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
Outline

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations
Outline

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations
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

- **4 execution arguments**: \((h, w, c, k)\) ⇒ variables to be determined.
Kernel Evaluation

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

\[
\text{convperf}(H, W, R, S, C, K, T, U; h, w, c, k) = \{
\begin{align*}
\text{height} & = h \times w \times (c + 1) \\
\text{width} & = 3 \times k \\
\text{time} & = \text{ceil}(\frac{H}{h}) \times \text{ceil}(\frac{W}{w}) \times \text{ceil}(\frac{C}{c}) \times \text{ceil}(\frac{K}{k}) \times \frac{RS}{T^2} \\
\text{mem} & = \frac{C}{c} \times \frac{K}{k} \times RS + \frac{W + S - 1}{w} \times \frac{H + R - 1}{h} \times \frac{K}{k}
\end{align*}
\}
\]
Kernel Evaluation

For a certain type of kernel that contains $n$ convs

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

\[
\text{blockperf}(TP, H, W, F; h, w, c_1, ..., c_n, k_1, ..., k_n) = \left\{ \begin{array}{l}
\text{conv}_i = \text{convperf}(H_i, W_i, R_i, S_i, C_i, K_i, T_i, U_i; h, w, c_i, k_i), \quad \forall \ i \in \{1, ..., n\} \\
\text{height} = \max_{1 \leq i \leq n} \text{conv}_i.\text{height}, \quad \text{width} = \sum_{i=1}^{n} \text{conv}_i.\text{width} \\
\text{time} = \max_{1 \leq i \leq n} \text{conv}_i.\text{time}, \quad \text{mem} = \max_{1 \leq i \leq n} \text{conv}_i.\text{mem} \\
\end{array} \right. \]

(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.

\[
\text{cost}_{\text{adapter}} = 1(h_{out}! = h_{in}) + 1(w_{out}! = w_{in}) + 1(c_{out,n} \text{ or } \min(c_{out,n}, k_{out,n})! = c_{in,1})
\]
Overview of Proposed Flow

Initialize best_solution and best_time. Set lower_bound = 0, upper_bound = MAX_INT.

\[ \text{lower_bound} + \text{min_gap} \leq \text{upper_bound} \]

Yes

target_time = \([\text{lower_bound} + \text{upper_bound}] / 2\)

Kernel candidate generation under target_time.

Data-path aware kernel placement under target_time.

Have a legal solution?

No

Update best_solution and best_time if needed. Set upper_bound = target_time.

Set lower_bound = target_time.

Neighbor-range search based on best_time.

Post refinement on best_solution.

Output best_solution.

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

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations
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 \(( < \frac{633}{2})\).
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

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

is no worse than

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

with regard to height, width, time and memory.
Optimization View

Solving The Optimal width For height = η (η = 1, ..., 633)

Minimize: width
\[ h, w, c, k_1, ..., k_x \]

Such that: height = h \times w \times (c + 1) = η

\[
width = \sum_{j=1}^{x} 3 \times k_j
\]

\[
time = \max_{1 \leq j \leq x} \text{ceil}\left(\frac{H_j}{h}\right)\text{ceil}\left(\frac{W_j}{w}\right)\text{ceil}\left(\frac{C_j}{c}\right)\text{ceil}\left(\frac{K_j}{k_j}\right)\frac{R_j S_j}{T_j^2}
\]

\[ \leq \text{target\_time} \]

\[
mem = \max_{1 \leq j \leq x} \frac{C_j K_j R_j S_j}{c k_j} + \frac{(W_j + S_j - 1)(H_j + R_j - 1)K_j}{whk_j}
\]

\[ \leq \text{memory\_limit} \]
Method to Solve It

- Factorize $\eta$ 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 $k$s.

Getting the $k$s

For $j = 1, ..., x$:

$$k^t_j = \text{ceil}(\text{ceil}(\frac{H_j}{h})\text{ceil}(\frac{W_j}{w})\text{ceil}(\frac{C_j}{c})\frac{R_j S_j K_j}{T_j^2 \times \text{target\_time}}))$$

$$k^m_j = \text{ceil}(\frac{C_j K_j R_j S_j}{c \times \text{memory\_limit}} + \frac{(W_j + S_j - 1)(H_j + R_j - 1)K_j}{wh \times \text{memory\_limit}}))$$

$$k_j = \max(k^t_j, k^m_j)$$
Final Pruning

An example solution of kernel sizing.
Outline

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations
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.
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 $ABCEDFGH$. 
function Placement(next_index, target_time, floor_height)

H_k \leftarrow \text{a sorted height set of all the kernel candidates}

for each height \( h \) in \( H_k \) do

if \( h + \text{floor.height} > \text{chip.height} \) then

break

end if

end for

\( \text{wIdle} \leftarrow \text{chip.width} \)

\( \text{max.height} \leftarrow 0 \)

for \( i = \text{next.index}, ..., \text{num.kernel} \) do

\( \text{w}_i \leftarrow \text{minimum width of the } i^{th} \text{ kernel's candidates meeting the requirements of } \text{target.time} \text{ and } h \)

\( \text{h}_i \leftarrow \text{the corresponding height of } \text{w}_i \)

if \( \text{w}_i > \text{wIdle} \) then

\( i \leftarrow i - 1 \)

break

else

\( \text{wIdle} \leftarrow \text{wIdle} - \text{w}_i \)

\( \text{max.height} \leftarrow \max(\text{max.height}, \text{h}_i) \)

end if

end for

if \( i < \text{next.index} \) then

continue

end if

Place the kernels of indices from \( \text{next.index} \) to \( i \) in a row on \( \text{floor.height} \)

if \( i \equiv \text{num.kernel} \) then

Update the best solution if needed

else

\( \text{floor.height} \leftarrow \text{floor.height} + \text{max.height} \)

Placement(\( i, \text{target.time}, \text{floor.height} \))

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

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations
Protocol Cost Optimization

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.
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

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

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

- **Let new** \( ker_{i,j}.height = floor_i.height = (ker_{i,j}.height + \Delta height_{i,j}) \), a new \( c_{\text{max}} \) can be uniquely determined by a given reference pair \((h_{\text{ref}}, w_{\text{ref}})\)

\[
c_{\text{max}} = floor_i.height / (h_{\text{ref}} * w_{\text{ref}}) - 1.
\]

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

\[
c_1 = ... = c_m = c_{\text{max}} = floor_i.height / (h_{\text{ref}} * w_{\text{ref}}) - 1.
\]

- **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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>C</td>
<td>234</td>
<td>185</td>
</tr>
<tr>
<td>D</td>
<td>139</td>
<td>123</td>
</tr>
<tr>
<td>E</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>F</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>G</td>
<td>221</td>
<td>98</td>
</tr>
<tr>
<td>H</td>
<td>77</td>
<td>49</td>
</tr>
<tr>
<td>I</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>J</td>
<td>193</td>
<td>69</td>
</tr>
<tr>
<td>K</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>140</td>
<td>18</td>
</tr>
<tr>
<td>M</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>N</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>O</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>P</td>
<td>154</td>
<td>85</td>
</tr>
<tr>
<td>Q</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>R</td>
<td>68</td>
<td>20</td>
</tr>
<tr>
<td>S</td>
<td>60</td>
<td>48</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Avg.</td>
<td>71.95</td>
<td>42.05</td>
</tr>
</tbody>
</table>
Outline

Overview

Kernel Sizing

Data-path-aware Kernel Placement

Protocol Optimization

Experimental Evaluations
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).
# Experimental Results on ISPD-20 Suite [1, 2]

<table>
<thead>
<tr>
<th>Case</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
<th>MT</th>
<th>WL</th>
<th>AC</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>35280</td>
<td>1186</td>
<td>15</td>
<td>1.02</td>
<td>35280</td>
<td>2047</td>
<td>13</td>
<td>1.04</td>
<td>37044</td>
<td>3611.5</td>
<td>11</td>
<td>1.14</td>
<td>35280</td>
<td>1565</td>
<td>12</td>
<td>1.03</td>
<td>34496</td>
<td>1314</td>
<td>15</td>
<td>35810</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>63504</td>
<td>3660</td>
<td>16</td>
<td>1.02</td>
<td>65856</td>
<td>4905</td>
<td>17</td>
<td>1.07</td>
<td>70560</td>
<td>6657</td>
<td>20</td>
<td>1.17</td>
<td>64512</td>
<td>3450</td>
<td>22</td>
<td>1.03</td>
<td>63504</td>
<td>2639.5</td>
<td>18</td>
<td>66143.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>64512</td>
<td>2471.5</td>
<td>217</td>
<td>1</td>
<td>65772</td>
<td>4278</td>
<td>281</td>
<td>1.05</td>
<td>76608</td>
<td>15696</td>
<td>69</td>
<td>1.38</td>
<td>63504</td>
<td>6308</td>
<td>163</td>
<td>1.04</td>
<td>64512</td>
<td>2408</td>
<td>185</td>
<td>66920</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>33712</td>
<td>2078.5</td>
<td>134</td>
<td>0.98</td>
<td>34944</td>
<td>3071.5</td>
<td>89</td>
<td>1.04</td>
<td>38304</td>
<td>9327.5</td>
<td>44</td>
<td>1.31</td>
<td>34048</td>
<td>6100.5</td>
<td>100</td>
<td>1.1</td>
<td>33712</td>
<td>2722</td>
<td>123</td>
<td>36434</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>39312</td>
<td>563</td>
<td>12</td>
<td>1</td>
<td>39690</td>
<td>590</td>
<td>16</td>
<td>1.03</td>
<td>36288</td>
<td>2080.5</td>
<td>7</td>
<td>1.26</td>
<td>35280</td>
<td>1565</td>
<td>12</td>
<td>1.13</td>
<td>39312</td>
<td>562</td>
<td>11</td>
<td>46032</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>66010</td>
<td>1489.5</td>
<td>18</td>
<td>1.02</td>
<td>70560</td>
<td>1475</td>
<td>14</td>
<td>1.07</td>
<td>76608</td>
<td>3237</td>
<td>15</td>
<td>1.36</td>
<td>65016</td>
<td>2650.5</td>
<td>18</td>
<td>1.15</td>
<td>65170</td>
<td>1489.5</td>
<td>12</td>
<td>81265</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>64512</td>
<td>2494.5</td>
<td>149</td>
<td>1.05</td>
<td>69888</td>
<td>2508</td>
<td>141</td>
<td>1.1</td>
<td>91728</td>
<td>7784</td>
<td>29</td>
<td>1.74</td>
<td>63504</td>
<td>6308</td>
<td>163</td>
<td>1.44</td>
<td>64512</td>
<td>2508.5</td>
<td>98</td>
<td>99397</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>39312</td>
<td>1033.5</td>
<td>60</td>
<td>1</td>
<td>43008</td>
<td>893</td>
<td>115</td>
<td>1.14</td>
<td>47040</td>
<td>4450</td>
<td>21</td>
<td>1.69</td>
<td>36400</td>
<td>2654</td>
<td>108</td>
<td>1.33</td>
<td>39520</td>
<td>1104.5</td>
<td>49</td>
<td>55465</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>49392</td>
<td>1288</td>
<td>13</td>
<td>1</td>
<td>52920</td>
<td>612</td>
<td>16</td>
<td>1.01</td>
<td>56448</td>
<td>3790</td>
<td>16</td>
<td>1.31</td>
<td>49392</td>
<td>1741.5</td>
<td>17</td>
<td>1.03</td>
<td>52136</td>
<td>617.5</td>
<td>13</td>
<td>54606</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>50176</td>
<td>1793</td>
<td>164</td>
<td>0.95</td>
<td>57792</td>
<td>1117.5</td>
<td>286</td>
<td>1.03</td>
<td>63504</td>
<td>8009.5</td>
<td>52</td>
<td>1.59</td>
<td>49392</td>
<td>4294</td>
<td>210</td>
<td>1.11</td>
<td>50274</td>
<td>2472.5</td>
<td>69</td>
<td>60164</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>252</td>
<td>423</td>
<td>7</td>
<td>1.4</td>
<td>504</td>
<td>400</td>
<td>14</td>
<td>1.51</td>
<td>576</td>
<td>236</td>
<td>3</td>
<td>1.09</td>
<td>828</td>
<td>267</td>
<td>10</td>
<td>1.36</td>
<td>432</td>
<td>240</td>
<td>3</td>
<td>1392</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>252</td>
<td>789</td>
<td>142</td>
<td>1.72</td>
<td>504</td>
<td>774</td>
<td>114</td>
<td>1.82</td>
<td>1280</td>
<td>910.5</td>
<td>60</td>
<td>2.49</td>
<td>1764</td>
<td>785</td>
<td>113</td>
<td>2.48</td>
<td>864</td>
<td>279</td>
<td>18</td>
<td>1980</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>2260992</td>
<td>3899</td>
<td>41</td>
<td>1.02</td>
<td>233625</td>
<td>6100</td>
<td>67</td>
<td>1.06</td>
<td>235929</td>
<td>6939.5</td>
<td>24</td>
<td>1.08</td>
<td>2276350</td>
<td>4313</td>
<td>58</td>
<td>1.03</td>
<td>2211840</td>
<td>3610.5</td>
<td>41</td>
<td>2226282</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>1651</td>
<td>437.5</td>
<td>8</td>
<td>1</td>
<td>1599</td>
<td>448.5</td>
<td>9</td>
<td>1</td>
<td>2268</td>
<td>707.5</td>
<td>0</td>
<td>1.59</td>
<td>1911</td>
<td>904</td>
<td>13</td>
<td>1.62</td>
<td>1482</td>
<td>480.5</td>
<td>10</td>
<td>3404</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>54720</td>
<td>614</td>
<td>19</td>
<td>1.06</td>
<td>52920</td>
<td>612</td>
<td>13</td>
<td>1.01</td>
<td>63504</td>
<td>1202</td>
<td>6</td>
<td>1.39</td>
<td>57624</td>
<td>649</td>
<td>12</td>
<td>1.08</td>
<td>52136</td>
<td>617.5</td>
<td>13</td>
<td>82036</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>60270</td>
<td>1096.5</td>
<td>78</td>
<td>1</td>
<td>66528</td>
<td>2273</td>
<td>102</td>
<td>1.46</td>
<td>115101</td>
<td>4015</td>
<td>24</td>
<td>2.1</td>
<td>63504</td>
<td>2519.5</td>
<td>134</td>
<td>1.6</td>
<td>57792</td>
<td>1101</td>
<td>85</td>
<td>135832</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>252</td>
<td>423</td>
<td>7</td>
<td>2.01</td>
<td>504</td>
<td>400</td>
<td>14</td>
<td>2.23</td>
<td>1152</td>
<td>178</td>
<td>1</td>
<td>0.87</td>
<td>2898</td>
<td>171.5</td>
<td>8</td>
<td>1.31</td>
<td>882</td>
<td>166</td>
<td>6</td>
<td>9922</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>252</td>
<td>789</td>
<td>11</td>
<td>1.37</td>
<td>504</td>
<td>774</td>
<td>114</td>
<td>2.92</td>
<td>1372</td>
<td>1443</td>
<td>30</td>
<td>2.69</td>
<td>14112</td>
<td>480.5</td>
<td>53</td>
<td>2.06</td>
<td>8064</td>
<td>259</td>
<td>20</td>
<td>26424</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>2396160</td>
<td>1349</td>
<td>47</td>
<td>0.93</td>
<td>239610</td>
<td>899.5</td>
<td>65</td>
<td>1</td>
<td>2495376</td>
<td>3551</td>
<td>25</td>
<td>1.24</td>
<td>2276350</td>
<td>4313</td>
<td>58</td>
<td>1.27</td>
<td>2396300</td>
<td>1897.5</td>
<td>48</td>
<td>3174500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>1651</td>
<td>437.5</td>
<td>8</td>
<td>1.59</td>
<td>2015</td>
<td>367.5</td>
<td>9</td>
<td>1.45</td>
<td>5720</td>
<td>555.5</td>
<td>0</td>
<td>1.99</td>
<td>6080</td>
<td>521.5</td>
<td>13</td>
<td>2.29</td>
<td>4102</td>
<td>208.5</td>
<td>4</td>
<td>14042</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Avg_i \* 1.25 \* 1.46 \* 1.61 \* 1.58

Avg_j 1.16 \* 1.30 \* 1.52 \* 1.37

Avg 1.00 \* 1.00
Thanks and Questions?
