EIGER
(EIG ALGORITHM IMPLEMENTATION)

Saad Bin Nasir

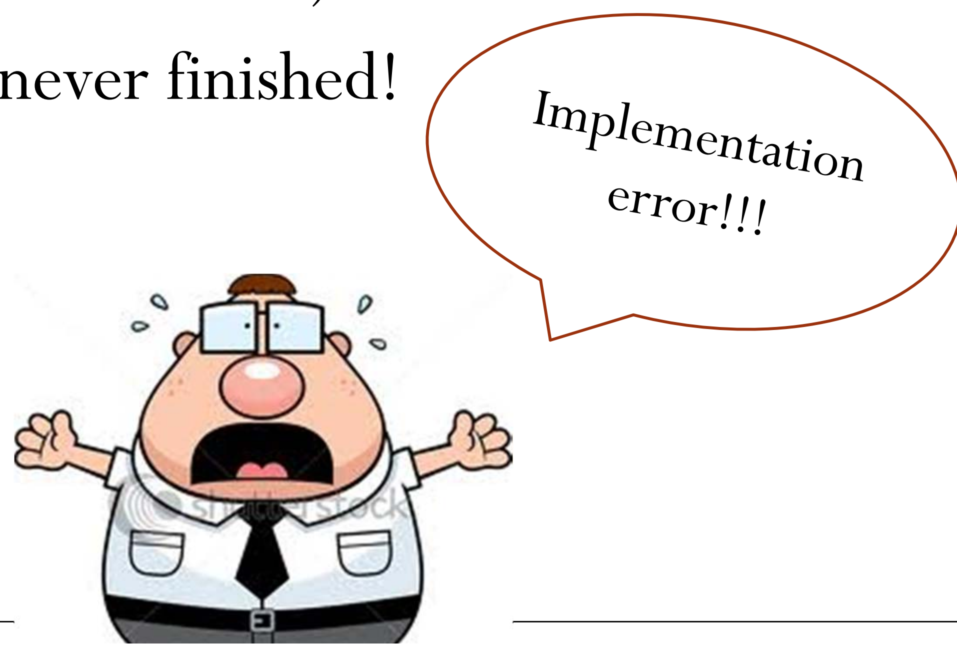
ECE-6133 Final Project

THE ALGORITHM

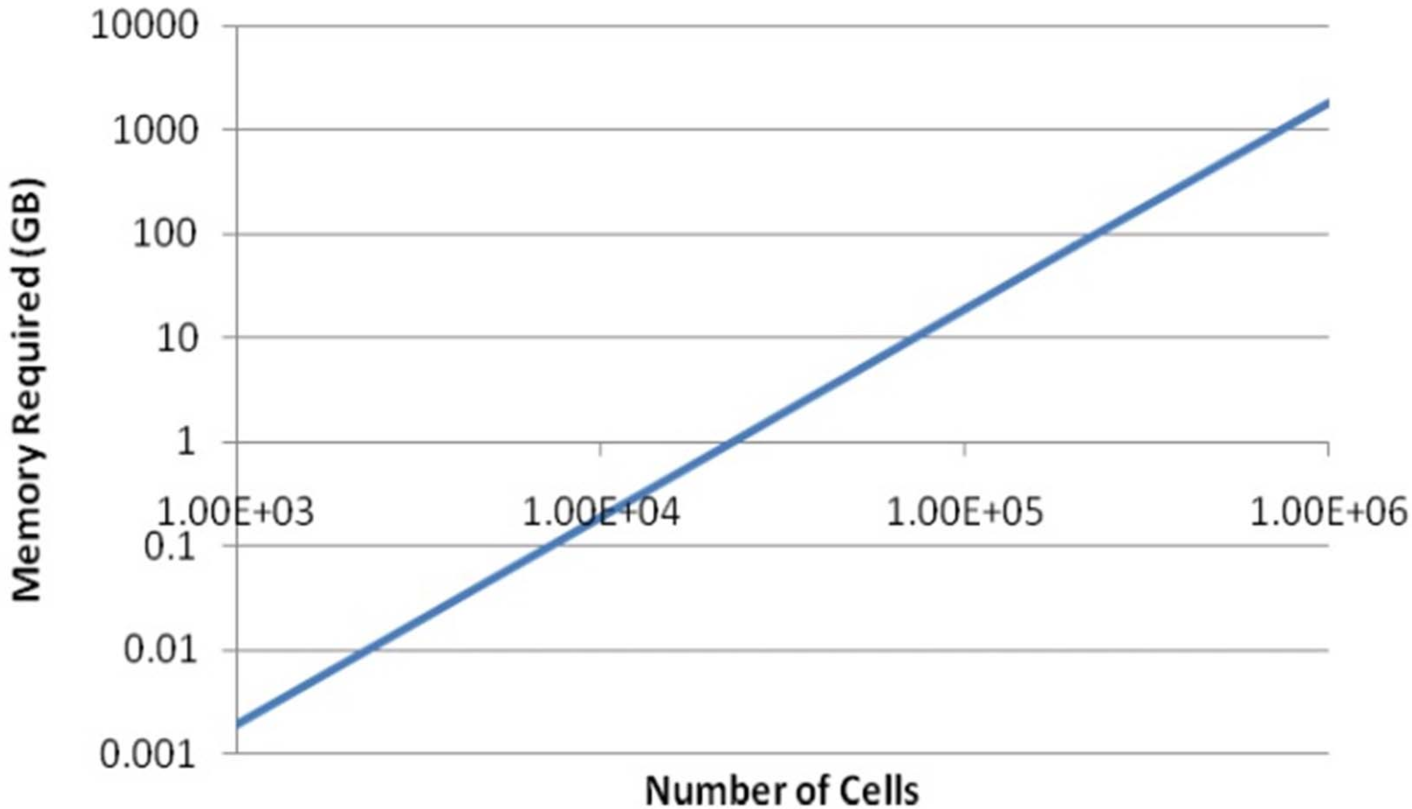
- Input = netlist
- Adjacency (A) and degree (D) matrices are computed
- Get laplacian (L) = A-D matrix
- Eigenvector corresponding to 2nd smallest eigenvalue of L gives a 1-dimensional placement of nodes
- ratio cut heuristic is computed within area skew constraint
- Output = partition corresponding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

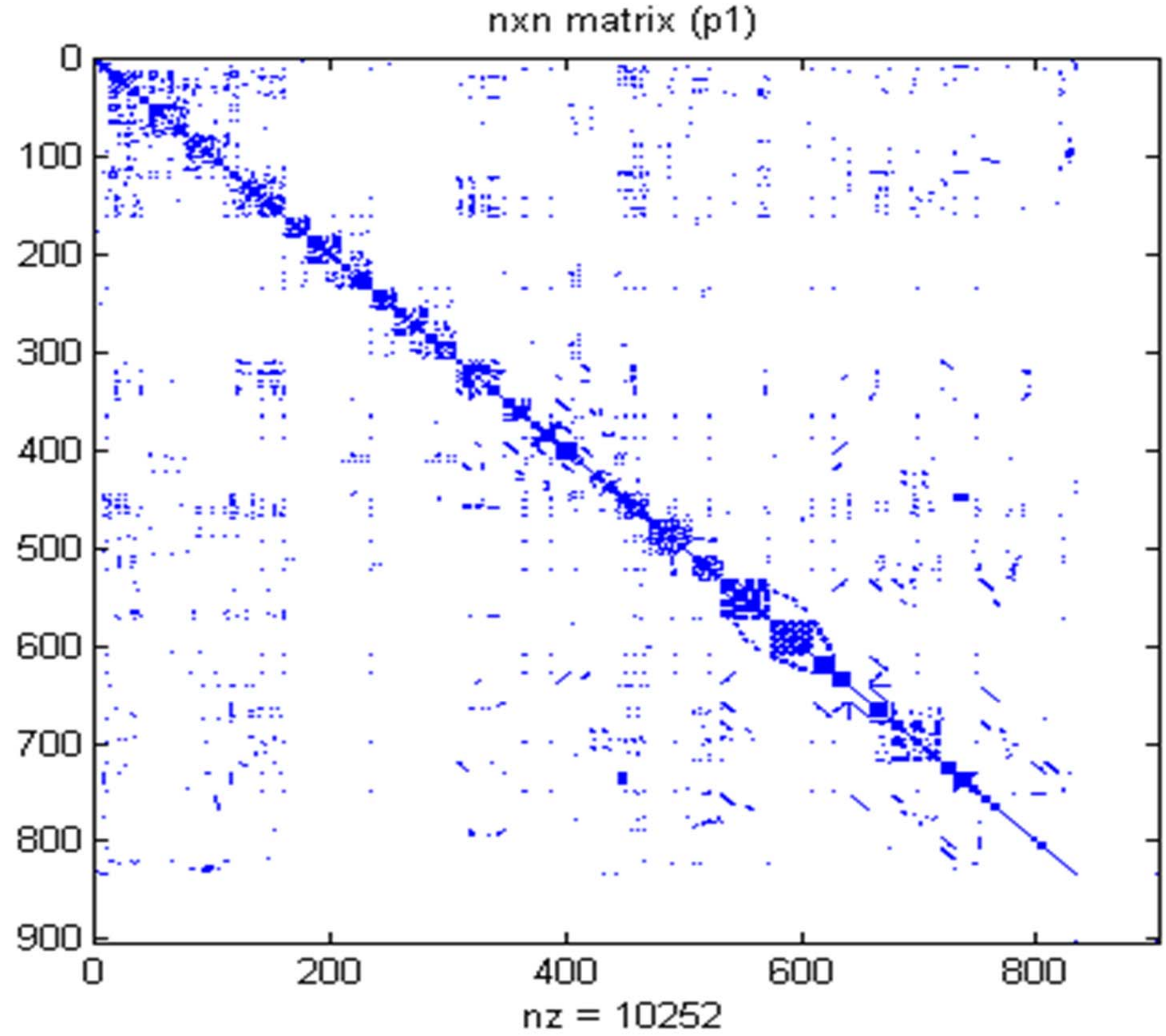


*From Cody Planteen's slide

A closer look reveals!

- Special structures in Matrices (Toeplitz, Hermition, Symmetric,etc)
- Dimensional Analysis!
  - Is it low rank? Seemingly, yes
  - Positive definite? No but Eigenvalues ≥ 0 so it is semi positive definite

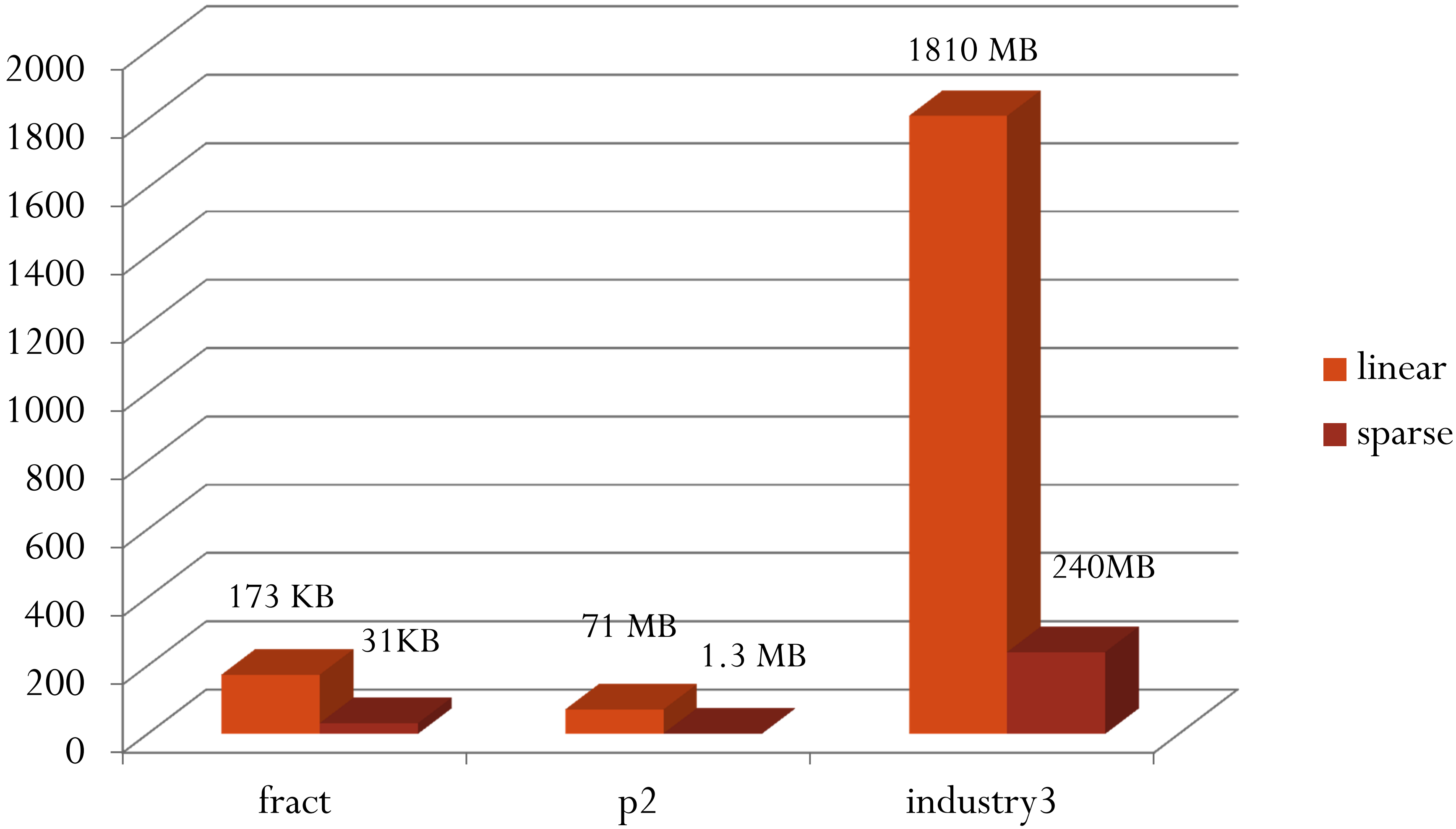
Sparse and Symmetric



Implementation

- Sparse Algebra
  - store only non-zero enteries, 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

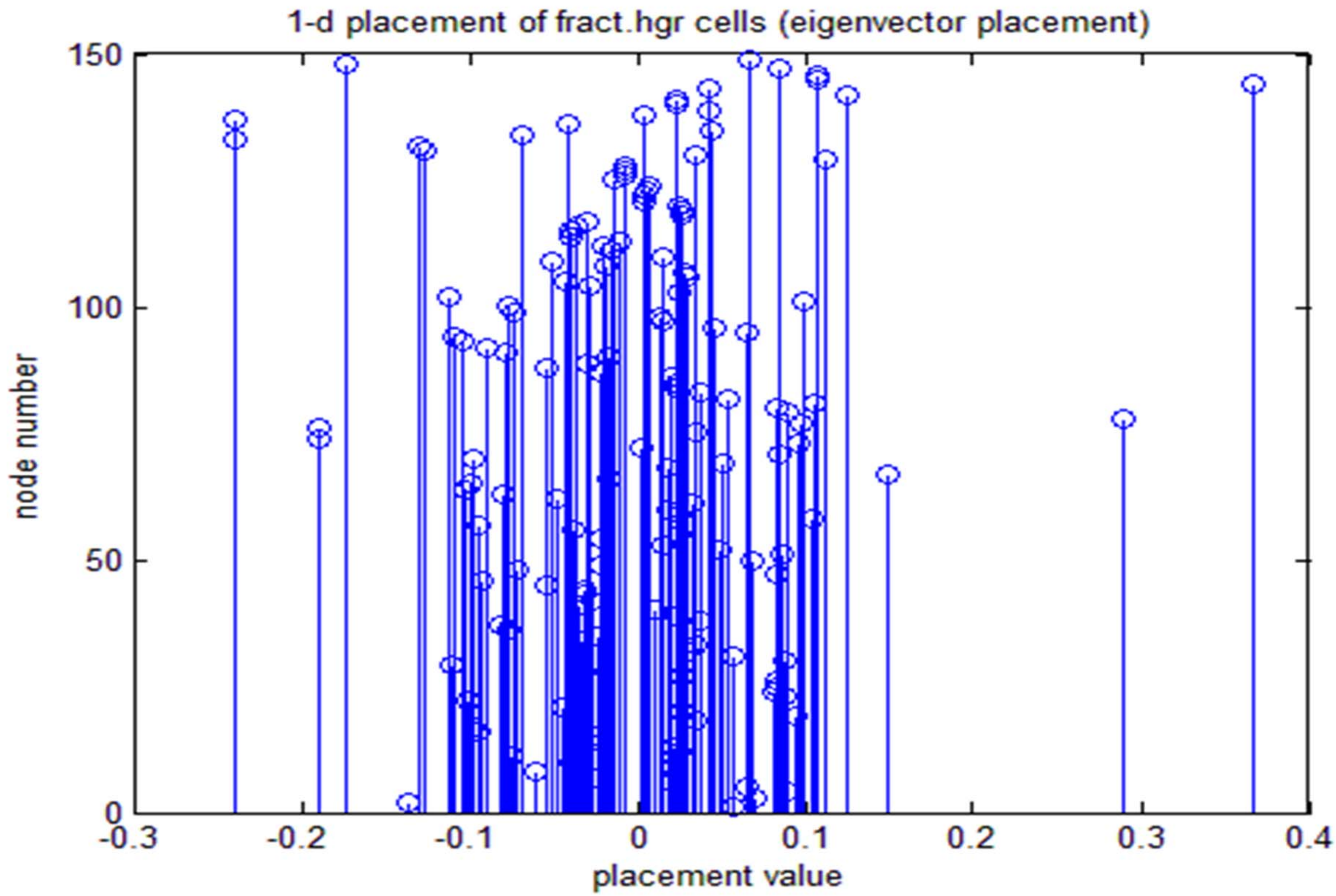


Storage gain = f (sparsity) = f (# of non-zero enteries)

Implementation Trade-off

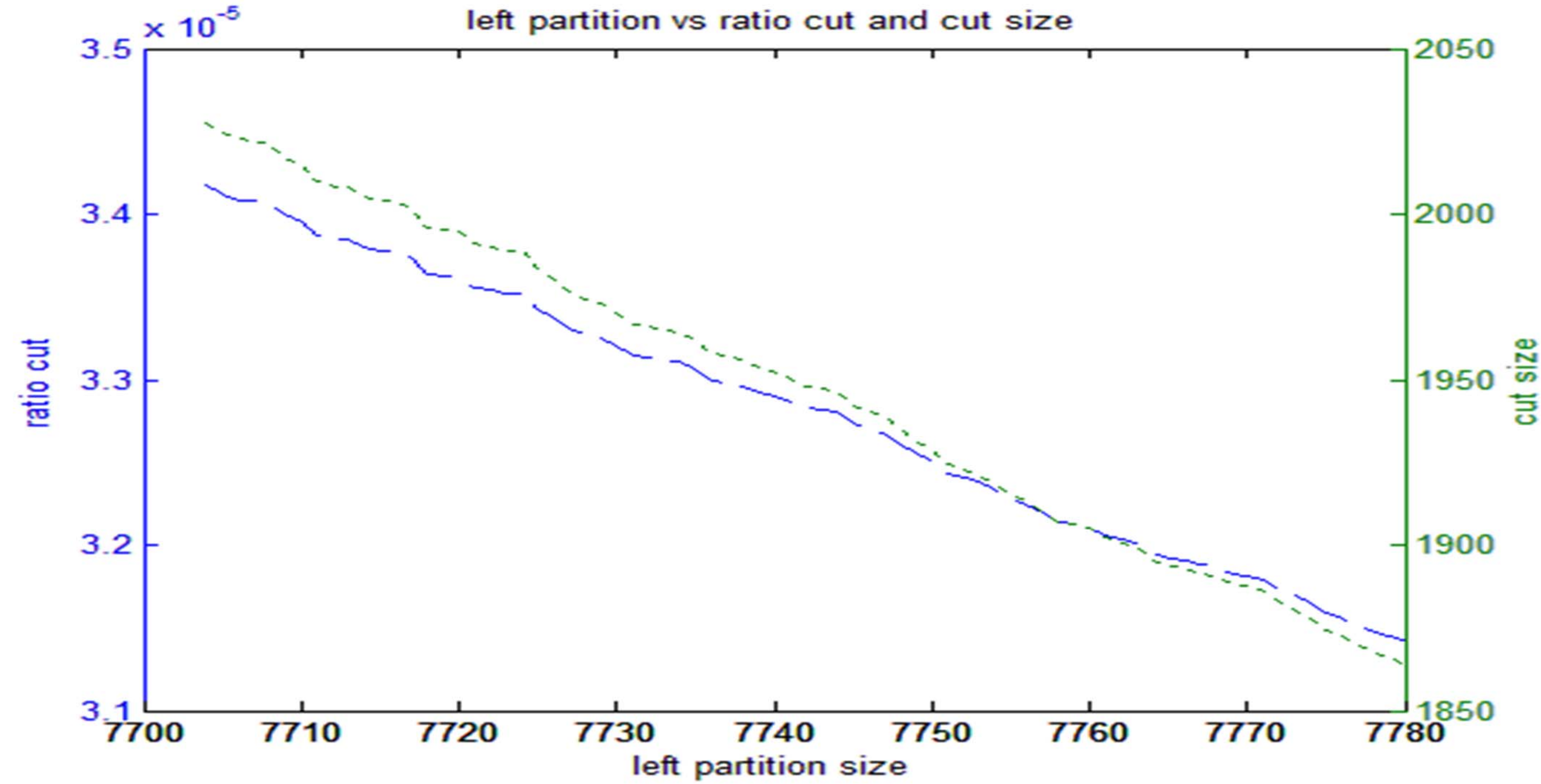
- Use linear algebra for speed
- Use sparse algebra for space
- mixed approach gives the required balance
  - Trade-off memory for precision
  - laplacian matrix requires large memory
  - Perform sparse computations till eigenvector calculation
  - Precision needed for eigenvalue/eigenvector calculation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

1-Dimensional placement



Validity : $\sum$ (placement value)$^2 = 1$

ratio cut/cut size plot



Industry3.hgr

Total nodes = 15406

Results!

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

Comparison

Design	Runtime(s) (this work)	Runtime(s) (Cody's Work)
fract	0.42	0
p1	0.569	0
structP	1.19	8
p2	24.34	28
biomedP	49.46	279
Industry2	61.85	2094
industry3	78.64	-
ibm01	43.85	2153

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