Battery-Aware Loop Nests Mapping for CGRAs

Yu PENG†, Nonmember, Shouyi YIN†(a), Member, LeiBo LIU†, and Shaojun WEI†, Nonmembers

SUMMARY Coarse-grained Reconfigurable Architecture (CGRA) is a promising mobile computing platform that provides both high performance and high energy efficiency. In an application, loop nests are usually mapped onto CGRA for further acceleration, so optimizing the mapping is an important goal for design of CGRAs. Moreover, obviously almost all of mobile devices are powered by batteries, how to reduce energy consumption also becomes one of primary concerns in using CGRAs. This paper makes three contributions: a) Proposing an energy consumption model for CGRA; b) Formulating loop nests mapping problem to minimize the battery charge loss; c) Extract an efficient heuristic algorithm called BPMap. Experiment results on most kernels of the benchmarks and real-life applications show that our methods can improve the performance of the kernels and lower the energy consumption.

key words: reconfigurable computing, loop nests, polyhedral model, energy consumption, discharge duration

1. Introduction

Coarse-grained Reconfigurable Architecture (CGRA) is a promising mobile platform providing high performance, high flexibility, and high energy efficiency. Compared to Application Specific Integrate Circuits (ASIC), CGRA is more flexible to run different applications and has lower cost and more suitable for the increasing functionalities; and compared to Field Programmable Gate Arrays (FPGA), CGRA has more power efficiencies.

A typical architecture CGRA is constituted of a host controller and a 2-D mesh processing element array (PEA). Figure 1 shows an instance of a CGRA. The host controller controls the execution of the whole system and exchanges data with PEA through data memory. Each processing element (PE) in the PEA includes an ALU and a register file. In different products, the size of PEA is also different, which can vary from 4 PEs (2 × 2 grid) to 64 PEs (8 × 8 grid). As to the PE, it can operate different word-size operations of fixed-point numbers. In addition, each PE in CGRA supports clock gating [1] or power gating [2]. Clock gating and power gating are both popular techniques used in integrated circuit design, and are used for reducing power consumption. Clock gating saves power by adding more logic to a circuit to prune the clock tree, while power gating saves power by shutting off the current to blocks of the circuit that are not in use. Therefore, each PE in CGRA can be shut down when it is not used.

Modern portable devices are usually powered by batteries. When the battery capacity is determined, prolonging battery lifetime becomes the key to saving the energy of batteries. The battery has some nonlinear characteristics that must be considered, namely, rate capacity effect and recovery effect [4]. Rate capacity effect is referred to the relationship between discharge current and energy conversion rate. The more discharge current is, the lower energy conversion rate becomes. Recovery effect means that the battery will recover some of its lost charge when it is not working. These nonlinear characteristics can affect the distribution of currents, which can determine the energy consumption. And the energy consumption here acts as the battery charge loss.

In the operating process of an application, the loop nests are often mapped onto the CGRA because of the great quantity of computing; that is to say, the main computing part of the application is the loop nests. Thus, large intensity of computation could cause much energy consumption, while battery capacity is limited so that the battery becomes exhausted very soon. Therefore, how to map the loop nests with minimal energy consumption is one of the primary concerns for designers to prolong the battery lifetime.

When implementing an application on a CGRA, we can partition the application into tasks. Task mapping has been researched for several years, and some useful methods have been proposed. In [5], a method called EPIMap is proposed to obtain optimal mapping scheme, which uses re-computation for resource limitation and obtained optimized II for a single-level loop and is based on modulo scheduling. In [6], the proposed approach called PolyMap uses polyhedral model to do mapping optimization of nested loop, where only the total execution time of CGRA is taken as an optimization metric and regardless of energy consumption. In [7], the active area of FPGA is considered as a key

---

*a* E-mail: yinsy@tsinghua.edu.cn
DOI: 10.1587/transinf.2014RCP0003
factor which affects the battery usage and lifetime in task mapping, and an optimal schedule which uses appropriate area of FPGA is chosen. In [8], task partitioning and task scheduling are combined to achieve an optimal scheme for task mapping in CGRA, which minimizes the amount of energy consumption. However, in some of the previous works, the energy consumption of mapping is not considered, such as [5] and [6]. On the other hand, some proposed works take energy consumption into consideration but are regardless of the characteristics of loop nests, such as [7] and [8]. Without the consideration of loop nests, results of the optimization would be limited.

Hence, here we consider energy consumption and the characteristics of loop nests together, and change the distribution of working currents to achieve the optimization scheme of energy consumption, through adjusting the coefficients of loop nests mapping. To optimize loop nests mapping and evaluate the performance comprehensively, this paper makes the following three contributions: a) New battery charge loss model for CGRAs proposed. We establish an analytic total charge loss (TCL) model that takes all the performance-influenced factors of CGRA into consideration. This model performs accurately to evaluate the executed scheme for mapping loop nests. b) Formulating loop nests mapping optimization problem. Our goal is minimizing the battery loss to prolong the battery’s lifetime. This formulation is based on Rakhmatov Battery Model [4] and polyhedral model [9]. c) Extracting an efficient heuristic approach. We use the insights from the problem formulation and extract an efficient heuristic algorithm called BPMap. The BPMap approach considers different situations in loop nests mapping and achieves an optimal scheme with minimum energy consumption.

The remainder of this paper is organized as follows: in Sect. 2, we introduce the background of this paper. We give our basic idea to improve mapping performance in Sect. 3. In Sect. 4, we present the formulation of the optimization problem. In Sect. 5, we give an efficient solution of the optimization problem. We give the experimental results in Sect. 6. Finally, we conclude in Sect. 7.

2. Background

In this section, we discuss the target architecture of CGRA for our experiments, the background of the battery model used and the overview of loop nests mapping.

2.1 Target Architecture

In terms of interconnection scheme, we can categorize CGRAs into two groups, namely array-based CGRAs and row-based CGRAs. As discussed in [10], row-based architectures exhibit low area complexity, high hardware utilization and relatively small configuration words, and we can optimize the kernel mapping and the final architecture instantiation jointly with tailored datapaths. Therefore we focus on row-based CGRAs, as shown in Fig. 1. The major characteristic of row-based CGRAs is that each row of the PEA executes in a control step and all the rows execute one after the other. However, at one configuration, the loading data is all loaded at a time, and so is the storing data. Besides, because of the row-to-row execution manner, PEs in row-based CGRAs usually transfer data through first-input first-output (FIFOs). A FIFO can store large amounts of data and works as a buffer, and the memory bandwidth refers to the number of data that the working FIFOs transmit in a cycle. To avoid data conflict, an interval between loading start time and storing start time is always set in the configuration context of CGRA. Thus, only if there are several FIFOs working synchronously and there are a large amount of loading data, loading and storing can be executable in parallel, and they are executed in different FIFOs. However, to reduce the power consumed and to save the area, designers often place few FIFOs in CGRA, and only one FIFO is usually working at the same time. Besides, to deal with a large amount of data, CGRAs usually use the data memory with large capacity, which is single-port. This is because for a large-capacity data memory, the memory bandwidth cannot meet the communication speed in double-port architecture. A single-port data memory only needs one FIFO at one time. With these considerations in mind, we assume that there is only one FIFO working at the same time in this paper, and the loading and storing durations are sequential for a PEA.

An example of row-based CGRAs is REMUS [3]. REMUS is a reconfigurable media computing processor, which mainly consists of a reconfigurable processor unit (RPU) and a RPU micro-controller (RMC). The basic architecture and die photograph of REMUS are shown in Fig. 2. The RPU contains four reconfigurable architectures (RCA), and each RCA is an 8 × 8 PEA. The PEs in two adjacent rows of the PEA are connected. The RCA is connected to a pair of FIFOs through an I/O bus. One FIFO is used for loading data and the other is used for storing data. Each PE can exchange data with memories through the asymmetric FIFOs. The data memory is single-port, as its capacity is large and its bandwidth cannot meet the communication speed for two ports. At one configuration, only one FIFO is selected to work at the same time, therefore the I/O bus can’t transfer reading data and writing data at the same time.
2.2 Battery Model

In this paper, we use Rakhmatov Battery Model [4] to depict the battery behavior. This model is based on the corresponding laws of chemistry, and describes the chemical reactions inside the battery concisely and commendably as an analytical form. In [4], the experimental results showed that the average error of this model in different cases was 2%, compared to factual situations. This model needs two parameters, namely $\alpha$ and $\beta$ as the battery capacity and the nonlinear effects of the battery, respectively. A large $\beta$ means the battery behaving like an ideal model. The analytical expression of the model is shown as follows:

$$\sigma = \sum_{k=0}^{n-1} \left( I_k \Delta_p + 2I_k \sum_{m=1}^{\infty} \frac{e^{-|p|m^2(T-t_i-T_0)} - e^{-|p|^2m^2(T-t_i)}}{\beta^2m^2} \right)$$

(1)

In (1), $T$ is the battery lifetime; $\sigma$ is the charge loss of $n$ tasks; $\Delta_p$, $I_k$ and $t_k$ stands for the duration, the discharge current and start time of the $k$th distinct discharge interval respectively. We can see that energy consumption can be determined by the duration and the corresponding current (namely the distribution of currents) with fixed $\alpha$ and $\beta$.

2.3 Loop Nests Mapping

2.3.1 Mapping Modes

There are two kinds of mapping modes for loop nests, namely temporal mapping and spatial mapping. Temporal mapping refers to mapping loop nests along the direction of time axis, so that the loop nests are mapped onto CGRA step by step with the time going on. Spatial mapping refers to mapping loop nests in the scale of space, so that several loop nests can be mapped at the same time if having enough space. Compared to temporal mapping, spatial mapping has fewer number of times the CGRA does the reconfiguration, and it has lower energy consumption. Therefore we consider spatial mapping in this paper.

2.3.2 Loop Body P&R

An efficient placement and routing (P&R) method for loop body is needed, and we consider the Split-Push Kernel Mapping Approach (SPKM) in [11]. This is because SPKM allows various types of orthogonal interconnections in CGRAs, and can generate mappings which have better qualities in terms of utilized CGRA resources [11]. We use SPKM to map one of the loop bodies onto the target CGRA, then we can get the loop body size (denoted as $W_{loop} \times L_{loop}$) after the mapping is done.

2.3.3 Polyhedral Model

Polyhedral model [9] is an expressive and highly flexible analytical representation, and it is often used for 2-D transformation because of great practicability. The essential parts of a polyhedral model are shown as follows.

i) Iteration Domain. We can derive a cluster of affine inequalities from the statement $S$ surrounding the bounds of loops. Using them we can define the iteration domain: $D_i(t_i) \geq 0$. Here $D_i$ is a matrix of affine constraints about execution of statement $S$, and $t_i$ is the iteration vector of $S$.

ii) Access Function. Each reference in a statement is also an affine function of loop indices and global parameters, and it could also be described by using the matrices which represents an affine mapping form the iteration space of statement $S$ to the data space of array $S$.

iii) Dependence Polyhedrons. We can use the Polyhedral Dependence Graph (PDG) to present dependence polyhedrons. PDG is a directed multi-graph with each vertex representing a statement, and the edge $(e \in E)$ between two nodes representing a polyhedral dependence on dynamic instances of them.

iv) Affine Transformation. Our representation for schedules is similar to that in [12], where the representation for transformation is split into smaller matrix and vector satisfying strong normalization rules. Let $A^S$ be the matrix operating on iteration vectors, $\rho^S$ the static statement ordering vector. The transformation matrix $\Phi^S$ is as follow:

$$\Phi^S = \begin{bmatrix} 0 & \cdots & 0 \\ A_{1,1}^S & \cdots & A_{1,d_s}^S & \rho_{1,0}^S \\ 0 & \cdots & 0 & \rho_{1}^S \\ \vdots & \ddots & \vdots \\ A_{d_s,1}^S & \cdots & A_{d_s,d_s}^S & 0 \\ 0 & \cdots & 0 & \rho_{d_s}^S \end{bmatrix}$$

(2)

where each row of $A^S$ is a one one-dimensional affine transformation, which can be also called an affine hyperplane.

3. Basic Idea

According to the previous knowledge we have, we describe our basic ideas in this section. We first depict the factors that affect energy consumption. Then in order to handle the factors, we propose our mapping method.

3.1 The Distribution of Currents

In order to describe energy consumption, we should depict the factors that affect energy consumption first. According to (1), the distribution of currents is the key factor with fixed $\alpha$ and $\beta$.

In (1), we notice that the distribution of currents in CGRA contains several pairs of $(\Delta_i, I_i, t_i)$, which is related to the operation mode of a CGRA. Figure 3 shows a typical operation mode of a CGRA. In general cases (such as the $(p-1)$th PEA operation), when a task is mapped into a PEA of a CGRA, the CGRA needs reconfiguration before
executing the task. After that, essential data need to transfer to the PEA for further computing. Then the task is executed on the PEA with these data. Finally, the output which results from the execution needs to be stored. General speaking, each stage has its corresponding discharging durations: reconfiguration duration refers to the time passed for the re-configuration of CGRA; loading duration refers to the time that is spent to transfer data from data memory to all needed PEs which can communicate with memory; executing duration refers to the time passed for executing the PEA; storing duration is the time that is spent to transfer data from PEs to data memory.

In CGRAs, application execution can be partitioned into many PEA operations. As discussed above, a PEA operation is a task sequence consisting of reconfiguration duration, data loading duration, PEA executing duration and data storing duration, and we use $\Delta_{CFG,p}$, $\Delta_{LD,p}$, $\Delta_{EXE,p}$ and $\Delta_{ST,p}$ to denote these durations of the $p$th PEA operation respectively. Each duration has its corresponding current, therefore we denote $I_{CFG,p}$, $I_{LD,p}$, $I_{EXE,p}$ and $I_{ST,p}$ as re-configuration current, loading current, executing current and storing current of the $p$th PEA operation, respectively. However, we can see that the CGRA needs no more reconfiguration in some PEA operations when the PEA would operate the same functionality as the previous one (such as the $p$th PEA operation), as shown in Fig. 3. In this case, we can assume that the PEA operation also has reconfiguration duration such that $\Delta_{CFG,p} = 0$. When all the durations of every PEA operation are determined, we will obtain the start time of each PEA operation, denoted as $t_p$. According to [6], each duration is determined by the form of loop nests mapping, so finding a suitable mapping method is the objective.

### 3.2 Loop Mapping Method

A loop nest is usually depicted by a senior language like C++, as shown in Fig. 4 (a). Figure 4 (a) shows a two-level loop nest, and the bounds of outer and inner loops are $M$ and $N$ respectively. The loop body surrounded by the two for statements, with two computing operators in it. To deal with the loop nest mapping, it is general for researchers to use the iteration domain introduced in Sect. 2.3.3. Thus, a loop body in the iteration domain is called a iteration, and is represented by a grey node in Fig. 4.

#### 3.2.1 A General Loop Mapping Method

In order to explain our motivation, we first introduce a general loop mapping method which is illustrated in Fig. 4 (b).

First, the loop nest is unrolled and an iteration domain is derived from the bounds of loops [9], where each node denotes an iteration (loop body); next, the iteration domain is tiled into small loop tiling blocks; finally each block is mapped onto PEA in the left-to-right and top-to-bottom order. Here we use the PEA resource tile (PRT) defined in [6] to describe the mapping relationship between PEA and the iteration domain. The PRT indicates the maximal size of iterations a PEA could hold, and each PRT refers to a PEA operation of CGRA.

However, the general method above is usually not a suitable one. Figure 4 (b) illustrates one reason: a loop-carried dependence $e$ is denoted by the blue arrow; when we try to map the PRT onto the PEA, the legality of $e$ is violated since its producer executes after its consumer, so that the mapping is failed. Besides, the method ignores the energy consumption, which may lead to large energy loss and low energy-efficiency. To solve the problem, we use the Polyhedral Model [9] to optimize loop nests mapping. This is because the Polyhedral Model has great practicability due to parallelism improvement, and can modify the direction of dependences across loops to guarantee the legality of parallel execution. Besides, since the PEA in CGRA is 2-D, we can map the innermost two loops of multi-level loop nests onto it directly. Therefore, we only consider the mapping of innermost two loops in this paper. According to the discussion above, we propose the following new mapping method.

#### 3.2.2 Loop Transformation and Tiling

First of all, we treat loop transformation as affine transformation (introduced in Sect. 2.3.3), and define $\Pi$ and $\Theta$ as the row related hyperplane and the column related hyperplane respectively. The affine transformation coefficients of hyperplane $\Pi$ and $\Theta$ are denoted as $(c_1, c_2)$ and $(d_1, d_2)$ respectively. For a loop nest, the original iteration domain is a rectangle with the width $M$ (outer loop bound) and the length $N$ (inner loop bound), then it will be transformed into a new
iteration domain depicted as a parallelogram, as shown in Fig. 4 (c). When the polyhedral parameters \( \{c_1, c_2, d_1, d_2\} \) are decided, the affine transformation will be fixed.

After the iteration domain is transformed, we tile the loop nest and generate some PRTs. In Fig. 4 (c), we can see that the transformed iteration domain is covered by PRTs, and the legality of \( x \) is kept and the mapping can be done. Moreover, the form of the PRTs has changed, compared to that in Fig. 4 (b). Like the work in [6], we define Regular PRT (R-PRT) as PRT full filled with iterations, and define Irregular PRT (I-PRT) as PRT partial filled with iterations. Figure 4 (c) uses the blue tiles and the yellow tiles to represent R-PRTs and I-PRTs respectively. Thus, loop nests mapping can be depicted by the transformed iteration domain and the PRTs. With the PEA size (Fig. 5 (b)), then we partition the equivalent loop nests into two cuts (Fig. 5 (c)), and map each cut onto the CGRA respectively. In Figs. 5 (d) and 5(e), the PEA executes 4 operators per cycle for both two cuts and there is no idle PE left; besides, the dependences across the loop bodies can be mapped onto the PEA.

We get some insights from this example. Although loop partitioning brings extra communication overhead, it prevents some dependences which must be placed in memories before from accessing memory; moreover, it can improve UR and parallelism, so that the number of PEA operations and total time of executing durations decrease. The total battery charging time decreases, so that we can reduce the energy consumption. On the other hand, since loop partitioning reduce the working time, we can insert more idle time (denoted as \( T_{idle} \)) in the execution process without violating the time constraints. According to [4], we can insert \( T_{idle} \) as late as possible to achieve the maximum recovery charge. If the battery is exhausted (failure) during loop \( l \) execution (after partitioning, each cut can be treated as a loop), we insert \( T_{idle} \) immediately before the execution of loop \( l \). Thus, we obtain \( T = \Delta_{total} + T_{idle} \), where \( \Delta_{total} \) is the sum of all the discharging durations. According to recovery effect, the battery will recover some of its lost charge during the idle time, so that we can prolong battery’s lifetime.

We use the Integrated Kernel Partitioning (IKP) in [13] and modify it for our cases. IKP can separate the loop into several cuts, and we use \( S_i \) denotes the \( i^{th} \) cut. In order to improve UR, we only consider the divisors of PEA size (denoted as \( d_j(size(PEA)) \), the \( j^{th} \) divisor in ascending order) as the candidates of the sizes of the generated cuts. For instance, we consider 1, 2, 4, 8 as the candidates when the PEA size is \( 8 \times 8 \). Therefore we add size constraints of cuts in (3) to the IKP process, so that the PEA will fill up with the cuts with the same functionality and no idle rows/columns left.

\[
\sum_i \left| \prod_j \left| size(S_i) - d_j(size(PEA)) \right| \right| = 0 \tag{3}
\]

In order to guarantee the dependence legality, we partition loop nests after polyhedral parameters are given. The whole process of loop partitioning is introduced below. First, we identify all the kinds of dependences across the original loops and find the longest one which is across \( \chi \) loops in the transformed iteration domain (the parallelogram). For example, \( \chi = 2 \) in Fig. 5. Then we fuse \( \chi \times \chi \) loops in the parallelogram into a new equivalent loop (Eloop). If the Eloop size is smaller than the PEA size, we obtain a integer \( \kappa \) which satisfies \( (\kappa - 1)size(Eloop) \leq size(PEA) \leq ksize(Eloop) \), then we fuse \( \kappa \) Eloops into a new Eloop. Finally we call the modified IKP approach and treat the Eloop as the input, then we get a set of cuts.

After loop partitioning, we use the SPKM approach [11] to map the cuts in the order of the direction of dependences. When all the iterations of one cut are finished, we start mapping the next cut. For example, we first map

---

Fig. 5 An example of loop mapping: (a) direct mapping; (b) loop fusion; (c) loop partitioning; (d) the first cut mapping; (e) the second cut mapping.
Cut $S_1$ in Fig. 5, and we map cut $S_2$ when all the references of $S_1$ have been mapped. We use $UR$ as a parameter of loop partitioning, which is shown as follows:

$$UR = \frac{\text{number (Active PE in PEA)}}{W_{\text{pea}} \times L_{\text{pea}}}$$  \hspace{1cm} (4)

The whole mapping flow is shown in Fig. 6. We can see that after partitioning we treat each cut as a separate loop and do loop tiling and P&R for it. Now we build a mapping framework and find all the parameters we need in it, which are $UR$, $(c_1, c_2, d_1, d_2)$. When the parameters change, the distribution of corresponding currents also becomes different, which lead to different amount of energy consumption. Through adjusting the parameters, we can change the distribution of currents to achieve the optimization scheme of energy consumption.

4. Energy Consumption Problem Formulation

With the discussion above, we can describe the whole energy consumption problem here. We first formulate all the durations of each PEA operation and the corresponding currents; then we calculate essential parameters to obtain the start time of each PEA operation; at last, we describe the performance metric and formulate the optimization problem.

4.1 Duration Formulation

4.1.1 Reconfiguration Duration

The reconfiguration duration is determined by the hardware structure and a constant (denoted as $\Delta_{\text{CFG}}$) for a given CGRA. We add two 0-1 variables $x_p$ and $\varepsilon$ to distinguish the not-reconfigured PEA operations ($x_p = 0$) and the reconfiguration-hidden CGRAs ($\varepsilon = 0$), respectively. The reconfiguration current is the current consumed when the controller writes configuration words into PE registers and is also a constant (denoted as $I_{\text{CFG}}$). We have:

$$\Delta_{\text{CFG},p} = \varepsilon \cdot x_p \cdot \Delta_{\text{CFG}}, I_{\text{CFG},p} = I_{\text{CFG}}$$  \hspace{1cm} (5)

4.1.2 Loading Duration and Storing Duration

The loading duration and storing duration are both related to the communication between PRTs. Communication is needed when there is a dependence between two PRTs. According to [6], we treat the length of the dependence as a function of communication volume, and we define $\sigma_{\text{E}}(\Theta) = \Theta(t_1^p) - \Theta(t_2^p)$, $\sigma_{\text{R}}(\Pi) = \Pi(t_1^p) - \Pi(t_2^p)$, where $t_1^p$ and $t_2^p$ are the iteration index of the target node and source node of dependence $e$. We define $\gamma_{LD}$ and $\gamma_{ST}$ to show the difference of communication volume between R-PRT and I-PRT during loading duration and storing duration respectively. As shown in Fig. 4, the average communication volume in I-PRT is almost half of that in R-PRT, so we let $\gamma_{LD}$ and $\gamma_{ST}$ approximately equal to 0.5 for simplicity. With the memory bandwidth $BW$, we have:

$$\Delta_{\text{LD},p} = \left\{ \begin{array}{cl} \frac{\sum_{e \in E} (\sigma_{\text{E}}(\Theta) + \sigma_{\text{R}}(\Pi) - \sigma_{\text{E}}(\Theta) - \sigma_{\text{R}}(\Pi))}{BW}, & \text{R-PRT} \\ \frac{\sum_{e \in E} (\sigma_{\text{E}}(\Theta) + \sigma_{\text{R}}(\Pi) - \sigma_{\text{E}}(\Theta) - \sigma_{\text{R}}(\Pi))}{BW}, & \text{I-PRT} \end{array} \right.$$  \hspace{1cm} (6)

$$\Delta_{\text{ST},p} = \left\{ \begin{array}{cl} \frac{\sum_{e \in E} (\sigma_{\text{E}}(\Theta) + \sigma_{\text{R}}(\Pi) - \sigma_{\text{E}}(\Theta) - \sigma_{\text{R}}(\Pi))}{BW}, & \text{R-PRT} \\ \frac{\sum_{e \in E} (\sigma_{\text{E}}(\Theta) + \sigma_{\text{R}}(\Pi) - \sigma_{\text{E}}(\Theta) - \sigma_{\text{R}}(\Pi))}{BW}, & \text{I-PRT} \end{array} \right.$$  \hspace{1cm} (7)

The loading current of transferring one data from the memory to PEA is constant (denoted as $I_{\text{LD}}$), so is the storing current of transferring one data (denoted as $I_{\text{ST}}$). But the whole loading/storing current of CGRA is also related to communication volume and memory bandwidth, and we denote $CV_{\text{LD}}, CV_{\text{ST}}$ as the communication volume of loading duration and storing duration respectively. We get:

$$I_{\text{LD},p} = I_{\text{LD}} \cdot \left[ \frac{CV_{\text{LD}}}{BW} \right], I_{\text{ST},p} = I_{\text{ST}} \cdot \left[ \frac{CV_{\text{ST}}}{BW} \right]$$  \hspace{1cm} (8)

4.1.3 Execution Duration

In execution duration, the execution duration of a PEA operation is the maximum data dependence length on PEA (max $L_{op}$). $E'$ denotes the set of the dependence placed on the PEA. The executing current is determined by the number of active PEs at one time (denoted as $n_{\text{pea}},p$) and the executing operators of PEs. We calculate the average current of working PEs which are carrying one loop, and we define $\text{num}(op)$ as the number of operators in a loop, $I_{\text{PE}}(op_k)$ as the current of the operator and $t_{\text{PE}}(op_k)$ as the time for a PE executing the operator. We have:

$$I_{\text{EXE},p} = n_{\text{pea}},p \times \frac{\sum_{k=1}^{\text{num}(op)} I_{\text{PE}}(op_k) t_{\text{PE}}(op_k)}{L_{\text{loop}}}$$  \hspace{1cm} (9)

4.2 The Determination of Start Time

In order to know the start time of every duration of any PEA operation, we must know the number of PRTs and R-PRTs in every PRT row first (denoted as $n_{\text{row}}(k)$ and $n_{\text{R-PRT}}(k)$ respectively, and $n_{\text{R-PRT}}(k)/n_{\text{PRT}}(k)$ is the number of PRTs/R-PRTs in the kth row). Before that, we should obtain the number of the rows which contain R-PRT (denoted as $N_{\text{row}}$). We use the calculation equations of $N_{\text{row}}, N_{\text{PRT}}, N_{\text{R-PRT}}$ (the number of R-PRTs, total PRTs (T-PRT) and
requested values. The calculations are similar to Case iii. 

\( N_{\text{row}} \) is shown in Fig. 7 (c). Here \( k_p \) and \( d_p \) are the number of terms and the common difference for Sequence \( n_{p\text{-pr}}(k) \). \( k_p \) and \( d_p \) are the number of terms and the common difference for Sequence \( n_{p\text{-pr}}(k) \): in Line 7, \( r < \left( \frac{\text{rem}}{2} \right) \) means that there are a few rows in the middle of the iteration domain having the same maximum values respectively; in Line 10, \( \text{rem}(a, b) \) is the remainder of \( a/b \), and \( \text{rem}(N_{\text{row}}, 2) = 0 \) means that the number of rows is even.

### 4.3 Performance Metric

We use the total charge loss (TCL) as a performance metric. Since energy consumption is essentially determined by the form of loop nests mapping, TCL can be computed when all the parameters discussed above are given. Since \( T \) is also a function of \( \Delta_k \), with a group of fixed \( \alpha \) and \( \beta \), (1) can be described as: \( \sigma = \sum_{i=1}^{n} F(\Delta_k, I_k, t_k) \). If the original loop has been partitioned into several cuts, each cut can be treated as a new loop. We calculate the TCL of each cut \( S_i \) independently, and then calculate the sum \( \text{TCL}_{\text{sum}} = \sum_i \text{TCL}_i \). With all the discussion above, we could give the analytical form of performance metric in (10). Here \( p \in [1, P] \) (we assume that \( P_0 = 1 \) and \( P_1 = P \) is the index number of the PEA operation, \( f \) is the clock frequency, \( I \) is the number of cuts, \( t_{p+1} = t_p + \Delta_{\text{CFG}} + \Delta_{\text{LD},p} + \Delta_{\text{EXE},p} + \Delta_{\text{ST},p} \), and \( t_1 = 0 \).

\[
\text{TCL} = \sum_{i=1}^{I} \sum_{p=P_{i-1}+1}^{P_i} \left[ F \left( \frac{\Delta_{\text{CFG}}}{f}, I_{\text{CFG}}, t_p \right) \right] \\
+ \left[ F \left( \frac{\Delta_{\text{LD},p}}{f}, I_{\text{LD},p}, t_p + \Delta_{\text{CFG}} \right) \right] \\
+ \left[ F \left( \frac{\Delta_{\text{EXE},p}}{f}, I_{\text{EXE},p}, t_p + \Delta_{\text{CFG}} + \Delta_{\text{LD},p} \right) \right] \\
+ \left[ F \left( \frac{\Delta_{\text{ST},p}}{f}, I_{\text{ST},p}, t_p + \Delta_{\text{CFG}} + \Delta_{\text{LD},p} + \Delta_{\text{EXE},p} \right) \right]
\]

(10)

### 4.4 Problem Establishment

Some constraints are needed to formulate the problem. First, the legality of dependences in PRTs should be guaranteed, and we give the constraints of legal affine transformation according to the left-to-right and top-to-bottom execution order of PRT. Second, mapping tasks are expected to be executed rapidly or more rapidly, and time constraints are needed. We define \( T_{\text{set}} \) as the maximum time that the mapping process can last under our acceptance, and it is not larger than \( T \). The working time must be not larger than \( T_{\text{set}} \). Third, the charge loss can not be larger than the battery capacity \( \alpha \), and the constraints of charge loss need to be added. Besides, the unimodular transformation constraints
of \((c_1, c_2, d_1, d_2)\) also need to be concerned, and the constraints make sure that the transformed space (the solution space for the optimization problem) is tight. Finally, we establish the optimization problem as follows:

\[
\begin{align*}
\text{Min} & \quad T_{\text{CL}} \\
\text{s.t.} & \quad \Theta(\vec{t}_1^i) - \Theta(\vec{t}_2^i) \geq 0, \; \Pi(\vec{t}_1^i) - \Pi(\vec{t}_2^i) \geq 0, \; \forall \vec{e} \in E \\
& \quad \Delta_{\text{total}} + T_{\text{idle}} \leq T_{\text{set}} \\
& \quad T_{\text{CL}} \leq \alpha \\
& \quad |c_2d_1 - c_1d_2| = 1 \\
& \quad c_2, c_1, d_1, d_2 \in \mathbb{Z}
\end{align*}
\]

Here (11b) gives the constraints of legal affine transformation; (11c) gives the constraints of time; (11d) gives the constraints of charge loss; (11e) and (11f) give the constraints of tightness of transformed space. Based on the performance metric, we could find the optimal coefficients \((UR, c_1, c_2, d_1, d_2)\) to finish the loop transformation.

5. Efficient Solution

The calculation of \(T_{\text{CL}}\) is non-convex and an enumerate-based approach is needed. We use the features of parameters and constraints to reduce the computation complexity of finding optimal solutions. We notice that (11e) is an equation of 4 variables. Moreover, \(UR\) is determined by the form of loop partitioning, so we can obtain it by searching the solution space of loop partitioning (due to the extra constraint in (3), the complexity is \(O(1)\) and related to the number of divisors of PEA size). According to these features, we can reduce the dimensionality of the problem to 3. Thus, we adopt Genetic Algorithm [14] to solve the problem to guarantee the accuracy of the results. In order to crossover the variables, an \(N\)-bit binary \(B_N\) is used to indicate a variable, where the MSB indicates the sign bit and the rest indicates the value. Thus, a solution of the problem with 3 variables could be represented by a \(3N\) bits binary (chromosome). We use \(1/T_{\text{CL}}\) as the fitness function and the definition domain \((DD)\) of variables is formed by (11b), (11c), (11d) and (11f).

In Algorithm 2, we first initialize the population number \((M = 4)\), the maximum generation number \((G = 40)\) and the bit width \((N = 4)\). Then we generate four different solutions \(P_j^i\) in \(DD\). With each \(P_j^i\) we calculate the fitness function, and use roulette wheel scheme [14] to select the population randomly. Roulette wheel scheme gives larger chosen probability to solutions with higher fitness. Then we partition the loop to generate \(UR\) for each \(P_j^i\). Then we perform one point crossover according to the crossover probability \((P_c)\) which is set to be 0.5, and perform one point mutation according to the mutation probability \((P_m)\) which is set to be 0.03. Through repeat the above steps we finally get the optimal parameters for the mapping. The time complexity of this algorithm is \(O(G \cdot M \cdot N)\).

Based on the discussion above, we present the implementation procedure of our optimal mapping scheme in Fig. 8. First we take the loop nests which depicted by high-level programming language like C as input, and preprocess them for easy management, such as branch removal and imperfect loops conversion. Next, we analyze the polyhedral model intermediate represent (IR) with dependence by LLVM-Polly framework [15]. Next, we do the loop distribution and current distribution to establish the TCL optimization problem. Then we call the optimal coefficients algorithm. At last, CGRA contexts are generated from the new polyhedral IR.

6. Experimental Results

To demonstrate the effectiveness of the proposed mapping method, we operate some experiments on REMUS processor [3]. REMUS is a reconfigurable media computing processor, which consists of a host controller (UNITY-2 core) and a reconfigurable processor unit (RPU). The RPU contains four reconfigurable architectures (RCA), and each of RCAs is an \(8 \times 8\) PEA. The primary parameters of REMUS are summarized in Table 1.

When carrying different operators, the PE has different working currents. The actual currents of different operators operated on the PEA are also summarized in Table 2.
Table 1  Primary parameters of REMUS.

| Parameter        | Value          |
|------------------|----------------|
| Process          | 65nm CMOS      |
| Memory Bandwidth | 256 bits/cycle |
| System Clock     | 200MHz         |
| Memory Accessing | 4              |
| Interconnection  | 4              |
| RAM Accessing    | 4              |
| Configuration    | 4              |
| Shared bus       | 4              |
| Interconnection  | 4              |
| Configuration    | 4              |
| Current          | 4              |
| Current          | 4              |
| Average PE       | 4              |
| Working Current  | 4              |

Table 2  The currents of different operators.

| Operator | Current (mA) | Operator | Current (mA) |
|----------|--------------|----------|--------------|
| A – B    | 1.94         | A + B    | 1.26         |
| A x B    | 3.37         | A – B    | 1.35         |
| A>B      | 1.93         | A < B    | 1.92         |
| A=B      | 1.24         | A = B    | 1.73         |
| A!B      | 1.75         | A! B     | 1.70         |
| Comp A:B| 1.98         | A & B    | 2.06         |
| A&B      | 1.94         | Transfer A to output | 1.97 |

Table 3  Sizes of the benchmarks.

| Benchmarks | Sizes: M x N x (w) |
|------------|-------------------|
| 2mm        | 1024 x 1024 x (3) |
| symm       | 1024 x 1024 x (6) |
| jacobi-1d-imper | 9998 x 100 x (3) |
| adi        | 1024 x 1024 x (6) |
| correlation| 1000 x 1000 x (4) |
| gile       | 512 x 512 x (3)   |
| pd         | 512 x 512 x (3)   |

Table 4  Comparison of TCL at β = 0.574 (unit: mA-s).

| Case | EPIMap | PolyMap | BES | BES | Increase to EPIMap | Increase to PolyMap | Increase to BES |
|------|--------|---------|-----|-----|-------------------|-------------------|----------------|
| 2mm  | 113.627| 105.366 | 57.51 | 57.51 | 57.51              | 57.51              | 73.02          |
| symm | 237.075| 236.926 | 134.254 | 134.254 | 134.254            | 134.254            | 154.36         |
| adi  | 122.814| 94.060  | 67.284 | 67.284 | 67.284             | 67.284             | 94.060         |
| gile | 253.923| 246.823 | 136.284 | 136.284 | 136.284            | 136.284            | 154.36         |
| pd   | 513.803| 267.352 | 133.785 | 133.785 | 133.785            | 133.785            | 317.53         |

Table 5  Comparison of TCL at β = 0.273 (unit: mA-s).

| Case | EPIMap | PolyMap | BES | BES | Increase to EPIMap | Increase to PolyMap | Increase to BES |
|------|--------|---------|-----|-----|-------------------|-------------------|----------------|
| 2mm  | 113.242| 105.366 | 57.51 | 57.51 | 57.51              | 57.51              | 73.02          |
| symm | 237.075| 236.926 | 134.254 | 134.254 | 134.254            | 134.254            | 154.36         |
| adi  | 122.814| 94.060  | 67.284 | 67.284 | 67.284             | 67.284             | 94.060         |
| gile | 253.923| 246.823 | 136.284 | 136.284 | 136.284            | 136.284            | 154.36         |
| pd   | 513.803| 267.352 | 133.785 | 133.785 | 133.785            | 133.785            | 317.53         |

Table 6  Comparison of TCL at β = 0.15 (unit: mA-s).

| Case | EPIMap | PolyMap | BES | BES | Increase to EPIMap | Increase to PolyMap | Increase to BES |
|------|--------|---------|-----|-----|-------------------|-------------------|----------------|
| 2mm  | 113.242| 105.366 | 57.51 | 57.51 | 57.51              | 57.51              | 73.02          |
| symm | 237.075| 236.926 | 134.254 | 134.254 | 134.254            | 134.254            | 154.36         |
| adi  | 122.814| 94.060  | 67.284 | 67.284 | 67.284             | 67.284             | 94.060         |
| gile | 253.923| 246.823 | 136.284 | 136.284 | 136.284            | 136.284            | 154.36         |
| pd   | 513.803| 267.352 | 133.785 | 133.785 | 133.785            | 133.785            | 317.53         |

All the loop nests that we use for experiments are from PolyBench2.0 [16] and Livermore [17]. They are loop nests containing multi-level loops: 2mm is used for two matrix multiplications; symm is about matrix multiplications of symmetric matrix; jacobi-1d-imper is used for 1-D Jacobi stencil computation; gilre is a case of matrix multiplication; glre is short for general linear recurrence equation; pd is short for Planckian distribution. After preprocessing, the size of the innermost two loops in these loop nests are shown in Table 3, where M, N indicates the bound of each loop and w indicates the number of the operations in the loop body.

To evaluate the effectiveness of our approach (BPMap), we compare with three of the latest presented approaches. The first reference is EPIMap [5]. EPIMap uses recomputation for resource limitation and obtains optimized II for a single-level loop and is based on modulo scheduling. The second reference is PolyMap [6]. PolyMap uses polyhedral model to do mapping optimization of nested loop, where only the total execution time of CGRA is taken as an optimization metric and regardless of energy consumption. The third reference is the battery-efficient scheduling algorithm (BES) [7], which uses task scheduling to achieve optimal energy consumption and treats the number of active PEs as a prime factor.

To demonstrate the superiority of our method, we perform four different kinds of experiments: 1) battery charge loss experiment, which shows that our method can reduce energy consumption; 2) battery lifetime experiment, which explains our approach prolongs the battery lifetime; 3) compilation time experiment to show the feasibility of our method.

6.1 Total Charge Loss Comparison

First we do the total charge loss (TCL) comparison. We notice that the battery charge loss can't be measured directly and a simulation analysis is needed: we first translate the RTL description of REMUS into a netlist through design compiler; then we put the source program with obtained configuration contexts and the netlist together, and input them into Synopsys Verilog Compile Simulator (VCS) to generate a series of waves; then we use PrimeTime and PrimePower to evaluate the energy consumption. Mapping results are verified with cycle accurate simulator.

Here we achieve the values of TCL with Time Constraint 1 given in Table 8. Firstly we evaluate the performance with a typical value of β in [4] (β = 0.574). Then we use β = 0.273 and β = 0.15 to do the comparison. The results are shown in Tables 4, 5 and 6. The unit of charge loss is mA-s.

From Tables 4, 5 and 6, we can conclude that the bat-
tery charge loss of our method is smaller than EPIMap, PolyMap and BES, no matter what the value of $\beta$ is. On one hand, this is because we optimize the distribution of currents through alter the parameters of mapping to minimize the TCL, while EPIMap and PolyMap only consider execution time optimization; on the other hand, we consider the characteristics of loop nests, but BES ignores them. EPIMap uses temporal mapping and regardless energy consumption so that its charge loss is the greatest. Due to different sizes of iteration domains, results for test cases are different. Besides, the charge loss increases when $\beta$ is diminishing, which is due to that a small $\beta$ means the more nonlinearity of the battery. We take the nonlinear feature of battery into account, so the performance of our approach is steadily better than other methods with different $\beta$, which is about 46% than EPIMap, 26% than PolyMap and 27% than BES.

We also use some test cases at $\beta = 0.574$ to illustrate the composition of energy consumption. In order to keep the accuracy, we don’t insert any idle time here. As discussed above, a PEA operation consists of four durations, and we use CFG, LD, EXE, ST to denote reconfiguration duration, loading duration, execution duration and storing duration. We calculate the corresponding energy consumption of each duration through using control variate method, as shown in Fig. 9. Here we make five experiments, and deal with 2mm by using different approaches from Experiment 1 to 4, and deal with glre by using BPMap in Experiment 5. For explicitness and clarity, we repeat showing the TCLs of CFG, EXE and ST in Fig. 10. Figure 11 shows the loop nest of 2mm after preprocessing, and we map the innermost two loops surrounded by the red dashed-line cycle onto the PEA.

In Figs. 9 and 10, we can see that the energy consumption of communication durations (loading and storing durations) is the greatest for all the experiments. This is because that we take the energy consumption of memory access into account when formulating the mapping problem. In addition, since PolyMap and BPMap consider spatial mapping which means that several loop nests can be mapped at the same time if having enough space, they have much less energy consumption in reconfiguration duration, compared to EPIMap and BES. Moreover, EPIMap uses recomputation, which brings much more memory access request, so its energy consumption is very high in communication durations; the recomputation technique also adds extra operators, so its energy consumption is also highest in execution duration. On the other hand, the remaining three methods don’t modify the number of the operators in 2mm, which means that they have the same workloads; in addition, REMUS works on a pipelined manner, therefore all the three methods must obey the dependence legality and the operators must be executed one after another. Thus, the TCLs of EXE duration by using three methods have not changed much. Furthermore, since both the two cases, 2mm and glre, have much more communication volume in loading duration than that in storing duration when they are mapped onto the PEA, the communication volume in loading duration is beyond the bandwidth and we need several cycles to transfer all the data. For example in Fig. 11, there are 4 loading data and 1 storing data in a loop body of 2mm (when considering using registers, we also need to load the immediate value, alpha, to the PEA one time at one configuration, and the average ratio of loading to storing is still more than 3:1), and finally the ratio of loading TCL to storing TCL of 2mm in every method is all between 3:1 and 4:1 (In our method, although we use loop partitioning to cut the loop body which brings 1 more loading data and 1 more storing data, the dependence e shown in Fig. 11 can be placed onto the PEA and the overall amount of loading data and storing data is not changed much and the ratio is still like other methods). As a result, much more energy consumption is generated during their loading duration than the storing duration.

Moreover, we use pd and correlation to illustrate the performance comparison with different values of parameters at $\beta = 0.574$. Table 7 shows the results, with our final solution listed in Row 1 (for pd) and Row 5 (for correlation). We can see that the TCL of our solution is the smallest. By comparing Row 1 with Row 2&3 (or Row 5 with Row 6&7), we conclude that when $c_1$, $c_2$, $d_1$ and $d_2$ are changed, the

![Fig. 9](image)  
**Fig. 9** Energy consumption of each duration (unit: mA-s).

![Fig. 10](image)  
**Fig. 10** Energy consumption of the three durations (unit: mA-s).

![Fig. 11](image)  
**Fig. 11** The loop nest of 2mm.
communication volume (CV) is changed, and the results are different. This is because the affine transformation can alter the direction of loop-carried dependences; when the coefficients are changed, some innermost loop carried dependences may turn to second-innermost loop carried dependences, and vice versa. Thus, the communication mode will change and the CV becomes different. CV is one of the factors that affect TCL, and we can see that a small CV can lead to a small TCL (comparison of Row 1&2&3). However, TCL is also related to other factors, when CV is not changed much in some cases, the case with smallest CV is not the case with smallest TCL (comparison of Row 5&6&7). In addition, whether using loop partitioning affects the value of $UR$, and loop partitioning let us achieve much lower energy consumption (Row 1 vs. Row 4, Row 5 vs. Row 8). Loop partitioning also reduces the CV, which results from our consideration of balancing the communication overhead when doing loop partitioning. Besides, the results in Row 2&3 are also better than other methods’ results shown in Table 4. This is because our partitioning achieves better execution speed and communication relationship between PRTs, with considering the features of row-based CGRAs and batteries. The best result is from the consideration of both the proper loop transformation and loop partitioning.

On the other hand, we give the total working time (TWT) comparison in the last column of Table 7. The case with largest TCL (Row 4 for pd, Row 8 for correlation) has the largest TWT. This is because the largest TWT refers to the largest discharging time. However, smaller TWT means more parallelism and faster working speed which can cause higher power consumption and higher currents, so that the case with the smallest TWT (Row 2 for pd, Row 6 for correlation) doesn’t have the smallest TCL. This phenomenon means that we sometimes need to face the tradeoff between the fastest solution and the most energy-efficient one.

### 6.2 The Comparison of Battery Lifetime

Then we compare the battery lifetime. In order to do the accurate evaluation, we use a 600 mA-h battery. We give time constraints for test cases according to loop nests execution time (using BPMap), and repeat the execution of each test case until the battery is exhausted. Hence the total execution time is the battery lifetime. The execution time and time constraints are given in Table 8. Here Time Constraint 1 is looser than Time Constraint 2, and both of them correspond to one time of the loop nests execution. Then we use the Time Constraint 1 and the Time Constraint 2 to calculate the battery lifetime respectively. The results are shown in Figs. 12 and 13, with $\beta = 0.574$.

From Figs. 12 and 13, we can conclude that our method is better than EPIMap, PolyMap and BES and prolongs the battery lifetime. This is due to the consideration of minimizing the charge loss and exploiting the characteristics of loop nests. Since the computation intensity of each case is different, the battery lifetime is different. Besides, the time constraint can affect the battery lifetime, and a looser time constraint corresponds to a longer battery lifetime. This results from battery recovery effect: loose time constraint pro-

### Table 7 Comparison with different values of parameters.

| Case | Row | Parameters | Partition? | TCL (mA-s) | CV (Byte) | TWT (cycle) |
|------|-----|------------|------------|------------|----------|-------------|
| pd   | 1   | 1 1 2 1 100% | Yes | 13.835 | 5.78E+05 | 1.33E+05   |
| correlation | 2   | 1 1 1 1 100% | Yes | 14.112 | 5.93E+05 | 1.25E+05   |
|      | 3   | 1 2 1 1 100% | Yes | 13.847 | 5.32E+05 | 1.35E+05   |
|      | 4   | 1 2 1 1 1 75% | No | 20.917 | 7.72E+05 | 2.67E+05   |
|      | 5   | 1 1 1 1 2 1 100% | Yes | 76.298 | 2.88E+06 | 5.00E+05   |
|      | 6   | 1 1 1 1 1 1 100% | Yes | 76.950 | 2.84E+06 | 4.89E+05   |
|      | 7   | 1 2 1 1 1 1 100% | Yes | 76.445 | 2.36E+06 | 4.93E+05   |
|      | 8   | 1 1 1 1 2 1 50% | No | 87.159 | 3.89E+06 | 1.70E+06   |

### Table 8 The time constraints for each test case (unit: cycle).

| Case    | Execution Time for one loop nests | Time Constraint 1 | Time Constraint 2 |
|---------|-----------------------------------|-------------------|-------------------|
| 2mm     | 7.33E+05                          | 1.00E+06          | 9.10E+05          |
| symmm   | 1.61E+06                          | 3.00E+06          | 2.70E+06          |
| jacobi-1d-imper | 8.06E+05 | 1.00E+06          | 9.40E+05          |
| adi     | 1.50E+06                          | 2.00E+06          | 1.75E+06          |
| correlation | 9.99E+05 | 1.50E+06          | 1.40E+06          |
| glre    | 1.90E+05                          | 3.00E+05          | 2.30E+05          |
| pd      | 1.95E+05                          | 3.00E+05          | 2.30E+05          |
produces large idle time, and there are more chances to recovery charge of battery. Our approach also place the idle time in a more effective way, so we get the longest battery lifetime.

6.3 Compilation Time Comparison

We compare the compilation time of our method with EPIMap, PolyMap and BES. The results are shown in Fig. 14. We can see that the compilation time of our method is more than that of EPIMap, PolyMap and BES. Longer compilation time is due to the searching process of the nonlinear battery model added. Because the number of dependences is diverse, the results in different cases are also different. However, the compilation time of our method is still acceptable.

7. Conclusion

The energy consumption of CGRA is becoming a more and more important concern for designers. In this paper, we build a synthetical performance model for loop nests mapping optimization in CGRAs, and take the energy consumption into account. Furthermore, the battery nonlinear characteristics are considered. Experiments results show that our proposed method can reduce the energy consumption and prolong the battery lifetime to a great extent, with acceptable compilation time.

References

[1] Q. Wu, M. Pedram, and X. Wu, “Clock-gating and its application to low power design of sequential circuits,” IEEE Trans. Circuits Syst. I, Fundam. Theory Appl., vol.47, no.3, pp.415–420, 2000.
[2] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zubby, H. Jacobson, and P. Bose, “Microarchitectural techniques for power gating of execution units,” Proc. 2004 International Symposium on Low Power Electronics and Design, pp.32–37, 2004.
[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, 2010.
[4] D. Rakhmatov and S. Vrudhula, “Energy management for battery-powered embedded systems,” ACM Trans. Embedded Computing Systems (TECS), vol.2, no.3, pp.277–324, 2003.
[5] M. Hamzeh, A. Shrivastava, and S. Vrudhula, “Epimap: Using epimorphism to map applications on CGRAs,” Proc. 49th Annual Design Automation Conference, pp.1284–1291, 2012.
[6] D. Liu, S. Yin, L. Liu, and S. Wei, “Polyhedral model based mapping optimization of loop nests for CGRAs,” Proc. 50th Annual Design Automation Conference, p.19, 2013.
[7] J. Khan and R. Vemuri, “Energy management for battery-powered reconfigurable computing platforms,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.14, no.2, pp.135–147, 2006.
[8] S. Yin, R. Shi, L. Liu, and S. Wei, “Battery-aware task mapping for coarse-grained reconfigurable architecture,” IEICE Trans. Inf. & Syst., vol.E96-D, no.12, pp.2524–2535, Dec. 2013.
[9] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, “A practical and fully automatic polyhedral program optimization system,” ACM SIGPLAN PLDI, 2008.
[10] S. Xydis, G. Economakos, D. Soudris, and K. Pekmestzi, “High performance and area efficient flexible DSP datapath synthesis,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.19, no.3, pp.429–442, 2011.
[11] J.W. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, “A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architectures,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.17, no.11, pp.1565–1578, 2009.
[12] A. Cohen, M. Sigler, S. Girbal, O. Temam, D. Parello, and N. Vasilache, “Facilitating the search for compositions of program transformations,” Proc. 19th Annual International Conference on Supercomputing, pp.151–160, 2005.
[13] G. Ansaloni, K. Tanimura, L. Pozzi, and N. Dutt, “Integrated kernel partitioning and scheduling for coarse-grained reconfigurable arrays,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.31, no.12, pp.1803–1816, 2012.
[14] D.E. Goldberg and J.H. Holland, “Genetic algorithms and machine learning,” Machine Learning, vol.3, no.2, pp.95–99, 1988.
[15] T. Grosser, “Enabling polyhedral optimizations in LLVM,” Ph.D. thesis, University of Passau, 2011.
[16] L.N. Pouchet, “Polybench: A 3.2.”
[17] F.H. McMahon, “The livermore fortran kernels: A computer test of the numerical performance range,” tech. rep., Lawrence Livermore National Lab., CA (USA), 1986.

Yu Peng received the B.S. degree in Communication Engineering from Jilin University, China, in 2012. Currently, he is working toward the M.S. degree in the Institute of Microelectronics, Tsinghua University, China. His research interests include reconfigurable computing and performance optimization of compilers 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 reconfigurable computing, mobile computing and SoC 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, Belgium, 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.