CU.POKer: Placing DNNs on Wafer-Scale AI Accelerator with Optimal Kernel Sizing

Bentian Jiang, Jingsong Chen, Jinwei Liu, Lixin Liu, Fangzhou Wang, Xiaopeng Zhang, Evangeline F.Y. Young

CSE Dept., The Chinese University of Hong Kong

Nov. 03, 2020

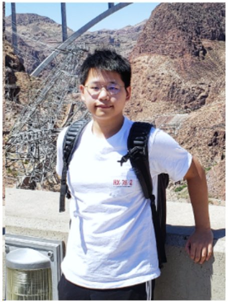
Speaker Biography

Biography

- Bentian Jiang is currently pursuing a Ph.D. degree with the Dept. of Computer Science & Engineering, The Chinese University of Hong Kong, under the supervision of Prof. Evangeline F.Y. Young.
- He is a recipient of several prizes in renowned EDA contests including the CAD Contests at ICCAD 2018 and ISPD 2018, 2019, 2020.

Research Interests

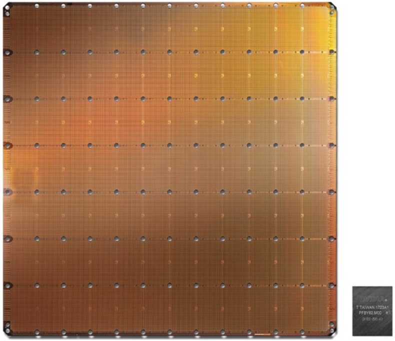
- Design for manufacturability
- Physical design



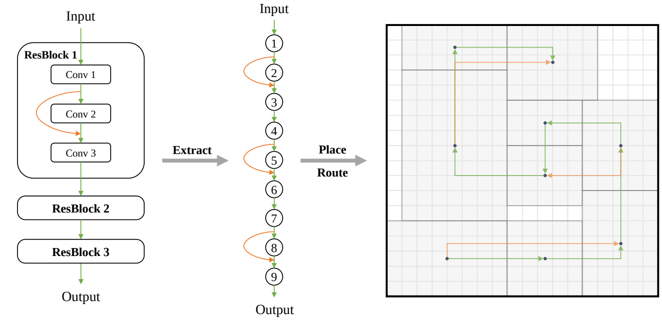
Outline

Outline

Simplified View CS-1 Compilation Flow



CS-1 WSE is one of the largest AI chip with more than 400,000 programmable compute cores. Figure from James *et al.* ISPD'20 [2]

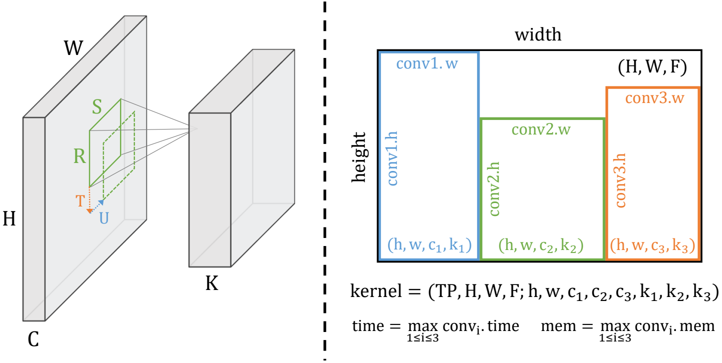


(a) Network Architecture (b) Kernel Graph (c) Execution Plan

CS-1 WSE compilation flow, the proposed framework focuses on the placement stage of compilation.

Kernel Definition

- *conv*: basic convolution kernel



(a) Arguments of *conv* (b) Performance of a kernel with 3 *convs*

- 8 formal arguments: (H, W, R, S, C, K, T, U) $\Rightarrow$ fixed input parameters.
- 4 execution arguments: (h, w, c, k) $\Rightarrow$ variables to be determined.

Kernel Evaluation

Performance Cuboid (height, width, time, memory) of conv

$$
\begin{aligned}
\text{convperf}(\underbrace{H,\ W,\ R,\ S,\ C,\ K,\ T,\ U;}_{\text{Formal arguments}}\ \underbrace{h,\ w,\ c,\ k}_{\text{Execution arguments}}) = \{ \\
\text{height} &= h \cdot w \cdot (c+1) \\
\text{width} &= 3k \\
\text{time} &= \text{ceil}(\frac{H}{h}) \cdot \text{ceil}(\frac{W}{w}) \cdot \text{ceil}(\frac{C}{c}) \cdot \text{ceil}(\frac{K}{k}) \cdot \frac{RS}{T^2} \\
\text{mem} &= \frac{C}{c} \cdot \frac{K}{k} \cdot RS + \frac{W+S-1}{w} \cdot \frac{H+R-1}{h} \cdot \frac{K}{k} \\
\}
\end{aligned}
\tag{1}
$$

Kernel Evaluation

- For a certain type of kernel that contains n convs

Performance Cuboid (height, width, time, memory) of Kernel

$$\begin{aligned}
&\text{blockperf}(TP, H, W, F; h, w, c_1, ..., c_n, k_1, ..., k_n) = \{ \\
&\quad conv_i = \text{convperf}(H_i, W_i, R_i, S_i, C_i, K_i, T_i, U_i; h, w, c_i, k_i), \quad \forall\ i \in \{1, ..., n\} \\
&\quad \text{height} = \max_{1 \le i \le n} conv_i.height, \qquad \text{width} = \sum_{i=1}^{n} conv_i.width \\
&\quad \text{time} = \max_{1 \le i \le n} conv_i.time, \qquad \text{mem} = \max_{1 \le i \le n} conv_i.mem \\
&\}
\end{aligned} \tag{2}$$

Problem Formulation

- Determine the execution parameters and the locations for all kernels.

Hard Constraints

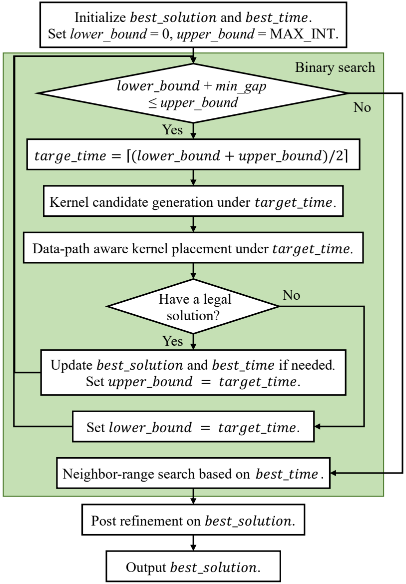
- All kernels must fit within the fabric area (633 x 633 tiles).
- No kernels may overlap.
- No kernel's memory exceeds the tile's memory limit.

Objectives to Minimize

- The maximum execution time among all placed kernels.
- The total L1 distance of all connected kernels.
- The total adapter cost of all connected kernels.

$$\text{cost}_{\text{adapter}} = \mathbf{1}(h_{out}! = h_{in}) + \mathbf{1}(w_{out}! = w_{in}) + \mathbf{1}(c_{out,n}! = c_{in,1})$$

Overview of Proposed Flow



Two-steps Search

- Binary search
  - Rapidly locate a good and feasible maximum execution time slot
- Neighbor-range search
  - Further improve the solution
- Post refinement
  - Optimize adapter cost and wirelength further

Searching under Target Time

- Kernel candidates generation with optimal shapes under given target time
- Data-path aware placement

Outline

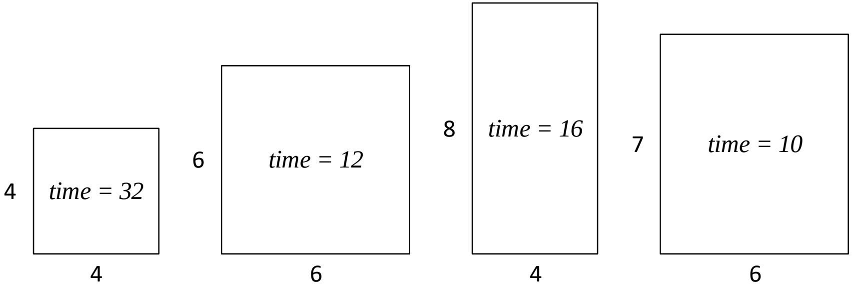
Kernel Sizing

- Goal: find all kernel candidates with optimal shapes and satisfying a given $target\_time$ constraint.
- Motivation 1: the optimal wire length can be achieved by using the kernels with optimal shapes only (under a given $target\_time$ constraint).
- Motivation 2: the optimal shaped kernel set is relatively small ($< 633/2$).

Optimal Shapes

Optimal shapes

- A kernel is regarded as having optimal shape if and only if there doesn't exist another kernel satisfying the same $target\_time$ constraint and having a better shape.



For $target\_time = 16$, only the second and the third shapes are regarded as optimal.

A Simplification

It seems that enforcing $c_1 = c_2 = ... = c_x = c$ in the cuboid performance equation will not sacrifice optimality.

Observation

For any argument $\{h, w, c_1, ..., c_x, k_1, ..., k_x\}$, there exist a $c = \max(c_1, ..., c_x)$ such that

$$ker_1 = blockperf(TP, H, W, F; h, w, c, ..., c, k_1, ..., k_x),$$

is no worse than

$$ker_2 = blockperf(TP, H, W, F; h, w, c_1, ..., c_x, k_1, ..., k_x)$$

with regard to $height$, $width$, $time$ and $memory$.

Optimization View

Solving The Optimal $width$ For $height = \eta$ $(\eta = 1, ..., 633)$

$$
\begin{aligned}
\underset{h,w,c,k_1,\ldots,k_x}{\text{Minimize:}}\ & width \\
\text{Such that:}\ & height = h \cdot w \cdot (c+1) = \eta \\
& width = \sum_{j=1}^{x} 3k_j \\
& time = \max_{1 \le j \le x} \mathrm{ceil}(\frac{H_j}{h})\mathrm{ceil}(\frac{W_j}{w})\mathrm{ceil}(\frac{C_j}{c})\mathrm{ceil}(\frac{K_j}{k_j})\frac{R_j S_j}{T_j^2} \\
& \quad \le target\_time \\
& mem = \max_{1 \le j \le x} \frac{C_j K_j R_j S_j}{c k_j} + \frac{(W_j + S_j - 1)(H_j + R_j - 1)K_j}{whk_j} \\
& \quad \le memory\_limit
\end{aligned} \tag{3}
$$

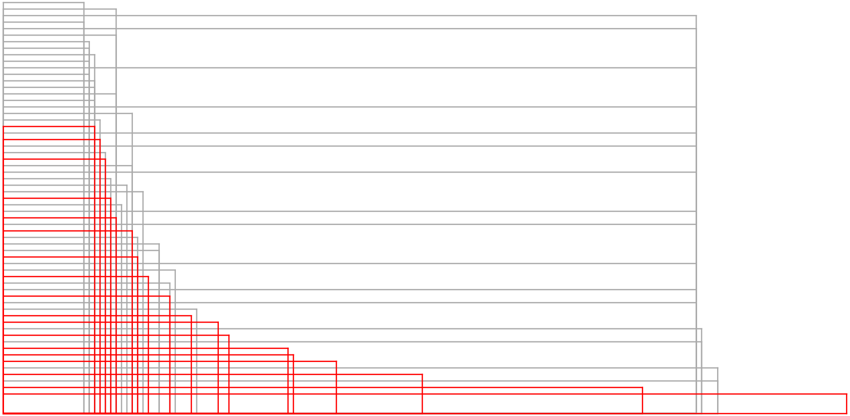
Method to Solve It

- Factorize η to get all the possible values of $\{h, w, c+1\}$.
- For each $\{h, w, c+1\}$, solve the following equations to get the minimum ks.

Getting the ks

$$
\begin{aligned}
&\text{For j} = 1, ..., x: \\
&\quad k_j^t = \text{ceil}(\text{ceil}(\frac{H_j}{h})\text{ceil}(\frac{W_j}{w})\text{ceil}(\frac{C_j}{c})\frac{R_j S_j K_j}{T_j^2 \cdot target\_time}) \\
&\quad k_j^m = \text{ceil}(\frac{C_j K_j R_j S_j}{c \cdot memory\_limit} + \frac{(W_j + S_j - 1)(H_j + R_j - 1)K_j}{wh \cdot memory\_limit}) \\
&\quad k_j = \max(k_j^t, k_j^m)
\end{aligned}
\tag{4}
$$

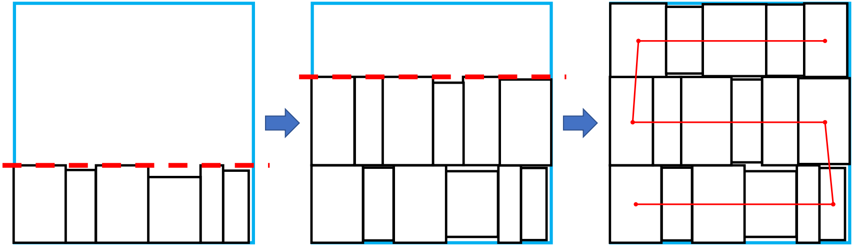
Final Pruning



An example solution (red) of kernel sizing after final pruning (with rotation consideration).

Outline

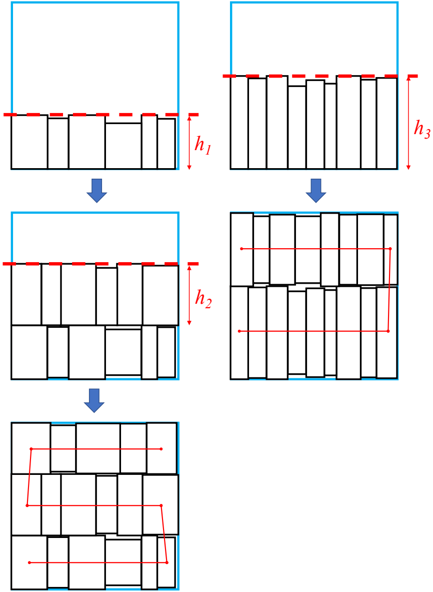
Data-path-aware Kernel Placement



Overall Flow

- Given a target time T, generate all the kernel candidates with optimal shapes and execution times under T.
- According to the connectivity graph, generate the topological order of the kernels for placement.
- Place the kernels compactly row by row in the topological order.

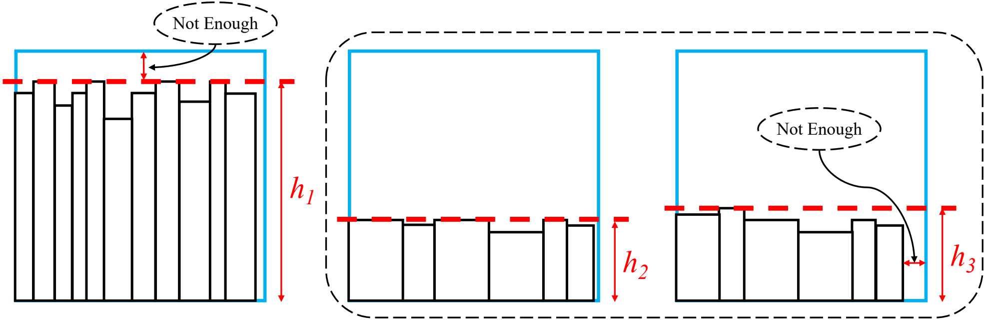
Algorithm



Data-path-aware Kernel Placement

function `Placement`$(next\_index, target\_time, floor\_height)$
  $H_k \leftarrow$ a sorted height set of all the kernel candidates
  for each height h in H_k **do**
    if $h + floor\_height > chip\_height$ **then**
      break
    end if
    $w_{idle} \leftarrow chip\_width$
    $max\_height \leftarrow 0$
    for $i = next\_index, \ldots, num\_kernel$ **do**
      $w_i \leftarrow$ minimum width of the i^{th} kernel's candidates meeting the requirements of $target\_time$ and h
      $h_i \leftarrow$ the corresponding height of w_i
      if $w_i > w_{idle}$ **then**
        $i \leftarrow i - 1$
        break
      else
        $w_{idle} \leftarrow w_{idle} - w_i$
        $max\_height \leftarrow \max(max\_height, h_i)$
      end if
    end for
    if $i < next\_index$ **then**
      continue
    end if
    Place the kernels of indices from $next\_index$ to i in a row on $floor\_height$
    if $i \equiv num\_kernel$ **then**
      Update the best solution if needed
    else
      $floor\_height \leftarrow floor\_height + max\_height$
      `Placement`$(i, target\_time, floor\_height)$
    end if
  end for
end function

Pruning

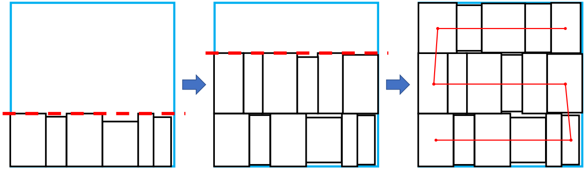


Two Pruning Steps

1. After placing one kernel, check if the remaining empty space on the fabric is less than the smallest total area of the kernels yet to be placed. If so, stop the current placement iteration.
2. Skip the "redundant" heights when traversing H_k to avoid unnecessary iterations.

Outline

Protocol Cost Optimization



Wasted Deadspace

- Not every kernel will have its height equal to the floor height.
- Suppose there are n kernels on the i^{th} floor of the layout, for each kernel $ker_{i,j}$, $j \in \{1, ..., n\}$, we have

$$ker_{i,j}.height \leq floor_i.height = \max_{1 \leq j \leq n} ker_{i,j}.height.$$

- If $ker_{i,j}.height < floor_i.height$, exists deadspace with

$$\Delta height_{i,j} = (floor_i.height - ker_{i,j}.height), \quad width_{i,j} = ker_{i,j}.width \tag{5}$$

Protocol Cost Optimization

Unifying (h, w) Pair for Each Floor

- Assume $ker_{i,j}$, the j^{th} kernel on the i^{th} floor, contains m $(conv)$, we have

$$ker_{i,j}.height = h \cdot w \cdot (c_{max} + 1) = \max_{1 \leq j \leq m} h \cdot w \cdot (c_j + 1).$$

- Let new $ker_{i,j}.height = floor_i.height = (ker_{i,j}.height + \Delta height_{i,j})$, a new c_{max} can be uniquely determined by a given reference pair $(\mathrm{h}_{\mathrm{ref}}, \mathrm{w}_{\mathrm{ref}})$

$$c_{max} = floor_i.height/(h_{ref} \cdot w_{ref}) - 1.$$

- A new assignment for $ker_{i,j}$'s arguments $(c_1, ..., c_m)$ is given by $c_1 = ... = c_m = c_{max}$
- We may unify the (h,w) for all kernels in the same floor with same $(\mathrm{h}_{\mathrm{ref}}, \mathrm{w}_{\mathrm{ref}})$ and hereby reduce the adapter cost since we place them in topological order and

$$\mathrm{cost}_{\mathrm{adapter}} = \mathbf{1}(h_{out}! = h_{in}) + \mathbf{1}(w_{out}! = w_{in}) + \mathbf{1}(c_{out,n}! = c_{in,1})$$

Protocol Cost Optimization

A Universal Scheme

- Greedy search for each floor, all possible reference pairs will be evaluated and the one leading to the best adapter cost will be committed.
- Regardless of kernel protocol functions.
- Worst case complexity is bounded by O(n^2), but there are only thousand kernels at most (negligible runtime) in practice.

Further Improvement

- The rest element, which is related to the protocol function, can be optimized via dynamic programming.

	W/O Adapter Opt.		W/ Adapter Opt.	
Case	AC*	Ratio	AC*	Ratio
A	15	1.00	15	1.00
B	18	1.00	18	1.00
C	234	1.00	**185**	**0.79**
D	139	1.00	**123**	**0.88**
E	11	1.00	11	1.00
F	13	1.00	**12**	**0.92**
G	221	1.00	**98**	**0.44**
H	77	1.00	**49**	**0.64**
I	13	1.00	13	1.00
J	193	1.00	**69**	**0.36**
K	9	1.00	**3**	**0.33**
L	140	1.00	**18**	**0.13**
M	41	1.00	41	1.00
N	10	1.00	10	1.00
O	13	1.00	13	1.00
P	154	1.00	**85**	**0.55**
Q	6	1.00	6	1.00
R	68	1.00	**20**	**0.29**
S	60	1.00	**48**	**0.80**
T	4	1.00	4	1.00
Avg.	71.95	1.00	**42.05**	**0.76**

Outline

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations & Case Study

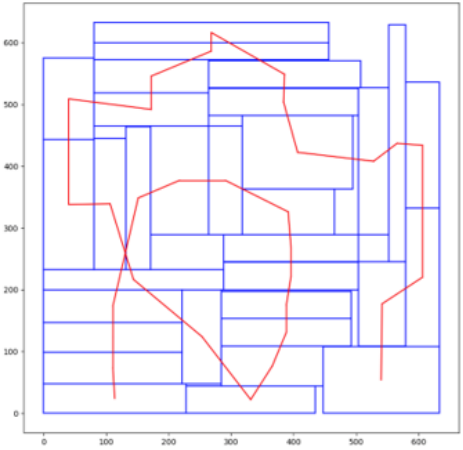
Simulated Annealing Placer

SA Placer with Twin Binary Sequences

- Most commonly used floorplan heuristic.
- SA-based placer with the twin binary sequences (TBS) representation [3].
- Compact packing is used to realize a layout from a given TBS.
- On 8 public benchmarks, 11% better than the best contestant (4th) using SA placer.

Actions

- Pick up a new kernel candidate.
- Swap two kernels.
- Rotate the sequences to change the packing topology.

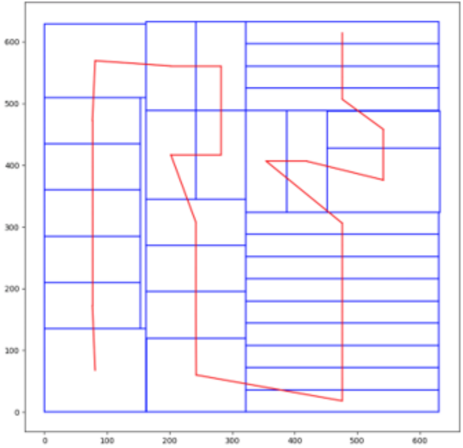


Kgraph-F by Simulated Annealing Placer.

Divide and Conquer Placer

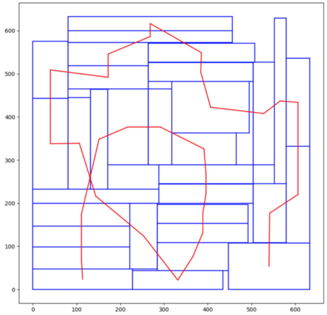
Slicing Placer

- Top-down phase for graph partition.
- Sub-graphs of each level should have
  - Similar total area
  - Fewer interconnections.
- Bottom-up phase to commit and merge placement results.
- On 8 public benchmarks, 32% better than the best contestant (4th) using SA placer.



Kgraph-F by Divide and Conquer Placer.

Comparisons with Conventional Floorplanning Heuristics



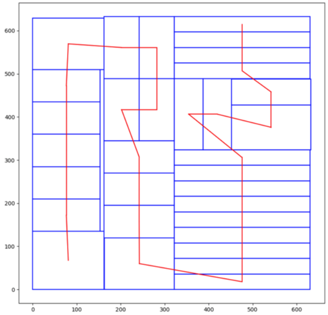
SA Placement:

Max_time: 76698

Wire_length: 3237

Adapter_cost: 15

Score: 110478



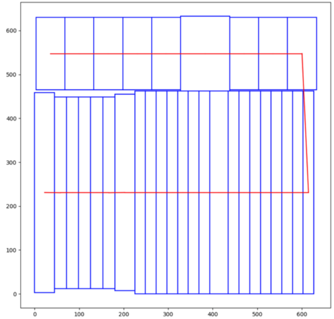
Slicing Placement:

Max_time: 65016

Wire_length: 2650.5

Adapter_cost: 18

Score: 93321



Our Final Method:

Max_time: 65170

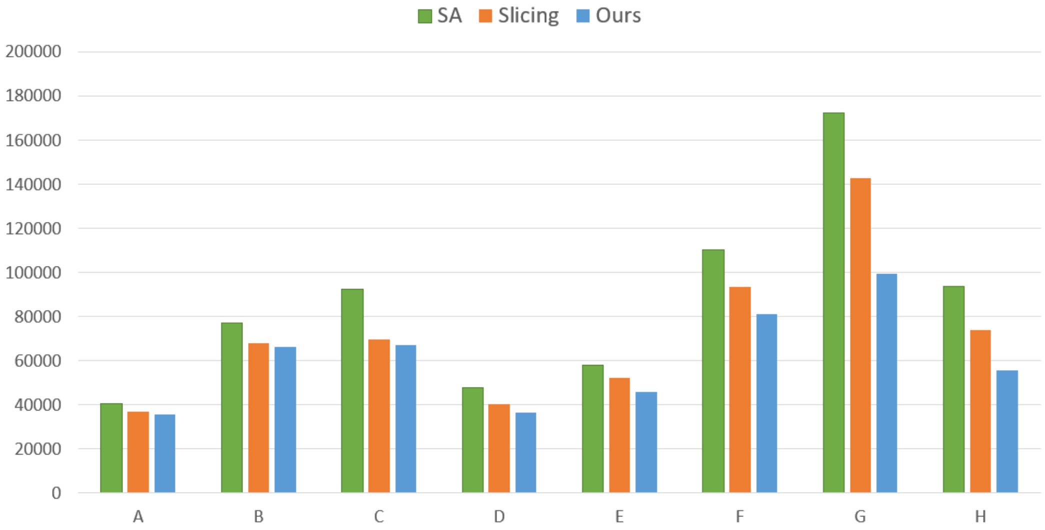
Wire_length: 1489.5

Adapter_cost: 12

Score: 81265

Layout comparisons with SA and DC placers on kgraph-f.

Comparisons with Conventional Floorplanning Heuristics

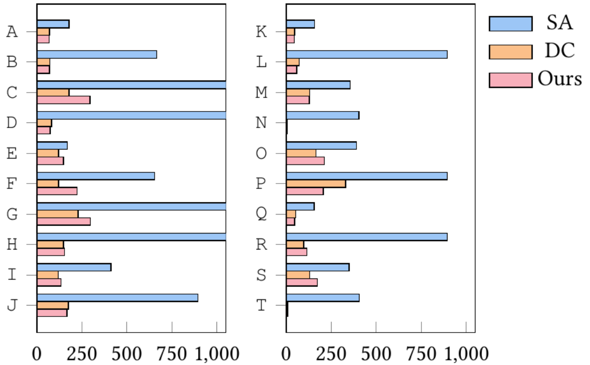


Performance comparisons with SA and DC placers on 8 public benchmarks.

Comparisons with Conventional Floorplanning Heuristics

Observations

- Common floorplanning heuristics cannot handle this challenge well.
- SA-based placer is too general, solution space is too large.
  - Connections are mostly aligned data paths with some forks.
  - Have many choices of candidate shapes.
- DC-based placer is fast, but has inevitable detour (layout layers number is strictly proportional to the size of input kernel graph).



Runtime comparisons with SA and DC placers.

Experimental Results on ISPD-20 Suite [1]

Our CU.POKer won the 1st place in ISPD 2020 contest

Comparing to GigaPlacer (2nd place)

- 16% better on all testcases, 25% better on hidden testcases

Comparing to CUPID (3rd place)

- 30% better on all testcases, 46% better on hidden testcases

Comparing to SA placer

- 52% better on all testcases, 61% better on hidden testcases

Comparing to Slicing placer

- 37% better on all testcases, 58% better on hidden testcases

Q&A

Thanks and Questions?

ISPD 2020 Contest: Wafer-Scale Deep Learning Accelerator Placement.
https://www.cerebras.net/ispd-2020-contest/.

James, M., Tom, M., Groeneveld, P., and Kibardin, V.
Ispd 2020 physical mapping of neural networks on a wafer-scale deep learning accelerator.
In *Proceedings of the 2020 International Symposium on Physical Design* (New York, NY, USA, 2020), ISPD '20, Association for Computing Machinery, p. 145–149.

Young, E. F., Chu, C. C., and Shen, Z. C.
Twin binary sequences: a nonredundant representation for general nonslicing floorplan.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22, 4 (2003), 457–469.