IEICE TRANS. INF. & SYST., VOL.E96–D, NO.8 AUGUST 2013

PAPER Special Section on Reconfigurable Systems

Affine Transformations for Communication and Reconfiguration Optimization of Mapping Loop Nests on CGRAs*

Shouyi YIN[†a)], ***Member***, **Dajiang LIU**[†], **Leibo LIU**[†], ***and*** **Shaojun WEI**[†], ***Nonmembers***

SUMMARY A coarse-grained reconfigurable architecture (CGRA) is typically hybrid architecture, which is composed of a reconfigurable processing unit (RPU) and a host microprocessor. Many computation-intensive kernels (e.g., loop nests) are often mapped onto RPUs to speed up the execution of programs. Thus, mapping optimization of loop nests is very important to improve the performance of CGRA. Processing element (PE) utilization rate, communication volume and reconfiguration cost are three crucial factors for the performance of RPUs. Loop transformations can affect these three performance influencing factors greatly, and would be of much significance when mapping loops onto RPUs. In this paper, a joint loop transformation approach for RPUs is proposed, where the PE utilization rate, communication cost and reconfiguration cost are under a joint consideration. Our approach could be integrated into compilers for CGRAs to improve the operating performance. Compared with the communication-minimal approach, experimental results show that our scheme can improve 5.8% and 13.6% of execution time on motion estimation (ME) and partial differential equation (PDE) solvers kernels, respectively. Also, run-time complexity is acceptable for the practical cases.

key words: *reconfigurable computing, loop transformation, polyhedra model, compiling*

1. Introduction

CGRA is a computing architecture combing the high performance of application specific integrated circuit (ASIC) with the much of the flexibility of general purpose processor (GPP). A CGRA typically consists of reconfigurable processing units (RPU) and a host microprocessor. With the presence of RPU, high performance of computation on some compute-intensive applications could be achieved and various applications could be implemented on CGRA.

However, tapping the computing potential of such hybrid architecture is a hard work and compilers for such system are urgently needed to leverage the synergies of the reconfigurable architectures. As the traditional sequential programming model is unfavorable to parallel computing and no acknowledged programming model is proposed for reconfigurable architecture, compiling for such architecture is a well-known challenge work.

RPUs typically handles computational intensive kernels in applications and much of them are loops. Mapping loops onto CGRAs to improve the execution performance is difficult due to their complex nature. Several techniques have been previously proposed to address the problems of mapping loops onto CGRAs. In [1], modular scheduling scheme is adopted to let a single loop to be executed in a loop pipeline fashion, which improves the throughout and the parallelism of CGRAs by overlapping the execution of different iterations of a loop. Since only a single iteration is analyzed, limited parallelism could be achieved. Loop unrolling [2], [3] is a common technique to generate a mapping scheme with greater parallelism. It unrolls a loop and transfer it into a DFG. Then, DFG based mapping optimization is performed to improve the execution performance. It is often used with other techniques such as loop shifting. This technology usually deals with a single loop for parallelization. In addition, the optimization opportunity in the step of DFG generation for original code is of little thought. Loop fission [4] is usually used to create smaller loops that have more speedup potential when parallelized, which is often applied to a single loop containing in the loop body one or more kernels. In [5], polyhedra model is firstly applied to the loop optimization for reconfigurable computing architectures, where both communication cost and PE utilization rate are considered. However, the reconfiguration cost is not taken into account in that work.

Most previous works have focus on loop transformation on CGRAs for communication-minimal parallelism, which is only one of the performance influencing factors of performance of CGRA. Since RPUs usually have limited resources and would be reconfigured many times to implement a computing task, the reconfiguration cost for changing the data path in RPUs could not be neglected. Thus, considering reconfiguration costs when applying loop transformation for communication minimal parallelization and PE utilization rate is of much significance.

In this paper, we focus on the polyhedral model [6] based loop transformations for coarse-grained reconfigurable architecture. The approach mainly find two hyperplanes to tile the original source code for both communication- and reconfiguration-minimal parallelism. When mapping loop nests on a 2-D PE array, we use scheduling hyperplane (Θ) to denote row related hyperplane for time tilling. By the scheduling hyperplane, the PEs are tiled in time domain, which means the tiled PEs execute in time sequencing. And we use space hyperplane (Π) to denote the column related hyperplane for space tiling. The

Manuscript received November 5, 2012.
Manuscript revised March 13, 2013.

[†]The authors are with Institute of Microelectronics, Tsinghua University, Beijing, China.

*This work was supported in part by the NNSF of China grant (No.61274131), the International S&T Cooperation Project of China grant (No.2012DFA11170), the Tsinghua Indigenous Research Project (No.20111080997) and the China National High Technologies Research Program (No.2012AA012701).

a) E-mail: yinsy@tsinghua.edu.cn

DOI: 10.1587/transinf.E96.D.1582

space hyperplane guarantees the tiled PEs in one row has no dependence and can execute in parallel. In our approaches, the three crucial factors in the architecture, PE utilization rate, communication cost and reconfiguration cost, are taken into a joint consideration. The proposed approaches include: 1). finding the best scheduling hyperplane. This step is of high priority because both communication volume and PE utilization rate are optimized in a unified way and it is also the basis reference of the second hyperplane. 2). finding the best space hyperplane independent of the scheduling hyperplane, where the trade-off between communication costs and configuration costs is made to get the global optimum by adding an angle constraint. This angle constraint represents the trade-off metric between communication cost and reconfiguration cost. With these two hyperplanes, a 2-dimensional loop nest could be partitioned into small tiles matching the size of an PE array. Then each tile can be mapped onto 2-dimensional PE array.

The rest of this paper is organized as follows. In Sect. 2, the target architecture of CGRA and the mathematical background of polyhedral model are provided. In Sect. 3, the relationships of PE utilization rate, communication cost and reconfiguration cost are analyzed in detail. Based on the previous section, Sect. 4 describes the formulation of optimization problems. Section 5 gives an efficient solution to the formulated problems. Then, Sect. 6 presents the experiments results that demonstrate the effectiveness of our optimization algorithm. At last, we conclude in Sect. 7.

2. Background and Notation

This section provides the target architecture of CGRA and the background of polyhedral model. In addition, the process of polyhedra model based mapping optimization of loop nests for CGRA is illustrated.

2.1 The Target Architecture of CGRA

The hardware architecture of coarse-grained reconfigurable computing is typically based on RPUs as a role of coprocessor and coupled to a host microprocessor, as depicted in Fig. 1, where the host microprocessor controls the execution of the whole system. The host microprocessor and RPU communicate with each other by memory (e.g., SRAM) out of chip. Each RPU was composed of one or more reconfigurable cell arrays (RCA), some data memory, a configuration memory, a configuration controller, etc. The different RCAs exchange data with a shared memory and the RCA internal memory stores the intermediate result of computation. The configuration memory stores the configuration contexts and the configuration controller controls the reconfiguration and execution of the RCA.

The typical structure of the RCA is a two-dimensional (2D) PE array connected by flexible routes. Each PE includes an ALU and some pass registers. The functionality of PE could be configured to be different word-level operations of fixed-point numbers. The routing style of PEs has

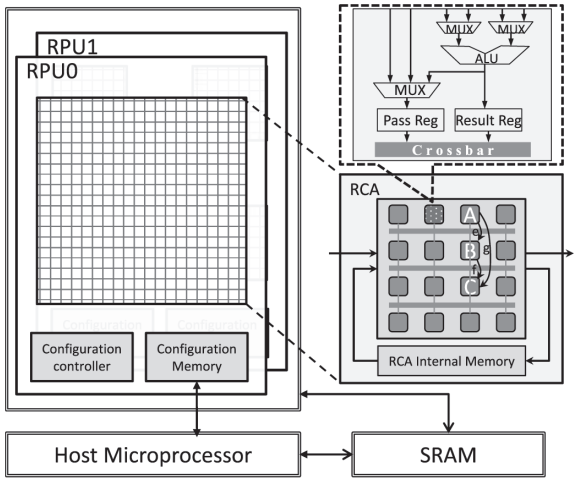


Fig. 1 Target architecture of coarse-grained reconfigurable computing.

great variety, and this paper mainly focus on the crossbar style connection [7], [8]. In crossbar style connection, the PEs in neighbor rows can communicate with each other directly through the horizontal bars. There is no direct connection between unneighbor PEs. The communication between unneighbor PEs must go through pass registers. For example, as shown in Fig. 1, the PE *A* can communicate with PE *B* directly since they are in neighbor rows. In contrast, the PE *A* must go through the pass register of PE *B* to communicate with PE *C*.

The function of each PE and the specific routing style could be reconfigurable by configuration contexts stored in the configuration memory. The reconfiguration cost (latency) depends on the size and bandwidth of the configuration memory greatly. The reconfiguration latency equals to the configuration context size divided by the configuration memory bandwidth. If the bandwidth is large enough, the latency could be very small. Moreover, if the size of configuration memory is big enough, the double-buffering technique can be used. Then the latency can be hidden greatly.

However, in practical implementation of CGRA (e.g. ReMUS [7] and PipeRench [8]), the configuration memory size and bandwidth is usually not big enough, due to the consideration of chip area and power consumption. For example, in ReMUS processor, the bandwidth of the configuration memory is 1024 bits and the configuration context size of one RCA operation is 128 words. So 4 cycles are need to finish the configuration of one RCA instance. In such cases, the reconfiguration cost cannot be ignored in performance analysis.

Our proposed loop optimization approach is based on the general reconfigurable architecture described above.

2.2 Polyhedra Model

The polyhedron model [6] is convenient alternative representation which combines analysis power, expressiveness and high flexibility. It first convert the original loop source code into polyhedral model intermediate representa-

tion (IR). Then affine transformations for optimization are performed on the polyhedral model IR. At last, the transformed polyhedral model IR is convert to optimized source code. This model is widely applied to GPP, GPU, etc and it is based on four main concepts: the iteration domain, the access function, dependence polyhedra and affine transformation. A program part that can be represented using the polyhedron model is called a Static Control Part or SCoP for short.

The iteration domain is defined by a system of affine inequalities, $\mathcal{D}_S(\vec{i_S}) \geq \vec{0}$, derived from the bounds of loops surrounding statement S. Using matrix to present the inequalities, the iteration space polytope is presented as:

$$\mathcal{D}_S \cdot \begin{pmatrix} \vec{i_S} \\ \vec{g_S} \\ 1 \end{pmatrix} \geq \vec{0} \tag{1}$$

where $\mathcal{D}_S$ is a matrix of n affine constraints on the execution of statement S. $\vec{i_S}$ is the iteration vector of statement S and $\vec{g_S}$ is a vector of global parameters.

Each reference in a statement is also affine functions of loop indices and global parameters, which could also be represented using matrices. if $\mathcal{F}_{kAS}(\vec{i_S})$ represents the access function of the k^{th} reference to an array A in statement S, then

$$\mathcal{F}_{kAS}(\vec{i_S}) = \mathcal{F}_{kAS} \cdot \begin{pmatrix} \vec{i_S} \\ \vec{g_S} \\ 1 \end{pmatrix} \tag{2}$$

where $\mathcal{F}_{kAS}$ is a matrix representing an affine mapping form the iteration space of statement S to the data space of array S. Each row in the matrix defines a mapping corresponding to one dimension of the data space.

The Polyhedral Dependence Graph (PDG) [9] is a directed multi-graph with each vertex representing a statement, and an edge, $e \in E$, from node S_i to S_j representing a polyhedral dependence from a dynamic instance of S_i to one of S_j: it is characterized by a polyhedron, Pe, called the dependence polyhedron that captures the exact dependence information corresponding to e. The dependence polyhedron is in the sum of the dimensionality of the source and target statements polyhedra. The h-transformation [10] h_e maps the target iteration vector $\vec{i_t}$ to the source iteration vector $\vec{i_s}$ that is the last access the same index of a array. So the dependence polyhedra can be represented as:

$$\mathcal{P}_e \equiv \left(\begin{array}{cc} \mathcal{D}_s & \\ & \mathcal{D}_t \\ \hline & h_e \end{array} \right) \begin{pmatrix} \vec{i_s} \\ \vec{i_t} \\ \vec{g} \\ 1 \end{pmatrix} \left(\begin{array}{cc} \geq & \vec{0} \\ \geq & \vec{0} \\ \hline = & \vec{0} \end{array} \right) \tag{3}$$

The affine transformation of a statement S is defined as an affine mapping that maps an instance of S in the original program to an instance in the transformed program. The transform function of a statement S is given by:

$$\Phi(\vec{i_S}) = \mathcal{T}_S \cdot \begin{pmatrix} \vec{i_S} \\ \vec{g_S} \\ 1 \end{pmatrix} \tag{4}$$

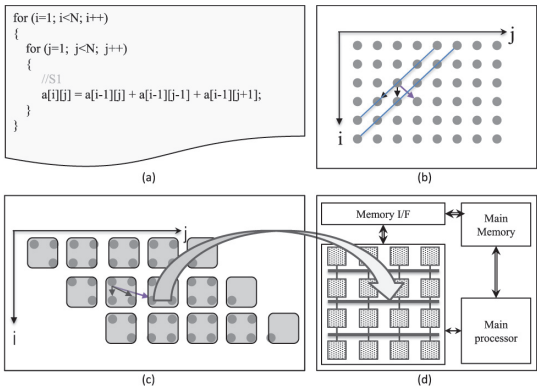


Fig. 2 The process of polyhedra model based mapping optimization.

where $\mathcal{T}$ is a row vector and the affine transformation is a one-dimensional mapping. Φ can also be called an affine hyperplane. An n-dimensional mapping can be represented as a combination of n (linearly independent) one-dimensional mappings. In the architecture of RCAs, we use space hyperplane Π to denote the space tiling (the hyperplane orthogonal to axis **j** in Fig. 2), and scheduling hyperplane Θ to denote the time tiling (the hyperplane orthogonal to axis **i**).

2.3 The Process of Polyhedra Model Based Mapping Optimization of Loops for CGRA

We first use polyhedral model for the mapping optimization of CGRAs in [5], where the mapping process is proposed as follow: (1) Convert the original loop nest into polyhedra intermediate representation (IR). (2) Find two 1-d affine transformations (i.e., two hyperplanes) on the polyhedra model IR, where some optimizations is performed. (3) Tile the iteration space by the two hyperplanes found in step (2) to match the size of RCA. (4) Map every tile generated in step (3) onto RCAs and generate configuration context for CGRA. The whole mapping process is expatiated in Fig. 2.

3. CGRA's Performance Factors Analysis

When mapping the loop nests onto CGRA, there are three major performance factors that affect the performance: communication cost, PE utilization rate and reconfiguration cost. In this section, we first illustrate the unity between PE utilization rate and communication cost. Then, the trade-off between communication cost and reconfiguration cost is analyzed.

The concept of communication cost refers to the data exchange between different loop instances. From the perspective of loop nests, the communication cost (or data exchange) is caused by the data dependence between different loop instances, which is presented by the arrows in Fig. 4 (c). Therefore the communication cost can be measured by the data volume need to be exchanged between different loop instances. Actually, when the loop nests are given, the above data volume can be calculated.

The communication cost affects the performance in two

respects. Due to the size limitation of RCA, only limited loop instances can be put into one RCA instance. There must be some data dependence (resulting in data communications) traversing the boundary of different RCA instance (the red arrows in Fig. 4 (c)). Since this kind of communication is between difference RCA instances, we call it as external communication. In practice, the external communication is implemented through shared memory. On the other hand, the data dependence between loop instances that are put in the same RCA instance will cause the data communications within RCA (the black arrows in Fig. 4 (c)). We call this kind of communication as internal communication. Since the internal communication is between different PEs, it can only be implemented by pass registers, which affects the PE utilization directly. It should be noted, the terms *internal* and *external* are just used to facilitate narration. In fact, both internal communication and external communication are caused by same data dependence in loop nests and have the same value.

3.1 The Unification between PE Utilization Rate and Communication Cost

The PE utilization rate is defined as the proportion of PEs used for computation in RCAs. Higher PE utilization rate means more PE were mapped with operators. The internal communication is the data exchange between different PEs in the same RCA instance. It can be only implemented by pass register, since the latency of internal memory access is not tolerable. Since the number of pass registers are limited in RCA, the use of pass register would occupy the resource of PEs. Therefore the PE utilization rate is inversely proportional to communication cost. And improving PE utilization rate is the same as reducing communication cost.

To illustrate this clearly, we define some tokens. As shown in Fig. 3, a combination of the framed R and circled + indicates an PE while the framed R and circled + mean a pass register and an ALU respectively. In addition, the horizontal bars between two nearby rows of PE means the crossbar style routes. As described in [5], a long dependence traversing more than one row (the vertical part of a dependence) in an RCA means more cost of pass registers, subsequently, lower utilization rate of PEs. As the crossbar style routes could just connect two PEs from two nearby rows, the life time of internal registers in RCA is one cycle. If a temporary value should maintain more than one control steps, it will be passed by pass a register step by step until it is consumed by an ALU operation. Thus, in an RCA, the longer the life time a temporary value is, the more number of pass registers will be used, which is generally limited in number in RCAs. Once all the pass registers in the row are occupied, some ALUs will be wasted because of lacking of pass register to pass temporary value. As a result, ALUs would be wasted in RCA because of long vertical dependence. As shown in Fig. 3, a vertical dependence with length of "1", "2", "3" and "4" will consume "0", "1", "2", "3" pass registers, respectively.

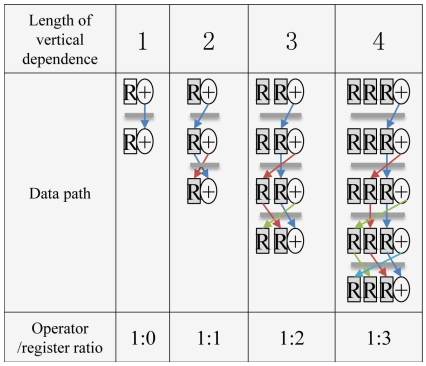


Fig. 3 The unity of PE utilization rate and internal communication.

For the feature of RCA described above, less internal communication (i.e. shorter data dependence) is expected for improving the PE utilization rate when finding the two hyperplanes. When the PE utilization rate is improved, the more PEs can be mapped with operators for computation. Therefore, the parallelization is also improved at the same time.

From the perspective of external communication, the more exchange data is, the more time is spent on the load and store of data (i.e. more external communication). In the optimization of loop in GPP, communication volume is given by the number of hyperplanes the dependence traverses along the hyperplane normal. The case in reconfigurable computing system is similar, where the minimal communication volume means a small quantity of data exchanges of different RCA through share memory or little temporary data in RCA internal memory. Reducing the communication cost is also helpful to reduce the data load and store latency.

From the discussion above, it is clear that short loop dependence (i.e. less loop communication) is good for increasing the PE utilization rate and reducing RCA external communication. In other words, reducing the communication cost will achieve both higher PE utilization rate and smaller data load and store latency. Therefore, the PE utilization and communication cost can be optimized in a unified way.

3.2 The Trade-Off between Communication Cost and Reconfiguration Cost

Reconfiguration cost means the time spent on the configuration of the RCA because of the change of data path in RCA. As an example to demonstrate the trade-off between communication cost and reconfiguration cost, we first focus on a simple code in Fig. 4 (a). The polyhedra representation of the original loop nest is depicted in Fig. 4 (b). There are two reference of the same array $\mathbf{a}$. One reference is $\mathbf{a}_w[i][j]$ with write operation, the other one is $\mathbf{a}_r[i-2][j-1]$ with read operation. So their is a dependence from reference (i, j) to reference $(i-2, j-1)$. The untransformed mapping scheme is depicted in Fig. 4 (c), where the nested data dependence could be see clearly. One of data dependence is marked with

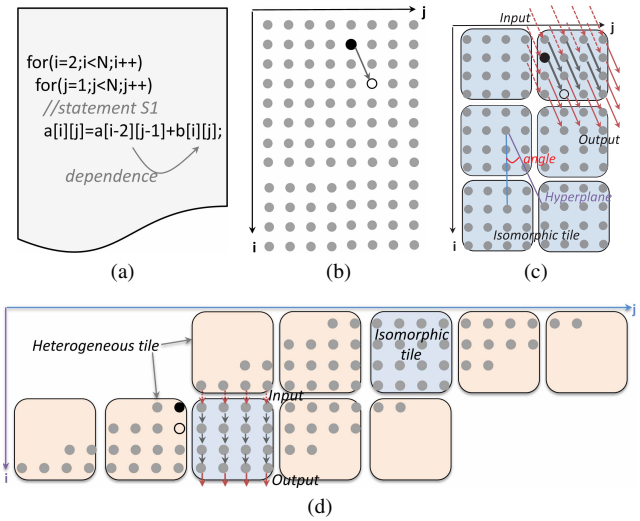


Fig. 4 A motivation example.

a doted arrow, where the black solid circular and vacant circular indicate the resource instance and target instance.

First, we do not apply any loop transformation to the original loop nest. We assume that the RCA is a 4×4 PE array and there is enough registers per PE to pass all the internal result in the RCA for pipelined operation. Then the original loop nest could be partitioned into small tiles matches the RCA size. With out any PE utilization and communication optimization, we find that 10 output data need to be export from the tile and 10 input data need to be import into the tile. So there are totally 20 data exchanges for one RCA operation. The communication cost is high. In spite of this, we find that the data path of almost all the tile is isomorphic. Consequently, the data path need not be reconfigured once the first operation instance is configured. In another word, the configuration cost of the original mapping is very low.

Then, we apply the communication minimal transformations to the original loop nest. With the same approaches in [9], we find two optimal hyperplanes for communication minimal parallelization. Then we figure out the tile size by RCA size and the operators of the loop statement. As a result, a transformation for communication cost is achieved, which is shown in Fig. 4 (d). In the optimized mapping approach, only 4 output data need to be export from the tile and 4 input data need to be import into the tile. So 8 data exchanges in total are required for one RCA operation compared with 20 in the original loop nest. However, we find that there are many heterogeneous tiles with different data path mapped to RCAs. That means the RCA need to be reconfigured many times because the data path changes frequently.

In this motivation example, we select the size parameter of the loop nest $N = 32$ for a quantitative analysis. As shown in Table 1, although the communication volume of the original mapping scheme is more than triple that of communication minimized scheme, the configuration times of the original mapping scheme is only once compared to 46 times of Communication Minimized Scheme.

Table 1 Communication volume and reconfiguration cost of the example loop nest with global parameter $N = 32$.

Transformations	Original Scheme	Communication Minimized Scheme
input data exchanges/tile	10	4
output data exchanges/tile	10	4
total data exchanges/tile	20	8
regular tiles	64	40
irregular tiles	0	38
total tiles	64	78
configuration times	1	46

Making a step further, we could conclude that the bigger the gap between the angle and 90° is, the more heterogeneous tiles will arise, where the angle means the angle between the two hyperplane partitioning the original loop nest. Consequently, there may be more configuration cost. Thus, this angle is a trade-off metric between Communication cost and Reconfiguration cost (C-R angle) and is an important constraint when formulating problems in next section.

From the discussion above, we find that there are close relationships between PE utilization rate, communication cost and configuration cost and they all have important influences on the performance of reconfigurable computing system. In this paper, instead of methods which may obtain suboptimal result for PE utilization rate, communication cost or reconfiguration cost requirement, we proposed a joint approach to optimize PE utilization rate, communication cost and configuration cost to get the global optimal results.

4. Problem Formulation

In this section, we describe the formulation of the problem. In order to simplify the problem, we make the following assumptions for the hardware architectures and applications. First, the hardware platform is a pipelined coarse-grained reconfigurable computing system, where the pass registers are limited in number. This is the most of the cases of pipelined reconfigurable computing system, e.g., there are only two registers in REMUS [7]. Then the input loop nest are static control parts (SCoP), where the loop bounds, if conditions and array subscripts are made of affine expressions involving only outer loop iterations, integer constants and integer literals. Next, we just consider the mapping of innermost two loops of multi-level loop nests. Then, the optimization is applied to perfectly nested loops. For imperfectly nested loops, the embedding approach proposed in [11] could convert the imperfectly nested loops into perfectly nested loops. At last, the length of all the data dependence are constants. The majority of daily life computation-intensive applications satisfy these assumptions.

Our problem is defined as: given the C programs satisfy the basic assumption described above, find the pipelined stage hyperplane Θ optimized for both communication cost and PE utilization rate. Then find another hyperplane Π (independent of the first one) which get a trade-off result in

over all consideration of communication cost and configuration cost.

4.1 Find the Hyperplane Θ for Unified Optimization of PE Utilization Rate and Communication Cost

Based on the theory of the polyhedra model in the previous subsection, we proposed an algorithm to find two hyperplanes to split the loop nest and map the split tiles to the RCAs in an RPU. As analyzed in Sect. 3, the configuration cost is sensitive to the angle of the two hyperplane that partition the original loop nest. So we firstly focus on the determination of the first hyperplane as a basis reference. In consideration the pass registers in the RCA are limited in number, we firstly determine the hyperplane Θ corresponding to the pipelined stage in the RCA for the unified optimization of PE utilization rate and communication cost.

We first give the constraints of good transformations. Considering the route style of pipelined PE rows in RCAs, all the data dependence should traverse at least one hyperplane of the Θ hyperplane set. So the pipelined stage constraint is defined as:

$$\Theta_{S_i}(\vec{i}_t) - \Theta_{S_j}(\vec{i}_s) \geq 1, <\vec{i}_s, \vec{i}_t> \in \mathcal{P}_e \tag{5}$$

As the h-transformation h_e could convert the iteration of source node into the iteration of target node, the pipelined stage constraint could also be represented as:

$$\Theta_{S_i}(\vec{i}_t) - \Theta_{S_j}(h_e(\vec{i}_t)) \geq 1, \vec{i}^t \in \mathcal{P}_e \tag{6}$$

With this constraints, the characterization of the pipelined PE rows that the operation in upper rows of RCA should be executed before the operations in lower rows is satisfied. Therefor, dependent operations could be executed without change their dependent relations, which is of most important to guarantee the correctness of a transformation.

In order to deduce the optimal target, we define the same cost function in [9]:

$$\delta_e(\vec{i}_t) = \Theta_{S_i}(\vec{i}_t) - \Theta_{S_j}(h_e(\vec{i}_t)), \vec{i}_t \in \mathcal{P}_e \tag{7}$$

$\delta_e(\vec{i}_t)$ has the same mathematic meaning as the work in [9] that the number of hyperplanes the dependence e traverse along the hyperplane in vertical direction. However, it has its own physical meaning in pipelined reconfigurable computing system, where $\delta_e(\vec{i}_t)$ indicates the pass register used. Further more, it also indicates the number of PE "wasted" in the RCA.

We also use the bonding function approach to restrict the cost function. Since the loop variables themselves are bounded by affine function of the global parameters, an affine form in the program global parameter $\vec{g}_S$ that bounds $\delta_e(\vec{i}_t)$ for every dependence edge e could be found. The expression of the bound function could be presented as $v(\vec{g}) = \vec{u} \cdot \vec{g} + w$, which could be simplified as w for constant data dependence. Such that

$$\Theta_{S_i}(\vec{i}^t) - \Theta_{S_j}(h_e(\vec{i}_t)) \leq v(\vec{g}_S), \vec{i}_t) \in \mathcal{P}_e$$

i.e.,
$$w - \delta_e(\vec{i}_t) \geq 0, \vec{i}^t \in \mathcal{P}_e \tag{8}$$

Then our optimization target is to find lexicographic minimal solution with w in the leading position and other transformation variables. Since the structural parameters are quite large, we first want to minimize their coefficients. So the optimization target finding the first hyperplane is:

$$minimize_{\prec}\ (w, c_1, c_2) \tag{9}$$

Where c_1, c_2 are the coefficients of the 1-D affine transformation Θ.

From the discussion above, we can summarize our formulation finding the pipelined stage hyperplane Θ as Eq. (10). Equation (10a) is responsible for the all dependence analysis of the original loop nest. Equation (10b) is the cost function of the transformation. Equation (10c) is the constraint for pipelined execution mode in RCAs. And Eq. (10d) is the affine bound of the cost function. Therefor, the first problem of finding the pipelined stage hyperplane is formulated.

$$Minimize_{\prec}(w, c_1, c_2)$$

$$Subject\ to \left(\begin{array}{c} \mathcal{D}_s \\ \hline \mathcal{D}_t \\ \hline h_e \end{array}\right)\left(\begin{array}{c} \vec{i}_s \\ \vec{i}_t \\ \vec{g} \\ 1 \end{array}\right)\left(\begin{array}{c|c} \geq & \vec{0} \\ \geq & \vec{0} \\ \hline = & \vec{0} \end{array}\right) \tag{10a}$$

$$\delta_e(\vec{i}_t) = \Theta_{S_i}(\vec{i}_t) - \Theta_{S_j}(h_e(\vec{i}_t)) \tag{10b}$$

$$\delta_e(\vec{i}_t) \geq 1, \vec{i}_t \in \mathcal{P}_e \tag{10c}$$

$$w - \delta_e(\vec{i}_t) \geq 0, \vec{i}_t \in \mathcal{P}_e \tag{10d}$$

4.2 Find the Hyperplane Π for Trade-Off Optimization of Communication Cost and Configuration Cost

After the determination of the first hyperplane as the basis reference, we now could further find the hyperplane Π in consideration the angle with the first one for the trade-off optimization of the communication cost and configuring cost.

First, we also need to give the good transformation constraint for the space hyperplane. As analyzed in Sect. 3, data dependence with big horizontal component has low route cost in RCAs because horizontal routes are implemented by MUXs without pass registers. So the transformation constraint for hyperplane Π is:

$$\Pi_{S_i}(\vec{i}_t) - \Pi_{S_j}(\vec{i}_s) \geq 0, <\vec{i}_s, \vec{i}_t> \in \mathcal{P}_e \tag{11}$$

Then, the place hyperplane Π must be independent of the first hyperplane Θ, which could be presented as:

$$c_1 d_2 - c_2 d_1 > 0 \ or \quad c_2 d_1 - c_1 d_2 > 0 \tag{12}$$

Where, c_1, c_2 indicate the coefficients of the Θ transformation found in the previous subsection and d_1, d_2 indicates the coefficients of the Π transformation.

Next, we will give constraint in consideration of configuration cost. As analyzed in Sect. 3, the configuration

cost is close related to the angle of the two hyperplanes selected to partition the original loop nest. In particular, the angle of the two hyperplane is closer to 90°, the less configuration times are needed to change the data path of RCAs. So the configuration constraint is presented as:

$$\left| \frac{c_1 d_1 + c_2 d_2}{\sqrt{{c_1}^2 + {c_2}^2}\sqrt{{d_1}^2 + {d_2}^2}} \right| < \epsilon, 0 < \epsilon < 1 \tag{13}$$

Where ϵ is an experiment parameter to control the gap between the actual angle and 90° and the left part of "<" is actually the cosine function of the angle between vector (c_1, c_2) and (d_1, d_2).

From the discussion above, we summarize the problem formulation finding the place hyperplane Π as Eq. (10). Equation (14a) is responsible for the iteration domain and data dependence analysis. Equation (14b) and Eq. (14e) have same meanings in Eq. (10). Equation (14d) is a independent constraint and Eq. (14f) is a configuration constraints. As a result, the whole problems are formulated now.

$$Minimize_{\prec}(w, d_1, d_2)$$

$$Subject\ to \left(\begin{array}{c} \mathcal{D}_s \\ \hline \mathcal{D}_t \\ \hline h_e \end{array} \right) \left(\begin{array}{c} \vec{i}_s \\ \vec{i}_t \\ \vec{g} \\ 1 \end{array} \right) \left(\begin{array}{c|c} \geq & \vec{0} \\ \geq & \vec{0} \\ \hline = & \vec{0} \end{array} \right) \tag{14a}$$

$$\delta_e(\vec{i}_t) = \Pi_{S_i}(\vec{i}_t) - \Pi_{S_j}(h_e(\vec{i}_t)) \tag{14b}$$

$$\delta_e(\vec{i}_t) \geq 0, \vec{i}_t \in \mathcal{P}_e \tag{14c}$$

$$c_1 d_2 - c_2 d_1 > 0 \ \ or \ \ c_2 d_1 - c_1 d_2 > 0 \tag{14d}$$

$$w - \delta_e(\vec{i}_t) \geq 0, \vec{i}_t \in \mathcal{P}_e \tag{14e}$$

$$\left| \frac{c_1 d_1 + c_2 d_2}{\sqrt{{c_1}^2 + {c_2}^2}\sqrt{{d_1}^2 + {d_2}^2}} \right| < \epsilon, 0 < \epsilon < 1 \tag{14f}$$

5. Efficient Solution

As discussed in the previous section, there are mainly two formulated problems of transform and mapping 2-D loop nests on to the RCA of pipelined reconfigurable computing system: *Problem* **1** is the selection of pipeline stage hyperplane Θ optimized for communication cost and PE utilization rate in a unified way, which is also a basis reference of the second hyperplane. *Problem* **2** is the selection of space hyperplane Π, where the communication cost is optimized under a specified angle range with the first basis reference hyperplane. In this section, the solution of the two problems are addressed, respectively.

Observe the Eq. (10) carefully, we find that all the constraints (i.e., (10a), (10b), (10c) and (10d)) are linear inequalities and the optimal target is to find a lexicographic minimum. Fortunately, the solution of this problem is within the power of Parametric Integer Programming [12]. As the work in [9], *Problem* **1** can be handled by PIP software easily.

Now, we move on to the other problem. Observes discovery in Eq. (14), every constraints are linear except that Eq. (14f) is a nonlinear constraint which is not easy to be transferred or removed. Actually, Eq. (14f) is a quadratic inequality of two unknowns, d_1 and d_2, since c_1 and c_2 has already been worked out in *Problem* **1**. Therefor, this inequality is only related to loop transformation. Actually, the loop level associated with data dependence is relative small, and the coefficients of the loop transformation is also relative small. The *Problem* **2** could be converted into a integer linear programming problem if we enumerate the some of the transformation coefficients in the polyhedra formed by all the linear inequalities.

We proposed a pseudo-optimum based stepwise expansion searching method to solve the *Problem* **2**. Instead of a brute-force search approach, we search the result in the stepwise expansion area of the optimal solution to speed up the search, where the initial the optimal solution is the optimal result without consideration of the nonlinear inequality (14f).

Our pseudo-optimum based stepwise expansion searching method includes three steps: 1). Using the PIP software, we first figure out the initial optimal result $(w^*, {d_1}^*, {d_2}^*)$ of *Problem* **2** without the nonlinear inequality (14f); 2). Give a stepwise expansion area that by adding a incremental k to all the components of the initial optimal result; 3). Search the optimal solution of *Problem* **2** in the area from the initial optimal solution $(w^*, {d_1}^*, {d_2}^*)$ to $(w^* + k, {d_1}^* + k, {d_2}^* + k)$ in lexicographic order. Once a solution satisfies all constraints in Eq. (14), we stop search and this solution is the optimal result. If there still no legal solution when the search goes through all the points in the area determined in 2), we add a incremental k to all the components of the finish points and repeat 2) in the rest area. The detail of the pseudo-optimum based stepwise expansion searching algorithm is presented as follow:

Algorithm 1 Pseudo-Optimum Based Stepwise Expansion Searching for *Problem* **2**.

```
1: find all dependence P_0, P_2, ..., P_n by dependence polyhedra;
2: figure out the initial optimal solution (w*, d1*, d2*) using PIP
3: repeat
4:   determine search local area A from (w*, d1*, d2*) to
     (w* + k, d1* + k, d2* + k)
5:   repeat
6:     if IsSearched(p) is false then
7:       feasibleA ← p is in the area A
8:       feasibleB ← IsSatisfyEqn14(p)
9:     IsSearched(p) ← true
10:    end if
11:    p ← p increase in lexicographic order in area A
12:  until feasibleA is false or feasibleB is true
13:  if feasibleA is false then
14:    k ← k+k
15:  end if
16:until feasibleA is true and feasibleB is true
```

Obviously, our result is the lexicographic minimum of the Problem **2** because the initial optimal result is obtained on relaxed constraints and our search index increase step by

step in lexicographic order. Generally, the data dependence length and transformation coefficient are relative small, so the 3-dimensional search space formed by (w, d_1, d_2) is also relative small.

6. Experiment Result

To verify and evaluate the proposed optimization approach, we conduct some experiments on REMUS processor [7]. As shown in Fig. 5, REMUS is a reconfigurable multi-media processor with an 8x8 RCA in an RPU clocked at 200 MHz, tape-out in SMIC 65 nm, where only two pass registers are available for each PE in RCAs.

Our loop transformation algorithm is first performed as source to source processing step to partition the original 2-D loop nest into small tiles matching the RCA size. Then, every tile is converted into data flow graph (DFG) and mapped on RCAs. Our design flow takes loop kernels in high level language, such as *C*, as input, and extract the polyhedra intermediate representation (IR) from source code by *Clan* [13] and perform dependence analysis by the Chunky Analyzer for Dependences in Loops (CAnDL) as the work [14]. Then the tile size is determined by operator decomposition of the statement in loops and the size of RCA. Finally, we convert the tiles into DFG and generate the configuration context running on REMUS [7].

Our test example include a set of real-life compute-intensive loop kernels: 1-D jacobi, Motion Estimation (ME) and stencil computations in partial differential equation (PDE). The proposed joint loop transformation in consideration communication cost, PE utilization rate and configuration cost (Joint PE+COM+CFG) is compared with two reference points. The first reference point is the loop unrolling based optimization scheme [2], where all the loops are unrolled and converted into DFGs. Therefor, the regularity of original code is disarranged and optimization is performed on the generated DFG. The second reference point is a work only perform unified optimization of PE utilization rate and communication cost (Combined PE+COM) [5], equal to find two independent communication minimal hyperplanes.

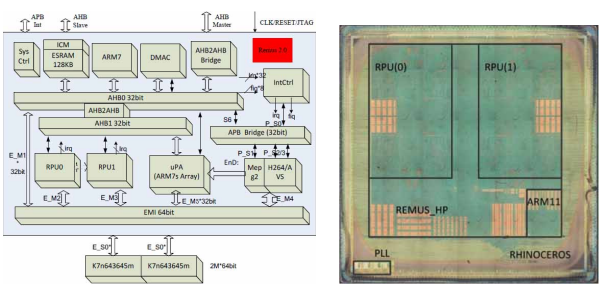
6.1 Performance Evaluation

The experiments of the three different approaches are demonstrated in Table 2. The first column lists the three different approaches. The notation t_{com},t_{cfg}, t_{op} and t_{total} indicate the data communication cycles, configuration cycles, RCA operation cycles and the total running cycles of a whole tasks, respectively, where $t_{total} = t_{com} + t_{cfg} + t_{op}$. The notation *PI* is the performance improvement of our proposed approach compared with the other approaches. Finally, the notation r_{pe} indicates the average utilization rate of PEs.

Here, t_{com} corresponds to the sum of external communication of a task in Sect. 3. Lower r_{pe} indicates that more times of RCA operations are needed, and more RCA operation time (t_{op}) is further needed. Thus, t_{op} involves the PE utilization rate r_{pe}, and further involves internal communications. In addition, t_{total} is the sum of t_{com}, t_{op} and t_{cfg}, so the joint influence of loop communications (external and internal communication), PE utilization rate and configuration cost is presented in t_{total}.

From the results, it is clear that the three factors (i.e., PE utilization rate, communication cost and configuration cost) have important influence on the performance of reconfigurable computing system. Take r_{pe} and t_{op} into consideration, we find that the higher r_{pe} is the less t_{op} is. Take the PDE solvers in Table 2 for example, $r_{pe} = 61.2\%$ in Combined PE+COM approach is the highest with its operation cycles $t_{op} = 1.1e3$, while the $r_{pe} = 40.2\%$ in Loop unrolling approach is the lowest with its operation cycles $t_{op} = 1.5e3$. For a given kernels, the total number of operations is fixed, so the higher the PE utilization rate is the less number of RCA operation times is. However, utilization rate is only one of total performance influencing factors. As a result, the Combined PE+COM approach with the highest r_{pe} in the PDE solvers don't get the best total performance of the three approach.

Compared with the combined PE+COM scheme, the kernels of 1-D JACOBI have almost the same operation cycles. The reason is that both our proposed approach and the



(a) Architecture (b) Die photograph

Fig. 5 REMUS: Reconfigurable multimedia processor [7].

Table 2 The performance comparison of joint PE+COM+CFG, combined PE+COM and loop unrolling based scheme.

Examples		1-D JACOBI	ME	PDE Solvers
Joint PE+COM+CFG	r_{pe}	77.2%	58.3%	49.3%
	$t_{com}(cycles)$	1.1e2	2.8e2	2.7e2
	$t_{op}(cycles)$	1.4e2	1.8e3	1.2e3
	$t_{cfg}(cycles)$	7.5e1	1.1e3	4.5e2
	$t_{total}(cycles)$	3.3e2	3.2e3	1.9e3
Combined PE+COM [5]	r_{pe}	77.2%	61.2%	52.6%
	$t_{com}(cycles)$	1.1e2	2.1e2	1.2e2
	$t_{op}(cycles)$	1.4e2	1.7e3	1.1e3
	$t_{cfg}(cycles)$	7.5e1	1.5e3	9.3e2
	$t_{total}(cycles)$	3.3e2	3.4e3	2.2e3
	PI	0%	5.8%	13.6%
Loop unrolling scheme [2]	r_{pe}	52.4%	53.0%	40.2%
	$t_{com}(cycles)$	1.8e2	6.1e2	4.1e2
	$t_{op}(cycles)$	1.7e2	1.9e3	1.5e3
	$t_{cfg}(cycles)$	1.1e2	1.8e3	1.1e3
	$t_{total}(cycles)$	4.6e2	4.3e3	3.0e3
	PI	28.3%	25.6%	36.7%

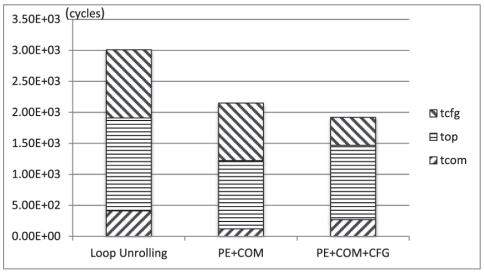


Fig. 6 The time component parts of the three approaches in PDE.

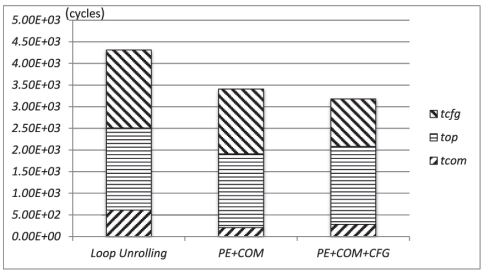


Fig. 7 The time component parts of the three approaches in ME.

combined PE+COM approach got the same transformation coefficients for the features of data dependence. However, our approach performs better than the loop unrolling based approaches [2] in all the three example cases, where the execution performance of 1-d JACOBI, ME and PDE solvers are improved by 28.3%, 25.6% and 36.7%, respectively.

Subsequently, we focus on the performance of our proposed approach and combined PE+COM approach [5] on the ME and PDE solver kernels, where the performance of our proposed approach is better than that of combined PE+COM [5] approach. For clearer comparison, we transfer the data in Table 2 into histogram if Fig. 6 and Fig. 7. As show in Figs. 6 and 7, our proposed approach is better than the loop unrolling scheme in every time components because loops are totally unrolled in the loop unrolling scheme and little optimization opportunity could be achieved in the disordered large DFG. In the other comparison, the communication cycles, operation cycles and PE utilization in our proposed approach don't perform better than that of the COM+PE method because less communication and PE utilization optimization opportunity could be found with the C-R angle constraint in Eq. (13). However, the reconfiguration cycles in our approach is lower than that of the COM scheme as less isomorphic tiles would arise with the cosine constraint in Eq. (13). As a result, the overall performance in our proposed approach is improved by 5.8% and 13.6% on average in the ME and PDE solver kernel, respectively.

Make a step further, we move on to the different size of the same kernel in Table 3 and we find that the amount of performance increase (13.6%) of small PDE kernels is bigger than that (11.3%) of bigger PDE kernels. So is the case with ME. Generally, the proportion of isomeric tiles of small kernels is bigger than that of big kernels after they are tiled by a same loop transformation. Therefor, the reduction of configuration cost in small kernels is of paramount importance and the upper hand of our proposed approach shows up.

Table 3 The impact of kernel size on performance.

Kernel of different size	Joint PE+COM+CFG	Combined PE+COM [5]	Improvement
PDE Solvers (N = 64)	1.9e3	2.2e3	13.6%
PDE Solvers (N = 128)	3.9e3	4.4e3	11.3%
ME (N = 64)	3.2e3	3.4e3	5.8%
ME (N = 128)	6.6e3	6.9e3	4.3%

Now REMUS [7] has been applied to set-top box (STB) and performs well in decoding of multimedia. Many application kernels (e.g., ME mentioned above) have been optimized by our proposed approach and it leads good results.

6.2 Run-Time Complexity

In our proposed algorithm, all the constraints in *Problem***1**, such as Eqs. (10a) (10b) (10c) (10d), are linear inequalities or equalities. And the optimization target of Eq. (10) is the lexicographic minimum of the unknowns in the inequalities. Thus, the optimal integer solution for the affine transformation coefficients could be solved by a parameter integer programming problem. As Paul Feautrier said in [12], although the theoretical complexity of solving a parameter integer programming (PIP) is quite high, it has a high probability of being polynomial. In practice, we have found the complexity of the algorithm to be commensurate to the complexity of the solution.

*Problem***2** could be actually divided into two subproblems: 1). Finding the initial pseudo-optimal solution when without the nonlinear inequalities Eq. (14d), which has the same complexity as *Problem***1**. 2). Searching the optimum a 3-dimensional space based on the pseudo-optimum. The complexity of this stepwise expansion searching is relative low because the boundary of the affine transformation coefficients is limited (-20 to 20 in most cases). To sum up the two subproblems, the complexity of *Problem***2** is almost equal to *Problem***1**.

In our algorithms, the complexity is sensitive to the number of dependence references of loop. To evaluate the effectiveness of our algorithms, we run more tests for more loop nests (mm, adi, jaocbi and gemm) with different number of dependence references. The compilation of the loop optimization is taken on an Intel Dual-Core CPU machine running at 1.9 GHz with 2 GB memory. Table 4 gives the maximal run-time of the test cases. In practice, the run-time is less than two seconds and the complexity of our algorithms is acceptable.

Table 4 Execution time on some cases.

test cases	mm	adi	jacobi	gemm
dependence references	1	2	3	4
running time (seconds)	0.23	0.51	0.97	1.35

7. Conclusion

PE utilization rate, communication cost and reconfiguration cost are the three of the most important performance influencing factors of reconfigurable computing system. In this work, we present a coalescent loop transformation algorithm to take into account the three factors as a whole to obtain two RCA corresponding hyperplanes. A pseudo-optimum based stepwise expansion search technique is proposed to speed up the execution of algorithm. At last, the effectiveness of our proposed algorithm is demonstrated with a few loop nest examples.

References

[1] J. Lee, K. Choi, and N. Dutt, "An algorithm for mapping loops onto coarse-grained reconfigurable architectures," ACM Sigplan Notices, pp.183–188, 2003.
[2] O. Dragomir and K. Bertels, "Extending loop unrolling and shifting for reconfigurable architectures," Architectures and Compilers for Embedded Systems (ACES), pp.61–64, Edegem, Belgium, 2010.
[3] S. Yin, C. Yin, L. Liu, M. Zhu, and S. Wei, "Configuration context reduction for coarse-grained reconfigurable architecture," IEICE Trans. Inf. & Syst., vol.E95-D, no.2, pp.335–344, Feb. 2012.
[4] Y. Lam, J. Coutinho, C. Ho, P. Leong, and W. Luk, "Multiloop parallelisation using unrolling and fission," Int. J. Reconfigurable Computing, vol.2010, p.1, 2010.
[5] D. Liu, S. Yin, C. Yin, L. Liu, and S. Wei, "Mapping optimization of affine loop nests for reconfigurable computing architecture," IEICE Trans. Inf. & Syst., vol.E95-D, no.12, pp.2898–2907, Dec. 2012.
[6] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical and fully automatic polyhedral program optimization system," ACM SIGPLAN PLDI, 2008.
[7] M. Zhu, L. Liu, S. Yin, Y. Wang, W. Wang, and S. Wei, "A reconfigurable multi-processor soc for media applications," Circuits and Systems (ISCAS), Proc. 2010 IEEE International Symposium on, pp.2011–2014, 2010.
[8] S.C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R.R. Taylor, "PipeRench: A reconfigurable architecture and compiler," Computer, vol.33, no.4, pp.70–77, 2000.
[9] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Affine transformations for communication minimal parallelization and locality optimization of arbitrarily nested loop sequences," Tech. Rep., Citeseer, 2007.
[10] P. Feautrier, "Some efficient solutions to the affine scheduling problem. part ii. multidimensional time," Int. J. Parallel Programming, vol.21, no.6, pp.389–420, 1992.
[11] N. Ahmed, N. Mateev, and K. Pingali, "Synthesizing transformations for locality enhancement of imperfectly-nested loop nests," Int. J. Parallel Programming, vol.29, no.5, pp.493–544, 2001.
[12] P. Feautrier, "Parametric integer programming," RAIRO Recherche opérationnelle, vol.22, no.3, pp.243–268, 1988.
[13] C. Bastoul, A. Cohen, S. Girbal, S. Sharma, and O. Temam, "Putting polyhedral loop transformations to work," Languages and Compilers for Parallel Computing, pp.209–225, 2004.
[14] D. Khaldi, C. Ancourt, and F. Irigoin, "Towards automatic c programs optimization and parallelization using the pips-pocc integration." http://www.cri.ensmp.fr/classement/doc/A-448.pdf

Shouyi Yin received the B.S., M.S. and Ph.D. degree in Electronic Engineering from Tsinghua University, China, in 2000, 2002 and 2005 respectively. He has worked in Imperial College London as a research associate. Currently, he is with Institute of Microelectronics at Tsinghua University as an associate professor. His research interests include mobile computing, wireless communications and SoC design.

Dajiang Liu received the B.S. degree from the School of Microelectronics and Solid-state Electronics in University of Electronic Science and Technology of China, Chengdu, China, in 2009. Currently he is working toward the Ph.D. degree in the Institute of Microelectronics, Tsinghua University, Beijing, China. His research interests include reconfigurable computing and optimization of compiler for reconfigurable computing.

Leibo Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999 and the Ph.D. degree in Institute of Microelectronics, Tsinghua University, in 2004. He now serves as an associate professor in Institute of Microelectronics, Tsinghua University. His research interests include reconfigurable computing, mobile computing and VLSI DSP.

Shaojun Wei was born in Beijing, China in 1958. He received Ph.D. degree from Faculte Polytechnique de Mons, Belguim, in 1991. He became a professor in Institute of Microelectronics of Tsinghua University in 1995. He is a senior member of Chinese Institute of Electronics (CIE). His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design.