Polyhedral Model based Mapping Optimization of Loop Nests for CGRAs

Dajiang Liu, Shouyi Yin, Leibo Liu, Shaojun Wei
Institute of Microelectronics
Tsinghua University, Beijing 100084, China
liudj09@mails.tsinghua.com, {yinsy, liulb, wsj}@tsinghua.edu.cn

ABSTRACT

The coarse-grained reconfigurable architecture (CGRA) is a promising platform that provides both high performance and high power-efficiency. The compute-intensive portions of an application (e.g. loops) are often mapped onto CGRA for acceleration. To optimize the mapping of loop nests to CGRA, this paper makes two contributions: i) Establishing a precise CGRA performance model and formulating the loop nests mapping as a nonlinear optimization problem based on polyhedral model, ii) Extracting an efficient heuristic loop transformation and mapping algorithm (PolyMAP) to improve mapping performance. Experiment results on most kernels of the PolyBench and real-life applications show that our proposed approach can improve the performance of the kernels by 21% on average, as compared to one of the best existing mapping algorithm, EPIMap. The runtime complexity of PolyMAP is also acceptable.

Categories and Subject Descriptors
C.3 [Special-purpose and application-based systems]: Real-time and embedded systems; D.3.4 [Processors]: Code generation, Compilers, Optimization

General Terms
Algorithms, Design, Performance

1. INTRODUCTION

The CGRA is becoming a promising platform that provides the potential for high performance, high energy efficiency, high flexibility and low cost. A CGRA is typically constituted of a host controller and a 2-D mesh processing element array (PEA), which is shown in Figure 1. The host controller controls the executions of the whole system. And each PE includes an ALU and a register file. The size of PEA can vary from 4 PEs arranged in a row up to 64 PEs arranged in an 8×8 grid. The functionality of PE could be configured to be different word-level operations of fixed-point numbers. The routing style of PEs also has great variety, such as connections between two PEs from neighbor rows, connections between each PE and its four or eight neighbors, buses connecting each node to other nodes in the same row or column. The degree of register file sharing also ranges from individual register files at each PE, to multiple register files each shared by a subset of PEs, to a single central register file shared by several or all PEs. On one hand these hardware characters bring CGRA with the potentials of high performance and high power-efficiency, but on the other hand the special features make the design of CGRA compiler is much more complex than a regular compiler.

Since the compute-intensive portions of an application (e.g. loops) are often mapped onto CGRA for acceleration, the most important task of CGRA compiler is automatically and efficiently mapping loop nests onto PEA. The mapping of loop nests could be partitioned into two subproblems, namely, loop transformation, and loop body placement and routing (P&R). The P&R of the loop body is to decide which PE to perform an operation and how to route data between PEs which has been well studied in the past several decades [1], while loop transformation is still the most challenging problem. To efficiently map loops on CGRAs, several loop transformation techniques have previously been proposed. Loop unrolling [2][3] is a common technique to generate a mapping scheme with greater parallelism. It unrolls a loop and transforms the unrolled loop into a data flow graph (DFG). Then the DFG is mapped onto CGRA. Modulo scheduling is another widely used technique, which improves the parallelism of CGRA by overlapping the execution of different iterations of a loop. In the modular scheduling works [4][5], initial interval (II) is usually used...
as a performance metric to guide the loop transformation. Recently, loop affine transformation is applied to CGRA. For example, in [6], loop affine transformation is used to optimize the PE utilization rate and communication cost between PEA and controller.

However, all the previous works have two major drawbacks. First, the CGRA’s architectural features are not well considered in loop transformation. The major architectural difference between CGRA and general purpose processor (GPP) is the reconfiguration mechanism of CGRA. The CGRA needs additional clock cycles to finish reconfiguration, while GPP has no such feature. As a result, the traditional performance metrics are not accurate and effective for CGRA. For example, in [4] and [5], II is used as performance metric as traditional software pipelining, which cannot reflect reconfiguration cost of CGRA. In [6], although PE utilization and communication cost are considered, the reconfiguration cost is also missed. Therefore, a new performance metric which reflects the overall CGRA’s architectural features (in other words, the performance influencing factors, such as reconfiguration, communication between controller and PEA, and 2-D parallelism) must be designed for loop transformation. Second, most previous works handle only the innermost loop of a loop nests and the parallelism is not explored adequately. For example, both loop unrolling and modulo scheduling usually deal with the innermost loop. Due to the 2-D topology of CGRA, the innermost 2-level loops should be taken into account to explore the parallelism.

Towards improving the whole performance of CGRA, this paper makes the following two important contributions:

1. **CGRA performance model establishment**: By analyzing the operation mechanism of a CGRA, we establish an analytic total execution time (TET) model that takes all the performance influencing factors of CGRA into an overall consideration based on polyhedral model[7]. Taking this TET model as an optimization target, we formulate the mapping of loop nests on CGRA as a constrained optimization problem.

2. **PolyMAP approach formation**: Based on genetic algorithm [8], we extract an efficient heuristic for the above optimization problem and further form a loop transformation and mapping approach (PolyMAP) to improve the performance of CGRA.

The rest of the paper is organized as follows: In section 2, we give the basic idea of our work and why a unified CGRA TET metric is needed. Then, section 3 describes the formulation of the optimization problem based on polyhedral model. Section 4 gives an efficient solution of the optimization problem and the whole mapping process. Then, section 5 gives the experimental results that demonstrate the effectiveness of our optimization works. At last, we conclude in section 6.

### 2. BASIC IDEA

Since the ultimate goal of loop mapping is to minimize the total execution time of the entire loop, the most rational way is to use the total execution time (TET) as performance metric to guide loop mapping. Due to the architectural features, the TET of CGRA is affected by the reconfiguration of PEA, data communication between the host controller and PEA, and computation of PEA. Moreover these factors are correlated. Therefore simply taking one or two factors (e.g., II and communication cost) as optimization targets may lead to poor mapping performance. The only possible way is to build a unified analytical model of TET.

The operation mechanism of a CGRA is shown in Figure 2, where CFC, LDC, CPC and STC denotes reconfiguration cycles, data loading cycles, PEA computing cycles and data storing cycles, respectively. In the process of executing a task, CGRA first reads configuration context from context memory to configure each PE. Then, input data is loaded from data memory for preparation of computing. Next, the PEA performs the computing of current parts of the task. Finally, the result data of this PEA operation is stored in data memory. Again and again like this, the whole task is executed, as shown in Figure 2(a). At some time, if the functionality of an PEA operation is the same as the previous one, configuration cycles are not needed in this PEA operation, as the case in Figure 2(b). Considering that most programs consist of a number of consecutive PEA operations \( p = [1, P] \) with different characteristics, CGRA performance can be defined in terms of the operating frequency \( f \), reconfiguration cycles, data communication cycles \( (CMC) \) (data loading and data storing) and PEA computing cycles as follow:

\[
TET = \frac{1}{f} \sum_{p \in P} (CFC_p + LDC_p + CPC_p + STC_p)
\]

As depicted in Figure 2(b), the CFC can be passed over in some PEA operations. Thus, we give a 0-1 variate of input \( x_p \) to distinguish the reconfigured and not reconfigured PEA operations. In addition, the reconfiguration time of PEA is a constant (denoted as \( CPC_p \)) in practice. As a result, the total configuration cycles of a task can be represented as follow:

\[
\sum_{p \in P} CFC_p = \sum_{p \in P} x_p \cdot CPC
\]

As shown in Figure 2, the communication cycles \( (CMC_p) \) in a PEA operation includes data loading cycles \( (LDC_p) \) and data storing cycles \( (STC_p) \):

\[
\sum_{p \in P} CMC_p = \sum_{p \in P} LDC_p + STC_p
\]

The computing cycle has a close relation to the PEA route style \( (RS) \) and the data dependence length \( (L^e) \) of input applications. So the computing cycles of a PEA operation could be represented as a function of \( RS \) and \( L^e \):

\[
\sum_{p \in P} CPC_p = \sum_{p \in P} Func(RS, L^e)
\]

From the above discussion, the CGRA TET could be represented by variables \( x_p \), \( LDC_p \), \( STC_p \) and \( L^e \). And all these
variables are closely related to the form of loop transformations. For example, the transformed loop body would lead to different times of reconfiguration and have different data dependence lengths compared to the original one. Moreover, the data volume that need to be transferred between loop instances would be also different. Therefore, there should be an optimal loop transformation that generates optimal variable $x_p$, $LDC_p$, $STC_p$ and $L^p$ leading to minimal TET. Accordingly, we need to formulate TET as a function of loop transformation. Then by minimizing TET, the optimal loop transformation and mapping could be achieved.

3. PROBLEM FORMULATION

To formulate TET and the optimization problem, we leverage the polyhedral model and affine transformation theory.

3.1 Polyhedral Model based Loop Transformation for CGRAs

The polyhedron model[7] provides a powerful abstraction to the transformation of loop nests by viewing every iteration of each statement as an integer point in a well-defined space called polyhedron. Then, affine transformations for optimization are performed on the such representation. The transformations finally reflect in the generated code with improved execution performance. It is feasible for Static Control Part (SCoP) of a program, where the loop bounds, if the conditions and array subscripts are made of affine expressions involving only outer loop iterations, integer constants and integer literals. The detailed information about the polyhedral model is presented in Appendix A. In this paper, we just consider the mapping of innermost two loops of multi-level loop nests. As shown in Figure 3(b), the original iteration domain of an input loop nest is a rectangle, and it would be affine transformed (i.e., finding two hyperplanes $\Theta$ and $\Pi$) into a new iteration domain depicted as a parallelogram in Figure 3(c). At last, the new iteration domain will be tiled into small tiles mapped onto PEs.

The input loop nests are divided into two parts: small loop body (the number of operators is less than the number of PEs in a PEA) and big loop body (the number of operators is more than the number of PEs in a PEA), which undergo different processes. For small loop body loops, we combine one or more loop bodies into a big body matching the size of PEA. For the big loop body loops, we would first distribute [9] them into small loops and handle them one by one. In practice, the size of PEA could be big enough (e.g., 8 × 8) to hold loop bodies in most cases. In addition, our optimization is applied to perfectly nested loops. For imperfectly nested loops, the embedding approach proposed in [10] could convert the imperfectly nested loops into perfectly nested loops.

As a result, our proposed approach could handle most cases of computation-intensive applications.

3.2 PEA Resource Tile

Let $L(M, N, W_{pb}, L_{ib})$ be the input 2-D nested loops, where $M, N$ indicate the bounds of outer and inner loop and $W_{pb}, L_{ib}$ indicate the size of the loop body, as shown in Figure 3(a). As the loop body is small, the $W_{pb} \times L_{ib}$ loop body size could be determined simply by an efficient P&R algorithm[1]. Let $W_{pea}, L_{pea}$ be the size of 2-D PEA, shown in Figure 3(c). We define a PEA Resource Tile (PRT) to indicate the maximal size of iterations a PEA could hold, as shown in Figure 3(d). The size of PRT is $\eta \times \zeta$, which could be simply obtained by

$$
\eta = \left\lfloor \frac{W_{pea}}{W_{ib}} \right\rfloor, \zeta = \left\lfloor \frac{L_{pea}}{L_{ib}} \right\rfloor
$$

Let the affine transformation coefficients of hyperplane $\Theta$ and hyperplane $\Pi$ are $(c_1, c_2)$ and $(d_1, d_2)$. In order to guarantee the transformed iteration domain is compact, we give the unimodular transformation constraints:

$$|c_1d_2 - c_2d_1| = 1
$$

Then, the transformed parallelogram (Figure 3(e)) would be covered by PRTs one by one. Every PRT on the parallelogram indicates a PEA operation of CGRA. As depicted in Figure 3(e), the slanting-line rectangle indicates a Regular PRT (R-PRT) that is full of iterations and the wave-line rectangle indicates an Irregular PRT (I-PRT) that is at the edge of the parallelogram and not full of iterations. The total number of PRT (T-PRT) is the sum of the R-PRT and I-PRT. We use $N_{rpt}, N_{ipt}, N_{rpt}, N_{ipt}$ to indicates the number of R-PRT, I-PRT, T-PRT and PRT rows, respectively. In Appendix B, we give the derivative process of the four parameters.

3.3 Determination of Optimization Target

In this subsection, we will further consummate the performance model as an optimize target with the number of PRT and the coefficients of loop transformation based on polyhedral model.

3.3.1 Configuration Cycles

After the original loop nest is transformed and tiled with PRT, we now could determine configuration cycles. As shown in Figure 3(e), the loop nest would be executed from top to bottom, from left to right one PEA operation by one PEA operation and a PEA operation needs to be reconfigured if its functionality is changed from the previous PEA operation. The switch between two nearby I-PRTs needs a reconfiguration (indicated by an arrow in Figure 3(e)) and the switch between an R-PRT and an I-PRT also needs a reconfiguration (also indicated by an arrow) because the functionality of the two adjacent PRTs is different due to the different number of iterations they hold. On the other hand, the switch between two R-PRTs does not need any reconfigurations. As a result, the reconfiguration times could be
roughly represented as follows:
\[ N_{cfg} = N_{ip} + N_{row} \]  
(7)

Then we put the result of Equation 7 into Equation 2, we obtained the whole reconfiguration cycles of the input loop task:
\[ \sum_{p \in P} CFG_p = (N_{ip} + N_{row}) \cdot CFG \]  
(8)

3.3.2 Communication Cycles
Let \( \vec{i}_p \) and \( \vec{j}_p \) be the iteration index of the source node and target node of dependence \( e \) in a loop nest. The set of dependence \( (E) \) could be calculated by the dependence polyhedron as described in Appendix A. As the work in [7], we take the number of hyperplanes the dependence \( e \) traverses as a communication cost function \( \delta \). Then, the communication cost of hyperplane \( \Theta \) and \( \Pi \) could be represented as follows:
\[ \delta_e(\Theta) = (\Theta(\vec{i}_p) - \Theta(\vec{j}_p)), \quad \delta_e(\Pi) = (\Pi(\vec{i}_p) - \Pi(\vec{j}_p)) \]  
(9)

We first consider the communication volume of the R-PRT, which holds \( n \times \zeta \) loop iterations. As shown in Figure 4(a), the communication volume of one R-PRT could be represented as follows:
\[ 2N_{ip} \sum_{e \in E} (\eta \cdot \delta_e(\Theta) + \zeta \cdot \delta_e(\Pi) - \delta_e(\Theta) \delta_e(\Pi)) \]  
(10)

The communication volume of I-PRT (Figure 4(b)) is different because some PEs are not occupied. Thus, the communication volume of I-PRT is a part of R-PRT and we give a scale factor \( \beta \in (0,1) \) to indicate the difference. Therefore, the communication volume of I-PRT could be represented as follow:
\[ 2\beta N_{ip} \sum_{e \in E} (\eta \cdot \delta_e(\Theta) + \zeta \cdot \delta_e(\Pi) - \delta_e(\Theta) \delta_e(\Pi)) \]  
(11)

Actually, once the loop nest is transformed and tiled, \( \beta \) could be determined. For instance, as depicted in Figure 4(a), the communication volume of the R-PRT is 14 (7 inputs and 7 outputs) and the communication volume of I-PRT (Figure 4(b)) is 8 (4 inputs and 4 outputs). As a result, the value of \( \beta \) in this transformation is 8/14 ≈ 0.57.

Considering Equation 3, the total number of communication cycles of the loop task is obtained:
\[ \sum_{p \in P} (CMC_p) = 2N_{ip} \sum_{e \in E} (\eta \cdot \delta_e(\Theta) + \zeta \cdot \delta_e(\Pi) - \delta_e(\Theta) \delta_e(\Pi)) \]  
+ \[ 2N_{ip} \beta \sum_{e \in E} (\eta \cdot \delta_e(\Theta) + \zeta \cdot \delta_e(\Pi) - \delta_e(\Theta) \delta_e(\Pi)) \]  
(12)

3.3.3 Computing Cycles
As analyzed in Equation 4, the computing cycles (CPC) has a close relationship with the \( RS \) and \( L_F^p \). We also define the length of dependence as the number of hyperplanes a dependence \( e \) transverse from the source node to target node. So the modified computing cycles could be represented as follow:
\[ \sum_{p \in P} (CPC_p) = N_{ip} Func\left( RS(\vec{i}_p), \Theta(\vec{i}_p) - \Theta(\vec{j}_p), \Pi(\vec{i}_p) - \Pi(\vec{j}_p) \right) \]  
(13)

There are mainly three interconnection topologies of CGRA\[11\], Mesh, Mesh plus and Morphosys. In the Mesh style CGRAs, a PE connects to its 4 neighbor PEs, \( Func(\cdot) \) could be represented as \( Max_{e \in E}(L_F^p) - 1 \) as dependence more than 1 would be passed by registers. In the Mesh plus CGRAs, \( Func(\cdot) \) could be represented as \( Max_{e \in E}(L_F^p) - 2 \) as dependence more than 2 would be passed by registers. In the Morphosys CGRAs, \( Func(\cdot) \) could be represented as \( Max_{e \in E}(L_F^p) - 4 \) as dependence more than 4 would be passed by registers.

From all the discussion above, now we could give the integral optimization target base on affine transformation:
\[ TET = (N_{ip} + N_{row}) \cdot CFG \]  
\[ = 2N_{ip} \sum_{e \in E} (\eta \cdot \delta_e(\Theta) + \zeta \cdot \delta_e(\Pi) - \delta_e(\Theta) \delta_e(\Pi)) \]  
+ \[ 2N_{ip} \beta \sum_{e \in E} (\eta \cdot \delta_e(\Theta) + \zeta \cdot \delta_e(\Pi) - \delta_e(\Theta) \delta_e(\Pi)) \]  
+ \[ N_{ip} Func\left( RS(\vec{i}_p), \Theta(\vec{i}_p) - \Theta(\vec{j}_p), \Pi(\vec{i}_p) - \Pi(\vec{j}_p) \right) \]  
(14)

3.4 Determination of Optimization Problem
Based on the preparation above, now we could establish the optimization problem as follow:
\[ Minimize \ TET \]  
Subject to \[ \Theta(\vec{i}_p) - \Theta(\vec{j}_p) \geq 0, \ e \in E \]  
(15a)
\[ \Pi(\vec{i}_p) - \Pi(\vec{j}_p) \geq 0, \ e \in E \]  
(15b)
\[ c_2d_1 - c_1d_2 = 1 \]  
(15c)
\[ c_1, c_2, d_1, d_2 \in Z \]  
(15d)

Where Equation 15a and Equation 15b give the constraints of legal affine transformation, which is the same as the work in [7]. Equation 15c gives the constraint of tightness of transformed space. Based on the performance metric, we could find the optimal coefficients \( (c_1, c_2) \) and \( (d_1, d_2) \) to perform affine transformation.

4. EFFICIENT SOLUTION AND POLYMAP
The calculation of TET in Equation 14 is non-convex and an enumerate-based approach is needed. Instead of enumerating all legal transformation coefficients in a brute-force way, we used the features of constraints to greatly reduce the computation complexity of finding optimal solutions for large design space. We note that the condition 15c is an equation of the 4 variables \( c_1, c_2, d_1 \) and \( d_2 \) and we could reduce the dimensionality of the problem to 3. Thus, the variable \( d_2 \) could be expressed by the combination of \( c_1, c_2 \) and \( d_1 \). As we only consider the innermost two loops of loop nests, the design space is not quite big in fact. Thus, we adopt Genetic Algorithm[8] to solve the problem to guarantee the accuracy of the results. In order to crossover the variables, a \( N \) bits binary \( BX \) 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 \( (c_1, c_2, d_1) \) could be represented by a \( 3N \) bits binary (chromosome). We use the inverse of TET as the fitness function and the definition domain (DO) of variables is formed by inequality 15a, 15b and 15d.
The target CGRA architecture is a $8 \times 8$ homogeneous PEA and every PE could perform fixed-point and logic operations, which is an instance of ADRES[11] template. Each PE has a access to the data memory and the interconnection topology is Morphosis. Each PE has its own local RF consisting of 5 rotating registers with 5 read and 5 write ports, which could be accessed by the local ALU and its 4 neighbor PEs. The configuration cycles of one PEA operation is 2 cycles if needed.

To evaluate the effectiveness of our approach (PolyMAP) we select two of the latest presented approaches as references. The first reference is EPIMap[5]. EPIMap used re-computation for resource limitation and obtained optimized II for a single-level loop. The second reference is communication minimized optimization of nested loop based on polyhedra model[6][labeled PolyCOM], where the communication of PEA is taken as an optimization metric. All the experiments were taken on an Intel Dual-Core CPU machine running at 1.9GHz with 2GB memory. Mapping results were verified with cycle accurate simulator.

5.1 The Accuracy of Performance Model

Our optimization approach is based on the deduced performance model in Equation 14. Thus, the accuracy of the performance model is very important to the results of the optimization. To evaluate the accuracy of our performance model, we use PolyMAP to mapping loops onto CGRA and give a report of execution time obtained by performance model. Then, the experimental results is also obtained by running configurations on the target CGRA. Figure 6(a) give the comparison between performance model and actual experiments results. We can see the execution time obtained by performance model is accurate enough compared with the actual experimental result. Consequently, our mapping optimization based on the performance model is well-founded.

5.2 The Execution Time Comparison

The performance comparison of three different mapping techniques is shown in Figure 6(b).

Comparison with PolyCom. Although both PolyCom and PolyMap use polyhedra model as the loop optimization tools to perform transformation, PolyMap always outperforms PolyCom by more than 36% on average. The poor performance of PolyCom is due to the improper performance metric. Figure 7 is used to give further explanation. The goal of PolyCom is to minimize the communication between host controller and PEA. Indeed, PolyCom works quite well for communication minimization. As shown in Figure 7, the communication cycles ($t_{com}$) of PolyCom on kernel adi and gemm are less than that of EPIMap and PolyMap. However, the communication cycles is only part of total execution time. The excessive optimization of communication cycles results in exacerbation of reconfiguration cost and computation cost. For instance, the configuration cycles ($t_{cfg}$) of PolyCom are more than that of EPIMap and PolyMap, as shown in Figure 7(a). On the other hand, PolyMap uses the unified metric, TET, which can achieve global optimal result. As a result, the overall performance of PolyMap exceeds that of PolyCom.

Comparison with EPIMap. As shown is Figure 6(b), PolyMap performs better than EPIMap by more than 21% on average. EPIMap works well on the mapping of resource limited CGRA and could achieve the minimized II on most.
### 6. CONCLUSIONS

Many compute-intensive loops are often mapped on to CGRAs for acceleration and there is not a unified metric to perform loop transformations. To this end, we first build a CGRA performance model considering synthetically performance influencing factors and formulate an optimization problem for coefficients to perform affine transformations based polyhedral model. Then, we extract an efficient heuristic to reduce the searching space. At last, the effectiveness of our proposed algorithm is demonstrated with PolyBench and real-life applications.

### 7. REFERENCES


---

### Figure 6: Experiments of PolyMap compared to PolyCom and EPIMap

<table>
<thead>
<tr>
<th>Performance Model</th>
<th>Experiment Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyCom</td>
<td>EPIMap</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>3000</td>
<td>3000</td>
</tr>
<tr>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
</tr>
</tbody>
</table>

### Figure 7: The time component parts of the three approaches on kernel adi and gemm

(a) Speedup of PolyMap compared to PolyCom and EPIMap

(b) Compilation time of PolyMap compared to PolyCom and EPIMap

---

5.3 The Compilation Time

Figure 6(c) demonstrated the compilation time of PolyMAP for different kernels, as compared to PolyCom and EPIMap. The compilation time of PolyMAP varies from several milliseconds to dozens of seconds and more than that of PolyCom and EPIMap on average. Actually, the number of dependence is an important factor influencing the compilation time. For instance, adi, dftd-2d, ftd-apm and jacobi-1d-imper have data dependence more than two and they all have a longer compilation time in PolyCom and PolyMAP. As a result, PolyMAP achieves better average performance at the cost of compilation time to find the optimal coefficients to perform loop transformation. However, the actual compilation time is acceptable.
APPENDIX

A. POLYHEDRAL MODEL

Polyhedron Model is based on four main concepts: the iteration domain, the access function, dependence polyhedra and affine transformation. A program part that can be represented using the polyhedron model is called a Static Control Part or SCOP for short.

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

$$D_S \cdot \begin{pmatrix} iS \noS \end{pmatrix} \geq 0$$  \hspace{1cm} (16)

where $D_S$ is a matrix of $n$ affine constraints on the execution of statement $S$. $iS$ is the iteration vector of statement $S$ and $\noS$ is a vector of global parameters.

Each reference in a statement is also an affine function of loop indices and global parameters, which could also be represented using matrices $\mathbf{c}$:

$$\mathbf{c} = \mathbf{F} \cdot \begin{pmatrix} iS \noS \end{pmatrix}$$ \hspace{1cm} (17)

Where $\mathbf{c}$ is a matrix representing an affine mapping form the iteration space of statement $S$ to the data space of array $S$.

The Polyhedral Dependence Graph (PDG) is a directed multi-graph with each vertex representing a statement, and an edge, $e \in E$, from node $S_i$ to $S_j$, representing a polyhedral dependence from a dynamic instance of $S_i$ to one of $S_j$: it is characterized by a polyhedron, $P_e$, called the dependence polyhedron that captures the exact dependence information corresponding to $e$. The dependence polyhedron is in the sum of the dimensionality of the source and target statement’s polyhedra. The $h$-transformation $[7]h_e$ maps the target iteration vector $i_{t}$ to the source iteration vector $i_{p}$ that is the last access the same index of a array. So the dependence polyhedra can be represented as:

$$P_e = \left( \begin{array}{c} D_s \\ h_s \end{array} \right) D_l$$  \hspace{1cm} (18)

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

$$\Phi(iS) = T_S \cdot \begin{pmatrix} iS \noS \end{pmatrix} \geq 0$$ \hspace{1cm} (19)

Where $T$ is a row vector and the affine transformation is a one-dimension mapping, which can be also called an affine hyperplane. When mapping loop nests on a 2-D PEA, we use $\Pi$ to denote row related hyperplane, and $\Theta$ to denote the column related hyperplane.

B. THE NUMBER OF PRTS

**Theorem 1.** Let $W$ and $L$ be the width and length of a right-angled triangle $T$. $c_1, d_1$ are mutually prime integers satisfying equation $W/L = c_1/d_1$. $T$ is covered by minimal number of squares $S$ with sides of 1. The $S$ well within $T$ is called $S_{TP}$. The $S$ with portions in $T$ and the left portions out of $T$ is called $S_{TP}$. Let $N_{TP}, N_{TP}$ and $N_{TP}$ be the number of $S_{TP}, S_{TP}$ and $S_{TP}$ and all $S$s. Then

$$N_{TP} = \frac{W}{c_1} \left( c_1 + d_1 - 1 \right)$$

$$N_{TP} = \frac{1}{2} \left( W + \frac{W}{c_1} (c_1 + d_1 - 1) \right)$$

PROOF. As shown in Figure 8(a), we first consider the situation of smallest repeat triangle (SRT) with two mutually prime square edges $c_1$ and $d_1$. We note that the slanting edge crossed one edge of the two squares at the end of the slanting edge and crossed two edge of the squares inside the slanting edge. Thus, the number of squares that the slanting edge crossed is

$$2(c_1 - 1 + d_1 - 1) + 2 = c_1 + d_1 - 1$$

Now we could extend to the situation of a right-angled triangle without two mutually prime square edges. With the condition $W/L = c_1/d_1$, $N_{TP}$ could be obviously obtained

$$N_{TP} = \frac{W}{c_1} \left( c_1 + d_1 - 1 \right)$$

The the number of the $S$s covering the whole triangle $T$ is the sum of area of $T$ and half of $N_{TP}$, we obtained

$$N_{TP} = \frac{1}{2} \left( W + \frac{W}{c_1} (c_1 + d_1 - 1) \right)$$

At last, the number of $S_{TP}$ could be calculated by $N_{TP}$ and $N_{TP}$

$$N_{TP} = N_{TP} - N_{TP} = \frac{1}{2} \left( W + \frac{W}{c_1} (c_1 + d_1 - 1) \right)$$

**Theorem 2.** Let $W$ and $L$ be the width and length of a right-angled triangle $T$. $c_1, d_1$ are mutually prime integers satisfying equation $W/L = c_1/d_1$. $T$ is covered by minimal number of rectangle $R$ with width of $n$ and length of $\zeta$. The $R$ well within $T$ is called $R_{TP}$. The $R$ with portions in $T$ and the left portions out of $T$ is called $R_{TP}$. Let $N_{TP}, N_{TP}$ and $N_{TP}$ be the number of $R_{TP}, R_{TP}$ and all $R$s. Let $\text{lcm}(a, b)$ be the
least common multiple of two integers $a$ and $b$. Then

$$N_p = \left[ \frac{W_c}{\eta c} \right] \left( \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right)$$

Then

$$N_p = \frac{1}{2} \left[ \frac{W_L}{\eta c} \right] \left( \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right)$$

$$N_p = \frac{1}{2} \left[ \frac{W_L}{\eta c} \right] \left( \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right)$$

**Proof.** As shown in Figure 9(b), we first determine the size of smallest repeat triangle (SRT). As the side of $R$ is no longer 1, the sides of SRT are no longer $c_1$ and $d_1$. Actually, the sides of SRT are multiples of $c_1$ and $d_1$ and the multiply could be represented as

$$\lambda = \text{lcm} \left( \frac{\text{lcm}(c_1, \eta)}{c_1}, \frac{\text{lcm}(d_1, \zeta)}{d_1} \right)$$

Now we could consider the situation of SRT, which could be normalized as situation in Theorem 1 by dividing the size of SRT by the size of $R$. As a result

$$N_p = \frac{1}{2} \left[ \frac{W_L}{\eta c} \right] \left( \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right)$$

$$N_p = \frac{1}{2} \left[ \frac{W_L}{\eta c} \right] \left( \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right)$$

$$N_p = \frac{1}{2} \left[ \frac{W_L}{\eta c} \right] \left( \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right)$$

**Theorem 3.** Let $M - 1$ and $N - 1$ be the length and width of original rectangle $O$. $(c_1, c_2)$ and $(d_1, d_2)$ be the coefficients two affine transformations $\Phi$ and $\Pi$ satisfying $|c_2d_1 - c_1d_2| = 1$. $O$ is transformed by $\Phi$ and $\Pi$ into a new parallelogram $P$ shown in Figure 3(e). $P$ is covered by minimal number of rectangle PRT with width of $\eta$ and length of $\zeta$. The PRT well in $P$ is called $R - PRT$. The PRT with portions in $P$ and the left sections out of $P$ is called $I - PRT$. Let $N_{r_1}, \eta_{r_1}, N_{i_1}, \eta_{i_1}$ and $N_{row}$ be the number of $R - PRT$'s, $I - PRT$s, all PRTs and the rows of PRT. Then

$$N_{i_1} = 2 \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

$$N_{i_1} = \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

$$N_{i_1} = \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

$$N_{row} = \left[ d_1(M - 1) + d_2(N - 1) \right]$$

where,

$$\lambda = \begin{cases} \frac{\zeta}{\text{lcm}(\text{lcm}(c_1, \eta), d_1, \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{if } c_1 = 0, d_1 = 0, \\
\frac{\zeta}{\text{lcm}(\text{lcm}(c_1, \eta), d_1, \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{else} \end{cases}$$

$$\lambda' = \begin{cases} \frac{\zeta}{\text{lcm}(\text{lcm}(c_2, \eta), d_1, \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{if } c_2 = 0, d_2 = 0, \\
\frac{\zeta}{\text{lcm}(\text{lcm}(c_2, \eta), d_1, \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{else} \end{cases}$$

**Proof.** Without loss of generality, we first discuss the case: $|c_1d_2| > 0$ and $|c_2d_1| > 0$. As shown in Figure 3(e), the size parameter $L_x, W_x, L_y$ and $W_y$ could be obviously represented as

$$W_x = \left| c_1 \right| (M - 1)$$

$$L_x = |d_1| (M - 1)$$

$$W_y = \left| c_2 \right| (N - 1)$$

$$L_y = |d_2| (N - 1)$$

Actually, the $N_{row}$ could be calculated by the number of $R - PRT$s in $T_1, T_2, T_3$ and $T_4$ and the number of $PRT$s in $R_1$ and $R_2$. With Theorem 2, the number of the $R - PRT$s in the 6 mentioned parts is

$$N_{row} = \left( \frac{W_xL_x}{\eta c} - \frac{W_xL_y}{\eta c} \right) \left( \frac{\lambda c_1}{\eta} + \frac{\lambda d_1}{\zeta} - 1 \right)$$

$$N_{row} = \left( \frac{W_xL_x}{\eta c} - \frac{W_xL_y}{\eta c} \right) \left( \frac{\lambda c_1}{\eta} + \frac{\lambda d_1}{\zeta} - 1 \right)$$

$$N_{row} = \left( \frac{W_xL_x}{\eta c} - \frac{W_xL_y}{\eta c} \right) \left( \frac{\lambda c_1}{\eta} + \frac{\lambda d_1}{\zeta} - 1 \right)$$

As shown in Figure 3(e), $N_{i_1}$ could be obtained by the $I - PRT$s in the 4 triangles, $T_1, T_2, T_3$ and $T_4$. Using Theorem 2, we obtained

$$N_{i_1} = 2 \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

$$N_{i_1} = \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

$$N_{i_1} = \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

$$N_{i_1} = \left[ \frac{\lambda c_1 + \lambda d_1}{\eta} - 1 \right]$$

Finally, the number of rows of PRTs could be calculated by

$$N_{row} = \frac{L_x + L_y}{\zeta} = \left| d_1(M - 1) + d_2(N - 1) \right|$$

In the case of $c_1d_2, d_2 = 0$, we redefine the parameter $\lambda$ and $\lambda'$ as follow

$$\lambda = \begin{cases} \frac{\zeta}{\text{lcm}(\text{lcm}(c_1, \eta), \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{if } c_1 = 0, d_2 = 0, \\
\frac{\zeta}{\text{lcm}(\text{lcm}(c_1, \eta), \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{else} \end{cases}$$

$$\lambda' = \begin{cases} \frac{\zeta}{\text{lcm}(\text{lcm}(c_2, \eta), \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{if } c_2 = 0, d_2 = 0, \\
\frac{\zeta}{\text{lcm}(\text{lcm}(c_2, \eta), \text{lcm}(d_1, \zeta), \text{lcm}(d_1, \zeta))}, & \text{else} \end{cases}$$

With the newly defined $\lambda$ and $\lambda'$, the result of Theorem 3 could also apply to the case of $c_1d_2d_2 = 0$. □