

DG-RePlace: A Dataflow-Driven GPU-Accelerated Analytical Global Placement Framework for Machine Learning Accelerators

Andrew B. Kahng, *Fellow, IEEE* and Zhiang Wang, *Student Member, IEEE*

Abstract—Global placement is a fundamental step in VLSI physical design. The wide use of 2D processing element (PE) arrays in machine learning accelerators poses new challenges of scalability and Quality of Results (QoR) for state-of-the-art academic global placers. In this work, we develop *DG-RePlace*, a new and fast GPU-accelerated global placement framework built on top of the OpenROAD infrastructure [24], which exploits the inherent dataflow and datapath structures of machine learning accelerators. Experimental results with a variety of machine learning accelerators using a commercial 12nm enablement show that, compared with *RePlace* (*DREAMPlace*), our approach achieves an average reduction in routed wirelength by 10% (7%) and total negative slack (TNS) by 31% (34%), with faster global placement and on-par total runtimes relative to *DREAMPlace*. Empirical studies on the *TILOS MacroPlacement Benchmarks* [26] further demonstrate that post-route improvements over *RePlace* and *DREAMPlace* may reach beyond the motivating application to machine learning accelerators.

I. INTRODUCTION

Global placement is a fundamental step in VLSI physical design that determines the locations of standard cells and macros in a layout. The backend design closure flow requires a fast placement engine for rapid design prototyping, feeding back to synthesis, and guiding optimization. However, emerging machine learning accelerators have introduced new challenges for global placement. On the one hand, machine learning accelerators with millions of standard cells and macros raise runtime concerns for the design closure process. On the other hand, machine learning accelerators featuring 2D processing element (PE) arrays, such as systolic arrays [21], have gained prominence because of their efficiency in convolutional neural network computations [8]. The dataflow and datapath architectures of these machine learning accelerators exhibit substantial differences compared to those of traditional datapath designs, requiring dedicated treatment during global placement to achieve decent Quality of Results (QoR).

To address aspects of the aforementioned challenges, several global placers have been proposed over the past decades. To improve runtime, researchers have focused on parallelizing global placement algorithms to leverage the computational substrates provided by multi-core CPUs and GPUs. [7] introduces a multi-threaded shared-memory implementation of *RePlace* [3] using off-the-shelf multi-core CPU hardware. [6] and [13] propose GPU-accelerated analytical placers by parallelizing the computation of the Logarithm-Sum-Exponential (LSE) wirelength function as well as the density function. Recently, *DREAMPlace* [14], [17] and *Xplace* [18] have

implemented the approach of *RePlace* on GPU by casting the placement problem as a neural network training problem; these works demonstrate the superiority of GPU-accelerated global placers. While *DREAMPlace* has already achieved significant runtime improvement relative to *RePlace*, our aim is to push these boundaries even further by leveraging optimized data structures and a new parallel wirelength gradient computation algorithm.

Other pioneering works exploit the dataflow and datapath structures during macro placement and global placement. [22] and [9] exploit RTL information and dataflow to guide macro placement. [16] integrates the dataflow information into the mixed analytical global placement framework through virtual objects. Furthermore, [5] introduces the first global placement framework that exploits the datapath regularity of 2D PE arrays. In our present work, we propose a new, fast GPU-accelerated global placement framework, exploiting both dataflow information and datapath regularity. Our approach ultimately guides global placement towards better Quality of Results (QoR). The main contributions of this paper are as follows.

- We propose *DG-RePlace*, a new and fast global placer that leverages the intrinsic dataflow and datapath structures of machine learning accelerators, to achieve high-quality global placement.
- *DG-RePlace* is built on top of the OpenROAD infrastructure with a permissive open-source license, enabling other researchers to readily adapt it for other enhancements.¹
- We propose efficient data structures and algorithms to further speed up the global placement. Experimental results on a variety of machine learning accelerators show that, our approach is respectively on average 22.49X and 1.75X faster than *RePlace* and *DREAMPlace* in terms of global placement runtime. Overall turnaround time is on par with that of *DREAMPlace*, despite (one-time) file IO runtime overheads that are due to OpenDB/OpenROAD integration.
- Experimental results using a variety of machine learning accelerators show that, compared with *RePlace* and *DREAMPlace*, our approach achieves an average reduction in routed wirelength by 10% and 7%, and total negative slack by 31% and 34%, respectively.
- Experimental results on the two largest *TILOS MacroPlacement Benchmarks* [26] testcases show that compared with *RePlace* and *DREAMPlace*, *DG-RePlace* achieves much

¹The source codes are available in the *DG-RePlace* GitHub repository [34].

better timing metrics (WNS and TNS) measured post-route optimization. This suggests that the proposed dataflow-driven methodology is not limited to machine learning accelerators.

The remaining sections are organized as follows. Section II introduces the terminology and background. Section III discusses our approach. Section IV shows experimental results, and Section V concludes the paper and outlines future research directions.

II. PRELIMINARIES

In this section, we begin by discussing the fundamentals of the systolic array structure in Section II-A. Following this, we delve into the electrostatics-based placement formulation in Section II-B, which is incorporated into our DG-RePIAce. Finally, we examine previous work on dataflow-driven placement in Section II-C. Table I summarizes important terms and their meanings; for clarity, we give 1-dimensional (x component) notation.

TABLE I: Terminology and Notation

Notation	Description
v	instance (standard cell or macro)
p	instance pin or input-output pin
e	net $e = \{p\}$
V	Set of instances $\{v\}$
E	Set of nets (hyperedges) $\{e\}$
P	Set of pins $\{p\}$
$WL_{grad_x}(p)$	Wirelength gradient on pin p
x_p	x coordinate of pin p
x_e^+	$\max_{i \in e} x_i, \forall e \in E$
x_e^-	$\min_{i \in e} x_i, \forall e \in E$
a_i^+	$\exp(\frac{x_i - x_e^+}{\gamma}), \forall i \in e, e \in E$
a_i^-	$\exp(-\frac{x_i - x_e^-}{\gamma}), \forall i \in e, e \in E$
b_e^+	$\sum_{i \in e} a_i^+, e \in E$
b_e^-	$\sum_{i \in e} a_i^-, e \in E$
c_e^+	$\sum_{i \in e} x_i a_i^+, e \in E$
c_e^-	$\sum_{i \in e} x_i a_i^-, e \in E$
X_v	Instance location, $\forall v \in V$
$FWL_x(v)$	Wirelength force on instance v
PU	Processing unit
PE	Processing element

A. Systolic Array Structure

A systolic array [4] is a 2D array of $M \times N$ processing elements (PEs), which performs massively parallel convolution and matrix multiplication operations. A PE is often composed of a MAC (Multiply-Accumulate) unit and registers, and PEs that are aligned in the same row collectively form a *processing unit* (PU). Figure 1 shows an example execution flow of a systolic array-based machine learning accelerator. Here, the input data horizontally propagate through the PEs, are multiplied by the weights in each PE, and then are accumulated vertically along the columns of the systolic array. This structure restricts data transfers (multiple bitwidth) to neighboring PEs, thus achieving better performance and efficiency.

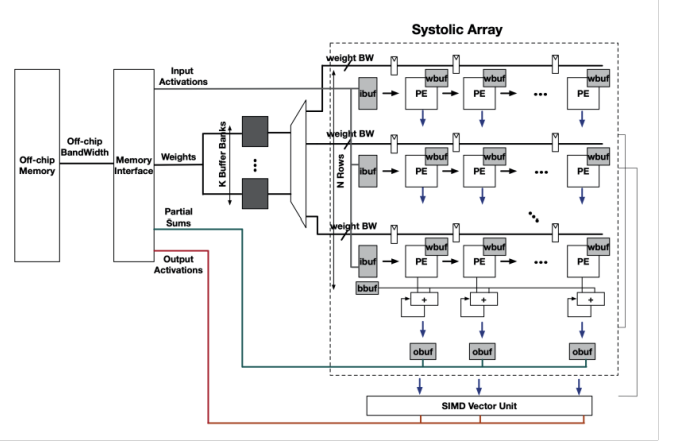


Fig. 1: Illustrative execution flow of a systolic array-based machine learning accelerator (figure reproduced from [4]).

B. Electrostatics-Based Placement

State-of-the-art academic global placers, such as *RePIAce* [3] and *DREAMPlace* [14], [17], usually adopt the electrostatics-based placement approach [12]. Let $(X_v, Y_v)^T$ denote the vectors of x-y coordinates of movable instances. The electrostatics-based placers formulate the global placement problem as follows:

$$\min_{X_v, Y_v} \sum_{e \in E} WL(e; X_v, Y_v) + \lambda \times D(X_v, Y_v) \quad (1)$$

where $WL(\cdot; \cdot)$ is the wirelength cost function, $D(\cdot)$ is the instance density cost function and λ is the weighting factor. In this work, we use the weighted-average wirelength (WA) as the wirelength cost function, where the x-component of WA for net e is given by

$$WL_e = \frac{\sum_{i \in e} x_i \cdot \exp(\frac{x_i}{\gamma})}{\sum_{i \in e} \exp(\frac{x_i}{\gamma})} - \frac{\sum_{i \in e} x_i \cdot \exp(-\frac{x_i}{\gamma})}{\sum_{i \in e} \exp(-\frac{x_i}{\gamma})} \quad (2)$$

where γ is a parameter that controls the smoothness and accuracy of the approximation to the half-perimeter wirelength (HPWL). With the notations in Table I, the gradient of WA wirelength to a pin location x_i is given as follows:

$$\frac{\partial WL_e}{\partial x_i} = \frac{(1 + \frac{x_i}{\gamma})b_e^+ - \frac{1}{\gamma}c_e^+}{(b_e^+)^2} \cdot a_i^+ - \frac{(1 - \frac{x_i}{\gamma})b_e^- + \frac{1}{\gamma}c_e^-}{(b_e^-)^2} \cdot a_i^- \quad (3)$$

C. Dataflow-Driven Placement

To obtain high-quality macro placement, human designers usually rely on their understanding of the dataflow of a design to determine the relative locations of macros. However, this manual process is very time-consuming, often takes several days to weeks to complete. To automate this process, Vidal-Obiols et al. [22] and *Hier-RTLMP* [10] introduce dataflow-driven multilevel macro placement approaches. However, both approaches apply the Simulated Annealing [11] algorithm to determine the locations of macros, resulting in poor runtime scalability [1]. Lin et al. [15] presents an analytical-based

placement algorithm to handle dataflow constraints in mixed-size circuits. Their approach initially assigns larger weights to nets connecting to datapath-oriented objects, and then gradually shrinks the weights according to the status of placement utilization.² However, their method requires dataflow constraints from designers and cannot handle the unique datapath regularity in machine learning accelerators (see Section III-C). In this work, we incorporate the physical hierarchy extraction approach in *Hier-RTLMP* [10] into the global placement framework to capture the dataflow information during global placement. Besides, we pay special attention to the datapath regularity in machine learning accelerators during placement.

III. OUR APPROACH

The architecture of our *DG-RePLAcE* framework is shown in Figure 2. The input is a synthesized hierarchical gate-level netlist and a floorplan .def file that contains the block outline and fixed IO pin or pad locations. The output is a .def file with placed macros and standard cells. *DG-RePLAcE* is built on top of the open-source OpenROAD infrastructure [24], and consists of four major steps.

- **Physical Hierarchy Extraction** (Section III-A): During this step, we convert the structural netlist representation of the RTL design into a clustered netlist. The instances within the same cluster are expected to remain close to each other during global placement.
- **Dataflow-Driven Initial Global Distribution** (Section III-B): During this step, we insert the dataflow information into the clustered netlist, and determine the location for each cluster through our new *GPU-accelerated parallel analytical placement* method. Then, every instance within a cluster is positioned at the cluster's center. Furthermore, we incorporate pseudo net constraints into the original netlist, ensuring that instances belonging to the same cluster are placed in close proximity to each other.
- **Datapath Constraints Construction** (Section III-C): This step extracts datapath information from the original netlist. Following this extraction, we transform the datapath information into pseudo net constraints.
- **Parallel Analytical Placement** (Section III-D): At this step, we execute the GPU-accelerated mixed-size global placement on the original gate-level netlist, integrating the pseudo net constraints derived from the *Dataflow-Driven Initial Global Placement* and the *Datapath Constraints Construction* steps. Here, we leverage parallel processing capabilities of GPU to accelerate the Nesterov's method in *RePLAcE*.

A. Physical Hierarchy Extraction

During this step, we transform the original logical hierarchy into a physical hierarchy. Much like the logical hierarchy, which is composed of logical modules, the physical hierarchy consists of physical clusters. In contrast to logical modules,

²[15] has reported an excellent dataflow-driven analytical mixed-size placer. Unfortunately, no testcases or executables can be released by their group.

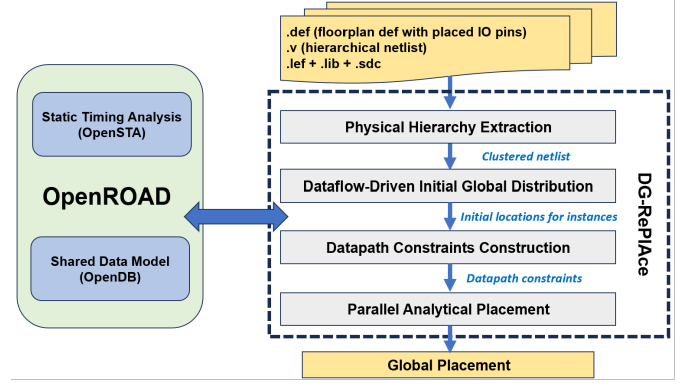


Fig. 2: Overview of the proposed *DG-RePLAcE* flow.

a physical cluster consists of instances that are expected to remain close to each other during global placement. Specifically, we employ the *Multilevel Autoclustering* component of the open-source *Hier-RTLMP* [10] to perform physical hierarchy extraction.³ Upon establishing the physical hierarchy, we convert the original gate-level netlist into a clustered netlist.

B. Dataflow-Driven Initial Global Distribution

In this section, we first describe how to insert dataflow information into the clustered netlist. Then, we explain how we use the *GPU-accelerated parallel analytical placement* framework to distribute the clustered netlist evenly, and how we solve the divergence issue by applying a bloat factor to each cluster. Finally, we discuss how we use the placed clustered netlist to guide the global placement process.

After physical hierarchy extraction, we have a clustered netlist in which the nodes are clusters and the nets are bundled connections. We then insert the dataflow information into the clustered netlist through virtual connections. The dataflow describes the way in which data moves between different functional units of a netlist. The dataflow can be visualized as the high-level conceptual movements of data and how they are processed step by step. Figure 4 shows the dataflow visualization of the Tabla01 design (see Section IV for details of this and other testcases). When backend engineers perform the place-and-route (P&R) flow for a netlist, understanding the dataflow is critical for optimizing power, performance and area (PPA), as the dataflow determines how the netlist is pipelined and how the parallel processing is implemented. We adopt the same idea as [22], [9], [10] and transform the dataflow information into *virtual connections* between clusters. The virtual connections ($Virtual_Conn(A, B)$) between clusters A and B are defined as

$$Virtual_Conn(A, B) = \frac{Info_Flow(A, B)}{2^{Num_Hops}} \quad (4)$$

Here, *Info_Flow* corresponds to connection bandwidth and *Num_Hops* is the length of the shortest path of registers between clusters. When calculating the virtual connections between clusters, we follow the same convention as [9], [10]. If

³The detailed algorithm is presented as Algorithm 2 in [10]. The source codes are available in [34].

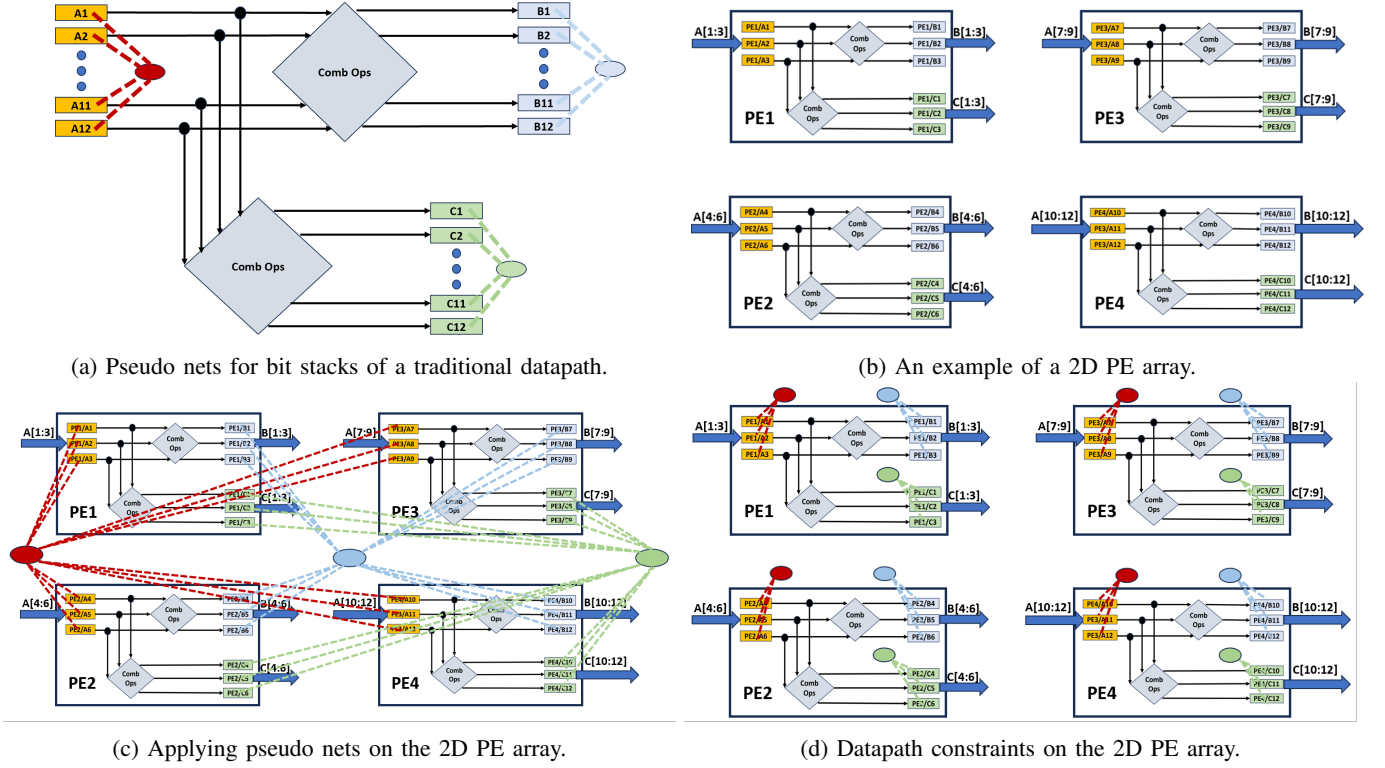


Fig. 3: Datapath constraints construction on the 2D PE array.

the register distance (Num_Hops) between clusters is greater than 4, then no virtual connection is added. In the example shown in Figure 4, if the register distance between $PU0$ and $Output\ Buffer$ is 2, then the calculated virtual connections are 16, given that the connection bitwidth is 64 bits.

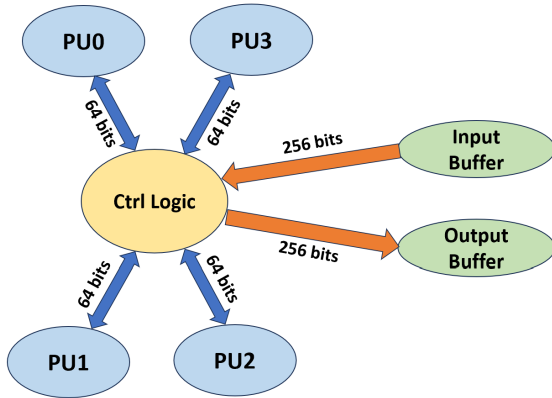


Fig. 4: Dataflow visualization of the Tabla01 design [4].

Upon incorporating the dataflow information into the clustered netlist, we call the *GPU-accelerated parallel analytical placement* framework, for which details are given in Section III-D, to evenly distribute the clustered netlist. However, directly working on the clustered netlist could lead to divergence issues, particularly if the layout has a large amount of whitespace. To solve the divergence issues, we introduce a bloat-shrink methodology guided by the final density overflow

of the cluster placement (refer to Equation (37) in [12] for the definition). This methodology includes two steps:

- **Bloat:** We first bloat each cluster by applying a bloat factor ($Bloat_Factor$), defined as

$$Bloat_Factor = \frac{\text{Area of placement region}}{\text{Total area of clusters}} \quad (5)$$

- **Shrink:** If the cluster placement diverges and ends with the density overflow $cluster_overflow$, we then shrink each cluster using a shrink factor ($Shrink_Factor$). $Shrink_Factor$ is determined by dividing the target density overflow ($target_overflow$) by the actual density overflow ($cluster_overflow$):

$$Shrink_Factor = \frac{target_overflow}{cluster_overflow} \quad (6)$$

Here, $target_overflow$ ($target_overflow = 0.2$ by default for cluster placement) is the convergence criterion for the Nesterov's approach.

After completing the placement of clusters, we place the instances within each cluster at the cluster's center to obtain a good initial placement. Furthermore, for each cluster, we add one pseudo net that connects all of the instances within the cluster. This ensures that instances belonging to the same cluster are placed in close proximity to each other. However, we notice that these high-fanout pseudo-nets could cause convergence problems. To address this issue, we transform the pseudo nets into multiple two-pin nets by the star model (i.e., by adding a pseudo vertex as the star's center, per cluster). To ensure that the global placer follows the pseudo net constraints imposed by the clustering constraints, we initially assign a

high penalty factor $Penalty_Factor_p$ to the pseudo nets, starting at the value of $Penalty_Factor_{p0}$. With each successive iteration, the penalty factor is progressively decreased to allow for a more even distribution of instances across the placement region. The adjustment of the penalty factor $Penalty_Factor_p$ is determined by the following equation:

$$Penalty_Factor_{p0} = \exp(iter_0) \quad (7)$$

$$Penalty_Factor_p = \frac{Penalty_Factor_{p0}}{\exp(iter)} \quad (8)$$

where $iter$ is the current iteration number, and $iter_0$ ($iter_0 = 4$ by default) is used to determine the initial value. To determine the default value of $iter_0$, we study $iter_0$ values ranging from 0 to 9, utilizing Tabla01 and Tabla02 (see Table II) as testcases. The score for our evaluation is the routed wirelength, which we normalize against the baseline results obtained from *RePlace*. Based on this experiment, we use $iter_0 = 4$ as the default.

C. Datapath Constraints Construction

After capturing the dataflow between clusters, we examine the detailed data movement within each cluster, i.e., datapath information. The datapath refers to the actual hardware components and interconnections that implement the dataflow, representing the paths that data traverse in a digital design. More specifically, the datapath is the circuit performing bit-wise data operations in parallel on multiple bits [2]. Each operation corresponds to a dedicated functional block, such as adder, register, buffer, multiplexer, multiplier, etc. Fang et al. [5] further point out that there is a significant difference between the datapath within a systolic array and that of traditional datapath designs, as shown in Figure 3.

Figure 3(a) shows the datapath in traditional datapath designs, characterized by a continuous bit-sliced structure for operations across different bits [2]. In such scenarios, a pseudo net can be applied to each alignment group (i.e., A[1:12], B[1:12] and C[1:12]), ensuring that the instances in each alignment group are placed in proximity. In contrast, as depicted in Figure 3(b), the datapath in a systolic array is not continuous, with operations for different bits across multiple PEs. This may lead to overlaps of PEs when a pseudo net is directly applied to each alignment group, as shown in Figure 3(c). To address this issue, we propose to assign pseudo nets to local alignment groups within each cluster. For example, in Figure 3(d), pseudo nets are independently applied to A[1:3], B[1:3] and C[1:3] within PE1. Here we also transform the pseudo nets into multiple two-pin nets by the star model. To maintain the integrity of local connectivity, we set the initial penalty factor $Penalty_Factor_{p0}$ to 1 for the pseudo nets induced by datapath constraints.

D. Parallel Analytical Placement

Our GPU-accelerated mixed-size parallel analytical placement framework uses the same Nesterov's method as *RePlace*, and is developed on top of the OpenROAD infrastructure. To minimize memory overhead, we integrate a data structure inspired by Gessler et al. [7], which optimizes data locality for

the frequently accessed components during global placement. As pointed out by [13], the fast computation of wirelength gradient and bin density is crucial for the efficiency of the global placer. We adopt the parallel bin density computation algorithm from Gessler et al. [7] (Algorithm 2 in [7]). For the fast computation of wirelength gradient, we introduce a novel parallel algorithm, presented in Algorithm 1. Our algorithm distinguishes itself from Algorithm 1 in *DREAMPlace* [14] primarily in *Lines 1-6* and *Lines 12-18*, where we leverage net-level parallelization rather than pin-level parallelization to eliminate the need for atomic additions. Furthermore, it differs from Algorithm 2 in *DREAMPlace* [14] primarily in *Lines 7-11* and *Lines 19-22*, where we implement pin-level computation parallelization with multiple threads rather than the sequential computation within a single thread. This approach is more efficient for managing high-fanout nets while maintaining comparable efficiency in handling low-fanout nets (see Section IV-C for details). Empirical results demonstrate that our algorithm is approximately 3.25X faster than the one implemented in *DREAMPlace* (Algorithm 2 in [14]).

Algorithm 1: Parallel Wirelength Gradient Computation.

Input: Instances V , Nets E , Pins P and Instance locations X_v
Output: Wirelength force for each instance $F_{WL_x}(v)$

```

1 for each thread  $0 \leq t < |E|$  do
2   Define  $e$  as the net corresponding to thread  $t$ ;
3    $x_e^+ \leftarrow \max_{p \in e} x_p$ ;  $\blacktriangleright x_e^+$  is in the global memory
4    $x_e^- \leftarrow \min_{p \in e} x_p$ ;  $\blacktriangleright x_e^-$  is in the global memory
5    $b_e^\pm \leftarrow 0$ ;  $c_e^\pm \leftarrow 0$ ;  $\blacktriangleright b_e^\pm, c_e^\pm$  are in the global memory
6 end
7 for each thread  $0 \leq t < |P|$  do
8   Define  $p$  as the pin corresponding to thread  $t$ ;
9   Define  $e$  as the net that pin  $p$  belongs to;
10   $a_p^\pm \leftarrow e^\pm \frac{x_p - x_e^\pm}{\gamma}$ ;  $\blacktriangleright a_p^\pm$  is in the global memory
11 end
12 for each thread  $0 \leq t < |E|$  do
13   Define  $e$  as the net corresponding to thread  $t$ ;
14   for pin  $p \in e$  do
15      $b_e^\pm \leftarrow b_e^\pm + a_p^\pm$ ;
16      $c_e^\pm \leftarrow c_e^\pm + x_p a_p^\pm$ ;
17   end
18 end
19 for each thread  $0 \leq t < |P|$  do
20   Define  $p$  as the pin corresponding to thread  $t$ ;
21   Compute the wirelength gradient of pin  $WL_{grad_x}(p)$ 
      using Equation (3);  $\blacktriangleright WL_{grad_x}(p)$  is in the global
      memory
22 end
23 for each thread  $0 \leq t < |V|$  do
24   Define  $v$  as the instance corresponding to thread  $t$ ;
25    $F_{WL_x}(v) \leftarrow 0.0$   $\blacktriangleright F_{WL_x}(v)$  is in the global memory
26   for pin  $p$  of  $v$  do
27      $F_{WL_x}(v) \leftarrow F_{WL_x}(v) + WL_{grad_x}(p)$ ;
28   end
29 end
30 return  $F_{WL_x}(v)$ 

```

IV. EXPERIMENTAL RESULTS

DG-RePlace is implemented with approximately 14K lines of C++ (and CUDA) with a Tcl command line interface on top of the OpenROAD infrastructure [24]. We run all experiments on a Linux server with an Intel Xeon E5-2690 CPU (48 threads) with 256 GB RAM and an NVIDIA TITAN V GPU.

To show the effectiveness of our global placer, the following three scenarios are evaluated and compared.

- *RePlace*: Global placement is done by *RePlace*, which is the default global placer in the OpenROAD project [24].
- *DREAMPlace*: Global placement is performed by the latest version of *DREAMPlace* [25], which is the state-of-the-art GPU-accelerated global placer.⁴
- *DG-RePlace*: Results are obtained using our global placer.

Our experiments use the following flow. (1) We first synthesize the design using a state-of-the-art commercial synthesis tool, preserving the logical hierarchy. (2) Next, we determine the core size of the testcase and place all the IO pins using a manually-developed script. (3) Then, the global placement is performed using different methods (*RePlace*, *DREAMPlace* and *DG-RePlace*). (4) Finally, we use a state-of-the-art commercial P&R tool, Cadence Innovus 21.1, to finish the legalization of macros, detailed placement of standard cells and routing. We follow the SP&R scripts in the public *MacroPlacement repository* [26]. All metrics are collected after post-route optimization. All studies use a commercial foundry 12nm technology (13 metal layers) with cell library and memory generators from a leading IP provider.

In this section, we first present the results for two types of machine learning accelerators: non-DNN machine learning accelerators (Tabla designs) and DNN machine learning accelerators (GeneSys designs), detailed in Section IV-A. Then, we study the respective effects of dataflow and datapath constraints by conducting an ablation study of *DG-RePlace*, in Section IV-B. Next, we discuss the runtime comparison between *DG-RePlace* and *DREAMPlace* in Section IV-C. In Section IV-D, we compare *DG-RePlace* with the dataflow-driven macro placer *Hier-RTLMP*, which uses the same method to perform physical hierarchy extraction. Lastly, we apply *DG-RePlace* to large non-machine learning test cases in Section IV-E, which demonstrates the versatility and potential benefit of the proposed dataflow-driven approach, beyond our motivating application context of large-scale machine learning accelerators.

A. Results on Machine Learning Accelerators

We have validated our global placer using two types of machine learning accelerators (Tabla and GeneSys) from the VeriGOOD-ML platform [4]. The Tabla accelerators are designed for training and inference for non-DNN machine learning algorithms, and the GeneSys accelerators are for DNN machine learning algorithms. Both Tabla and Genesys adopt the systolic array structure, thus each design has an

⁴The default hyperparameter settings that we use for *DREAMPlace* are from [27].

TABLE II: Benchmarks. “Macro Util” stands for macro utilization, which is defined as the total area of macros divided by the core area.

Designs	PE Array	# Macros	# Std Cells	# Nets	Macro Util
Tabla01	4 × 8	368	232K	252K	0.60
Tabla02	4 × 16	1232	441K	486K	0.59
Tabla03	8 × 8	760	372K	408K	0.58
Tabla04	8 × 16	2488	741K	830K	0.54
GeneSys01	16 × 16	368	986K	1056K	0.46
GeneSys02	16 × 16	368	1055K	1135K	0.52

TABLE III: Experimental results. We highlight best values of metrics in blue bold font. Data points for WL, Power, WNS and TNS are normalized.

Design	Global Placer	WL	Power	WNS	TNS	GP (s)	TAT (s)
Tabla01	<i>RePlace</i>	1.00	1.00	-0.180	-81.349	150	204
	<i>DREAMPlace</i>	0.98	1.01	-0.151	-55.844	14	27
	<i>Hier-RTLMP</i>	1.10	1.00	-0.156	-74.929	-	2351
	<i>DG-RePlace</i>	0.93	0.98	-0.180	-62.622	7	31
Tabla02	<i>RePlace</i>	1.00	1.00	-0.197	-22.695	374	476
	<i>DREAMPlace</i>	0.95	0.98	-0.188	-24.188	24	47
	<i>Hier-RTLMP</i>	1.12	1.02	0.160	-17.807	-	3613
	<i>DG-RePlace</i>	0.91	0.98	-0.187	-19.642	13	50
Tabla03	<i>RePlace</i>	1.00	1.00	-0.092	-43.136	279	364
	<i>DREAMPlace</i>	1.21	1.05	-0.154	-84.152	21	41
	<i>Hier-RTLMP</i>	0.98	0.99	-0.136	-88.578	-	3872
	<i>DG-RePlace</i>	0.88	0.97	-0.084	-14.910	16	47
Tabla04	<i>RePlace</i>	1.00	1.02	-0.174	-48.756	689	883
	<i>DREAMPlace</i>	0.83	0.96	-0.177	-62.990	47	81
	<i>Hier-RTLMP</i>	1.10	1.06	-0.281	-222.119	-	8418
	<i>DG-RePlace</i>	0.85	0.97	-0.219	-54.755	20	82
GeneSys01	<i>RePlace</i>	1.00	1.00	-0.191	-94.176	630	850
	<i>DREAMPlace</i>	0.89	0.98	-0.213	-101.598	61	101
	<i>Hier-RTLMP</i>	0.97	1.00	-0.110	-23.151	-	3134
	<i>DG-RePlace</i>	0.89	0.98	-0.162	-45.939	40	100
GeneSys02	<i>RePlace</i>	1.00	1.00	-0.132	-14.937	752	972
	<i>DREAMPlace</i>	0.89	0.97	-0.108	-14.979	64	112
	<i>Hier-RTLMP</i>	N/A					
	<i>DG-RePlace</i>	0.94	0.99	-0.062	-7.78	44	111

$m \times n$ PE array. The major characteristics of the testcases are summarized in Table II.

Table III shows the experimental results after completion of post-route optimization. Rows represent testcases and global placement flows, and columns give information on total routed wirelength, power, worst negative slack (WNS), total negative slack (TNS), runtime of global placement (GP) and turnaround time (TAT).⁵ The metrics are normalized to protect foundry IP: (i) wirelength and power are normalized to the *RePlace* results, and (ii) timing metrics (WNS and TNS) are normalized to the clock period which we leave unspecified.

We can observe the following conclusions.

- Our approach outperforms both *RePlace* and *DREAMPlace* in terms of routed wirelength, achieving average reductions of 10% and 7%, respectively.
- Our approach outperforms both *RePlace* and *DREAMPlace* in terms of TNS, achieving average reductions of 31% and 34%, respectively.
- Our approach achieves similar speedup as *DREAMPlace* in terms of total turnaround time, but our approach is

⁵The runtime of global placement refers to the runtime required to distribute the original netlist across the placement region using the Nesterov’s method. For *DREAMPlace*, we extract the relevant information from the following log file entry: “[INFO] DREAMPlace - non-linear placement takes xx seconds”.

about $1.75X$ faster than *DREAMPlace* in terms of global placement runtime.⁶ The detailed runtime analysis is presented in Section IV-C.

- For the Tabla03 design, our approach significantly outperforms both *RePlace* and *DREAMPlace* in all the metrics (wirelength, power and timing). We attribute this to *DG-RePlace*'s ability to identify the dataflow and datapath of the design, which enables it to generate the placement in accordance with dataflow and datapath constraints. Figure 5(c) shows the post-route layout of the Tabla03 design for the global placement generated by *DG-RePlace*. We can see that it perfectly matches the dataflow pattern illustrated in Figure 4. By contrast, in the layout from *DREAMPlace* (Figure 5(b)), we can see that PU2 (in orange) gets mixed up with other Processing Units (PUs), leading to a significant degradation in wirelength, power and performance.
- For the GeneSys02 design, our approach delivers significantly better timing compared to both *RePlace* and *DREAMPlace*. This is because *DG-RePlace* follows the dataflow pattern inherent in the GeneSys02 design, as shown in Figure 5(f). We also observe that the *Input Buffer* module (highlighted in purple) becomes mixed up with other modules when placed by *RePlace* (Figure 5(d)) and *DREAMPlace* (Figure 5(e)). While *DREAMPlace*'s solution generates significantly better wirelength, its power improvement is very limited. We attribute this to the GeneSys02 design's extreme macro dominance, where the leakage power and internal power constitute 67% of total power consumption. We leave how to achieve a better power and timing tradeoff as a direction for future work.

B. Ablation Study

To demonstrate the effect of dataflow and datapath constraints, we run an ablation study [31] by removing dataflow or datapath constraints to understand their respective contributions to the overall performance of *DG-RePlace*. We conduct two separate experiments using variants of *DG-RePlace*. The first variant, referred to as *DG-RePlace_{nf}*, is executed without the dataflow constraint. The second variant, designated as *DG-RePlace_{np}*, is executed without the datapath constraint. These modifications allow us to assess the individual contributions of each constraint. The experimental results are presented in Table IV. In this table, WL_{avg} and TNS_{avg} respectively represent the average normalized routed wirelength and average total negative slack over all the testcases in Table II, compared with those from *RePlace*. We observe that both *DG-RePlace_{nf}* and *DG-RePlace_{np}* generate better results than *RePlace* in terms of TNS, but *DG-RePlace* always generates the best results. This suggests that both dataflow and datapath constraints are important components of *DG-RePlace*.

⁶We notice that for testcases GeneSys01 and GeneSys02, *DREAMPlace* is about $25X$ faster than *RePlace*, which matches the results reported in [14].

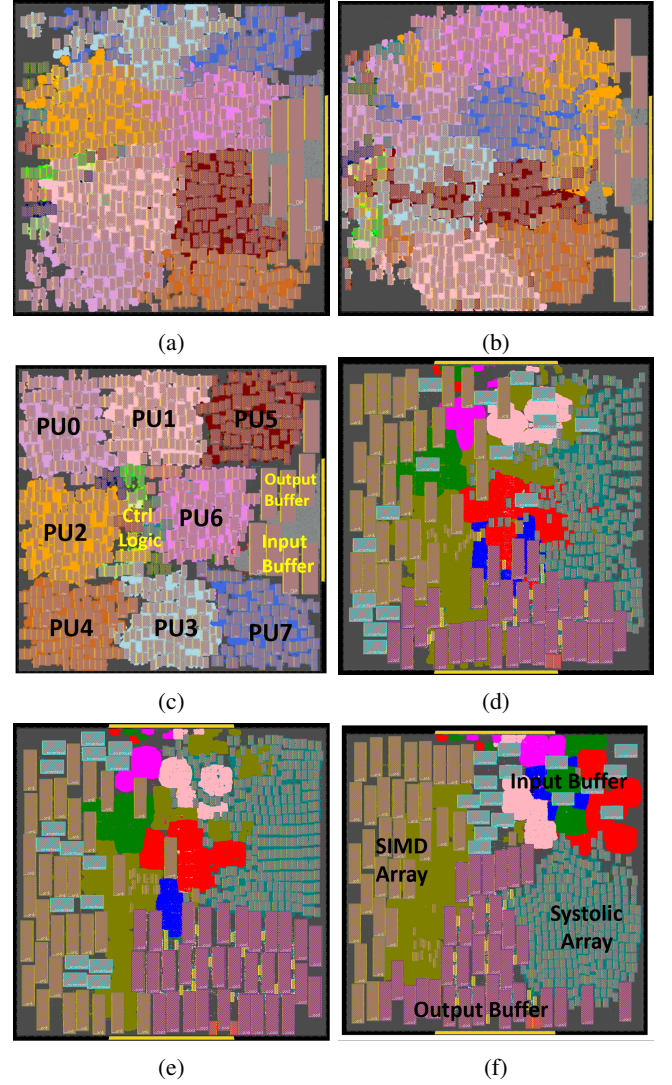


Fig. 5: Post-route layouts of Tabla3 and GeneSys02 designs with different flows. Design (Flow): (a) Tabla03 (*RePlace*); (b) Tabla03 (*DREAMPlace*); (c) Tabla03 (*DG-RePlace*); (d) GeneSys02 (*RePlace*); (e) GeneSys02 (*DREAMPlace*); (f) GeneSys02 (*DG-RePlace*). For the same design, each module maintains consistent coloring across different layouts.

TABLE IV: Effect of dataflow and datapath constraints (averages over all testcases). We highlight best values of metrics in blue bold font. Data points are normalized.

Metrics	<i>RePlace</i>	<i>DG-RePlace_{nf}</i>	<i>DG-RePlace_{np}</i>	<i>DG-RePlace</i>
WL_{avg}	1.00	0.92	0.91	0.90
TNS_{avg}	1.00	0.61	0.80	0.61

C. Runtime Comparison Against DREAMPlace

In this section, we compare the runtime of *DG-RePlace* against that of the leading GPU-accelerated global placer, *DREAMPlace*. As shown in Table III, the global placement runtime of *DG-RePlace* is less than that of *DREAMPlace*, while its overall turnaround time is similar. We will first discuss the global placement runtime, and then examine the overall turnaround time.

TABLE V: The iterations required for convergence of *RePlace*, *DREAMPlace* and *DG-RePlace*. We highlight best values in blue bold font.

Design	<i>RePlace</i>	<i>DREAMPlace</i>	<i>DG-RePlace</i>
Tabla01	410	450	380
Tabla02	460	546	390
Tabla03	460	507	380
Tabla04	520	687	490
GeneSys01	520	593	470
GeneSys02	510	598	450

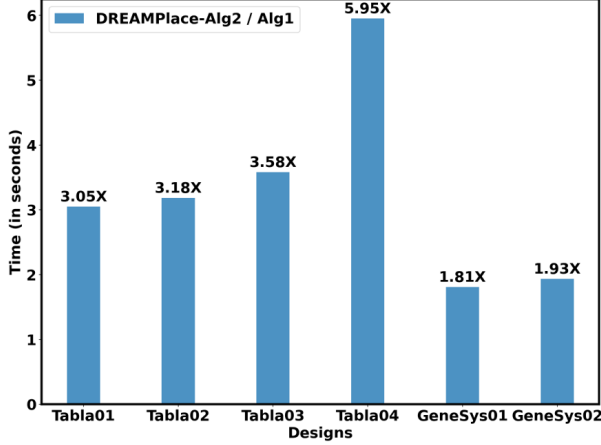


Fig. 6: Runtime comparison for different implementations of wirelength gradient computation.

The global placement runtime efficiency of *DG-RePlace* can be attributed to the following two factors.

- *DG-RePlace* achieves convergence with fewer iterations, due to an improved initial placement generated by our *Dataflow-Driven Initial Global Placement*. The iterations required for convergence of *RePlace*, *DREAMPlace* and *DG-RePlace* are presented in Table V.⁷ On average, *DG-RePlace* achieves convergence in 24% fewer iterations compared to *DREAMPlace*.
- Our parallel wirelength gradient algorithm (Algorithm 1) outperforms the one used by *DREAMPlace*, denoted as *DREAMPlace-Alg2*. For a fair comparison, we implement the wirelength gradient algorithm used by *DREAMPlace* (Algorithm 2 in [14]). The result is shown in Figure 6, which suggests that our algorithm is on average 3.25X faster. To further confirm that the increased runtime overhead of *DREAMPlace-Alg2* is due to high-fanout nets, we remove all nets connecting over 100 instances.⁸ After removing all of these high-fanout nets, *DREAMPlace-Alg2* achieves the same runtime as our Algorithm 1.

The longer overall turnaround time for *DG-RePlace* results from the *file IO for testcase reading and writing* and *physical hierarchy extraction*. Figure 7 shows the detailed runtime breakdown of *DG-RePlace*. We see that *file IO for testcase reading and writing* accounts for 50% of the overall turnaround time. This is due to complexity of the industry-

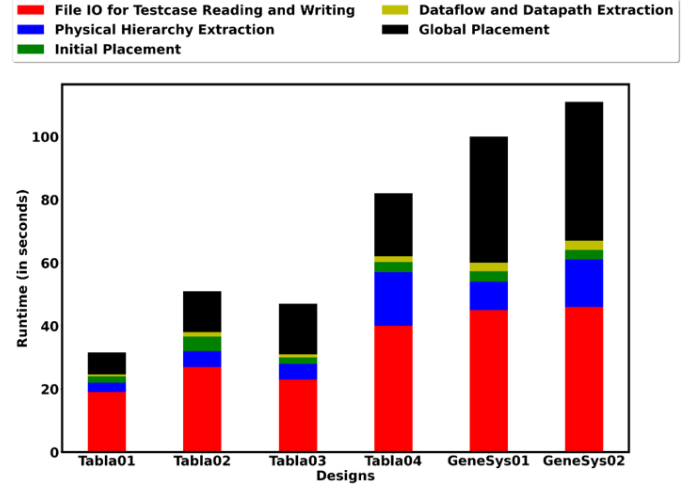


Fig. 7: Runtime breakdown of *DG-RePlace*.

strength database (OpenDB [32] in OpenROAD [24]) that we use, which brings increased loading times for designs. On the other hand, considering that the design is typically loaded just once, substituting *DG-RePlace* for *DREAMPlace* in scenarios where global placement is executed many times (e.g., $O(1000)$ times in *AutoDMP* [1]) will significantly improve runtimes for such scenarios.

D. Comparison with Hier-RTLMP

In this section, we compare *DG-RePlace* with the dataflow-driven multilevel macro placer *Hier-RTLMP* [10]. *Hier-RTLMP* uses the same physical hierarchy extraction approach and also considers dataflow information when determining locations of macros.⁹ The results are presented in Table III, and Figure 8 shows the post-route layouts. According to Table III, *DG-RePlace* achieves 15% wirelength reduction compared to *Hier-RTLMP*. *Hier-RTLMP* has worse wirelength because it models each cluster as a rectangular shape, as shown in Figure 8, potentially leading to unnecessary signal net detours.

Moreover, *Hier-RTLMP* fails to generate macro placement for the GeneSys02 design. *Hier-RTLMP* uses the Sequence Pair [19] representation and Simulated Annealing [11] algorithm to determine shapes and locations for clusters level by level. Therefore, it may not be able to obtain a feasible solution when it tries to place macros within a cluster whose location and shape have been determined in the previous step. Additionally, as has been pointed in [1], the use of Simulated Annealing algorithm in *Hier-RTLMP* makes it suffer from poor runtime scalability. Figure 9 shows how the speedup achieved by *DG-RePlace* over *Hier-RTLMP* changes with the number of PUs (#PU) and the number of PEs per PU (#PE per PU) for Tabla designs. We see that when the total number of PEs increases from 32 (Tabla01) to 128 (Tabla04), the speedup provided by *DG-RePlace* over *Hier-RTLMP* increases from 76X to 103X. Such speedups are enabling for architects or front-end designers who seek to identify the optimal #PU

⁷The *stop_overflow* hyperparameter is 0.1 for *RePlace* [28], *DREAMPlace* [27] and *DG-RePlace*.

⁸*ignore_net_degree* is 100 by default in *DREAMPlace* [27].

⁹We use the latest version of *Hier-RTLMP* from the OpenROAD repository [33].

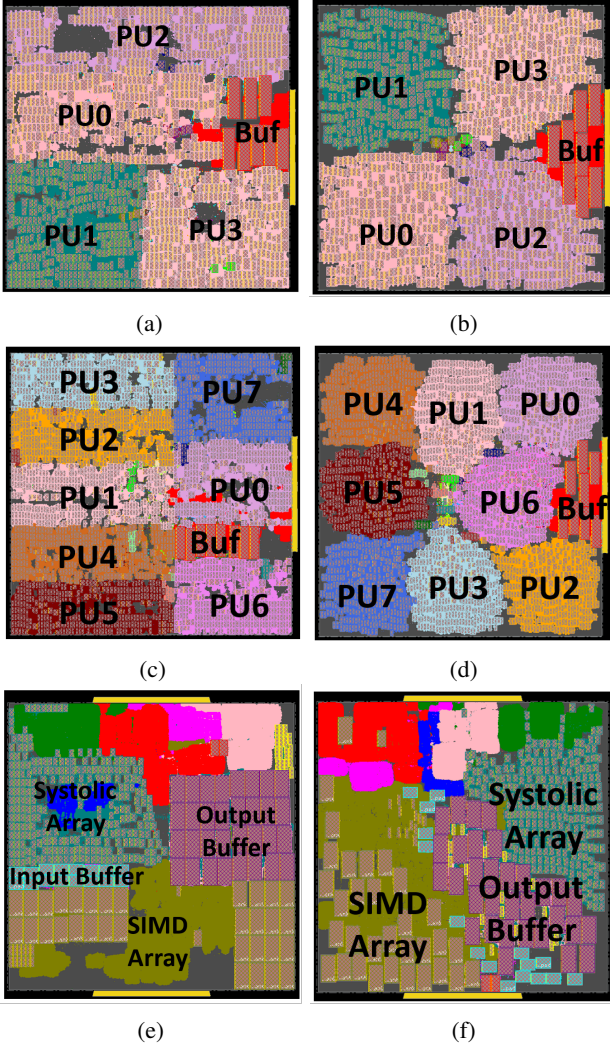


Fig. 8: Post-route layouts of Tabla02, Tabla04 and GeneSys01 designs with different flows. (a) Tabla02 (*Hier-RTLMP*); (b) Tabla02 (*DG-RePlace*); (c) Tabla04 (*Hier-RTLMP*); (d) Tabla04 (*DG-RePlace*); (e) GeneSys01 (*Hier-RTLMP*); (f) GeneSys01 (*DG-RePlace*). For the same design, each module maintains consistent coloring across different layouts. In this figure, “Buf” stands for input and output buffer.

and #PE per PU during the initial stages of machine learning accelerator development.

We also observe that for the GeneSys01 design, *Hier-RTLMP* generates the best timing metrics in terms of both WNS and TNS. We attribute this to the relatively low macro utilization of GeneSys01 (see Table II). In such contexts, *Hier-RTLMP* is able to generate reasonable macro tilings that are aligned with the dataflow structure, as shown in Figure 8(e).

E. Results on TILOS MacroPlacement Benchmarks

The dataflow-driven approach proposed in this work is inspired by the unique demands of dataflow and datapath structures in modern, highly scaled machine learning accelerators. However, benefits of our proposed method may reach beyond machine learning accelerators. To demonstrate the

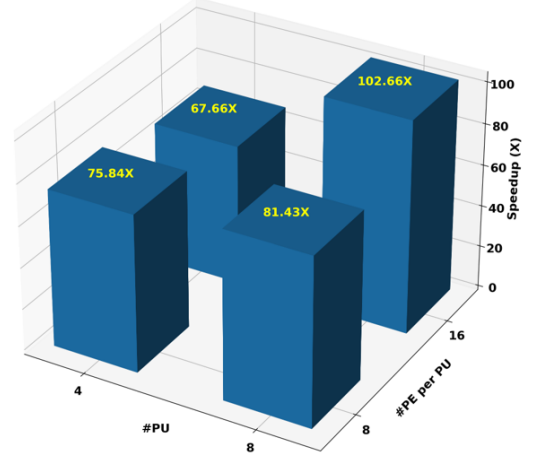


Fig. 9: Speedup of *DG-RePlace* over *Hier-RTLMP* for Tabla designs.

generality and effectiveness of our *DG-RePlace*, we conduct evaluations on the two largest benchmarks – BlackParrot (Quad-Core) [29] (827K instances, 196 macros) and MemPool Group [30] (2529K instances, 326 macros) – from the *TILOS MacroPlacement Benchmarks* [26]. The experimental results are summarized in Table VI, and the post-route layouts are presented in Figure 10.

For the BlackParrot design, *DG-RePlace* dominates *RePlace* and *DREAMPlace* across all metrics, including wirelength, power and timing. Figures 10(a), (b) and (c) show the post-route layouts from *RePlace*, *DREAMPlace* and *DG-RePlace*, respectively. It is clear that one of the CPU cores (marked in yellow) gets mixed up when placed by *RePlace* and *DREAMPlace*, resulting in worse wirelength, power and timing. Additionally, we notice that *DG-RePlace* is 2X faster than *DREAMPlace* in terms of global placement runtime, but has total turnaround time larger than *DREAMPlace*. This is because it takes 134 seconds to load the design into OpenROAD. Subtracting the loading time, the turnaround time for *DG-RePlace* drops to 66 seconds. These findings are consistent with the runtime analysis presented in Section IV-C.

TABLE VI: Experimental results on *TILOS MacroPlacement* benchmarks. We highlight best values of metrics in blue bold font. Data points for WL, Power, WNS and TNS are normalized. *DREAMPlace** represents running *DREAMPlace* with updated hyperparameters: *ignore_net_threshold* = 1e9 and *iterations* = 5000.

Design	Global Placer	WL	Power	WNS	TNS	GP (s)	TAT (s)
BlackParrot	<i>RePlace</i>	1.00	1.00	-0.123	-108.15	387	653
	<i>DREAMPlace</i>	0.92	0.98	-0.023	-2.623	61	88
	<i>DG-RePlace</i>	0.90	0.97	-0.014	-0.078	32	200
MemPool Group	<i>RePlace</i>	1.00	1.00	-0.073	-99.989	1896	2712
	<i>DREAMPlace</i>	0.92	0.97	-0.086	-134.421	72	167
	<i>DREAMPlace*</i>	0.92	0.97	-0.069	-108.193	178	284
	<i>DG-RePlace</i>	0.95	0.98	-0.067	-38.71	122	591

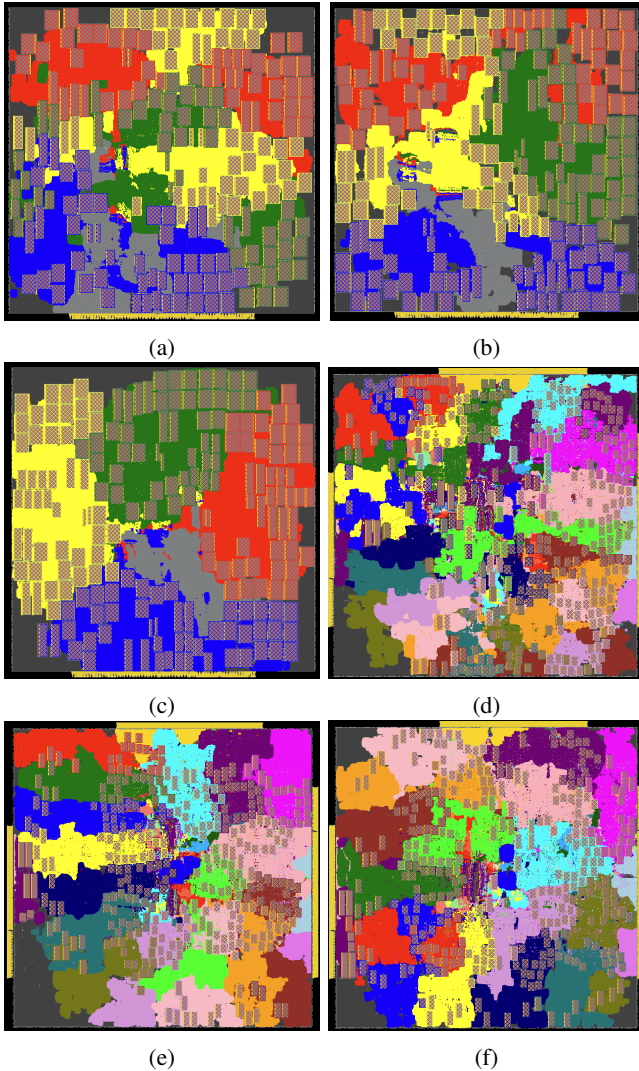


Fig. 10: Post-route layouts of BlackParrot and MemPool designs with different flows. Design (Flow): (a) BlackParrot (*RePlace*); (b) BlackParrot (*DREAMPlace*); (c) BlackParrot (*DG-RePlace*); (d) MemPool Group (*RePlace*); (e) MemPool Group (*DREAMPlace*); (f) MemPool Group (*DG-RePlace*). The layouts from *DREAMPlace* are generated with the default hyperparameter settings [27]. For the same design, each module maintains consistent coloring across different layouts.

For the MemPool Group design, *DG-RePlace* achieves significantly better timing (TNS) compared to both *RePlace* and *DREAMPlace*. However, *DG-RePlace* suffers from 3% wirelength degradation over *DREAMPlace*. To understand this wirelength increase, we use early global route (eGR) in a commercial place-and-route tool (Cadence Innovus 21.1) to examine the congestion maps for placements generated by *DG-RePlace* and *RePlace*. We observe that the placement from *DG-RePlace* is free from congestion, while there are 0.05% horizontal and 0.02% vertical congestion in the placement from *DREAMPlace*. This accounts for the increased wirelength with *DG-RePlace*, as *DG-RePlace* tries to distribute instances more evenly to prevent congestion – but at the cost of wirelength degradation. We leave how to achieve

better wirelength and congestion tradeoffs as a direction for future work.

We also notice that *DREAMPlace*, using its default parameter settings [27], terminates because it reaches its maximum number of iterations (*iteration* = 1000 by default). To prevent such early termination, we rerun *DREAMPlace* with updated hyperparameters that leave ample margin: *ignore_net_threshold* = 1e9 and *iterations* = 5000; we denote these runs as *DREAMPlace**. These hyperparameters are also set to be the default limits on net filtering and iteration count for *RePlace* and *DG-RePlace*. As shown in Table VI, *DREAMPlace** delivers better results compared to the default configuration of *DREAMPlace*, but at the cost of increased global placement runtime and total turnaround time. Even with the updated hyperparameters, *DG-RePlace* continues to outperform *DREAMPlace** in terms of timing metrics (WNS and TNS).

Additionally, for the MemPool Group design, the global placement runtime of *DG-RePlace* exceeds that of *DREAMPlace*. To delve into the reasons behind the global placement runtime degradation, we examine the detailed runtime breakdown for both *DG-RePlace* and *DREAMPlace*. The results are presented in Table VII. We can observe the following conclusions.

- The global placement runtime of *DREAMPlace* increases by $2.47\times$ when considering all the signal nets during placement.
- With the same hyperparameter settings, *DG-RePlace* is $1.46\times$ faster than *DREAMPlace** in terms of global placement runtime, while its total turnaround time is larger than that of *DREAMPlace**.
- *DG-RePlace* converges in fewer iterations compared to *DREAMPlace**, which accounts for its $1.46X$ speedup

TABLE VII: Runtime breakdown for the MemPool Group and MegaBoom_X4 benchmarks. Effective TAT is the net turnaround time, calculated by subtracting the time spent on handling input and output files (IO) from the total turnaround time (TAT). *DREAMPlace** represents running *DREAMPlace* with the updated hyperparameters *ignore_net_threshold* = 1e9 and *iterations* = 5000.

Design	Global Placer	Convergence Iterations	GP (s)	IO (s)	TAT (s)	Effective TAT (s)
MemPool Group	<i>RePlace</i>	690	1896	332	2712	2380
	<i>DREAMPlace</i>	1001	72	95	167	72
	<i>DREAMPlace*</i>	1091	178	95	284	189
	<i>DG-RePlace</i>	620	122	332	591	259
MegaBoom_X4	<i>RePlace</i>	870	7433	370	8546	8176
	<i>DREAMPlace</i>	993	319	230	550	319
	<i>DREAMPlace*</i>	1036	881	230	1113	883
	<i>DG-RePlace</i>	770	418	370	937	567

TABLE VIII: Experimental results on the MegaBoom_X4 design. Data points for WL are normalized.

Design	# Std Cells	# Nets	Global Placer	WL	Horizontal Congestion	Vertical Congestion
MegaBoom_X4	5807K	5831K	<i>RePlace</i>	1.00	0.01%	0.07%
			<i>DREAMPlace</i>	1.00	0.02%	0.08%
			<i>DREAMPlace*</i>	1.00	0.01%	0.08%
			<i>DG-RePlace</i>	1.00	0.00%	0.08%

of global placement runtime. To further confirm this speedup, we run the same experiments on another large design, MegaBoom_X4 (four-core RISC-V MegaBoom [36]), which has more than 5.8M instances. Experimental results on MegaBoom_X4 shown in Table VII give support to our analysis. Detailed metrics for MegaBoom_X4 are presented in Table VIII.¹⁰

- After subtracting the time spent on handling file input and output (IO) from the total turnaround time (TAT), the net turnaround time (“Effective TAT”) of *DG-RePlace* is larger than that of *DREAMPlace**. As pointed out in Section IV-C, we attribute this to the *physical hierarchy extraction* process. Enhancing the efficiency of the *physical hierarchy extraction* process is a key objective for our future research efforts.

V. CONCLUSION AND FUTURE WORK

In this work, we develop *DG-RePlace*, a new and fast GPU accelerated global placement framework which is built on top of the OpenROAD infrastructure [24], and which exploits the inherent dataflow and datapath structures of machine learning accelerators to achieve superior results. Experimental results show that *DG-RePlace* outperforms both *RePlace* and *DREAMPlace* in terms of routed wirelength and total negative slack metrics. Extensions to *DG-RePlace* that we are currently exploring include: (i) incorporation of density screens for routability and virtual resizing for timing optimization; (ii) application of ML-based multi-objective optimization methods to autotune the hyperparameters of *DG-RePlace*, potentially achieving better tradeoffs across wirelength, congestion, power and timing; and (iii) improving the runtime of the *physical hierarchy extraction* process. In combination with open-sourcing and OpenROAD integration, we believe that this work will add to the foundations for new research on fast and high-quality global placers.

REFERENCES

- [1] A. Agnesina, P. Rajvanshi, T. Yang, G. Pradipta, A. Jiao et al., “AutoDMP: automated DREAMPlace-based macro placement”, *Proc. ISPD*, 2023, pp. 149–157.
- [2] S. Chou, M.-K. Hsu and Y.-W. Chang, “Structure-aware placement for datapath-intensive circuit designs”, *Proc. DAC*, 2012, pp. 762–767.
- [3] C.-K. Cheng, A. B. Kahng, I. Kang and L. Wang, “RePlace: advancing solution quality and routability validation in global placement”, *IEEE Trans. on CAD* 38(9) (2019), pp. 1717–1730.
- [4] H. Esmailzadeh, S. Ghodrati, J. Gu, S. Guo et al., “VeriGOOD-ML: an open-source flow for automated ML hardware synthesis”, *Proc. ICCAD*, 2021, pp. 1–7.
- [5] D. Fang, B. Zhang, H. Hu, W. Li, B. Yuan et al., “Global placement exploiting soft 2D regularity”, *Proc. ISPD*, 2022, pp. 203–210.
- [6] J. Cong and Y. Zou, “Parallel multi-level analytical global placement on graphics processing units”, *Proc. ICCAD*, 2009, pp. 681–688.
- [7] F. Gessler, P. Brisk and M. Stojilović, “A shared-memory parallel implementation of the RePlace global cell placer”, *Proc. VLSID*, 2020, pp. 78–83.
- [8] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal et al., “In-datacenter performance analysis of a tensor processing unit”, *Proc. ISCA*, 2017, pp. 1–12.
- [9] A. B. Kahng, R. Varadarajan and Z. Wang, “RTL-MP: toward practical, human-quality chip planning and macro placement”, *Proc. ISPD*, 2022, pp. 3–11.
- [10] A. B. Kahng, R. Varadarajan and Z. Wang, “Hier-RTLMP: a hierarchical automatic macro placer for large-scale complex IP blocks”, *IEEE Trans. on CAD*, 2023, <https://ieeexplore.ieee.org/document/10372220>
- [11] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, “Optimization by simulated annealing”, *Science* 220(4598) (1983), pp. 671–680.
- [12] J. Lu, P. Chen, C.-C. Chang, S. Lu, D. J.-H. Huang et al., “ePlace: electrostatics-based placement using fast fourier transform and nesterov’s method”, *ACM Trans. Des. Autom. Electron. Syst.* 20(2) (2015), pp. 1–34.
- [13] C.-X. Lin and M. D. F. Wong, “Accelerate analytical placement with GPU: a generic approach”, *Proc. DATE*, 2018, pp. 1345–1350.
- [14] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar et al., “DREAMPlace: deep learning toolkit-enabled GPU acceleration for modern VLSI placement”, *IEEE Trans. on CAD* 40(4) (2020), pp. 748–761.
- [15] J.-M. Lin, W.-F. Huang, Y.-C. Chen, Y.-T. Wang and P.-W. Wang, “DAPA: a dataflow-aware analytical placement algorithm for modern mixed-size circuit designs”, *Proc. ICCAD*, 2021, pp. 1–8.
- [16] J.-M. Lin, Y.-L. Deng, Y.-C. Yang, J.-J. Chen and P.-C. Lu, “Dataflow-aware macro placement based on simulated evolution algorithm for mixed-size designs”, *IEEE Trans. on VLSI Systems* 29(5) (2021), pp. 973–984.
- [17] P. Liao, S. Liu, Z. Chen, W. Lv, Y. Lin and B. Yu, “DREAMPlace 4.0: timing-driven global placement with momentum-based net weighting”, *Proc. DATE*, 2022, pp. 939–944.
- [18] L. Liu, B. Fu, M. D. F. Wong and E. F. Y. Young, “Xplace: an extremely fast and extensible global placement framework”, *Proc. DAC*, 2022, pp. 1309–1314.
- [19] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence-pair”, *IEEE Trans. on CAD* 15(12) (1996), pp. 1518–1524.
- [20] R. X.T. Nijssen and J. A.G. Jess, “Two-dimensional datapath regularity extraction”, *Proc. ACM/SIGDA Physical Design Workshop*, 1996, pp. 111–117.
- [21] P. Quinton, “An introduction to systolic architectures”, *Future Parallel Computers: An Advanced Course*, P. Treleaven and M. Vanneschi, Eds., Springer, 1987, pp. 387–400.
- [22] A. Vidal-Obiols, J. Cortadella, J. Petit, M. Galceran-Oms and F. Martorell, “Multilevel dataflow-driven macro placement guided by RTL structure and analytical methods”, *IEEE Trans. on CAD* 40(12) (2021), pp. 2542–2555.
- [23] S. Ward, D. Ding and D. Z. Pan, “PADE: A high-performance placer with automatic datapath extraction and evaluation through high-dimensional data learning”, *Proc. DAC*, 2012, pp. 756–761.
- [24] OpenROAD, <https://github.com/The-OpenROAD-Project/OpenROAD>.
- [25] DREAMPlace 4.0.0, <https://github.com/limbo018/DREAMPlace>.
- [26] MacroPlacement, <https://github.com/TILOS-AI-Institute/MacroPlacement>.
- [27] Default hyperparameter settings for DREAMPlace, https://github.com/limbo018/DREAMPlace/blob/master/test/ispd2019/lefdef/ispd19_test1.json.
- [28] Default hyperparameter settings for RePlace, <https://github.com/The-OpenROAD-Project/RePlace/blob/cf289bb141a995d8304656cd994ba3f2f95b2f8a/src/nesterovPlace.cpp#L24>.
- [29] BlackParrot repo, <https://github.com/black-parrot/black-parrot>.
- [30] MemPool repo, <https://github.com/pulp-platform/mempool>.
- [31] Ablation (artificial intelligence), [https://en.wikipedia.org/wiki/Ablation_\(artificial_intelligence\)#:~:text=An%20ablation%20study%20investigates%20the,component%20to%20the%20overall%20system](https://en.wikipedia.org/wiki/Ablation_(artificial_intelligence)#:~:text=An%20ablation%20study%20investigates%20the,component%20to%20the%20overall%20system).
- [32] OpenDB, <https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/odb>.
- [33] Hier-RTLMP, <https://github.com/The-OpenROAD-Project/OpenROAD/tree/master/src/mpl2>.
- [34] The DG-RePlace repository, <https://github.com/ABKGroup/DG-RePlace>.
- [35] Himax Technologies, Inc., <https://www.himax.com.tw/company/about-himax>.
- [36] BOOM: the berkeley out-of-order risc-v processor, <https://github.com/riscvboom/>.

¹⁰Since we cannot finish the place-and-route flow for the MegaBoom_X4 design, we report metrics (wirelength, horizontal and vertical congestion) after early global routing in the commercial tool.



Andrew B. Kahng is a distinguished professor in the CSE and ECE Departments of the University of California at San Diego. His interests include IC physical design, the design-manufacturing interface, large-scale combinatorial optimization, AI/ML for EDA and IC design, and technology roadmapping. He received the Ph.D. degree in Computer Science from the University of California at San Diego.



Zhiang Wang received the M.S. degree in electrical and computer engineering from the University of California at San Diego, La Jolla, in 2022. He is currently pursuing the Ph.D. degree at the University of California at San Diego, La Jolla. His current research interests include partitioning, placement methodology and optimization.