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

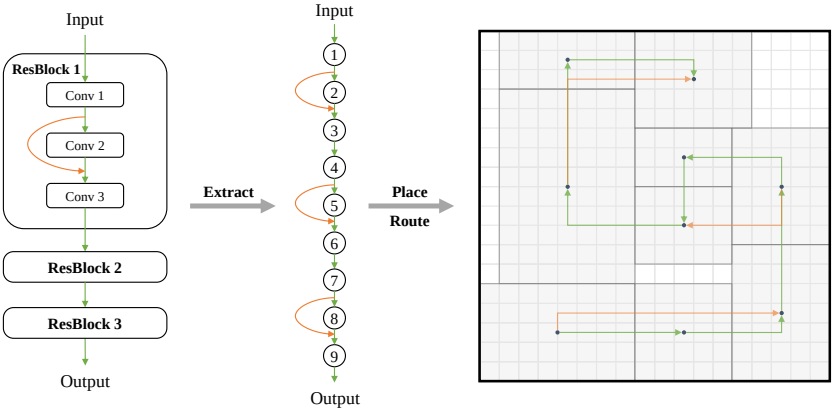
July. 16, 2020

香港中文大學
The Chinese University of Hong Kong

Outline

Outline

Simplified View CS-1 Compilation Flow

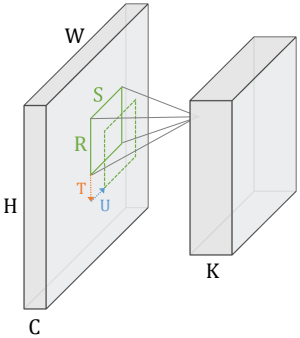


(a) Network Architecture (b) Kernel Graph (c) Execution Plan

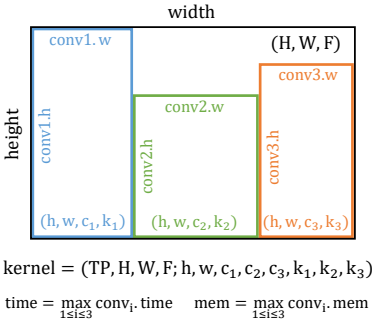
CS-1 WSE compilation flow, the proposed framework focuses on the placement stage of compilation.

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) ⇒ fixed input parameters.
- 4 execution arguments: (h, w, c, k) ⇒ variables to be determined.

Kernel Evaluation

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

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

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

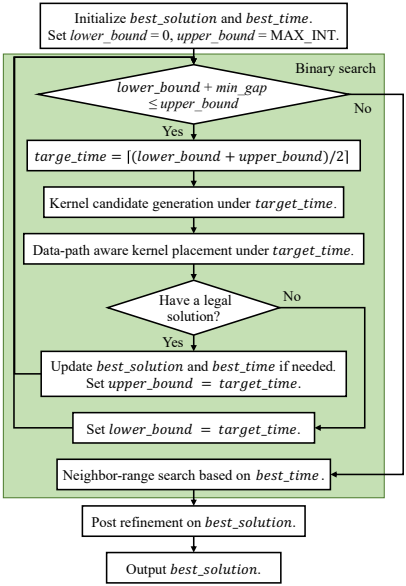
- All kernels must fit within the fabric area (633 x 633 tiles).
- No kernels may overlap.
- No kernel's memory exceeds the tile's memory limit.

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.

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

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 under given target time
- Data-path aware placement

Outline

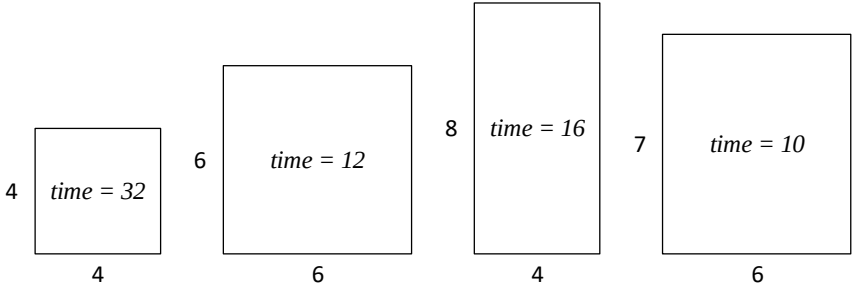
Kernel Sizing

- 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.

Theorem For This

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 $height = \eta$ $(\eta = 1, ..., 633)$

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

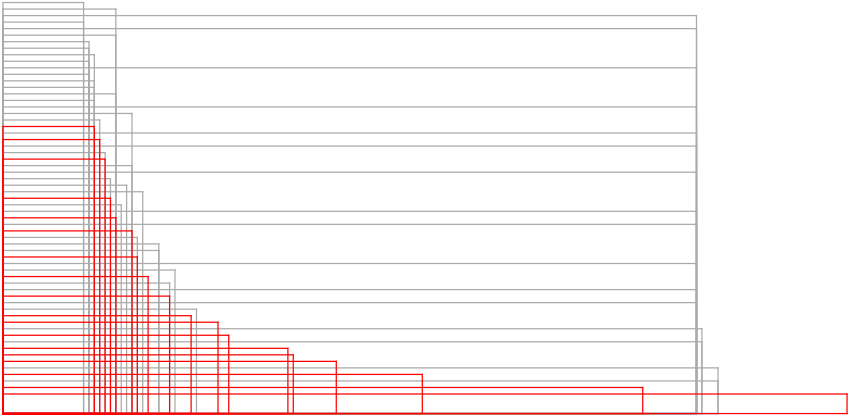
Method to Solve It

- 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

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

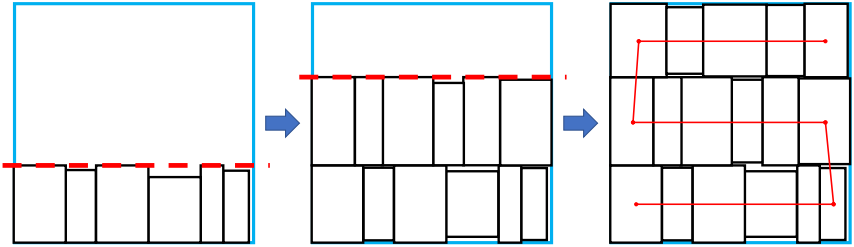
Final Pruning



An example solution of kernel sizing.

Outline

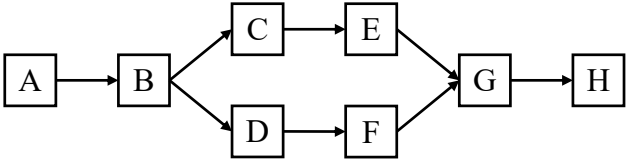
Data-path-aware Kernel Placement



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.

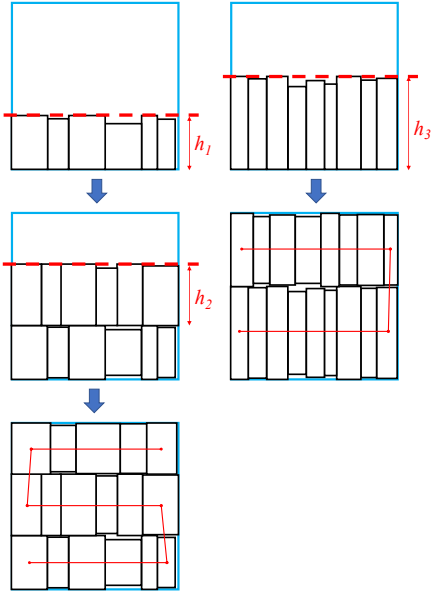
Topological Order Generation



Depth-first Search

- The topological order is generated by depth-first search on the connectivity graph.
- Depth-first search can handle the forks in the connectivity graph.
- For the above connectivity graph, the topological order is $ABCEDFGH$.

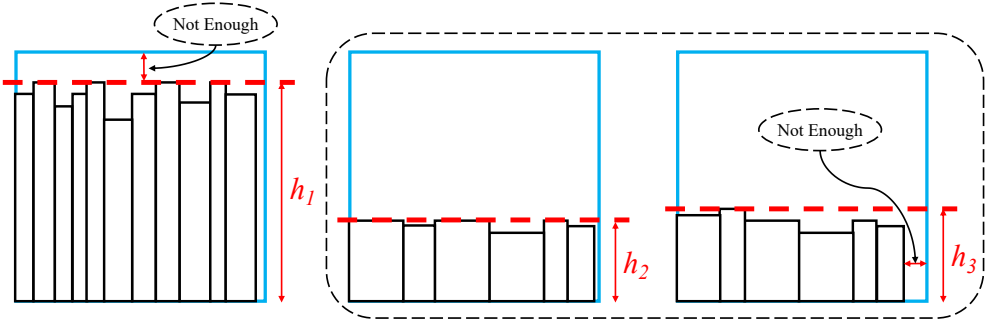
Algorithm



Data-path-aware Kernel Placement

```
function Placement(next_index, target_time, floor_height)
    H_k ← 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 ← chip_width
        max_height ← 0
        for i = next_index, ..., num_kernel do
            w_i ← minimum width of the i^th kernel's candidates meeting the requirements of target_time and h
            h_i ← the corresponding height of w_i
            if w_i > w_idle then
                i ← i − 1
                break
            else
                w_idle ← w_idle − w_i
                max_height ← 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 ≡ num_kernel then
            Update the best solution if needed
        else
            floor_height ← floor_height + max_height
            Placement(i, target_time, floor_height)
        end if
    end for
end function
```

Pruning

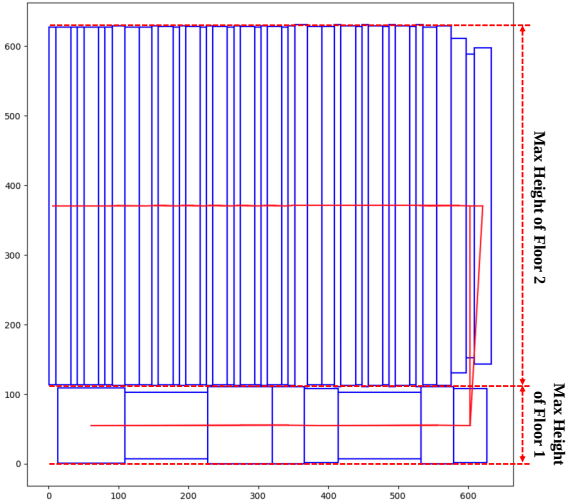


Two Pruning Steps

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



Solution before post refinement.

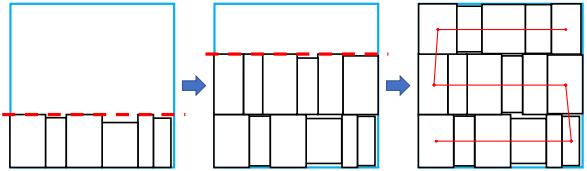
Observations

- No explicit correlation between the protocol cost and the other two costs.
- Determined after the statuses of all connected kernels were known.
- Revision on single kernel may affect both its input and output ports.

Properties

- May need to revise multiple relevant kernels simultaneously.
- Not compatible with the previous sequential placement flow.

Protocol Cost Optimization



Wasted Deadspace

- Not every kernel will have its height equal to the floor height.
- Suppose there are n kernels on the ith 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

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

Protocol Cost Optimization

Unifying (h, w) Pair for Each Floor

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

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

- 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 $(\mathrm{h_{ref}}, \mathrm{w_{ref}})$

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

- A new assignment for $ker_{i,j}$'s arguments $(c_1, ..., c_m)$ is then given by

$$c_1 = ... = c_m = c_{max} = floor_i.height/(h_{ref} * w_{ref}) - 1.$$

- This is one of the optimal assignments for arguments $(c_1, ..., c_m)$, by following a similar argument as in the proof of page 10.

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 by simulated annealing.

	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

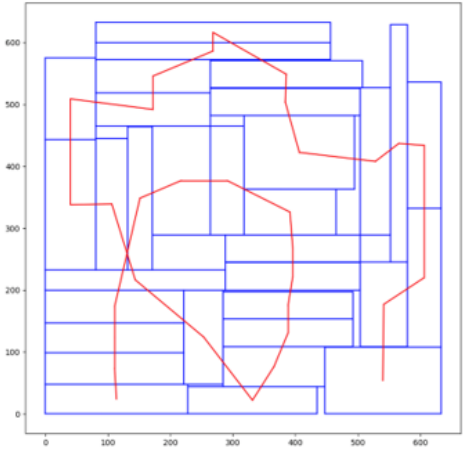
Simulated Annealing Placer

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.
- 11% better than NTU428 SA placer.

Actions

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

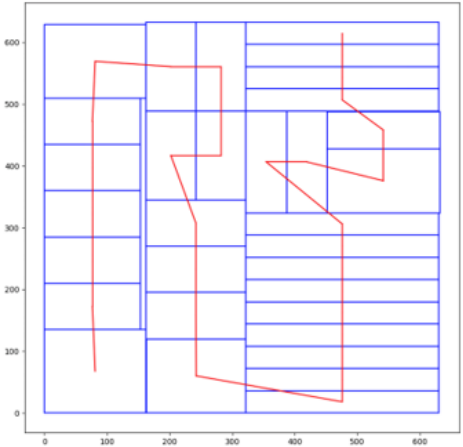


Kgraph-F by Simulated Annealing Placer.

Divide and Conquer Placer

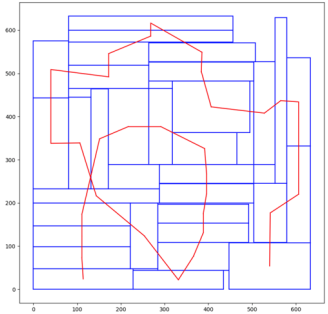
Slicing Placer

- 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.
- 32% better than NTU428 SA placer.



Kgraph-F by Divide and Conquer Placer.

Comparisons with Conventional Floorplanning Heuristics



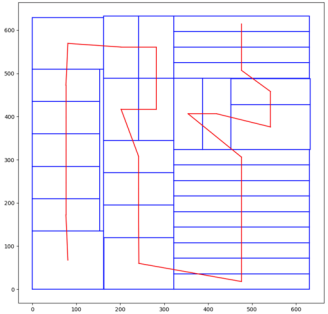
SA Placement:

Max_time: 76698

Wire_length: 3237

Adapter_cost: 15

Score: 110478



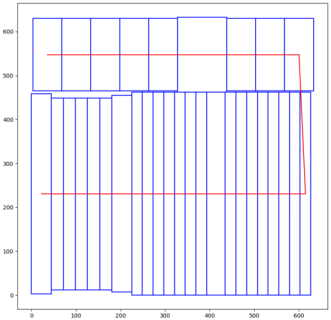
Slicing Placement:

Max_time: 65016

Wire_length: 2650.5

Adapter_cost: 18

Score: 93321



Our Final Method:

Max_time: 65170

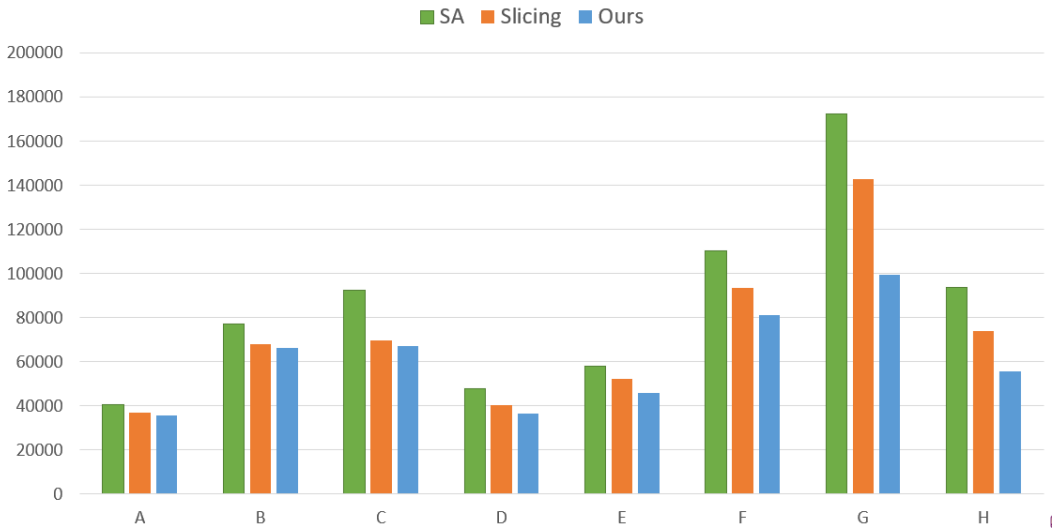
Wire_length: 1489.5

Adapter_cost: 12

Score: 81265

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

Comparisons with Conventional Floorplanning Heuristics

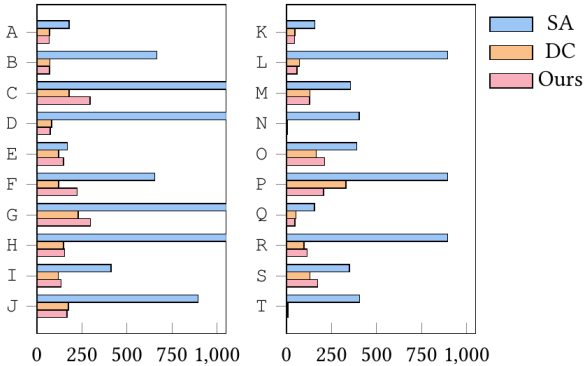


Performance comparisons with SA and DC placers on 8 public benchmarks.

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, 2]

	GigaPlacer				CUPID Placer				SA Placer				DC Placer				Ours			
Case	MT	WL	AC	Score*	MT	WL	AC	Score*	MT	WL	AC	Score*	MT	WL	AC	Score*	MT	WL	AC	Score
A	35280	1186	15	1.02	35280	2047	13	1.04	37044	3611.5	11	1.14	35280	1565	12	1.03	34496	1314	15	**35810**
B	63504	3660	16	1.02	65856	4905	17	1.07	70560	6657	20	1.17	64512	3450	22	1.03	63504	2639.5	18	**66143.5**
C	64512	2471.5	217	1	65772	4278	281	1.05	76608	15696	69	1.38	63504	6308	163	1.04	64512	2408	185	**66920**
D	33712	2078.5	134	**0.98**	34944	3071.5	89	1.04	38304	9327.5	44	1.31	34048	6100.5	100	1.1	33712	2722	123	36434
E	39312	563	12	1	39690	590	16	1.03	36288	2080.5	7	1.26	35280	1565	12	1.13	39312	562	11	**46032**
F	66010	1489.5	18	1.02	70560	1475	14	1.07	76608	3237	15	1.36	65016	2650.5	18	1.15	65170	1489.5	12	**81265**
G	64512	2494.5	149	1.05	69888	2508	141	1.1	91728	7784	29	1.74	63504	6308	163	1.44	64512	2508.5	98	**99397**
H	39312	1033.5	60	1	43008	893	115	1.14	47040	4450	21	1.69	36400	2654	108	1.33	39520	1104.5	49	**55465**
I	49392	1288	13	1	52920	612	13	1.01	56448	3790	16	1.31	49392	1741.5	17	1.03	52136	617.5	13	**54606**
J	50176	1793	164	**0.95**	57792	1117.5	286	1.03	63504	8009.5	52	1.59	49392	4294	210	1.11	50274	2472.5	69	60164
K	252	423	7	1.4	504	400	14	1.51	576	236	3	1.09	828	267	10	1.36	432	240	3	**1392**
L	252	789	142	1.72	504	774	114	1.82	1280	910.5	60	2.49	1764	785	113	2.48	864	279	18	**1980**
M	2260992	3899	41	1.02	2336256	5100	67	1.06	2359296	9359	24	1.08	2276350	4313	58	1.03	2211840	3610.5	41	**2226282**
N	1651	437.5	8	1	1599	448.5	9	1	2268	707.5	0	1.5	1911	904	13	1.62	1482	480.5	10	**3404**
O	54720	614	19	1.06	52920	612	13	1.01	63504	1202	6	1.39	57624	649	12	1.08	52136	617.5	13	**82036**
P	60270	1096.5	78	1	66528	2273	102	1.46	115101	4015	24	2.1	63504	2519.5	134	1.6	57792	1101	85	**135832**
Q	252	423	7	2.01	504	400	14	2.23	1152	178	1	**0.87**	2898	171.5	8	1.31	882	166	6	9922
R	252	789	11	1.37	504	774	114	2.92	1372	1443	30	2.69	14112	480.5	53	2.06	8064	259	20	**26424**
S	2396160	1349	47	**0.93**	2396160	1899.5	65	1	2495376	3551	25	1.24	2276350	4313	58	1.27	2396300	1897.5	48	3174500
T	1651	437.5	8	1.59	2015	367.5	9	1.45	5720	555.5	0	1.99	6080	521.5	13	2.29	4102	208.5	4	**14042**
Avg^*_h				1.25				1.46				1.61				1.58				**1.00**
Avg^*				1.16				1.30				1.52				1.37				**1.00**

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.