A Parallel Architecture for Discrete Relaxation Algorithm

JUN GU, STUDENT MEMBER, IEEE, WEI WANG, AND THOMAS C. HENDERSON, SENIOR MEMBER, IEEE

Abstract—**Discrete relaxation techniques have proven useful in solving a wide range of problems in digital signal and digital image processing, artificial intelligence, operations research, and machine vision. Much work has been devoted to finding efficient hardware architectures. This paper shows that a conventional hardware design for a Discrete Relaxation Algorithm (DRA) suffers from $O(n^2m^3)$ time complexity and $O(n^2m^2)$ space complexity. By reformulating DRA into a parallel computational tree and using a multiple tree-root pipelining scheme, time complexity is reduced to $O(nm)$, while the space complexity is reduced by a factor of 2. For certain relaxation processing, the space complexity can even be decreased to $O(nm)$. Furthermore, a technique for dynamic configuring an architectural wavefront is used which leads to an $O(n)$ time highly concurrent DRA3 architecture.**

Index Terms—**Algorithm-configured dynamic architectural wavefront system, associative circular pipelining, Discrete Relaxation Algorithm (DRA), interleaved processing, multiprocessor architecture, recursive systolic computation, VLSI.**

I. Introduction

DISCRETE relaxation is a very general computational technique for a wide range of theoretical and engineering problems. Since its invention it has demonstrated powerful and extensive applications in many areas. Some of them are listed below:

1) Digital Signal Processing: for digital signal and digital image processing, particularly in the restoration and identification of moving objects from ambiguous environment;

2) Artificial Intelligence: propagating numeric constraints, doing heuristic optimal search, finding *N*-ary relations, determining satisfiability of propositional logic statements, and theorem proving, etc.;

3) Computer Vision: dealing with the problems such as graph and subgraph homomorphisms, graph coloring problems, for line finding, stereopsis, line-labeling, and semantics-based region growing;

4) Robotics: for solving vision problems, packing problem, and space planning problem, etc.;

5) Database Operation: for the relational homomorphism problem, database consistency-maintenance, query-answering, and redundancy-checking, etc.;

6) VLSI Engineering: for developing various hardware accelerators for design automation, such as for building a relaxation-based simulator and tester for logic simulation, fault checking, and testing.

For a review of the numerous applications of relaxation processes see [1]–[3], [5], [7]. A relatively comprehensive bibliography can be found in [10].

Since the introduction of the discrete relaxation technique in the mid-1970's, many DRA applications have been found, and much work has been done to develop optimal DRA algorithms [5], [6]. On the other hand, people have also tried to build efficient hardware architectures to support real-time DRA processing. Unfortunately, due to high computational cost including space complexity, time complexity, and data communication cost, a conventional hardware architecture implementation is not feasible [8]. Current research in this aspect is blocked, and has appeared only in a virtual software simulator format [4].

The research described in this paper concerns the hardware implementation of discrete relaxation architecture. In Section II we briefly describe the Discrete Relaxation Algorithm, and give an example for solving the region coloring problem. Then we define the DRA hardware implementation problem in Section III. Complexity analyses and our first design of the DRA1 chip are presented in Section IV. In Section V, we concentrate on the design of the parallel DRA computational tree and a tree-root pipelining scheme, and its corresponding $O(nm)$ time SIMD multiprocessor architecture DRA2. Finally, it is shown that we have adopted a dynamically configurable, highly parallel routing scheme on the DRA3 switch lattice, so that an $O(n)$ time algorithm-configured architectural wavefront system, DRA3, is found.

These three architectures are designed for the same 8-object 8-label DRA problems. The first two systems are implemented using PPL (Path Programmable Logic) [13], [14] at the University of Utah. The DRA1 chip requires two 4K memory blocks and maximum execution time of over an hour in a 3 μ NMOS process, which makes such a hardware implementation infeasible [8]. The DRA2 design eliminates excessive memory requirements and performs the DRA computation in microseconds, at the worst case in milliseconds. This chip was fabricated using a 3 μ NMOS process by MOSIS [9]. DRA3 will be fabricated using a 1.0 μ GaAs technology. The clock speed of

Manuscript received July 25, 1986; revised January 20, 1987. This work was supported in part by the National Science Foundation under Grants MCS-82-21750, DCR-85-06393, and DMC-85-02115; and in part by a University of Utah Research Fellowship and an ACM/IEEE Scholarship.

J. Gu and T. C. Henderson are with the Department of Computer Science, University of Utah, Salt Lake City, UT 84112.

W. Wang is with the Department of Electrical Engineering, University of Utah, Salt Lake City, UT 84112.

IEEE Log Number 8716850.

DRA3 is expected to be over 500 MHz [23]. In this paper, we try to describe our research ideas and the architectural concepts for these DRA architectures as concisely as possible. For detailed references see [8]–[12], [23].

II. The Discrete Relaxation Algorithm (DRA)

A. Boolean Formulation of Discrete Relaxation Algorithm

Instead of seeking a real number solution in a numerical relaxation situation [1], the solution to be found in the discrete relaxation case involves the assignment of set of *labels* at each unknown such that some constraint relation among the labels is satisfied by neighboring unknowns [5], [6]. Whereas the unknowns in numerical relaxation take on real number values, the unknowns in a *labeling problem* take on a Boolean vector value with each element in the vector corresponding to a possible label.

The generalized problem involves a set of unknowns which usually represents a set of objects to be given names, a set of labels which are the possible names for the unknown, and a *compatibility model* containing ordered groups of units which mutually constrain one another and ordered groups of unit-label pairs which are compatible. The compatibility model is sometimes called a *world model*. This model tells us which objects mutually constrain one another and which labelings are permitted or legal for those objects which do constrain one another. The problem is to find a label for each object such that the resulting set of object-label pairs is consistent with the constraints of the world model.

Boolean vector operations are denoted by $'$, $\times$, t, $*$, $+$ and $\cdot$ which represent complementation, vector multiplication, transpose, Boolean "and," Boolean "or," and Boolean vector dot product, respectively. Our basic definitions which are used in formulating DRA are given in the following.

Definition 1: Let $U = \{u_1, \cdots, u_n\}$ be the set of unknowns, and $\Lambda = \{\lambda_1, \cdots, \lambda_m\}$ be the set of possible labels. Then

1) C is an m by m *compatibility matrix for label pairs*, where $C(i, j) = 1$ if λ_i is compatible with λ_j; 0 otherwise.

2) $\Lambda_i = (l_1, \cdots, l_m)^t$ is the column vector describing the set of labels (i.e., zero or one) possible for u_i, where $l_j = 1$ if λ_j is compatible with u_i; 0 otherwise.

3) $\Lambda_{ij} = (\Lambda_i \times \Lambda_j^t) * ((\text{Nei}(i, j)'E) + C)$ is an m by m *compatibility matrix for* u_i and u_j, where E is the m by m matrix for all 1's, and $\text{Nei}(i, j) = 1$ if u_i neighbors u_j; 0 otherwise. We use Λ_k to denote the kth row of Λ_{ij}.

Definition 2: A *labeling* is a vector $L = (L_1, \cdots, L_n)^t$, where $L_i = (l_{i1}, \cdots, l_{im})$ in Λ_i is a Boolean vector with $l_{ij} = 1$ if label λ_j is a possible label for object u_i; 0 otherwise.

Definition 3: A labeling is *consistent* if for every i and k,

$$l_{ik} \le \prod_{j=1}^{n} \left[\sum_{p=1}^{m} (l_{ik} * l_{jp} * \Lambda_{ij}(k, p)) \right]. \tag{1}$$

Once a formal definition of local consistency such as (1) has been given, it is easy to see that a situation very similar to classical relaxation now holds. However, instead of having to manipulate (1) into a form amenable to iterative solution, we merely note that (1) can be rewritten

$$l_{ik} \le l_{ik} * \prod_{j=1}^{n} \left[\sum_{p=1}^{m} (l_{jp} * \Lambda_{ij}(k, p)) \right] \tag{2}$$

for every i and k. Since the l_{ik} on the right-hand side is independent of j and p, it is clear that if (2) does not hold it can be made to hold by setting l_{ik} equal to the value on the right-hand side. This is, in fact, equivalent to discarding label k for u_i if at some neighbor u_j there does not exist a compatible label. If the l_{ik}'s, $k = 1, m$ are now gathered together in vector form:

$$\begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} \le \begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} * \begin{bmatrix} \sum_{p=1}^{m} (l_{1p} * \Lambda_{i1}(1, p)) \\ \sum_{p=1}^{m} (l_{1p} * \Lambda_{i1}(2, p)) \\ \vdots \\ \sum_{p=1}^{m} (l_{1p} * \Lambda_{i1}(m, p)) \end{bmatrix} * \cdots * \begin{bmatrix} \sum_{p=1}^{m} (l_{np} * \Lambda_{in}(1, p)) \\ \sum_{p=1}^{m} (l_{np} * \Lambda_{in}(2, p)) \\ \vdots \\ \sum_{p=1}^{m} (l_{np} * \Lambda_{in}(m, p)) \end{bmatrix} \tag{3}$$

or

$$\begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} \le \begin{bmatrix} l_{i1} \\ l_{i2} \\ \vdots \\ l_{im} \end{bmatrix} * \begin{bmatrix} L_1 * \Lambda_{i1}(1)^t \\ L_1 * \Lambda_{i1}(2)^t \\ \vdots \\ L_1 * \Lambda_{i1}(m)^t \end{bmatrix} * \cdots * \begin{bmatrix} L_n * \Lambda_{1n}(1)^t \\ L_n * \Lambda_{1n}(2)^t \\ \vdots \\ L_n * \Lambda_{1n}(m)^t \end{bmatrix} \tag{4}$$

or

$$L_i \le L_i * \prod_{j=1}^{n} \left(\left\{[L_j] \times [\Lambda_{ij}(1)^t \cdots \Lambda_{ij}(m)^t]\right\}^t\right). \quad (5)$$

Let

$$P = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_n \end{bmatrix}, \quad (6)$$

where the column vector:

$$P_i = \prod_{j=1}^{n} \left(\left\{[L_j] \times [\Lambda_{ij}(1) \cdots \Lambda_{ij}(m)]\right\}^t\right). \quad (7)$$

Gathering together the L_i's, $i = 1, n$, we have

$$L \le L * P. \quad (8)$$

This formulation emphasizes the relation to classical relaxation. The relaxation is achieved by repeating

$$L \Leftarrow L * P \quad (9)$$

until L does not change value.

B. An Example of Region Coloring Problem

Suppose that we are analyzing a picture of a scene, with the aim of describing it, and that we have detected a set of objects $u_1, \cdots, u_n$ in the scene, but have not identified them unambiguously. The relationships that exist among the objects are used to eliminate the ambiguity.

An example for eliminating the ambiguity in a region coloring problem is given here to demonstrate these ideas and computation procedures. For simplicity, consider the case of three regions to be colored red, green or blue with the constraints:

1) Region 1 must be red,
2) Region 3 must be blue, and
3) No two regions may be colored the same color.

Thus, u_i = Region i (for $i = 1, 2, 3$) and:

$$U = \{u_1, u_2, u_3\}$$

$$\Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$$

where λ_1 is red, λ_2 is green, and λ_3 is blue. Since Region 1 must be red, we have:

$$\Lambda_1 = [1\ 0\ 0]^t$$

and since Region 3 must be blue:

$$\Lambda_3 = [0\ 0\ 1]^t.$$

Finally, since there is no restriction on Region 2's color, we have all possibilities:

$$\Lambda_2 = [1\ 1\ 1]^t.$$

Since only similar colors are incompatible, we have:

$$C = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad (10)$$

for different objects, and

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

for the same object.

We see then that C actually depends on the objects under consideration; i.e., technically, we should write C_{ij} which is identified as:

$$C_{ij} = \begin{bmatrix} \mathrm{Nei}(i,j)' & \mathrm{Nei}(i,j) & \mathrm{Nei}(i,j) \\ \mathrm{Nei}(i,j) & \mathrm{Nei}(i,j)' & \mathrm{Nei}(i,j) \\ \mathrm{Nei}(i,j) & \mathrm{Nei}(i,j) & \mathrm{Nei}(i,j)' \end{bmatrix} \quad (12)$$

where

$$\mathrm{Nei}(i,j) = \begin{cases} 0 \text{ if Region } i \text{ does not neighbor Region } j, \\ 1 \text{ if Region } i \text{ does neighbor Region } j. \end{cases}$$

Now we can calculate Λ_{ij} as:

$$\Lambda_{11} = ([1\ 0\ 0]^t \times [1\ 0\ 0]) * ((0'E) + C)$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (13)$$

$$\Lambda_{12} = ([1\ 0\ 0]^t \times [1\ 1\ 1]) * ((1'E) + C)$$
$$= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (14)$$

$$\Lambda_{13} = ([1\ 0\ 0]^t \times [0\ 0\ 1]) * ((1'E) + C)$$
$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (15)$$

$$\Lambda_{21} = ([1\ 1\ 1]^t \times [1\ 0\ 0]) * ((1'E) + C)$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (16)$$

$$\Lambda_{22} = ([1\ 1\ 1]^t \times [1\ 1\ 1]) * ((0'E) + C)$$
$$= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (17)$$

$$\Lambda_{23} = ([1\ 1\ 1]' \times [0\ 0\ 1]) * ((1'E) + C)$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad (18)$$

$$\Lambda_{31} = ([0\ 0\ 1]' \times [1\ 0\ 0]) * ((1'E) + C)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad (19)$$

$$\Lambda_{32} = ([0\ 0\ 1]' \times [1\ 1\ 1]) * ((1'E) + C)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \qquad (20)$$

$$\Lambda_{33} = ([0\ 0\ 1]' \times [0\ 0\ 1]) * ((0'E) + C)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \qquad (21)$$

The (p, q) entry of Λ_{ij} tells if λ_p at object i is compatible with λ_q at object j. For example, Λ_{11} reveals that only λ_1 is compatible with λ_1 at object 1; i.e., that Region 1 must be colored red.

Finally we continue the iteration process. According to (2),

for $i = 1$ and $k = 1$:

$$\begin{aligned} l_{11}^{(n)} \le l_{11}^{(n-1)} &* [l_{11}^{(n-1)} * \Lambda_{11}(1, 1) + l_{12}^{(n-1)} * \Lambda_{11}(1, 2) \\ &+ l_{13}^{(n-1)} * \Lambda_{11}(1, 3)] \\ &* [l_{21}^{(n-1)} * \Lambda_{12}(1, 1) + l_{22}^{(n-1)} * \Lambda_{12}(1, 2) \\ &+ l_{23}^{(n-1)} * \Lambda_{12}(1, 3)] \\ &* [l_{31}^{(n-1)} * \Lambda_{13}(1, 1) + l_{32}^{(n-1)} * \Lambda_{13}(1, 2) \\ &+ l_{33}^{(n-1)} * \Lambda_{13}(1, 3)] \\ 1 \le 1 &* [1 * 1 + 0 * 0 + 0 * 0] \\ &* [0 * 0 + 0 * 0 + 1 * 1] \\ &* [1 * 0 + 1 * 1 + 1 * 1] \\ 1 \le 1 & \\ 1 \le 1 & \text{ which is true.} \end{aligned} \qquad (22)$$

This says that the color red is all right for Region 1. To determine if the color red is possible for Region 2, we must find l_{21}.

For $i = 2$ and $k = 1$:

$$\begin{aligned} l_{21}^{(n)} \le l_{21}^{(n-1)} &* [l_{11}^{(n-1)} * \Lambda_{21}(1, 1) + l_{12}^{(n-1)} * \Lambda_{21}(1, 2) \\ &+ l_{13}^{(n-1)} * \Lambda_{21}(1, 3)] \\ &* [l_{21}^{(n-1)} * \Lambda_{22}(1, 1) + l_{22}^{(n-1)} * \Lambda_{22}(1, 2) \\ &+ l_{23}^{(n-1)} * \Lambda_{22}(1, 3)] \\ &* [l_{31}^{(n-1)} * \Lambda_{23}(1, 1) + l_{32}^{(n-1)} * \Lambda_{23}(1, 2) \\ &+ l_{33}^{(n-1)} * \Lambda_{23}(1, 3)] \\ 1 \le 1 &* [1 * 0 + 0 * 0 + 0 * 0] \\ &* [1 * 1 + 1 * 0 + 1 * 0] \\ &* [0 * 0 + 0 * 0 + 1 * 1] \\ 1 \le 1 &* 0 \\ 1 \le 0 & \text{ which is false.} \end{aligned} \qquad (23)$$

Thus, l_{21} must be set to zero. Likewise, for $i = 2$ and $k = 3$, l_{23} is set to zero, and blue is not a possible label for Region 2. Finally,

for $i = 2$ and $k = 2$:

$$\begin{aligned} l_{22}^{(n)} \le l_{22}^{(n-1)} &* [l_{11}^{(n-1)} * \Lambda_{21}(2, 1) + l_{12}^{(n-1)} * \Lambda_{21}(2, 2) \\ &+ l_{13}^{(n-1)} * \Lambda_{21}(2, 3)] \\ &* [l_{21}^{(n-1)} * \Lambda_{22}(2, 1) + l_{22}^{(n-1)} * \Lambda_{22}(2, 2) \\ &+ l_{23}^{(n-1)} * \Lambda_{22}(2, 3)] \\ &* [l_{31}^{(n-1)} * \Lambda_{23}(2, 1) + l_{32}^{(n-1)} * \Lambda_{23}(2, 2) \\ &+ l_{33}^{(n-1)} * \Lambda_{23}(2, 3)] \\ 1 \le 1 &* [1 * 1 + 0 * 0 + 0 * 0] \\ &* [1 * 0 + 1 * 1 + 1 * 0] \\ &* [0 * 0 + 0 * 0 + 1 * 1] \\ 1 \le 1 & \text{ which is true.} \end{aligned} \qquad (24)$$

We see then that the value of l_{11}, l_{22}, and l_{33} are not affected by the change of l_{21} and l_{23} to zero. In fact, the system of equations stabilizes after the change of l_{21} and l_{23}, and the result is $l_{11} = l_{22} = l_{33} = 1$, while all other hypotheses are zero. Thus, the only consistent labeling is to label Regions 1, 2 and 3 the colors red, green, and blue, respectively.

III. The Hardware Implementation Problems for DRA

A. Assumptions and Statements

To concentrate our attention on tackling the major design complexities without losing generality for developing various DRA architectures, the following assumptions are adopted in our implementations.

Assumption 1: The numbers of objects and labels are assumed to be equal, i.e., $n = m$. Since the extensibility for different numbers of objects and labels has first been considered to be an important factor in system design, DRA2 and DRA3 are able to be specified to an arbitrary number of objects and labels [10].

Assumption 2: For useful DRA applications, we have chosen the minimal value of n and m to be 8. This is reasonable, for example in image analysis applications.

Assumption 3: Contrary to the assumption in [4] that the input data are always ready before the DRA computation begins, we assume that both data load-in time and their hardware support cannot be ignored in a VLSI DRA computation. This assumption provides a greater challenge for an advanced DRA system implementation.

Assumption 4: The time for binary operation is generally derived from the gate delay of the combinational logic circuits, which is, in our DRA designs, always less than one global clock cycle. Similarly, it is common that one clock cycle is usually less than one Read/Write memory cycle. That is, we have that

$$t_{\text{gate delay}} < t_{\text{clock cycle}} \leq t_{\text{memory R / W cycle}}. \tag{25}$$

Whenever we analyze the time complexity, the following three statements hold.

Statement 1: Since the fact that the time complexity for each DRA computing iteration is the same, theoretically, we estimate the time complexity only within one iteration of computation.

Statement 2: As for the worst case of iteration times, which is solely determined by the convergence property of the algorithm and problem feature, we give its estimation in terms of the corresponding VLSI fabrication technology.

Statement 3: We also follow the traditional notation for time complexity analysis for the DRA system, i.e., we describe it according to object number n and labels number m, and not the bit number of input data. For example, following statement 1 that, it takes $O(n)$ time for computing DRA problem in DRA3 architecture, here n is the number of objects, whereas we have actually loaded in n by m bits of input data for each computation.

B. DRA Hardware Implementation Problems

Definition 4: The DRA2 Hardware Implementation problem has been defined as finding the *consistent labeling matrix L*:

$$L = (L_1, L_2, \cdots, L_i, \cdots, L_n)^t = \begin{bmatrix} l_{11}, & l_{12}, & \cdots, & l_{1m} \\ l_{21}, & l_{22}, & \cdots, & l_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1}, & l_{n2}, & \cdots, & l_{nm} \end{bmatrix} \tag{26}$$

for the given *Region Coloring Problem*, provided by the *initial labeling matrix*:

$$\Lambda = (\Lambda_1, \Lambda_2, \cdots, \Lambda_i, \cdots, \Lambda_n)^t = \begin{bmatrix} l_{11}^{(o)}, & l_{12}^{(o)}, & \cdots, & l_{1m}^{(o)} \\ l_{21}^{(o)}, & l_{22}^{(o)}, & \cdots, & l_{2m}^{(o)} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1}^{(o)}, & l_{n2}^{(o)}, & \cdots, & l_{nm}^{(o)} \end{bmatrix} \tag{27}$$

and the *object-label pairs' compatibility matrices* C_{ij} for every i and j $(i, j = 1, 2, \cdots, n)$.

Definition 5: The DRA3 Hardware Implementation problem has been defined as finding the *consistent labeling matrix L*:

$$L = (L_1, L_2, \cdots, L_i, \cdots, L_n)^t = \begin{bmatrix} l_{11}, & l_{12}, & \cdots, & l_{1m} \\ l_{21}, & l_{22}, & \cdots, & l_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1}, & l_{n2}, & \cdots, & l_{nm} \end{bmatrix} \tag{28}$$

for the given *World Model*, provided by the *initial labeling matrix*:

$$\Lambda = (\Lambda_1, \Lambda_2, \cdots, \Lambda_i, \cdots, \Lambda_n)^t = \begin{bmatrix} l_{11}^{(o)}, & l_{12}^{(o)}, & \cdots, & l_{1m}^{(o)} \\ l_{21}^{(o)}, & l_{22}^{(o)}, & \cdots, & l_{2m}^{(o)} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1}^{(o)}, & l_{n2}^{(o)}, & \cdots, & l_{nm}^{(o)} \end{bmatrix} \tag{29}$$

and the *object-label pairs' compatibility matrices* C_{ij} for every i and j $(i, j = 1, 2, \cdots, n)$.

The above definitions represent our research strategy for developing DRA architectures. The DRA2 system is aimed at first reducing both the space and time complexities for a particular DRA application problem, i.e., the region coloring problem. The second one, DRA3 design, generalizes DRA2 for an arbitrary DRA problem and leads to a faster and more general purpose architecture.

IV. A Conventional DRA1 Design and the Complexity Analysis

The entire DRA computation consists of mainly two parts. The first part generates the n^2 m by m compatibility matrices for object pairs u_i and u_j; i.e., computing each matrix $\Lambda_{ij}(k, p)$ $(i, j = 1, \cdots, n; k, p = 1, \cdots, m)$ from the most recently evaluated (or originally given at the first iteration) labelings; as depicted in (13)–(21). The second part as shown in (22)–(24), is an iteration process for the relaxation of the current object pairs' constraints among all different objects to produce new labelings; i.e., computing each new labeling l_{ik} ($i = 1, \cdots, n$; $k = 1, \cdots, m$), provided by the currently known object constraint matrices $\Lambda_{ij}(k, p)$. Reviewing the DRA formulation and the computational procedure for the region coloring example in Section II reveals the complexity issues met in the DRA1 system design.

Time Complexity: With (22)–(24), at least $n \times m$ memory Read/Write operations must be performed for computing an n by m matrix of $l_{jp}^{(n-1)} * \Lambda_{ij}(k, p)$ elements. Because there are $n \times m$ iteration equations (for computing new labels l_{ik}, where $i = 1, \cdots, n$; $k = 1, \cdots, m$) which need to be evaluated, the sequential computation time would be $O(n^2m^2)$. In the first DRA1 design, in addition to using the sequential evaluation strategy, another $O(m)$ time is spent on the complicated nested control loop for memory Read/Write operations during each iteration evaluation [8]. Thus the entire computation time within each iteration costs $O(n^2m^3)$. There are other binary computations which must be taken into account. According to assumption 4, their contributions compared to R/W operations of memory are ignored.

Space Complexity: During each iteration of DRA computation, several intermediate arguments (initial arguments at the first iteration) take certain amounts of space be stored. The initial label matrix Λ ($L^{(o)}$ matrix) and the next iteration result of Λ each takes $O(n^2)$ space. The matrices $\Lambda_i \times \Lambda_j^t$ and the label pairs compatibility matrices $C_{ij}(k, p)$ as well as the resulting object pairs compatibility matrices $\Lambda_{ij}(k, p)$ ($i, j = 1, \cdots, n$; $k, p = 1, \cdots, m$) each takes $O(n^2m^2)$ space. The total space complexity is:

$$O(2n^2 + 3n^2m^2) = O(n^2m^2). \tag{30}$$

Data Routing Complexity: It is meaningful to consider the routing complexity only if a parallel DRA architecture may possibly exist. We are to discuss this issue in Section V and Section VI when developing parallel DRA architectures. In fact, the data routing complexity during each iteration is over $O(n^2m^2)$. For an 8-object 8-label DRA parallel system, there are at least 8K data being routed or communicated during each iteration.

The DRA1 architecture, unfortunately, serially computes each intermediate element. The computation strategy imbedded in this design is purely *I/O-bound*. Assuming $t_{\text{read}} \approx t_{\text{write}} \approx 500$ ns for an NMOS process and the time complexity is $O(n^2m^3)$ of each iteration. Multiplying the possible maximum iteration times, which is on the order of $O(ea^3)$ and is determined by the feature of the computational model [6], the worst case execution time is over an hour. For practical applications, the number of objects could be 8, 16, or 32, the corresponding memory requirements for these different cases are 8K, 32K, and 128K, respectively. As shown in the DRA1 design, this has greatly added to the circuit size which has been a bottleneck in DRA1 system design when n becomes larger.

V. Parallel Tree-Root Pipelining and the DRA2 System

In the analyses above, most of the computing time was spent on the R/W operations for matrices $\Lambda_{ij}(k, p)$; moreover, one half of the $O(n^2m^2)$ space is taken for the generation of matrices $\Lambda_{ij}(k, p)$, which blocks the DRA1 VLSI computation. In this section we describe a DRA2 architecture which is completely independent of the $\Lambda_{ij}(k, p)$ evaluation [9].

A. A Parallel Tree-Structured Reformulation for DRA

It should be clear that any attempt to speed up an I/O-bound computation must rely on an increase in the memory bandwidth. Speeding up a compute-bound computation, however, may frequently come from the concurrent use of many processing elements [16]. The degree of parallelism in a special-purpose system is largely determined by the underlying algorithm. In order to solve the complexity arising in the conventional DRA1 design, the following three steps have been taken to design a hardware algorithm that supports a high degree of concurrency in DRA computation.

1) Constructing the Parallel Computation Tree: When more effort is spent analyzing (2), we see that element $\Lambda_{ij}(k, p)$ can be decomposed as:

$$\Lambda_{ij}^{(n-1)}(k, p) = l_{ik}^{(o)} l_{jp}^{(o)} C_{ij}(k, p) \tag{31}$$

which can form a leaf node as shown in Fig. 1, so that (2) can be hierarchically formed as a tree-like structure with each level imbedded in the parallel computation for their leaves' operands as shown in Fig. 3.

2) Speeding Up Iteration: Once a DRA problem in processing is identified as convergent, its current computing labeling reaches its final consistent labeling asymptotically in such a way that the error vectors, i.e., the difference vectors between these two labelings, monotonically decrease. Therefore, the node computation in Fig. 1 can be speeded up by replacing the initial given elements $l_{ik}^{(o)}$ and $l_{jp}^{(o)}$ with the most recently $n - 1$th iterated results $l_{ik}^{(n-1)}$ and $l_{jp}^{(n-1)}$. The modified computation composed of the leaf node is shown in Fig. 2. The computation tree for $l_{ik}^{(n)}$ is formed as shown in Fig. 3. In Section V-B it is shown that the bottommost leaves' operands have been associated with a data pipelining channel, completing a multiple parallel tree-root pipelining scheme which supports a highly concurrent DRA computation. For convenient notation in the rest of the paper, we classify these three kinds of bottommost leaves' operands of Fig. 3 in the following definition.

Definition 6: 1) We define the labeling elements of the l_{jp}'s ($j = 1, \cdots, n$ and $p = 1, \cdots, m$), which are inside the pipelining channel, to be *l-pipe*, and those on the tree root to be *l-root*. Both *l-pipe* and *l-root* have the same logic values except they are topologically separated in the system layout. 2) Signal l_{ik} passes through all roots horizontally through each row array of k; it is defined as the broadcasting signal b_k. 3) The $C_{ij}(k, p)$ matrix ele-

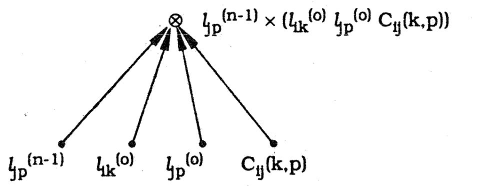

$l_{jp}^{(n-1)} \times (l_{ik}^{(o)}\, l_{jp}^{(o)}\, C_{ij}(k,p))$
$l_{jp}^{(n-1)}$
$l_{ik}^{(o)}$
$l_{jp}^{(o)}$
$C_{ij}(k,p)$

Fig. 1. Leaf node for computing $l_{jp} \times \Lambda_{ij}(k, p)$.

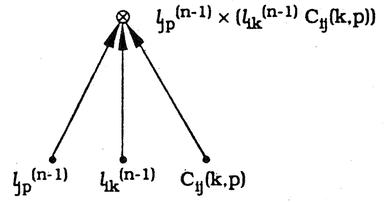


Fig. 2. Modified leaf node computation for $l_{jp} \times \Lambda_{ij}(k, p)$.

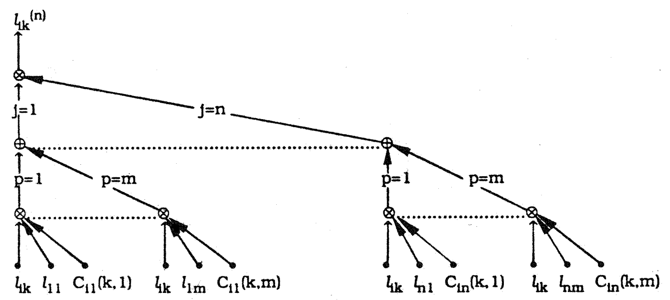

$l_{ik}^{(n)}$
j=1
j=n
p=1
p=m
p=1
p=m
l_{ik} l_{11} $C_{11}(k,1)$
l_{ik} l_{1m} $C_{11}(k,m)$
l_{ik} l_{n1} $C_{in}(k,1)$
l_{ik} l_{nm} $C_{in}(k,m)$

Fig. 3. A parallel tree for computing nth l_{ik}.

ments, which are systematically distributed along each tree root, are defined as the *$C_{ij}(k, p)$-Pattern.*

3) Introducing a Time Dimension in the Computation: To compute an n-object m-label DRA problem, a total of n by m l_{ik}'s need to be evaluated. This means at least 64 computation trees as shown in Fig. 3 need to be built inside the circuit for our $n = m = 8$ case, which greatly increases the circuit size. To minimize this problem, each operand at the bottom of the tree has been constructed in a time-dimension. As the time changes forward, different l_{ik}'s ($i = 1, \cdots, n$; $k = 1, \cdots, m$) can be generated. These time-varying characteristics can best be described by the following *spatial-temporal (ST) index equations*:

$$i \leftarrow i + t \bmod n, \tag{32}$$

$$j \leftarrow j + t \bmod n. \tag{33}$$

Denoted by $A \leftarrow B$ or we say that A is generated by B. We call i, or $i + t$, and j, or $j + t$ the spatial-temporal indexes since they characterize the indexes in the recursive formulas of (1)–(9). They constitute the theoretical basis for the recursive systolic computation and interleaved processing in the DRA2 and DRA3 systems, while the basic DRA-PE cell realizes the computation in the formula. It is also of interest to see that in the DRA2 system [9], the ST indexes generate the dynamic forward DRA computational wavefront on a statically configured DRA2 architecture; while in the DRA3 architecture [10], it forms the dynamically configurable DRA3 architectural wavefront for a virtually static computation.

B. Implementation Issues for the DRA2 Chip

1) System Architecture and Block Diagram: The block diagram of the DRA2 circuit is illustrated in Fig. 4. The chip consists of four functional blocks.

a) Compatibility Matrix Registers (CMR). C_{ij} Registers are a set of eight 8-bit shift registers in the leftmost part of the circuit; they are used for storing each C_{ij} matrix. Another set of C_{ii} Registers in the rightmost part of the circuit are for storing C_{ii}.

b) 8 × 8 Multiprocessor SIMD Array (MSA). The MSA is composed of 8 by 8 simple and regular cells. They are predefined to map the highly parallel computation algorithm of Fig. 3 onto silicon. A number of parallel horizontal and vertical communication wires are designed around the four edges of the cells to make use of higher degrees of parallelism in the computation.

c) L-matrix Shift Register (LSR). It is used for 1) the input and output data paths for the original and final labeling matrices, 2) the pipelining channel for tree-root operand broadcasting and pipelining, forming a recursive DRA computational wavefront, and 3) performing temporarily the data storing and updating.

d) Control Module (CM). This module includes four units. An 8-Bit Comparator is located on top of the first 8-bit shift register of the LSR to sense the equality between the nth output vector $L_i^{(n)}$ of the MSA array and the corresponding $n - 1$th row vector $L_i^{(n-1)}$ inside the LSR. A Timer is served as both the systole pacer and tagged-bit signal generator for iteration control. An 8-Bit State Register is used for collecting comparison results from the Comparator and monitoring iteration states. Finally a Finite State Machine (FSM) is built for performing a self-timed synchronization among these functional blocks and host computer.

This diagram of four functional blocks also serves as the PPL layout floor plan for efficient layout (in subsection V-D).

2) Multiprocessor SIMD Array and the Cell Design: The basic principle of the Multiprocessor SIMD architecture for DRA2 is illustrated in Fig. 5. By replacing a single Processing Element with an array of 8 by 8 PE's, a higher computation throughput can be achieved without increasing memory bandwidth. The function of the memory (i.e., the L matrix shift registers) in the diagram is to pulse data *l-pipe*$_{jp}$ ($j = 1, \cdots, n$; $p = 1, \cdots, m$) through the array of cells. Then new data *l-pipe*$_{ik}$ ($i = 1, \cdots, n$; $k = 1, \cdots, m$) are returned to memory in a rhythmic fashion. The crux of this approach is to ensure that once the data are brought out from the memory they can be used effectively at each cell they pass while being pumped through the entire array.

To perform the parallel DRA2 computation, two cells [as illustrated in Fig. 6(a) and (b)] with almost identical logic and structure were used in constructing the entire array. The only difference is that the first cell performs

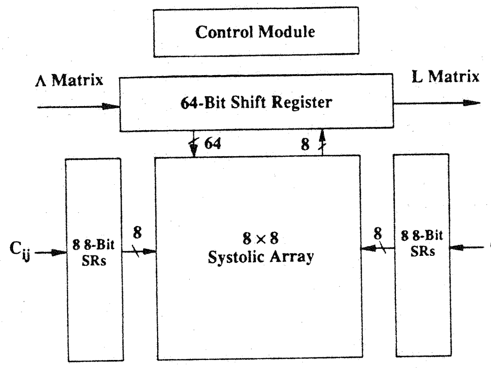


Fig. 4. Circuit block diagram for the DRA2 system.

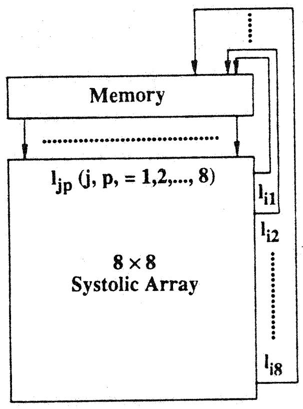


Fig. 5. Basic principle of the parallel DRA2 system.

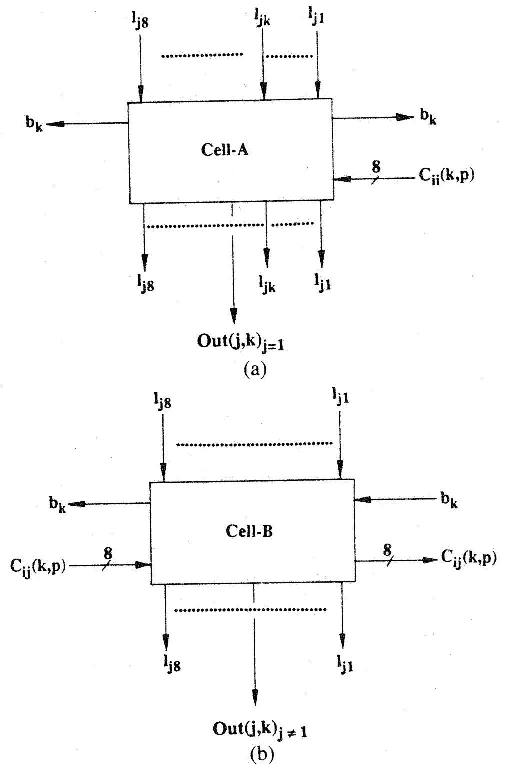


Fig. 6. Two cells in multiprocessor DRA2 array. (a) Cell-A. (b) Cell-B.

the generation of the multiple broadcasting signals b_k ($k = 1, \cdots, m$) for each row array while the second cell is only transparent to the passing of the b_k signals. The construction of the multiprocessor SIMD array using these two cells is illustrated in Fig. 7.

$$b_k = l_{jk}, \text{ at column } j = 1. \tag{34}$$

$$\begin{aligned} \mathrm{Out}(j, k)_{j=1} &= \sum_{p=1}^{m} (l_{jp} \times \Lambda_{ij}(k, p)) \\ &= \sum_{p=1}^{m} (l_{jp} \times l_{ik} \times C_{ii}(k, p)) \\ &= \sum_{p=1}^{m} (l_{jp} \times b_k \times C_{ii}(k, p)) \\ &= \sum_{p=1}^{m} (\overline{\overline{l_{jp}} + \overline{b_k} + \overline{C_{ii}(k, p)}}). \end{aligned} \tag{35}$$

$$b_{k(in)} = b_{k(out)}, \text{ at columns } j \neq 1. \tag{36}$$

$$\begin{aligned} \mathrm{Out}(j, k)_{j\neq 1} &= \sum_{p=1}^{m} (l_{jp} \times \Lambda_{ij}(k, p)) \\ &= \sum_{p=1}^{m} (l_{jp} \times l_{ik} \times C_{ij}(k, p)) \\ &= \sum_{p=1}^{m} (l_{jp} \times b_k \times C_{ij}(k, p)) \\ &= \sum_{p=1}^{m} (\overline{\overline{l_{jp}} + \overline{b_k} + \overline{C_{ij}(k, p)}}). \end{aligned} \tag{37}$$

According to Fig. 3 and (34)–(37), these two cells are implemented in two levels of NOR gate combinational logic. Their PPL [13], [14] layouts can be easily identified in Fig. 13.

We see from Fig. 6 that in essence the inner summation part of (2) is carried out in Cell-A and Cell-B, while the outer multiplication part of that equation is implemented in each horizontal array in Fig. 7.

It is also noted from Figs. 6 and 7 that the DRA2 architecture can readily be extended to different numbers of objects and labels. For example, a row array of the number k in Fig. 7 can be added in order to attach one more label for all objects, while a column array of the number j in the same figure can be added to extend the system for one more object, provided that the corresponding changes inside each cell are made. The critical analysis indicates that the monolithic DRA2 systems for 8, 16, and 32 objects are technically implementable in the 3.0 μ NMOS, 2.0 μ and 1.2 CMOS, and 1.0 μ GaAs processes [12].

3) Circuit Features and Design Techniques: In addition to designing the simple and regular cells, several efficient techniques, such as interleaved processing, multiple signal broadcasting, and self-timed synchronization, were applied to the implementation of the SIMD multiprocessor DRA2 architecture.

Recursive Systolic Computation and Interleaved Processing: Since the introduction of the time dimension in subsection V-A, the multiprocessor SIMD array in Fig. 7 possesses a time-varying characteristic which makes recursive systolic computation and interleaved processing

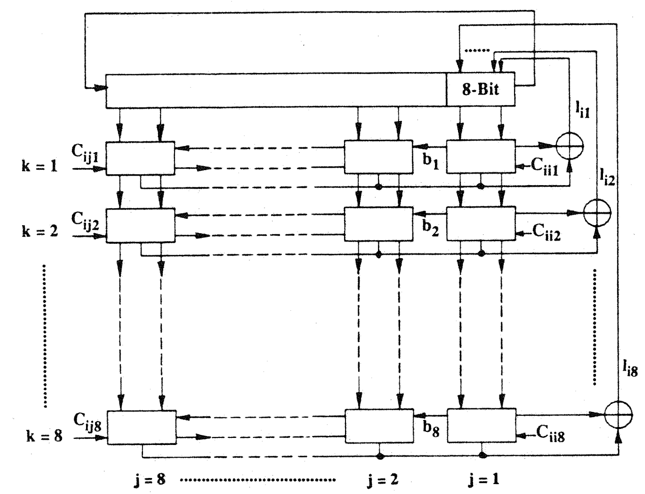


Fig. 7. Construction of SIMD multiprocessor array using Cell-A and Cell-B.

possible. Let us focus on the first column ($j = 1$) array. It is clearly indicated that the first input vector, which is the ith row vector of L, the labeling matrix, at the $n - 1$th iteration, is fed into the first column of DRA2 array as

$$(l_{j1}, l_{j2}, l_{j3}, \cdots, l_{jm}),$$

where $j = 1, \cdots, n$. The corresponding output vector L_i of the multiprocessor SIMD array, which is the ith row vector of L labeling matrix at the nth iteration, is generated:

$$(l_{i1}, l_{i2}, l_{i3}, \cdots, l_{im}),$$

where i is fixed at a time $t = i$. As time moves forward, the elements in the L shift register have shifted from the left to the right in an 8-clock-pace fashion. For example, in the DRA2 system, at time $t = i = 1$, vector L_1

$$(l_{11}, l_{12}, l_{13}, l_{14}, l_{15}, l_{16}, l_{17}, l_{18})$$

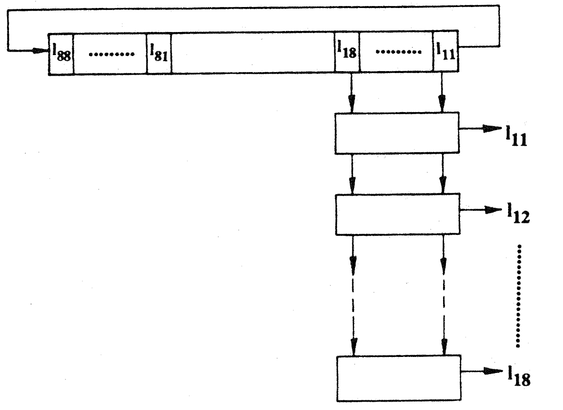
is generated [see Fig. 8(a)], and at time $t = i = 2$, vector L_2

$$(l_{21}, l_{22}, l_{23}, l_{24}, l_{25}, l_{26}, l_{27}, l_{28})$$

is generated, etc. [see Fig. 8(b)].

Each L_i vector is computed based on the interleaved utilization of the multiprocessor SIMD array, whereas eight L_i vectors form an entire computational wavefront of the L labeling matrix, of the nth relaxation iteration. Note that we use n computing trees for generating $n \times m$ l_{ik}'s in $O(nm)$ time; we may also use $O(nm)$ computing trees to compute the same number of l_{ik}'s in $O(n)$ time, provided that the latter has a uniformly progressing wavefront in time and in space.

Multiple Signal Broadcasting: The broadcasting technique is probably one of the most obvious ways to make multiple use of each input element. It plays an important role in making the parallel computation tree of Fig. 3 implementable. Two multiple broadcasting schemes are used in DRA2 architecture. In the first, n by



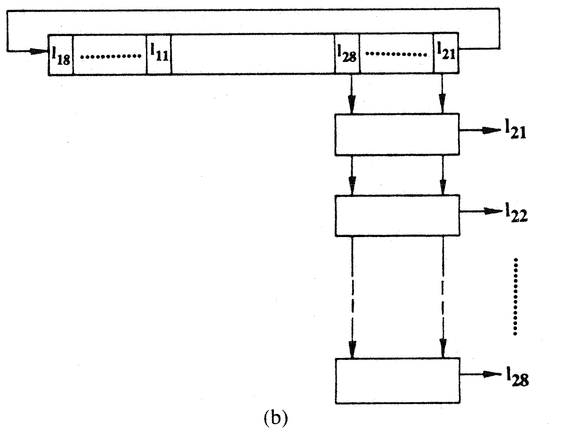


Fig. 8. Computational wavefront pipelining and circulation for interleaved processing.

m vertical broadcasting lines from each pipelining operand are connected to the bottom most leaves' node of each parallel computing tree (passing each *l-pipe* to *l-root*). Secondly, as depicted in Figs. 6 and 9, Cell-A at column $j(=1)$ is used to jog signal $l_{ik}^{(n-1)}$ (which is the nth *l-pipe*$_{jp}$) and then propagate it horizontally from right to left through the entire row array. Thus, the output vector of the multiprocessor SIMD array, i.e., $(l_{i1}, l_{i2}, l_{i3}, l_{i4}, l_{i5}, l_{i6}, l_{i7}, l_{i8})$, can be generated simultaneously in a highly concurrent manner.

Definition 7: The second multiple data routing scheme for jogging b_k at column $j = 1$, as illustrated in (32) and (34), and Fig. 9, is defined as the *J-Pattern*.

Associative Circular Pipelining: During each clock cycle, many operations are performed by bringing the right data together using the associative circular pipelining in the DRA2 design. Figs. 7, 8, 9, and 10 show the implementation of two circular pipelining loops.

The first pipeline is designed for the l_{jp} elements. This pipeline can be viewed in two parts, the master pipeline part and the slave pipeline part. The master pipeline refers to a serial horizontal circular pipeline of the l_{jp} elements inside the main pipelining channel, LSR. While the master pipeline is going on, a parallel horizontal virtual circular pipeline is followed in k row-horizontal array. The parallel pipeline in the DRA array is a slave pipeline of the master one which is supported by multiple signal broadcasting.

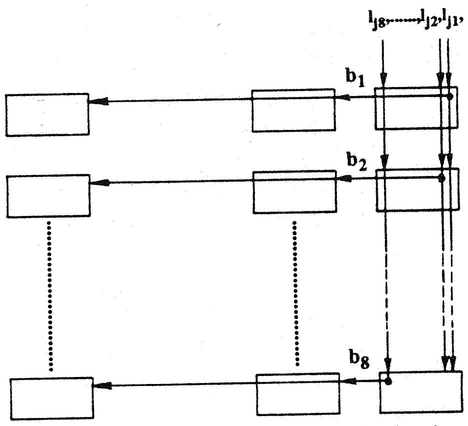


Fig. 9. Broadcasting scheme for the b_k signals.

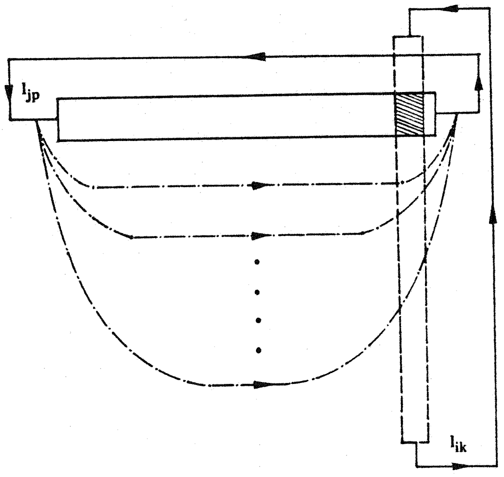


Fig. 10. Associative circular pipelining.

Viewing Fig. 10, the second pipeline loop for l_{ik} is a parallel vertical circular pipeline which crosses the serial horizontal pipeline at the rightmost first m-bit registers of the main pipeline channel, updating the old labels in the channel with new labels.

In addition to providing impressive parallelism, an extra advantage of both pipelining scheme is that they decrease the routing and control complexities significantly.

Self-Timed Synchronization and Tagged-Bit Control: By using recursive systolic computation and interleaved processing, the computational task has been decomposed into the smallest computing piece L_i. To compute each vector L_i, the globally synchronized multiprocessor SIMD array of Fig. 7 is used. For completing the entire relaxation computation, this synchronous array is imbedded into a self-timed system. The self-timed asynchronous scheme may be costly in terms of extra hardware and delay in each element, but it has the advantage that the time required for a communication event between two elements is independent of the size of the entire system [17]. Also, it is easy to design and validate a self-timed state machine in PPL methodology [18].

Among the 64-bit L shift registers, the rightmost first 8-bit SR is one which is able to parallel load in the nth output vector from the multiprocessor SIMD array in order to update the current $n - 1$th L_i row vector. This iteration and updating process is the core of the relaxation process described in (8). To sense the completion of computation, a Comparator is built on top of the first 8-bit SR. If two vectors are equal, a *row-eq* signal of 1 is produced and stored into *8-bit States SR* of the Control Module; otherwise a 0 signal is sent. As soon as the State Register gets eight 1's, which means the equality of (8) is reached, an *all-eq* signal is issued to the FSM. Since the control processes in this system are based on the data validity of a *control data flow*, reliable and fast execution in a data-driven environment is created. The control mechanism used in the parallel DRA2 architecture is shown in Fig. 11.

To ensure that the iteration cycle completes at the end of $n \times m$ cycles, a *tagged-bit* is derived from an ANDed term of both the l_{11} bit and the 64th-count of the Timer,

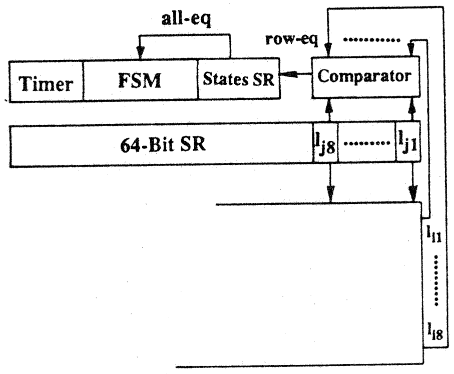


Fig. 11. Self-timed synchronization.

which has served as a reliable alignment signal for computation in the control flow. In Fig. 12 the state graph of the FSM is illustrated.

C. Performance Evaluation

With the parallel tree-structured reformulation and tree-root pipelining in its system design, the DRA2 architecture takes advantage of a high degree of pipelining and multiprocessing. *It gets rid of the need for storing and computing each element associated with the object pairs matrices* $\Lambda_{ij}(k, p)$. One-half of the $O(n^2m^2)$ space is eliminated; the other half is reserved for input data $C_{ij}(k, p)$, the label pairs compatibility matrices. In certain DRA computations, as shown in the *Region Coloring Problem*, the space complexity can even be reduced to $O(nm)$; i.e., only an $O(nm)$-bit shift register is required to store all n by m intermediate (or original) labeling elements [9]. Obviously, the DRA2 computation during each iteration only takes $O(nm)$ clock cycles. Assuming a clock cycle is about 120 ns (using a 3 μ NMOS process and the PPL design methodology), DRA2 performs the DRA computation in microseconds, and in the worst case, i.e., multiplying the maximum possible iteration time, $O(ea^3)$, in milliseconds [9].

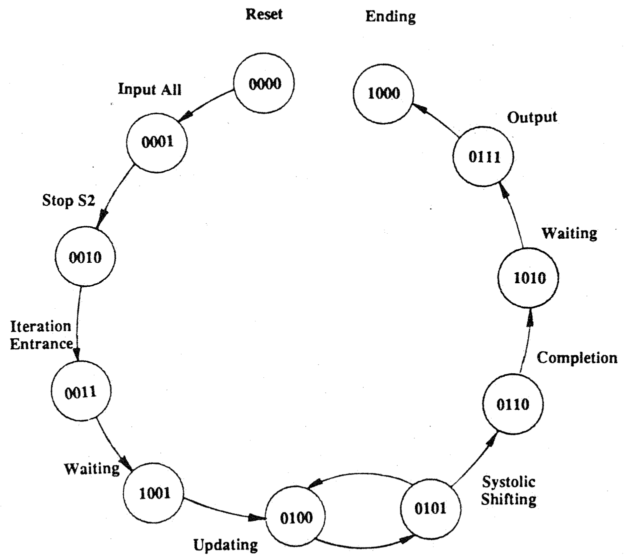


Fig. 12. State graph of the finite state machine.

D. *The PPL Layout for DRA2 System*

The DRA2 chip was built by assembling the four functional blocks in Fig. 4 using PPL (Path Programmable Logic) tools at the University of Utah [13], [14]. Since parallel computation and the multiprocessor SIMD array greatly simplify the design difficulties, the PPL layout is very simple and straightforward. An overview of the complete PPL layout, which is a PPL mapping of the block *floor plan* in Fig. 4, is shown in Fig. 13.

For details associated with the complete system design, the simulation results, interfacing strategy with host computer, timing and wiring delay analysis, testing, pinout description, and fabrication of the DRA2 chip see [9], [11] (see Fig. 14).

VI. An $O(n)$ Time Algorithm-Configured Dynamic Architectural Wavefront System

Compared to the conventional DRA1 design, the DRA2 system achieves a very impressive performance improvement in terms of speed, size, and memory access requirements. However, there are still several bottlenecks in developing a fast general purpose DRA architecture.

A. *Complexity Issues Revisited*

Time Complexity: The $O(nm)$ time of the DRA2 system is suitable only for a small number of objects, although the number of objects and labels in DRA2 architecture can be extended to a larger number of objects and labels. The larger the number of objects and labels are, the slower the DRA2 system becomes. It is desirable that a much faster DRA architecture be designed for real-time processing of a larger number of objects and labels.

Data Preprocessing Complexity: One of the remarkable characteristics of the DRA problem is that a large number of data, such as $O(n^2m^2)$ elements of label pairs compatibility matrices $C_{ij}(k, p)$, must be loaded into the system before the computation is performed. In addition to this, the data ordering and format which might be processed during the load-in time must be arranged to support efficient DRA computation. Following assumption 3, the term *Data Preprocessing Complexity* (*DPC*) is used here to refer to the complexity issues which arise in the analysis of the time and space, as well as hardware complexities in this kind of data-preprocessing-intensive VLSI computation. Some labeling solutions avoid this problem by assuming that the data is always ready for computation.

Data Routing Complexity: One of the proposed DRA3 architectures [10] is shown in Fig. 18, in which the data routing complexity in the horizontal direction between two DRA modules is of $O(m^2)$; a total of $O(nm^3)$ connections are required for nm DRA modules to be routed. The data routing complexity becomes another dominant factor in designing the DRA3 system as the number of labels increases.

Fabrication Difficulty: Although we have separated the main pipelining channel and have used the larger buffers to drive between shift registers, its characteristic turns out to be worse as the numbers of n and m increase.

B. *$C_{ij}(k, p)$-Pattern Analysis*

The DRA3 architecture overcomes these difficulties based on the use of the dynamically configured architectural wavefront on the DRA3 switch lattice [10]. It runs the DRA computation in $O(n)$ time without the extra requirement for data preprocessing hardware and without requiring the horizontal routing among the DRA modules, provided that only an $O(nm^2)$ number of switches must be added to the original multiple broadcasting wires of the DRA2 system.

In fact, computing time, data preprocessing complexity, and data routing complexity form a coherent mixture in the DRA3 system design. We illustrate its major architectural concepts beginning with the analysis of $C_{ij}(k, p)$-*Patterns* in the case of $n = m = 3$.

Referring to (2) and Fig. 8, the first computational wavefront for computing the nth labels $l_{ik}^{(n)}$ is formed at time $t = i = 1$ as follows.

For $i = 1$ and $k = 1, 2, 3$:

$$\begin{aligned} l_{1k}^{(n)} \le l_{1k}^{(n-1)} &* [l_{11}^{(n-1)} * \Lambda_{11}(k, 1) + l_{12}^{(n-1)} * \Lambda_{11}(k, 2) \\ &+ l_{13}^{(n-1)} * \Lambda_{11}(k, 3)] \\ &* [l_{21}^{(n-1)} * \Lambda_{12}(k, 1) + l_{22}^{(n-1)} * \Lambda_{12}(k, 2) \\ &+ l_{23}^{(n-1)} * \Lambda_{12}(k, 3)] \\ &* [l_{31}^{(n-1)} * \Lambda_{13}(k, 1) + l_{32}^{(n-1)} * \Lambda_{13}(k, 2) \\ &+ l_{33}^{(n-1)} * \Lambda_{13}(k, 3)]. \qquad (38) \end{aligned}$$

A snapshot of the $C_{1j}(k, p)$ ($j, k, p = 1, 2, 3$) matrices distribution pattern which is matched with the $l_{1k}^{(n)}$ wavefront at $t = 1$ is shown in Fig. 15. At time $t = 2$, a computational wavefront for generating $l_{2k}^{(n)}$ is produced. For

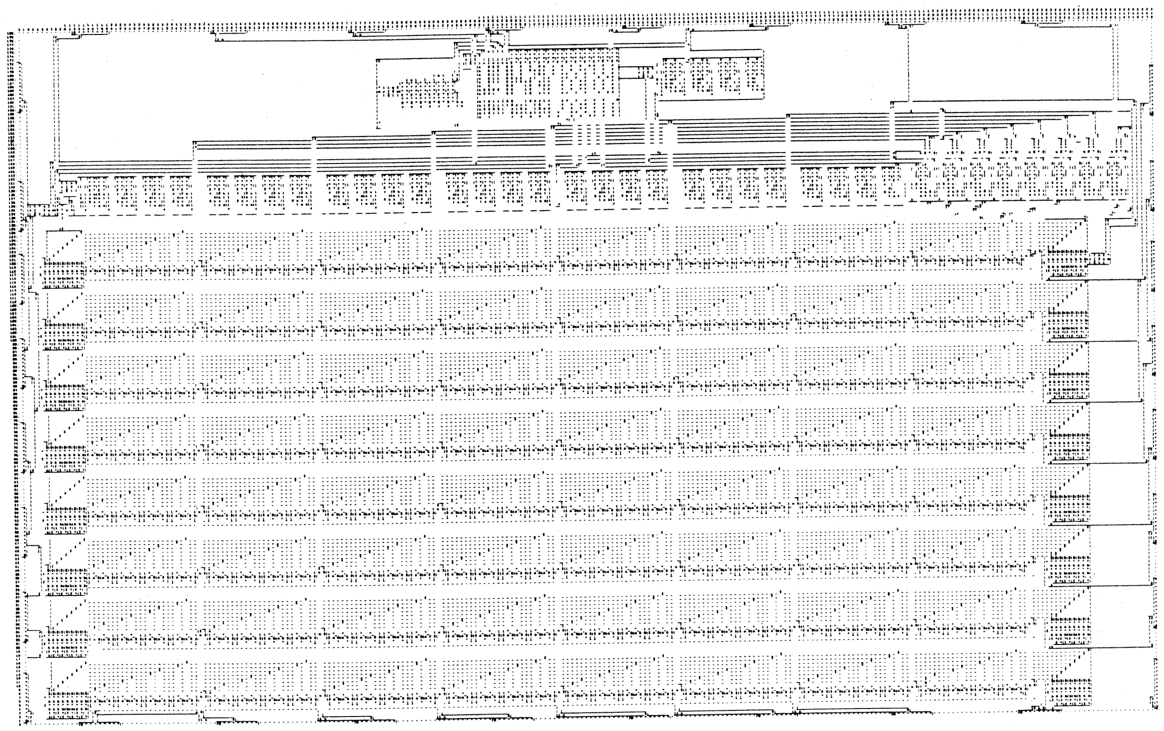


Fig. 13. The PPL layout of the multiprocessor SIMD DRA2 system.

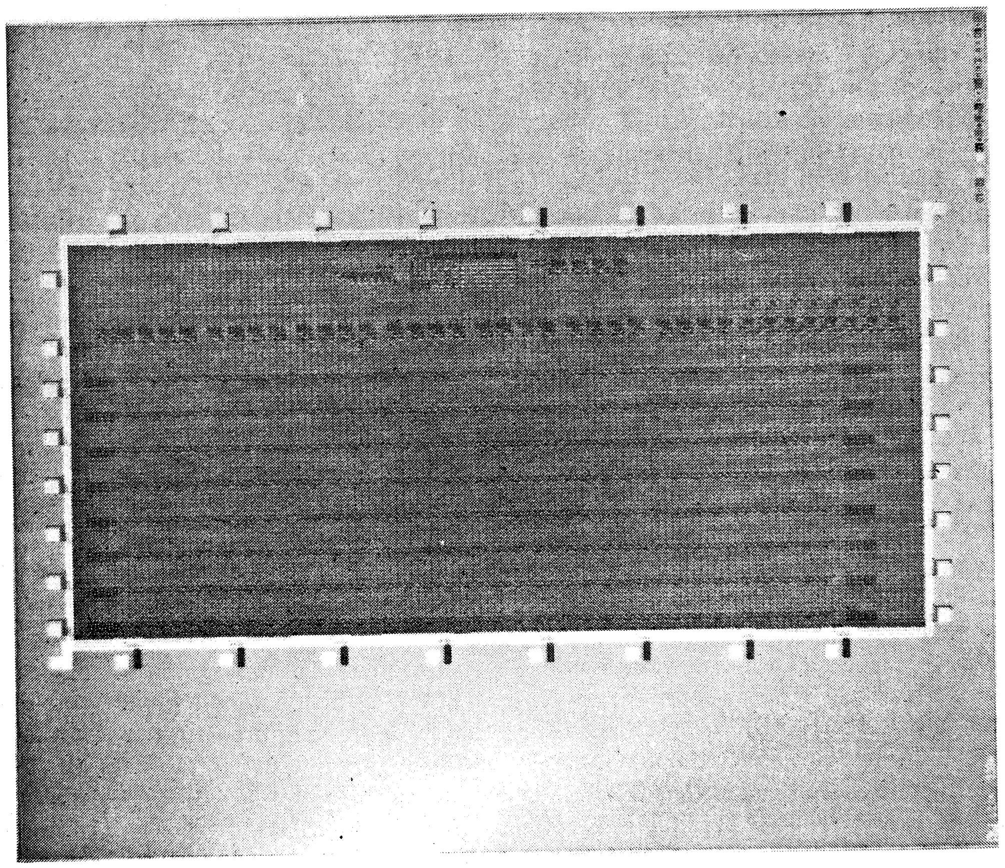


Fig. 14. The photomicrograph of the DRA2 chip.

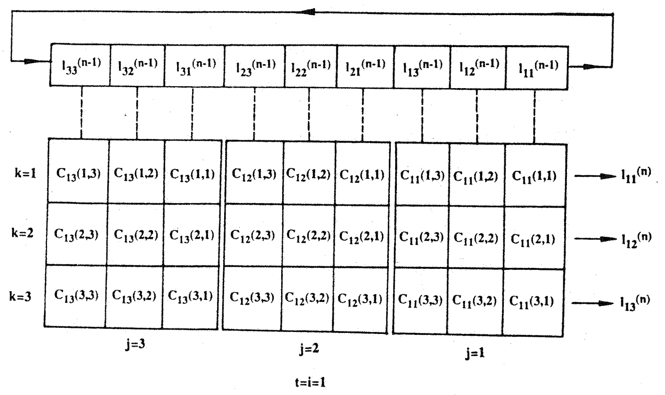

l33(n-1) l32(n-1) l31(n-1) l23(n-1) l22(n-1) l21(n-1) l13(n-1) l12(n-1) l11(n-1)
k=1 C13(1,3) C13(1,2) C13(1,1) C12(1,3) C12(1,2) C12(1,1) C11(1,3) C11(1,2) C11(1,1) l11(n)
k=2 C13(2,3) C13(2,2) C13(2,1) C12(2,3) C12(2,2) C12(2,1) C11(2,3) C11(2,2) C11(2,1) l12(n)
k=3 C13(3,3) C13(3,2) C13(3,1) C12(3,3) C12(3,2) C12(3,1) C11(3,3) C11(3,2) C11(3,1) l13(n)
j=3
j=2
j=1
t=i=1

Fig. 15. The $C_{1j}(k, p)$ pattern for generating l_{1k}'s.

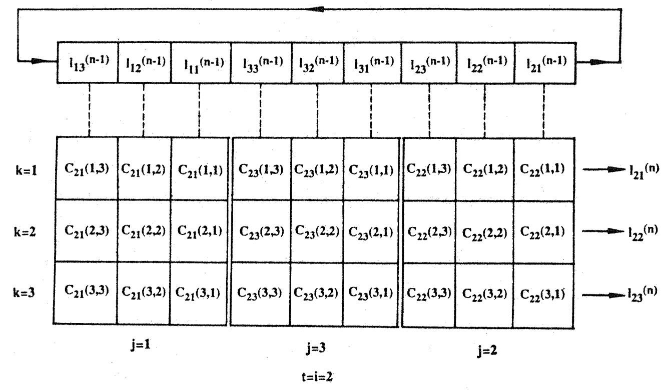

l13(n-1) l12(n-1) l11(n-1) l33(n-1) l32(n-1) l31(n-1) l23(n-1) l22(n-1) l21(n-1)
k=1 C21(1,3) C21(1,2) C21(1,1) C23(1,3) C23(1,2) C23(1,1) C22(1,3) C22(1,2) C22(1,1) l21(n)
k=2 C21(2,3) C21(2,2) C21(2,1) C23(2,3) C23(2,2) C23(2,1) C22(2,3) C22(2,2) C22(2,1) l22(n)
k=3 C21(3,3) C21(3,2) C21(3,1) C23(3,3) C23(3,2) C23(3,1) C22(3,3) C22(3,2) C22(3,1) l23(n)
j=1
j=3
j=2
t=i=2

Fig. 16. The $C_{2j}(k, p)$ pattern for generating l_{2k}'s.

$i = 2$ and $k = 1, 2, 3$:

$$\begin{aligned} l_{2k}^{(n)} \le l_{2k}^{(n-1)} * [l_{11}^{(n-1)} * \Lambda_{21}(k, 1) + l_{12}^{(n-1)} * \Lambda_{21}(k, 2) \\ + l_{13}^{(n-1)} * \Lambda_{21}(k, 3)] \\ * [l_{21}^{(n-1)} * \Lambda_{22}(k, 1) + l_{22}^{(n-1)} * \Lambda_{22}(k, 2) \\ + l_{23}^{(n-1)} * \Lambda_{22}(k, 3)] \\ * [l_{31}^{(n-1)} * \Lambda_{23}(k, 1) + l_{32}^{(n-1)} * \Lambda_{23}(k, 2) \\ + l_{33}^{(n-1)} * \Lambda_{23}(k, 3)]. \end{aligned} \tag{39}$$

The $C_{2j}(k, p)$ $(j, k, p = 1, 2, 3)$ matrices distribution pattern at this time is shown in Fig. 16.

From Figs. 15 and 16, several criteria for DRA3 architecture are derived.

1) At time $t = i$, to compute each new labeling vector L_i, different computational wavefronts are formed and the associated $C_{ij}(k, p)$ matrices are distributed. More accurately, at time t, each matrix $C_{tj}(k, p)$ elements (where $t, j = 1, \cdots, n$ and $k, p = 1, \cdots, m$) stored in the kth row of a parallel memory in Figs. 15 and 16 are retrieved. Thus a parallel Row-Readable RAM is required.

2) It is clearly indicated that the $C_{ij}(k, p)$ elements associated with each dynamically pipelined operand in the main tree-root pipelining channel are one unit of j skewed and horizontally circulated. The cyclic shifting of the bits of $C_{ij}(k, p)$-*Patterns* elements to the right for m-bit position corresponds to a *Block-data Shuffling* operation [21]. We summarize and extend all distribution C matrices' patterns in Fig. 17. (Each symbol actually is a matrix distributed in Figs. 15 and 16.)

The first requirement requires a simple associative processing, i.e., parallel Row-Readable RAM's; while the second criterion demands higher order design complexity in time and data preprocessing, as well as in data routing.

A candidate design for the DRA3 system, where $n = m = 8$, is presented in Fig. 18. The DRA3 system is made up by using 8 by 8 DRA modules. Each module consists of a PE cell (same structure as Cell-A and Cell-B in DRA2 system) and a local Row-Readable parallel RAM for $C_{ij}(k, p)$-*Pattern*.

A preshuffling chip is implemented and fabricated using a 3 μ NMOS PPL design methodology in order to make the $C_{ij}(k, p)$-*Pattern* the same format as in Fig. 17 [22]. However the pre-processing hardware costs $O(b^3)$ time (noting that b is the number of bits to be shuffled) and takes more than half of the chip, provided that a difficult inter-chip routing problem remains unsolved [10].

$t = 1$:	$C_{13}(k, p)$	$C_{12}(k, p)$	$C_{11}(k, p)$
$t = 2$:	$C_{21}(k, p)$	$C_{23}(k, p)$	$C_{22}(k, p)$
$t = 3$:	$C_{32}(k, p)$	$C_{31}(k, p)$	$C_{33}(k, p)$

Fig. 17. A circularly skewed $C_{ij}(k, p)$-*Pattern* when generating l_{ik}'s.

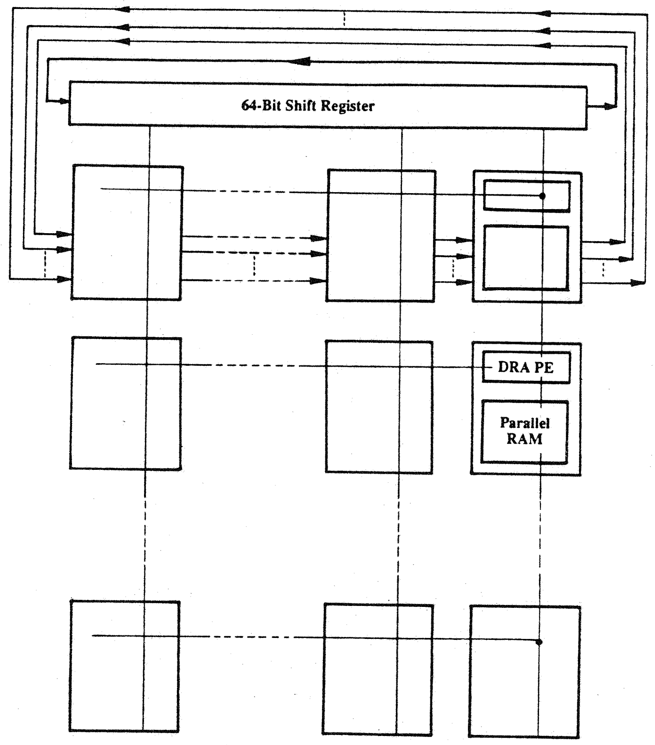


Fig. 18. A candidate design for the DRA3 system.

C. A Dynamically Configurable DRA3 Architecture

The configurable, highly parallel computer system is a multiprocessor architecture that provides a programmable interconnection structure integrated with the PE's [15]. The original idea is that, the computer processing begins with the controller broadcasting a command to all switches to invoke a particular *static* configuration setting. The design of the DRA3 system has revealed that by adopting an advanced *dynamic* configuring strategy on the DRA3 switch lattice, not only can the DRA computational wavefront be generated while retaining the benefits of uniformity and locality that DRA-PE exploits, but also the combined limitations imposed in upgrading DRA3 architecture can be completely eliminated.

1) The Architectural-Computational Wavefront (ACW) Notation: We rewrite Weiser and Davis's definition of wavefront [19]:

Definition 8: A *wavefront*, denoted as A, represents an ordered set of data elements: $\{a(1, m), a(2, m), \cdots, a(N-1, m), a(N, m)\}$, where m is the "time" subscript. The elements $a(I, m)$ for all m belong to the Ith data stream. For simplicity, the "time" subscript in the elements of a wavefront is omitted and $a(i, m)$ will be simply represented as $a(i)$.

We extend the definition as follows.

Definition 9: The *Architectural-Computational Wavefront (ACW) Notation* [10] consists of two different wavefronts. The *Computational Wavefront* is an ordered set of data elements: $\{c(1, i+t), c(2, i+t), \cdots, c(N, i+t)\}$, where $i + t$ is the spatial-temporal index in (32). The *Architectural Wavefront* is an ordered set of architectural configurations: $\{a(1, j+t), a(2, j+t), \cdots, a(N, j+t)\}$, where $j + t$ is the spatial-temporal index in (33).

The following distinctions are made which are necessary for building a DRA3 system:

1) The computational wavefront dynamically progresses on a static architecture, as the ST indexes increase. The architectural wavefront progresses, by dynamically configuring the system architecture, in a way against the computational wavefront, as the ST indexes increase. For instance, as we have seen that the ST index in DRA2 forms a dynamic DRA computational wavefront on a static DRA2 architecture; we will soon see that the ST index of DRA3 generates a dynamic DRA3 architectural wavefront for a virtual static DRA computation.

2) The equivalence between computational wavefront and architectural wavefront holds. Thus either one characterizes both of them.

3) Both architectural and computational wavefront notation have spatial parameters, such as i and j, and temporal parameter, such as t, so that both can be manipulated on the space and time domains.

4) The conventional wavefront notation assumes a uniform progression of the data stream. This restriction is eliminated by combining ST indexes with ACW notation, as it is obvious that these indexes may behave synchronously or asynchronously.

5) The ACW notation first differentiates the computational wavefront and architectural wavefront and is more suitable for exploring architectural configurability and the synthesis of highly configurable system.

Referring to (2), (32), and (33), there is always a data dependence relation among these arguments with respect to time t and positions k and j, that:

$$l\text{-}pipe_{i+t,k}^{(n)} \leftarrow \sum_{p=1}^{m} l\text{-}root_{i+t,k}^{(n-1)} l\text{-}root_{j+t,p}^{(n-1)} C_{i+t,j+t}(k, p). \tag{40}$$

In Fig. 7 we see that this data dependency equation is topologically mapped onto silicon in terms of the parallel tree structure. The vertical broadcasting of the *l-pipe* operands from pipelined channel to DRA array is described by a routing function:

$$l\text{-}pipe_{i+t,k}^{(n-1)} \rightarrow l\text{-}root_{j+t,p}^{(n-1)} \updownarrow C_{i+t,j+t}(k, p), \tag{41}$$

where the symbol $\updownarrow$ denotes an alignment relation such that $A \updownarrow B$ if and only if A and B have the same physical column position of j. The J-Pattern of Definition 7 is routed by equation:

$$l\text{-}root_{j+t,p}^{(n-1)} \rightarrow l\text{-}root_{i+t,k}^{(n-1)} \updownarrow C_{i+t,j+t}(k, p). \tag{42}$$

The DRA computation is valid if and only if these two dependency equations are true at any time t and positions k and j, as well as the iteration times n. Equation (41) stands for a static network in the DRA3 architecture. Equation (42) is a virtually static network; a potential speedup in the DRA3 computation can be reached by dynamic configuring (41) in an ACW notation.

2) Dynamic Configuring the DRA3 Architectural Wavefront: Though symbol j represents the horizontal spatial shifting (in Figs. 15 and 16) while the symbol t represents the temporal unit, the computational wavefront progression of $l\text{-}root_{j+t,p}$ proceeds [referring to (32) and (33)] as the increasing of $j + t$, where the combined parameter $j + t$ must not be associated with any dimensions. By Definition 9, it is obviously correct that the l_{ik} computational wavefront moves from left to right, as the ST indexes increase and is equivalent to that of the J-Pattern (one configuration of the architectural wavefront) shifts in the reverse horizontal direction, as the ST indexes increase. The DRA computation can be performed by manipulating *either* a computational wavefront *or* configuring dynamically an architectural wavefront.

In Fig. 19, we create a dynamically configurable DRA3 architectural wavefront by building an algorithm-configured switch lattice. A total number of $O(nm^2 + nm)$ switches are added between each vertical wire and horizontal wire.

In Fig. 19, there are n by m identical DRA modules denoted by M_{jk} ($j = 1, \cdots, n$ and $k = 1, \cdots, m$). Let SN_{jk} denote a Switching Node which connects vertical broadcasting wire l_{jk} and horizontal broadcasting wire b_k inside of each Module. Let BS_{jk} denote a Bus Switch which connects the labeling output of the multiprocessor SIMD array to the operands of the "pipelining channel," noting that the pipelining channel is now static and is replaced with an $O(nm)$-bit RAM. These switches are programmed by a *Self-Timed System Controller*, as is shown in DRA2 circuit, to avoid the time delay that occurred

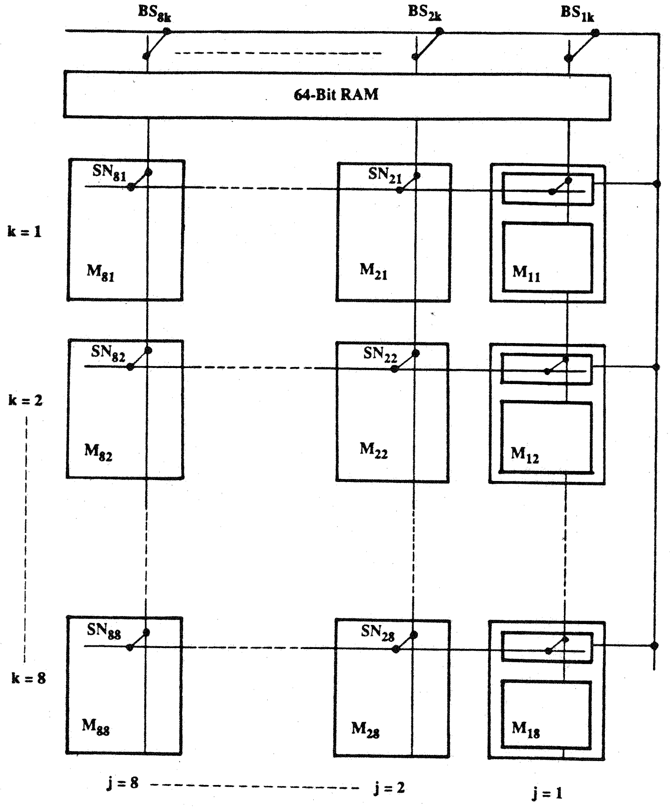


Fig. 19. The DRA3 architecture.

during the broadcasting on the switch lattice. For DRA3 computation, we give the following.

Definition 10: We define the *DRA3 architectural wavefront* to be the interconnection patterns of the DRA3 switch lattice, of which both the switches SN_{jk} for a column j, i.e., the J-Pattern at a specified column j, and switches BS_{jk} for that column j are invoked simultaneously, where index j is defined in (33).

During the DRA3 computation, at index $j \pmod n$, the DRA3 architectural wavefront shifts from right to left in $O(n)$ clock cycles, therefore, an entire DRA computation, which is virtually static, is dynamically generated.

It would now be of interest to examine whether the complexity issues in Section VI-A are solved upon building the DRA3 architecture. First, a time upper bound of $O(n)$ is reached. Secondly, there is no particular hardware support and no special requirements for data-preprocessing. Under dynamic parallel configuring of DRA3, the $C_{ij}(k, p)$*-Pattern* is naturally distributed in matrix index order in the original parallel memory. For an intuitive understanding, we redraw the $C_{ij}(k, p)$*-Pattern* of Fig. 17 in Fig. 20 which is associated with our DRA3 computation strategy.

The third problem, an intensive multiple data routing requirement, is eliminated by adding $O(nm^2 + nm)$ switches on the DRA3 switch lattice. As for the long pipelining channel, it is replaced with $O(nm)$ bits of smaller and more reliable RAM cells, under the dynamic data routing. Finally, since $C_{ij}(k, p)$*-Pattern* is designed for $O(n^2)$ arbitrary m by m C-matrices, the DRA3 system is good for any general purpose DRA computation.

$t = 1$:	$C_{13}(k, p)$	$C_{12}(k, p)$	$C_{11}(k, p)$
$t = 2$:	$C_{23}(k, p)$	$C_{22}(k, p)$	$C_{21}(k, p)$
$t = 3$:	$C_{33}(k, p)$	$C_{32}(k, p)$	$C_{31}(k, p)$

Fig. 20. A $C_{ij}(k, p)$*-Pattern* under dynamic reconfiguration of DRA3.

VII. Conclusions

We have described several VLSI architectures for speeding up the computation of the Discrete Relaxation Algorithm. The key issues are a new tree-root pipelining scheme and a technique to dynamically configure the architectural wavefront in terms of DRA algorithm. The implementations of these architectures offer much greater processing performance than general purpose processors. Further research in this area is to imbed the highest degree of flexibility in DRA design by allowing programmability in cells as well as reconfigurability of cell interconnections, for generating efficient and faster dynamically configurable MIMD DRA architectures.

References

[1] R. V. Southwell, *Relaxation Methods in Engineering Science.* London: Oxford University Press, 1940.

[2] D. Waltz, "Understanding line drawings of scenes with shadows," in *Psychology of Computer Vision*, P. H. Winston, Ed. New York: McGraw-Hill, 1975, pp. 19–91.

[3] D. H. Ballard and C. M. Brown, *Computer Vision.* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[4] J. T. Mccall, J. T. Tront, F. G. Gray, R. M. Haralick, and W. M. Mccormack, "Parallel computer architectures and problem solving strategies for the consistent labeling problem," *IEEE Trans. Comput.*, vol. C-34, pp. 973–980, Nov. 1985.

[5] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operations," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 420–433, June 1976.

[6] T. C. Henderson, *Discrete Relaxation.* London: Oxford University Press, to be published.

[7] A. Bundy, Ed., "Catalogue of artificial intelligence tools," in *Symbolic Computation Artificial Intelligence*, L. Bolc, A. Bundy, P. Hayes, and J. Siekmann, Eds. Berlin: Springer-Verlag, 1984.

[8] D. C-L. Ku, "DRA1 chip implementation report," Dep. Comput. Sci., Project Rep., Mar. 19, 1986.

[9] W. Wang, J. Gu, and T. C. Henderson, "A pipelined architecture for parallel image relaxation operations," *IEEE Trans. Circuits Syst.*, vol. CAS-34, no. 11, 1987; for detailed version see Dep. Comput. Sci., Univ. Utah, Tech. Rep. UUCS-TR-86-008, Mar. 1986.

[10] J. Gu, W. Wang, and T. C. Henderson, "An $O(n)$ time discrete relaxation architecture for real-time processing of the consistent labeling problem," Dep. Comput. Sci., Univ. Utah, Tech. Rep. UUCS-TR-86-116, Dec. 1986.

[11] MOSIS, "DRA2 chip fabrication report," Inform. Sci. Inst., Univ. Southern California, Dec. 1986.

[12] M. Gaspar, "Discussion with the PPL design methodology and MOSIS fabrication technology," private communication, Dec. 1986.

[13] K. F. Smith, T. M. Carter, and C. E. Hunt, "Structured logic design of integrated circuits using the storage/logic array (SLA)," *IEEE Trans. Electron Devices*, vol. ED-29, no. 4, Apr. 1982.

[14] K. F. Smith, *PPL Manual*, Dep. Comput. Sci., Univ. Utah, Tech. Rep. UUCS-86-003, Jan. 1986.

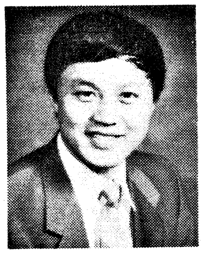
[15] L. Snyder, "Introduction to the configurable, highly parallel computer," *Computer*, vol. 15, pp. 47–56, Jan. 1982.

[16] H. T. Kung, "Putting inner loops automatically in silicon," in *Lecture Notes in Computer Science Vol. 163: VLSI Engineering*, T. L. Kunii, Ed. Berlin: Springer-Verlag, 1985, pp. 70–104.

[17] A. L. Fisher and H. T. Kung, "Synchronizing large VLSI processor arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 734–740, Aug. 1985.

[18] A. B. Hayes, "Self-timed IC design with PPL's," in *Proc. Third*

CalTech Conf. VLSI, R. E. Bryant, Ed. Rockville, MD: Computer Science Press, Jan. 1983, pp. 257–273.
[19] U. Weiser and A. Davis, "A wavefront notation tool for VLSI array design," in *Proc. CMU Conf. VLSI System and Computations*, H. T. Kung, B. Sproull, and G. Steele, Eds. Rockville, MD: Computer Science Press, Oct. 1981, pp. 226–234.
[20] S. Y. Kung, *et al.*, "Wavefront array processor: Language, architecture, and applications," *IEEE Trans. Comput.*, vol. C-31, pp. 1054–1066, Nov. 1982.
[21] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.
[22] W. Burger, "A virtual shuffle dynamic RAM controller," Dep. Comput. Sci., Univ. Utah, Project Rep., Feb. 1986.
[23] J. Gu and K. F. Smith, "Design and implementation of 1.0 μ GaAs path programmable logic system for ultra fast VLSI architectures," Dep. Comput. Sci., Univ. Utah, Research Tech. Rep., to appear, 1987.



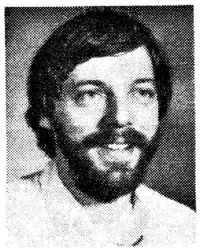
Jun Gu (S'86) received the B.S. degree in electrical engineering from the University of Science and Technology of China (USTC), Hofei, People's Republic of China, in 1982, and pursued the M.S. degree on the subject of bioengineering and optical information processing, as well as VLSI engineering, from the Department of Electrical Engineering, University of Utah, in 1985.

He worked as a faculty member, from 1982 to 1983, at the Department of Electrical Engineering, USTC. Since September 1985, he has been a graduate student of Computer Science at the University of Utah, Salt Lake City, where he is currently a Ph.D. candidate. He was supported by an ASC Fellowship during 1984–1985 and 1985–1986, a University of Utah Research Fellowship during 1986–1987, and this year he has been awarded the ACM/IEEE scholarship. His current research interests include AI and robot vision, computer architecture, fault-tolerant computing, intelligent silicon compilation, and GaAs PPL systems for ultra fast VLSI architectures.

Mr Gu is a student member of the Association for Computing Machinery and the American Association for Artificial Intelligence, and a member of Sigma Xi.

Wei Wang received the B.S. degree in electrical engineering and computer science from the Hofei Institute of Technology, Hofei, People's Republic of China, in 1982.

She worked as a member of the technical staff at the National Institute of Electrical and Mechanical Engineering, Beijing, from 1982 to 1984. In 1984 she joined the faculty of the Department of Computer Science at the University of Science and Technology of China (USTC). Currently she is a graduate student in the Department of Electrical Engineering at the Unviersity of Utah, Salt Lake City. Her current research interests are in the areas of communication system, coding theory, digital signal processing, and VLSI architecture.



Thomas C. Henderson (M'80–SM'86) received the BA degree with Honors in mathematics from Louisiana State University in 1973, and the Ph.D. degree in computer science from the University of Texas at Austin in 1979.

He worked at DFVLR in Germany during 1980, and was at INRIA in France in 1981. He joined the Department of Computer Science at the University of Utah, Salt Lake City, in 1982 where he is presently Associate Professor and Associate Chairman. His current areas of technical interest include multisensor systems, artificial intelligence, computer vision, architectures for robotics, and CAD-based robotics.