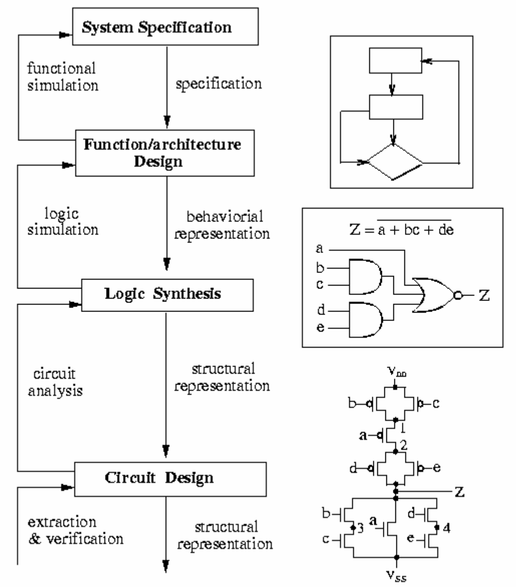
Traditional VLSI Design Flow Revisited (1/2)



System Specification

functional simulation

specification

Function/architecture Design

logic simulation

behaviorial representation

Logic Synthesis

circuit analysis

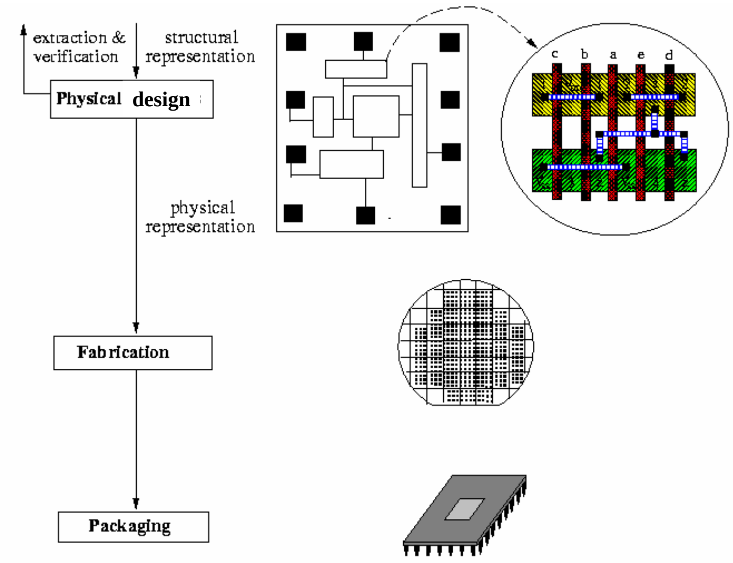
structural representation

Circuit Design

extraction & verification

structural representation

Traditional VLSI Design Flow Revisited (2/2)



extraction & verification

structural representation

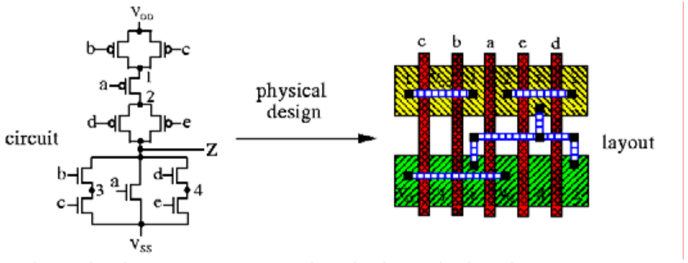
Physical design

physical representation

Fabrication

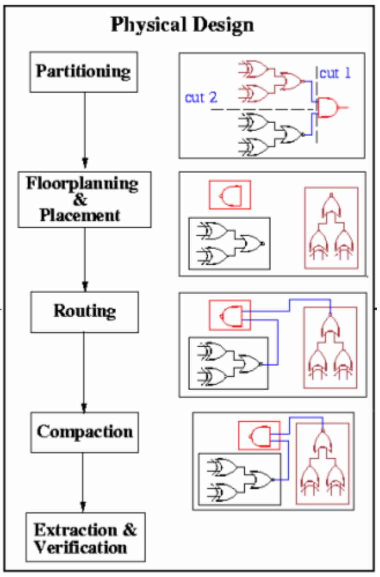
Packaging

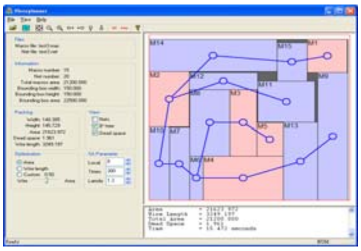
Physical Design



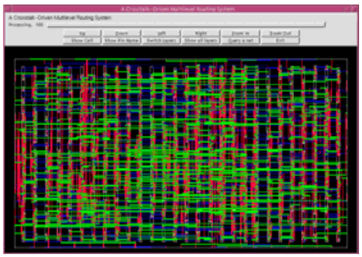
- Physical design converts a circuit description into a geometric description.
- The description is used to manufacture a chip.
- Physical design cycle:
    1. Logic partitioning
    2. Floorplanning and placement
    3. Routing
    4. Compaction
- Others: circuit extraction, timing verification, and design rule checking

Physical Design Flow



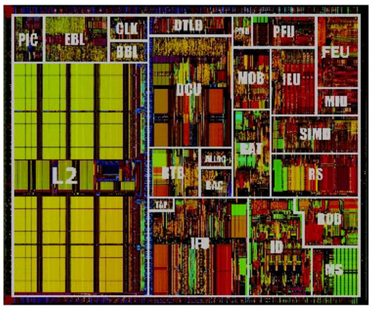


B*-tree based floorplanning system

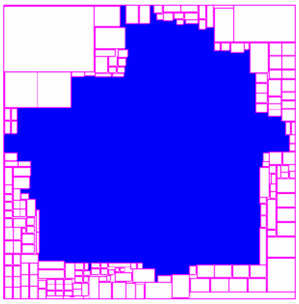


A routing system

Floorplan Examples

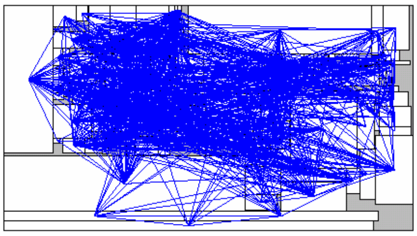


Intel Pentium 4



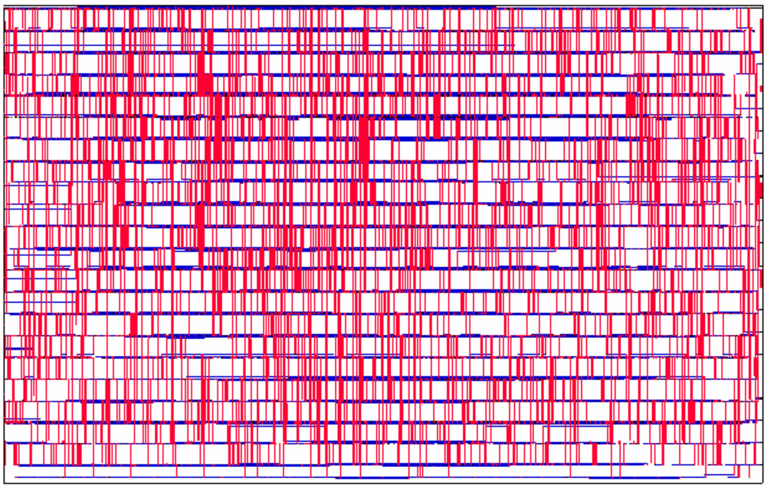
Mixed-size design

A floorplan with interconnections



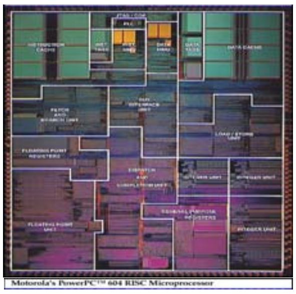
Routing Example

- 0.18um technology, two layers, pitch = 1 um, 8109 nets.

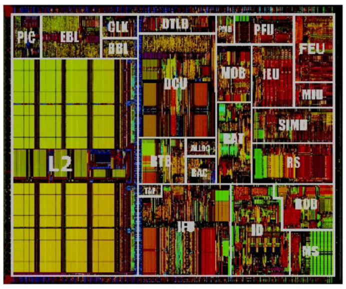


Floorplanning

- Course contents
  - Floorplanning basics
  - Normalized Polish expression for slicing flooprlans
  - B*-trees for non-slicing floorplans
- Readings
  - Chapters 8 and 5.6



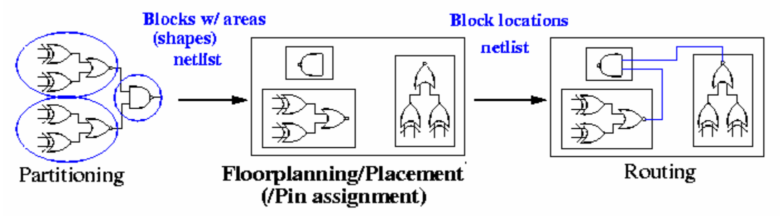
PowerPC 604



Pentium 4

Floorplanning

- Partitioning leads to
  - Blocks with well-defined **areas and shapes** (rigid/hard blocks).
  - Blocks with approximate areas and no particular shapes (flexible/soft blocks).
  - A **netlist** specifying connections between the blocks.
- Objectives
  - Find **locations** for all blocks.
  - Consider shapes of soft block and pin locations of all the blocks.



Blocks w/ areas (shapes) netlist

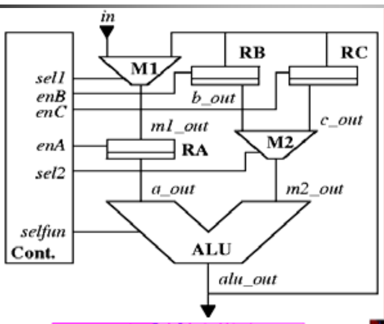
Block locations netlist

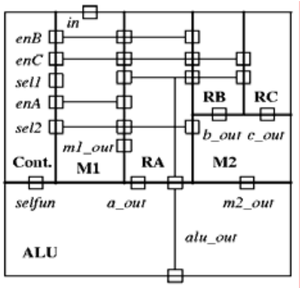
Partitioning

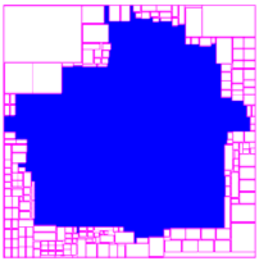
Floorplanning/Placement (/Pin assignment)

Routing

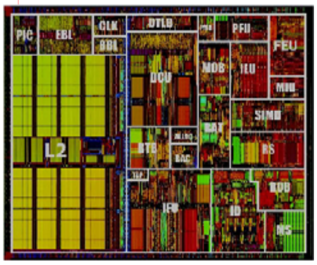
Floorplan Examples







Mchip 4



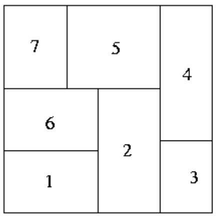
Pentium 4

Early Layout Decision Methodology

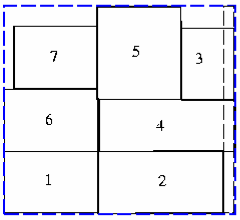
- An IC is a 2-D medium; considering the dimensions of blocks in early stages of the design helps to improve the quality.
- Floorplanning gives early feedback
  - Suggests valuable architectural modifications
  - Estimates the whole chip area
  - Estimates delay and congestion due to wiring
- Floorplanning fits very well in a *top-down* design strategy; the *step-wise refinement* strategy also propagated in software design.
- Floorplanning considers the *flexibility* in the shapes and terminal locations of blocks.

Floorplanning Problem

- Inputs to the floorplanning problem:
  - A set of blocks, hard or soft.
  - Pin locations of hard blocks.
  - A netlist.
- Objectives: minimize area, reduce wirelength for (critical) nets, maximize routability (minimize congestion), determine shapes of soft blocks, etc.

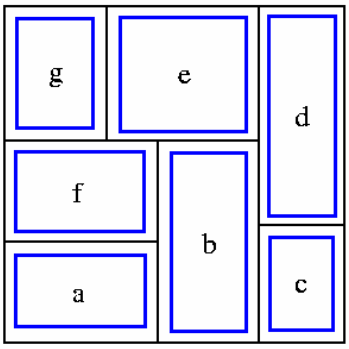


An optimal floorplan, in terms of area

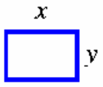


A non–optimal floorplan

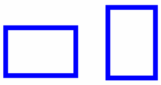
Floorplan Design



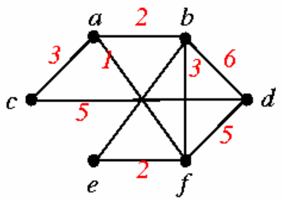
- *Modules:*



- *Area: A=xy*
- *Aspect ratio: r <= y/x <= s*
- *Rotation:*

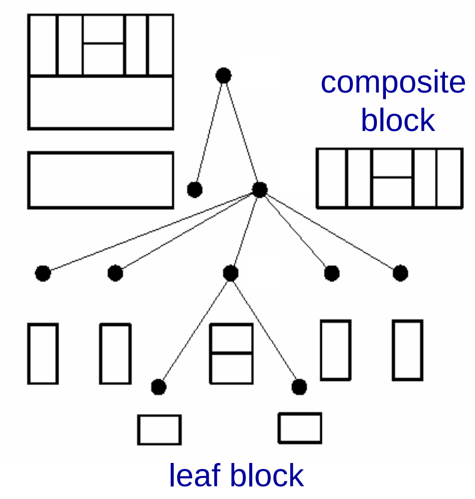


- *Module connectivity*



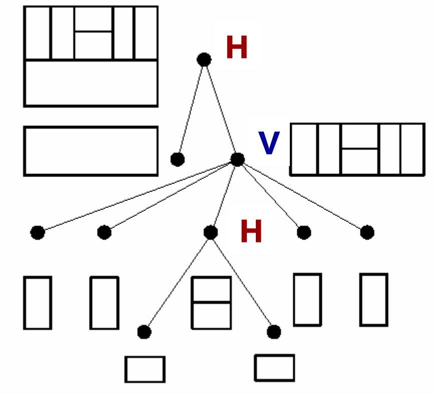
Floorplan Elements

- **Leaf block (cell/module):** a block at the lowest level of the hierarchy; it does not contain any other block.
- **Composite block (cell/module):** a block that is composed of either leaf blocks or composite blocks. The entire IC is the highest-level composite block.



Slicing Floorplan + Slicing Tree

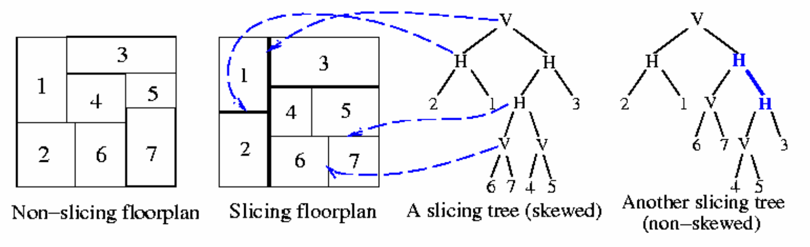
- A composite block's subblocks are obtained by a horizontal or vertical *bisection* of the composite block.
- Slicing floorplans can be represented by a **slicing tree**.
- In a slicing tree, all blocks (except for the top-level block) have a *parent*, and all composite blocks have *children*.
- A slicing floorplan is also called a floorplan of order 2.



H: horizontal cut
V: vertical cut
different from the definitions in the text!!

Skewed Slicing Tree

- **Rectangular dissection:** Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure:** a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- **Slicing tree:** A binary tree, where each internal node represents a vertical cut line or horizontal cut line, and each leaf a basic rectangle.
- **Skewed slicing tree:** One in which no node and its **right** child are the same.



Non–slicing floorplan

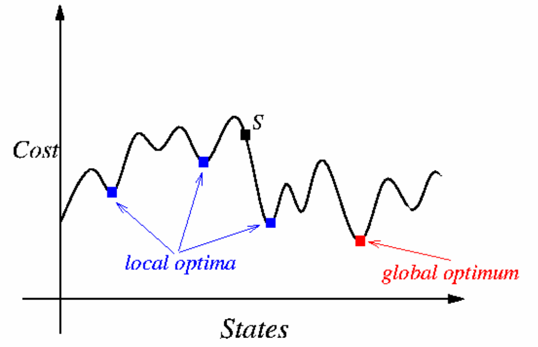
Slicing floorplan

A slicing tree (skewed)

Another slicing tree (non–skewed)

Floorplan Design by Simulated Annealing

- Slicing Floorplan: Wong & Liu, “A new algorithm for floorplan design,” DAC-86.
- Compacted Floorplan: Chang, Chang, Wu, and Wu, “B*-tree: A new representation for non-slicing floorplans,” DAC-2K.
- Kirkpatrick, Gelatt, and Vecchi, “Optimization by simulated annealing,” *Science*, May 1983.



Simulated Annealing Basics

- Non-zero probability for “up-hill” moves.
- Probability depends on
  1. magnitude of the “up-hill” movement
  2. total search time

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{“}down-hill\text{''} \ moves */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{“}up-hill\text{''} \ moves */ \end{cases}$$

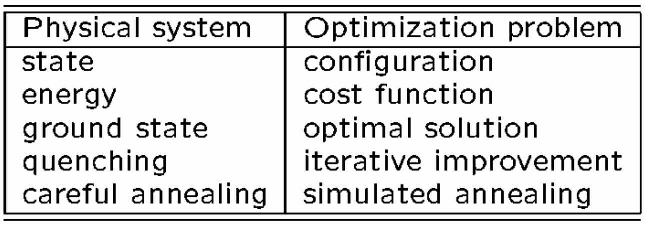
- ΔC = *cost*(*S*') - *Cost*(*S*)
- *T*: Control parameter (temperature)
- Annealing schedule: $T=T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0$, $r < 1$.

Generic Simulated Annealing Algorithm

```
begin
Get an initial solution $S$;
Get an initial temperature $T > 0$;
while not yet “frozen” do
    for $1 \leq i \leq P$ do
        Pick a random neighbor $S'$ of $S$;
        $\Delta \leftarrow cost(S') - cost(S)$;
        /* downhill move */
        if $\Delta \leq 0$ then $S \leftarrow S'$
        /* uphill move */
        if $\Delta > 0$ then $S \leftarrow S'$ with probability $e^{-\frac{\Delta}{T}}$;
$T \leftarrow rT$;  /* reduce temperature */
return $S$
end
```

Basic Ingredients for Simulated Annealing

- Analogy:

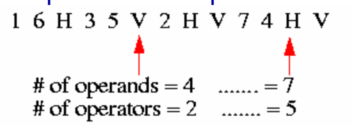


Physical system	Optimization problem
state	configuration
energy	cost function
ground state	optimal solution
quenching	iterative improvement
careful annealing	simulated annealing

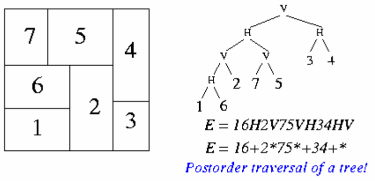
- Basic Ingredients for Simulated Annealing:
  - **Solution space**
  - **Neighborhood structure**
  - **Cost function**
  - **Annealing schedule**

Solution Representation: Slicing Flooprlan

- An expression $E = e_1\ e_2 \ldots\ e_{2n-1}$, where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \leq i \leq 2n\text{-}1$, is a **Polish expression** of length $2n$-1 iff
  1. every operand j, $1 \leq j \leq n$, appears exactly once in E;
  2. (**the balloting property**) for every subexpression $E_i = e_1 \ldots e_i$, $1 \leq i \leq 2n\text{-}1$, # operands > # operators.



- Polish expression ↔ Postorder traversal.
- *ijH*: rectangle *i* on bottom of *j*; *ijV*: rectangle *i* on the left of *j*.

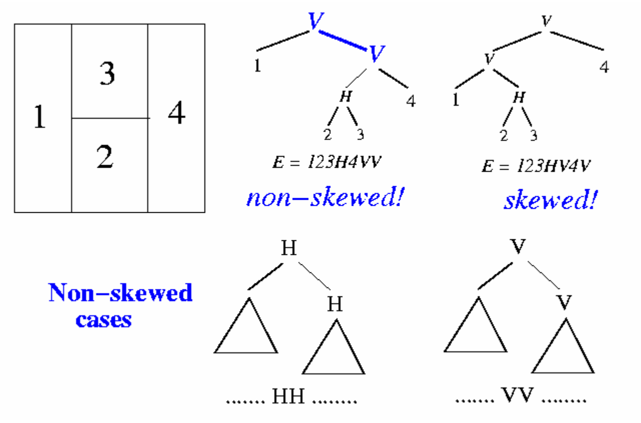


$E = 16H2V75VH34HV$

$E = 16{+}2{*}75{*}{+}34{+}{*}$

Postorder traversal of a tree!

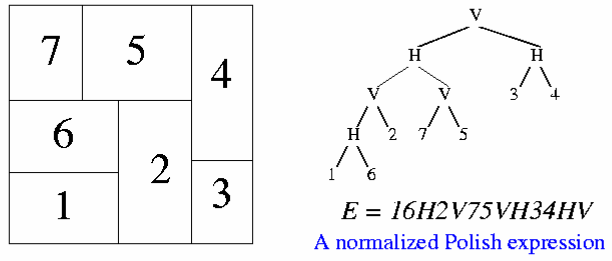
Redundant Representations



- **Question:** How to eliminate ambiguous representation?

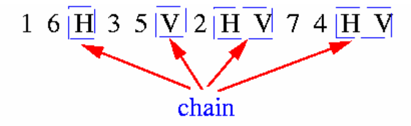
Normalized Polish Expression

- A Polish expression $E = e_1\ e_2 \ldots e_{2n\text{-}1}$ is called **normalized** iff E has no consecutive operators of the same type (H or V).
- Given a **normalized** Polish expression, we can construct a **unique** rectangular slicing structure.



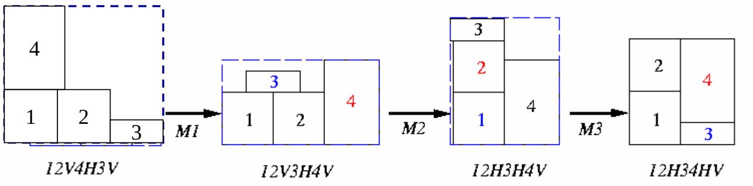
Neighborhood Structure

- **Chain:** *HVHVH* … or *VHVHV* …



- **Adjacent:** 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and *V* are adjacent operand and operator.
- 3 types of moves:
  - *M1* (**Operand Swap**): Swap two adjacent operands.
  - *M2* (**Chain Invert**): Complement some chain (*V* = *H*, *H* = V).
  - *M3* (**Operator/Operand Swap**): Swap two adjacent operand and operator.

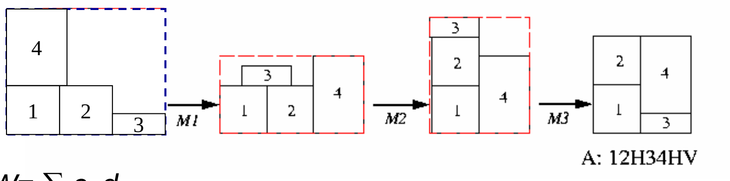
Effects of Perturbation



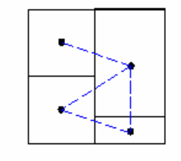
- **Question:** The balloting property holds during the moves?
  - *M1* and *M2* moves are OK.
  - **Check the *M3* moves!** Reject “illegal” *M3* moves.
- **Check *M3* moves:** Assume that *M3* swaps the operand e_i with the operator e_{i+1}, $1 \le i \le k-1$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
  - N_k: # of operators in the Polish expression $E = e_1\ e_2 \dots e_k$, $1 \le k \le 2n-1$

Cost Function

- $\phi = A + \lambda W$.
  - *A*: area of the smallest rectangle
  - *W*: overall wiring length
  - λ : user-specified parameter

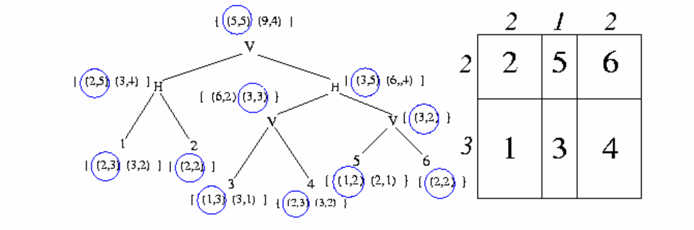


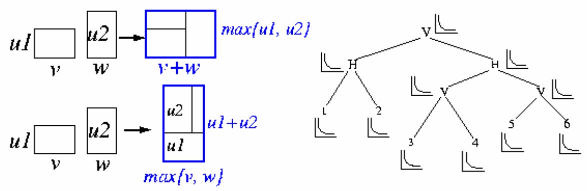
- $W = \sum_{ij} c_{ij}\, d_{ij}$.
  - c_{ij}: # of connections between blocks *i* and *j*.
  - d_{ij}: center-to-center distance between basic rectangles *i* and *j*.



Area Computation for Hard Blocks

- Allow rotation

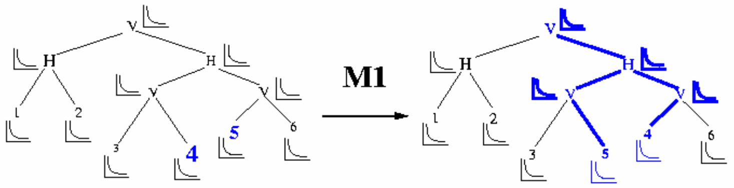




- Wiring cost?
  - Center-to-center interconnection length

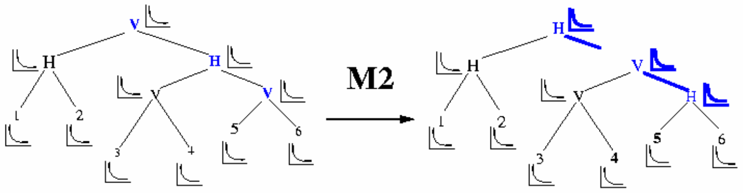
Incremental Computation of Cost Function

- Each move leads to only a minor modification of the Polish expression.
- At most **two paths** of the slicing tree need to be updated for each move.

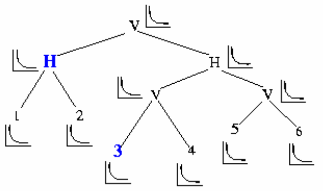


E = 12H3$\boxed{4V5}$6VHV   E = 12H3$\boxed{5V4}$6VHV

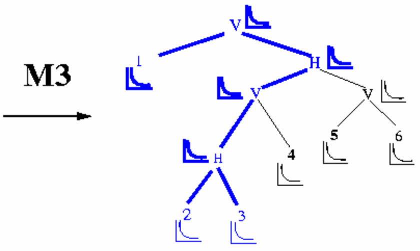
Incremental Computation of Cost Function (cont'd)



E = 12H34V56$\boxed{VHV}$ E = 12H34V56$\boxed{HVH}$



E = 12H34V56VHV



E = 123H4V56VHV

Annealing Schedule

- Initial solution: 12*V*3*V* … *nV*.

1	2	3		n

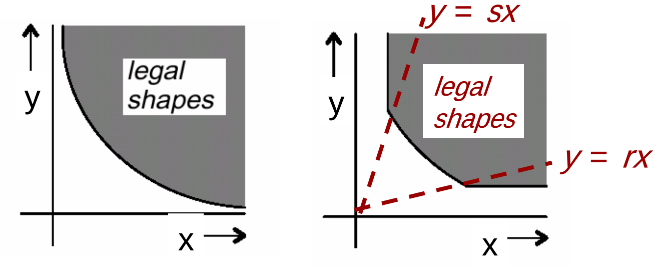
- $T_i = r^i\, T_0$, i = 1, 2, 3, …; r =0.85.
- At each temperature, try *kn* moves (*k* = 5-10).
- Terminate the annealing process if
  - # of accepted moves < 5%,
  - temperature is low enough, or
  - run out of time.

Algorithm: Wong-Liu (*P*, ε, *r*, *k*)

```
1 begin
2 $E \leftarrow$ 12V3V4V … nV; /* initial solution */
3 $Best \leftarrow E$; $T_0 \leftarrow \frac{\Delta_{avg}}{ln(P)}$ ; $M \leftarrow MT \leftarrow uphill \leftarrow 0$; $N = kn$;
4 repeat
5   $MT \leftarrow$ uphill $\leftarrow$ reject $\leftarrow$ 0;
6   repeat
7     SelectMove($M$);
8     Case $M$ of
9     $M_1$:  Select two adjacent operands $e_i$ and $e_j$; $NE \leftarrow$ Swap($E$, $e_i$, $e_j$);
10    $M_2$:  Select a nonzero length chain $C$; $NE \leftarrow$ Complement($E$, $C$);
11    $M_3$:  done $\leftarrow$ FALSE;
12        while not (done) do
13            Select two adjacent operand $e_i$ and operator $e_{i+1}$;
14            if ($e_{i-1} \neq e_{i+1}$)  and ($2\, N_{i+1} < i$) then done $\leftarrow$ TRUE;
13'           Select two adjacent operator $e_i$ and operand $e_{i+1}$;
14'           if ($e_i \neq e_{i+2}$) then done $\leftarrow$ TRUE;
15        $NE \leftarrow$ Swap($E$, $e_i$, $e_{i+1}$);
16    $MT \leftarrow MT+1$; $\Delta cost \leftarrow cost(NE) - cost(E)$;
17    if ($\Delta cost \leq 0$) or ($Random < e^{\frac{-\Delta cost}{T}}$ )
18     then
19        if ($\Delta cost > 0$) then $uphill \leftarrow uphill + 1$;
20        $E \leftarrow NE$;
21        if  $cost(E) < cost(best)$ then $best \leftarrow E$;
22      else $reject \leftarrow reject + 1$;
23   until ($uphill > N$) or ($MT > 2N$);
24   $T \leftarrow rT$; /* reduce temperature */
25 until ($reject/MT > 0.95$) or ($T < \varepsilon$) or $OutOfTime$;
26 end
```

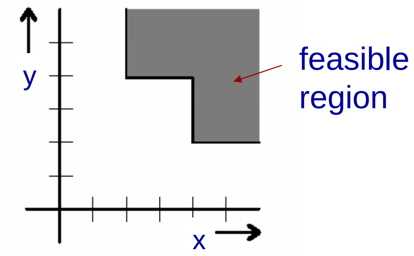
Shape Curve for Floorplan Sizing

- A soft (flexible) block *b* can have different aspect ratios, but is with a fixed area *A*.
- The shape function of *b* is a hyperbola: *xy* = *A*, or *y* =*A*/x, for width *x* and height *y*.
- Very thin blocks are often not interesting and feasible to design; add two straight lines for the constraints on aspect ratios.
  - Aspect ratio: $r \le y/x \le s$.



Shape Curve

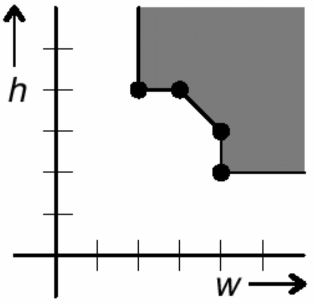
- Since a basic block is built from discrete transistors, it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a block is rigid/hard: it can only be rotated and mirrored during floorplanning or placement.



The shape curve of a 2 × 4 hard block.

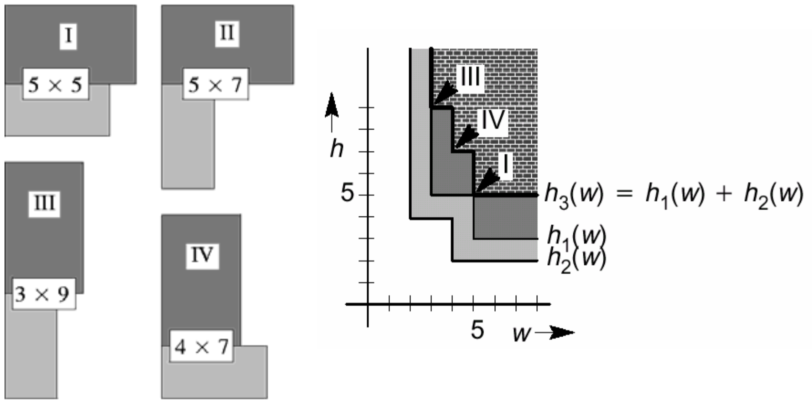
Shape Curve (cont'd)

- In general, a *piecewise linear* function can be used to approximate any shape function.
- The points where the function changes its direction, are called the corner *(break) points* of the piecewise linear function.



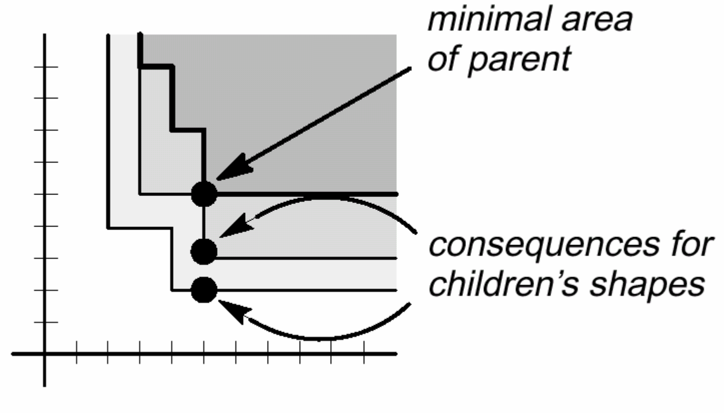
Vertical Abutment

- Composition by vertical abutment (horizontal cut) ⇒ the addition of shape functions.



Deriving Shapes of Children

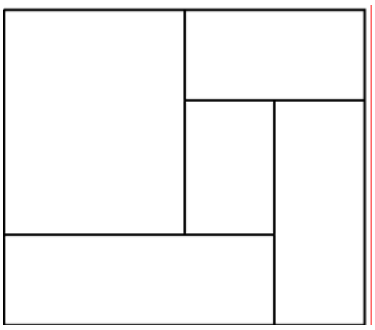
- A choice for the minimal shape of a composite block fixes the shapes of the shapes of its children blocks.



Slicing Floorplan Sizing

- The shape functions of all leaf blocks are given as piecewise linear functions.
- Traverse the slicing tree to compute the shape functions of all composite blocks (bottom-up composition).
- Choose the desired shape of the top-level block; as the shape function is piecewise linear only the corner points of the function need to be evaluated, when looking for the minimal area.
- Propagate the consequences of the choice down to the leaf blocks (top-down propagation).
- The sizing algorithm runs in polynomial time for slicing floorplans
  - NP-complete for non-slicing floorplans

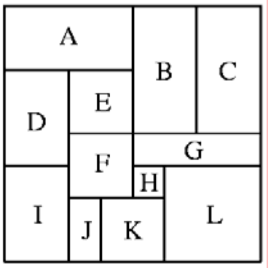
Wheel or Spiral Floorplan

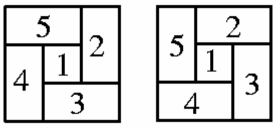


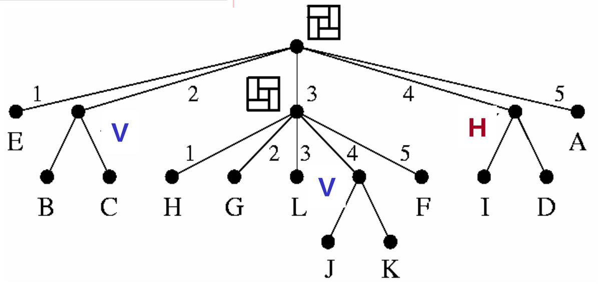
- This floorplan is not slicing!
- **Wheel** is the smallest non-slicing floorplans.

- Limiting floorplans to those that have the slicing property can facilitate floorplanning algorithms.
- Taking the shape of a wheel floorplan and its mirror image as the basis of operators leads to hierarchical descriptions of *order 5.*

Order-5 Floorplan Examples

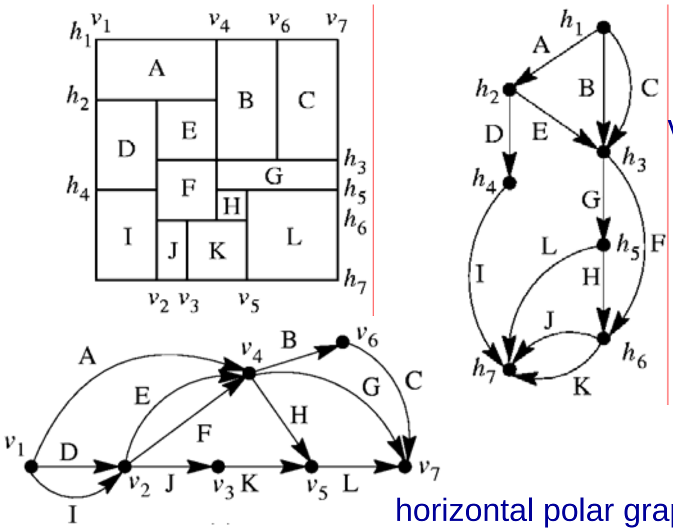






General Floorplan Representation: Polar Graphs

- **vertex：channel segment; edge: cell/block/module.**

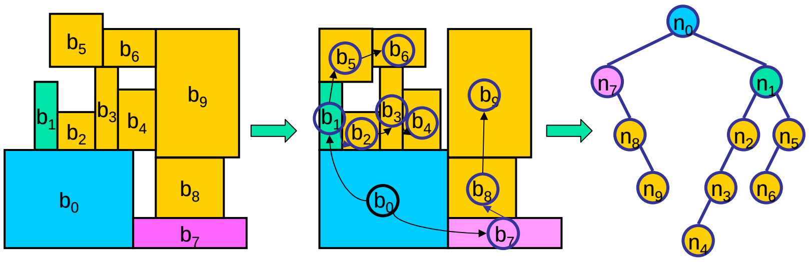

h_1 v_1 v_4 v_6 v_7
A B C D E F G H I J K L
h_2 h_3 h_4 h_5 h_6 h_7
v_2 v_3 v_5

vertical polar graph

horizontal polar graph

B*-Tree: Compacted Floorplan Representation

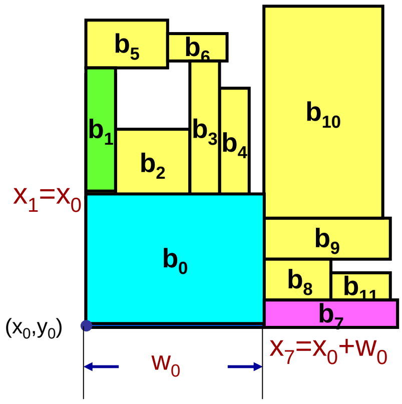
- Chang et al., “B*-tree: A new representation for non-slicing floorplans,” DAC-2K.
  - Compact modules to left and bottom.
  - Construct an ordered binary tree (B*-tree).
    - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$).
    - Right child: the first block above, with the same x-coordinate ($x_j = x_i$).

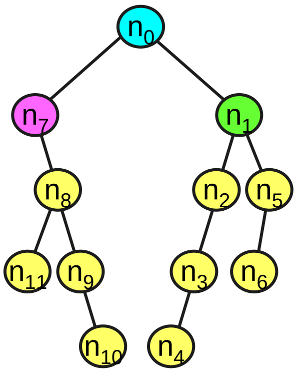

b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
n_0 n_1 n_2 n_3 n_4 n_5 n_6 n_7 n_8 n_9

A non-slicing floorplan **Compact to left and down** **B*-tree**

B*-tree Packing

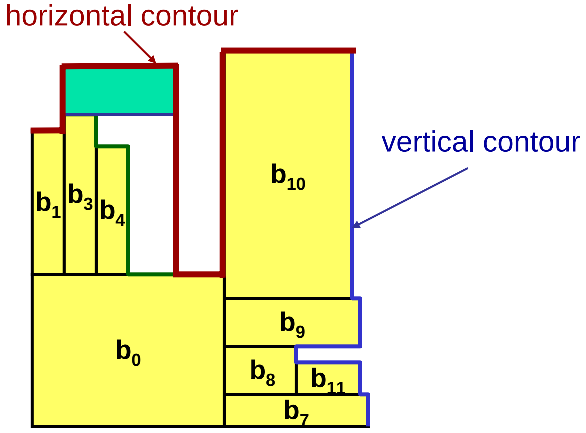
- x-coordinates can be determined by the tree structure.
  - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$).
  - Right child: the first block above, with the same x-coordinate ($x_j = x_i$).
- y-coordinates?





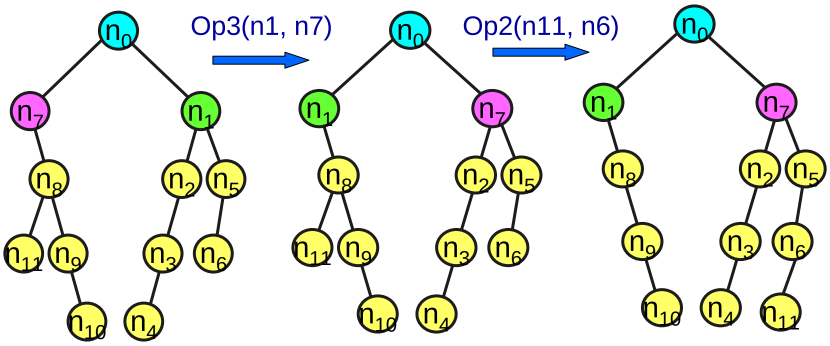
Computing y-coordinates

- Horizontal contour: Use a doubly linked list to record the current maximum y-coordinate for each x-range
- Reduce the complexity of computing a y-coordinate to amortized O(1) time



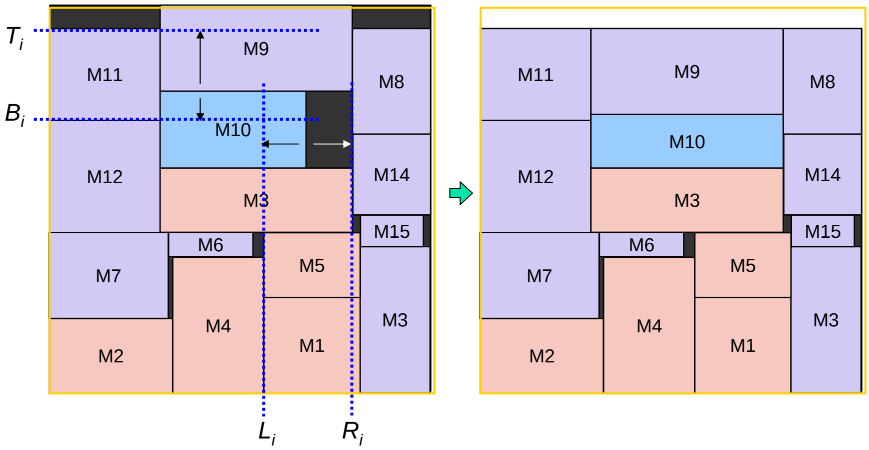
B*-Tree Perturbation

- Op1: rotate a macro
- Op2: delete & insert
- Op3: swap 2 nodes
- Op4: resize a soft macro



Simple Floorplan Sizing

- Key: Line up with adjacent modules
- Advantage: fast and reasonably effective



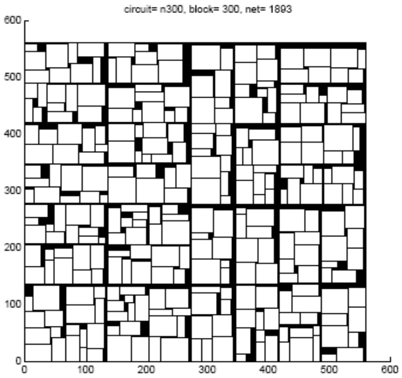
B*-tree Floorplanning

- Was considered as the best representation for packing in a recent survey (Chan et al., ISPD-05)
- More than 100 citations in ACM/IEEE papers since its publication at DAC-2K (> 70% of floorplanning papers)
- Package is available at http://eda.ee.ntu.edu.tw/research.htm/

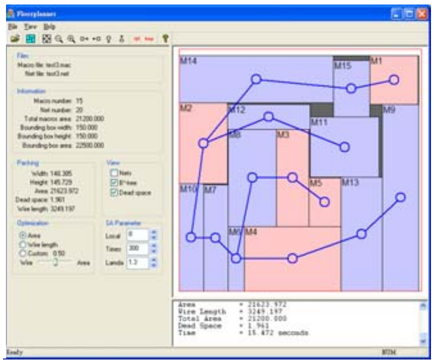
MCNC Ckts	SP (Japan) 1995	Q-Seq (Japan) 2002	O-tree (USA/ Japan) 1999	CBL (China) 2001	Slicing (USA) 2005	TCG (Ours) DAC 2001	TCG-S (Ours) DAC 2002	CS (Ours) TVLSI 2003	B*-tree (Ours) DAC 2000
apte	48.12	**46.92**	47.1	NA	**46.92**	**46.92**	**46.92**	**46.92**	**46.92**
xerox	20.69	19.93	20.1	20.96	20.20	19.83	**19.796**	19.83	**19.796**
hp	9.93	9.03	9.21	NA	9.03	**8.947**	**8.947**	**8.947**	**8.947**
ami33	1.22	1.194	1.25	1.20	1.183	1.20	1.185	1.18	**1.168**
ami49	38.84	36.75	37.6	38.58	**36.24**	36.77	36.4	**36.24**	36.4

Best chip areas are in red

B*-tree Based Floorplanner



GSRC: n300 (300 modules)



Files
Macro file: test3.mac
Net file: test3.net

Information
Macro number: 15
Net number: 20
Total macros area: 21200.000
Bounding box width: 150.000
Bounding box height: 150.000
Bounding box area: 22500.000

Packing
Width: 148.385
Height: 145.729
Area: 21623.972
Dead space: 1.961
Wire length: 3249.197

View
☐ Nets
☑ B*-tree
☑ Dead space

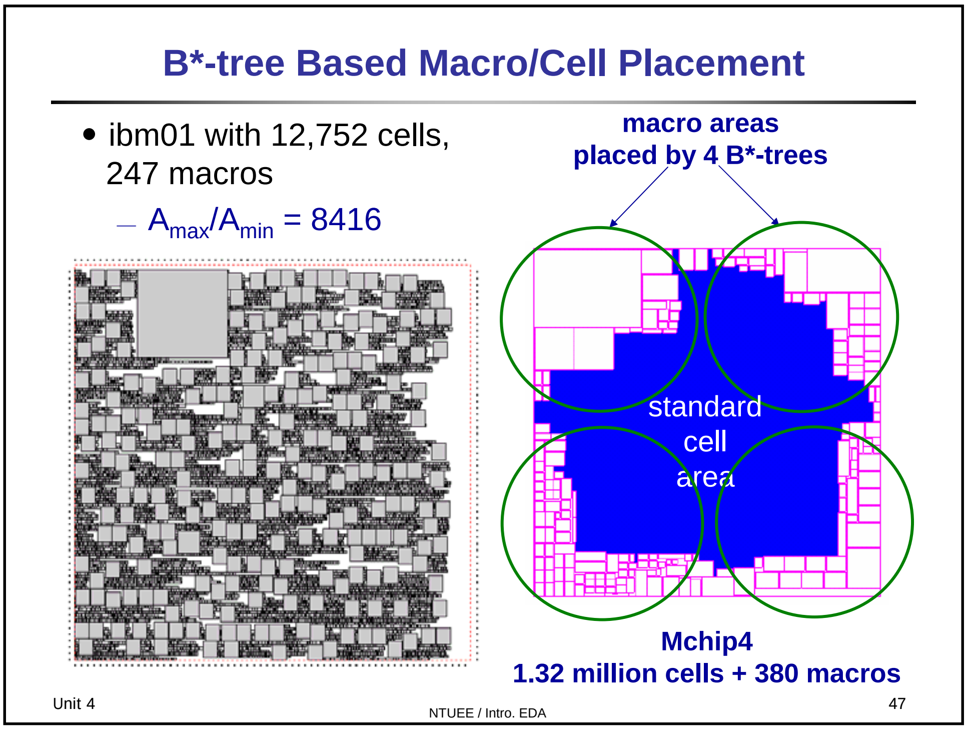
Optimization
◉ Area
○ Wire length
○ Custom: 0.50
Wire — Area

SA Parameter
Local: 8
Times: 300
Lamda: 1.3

```
Area          = 21623.972
Wire Length   = 3249.197
Total Area    = 21200.000
Dead Space    = 1.961
Time          = 15.472 seconds
```

B*-tree floorplanning system

Courtesy of Tung-Chieh Chen



B*-tree Based Macro/Cell Placement

- ibm01 with 12,752 cells, 247 macros
  - A_{max}/A_{min} = 8416

macro areas
placed by 4 B*-trees

Mchip4
1.32 million cells + 380 macros