PT-Map: Efficient Program Transformation Optimization for CGRA Mapping

Bizhao Shi, Tuo Dai, Jiaxi Zhang, Xuechao Wei, and Guojie Luo
School of Computer Science, Peking University
Center for Energy-efficient Computing and Applications, Peking University
Beijing, China
{bshi,daitoto,zhangjiaxi,xuechao.wei,gluo}@pku.edu.cn

ABSTRACT

Coarse-Grained Reconfigurable Array (CGRA) is a parallel architecture providing high energy efficiency and spatial-temporal reconfigurability. Beyond loop scheduling for throughput optimization, program transformation is also crucial in CGRA mapping to optimize overall performance and efficiency. However, existing studies on program transformation optimization face challenges in exploring the transformation space systematically and evaluating candidates efficiently, leading to sub-optimal results. To tackle these challenges, this paper introduces PT-Map, an efficient program transformation optimization framework for CGRA mapping. PT-Map defines a comprehensive transformation space and employs a CGRA-specialized top-down exploration approach. It also incorporates a bottom-up evaluation scheme using architectural parameters and a graph neural network-based predictive model. Experiments demonstrate that PT-Map achieves up to 2.95×/1.80× speedups and 59.0%/23.2% energy-delay-product (EDP) reductions over the state-of-the-art approaches MapZero and PBP, respectively.

ACM Reference Format:
Bizhao Shi, Tuo Dai, Jiaxi Zhang, Xuechao Wei, and Guojie Luo. 2024. PT-Map: Efficient Program Transformation Optimization for CGRA Mapping. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3656257

1 INTRODUCTION

Coarse-grained Reconfigurable Arrays (CGRAs) have gained attention for their performance, word-level spatial-temporal reconfigurability, and energy efficiency. Fig. 1a shows a 4x4 mesh-connected CGRA. Each PE performs operations by ALU and routes the data by MUX and local register file (LRF). Besides the PE array, there is an on-chip data buffer (DB), context buffer (CB), and auxiliary modules like global register file (GRF) and data synchronizer. The regular parallel architectures of CGRAs present numerous opportunities and challenges for application mapping. As CGRAs usually execute computation-intensive innermost loops in a pipelined manner, there are generally two distinct spaces for the mapping optimization: **loop scheduling** and **program transformation**. Loop scheduling (Fig. 1b) minimizes initiation interval (II) of the pipeline considering SW/HW constraints like operator scheduling, placement, and data routing [2, 10, 11]. Program transformation (Fig. 1c) compilers explore equivalent transformations of the entire program to optimize overall performance and efficiency [12, 13, 17]. In this paper, we focus on the program transformation optimization.

The key to program transformation optimization lies in exploring the transformation space and selecting the high-performance or high-efficient candidates with proper spatial-temporal granularities (e.g. #operations in DFGs, tripcount, and working set size). However, obtaining the actual performance requires multiple time-consuming loop schedulings for the pipelined loops, making it difficult to apply common tuning-based techniques. Existing approaches generate many candidate transformations and rank them based on performance profiling. Multiple loop schedulings are then performed to find the highest-ranking transformation candidate with all mappable innermost loops. Currently, there have been some program optimization approaches with heuristic-based explorations and some specific objectives, including total execution cycles [12], imperfect loop structures [13], data reuse [6] and others. PolyMap [12] builds the CGRA execution model using the polyhedral variables and proposes a genetic-based exploration algorithm. IP [17] explores the loop interchanges before modulo scheduling. PBP [13] proposes an analytical model for performance profiling and explores loop fusion/fission and interchange. Huang et al. [6] optimizes data reuse with a highly-reduced transformation space. However, due to the flexibility of CGRA architectures and applications, the existing studies struggle to achieve satisfactory results for two challenges: 1) The comprehensive transformation dimensions should be represented and explored efficiently, taking into account CGRA execution characteristics. 2) Adaptive performance profiling methods for candidate transformations are required to accommodate SW&HW flexibility.

To address these challenges, we present PT-Map, a novel program transformation optimization framework for CGRA mapping. In contrast to existing approaches, PT-Map leverages the fine-grained execution capabilities of CGRAs and effectively explores both inter-loop and intra-loop transformation opportunities in a hierarchical manner. Additionally, PT-Map introduces an adaptive performance profiling approach to identify the promising transformation candidates. Our contributions are summarized as follows:

- PT-Map proposes a CGRA-specialized top-down exploration method to efficiently explore the comprehensive program transformation space, including inter-loop and intra-loop transformation dimensions.
- PT-Map proposes an adaptive bottom-up performance profiling approach with the assistance of a graph neural network, which bridges the fine-grained pipelined loops and the program-level

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656257

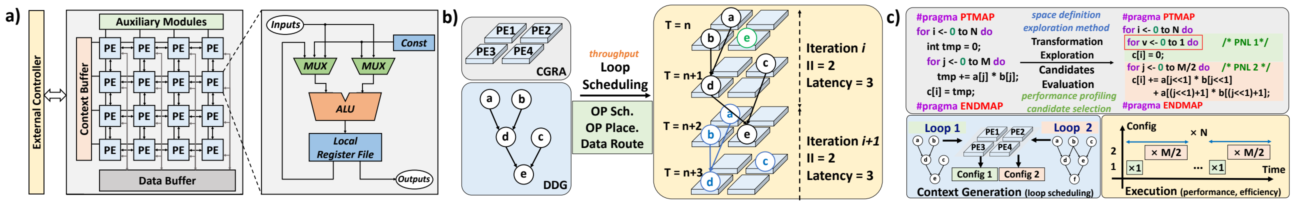


Figure 1: a) CGRA example with a 4x4 PE array. b) Example of loop scheduling. c) Example of program transformation.

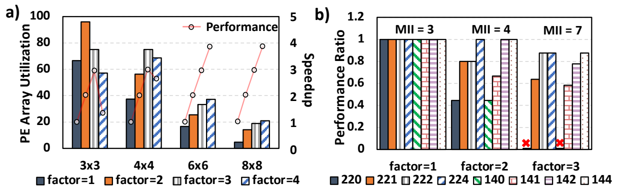


Figure 2: a) Impact of loop unrolling for PE utilization and performance. b) Accuracy of the MII-based analytical model.

optimization objectives with the consideration of both computation and memory access.

- PT-Map achieves 3.45×/2.95× speedups and 66.1%/59.0% EDP reductions compared to the state-of-the-art loop scheduling mappers, LISA [11] and MapZero [10]; and achieves up to 1.80× speedup and 23.2% EDP reduction compared to the state-of-the-art program transformation mapper PBP [13].

2 MOTIVATION

To motivate our work, we first illustrate and analyze the limitations of existing studies through two examples. In Fig. 2a, we use matrix multiplication as an example, where the input/output matrices are all 24x24. We evaluate the PE array utilization and the normalized performance under various loop unrolling factors. Here, the case of factor=1 denotes that only inter-loop transformations are applied. Our findings indicate that inter-loop transformations commonly explored in the existing works result in under-utilizing large PE arrays (only 4.7% in the 8x8 CGRA), while the utilization and performance can be significantly improved through loop unrolling. Moreover, for 3x3 and 4x4 CGRAs, fluctuations in the case of factor=4 primarily stem from the nonlinear variation of mapped II with the unroll factor. This example demonstrates the potential to fully exploit CGRA's parallelism through intra-loop transformation, specifically a proper loop unrolling factor, which has not been explored in the existing studies. Therefore, **the comprehensive space should contain both inter-loop and intra-loop transformation dimensions.**

Despite the opportunities presented by loop unrolling, several challenges remain. 1) The most straightforward one is the significantly expanded design space. With the loop unrolling, there are more about 32768× transformation choices for the case in GEMM, which will challenge the heuristics-based generation and exploration methods in the existing mappers. Therefore, **within the comprehensive transformation space, the efficient exploration method specialized for CGRAs is critical to the overall efficiency.** 2) Considering the larger DFGs and large IIs for the unrolled loops, there are much more mapping choices between DFG nodes and PEs. The MII-based analytical model (used in [6, 13]) may produce an inaccurate estimation and further affect the ranking of transformation candidates. Here, we use the vector reduction as another example. Fig. 2b showcases the ratio of actual performance and the estimated performance provided by the MII-based analytical model on different CGRA architectures with the same number of PEs. The legend number *abc* (e.g., 220) denotes an $a \times b$ CGRA with c LRFs per PE. The programs with the same unrolling factors have the same MIIs. Notably, all of the performance ratios reach 1.0 when factor=1, while the larger unrolling factors leads to more obvious errors in the MII-based model. Even worse, such an oversimplified model will have a greater impact when the PEs are heterogeneous and interconnections are with complex patterns. Therefore, **it is necessary to build an adaptive and accurate performance model considering SW&HW flexibility.**

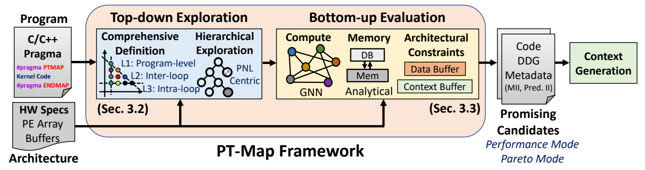


Figure 3: The overview of PT-Map.

3 PT-MAP FRAMEWORK

3.1 Overview

Fig. 3 demonstrates the workflow of PT-Map. The inputs to PT-Map consist of two components: a) C/C++ programs annotated between `#pragma PTMAP` and `#pragma ENDMAP`, where the annotations define the scope of the application mapped on CGRA; b) CGRA architecture information, including the specification of PE, array shape, buffer size, and interconnect. PT-Map comprises two main stages: a) it constructs the transformation space using several primitives and explores the space in a top-down fashion, and then, b) as the multiple transformations obtained, PT-Map constructs the program-level profiling from the bottom up with the nested loop-level estimation and further ranks these candidates. The outputs of PT-Map are two lists of transformations ranked in the performance mode or the balanced mode (also Pareto mode). Finally, the context generation determines the highest-ranking transformation candidate with all mappable innermost loops by loop schedulings.

3.2 Top-down Exploration

Based on the motivating examples, PT-Map introduces program-level, inter-loop, and intra-loop transformation opportunities to its program transformation space. Tab. 1 lists the definitions and major impacts of the transformation primitives in the context of CGRA. The program-level transformation refers to the heuristics or manually-designed fusion or fission. It usually restructures the whole program and leads to different loop nesting. The inter-loop transformations include reordering, tiling, and flattening. They have composite impacts on CGRA mapping qualities, such as temporal granularity (loop tripcount), DB utilization, and memory access patterns. The intra-loop transformations in PT-Map focus on unrolling, mainly affecting the spatial granularities of the pipelined loops, the utilization of PE array and CB. Given the multiple transformation dimensions with the complex impacts on CGRA, organizing them using an efficient exploration is crucial.

In order to fully utilize the fine-grained nature of CGRA execution and bridge it with the whole program transformation space, PT-Map proposes a top-down exploration approach that efficiently generates transformation candidates. This approach adopts a *per-*

Table 1: Transformation primitives definition and impacts.

Granularity	Primitive	Design Choices	Impacts
Program	Fusion/Fission	Heuristic/Manual	Overall Program Structure
Inter-loop	Reordering	Permutation	Compute: Temporal Granularity
	Tiling	$2^x, x \in [4, 10]$	Utilization: CB/DB
	Flattening	Index Pair	Data: Pattern, Locality, Dependency
Intra-loop	Unrolling	$1 \sim 8$	Compute: Spatial Granularity Utilization: PE Array, CB

fectly nested loop (PNL)-centric perspective, explicitly dividing the exploration into three levels: program-level, out-PNL, and in-PNL. The explored dimensions at the different levels are shown in Fig. 4a, which effectively prunes redundant transformations that have minimal impact on CGRA execution. Besides, PT-Map proposes a loop index tree (LIT) for the efficient representations during the exploration. Fig. 4b illustrates the source code of a fused GEMM and the corresponding LIT. Each node of a LIT denotes a loop index in the program, and the direct edges are established based on the loop nesting relationship. Notably, there is a virtual root node that represents the unified entry point of the entire program. Leveraging the LIT, it is easy to determine whether a code snippet corresponds to a sub-LIT with node i as the root, indicating a PNL.

• **Program-level exploration:** PT-Map focuses on heuristic-based loop fusion/fission choices (e.g., maxfuse/nofuse/smartfuse in PLuTo [1]) during the program-level exploration. To accommodate the potential restructuring of the entire program, PT-Map constructs an individual LIT for each fused/fissioned program. The PNL-related explorations are then performed within the corresponding LITs.

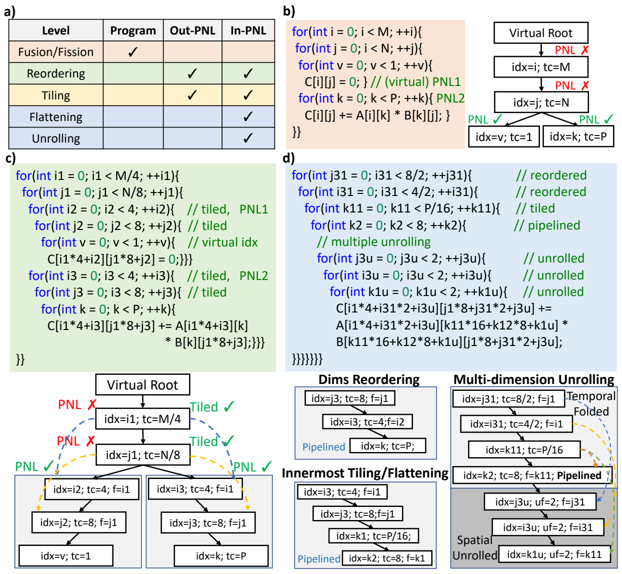
• **Out-PNL exploration:** Fig. 4c shows the out-PNL exploration. It follows a breadth-first-search (BFS) style and aims to lower the transformations of non-PNL nodes to their PNL children nodes. When visiting a non-PNL node i, PT-Map examines the potential and validity for loop tiling and distributes the tiled index to its children through loop reordering while considering data dependency constraints. Ideally, every tiled node can reach or become a new PNL through loop reordering. If the tiled nodes cannot reach any PNL, tiling is disregarded for the node i. During the exploration, if the current node is a PNL node, it enters the in-PNL exploration.

• **In-PNL exploration:** The in-PNL exploration consists of three main stages: loop reordering enumeration, innermost tiling or flattening, and multi-dimensional unrolling. First, PT-Map generates valid loop orders through loop reordering, focusing on the innermost three levels. Additionally, if a tiled node j is derived from the parent non-PNL node, the innermost three levels must include j. After loop order enumeration, PT-Map considers loop tiling or flattening for the pipelined loop to control temporal granularity. Fig. 4d demonstrates an example for tiling the current innermost loop. Finally, PT-Map generates multi-dimensional unrolling, ensuring that the unrolling nodes are placed under the pipelined level to affect the actual spatial granularity. After the exploration, various in-PNL transformations are recorded and backtracked to the non-PNL parents.

Upon completing the exploration within each LIT (as well as their PNL children) resulting from the program-level explorations, the outcomes can be viewed as a result forest. Non-leaf nodes represent the corresponding non-PNL nodes, while leaf nodes indicate the result arrays (RAs) of the original PNLs, which stores valid transformations (Fig. 5a). This approach records multiple valid PNL transformations in a single leaf node, reducing the overhead of representing transformation candidates in cases with multiple PNLs.

3.3 Bottom-up Evaluation

The evaluation performs the performance profiling and ranking



a)

Level	Program	Out-PNL	In-PNL
Fusion/Fission	✓		
Reordering		✓	✓
Tiling		✓	✓
Flattening			✓
Unrolling			✓

b)

```
for(int i = 0; i < M; ++i){
 for(int j = 0; i < N; ++j){
  for(int v = 0; v < 1; ++v){
   C[i][j] = 0; } // (virtual) PNL1
  for(int k = 0; k < P; ++k){ PNL2
   C[i][j] += A[i][k] * B[k][j]; }
}}
```

c)

```
for(int i1 = 0; i1 < M/4; ++i1){
 for(int j1 = 0; j1 < N/8; ++j1){
  for(int i2 = 0; i2 < 4; ++i2){   // tiled,  PNL1
   for(int j2 = 0; j2 < 8; ++j2){ // tiled
    for(int v = 0; v < 1; ++v){   // virtual idx
     C[i1*4+i2][j1*8+j2] = 0;}}}
  for(int i3 = 0; i3 < 4; ++i3){   // tiled,  PNL2
   for(int j3 = 0; i3 < 8; ++j3){ // tiled
    for(int k = 0; k < P; ++k){
     C[i1*4+i3][j1*8+j3] += A[i1*4+i3][k]
                            * B[k][j1*8+j3];}}}
}}
```

d)

```
for(int j31 = 0; i31 < 8/2; ++j31){          // reordered
 for(int i31 = 0; i31 < 4/2; ++i31){         // reordered
  for(int k11 = 0; k11 < P/16; ++k11){       // tiled
   for(int k2 = 0; k2 < 8; ++k2){            // pipelined
    // multiple unrolling
    for(int j3u = 0; j3u < 2; ++j3u){          // unrolled
     for(int i3u = 0; i3u < 2; ++i3u){         // unrolled
      for(int k1u = 0; k1u < 2; ++k1u){        // unrolled
       C[i1*4+i31*2+i3u][j1*8+j31*2+j3u] +=
       A[i1*4+i31*2+i3u][k11*16+k12*8+k1u] *
       B[k11*16+k12*8+k1u][j1*8+j31*2+j3u];
}}}}}}}
```

Figure 4: Top-down exploration in PT-Map: a) explored transformations at different levels; b) code and LIT of the fused GEMM; c) out-PNL exploration; d) in-PNL exploration.

for the transformation candidates. Although the overall programs have complex structures, the CGRA execution in a PNL has regular behavior. Therefore, PT-Map proposes a bottom-up method to construct the program-level profiling from the PNL-level profiling.

The hierarchical profiling and ranking are based on our insights: 1) there is no pipelined loop executed on CGRA at non-PNL nodes, therefore, the computation cycles at these nodes can be calculated statically through their tripcounts; 2) the limited on-chip DB capacity makes the off-CGRA data access volume mainly determined by the critical PNLs. Therefore, based on the comprehensive PNL-level analysis, the program-level static analysis is effective for CGRA.

3.3.1 PNL-level Evaluation. The PNL-level evaluation in PT-Map (Fig. 5b) mainly contains GNN-assisted computational performance profiling and memory profiling for further ranking and selections.

• **GNN-assisted PNL performance profiling:** To provide accurate and adaptive computational performance estimations, PT-Map formulates the computational cycles on CGRA for a PNL first and then proposes a graph neural network (GNN)-based predictive method. Consider a PNL transformation p with the pipelined loop l, the computation cycles for $Cycle(\cdot)$ can be calculated as:

$$Cycle(l) = TC_l \times II_{map,l} + ProEpi_l \tag{1}$$

$$Cycle(p) = Cycle(l) \times \prod_{idx \in O(l)} TC_{idx} \tag{2}$$

where TC_l, $II_{map,l}$, and $ProEpi_l$ denote the tripcount, the mapped II, and the pipeline filling and draining cycles, respectively. Moreover, $O(l)$ denotes the set of the temporal folded loops (without l) of p.

With this formulation, the GNN model in PT-Map is designed to accurately predict II_{map} and $ProEpi$, which can only be obtained through loop scheduling. The motivation for using GNNs is twofold: a) the nature of SW/HW representations (DFG and TEC/MRRG) and the related graph-based formulations [2] for loop scheduling allow GNNs to extract features effectively; b) the generalizability of GNNs enables its application to various PNLs and CGRAs. The specializations of GNN in PT-Map include predictive tasks, input representations, and the model architecture with training strategies.

a) Predictive tasks: PT-Map designs three sub-tasks (Tab. 2) to reduce the difficulty of predictions. For II predictions, PT-Map int-

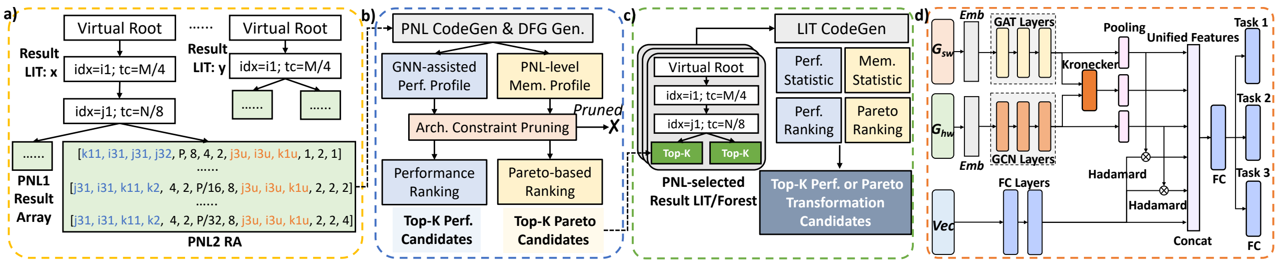


Figure 5: Bottom-up evaluation in PT-Map: a) the examples of generated result forest, result LIT, and result array for each PNL; b) the PNL-level evaluation; c) the program-level evaluation; d) the proposed GNN architecture.

Table 2: Task design for the GNN model in PT-Map.

Name	Task		Loss (Label: y_i, Output: $\hat{y}_i$)
II Equivalence	Classification	$MII? = II_{map}$	$CE(y_0, \hat{y}_0)$
II Residual	Regression	II_{res}	$MSE(y_1, \hat{y}_1) + \alpha \times MSE(1, \frac{x_{MII}+\hat{y}_1}{x_{MII}+y_1})$
ProEpi Prediction	Regression	$ProEpi$	$MSE(y_2, \hat{y}_2)$

roduces the prior MII and defines a residual variable II_{res} as $II_{res} \equiv II_{map} - MII$, which is a non-negative integer. With II_{res}, PT-Map converts the mapped II predictions into two sub-tasks (II Equivalence and II Residual). If the output $\hat{y}_0$ is true, the final II prediction will be set as MII. Otherwise, PT-Map checks the residual output $\hat{y}_1$ and adds to the MII as the final prediction. The ProEpi prediction is a relatively independent regression task. Besides, Cross-Entropy (CE) is used as the loss function for the classification task, and Mean-Square-Error (MSE) is used for the regression task. It is worth mentioning that the two-item loss function in the II residual task minimizes both the absolute error and the relative error of II_{res}, especially for cases with small MIIs. The small-MII cases can easily produce a large relative error and affect the performance ranking significantly. Overall, the predicted metrics in integers are:

$$II_{pred} = MII + (\hat{y_0} == False) * round(\hat{y_1}) \quad (3)$$

$$ProEpi_{pred} = round(\hat{y_2}) \quad (4)$$

b) Input representations: The GNN in PT-Map takes three inputs: pipelined loop representation (G_{sw}), architecture representation (G_{hw}), and mapping meta-data (Vec). Additional attributes are designed to fully exploit the DFG and TEC features (Tab. 3). In G_{sw}, we extend operator schedules under the ASAP/ALAP strategies; we also count the node degrees for data routing. In G_{hw}, we add a list of supported operators (op_list) for each PE to accommodate the heterogeneous array. We also add LRF and GRF sizes in G_{hw} for data routing ability. GRFs on G_{hw} act as PEs with interconnects but an empty op_list. Besides, Vec includes MII as a prior, Max Fanout for data routing, and Critical Path Length for ProEpi estimation.

c) Model architecture: The proposed GNN model architecture (Fig. 5d) includes stacked GNN layers to extract features for G_{sw} and G_{hw} embeddings from multi-hop neighbors. GAT [16] is used for G_{sw} with the multi-attribute edges, while GCN [9] is used for G_{hw} with homogeneous inter-PE connections (edges). After extraction, it is worth mentioning that PT-Map uses Kronecker product for feature alignment, allowing interactions between G_{sw} and G_{hw} gradients in the backpropagation. It is also similar with the compatibility graph in [2]. Pooling operators generate SW&HW graph-level feature vectors, which are then aligned with features of Vec using Hadamard products. These features are concatenated as a unified vector and mapped through an FC layer. Each task has its own FC layers to generate the corresponding output. Tasks are trained alternately using their own optimizers until convergence.

• **PNL-level memory profiling:** For each PNL transformation p, PT-Map performs analytical analysis of memory access. Here,

Table 3: Input attributes for the GNN predictive model.

Field	Basics	Attributes

G_{sw}	DDG	`base:` Node: {type, fan-in/out}, Edge: {type, order}
		`extend:` Node: {ASAP, ALAP, in/out-degree}

G_{hw}	TEC	`base:` PE Array Shape, Topology `extend:` Node: {op_list, #LRF, #GRF}
Vec	–	`extend:` MII, Max Fanout, Critical Path Length

PT-Map mainly considers the off-CGRA accesses volume $Vol(p)$ because the load/store unit and the context controller manage the on-CGRA data access. The off-CGRA data access can be formulated as a two-level cache analysis problem, where the CB/DB on CGRA is at the first level, and the off-CGRA cache/memory is at the second level. The off-CGRA data access contains two main types: computation and context. PT-Map gets the context-related volume through a simple simulation considering the regularity of the context loading and eviction. For the computation-related access volume, PT-Map adopts the analytical approach in [5] to obtain the estimations.

3.3.2 PNL-level Ranking and Selection.

With the computation and memory access estimations, PT-Map designs the hardware-aware pruning strategies and two-mode ranking methods.

• **Architectural constraint-based pruning:** The exploration in PT-Map generates transformations with different spatial-temporal granularities. Fine granularity leads to low utilization, while coarse granularity may even result in infeasible mappings. As the fine-grained candidates can be eliminated naturally through the performance ranking, PT-Map prunes the too coarse-grained transformations by the CB (in spatial) and DB (in temporal) capacities:

a) CB constraint: The mapped II directly affects CB utilization. If the mapped II exceeds the capacity, frequent context loading during the pipeline execution incurs overhead and may cause execution errors. PT-Map uses the predicted II from Eqn. (3) for pruning.

b) DB constraint: The pipelined loop tripcounts influence the working set and DB utilization. Working sets exceeding the capacity introduce significant overhead (data loading/eviction) and pipeline stalls. The working set calculation can be considered as a capacity miss detection problem. Here, PT-Map uses the previous PNL-level memory profiling and checks the capacity miss. If the count of capacity miss at the pipelined loop level is positive, the working set is too large for the DB. And this transformation should be pruned.

Overall, a transformation candidate will be pruned if its pipelined loop violates at least one of the two constraints.

• **Two-mode ranking and selection:** With the estimated computation cycles $Cycle(p)$ and the off-CGRA data access volume $Vol(p)$ of each candidate p for a PNL, PT-Map performs the ranking and top-K selection within two modes: a) performance mode; b) Pareto mode. In the performance mode, PT-Map ranks the transformations in ascending order by $Cycle(p)$ (the first priority) and $Vol(p)$ (the second priority). In the Pareto mode, PT-Map computes the Pareto Hypervolume $PVol(H(p))$, where $H(p) = (Cycle(p), Vol(p))$, and the reference point is carefully selected for the calculation. PT-Map

Table 4: Settings of the GNN model and its training.

Field	Component	Configurations
Dataset	Program	Random C Programs
	Architecture	PE Array: X × Y, where X, Y ∈ 2 ≤ X ≤ Y ≤ 8 PE Functions: Arithmetic, Logic, Memory Topology: ADRES [15], HyCube [7] LRF and GRF: 0, 1, 2, 4, 8
	Compilation	Compiler: RAMP [2]; Max II: 20; Time Limit: 1 hour
Training	Model	Stacked GCN/GAT/FC Layers: 3/3/2; Activation Func.: ReLU Hidden Dimensions: 128; Pooling Operation: Average
	Hyperparam.	Optimizer: Adam [8]; Learning Rate: 3e-4; α: 0.5 Batch Size: 256; Epochs: 300

Table 5: Apps and their #PNLs in the real benchmark.

App.	GEM	TRI	COV	DOI	TMM	ATA	BLU	HAR	CON	TCO	WIN
#PNLs	4	3	7	6	2	4	2	9	4	4	13

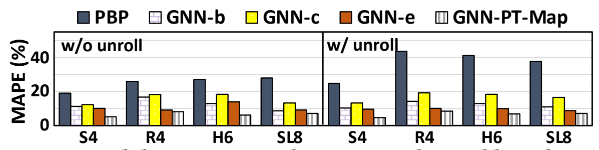


Figure 6: Model accuracy evaluation on the real benchmark.

ranks the candidates in descending order by $PVol(H(p))$.

3.3.3 Program-level Evaluation. After the top-K candidates are selected from each PNL, PT-Map performs statistical evaluations at the program level (or tree/forest level) with the PNL profiling data (Fig. 5c). In particular, PT-Map collects a list L for a given LIT T, where the specific candidate selection for the i^{th} PNL is stored in $L[i]$. And PT-Map performs the program-level profiling as:

$$f(T, L) = \sum_{i}^{\#PNL} (f(p_i(L[i])) * \prod_{j \in O(p_i)} TC_j), \quad f \in \{Cycle(\cdot), Vol(\cdot)\}$$

With these statistics, PT-Map further selects the top-K candidates in the corresponding mode, respectively. The selected program-level candidates will be the final outputs for context generation.

4 EVALUATIONS

4.1 Evaluation Setup

• **PT-Map Implementation:** The top-down exploration approach is implemented on the top of PLuTo [1]. The primitives are extended as PLuTo interfaces, and the validity is ensured by polyhedral model and data dependency checking. The bottom-up evaluation is implemented in C/C++ as standalone components or LLVM passes. The GNN model is implemented using PyG. PT-Map also extends RAMP [2] to support heterogeneous PEs and non-mesh topologies. Considering the overhead, we keep the top-20 candidates.

• **Benchmarks:** There are two benchmarks for evaluation. The synthetic one is for the GNN model training. We collect the paired data of inputs $\langle G_{sw}, G_{hw}, Vec\rangle$ and outputs $\langle II_{map}, ProEpi\rangle$ according to the settings in Tab. 4. In software, we implement a random C program generator that specifically targets single-level loops. These loops consist of scalars, arrays, affine or indirect memory access, and common arithmetic operators without complex control. In hardware, we randomly sample design choices from the shown architecture space. RAMP is used as the mapper with a 1-hour compilation time limitation. Finally, there are about 400,000 paired mapping data, and we split the training and test sets with a 3:1 ratio. The model and hyperparameters are also illustrated. The real benchmark is for evaluating mapping quality and compilation time, which includes loop-intensive programs (with multiple PNLs) from different domains: gemver (GEM), trisolv (TRI), covariance (COV), doitgen (DOI), 3mm (TMM), and atax (ATA) in PolyBench/C 3.2 (widely used in the evaluations of previous compilers [12, 13, 17]); blur2d (BLU) and harris (HAR) in image processing; convolution (CON), transposed convolution (TCO), and Winograd convolution

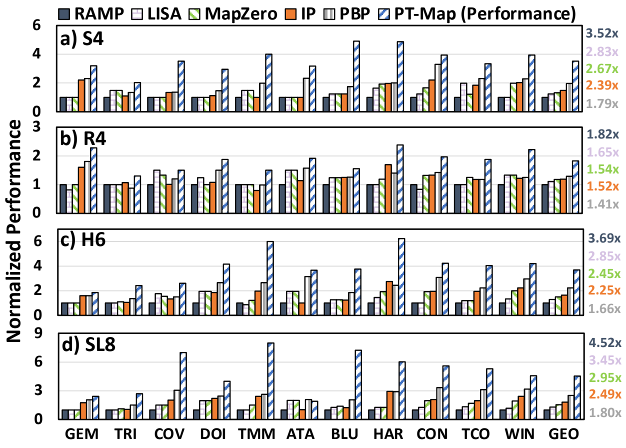


Figure 7: Normalized performance relative to RAMP.

(WIN) in deep learning. Tab. 5 presents their #PNLs in default.

• **Architectures:** There are four selected architectures: a) 4 × 4 standard CGRA (S4), b) 4 × 4 reduced CGRA (R4), similar to the reduced architecture in [4], c) 6 × 6 HyCube-like CGRA (H6), and d) 8 × 8 CGRA with less routing resource (SL8). The CB capacity is set as 8, considering the context-switching overhead. And the DBs are set as 4/4/6/8KB for S4/R4/H6/SL8, respectively. The performance is evaluated on a cycle-accurate simulator. As for the evaluation of relative energy efficiency, these architectures are synthesized at 45nm@200MHz to get reference power consumption, and the off-CGRA data access energy is obtained through CACTI.

• **Baselines:** There are two classes of baseline for comparisons: a) SOTA loop scheduling mappers: RAMP [2], LISA [11], and MapZero [10], where LISA and MapZero are GNN-assisted loop schedulers; b) SOTA program transformation mappers: IP [17] and PBP [13]. The loop scheduling back-ends for context generation in IP, PBP, and PT-Map are all set as our extended RAMP for fair comparisons. All methods are performed on a server with a 24-core CPU@3.0 GHz, 128 GB DDR4 memory, and NVIDIA RTX 3080Ti.

4.2 Experimental Results

• **Model accuracy:** The accuracy of the GNN model in PT-Map (marked as GNN-PT-Map) is evaluated on the real benchmark, with the mean absolute percentage error (MAPE) serving as the metric. To assess the specialized designs in GNN-PT-Map, we select four baselines for comparisons: a) the MII-based model in PBP; b) GNN-b, which uses only the basic features in G_{sw} and G_{hw}; c) GNN-c, which has no Kronecker or Hadamard product for feature alignment (similar with the graph feature extraction in MapZero [10]); and d) GNN-e, which directly predicts computation cycles without the three sub-tasks. Fig. 6 presents the results: the MII-based analytical model performs unstable when loop unrolling is involved or when dealing with heterogeneous PEs and non-mesh topologies. Even for the regular S4, the MAPE remains at 24.7%. In contrast, the GNN-based models outperform PBP significantly. And GNN-PT-Map further achieves the lowest MAPEs below 8.5% across all cases. This high accuracy enables PT-Map to identify promising candidates.

• **Performance:** Fig. 7 shows the performance comparisons of PT-Map (in the performance ranking mode) and the baselines. PT-Map's speedups to the baselines are marked in the corresponding colors. Compared to RAMP, LISA and MapZero achieve their speedups by exploring the loop pipelining better, while IP and PBP achieve the speedups by exploring loop interchange and loop fusion/fission in their inter-loop transformation spaces. In contrast, PT-Map consis-

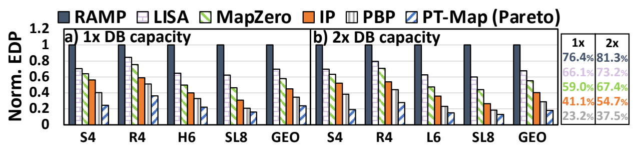


Figure 8: EDP reduction with different DB capacities.

Table 6: Normalized performance comparisons on SL8.

App.	GEM	TRI	COV	DOI	TMM	ATA	BLU	HAR	CON	TCO	WIN	GEO
RAMP	0.41	0.37	0.14	0.25	0.13	0.53	0.13	0.16	0.17	0.18	0.21	0.22
AL	0.49	0.53	0.19	0.47	0.86	0.58	0.46	0.22	0.88	0.24	0.28	0.41
AM	0.88	0.76	fail	fail	0.82	0.88	0.79	fail	0.89	0.92	fail	–
PT-Map	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

tently discovers promising inter/intra-loop transformation opportunities that achieve better performance on all architectures, though it uses the slightly weaker loop scheduler than LISA and MapZero. Specifically, PT-Map achieves geomean speedups ranging from 1.65×/1.54× (on R4) to 3.45×/2.95× (on SL8) compared to LISA and MapZero, respectively. Besides, PT-Map achieves 1.52×/1.41× (on R4) to 2.49×/ 1.80× (on SL8) geomean speedups compared to IP and PBP, respectively. Thanks to the detailed in-PNL exploration and the accurate evaluation, PT-Map achieves higher speedups on larger CGRAs and maximizes its advantages in the applications (e.g., BLU and TMM) with adequate unrollable loop dimensions. While the applications with ample fusion choices (e.g., HAR and WIN) narrow the performance gap between PT-Map and PBP. Compared with PBP exploring the loop fusion, the multi-dimensional loop unrolling in PT-Map has a more direct impact on spatial granularity within one PNL and resource utilization. The only abnormal case is ATA on SL8; due to an inaccurate prediction, a sub-optimal candidate is selected, resulting in 9.1% lower performance to PBP.

• **Energy efficiency:** We also evaluate the energy-delay-product (EDP) comparisons (PT-Map in Pareto mode). To demonstrate the awareness of data access in PT-Map, we conduct additional experiments with the doubled DB capacity. For a fair comparison, the same memory profiling method and ranking metric $PVol(\cdot)$ are employed for IP and PBP. Fig. 8 shows the results: with the original capacity, PT-Map achieves geomean EDP reductions of 76.4%/66.1%/59.0% compared to RAMP/LISA/MapZero, respectively. It also achieves 41.1%/23.2% EDP reductions compared to IP and PBP, respectively. With the doubled capacity, PT-Map achieves even higher reduction ratios: 81.3%/73.2%/67.4%/54.7%/37.5% compared to RAMP/LISA/MapZero/IP/PBP, respectively. LISA and MapZero achieve reductions by decreasing the delay, while IP and PBP lack explicit tiling exploration and introduce side effects on memory access. They all lack the data access awareness for energy efficiency optimization. In contrast, PT-Map generates various tiled loops in out-PNL/in-PNL explorations, resulting in efficient data access and proper temporal granularity for different DB capacities.

• **Ablation study:** We replace the exploration or the evaluation methods in PT-Map to assess their effectiveness on SL8: a) AL: using the active learning tool, OpenTuner [3], to tune the performance within the space defined in Tab. 1 in four hours; b) AM: using the MII-based model in PBP for performance profiling. Tab. 6 lists the normalized performance comparisons. AL exhibits volatile fluctuations during the search process and generates many illegal transformations. Therefore, the overall mapping quality is unstable, particularly for programs with large #PNLs. As for AM, the MII-based model favors transformations with coarser spatial granularity (large unroll factors), and the uncertainty of the actual mapped IIs causes mapping infeasibility. In the four cases with large #PNLs, all top-20 candidates contain unmappable innermost loops. The

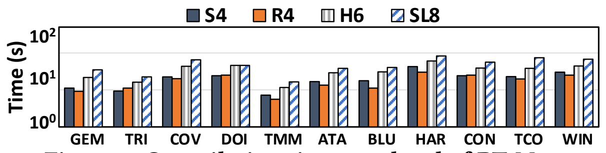


Figure 9: Compilation time overhead of PT-Map.

ablation study proves that both the top-down exploration and the GNN-assisted bottom-up evaluation are essential in PT-Map.

• **Generality:** We evaluate PT-Map on an unseen 4 × 4 HReA-like [14] CGRA architecture. The GNN model is fine-tuned using only 400 random programs to enhance accuracy. With the fine-tuned model, PT-Map achieves 2.94×/2.24×/1.58× geomean speedups and 60.2%/48.6%/26.7% compared to MapZero/IP/PBP, respectively. It shows the good generality and adaptability of PT-Map.

• **Compilation time:** Fig. 9 shows the compilation time of PT-Map. Due to the significantly smaller spaces of IP and PBP compared to PT-Map, we have not included their compilation time here. PT-Map's compilation time is on the order of $10^1 \sim 10^2$ seconds. The longest case is mapping HAR on SL8 (83.4 seconds). As the array size increases, the granularity of mappable loops becomes coarser, necessitating the evaluation of more feasible transformations. Given the inherent complexity of the program transformation optimization problem, we claim that these time overheads are acceptable, and the time is also much shorter than the ILP-based or SA-based loop schedulers taking hours to get a feasible solution. In the future, advanced parallelizations can be explored for further reduction.

5 CONCLUSION

This paper presents PT-Map, a novel CGRA mapping framework for program transformation optimization. The top-down exploration in the comprehensive transformation space and the bottom-up GNN-assisted performance profiling make PT-Map outperform the state-of-the-art approaches in both performance and energy efficiency.

ACKNOWLEDGEMENT

This work was partly supported by the National Natural Science Foundation of China (Grant No. 62090021) and the National Key R&D Program of China (Grant No. 2022YFB4500500).

REFERENCES

[1] U. Bondhugula et al. 2008. A practical automatic polyhedral parallelizer and locality optimizer. In *PLDI*.
[2] S. Dave et al. 2018. RAMP: Resource-aware mapping for CGRAs. In *DAC*.
[3] J. Ansel et al. 2014. OpenTuner: An Extensible Framework for Program Autotuning. In *PACT*.
[4] Y. Guo et al. 2020. Pillars: An Integrated CGRA Design Framework. In *WOSET*.
[5] T. Gysi et al. 2019. A fast analytical model of fully associative caches. In *PLDI*.
[6] L. Huang et al. 2023. Optimizing Data Reuse for CGRA Mapping Using Polyhedral-based Loop Transformations. In *DAC*.
[7] M. Karunaratne et al. 2017. HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect. In *DAC*.
[8] D. Kingma et al. 2015. Adam: A method for stochastic optimization. In *ICLR*.
[9] T. Kipf et al. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[10] X. Kong et al. 2023. MapZero: Mapping for Coarse-grained Reconfigurable Architectures with Reinforcement Learning and Monte-Carlo Tree Search. In *ISCA*.
[11] Z. Li et al. 2022. LISA: Graph Neural Network based Portable Mapping on Spatial Accelerators. In *HPCA*.
[12] D. Liu et al. 2013. Polyhedral model based mapping optimization of loop nests for CGRAs. In *DAC*.
[13] D. Liu et al. 2021. Polyhedral-based Pipelining of Imperfectly-Nested Loop for CGRAs. In *ICCAD*.
[14] L. Liu et al. 2017. HReA: An energy-efficient embedded dynamically reconfigurable fabric for 13-dwarfs processing. *IEEE TCAS-II* (2017).
[15] B. Mei et al. 2003. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *FPL*.
[16] P. Veličković et al. 2018. Graph attention networks. In *ICLR*.
[17] S. Yin et al. 2015. Joint affine transformation and loop pipelining for mapping nested loop on CGRAs. In *DATE*.