Resizing Gates in Series-Parallel Graphs and Trees to Minimize Power Consumption *

Edward Y.C. Cheng and Sartaj Sahni
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611-6120
{yccheng, sahni}@cise.ufl.edu

August 2, 2000

Abstract

We study the problem of resizing gates so as to reduce overall power consumption while satisfying a circuit's timing constraints. Polynomial time algorithms for series-parallel and tree circuits are obtained.

Keywords and Phrases: power consumption, timing constraints, gate resizing, series-parallel graphs, trees.

1 Introduction

Power consumption, speed and area are three important and related characteristics of a circuit. With the increase in circuit density and the enhanced use of battery operated devices, the emphasis on power consumption has increased. By reducing power consumption, we simultaneously reduce heat dissipation and increase battery life.

In this paper we consider the problem of minimizing the power consumed by a circuit subject to satisfying the circuit's timing constraints. Power reduction is obtained by gate resizing – larger gates are replaced by smaller ones that have higher delay but lower power consumption. Power reduction via gate resizing has been considered in [4], for example.

In the general gate resizing problem (GGR), for each gate in the circuit we have a list of (delay, capacitance) pairs. Each pair gives the delay and capacitance associated with a possible implementation of that gate. Dynamic power is the dominant component of the power consumed by a CMOS gate [4, 9, 6]. The dynamic power consumed by a CMOS gate is linearly proportional

*This research was supported, in part, by the Army Research Office under grant DAA H04-95-1-0111.

to the product of the capacitive load at the gate output and the output switching activity of the gate. The dynamic power consumed by a circuit is the sum of the dynamic power consumed by the gates in the circuit. Consequently, when a gate v is resized, the change in the dynamic power consumed by the circuit is given by

$$c * \Delta C(v) \sum_{u \text{ is an input of } v} E(u)$$

where c is a constant, $\Delta C(v)$ is the change in the capacitance of gate v, and $E(u)$ is the switching activity at u. As in [5], we assume that the transition probabilities for the primary inputs of the circuit are known. From these transition probabilities, the switching activity at each gate output can be computed using the symbolic simulation methods of [5]. Since resizing one or more gates does not affect the switching activity at gate outputs/inputs, the change in the dynamic power consumption of a circuit due to resizing a gate v may be computed from the known and nonvarying switching activity at the gate inputs and the change in the gate's capacitance. Therefore, we may assume that for each gate we have (delay, power consumption) pairs instead of (delay, capacitance) pairs. Each (delay, power consumption) pair characterizes an implementation of the gate. In practice, this means that before using the algorithms developed in this paper, we will need to transorm the given (delay, capacitance) pairs (which are the inputs to the algorithms of [4]) for each circuit gate into equivalent (delay, power consumption) pairs by multiplying the gate's capacitance with the sum of the switching activities at the gate's inputs and some constant. This transformation is circuit dependent.

In the GGR problem, we begin with a realization for each gate (i.e., a selection of a (delay, power consumption) pair) such that the timing constraints are satisfied. We wish to change the realization of some or all of the gates by replacing their assigned pair with one that has larger delay (i.e., gate resizing) and such that the timing constraints remain satisfied and the power consumption of the resized circuit (this is the sum of the power consumption at each gate) is minimum. The GGR problem (referred to as the incomplete library problem in [4]) is equivalent to the BCI problem studied in [1]. The BCI problem was shown to be NP-Complete in [1], even for circuits that were simply a chain of single input single output gates. Bahar et al. [9] propose a greedy heuristic for the GGR problem on general circuits. This heuristic resizes one gate at a time and accounts for false paths in its timing analysis. Chen and Sarrafzadeh [4] have proposed a heuristic that resizes several gates at a time. The heuristic of [4], however, does not account for false paths. Chen and Sarrafzadeh also propose a pseudo polynomial time algorithm for the Low Power Complete Library-Specific Gate Resizing (CGR) problem. In this problem, each gate can be realized to have any delay (delays are assumed to be integral). Further, the power consumed by gate v decreases by the constant $c(v)$ for each unit increase in delay. Let $d_l(v)$ by the delay of the initially assigned realization of gate v and let $d(v) \geq d_l(v)$ be the delay of the resized gate v. Then the power

reduction ΔP resulting from resizing an n gate circuit is:

$$\Delta P = \sum_{i=1}^{n} c(i)(d(i) - d_l(i))$$

In this paper, our focus is the development of efficient polynomial time algorithms for gate resizing problems restricted to series-parallel and tree circuits. In Section 2.2 we develop a linear algorithm for the CGR problem for series-parallel circuits. In Section 2.3 we extend this linear algorithm for series-parallel graphs to obtain an $O(n \log^2 n)$ time algorithm that works when there is an upper bound on the delay of each gate. That is, each gate v has realizations with integral delays in the range $[d_l(v), d_u(v)]$. As in the CGR problem, each unit increase in delay reduces power consumption by $c(v)$. We call this the CUGR problem. The CUGR problem for tree circuits can also be solved in linear time (Section 3).

Cheng and Sahni [7] have shown that for general circuits, the CGR problem with multigate modules is NP-hard. They also present a uniform framework for the solution of the CGR, CUGR, and ConvexCGR problems for general circuits, and a heuristic for the GGR problem for general circuits.

Throughout this paper we assume that a circuit is represented as a directed acyclic graph. The vertices of this graph represent gates and the edges represent signal flow. Primary inputs may be modeled as vertices with no incoming edge and primary outputs may be modeled as vertices with no outgoing edge.

Figure 1 gives the digraph for an example circuit. The vertices corresponding to primary inputs are labeled with the time at which the primary input is available; vertices corresponding to primary outputs are labeled with the time by which the output signal must arrive; and the remaining vertices (these correspond to circuit gates) are labeled with the (delay, power consumption) pair corresponding to their initial implementation.

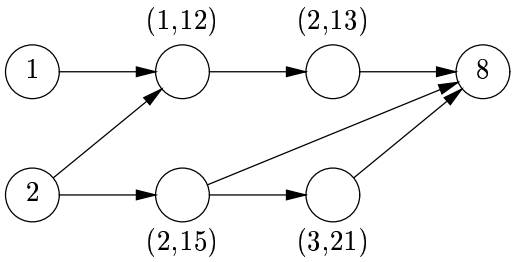


Figure 1: Digraph corresponds to a circuit

2 Series-Parallel Circuits

2.1 Definition

Series-parallel circuits were considered in [1]. A series-parallel circuit may be defined recursively as below [1]:

SP1: a chain of gates is a series-parallel circuit (Figure 2(a)).

SP2: several chains of gates joined at the ends to a common first gate and a common last gate (Figure 2(b)) define a simple parallel circuit. A simple parallel circuit is a series parallel circuit.

SP3: a circuit obtained from a series-parallel circuit C by replacing any interconnect of C by another series-parallel circuit is a series-parallel circuit (Figure 2(c)).

Figure 2 gives example series-parallel circuits as well as a circuit that is not series-parallel.

2.2 Complete Library Gate Resizing (CGR)

Our strategy to solve the CGR problem for series-parallel circuits is to reduce the circuit to one that has a single gate. The CGR problem for the reduced single gate circuit is easily solved, and finally the solution to this single gate problem is used to reconstruct the solution for the initial circuit.

To transform an arbitrary series-parallel circuit into an equivalent single gate circuit (i.e., a single gate circuit with the same maximum power reduction), we first obtain the series parallel decomposition of the circuit using the linear time algorithm of [2]. This series-parallel decomposition essentially tells us how to build the original circuit using chains (SP1), simple parallel circuits (SP2), and replacing interconnects of C by series-parallel circuits (SP3). During this rebuild process, we shall replace each chain and simple parallel circuit by a single gate. Consequently, when the rebuild is complete, we will be left with a single gate. The replacement rules for chains and simple parallel circuits are given below:

Chain Suppose the chain has n gates labeled with delays $d_1, d_2, \ldots, d_n$. Let c_i be the power reduction obtained by increasing d_i by 1. The signal delay through the chain is $\sum_{i=1}^{n} d_i$. For each unit increase in signal delay over $\sum_{i=1}^{n} d_i$, the maximum possible power reduction is $\max_{1 \leq i \leq n}\{c_i\}$. Therefore the chain is equivalent to a gate v with delay $\sum_{i=1}^{n} d_i$ and $c(v) = \max_{1 \leq i \leq n}\{c_i\}$ (Figure 3(a)). The power reduction ΔP obtained by making this replacement is zero.

Simple Parallel Circuit First transform each chain in the simple parallel circuit into an equivalent single gate using the transformation of Figure 3(a). This results in the parallel circuit

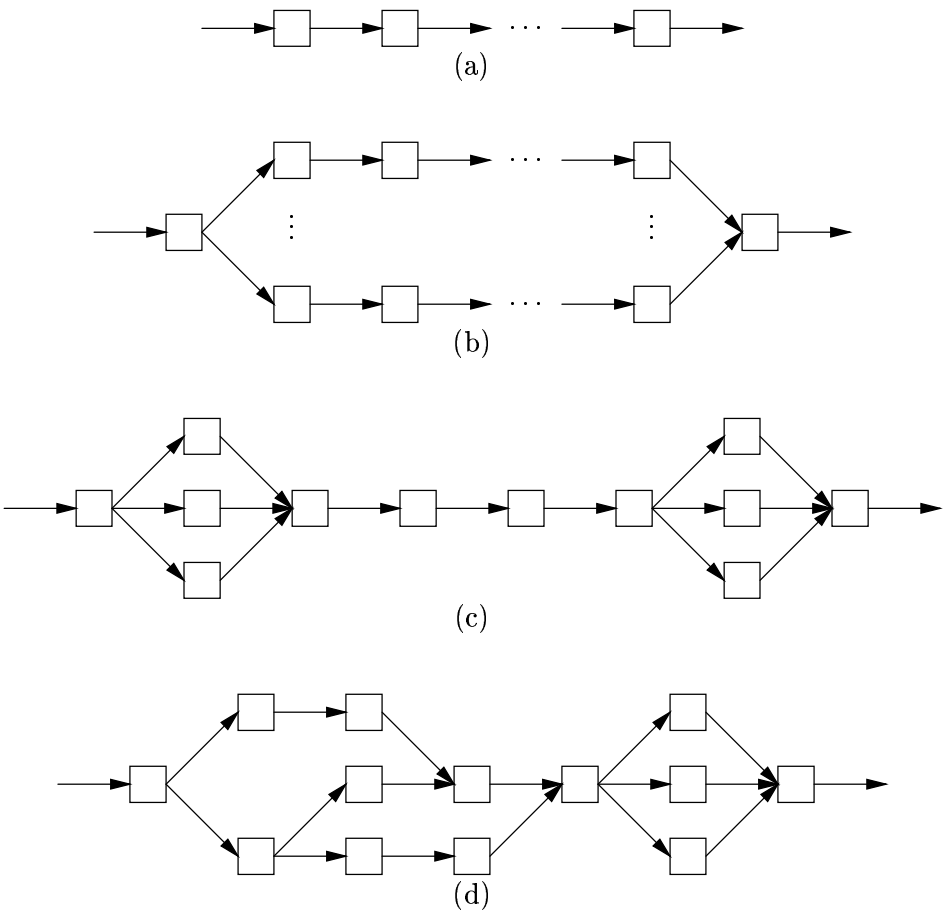


Figure 2: Circuit Examples (Source : [1]). (a) Chain; (b) Simple Parallel Circuit; (c) Series-Parallel Circuit; (d) Non-Series-Parallel Circuit

of Figure 3(b). The signal delay between the output of gate s and the input of gate t is $\max_{1\leq j\leq n}\{d_j\}$. We can increase the delays of all gates between s and t to $\max_{1\leq j\leq n}\{d_j\}$ without affecting the overall circuit delay. This gives us a power reduction $\Delta P = \sum_{i=1}^{n} c_i(\max_{1\leq j\leq n}\{d_j\} - d_i)$. Each unit increase in delay between s and t beyond $\max_{1\leq j\leq n}\{d_j\}$ gives us a power reduction of $\sum_{i=1}^{n} c_i$. Therefore the n gates between s and t are equivalent to a single gate with delay $\max_{1\leq j\leq n}\{d_j\}$ and $c(v) = \sum_{i=1}^{n} c_i$. Consequently, a simple parallel circuit may be replaced by the three gate chain shown in Figure 3(b). This chain can, in turn, be replaced by a single gate using the chain transformation of Figure 3(a).

Using the above transformations on the series parallel decomposition yields a single gate circuit. The input to this gate is the primary input of the original circuit and the gate output is the primary output of the original circuit. Let v be the single gate that remains. Let t_i and t_o be the arrival time of the input and the required time for the output respectively. Since we start with a circuit that can

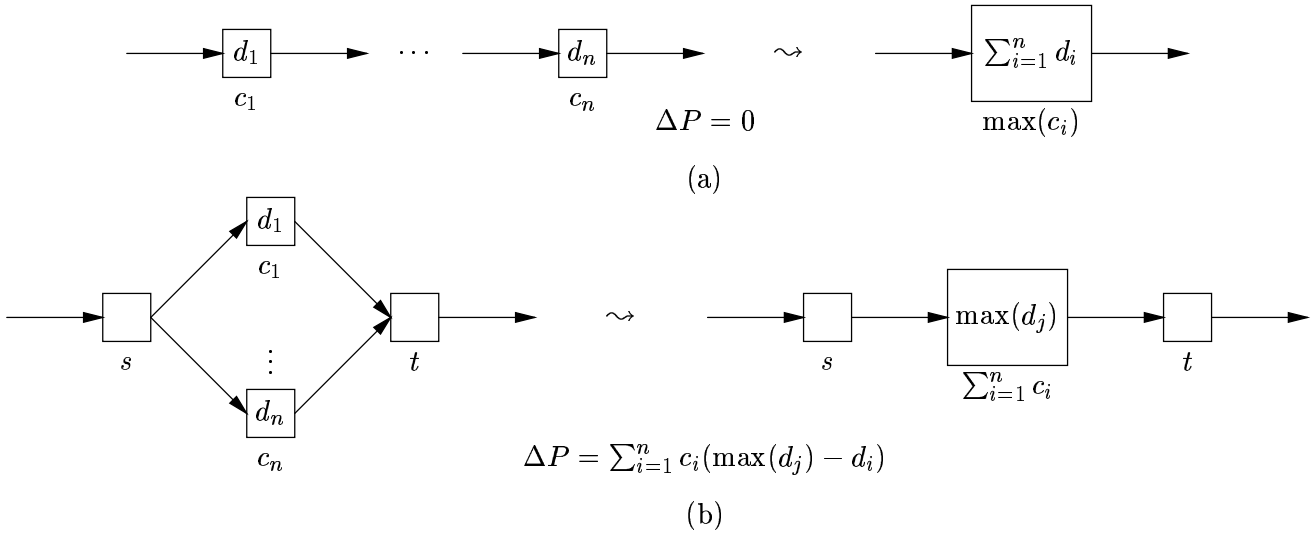


Figure 3: Transformation of Series-Parallel Circuits. (a) Chain; (b) Simple Parallel Circuit

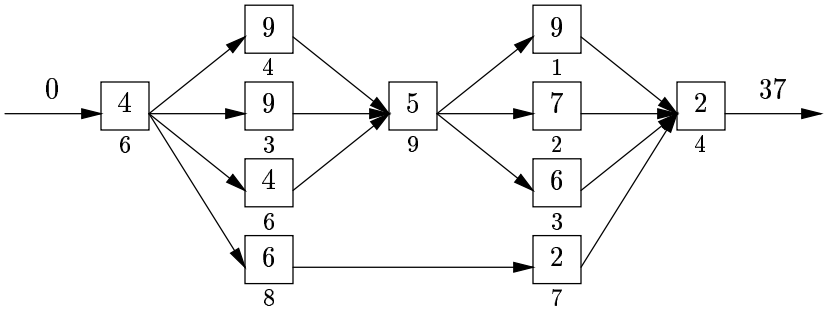
meet its arrival time requirements (i.e., a feasible circuit) and since the transformations of Figure 3 do not affect feasibility, $t_i + d(v) \leq t_o$. The additional power reduction possible is $(t_o - t_i - d(v))c(v)$ The maximum power reduction $\Delta P_{\max}$ for the original circuit is $(t_o - t_i - d(v))c(v)$ + sum of the ΔPs from the simple parallel circuit transformation (Figure 3(b)).

To obtain the delay values for each gate of the original circuit that will result in a power reduction of $\Delta P_{\max}$, simply follow the reduction process backwards. The total time taken is linear in the number of gates in the original circuit. Figures 4 and 5 show an example. Each gate is represented by a box, the number inside a box is the gate delay, the number below a box is the gate's c value, the primary input is available at time 0, and the primary output is needed at time 37.

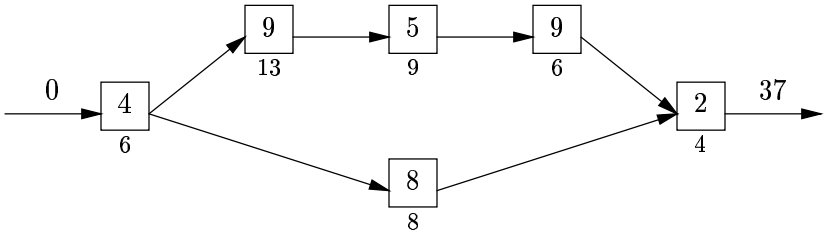
2.3 Complete Library with Upper Bounds (CUGR) and Convex $c(v)$s

The power reduction per unit delay increase function $c(v)$ for gate v is convex iff there exist positive $\delta_1, \delta_2, \ldots, \delta_m$ and $c_1 \geq c_2 \geq \cdots \geq c_m$ such that $c(v) = c_1$ for delay increases between 0 and δ_1; $c(v) = c_2$ for delay increase between $\delta_1 + 1$ and $\delta_1 + \delta_2$; $c(v) = c_3$ for delay increases between $\delta_1 + \delta_2 + 1$ and $\delta_1 + \delta_2 + \delta_3$; and so on. Figure 6 shows the power consumption as a function of the increase in delay relative to the gate's initial delay $d_l(v)$. P_0 is the power consumption when the gate has its initial delay $d_l(v)$.

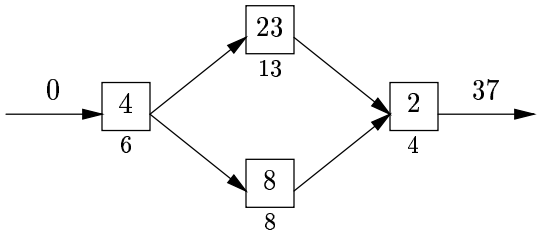
The CUGR problem can be modeled using gates with convex power reduction functions $c(v)$. For example if gate v provides a power reduction of c for each unit increase in delay between $d_l(v)$ and $d_u(v)$ then we may use $c(v)$ with $\delta_1 = d_u(v) - d_l(v)$ and $c_1 = c$. Because of this correspondence



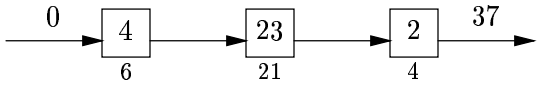
(a) $\Delta P_1 = 0$



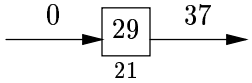
(b) $\Delta P_2 = 6 * 5 + 2 * 2 + 3 * 3 = 43$



(c) $\Delta P_3 = 0$



(d) $\Delta P_4 = 8 * 15 = 120$



$\Delta d = 37 - 29 = 8$

(e) $\Delta P_5 = 21 * \Delta d = 168$

Figure 4: Transformation of a Series-Parallel Circuit into a single gate.

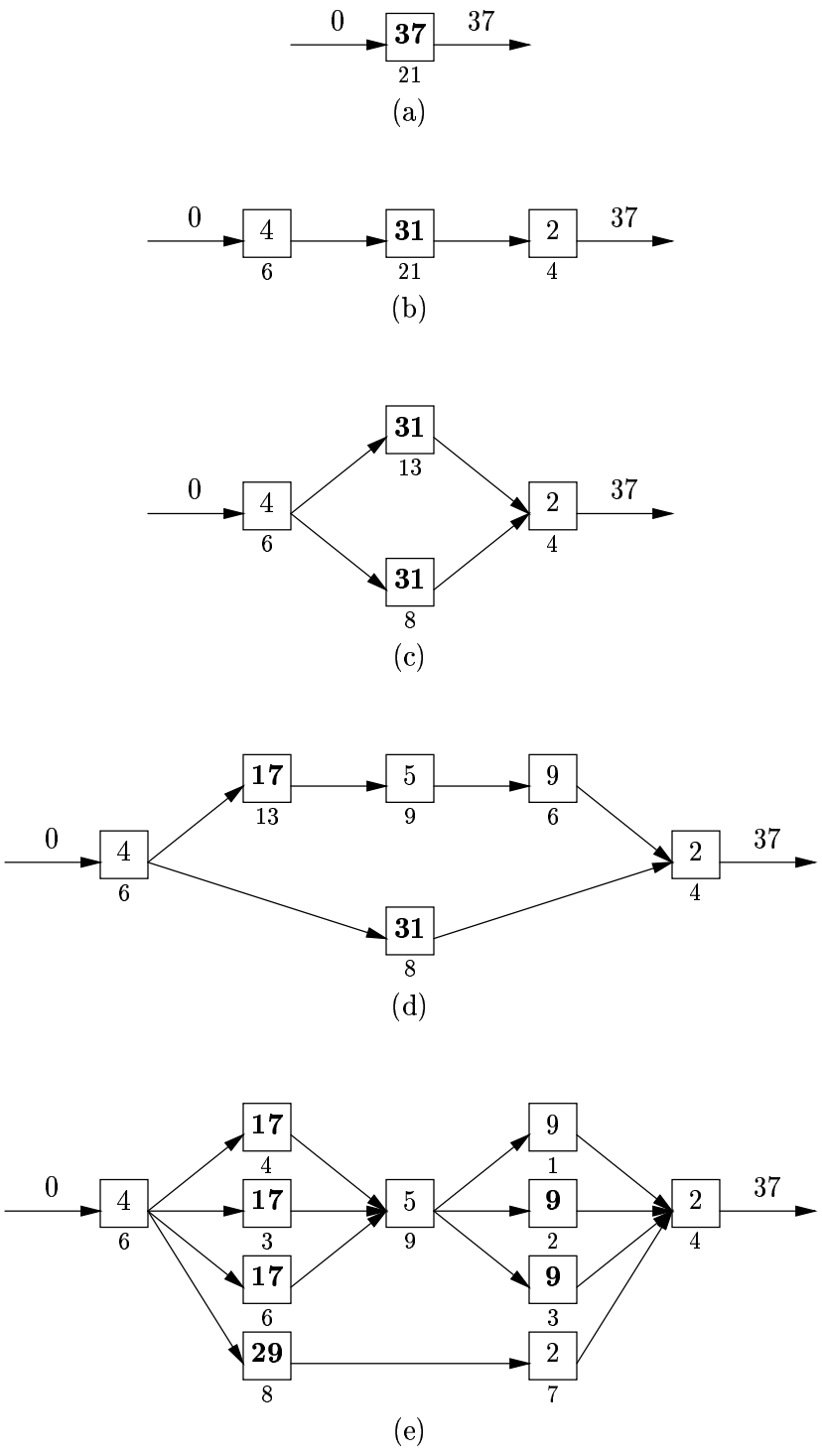


Figure 5: Computation of new delay for each gate.

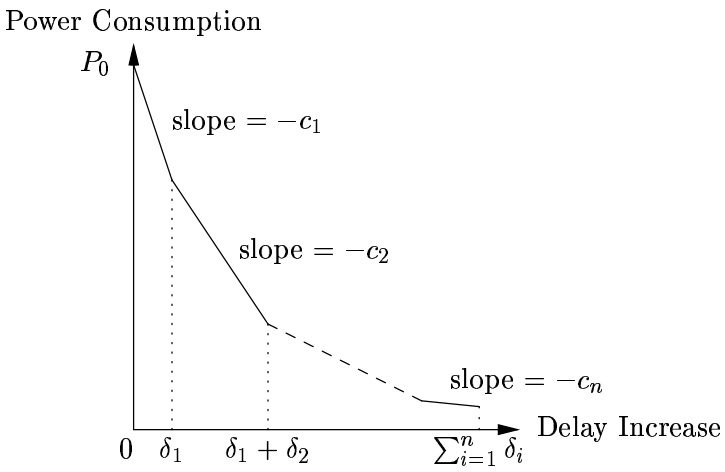


Figure 6: Convex delay–power-consumption graph

between the CUGR problem and the convex gate resizing problem ConvexCGR, we consider only the ConvexCGR problem in this section.

To solve the ConvexCGR problem for series-parallel circuits, we need only develop methods to transform a chain of convex gates into an equivalent convex gate and to transform a simple parallel circuit comprised of convex gates into an equivalent convex gate. These transformations can then be used in place of the transformations of Section 2.2 to obtain an algorithm for the ConvexCGR (and hence also for the CUGR) problem. We assume that a convex gate is given by a list of tuples $\{(\delta_1, c_1), (\delta_2, c_2), \ldots, (\delta_m, c_m)\}$, $c_1 > c_2 > \cdots > c_m$. This list is called the DP (delay-power) list of the gate.

Chain of Convex Gates A chain of n convex gates with initial delays $d_1, d_2, \ldots, d_n$ is replaced by a convex gate with initial delay $\sum_{i=1}^{n} d_i$ as in Figure 3(a). The DP list for the new gate is obtained by merging the DP lists of the n gates in the original chain into a single list sorted by non-increasing c_is. During this process, pairs with the same c_i value are combined into a single pair. For example if $(5, 24)$ and $(2, 24)$ are pairs in DP_1 and DP_2 , respectively, the combined pair is $(7, 24)$. Suppose we have a 3 gate chain with $DP_1 = \{(3, 28), (5, 24), (3, 21)\}$, $DP_2 = \{(2, 24), (4, 23)\}$ and $DP_3 = \{(9, 26)\}$. The DP list for the replacement gate for this chain is $\{(3, 28), (9, 26), (7, 24), (4, 23), (3, 21)\}$.

Simple Parallel Circuit with Convex Gates First the chains in the circuit are transformed into equivalent single convex gates. Then the delays of these equivalent convex gates are increased to $\max_{1 \le j \le n}\{d_j\}$ (see Figure 3(b)). This increase in delay provides us a power reduction ΔP and changes the tuples at the front of the DP lists of the gates whose delay is increased. Let the new DP lists be $DP'_1, DP'_2, \ldots, DP'_n$. Now the gates between s and t are

replaced by a gate with delay $\max_{1 \le j \le n}\{d_j\}$. The DP list of the gate may be obtained from the DP_i' lists. Figure 7 shows the process when $n = 2$. Here L_1 and L_2 denote the two DP lists DP_1' and DP_2' and L denotes the DP list of the replacement convex gate. Finally, s, t and the replacement gate are combined into a single convex gate using the method for a chain of convex gates. Suppose we have two parallel convex gates with delays 3 and 2 respectively and the corresponding DP lists $DP_1 = \{(3, 28), (5, 24), (3, 21)\}$ and $DP_2 = \{(2, 24), (4, 23)\}$. The delay of the equivalent convex gate is thus 3, which reduces the power consumption of the second gate by 24. The DP list of the second gate is modified to $\{(1, 24), (4, 23)\}$. By performing the *Parallel_Merge* operation we obtain the DP list $\{(1, 52), (2, 51), (2, 47), (3, 24), (3, 21)\}$ for the equivalent gate.

Algorithm *Parallel_Merge*(L_1, L_2)

```
/* Merge L1 and L2, consider two gates in parallel */

p1 ← head(L1);
p2 ← head(L2);
L ← NULL;
while L1 not empty and L2 not empty do
  if δ(p1) < δ(p2) then
    Insert (δ(p1), c(p1) + c(p2)) into L;
    δ(p2) ← δ(p2) − δ(p1);
    p1 ← next(p1);
  else if δ(p1) > δ(p2) then
    /* Similar to the "if" part above, interchange p1 and p2. */
  else /* δ(p1) = δ(p2) */
    Insert (δ(p1), c(p1) + c(p2)) into L;
    p1 ← next(p1);
    p2 ← next(p2);
  end if
end while
if L1 is empty then
  Append the remaining nodes of L2 starting from p2 to L;
else
  Append the remaining nodes of L1 starting from p1 to L;
end if
return L;
```

Figure 7: Algorithm *Parallel_Merge*

2.4 Time Complexity of Convex GR Problem

A straightforward implementation of our algorithm of the previous section for the ConvexCGR problem uses sorted chains (linked lists) to represent the DP lists. The time needed to combine/merge the DP lists L_1 and L_2 of two gates is $O(|L_1| + |L_2|)$ regardless of whether we do a series or parallel merge. If we start with gates having DP lists with k tuples each, then the time needed to transform an n gate series-parallel circuit into its equivalent single gate is $O(kn^2)$. To see this, observe that each series/parallel combine step reduces the number of gates by at least 1. Therefore, there can be at most $n-1$ combine steps. Further, after q combines, the size of a DP list is $O(kq) = O(kn)$. So the cost of $O(n)$ combines is $O(kn^2)$. An example circuit that exhibits kn^2 worst-case behavior is given in Figure 8.

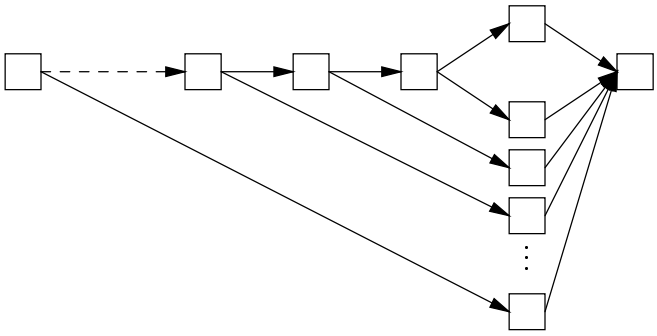


Figure 8: Worst-case merging of n gates

We can reduce the asymptotic time complexity to $O(kn \log^2 n)$ by using balanced binary search trees (BBSTs) [16] to represent the DP lists. Each DP list is represented by a BBST such that the external nodes represent the pairs (δ_i, c_i) in the DP list in right to left order (i.e., in decreasing order of power reduction). Each internal node x contains a triple of the form $(D(x), C(x), M(x))$, where $D(x)$ is the sum of the delays of the DP list pairs in the left subtree of x, $C(x)$ is a corrective factor needed to compute the c_i values of pairs in the left subtree of x, and $M(x)$ is a pointer to the rightmost external node in the left subtree of x. Each external node y stores a pair $(d(y), c(y))$ such that $d(y)$ is the δ value of the DP list pair represented by node y; the c value of this DP list pair is

$$c(y) + \sum_{\{x\,:\,y \text{ is in the left subtree of } x\}} C(x)$$

Figure 9 shows a possible BBST for the DP list $\{(3,28),(5,24),(3,21)\}$. The leftmost external node contains the pair (3,13) which represents the DP list pair (3,28). The correct c value for the DP list pair is obtained by adding to 13 the C values in the ancestors of the external node.

To insert a new DP list pair into our BBST, we must be able to trace a path from the root to

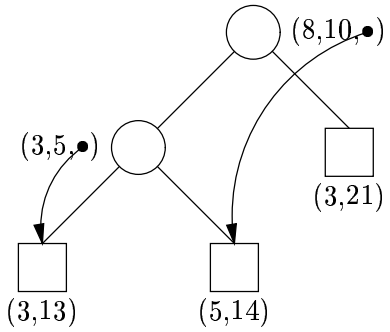


Figure 9: BBST used to represent DP list $\{(3,28),(5,24),(3,21)\}$

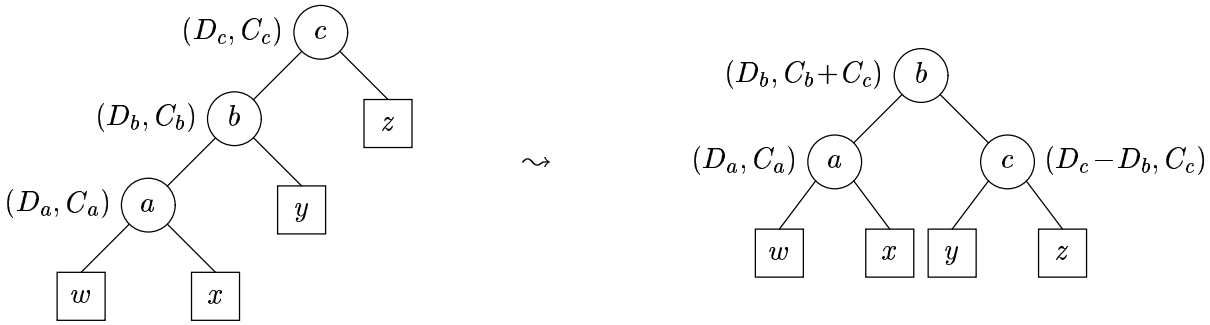
an appropriate external node. This path tracing is facilitated by the pointer $M(x)$ in internal node x. By using the $c()$ value in the external node $M(x)$ and the C values in the nodes from the root to x, we can compute the maximum c value of any DP pair in the left subtree of x.

Since insertions require rotations, we show how the $D()$ and $C()$ values in internal nodes are to be changed when rebalancing rotations are done (Figure 10). Note that $M()$ values remain unchanged during tree rotations, and are omitted from the figure. The tuple $(D(x), C(x))$ of each internal node x is shown next to the node.

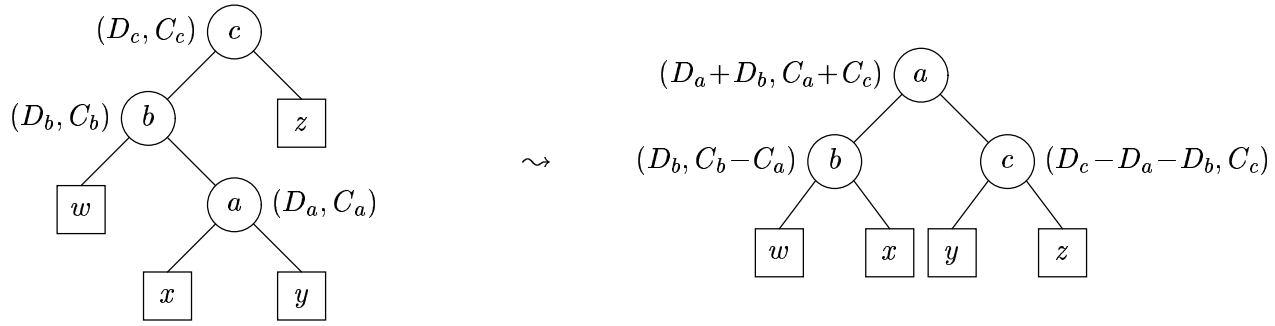
To merge the DP lists of two gates in a chain, we first perform an inorder traversal of the smaller DP list's BBST to extract the DP list's pairs. Then, these pairs are inserted into the BBST for the larger DP list. During this insertion, pairs with the same c value are combined into a single pair. If the two DP lists are L_1 and L_2, the time needed to do the series merge is $|L_1|\log(|L_1|+|L_2|)$, where L_1 is the smaller DP list.

For a parallel merge of two DP lists L_1 and L_2 (L_1 is the smaller list), we need to identify for each (δ_k, c_k) in L_1, the external nodes z in the BBST for L_2 for which $\sum_{i=1}^{k-1}\delta_i < f(z) \leq \sum_{i=1}^{k}\delta_i$, where the δ_is are defined with respect to L_1 and $f(z)$ is the sum of the $d()$ values of the external nodes in the BBST of L_2 that lie to the left of z plus the $d()$ value of z. Let x and y be the leftmost and rightmost such external nodes (see Figure 11). These nodes can be found in $O(\log|L_2|)$ time using $\sum_{i=1}^{k-1}\delta_i$ and $\sum_{i=1}^{k}\delta_i$, and the D values in the triples of the internal nodes of the BBST for L_2. Actually, node x may already be known from the processing of the pair (δ_{k-1}, c_{k-1}) of L_1. We need to increase the c values of the external nodes from x to y. This can be done in logarithmic time by changing the C correctors stored in the internal nodes on the paths from x and y to their common ancestor (see Figure 11).

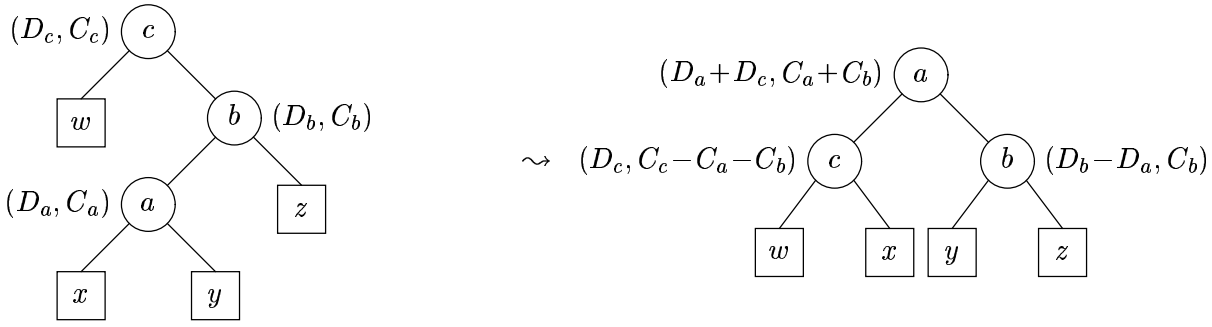
In addition to the above change in C correctors, we may need to insert a new external node. If $f(y) = \sum_{i=1}^{k}\delta_i$, then no insertion is needed, we increase $c(y)$ by c_k. Otherwise, we change the original node to $(\sum_{i=1}^{k}\delta_i - f(y) - d(y), c(y)+c(k))$ and insert $(f(y) - \sum_{i=1}^{k}\delta_i, c_k + c(y))$ into the BBST. The inserted external node is the x node for the next pair of L_1.



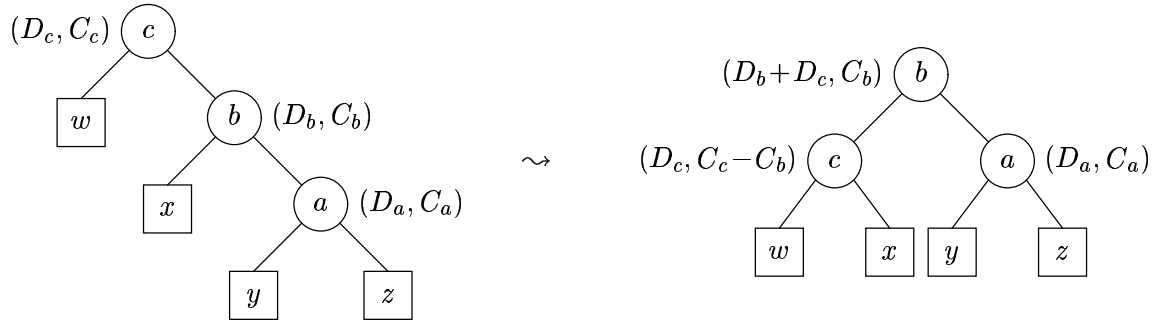
(a) LL rotation



(b) LR rotation



(c) RL rotation



(d) RR rotation

Figure 10: Update of $D()$s and $C()$s for internal nodes during tree rotations

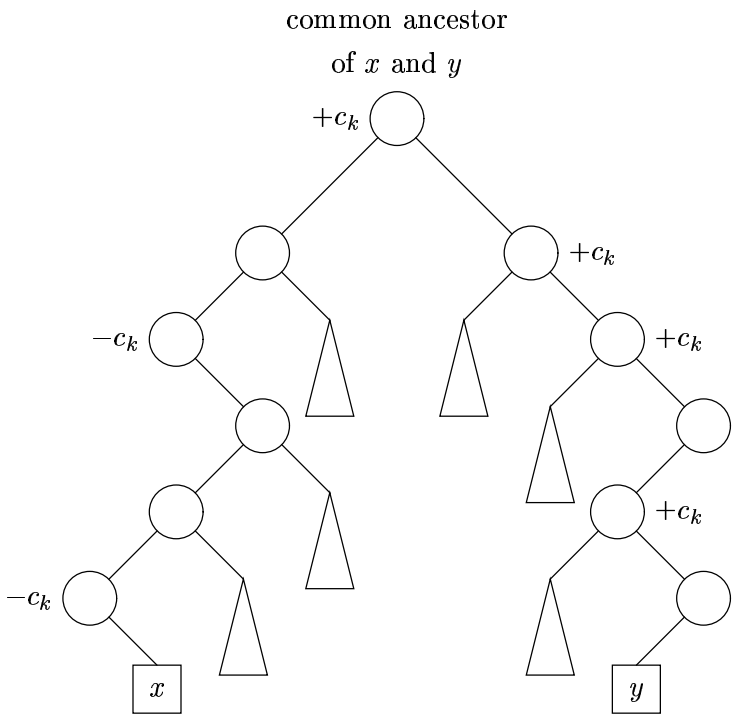


Figure 11: Change of c values of internal nodes of L_2 for the kth tuple of L_1

When the BBST method is used on the worst-case example for the linked list method (Figure 8), $|L_1| = k$ and $|L_2| \leq kn$ for each of the $n-1$ merges. Therefore, the run time is $O(kn \log kn)$. The worst-case for the BBST method is when we continually merge DP lists of the same size. The worst case is described by $\log n$ stages of merges; each stage involving pairs of DP lists of the same size. In stage 1, $\frac{n}{2}$ pairs of lists each of size k are merged in $O(\frac{n}{2} k \log 2k)$ time to produce $\frac{n}{4}$ DP lists of size $2k$ each; in stage 2, $\frac{n}{8}$ pairs of DP lists of size $2k$ each are merged in $O(\frac{n}{4} 2k \log 4k)$ time to produce $\frac{n}{8}$ DP lists of size $4k$ each; and so on. The total time is $O(\sum_{i=1}^{\log n} \frac{n}{2} k \log kn) = O(kn \log kn \log n)$.

Figure 12 shows a circuit on which this worst-case bound is achieved. Let C_0 be a circuit with a single module. $C_i, i > 0$, is a simple parallel circuit obtained from C_{i-1} as shown in Figure 12. The number of modules in C_i is $2^{i+1} + i(i-1)$ and C_i requires 2^{i-1} parallel merges and $2^i - 1$ series merges. The total cost is $O(kn \log kn \log n)$.

3 Tree Circuits

Gates in circuits with a tree topology (for example, distribution trees) can be resized by transforming the trees into equivalent single gate circuits using the basic transformation shown in Figure 13. The transformation of Figure 13 first transforms a node, all of whose children are leaves, into an

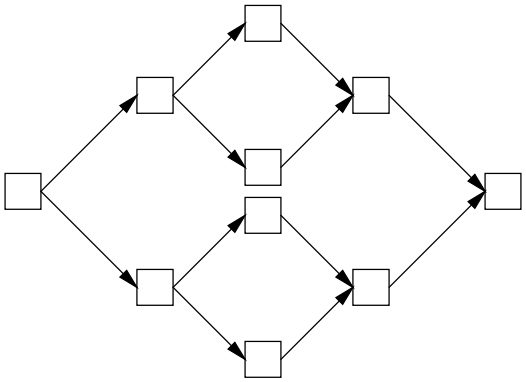


Figure 12: Circuit C_2 that exert the worst case behavior

equivalent simple parallel circuit by the introduction of additional gates/nodes with delay $r - r_i$, where r_i is the required time for the output of leaf i and $r = \max(r_i)$. The c values for the new gates are 0. The simple parallel circuit can now be transformed into an equivalent single gate using the transformation of Figure 3. By repeatly applying this transformation on any tree, the tree can be transformed into an equivalent single gate.

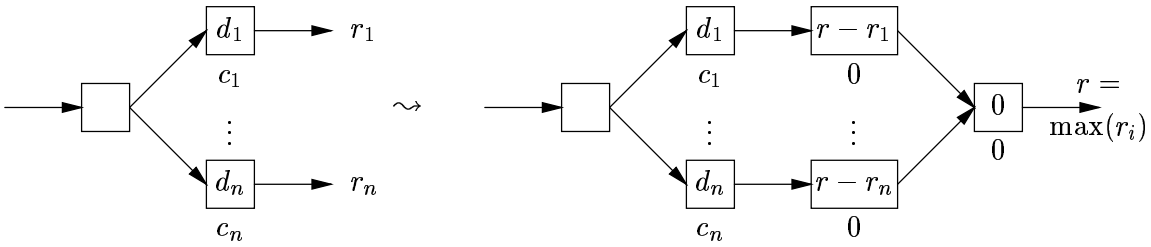


Figure 13: Transformation of a basic tree to a simple parallel circuit

Although the preceding transformation was described specifically for the CGR problem, it is easily extended to the CUGR and ConvexCGR problems using the ideas of Section 2.

4 Conclusion

We have developed polynomial time algorithms for the CGR, CUGR, and ConvexCGR problems for series-parallel and tree circuits.

References

[1] Wing-Ning Li, Andrew Lim, Prathima Agrawal, and Sartaj Sahni. On the circuit implementation problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(8):1147–1156, August 1993.

[2] J. Valdes, R. Tarjan, and E. Lawler. The recognition of series and parallel digraphs. *SIAM Journal of Computing*, 11(2):298–313, May 1982.

[3] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, New Jersey, 1962.

[4] De-Sheng Chen and Majid Sarrafzadeh. An exact algorithm for low power library-specific gate re-sizing. In *Proceeding of the 33rd Design Automation Conference*, pages 783–788, Las Vegas, Nevada, 1996.

[5] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of average switching activity in combinational and sequential circuits. In *Proceedings of the 29th Design Automation Conference*, pages 253–259, Anaheim, California, 1992.

[6] F. Najm. Transition density, a stochastic measure of activity in digital circuits. In *Proceedings of the 28th Design Automation Conference*, pages 644–649, San Francisco, California, 1991.

[7] E. Cheng and S. Sahni. Gate resizing to reduce power consumption. In *Technical Report*, University of Florida, 1998.

[8] E. Cheng and S. Sahni. Resizing gates in series-parallel graphs and trees to minimize power consumption. In *Technical Report*, University of Florida, 1998.

[9] R. Iris Bahar, Gary D. Hachtel, Enrico Macii, and Fabio Somenzi. A symbolic method to reduce power consumption of circuits containing false paths. In *IEEE International Conference of Computer Aided Design*, pages 368–371, San Jose, California, November 1994.

[10] I. Rival, editor. *Graphs and Orders: the Role of Graphs in the Theory of Ordered Sets and its Applications.* D. Reidel Publishing Company, Dordrecht, Holland, May 1984.

[11] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[12] Salah E. Elmaghraby. *Activity Networks: Project Planning and Control by Network Models.* John Wiley and Sons, New York, 1977.

[13] Naveed Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, 2nd edition, 1995.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[15] Sartaj Sahni. *Data Structures, Algorithms, and Applications in C++*. McGraw Hill, 1998.

[16] Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta. *Fundamentals of Data Structures in C++*. W. H. Freeman and Company, 1995.