



香港中文大學

The Chinese University of Hong Kong

**CENG5030**

## Part 1-2: Voltage Scaling

— A Dynamic Programming Approach

**Bei Yu**

(Latest update: January 14, 2019)

Spring 2019

# Overview

Introduction

Background: NP problem

Background: Dynamic Programming

DAC'07: Voltage Partitioning

ICCAD'06: Voltage Assignment on Netlist

ICCAD'07: Voltage Assignment on Slicing Floorplanning



# Overview

Introduction

Background: NP problem

Background: Dynamic Programming

DAC'07: Voltage Partitioning

ICCAD'06: Voltage Assignment on Netlist

ICCAD'07: Voltage Assignment on Slicing Floorplanning



# Multi-Voltage Design @IBM<sup>1</sup>

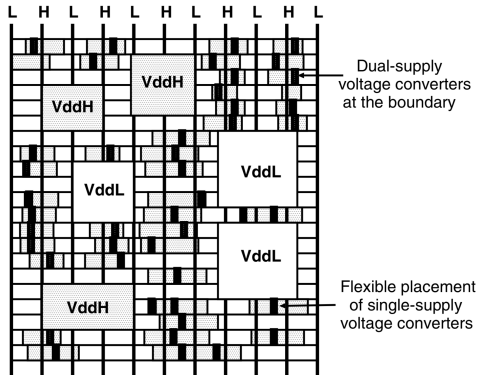


Figure 3: Flexible voltage island layout style.

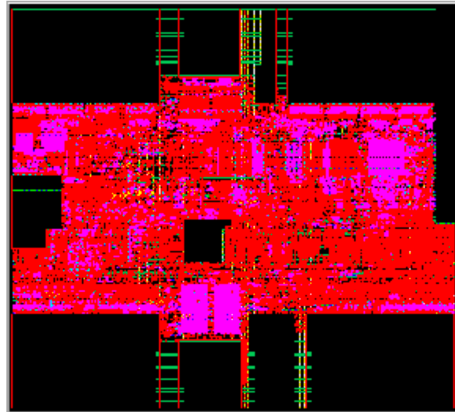
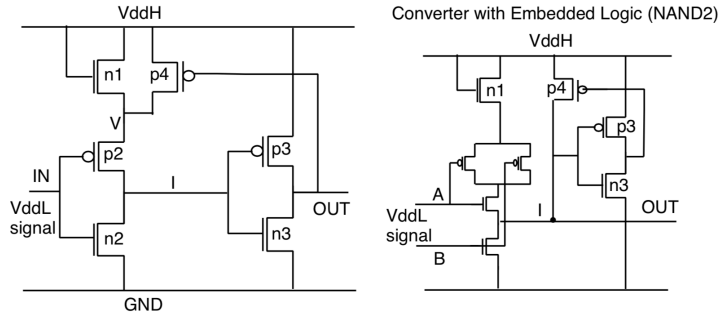


Figure 4: A processor with flexible voltage islands.

<sup>1</sup>Ruchir Puri et al. (2003). "Pushing ASIC performance in a power envelope". In: *Proc. DAC*, pp. 788–793.

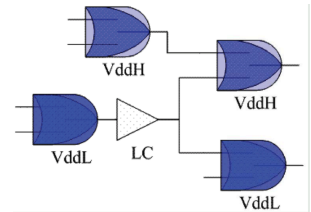


## 4/27

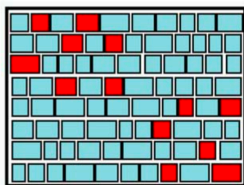


Level-converter is used to avoid excessive static power consumption between the low and high voltage regions.

<sup>2</sup>Ruchir Puri et al. (2003). "Pushing ASIC performance in a power envelope". In: *Proc. DAC*, pp. 788–793.



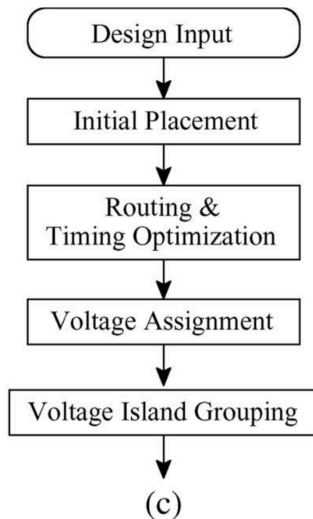
# Placement Level Multi-Voltage<sup>3</sup>



(a)



(b)



<sup>3</sup>Huaizhi Wu and Martin DF Wong (2009). "Incremental improvement of voltage assignment". In: *IEEE TCAD* 28.2, pp. 217–230.



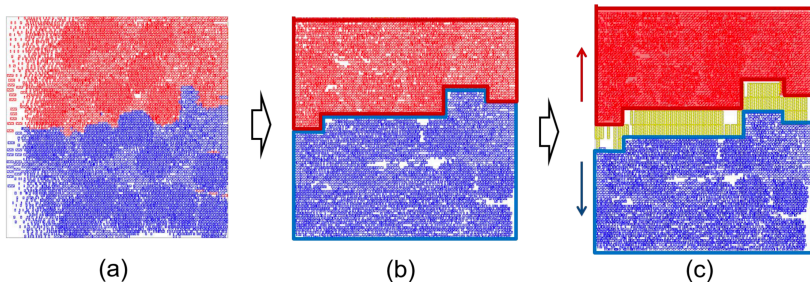
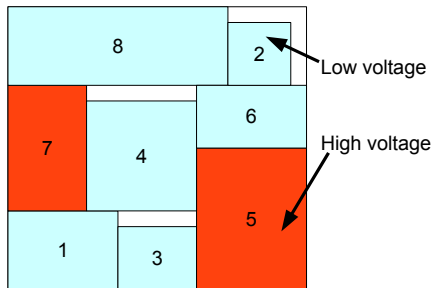
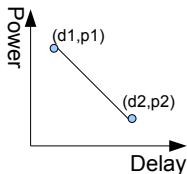


Fig. 3. Example of optimization: (a) layout-aware partitioning, (b) region definition of power domains, and (c) level shifter insertion in the updated floorplan. In blue are instances assigned to the bottom domain and in red are instances assigned to the top domain. In yellow are level shifters. Design: AES ( $\sim 11K$  instances). Technology: 28LP.

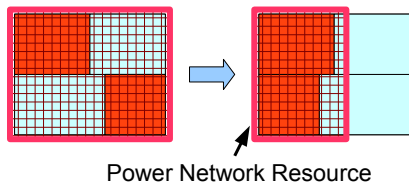
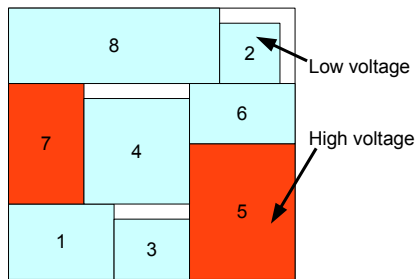
# Floorplanning Level Multi-Voltage



- ▶ Modules are assigned high-voltage or low-voltage.
- ▶ Low voltage → **high** delay.
- ▶ Trade off between the power saving and performance.



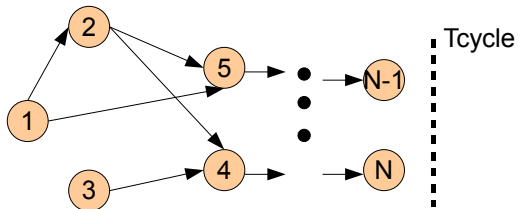
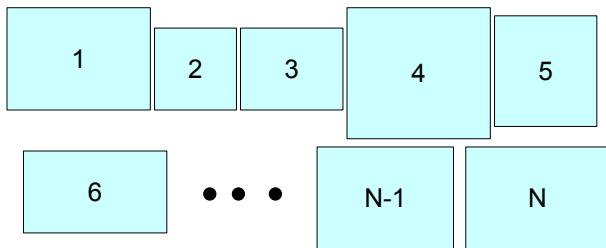
# Floorplanning Level Multi-Voltage



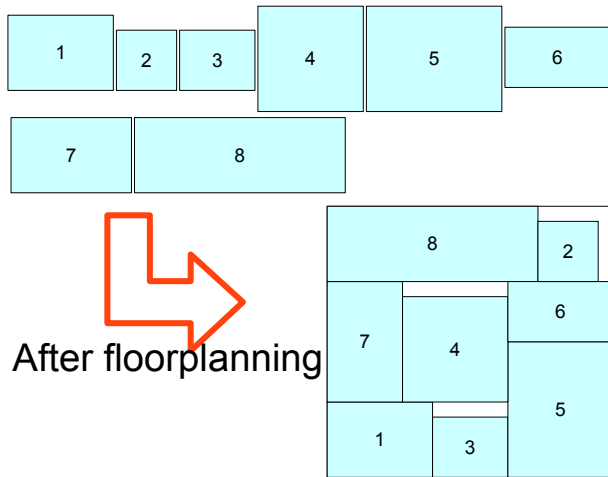
- ▶ Modules are assigned high-voltage or low-voltage.
- ▶ Low voltage → **high** delay.
- ▶ Trade off between the power saving and performance.
- ▶ Consider Power Network Resource
  - ▶ High voltage modules should pack close
  - ▶ Generate **Voltage Island**



# What's Netlist?



# What's Floorplanning?



After floorplanning



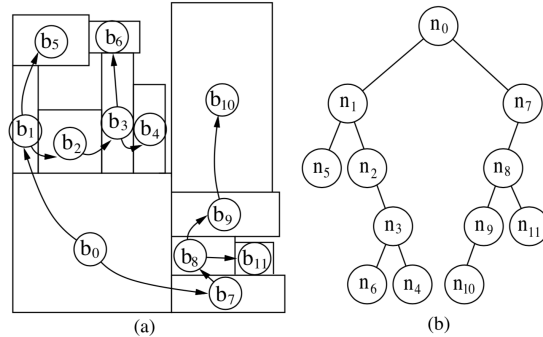


Figure 6: (a) An admissible placement. (b) The vertical B\*-tree representing the placement.

<sup>5</sup>Yun-Chih Chang et al. (2000). "B\*-Trees: A New Representation for Non-Slicing Floorplans". In: *Proc. DAC*, pp. 458–463.



# Classic Design Flow<sup>7</sup>

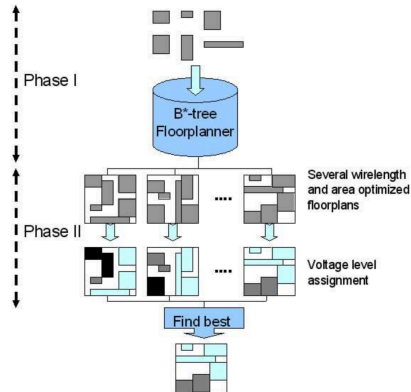


Fig. 4. Voltage island floorplanning framework.

- ▶ Integer linear programming (ILP) based
- ▶ More complicated ILP formulation is developed in ICCAD'07<sup>6</sup>.

<sup>6</sup>Wan-Ping Lee, Hung-Yi Liu, and Yao-Wen Chang (2007). "An ILP algorithm for post-floorplanning voltage-island generation considering power-network planning". In: *Proc. ICCAD*, pp. 650–655.

<sup>7</sup>Wai-Kei Mak and Jr-Wei Chen (2007). "Voltage island generation under performance requirement for SoC designs". In: *Proc. ASPDAC*, pp. 798–803.



- Let the available set of operating voltages for core  $i$  be  $\{V_{i1}, V_{i2}, \dots, V_{iL_i}\}$  where  $V_{ip}$  denotes the voltage of core  $i$  when it operates at voltage level  $p$  ( $1 \leq i \leq m$ ,  $1 \leq p \leq L_i$ ).
- Let  $P_{ip}$  denote the power consumption of core  $i$  when it operates at voltage level  $p$  ( $1 \leq i \leq m$ ,  $1 \leq p \leq L_i$ ).
- Let  $S_{ij}$  denote the number of signals from core  $i$  to core  $j$  ( $1 \leq i \leq m$ ,  $1 \leq j \leq m$ ).
- Let  $P_{ip,jq}$  denote the extra power consumption for the level shifters added (if any) to the interconnects from core  $i$  to core  $j$  when core  $i$  operates at voltage  $V_{ip}$  and core  $j$  operates at voltage  $V_{jq}$ . (Note that  $P_{ip,jq}$  is 0 when  $i = j$  or  $V_{ip} \geq V_{jq}$ , since no level shifters are inserted in these cases. For other cases,  $P_{ip,jq}$  can be pre-computed using the switching activity information of the signals.)
- Let  $\mathcal{N}_i$  denote the set of physically neighboring cores of core  $i$  ( $1 \leq i \leq m$ ).

We define binary decision variables  $b_{ip}$  ( $1 \leq i \leq m$ ,  $1 \leq p \leq L_i$ ) such that  $b_{ip}$  is 1 if core  $i$  is assigned voltage level  $p$ , and  $b_{ip}$  is 0 otherwise. It is easy to see that we should assign a single voltage for each core, so we have

$$\sum_{p=1}^{L_i} b_{ip} = 1 \quad \text{for } 1 \leq i \leq m \quad (1)$$

$$b_{ip} = 0 \text{ or } 1 \quad (2)$$



The core power consumption  $P_{cores}$  is the total power consumption of all cores. We have

$$P_{cores} = \sum_{i=1}^m \sum_{p=1}^{L_i} (P_{ip} \cdot b_{ip}) \quad (3)$$

- Let the available set of operating voltages for core  $i$  be  $\{V_{i1}, V_{i2}, \dots, V_{iL_i}\}$  where  $V_{ip}$  denotes the voltage of core  $i$  when it operates at voltage level  $p$  ( $1 \leq i \leq m$ ,  $1 \leq p \leq L_i$ ).
- Let  $P_{ip}$  denote the power consumption of core  $i$  when it operates at voltage level  $p$  ( $1 \leq i \leq m$ ,  $1 \leq p \leq L_i$ ).
- Let  $S_{ij}$  denote the number of signals from core  $i$  to core  $j$  ( $1 \leq i \leq m$ ,  $1 \leq j \leq m$ ).
- Let  $P_{ip,jq}$  denote the extra power consumption for the level shifters added (if any) to the interconnects from core  $i$  to core  $j$  when core  $i$  operates at voltage  $V_{ip}$  and core  $j$  operates at voltage  $V_{jq}$ . (Note that  $P_{ip,jq}$  is 0 when  $i = j$  or  $V_{ip} \geq V_{jq}$ , since no level shifters are inserted in these cases. For other cases,  $P_{ip,jq}$  can be pre-computed using the switching activity information of the signals.)
- Let  $\mathcal{N}_i$  denote the set of physically neighboring cores of core  $i$  ( $1 \leq i \leq m$ ).

We define binary decision variables  $b_{ip}$  ( $1 \leq i \leq m$ ,  $1 \leq p \leq L_i$ ) such that  $b_{ip}$  is 1 if core  $i$  is assigned voltage level  $p$ , and  $b_{ip}$  is 0 otherwise. It is easy to see that we should assign a single voltage for each core, so we have

$$\sum_{p=1}^{L_i} b_{ip} = 1 \quad \text{for } 1 \leq i \leq m \quad (1)$$

$$b_{ip} = 0 \text{ or } 1 \quad (2)$$

The level shifter power consumption  $P_{shifters}$  is the power consumed by the added level shifters. Level shifters consume power just like conventional buffers. As we discussed before, level shifters must be inserted to the interconnects from low voltage cores to high voltage cores. We have

$$P_{shifters} = \sum_{i=1}^m \sum_{j=1}^m \sum_{p=1}^{L_i} \sum_{q=1}^{L_j} (P_{ip,jq} \cdot b_{ip} \cdot b_{jq})$$

We define the fragmentation cost  $F$  to model the power network complexity problem. The fragmentation cost is equal to the number of pair of physically adjacent cores operating at different voltage levels. We have

$$F = \sum_{i=1}^m \sum_{j \in \mathcal{N}_i} \sum_{\substack{i < j \leq m \\ 1 < p \leq L_i \\ 1 < q \leq L_j \\ V_{ip} \neq V_{jq}}} (b_{ip} \cdot b_{jq})$$

Notice that the condition  $i < j$  is to ensure that no pair is considered twice.

However, the equations for  $P_{shifters}$  and  $F$  above are illegal in an ILP because they are non-linear. As a result, we introduce Boolean variables  $b'_{ip,jq}$  ( $1 \leq i \leq m$ ,  $1 \leq j \leq m$ ,  $1 \leq p \leq L_i$ ,  $1 \leq q \leq L_j$ ) to replace  $b_{ip} \cdot b_{jq}$  by enforcing the following artificial constraints in our ILP:

$$b'_{ip,jq} \geq b_{ip} + b_{jq} - 1 \quad (4)$$

$$b'_{ip,jq} = 0 \text{ or } 1 \quad (5)$$



# Overview

Introduction

Background: NP problem

Background: Dynamic Programming

DAC'07: Voltage Partitioning

ICCAD'06: Voltage Assignment on Netlist

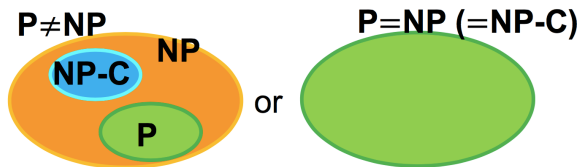
ICCAD'07: Voltage Assignment on Slicing Floorplanning



# NP-Completeness [Garey & Johnson, 1979]<sup>8</sup>

- ▶ **Decision Problem** (Yes/No Problem)
- ▶  $\mathcal{NP}$ : Set of problems w. **Nondeterministic Polynomial time algorithm**
- ▶  $\mathcal{P}$ : Set of problems w. **(Deterministic) Polynomial time algorithm**
- ▶  **$\mathcal{NP}$ -Complete**: **hardest** problems in  $\mathcal{NP}$

If a problem in  $\mathcal{NP}$ -Complete solved in polynomial time  $\rightarrow$  any problem in  $\mathcal{NP}$  solved in polynomial time



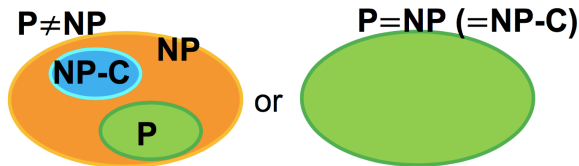
<sup>8</sup>Some contents & figures on this part come from Prof. Takahashi



# NP-Completeness [Garey & Johnson, 1979]<sup>8</sup>

- ▶ **Decision Problem** (Yes/No Problem)
- ▶  $\mathcal{NP}$ : Set of problems w. **Nondeterministic Polynomial time algorithm**
- ▶  $\mathcal{P}$ : Set of problems w. **(Deterministic) Polynomial time algorithm**
- ▶  **$\mathcal{NP}$ -Complete**: **hardest** problems in  $\mathcal{NP}$
- ▶ **Conjecture**:  $\mathcal{P} \neq \mathcal{NP}$

If a problem in  $\mathcal{NP}$ -Complete solved in polynomial time  $\rightarrow$  any problem in  $\mathcal{NP}$  solved in polynomial time

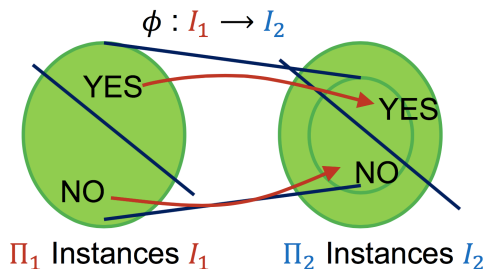




<sup>8</sup>Some contents & figures on this part come from Prof. Takahashi



# Polynomial Time Reduction

- Provides difficulty relation between problems



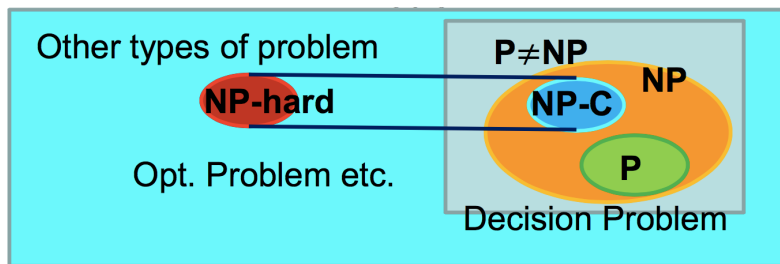
$\Pi_1$	$\propto$	$\Pi_2$
Easy		Easy
Difficult		Difficult

- SAT is  $\mathcal{NP}$ -Complete  $\rightarrow$  3SAT, Hamilton, TSP, Coloring...



# NP-Hardness [Garey & Johnson,1979]

- ▶ Optimization problem
- ▶ Is neither in  $\mathcal{NP}$  nor in  $\mathcal{NP}$ -Complete
- ▶  **$\mathcal{NP}$ -hard** if a related decision problem is  $\mathcal{NP}$ -complete
- ▶ E.g. Travelling Salesman Problem (TSP)
- ▶ No polynomial time algorithm



# Strategies of Algorithm Design

1. Check whether problem is easy or not?
2. If possible, prove is  $\mathcal{NP}$ -hard or  $\mathcal{NP}$ -complete
3. For easy problem (in  $\mathcal{P}$ ):
4. For not easy problem (in  $\mathcal{NP}$ -hard):



# Overview

Introduction

Background: NP problem

**Background: Dynamic Programming**

DAC'07: Voltage Partitioning

ICCAD'06: Voltage Assignment on Netlist

ICCAD'07: Voltage Assignment on Slicing Floorplanning



## Case 1: Calculating Binomial Coefficient

Consider the problem of calculating the binomial coefficient

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \\ 0 & \text{otherwise.} \end{cases}$$

Suppose  $0 \leq k \leq n$ . If we calculate  $\binom{n}{k}$  directly by

```
function C(n, k)
  if k = 0 or k = n then return 1
  else return C(n - 1, k - 1) + C(n - 1, k)
```

many of the values  $C(i, j)$ ,  $i < n$ ,  $j < k$ , are calculated over and over.



# Case 1: Calculating Binomial Coefficient

Consider the problem of calculating the binomial coefficient

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \\ 0 & \text{otherwise.} \end{cases}$$

Suppose  $0 \leq k \leq n$ . If we calculate  $\binom{n}{k}$  directly by

```
function C(n, k)
    if k = 0 or k = n then return 1
    else return C(n - 1, k - 1) + C(n - 1, k)
```

many of the values  $C(i, j)$ ,  $i < n$ ,  $j < k$ , are calculated over and over.

## Question

Can we have a better Algorithm?



## Case 2: Knapsack Problem

$$\begin{aligned} \max \quad & \sum_{i=1}^n x_i v_i \\ \text{s.t.} \quad & \sum_{i=1}^n x_i w_i \leq W \end{aligned}$$

- ▶  $v_i$ : value of object  $i$
- ▶  $w_i$ : weight of object  $i$
- ▶  $x_i \in \{0, 1\}$

### Question

- ▶ Design a Dynamic Programming Algorithm to Solve it.
- ▶ What is  $x_i$  can be floating value?



## Principle of Optimality

In an optimal **sequence** of decisions or choices, each subsequence must also be **optimal**.



# Overview

Introduction

Background: NP problem

Background: Dynamic Programming

**DAC'07: Voltage Partitioning**

ICCAD'06: Voltage Assignment on Netlist

ICCAD'07: Voltage Assignment on Slicing Floorplanning



Specifically, if  $d = 3$ , then  $L_1 = \langle u_1, u_2, \dots, u_{|L_1|} \rangle$ ,  $L_2 = \langle u_{|L_1|+1}, \dots, u_{|L_1|+|L_2|} \rangle$ , and  $L_3 = \langle u_{|L_1|+|L_2|+1}, \dots, u_n \rangle$ .

Before presenting our algorithm for OVPP, we first define four notations for clear presentation:

- $C_{i,j} \equiv \sum_{q=i}^j c_q$ ,  $i \leq j$ , i.e., the cumulative capacitance from the unit  $i$  to the unit  $j$ .
- $E_p^i \equiv$  the total energy of  $\langle \langle u_1, \dots, u_i \rangle, \langle u_{i+1}, \dots, u_p \rangle \rangle$ ,  $i \in \{1, \dots, p-1\}$ , i.e., the total energy of an ordered *bi-partition* of the first  $p$  units with a cut right behind the unit  $i$ . By Definition 1,  $E_p^i = C_{1,i} \cdot v_i^2 + C_{i+1,p} \cdot v_p^2$ .
- $E_p^* \equiv \min_{i=1, \dots, p-1} E_p^i$ , i.e., the minimum total energy of ordered *bi-partitions* of the first  $p$  units.
- $S_p^* \equiv j$  such that  $E_p^j = E_p^*$ , i.e., an optimal cut position which yields the minimum total energy  $E_p^*$ .

Now we can have the following recurrence as a recursive solution to OVPP for *dual-Vdd*'s. For  $p = 2, \dots, n-1$ ,

$$E_{p+1}^* = \begin{cases} E_{p+1}^{S_p^*} & \text{if } v_{p+1} = v_p, \\ \min_{i=S_p^*, \dots, p} E_{p+1}^i & \text{if } v_{p+1} > v_p. \end{cases} \quad (1)$$

The correctness of the recurrence can be proven by mathematical induction. Specifically, given that an optimal cut position  $S_p^*$  for the first  $p$  functional units is known, and that we request for an optimal cut position of the first  $p+1$  functional units, then Equation (1) implies to keep the known optimal position  $S_p^*$  if  $v_{p+1} = v_p$ , or to find an optimal cut from the known optimal position to the position right behind the functional unit  $p$  if  $v_{p+1} > v_p$ . Note that it is trivial to show the initial conditions  $S_2^* = 1$  and  $E_2^* = c_1 \cdot v_1^2 + c_2 \cdot v_2^2$ . By the recurrence,  $E_n^*$  yields the minimum total energy of ordered bi-partitions of all  $n$  functional units, and  $S_n^*$  records an optimal cut position for  $E_n^*$ .

Given Equation (1), the minimum total energy of all  $n$  functional units using *triple-Vdd*'s can further be computed by

$$\min_{p=2, \dots, n-1} \{E_p^* + C_{p+1,n} \cdot v_n^2\}. \quad (2)$$

$u_i$	$u_1$	$u_2$	$u_3$	$u_4$	$u_5$	$u_6$
$c_i$	2.0	0.2	3.0	1.0	1.5	1.5
$v_i$	0.8	1.0	1.0	1.0	1.1	1.2

(a) Ordered Functional Units

$i$	1	2	3	4	5	6
$C[i]$	2.0	2.2	5.2	6.2	7.7	9.2
$S_i^*$	-	1	1	1	4	4
$E_i^*$	-	1.48	4.48	5.48	8.02	10.52

(b) Dual-Vdd Partitioning

$S_r$	-	2	3	4	5	-
$E_{min}$	-	11.56	10.24	9.80	10.18	-
$S_l$	-	1	1	1	4	-

(c) Triple-Vdd Partitioning



## 4. PROBLEM COMPLEXITY

Although Gu *et al.* claim that their polynomial-time algorithm is exact for VPP [7], we show that their claim is incorrect by proving the NP-hardness of VPP. Before proving the NP-hardness, we first point out the flaw in the lemma for which Gu *et al.* try to prove for the correctness of their algorithm. Lemma 1 of [7] (with the notations used in this paper) contains the inequality:  $v(F_i) \leq v_j, \forall u_j \in F_i$ . However, by Definition 1, the direction of the inequality should be reversed, and the mistake in their optimality claim thus follows. Now, we can prove the following theorem by restriction [6]: reducing the perfect number-partitioning problem [6], a well-known NP-complete problem, to VPP.

THEOREM 1. *VPP is NP-hard.*

## 5.4.2 Performance Bound of FOVP for VPP

Since VPP is NP-Hard, it is desired to develop an efficient approximation algorithm with a guaranteed constant performance bound. Fortunately, we have the following good property from Definition 1.

PROPERTY 1. *Any algorithm is an  $\alpha^2$ -approximation algorithm for VPP, if the maximum available voltage is  $\alpha$  times as large as the minimum one.*

Now we can further show that the FOVP algorithm never reaches the  $\alpha^2$  performance bound, i.e., FOVP never produces such a worst-case solution, in which all the functional units are operated at the globally maximum mapped voltage.

THEOREM 3. *The FOVP algorithm never produces a worst-case solution to VPP.*

Ckt	# of Units	Power Saving	Running Time (s)		Speedup
		[7] and FOVP	[7]	FOVP	
t1	1001	68.02%	0.09	<0.01	N/A
t2	2000	67.69%	0.36	0.01	36X
t3	2999	67.34%	0.84	0.01	84X
t4	3999	67.53%	1.61	0.02	80X
t5	4999	67.66%	2.43	0.02	121X
t6	5999	67.84%	3.62	0.02	181X
t7	6999	67.50%	5.17	0.03	172X
t8	7999	67.61%	6.47	0.03	215X
t9	8999	67.75%	8.29	0.04	206X
t10	9999	67.64%	10.20	0.04	255X

► On perfect-number partition: [https://en.wikipedia.org/wiki/Partition\\_problem](https://en.wikipedia.org/wiki/Partition_problem)

► Correction:<sup>10</sup>

<sup>10</sup>Tao Lin et al. (2010). "A revisit to voltage partitioning problem". In: *Proc. GLSVLSI*, pp. 115–118.



# Overview

Introduction

Background: NP problem

Background: Dynamic Programming

DAC'07: Voltage Partitioning

ICCAD'06: Voltage Assignment on Netlist

ICCAD'07: Voltage Assignment on Slicing Floorplanning



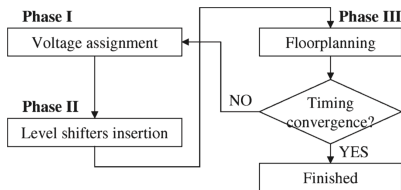
# ICCAD'06: Voltage Assignment on Netlist<sup>11</sup>

Given  $k$  choices of supply voltages  $VDD_j$ ,  $1 \leq j \leq k$ , an  $n$ -vertex DAG  $G = (V, E)$  and delay  $d_i$  for each vertex  $v_i \in V$ ,  $d_i \in \{d_i^1, d_i^2, \dots, d_i^k\}$ , where  $d_i^j$  denotes the delay of a vertex  $v_i$  operated at the  $j$ th voltage domain  $VDD_j$ , according to static timing analysis (STA), the arrival time  $a_i$  and the required time  $r_i$  of  $v_i$  are derived as follows:

$$a_i = \begin{cases} \max_{v_j \in FI_i} a_j, & FI_i \neq \phi \\ 0, & FI_i = \phi \end{cases} \quad (2)$$

$$r_i = \begin{cases} \min_{v_j \in FO_i} a_j - d_i, & FO_i \neq \phi \\ T_{\text{cycle}}, & FO_i = \phi \end{cases} \quad (3)$$

where  $FI_i$  and  $FO_i$  are sets of the fan-in and fan-out vertices of  $v_i$ , respectively, and  $T_{\text{cycle}}$  is the clock cycle time of the netlist. Using the STA model, we define the static-timing constraint as follows.



(a) Algorithm Flow

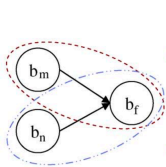
(b) Notations

## Question:

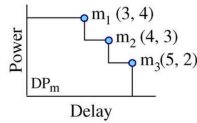
- ▶ How to define a slack for each vertex  $v_i$ ?
- ▶ Please provide a mathematical formulation minizing total power consumption.

<sup>11</sup>Wan-Ping Lee, Hung-Yi Liu, and Yao-Wen Chang (2006). "Voltage island aware floorplanning for power and timing optimization". In: *Proc. ICCAD*, pp. 389–394.

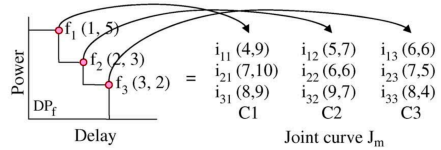




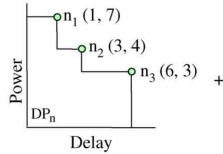
$J_m =$



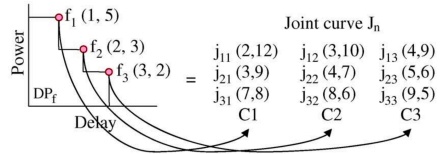
+



$J_n =$



+



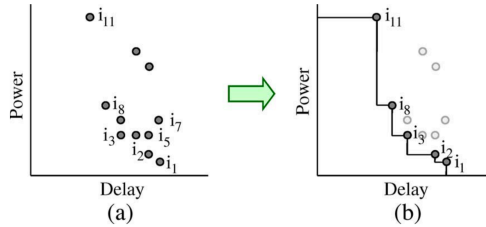


Fig. 11. Two processes of redundant-point pruning: sorting and pruning. (a) Sort all points by their  $y$ -coordinates.  $i_j$  represents that this point is the  $j$ th lowest in the curve. (b) The final pruned result; there are five nonredundant points.



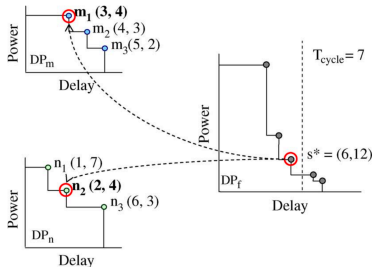


Fig. 12. Backtracing procedure for finding an optimal solution. According to  $T_{\text{cycle}}$ , we identify the best resulting  $s^*$  point in the DP curves of POs. Then, we backtrace an optimal solution of each block.

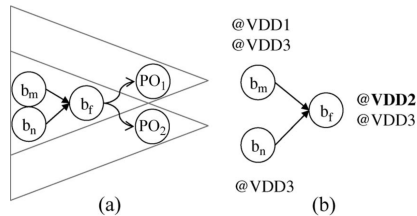


Fig. 13. According to Definition 10,  $b_f$  is the common block in  $PO_1$ 's and  $PO_2$ 's fan-in cones. (a)  $PO_1$  and  $PO_2$  share some blocks, as in the overlapping portion. (b) After backtracing a solution, these overlapped blocks may be set in several different voltages. Assign the highest one to the common block  $b_f$  and then run the voltage-assignment algorithm again to find a better solution.

- Further speed-up: dual to min-cost flow<sup>12</sup>
- Overcome reconverge issue:<sup>13</sup>

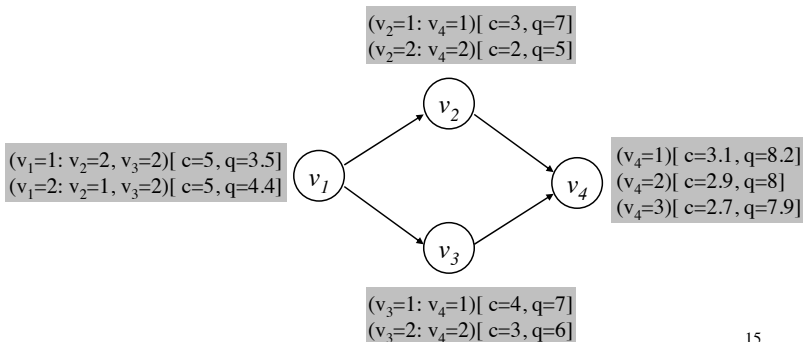
<sup>12</sup>Qiang Ma and Evangeline FY Young (2008). "Network flow-based power optimization under timing constraints in MSV-driven floorplanning". In: *Proc. ICCAD*, pp. 1–8.

<sup>13</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Relaxation

← backward solution propagation

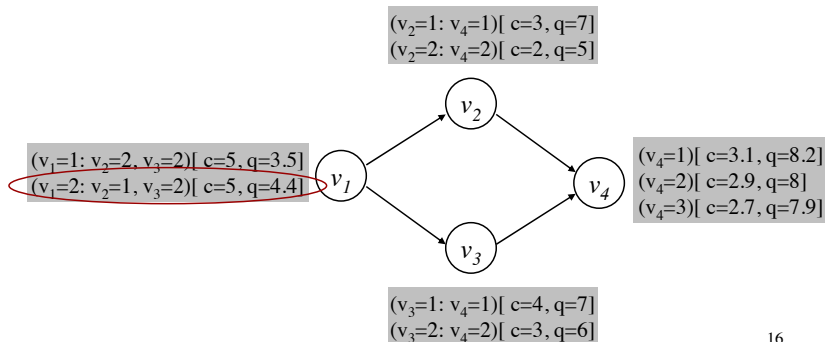


15

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Relaxation

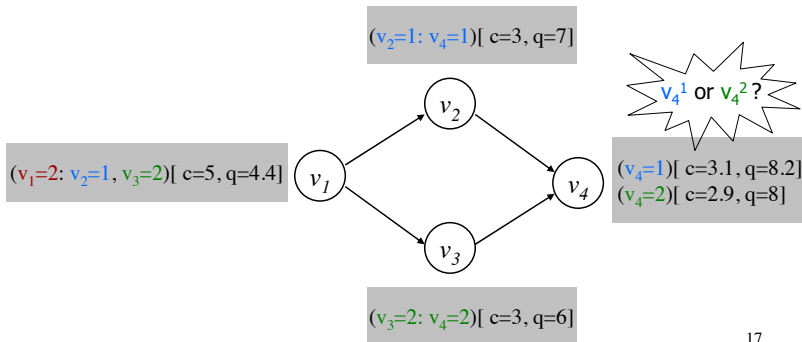


16

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Relaxation

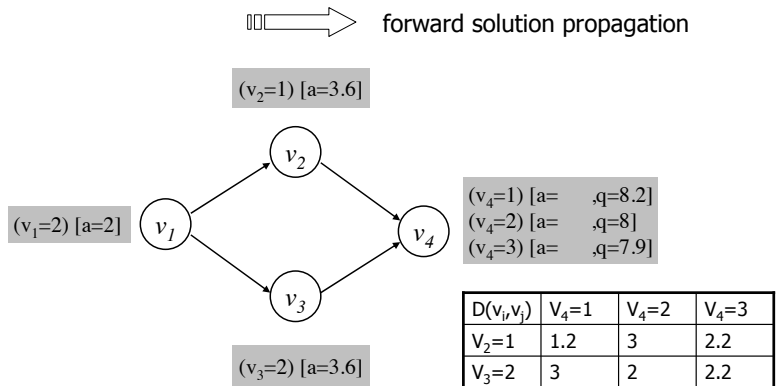


17

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Restoration

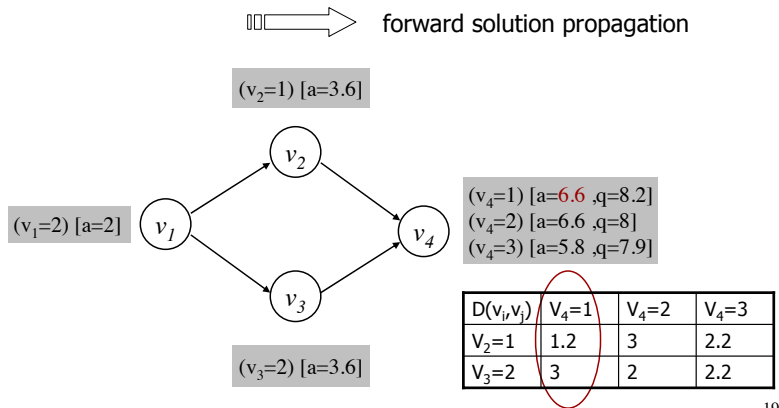


18

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Restoration

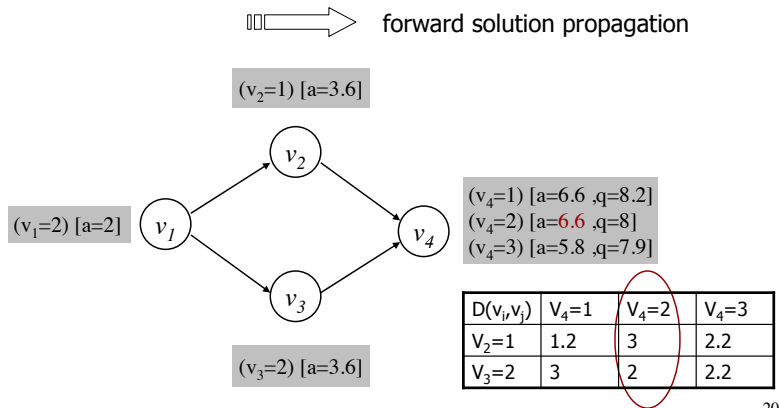


19

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Restoration

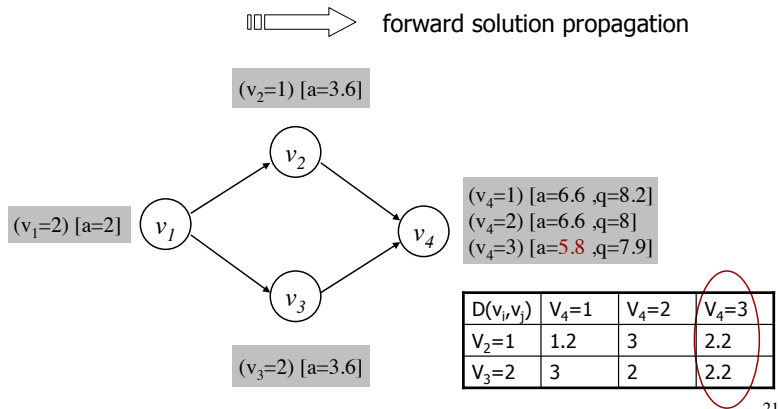


20

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



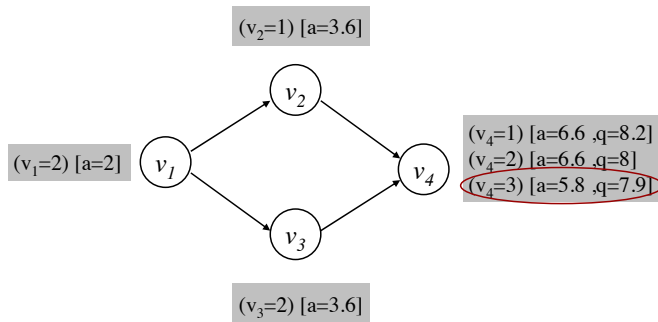
# Consistency Restoration



<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Consistency Restoration



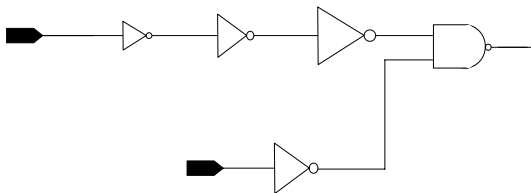
22

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

Circuit in consideration



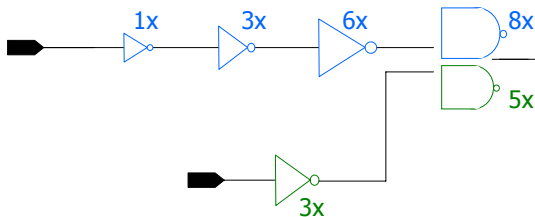
24

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

After relaxation



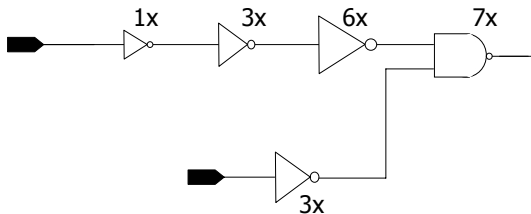
25

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

After Phase I restoration



26

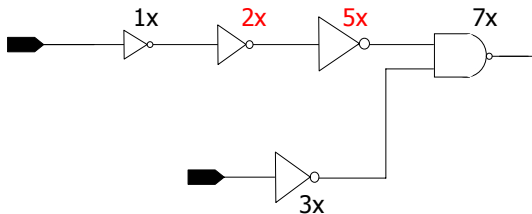
<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

Phase II begins

← solution propagation direction

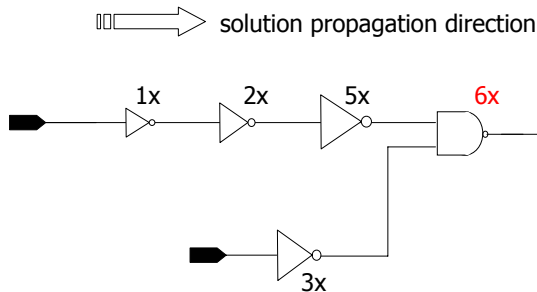


27

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

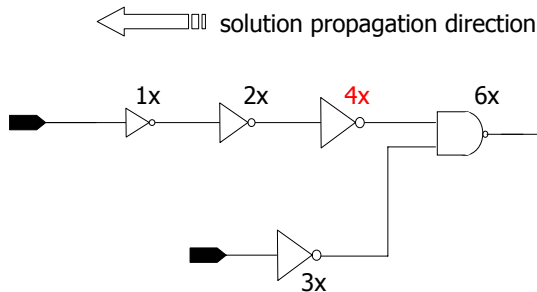


28

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

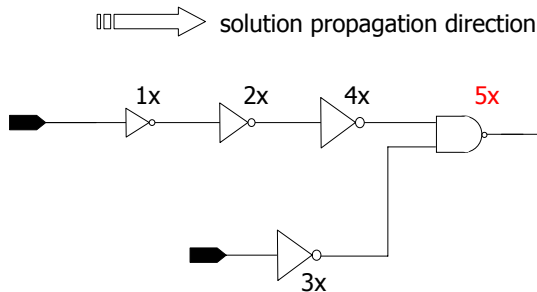


29

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement



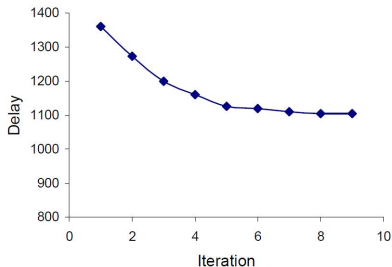
30

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Iterative Refinement

- Monotonic improvement of solution by iterative refinement



31

<sup>14</sup>Yifang Liu and Jiang Hu (2009). "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment". In: *Proc. ISPD*, pp. 27–34.



# Overview

Introduction

Background: NP problem

Background: Dynamic Programming

DAC'07: Voltage Partitioning

ICCAD'06: Voltage Assignment on Netlist

ICCAD'07: Voltage Assignment on Slicing Floorplanning



# ICCAD'07: Voltage Assignment on Slicing Floorplanning<sup>15</sup>

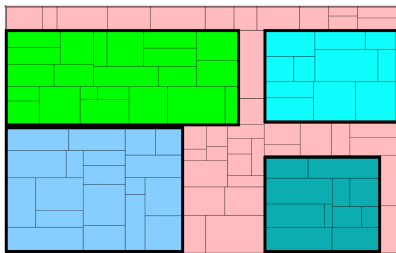
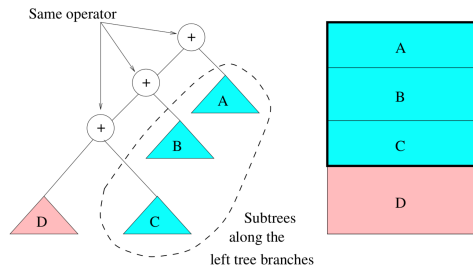


Fig. 3. One resultant floorplan of  $n100$  with four voltage islands.



A, B and C, not in one subtree, can form one voltage island.

Fig. 2. An example of forming island across subtrees.

<sup>15</sup>Qiang Ma and Evangeline FY Young (2007). "Voltage island-driven floorplanning". In: *Proc. ICCAD*, pp. 644–649.

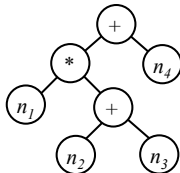


# Normalized Polish Expression (NPE)

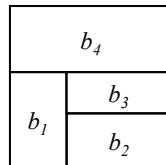
- Slicing floorplan representation
- A sequence of *operands* and *operators*
  - An *operand* denotes a block
  - An *operator* denotes a cut direction
    - '+' denotes a horizontal cut
    - '\*' denotes a vertical cut

$n_1 n_2 n_3 +^* n_4 +$

NPE



Slicing tree

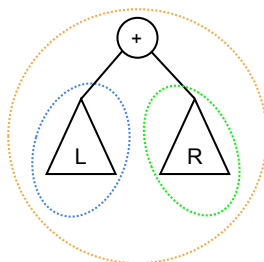


Slicing Floorplan



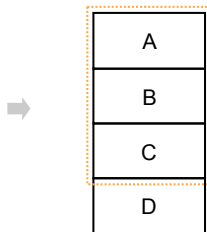
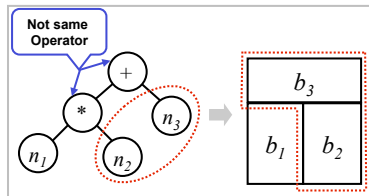
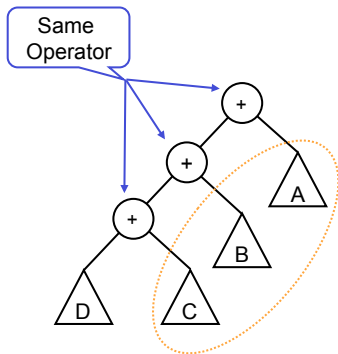
# Optimal Island Partitioning

- Given a candidate floorplan represented by NPE, we can perform *optimal* island partitioning and voltage assignment on the slicing tree
- Procedure TreePart(TreeNode  $u$ , Num\_island  $k$ )
  - Optimally partition a tree rooted at  $u$  into  $k$  islands
  - Solved by dynamic programming
- When  $k = 1$ 
  - Case 1 : Island in left subtree
  - Case 2 : Island in right subtree
  - Case 3 : The whole tree rooted at  $u$  form an island
  - Case 4 : Island is formed across the left and right subtrees



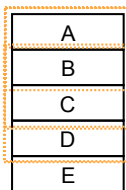
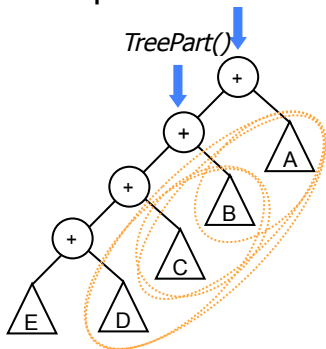
# Optimal Island Partitioning

- Case 4 : Island is formed across subtrees
  - ⊙ A set of contiguous right subtrees may also form an island when operators along the left tree branches are the same



# Optimal Island Partitioning

- The procedure **NonSubtree()** deals with case 4



NonSubtree (TreeNode  $u$ , num\_island  $k$ )

- ◆  $min\_cost = \infty$
- ◆  $S = \text{right\_child}(u)$
- ◆  $op = \text{operator}(u)$
- ◆ While  $\text{operator}(\text{left\_child}(u))$  is  $op$ 
  - $u = \text{left\_child}(u)$
  - $S = S \cup \text{right\_child}(u)$
  - $C = \text{TreePart}(\text{left\_child}(u), k-1) + \text{cost}(S)$
  - If  $min\_cost > C$ ,  $min\_cost = C$
- ◆ Return( $min\_cost$ )

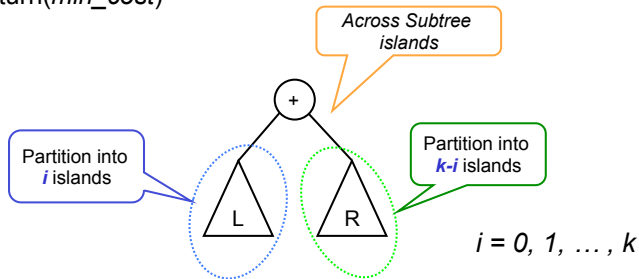


# Optimal Island Partitioning

- When  $k$  is more than 1, exhaust all different ways of distributing the  $k$  islands by *dynamic programming*

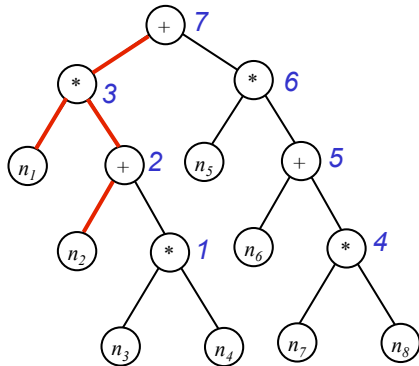
TreePart (TreeNode  $u$ , num\_island  $k$ )

- ◆  $\text{min\_cost} = \text{NonSubtree}(u, k)$
- ◆ For  $i = 0$  to  $k$ 
  - $C = \text{TreePart}(\text{left\_child}(u), i) + \text{TreePart}(\text{right\_child}(u), k-i)$
  - If  $\text{min\_cost} > C$ ,  $\text{min\_cost} = C$
- ◆ Return( $\text{min\_cost}$ )



# Optimal Island Partitioning

- Use the Cost Table to speed up *Procedure TreePart* ( $u, k$ )
  - Store the best partitioning solution of each node
    - Minimize the number of recursive calls - a dynamic programming technique
  - After each move, only the nodes lying on the path from the perturbed node to the root need to be updated



$n_1 n_2 n_3 n_4 * + n_5 n_6 n_7 n_8 * + +$

Cost Table:

Node	No. of Islands					
	1	2	3	...	K-1	K
1 ( *)						
2 (+)						
3 ( *)						
4 ( *)						
5 (+)						
6 ( *)						
7 (+)						



# Takeaway

- ▶  $\mathcal{NP}$  completeness and  $\mathcal{NP}$  hardness
- ▶ Dynamic Programming: when and how
- ▶ How to evaluate previous work

