ACKNOWLEDGMENT

The author wishes to thank S. Verma for work done on the software developments and other contributions to this write-up. We are also greatful to I. Pomeranz and S. Yadavalli for making the COMPACTSET vectors available. We thank the anonymous referees for their thoughtful comments, and to the referee that pointed out the important reference of [5] and [22].

REFERENCES

[1] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "Smart and fast: Test generation for VLSI scan-design circuits," *IEEE Design and Test*, pp. 43–54, Aug. 1986.

[2] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the role of independent fault sets in the generation of minimal test sets," *Int. Test Conf.*, pp. 1100–1107, 1987.

[3] B. Ayari and B. Kaminska, "A new dynamic test vector compaction for automatic test pattern generation," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 353–358, 1994.

[4] C. Barnhart, E. L. Johnson, R. Anbil, and L. Hatay, "A column generation technique for the long-haul crew assignment problem," Tech. Rep. COC-91-01, Computat. Optimiz. Center, School of Indust. Syst. Eng., Georgia Inst. Technol., Atlanta, GA, 1991.

[5] J.-S. Chang and C.-S. Lin, "Test set compaction for combinational circuits," in *Asian Test Symp.*, 1992, pp. 20–25.

[6] K. T. Cheng and V. D. Agarwal, "A simulation-based directed-search method for test generation," in *Proc. Int. Conf. Computers, Des.*, Oct. 1987, pp. 48–51.

[7] J. Desrosiers, Y. Dumas, M. Desrochers, F. Soumis, B. Sanso, and P. Trudeau, "A breakthrough in airline crew scheduling," Tech. Rep. G-91-11, GERAD, Montreal, Canada, 1991.

[8] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Trans. Comput.*, vol. C-32, pp. 1137–1144, Dec. 1983.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability.* San Francisco, CA: W.H. Freeman, 1979.

[10] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 215–222, Mar. 1981.

[11] N. Hall and D. S. Hochbaum, "The multicovering problem," in *Europ. J. Operat. Res.*, vol. 62, pp. 323–339, 1992.

[12] D. S. Hochbaum, "Approximation algorithms for the weighted set covering and vertex cover problems," *SIAM J. Comput.*, vol. 11, no. 3, pp. 555–556, 1982. An extended version: W.P. #64-79-80, GSIA, Carnegie-Mellon Univ., Apr. 1980.

[13] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-effective generation of minimal test sets for stuck-at-faults in combinational logic circuits," in *30th ACM/IEEE Design Automation Conf.*, 1993, pp. 102–106.

[14] T. Kirkland and M. R. Mercer, "A topological search algorithm for atpg," in *Proc. 24th ACM/IEEE Design Automation Conf.*, June 1987, pp. 502–508.

[15] B. Krishnamurthy and S. B. Akers, "On the complexity of estimating the size of a test set," in *IEEETC*, Aug 1987, pp. 750–753.

[16] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 625–636, Jan. 1992.

[17] J. F. McDonald, "Test set reduction using the subscripted d-algorithm," in *Int. Test Conf.*, pp. 115–121, 1987.

[18] M. Minoux, "Column generation techniques in combinatorial optimization: A new application to crew pairing problems," in *AGIFORS Proc., France*, 1984.

[19] ——, *Mathematical Programming: Theory and Algorithms.* New York: Wiley, 1986.

[20] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A method to generate compact test sets for combinatorial circuits," in *Int. Test Conf.*, 1991, pp. 194–203.

[21] B. Rannou, "A new approach to crew pairing optimization," in *26th AGIFORS Symp.*, England, 1986.

[22] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "ROTCO: A Reverse order test compaction technique," in *EURO-ASIC Conf.*, 1992, pp. 189–194.

[23] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Develop.*, vol. 10, pp. 278–291, July 1966.

[24] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 126–137, Jan. 1988.

[25] Y. Takamatsu and K. Kinoshita, "Cont: A concurrent test generation algorithm," in *FTCS-17*, July 1987, pp. 22–27.

[26] E. Trischler, "Atwig, An automatic test pattern generator with inherent guidance," in *Proc. 1984 Int. Test Conf.*, Oct. 1984, pp. 86–87.

[27] G.-J. Tromp, "Minimal test sets for combinational circuits," in *Int. Test Conf.*, 1991, pp. 194–203.

A Neural Network Approach to PLA Folding Problems

Kazuhiro Tsuchiya and Yoshiyasu Takefuji

***Abstract*—A near-optimum parallel algorithm for solving PLA folding problems is presented in this paper where the problem is NP-complete and one of the most fundamental problems in VLSI design. The proposed system is composed of $n \times n$ neurons based on an artificial two-dimensional maximum neural network where n is the number of inputs and outputs or the number of product lines of PLA. The two-dimensional maximum neurons generate the permutation of inputs and outputs or product lines. Our algorithm can solve not only a simple folding problem but also multiple, bipartite, and constrained folding problems. We have discovered improved solutions in four benchmark problems over the best existing algorithms using the proposed algorithm.**

I. INTRODUCTION

Programmable Logic Array (PLA) is a very effective and efficient means to implement multiple output combinational logic circuits because of its structured array design which makes it possible to transform its canonical logic form directly into a physical layout [1]–[3]. It is especially important in very large scale integrated circuits (VLSI) or ultralarge scale integrated circuits (ULSI) where regular structures and simple designs are required in order to shorten the design and test time.

Fig. 1 shows an example of a representation of a PLA. Fig. 1(a) shows the general structure of the PLA using an NMOS NOR circuit. It consists of two arrays referred to as AND array (plane) and OR array (plane). The AND array has input lines as columns and the OR array has output lines as columns. Product lines (terms) run through both of the arrays as rows. The outputs are the sum-of-products of the inputs. This PLA has seven input lines (*#A* through *#G*), two output lines (*#H* and *#I*), and six product lines (#1 to #6). Fig. 1(a) is replaced by Fig. 1(b) in the PLA folding problem where the transistors are replaced by black points. In this case, vertical intertransistor lines are drawn instead of the input and output lines. A net is composed of either an intertransistor line with transistors connected to it or a

Manuscript received August 11, 1994; revised May 26, 1995 and March 8, 1996. This paper was recommended by Associate Editor C.-K. Cheng.

K. Tsuchiya is with the Faculty of Environmental Information, Keio University, Fujisawa 252, Japan. He is also with Fuji Facom Co., Fuji-machi, Hino, Tokyo 191, Japan.

Y. Takefuji is with the Faculty of Environmental Information, Keio University, Fujisawa 252, Japan. He is also with the Department of Electrical Engineering and Applied Physics, Case Western Reserve University, Cleveland, OH 44106 USA.

Publisher Item Identifier S 0278-0070(96)07184-9.

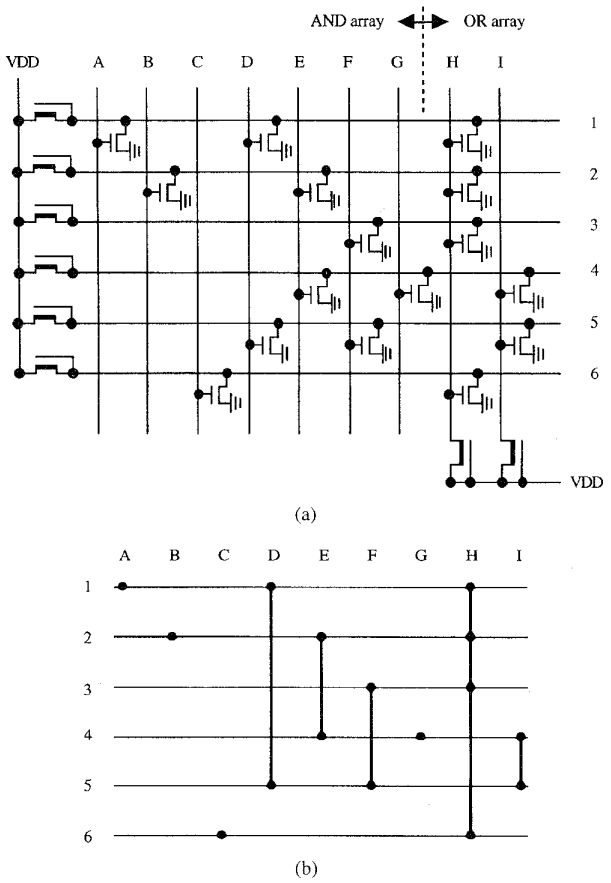

Fig. 1. An example of the representation of a PLA. (a) An example of a PLA by NMOS NOR circuit. (b) The representation of (a) in the PLA folding problem.

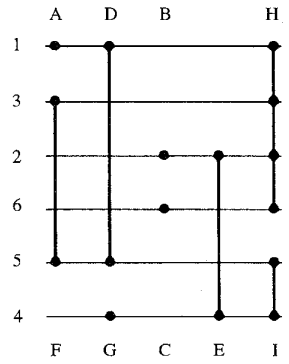

Fig. 2. An example of SF.

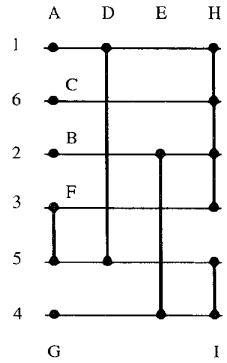

Fig. 3. An example of MF.

transistor connected by no intertransistor lines. The number of nets is the same as the total number of input lines and output lines.

PLA's are, generally, very sparse and large even after logic minimization [4], [5]. PLA column folding is a technique to fold (pack) the nets with the minimum number of columns by optimizing the permutation (order) of rows. Similarly, PLA row folding is performed by folding the product lines. Since both folding methods are basically the same, this paper focuses on column folding for the purpose of illustration and pedagogy, although the proposed algorithm is applicable to both folding methods.

Fig. 2 shows an example of PLA column folding for the PLA in Fig. 1. This type of folding is called simple folding (SF) where a pair of input nets or output nets share the same column, and the number of pairs is maximized. Note that the input lines and the output lines are either on the upper or on the lower sides of the columns so that neither input lines nor output lines intersect each other. Usually, the input nets and the output nets are folded within the AND array and the OR array, respectively, in the PLA folding problem due to electrical or physical constraints. A more general type of SF is called multiple folding (MF) where the input nets and the output nets are folded as much as possible to minimize the number of columns within the AND array and the OR array, respectively. Fig. 3 depicts an example of MF for the PLA in Fig. 1. Although the area can be smaller than that of SF, routing the input lines and the output lines may be complicated and another metal or polysilicon layer may even be necessary. MF is especially effective when the PLA is used as a component of a large system which needs more than one metal or polysilicon layer. Bipartite folding (BF) is a special type of SF, where column breaks (cuts) between the two nets in the same column must occur at the same horizontal level in both the AND array and the OR array. Note that if there exists a net across the break in a column, only the net can be allocated in the column. Fig. 4 shows an example of BF for the PLA in Fig. 1 where the column breaks occur between product lines #1 and #6. Although the area could be larger than that of SF, there are some advantages for BF, which leads to a reduction of the chip area. For example, since the upper region above the break and the lower region below the break can be considered as two individual PLA's, subsequent BF can be further applied to those PLA's. In addition, since augmented circuits can be placed at the break as input decoders and output buffers for testable design of the PLA, the space for them is saved in terms of total chip area, and additional routing and layout are simplified. Constrained folding is a restricted folding where some constraints such as the order and/or place of the lines are given and accommodated with other foldings. These PLA folding problems have proved to be NP-complete [6]–[8]. The number of possible solutions is $O(C!)$ or $O(R!)$ where C and R are the total number of columns and rows, respectively.

Since the PLA folding leads to a significant reduction of silicon area, the problem is one of the most fundamental and important layout problems in VLSI or ULSI. Greer proposed an MF implementation for the first time [9]. Wood presented an SF implementation [10] in 1979. Since then, many algorithms have been proposed to solve the PLA folding problems [7], [8], [11]–[17]. Hachtel *et al.* applied a graph-theoretic formulation and a heuristic algorithm to SF with the time complexity $O(C^3)$ [7], [11]. Micheli *et al.* developed a computer program called PRESURE which can accomplish not

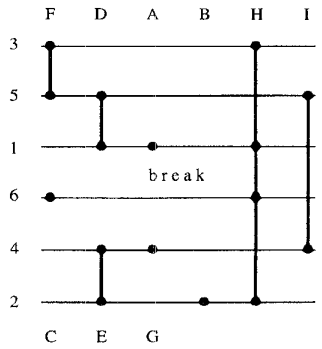

Fig. 4. An example of BF.

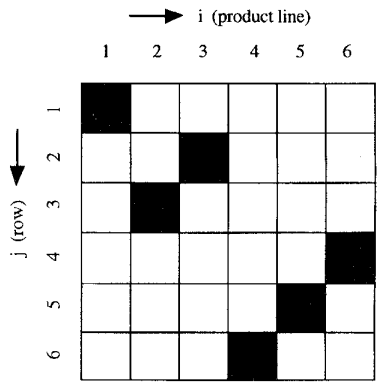

Fig. 5. The 6×6 neural network array for Fig. 2

only SF or MF but also constrained row and column folding [12]. Egan *et al.* proposed BF for the first time and used a branch and bound algorithm to discover optimal solutions [8]. Hwang *et al.* also applied a graph-theoretic formulation and employed a best-first search for finding optimal SF, MF, or constrained folding with the time complexity $\max(O(C^4), O(R^4))$ [13]. Lecky et al. transformed the folding problem to a maximum clique problem as one of graph theory problems and used a greedy algorithm with the time complexity of $\max(O(C^5), O(R^5))$ for SF or MF, and $\max(O(C^2 \log C), O(R^2 \log R))$ for BF [14]. Lussio et al. presented a heuristic algorithm which builds MF row by row with the time complexity of $O(C^2R + R^2C)$ [15]. Hsu *et al.* combined logic minimization and folding where a PLA personality matrix was converted into a network and the optimal BF was obtained by partitioning the network [16]. Liu *et al.* proposed a heuristic algorithm based on a matrix representation for BF [17]. To our knowledge, no parallel algorithm for the problem has been proposed. In this paper, a near-optimum parallel algorithm using a two-dimensional maximum neural network is proposed for the PLA folding problems, where the state of the neurons represents a column or row permutation and motion equations are given for SF, MF, BF and constrained folding to evaluate every neuron.

In Section II, we review the basic concept of artificial neural networks and explain our neural network representation used to solve the problem. In Section III, we describe the neural network parallel algorithm and discuss the experimental results where several benchmark problems are used to justify the effectiveness of our algorithm. We summarize this paper in Section IV.

II. The Neural Network Representation

The first artificial neural network using sigmoidal neurons was introduced by Hopfield and Tank for solving combinatorial optimization problems [19]. Takefuji *et al.* have proposed a hysteresis McCulloch-Pitts neural network and a one-dimensional maximum (winner-take-all) neural network for NP-complete problems [20]–[23]. The mathematical model of the artificial neural networks consists of two components; neurons and synaptic links. The output signal transmitted from a neuron propagates to other neurons through the synaptic links. Every artificial neuron has the input U and the output V. The output V is given by the neuron's input/output function f. For example, conventional neuron's input/output functions such as that of the sigmoidal neuron model or the McCulloch-Pitts neuron model are given by

$$V_{i,j} = f(U_{i,j}) \tag{0}$$

where the subscript i, j means the (i, j)th neuron.

Our system is composed of an $R \times R$ neural network array for a PLA column folding problem. The solution in Fig. 2 is provided by the state of a 6×6 neural network array as shown in Fig. 5. Note that each square represents an output state of the (i, j)th neuron. The black squares and the white squares show that the outputs of the neurons generate 1's and 0's, respectively. The nonzero output of the (i, j)th neuron means that product line $\#i$ is assigned to row $\#j$. Because of the PLA folding constraint, one and only one product line must be assigned to each row and all the product lines must be assigned. This means that one and only one neuron must generate a nonzero output per row and per column in the $R \times R$ neural network array. In order to satisfy this constraint, the two-dimensional maximum neuron model is newly introduced. The input/output function of the neuron model is given by

step 1. $V_{a,b} = 1$ if $U_{a,b} = \max\{U_{i,j}\}$
step 2. $V_{c,d} = 1$ if $U_{c,d} = \max\{U_{i,j} \mid i \neq a, j \neq b\}$
step 3. $V_{e,f} = 1$ if $U_{e,f} = \max\{U_{i,j} \mid i \neq a, c, j \neq b, d\}$
$\vdots$ $\vdots$ $\vdots$ $\vdots$
step R. $V_{g,h} = 1$ if $U_{g,h} = \max\{U_{i,j} \mid i \neq a, c, e, \ldots, j \neq b, d, f, \ldots\}$;

$$V_{k,l} = 0 \quad \text{otherwise.} \tag{1}$$

where $V_{i,j} = 1$ means that product line $\#i$ is embedded in row $\#j$. Those steps are to be executed sequentially. For example, product line $\#a$ should be assigned to row $\#b$ if $U_{a,b}$ is the largest among all $U_{i,j}$'s, then product line $\#c$ should be assigned to row $\#d$ if $U_{c,d}$ is the largest among all $U_{i,j}$'s except for $i = a$ or $j = b$, and so on. Note that if more than one neuron has the largest input, only one neuron among them should be selected. In R^2 maximum neurons, R neurons always generate nonzero outputs and the other $(R^2 - R)$ neurons generate zero so that not more than one product line is assigned per row. This neuron model provides a faster convergence speed and higher convergence rate than those of the conventional sigmoidal neuron or McCulloch-Pitts neuron models. The input of the (i, j)th neuron is determined and updated by the following motion equation.

The motion equation represents the synaptic links. It shows interconnections between the (i, j)th neuron and other neurons. The motion equation of the (i, j)th neuron with a discrete input/output function is given by

$$\frac{\Delta U_{i,j}}{\Delta t} = -\frac{\Delta E(V_{1,1}, \ldots, V_{i,j}, \ldots, V_{n,n})}{\Delta V_{i,j}}. \tag{2}$$

This means that the change of the input of the (i, j)th neuron is given by the partial derivatives of the computational energy function E with respect to the output of the (i, j)th neuron where E follows

an $(n \times n)$-variable function: $E(V_{1,1}, \ldots, V_{i,j}, \ldots, V_{n,n})$. The goal of the artificial neural networks for solving optimization problems is to minimize the fabricated computational energy function E in (2), for which the artificial neural networks provide a parallel gradient descent method. The partial derivatives of E with respect to $V_{i,j}$, however, are difficult to describe for some problems.

The left term in (2), instead, can usually be constructed by considering the necessary and sufficient constraints and/or the cost function from the given problem. The proposed two-dimensional maximum neural network needs only the cost function because the neuron model includes the constraints. The first-order Euler method is used in our algorithm to update $U_{i,j}$ by the left term. The number of columns is the cost to be minimized for the PLA folding problem. Therefore, the motion equation is approximated by

$$\frac{\Delta U_{i,j}}{\Delta t} = Q - P \tag{3}$$

where Q and P are the user-defined objective number of columns and the evaluated number of columns, respectively. In this paper, Q is set to be the same as the maximum number of transistors in the same product line, which is the lower limit of Q. It means that $\frac{\Delta U_{i,j}}{\Delta t}$ is always negative or zero. P is evaluated in the following ways:

1) For BF, given a row permutation by (1), the number of columns is simply evaluated by folding the input nets and output nets whose transistors are all in the upper row above the break in the upper region of the PLA and folding those in the lower row below the break in the lower region.
2) For MF, a left-edge-first algorithm is applied [24] which can fold the input nets and output nets with the smallest number of columns within a given row permutation by (1). The left-edge-first algorithm for MF is shown in Appendix I.
3) For SF, a modified left-edge-first algorithm is used. Here a constraint that two input nets or two output nets at the most share the same column is added to the left-edge-first algorithm.
4) For the constraint folding, other constraints are added to the above algorithms.

In order to calculate (3) for the (i,j)th neuron which generates a zero output, it is assumed that product line $\#i$ is assigned to row $\#j$. For example, to calculate $\Delta U_{5,4}/\Delta t$ in Fig. 2, product line #5 is assigned to row #4 temporarily as shown in Figs. 6(a) and (b) while product line #6 in row #4 is moved to row #5 temporarily, and then the modified left-edge-first algorithm is applied to compute P. P is 6 in this case and Q is 3. Therefore, $\Delta U_{5,4}/\Delta t = 3 - 6 = -3$. To calculate $\Delta U_{5,5}/\Delta t$, the assignment of product line #5 to row #5 must be maintained because product line #5 has already been assigned to row #5 by (1). Since P is 5 as shown in Fig. 2 or Fig. 6(b), $\Delta U_{5,5}/\Delta t = -2$. Equation (3) describes the degree of penalty which discourages the highly penalized neurons from generating a nonzero output.

In order to improve the global minimum convergence and to accelerate the simulation speed, (3) was replaced by

$$\text{If } (t \bmod 10) < \omega \text{ then } \frac{\Delta U_{ij}}{\Delta t} = (Q - P)V_{i,j}, \quad \text{else } \frac{\Delta U_{ij}}{\Delta t} = Q - P \tag{4}$$

where t and ω are the number of iteration steps and a constant parameter, respectively. Note that ω is one and only one parameter for the proposed system. This method helps the state of the system to escape from local minima. The first equation is activated when the (i,j)th neuron generates a nonzero output. Convergence theorem/proof of the two-dimensional maximum neural network is given in Appendix II.

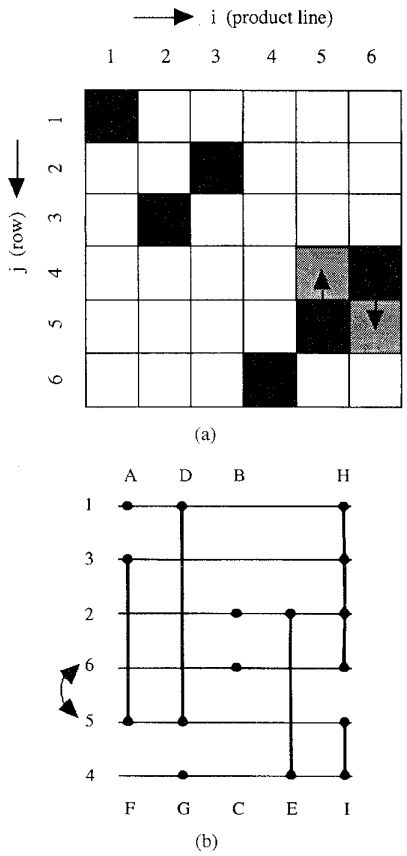

Fig. 6. How to calculate $\Delta U_{5,4}/\Delta t$. (a) The neural network array representation. (b) The layout representation.

III. Parallel Algorithm and Experimental Results

A simulator based on the proposed neural network was developed and a synchronous parallel system and a sequential system were simulated. Both systems were implemented on a Macintosh PowerBook 170 and an HP 9000/710 computer, although the parallel algorithm is executable either on a sequential machine or on a parallel one. The following steps describe the proposed algorithm based on the synchronous parallel system for a PLA column folding problem. Note that t_limit is the maximum number of iteration steps for the system termination condition and that best_P is the smallest number of columns the system discovers.

step 0. Set $t = 0$ and best_$P = 10\,000$, and set t_limit and ω for the PLA folding problem.

step 1. Initialize values of $U_{i,j}(t)$ for $i,j = 1, \ldots, R$ using uniformly randomized numbers.

step 2. Evaluate $V_{i,j}(t)$ for $i,j = 1, \ldots, R$, using (1).

step 3. Compute P and (4) of the $R \times R$ neural network for $i,j = 1, \ldots, R$ to obtain $\Delta U_{i,j}(t)$

$$\Delta U_{i,j}(t) = \frac{\Delta U_{i,j}}{\Delta t} \tag{5}$$

step 4. If best_$P \geq P$, then generate the solution and set best_$P = P$.

step 5. Update $U_{i,j}(t+1)$ for $i,j = 1, \ldots, R$, based on the first-order Euler method

$$U_{i,j}(t+1) = U_{i,j}(t) + \Delta U_{i,j}(t) \tag{6}$$

TABLE I
FOUR BENCHMARK PROBLEMS AND COMPARISON

Problem # [Refrence]	The number of rows *1	The number of nets *1	Folding type	The number of columns		CPU time [sec]	Conv. rate [%] *2
				by this work	by reference		
1 [17] (alu1)	19	20	BF	12	12 [17]	0.85	100
			SF	11	NA	0.43	100
			MF	7	NA	9.5	54.4
2 [18]	18	32 (30)	BF	20	21 [8][16]	0.83	100
			SF	15	NA	4.4	98.6
			MF	11	NA	8.6	64.3
3 [13]	21	38	BF	26	NA	5.9	92.0
			SF	21	21 [13][14] 25 [12][14]	4.7	100
			MF	19	NA	25	78.9
4 [7][11]	52 (39)	42	BF	25	25 [8][16]	27	78.1
			SF	22	25 [11]	23	99.9
			MF	13 14	18 [15]	190 95	4.3 67.4

NA: Not available

*1: Virtually 30 nets in problem #2 and 39 distinct product lines in problem #4.

*2: Within 2000 iteration steps for MF, 1000 iteration steps otherwise.

step 6. Evaluate $V_{i,j}(t+1)$ for $i, j = 1, \ldots, R$, using (1).

step 7. If $t = t\_limit$ then terminate this procedure else increment t by 1 and go to the step 3.

The first-order Euler method was used to solve R^2 equations in (4) numerically. Steps 3 through 5 can run in parallel. It means that the computational time of steps 3 and 5 becomes $1/R^2$ if the system runs on a parallel machine with R^2 processing elements. In the sequential algorithm, (5) and (6) are calculated sequentially, neuron by neuron. The results of the above mentioned parallel algorithm and the sequential one were almost the same in terms of solution quality. Therefore, we only show the results of the parallel algorithm.

We have examined the four benchmark problems to test our algorithm for BF, SF, and MF. Table I shows our results and those of the best existing algorithms. Our algorithm discovered significantly improved solutions which have a smaller number of columns over the best existing algorithms especially in the larger size problem, #4. The result of 13 columns for MF in problem #4 leads to 28% reduction of the area by [15]. Our simulation results show that the solution quality of the proposed neural network does not degrade with the problem size within the range of Table I. Table I also shows the average CPU time and the convergence rate. Note that the average CPU time was measured on the HP 9000/710 and that the convergence rate is the convergence frequency to the number of columns shown in Table I when different initial uniform-random states were used. The average CPU time was more than ten times longer than that of the existing algorithms. However, it is expected to be much less if it is run on a parallel machine. The convergence rate was more than 50% in most of the cases. Note that although the convergence rate to 13 columns for MF in problem #4 was 4.3%, the one to 14 was 67.4%.

IV. CONCLUSION

In this paper we have proposed a near-optimum parallel algorithm using the two-dimensional maximum neural network for the PLA folding problems in VLSI design. The proposed algorithm requires an $n \times n$ neural network where n is the number of product lines or nets. Our algorithm is not only applicable to simple folding but also to multiple folding, bipartite folding, and constrained folding. The simulation results demonstrate the effectiveness of the proposed algorithm in four benchmark problems. We have discovered significant improvements in terms of solution quality using the algorithm.

We are planning to apply the proposed algorithm to other kinds of layout problems such as gate assignment problems. We will also try to install the system on a parallel machine in the future in order to solve larger size problems and to measure the performance of the proposed algorithm in terms of the computational time.

APPENDIX I
THE LEFT-EDGE-FIRST ALGORITHM FOR MF

The following procedure is used for the left-edge-first algorithm to fold the input nets and output nets with the smallest number of columns within a given row permutation. The proof of this algorithm is given in [24]. Note that NET is the number of nets allocated in the columns, COLUMN is the column number being used for assigning the nets, and ROW is the row number being used for the COLUMN.

step 0. Set NET = 0, COLUMN = 1, and ROW = 1.

step 1. Continue the following procedure from step 1-a through step 1-d while NET < "the number of input lines."

step 1-a. Find an input net which is not assigned to any column, whose topmost transistor is located in ROW or below ROW but above those of the other input nets which are not assigned to any column.

step 1-b. If there is more than one input net which satisfies the above condition, choose one.

step 1-c. If there is no net satisfying the above condition in step 1-a, then set COLUMN = COLUMN+1, ROW = 1, and go to step 1-a.

step 1-d. Assign the net to COLUMN and set NET = NET+1, ROW = "the row in which the bottom transistor of the net is located" + 1.

step 2. set ROW = 1 and continue the following procedure from step 2-a through step 2-b while NET < "the number of input lines and output lines."

step 2-a. Find an output net which is not assigned to any column, whose topmost transistor is located in ROW or below ROW but above those of the other output nets which are not assigned to any column.

step 2-b. If there is more than one output net which satisfies the above condition, chose one.

step 2-c. If there is no net satisfying the above condition in step 2-a, then set COLUMN = COLUMN+1, ROW = 1, and go to step 2-a.

step 2-d. Assign the net to COLUMN and set NET = NET+1, ROW = "the row in which the bottom transistor of the net is located" + 1.

Appendix II
Convergence Property of the Two-Dimensional Maximum Neural Network

The convergence property of the two-dimensional maximum neural network is determined by the time derivatives of the energy of the system, $\frac{\Delta E}{\Delta t}$. Lemma 1 is introduced to prove that the proposed system is always allowed to converge to the equilibrium state or the optimal (near-optimum) solution.

Lemma 1: $\frac{\Delta E}{\Delta t} \leq 0$ is satisfied under two conditions such as

(1) $\frac{\Delta U_{i,j}}{\Delta t} = -\frac{\Delta E}{\Delta V_{i,j}} = Q - P < 0$ and

(2) The input/output function of the neuron model is given by

step 1. $V_{a,b} = 1$ if $U_{a,b} = \max\{U_{i,j}\}$

step 2. $V_{c,d} = 1$ if $U_{c,d} = \max\{U_{i,j} \mid i \neq a,\ j \neq b\}$

step 3. $V_{e,f} = 1$ if $U_{e,f} = \max\{U_{i,j} \mid i \neq a, c,\ j \neq b, d\}$

$\vdots \quad \vdots \quad \vdots \quad \vdots$

step n. $V_{g,h} = 1$ if $U_{g,h} = \max\{U_{i,j} \mid i \neq a, c, e, \ldots, j \neq b, d, f, \ldots\}$;

$$V_{k,l} = 0 \quad \text{otherwise.} \tag{1}$$

Proof: Consider the derivatives of the computational energy E with respect to time t.

$$\frac{\Delta E}{\Delta t} = \sum_i \sum_j \frac{\Delta U_{i,j}}{\Delta t} \frac{\Delta V_{i,j}}{\Delta U_{i,j}} \frac{\Delta E}{\Delta V_{i,j}}$$

$$= -\sum_i \sum_j \left(\frac{\Delta U_{i,j}}{\Delta t}\right)^2 \frac{\Delta V_{i,j}}{\Delta U_{i,j}}$$

where $\frac{\Delta E}{\Delta V_{i,j}}$ is replaced by $-\frac{\Delta U_{i,j}}{\Delta t}$. (condition 1)

Let $\frac{\Delta U_{i,j}}{\Delta t}$ be

$$\frac{U_{i,j}(t+\Delta t) - U_{i,j}(t)}{\Delta t}$$

and $\frac{\Delta V_{i,j}}{\Delta U_{i,j}}$ be

$$\frac{V_{i,j}(t+\Delta t) - V_{i,j}(t)}{U_{i,j}(t+\Delta t) - U_{i,j}(t)}.$$

Assume that only the input of the (a,b)th neuron is changed during time t and $t+\Delta t$ in the system.

$$-\sum_i \sum_j \left(\frac{\Delta U_{i,j}}{\Delta t}\right)^2 \frac{\Delta V_{i,j}}{\Delta U_{i,j}}$$

$$= -\sum_i \sum_j \left(\frac{U_{i,j}(t+\Delta t) - U_{i,j}(t)}{\Delta t}\right)^2 \frac{V_{i,j}(t+\Delta t) - V_{i,j}(t)}{U_{i,j}(t+\Delta t) - U_{i,j}(t)}$$

$$= -\sum_i \sum_j \frac{U_{i,j}(t+\Delta t) - U_{i,j}(t)}{(\Delta t)^2} (V_{i,j}(t+\Delta t) + V_{i,j}(t))$$

$$= -\frac{U_{a,b}(t+\Delta t) - U_{a,b}(t)}{(\Delta t)^2} (V_{a,b}(t+\Delta t) - V_{a,b}(t))$$

$\frac{\Delta U_{a,b}}{\Delta t}$ is negative (condition 1). Therefore, it is necessary and sufficient to consider the following two cases (condition 2):

1) $V_{a,b}(t+\Delta t) = V_{a,b}(t)$
2) $V_{a,b}(t+\Delta t) = 0,\ V_{a,b}(t) = 1$

If case 1 is satisfied, then

$$-\sum_i \sum_j \left(\frac{\Delta U_{i,j}}{\Delta t}\right)^2 \frac{\Delta V_{i,j}}{\Delta U_{i,j}} = 0$$

because $V_{a,b}(t+\Delta t) - V_{a,b}(t) = 0$.

If case 2 is satisfied, then

$$-\sum_i \sum_j \left(\frac{\Delta U_{i,j}}{\Delta t}\right)^2 \frac{\Delta V_{i,j}}{\Delta U_{i,j}} < 0$$

because $V_{a,b}(t+\Delta t) - V_{a,b}(t) = -1$ and $\frac{U_{i,j}(t+\Delta t) - U_{i,j}(t)}{\Delta t} = \frac{\Delta U_{i,j}}{\Delta t} = Q - P < 0$. (condition 1)

Therefore $\frac{\Delta E}{\Delta t} \leq 0$. □

References

[1] H. Fleisher and L. I. Maissel, "An introduction to array logic," *IBM J. Res. Develop.*, vol. 19, pp. 98–109, Mar. 1975.

[2] J. W. Jones, "Array logic macros," *IBM J. Res. Develop.*, vol. 19, pp. 120–126, Mar. 1975.

[3] M. S. Schmookler, "Design of large ALU's using multiple PLA macros," *IBM J. Res. Develop.*, vol. 24, pp. 2–14, Jan. 1980.

[4] S. J. Hong, R. G. Cain, and D. L. Oskapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, pp. 443–458, Sept. 1974.

[5] R. Brayton, G. D. Hachtel, L. Hemachanandra, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "A comparison of logic minimization strategies using espresso—An APL program package for partitioned logic minimization," in *Proc. Int. Symp. Circuits Syst.*, Rome, 1982, pp. 42–48.

[6] M. Luby, U. Vazirani, V. Vazirani, and A. L. Sangiovanni-Vincentelli, "Some theoretical results on optimal PLA folding problem," in *Proc. IEEE Int. Conf. Circuits Comput.*, 1982, pp. 165–170.

[7] G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "Some results in optimal PLA folding," in *Proc. Int. Circuits Comp. Conf.*, New York, 1980, pp. 1023–1028.

[8] J. R. Egan and C. L. Liu, "Bipartite folding and partitioning of a PLA," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 191–199, July 1982.

[9] D. L. Greer, "An associative logic matrix," *IEEE J. Solid-State Circuits*, vol. SC-11, pp. 679–691, Oct. 1976.

[10] R. A. Wood, "A density programmable logic array chip," *IEEE Trans. Comput.*, vol. C-28, pp. 602–608, Sept. 1979.

[11] G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "An algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 63–77, Apr. 1982.

[12] G. D. Micheli and A. L. Saniovanni-Vincentelli, "Multiple constrained folding of programmable logic arrays: Theory and applications," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 151–167, July 1983.

[13] S. Y. Hwang, R. W. Dutton, and T. Blank, "A best-first search algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 433–442, July 1986.

[14] J. E. Lecky, O. J. Murphy, and R. G. Absher, "Graph theoretic algorithm for the PLA folding problem," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1014–1021, Sept. 1989.

[15] F. Lussio and M. C. Pinitti, "Suboptimal solution for PLA multiple column folding," *Computer-Aided Design*, vol. 22, pp. 515–520, Oct. 1990.

[16] Y.-C. Hsu, Y.-L. Lin, H.-C. Hsieh, and T.-H. Chao, "Combining logic minimization and folding for PLA's," *IEEE Trans. Computer*, vol. 40, pp. 706–713, June 1991.

[17] C.-Y. Liu and K. K. Saluja, "An efficient algorithm for bipartite PLA folding," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1839–1847, Dec. 1993.

[18] S. Kang and W. M. vanCleemput, "Automatic PLA synthesis from a DDL-P description," in *Proc. 18th Design Automation Conf.*, 1981, pp. 391–397.

[19] J. Hopfield and D. Tank, "Neural computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.

[20] Y. Takefuji, *Neural Network Parallel Computing*. Norwell, MA: Kluwer Academic, 1992.

[21] K. C. Lee, N. Funabiki, and Y. Takefuji, "A parallel improvement algorithm for bipartite subgraph problem," *IEEE Trans. Neural Networks*, vol. 3, pp. 139–145, Jan. 1992.

[22] K. C. Lee and Y. Takefuji, "A generalized maximum neural network for the module orientation problem," *Int. J. Electron.*, vol. 72, no. 3, pp. 331–355, 1992.

[23] N. Funabiki, Y. Takefuji, and K. C. Lee, "A neural network model for a near-maximum clique," *J. Parallel and Distributed Computing*, vol. 14, pp. 340–344, 1992.

[24] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 18th Design Automation Wkshp.*, 1971, pp. 155–169.