A PLACEMENT ALGORITHM FOR ARRAY PROCESSORS[+]

Dah-juh Chyan*
and
Melvin A. Breuer

DIGITAL INTEGRATED SYSTEMS CENTER REPORT
DISC/82-7

DEPARTMENT OF ELECTRICAL ENGINEERING-SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90089-0781

DECEMBER 1982

*Dah-juh Chyan is a graduate student at the University of Southern California and is also employed at Xerox Corp., El Segundo, CA 90245.

[+]This work was supported in part by the National Science Foundation under Grant ECS-8005957 and by the Xerox Corp.

A Placement Algorithm for Array Processors

ABSTRACT

In this report a concurrent pairwise exchange placement algorithm executing on an array processor is presented. Two force functions and their effects are discussed. The oscillation phenomenon caused by the concurrent computation is investigated and some solutions are suggested. A design for the array processor is presented along with a complexity analysis which indicates that this algorithm is $O(N^2)$ faster than a conventional sequential placement algorithm.

KeyWords

Placement, array-processor, pairwise exchange, oscillation, VLSI

1. Introduction

Conventional placement algorithms, such as the interactive placement-improvement algorithms [1] or the min-cut placement algorithm[2], were designed to run sequentially on a single CPU. In the past ten years, however, thanks to advanced VLSI technology, computing power has become so cheap that it is feasible to employ specialized processing units such as array processors to perform such tasks. For example, low cost array processors have been successfully utilized in signal processing applications[3] as well as design automation[4]. In this paper we present a pairwise exchange placement algorithm designed to run on array processors. This algorithm takes advantage of the concurrent processing power provided by an array of special purpose processors. Each processor consists of a simple ALU, a local memory and a special control unit. Due to its simplicity, many processors may be put on a single VLSI chip, thus reducing the implementation cost. A program which simulates this concurrent placement machine has been coded in PASCAL, and experimental results have been obtained. An oscillation problem was discovered as a result of these simulations. Several types of oscillation have been identified, and methods to reduce this phenomenon are suggested. Even though oscillation is undesirable because it makes it more difficult for the placement algorithm to converge, it may contribute, from a statistical point of view, to a better placement.

2. The Concurrent Placement Algorithm

Consider the two dimensional array of processors shown in Fig. 1. Each processor is capable of performing some global communication through a serial line. Adjacent pairs of processors are connected via a local bus which can be used to exchange data at a high bandwidth. Each processor has an ALU, a local memory and some control logic. There is also a global control unit which is capable of initializing all the processors and is responsible for controlling the global communication.

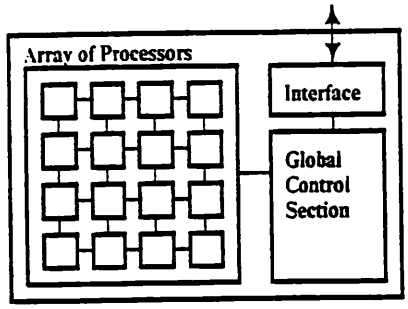


Fig. 1 Array of processors

We consider the following placement problem:

Given a set of modules (either function blocks on a VLSI chip or packaged components) and a net list

which describes the interconnections among the modules as well as a regular array of positions for modules lying in rows and columns, find an assignment of modules to positions so that the resulting placement has a near-optimal minimal total wire length. The following is a concurrent placement algorithm for solving this problem:

1. Choose an arbitrary or predefined initial placement which assigns each module to one processor. Each processor carries the ID number of its assigned module in addition to the ID numbers and locations of modules connected to it.
2. FOR Iteration = 1 to maximum-Iteration DO
   BEGIN
3. FOR Cycle = 1 to 4 DO
   BEGIN
4. **Form-Interchange-Pair (Cycle);** --concurrent operation
5. **Interchange-decision-making;** --concurrent operation
6. **Interchange-data-between-pairs;** --concurrent operation
7. **Broadcast-new-location;** --sequential operation
   END;
   IF stabilized THEN EXITLOOP; --global decision
   END;
8. Report final placement: The computation stops and the last placement is reported through the global control unit.

The final result usually is a relative placement because the physical module size is neglected. A post-placement process which takes the physical size into account may be desirable. This algorithm will stop if there are no more changes made in one iteration(the placement has stabilized); otherwise it runs for a maximum number of iterations. Steps 1, 2, 3 are self-explanatory. In the following, the actions in steps 4, 5, 6, 7 and 8 are described.

step 4: Form-Interchange-Pair (Cycle)

Formation of pairs is determined by the value of 'Cycle.' The pair will be selected in the manner similar to a sorting algorithm, known as Alternate Sorting. Since we are dealing with a two-dimensional array of processors, the interchange will be performed alternately between X-dimension neighbors and then Y-dimension neighbors. An example of the formation of pairs is depicted in Fig.2.

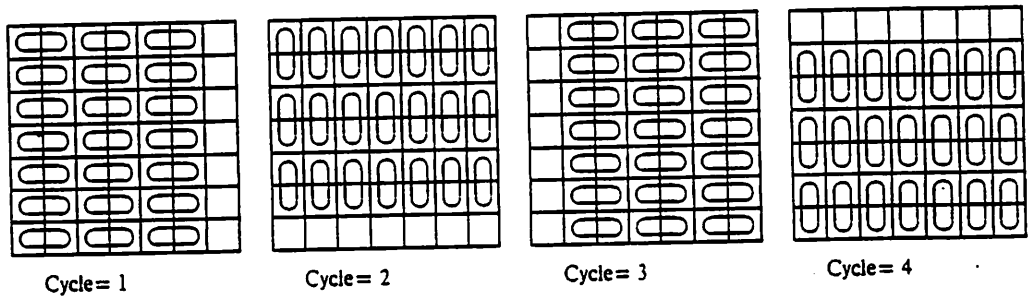


Fig.2. Formation of interchange pairs

step 5: Interchange-decision-making

The same computation is performed in all pairs simultaneously aimed at determining if the total wire length can be reduced by interchanging the positions of the modules in the pair. For any pair, if the gain is greater than a certain threshold, an interchange will be carried out in the next step. The function used in determining the gains is called the net-gain function. Several useful net-gain functions, each providing its unique features, will be discussed in a later section.

step 6: Interchange-data-between-pairs

In this step those modules which were determined to be interchanged in previous steps are actually interchanged. The information which needs be interchanged includes the module ID numbers and the connectivity information. The data being exchanged is sent through a local bus which connects each pair of processors.

Up to this step all the actions take place simultaneously among all interchange pairs. The next step is the only action that has to be performed sequentially through all the pairs.

step 7: Broadcast-new-location

Because a complete graph communication network is very costly in terms of VLSI real estate and logic, it was decided that a global bus be used. All processors can read from the bus simultaneously, but only one processor is allowed to write onto the bus at any one time. Accordingly, the interchange pairs have to sequentially broadcast their new locations (or simply the movement) to all other processors. The order of broadcasting can be simply a row-major ordering. All processors update their own data upon receiving the broadcasted information. Clever hardware design may be employed to shorten the broadcasting time. We will show an example in section 5.

3. The net-gain function

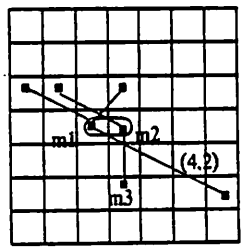
The net-gain function for each pair is actually the combined result of two values. Each module in the pair contributes one value that is a function of the relative locations between this module and its connected modules. If the value of this function is positive, it will cause a reduction in total wire length if we move this module to the opposite side of this pair. When adding the two values, if the resulting sum is positive, an interchange should be performed; other, one should not take place. We will call the relative location function a *'force function'* for reasons that will become clear later.

The force function is an important factor in determining the optimality of a placement. Unfortunately, there is no single function which is superior to all others. In the following, we will analyze two types of force functions which have produced good results.

3.1 Constant force function

A constant force function is a function in which each connected module pair contributes a constant force of unit value in the x or y direction so as to pull each module toward the other. The force function is derived based upon the positions of a module before and after an interchange. Consider a module pair interchange in the x-direction where a module **m** moves from the right to the left. All modules to which **m** is connected which lie to the left (right) of the module in question contributes a force of +1 (-1). When **m** moves from left to right the sign of the force is reversed. A module which is in the same column as **m** always contributes a force of -1. The force is independent of the distance between the modules.

Assume we have the placement situation shown in Fig. 3. The force function in the x-direction is 1 for **m1** and 0 for **m2**. The net-gain for this pair will be 1. Note that **m3** contributes -1 to the force for **m2**. Also note that when a horizontal interchange pair is considered, the relative positions in the vertical direction are irrelevant. A similar computation is applied to a vertical direction interchange pair.



(4,2) means 4 units long in x-direction
2 units long in y-direction

Fig. 3 Example of forces

3.2 Linear force function

A linear force function is one in which each connected module contributes an attraction force between the two modules which is connected proportional to the distance between the modules, i.e, it is a Hooke's Law system. However, unlike the constant force function, using the linear force function will not reduce the total wire length $\Sigma(d_i)$ in a strict mathematical sense. Instead, it will reduce the sum of the squares of all wires, i.e., $\Sigma(d_i)^2$. Experimentally, however, the linear force function still leads to a small total wire length. We will again use Fig. 3 to illustrate the linear force function, but this time the distance in the horizontal direction is taken into account. The net force function for **m1** will have a value 3, and the net force function for **m2** will have a value 1 (note that **m3** being in the same column as **m2** contributes -1 to the function in spite of x-direction distance is 0.) The net-gain for this pair will be 4. Note that when a horizontal interchange pair is considered, the distance in the vertical direction is not used to calculate the gain. The advantages and disadvantages of the constant force function and the linear force function are discussed next.

3.3 Experimental results for different Force Functions

The two functions described have been modeled in a simulation program. In the tested cases, the linear force function was found to lead to a smaller total length. It was also found that the linear force function lead to faster convergence. We believe that local optimization contributes to this result. Local optimization is a more serious problem for a concurrent placement algorithms than it is for a conventional placement algorithms because the module has to travel through every location along the path to reach its final place, thus increasing the probability of getting trapped in a local optimal position along the path. Since the linear force function is more sensitive to long wires, it did better in avoiding a local optimal placement, and consequently has a better chance of obtaining a better final result. When the constant force function is used, it is quite possible that some modules get trapped in local optimal positions. In the best case, a module may be able to get out of a trap but has delayed the process of convergence; in the worst case, a module may never leave a locally optimal position. It appears that the linear force function is much better than the constant force function, but before we jump to any conclusions, we have one more factor to study, namely oscillation. In the following section, we investigate the oscillation phenomenon and discuss its implication on the two force functions .

4. Oscillation

In this section we will discuss a unique property of concurrent placement algorithms, namely oscillation. Oscillation refers to the phenomenon that a number of modules perform endless exchange of positions. The reason for oscillation is that erroneous decisions are possible when numerous pairs of modules are considered simultaneously for exchange. Because the algorithm can not predict the movements of other modules, it can only assume that the other modules will not change positions. Unfortunately, this assumption is not always true and a decision to exchange positions may turn out to be counterproductive, i.e., instead of reducing the total wire length, it may increase it. Such an interchange is referred to as a mal-interchange. However, a mal-interchange is only a necessary condition for oscillation; to cause an oscillation also requires some patterns of placements to form and appear repeatedly. In the following, we show some simple patterns which cause oscillations. More complex oscillation patterns are also possible, but are rarely observed.

4.1 Patterns of Oscillation

We will show two simple oscillation patterns. The first one will cause oscillations for both the constant force function and the linear force function. The second example will cause oscillation only when the linear force function is used.

Oscillation pattern I: This kind of oscillation occurs most often. It happens when two pairs of modules are connected in the manner shown in Fig. 4.

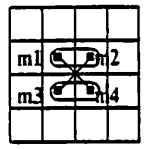


Fig. 4 Oscillation pattern I

The force pulling **m1** to the right is +1, and the force pulling **m2** to the left is also +1, therefore, **m1** and **m2** decide to change positions. So do **m3** and **m4**. An endless exchange thus occurs.

Oscillation pattern II: This type of oscillation only occurs for linear (or higher order) force function. It happens when two pairs of modules are connected in the manner shown in Fig. 5.

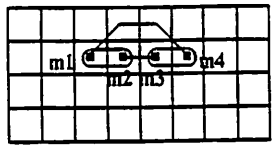


Fig. 5 Oscillation pattern II

The force to pull **m1** to the right is +3, the force to pull **m2** to the left is -1, i.e., **m2** resists the movement to the left; unfortunately, the sum is +2 and therefore, **m1** and **m2** change positions. So do **m3** and **m4**; an endless exchange thus occurs. In general, the linear force function has a higher probability of causing oscillation.

4.2 Methods of reducing oscillation

There are many ways to reduce the oscillation phenomenon. We present two methods here, one of which interlaces interchanges, the second one employs a threshold value for the total force.

(1) Interlacing the exchanging pairs

If one forms the pairs in the interlaced manner shown in Fig. 6a, it can be shown that the occurance of pattern I oscillation will be reduced by at least one half when the constant force function is used.

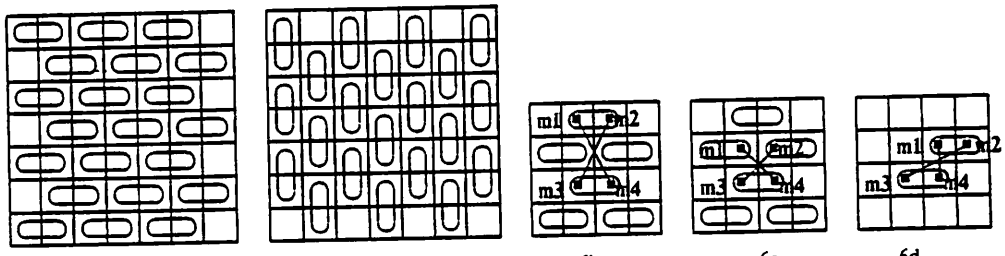


Fig. 6 Interlacing the exchange pair

The patterns which may cause an oscillation are shown in Fig. 6b, 6c and 6d. It is easy to verify that the patterns in Fig. 6c and 6d do not cause oscillations when a constant force function is used. Therefore, the chance of pairs forming an oscillation pattern as shown in Fig. 6b is reduced to only one half of that in non-interlaced case. The reduction in the probability of oscillation should be even greater than one half when we consider that oscillation happens mostly between pairs lying in adjacent rows (or columns.) Actually, it was very surprising to find that in many test cases the interlaced placement scheme does terminate all the oscillations. The interlace scheme not only terminates oscillation, but also modifies the placement shown in Fig. 7a to that in Fig. 7b.

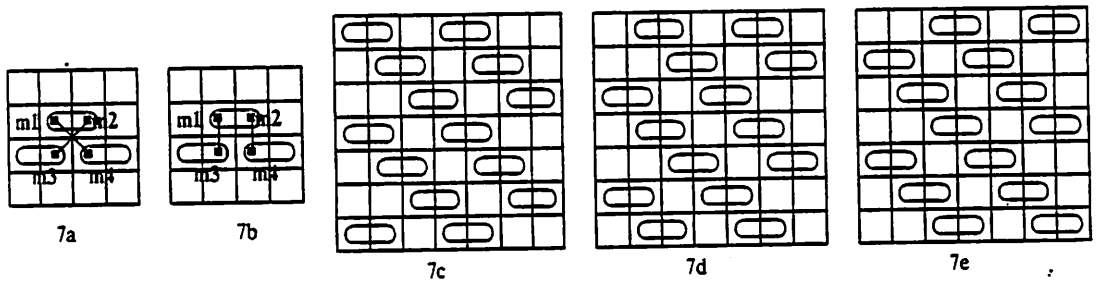


Fig. 7

Other methods based on the interlaced concept can also be devised. Three way interlacing is an example. This is shown in Fig. 7c, 7d and 7e. It further reduces the probability of oscillation, but at the cost of slowing down the placement process.

(2) Employing a non-zero threshold value

In section 3 we chose the threshold to be 0, i.e., if the net-gain is greater than 0, we allow the exchange take place. This decision was made because when the force exceeds 0, it indicates a possible reduction in the potential function. However, we showed in section 4.1 that both pattern I and pattern II with net-gain functions equal to 2 do not lead to a reduction in total wire length. This fact suggests a change in the threshold value may be useful. For example, we may choose the threshold to be 2. However, this still does not guarantee the complete elimination of oscillation. In Fig. 8 we show an example where a threshold equal to 2 still results in oscillation.

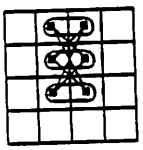


Fig. 8 Oscillation occurs for threshold less than 4

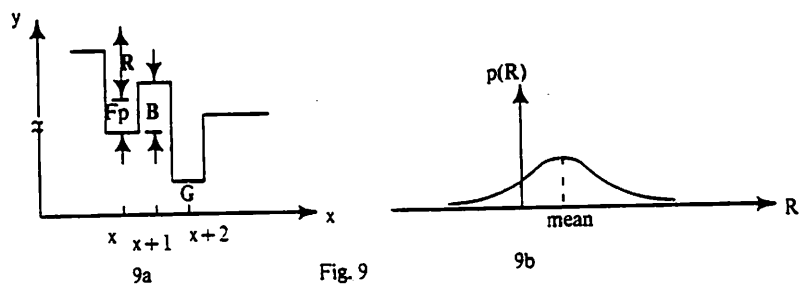
One approach to terminate oscillation is to increase the threshold value gradually after the placement process has proceeded for some time. The disadvantage with this scheme is that we may sacrifice some

real improvement in the objective function. When to start increasing the threshold value thus becomes a crucial decision. This method is an effective way to terminate the oscillation despite some drawbacks.

4.3 Effects of oscillation

Oscillation caused by mal-interchanges is not a desirable phenomenon because it creates some difficulty in deciding when to stop the placement process. But it does not mean that mal-interchanges only cause negative effects. Experiment results actually showed that a placement scheme with more oscillations or mal-interchanges usually lead to a lower value of total wire length. In one experiment, we modified the concurrent placement algorithm so that we formed the pairs in the same manner as in Fig. 2, but only one pair of modules is allowed to make a decision and carry out an exchange at a time, any new location is broadcast and updated immediately, i.e., we solved the test placement problem in a totally sequential manner. No oscillation is possible and the placement process terminates relatively fast, however, the result is very unsatisfactory, i.e., the resultant total wire legth is quite high in comparison to the result of an concurrent placement algorithm. One other experiment used the interlace placement scheme. The result is not quite as good as that for the non-interlaced concurrent technique. To explain why the non-interlaced concurrent algorithm outperforms the other techniques, we offer the following reasoning . Our force F is the (directed) sum of individual forces corresponding to each connected module. The individual forces are calculated under the assumption that the connected modules do not change their locations. However, we are actually uncertain about this. This uncertainty introduces a random factor into our force. It is reasonable to say that our calculated force F_c is the sum of a force F_p and a random variable R, i.e., $F_c = F_p + R$, where F_p is the force we would obtain if we could predict the resultant placement of this cycle. If we always use F_p, no mal-interchanges can happen. But because of the random factor R, we sometimes get mal-interchanges. Since we do not have much information about this random variable, we assume it is a normal distribution function and its mean value may be obtained from experiments. It is possible that this random factor R may help overcome the local optimization barrier. Even though we could not prove our reasoning rigorously, we believe the randomness concept should be useful to some extent. Fig. 9a is an abstract depiction of this idea. Let the Y-axis represent the total wire length of the system and the X-axis the relative location of module m. Assume m is originally at location x. If the force F_p is not enough to overcome the barrier B, the random variable R added to F_p sometimes makes it possible for m to move away from a local optimal position so that module m has a chance to reach a better position, e.g., point G. Let the probability of m leaving the trap be equal to P, and the p(R) in Fig. 9b be the probability density function of random variable R. Then

$$P = \int_{B-Fp}^{\infty} p(R)\, dR$$



9a Fig. 9 9b

5. *One Sample Design of a Concurrent Placement Machine*

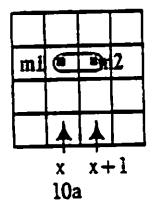
In this section we present a sample design of a concurrent placement machine. Our machine is capable of pipelining the broadcasting process while part of decision making process is in progress, i.e., most computations are done immediately after the broadcast information is received. Therefore, most arithmetic operations are being performed while the broadcasting process is still undergoing. This machine will support both the constant force function and the linear force function. We assume that it is the responsibility of the global control unit to stop the placement process or to terminate oscillation. Before presenting our design we have to analyze the two force functions in order to find out what hardware we will need.

5.1 Analysis of the Computation

In this design we are trying to minimize the computation required to calculate the value of the force function. Therefore, we decided not to recalculate the forces from scratch for every iteration. We will maintain a current force value. This value will be modified by an appropriate amount whenever a relocation of interconnected modules occurs. This approach complicates the hardware design, but reduces the computation time substantially.

(1) Constant Force Function

Assume we have a pair of modules **m1** and **m2** as shown in Fig. 10a. We find that for a constant force function, only the movements of modules in columns x and x+1 contribute to modify the forces on **m1** and **m2**. This follows because as long as the modules stay outside column x and x+1, their contributions to the resultant forces on **m1** and **m2** remain unchanged. Fig. 10b is a transfer table showing the movements of modules that are connected to **m1**. n1 is the number of modules that stay in column x. n2 is the number of modules originally in column x and found to be in x+1 after the broadcasting. Similar definitions apply to n3 and n4.



10a

original \ after	x	x+1
x	n1	n2
x+1	n3	n4

10b

Fig. 10

The force contributed by modules connected to **m1** can be divided into two parts: F1, the force contributed by the modules which are not in the same column as **m1**, and F2, the force due to those modules in the same column of **m1**. Thus

$$F1 = \left(\sum_{x(j)>x} 1 + \sum_{x(j)<x} (-1)\right)$$

$$F2 = \sum_{x(j)=x} 1$$

where j is any module connected to **m1**, and x(j) is the x- location of module j.

First, let us consider the case where **m1** does not change its location in a cycle. It is obvious that after the broadcasting, F1 should be updated to F1', where

$$F1' = F1 + n2 - n3$$

Similarly, $F2' = F2 - n2 + n3$

Let $E1 = F1 - F2$, $E2 = -F1 - F2$

The value of the force function is equal to E1 if **m1** is on the left side of the module pair; the force is equal to E2 if **m1** is on the right side. After each cycle

$$E1' = F1' - F2' = F1 + n2 - n3 - (F2 - n2 + n3) = E1 + 2(n2 - n3) = E1 + 2(n1 + n2) - 2(n1 + n3)$$

Notice that $n1 + n2$ is the number of modules originally in column x; $n1 + n3$ is the number of modules in column x after the cycle.

$$E2' = -F1' - F2' = -(F1 + n2 - n3) - (F2 - n2 + n3) = -F1 - F2 = E2$$

We have designed our machine so that prior to a broadcasting process, we add twice the number of modules that are in column x to E1. During the broadcasting process, **m1** will update the locations of its connected modules. After the broadcasting process, we subtract twice the number of modules that are now in x. Thus we obtain a new value for E1. E2 will remain unchanged in this case. Two similar terms are used when processing the y-direction exchange.

Now we consider the case where **m1** changes position. Expressions for F1', F2' are now different.

$$F1' = F1 - n1 - 2n3 - n4$$

$$F2' = F2 - n1 + n4$$

and $E1' = F1' - F2' = F1 - F2 - 2(n3 + n4) = E1 - 2(n3 + n4)$

$$E2' = -F1' - F2' = -F1 - F2 + 2(n1 + n3) = E2 + 2(n1 + n3)$$

Notice that $n3 + n4$ equals the number of modules originally in column $x + 1$, and $n1 + n3$ equals the number of modules eventually in column x. Therefore, before broadcasting, we subtract from E1 twice the number of modules in column $x + 1$. After the broadcasting, we add to E2 twice the number of modules in column x.

Note that for the constant force function, we only need to know the number of modules in column x or $x + 1$. The arithmetic operations are performed immediately before and after the broadcasting process. This is not the case for the linear force function. In the following, we omit the lengthy analysis and briefly describe what should be done for the linear force function.

(2) Linear force function

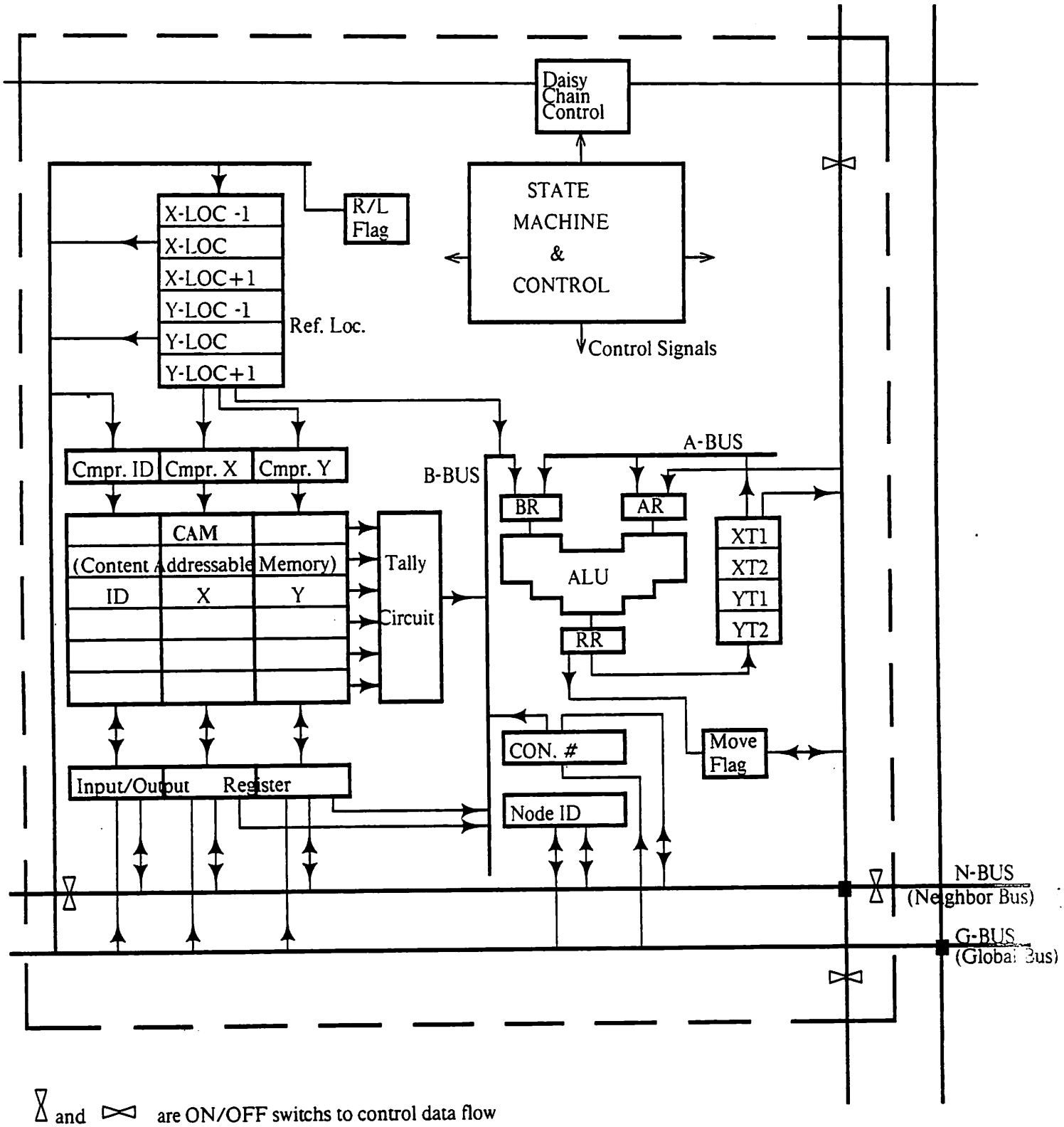
The difference between (2) and (1) is that not only must the movements of modules in columns x and x+1 be taken into account, all other connected modules which move also participate in generating the new force. We noticed that we have to add 1 to F1 when a connected module moves to the right, and subtract F1 by 1 when a connected module moves to the left. Let NT be the total number of modules connected to **m1**. We subtract NT from the force if **m1** moves to the right. Some other operations will involve n1+n2, n1+n3 and n2+n4. After examining the necessary operations needed to calculate the force value, we note that basically we need to maintain two values for both the x- and y- dimensions. Some arithmetic calculations are involved. In addition to that, we need the capability to know the number of connected modules in a given column or row.

5.2 The Actual Design

Fig. 11 is a detailed block diagram of one processor. To speed up the search and count operation we decided to use a content addressable memory(CAM) to hold the ID and X-, Y- locations of the connected modules, and to use a tally circuit to determine the number of modules that match a certain pattern, e.g., a coordinate. The pattern to be matched is stored in either Cmpr-ID, Cmpr-X or Cmpr-Y. We use X-LOC-1, ... to store the relavent coordinates for the processors. We use registers XT1, XT2, YT1 and YT2 to maintain the four values mentioned in section 5.1. There are two internal buses- A-BUS and B-BUS. They are used to transfer the operands to the ALU. N-BUS(Neighbor Bus) is a bus that connects a processor to all its adjacent processors. Some switching circuitry is needed to control the data flow. The G-BUS is a global bus. During the system initialization phase all initial data is transmitted through the G-BUS. The broadcast data is also sent out through the G-BUS. The data being broadcast includes (1) the ID of the module doing the broadcasting, and (2) its new X- or Y-location. Note that a module that does not change locations does not broadcast any data. This is achieved by using a daisy chain control circuit. This circuit allows modules that do not change locations to simply pass a token to the next one in the daisy chain. If a module does change its location, it holds the token until it finishes its broadcasting. For each cycle the global control generates one token and passes it to first processor in the daisy chain. By so doing, the time for the broadcasting process is proportional to the number of movements in that cycle. This number is expected to decrease as the placement improves. The daisy chain control also assures that at most one processor is putting data on the global bus. Global control circuit will make sure that all processors are running synchronously.

5.3 The Time Complexity

In this section we compare the time complexity of a conventional placement algorithm to our concurrent placement algorithm. We choose the force-directed interchange algorithm presented in [1] (page 259) because it is very similar to our algorithm. Basically, the concurrent machine can make up



and are ON/OFF switchs to control data flow

Fig. 11 BLOCK DIAGRAM OF ONE PROCESSOR

to $N^2/2$ decisions in each cycle, assuming we have an NxN array of processors, while a sequential machine can only make one decision in one cycle. It is thus fair to say that if a sequential machine takes P cycles to complete a placement, the concurrent machine will take $P/(N^2/2)$ cycles to finish. Here we ignore the effect caused by oscillation. Let the total time for a sequential machine to complete a job be Ts, and the total time for concurrent machine be Tc. Then

$$Ts = P * Cs$$

and $$Tc = P/(N^2/2) * Cc \quad (* \text{ indicates multiplication})$$

where Cs is the time for the sequential machine to finish one cycle, and Cc is the time for the concurrent machine to finish one cycle.

Now we have to compare Cs and Cc. As mentioned in [1] the sequential algorithm has to maintain a Q-list and keep updating it. It takes at least $O(N^2)$ time to do so, i.e., $Cs = O(N^2)$. Note that the total number of modules is proportional to N^2. It is more difficult to calculate Cc because Cc is equal to the sum of exchange time (Tx), the broadcasting time (Tb) and a small constant. Let C_{max} be the maximum number of modules that can be connected to any given module. C_{max} is $O(N^2)$. The exchange time is $O(C_{max})$, and therefore Tx is $O(N^2)$. Broadcasting time varies from cycle to cycle. Since each exchanged module needs a constant time to do the broadcasting, the broadcasting time is proportional to the number of exchanges. Initially, there will be many exchanges taking place. As the placement gradually settles down, the number of exchanges becomes smaller and smaller. Therefore, we can only estimate the average number of exchanges in one cycle. Let Te be the total number of exchanges in the entire placement process. Then the average number of exchange per cycle is $Ve = Te/(2P/N^2)$. We can estimate the total number of exchanges by using the following argument: From the experimental results, the number of exchanges in the i-th cycle is approximately $K*(1/i)$, for $i = 1, 2, 3, \ldots, 2P/N^2$, where K equals the number of movement in the first cycle. K is at most $N^2/2$. The total number of exchanges Te is then given by the expression

$$Te = \sum_i K*(1/i) \simeq K*\int (1/i)\, di \simeq K* \text{Log}(2P/N^2) \simeq (N^2/2)*\text{Log}(2P/N^2)$$

where i ranges from 1 to $2P/N^2$

Therefore $Ve \simeq ((N^2/2)*\text{Log}(2P/N^2))/(2P/N^2) = N^4*\text{Log}(2P/N^2)/4P$

From [1] we know that the complexity of P is at least $O(N^4)$ because it is proportional to the square of the total number of modules, which is $O(N^2)$. Therefore we obtain $Ve = O(\text{Log } N)$. It is obvious that Cc is dominated by the exchange time Tx which is $O(N^2)$. Cc is $O(N^2)$. Since Cs and Cc are of the same order of complexity, we conclude that the concurrent placement scheme can speed up the placement process by a factor of $O(N^2)$.

6. *Conclusion*

In this paper we presented a concurrent placement algorithm and an array architecture which can implement this algorithm. We discussed the oscillation problem and its effects. We compared the time complexity of the concurrent algorithm with a conventional (sequential) placement algorithm. The

conclusion is that we can speed up the computation by a factor of $O(N^2)$ while retaining the same if not better quality of results.

Several problems have not been addressed in this paper. The first deals with how to deal with placement problems which are larger than the concurrent machine can handle. For this case a divide and conquer approach is being investigated. A second problem deals with the expandability of our machine. If our machine is implemented on a chip, it would be desirable to interconnect these chips so that larger problems can be processed. Finally it is possible to modify the computational formula to deal with the problem of "weighted" signal nets. In conclusion, we believe that the use of array processors is a powerful and useful tool to be used in solving design automation problems.

REFERENCES

1. M. Hanan and J. M. Kurtzberg, "Placement Techniques", Chap. 5 in *Design Automation of Digital Systems: Theory and Techniques, Vol. 1*, Editor, M. A. Breuer, Prentice-Hall, 1972. pp. 213-282.
2. M. A. Breuer, "Min-Cut Placement", *J. of Design Automation and Fault Tolerant Computing Vol. 1*, October 1977, pp. 343-382.
3. H. T. Kung and C. Leiserson "Algorithms for VLSI processor Arrays", Sec 3, Chap. 8 in *Introduction to VLSI systems* by C. Mead and L. Conway, Addison-Wesley, 1980 pp. 271-291.
4. M. A. Breuer and K. Shamsa, "A Hardware Router", *J. of Digital System, Vol. 4*, Winter 1980, pp. 393-408.