modern embedded computing architectures demand high performance and power efficiency. The Application-specific integrated circuits (ASICs) are not programmed and therefore limited in usage. General-purpose graphic processing units (GPGPUs) are used to accelerate regular parallel loops. Field programmable gate arrays (FPGAs) have poor power efficiency due to their fine grained reconfigurability. To balance high performance, programmability and power efficiency, a large number of Coarse-Grained Reconfigurable Architectures (CGRAs) including Morphosys[1], ADRES[2] and Piperench[3] are proposed in the past decade.

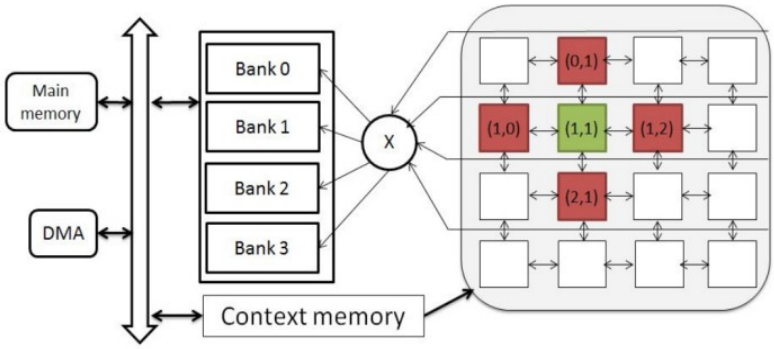

Figure 1: Brief Architecture of CGRA

Typically, CGRA consists of a 2-D processing element (PE) array. The PE array is connected with a mesh like reconfigurable network. Fig. 1 shows a brief architecture of CFRA. Each PE can execute word level operations (such as logic and arithmetic operations). The function and connection of PE are configured before each execution cycle. Each PE has a local register file to store the output data operations and the register file is accessible to the nearest four neighboring PEs in the next cycle.

In most real-time multimedia and DSP applications, the loops take up a large part of execution time, especially by using modulo scheduling method. The modulo scheduling is a well-known software pipelining method, which exploits instruction level parallelism (ILP) in loops by overlapping the execution of successive iterations. Multiple elements of an array are often requested to be simultaneous and will cause the hardware latency. There are usually two schemes to overcome the problem of multiple memory request. Using a single memory bank with multiple ports is a direct way; or use multiple banks architecture, but it poses a new issue of data distribution. One approach to solve the data distribution is to duplicate the data into several banks, which will result in the memory consistency problem and has power overhead. Another solution is to divide the original arrays into several parts. Each part of the arrays is not overlapped and allocated in different banks.

Previous memory partitioning method mainly partitioned the input array in a coarse grained level (the whole array as an atomic unit), but this paper proposes a more fine grained method to meet the potential parallelism.

To this end, this paper makes two contributions:

1.A maximum-clique based memory partition method is proposed to reduce the memory access conflicts.

2. A data routing cost model is proposed based on three different routing mediums.

3. An optimal modulo scheduling algorithm with memory partition is proposed on the basis of the memory partition and data routing cost.

The remaining of this paper is organized as follows: Section 2 presents the architecture of CGRA with multiple banks scratchpad memory and a motivated example of memory partition; Section 3 formulates the mapping problem; Section 4 provides the data routing cost model and modulo scheduling method; Section 5 analyzes the experimental results; and finally Section 6 concludes this paper.

2. Background and Problem Formulation

In Fig. 1, PE (1,1) can connect PE (0,1),(2,1),(1,0) and (1,2). The connection and function of PEs are configured the last execution cycle. PEs can be homogenous or heterogeneous. Hamzeh extended CGRA resource to a time-extended CGRA[4]. The load operation is expensive than others. Intuitively, optimizing the memory (load/store) operation will improve the system performance. In MorphoSys and ADRES, the on-chip SRAM space was divided into equal size memory module called banks, which can be accessed in parallel. In each time slot, PE array may have several accesses to the local memory. Latency must be added to avoid the bus conflicts for a single large bank memory architecture.

An early work [5] presented a multi-bank double buffering memory architecture, where PEs in a row can access the corresponding bank. Fig. 1 shows details of CGRA, which contains 4×4 PE array. Each PE reads data from the local memory or registers files of the neighboring PEs and writes output data to a dedicate register. The address generation unit (AGU) between PE array and four memory banks provide four ports. A hardware stall should be introduced if there are two data references to the same bank simultaneously. In the modulo scheduling method, initial interval (II) is a more important indicator than the critical path length when the number of iteration is relative larger. The II is defined in (2.1). We assume that the operation mode of PE array is synchronous and ignore the impact of epilogue and prologue. As to each stage, the execution is completed until all operations are over. t_{exe} is the maximum value of all operations and it is a constant. SC indicates the stage count number and T_i indicates the additional hardware stalls (caused by memory bank access conflict) between stage i+1 and stage i.

$$II = SC \cdot t_{exe} + \sum_{i=1}^{SC} T_i \tag{2.1}$$

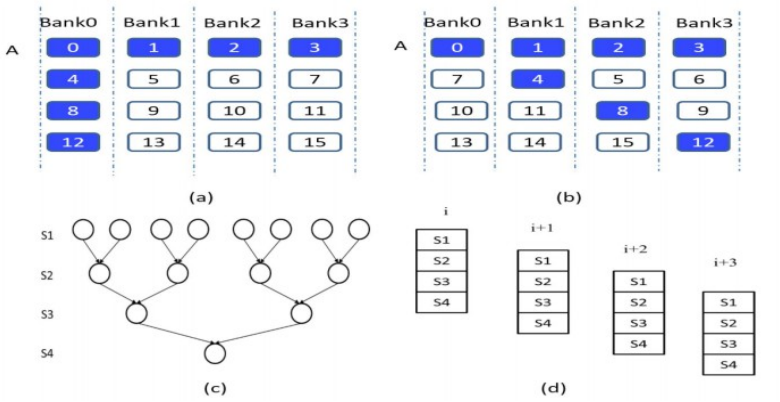

Figure2: (a) Cyclic allocation of array A, (b) another allocation of array A, (c) stage arrangement of example, (d) modulo scheduling of the example.

Equation (2.2) shows an example form DCT (Discrete Cosine Transform). A and B are two 4-by-4 symmetric matrixes. A^T is transpose of A. α and β are the scalar coefficients. The application is mapping onto a 4×4 PE array with 4 memory banks. The topmost stage includes 8 load operations as shown in Fig. 2(a). Elements in dark representing memory access in a single stage. In Fig. 2(a), A0, A4, A8, A12 are allocated in bank 0. and load stall are 2 cycles; therefore, II is equal to $2+T_1=8$ cycles. In Fig. 2(c), array A is partitioned by our method and the II is 4 cycles. This example shows that the distribution of input array can affect the execution time of applications on CGRA.

$$B = \alpha \cdot A \cdot A^T + \beta \cdot B \tag{2.2}$$

Most of previous module scheduling method ignored the impact of data distribution. Hatanaka[6] and Park[7] considered the operations placement and scheduling as the main task, and they ignored the impact of memory data distribution. As far as we've known, only Park[7] mentioned the memory data distribution; however they just clustered the input data in a coarse level (the whole array as the atomic unit) and the benefit of data partitioning. There are still a lot of space to dig the potential benefits from more fine-grained aspects, such as partitioning each element of array. Hamzeh[8] proposed a modulo scheduling method with register allocation based on maximum clique. Details of our partitioning algorithm will be described in Section 3 and 4.

3. Problem Formulation

In most DSP and multimedia applications, the nested loops occupy the majority of execution time. CGRAs are tending to accelerate the innermost loop. The CGRAs' compiler is to transform the loop kernels into the configuration instructions. The configuration instructions highlight the function of each PE and the routing path of data. The instructions can be reconfigured before the next cycle, if necessary. The memory architecture and execution model are based on the following claims.

1. The memory banks have a uniform latency for load/store operations.
2. The memory is big enough to allocate all data, and communication to the off-chip memory is not considered in this paper.
3. Each element of array has a fixed allocation during the execution.
4. The execution time of loops only depends on the initial interval (II) rather than the depth of loops.
5. An allocation with the smallest cost indicates the minimal execution time.

Due to the topological connections of PEs, this problem is much more complex on CGRAs than the regular VLIW processers. Modulo scheduling method increases the throughout of a nested loop. The target of modulo scheduling method is to find a feasible schedule pattern with minimal initial interval (II) [9].

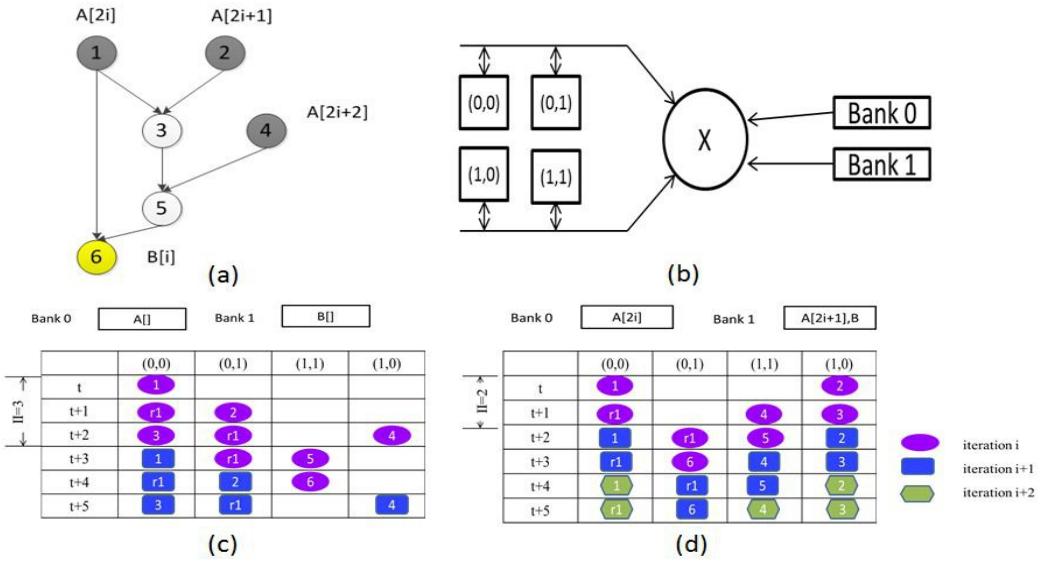

Figure 3: (a)a simple DFG, (b) a 2×2PE,(c) a feasible mapping of DFG with II=3, (d) another mapping with array partition with II=2. II is 2 because a[2i] and a[2i+1] is loaded to PE (0,0) and (1,0) simultaneously after array partitioning.

Definition1. Let $D_I=(V_I, E_I)$ be the input DFG, and ***CR*** be the description of time extended CGRA resources. The role of compiler is to find a mapping function $M=(M_v, M_e)$ .The mapping function $M=(M_v, M_e)$ includes two aspects. One is the node mapping function $M_v(V_i)=(V_c, t_i)$. It indicates that node V_i in DFG D_I is mapping to PE V_c at time step t_i . The other aspect is $M_e(e=(V_i, V_j))=Path(M_v(V_i), M_v(V_j))$. It indicates that edge in DFG is mapping to . For example, in figure3(c), M_v (1) = ((0,0),t), M_e ((1,6)) = (((0,0), t), ((0,0), t +1), ((0,1) , t+ 2) ,((0,1), t+3),((1,1), t+ 4)).

Memory partitioning is to separate the original array to different non-overlapped parts to avoid memory access conflict. Multiple access requests a single bank simultaneously, which has caused conflict. Potential conflict exists in all the operations are performed simultaneously. Based on the conflict relationship, we propose a data conflict graph (DCG). Different II indicates the different execution sequences and it will cause the change of DCG.

The partition of an original array can be described as a function f(d), the number of bank to which d is allocated. The bank conflict between two memory accesses d1 and d2 means that f(d1) = f(d2) simultaneously. This paper assumes that each bank has only one port.

In CGRAs, each PE usually has a register file, which is connected to nearest neighbor PEs; therefore there are three ways to routing data, routing PE, register file and spilling through local memory. Routing PE is a PE which does not execute any operation but only transfers data. As to the register files, the routing cost is lowest but the resource is limited and Routing PE occupies the computation resource. Transfer data with local memory is expensive for CGRAs, but it saves computation resource. We conduct a routing cost model with the above routing mediums.

Definition2. Given an edge f $e=(u,v)$ orm DFG, a mapping function $M=(M_v, M_e)$. Assume that $M_v(u)=(PE_i, t_i)$, $M_v(v)=(PE_n, t_n)$,edge mapping function $M_e(u,v)=Path(PE_i, PE_n)=(PE_i, t_i)$, $(PE_{i+1}, t_{i+1})...(PE_n, t_n)$ and $t_{i+1}-t_i=1\,cycle$.

The routing cost of register and memory are defined as follows:

$$C_{path}=\sum_{i=1}^{n-1} Pathcost(PE_i, PE_{i+1}) \tag{3.1}$$

$$Pathcost=\begin{pmatrix} C_{reg}, if\ dis(PE_i, PE_{i+1}=0) \\ C_{PE}, if\ dis(PE_i, PE_{i+1}=1) \\ C_{reg}, if\ PE_{i+1}\ is\ destnation \\ \infty, otherwise \end{pmatrix} \tag{3.2}$$

Memory partitioning gives an DFG ***G=(V,E)***, CGRA hardware resource description ***C*** , memory port number ***N*** only to find an optimal scheduling pattern and memory partitioning function so that:

Minimize II, subject to

$\forall R_i, R_j \in G, if\ T(R_i)=T(R_j), \wedge R_i \neq R_j, then\ MP(R_i)\neq MP(R_j)$

$\forall R_i \in G, \sum |MP(R_i)| \leqslant N$

$\forall R_i, R_j \in G,$ if Ri and Rj are in same node in DFG and they are not in the same iteration, then $MP(R_i)=MP(R_j)$

.

The first condition indicates that if the memory reference R_i and R_j are be mapped at the same time step, they should not be partitioned to the same bank. While the second condition emphasizes the modulo constrain that R_i and R_j should be partitioned to the same bank. The last condition is the resource constrain which indicates that the memory access number is less than the memory port N.

4. Modulo Scheduling alogrithm with Memory Partitioning

In this section, we propose a modulo scheduling algorithm with heuristic memory partitioning method. For example, in Fig. 3(a), the conflict relation is between op 1 and op 2 and the references are (2i+2) and(2i). The partition function is $MP(R_i)$ % 2 . The iteration distance is (2(i+2)-2· i)=4.

Algorithm 1 Modulo Scheduling Method with Memory Partitioning

Input: DFG ***D=(V,E)*** ,CGRA=***C***, Memory Port ***N***
1: D_q=*DFS*(*D*), Assign(D_q), D_c Construct_conflict_graph(D_q)
2: *MII*DeterminII(D_q, D_c ,*C*,*N*)
3: *II*
4: S
5: **while** TRUE **do**
6: Placement_and_routing(,S,II)
7: **if** no mapping is found **then**
8: *II* ,**break**
9: **else** S update_schedule(,S,II,)
10: **end if**
11: **end while**
12: **return** S & II

The MSDP algorithm is presented in Algorithm 1. Line 1 is the pre-process step, at first assigning the stage of each operation and traversing DFG to record all memory operations and create the data conflict graph (DCG) D_c. According to DCG, the conflict number N can be determined. Line 2 determines the II which respects the input DFG and the CGRA description. Line 5-11 shows the iterative algorithm of data distribution pattern and II. If no mapping is found, the **while** loop is **break** and II is increased. When the scheduling is found, the data allocation can find the minimal bank conflict number allocation pattern of each array.

5. Experiment Results

This paper had taken some kernel of DSP and multimedia applications for experiments. To evaluate the effectiveness of our approach, we selected edge-centric modulo scheduling (EMS) and EPImap as the scheduling methods. Details of benchmarks are shown in Table 5.1.

Application[1]	#ops	#mems	#edges	length	MII
FFT	36	16	35	9	4
MPEG2	56	21	67	8	6
JPEG	51	17	52	16	5
H.264	44	18	48	7	5
Lowpass	24	10	23	9	3
Sobel	27	7	34	4	2
Jacobi	25	13	26	10	4

[1]The benchmarks are selected from SPEC2006. This table shows the details of DFG (data flow graph) of the loops in selected applications. #op denotes the number of operations in DFG, #mems denotes the number of load/store operations, #edges denotes the number of edges (data dependence) and length is the length of critical path, MII is the theoretically minimal II for a 4×4 PE array with 4 memory banks.

Table 1 : Details of Benchmarks

The target CGRA architecture is shown in Section 2. The local memory has 4 banks with each bank having one port. A bank cannot connect more than one PE simultaneously. The size of each local bank is 1K Byte. Each PE has a local register files consisting of four registers, which can be accessed by its neighboring PEs. The execution time of configuration of each step can be hidden. All verifications were taken on the operating system of Window7 with 4GB memory and 1.9G Hz Intel Dual-Core CPU.

Fig. 4 shows average II by using MSDP with and without memory routing. If without the memory routing, it indicates CGRA route data only uses the register files and the routing PE. We observe that JPEG and matrix multiplication achieve MII using MSDP with memory routing. The performance improvements can explained by the ratio of LE_s/E_s. In the benchmark JPEG, the LE_s/E_s ratio is 21/52. Routing these 21 long edges only by the register files, the routing PEs will cost more computation PE than by memory routing; therefore, the achieved II are 7 and 5, respectively. The result verifies the correctness of the proposed routing cost model proposed in Section 3.

Figure 4: Performance of MSDP to a 4×4 CGRA with memory routing vs. without memory routing, each PE has 4 registers.

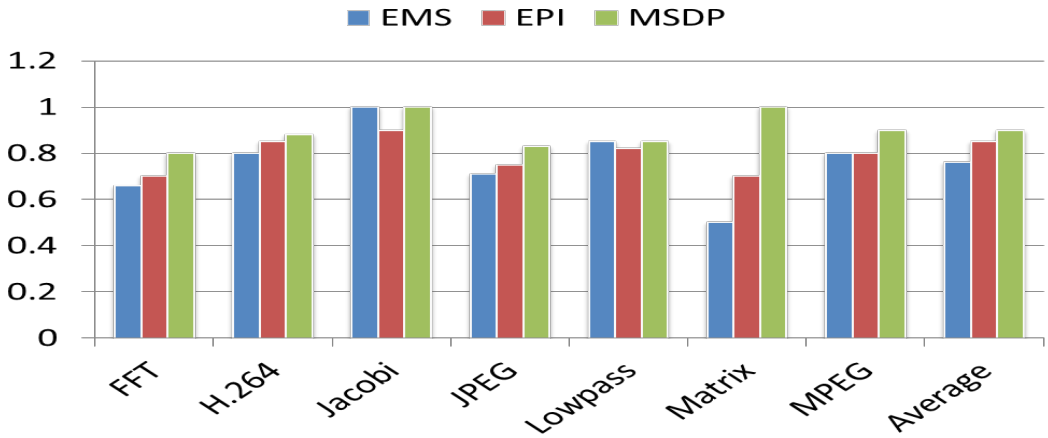

Figure 5: Performance of MSDP vs. EPImap and EMS.

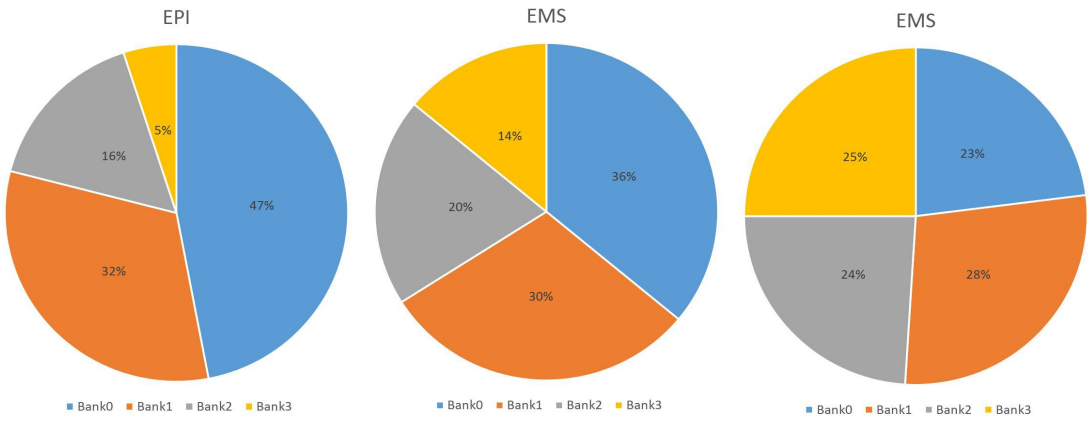

Figure 6: Bank Distribution of Application H. 264.

Fig. 5 shows the II of three modulo scheduling methods, which are Modulo scheduling with Data partitioning (MSDP) and other two modulo scheduling techniques (EPImap and EMS). The first observation on the average, MSDP achieves a lower practical II than both EMS and EPImap in most applications. In best case of Matrix multiplication, the MII/II ratio of MSDP is 2X of EMS. On average, MSDP with MII/II ratio of 1.23X and 1.15X better than EMS and EPImap, respectively.

The bank distribution is shown in Fig. 6 that the distribution of our MSDP method is more balanced than EMS. If the size of original array is larger than local memory, MSDP has fewer access of off-chip memory, which is expensive.

6. Conclusion

This paper introduced the relations among memory partitioning, modulo scheduling and register allocation on coarse-grained reconfigurable architectures. By partitioning data onto different banks for reducing the pressure of local memory data bus, the methodology has succeeded in reducing the memory access and improve the performance of CGRA; and the experimental results show that the proposed memory partitioning method has reduced the execution time by 23% when compared with state-of-the-art modulo scheduling method.

References

[1] H. Singh, M.-H. Lee, G. Lu,, F. J. Kurdahi,, N. Bagherzadeh,, , M. C. Eliseu Filho. *Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications*[J], Computers, IEEE Transactions on, vol. 49, no. 5, pp, 465–481(2000)

[2] B. Mei, S. Vernalde, D. Verkest,, H. De Man,, R. Lauwereins,. *Adres:An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix* [M], in Field Programmable Logic and Application.Springer Berlin Heidelberg. pp, 61–70(2003)

[3] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor. *Piperench: A reconfigurable architecture and compiler,Computer*[J], vol. 33, no. 4, pp, 70–77(2000)

[4] M. Hamzeh, A. Shrivastava, and S. Vrudhula, *Epimap: using epimorphism to map applications on cgras*[C], Proceedings of the 49th Annual Design Automation Conference. ACM DAC2012. San Francisco. pp,1284–1291(2012)

[5] Y. Kim, J. Lee, A. Shrivastava, J. W. Yoon, D. Cho, and Y. Paek, *High throughput data mapping for coarse-grained reconfigurable architectures* [J], Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 30(11):1599–1609(2011)

[6] A. Hatanaka and N. Bagherzadeh, *A modulo scheduling algorithm for a coarse-grain reconfigurable array template*[C], Parallel and Distributed Processing Symposium IPDPS 2007. IEEE International. IEEE, Long Beach CA.pp, 1–8(2007)

[7] H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H.-s. Kim. *Edgecentric modulo scheduling for coarse-grained reconfigurable architectures*[C], International conference on Parallel architectures and compilation techniques . ACM ,Toronto pp, 166–176(2008)

[8] M. Hamzeh, A. Shrivastava, S. Vrudhula. *REGIMap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)*[C], Proceedings of the 50th Annual Design Automation Conference. DAC2013, ACM, Austin. pp, 1-10(2013)

[9] C.-G. Lyuh and T. Kim.*Memory access scheduling and binding considering energy minimization in multi-bank memory systems*[C], Proceedings of the 41st annual Design Automation Conference DAC2004, ACM. San Diego pp, 81–86(2004)