Joint Affine Transformation and Loop Pipelining for Mapping Nested Loop on CGRAs

Shouyi Yin*, Dajiang Liu*, Leibo Liu*, Shaojun Wei*, Yike Guo†
*Institute of Microelectronics, Tsinghua University, Beijing, 100084, China
†Department of Computing, Imperial College, London, SW7 2AZ, UK

***Abstract*—Coarse-Grained Reconfigurable Architectures (CGRAs) are the promising architectures with high performance, high power- efficiency and attractions of flexibility. The computation-intensive portions of application, i.e. loops, are often implemented on CGRAs for acceleration. The loop pipelining techniques are usually used to exploit the parallelism of loops. However, for nested loops, the existing loop pipelining methods often result in poor hardware utilization and low execution performance. To tackle this problem, this paper makes two contributions: 1) a pipelining-beneficial affine transformation method which can optimize the initiation interval (II) of nested loop and enable multiple loop pipelines merging; 2) a multi-pipeline merging method which can improve hardware utilization further. The experimental results show that our approach can improve the performance of nested loop by up to 56% on average, as compared to the state-of-the-art techniques.**

***Keywords*-reconfigurable computing, CGRA, loop pipelining, affine transformation, polyhedral model**

I. Introduction

Coarse-Grained Reconfigurable Architecture (CGRA) [1] [2] is a promising parallel computing architecture with both high performance and high flexibility, which is typically composed of a host controller, a Processing Element Array (PEA). The PEA is typically a 2-D mesh PE array connected by flexible routes, where PEs run in a parallel way which enables it to efficiently execute various applications. The functionality of PE could be configured to different word-level operations of fixed-point numbers according to the configuration words. The routing style of PEs has great variety, such as mesh topology in which each PE could communicate with its four nearest neighbors, mesh plus topology and morphosys topology [1]. The data memory is used to store input and output data for PEA.

Since a program usually spends 90% of its execution time in only 10% of the code (loop), the most important task of CGRA compiler is automatically and efficiently mapping loops onto PEA. The loop pipelining techniques are widely used to exploit loop-level parallelism and modulo scheduling is one of most important techniques in loop pipelining. For nested loops, the existing loop pipelining methods for CGRA often result in insufficient parallelism and poor hardware utilization. This is because that the existing methods [3] [4] are only good at pipelining single level loop, e.g., the innermost loop in nested loop. Although they can find the best II for the innermost loops, if the parallelism of innermost loop is insufficient compared to CGRA resource, they will lead to poor PE utilization rate. Moreover, the existing loop pipelining methods has little consideration of loop carried dependences in a nested loop, which may generate significant memory access overhead. When mapping nested loop on CGRA, in order to reduce memory access overhead, it is necessary to assign more data dependences to be transferred by routing PEs and distributed registers. However, in the existing loop pipelining methods, only the data dependences carried by innermost loop can be assigned to routing PEs and distributed registers, which results in high memory access overhead of transferring outer loop carried dependences.

Based on the analysis of these drawbacks of existing loop pipelining methods, we find that the key of better nested loop pipelining on CGRA is adapting the data dependence pattern of nested loop to the CGRA's architectural feature. This paper makes the following three contributions: (1) For the first time in CGRA compilers, we propose the use of affine transformation to facilitate nested loop pipelining. The affine transformation is tailored to exploit the parallelism in inner loops and reduce the outer loop carried dependence. (2) By analyzing the hardware features of a CGRA, we establish an analytic performance model that takes the both PE utilization rate and memory access cost of CGRA into an overall consideration based on polyhedral model [5]. Taking this performance model, we formulate the mapping of loop nests on CGRA as a constrained optimization problem. (3) Using the insights from problem formulation, we design a joint affine transformation and multi-pipeline merging approach.

II. Motivation

In this section, we motivate our method with an example of a two level perfect loop nest. The iteration domain and data flow graph (DFG) of the original loop are shown in Figure 2(a) and 2(b), where i and j indicate the outer and inner iteration variables, and the red arrows and blue arrows indicates the loop carried dependence with distance vector of [0,1] and [1,0], corresponding to the colored arrows in Figure 2(b). We also note that, in Figure 2(b), the inner loop carried dependence

This work was supported in part by the National NSFC grant (No.61274131), the International S&T Cooperation Project of China Grant (No. 2012DFA11170), the Tsinghua Indigenous Research Project (No.20111080997), the China National High Technologies Research Program (No. 2012AA012701 and 2012AA010904) and the China Major S&T Project (No.2013ZX01033001-001-003).

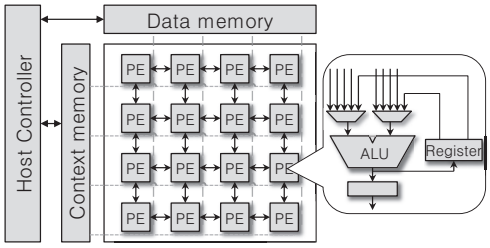


Fig. 1. Typical architecture of CGRA

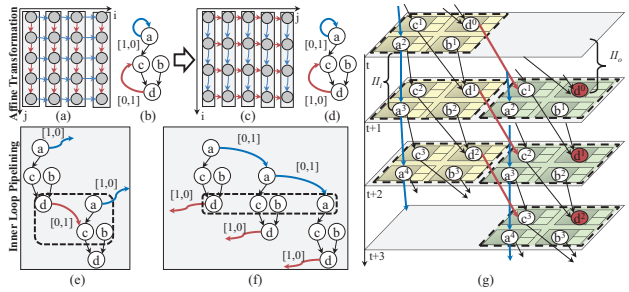


Fig. 2. Pipelining performance improved by affine transformation and multi-pipeline merge

[0,1] causes a longer delay of the recurrence than the outer loop carried dependence [1,0].

To improve the PE utilization rate, we perform affine transformation (e.g., loop interchange in this example) on the original loop nest and we get the iteration domain and DFG of the new loop nest, as shown in Figure 2(c) and 2(d). After this affine transformation, the delay of the recurrence at innermost level, as the blue arrows in Figure 2(c) and 2(d), is shorten to one cycles. When pipelining the inner loop of the new loop nest, II of 1 could be obtained and 4 operators of the kernel (dashed rectangular) could be executed in a parallel way (Fig. 2(f)), which is lower than the II=2 obtained in the original approach Fig. 2(e). As a result, the PE utilization rate could be improved to 50% if mapping is performed on a 2×4 PEA. Moreover, pipeline merging could further improve the PE utilization rate. As shown in Fig.2(g), we merge two pipelines together and map the merged pipelines as a whole onto a 2×4 PEA. After merging, two kernels, including 8 operators, could be executed in a parallel way on the 2×4 PEA and this merging improves the PE utilization rate to 100%.

Since there are quite a lot loop carried dependences in a nested loop and memory accessing overhead is relative high, getting less loop carried dependences to be delivered by data memory is very effective to improve the execution performance of a loop nest on CGRA. As shown in Fig. 2(g), some outer loop carried dependences (e.g., red arrows) could be delivered by routing PEs with less communication overheads after pipeline's merging.

This example illustrates the joint affine transformation and multi-pipeline merging approach is effective to improve PE utilization rate and reduce memory access overhead.

III. Problem Formulation

In this section, we take both affine transformation and pipeline merging into joint consideration and formulate the performance of nested loop pipelining on CGRA.

A. Affine Transformations and Multi-pipeline Merging

Taking a N-level perfectly nested loop (L) with regular iteration domain as example (as shown in Fig. 3), we formulate the pipelining-beneficial affine transformations and multi-pipeline merging based on the polyhedral model [5]. The imperfectly nested loop and triangle domain nested loop could also be handle with similar approach. Let Φ be the transformation matrix of L, which acts on the N iteration variables, $i_1, i_2, \cdots, i_N$ from the outermost level to the innermost level, and B_n be the iteration count of the nth level loop in this L. We use three symbols, E_{in1}, E_{in2} and E_{out} to denote different kinds of dependence. E_{in1} denotes the dependences only carried by innermost loop. E_{in2} denotes the dependences carried by innermost two loops but not by outer $N-2$ level loops. E_{out} denotes the dependences carried by outer $N-2$ level loops. Obviously, $E_{in1} \in E_{in2}$ and $E_{out} = E \backslash E_{in2}$.

As all the dependences should be preserved, the following lexicographic ordering should be held:

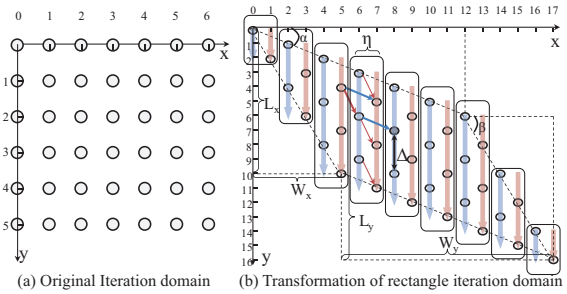
$$\Phi(\vec{i}_s^e) \prec \Phi(\vec{i}_t^e), \forall e \in E \tag{1}$$

From the perspective of nested loop, multi-pipeline merging is equivalent to mapping the innermost 2-level loop simultaneously onto CGRA. Therefore the pipelining-beneficial affine transformation should consist of two parts: 1) Loop interchange is performed to select 2 loops as the innermost 2 levels for parallelization, 2) Loop skewing is performed to enhance parallelism at the innermost 2-level and enable merging multiple innermost loop pipelines. As a result, the legal affine transformation matrix on N-level nested loop should have the following form:

$$\Phi = \begin{bmatrix} S_1 \\ I_1 \\ \cdots \\ I_{N-2} \\ S_2 \end{bmatrix} = \begin{bmatrix} 0 \cdots \Theta \\ I_1 \\ \cdots \\ I_{N-2} \\ 0 \cdots \Pi \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & c_1 & c_2 \\ 0 & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 & 0 \\ 0 & \cdots & 0 & d_1 & d_2 \end{bmatrix} \tag{2}$$

where each row of matrix Φ representing a 1-D affine transformation. The rows can be classified into two types: 1) Interchange type (I) that has one and only one "1" in the row; 2) Skewing type (S) has non-zero coefficients in the last two elements and the first N-2 elements are all 0. There are N-2 rows of I type and 2 rows of S type. All these 1-d transformations are arranged together to form the scheduling matrix Φ of this N-dimension loop. S_1 and S_2 could also be represented as $(0,\cdots,\Theta)$ and $(0,\cdots,\Pi)$ where $\Theta = (c_1, c_2)$ and $\Pi = (d_1, d_2)$ are the hyperplanes of innermost two loops, respectively. To guarantee that inner loop pipelining and multi-pipeline merging can be performed, the affine transformation on innermost 2 loop should satisfy the following constraints:

$$\Theta(\vec{i}_t^e) - \Theta(\vec{i}_s^e) \geq 0, \quad \Pi(\vec{i}_t^e) - \Pi(\vec{i}_s^e) \geq 0, e \in E_{in2} \tag{3}$$



(a) Original Iteration domain (b) Transformation of rectangle iteration domain

Fig. 3. Pipelining-beneficial affine transformation and multi-pipeline merging

After affine transformation, we would try to merge η (merging factor, MF for short) pipelines and map them onto CGRA. Consequently, the MF η, initiation interval of the inner loop (II_i), initiation interval of outer loop (II_o) would be obtained in this process.

B. Performance Modeling and Problem Formulation

After the skew of the innermost loop with hyperplane Θ and Π, the iteration domain of innermost 2 levels is a parallelogram, as shown in Fig.3. We use the small circle to indicate iteration instance. The innermost loops are pipelined and η pipelines are merged to form a strip. The strips are executed sequentially. For example, there are 2 pipelines, the red one and blue one, in each strip.

1) Control Steps and NLET: To formulate the Nested Loop Execution Time (NLET), we first calculate the number of control steps to execute one strip. In order to obtain the number of control steps of one strip, we need to calculate the number of iterations in each pipeline of the strip. We first calculate the start point and end point of each pipeline along y-axis in the case of parallelogram iteration domain in Fig.3(b):

$$S(x) = \begin{cases} x\tan\alpha, & x < W_y \\ L_y + (x - W_y)\tan\beta, & W_y \le x < W_x + W_y \end{cases} \quad (4)$$

$$E(x) = \begin{cases} x\tan\beta, & x < W_x \\ L_x + (x - W_x)\tan\alpha, & W_x \le x < W_x + W_y \end{cases} \quad (5)$$

where $\tan\alpha$ is equal to $|d_1/c_1|$, $\tan\beta$ is equal to $|d_2/c_2|$, W_x is equal to $c_1(B_N - 1)$, L_x is equal to $d_1(B_N - 1)$, W_y is equal to $c_2(B_{N-1} - 1)$, and L_y is equal to $d_2(B_{N-1} - 1)$.

With these two values, we could calculate the distance between start point and end point of a pipeline. However, not every integer point on the line has a iteration instance. The iteration instance appears every Δ integer points along the line, as shown in Fig.3(b) , where Δ is equal to $|c_1d_2 - c_2d_1|$. As a result, the number of iteration on each line could be represented as follows:

$$N(x) = \left\lceil \frac{E(x) - S(x)}{\Delta} \right\rceil \quad (6)$$

As each innermost loop would be pipelined, the number of control steps of each innermost loop pipeline could be represented with II_i and loop latency L_b as follows:

$$P(x) = (N(x) - 1)II_i + L_b \quad (7)$$

We use II_o to denote the initiation interval between pipelines in one strip. Then the number of control steps of executing sth strip could be represented with II_o and the number of control steps of the last pipeline in the strip as follows:

$$N_{CS}(s) = (\eta - 1)II_o + P(s\eta + \eta - 1) \quad (8)$$

And the total number of strips (N_S) is: $\lceil \frac{W_x + W_y}{\eta} \rceil$.

Therefore, the NLET could be represented as follows:

$$NLET = \prod_{n=1}^{N-2} B_n \cdot \sum_{s=1}^{N_S} T_{LD}(s) + T_{EX} \cdot \prod_{n=1}^{N-2} B_n \cdot \sum_{s=1}^{N_S} N_{CS}(s) \quad (9)$$

where $T_{LD}(cs)$ denotes the latency of data load in control step cs and T_{EX} denotes the latency of computation of PE. In Equation (9), we let

$$T_{LD}(s) = \sum_{cs=1}^{N_{CS}(s)} T_{LD}(cs) \quad (10)$$

The physical meaning of $T_{LD}(s)$ is the total communication time for executing strip s.

2) Communication Time: The communication time in strip s is directly proportional to the number of data dependences that delivered by data memory. As shown in Fig.2(f), the dependences in E_{in1} (as the blue arrows) can be delivered by routing PEs and distributed registers and has no memory access communication costs. The dependences in E_{out} can only be delivered by data memory and will cause communication costs. The dependences in $E_{in2\setminus 1}$ can be divided into two types: 1) the dependence within one strip, as the fine red arrows in Fig.3(b), which can also be delivered by routing PEs and distributed registers, 2) the dependence across the border of strip, as the fine blue arrows in Fig.3(b), which needs to be delivered by data memory and has memory access communication cost. We use a 0-1 variable to indicate whether a dependence in $E_{in2\setminus 1}$ is across the strip's border or not as follows:

$$\rho(m, e) = \begin{cases} 1, & m + \delta(e) > \eta \\ 0, & otherwise \end{cases} \quad (11)$$

where $\delta(e) = \Pi(\vec{i}^t) - \Pi(\vec{i}^s)$ indicates the number of hyperplanes Π dependence $< \vec{i}^t, \vec{i}^s >$ traverses, and m indicates the mth pipeline in a strip.

Therefore, the communication time in strip s, $T_{LD}(s)$, could be represented as follows:

$$T_{LD}(s) = \frac{1}{BW} \sum_{m=0}^{\eta-1} \left(N(s\eta + m) \cdot \left(|E_{out}| + \sum_{e \in E_{in2\setminus 1}} \rho(m, e) \right) \right) \quad (12)$$

where BW denotes the data memory bandwidth and $|E_{out}|$ denotes the number of dependences in E_{out}.

3) Optimization Problem Establishment: Substituting Equations (8) and (12) into Equation (9), we can get the final expression of NLET. Then we could establish the optimization problem for nested loop pipelining as follows:

Given a N-dimensional nested loop L with dependence set E and a target CGRA C, find a tuple (Φ, η, II_i, II_o) **such that**:

(i) $\forall e \in E, \Phi(\vec{i}_s^e) \prec \Phi(\vec{i}_t^e)$;
(ii) $\forall e \in E_{in2}, \Theta(\vec{i}_t^e) - \Theta(\vec{i}_s^e) \geq 0$;
(iii) $\forall e \in E_{in2}, \Pi(\vec{i}_t^e) - \Pi(\vec{i}_s^e) \geq 0$;
(iv) the merged η pipelines with II_i and II_o is successfully mapped on CGRA C;
(v) $NLET$ is minimized.

IV. Efficient Solution

We first generate the legal transformation matrix according to the constraints and then find the feasible P&R of merged pipeline on CGRA. By using genetic algorithm, we finally achieve the optimal solution (Φ, η, II_i, II_o) and the corresponding mapping scheme of the given nested loop.

A. Solution Space Generation

In our approach, the pipelining-beneficial affine transformation for nested loop has unique property compared to general affine transformation. The loop interchange is performed at all dimensions and loop skewing is only performed at innermost two dimensions of the interchanged nested loop. Therefore the transformation matrix Φ must have the same pattern as shown in Equation (2). To generate the solution space, we first generate the general solution space for all affine transformations. Then we use the constraint of matrix pattern to reduce the general space and finally we can get the solution space for our optimization problem. The following algorithm (**Algorithm 1**) generates solution space of Φ for a nested loop:

Algorithm 1 Solution Space Generation

```
1: L_d ← The search space built by Pouchet's heuristic [5], d ∈ (1, N)
2: U ← The matrixes that have the same pattern of Equation (2)
3: for each matrix U_n in U do
4:     for each dimension d do
5:         U_nd ← The d_th row of U_n
6:         S_nd ← L_d ⋂ U_nd
7:         if S_nd = ∅ then
8:             S_n ← ∅, break
9:         end if
10:    end for
11:    if S_n ≠ ∅ then
12:        E^n_in2 ← ∅
13:        for each e ∈ E do
14:            for each p ∈ I-type row set P of S_n do
15:                if p · e = 0 then
16:                    E^n_in2 ← E^n_in2 ⋃ e
17:                else
18:                    E^n_in2 ← E^n_in2 \ e, break
19:                end if
20:            end for
21:        end for
22:        for each e ∈ E^n_in2 do
23:            Compute T-type row set R satisfying R · e ≥ 0
24:            if R ⋂ Θ = ∅ then
25:                S_n = ∅, break
26:            end if
27:            if R ⋂ Π = ∅ then
28:                S_n = ∅, break
29:            end if
30:        end for
31:    end if
32: end for
33: S ← ⋃ S_n
```

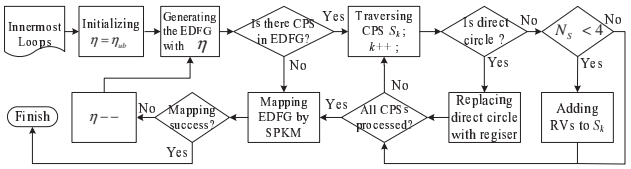


Fig. 4. The processing flow of E-SPKM

B. Multi-Pipeline Merging

After the nested loop is transformed with matrix Φ, we should merge innermost loop pipelines as many as possible to get the best parallelism and reduce memory access overhead further. The upper bound of η, η_{ub}, can be obtained as follows:

$$\eta_{ub} = \left\lfloor \frac{sizeof(\text{PEA})}{sizeof(\text{Loop Body})} \right\rfloor \tag{13}$$

To merge loop pipelines together, we design an extended-SPKM method (E-SPKM) based on the Split-Push Kernel Mapping (SPKM) method [6], which is best existing mapping method to obtain minimum area mapping for kernels.

The whole flow of E-SPKM is shown in Figure 4. First we initialize $\eta = \eta_{ub}$. Then we duplicate η DFGs to form an extended-DFG (EDFG). Then we check whether there is a cyclic path subgraph (CPS) in the EDFG. If there is a direct circle (the producer and consumer are the same operator), we use a register to replace it. If there is an indirect circle, we check the number of vertices (N_{S_k}) and insert routing vertexes (RVs) to form a matching-cut. The number of inserted RVs is $4 - N_{S_k}$, according to the theory of matching-cut [6]. After the processing for all CPSs, the EDFG is capable to be processed by SPKM method. If SPKM fails to map EDFG, we reduce η and repeat this procedure until it succeeds. When E-SPKM is finished, the innermost loop initiation interval (II_i) can be obtained by calculating the mapping path length of the innermost loop carried dependence. And the initiation interval between two consecutive pipelines (II_o) can be obtained by calculating the mapping path length of the inner 2-level loop carried dependence.

C. The Nested Loop Pipelining Flow

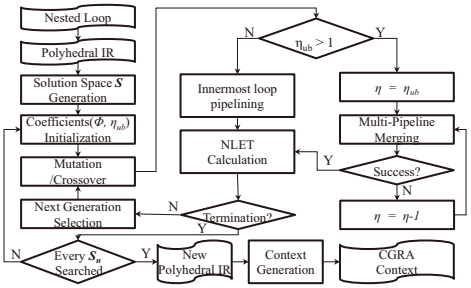


Fig. 5. The Nested Loop Pipelining Flow

In Fig. 5, the whole flow of our approach is demonstrated. The implementation of our approach is integrated into the

framework of LLVM-Polly. We take loop kernels in high-level specifications like C/C++ as input. Next, we analyze the polyhedral model intermediate represent (IR) and perform the proposed approach on the polyhedral model IR. Finally, we get the optimal mapping scheme of nested loops and generate CGRA context in the back-end phase.

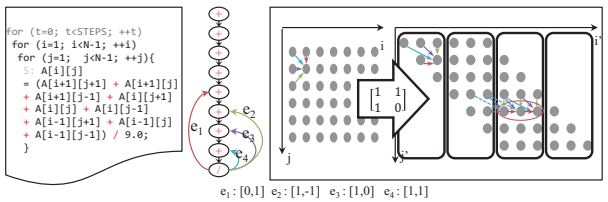
V. Experimental Results

A. Experiment Setup

To verify and evaluate the proposed optimization approach, we report performance results for several important nested loops taken from PolyBench 3.2 [7]. We select three state-of-the-art loop mapping approaches on CGRA as references, innermost loop pipelining (ILP) [3], epilog-prolog merging (EPM) [4] and outer loop pipelining (OLP) [8]. The performance results are obtained from a cycle-accurate CGRA simulator. The target CGRA has a 4 $\times$ 4 homogeneous PEA, where every PE in the PEA could perform a word level (32 bits) fixed-point logic operations. The latency of PE operation and memory access operation (Load/Store) are 1 and 2 clock cycles, respectively.

B. Case Study

The performance of nested loop running on CGRA highly depends on the PE utilization rate and memory access overhead. We use seidel-2D to illustrate the joint effect of pipelining-beneficial affine transformation and multi-pipelining merging, which improves the performance from both aspects.



(a)Seidel-2D loop nest (b) DDG (c) Transformation enables multi-pipeline merging

Fig. 6. Seidel-2D example

In Fig.6(a), where the innermost 2 loops, i and j are selected to be mapped onto CGRA. The dependence data graph (DDG) is shown in Fig.6(b) and there are four dependences, $e_1 : [0, 1]$, $e_2 : [1, -1]$, $e_3 : [1, 0]$ and $e_4 : [1, 1]$. If using ILP or EPM method, the innermost loop with dependence e_1 would be pipelined and the minimal II_i of 5 could be obtained. If our approach is adopted, it will first perform affine transformation on the original iteration domain, as shown in Fig.6(c). After this transformation, the dependence pattern is changed. The dependence e_2 becomes the innermost loop carried dependence. This new dependence pattern can result in $II_i = 4$ and obtain utilization rate of 65.2%. For further analysis, the performance comparison of seidel-2D is given in Table I. The notation CPT and CMT denotes "Computation Time" and "Communication Time". Since no pipeline merging is involved in both ILP and EPM, we first apply our approach with MF of 1 for fair comparison. When MF=1, our approach achieves CPT=8.6×10^5 and NLET=5.4×10^5, which are both less than those of ILP and EPM. This improvement is attributed to the smaller II_i, resulting in higher PE utilization rate.

In addition, the memory access overhead is also reduced greatly. Let's take one row of iteration in a strip for example, as in the red oval in Fig.6(c), where all the input dependences of each iteration are demonstrated with arrows. After pipeline's merging with MF=3, all the dependence within strip are promoted to intra-PEA communications (the solid arrows), and can be delivered by routing PE or distributed registers. As a result, the communication overhead is further reduced to 2.6×10^5 cycles, which are much less than the number of 5.7×10^5 cycles when MF=1.

TABLE I
Performance Comparison of Seidel-2D (Unit of CPT, CMT, NLET: clock cycle)

Approaches	MF	II_i	II_o	UR	CPT	CMT	NLET
ILP	1	5	N/A	17.5%	1.0×10^6	5.8×10^5	1.6×10^6
EPM	1	5	N/A	19.2%	9.3×10^5	5.6×10^5	1.5×10^6
Ours	1	4	N/A	21.4%	8.6×10^5	5.7×10^5	1.4×10^6
Ours	3	4	8	65.2%	2.9×10^5	2.6×10^5	5.4×10^5

C. Performance Comparison

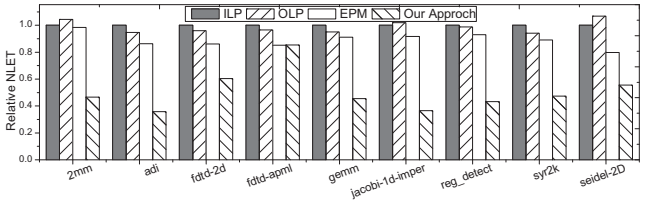


Fig. 7. Relative NLET Comparison

1) NLET Comparison: As shown in Fig. 7, the relative NLET of benchmarks mapped by the four approaches are compared. As ILP only handle the innermost loop and all other outer loops are executed in sequential mode, it gets the worst execution performance among the four approaches. In OLP approach, the epilogue corresponding to one outer loop iteration is overlapped with the prologue corresponding to one or more subsequent outer loop iterations. In EPM approach, the epilogue and prologue of two nearby outer loop are merged together. Thus, the initiation time of outer loop is put forward and some memory communications between epilogue and prologue are delivered by PEA-internal routing PEs and distributed registers. In our approach, we try to improve parallelism and reduce memory communications from more fundamental aspects, including dependence pattern transformation and pipelines merging. Therefore the transformed dependence pattern and merged loop kernel can lead to more improvement. In the experiments, we do obtain

better performance, about 56% improvement on average when compared to the ILP, and about 53% improvement on average when compared to the OLP and about 51% improvement on average when compared to the EPM.

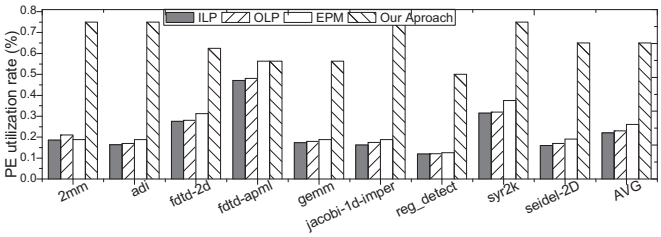


Fig. 8. PE Utilization Rate Comparison

2) PE Utilization Rate Comparison: Fig.8 demonstrated the PE utilization rate achieved by the four approaches. As ILP only exploits the parallelism from innermost loop which is very restricted by the dependences pattern of the innermost loop, the number of concurrently executable operators from innermost is limited and it get poor PE utilization rate, about 23% on average. In both OLP and EPM, although the PE utilization rate in the epilogue and prologue phase is improved, the maximal number of concurrently executable operators of steady-state loop kernel is not changed. Thus, OLP and EPM obtain average utilization rates of 23% and 27%, which are both just a little bit better than ILP. In our approach, since a pipelining-beneficial dependence pattern is achieved, the II of innermost pipeline is reduced, which improves the PE utilization rate. Moreover, multi-pipeline merging also plays an important role in increasing utilization rate. Therefore, we get a utilization rate of 66% on average, which is much higher than that of both ILP and EPM.

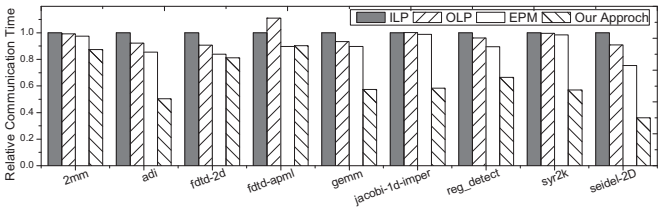


Fig. 9. Relative Communication Time Comparison

3) Memory Access Overhead Comparison: As shown in Fig. 9, the relative memory communication time obtained by the four approaches is demonstrated. As only the innermost loop is parallelized in ILP, all the outer loop carried dependence can only be delivered by data memory. In both OLP and EPM, although the data transfer between nearby epilogue and prologue can be delivered by PEA internal distributed register and routing PEs with low cost, there are still a large number of dependence would be stored and loaded in memory. As a result, the communication cost of OLP and EPM are still very high, just 11% and 9% less on average than ILP, respectively. In our approach, both affine transformation and multi-pipeline merging can promote data dependences to be delivered by PEA internal routing PEs and distributed registers. As a result, our approach could have a communication cost reduction of 40% on average, as compared to ILP.

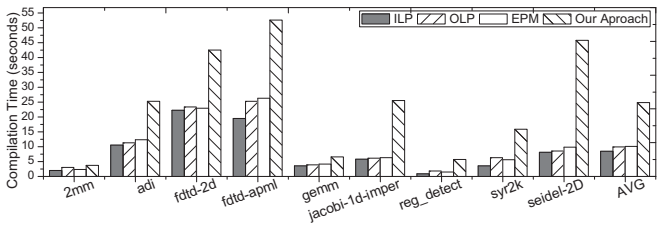


Fig. 10. Average Compilation Time Comparison

4) No Significant Compilation Time Overhead: We also evaluate the compilation time overhead of our approach. Fig.10 shows the average mapping time of all the four approaches running on an Intel Dual-Core CPU machine at 1.9GHz with 2GB memory. Since our approach makes comprehensive optimization, it results in a longer compilation time. The average compilation time of our approach is 2.9×, 2.6× and 2.5× more than that of ILP, OLP and EPM, respectively.

VI. Conclusion

Loop nests are the major computation-intensive portions in application. Nested loop pipelining is crucial for the execution performance of CGRA. We propose a joint affine transformation and multi-pipeline merging approach, which can improve the parallelism of pipelines and reduce the number of dependences need to be delivered by data memory. As a result, higher PE utilization rate and smaller memory access overhead can be achieved, resulting in better execution performance of nested loop on CGRA finally.

References

[1] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture enhancements for the adres coarse-grained reconfigurable array," in *High Performance Embedded Architectures and Compilers*. Springer, 2008, pp. 66–81.

[2] L. Liu, C. Deng, D. Wang, M. Zhu, S. Yin, P. Cao, and S. Wei, "An energy-efficient coarse-grained dynamically reconfigurable fabric for multiple-standard video decoding applications," in *Custom Integrated Circuits Conference (CICC), 2013 IEEE*. IEEE, 2013, pp. 1–4.

[3] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "Epimap: using epimorphism to map applications on cgras," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 1284–1291.

[4] Y. Kim, J. Lee, T. X. Mai, and Y. Paek, "Improving performance of nested loops on reconfigurable array processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 32, 2012.

[5] L.-N. Pouchet, C. Bastoul, A. Cohen, and J. Cavazos, "Iterative optimization in the polyhedral model: Part ii, multidimensional time," in *ACM SIGPLAN Notices*, vol. 43, no. 6. ACM, 2008, pp. 90–100.

[6] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, "A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architectures," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 11, pp. 1565–1578, 2009.

[7] L. Pouchet, "Polybench: The polyhedral benchmark suite (2011)," *URL http://www-roc. inria. fr/~ pouchet/software/polybench*.

[8] G. D. Kalyan Muthukumar, "Software pipelining of nested loops," *Compiler Construction Lecture Notes in Computer Science*, vol. 2027, pp. 165–181, 2001.