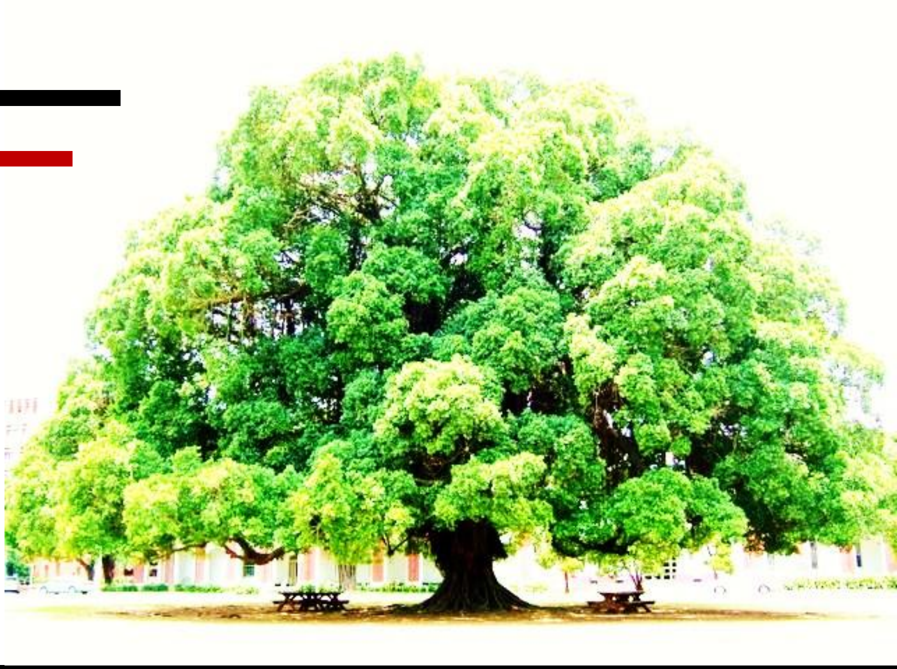
A Flexible Fixed-outline Floorplanning Methodology for Mixed-size Modules

Presenter : Kai-Chung Chan,

Co-Worker : Chao-Jam Hsu,

Advisor : Jai-Ming Lin

Department of Electrical Engineering,
National Cheng Kung University,
Tainan, Taiwan

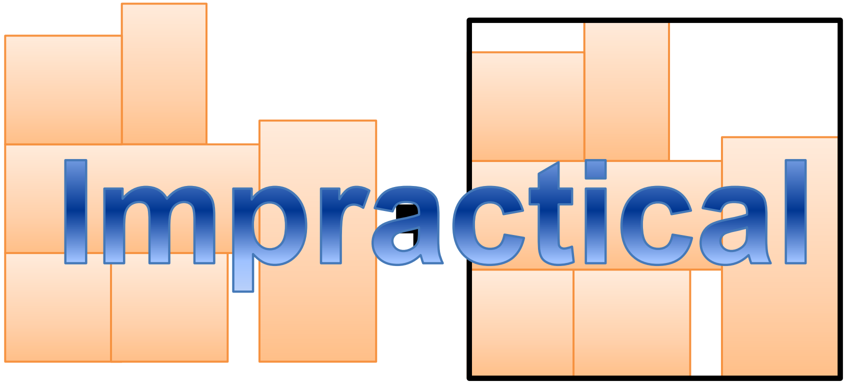


Outline

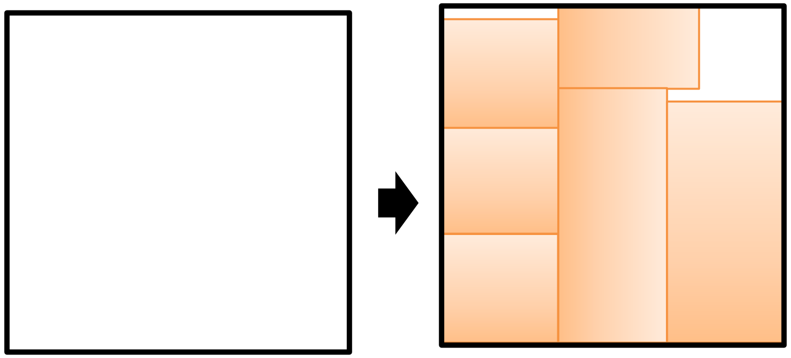
- Introduction
- Problem Formulation
- Algorithm Flow
  - Global Distribution Stage
  - Legalization Stage
- Experimental Results
- Conclusions & Future Works

Fixed-outline Floorplanning

- Floorplanning still is a crucial stage in physical for a system-on-chip(SOC) design.
- Two kinds of problems :
  - Outline-free floorplanning: determine chip outline after floorplanning stage
  - Fixed-outline floorplanning: peferm floorplanning in the designated region



Outline-free



Fixed-outline

Floorplanning Consideration of Other Issues

- As the design complexity continue increasing, several issues need to be considered during floorplainning such as routability or thermal effect.
- Most of existing floorplanners concentrate on reducing wirelength
  - Difficult to extend them to consider other issues
- It is important to have a floorplanner which can be used to consider different requirements and meet the fixed-outline constraint.
- In this paper, we propose a floorplanning methodology which is easily extended to consider other issues.

Problem Formulation

- **Input**
  - (W_f, H_f) : the width and height of a specified region
  - M : a set of modules including soft modules and hard modules.
    - A_i : the area of module i
    - w_i : the width of module i
    - h_i : the height of module i
    - **Aspect ratio (w_i / h_i)** constraint for each soft module
  - N : a set of nets connecting the module set M

Problem Formulation (cont)

Output

- (x_i , y_i) : The left-bottom coordinates of each module *i*
- (w_i , h_i) : The dimensions of each module *i*

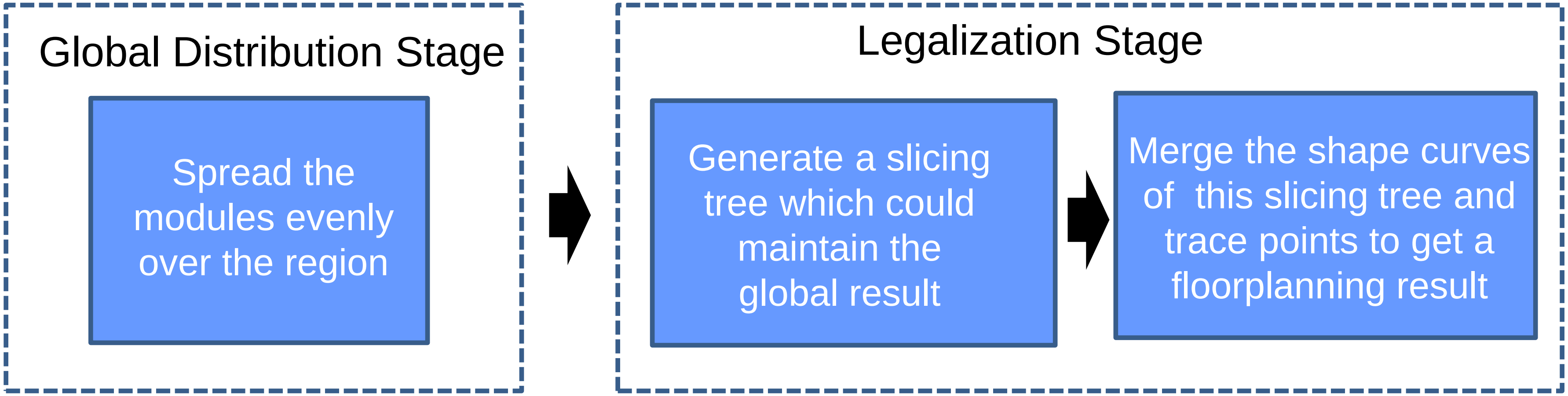
Objective

- **minimize total wirelength under the following constraints:**

  1 : There are no overlaps between the modules

  2 : All modules are placed in the fixed-outline region

Algorithm Flow



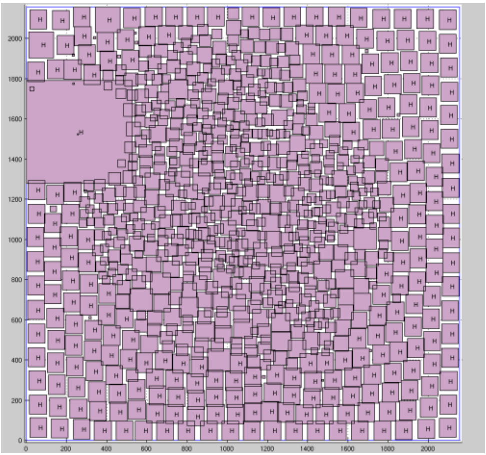
Global Distribution Stage

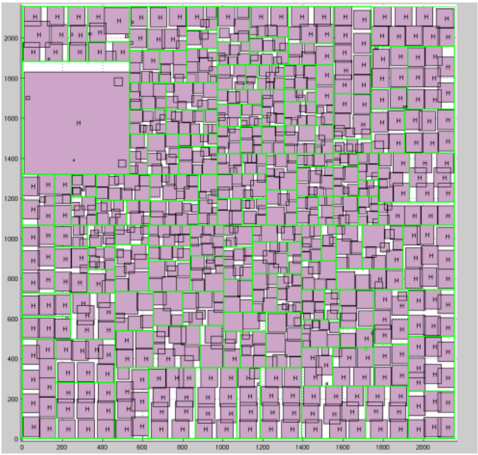
Spread the modules evenly over the region

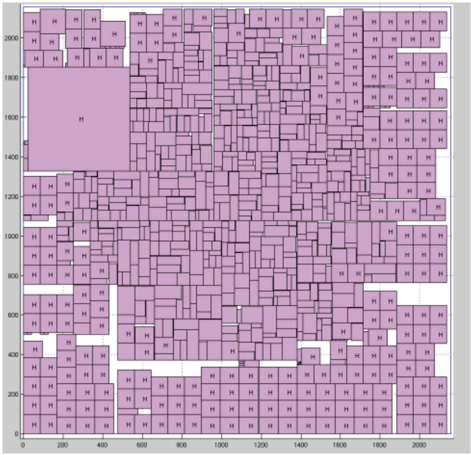
Legalization Stage

Generate a slicing tree which could maintain the global result

Merge the shape curves of this slicing tree and trace points to get a floorplanning result

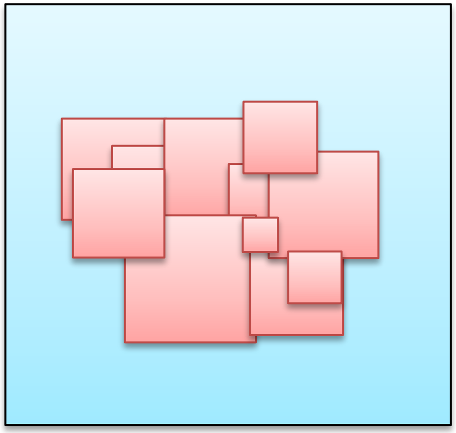




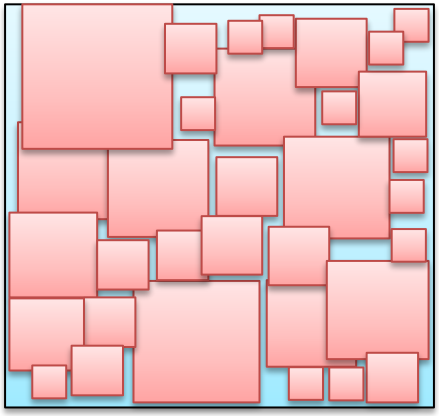


Global Distribution Stage

- It intents to spread the modules over the specified region and minimize total wirelength.



Initial floorplanning



Global distribution result

- This paper adopts a mathematical analysis approach to achieve this target.

Mathematical Formulation

- Use the following mathematical formulation to spread modules over the region

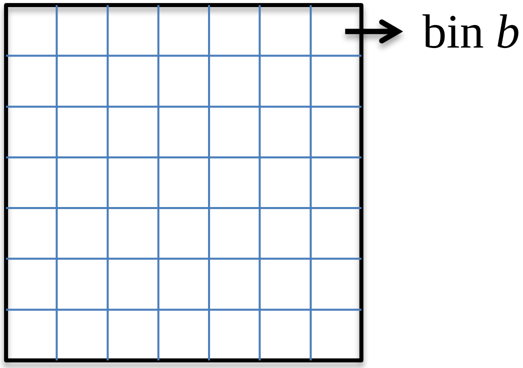
min $W(x,y)$ - - - - - - - - - - - - - - - - > Wirelength term

s.t. $D_b \leq M_b$, for each bin b - - - - > Density term

$W(x,y)$: total wirelength

D_b : the total module area in bin b

M_b : the maximal allowable
module area in bin b



Solve Mathematical Formulation

- Transform the formulation into the following differentiable function:

$$\overset{\textbf{log-sum-exp}}{\boxed{\min W(x,y)}} + \overset{\textbf{bell-shape}}{\boxed{\lambda\Sigma_b(D_b(x,y)-M_b)^2}}$$

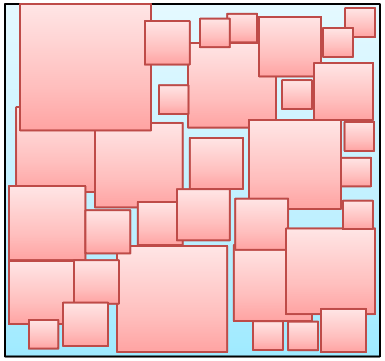
λ : the ratio weight between density and wirelength

  - It can be solved by conjugate-gradient method.

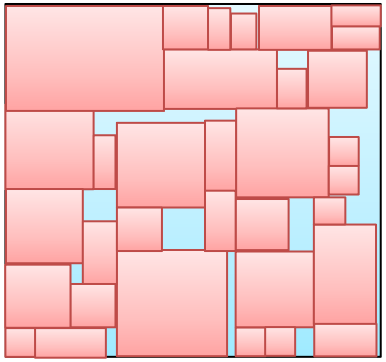
- Any approach which can spread modules over the specified region can be used in this stage.
  - It is suitable for other functions which consider other issues.

Legalization Stage

- Remove all overlaps by changing the locations and dimensions of modules
- Move module as little as possible to keep the good result of global



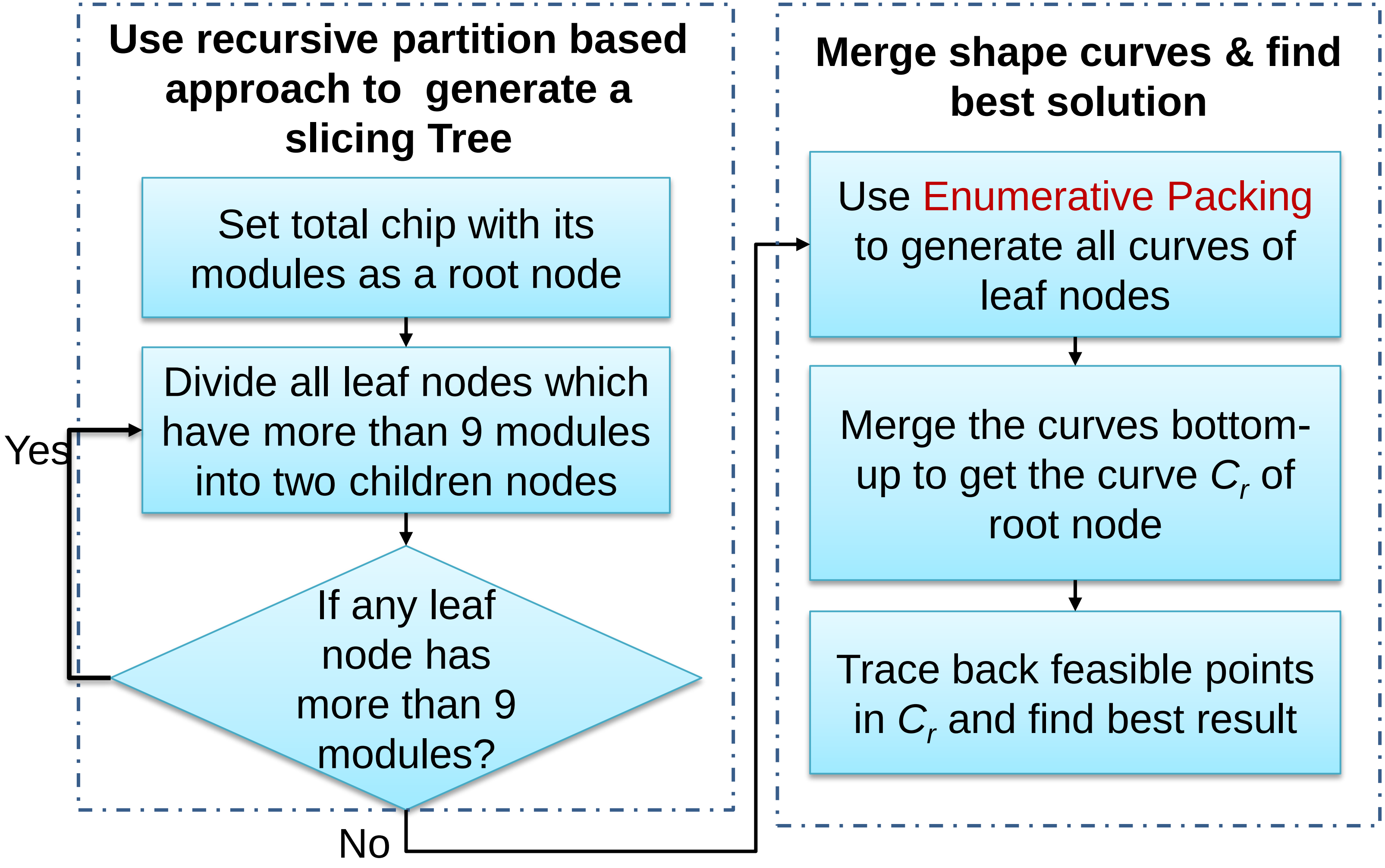
Global distribution result



Legalization result

- Use a two stage approach to achieve it.
  1. Propose a **partition based approach** to generate a slicing tree from the global distribution results.
  2. Enumerative pack and merge shave the curves in this slicing tree to find a best solution which satisfy the constraint.

Two Stage Approach to Legalize



Use recursive partition based approach to generate a slicing Tree

Set total chip with its modules as a root node

Divide all leaf nodes which have more than 9 modules into two children nodes

If any leaf node has more than 9 modules?

Yes

No

Merge shape curves & find best solution

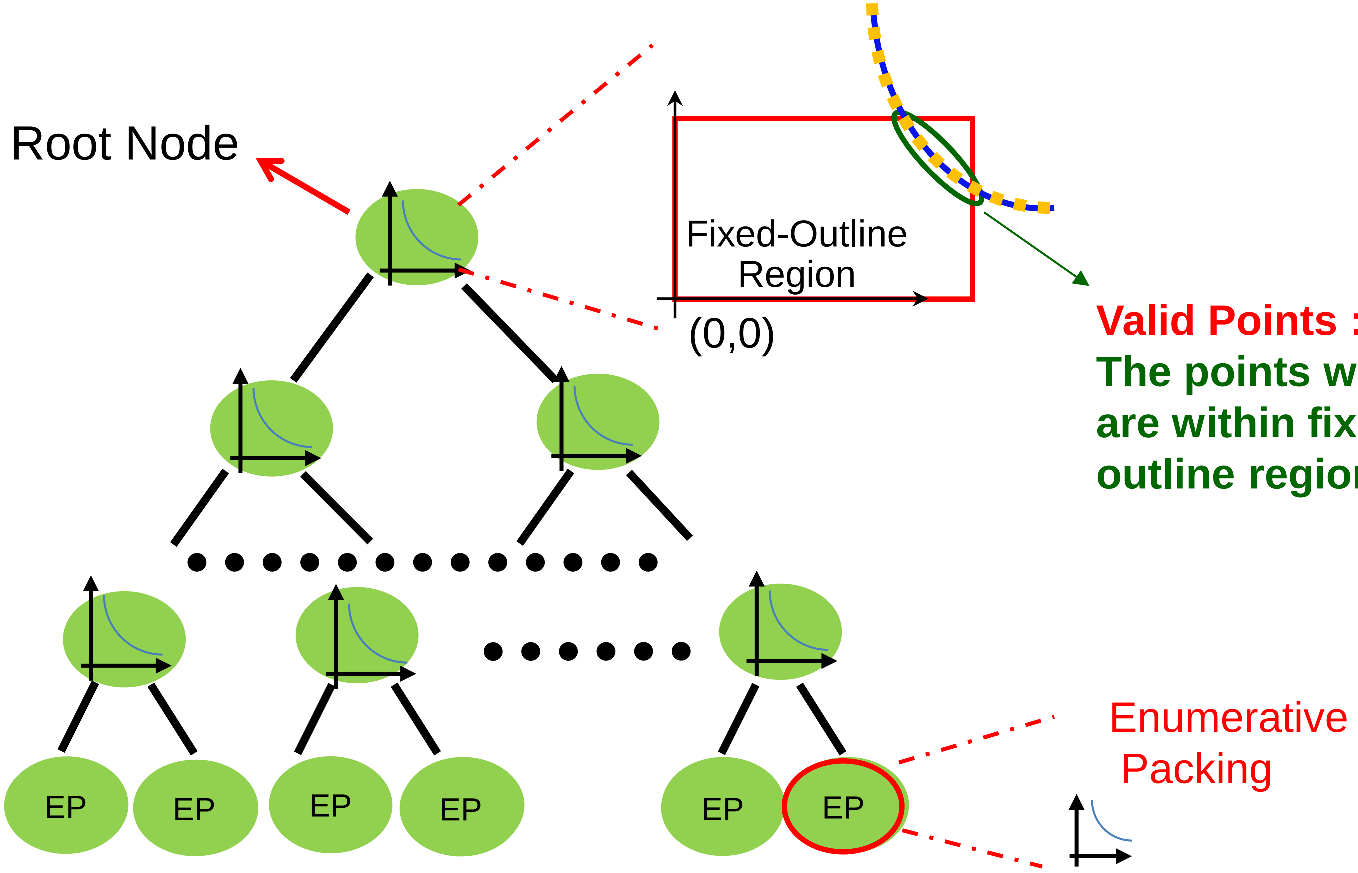
Use Enumerative Packing to generate all curves of leaf nodes

Merge the curves bottom-up to get the curve C_r of root node

Trace back feasible points in C_r and find best result

Two Stage Approach to Legalization

- The flow of our legalization approach

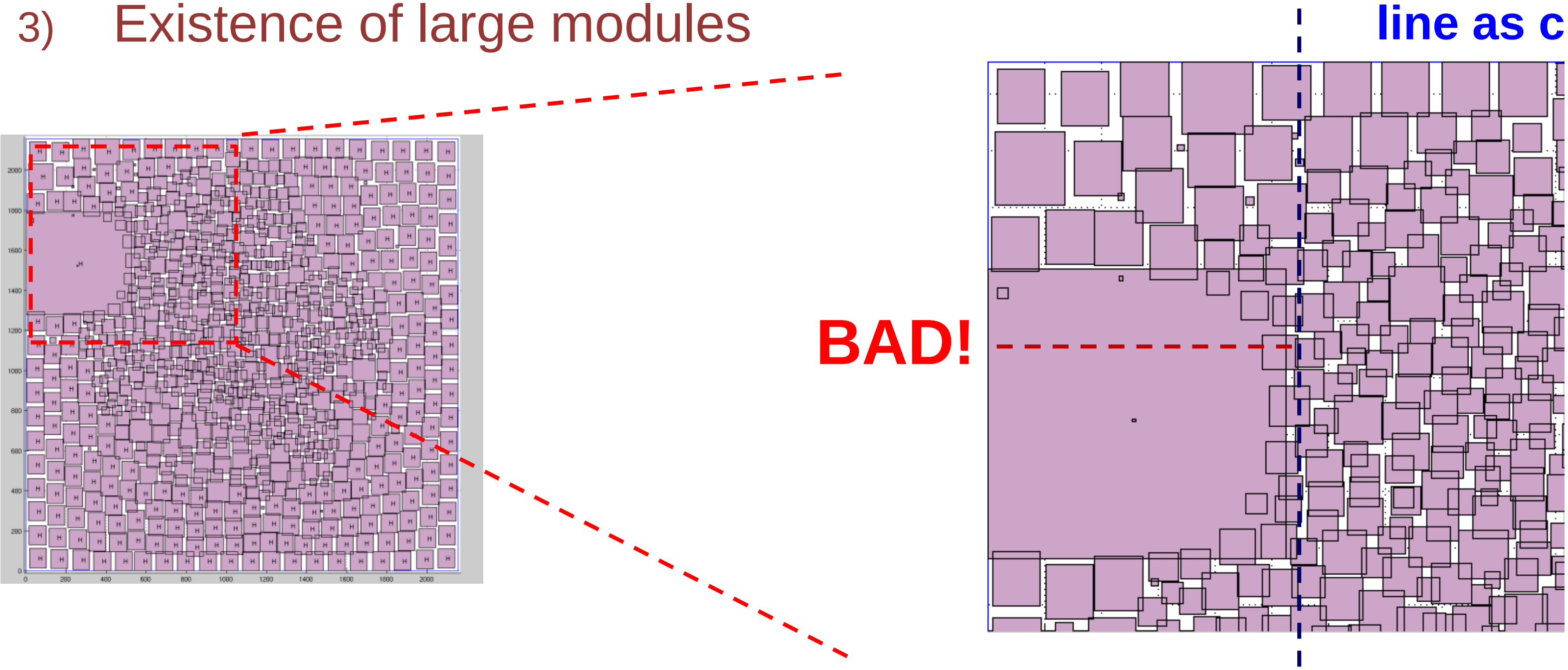


Valid Points : **The points which are within fixed-outline region.**

Enumerative Packing

Difficulty in Applying a Partition Based Approach to Generate a Slicing Tree

- It is not easy to find the location of a cutline because of the following phenomenon:
  1) No straight line without passing across the modules
  2) Un-uniform distribution of modules in a local region
  3) Existence of large modules

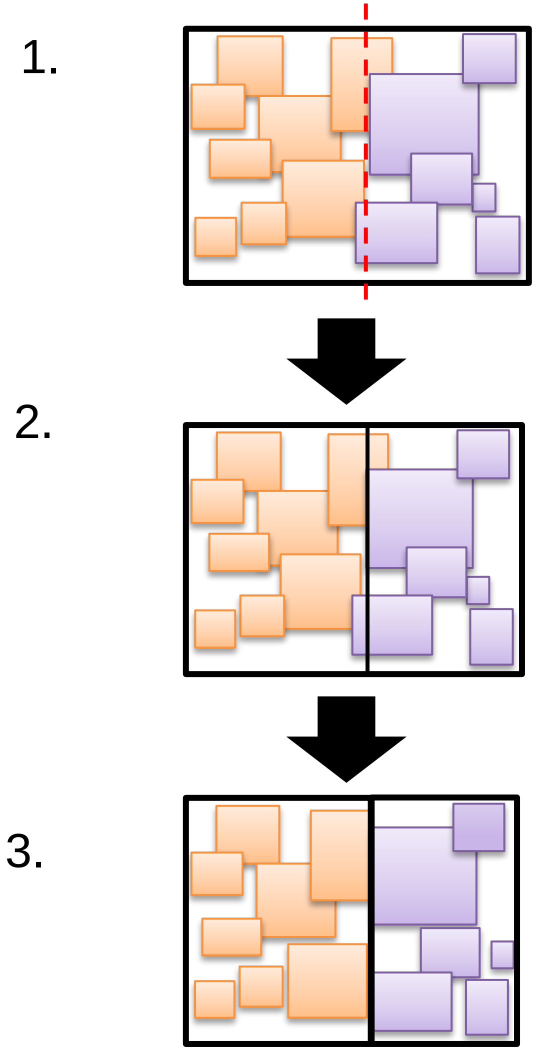


Select the middle line as cutline?

BAD!

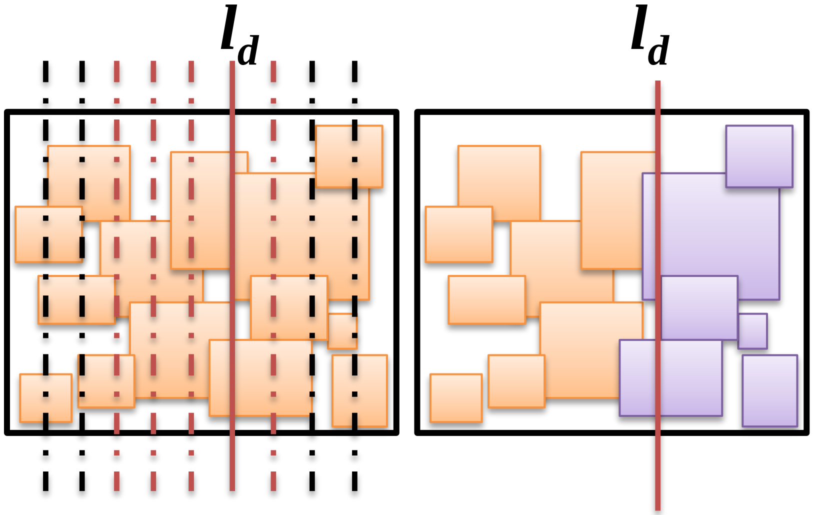
Flow of Partition Based Approach

1. Divide modules into two parts
   - Find a cutline passing across least modules
2. Determine dimensions of regions for placing modules in each part
   - Take additional consideration while existence of a larger module in a region
3. Re-spread modules in each region
   - Move crossed modules into the region where they belong to
   - Spread modules again by using an analytical approach



Division of Modules

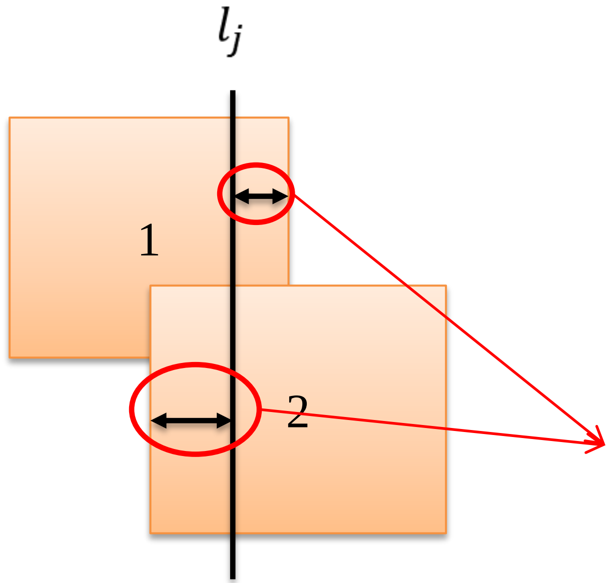
- Find a cutline passing across least modules



- Divide a region into several uniform-tiles by a set of straight lines first
  - Delete those lines which are close to the boundary of region
  - Compute the cost of remaining lines, and select the line whose cost is least.

Division of Modules

- Crossing Cost
  - While a line pass across a module, there will be a jag produced by this line and module's boundary.
  - The cost of a line is the sum of total jags of its.



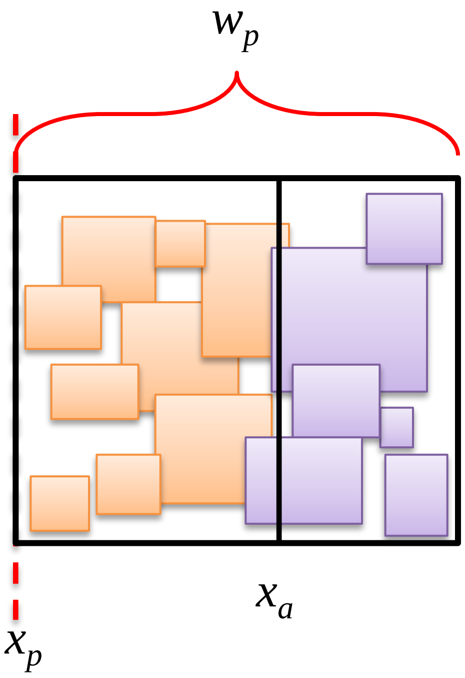
$$\varphi(l_j) = \sum_{m_k \in M_j} Jag(l_j, m_k)$$

$$jag(l_j, m_k) \begin{cases} \left(x_k + \frac{w_k}{2} - x_j\right) & if\ x_k \le x_j \le (x_k + \frac{w_k}{2}) \\ \left(x_j - x_k + \frac{w_k}{2}\right) & else\ if\ (x_k - \frac{w_k}{2}) \le x_j \le x_k \end{cases}$$

Determination of Dimensions of Regions

- The cutline with minimum cost doesn’t ensure that it has enough space for modules to place
- To ensure enough space, we have to balance the whitespace in each region.

$$x_a = x_p + w_p \frac{\sum_{m_i \epsilon P_{left}} \alpha_i A_i}{\sum_{m_i \epsilon P_{left}} \alpha_i A_i + \sum_{m_i \epsilon P_{right}} \alpha_i A_i}$$



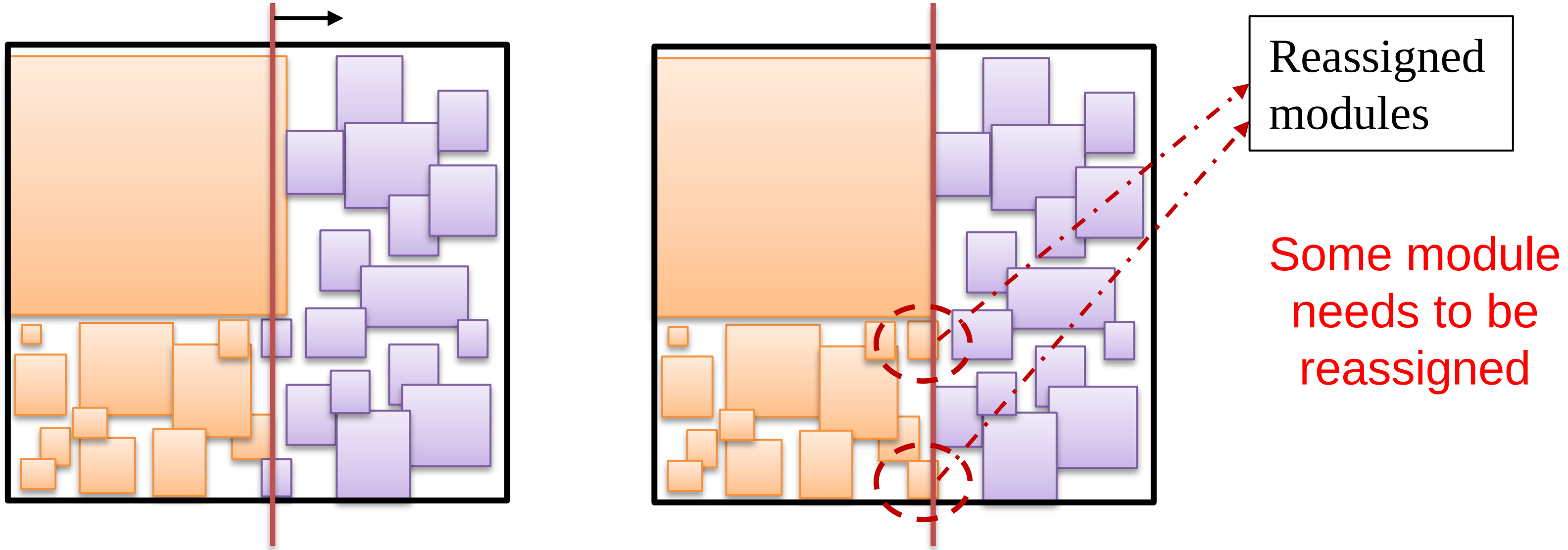
P_{left} : The partition of left child

P_{right} : The partition of right child

α_i : A weighted parameter, a hard module usually has bigger α_i than a soft one.

Determination of Dimensions of Regions in Special Condition

- While the width or height of sub-region is smaller than the width or height of any module in it, it has to shift cutline location to obtain a feasible space.
  - Some modules need to be re-assigned to balance the white space

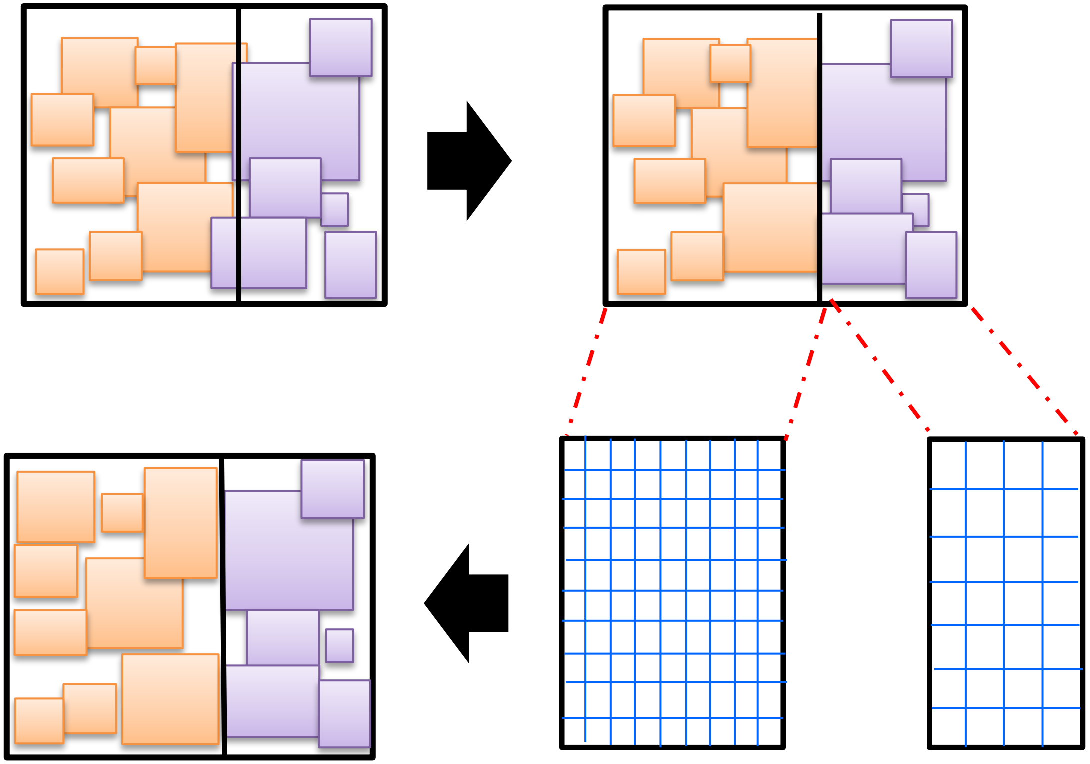


Reassigned modules

Some module needs to be reassigned

Re-spread Modules in Each Region

- Move modules into the specified region
  - Spread modules in each region more uniform
    - Divide each region into many regular bins according to the number of modules in it.
    - Use conjugate gradient to solve the bell-shape function



Experimental Results

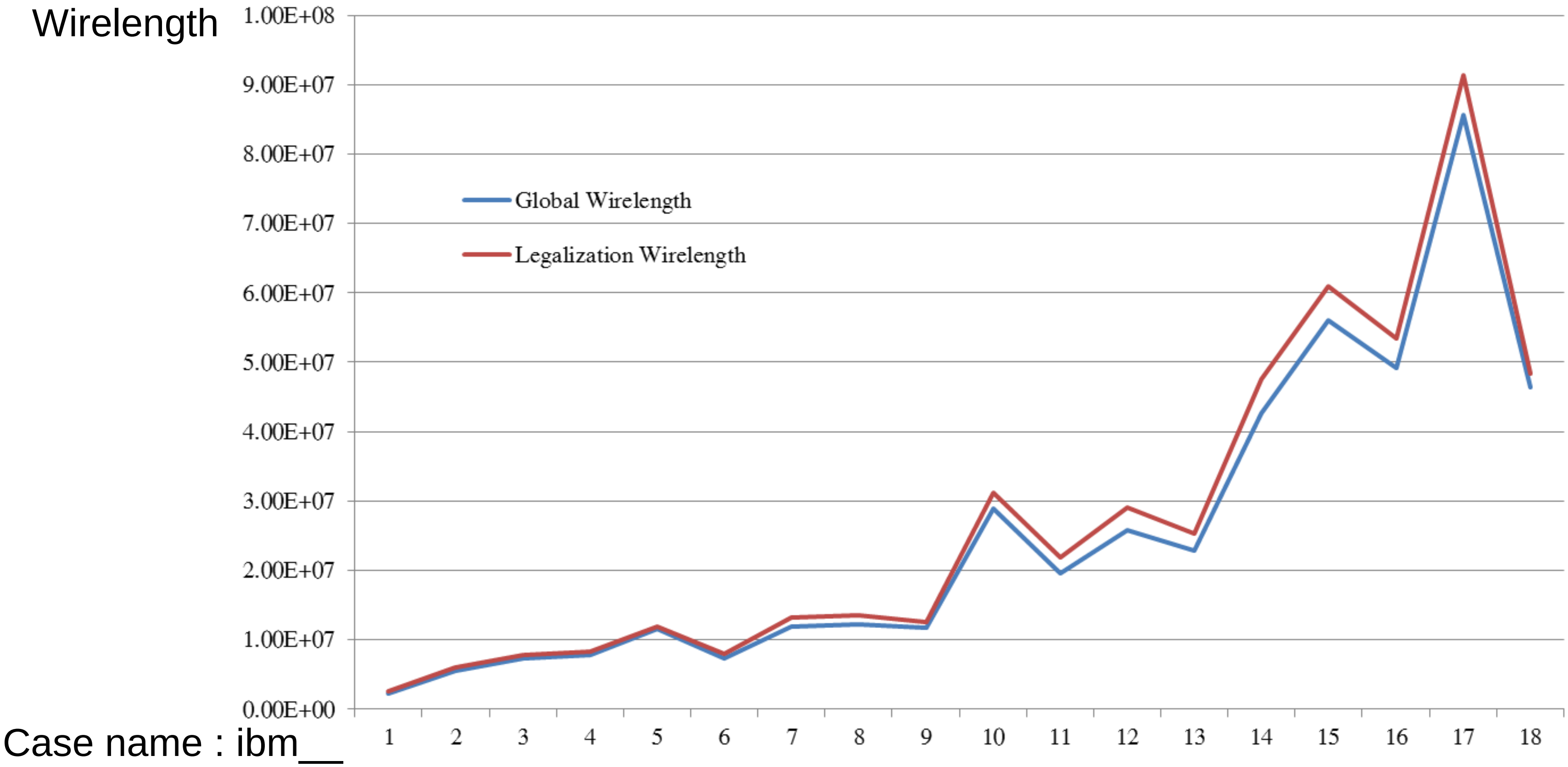
- Environment :
  - Language : C++
  - CPU : Intel Xeon ® 2.4 GHz
  - Memory : 4GB
  - Benchmarks : From IBM/ISPD suite

Experimental Results

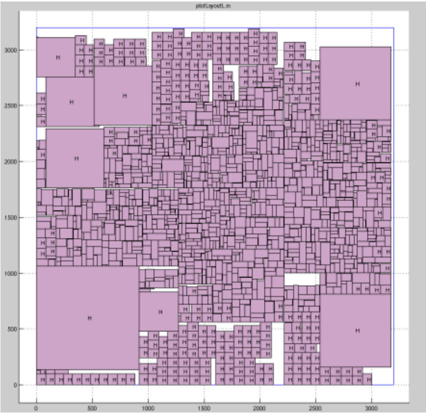
				PATOMA[1]		DeFer[2]		Ours	
Circuit	#Soft	#Hard	#Net	WL(e+06)	Time(s)	WL(e+06)	Time(s)	WL(e+06)	Time(s)
ibm01	665	246	4236	2.84	7.04	2.66	1.44	2.64	10.52
ibm02	1200	271	7652	X	X	6.55	14.48	5.99	73.5
ibm03	999	290	7956	12.59	5.42	8.77	3.6	7.89	15.58
ibm04	1289	295	10055	X	X	8.94	3.04	8.38	25.08
ibm05	564	0	7887	12.27	14.21	12.64	3.55	11.81	11.89
ibm06	571	178	7211	X	X	7.87	3.66	7.92	8.26
ibm07	829	291	11109	X	X	13.81	3.87	13.00	13.09
ibm08	968	301	11536	X	X	13.95	5.44	13.63	12.81
ibm09	860	253	11008	X	X	12.85	2.6	12.49	13.18
ibm10	809	786	16334	48.47	21.71	33.25	11.63	31.2	20.64
ibm11	1124	373	16985	20.87	33.87	21.99	4.84	21.84	19.74
ibm12	582	651	11873	X	X	29.72	10.95	29.08	8.94
ibm13	530	424	14202	X	X	25.95	6.03	25.32	14.13
ibm14	1021	614	26675	71.87	23.59	50.83	9.69	47.58	41.45
ibm15	1019	393	28270	X	X	64.18	9.71	60.88	42.82
ibm16	633	458	21013	X	X	56.88	16.79	53.43	27.4
ibm17	682	760	30556	102.45	41.75	95.92	10.43	91.29	37.44
ibm18	658	285	21191	50.28	38.24	49.12	7.93	48.41	35.49
Normalized				1.225	0.96	1.047	0.3	1	1

Experimental Results

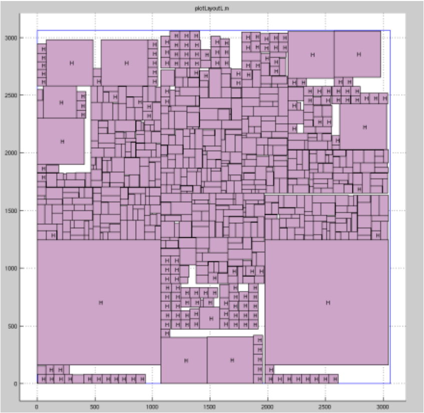
- Demonstrate the consistency of the results in global distribution stage and legalization stage



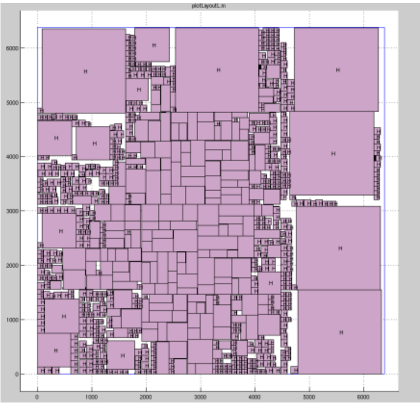
Experimental Results



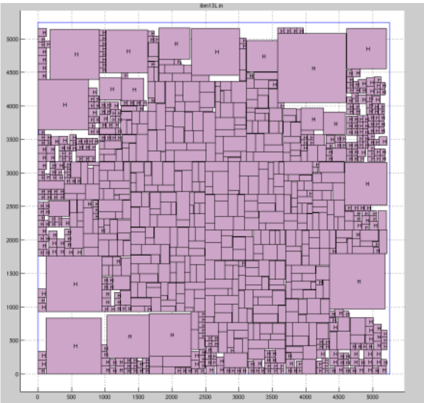
Ibm04
HPWL
8.38e+06
Time
25.08



Ibm06
HPWL
7.92e+06
Time
8.26



Ibm12
HPWL
29.08
Time
8.94



Ibm16
HPWL
56.88e+06
Time
16.79

Conclusions & Future Work

Conclusions

- In this paper, a two stage approach is proposed to handle fixed outline floorplanning for mixed-size modules.
- The approach is flexible and efficient.
  - **Flexible** : can be integrated to other method such as force-directed method, or extended to handle other issues
  - **Efficient** : the longest time was spent is 70 seconds.
- We can get the wirelength by 4.7% and 22.5% better than DeFer and PATOMA respectively.

Future Work

- Consider routability and thermal issues and extend it to handle floorplanning in 3D IC