IEICE TRANS. INF. & SYST., VOL.E95–D, NO.12 DECEMBER 2012

PAPER Special Section on Parallel and Distributed Computing and Networking

Mapping Optimization of Affine Loop Nests for Reconfigurable Computing Architecture*

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

SUMMARY Reconfigurable computing system is a class of parallel architecture with the ability of computing in hardware to increase performance, while remaining much of flexibility of a software solution. This architecture is particularly suitable for running regular and compute-intensive tasks, nevertheless, most compute-intensive tasks spend most of their running time in nested loops. Polyhedron model is a powerful tool to give a reasonable transformation on such nested loops. In this paper, a number of issues are addressed towards the goal of optimization of affine loop nests for reconfigurable cell array (*RCA*), such as approach to make the most use of processing elements (*PE*) while minimizing the communication volume by loop transformation in polyhedron model, determination of tilling form by the intra-statement dependence analysis and determination of tilling size by the tilling form and the RCA size. Experimental results on a number of kernels demonstrate the effectiveness of the mapping optimization approaches developed. Compared with DFG-based optimization approach, the execution performances of 1-d jacobi and matrix multiplication are improved by 28% and 48.47%. Lastly, the run-time complexity is acceptable for the practical cases.

key words: *reconfigurable computing, affine loop, polyhedron model, parallel computing*

1. Introduction

Reconfigurable computing is a computer architecture combing some of the flexibility of software with the high performance of hardware. During the past decade a large number of reconfigurable computing architecture have been developed by the research community, which demonstrated the high performance for a selected set of applications. For example, Morphsys [1] is implemented by a tiny RISC and 8x8 processing array which can be 6.5 times faster than Splash2 on the automatic target recognition processing; Garp [2] is a fine grain 1-D reconfigurable architecture which is 9 times faster than UltraSPARC on image dithering;Rapid is a 1-D pipeline coarse grain processor which performs very close to its peak of 1.6 GOPS on 2-D DCT; and REMUS [3], a reconfigurable multi-processor SoC for media application consisting of 512 processing engines and two ARMs, is also been implemented to accelerate the decoding of H.264 high-profile streams. With the presence of reconfigurable processor unit (RPU) in these architecture, high performance of computation on some compute-intensive applications is achieved.

Reconfigurable computing system are typically based on reconfigurable processing units (RPUs) acting as coprocessor units and coupled a host processor. To exploit the computation power of the hybrid architecture, efficient reconfigurable compilers are badly needed to leverage the synergies of the reconfigurable architectures. Programming reconfigurable architecture is a well-known challenge work, as the traditional sequential programming pattern is no longer suitable for this architecture. Generally, partitioning the computation between the host system and RPU is a key aspect in the middle-end compilation of reconfigurable architecture. As a result, the original computations are partitioned into a software component and a reconfigware component. The software component is compiled with traditional compilers. Then the hardware component will be mapped to RPUs with special representations. In the middle-end of compilers, architecture-driven transformations (e.g., loop transformations) is very important to exploit the parallelism of code.

For the compiling for the hybrid architecture, several compilers have been proposed, ranging from a software assisting in manual circuit creation to an automatic high level programming language compiler. The NAPA C [4] compiler is a low-level circuit generator and *pragma*-based directives are provided by programmer manually to specify where data is to reside and where computation is to occur. Thus, this low-level compiler limits the widespread use for reconfigurable computing system. In SA-C [5], a set of data-parallel semantics are defined to hide the reconfigurable hardware details, however, the performance still relies on the programmer's capability and runtime configuration executable file could not be generated. High level compilers, e.g., Garp C compiler [6], NEC electronics' Musketeer [7] and Template-based compiler [8] are also proposed recently. In all these compilers, partitioning the computations between the processor and an RPU (hardware part) is a necessity of the compiling flow. After the partition, a large data flow graph (DFG) is generated from the hardware part. Then the large DFG is partitioned into small sub-graph with its size corresponding to the reconfigurable cell array (RCA) size where optimization for communication and synchronization is under consideration. However, the level of optimization based on the large DFG is not so high because

Manuscript received December 26, 2011.
Manuscript revised April 28, 2012.

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

*This work was supported by NNSF of China grant 60803018, the National High-Technology Research and Development Program (863 Program) of China grant 2009AA011702 and the International S&T Cooperation Project of China grant 2012DFA11170

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

DOI: 10.1587/transinf.E95.D.2898

that the optimization opportunity in the step of DFG generation is of little thought. Not only that, the regular source code of front-end is disorganized. As a result, less opportunity could be embraced from the DFG-based optimization, e.g., communication volume, utilization rate of processing elements (PE).

To get more optimization opportunity for the compiling of reconfigurable computing system, our attention is turned to the original source code. In the source code of compute-intensive programs, nested loop costs most of the running time. A mathematic model, the polyhedron model, offers a powerful abstraction to give a unified framework for dependence analysis and transformation of loop with affine bounds and array access functions (regular code). A dynamic instance of each statements in a loop nest is presented as an integer point in a well-defined space which is the statement's polyhedron. Using linear integer programming, an optimal loop transformation under some constrictions could be worked out to exploit the computation power of parallel architecture. Tiling is a key transformations in optimizing for data locality, parallelism, communication volume and synchronization. Tiling for data locality optimization requires grouping points in an iteration space into smaller blocks to maximize data reuse. Tiling for parallelism and synchronization is mainly used to partition the iteration space into small tiles that maybe concurrently executed on different processors with reduced communication volume and synchronization times. Hence, we focus on the key aspects of loop transformation, tilling for reconfigurable cell array (RCA).

This paper focuses on RCAs in reconfigurable processor unit (RPU) and addresses the optimization of mapping affine loop nests on a two dimensional (2-D) RCA. The proposed optimization approaches include: finding two proper hyperplanes for a two dimensional RCA for the target of making best use of PEs and minimizing the communication volume between RCA, determining the tile form (length/width ratio) and determining proper tile size to fill up an RCA. The proposed approaches make most use of PEs in an RCA and improve the PE utilization rate of RCA. In addition, communication volume between different operation instances of the same RCA or different RCAs is minimized to reduce the memory access time.

The rest of this paper is organized as follows. In Sect. 2, we briefly describe the current architecture for reconfigurable computing platforms. In Sect. 3, we analysis the key factors that affect the performance of RCA. In the next sections, the optimization for the tiling shape is discussed and in Sect. 5 the tiling form and size are determined. Then, Sect. 6 gives experiments results that demonstrate the effectiveness of our optimization works. At last, we conclude in Sect. 7.

2. Overview of Reconfigurable Computing Architecture

2.1 General Architecture of Pipelined Coarse-Grained Reconfigurable Computing System

There are two main types of reconfigurable computing hardware architecture according to granularity. In this paper, we distinguish efforts on fine-grained RPUs, for example, field-programmable gate array (FPGAs), from the ones targeting coarse-grained RPUs and we focus on the coarse-grained reconfigurable computing architecture. The general architecture of pipelined coarse-grained reconfigurable computing is typically based on reconfigurable processing units (RPUs) as roles of coprocessor and coupled to a host processor, where the host processor controls the execution of the whole system. Each RPU consists of one or more RCAs, a data memory, a configuration memory, a control unit, etc. The configuration memory stores the configuration context and control unit controls the reconfiguration and execution of RCAs. The data memory provides operand data to RCA through high bandwidth data bus and the RCA runs in a pipelined way.

Since our proposed optimization approach is based on the general reconfigurable computing architecture described above, it can be apply to most of the pipelined coarse-grained reconfigurable computing with two-dimensional RCAs.

2.2 REMUS

REMUS [3], a reconfigurable multi-media system, consists of one RISC processor, two RPUs and some assistant modules, which is depicted in Fig. 1. An ARM11, a typical embedded RISC, is chosen as the host system for application control and reconfigurable schedule. An RPU is a powerful dynamic reconfigurable system consisting of four RCAs and each RCA has 8x8 two dimensional (2-d) PEs. Algorithms

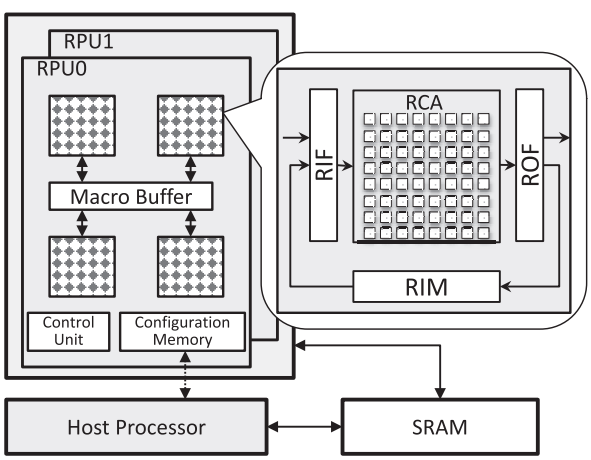

Fig. 1 The brief architecture of REMUS.

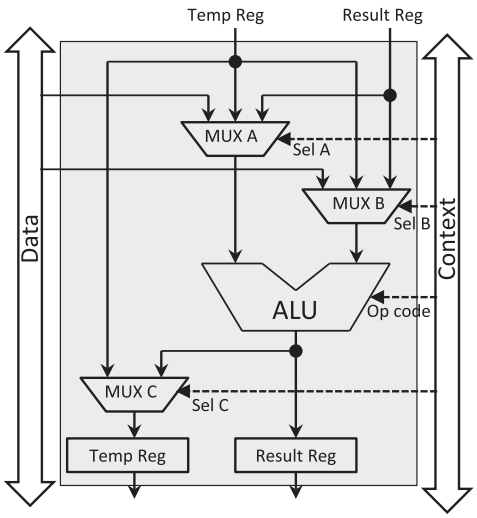

Fig. 2 RC and Route.

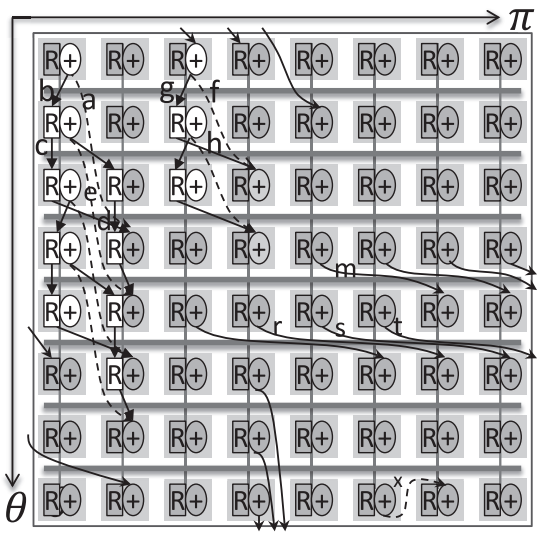

Fig. 3 The legal routing of the PE in RCA.

can be mapped into an RPU simultaneously and be executed independently to achieve high performance. An RCA Input FIFO (RIF) and an RCA Outpur FIFO (ROF) are directly connected to RCA for data buffer and a RIM is designed for the data exchange of different running instances of the same RCA. With a Macro Buffer (MB), different RCAs could exchange data with each other. The SRAM is an off-chip memory taking charge of the data exchange between host processor and RPUs.

RCA8x8 is the basic unit in an RPU, defined as the minimal function block,which has three parts: a DBI, a context interface and a 2-dimension PE array. A DBI is a flexible data exchange unit to prepare data from internal or external memory for RCA. A context interface mainly reads context from configuration memory to change the data path of RCA to support kinds of applications. RCA8x8 is a coarse-grained reconfigurable computing array of 8/16 bits, 64 cells and routes consists of a reconfigurable network, where the reconfigurable cells (equivalent to PE in this paper) and routes that can be re-functioned and restructured by the context interface. As shown in Fig. 2, a common ALU architecture with arithmetic operations is adopted. A temp register is also designed to retain data in a operation step for further process. With the input select MUXA and MUXB, the ALU could choose data from input data FIFO, temp registers of any PE in the previous rows and the operation result of any PE in the previous rows and thus the routes could transmit data between two nearby rows, point-to-point.

3. Performance Characterization of Reconfigurable Computing

In this section, the architecture of RPU will be analyzed in detail to characterize key factors that affect the performance of RPU and then the implications for compiler optimization be discussed. Before further discussion, we identify that the phrase "row" and "column" indicate the line parallel to the axis π in Fig. 3 and the line parallel to θ respectively. θ and π are two axis of two-dimensional Cartesian coordinate system indicating the location of PEs in an RCA.

3.1 The Routing Style in RCAs

The RCA is an important component in an reconfigurable computing system. It consists of many 1-dimensional or 2-dimensional reconfigurable array. A PE in the RCA is mainly composed of an ALU and a local register and it could communicate with other PEs in the nearby rows.

From the inherent character of the PEs and Routes, several mapping constraints could be figured out. To expatiate this, an RCA is depicted in Fig. 3. A combination of the framed R and circled + indicates an PE while the framed R and circled + mean a temp register and an ALU respectively. In addition, the horizontal bars between two nearby rows of PE means the routes. The arrows in Fig. 3 demonstrates some data dependence between PEs when the routes are once configured. The solid arrow and dotted arrow indicate a valid data dependence and an invalid dependence respectively. From the hardware features of the RCA, we can find the flowing constraints of data dependence on the RCA.

A. A dependence must have positive θ axis part. As the rows in a RCA corresponds to the control step of time when operators mapped on to the RCA. Thus, the operators in the upper rows should be executed before the operators in the lower row.
B. Data dependence from the same rows is forbidden, in another word, the vertical part (the part along θ axis) of dependence could not be zero. In the direction of π axis, operations in the same row are executed concurrently and operation sets are executed row by row along the θ axis. So operations mapped to the same row are belong to the same control step in data path and could not depend on each other. For example, the dependence

$\vec{x}$ in the last row in Fig. 3 is prohibitory.

C. A long dependence traversing more than one row in an RCA means more cost of temp register, subsequently, lower utilization rate of PEs. As the 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 step, it will be passed by temp 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 temp register will be used. Once all the temp register in the row in occupied, some ALU will be wasted because of lacking of temp register to pass temporary value. As a result, ALUs would be wasted in RCA because of long dependence along the axis θ. The utilization rate of PEs in an RCA is a crucial factor affecting the performance of an RPU because low utilization rate of PEs means more number of RCA operation times. However, free dependence along θ is also not recommended because free dependence along θ cause the result output of internal PEs and more overhead of memory access. Thus, those dependence that traversing one row is highly recommended for the consideration of PE utilization rate. For example, in Fig. 3, there is a data dependence $\vec{a}$ traversing three columns, the dependence will be transformed into dependence $\vec{b}$, $\vec{c}$ and $\vec{d}$ with two temp register transferring temporary value. The same objection applies to the dependence $\vec{e}$. We presume that a program is full of dependence like this. In this case , two temp registers in the same row are needed to transfer temporary data because of the overlap of dependence in the direction of axis θ. Thus, the ratio of temp register number and ALU number is $2:1$, and half of the ALU in RCA will be wasted.

D. A long dependence traversing more than one columns have very small data transfer cost in an RCA. As there is a MUX for every PE to select any data from the previous column, the long dependence traversing more than one columns in RCA almost has no transfer cost except for the delay on MUX. For example, the dependence $\vec{m}$, $\vec{r}$, $\vec{s}$ and $\vec{t}$ in Fig. 3 are all valid and have low transfer cost.

The constraints of dependence above in RCA exposes important optimization opportunity in compiling for reconfigurable computing system, especially for loop transformation stage of compute-intensive programs.

3.2 The Latency of Memory Access

In this architecture, although it takes only fixed cycles to accomplish an RCA operation for 8 rows of operators, the memory access latency time due to input and output of RCA is not allowed to be neglected. The latency is generally proportional to the number of input and output data of an RCA. Let M_{in}, M_{out}, P, R, be the number of input data, number of output data, total number of operators of a program and utilization rate of RCA respectively. Then the total running cycles in rough estimation of the program could be given by:

$$T = ((\lceil M_{in} + M_{out})/\alpha \rceil + \beta) \cdot \lceil P/(\gamma \cdot R) \rceil \quad (1)$$

where the α is an experience parameter (generally 6 for REMUS) indicating the bandwidth of input and output data considering the influence of all the memory. β indicates the operate cycles for an RCA operation (generally 8 for REMUS [3]). γ is the total number (64 for REMUS [3]) of PEs in an RCA. We also presume that the data path structure and the input&output number are the same between different RCA operations, which is reasonable for the tiled big affine loop nests because of the similarity of tiles. In this estimation model, we could find a synthetical influence of PE utilization rate and communication cost.

Having identified the key factors that influence the performance of RCA, we now discuss the compile-time optimization to address these issues. We mainly focus on affine loop nests since loop nests taking up most of run time in compute-intensive applications.

4. Optimization of Tile Shape

In this section, an approach is proposed and developed to optimize the compilation for performing program (i.e., affine loop nests) transformations that enable make good use of PEs of RCA while minimizing the communication volume of RCA. This approach is based on the polyhedron model, a powerful algebraic framework for presenting programs and transformations. Regular affine loop nest that loop bounds are affine functions of outer loop indices and global parameters are the objective our research since it takes up most of the running time in compute-intensive programs.

4.1 The Background and Notation of Polyhedron Model

This subsection gives a quick overview of polyhedron model, and the vocabulary and notations will be interpreted. The bold font letter indicates a vector.

The polyhedron model is convenient alternative representation which combines analysis power, expressiveness and high flexibility. It is based on three main concepts: the iteration domain, the scattering and the access function. A program part that can be represented using the polyhedron model is called a *Static Control Part* or *SCoP* for short.

A statement S within a *SCoP*, surrounded by d_S loops is represented by an d_S-dimensional polytope refered to an iteration space of S, where the d_S is depth of the loop nests. The coordinates of a point (*iteration vector* $\mathbf{i}_S$) in the polytope corresponds to the values of the loop indices of the surrounding loops, starting from the outermost to the innermost. A point in the polytope corresponds to an instance of statement S in program execution. The iteration space is defined by a system of affine inequalities, $\mathcal{D}_S(\mathbf{i}_S) \geq \mathbf{0}$, derived from the bounds of loops surrounding S. Using matrix to present the inequalities, the iteration space polytope is presented as:

$$\mathcal{D}_S \cdot \begin{pmatrix} \mathbf{i}_S \\ \mathbf{g}_S \\ 1 \end{pmatrix} \geq \mathbf{0} \tag{2}$$

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

The scattering function gives the execution order of the instances in loop nests. A very useful example of multi-dimensional scattering functions is the *scheduling of the original program*. The idea is to build an abstract syntax tree (AST) [9] of the program and to read the scheduling for each statement.

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

$$F_{kAS}(\mathbf{i}_S) = \mathcal{F}_{kAS} \cdot \begin{pmatrix} \mathbf{i}_S \\ \mathbf{g}_S \\ 1 \end{pmatrix} \tag{3}$$

Where $\mathcal{F}_{kAS}$ is a matrix representing an affine mapping from 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.

An array is said to have an order of magnitude reuse if the rank of the access matrix of the array reference is less than the iteration space dimensionality of the statement in which it is accessed. Thus, the condition for magnitude reuse of an array A due to a reference $\mathcal{F}_{kAS}(\mathbf{i}_S)$ is: $rank(\mathcal{F}_{kAS}) < dim(\mathbf{i}_S)$. Loops whose iterations do not occur in the affine access function of a reference are said to be *redundant loops* for the reference.

Polyhedron compiling usually involves three steps: first input programs should be represented into polyhedron formalism, then apply a transformation to this representation, and finally generate the target code.

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

$$\Phi(\mathbf{i}_S) = \mathcal{T}_S \cdot \begin{pmatrix} \mathbf{i}_S \\ \mathbf{g}_S \\ 1 \end{pmatrix} \tag{4}$$

Where $\mathcal{T}$ is a row vector, the affine transformation is a one-dimensional mapping. $\Phi(\mathbf{i}_S)$ 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 which case $\mathcal{T}$ is matrix with n rows. In the architecture of RCAs, we use Π_S to denote the space tiling (to PEs in a row of RCA) and Θ_S to denote the time tiling (to PEs belonging to different control step in the data path of RCA).

A lot of work has been done to analyze the dependence in polyhedron. The dependence model here is the same as the one used in [10], [11] and [12]. The Polyhedron Dependence Graph (PDG) is used in most polyhedron compiling work, which 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 polyhedron dependence from a dynamic instance of S_i to one of S_j. It is characterized by a polyhedron, $\mathcal{P}_e$, called the *dependence polyhedron* that captures the exact dependence information corresponding to e.

In our work, we mainly focus on finding two good hyperplane (Π and Θ) of a affine loop nests to satisfy the constraint conditions of RCA in reconfigurable computing system. The hyperplane Θ is normal to axis θ in Fig. 3 and indicates the row of an RCA. The hyperplane Π is normal to axis π in Fig. 3 and indicate the column of an RCA.

4.2 Finding Good Θ Hyperplane

Based on the theory of polyhedron model in the previous subsection, we propose an algorithm to find two hyperplanes to split the affine loop nests and map the split tile to the RCAs of RPU. Considering the constraints of data dependence in RCA in Sect. 3, the 1st and 2nd are the most rigid one because the row by row control step means the operator in lower row must be executed after the operator in the upper row. Thus, the hyperplane Θ normal to the θ axis is of a prior consideration in the hyperplane finding process.

As the work in [12], we should first give the constraints of good transformations. Let there be a dependence from statement instance $\mathbf{s}$ of S_i to $\mathbf{t}$ of S_j corresponding to an edge e of PDG of a program. After exact dependence analysis, we get

$$\mathbf{s} = h_e(\mathbf{t}), \mathbf{t} \in \mathcal{P}_e \tag{5}$$

Where $\mathcal{P}_e$, described in [13], indicates the dependence polyhedron that capture the exact dependence information corresponding to e and h_e here represents the h-transformation (affine transformations mapping the iteration vector $\mathbf{t}$ of target to source $\mathbf{s}$).

As analyzed in Sect. 3, the operations are executed row by row corresponding to the time control step. So we first need to find a hyperplane Θ that all the dependence traversing at least one hyperplane. So the time control step constraint is defined as :

$$\Theta_{S_j}(\mathbf{t}) - \Theta_{S_i}(\mathbf{s}) \geq 1, \mathbf{t} \in \mathcal{P}_e \tag{6}$$

With this condition, the characterization of RCA that the operation in upper rows should be executed before that in lower rows could be satisfied. In another word, the dependent operation could be executed without the change of their dependent relations, which is of most important to guarantee the correctness of a program.

In order to deduce an optimal target, we define the same cost function as it in [12]:

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

The affine form $\delta_e(\mathbf{t})$ means the number of hyperplanes

the dependence e traverses along the hyperplane normal in mathematic which is the same as the work in [13], however, it has a totally different physical meaning in reconfigurable computing system (i.e. REMUS here). $\delta_e(\mathbf{t})$ also indicates the number of temp registers used. Further more, it also indicates the number PEs have been "wasted" in the RCA. For example, in the case of mapping a loop nest with only one operator in its body statement to RCAs with one temp register per PE, $\delta_e(\mathbf{t})$ with the value of one means there is no temp register is used and no PE "wasted". $\delta_e(\mathbf{t})$ with the value of two means that one temp register will be used to transmit temporary value and no PE will be "wasted". $\delta_e(\mathbf{t})$ with the value of three means that two temp register will be used to transmit temporary value and one PE nearby will be "wasted" because of lacking of temp register.

As the work in [12], we also use the bounding function approach to find a good tile shape out of several possibilities. Since the loop variables themselves are bounded by affine functions of the parameters, an affine form in the program parameters, $\mathbf{p}$, that bounds $\delta_e(\mathbf{t})$ for every dependence edge e could be always found. i.e., there exists $v(\mathbf{p}) = \mathbf{u} \cdot \mathbf{p} + w$, such that

$$\begin{aligned} &\Theta_{S_j}(\mathbf{t}) - \Theta_{S_i}(h_e(\mathbf{t})) \leq v(\mathbf{p}), \mathbf{t} \in P_e, \forall e \in E \\ &i.e., \qquad v(\mathbf{p}) - \delta_e(\mathbf{t}) \geq 0, \mathbf{t} \in P_e, \forall e \in E \end{aligned} \tag{8}$$

Now Farkas Lemma can be applied to Eq. (8).

$$v(\mathbf{p}) - \delta_e(\mathbf{t}) \equiv \lambda_{e0} + \sum_{k=1}^{m_e} \lambda_{ek} P_e^k, \lambda_{ek}^T \geq \mathbf{0} \tag{9}$$

Where P_e^k is the face of P_e. The above is an identity and the coefficients of each of the loop indices in $\mathbf{t}$ and parameters in p on the left and right hang side can be gathered and equated, to get linear equalities entirely in coefficients of affine mappings for all statements, $\mathbf{u}$ and w. Using integer linear programming system, the above inequalities can be at once be solved by finding lexicographic minimal solution with $\mathbf{u}$ and w in the leading position and other variables:

$$minimize_{\prec} \ (u_1, u_2, \ldots, u_k, w, \ldots, c_i^{'}s, \ldots) \tag{10}$$

Using parameter integer programming (PIP) [14] software, the lexicographic minimal solution could be found easily under the time step constraint condition in (6). The solution gives a optimal Θ hyperplane for along the direction of θ axis in Fig. 3.

4.3 Finding Good Π Hyperplane

After the determination of the time step hyperplane Θ, now move on to find a space hyperplane Π, independent of the hyperplane Θ, and normal to π axis under the constraint condition analyzed in Sect. 3. At first, the constraint conditions is different from (6) as different hardware constraint. The new constraint condition is given by:

$$\Pi_{S_j}(\mathbf{t}) - \Pi_{S_i}(h_e(\mathbf{t})) \geq 0 \tag{11}$$

With this constraint condition, the target loop in the transformed iteration space can be blocked rectangularly as all dependence have positive components along that hyperplane.

In addition, this hyperplane Π must be independent of the first hyperplane Θ_S, the sub-space orthogonal to Θ_S is given by:

$$\Pi = I - \Theta_S^T(\Theta_S \Theta_S^T)^{-1}\Theta_S \tag{12}$$

This new constraints make sure of independence with the found hyperplane solution Θ. $\Pi \cdot \Theta \geq 1$ or $\Pi \cdot \Theta \leq -1$ gives the necessary constraint to be added for statement S to make sure that Π has a non-zero component in the sub-space orthogonal to Θ. With new constraints (11)(12), we run PIP software again to find the new solution corresponding to the hyperplane Π that the dependence length at the fringe of a tile has been minimized. Thus, the communication volume between RCA and memory(e.g., RIM, MB and SRAM) has been optimized.

4.4 Complexity of Hyperplane Determination Algorithms

In the Algorithms of finding the two independent hyperplanes, all the constraints, such as Eqs. (6)(8)(9)(11)(12), are linear inequalities or equalities. And the optimization target Eq. (10) is the lexicographic minimum of the unknowns in the inequalities. Therefor, we can find the optimal integer solution for the transformation coefficients by solving a parameter integer programming problem [15]. Although the theoretical complexity of solving a parameter integer programming [14] is quite high, in practice, we have found it to share the well known property of the simplex, which while exponential in the worst case, has a high probability of being polynomial.

5. Optimization of Tile Form and Size

Having fixed the shape of the tile mapping to an RCA, now we go on to get the form (i.e., the length/width ratio) and size of the tile.

As shown in Fig. 1, the RCA in REMUS [3] has 8x8 square PEs with each of them representing an operator in programs. If a statement has only one operator, an instance of the statement could be mapped to a PE in RCA directly. Actually, in most of the case, their are more than one operator in a statement, e.g., their are two operator in statement $S1$ in Fig. 4. Through analyzing the operator data dependence of intra-statement, we obtain the length/width ζ_S/η_S (a rectangular of PEs with length of ζ_S and width of η_S) of loop statement.

In Fig. 4, the two operator should be executed in different control step (i.e. different rows nearby of PE in RCA). Thus, an instance of statement S would occupy 2×1 PEs (two for adders). As the shape of RCA is square, the ration (L/W)of tile should be inversely proportional to the ratio (L/W) of the statement. In Fig. 4, the tile form is 1×2 as the ration of $S1$ is 2×1. Make a step forward, the length

```
for(i=1;i<N;i++)
    for(j=2;j<N-1;j++)
        a[i][j]=a[i-1][j]+a[i-1][j-1]
               +a[i-1][j+1];  \\S1
```

Fig. 4 An 2-dimensional loop nests example.

and width of the tile could be calculated. Let the length (total control step) of RCA be $\mathcal{L}_{RCA}$ and the width of RCA is $\mathcal{W}_{RCA}$, we obtain the tile sizes:

$$\begin{aligned}\mathcal{L}_{tile} &= \mathcal{L}_{RCA}/\zeta_S \\ \mathcal{W}_{tile} &= \mathcal{W}_{RCA}/\eta_S\end{aligned} \tag{13}$$

Where $\mathcal{L}_{tile}$ (round down value) is the length of the tile and $\mathcal{W}_{tile}$ (round down value) indicates the width of the *tile*. In Fig. 4, $\mathcal{L}_{tile} = 4$ and $\mathcal{W}_{tile} = 8$. Obviously, the algorithm of optimization of tile form and size is quite simple and is within polynomial time.

6. Experimental Results

To verify and evaluate the proposed optimization approach, we conduct our experiments on REMUS [3]. REMUS is a reconfigurable multi-media processor with four RCA8x8 in an RPU cloked at 200 MHz, tape-out in SMIC 130 *nm*. To understand the proposed algorithm better, every step of the algorithm is demonstrated through the example of 1-d Jacobi. Then a serial of experiments are carried out to evaluate the performance of the polyhedral optimized mapping algorithm.

6.1 Case Study

We take a perfectly nested version of 1-d Jacobi in Fig. 4 as example to demonstrate the optimization flow. The original loop nest is depicted in Fig. 5. Dependence analysis produces the h-transformation and dependence polyhedron:

$$\begin{aligned} flow: a[i'][j'] &\longrightarrow a[i-1][j] \\ h: i'=i-1, j'=j;\ D_1: 1 \le i \le N-1, 2 \le j \le N-2 & \\ flow: a[i'][j'] &\longrightarrow a[i-1][j-1] \\ h: i'=i-1, j'=j-1;\ D_2: 1 \le i \le N-1, 2 \le j \le N-2 & \\ flow: a[i'][j'] &\longrightarrow a[i-1][j+1] \\ h: i'=i-1, j'=j+1;\ D_3: 1 \le i \le N-1, 2 \le j \le N-2 & \end{aligned}$$

A. *Finding the* Θ *hyperplane for time control step.*
Dependence 1: Tiling constraint Θ:

$$c_i i + c_j j - c_i(i-1) - c_j j \ge 1 \ \Rightarrow c_i \ge 1 \tag{14}$$

The bonding constraint is $w - c_i \ge 0$ here because it is a constant dependence.
Dependence 2: Tiling constraint:

$$c_i i + c_j j - c_i(i-1) - c_j(j-1) \ge 1 \ \Rightarrow c_i + c_j \ge 1 \tag{15}$$

The bonding constraint is $w - c_i - c_j \ge 0$.

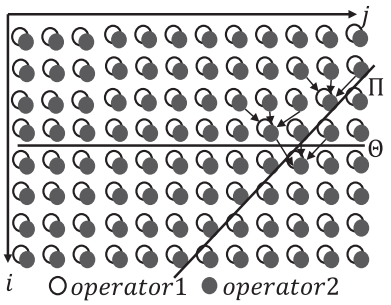

Fig. 5 The original loop nests of 1-D jacobi.

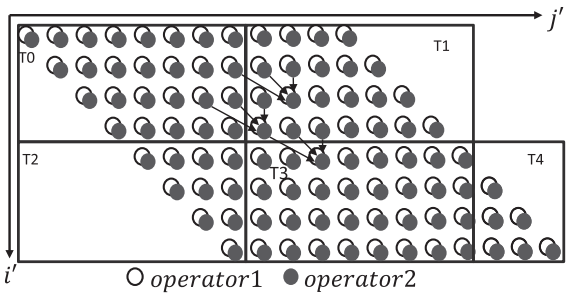

Fig. 6 The transformed loop nests of 1-D jacobi.

Dependence 3: Tiling constraint:

$$c_i i + c_j j - c_i(i-1) - c_j(j+1) \ge 1 \ \Rightarrow c_i - c_j \ge 1 \tag{16}$$

The bonding constraint is $w - c_i + c_j \ge 0$. Collecting all the constraints and optimization target together, we obtain:

$$\begin{aligned} & c_i \ge 1 \\ & w - c_i \ge 0 \\ & c_i + c_j \ge 1 \\ & w - c_i - c_j \ge 0 \\ & c_i - c_j \ge 1 \\ & w - c_i + c_j \ge 0 \\ & minimize \prec (w, c_i, c_j) \end{aligned} \tag{17}$$

The lexicographic minimal solution for vector $(w, c_i, c_j) = (1, 1, 0)$. Thus, we obtain $c_i = 1, c_j = 0$

B. *Finding the* Π *hyperplane.* The next solution for hyperplane Π must meet the constraint in 11 and 12 (independent constraint). As the flow of finding hyperplane Θ, we get the lexicographic minimal solution for hyperplane Π: $c_i' = 1, c_j' = 1$.

C. *Determining the tile form and size.* Through intra-statement dependence analysis, we find that the statement $S1$ has two operators at different control step which implies a temp register is needed to pass the temporary value from *operator*1 to *operator*2. *operator*1 and *operator*2 indicate the first addition operator and the second addition operator in the statement $S1$ in

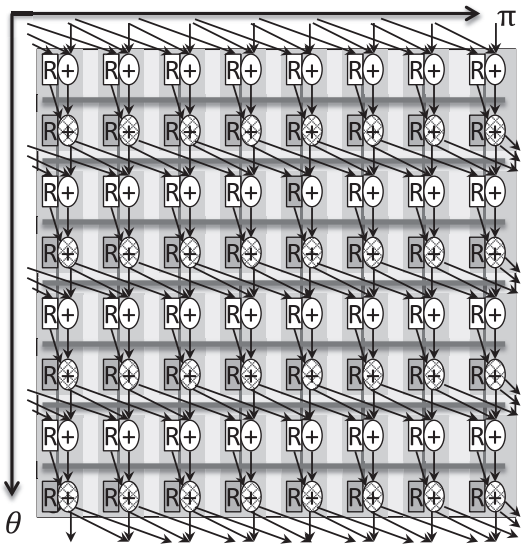

Fig. 7 The mapping of the tiled loop nests.

Fig. 4, respectively. Thus, a PEs 2×1 are occupied for an instance of statement $S1$. Using the tile size determining algorithm described in Sect. 5, the length ($\mathcal{L}_{tile}$) and width ($\mathcal{W}_{tile}$) of a tile are 4 and 8 respectively. The tiled loop nest is depicted in Fig. 6.

D. *Mapping a tile to an RCA*. After the step 1, 2 and 3, the shape and size of the tile is determined. Now the tile could be mapped onto the RCA easily since we don't beak the regularity of loop in the transformation. For example, in Fig. 6 , only the tiles at the fringe of the tiled loop nest (e.g., T0, T1, T2 and T4) are not regular and the internal tile (e.g., T3) is very regular. Actually, there are a lot of regular tiles when the size of a program (i.e., global parameter n here) is very large. In Fig. 7, a complete and regular tile (T3) is mapped onto RCA 8×8, where the circled "+", grid circled "+", framed R indicate the $operator1$, the $operator2$ and a temporary register, respectively.

In this tiling example, a complete optimization and mapping of affine loops on reconfigurable computing system is demonstrated. With proposed algorithm, a regular affine loop nest could be optimized for PE utilization rate maximization and communication volume minimization for REMUS [3] RCA without broken the regularity of original program and they could be easily mapped onto an RCA.

6.2 Performance Evaluation of Polyhedral Optimized Mapping Algorithm

We use several loop nests program as the test examples to evaluate the performance of our proposed optimization algorithm. At first, Some notations used in performance evaluation are described in Table 1.

The comparison of the same affine loop nests program mapped with DFG-based optimization approach [8] and our proposed front-end source code based optimization algorithm are shown in Table 2 and 3. DFG-based optimization approach is an algorithm that directly transfer the compute-intensive part of programs into a large Data Flow Graph (DFG) with loop unrolling, scalar replacement and other technologies. In this step, optimization for data dependence is almost not under consideration and the regularity of source code is gummed up. Then, from the generated large DFG, optimization opportunity is explored by hard work for DFG partition.

In the experiments, a 1-d jacobi loop nest with totally 496006 operators and a matrix multiplication with $1.99e{+}6$ operators are carried out with our optimizing approach and DFG-based optimization approach, where the sizes of the two matrix are both 100×100.

As the comparison shown in Table 3, our proposed approach can obviously improve the operation performance of REMUS on the computation of affine loop nests, where the execution performance of 1-d jacobi and matrix multiplications are improved by 28% and 48.47% respectively.

The total execution cycles are the sum of operation cycles (t_{op}), communication cycles (t_{commu}) and the reconfiguration cycles (t_{cfg}). First, we find that the t_{cfg} in our proposed approach is smaller than that of DFG-based approach despite that both approach have very small proportion in the total execution cycles (t_{total}). For DFG-based optimization mapping approach, template-based [8] technologies (tem-

Table 1 Notations for performance evaluation.

Notations	Explanation
t_{in}	represents RCA input data transfer delay cycles
t_{out}	represents RCA output data transfer delay cycles
$t_{commu} = t_{in} + t_{out}$	represents the communication cycles
r_{pe}	represents the PE utilization rate in an RCA
t_{cfg}	represents the configuration cycles
t_{op}	represents the total operation cycles of an RCA
$t_{total}=t_{cfg}+t_{op}$ $+t_{commu}$	represents the total execution cycles

Table 2 The comparison of PE utilization rate and communication cost.

Test Example	DFG-based mapping		Our proposed mapping		Improvement	
	r_{pe} (%)	t_{commu} (cycles)	r_{pe} (%)	t_{commu} (cycles)	r_{pe} (%)	t_{commu} (%)
1-d jacobi 500×500	85.2%	$5.83e4$	99.2%	$3.17e4$	16.4%	45.6%
Matrix multiplication $A_{100x100} \times B_{100x100}$	46.87%	$2.65e5$	99.7%	$1.57e5$	109.2%	40.54%

Table 3 The comparison of total execution cycles.

Mapping Approaches		1-d Jacobi 500×500	Matrix Multiplication $A_{100x100} \times B_{100x100}$
DFG-based mapping	$t_{op}(cycles)$	$7.42e4$	$5.30e5$
	$t_{cfg}(cycles)$	85	65
	$t_{commu}(cycles)$	$5.83e4$	$2.65e5$
	$t_{total}(cycles)$	$1.32e5$	$7.96e5$
Our proposed mapping	$t_{op}(cycles)$	$6.29e4$	$2.52e5$
	$t_{cfg}(cycles)$	45	10
	$t_{commu}(cycles)$	$3.17e4$	$1.58e5$
	$t_{total}(cycles)$	$9.54e4$	$4.10e5$
Improvement	$t_{total}(\%)$	28%	48.47%

Table 4 The complexity on 1-d jacobi and matrix multiplication.

Mapping Approach		1-d Jacobi 500×500 (3 dependence)	Matrix Multiplication $A_{100x100} \times B_{100x100}$ (1 dependence)
Runtime (seconds)	Our Proposed Approach	1.26	0.39
	DFG-based Approach [8]	0.89	0.18

plate extraction and template matching) are used on the generated DFG in the executable synthesis to reduce the reconfiguration cost. On the other hand, optimization opportunity is directly exploited from the front-end source code using Polyhedral Model in our proposed approach. Our approach tiles the original loop nests into structural similar tiles with its size corresponding to the size of an RCA. Thus, the data path of RCA hardly need to be changed and the reconfiguration cost is reduced. Then, the RCA operation cycles (t_{op}) in our proposed approach is also smaller than that in DFG-based optimization approach, where t_{op} is inversely proportional to PE utilization rate r_{pe}. In Table 2, we could find that the utilization rates of 1-jacobi and matrix multiplication optimized using our approach are improved by 16.4% and 109.2% compared that using DFG-based approach. In our approach, the data dependence of time control step part is minimized with time constraints. Thus, an optimal hyperplane is found to make full use of the PE's ALU function. As a result, our proposed approach has more PE utilization rate and less RCA operation cycles. At last, communication cost of 1-d jacobi and matrix multiplication is also improved by 45.6% and 40.54%, respectively. The improvement of communication cost is contributed in two aspects. One is the minimization of the length of data dependence, which is carried out by finding the direction of hyperplanes where dependence could traverse fewer of them. The other one is also the maximization of PE utilization rate. For the program with a specific number of operator, the higher the PE utilization rate is, the less RCA operation number of times the program costs. Thus, the communication cost for in whole execution process is also reduced because of less number of times of RCA operation.

For the above reasons, the execution performances of REMUS on 1-d jacobi and matrix multiplication are improved a lot with our proposed approach.

6.3 Run-Time Complexity

In our proposed algorithm, the complexity of finding hyperplane Θ and hyperplane Π is in polynomial time and the determination of tile form and size is also in polynomial time. In practice, the run-time is less than two seconds four our experiment cases. The complexity of 1-d jacobi and matrix multiplication optimized by different approaches is given in Table 4.

Table 4 shows the complexity of our approach is acceptable in in the case of 1-d Jacobi and matrix multiplication even though that the execution time is slightly longer than that of DFG based scheme [8]. In addition, we find that the runtime of optimization is closely related to the number of dependence reference because more dependence give rise to more inequality constraints in the PIP problem [15].

7. Conclusion

In this paper, a front-end source code based optimization algorithm for PE utilization rate and communication volume using Polyhedron Model for reconfigurable computing architecture is proposed. By analyzing the key factors that influence the performance of RCA operation, a performance evaluation model is deduced. Then we use polyhedron model to optimize affine loop nest for both PE utilization rate and communication volume. Next tilling form and size are determined by RCA size and intra-statement operator dependence. At last, some typical examples are carried out to demonstrate the effectiveness of our proposed algorithm within acceptable rum-time complexity .

References

[1] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves Filho, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Trans. Comput., vol.49, no.5, pp.465–481, 2000.

[2] J. Hauser and J. Wawrzynek, "Garp: A mips processor with a reconfigurable coprocessor," FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on, pp.12–21, IEEE, 1997.

[3] M. Zhu, L. Liu, S. Yin, Y. Wang, W. Wang, and S. Wei, "A reconfigurable multi-processor soc for media applications," Proc. 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp.2011–2014, IEEE, 2010.

[4] M. Gokhale and J. Stone, "Napa c: Compiling for a hybrid risc/fpga architecture," Proc. IEEE Symposium on FPGAs for Custom Computing Machines, 1998, pp.126–135, IEEE, 1998.

[5] W. Böhm, J. Hammes, B. Draper, M. Chawathe, C. Ross, R. Rinker, and W. Najjar, "Mapping a single assignment programming language to reconfigurable systems," J. Supercomputing, vol.21, no.2, pp.117–130, 2002.

[6] T. Callahan, J. Hauser, and J. Wawrzynek, "The garp architecture and c compiler," Computer, vol.33, no.4, pp.62–69, 2000.

[7] T. Toi, N. Nakamura, Y. Kato, T. Awashima, K. Wakabayashi, and L. Jing, "High-level synthesis challenges and solutions for a dynamically reconfigurable processor," Proc. 2006 IEEE/ACM international conference on Computer-aided design, pp.702–708, ACM, 2006.

[8] C. Yin, S. Yin, L. Liu, and S. Wei, "Compiler framework for reconfigurable computing architecture," IEICE Trans. Electron., vol.E92-C, no.10, pp.1284–1290, Oct. 2009.

[9] 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.

[10] A. Lim and M. Lam, "Maximizing parallelism and minimizing synchronization with affine partitions," Parallel Computing, vol.24, no.3-4, pp.445–475, 1998.

[11] A. Lim, G. Cheong, and M. Lam, "An affine partitioning algorithm to maximize parallelism and minimize communication," Proc. 13th International Conference on Supercomputing, pp.228–237, ACM, 1999.

[12] U. Bondhugula and J. Ramanujam, "Pluto: A practical and fully automatic polyhedral parallelizer and locality optimizer," 2007.

[13] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A

practical automatic polyhedral parallelizer and locality optimizer," ACM SIGPLAN Notices, vol.43, no.6, pp.101–113, 2008.
[14] P. Feautrier, "Pip/piplib, a parametric integer linear programming solver, 2006."
[15] P. Feautrier, "Parametric integer programming," RAIRO Recherche opérationnelle, vol.22, no.3, pp.243–268, 1988.

Dajiang Liu was born in 1986. He 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.

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.

Chongyong Yin was born in 1984. He received the B.S. degree from the Department of Microelectronics, Xi'dian University, Shanxi, China, in 2006. 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 its compiler design.

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.