A Fast Algorithm for Transistor Folding *

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

February 14, 2002

Abstract

Transistor folding reduces the area of row-based designs that employs transistors of different size. Kim and Kang [1] have developed an $O(m^2 \log m)$ algorithm to optimally fold m transistor pairs. In this paper we develop an $O(m^2)$ algorithm for optimal transistor folding. Our experiments indicate that our algorithm runs 3 to 60 times as fast for m values in the range $[100,\ 100,000]$.

Keywords and Phrases: transistor folding, row-based design, area minimization, complexity.

1 Introduction

In high-performance circuit design, the transistor sizing problem was investigated widely in the past (for example, [7, 8, 9, 10]). The objective of transistor sizing is to reduce the circuit delay by increasing the area of transistors. One by-product of transistor sizing is the generation of layouts of transistors of widely varying size. In row-based layout synthesis ([3, 4, 5, 6]), we group pMOS and nMOS transistors together and place them in rows. The layout area for these designs is wasted due to nonuniform cell heights. The layout area required can be reduced by folding large transistors so

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

that their height is reduced. Transistor folding to optimize layout area has been considered in [1] and [2]. Her and Wong [2] have developed an $O(m^6)$, where m is the number of transistor pairs, dynamic programming algorithm for the general transistor folding problem. (If only s heights are possible for the folded transistors, the complexity of Her and Wong's algorithm is $O(m^3 s^3)$. In general, s is $O(m)$.) Kim and Kang [1] have developed a more practical algorithm for the case of row-based designs. The complexity of their algorithm is $O(m^2 \log m)$ or $O(s(m+s)\log m)$. They also show that the area of row-based designs can be reduced by as much as 30% by performing transistor folding. In this paper, we consider the row-based-design transistor-folding problem considered in [1] and develop an $O(m^2)$ or $O(s(m+s))$ algorithm to minimize area. We also report on experiments conducted by us that show that our algorithm actually runs much faster than the algorithm of [1]. The test circuits used in our experiments have between 100 and 100,000 transistor pairs. So, our tests are similar to those conducted in [1] where the circuits had from 192 to 88,258 transistor pairs.

2 Problem Formulation

We are given a CMOS circuit with a row of m transistor pairs. Each transistor pair consists of a pMOS transistor and its dual nMOS transistor. Let p_i and n_i, respectively, be the heights of the pMOS and nMOS transistors in the ith pair, $1 \leq i \leq m$. p_i and n_i are integers that give transistor height in multiples of the minimum resolution λ. Figure 1 shows a CMOS circuit with 4 pairs of transistors, $p_2 = 10$ and $n_2 = 12$. If the folding height of pMOS transistors is 4 and that of nMOS transistors is 3, then the circuit layout is as in Figure 2. The second pMOS transistor is divided into three columns of height 4, 4, and 2 respectively, and the second nMOS transistor is divided into four columns of height 3 each. The area occupied by the folded transistor pair is shown by a

shaded box in Figure 2. In practice, the height of the layout area is slightly larger than the sum of the pMOS and nMOS folding heights, and the layout width is slightly larger than the number of transistor columns because of overheads.

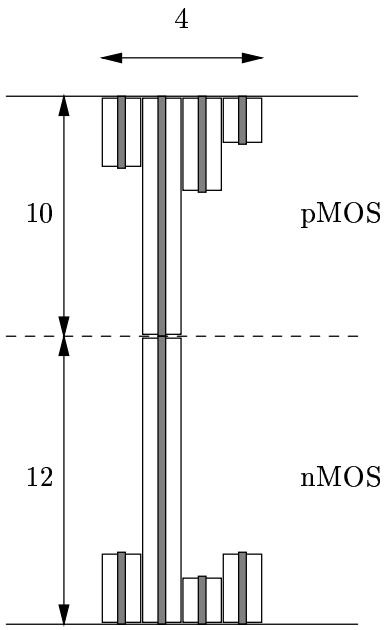

Figure 1: An example circuit with 4 pairs of transistors.

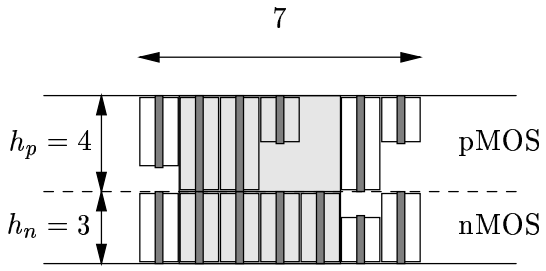

Figure 2: The Circuit of Figure 1 after folding with $h_p = 4$ and $h_n = 3$.

Let h_p and h_n be the folded heights of the pMOS and nMOS transistors, respectively. The height of the folded layout is $h_p + h_n + c_v$ and the width is $\sum_{i=1}^{m} \max(\lceil \frac{p_i}{h_p} \rceil, \lceil \frac{n_i}{h_n} \rceil) + c_h$ where c_v and c_h are, respectively, vertical and horizontal overheads. The area of the folded layout is [1]:

$$A = (h_p + h_n + c_v)(\sum_{i=1}^{m} \max(\lceil \frac{p_i}{h_p} \rceil, \lceil \frac{n_i}{h_n} \rceil) + c_h) \quad (1)$$

In practice, there is a technological constraint on how small h_p and h_n can be. It is required [1] that $h_p \geq PMIN$ and $h_n \geq NMIN$.

Kim and Kang [1] give two algorithms to determine h_p and h_n so that the layout area is minimized. The first algorithm is an exhaustive search algorithm that simply tries out all integer choices for h_p and h_n such that $PMIN \leq h_p \leq \max_{1 \leq i \leq m}\{p_i\} = \max(P)$ and $NMIN \leq h_n \leq \max_{1 \leq i \leq m}\{n_i\} = \max(N)$. (Here $P = \{p_1, p_2, \ldots, p_m\}$ and $N = \{n_1, n_2, \ldots, n_m\}$.) The complexity of the exhaustive search algorithm is $O(\max(P) \cdot \max(N) \cdot m) = O(m^3)$ because $\max(P)$ and $\max(N)$ are $O(m)$ for practical circuits [1].

The second algorithm given in [1] works in two phases. In the first phase, the algorithm constructs a subset S^P of $[PMIN,\ \max(P)]$ and another subset S^N of $[NMIN,\ \max(N)]$ with the property that the optimal h_p is in S^P and the optimal h_n is in S^N. The basic observation used to arrive at S^P and S^N is that if the heights h_1 and $h_1 + k$ divide a transistor into the same number of columns then h_1 is preferred over $h_1 + k$ (for example if $p_1 = 14$, then folding heights 7,8,9,10,11,12 and 13 all fold the transistor into two columns; 7 is preferred over the remaining choices). In the second phase the optimal combination (h_p, h_n) is determined from S^P and S^N. The complexity of the second phase is $O(s(m+s)\log m) = O(m^2 \log m)$, where $s = |S^P| + |S^N|$, and that of the first phase is $\Theta(\sum_{i=1}^{m}(p_i + n_i)) = O(m^2)$ (assuming $\max(P)$ and $\max(N)$ are $O(m)$).

3 Our Algorithm

3.1 Phase I

Our algorithm is also a two phase algorithm. The first phase of our algorithm is identical to the first phase of Kim and Kang's algorithm [1]. We compute the subsets S^P and S^N using the code of Figure 3. The arrays *SPL* and *SNL* are initialized to zero in the first two **for** loops. Then we determine the members of S^P and S^N; we set $SPL[i] = 1$ if and only if $i \in S^P$ and $SNL[i] = 1$ if and only if $i \in S^N$. Finally, S^P and S^N are computed in compact form from *SPL* and *SNL* respectively. Note that we can compute S^P and S^N in either ascending or descending order easily by controlling the direction of traversal of the *SNL* and *SPL* arrays respectively, in the last two **for** loops. The algorithm presented in Figure 3 computes S^P in ascending order and S^N in decending order.

3.2 Phase II

Assume that the transistor pairs have been reordered so that $\frac{p_i}{n_i} \leq \frac{p_{i+1}}{n_{i+1}}$; also assume that $\frac{p_0}{n_0} = 0$ and $\frac{p_{m+1}}{n_{m+1}} = \infty$. The formula, Equation 1, for the layout area can be rewritten as

$$A = (h_p + h_n + c_v)(\sum_{i=1}^{k-1} \lceil \frac{n_i}{h_n} \rceil + \sum_{i=k}^{m} \lceil \frac{p_i}{h_p} \rceil + c_h) \tag{2}$$

where $k \in [1,\ m+1]$ is such that

$$\frac{p_{k-1}}{n_{k-1}} < \frac{h_p}{h_n} \leq \frac{p_k}{n_k} \tag{3}$$

Algorithm Phase I $(P, N, PMIN, NMIN)$

```
/* Compute S^P and S^N */

/* Initialize SPL[] and SNL[] */
for i = PMIN to max(P) do
   SPL[i] ← 0;
for i = NMIN to max(N) do
   SNL[i] ← 0;

/* set SPL[] and SNL[] */
for i = 1 to m do
   for j = 1 to p_i do
      SPL[⌈p_i / j⌉] ← 1;
   for j = 1 to n_i do
      SNL[⌈n_i / j⌉] ← 1;
end for

/* collect items from SPL[] and SNL[] and store them into S^P[] and S^N[] */
SPsize ← 0; SNsize ← 0;
for i = PMIN to max(P) do
   if SPL[i] = 1 then
      S^P[SPsize++] ← i;
for i = max(N) downto NMIN do
   if SNL[i] = 1 then
      S^N[SNsize++] ← i;
```

Figure 3: Computing S^P and S^N

Let $L_N(h_n, k) = \sum_{i=1}^{k-1} \lceil \frac{n_i}{h_n} \rceil$ and $L_P(h_p, k) = \sum_{i=k}^{m} \lceil \frac{p_i}{h_p} \rceil$, we can rewrite Equation 2 as

$$A = (h_p + h_n + c_v)(L_N(h_n, k) + L_P(h_p, k) + c_h) \tag{4}$$

From Equation 3 and the ordering of the transistors by $\frac{p_i}{n_i}$, it follows that if h_n is held fixed and h_p increased, the value of k cannot decrease. This observation results in the algorithm of Figure 4.

Algorithm Phase II $(P, N, S^P, S^N, c_h, c_v)$
  /* S^P is in ascending order and S^N is in descending order */
  Sort P and N in increasing $P[i]/N[i]$ ratio;
  Compute $L_N[][]$ and $L_P[][]$;
  for each $h_n \in S^N$ **do**
    $k \leftarrow 1$;
    for each $h_p \in S^P$ **do**
      while $P[k]/N[k] < h_p/h_n$ **do**
        $k \leftarrow k + 1$;
      $A = \min(A, (h_p + h_n + c_v) * (L_N[h_n][k] + L_P[h_p][k] + c_h))$;
    end for
  end for

Figure 4: Compute optimal h_p and h_n

Since S^P and S^N can be computed in ascending and descending order respectively by Algorithm Phase I of Figure 3, no sorting is needed to evaluate the members of S^P and S^N in the specified order. We can sort the transistors into increasing (actually nondecreasing) order of $\frac{p_i}{n_i}$ in $O(m \log m)$ time; and the arrays L_N and L_P can be computed in $\Theta(m|S^N|)$ and $\Theta(m|S^P|)$ time respectively. Each iteration of the outer **for** loop takes $O(|S^P| + m)$ time. Therefore, the time needed for all $|S^N|$ iterations is $O(|S^N|(|S^P| + m))$. We can change this complexity to $O(\min\{|S^N|, |S^P|\}(\max\{|S^N|, |S^P|\} + m))$ by interchanging the inner and outer **for** loop headers.

Further improvement in run time is possible. Consider the algorithm of Figure 4. Let k_i be the

k value that satisfies Equation 3 when we use the first (i.e., largest) h_n value h_{n_1} and the ith (i.e. ith smallest) h_p value h_{p_i}. On the next iteration of the outer **for** loop, $h_{n_2} \leq h_{n_1}$, so $\frac{h_{p_i}}{h_{n_1}} \leq \frac{h_{p_i}}{h_{n_2}}$, and the k value that satisfies Equation 3 is at least k_1. Hence if we save k_1 from the first iteration, we can start the search for the new k value at k_1. This observation leads to the refinement shown in Figure 5. Although its worst-case complexity is the same as that of Figure 4, it is expected to run faster in practice.

Algorithm Refined Phase II $(P, N, S^P, S^N, c_h, c_v)$
  /* S^P is in ascending order and S^N is in descending order */
  Sort P and N in increasing $P[i]/N[i]$ ratio;
  Compute $L_N[][]$ and $L_P[][]$;
  Initialize K_{h_p} to 0 for all $h_p \in S^P$
  if $|S^N| \leq |S^P|$ **then**
    for each $h_n \in S^N$ **do**
      $k \leftarrow 1$;
      for each $h_p \in S^P$ **do**
        $K_{h_p} \leftarrow \max(k, K_{h_p})$;
        while $P[k]/N[k] < h_p/h_n$ **do**
          $k \leftarrow k + 1$;
        $K_{h_p} \leftarrow k$;
        $A = \min(A, (h_p + h_n + c_v) * (L_N[h_n][k] + L_P[h_p][k] + c_h))$;
      end for
    end for
  else
    /* same as "if", but interchange the inner and outer for loop headers, and replace K_{h_p} by K_{h_n} */
  end if

Figure 5: Refined Phase 2 algorithm

4 Algorithmic Variants

Although the use of the preprocessing phase, Phase I, dramatically reduces the run time when processing is completed using the technique of Kim and Kang [1], it is possible that the cost of

this preprocessing when used in conjunction with our refined Phase II algorithm of Figure 5 may exceed the benefits that accrue from the preprocessing. Therfore, we formulate another version of our algorithm which does not employ the Phase I preprocessing algorithm of Figure 3. The new version, Figure 6 is just the algorithm of Figure 5 with the **while** loops replaced by **for** loops.

Algorithm New Phase II (P, N, c_h, c_v)
  Sort P and N in increasing $P[i]/N[i]$ ratio;
  Compute $L_N[][]$ and $L_P[][]$;
  Initialize K_{h_p} to 0 for all h_p;
  if $|S^N| \leq |S^P|$ **then**
    for $h_n = MAXN$ downto $PMIN$ **do**
      $k \leftarrow 1$;
      for $h_p = PMIN$ to $MAXP$ **do**
        $K_{h_p} \leftarrow \max(k, K_{h_p})$;
        while $P[k]/N[k] < h_p/h_n$ **do**
          $k \leftarrow k + 1$;
        $K_{h_p} \leftarrow k$;
        $A = \min(A, (h_p + h_n + c_v) * (L_N[h_n][k] + L_P[h_p][k] + c_h))$;
      end for
    end for
  else
    /* same as "if", but interchange the inner and outer for loop headers, and replace K_{h_p} by K_{h_n} */
  end if

Figure 6: New Algorithm without Phase I

Figure 7 shows a variant of the algorithm of Figure 6 in which only one of the arrays L_P and L_N is precomputed, the values of the other array are computed as needed. This variant reduces the space requirements of the algorithm, and, as we shall see, also reduces the time requirements.

5 Experimental Results

The phase 1 algorithm of Figure 3, the phase 2 algorithm of Figure 5, the algorithmic variants of Figures 6 and 7, and the two algorithms of [1] were implemented, by us, in C and run on a

Algorithm Phase II without LN (P, N, c_h, c_v)

Sort P and N in increasing $P[i]/N[i]$ ratio;
Compute $L_P[][]$ only;
Initialize K_{h_p} to 0 for all h_p
if $|S^N| \leq |S^P|$ **then**
  for $h_n = MAXN$ downto $PMIN$ **do**
    $k \leftarrow 1$;
    $l_n = 0$
    for $h_p = PMIN$ to $MAXP$ **do**
      $K_{h_p} \leftarrow \max(k, K_{h_p})$;
      while $P[k]/N[k] < h_p/h_n$ **do**
        $l_n = l_n + \lceil \frac{N[k]}{h_n} \rceil$;
        $k \leftarrow k + 1$;
      $K_{h_p} \leftarrow k$;
      $A = \min(A, (h_p + h_n + c_v) * (l_n + L_P[h_p][k] + c_h))$;
    end for
  end for
else
  /* same as "if", but interchange the inner and outer for loop headers, and replace K_{h_p} by K_{h_n}, L_P by L_N, and l_n by l_p */
end if

Figure 7: New Algorithm without Phase I and LN[][] Precomputation

SUN SPARCstation 4. Similar programming methodologies were used to develop the codes for our algorithms and those of [1]. As a result, we expect that almost all of the performance difference exhibited in our experiments is due to algorithmic rather than programming differences. Since we were unable to obtain the test data used in [1], we generated random data. We ignore any possible correlation between pMOS and nMOS transistors. For our test data, the number of transistor pairs ranged from 100 to 100,000. This covers the range in transistor numbers (192 to 88,258) in the circuits of [1]. For our first test set, the sizes of the pMOS and nMOS transistors were generated using a uniform random number generator with range $[30, 90]$ for pMOS and $[20, 60]$ for nMOS. These size ranges correspond to those for the circuit *fract* that was used in [1], the circuit *fract* has 598 transistors. Since all of the tested algorithms generate optimal solutions, run time is the only comparative factor. This time is provided in Table 1. Other than in the column labeled "Phase 1", all times are the times needed for the entire area minimization process (i.e., phase 1 plus phase 2). The exhaustive search algorithm was not run for $m > 10,000$ as its run time becomes prohibitive. In the case of the algorithm proposed in [1], the phase 2 time is significantly larger than the phase 1 time. Our algorithm for phase 2 (5) has brought this time down to approximate the phase 1 time. Equally interestingly, on the data sets used by us, the preprocessing of phase 1 is no longer of any use. When this preprocessing is eliminated (as in Figure 6), the run time reduces by an additional 30%. The variant of Figure 7 provides modest run time improvement (due mainly to not having to reference a two-dimensional array to get one of the L values), but provides a 50% reduction in space needs.

For small circuits ($m \leq 10,000$), our algorithm of Figure 7 provides a speedup of 3.8 to 7.1 over the algorithm of [1]. On larger circuits, the speedup is more dramatic. For instance, when $m = 100,000$ our algorithm of Figure 7 is almost 27 times as fast as that of [1].

m	Exhaustive	Phase 1	Kim & Kang	Fig. 5	Fig. 6	Fig. 7
100	1.46	0.02	0.50	0.09	0.07	0.07
300	4.41	0.07	0.92	0.27	0.19	0.19
500	7.34	0.11	1.29	0.44	0.32	0.30
600	8.79	0.15	1.48	0.53	0.38	0.37
1000	14.67	0.25	2.35	0.88	0.63	0.61
5000	74.59	1.20	13.54	4.48	3.17	3.03
10000	149.12	2.43	35.09	9.22	6.43	6.22
50000	–	12.42	473.72	47.94	32.12	30.31
100000	–	24.24	1710.36	96.33	69.48	63.58

Time in seconds

Table 1: Run time using a uniform distribution

We experimented with two other data sets. Table 2 reports the run times for circuits in which the range of the uniform random number generator was set to $[30, 180]$ for pMOS transistor sizes and $[20, 120]$ for nMOS sizes and Table 3 gives the run times when the transistor sizes are from a normal distribution with mean 40 and standard deviation 10 for pMOS transistors and mean 30 and standard deviation 10 for nMOS transistors. The speedups range from a low of 4.1 to a high of 66.3.

m	Exhaustive	Phase 1	Kim & Kang	Fig. 5	Fig. 6	Fig. 7
100	6.35	0.05	1.73	0.16	0.17	0.16
300	19.87	0.13	3.36	0.54	0.43	0.42
500	33.15	0.22	4.44	0.91	0.69	0.66
600	39.77	0.26	4.95	1.09	0.83	0.80
1000	66.31	0.44	6.67	1.81	1.36	1.29
5000	336.92	2.19	26.71	9.24	6.74	6.41
10000	673.43	4.44	57.63	18.64	13.42	12.72
50000	–	22.05	516.08	98.22	75.97	65.80
100000	–	44.17	3777.96	208.33	158.13	139.05

Time in seconds

Table 2: Run time using a uniform distribution with larger limits

m	Exhaustive	Phase 1	Kim & Kang	Fig. 5	Fig. 6	Fig. 7
100	0.90	0.02	0.22	0.04	0.03	0.03
300	3.22	0.06	0.53	0.13	0.10	0.09
500	5.79	0.10	0.83	0.21	0.17	0.16
600	6.70	0.12	1.00	0.25	0.20	0.19
1000	12.69	0.19	1.68	0.43	0.36	0.34
5000	68.26	0.96	13.67	2.42	1.89	1.77
10000	129.67	1.92	38.07	4.79	3.65	3.41
50000	–	9.60	677.42	27.20	19.29	18.04
100000	–	19.22	2663.71	59.49	45.33	40.12

Time in seconds

Table 3: Run time using a normal distribution

6 Conclusion

We have developed a transistor folding algorithm that is both theoretically and practically faster than the algorithm proposed in [1]. Our algorithm is also simpler to code. Experiments suggest that our algorithm runs 3 to 60 times as fast as the algorithm of [1] on circuits with 100 to 100,000 transistor pairs. These circuit sizes are comparable to those used in [1].

References

[1] Jaewon Kim and S. M. Kang. An efficient transistor folding algorithm for row-based CMOS layout design. In *Proceedings 34th Design Automation Conference*, pages 456–459, 1997.

[2] T. W. Her and D. F. Wong. Cell area minimization by transistor folding. In *Proceedings 1993 Euro-DAC*, pages 172–177, 1993.

[3] Y. C. Hsieh, C. Y. Hwang, Y. L. Lin, and Y. C. Hsu. LiB: A CMOS cell compiler. *IEEE Transactions on Computer-Aided Design*, 10(8), 1991.

[4] A. Stauffer and R. Nair. Optimal CMOS cell transistor placement: A relaxation approach. In *IEEE International Conference on Computer-Aided Design*, November 1988.

[5] T. Uehara and W. VanCleemput. Optimal layout of CMOS functional arrays. *IEEE Transactions on Computers*, C-30(5), 1981.

[6] S. Wimer, R.Y. Pinter, and J.A. Feldman. Optimal chaining of CMOS transistors in a functional cell. *IEEE Transactions on Computer-Aided Design*, 30(5), 1987.

[7] Sachin S. Sapatnekar, Vasant B. Rao, Pravin M. Vaidya, and Sung-Mo Kang. An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. *IEEE Transaction on Computer-Aided Design*, 12(11):1621–1634, November 1993.

[8] J. P. Fishburn and A. E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *IEEE International Conference on Computer-Aided Design*, pages 326–328, November 1985.

[9] J. Shyu, J. P. Fishburn, A. E. Dunlop, and A. L. Sangiovanni-Vincentelli. Optimization-based transistor sizing. *IEEE Journal on Solid-State Circuits*, pages 400–409, April 1988.

[10] Z. Dai and K. Asada. Mosiz: A two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates. In *Proceedings 1989 Custom Integrated Circuits Conference*, pages 17.3.1–17.3.4, May 1989.