TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/15 pp330–343
DOI: 10.26599/TST.2022.9010001
Volume 28, Number 2, April 2023

Loop Subgraph-Level Greedy Mapping Algorithm for Grid Coarse-Grained Reconfigurable Array

Naijin Chen*, Fei Cheng*, Chenghao Han, Jianhui Jiang, and Xiaoqing Wen

Abstract: To solve the problem of grid coarse-grained reconfigurable array task mapping under multiple constraints, we propose a Loop Subgraph-Level Greedy Mapping (LSLGM) algorithm using parallelism and processing element fragmentation. Under the constraint of a reconfigurable array, the LSLGM algorithm schedules node from a ready queue to the current reconfigurable cell array block. After mapping a node, its successor's indegree value will be dynamically updated. If its successor's indegree is zero, it will be directly scheduled to the ready queue; otherwise, the predecessor must be dynamically checked. If the predecessor cannot be mapped, it will be scheduled to a blocking queue. To dynamically adjust the ready node scheduling order, the scheduling function is constructed by exploiting factors, such as node number, node level, and node dependency. Compared with the loop subgraph-level mapping algorithm, experimental results show that the total cycles of the LSLGM algorithm decreases by an average of 33.0% ($\mathrm{PEA}_{4\times4}$) and 33.9% ($\mathrm{PEA}_{7\times7}$). Compared with the epimorphism map algorithm, the total cycles of the LSLGM algorithm decrease by an average of 38.1% ($\mathrm{PEA}_{4\times4}$) and 39.0% ($\mathrm{PEA}_{7\times7}$). The feasibility of LSLGM is verified.

Key words: Grid Coarse-Grained Reconfigurable Array (GCGRA); mapping; loop subgraph; scheduling

1 Introduction

Grid coarse-grained reconfigurable computing systems exhibit the characteristics of low power consumption, dynamic online configuration, etc. They have been applied to the Internet of things[1], acceleration[2], and other fields. Field Programmable Gate Array (FPGA) is a typical representative of fine-grained reconfigurable architecture, which demonstrates obvious advantages in the bit-level calculation. However, FPGA has a long configuration time and power consumption while dealing with arithmetic and logic operations with bit width. Under this background, various Coarse-Grained Reconfigurable Architecture (CGRA) models have been proposed[3, 4].

Thus, CGRA has become an international research hotspot. Many related research results are found. A compiler mapping simulator is a paramount aspect of the CGRA research[5, 6].

The Processing Element Array (PEA) of Grid Coarse-Grained Reconfigurable Architecture (GCGRA) exhibits the characteristics of simple interconnection, less wiring, low power consumption, etc. PEA can finish basic arithmetic and logic operations, such as monocular, binocular, and trinocular, and it can accomplish complex arithmetic and logic operations. Because the loop of computing-intensive tasks has numerous running times, it can be transformed into intermediate representations, such as a Data Flow Graph (DFG). According to the

• Naijin Chen, Fei Cheng, and Chenghao Han are with School of Computer and Information Science, Anhui Polytechnic University, Wuhu 241000, China. E-mail: chennaijin@ict.ac.cn; {957189105, 1048551181}@qq.com.
• Jianhui Jiang is with School of Software Engineering, Tongji University, Shanghai 201804, China. E-mail: jhjiang@tongji.edu.cn.
• Xiaoqing Wen is with Department of Computer Science and Networks, Kyushu Institute of Technology, Fukuoka 820-8502, Japan. E-mail: wen@cse.kyutech.ac.jp.
* To whom correspondence should be addressed.
Manuscript received: 2021-10-29; revised: 2021-12-27; accepted: 2022-01-24

interconnection, area, and other constraints of PEA, the nodes of DFG are mapped onto one or several PEAs using an algorithm. As such, the execution efficiency of critical loop tasks is significantly improved. The DFG mapping problem is the key issue to be solved by the CGRA compilation simulator, which has been proven to be an NP-hard problem[3–6].

As for different computing platforms, various mapping scheduling algorithms have been proposed[7–15]. Owing to the characteristics of GCGRA flexible configuration in Processing Element (PE), it attracted considerable attention. Several scheduling algorithms have been proposed based on GCGRA platforms[10–13]. However, the existing algorithms exhibit the following defects.

(1) Because the grid PEA mapping algorithm places too much emphasis on reducing cycle start intervals, the number of PEs is not fully utilized.

(2) The communication cost between grid PEA blocks is high. In this study, we consider resolving these defects. Its contributions are as follows:

(a) The operation mechanism and formal definition associated with temporal mapping are given, and Verilog Hardware Description Language (HDL) is used to simulate the delays of 4-bit signed addition, subtraction, and multiplication.

(b) A Loop Subgraph-Level Greedy Mapping (LSLGM) algorithm for grid PEA is designed and implemented. The LSLGM algorithm considers the interconnection constraints of grid PEA, parallelism of operation nodes, usage of hardware PE fragments, etc.

The remainder of this article is organized as follows. In Section 2, related works are introduced. In Section 3, we introduce the cluster architecture of CGRA and sequential logic simulation of three operations. In Section 4, we define the problems associated with temporal mapping. In Section 5, we introduce experimental motivation and develop Loop Subgraph-Level Mapping (LSLM) and LSLGM algorithms. In Section 6, we demonstrate the efficiency of our approach on several indexes and analyze our experimental results. Finally, we conclude this study in Section 7.

2 Related Work

CGRA is a multicore system. The programmable register bits in CGRA are wide, and the power consumption is low. At present, CGRA has become a research hotspot. Many related research results exist, which are described from two aspects below.

Typical studies related to non-GCGRA structures are described as follows:

Liu et al.[5] described the current research status of existing CGRA from the perspectives of programming, computing, and execution. In addition, the parallel computing, virtualization, and storage efficiency of CGRA are analyzed. For the mapping problem of parallelizable CGRA, a depth greedy mapping algorithm has been designed to minimize the communication costs between row configuration blocks[6]. Aiming at solving the mapping scheduling problem of row parallel reconfigurable array in processing multitree DFG, a Row-Column Pruning Mapping (RCPM) algorithm was proposed[7]. Compared with the place and route algorithm, the average execution total delays of RCPM reduced by 15.7% ($RCA_{4\times4}$) and 18.4% ($RCA_{5\times5}$). Compared with the split-push kernel mapping algorithm, the average execution total delays of RCPM decreased by 30.0% ($RCA_{4\times4}$) and 29.8% ($RCA_{5\times5}$). Chen and Feng[8] evaluated the interconnection delay of PEA with routing, row-column bus, and point-to-point relationships. The experimental results showed that the PEA with point-to-point interconnection obtained less interconnection delay. From the perspective of cyclic acceleration, Balasubramanian and Shrivastava[9] proposed a compute-intensive loop acceleration by randomized Iterative Modulo Scheduling (IMS), optimized mapping method to remove invalid mappings, and achieved improved effective scheduling results.

Typical studies related to GCGRA structures are as follows.

Lee et al.[10] introduced the transient fault recovery time structural model of CGRA and proposed a three-mode redundancy fault tolerance model for error detection and correction of CGRA. They analyzed the maximum recovery time of faults. A multiobjective optimization Genetic Mapping (GenMap) was designed. Compared with the traditional mapping algorithm, the energy consumption of GenMap reduced by 12.1%–46.8%[11]. The partition mapping module of the CGRA compiler was restricted by repeated computing resources, universal formal description, etc. Hamzeh et al.[12] proposed the EPImorphism Mapping (EPIMap) algorithm. EPIMap exhibits a less compilation time than edge-centric modulo scheduling. The discussion focused on two main defects of the existing CGRA.

One was the issue of effectively using PE register files, and the other was large compilation time. A heuristic mapping algorithm has been proposed considering PE internal registers[13]. This mapping algorithm optimizes the compiler performance.

3 CGRA PEA Cluster (PEAC) and Delay Solution of Partial Operations

3.1 CGRA PEAC

A general PEAC architecture based on CGRA is shown in detail (see Fig. 1). This architecture includes the Main Processor (MP), Main Memory (MM), high bus, Direct Memory Access (DMA), and Reconfigurable Processing Unit (RPU). To alleviate the speed contradiction among MP, MM, and RPU and to optimize the performance of storage and computation for computer systems, the data or instruction cache module can be added among MP, MM, and RPU. The RPU comprises several configuration controllers, reconfigurable PEAs, interconnected resources, etc. PEA architectures can be homogeneous or heterogeneous. Each PEA includes several homogeneous or heterogeneous PEs, data or instruction memory, and other components. They are shown in detail in Fig. 2.

3.2 Target mapping structure

To facilitate research, we only consider the mapping of a single grid PEA. The internal structure of grid PEA and PE is shown in Fig. 2. The grid PEA demonstrates the following two properties. First, after the calculation task is mapped successfully, it can be executed on PE. One PE can handle fixed-point operations, and two PEs can handle floating-point mantissa and exponent. In this

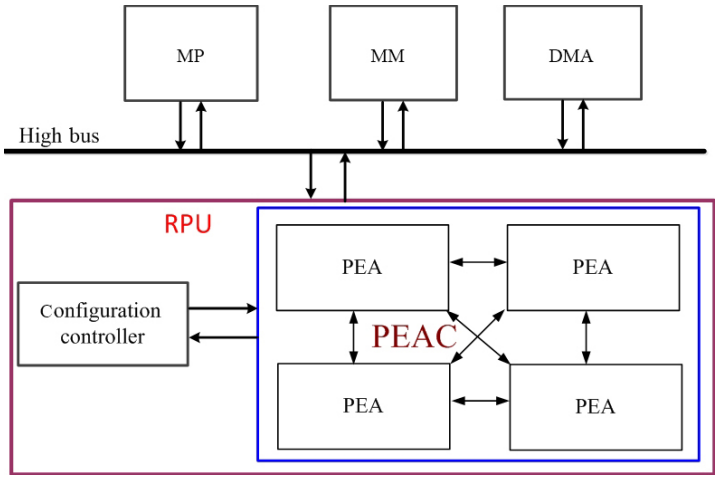


Fig. 1 PEAC architecture of CGRA.

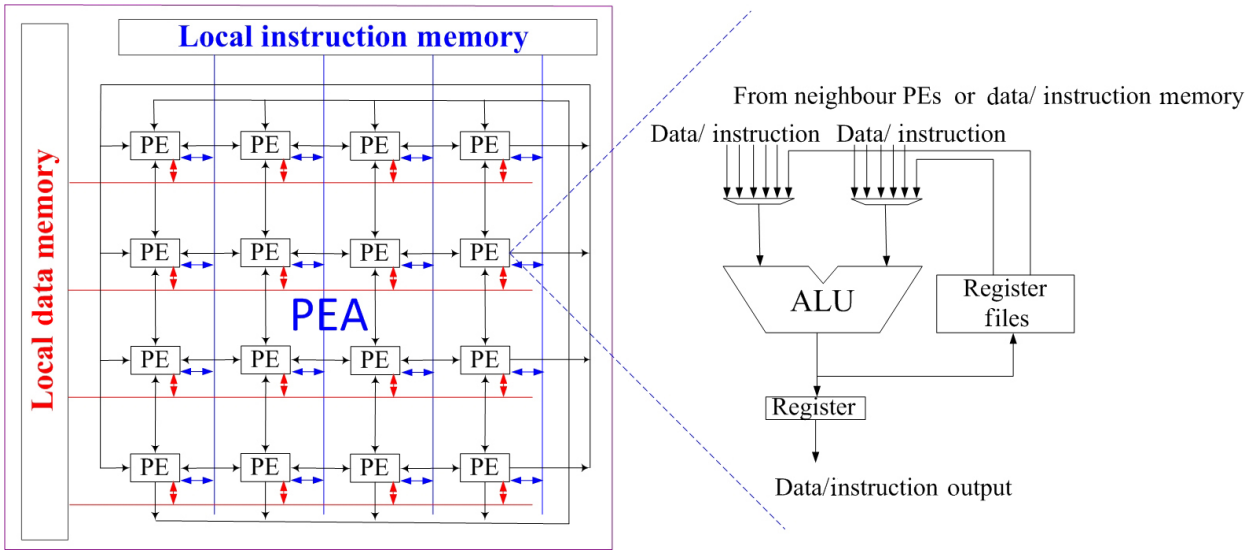


Fig. 2 Internal structure of grid PEA and PE.

study, we consider only isomorphic PE and fixed-point operations. Second, the interconnection mode between PEs is point-to-point. It can form pipeline or nonpipeline operations inside and outside PEA blocks.

Our research is based on four preconditions.

(1) Fixed-point operations of operands are studied, and a loop DFG of a program is extracted.

(2) Temporal mapping methods of operation level will be studied, and the temporal mapping algorithm is evaluated by one PEA.

(3) As some computing-intensive tasks exhibit dependencies between one cycle and others, a loop single subgraph is mapped in proper order.

(4) Storage and fetch data are collected in a synchronous state.

3.3 Delay solution of partial operations

In general, the participating operands have 4 bits or more, which is called coarse granularity. We study the operation based on a coarse-grained array, so it is uniformly agreed that the operands have 4 bits. Verilog HDL is employed to solve the operation delays of source code signed addition, subtraction, and multiplication. Experiments show that the delays of 4-bit signed addition and subtraction are 2 cycles, and the delay of multiplication is 6 cycles. The conventional configuration and other arithmetic logic operation delays are unified into one cycle, and the number of PE occupied by each operation is 1 cycle.

Consider the Signed-4-bit Source Code Multiplication (SSCM) operation delay solution as an example, the delay of SSCM is obtained by 4 times addition, 4 times shift, and symbol bit processing of operation results. Figure 3 shows that the delay is 6 cycles.

In Algorithm 1, Step 1 initializes the partial product and assigns the absolute values of the multiplicand and multiplier. Step 2 shows that the end of the multiplier is 0, the partial product adds 0, and then the partial product and multiplier are moved right by 1 bit. If the end of the multiplier is 1, the partial product adds X, and then the partial product and multiplier are moved right by 1 bit. Step 3 indicates symbol bit calculation. Step 4 is the end of SSCM.

4 Problem Definition

The key to GCGRA acceleration is minimizing the execution delay of each PEA and its reuse time. Therefore, under the area constraint of a PEA, to obtain a higher speedup, the critical loop of computing-intensive exhibits a smaller computing delay (i.e., S_{SD}). If a PEA has more reusable time of reconfigurable hardware (i.e., M), the hardware configuration cost will increase, which will affect the acceleration of GCGRA. The optimization objectives of GCGRA include the following two points:

(1) Pursuing the minimization of S_{SD}. (2) Minimization of M.

The relevant definitions of the mapping problem are as follows.

Definition 1 (Calculation pattern): Let PEA be a

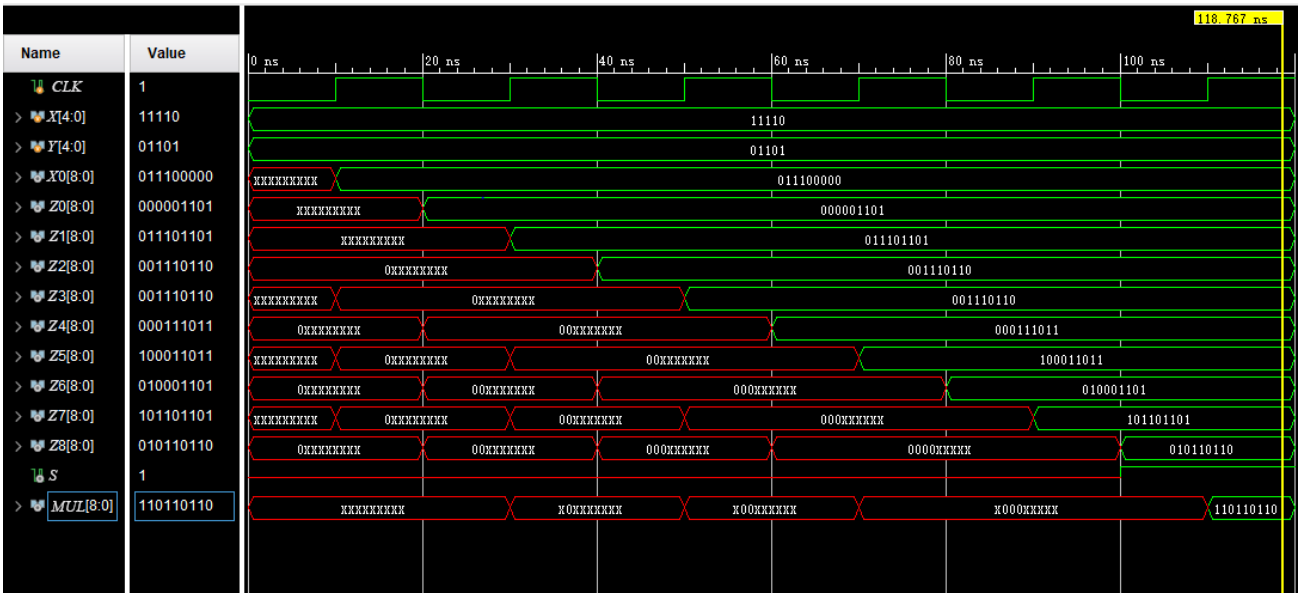


Fig. 3 Sequential waveforms of 4-bit signed multiplication.

Algorithm 1 SSCM

Input: X[4:0], Y[4:0]
Output: Delay of SSCM
Step 1:
$MUL[8:0]=000000000$; $X[8:0]=000000000$; $Y[3:0]=Y[3:0]$; $X[7:4]=X[3:0]$; $MUL[3:0]=Y\ [3:0]$;
Step 2:
for (i = 1 to 4) **do**
  if ($MUL[0]==1$) **then**
    $MUL[8:0]=MUL[8:0]+X[8:0]$;
  end if
  $MUL \gg 1$; $MUL[8]=0$;
end for
Step 3:
if ($X[4]\oplus Y[4]$) **then**
  $MUL[8]=1$;
else
  $MUL[8]=0$;
end if
Step 4: end SSCM.

set of isomorphic or heterogeneous PE units, the function f: PE→PE is called a monadic operator, the function f: PE×PE→PE is called a binary operator, the function f: PE×PE×PE→PE is called a triple operator, and the function f: PE×PE×⋯×PE→PE is called a multivariate operator.

Definition 2 (Computing delay S_{SD}): A loop DFG can be represented as a quadruples $G = (V, E, W, D)$[14], set of vertices $V = \{v_i | v_i$ is an ordered operator, $1 \leqslant i \leqslant n\}$; set of edges $E = \{e_{ij} | e_{ij} = \langle v_i, v_j \rangle, 1 \leqslant i, j \leqslant n\}$; set of weight $W = \{w_i |$ the area of hardware resources occupied by $v_i, 1 \leqslant i \leqslant n\}$; D represents a set of delays, $d_i \in D$ represents the delay of the v_i operation node. A two-dimensional ($row \times col$) PEA is a quadruple, PEA= (PE, I, O, E_L)[15], where PE = $\{PE_{(1,1)}, PE_{(1,2)}, \ldots, PE_{(m,n)}\}$ is a finite set. Temporal mapping of $G = (V, E, W, D) \xrightarrow{f}$PEA = (PE, I, O, E_L) is monomorphism. Let $V = \{v_1, v_2, \ldots, v_n\}$, $P = \{P_1, P_2, \ldots, P_M\}$ is the mapping set of V, $\bigcup_{i=1}^{M} P_i = V$ and $P_i \cap P_j = \phi (i \neq j)$.

The calculation definition of S_{SD} is as follows:

(1) Suppose there are two nonintersecting directed cyclic subDFGs, let $V_1 = \{v_1, v_2, \ldots, v_m\}$, $V_2 = \{v_m, v_{m+1}, \ldots, v_k\}$, $V_1 \subseteq V$, $V_2 \subseteq V$, $V' = V_1 \cup V_2$, $V' \subseteq V$, V_1 and V_2 have mapped onto a P_i simultaneously, $i \in [1, M]$. In the set V_1, if several directed paths exist from v_1 to v_m, let the longest path of [v_1,v_m] is R_1, it has a maximum delay , which can be expressed as $delay(R_1)$. In the set V_2, if several directed paths exist from v_m to v_k, let R_2 represent the longest paths exist from v_m to v_k with the maximum delay, which can be expressed as $delay(R_2)$, then $S_{SD} = max(delay(R_1), delay(R_2))$.

(2) Let directed cyclic subDFGs with two intersections, $V_1 = \{v_1, v_2, \ldots, v_m\}$ and $V_2 = \{v_m, v_{m+1}, \ldots, v_k\}$, $V_1 \subseteq V$, $V_2 \subseteq V$, $V' = V_1 \cup V_2$, $V' \subseteq V$, V_1 and V_2 are mapped to a certain P_j concurrently, $j \in [1, M]$, in set V' , if several directed paths exist from vertex v_1 to vertex v_k, let R_{max} represent the longest paths from v_1 to v_k with the maximum delay, which can be expressed as $delay(R_{max})$, then $S_{SD} = delay(R_{max})$.

Definition 3 (Mesh PEA area): The interconnection relationship of the execution PEA is point-to-point, and each row or column's PEs meet certain requirements, then the mesh PEA area can be expressed as $A_{RPU} = row \times col$.

Definition 4 (Hardware fragment PE): Under the constraints of coarse-grained reconfigurable array area, interconnection mode, and other elements, it is generated in the process of dynamically dividing the loop DFG by a mapping algorithm. It demonstrates the characteristics of configuration, computation, and data transmission, and its area is the number of remaining execution PEs.

Definition 5 (T_{TOTAL} of loop DFG): As the grid PEA is point-to-point interconnection, the interconnection delay inner PEA block is approximately 0. The total cycles of a DFG consumed by one PEA execution can be expressed as $T_{TOTAL} = T_{con} + T_{in} + T_{out} + T_c$, where T_{con} represents the configuration time. Configuration time includes PE operator or routing configuration time (about 1 cycle), configuration time for one PEA, and control word configuration time for dynamic connection switching (about 17 cycles); T_{in} represents the input time for loop DFG; T_{out} represents the output time for loop DFG; T_c represents the calculation time for loop DFG. In summary: $T_{TOTAL} = T_{con} + T_{in} + T_{out} + T_c = (\alpha \times C_{CON} + \eta \times M) + \beta \times (N_1 + N_{org1}) + \gamma \times (N_2 + N_{org2}) + \delta \times S_{SD}, \alpha, \eta, \beta, \gamma, \delta$ are correction coefficients, and their values are determined by different CGRAs and interconnection modes, the unified agreement is $\alpha = 1, \eta = 17, \beta = \gamma = 1$, and $\delta = 1$. The meanings of $C_{CON}, N_1, N_2, N_{org1}, N_{org2}, M, S_{SD}$, and other symbols are consistent with those in Ref. [6].

Regulation 1 For a computationally intensive task, the original input times (i.e., N_{org1}) and original output times (i.e., N_{org2}) of DFG are fixed values. For comparison, we uniformly stipulate that the data communication costs between one PEA and others are

calculated according to the number of nonoriginal inputs (i.e., N_1) and nonoriginal outputs (i.e., N_2) .

Regulation 2 Considering the dependency of loop subgraphs, we partition multiple loop subgraphs in a sequential mapping manner, loop subgraphs are mapped onto PEAs sequentially.

5 Experimental Motivation and LSLGM Algorithm Design

5.1 Experimental motivation

Our experimental motivation includes the following two points:

(1) The LSLGM algorithm is designed and implemented based on a class of grid PEA architecture, which considers the execution delay of the operations and the times of using PEA or other indexes.

(2) The optimizations of communication costs between one PEA and others are considered.

5.2 LSLGM algorithm design

The scheduling of starting point for LSLGM is PE (0, 0), and the nodes of DFG are mapped onto PEA by row in first. The LSLGM algorithm has considered grid PEA parallelism and delays of equalization, etc. The relevant strategies are as follows.

Strategy 1 Considering maximizing the number of parallel execution nodes of grid PEA.

LSLM scans the ready nodes of subDFGs by level in proper order. Priority is given to nodes with smaller levels in the same case, and nodes that meet the requirements are mapped to a level on the PEA. Thus, the number of mapped parallel nodes in the PEA can be maximized to reduce S_{SD}. From the above, Strategy 1 focuses on optimizing S_{SD}.

Strategy 2 Dynamically considering the delays of equalization for the mapping nodes.

The specific methods for Strategy 2 are as follows.

(1) Based on Strategy 1, nodes with current level 0 are arranged in the ready queue by layer mapping; also, current level nodes with an indegree of 0 are arranged in the ready queue, and the delays of nodes are calculated. Nodes with larger delays are scheduled and mapped onto PEA due to high priority, and the same delay nodes are placed in the same row of PEA as far as possible.

(2) After mapping a node, the indegree of its direct successor will be updated. If the indegree of a node's direct successor is 0, it is directly partitioned in the ready queue.

The first row is full, and it is placed in the second row to minimize S_{SD}. When the first row is full, the second row will be mapped in turn. As such, a small execution delay in PEA will be obtained.

Example 1 The delay equalization mapping node is shown in Fig. 4. The target architecture is grid $\mathrm{PEA}_{1\times2}$ (see Fig. 4a). A critical loop subgraph is being mapped (see Fig. 4b). Regarding mapping methods, Scheme 1 adopts level mapping. Figure 4e shows the mapping result. Scheme 2 adopts equalization mapping. Figure 4g shows the mapping result. The mapping parameters of the two schemes show that Scheme 1 has $M = 4$ and $S_{\mathrm{SD}} = 24$; Scheme 2 has $M = 4$ and $S_{\mathrm{SD}} = 16$. The S_{SD} for equalization mapping has been reduced by 8 cycles. The results show that the advantages of equalization mapping are obvious.

From above, Strategy 2 further optimized the node execution delay S_{SD} in PEA based on Strategy 1.

Strategy 3 Considering communication cost optimization between one PEA and others and effectively using reconfigurable hardware fragments.

Under the condition of ensuring S_{SD} minimization, if a node, which does not meet the requirements, is encountered in DFG mapping, it will stop scheduling. However, the remaining hardware area may also meet the requirements of the ready node behind the current node. As such, numerous hardware fragments will be generated.

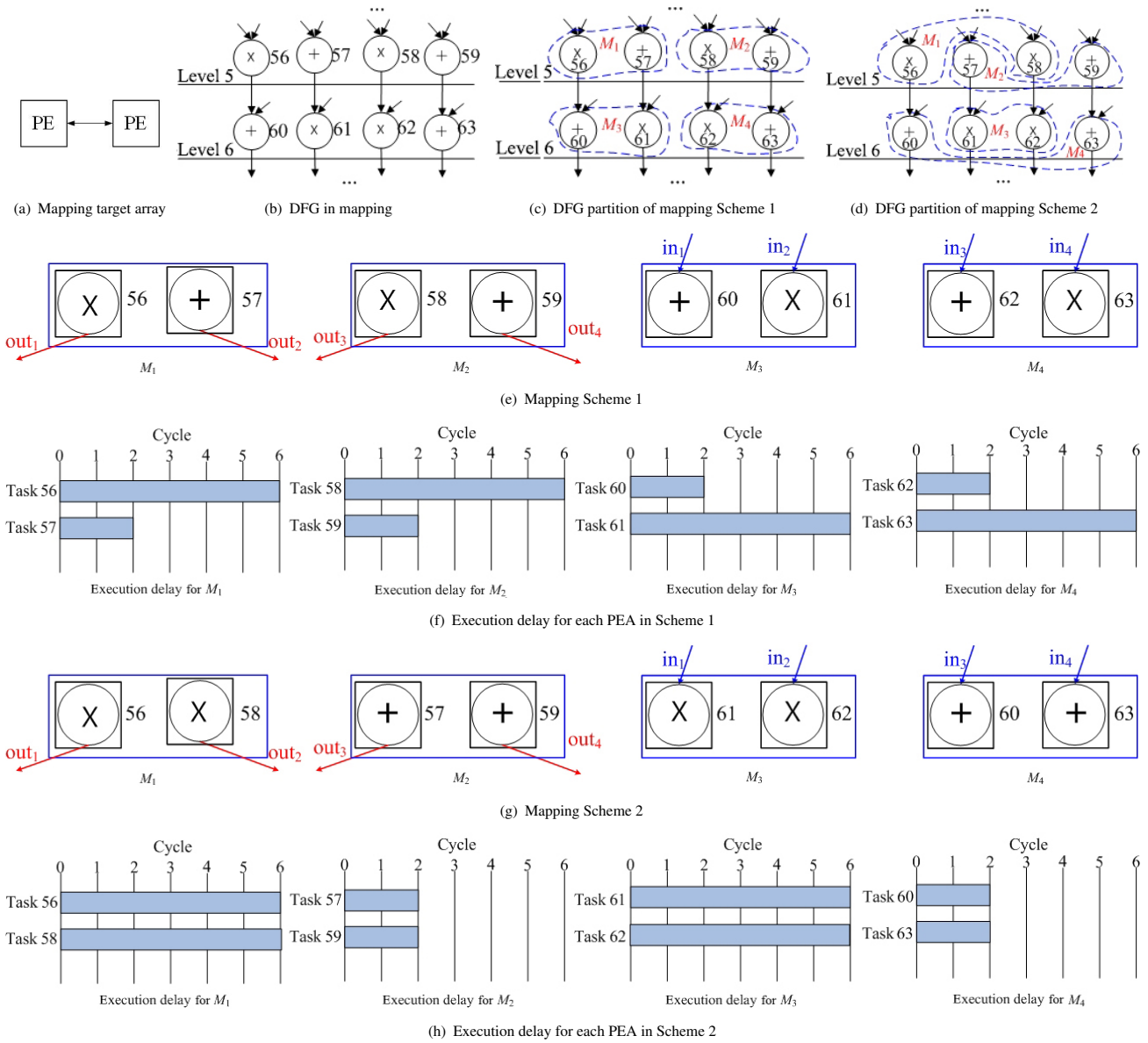
First, LSLGM dynamically tracks the process of DFG partitioning. When the nodes in LSLGM blocks that do not meet the requirements, it searches for nodes that meet the requirements and maps them onto the current PEA. Thus, the optimization of M is obtained. Second, under the condition of ensuring S_{SD} minimization, the successor of the currently block nodes is mapped as soon as possible. As a result, the optimization of N_1 and N_2 is achieved.

From above, Strategy 3 further optimized the communication costs between blocks N_1 and N_2 and the number of M.

According to Strategy 1, we implemented the LSLM by Algorithm 2. The purpose of the LSLM algorithm is to minimize S_{SD}. A single-objective optimization scheduling function $y(v_i)$ is constructed by LSLM. The scheduling is performed using $y(v_i)$, and the operations with the greatest function value exhibit a high priority by arrangement. The function form is as follows:

$$y(v_i) = num(v_i) + high(v_i) \tag{1}$$

where $num(v_i)$ denotes node number of v_i, and $high(v_i)$ denotes the level number of node high v_i in



(a) Mapping target array (b) DFG in mapping (c) DFG partition of mapping Scheme 1 (d) DFG partition of mapping Scheme 2

(e) Mapping Scheme 1

(f) Execution delay for each PEA in Scheme 1

(g) Mapping Scheme 2

(h) Execution delay for each PEA in Scheme 2

Fig. 4 Illustration of the equalization time for mapping nodes.

DFG.

In Algorithm 2, Step 1 reads the loop subgraph data table and initializes variables in LSLM. Step 2 scans the entire data table, calculates the weights of the ready nodes, and sorts them according to Eq. (1). The time complexity is $O(n)$, where n represents the number of nodes. Steps 3 and 4 scan the entire data table and select the computing-ready nodes with high priority for mapping in turn. When a node is mapped, the indegree of the successor node decreases by 1. Step 5 shows that if a PEA is full or not but the node cannot be mapped according to the hardware constraints, the PEA and related variables are initialized, then return to Step 3. The time complexity is about $O(n)$. Step 6 indicates that if all DFG nodes are mapped, which will jump out of the loop. Step 7 shows that six parameters such as $M, N_1, N_2, S_{\mathrm{SD}}, C_{\mathrm{CON}}$, and T_{TOTAL} are calculated by corresponding functions. Step 8 is the end of LSLM. In summary, the time complexity of the LSLM is about $O(n)$.

According to Strategies 1, 2, and 3, the LSLGM algorithm was designed and implemented. The purpose of the LSLGM algorithm is to optimize S_{SD}, M, N_1, N_2, and other indicators; the LSLGM algorithm constructs a multiobjective optimization scheduling function $h(v_i)$. The scheduling mapping is performed using $h(v_i)$. The operation with the greatest value exhibits high priority. The function form is as follows:

$$h(v_i) = num(v_i) + high(v_i) + delay(v_i) + p(v_i|v_j) \quad (2)$$

Algorithm 2 LSLM

Input: loop_subDFG
Output: Configuration information, M, N_1, N_2, S_{SD}, C_{CON}, and T_{TOTAL}
Constraint: A_{RPU}=16 or 49, illegal data dependencies are not allowed, GCGRA
Step 1:
Read *loop_subDFG_table*( ); initialize PEA matrix and variables, $M=0, N_1=0, N_2=0, S_{\mathrm{SD}}=0, C_{\mathrm{CON}}=0, T_{\mathrm{TOTAL}}=0$, and $n=0$;
Step 2:
for 1 (v_i = 1 to *node number*) **do**
Compute the node weight values by $y(v_i)$ and sort them;
end for 1
Step 3:
for 2 (v_i = 1 to *node number*) **do**
Step 4:
Select the minimum value of $y(v_i)$ and $I_{indegree}[v_i].id$ = 0; PEA[*matrix-row*][*num*[*matrix-row*]].*id* = $N_{node}[v_i]$.*id*; $N_{node}[v_i]$.*flag* = 1; *n*++; *num*[*matrix-row*]++; v_i successor $I_{indegree}(S_{succ}(v_i))-1$;
Step 5:
if(a PEA is filled with nodes || a PEA is not filled with nodes but cannot map by hardware constraint)
Change PEA block; clear PEA to zero, initializate variables, and update ready queue by $y(v_i)$;
Compute $M, N_1, N_2, S_{\mathrm{SD}}, C_{\mathrm{CON}}$, and T_{TOTAL} by *Cal_parameter*( ); *matrix-row* = 0; *num*[*matrix-row*] = 0;
go to Step 3;
Step 6:
if (*n*=*node number*) **break;**
end for 2
Step 7:
$M, N_1, N_2, S_{\mathrm{SD}}, C_{\mathrm{CON}}$ are got by $M( )$, $N_1( )$, $N_2( )$, $S_{\mathrm{SD}}( )$, $C_{\mathrm{CON}}( )$, respectively; T_{TOTAL} is obtained by *delays*( );
Step 8: end LSLM.

where $delay(v_i)$ represents the delay of v_i. After v_i's direct predecessor node v_j is mapped completely, $p(v_i|v_j)$ represents the probability when v_i has become a ready node. After LSLGM has obtained a smaller S_{SD} and smaller communication costs between PEAs, it optimizes M, N_1, N_2, and other indicators using greedy mapping in the process of scheduling.

LSLGM is designed and implemented by Strategies 1, 2, and 3, which is described as follows (Algorithm 3).

In Algorithm 3, Step 1 reads the loop subgraph data table and initializes variables in LSLGM. Step 2 scans the entire data table, calculates the weights of ready nodes according to Eq. (2), and sorts them; the time complexity is about $O(n)$, where n represents the number of DFG nodes. Step 3 scans the entire data table and PEA array if the ready node is mapped to end this

Algorithm 3 LSLGM

Input: loop_subDFG
Output: $M, N_1, N_2, S_{\mathrm{SD}}, C_{\mathrm{CON}}$, and T_{TOTAL}
Constraint: A_{RPU}=16 or 49, illegal data dependencies are not allowed, GCGRA
Step 1:
Read $loop\_subDFG\_table( )$; initialize PEA matrix and variables $M=0$, $N_1=0$, $N_2=0$, $S_{\mathrm{SD}}=0$, $C_{\mathrm{CON}}=0$, $T_{\mathrm{TOTAL}}=0$, and $n=0$;
Step 2:
for 1 (v_i=1 to $node\ number$) **do**
Compute the node weight values by $h(v_i)$ and sort them;
end for 1
Step 3:
for 2 (v_i = 1 to *node number*) **do**
if 1 (ready node is mapped) **continue**;
end if 1
end for 2
for 3 (i=0 to PEA_row) **do**
for 4 (j=0 to PEA_col) **do**
if 2 (Current PEA has placement positions) **then**
Select the minimum value of $y(v_i)$ and $I_{indegree}[v_i].id$=0; As for up, down, left, and right of current PEA, ready nodes are placed by place(*p,*i,*j); mapped node v_i successor $I_{indegree}(S_{succ}(v_i))$–1;
else break;
end if 2
end for 4
end for 3
Step 4:
if 3 (Same level nodes are mapped) **then**
Scan loop_subDFG_table to search for the next level ready nodes of DFG;
end if 3
if 4 (The next level ready nodes exhibit predecessor nodes) **then**
if 5 (Predecessor nodes have mapped) Map ready nodes to current PEA by place (*p,*i,*j); mapped node v_i successor $I_{indegree}(S_{succ}(v_i))-1$; **break;**
else continue
end if 5
else
Ready nodes are placed by place (*p,*i,*j); mapped node v_i successor $I_{indegree}(S_{succ}(v_i))$–1;
break
end if 4
Step 5:
if 6 (A PEA is filled with nodes || a PEA is not filled with nodes, but it cannot map by hardware constraint) **then**
Change PEA block; clear PEA to zero, initialize variables, update ready queue by $y(v_i)$;
Compute $M, N_1, N_2, S_{\mathrm{SD}}, C_{\mathrm{CON}}$, and T_{TOTAL} by $Cal\_parameter( )$; $matrix\text{-}row$=0; and num [$matrix\text{-}row$]=0;
goto Step 3;
end if 6
Step 6:
if 7 (*n*=*node number*)
break;
end if 7
Step 7:
$M, N_1, N_2, S_{\mathrm{SD}}, C_{\mathrm{CON}}$ are got by $M( )$, $N_1( )$, $N_2( )$, $S_{\mathrm{SD}}( )$, $C_{\mathrm{CON}}( )$, respectively; T_{TOTAL} is obtained by *delays*( );
Step 8: end LSLGM.

time cycle. If the current PEA is empty, the mapping will be terminated; otherwise, the high priority and ready nodes are selected to map. Each node is mapped, and the indegree of the successor node decreases by 1. Step 4 indicates that the current layer nodes have been mapped; then, Step 4 found the next ready node. If this ready node exhibits no precursor, it will be mapped directly. If the precursor of the current ready node has been placed in the current PEA or the front PEA and the PEA has a legal vacancy, the current ready node will be mapped at once; otherwise, the scheduling will be ended. The time complexity is about $O(n)$. Step 5 illustrates that if a PEA is full or not but the ready node cannot be mapped according to the hardware constraints, PEA and related variables are initialized, and it is returned to Step 3. The time complexity is about $O(1)$. If all nodes of the loop DFGs are mapped completely in Step 6, this cycle will be ended. The time complexity is about $O(n \times PEA\_row \times PEA\_col)$. PEA_row and PEA_col represent the number of rows and columns in one PEA, respectively. Step 7 is the end of LSLGM.

In summary, the time complexity of LSLGM is about $O(n \times PEA\_row \times PEA\_col)$.

Example 2 To illustrate the mapping effect of LSLGM, the following examples are considered. A loop code is converted into a DFG. As shown in Figs. 5a and 5b, a loop subDFG contains 32 original inputs, 4 original outputs, 28 operation nodes, 24 nonoriginal edges, 16 addition operations, and 12 multiplication operations. The partition results of LSLM and LSLGM are shown in Figs. 5c and 5d. The LSLM and LSLGM mapping scheduling results are shown in Figs. 5f and 5g. The comparison of LSLM and LSLGM mapping results are shown in Table 1. The results show that LSLGM exhibits better optimization.

6 Experiment and Comparison Analysis

6.1 Benchmarks

To compare different mapping algorithms, we designed two types of benchmark programs.

(1) Binary image algebra or logic operations (all of them are expanded by 4 times): Binary Image ADDITION (BIAD), Binary Image AND (BIA), Binary Image NOT (BIN), and Binary Image XOR (BIX).

(2) An example of loop 4 (LOOP4) times expansion, Matrix Multiplication 4 (MM4) expansion, FDCT3, FDCT4, EWF3, and EWF4[7]. Table 2 shows these benchmarks. The latency and amount of resources used for various operations have been explained previously; 10 benchmarks were used. The number of operands are listed in Table 2 ("add" represents addition, "sub" represents subtraction, "mul" represents multiplication, "as" represents assignment, "fe" represents fetch content, "aj" represents assignment judge, "ju" represents judge, "gr" represents greater than, "le" represents less than, "eq" represents equal, "and" represents and operation, and "ej" represents equal judge).

6.2 Experimental analysis and comparison

To facilitate comparison, we used the BIAD, BIA, BIN, BIX, LOOP4, MM4, FDCT3, FDCT4, EWF3, and EWF4 benchmark programs (Table 2). The A_{RPU} values are 16 ($\mathrm{PEA}_{4\times4}$) and 49 ($\mathrm{PEA}_{7\times7}$) at random. For different A_{RPU} values, LSLGM, LSLM, and EPIMap are tested by a set of benchmarks. Five indexes (i.e., M, N_1, N_2, S_{SD}, and C_{CON}) are considered in this study. We implemented LSLM, LSLGM, and EPIMap in C++.

6.2.1 Comparison of LSLM, LSLGM, and EPIMap

(1) Comparing M of LSLM, LSLGM, and EPIMap for GCGRA

From Fig. 6, when $A_{\mathrm{RPU}}=16$ and $A_{\mathrm{RPU}}=49$, compared with LSLM and EPIMap, LSLGM acquired all optimizations of M. The reason is that LSLGM adopts greedy thinking and puts all nodes that meet the mapping conditions as soon as possible.

(2) Comparing N_1 and N_2 of LSLM, LSLGM, and EPIMap for GCGRA

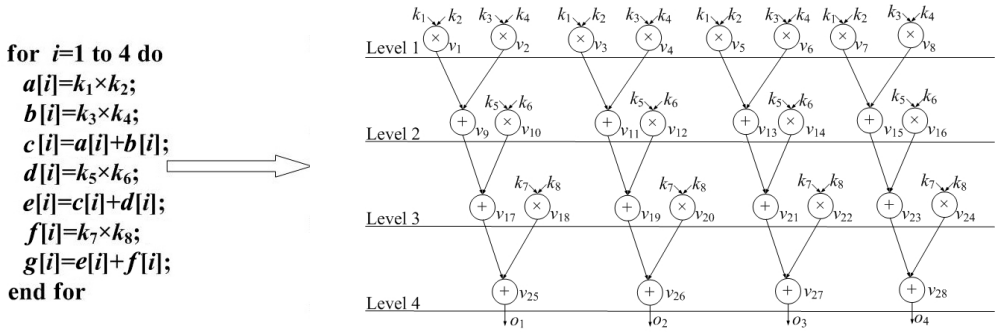
From Figs. 7 and 8, when $A_{\mathrm{RPU}} = 16$ and $A_{\mathrm{RPU}} = 49$, compared with LSLM, except for LOOP4 and FDCT4, LSLGM exhibits comprehensive optimization in terms of N_1 and N_2. N_1 and N_2 indexes obtained by LSLGM are better than those by EPIMap because EPIMap mainly pursues the minimum II (initiation interval); hence, N_1 and N_2 values are not well-considered.

(3) Comparing S_{SD} of LSLM, LSLGM, and EPIMap for GCGRA

From Fig. 9, when $A_{\mathrm{RPU}} = 16$ and $A_{\mathrm{RPU}} = 49$, LSLGM considers multiple index optimization and is inferior to LSLM and EPIMap in S_{SD} because LSLM focuses on optimizing the row parallelism of nodes, and EPIMap focuses on optimizing the loop II of computing nodes, which decreases S_{SD}.

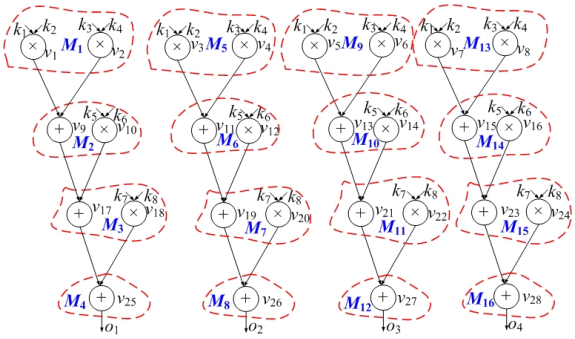
(4) Comparing C_{CON} of LSLM, LSLGM, and EPIMap for GCGRA

From Fig. 10, when $A_{\mathrm{RPU}} = 16$ and $A_{\mathrm{RPU}} = 49$, LSLM and EPIMap consider the optimization of S_{SD},

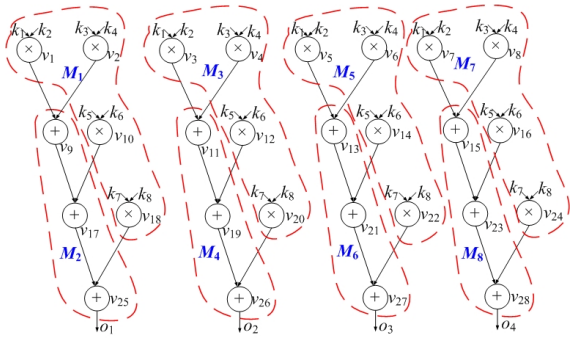


(a) Loop code

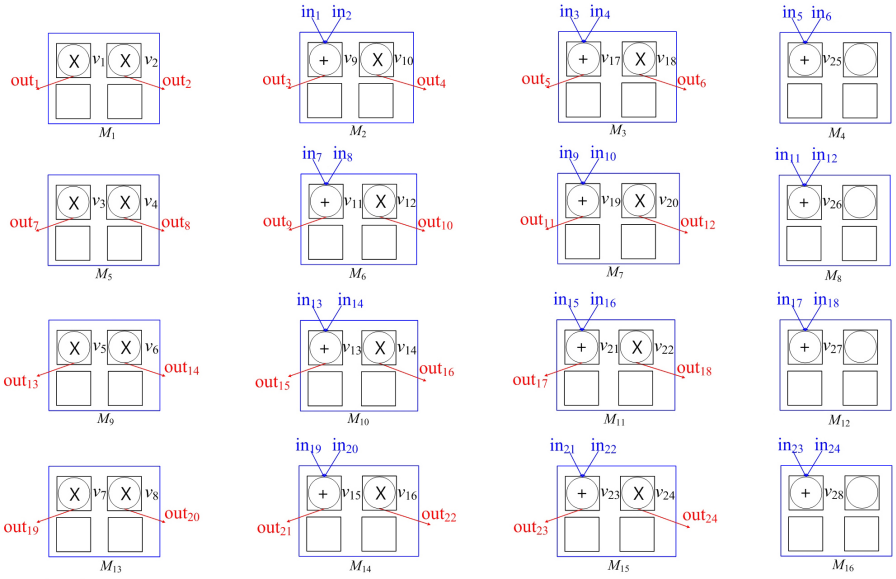
(b) Loop 4 times for unrolling of DFG



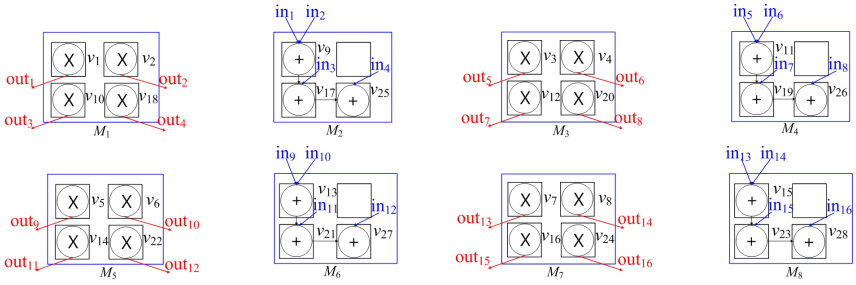
(c) Loop subDFG partition



(d) Loop subDFG greedy partition



(e) Mapping results of LSLM ($PEA_{4\times4}$)



(f) Mapping results of LSLGM ($PEA_{4\times4}$)

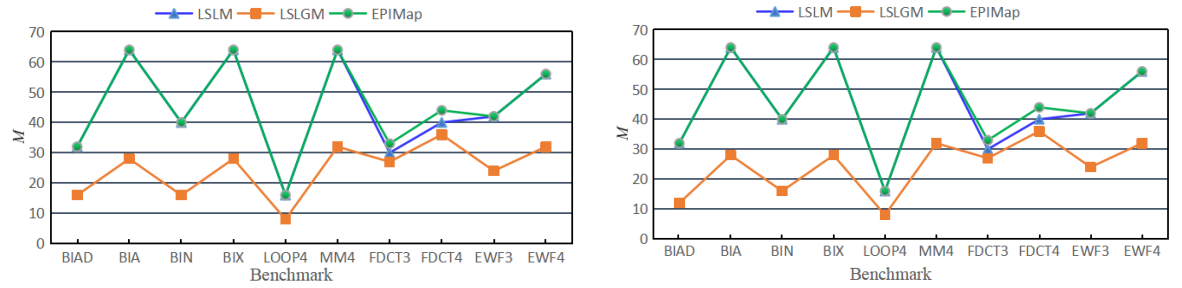
Fig. 5 Example of loop code unrolling.

Table 1 Mapping parameter comparison of LSLM and LSLGM.

Algorithm	M	N_1	N_2	S_{SD}	C_{CON}	T_{TOTAL} (clock cycle)
LSLM	16	24	24	80	300	428
LSLGM	8	16	16	48	164	244

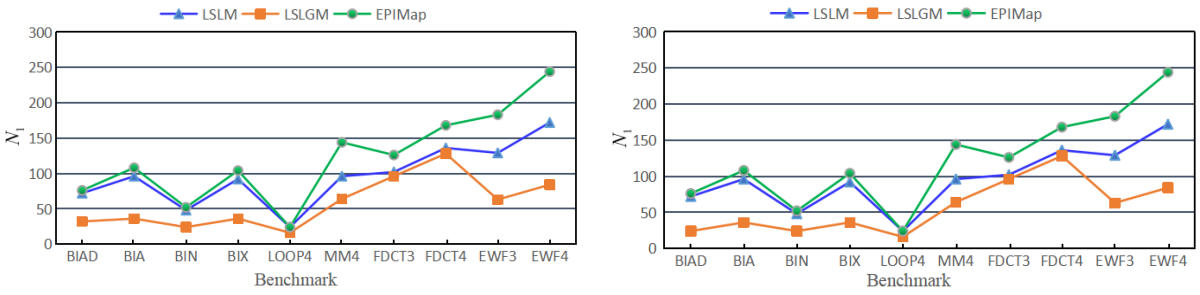
Table 2 Number of operations of benchmarks.

Benchmark	total	add	sub	mul	as	fe	aj	ju	gr	le	eq	and	ej
BIAD	76	28	–	12	12	12	8	–	4	–	–	–	–
BIA	108	24	–	24	–	24	16	8	8	–	–	4	–
BIN	56	12	–	12	–	12	12	–	4	–	–	–	4
BIX	104	28	–	24	–	24	16	4	8	–	–	–	–
LOOP4	28	12	–	16	–	–	–	–	–	–	–	–	–
MM4	112	48	–	64	–	–	–	–	–	–	–	–	–
FDCT3	126	39	39	48	–	–	–	–	–	–	–	–	–
FDCT4	168	52	52	64	–	–	–	–	–	–	–	–	–
EWF3	102	84	–	18	–	–	–	–	–	–	–	–	–
EWF4	136	112	–	24	–	–	–	–	–	–	–	–	–



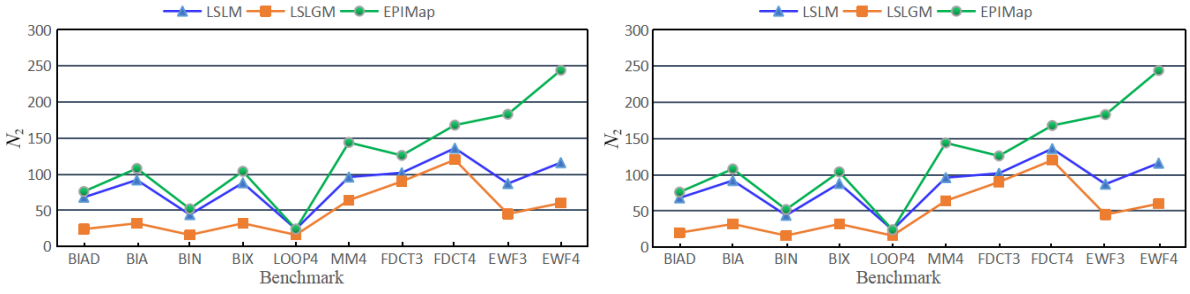
(a) $A_{RPU} = 16$ (b) $A_{RPU} = 49$

Fig. 6 Comparison of M on LSLM, LSLGM, and EPIMap.



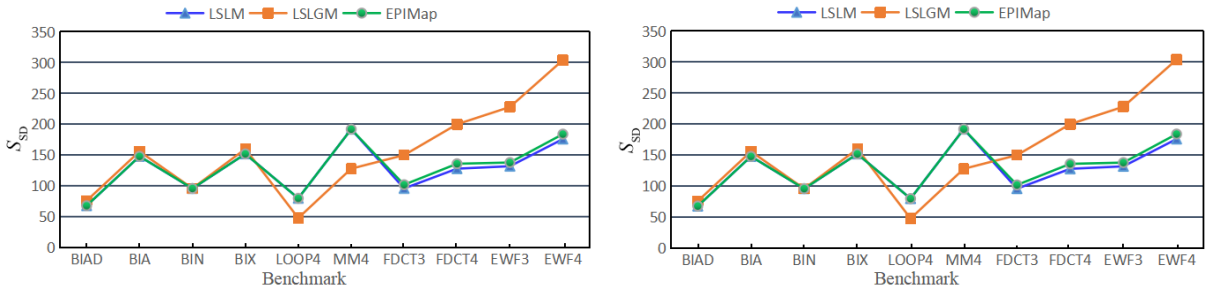
(a) $A_{RPU} = 16$ (b) $A_{RPU} = 49$

Fig. 7 Comparison of N_1 on LSLM, LSLGM, and EPIMap.



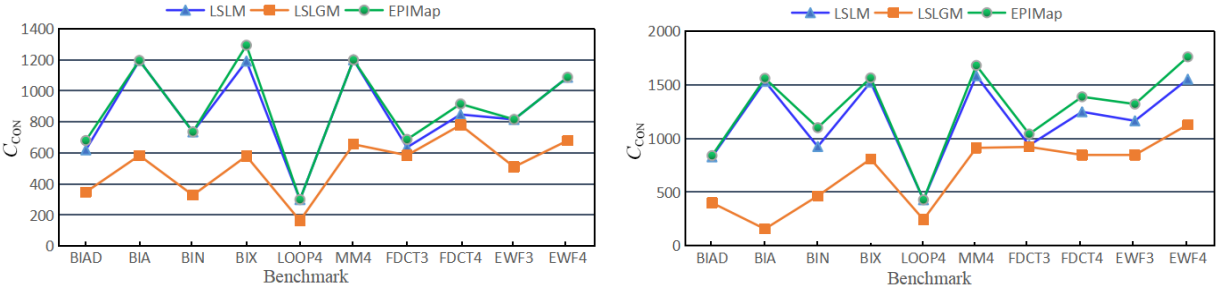
(a) $A_{RPU} = 16$ (b) $A_{RPU} = 49$

Fig. 8 Comparison of N_2 on LSLM, LSLGM, and EPIMap.



(a) $A_{RPU} = 16$ (b) $A_{RPU} = 49$

Fig. 9 Comparison of S_{SD} on LSLM, LSLGM, and EPIMap.



(a) $A_{RPU} = 16$ (b) $A_{RPU} = 49$

Fig. 10 Comparison of C_{CON} on LSLM, LSLGM, and EPIMap.

which increases M. The PEA fixed-configuration costs are increased by M. Therefore, compared with LSLM and EPIMap, LSLGM has optimized for C_{CON}.

(5) Comparing T_{TOTAL} of LSLM, LSLGM, and EPIMap for GCGRA

From Table 3, when $A_{RPU} = 16$ and $A_{RPU} = 49$, compared with LSLM, owing to insufficient consideration of the data access mechanism between PEAs by LSLM, LSLM's T_{TOTAL} is larger. LSLGM achieved an overall optimization of 33.0% (A_{RPU}=16) and 33.9% (A_{RPU}=49) of T_{TOTAL}. Owing to subgraph mapping, the area of reconfigurable processing unit has little effect on the T_{TOTAL} for the three algorithms except BIAD. The overall performance of LSLGM is better than that of EPIMap because EPIMap exhibits a larger S_{SD} and C_{CON}. The average improvement percentages of T_{TOTAL} obtained by LSLGM are 38.1% ($A_{RPU} = 16$) and 39.0% ($A_{RPU} = 49$). From Table 3, based on ten benchmarks, compared with LSLM, the percentage of LSLGM improvement is represented by Δ_1%; compared with EPIMap, the percentage of LSLGM improvement is represented by Δ_2%.

6.2.2 Applications of LSLGM

Compared with LSLM and EPIMap, LSLGM obtains better optimization in terms of M, N_1, and N_2. However, the value of S_{SD} obtained by LSLGM is worse than that of EPIMap. Because the number of modules and

Table 3 Comparison of mapping T_{TOTAL} on LSLM, EPIMap, LSLGM at A_{RPU} = 16 and 49.

Benchmark	LSLM		LSLGM		EPIMap		Δ_1 (%)		Δ_2 (%)	
	A_{RPU}=16	A_{RPU}=49	A_{RPU}=16	A_{RPU}=49	A_{RPU}=16	A_{RPU}=49	A_{RPU}=16	A_{RPU}=49	A_{RPU}=16	A_{RPU}=49
BIAD	828	828	480	400	840	840	–42.0	–51.7	–42.9	–52.4
BIA	1532	1532	808	808	1560	1560	–47.3	–47.3	–48.2	–48.2
BIN	924	924	464	464	1100	1100	–50.0	–50.0	464	–57.8
BIX	1524	1524	808	808	1564	1564	–47.0	–47.0	–48.3	–48.3
LOOP4	428	428	244	244	428	428	–43.0	–43.0	–43.0	–43.0
MM4	1584	1584	912	912	1680	1680	–42.4	–42.4	–45.7	–45.7
FDCT3	936	936	921	921	1041	1041	–1.6	–1.6	–11.5	–11.5
FDCT4	1248	1248	1228	1228	1388	1388	–1.6	–1.6	–11.5	–11.5
EWF3	1164	1164	846	846	1320	1320	–27.3	–27.3	–35.9	–35.9
EWF4	1552	1552	1128	1128	1760	1760	–27.3	–27.3	–35.9	–35.9
Average	–	–	–	–	–	–	–33.0	–33.9	–38.1	–39.0

the communication costs between PEAs are reduced, the parallelism between operations in each mapping module is not optimal. Therefore, LSLGM is suitable for the small-space storage or fetch costs and few partition modules.

7 Conclusion

In this study, an LSLGM algorithm considering the data communication cost, calculation delay, and number of configuration blocks is proposed. LSLGM, LSLM, and EPIMap are tested and compared via a set of benchmark programs. The experimental results show that LSLGM demonstrates advantages in M, N_1, and N_2 optimization. LSLGM is feasible in reducing N_1, N_2, and T_{TOTAL} based on GCGRAs.

Acknowledgment

This research was supported by the Natural Science Foundation of Anhui Province (No. 1808085MF203) and the Natural Science Foundation of China (Nos. 61972438 and 61432017).

References

[1] M. Brandalero, L. Carro, A. C. S. Beck, and M. Shafique, Multi-target adaptive reconfigurable acceleration for low-power IoT processing, *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 83–98, 2021.

[2] I. Bae, B. Harris, H. Min, and B. Egger, Auto-tuning CNNs for coarse-grained reconfigurable array-based accelerators, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2301–2310, 2018.

[3] J. M. P. Cardoso, P. C. Diniz, and M. Weinhardt, Compiling for reconfigurable computing: A survey, *ACM Comput. Surv.*, vol. 42, no. 4, p. 13, 2010.

[4] G. Charitopoulos, D. N. Pnevmatikatos, and G. Gaydadjiev, MC-DeF: Creating customized CGRAs for dataflow applications, *ACM Trans. Archit. Code Optim.*, vol. 18, no. 3, p. 26, 2021.

[5] L. B. Liu, J. F. Zhu, Z. S. Li, Y. N. Lu, Y. D. Deng, J. Han, S. Y. Yin, and S. J. Wei, A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications, *ACM Comput. Surv.*, vol. 52, no. 6, p. 118, 2020.

[6] N. J. Chen, Z. Wang, R. X. He, J. H. Jiang, F. Cheng, and C. H. Han, Efficient scheduling mapping algorithm for row parallel coarse-grained reconfigurable architecture, *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 724–735, 2021.

[7] N. J. Chen and J. H. Jiang, Mapping algorithm of coarse grained reconfigurable cell array for multi-branch tree data flow graph, (in Chinese), *J. Comput. -Aided Des. Comput. Graphics*, vol. 28, no. 7, pp. 1180–1187, 2016.

[8] N. J. Chen and Z. Y. Feng, Interconnect delay performance evaluation for non-crossing level and row operands parallel RCA, (in Chinese), *J. Tianjin Univ. (Sci. Technol.)*, vol. 50, no. 4, pp. 429–436, 2017.

[9] M. Balasubramanian and A. Shrivastava, CRIMSON: Compute-intensive loop acceleration by randomized iterative modulo scheduling and optimized mapping on CGRAs, *IEEE Trans. Comput. -Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3300–3310, 2020.

[10] G. Lee, E. Cetin, and O. Diessel, Fault recovery time analysis for coarse-grained reconfigurable architectures, *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, p. 42, 2018.

[11] T. Kojima, N. A. V. Doan, and H. Amano, GenMap: A genetic algorithmic approach for optimizing spatial mapping of coarse-grained reconfigurable architectures, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 11, pp. 2383–2396, 2020.

[12] M. Hamzeh, A. Shrivastava, and S. Vrudhula, EPIMap: Using Epimorphism to map applications on CGRAs, in *Proc. 2012 DAC Design Automation Conf.*, San Francisco, CA, USA, 2012, pp. 1280–1287.

[13] M. Hamzeh, A. Shrivastava, and S. Vrudhula, REGIMap: Register-aware application mapping on Coarse-Grained Reconfigurable Architectures (CGRAs), in *Proc.*50^{th} *ACM/EDAC/IEEE Design Automation Conf. (DAC)*, Austin, TX, USA, 2013, pp. 1–10.

[14] N. J. Chen, J. H. Jiang, X. Chen, Z. Zhou, and Y. Xu, An improved level partitioning algorithm considering minimum execution delay and resource restraints, (in Chinese), *Acta Electron. Sin.*, vol. 40, no. 5, pp. 1055–1066, 2012.

[15] N. J. Chen, Z. Y. Feng, and J. H. Jiang, Bypass node non-redundant adding algorithm for crossing-level data transmission in two-dimension reconfigurable cell array, (in Chinese), *J. Commun.*, vol. 36, no. 4, p. 2015132, 2015.

Naijin Chen received the PhD degree in computer science and technology from Tongji University, Shanghai, China in 2013. He obtained the postdoctoral certificate from Tianjin University, Tianjin, China in 2016. He is a member of China Computer Federation. He is currently a professor at Anhui Polytechnic University, Wuhu, China. His current research interests include reconfigurable computing and compiling, fault tolerant computing, reliability evaluation of high-level circuits, approximate computing, formal verification, semantic big data representation and reasoning, and pattern recognition and image processing.

Jianhui Jiang received the PhD degree in traffic information engineering and control from Shanghai Tiedao University (in April 2000, it was merged to Tongji University), China in 1999. Since 2011, he has been the associate dean at the School of Software Engineering, Tongji University. He is a professor and PhD supervisor at Tongji University. He is a senior member of China Computer Federation. His main research interests include reconfigurable computing and compiling, dependable systems and networks, software reliability engineering, and VLSI test and fault tolerance.

Fei Cheng received the BS degree from Anhui Polytechnic University, Wuhu, China in 2019. He is now a master student at School of Computer and Information Science, Anhui Polytechnic University, Wuhu, China. His current research interests include reconfigurable computing and compiling, formal verification, fault tolerant computing, semantic big data representation and reasoning, and pattern recognition and image processing.

Chenghao Han received the BS degree from Suzhou University, Suzhou, China in 2020. He is now a master student at School of Computer and Information Science, Anhui Polytechnic University, Wuhu, China. His current research interests include reconfigurable computing and compiling, formal verification, and fault tolerant computing.

Xiaoqing Wen received the BEng degree from Tsinghua University, China in 1986, the MEng degree from Hiroshima University, Japan in 1990, and the PhD degree from Osaka University, Japan in 1993. From 1993 to 1997, he was an assistant professor at Akita University, Japan. He was a visiting researcher at University of Wisconsin, Madison, USA from Oct. 1995 to Mar. 1996. He joined SynTest Technologies, Inc., USA in 1998, and served as its chief technology officer until 2003. In 2004, he joined Kyushu Institute of Technology, Japan, where he is currently a professor at Department of Computer Science and Networks. His research interests include VLSI test, diagnosis, and testable design. He co-authored and co-edited two books: *VLSI Test Principles and Architectures*: *Design for Testability* (Morgan Kaufmann, 2006) and *Power-Aware Testing and Test Strategies for Low Power Devices* (Springer, 2009). He also holds 42 U.S. patents and 14 Japan patents on VLSI testing. He is a fellow of the IEEE, a senior member of the IEICE, a senior member of the IPSJ, and a member of the REAJ.