EIGER
(EIG ALGORITHM IMPLEMENTATION)

Saad Bin Nasir
ECE-6133 Final Project
THE ALGORITHM

- Input = netlist
- Adjacency (A) and degree (D) matrices are computed
- Get laplacian (L) = A-D matrix
- Eigenvector corresponding to 2nd smallest eigenvalue of L gives a 1-dimensional placement of nodes
- Ratio cut heuristic is computed within area skew constraint
- Output = partition corresponding to minimum ratio cut
IMPLEMENTATION

- Matrices => Matlab should be the first choice
- Use power of matlab
  - Vectorization – minimize # of loops required
  - Pre-allocation – saves huge amounts of runtime
- Making “A” and “D” matrices from netlist takes time
- Simplest model (V=149 nodes) took no time!
- 10000 node netlist never finished!
It turns out Matrices are big!

*From Cody Planteen’s slide*
A closer look reveals!

- Special structures in Matrices (Toeplitz, Hermitian, Symmetric, etc)
- Dimensional Analysis!
  - Is it low rank? Seemingly, yes
  - Positive definite? No but Eigenvalues $\geq 0$ so it is semi positive definite
Sparse and Symmetric
Implementation

• Sparse Algebra
  • store only non-zero entries, rest are assumed 0 by default
  • Three vector required for one matrix
    • row vector, column vector and the value vector

• Trade offs
  • saves memory
  • algorithm translation required
    • could be slow
  • matrix operations require more time
  • specialized eigenvalue calculation engine required
Storage Requirement Comparison

Storage gain = f (sparsity) = f (# of non-zero entries)
Implementation Trade-off

- Use linear algebra for speed
- Use sparse algebra for space
- Mixed approach gives the required balance
  - Trade-off memory for precision
  - Laplacian matrix requires large memory
  - Perform sparse computations till eigenvector calculation
  - Precision needed for eigenvalue/eigenvector calculation (Matlab uses ARPACK)
- Perform ratio cut heuristics on full Laplacian matrix (stored in lower precision, 32-bit floating point)
1-Dimensional placement

Validity: \[ \sum (\text{placement value})^2 = 1 \]
ratio cut/cut size plot

![Graph showing ratio cut vs cut size for left partition, with a total of 15406 nodes.](Industry3.hgr)
## Results!

<table>
<thead>
<tr>
<th>Design</th>
<th>Total Cells</th>
<th>Left Partition</th>
<th>Right Partition</th>
<th>Area Skew(%)</th>
<th>Run Time(s)</th>
<th>Cut size</th>
<th>Ratio Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>fract</td>
<td>149</td>
<td>75</td>
<td>74</td>
<td>0.67</td>
<td>0.42</td>
<td>23.75</td>
<td>4.3e-3</td>
</tr>
<tr>
<td>p1</td>
<td>902</td>
<td>455</td>
<td>447</td>
<td>0.88</td>
<td>0.569</td>
<td>84.7857</td>
<td>4.16e-4</td>
</tr>
<tr>
<td>structP</td>
<td>1952</td>
<td>985</td>
<td>967</td>
<td>0.92</td>
<td>1.19</td>
<td>174.25</td>
<td>1.83e-4</td>
</tr>
<tr>
<td>p2</td>
<td>3029</td>
<td>1529</td>
<td>1500</td>
<td>0.95</td>
<td>24.34</td>
<td>313.29</td>
<td>1.36e-4</td>
</tr>
<tr>
<td>biomedP</td>
<td>6514</td>
<td>3289</td>
<td>3225</td>
<td>0.98</td>
<td>49.46</td>
<td>1.85e+3</td>
<td>1.74e-4</td>
</tr>
<tr>
<td>industry2</td>
<td>13419</td>
<td>6709</td>
<td>6710</td>
<td>0.0074</td>
<td>61.85</td>
<td>720.5</td>
<td>1.6e-5</td>
</tr>
<tr>
<td>industry3</td>
<td>15406</td>
<td>7780</td>
<td>7626</td>
<td>0.99</td>
<td>78.64</td>
<td>1.86e+3</td>
<td>3.14e-4</td>
</tr>
<tr>
<td>ibm01</td>
<td>14111</td>
<td>7125</td>
<td>6986</td>
<td>0.98</td>
<td>43.85</td>
<td>380.6</td>
<td>7.64e-6</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>Design</th>
<th>Runtime(s) (this work)</th>
<th>Runtime(s) (Cody’s Work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fract</td>
<td>0.42</td>
<td>0</td>
</tr>
<tr>
<td>p1</td>
<td>0.569</td>
<td>0</td>
</tr>
<tr>
<td>structP</td>
<td>1.19</td>
<td>8</td>
</tr>
<tr>
<td>p2</td>
<td>24.34</td>
<td>28</td>
</tr>
<tr>
<td>biomedP</td>
<td>49.46</td>
<td>279</td>
</tr>
<tr>
<td>Industry2</td>
<td>61.85</td>
<td>2094</td>
</tr>
<tr>
<td>industry3</td>
<td>78.64</td>
<td>-</td>
</tr>
<tr>
<td>ibm01</td>
<td>43.85</td>
<td>2153</td>
</tr>
</tbody>
</table>

For smaller designs - linear matrix computations are fast
For larger designs – using power of sparsity pays dividends
Conclusion and Extensions

• Significant gains in runtime and storage requirements achieved using a combination of linear and sparse algebra

• Optimized package can be written in a language like C

• Gives motivation and insights in to exploring the usage of sparse algebra based mathematics into solving the problems of physical design of VLSI

• Exciting mathematics being developed for low rank matrices
  • Randomization, compressed sensing, etc
Questions?

Thank you