Multi-Electrostatics Based Placement for Non-Integer Multiple-Height Cells

Yu Zhang\textsuperscript{1,2}, Yuan Pu\textsuperscript{1}, Fangzhou Liu\textsuperscript{1}, Peiyu Liao\textsuperscript{1}, Keren Zhu\textsuperscript{1}, Kai-Yuan (Kevin) Chao\textsuperscript{4}, Yibo Lin\textsuperscript{2,3*}, Bei Yu\textsuperscript{1*}

\textsuperscript{1}Chinese University of Hong Kong \quad \textsuperscript{2}Peking University
\textsuperscript{3}Institute of Electronic Design Automation, Peking University
\textsuperscript{4}Siemens Digital Industries Software
1 Background

2 Algorithm

3 Experiments
Standard-cell libraries can be developed with different cell heights, enabling a more flexible optimization of area, timing, and power.

- Large cells provide higher pin accessibility, drive strength, and shorter delay time.
- Small cells have smaller areas, pin capacitance, and power consumption.
- C1: at least two cell rows
- C2: even number of rows
- C3: cells placed at sites on rows of the same height
- C4: horizontal spacing
- C5: vertical spacing
- C6: breaker cells insertion
Row-based Placement Flow for NIMH Cells

Observation

- Traditional flow causes significant disruptions in the initial placement results, resulting in inferior wirelength.
Therefore, we propose to

- Adaptively generate regions for each cell type during global placement to identify more desired solutions;
- Introduce a multi-electrostatics-based global placement algorithm to directly solve the global placement problem with NIMH cells.
Overall Flow

- **Circuit Netlists**
- **LEF Library**
  - Random Initial Placement
  - \( \lambda \) Initialization
  - Placement Optimization
  - Legalization
  - Placement Result

- **Gradient Computation**
- **Nesterov's Optimization**
- **\( \lambda \) Update**
  - **OVFL \( < 15\% \)**
    - **Cluster Generation**
    - **Re-optimization**
      - **OVFL\( _c \) \( < 7\% \)**

- 8T cell
- 12T cell
Considering the density constraint for each cell type $c$ as a distinct electrostatic system, we frame the placement problem with NIMH cells as follows:

$$\min_{x,y} \tilde{W}(x,y) \quad \text{s.t.} \quad \Phi_c(x,y) = 0, \ \forall c \in C.$$ (1)

We leverage the augmented Lagrangian method (ALM) to solve this optimization problem:

$$\min_{x,y} f(x,y) = \tilde{W}(x,y) + \sum_{c \in C} \lambda_c (\Phi_c(x,y) + \frac{1}{2} \mu \theta \lambda \Phi_c(x,y)^2),$$ (2)

where $\lambda_c$ represents the density multiplier for each cell type.
• The constrained optimization problem is transformed into an unconstrained optimization problem;

• The ALM formulation can be interpreted as a combination of the multiplier method and the quadratic penalty method.
The $x$-directed gradient of our ALM objective function can be derived as follows:

$$\frac{\partial f(x, y)}{\partial x_i} = \frac{\partial \tilde{W}(x, y)}{\partial x_i} + \lambda_c\left(\frac{\partial \Phi_c(x, y)}{\partial x_i} + \mu \theta \lambda_c(x, y) \frac{\partial \Phi_c(x, y)}{\partial x_i}\right), \forall i \in \mathcal{V}_c. \quad (3)$$

Then, the preconditioned$^2$ gradient would be input into Nesterov’s optimizer$^3$ for a gradient descent step.


Given that the dual function, \( Z(\lambda) = \max f(x, y)|_{\lambda} \), associated with Eq. 2 is not smooth but piecewise linear, we utilize the subgradient method to update \( \lambda \) as,

\[
\lambda^{k+1} \leftarrow \min(\lambda_{\max}, \max(0, \lambda^k + \alpha^k g_{\text{sub}}(\lambda))),
\]

(4)

where \( g_{\text{sub}}(\lambda) = (\ldots, \Phi_c(x, y) + \frac{1}{2} \mu \theta \lambda \Phi_c(x, y)^2, \ldots) \). However, the convergence of the traditional subgradient method highly depends on \( \alpha^k \).
The main idea of the surrogate subgradient\(^4\) method is to obtain \(\alpha^k\) such that distances between Lagrangian multipliers \(\lambda^k\) at consecutive iterations decrease, i.e.,

\[ \|\lambda^{k+1} - \lambda^k\| = \eta^k \|\lambda^k - \lambda^{k-1}\|, \tag{5} \]

where \(0 < \eta^k < 1\). Eq. 4 and Eq. 5 imply

\[ \alpha^k = \eta^k \frac{\alpha^{k-1} \|g_{\text{sub}}(f^{k-1})\|}{\|g_{\text{sub}}(f^k)\|}. \tag{6} \]

---

Considering NIMH constraints, we prioritize clustering cells of the same type together. To achieve this, we introduce pseudo-nets for cells with the same height. The score function $c(i,j)$ in the modified BestChoice clustering algorithm is defined as follows:

$$c(i,j) = \sum_{e \in E_{i,j}} \frac{\omega_e}{a_i + a_j}, \quad (7)$$

where $\omega_e$ is a corresponding edge weight defined as:

$$\omega_e = \begin{cases} 
1, & h_i = h_j \text{ and } e \text{ is a real net}, \\
\frac{1}{e \sqrt{|x_i - x_j| + |y_i - y_j|}}, & h_i = h_j \text{ and } e \text{ is a real net}, \\
1, & h_i = h_j \text{ and } e \text{ is a pseudo-net}, \\
0, & h_i \neq h_j.
\end{cases} \quad (8)$$
We conducted experiments using eight design blocks (sha3, aes_core, des, fpu, des3, mor1kx, jpeg, aes_128) obtained from the OpenCores website.

Table: Statistics of the OpenCores benchmarks

<table>
<thead>
<tr>
<th>Design</th>
<th>#Cells</th>
<th>#Nets</th>
<th>Util (%)</th>
<th>Clock (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha3</td>
<td>1337</td>
<td>1397</td>
<td>69.05</td>
<td>100</td>
</tr>
<tr>
<td>aes_core</td>
<td>4733</td>
<td>4808</td>
<td>69.84</td>
<td>400</td>
</tr>
<tr>
<td>des</td>
<td>18274</td>
<td>18372</td>
<td>67.11</td>
<td>250</td>
</tr>
<tr>
<td>fpu</td>
<td>30495</td>
<td>31225</td>
<td>67.65</td>
<td>270</td>
</tr>
<tr>
<td>des3</td>
<td>58017</td>
<td>58116</td>
<td>67.05</td>
<td>250</td>
</tr>
<tr>
<td>mor1kx</td>
<td>61220</td>
<td>58952</td>
<td>67.32</td>
<td>200</td>
</tr>
<tr>
<td>jpeg</td>
<td>210968</td>
<td>233898</td>
<td>68.49</td>
<td>300</td>
</tr>
<tr>
<td>aes_128</td>
<td>250672</td>
<td>225888</td>
<td>57.29</td>
<td>300</td>
</tr>
</tbody>
</table>
### Experimental Result

**Table:** WNS (ns), TNS (ns), HPWL (10^5um) and CPU Runtime (s) with State-of-the-art Row-based Placers.

<table>
<thead>
<tr>
<th>test case</th>
<th>Cells</th>
<th>ICCAD’21-imp</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8T</td>
<td>12T</td>
<td>Total</td>
</tr>
<tr>
<td>sha3</td>
<td>662</td>
<td>675</td>
<td>1337</td>
</tr>
<tr>
<td>aes_core</td>
<td>2511</td>
<td>2222</td>
<td>4733</td>
</tr>
<tr>
<td>des</td>
<td>8853</td>
<td>9421</td>
<td>18274</td>
</tr>
<tr>
<td>fpu</td>
<td>15266</td>
<td>15229</td>
<td>30495</td>
</tr>
<tr>
<td>des3</td>
<td>29683</td>
<td>28334</td>
<td>58017</td>
</tr>
<tr>
<td>mor1kx</td>
<td>30168</td>
<td>29873</td>
<td>60041</td>
</tr>
<tr>
<td>jpeg</td>
<td>107866</td>
<td>103102</td>
<td>210968</td>
</tr>
<tr>
<td>aes_128</td>
<td>123825</td>
<td>126847</td>
<td>250672</td>
</tr>
<tr>
<td>average ratio</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- On average, our method achieves a 12% reduction in HPWL while exhibiting a remarkable $23.5 \times$ faster runtime.
- In large cases involving over 200,000 standard cells, our method shows up to $42.85 \times$ speedup while delivering better placement solution quality.
- Our method improves 22% and 49% in WNS and TNS, respectively.
THANK YOU!