Optimum Prefix Adders in a Comprehensive Area, Timing and Power Design Space

Jianhua Liu[1], Yi Zhu[1], Haikun Zhu[1],
John Lillis[2], Chung-Kuan Cheng[1]

[1]Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093

[2]Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607

Outline

- Previous Works and Motivation
- Area/Timing/Power Model
- ILP Formulation of Prefix Adder
- Experimental Results
- Conclusions

Previous Works and Motivation

- Parallel prefix adder is the most flexible and widely-used binary adder for ASIC designs.
- Prefix network formulation:

Pre-processing:

$$g_i = a_i b_i$$

$$p_i = a_i \oplus b_i$$

Prefix Computation:

$$G_{[i:k]} = G_{[i:j]} + P_{[i:j]} G_{[j-1:k]}$$

$$P_{[i:k]} = P_{[i:j]} P_{[j-1:k]}$$

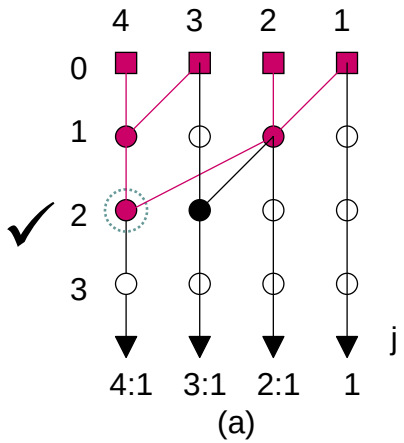
Post-processing:

$$c_{i+1} = G_{[i:0]} + P_{[i:0]} \cdot c_0$$

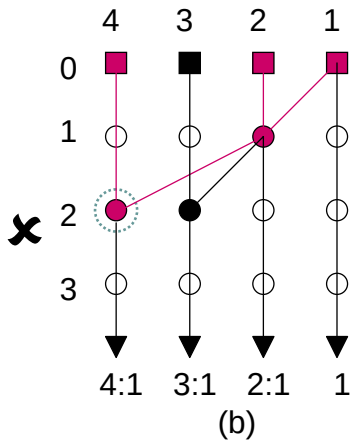
$$s_i = p_i \oplus c_i$$

Previous Works and Motivation

- Each output network is an **alphabetical tree**:
  - Output i is the root of a binary tree covering inputs 1-i.
  - An in-depth traversal of the tree terminals follows the sequence of the inputs

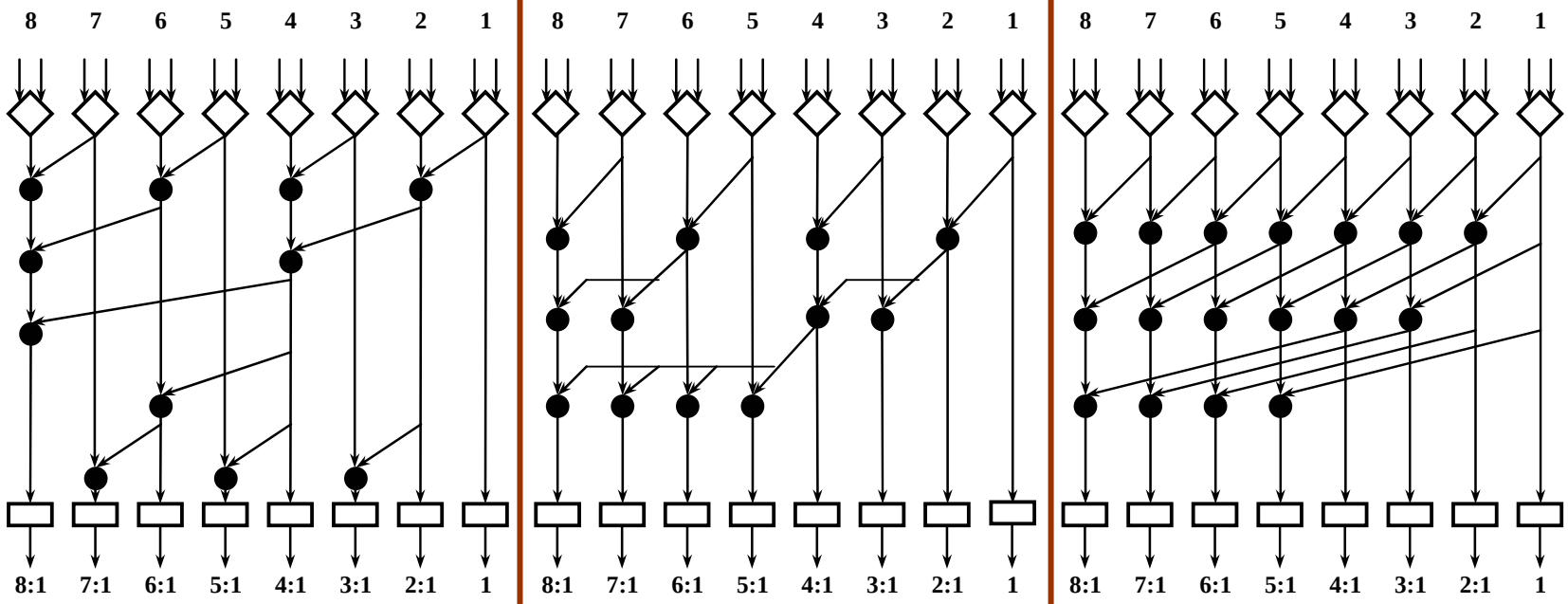


(a)



(b)

Previous Works and Motivation



Brent-Kung:
Logical levels: $2\log_2 n - 1$
* Max fanouts: 2
Wire tracks: 1

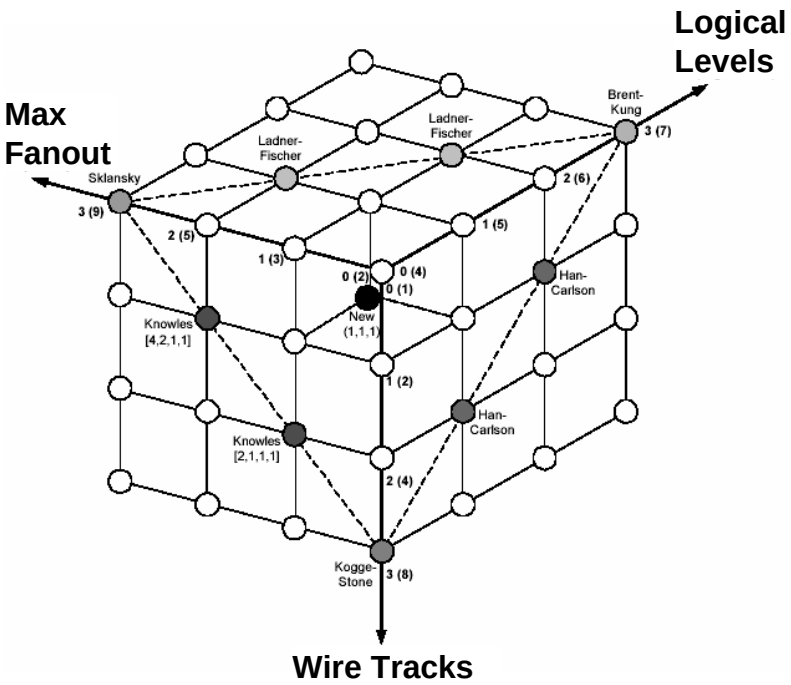
Sklansky:
Logical levels: $\log_2 n$
Max fanouts: n/2
Wire tracks: 1

Kogge-Stone:
Logical levels: $\log_2 n$
Max fanouts: 2
Wire tracks: n/2

**** Max Fanouts is based on the regular buffer insertions at all empty space***

Previous Works and Motivation

- The design space of prefix adder is considered as the tradeoff among logical levels, max fanouts and wire tracks. * Harris D, "A Taxonomy of Parallel Prefix Networks" Nov. 2003.



$$\text{Logical levels}: L = \log_2 n + l$$
$$\text{Max fanouts} : F = 2^f + 1$$
$$\text{Wire tracks} : T = 2^t$$
$$l + f + t = \log_2 n - 1$$

Timing: $L \times F \times Dgp$

(*Dgp*: delay of one GP adder with unit load)

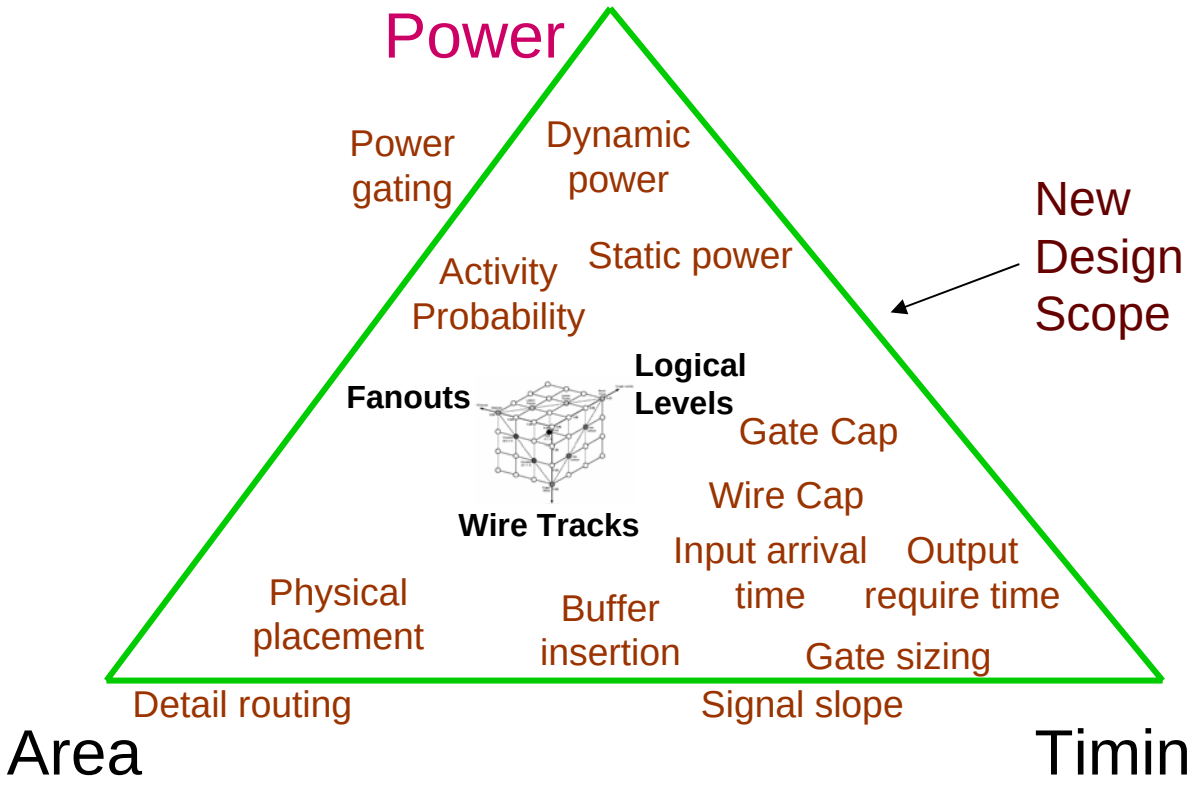
Area: $L \times (Hgp + T \times Hwt) \times n$

(*Hgp*: Height of one GP adder
Hwt: Height of one wire track)

Favor the minimal logical levels

Previous Works and Motivation

- Increasing impact of physical design.
- Power becomes a critical concern.

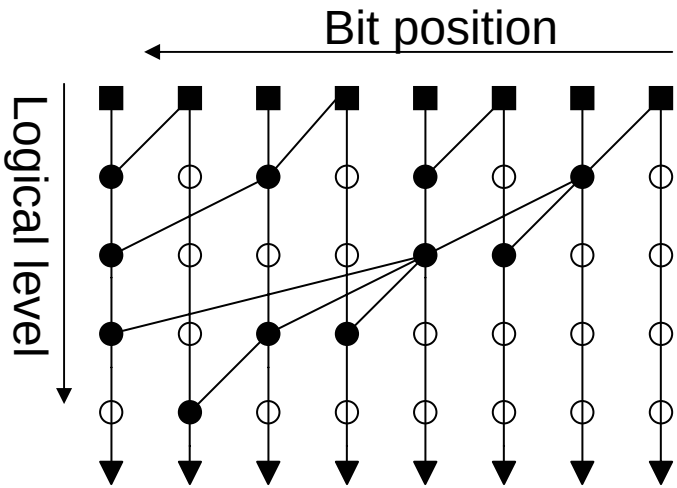


Previous Works and Motivation

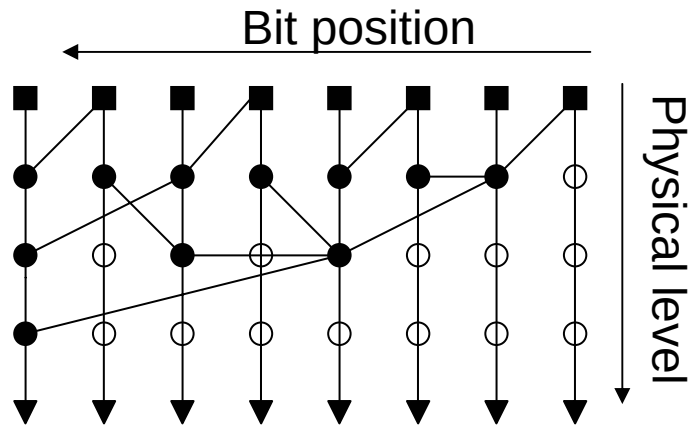
- Input: bit width, physical area, input arrival times, output required times.
- Output: placed prefix adder
- Constraint: alphabetical tree rooted at each output *i* to cover inputs *1* to *i*,
  area and timing requirements
- Objective: minimize power consumption

Models – Area Model

- Distinguish physical placement from logical structure, but keep the bit-slice structure.



Logical view of
Brent-Kung adder



Compact placement

Physical view of
Brent-Kung adder

$$A = n \times m$$

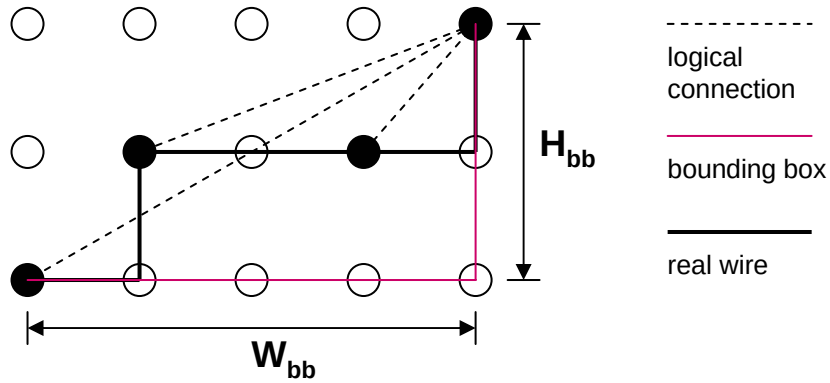
n: Bit width
m: Physical depth

Models – Timing Model

- Cload includes both gate and wire capacitance. Wire capacitance is proportional to wire length.

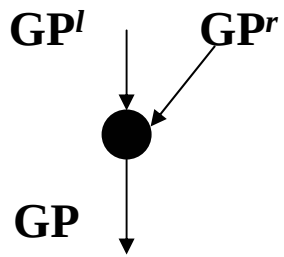
$$C_{load} = C_{wire} + C_{gate}$$
$$C_{wire} = \lambda^w \times (H_{bb} + W_{bb})$$
$$C_{gate} = \Sigma\, C_{in}$$

$(\lambda^w = 0.5)$



- Use a linear timing model derived from logical effort.

$$Delay\_GP^l = 1.5\, C_{load} + 2.5$$
$$Delay\_GP^r = 2.0\, C_{load} + 2.5$$



** Harris D, Sutherland I, "Logical Effort of Carry Propagate Adders", 2004.*

Models – Power Model

- Total power consumption:

  Dynamic power + Static Power

- Static power: leakage current of device

  $P_{sta} = \lambda^s$ $(\lambda^s = 0.5)$

- Dynamic power: current switching capacitance

  $P_{dyn} = \rho \times C_{load}$

- ρ is the switching probability

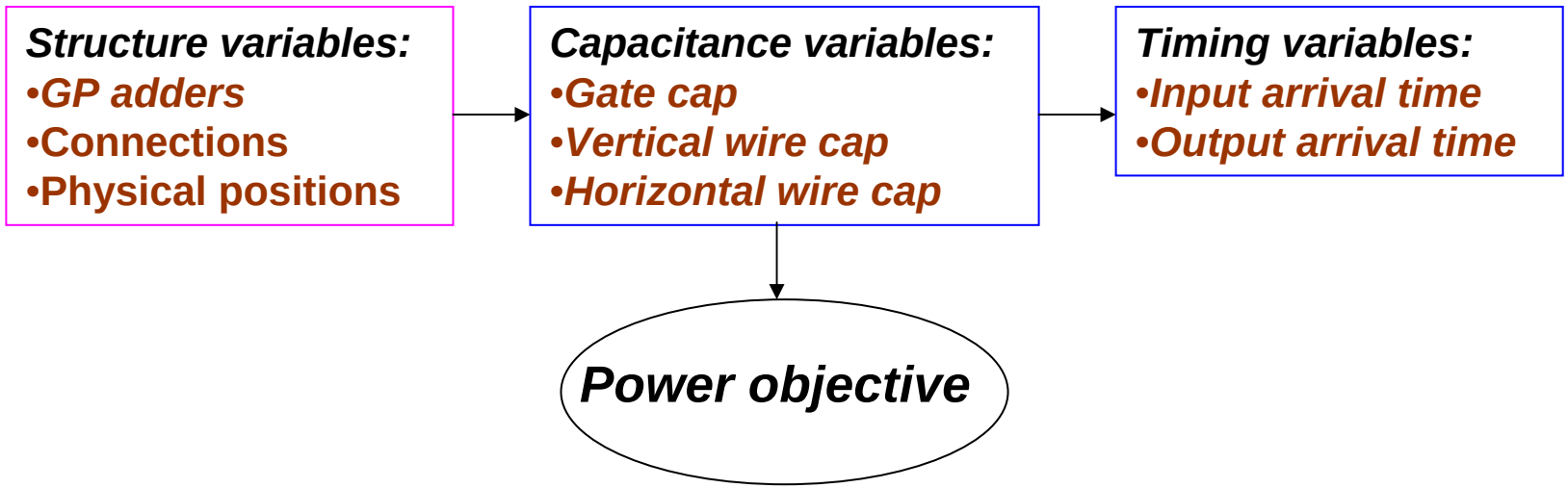
  $\rho = j$ (j is the logical level*)

$$P_{total} = P_{dyn} + P_{sta} = j \cdot C_{load} + \lambda^s$$

* Vanichayobon S, etc, "Power-speed Trade-off in Parallel Prefix Circuits", 2002

ILP on Prefix Adder – Overall Picture

- We propose to formulate prefix computation as Integer Linear Programming (ILP) problem.
- Optimum solution can be produced by contemporary ILP solver.



Structure variables defines the ILP solution space

ILP – Linear Programming

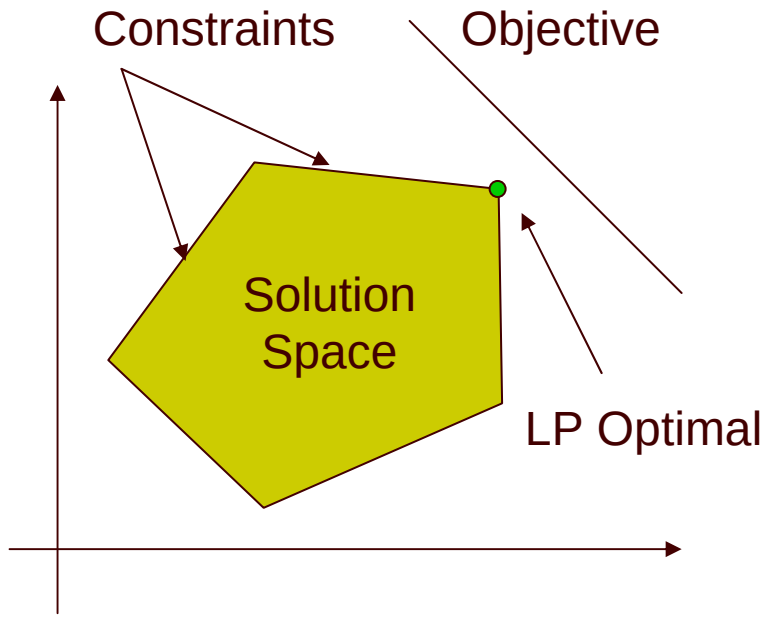
- Linear Programming: linear constraints, linear objective, fractional variables.

Maximize: $x_1 + 2\,x_2 + 3\,x_3$

subject to: $-\,x_1 + x_2 + x_3 \leq 20$

$x_1 - 3\,x_2 \leq 30$

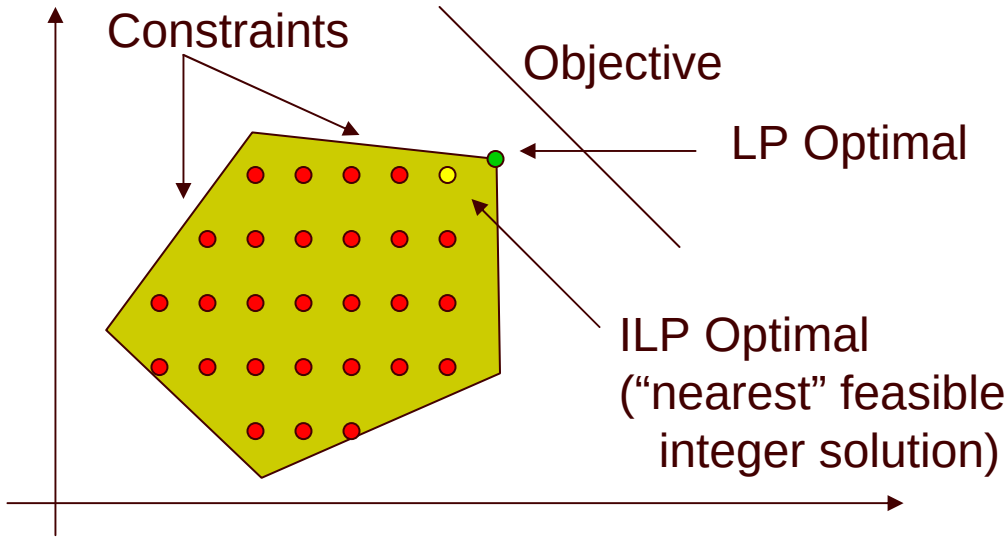
$$\begin{bmatrix} -1 & 1 & 1 \\ 1 & -3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 20 \\ 30 \end{bmatrix}$$



LP problems are polynomial time solvable (interior point algorithm, Karmarkar 1984)

ILP – Integer Linear Programming

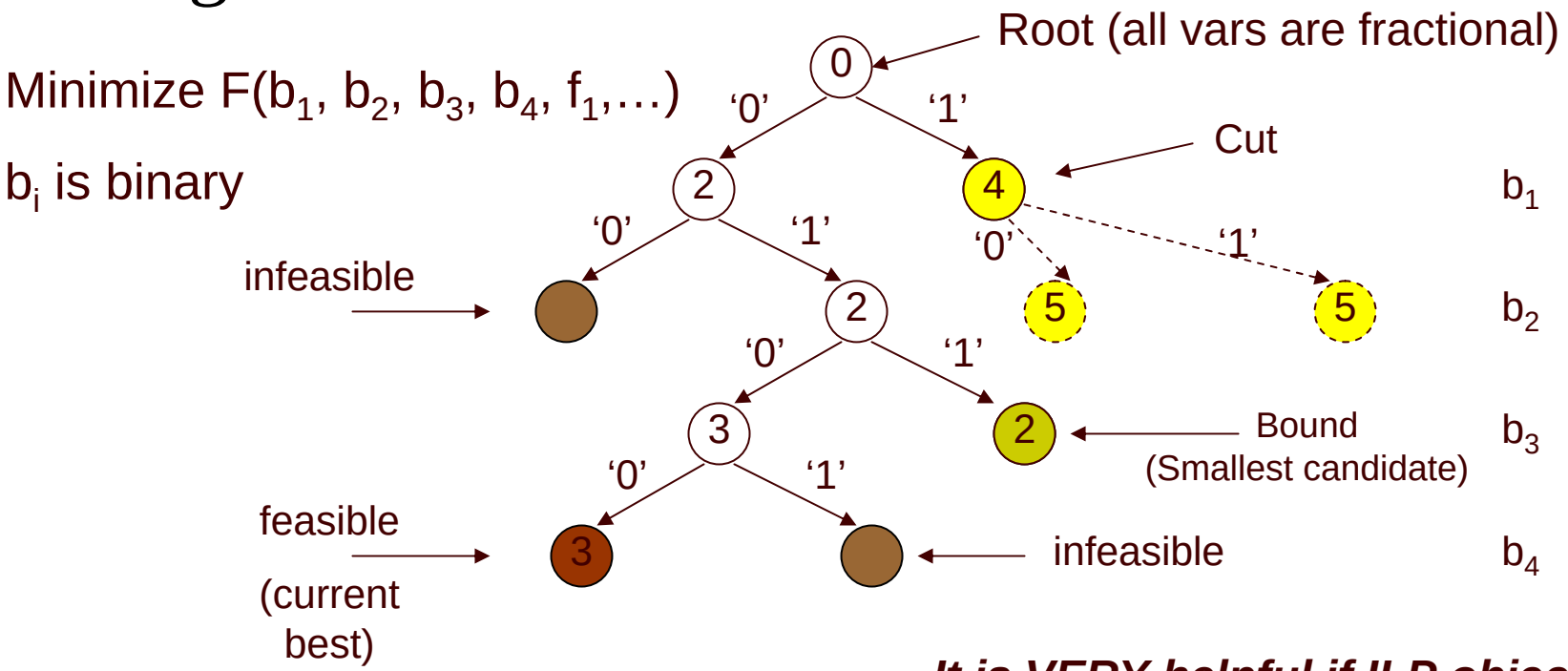
- Integer Linear Programming: all variables are integers.



ILP problem with bounded variables is NP-hard.

ILP – Branch and Bound

- Brach and bound with linear relaxation algorithm in ILP solvers:



It is VERY helpful if ILP objective is close to LP objective

ILP – Pseudo-Linear Constraint

- A constraint is called pseudo-linear if it's not effective until some integer variables are fixed.

Problem:

Minimize: x_3
Subject to: $x_1 \geq 300$
$x_2 \geq 500$
$x_3 = \min(x_1, x_2)$

LP objective: 0

ILP objective: 300

ILP formulation:

Minimize: x_3
Subject to: $x_1 \geq 300$
$x_2 \geq 500$
$x_3 \leq x_1$
$x_3 \leq x_2$
$x_3 \geq x_1 - 1000\, b_1$ (1)
$x_3 \geq x_2 - 1000\,(1 - b_1)$ (2)
b_1 is binary

- Pseudo-linear constraints mostly arise from IF/ELSE scenarios
  - binary decision variables are introduced to indicate true or false.

ILP – Summary

- Integer Linear Programming is a powerful solution space search tool guided by Linear Programming.
- However, pseudo-linear constraints may compromise the efficiency.

ILP on Prefix Adder – Structure

- ILP decision variables represent GP adders and interconnects in logical view.
- Alphabetical tree rooted at each output.
  - Each GP adder has exact one left input and one right input. (*fanin const.*)
  - At least one input is from the previous level. (*logical level const.*)
  - Every GP adder roots an alphabetical tree covering a continuous segment. (*root const.*)

ILP on Prefix Adder – Structure

Variables:

- *gp(i,j)* {0,1}: GP adders in the $n \times d$ array (*d*: logical depth)
- *wl(i,j,h)* {0,1}: The wire from *(i,h)* to the left fanin of *(i,j)*
- *wr(i,j,k,l)* {0,1}: The wire from *(k,l)* to the right fanin of *(i,j)*

Constraints:

- (*fanin const.)* One left/right fanin for each GP adder

$$\sum_{h} wl(i,j,h) = gp(i,j) \qquad \forall (i,j) \;\; i > h$$

$$\sum_{(k,l)} wr(i,j,k,l) = gp(i,j) \qquad \forall (i,j) \;\; i > k \;\&\; j > l$$

- (*logical level const.*) At least one fanin from the previous level

$$wl(i,\, j,\, j-1) + \sum_{k} wr(i,j,k,j-1) \geq gp(i,j) \qquad \forall (i,j)$$

ILP on Prefix Adder – Structure

- The segment information is necessary for root constraint. The GP segments of two children must be adjacent.

Variables:

- *gpl(i,j), gpr(i,j)* int [1,n]:
  The segment covered by *gp(i,j)* is [*gpl(i,j)*:*gpr(i,j)*]

Constraints:

- (*root const.*) The GP segments of two children must be adjacent

$$gpl(i,j) = gpl(i,h) \quad if \quad wl(i,j,h) = 1 \qquad (1)$$

$$gpr(i,j) = gpr(k,l) \quad if \quad wr(i,j,k,l) = 1$$

$$gpr(i,h) = gpl(k,l) + 1 \quad if \quad wl(i,j,h) = 1 \,\&\, wr(i,j,k,l) = 1$$

Conditional constraint (1) in ILP formulation: ⇒

$$gpl(i,j) \geq gpl(i,h) - n \cdot (1 - wl(i,j,h))$$

$$gpl(i,j) \leq gpl(i,h) + n \cdot (1 - wl(i,j,h))$$

ILP on Prefix Adder – Structure

- Physical position variable attached to each GP adder describes physical level.
- No overlap in the physical view. (*overlap const.*)

Variables:

- *phy(i,j)* int [1,m]: The physical position of *gp(i,j)* is (*i*,*phy(i,j)*). (*m*: physical depth)

Constraints:

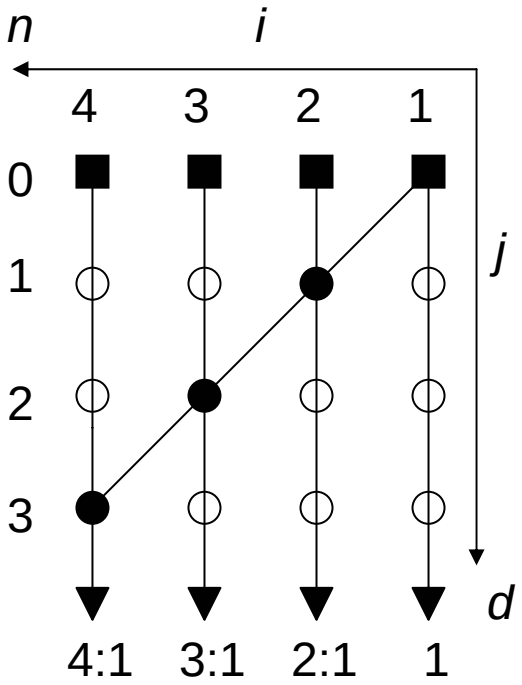
- (*overlap const.*) Each physical position contains at most one GP adder

$$phy(i, j) \neq phy(i, h) \qquad \forall i, j \neq h$$

ILP on Prefix Adder – Example

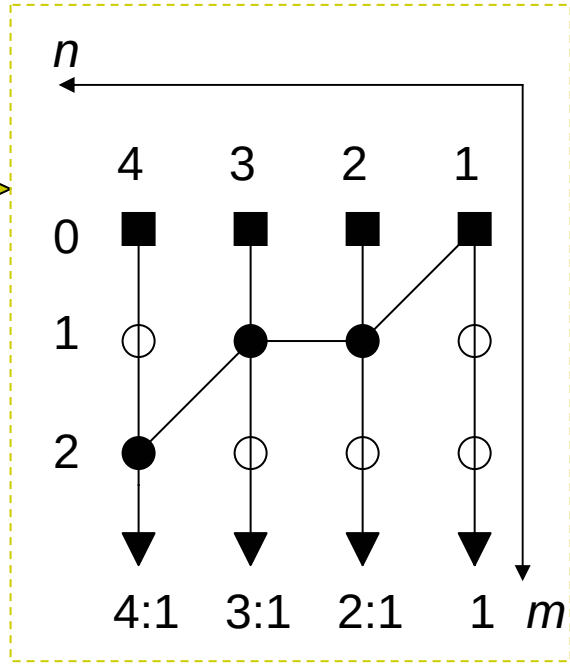
gp(2,1)=1, wl(2,1,0)=1, wr(2,1,1,0)=1
gp(3,2)=1, wl(3,2,0)=1, wr(3,2,2,1)=1
gp(4,3)=1, wl(4,3,0)=1, wr(4,3,3,2)=1

[gpl(2,1):gpr(2,1)] = [2:1]
[gpl(3,2):gpr(3,2)] = [3:1]
[gpl(4,3):gpr(4:3)] = [4:1]



Logical view

phy(2,1)=1
phy(3,2)=1
phy(4,3)=2



Physical view

ILP on Prefix Adder – Capacitance

- Gate capacitance is calculated based on logical fanouts.
  - Gate cap equals to the number of fanouts, when input cap of GP adder is 1 unit. (*gate const.*)
- Wire capacitance depends on physical placement.
  - Vertical wire cap is proportional to the max vertical height of each fanout. (*wire const.*)
  - Horizontal wire cap is proportional to the max horizontal width of each fanout. (*wire const.*)

ILP on Prefix Adder – Capacitance

Variables:

- *Cg(i,j)* float: Gate load capacitance of *(i,j)*
- *Cwv(i,j)* float: Vertical wire load capacitance of *(i,j)*
- *Cwh(i,j)* float: Horizontal wire load capacitance of *(i,j)*

Constraints:

- (*gate const.*) Gate load capacitance:

$$Cg(i,j) = \sum_{h} wl(i,h,j) + \sum_{(k,l)} wr(k,l,i,j)$$

- (*wire const.*) Wire load capacitances:

$$Cwv(i,j) \geq \lambda^{w}(phy(i,h) - phy(i,j)) \quad if \quad wl(i,h,j) = 1 \qquad (\lambda^{w} = 0.5)$$

$$Cwv(i,j) \geq \lambda^{w}(phy(k,l) - phy(i,j)) \quad if \quad wr(k,l,i,j) = 1$$

$$Cwh(i,j) \geq \lambda^{w}(k - i) \quad if \quad wr(k,l,i,j) = 1$$

ILP on Prefix Adder – Timing

- The output time is the max path delay. (*output const.*)
- Input arrival times equal to the output times of two children. (*input const.*)
- According to the timing model, gate delay is calculated based on load capacitance.

$$\text{Delay\_GP}^{l} = 1.5\ C_{\text{load}} + 2.5$$
$$\text{Delay\_GP}^{r} = 2.0\ C_{\text{load}} + 2.5$$

ILP on Prefix Adder – Timing

Variables:

- *Tl(i,j)* float: Left input arrival time of *(i,j)*
- *Tr(i,j)* float: Right input arrival time of *(i,j)*
- *T(i,j)* float [0,*Tmax*]: Output time of *(i,j)*
  (*Tmax*: Output required time.)

Constraints:

- (*input const.*) Input arrival times:

$$Tl(i,j) = T(i,h) \quad if \quad wl(i,j,h) = 1$$

$$Tr(i,j) = T(k,l) \quad if \quad wr(i,j,k,l) = 1$$

- (*output const.*) Output time:

$$T(i,j) \geq Tl(i,j) + 1.5 \cdot Cload(i,j) + 2.5$$

$$T(i,j) \geq Tr(i,j) + 2.0 \cdot Cload(i,j) + 2.5$$

$$(Cload(i,j) = Cg(i,j) + Cwv(i,j) + Cwh(i,j))$$

ILP on Prefix Adder – Power

- Total power consumption is the summation of power consumption on each GP adder.
- The objective is to minimize total power consumption.

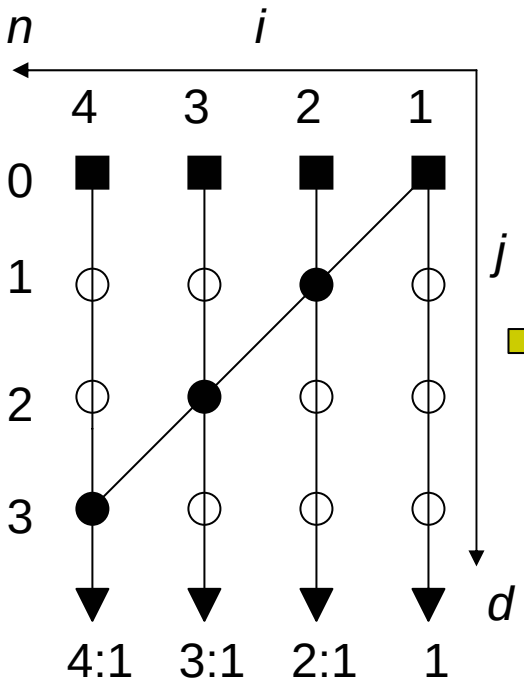
$$Minimize: \sum_{(i,j)} j \cdot Cload(i,j) + \lambda^S \cdot gp(i,j)$$

ILP on Prefix Adder – Example

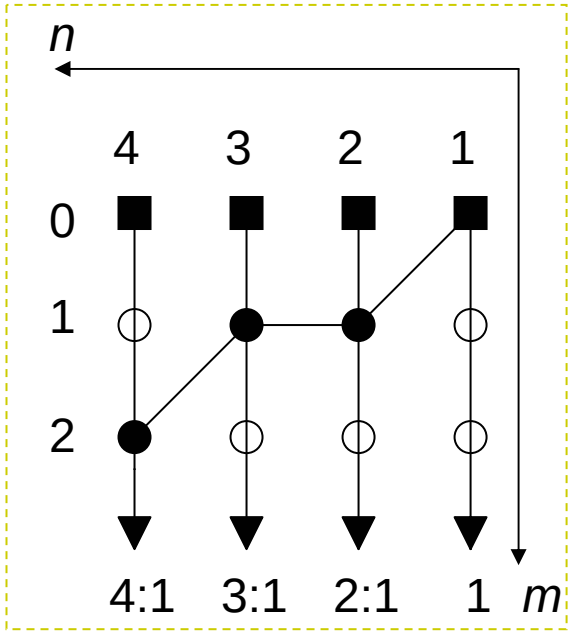
Cg(2,1)=1, Cwv(2,1)=0, Cwh(2,1)=0.5 *Cload(2,1)=1.5*

Cg(3,2)=1, Cwv(3,2)=0.5, Cwh(3,2)=0.5 *Cload(3,2)=2*

Cg(4,3)=0, Cwv(4,3)=0, Cwh(4,3)=0 *Cload(4,3)=0*



Logical view



Physical view

Tl(2,1)=0, Tr(2,1)=0,
T(2,1)=0+2×1.5+2.5=5.5

Tl(3,2)=0, Tr(3,2)=5.5,
T(3,2)=5.5+2×2+2.5=12

Tl(4,3)=0, Tr(4,3)=12,
T(4,3)=12+2×0+2.5=14.5

Power = 1×Cload(2,1) +
2×Cload(3,2) +
3×Cload(4,3) +
3×3 = 14.5

ILP on Prefix Adder – Extension

- Gate sizing and buffer insertion are two important optimization technologies to improve performance.
- Gate sizing: decrease gate delay, increase input capacitance.
- Buffer insertion: introduce new element, impact placement.
- Gate sizing and buffer insertion can be supported by ILP formulation.

Experimental Results

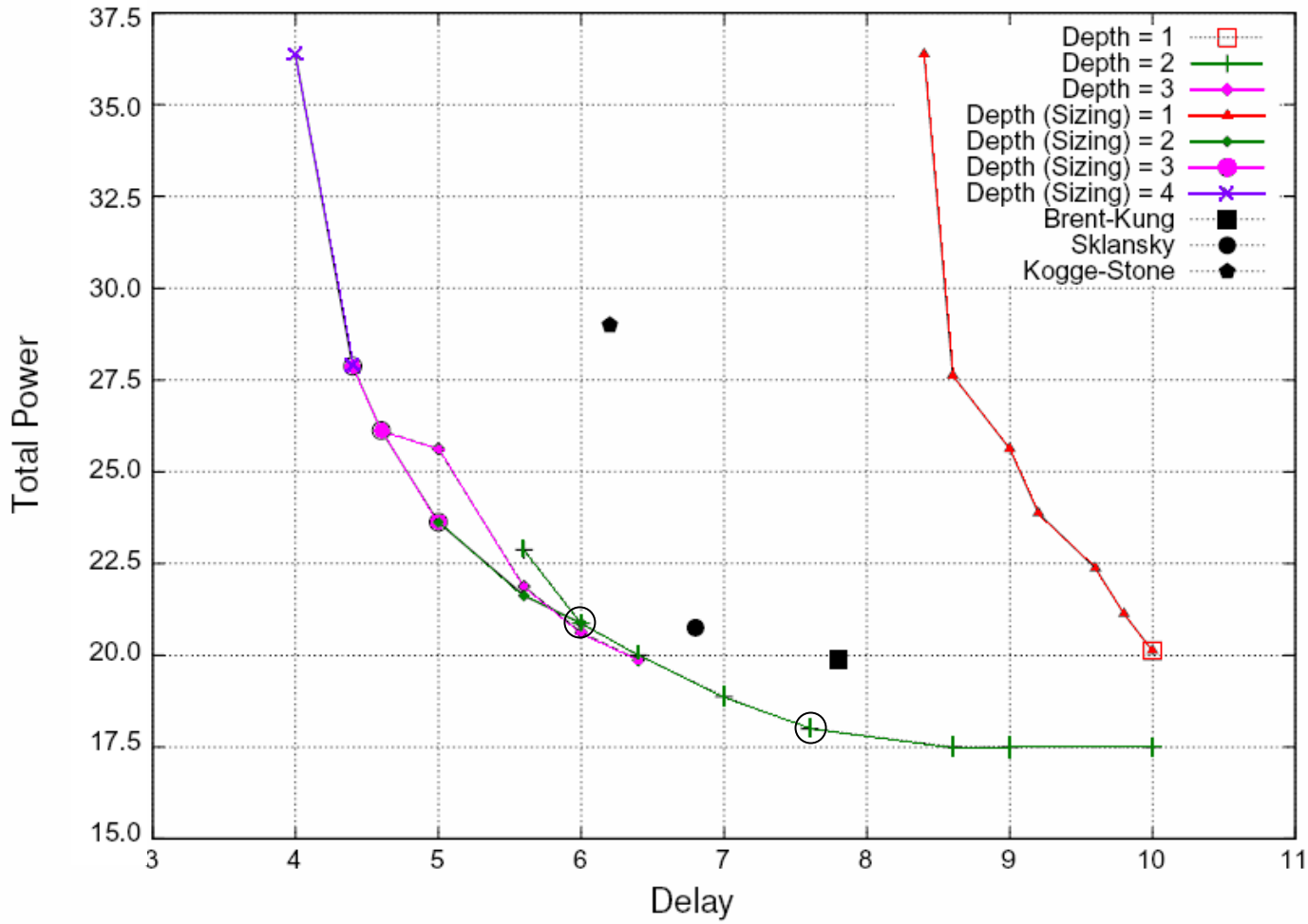
- Optimum prefix adders solved by CPLEX 9.1
- 8-bit prefix adders
  - Uniform input arrival time
  - Non-uniform input arrival time
- Hierarchical 64-bit prefix adders
- 64-bit prefix adder implementation (Synopsys flow, TSMC 90nm technology)
  - Module Compiler
  - Astro
  - Prime Power

Experimental Results – 8-bit Uniform

Method	Timing (D_{FO4})	Depth	Power (P_{FO4})	CPU (s)	Method	Timing (D_{FO4})	Depth	Power (P_{FO4})	CPU (s)
ILP	10.0	1	20.1	0.31	*K-S*	*6.2*	*3*	*29.0*	-
ILP	10.0	2	17.5	124	ILP	6.0	2	20.9	259
ILP (S)	9.0	1	25.6	2.83	ILP	5.6	2	22.9	45.7
ILP	9.0	2	17.5	83.4	ILP (S)	5.6	2	21.6	756
ILP (S)	8.6	1	27.6	1.28	ILP	5.6	3	21.9	1237
ILP	8.6	2	17.5	93.2	ILP (S)	5.0	2	23.6	1208
B-K	*7.8*	*3*	*19.9*	-	ILP	5.0	3	25.6	4563
ILP	7.6	2	18.0	112	ILP	4.6	3	26.1	7439
ILP	7.0	2	18.6	99.6	ILP (S)	4.2	3	27.9	9654
Skl	*6.8*	*3*	*20.8*	-	ILP (S)	4.0	4	36.4	20211

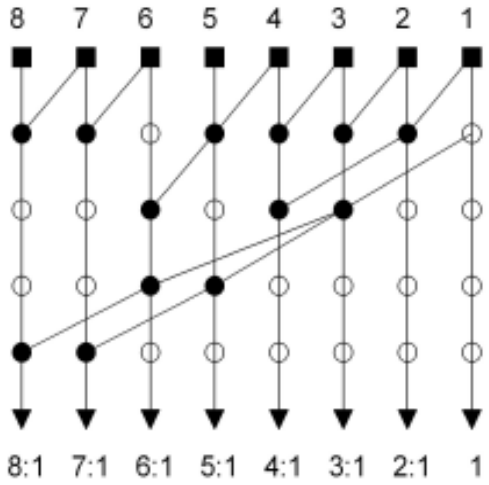
(S): Gate sizing, B-K: Brent-Kung, Skl: Sklansky, K-S: Kogge-Stone

Experimental Results– 8-bit Uniform

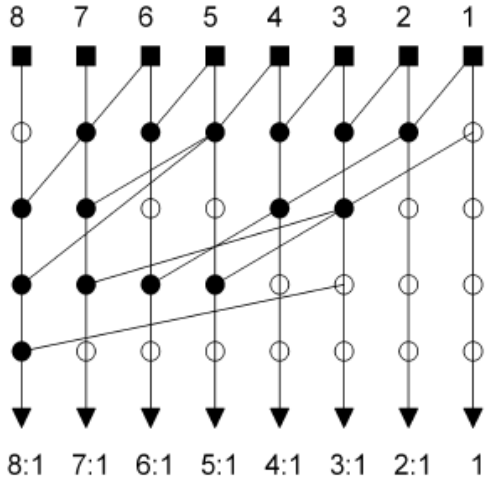


Experimental Results – 8-bit Uniform

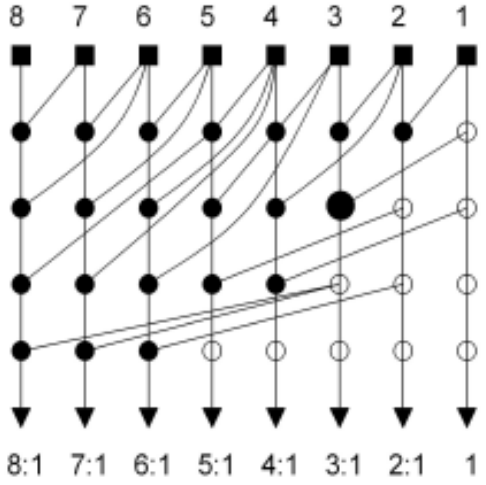
- Some typical ILP results:



Fastest depth2



Fastest depth3

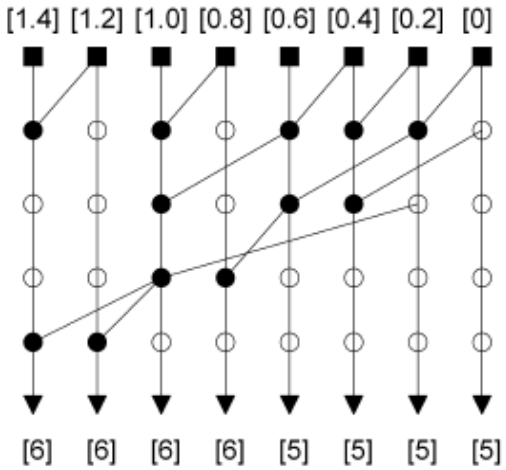


Fastest depth4

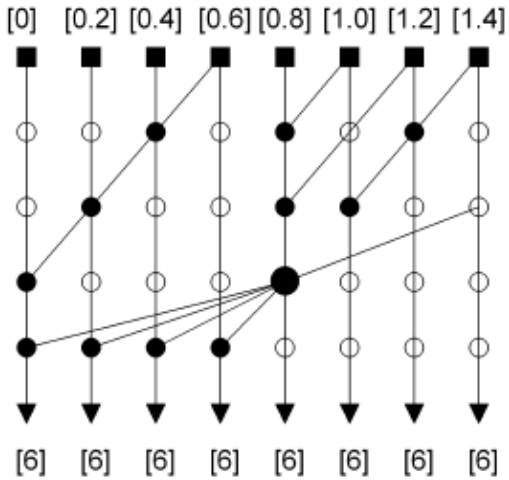
All the 8-bit fastest prefix adders have 4 logical levels

Experimental Results – 8-bit Non-Uniform

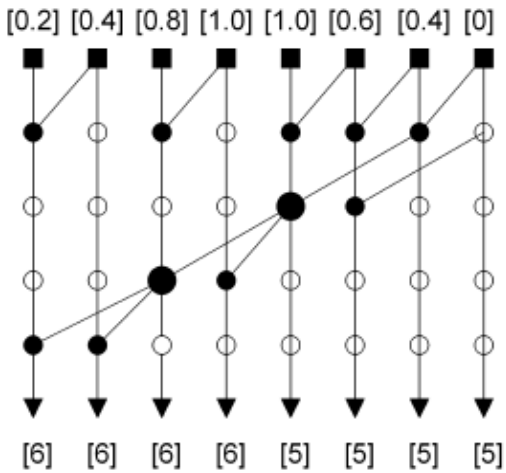
Case	Power	Depth	Power*	Depth*
Increasing arrival times	20.8	3	26.1	3
Decreasing arrival times	25.1	3	26.1	3
Convex arrival times	21.6	2	23.6	3



Increasing arrival times



Decreasing arrival times

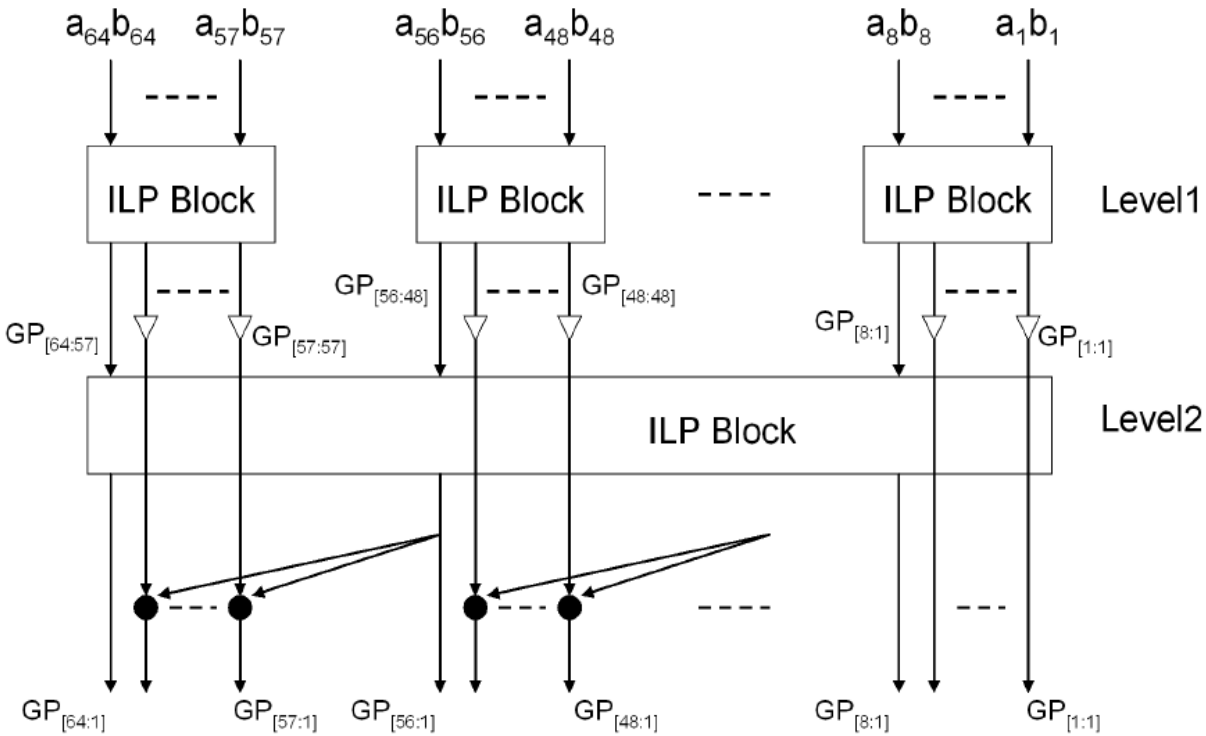


Convex arrival times

**** Use the worst input arrival time for all inputs***

Experimental Results – 64-bit Hierarchical

- For high bit-width application, ILP method can be applied in a hierarchical design strategy.

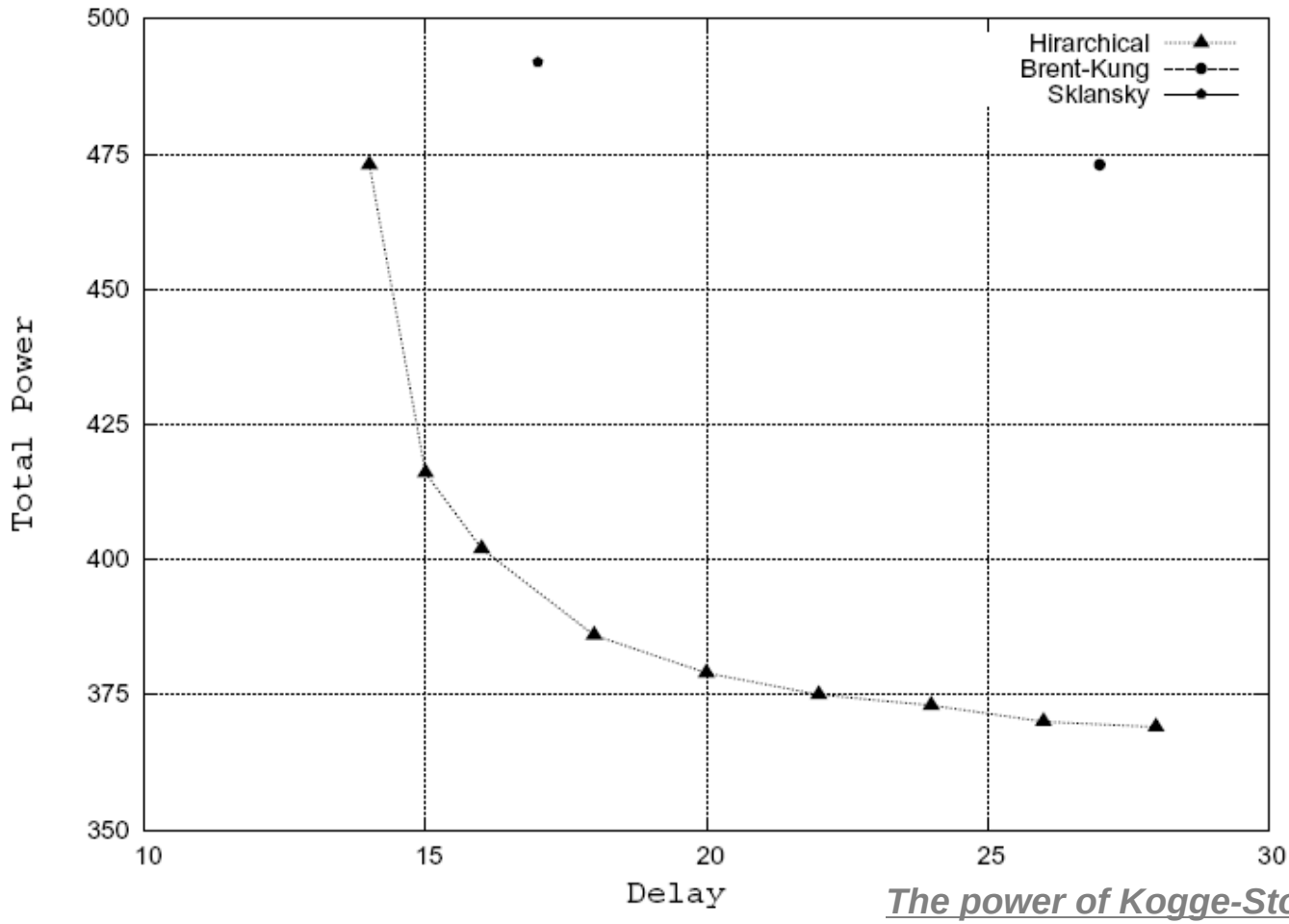


Experimental Results – 64-bit Hierarchical

- Hierarchical ILP designs in solution space: (The physical depth is set to 6)

Method	Timing (D_{FO4})	Power (P_{FO4})	Method	Timing (D_{FO4})	Power (P_{FO4})
Hierarchical ILP	28	369	Hierarchical ILP	18	386
Brent-Kung	27	473	Sklansky	17	492
Hierarchical ILP	26	370	Hierarchical ILP	16	402
Hierarchical ILP	24	373	Hierarchical ILP	15	416
Hierarchical ILP	22	375	Kogge-Stone	15	3032
Hierarchical ILP	20	379	Hierarchical ILP	14	473

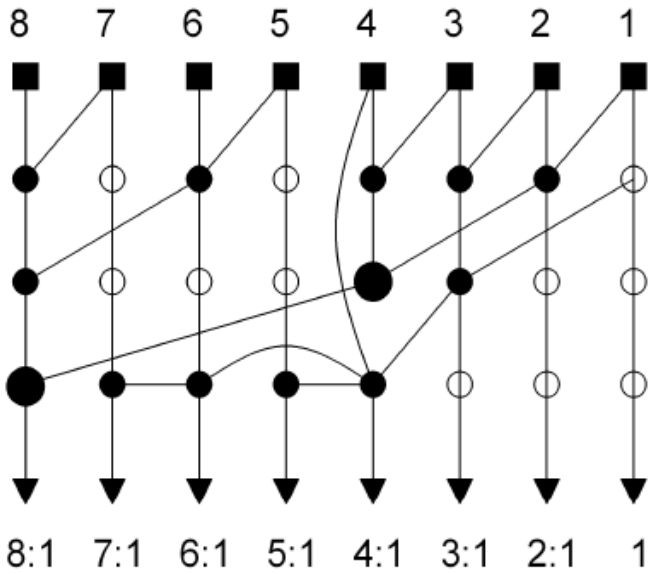
Experimental Results – 64-bit Hierarchical



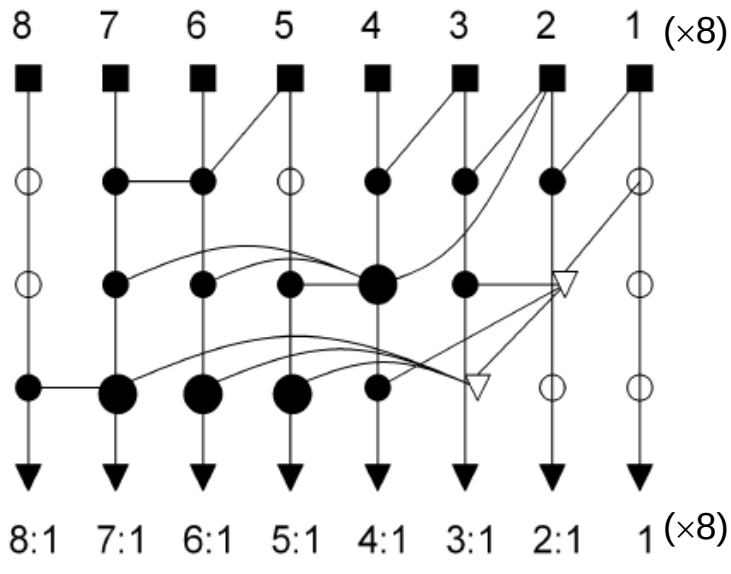
The power of Kogge-Stone add is much larger than other prefix adders.

Experimental Results – 64-bit Hierarchical

- The fastest 64-bit hierarchical ILP adder:



Hierarchical ILP – level 1 (Physical view)



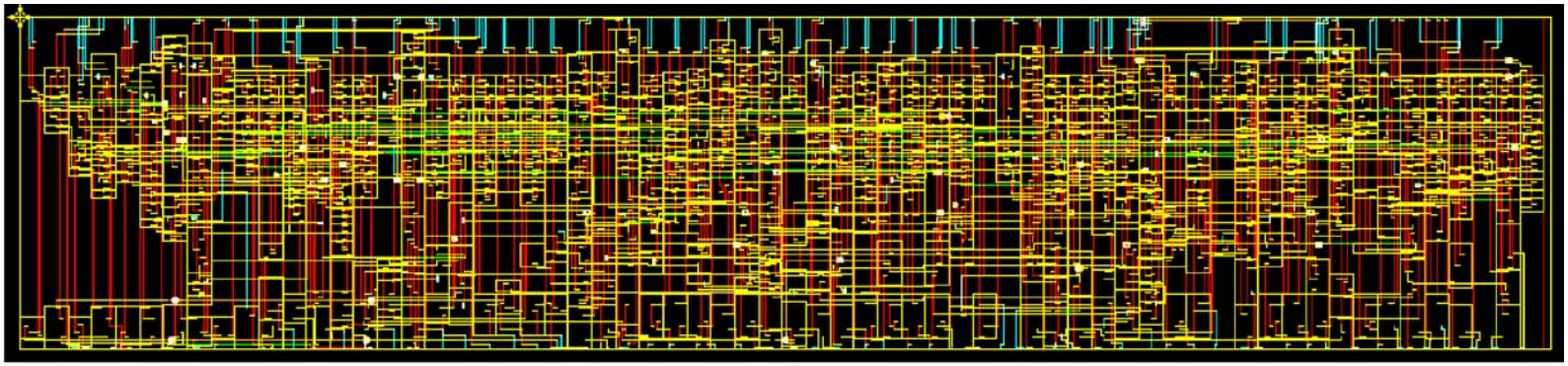
Hierarchical ILP – level 2 (Physical view)

Experimental Results – 64-bit Implementation

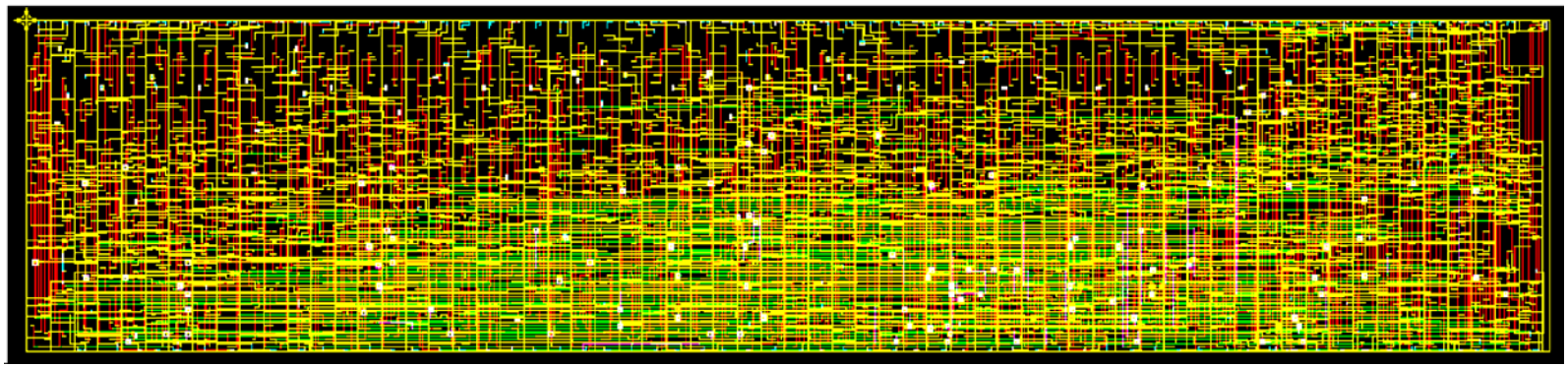
- 64-bit ILP prefix adders compared with 64-bit fast prefix adders generated by Module Compiler with relative placement.

ILP		Module Compiler		Power Saving [Wire] (%)
Timing (ns)	Total Power [Wire Power] (mW)	Timing (ns)	Total Power [Wire Power] (mW)	
0.74	1.9 [0.93]	0.75	4.9 [2.8]	61% [67%]
0.76	1.8 [0.90]	0.83	3.5 [2.1]	49% [57%]
1.13	1.15 [0.65]	1.24	2.3 [1.5]	50% [57%]

Experimental Results – 64-bit Implementation



64-bit ILP Prefix Adder Physical View



64-bit MC Prefix Adder Physical View

Conclusions

- We propose an ILP method to solve minimal power prefix adders.
- The comprehensive area/timing/power model involves physical placement, gate/wire capacitance and static/dynamic power consumption.
- The ILP method can handle gate sizing, buffer insertion for both uniform and non-uniform input arrival time applications.
- The ILP method can be applied in hierarchical design methodology for high bit-width applications.


Thank You