Implementing probabilistic algorithms on VLSI architectures

Marius V.A. Hâncu and Kenneth C. Smith

Department of Electrical Engineering, University of Toronto, Toronto, Ontario M5S 1A4, Canada

Received 5 January 1987

Abstract. In this paper, we pursue the use of probabilistic (randomized) algorithms in VLSI architectures, in order to reduce the amount of computation, and, correspondingly, the time of computation as well as chip area.

As a case example, we propose two VLSI solutions to the well-known problem of the nearest pair of points in computational geometry. These implementations are based on Rabin's and Weide's probabilistic algorithms. The chip area and time of computation which result are each $O(n)$. This is a marked improvement over both the straightforward deterministic approach (leading to an $O(n^2)$ computational time) and the deterministic algorithms known as being the best (leading to $O(n \log n)$ computational time).

In suggesting a VLSI solution to the nearest-pair problem, we introduce two new systolic structures, a systolic grouper and a systolic minimum-distance processor. We also make use of a new class of systolic arrays introduced earlier, probabilistic systolic arrays.

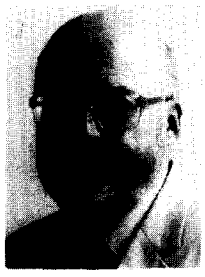
Keywords. Computational geometry, nearest-pair problem, probabilistic algorithms, systolic arrays, VLSI.

1. Introduction

VLSI architectures designed until now execute exclusively deterministic algorithms quite successfully in many application areas. However, at the same time, probabilistic concepts have been acknowledged in the design of digital computers for many years, without becoming predominant or even very visible. Inspired by an interest in these concepts, we have discovered that a special class of algorithms, namely *probabilistic (randomized) algorithms*, quite well-known since

North-Holland
INTEGRATION, the VLSI journal 6 (1988) 59–82

the '70s in the computer-science community, but not yet implemented in hardware to our knowledge, represent very good candidates for competitive and interesting VLSI implementations.



Kenneth C. Smith is a Professor in the Department of Electrical Engineering, the Department of Computer Science and the Faculty of Library and Information Science at the University of Toronto. He is the author or co-author of well over 100 journal and symposium papers in a wide range of topics in electronics and computer architecture, and the contributor to several books, including the text, 'Microelectronic Circuits' co-authored with Adel S. Sedra. As well, he is an advisor to several Canadian companies in the area of new product development utilizing microprocessors.

Born in Toronto, Canada in 1932, he received the B.A.Sc. in Engineering Physics, the M.A.Sc. in Electrical Engineering and the Ph.D. in Physics in 1954, 1956 and 1960, respectively, from the University of Toronto. He was elected Fellow of the IEEE "For contributions to Digital Circuit Design" in 1978, and received an IEEE Computer Society Certificate of Appreciation "For Contributions to Multiple-Valued Circuit and Device Technology" in 1984.

From 1956 to 1958 he worked at the Digital Computer Laboratory at the University of Illinois, Urbana, Illinois, on high-speed digital-circuit design on secondment from the University of Toronto. For the academic year 1960–61 he was an Assistant Professor of Electrical Engineering and a Research Associate at the Institute of Computer Science at the University of Toronto. From 1961–65 he was on the professorial staff of the Department of Computer Science at the University of Illinois, Urbana, where he served as Chief Engineer of the Illiac II high-speed multiprocessor project and as a Chief Engineer and consultant for the Illiac III pattern- recognition project, the latter until 1970.

He returned to the University of Toronto in 1965 to join the Departments of Electrical Engineering and Computer Science as an Associate Professor, working in the area of linear and digital electronics and computer organization. He was promoted to the rank of Full Professor in 1970 and served as Chairman of the Department of Electrical Engineering from 1976 to 1981. As well, he is a founding member of the Computer Systems Research Institute at the University of Toronto.

From 1962 he has participated in the formation of several small high-technology companies both in U.S.A. and Canada in several areas including power-supply, medical instrument and microprocessor-based systems. In 1968 he became a founding Director of EEC, a research and development-oriented consulting company, of which he was the President from 1974–1976 and for which he is currently an Executive Director.

He has participated widely in technical and professional organizations, including the IEEE, CSEE and CSPE. He has contributed to the Circuits and Systems Society of the IEEE, as a member of Adcom, General Chairman of the 1973 International Symposium for Circuits and Systems, IEEE Press Book Representative and Chairman of Future Conferences and Planning, and to the Solid-State Circuits Conference of the IEEE, as member of the Program Committee, member of the Executive Committee and Awards Chairman. He has also had related roles in the Multiple-Valued Logic Symposium and Technical Committee of the Computer Society of the IEEE. As well he has been the Chairman of the Publications Council of the Canadian Society of Electrical Engineering (CSEE), and a Director of the Canadian Society for Professional Engineers (CSPE).

In addition, he has acted as a program evaluator for the Canadian Engineering Accreditation Board, and as a consultant on behalf of EEC for a large number of Canadian and U.S. companies with specialization in the area of electronic and computer systems and emphasis on new-product design and startups.

Marius V.A. Hâncu was born in 1948 in Barlad, Romania. He received in Dipl. Ing. degree in electronics in 1971 from the Polytechnical Institute of Bucharest, Romania. From 1971 he was on the faculty of the Department of Electronics at the Polytechnical Institute, Bucharest, Romania. Since 1983 he has been involved in research on VLSI architectures in the Department of Electrical Engineering at the University of Toronto, Canada, where is a Ph.D. candidate. He will join Bell Northern Research, Ottawa, Canada, in January 1987, as a member of the scientific staff. His current research interests include VLSI architectures, VLSI communication and fault-tolerant design.

These algorithms, to which Rabin [23] has brought a major contribution, allow in many cases a massive reduction in the expected (average) time of running a computation, while knowingly accepting no speedup in the treatment of the worst cases. As such, their worst-case performance is not their main merit. Rather, they deliver in general a much better performance on the average on the full set of problem instances.

Probabilistic (randomized) algorithms are defined [23] as algorithms which "include a finite number of random steps, proceeding otherwise completely deterministically".

Our idea [8,9] has been to explore the reduction in computational time which could be obtained by running these algorithms in a VLSI environment, in comparison with known deterministic algorithms. The present paper extends the work [8] and [9], presenting a comparison between the proposed VLSI implementations of Rabin and Weide algorithms, respectively. It also provides a comparison between implementations providing exact and approximate solutions to the nearest-pair problem.

In doing this, we hope to attract the attention of VLSI architects to this valuable class of algorithms, which though introduced 10 years ago, are not yet popular beyond the computer science community.

Part of this relative lack of popularity is justified by the fact that probably all of the probabilistic algorithms proposed in the literature were not designed with a VLSI implementation in mind. As such, they have intricate data dependencies, which neither map easily into regular VLSI structures nor allow for the neighbour-to-neighbour interconnections favoured in a VLSI environment.

As being arguably the best known probabilistic algorithm, and one having characteristics making it amenable to implementation with regular VLSI structures (such as systolic arrays), we have chosen for the start of our discussion the one proposed by Rabin [23].

This algorithm solves the *closest-pair problem* in computational geometry in a K-dimensional space ($K = 2$, throughout our discussion, to follow Rabin's original paper [23]). This problem and solutions to it, are described in [5], [23], [25], [29] and [31].

As it is a relatively new field of computer science, in this context we would like to make some introductory remarks about computational geometry itself. Its foundations were organized by Michael Shamos in his Ph.D. thesis [25], after preliminary work by other researchers. As the object of computational geometry, in [15] it is mentioned that the study of "computational complexity of geometric problems within the framework of algorithm analysis and design", is important.

As some of the important classes of problems in computational geometry, Preparata and Lee, in the above-mentioned survey paper [15], include:

(i) convex hull problems (i.e. finding the smallest convex set containing a set of points);
(ii) intersections;
(iii) geometric-searching problems;

(iv) proximity and related problems:
- *closest pair* (the problem treated in this paper),
- all nearest neighbours (finding the nearest neighbour for each point in a given set),
- Euclidian minimum spanning tree (finding the tree that interconnects all the points in a set, with a minimum edge length),
- farthest pair;

(v) geometric optimization problems.

Now that we have provided this brief introduction to the object of computational geometry, we would like to define more formally the *closest-pair problem*. In this problem, for $S = \{x_1, \cdots, x_n\}$, n points in a K-dimensional space, it is required to find the nearest pair(s) x_i, x_j for which $d(x_i, x_i) = \min\ d(x_p, x_q)$ where d is the distance defined in that space, with $p, q \in [1, n]$ integers.

The closest-pair problem is not only a fundamental problem in computational geometry, it also has applications in image processing (the detection of potentially colliding objects on a radar screen, for example) and related fields.

In analysing the complexity of existing solutions to this problem, we must highlight the comparative performance of deterministic and probabilistic algorithms designed for solving it.

The straightforward deterministic approach to finding the nearest pair in a set of n points involves $C_n^2 = \frac{1}{2}n(n-1)$ (i.e. $O(n^2)$) distance calculations and comparisons to find the required minimum. Other, more involved, recursive (but also deterministic) algorithms, provided in [3] and [31], require only $O(n \log n)$ distance computations.

In comparison with the deterministic algorithm just mentioned, the probabilistic algorithms by Rabin [23] and Weide [29] are faster on the average. In both of the latter, the expected number of distance computations is only $O(n)$.

As an important feature of these two probabilistic algorithms, we should like to point out that they provide exact answers; that is, for further emphasis, the minimum distance is found with certitude. The only probabilistic aspect of the answer lies in the computational time required. While taking $O(n)$ on the average, it could take longer, but with small probability.

In principle, both algorithms by Rabin and Weide use an idea first enunciated by Yuval [31]: consider clusters of points and check for the nearest pair only within these clusters. A rectangular grid system is used to separate the initial set of points, in order to define the point clusters.

While the grid used by Rabin [23] is a two-level one (organized with two levels of resolution) the grid system proposed by Weide [29] is simpler, operating only with one level of resolution.

The probabilistic aspect of these algorithms is produced by the way the mesh-size of the separation grids is defined. This is a very important step, as the cluster size critically determines if finally the expected computational time will be $O(n)$, or more. Both algorithms randomly sample sparser sets of points (Rabin's algorithm) or distances (Weide's algorithm) in order to define the mesh-size of the separation grids.

A critical step in implementing this algorithm in $O(n)$ time is a sorting (or in effect, grouping) process, used in cluster formation. A software sort solves this problem in time $O(n \log n)$[1]. In view of the importance of the shorting process, we have explored a hardware systolic group sorter, similar to the systolic ripple sorter described in [16] and [26], but containing some new features. In [26] it is shown that the systolic ripple sorter is the most efficient available hardware solution for sorting and the most suitable for VLSI implementaion. We believe these qualities are retained in our modified version, which in fact implements a systolic grouper.

The cluster-definition process is followed by distance calculations (inside the clusters only) and comparisons. To remain $O(n)$ in area and time, we again use the high degree of parallelism offered by systolic structures. A *minimum-distance processor* performing the required functions is another novel systolic structure proposed in this paper.

The sampling process implied by the probabilistic algorithms used can be implemented in parallel with other operations in *probabilistic systolic arrays* [9] thus improving the compactness of the hardware implementation. Such arrays are introduced later in this paper. They have several other applications as previously mentioned.

2. Rabin's algorithm for the closest-pair problem

In presenting this algorithm, we will closely follow Rabin's original paper [23] on probabilistic algorithms.

As previously mentioned, the starting idea, suggested initially by Yuval [31], is to consider clusters of the points and to check for the nearest pair within these clusters. In order to obtain the clusters, Rabin uses two sparse subsets, obtained by a double sampling from the initial set S:

Given $S = \{x_1, \cdots, x_n\}$ the initial set of n points, $m_1 = n^{2/3}$ of them, sampled at random, define a sparser set $S_1 = \{x_i, \cdots, x_{i_{m_1}}\}$. An even sparser subset $S_2 = \{x_j, \ldots, x_{j_{m_2}}\}$ is obtained by sampling randomly $m_2 = (n^{2/3})^{2/3} = n^{4/9}$ points from S_1.

These are the only random steps in the whole process. The rest of the algorithm proceeds purely deterministically.

The sampled subset of points serves to define the *mesh-size* of the rectangular (in the planar case) lattice (grid), which decomposes the denser S into subsets (clusters of points located within a square of the grid). This takes place in two steps.

In the first step, we find $\delta(S_2)$, where $\delta(S_2) = \min d(x_i, x_j)$, where $x_{i,j} \in S_2$ (thus $\delta(S_2)$ is the minimum distance for the subset S_2). Because the number of pairs of points in S_2 is

$$C_{m_2}^2 = \tfrac{1}{2} m_2 (m_2 - 1) < m_2^2 = (n^{4/9})^2 = n^{8/9} < n$$

we can afford to compute all distances in this set and still be $O(n)$. The numbers of comparisons needed to establish the minimum distance in this subset is

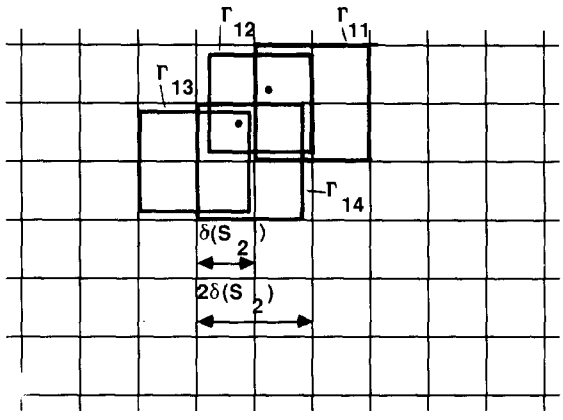


Fig. 1. Decomposing lattices.

$m_2 - 1$, resulting in $O(n)$ total expected time for computation of $\delta(S_2)$. This result was ensured by Rabin via a careful choice of sampling indexes m_1 and m_2, a choice which proves to be quite critical.

This basic distance $\delta(S_2)$ serves now to determine the mesh-size of the lattice used to decompose S_1 into clusters. Four lattices $\Gamma_{11}, \ldots, \Gamma_{14}$ are constructed, with the mesh size $2\delta(S_2)$. They can be obtained from each other by horizontal or vertical translations of distance $\delta(S_2)$.

The elementary $2\delta(S_2) \times 2\delta(S_2)$ squares in these lattices are represented in Fig. 1. It can be observed from the example of Fig. 1 that four lattices are necessary in Rabin's algorithm in order to deal with pairs of points located on both sides of one line in the basic grid which has a mesh size of $2\delta(S_2)$.

Denoting similarly $\delta(S_1) = \min d(x_i, x_j)$, $x_{ij} \in S_1$ (thus $\delta(S_1)$ is the minimum distance for the subset S_1), since $\delta(S_1) \leqslant \delta(S_2)$($S_1$ being obtained by sampling S_2), Rabin shows that the nearest pair, for points x_i, x_j in S_1, falls within the same square of at least one of the Γ_{1i}, $1 \leqslant i \leqslant 4$, which provides the important conclusion that we can limit our distance processing (d computations and comparisons) to clusters of points from S_1 located each in only one cell of any of the four lattices.

The four lattices $\Gamma_{11}, \cdots, \Gamma_{14}$ divide S_1 via the corresponding decompositions (the details are given later in this section):

$$S_1 = S_{11}^{(i)} \cup \cdots \cup S_{1k_i}^{(i)}, \quad 1 \leqslant i \leqslant 4$$

where $S_{1j}^{(i)}$ is the subset of S_1 located in a certain cell (square) of Γ_i, $1 \leqslant i \leqslant 4$.

Now the inter-point distances $d(x_p, x_q)$ for all $x_p, x_q \in S_{1j}^{(i)}$ (points in the same cell) are computed. The nearest pair, which has been shown above to be located in one of the cells, can then be obtained by comparing these distances with each other, in order to get $\delta(S_1)$, the distance corresponding to the closest pair in S_1.

To realize the clustering process in subset S_2, the original (true) coordinates of a point $x_i = (x_i, x_i)$ are replaced by the integral coordinates, on a $2\delta(S_2)$ scale,

(a_i, b_i), where $a_i = [x_{i_1}/2\delta(S_2)]$, $b_i = [x_{i_2}/2\delta(S_2)]$. Here [ ] means "integral part of...". Then we group together the points having, in the same lattice, the same integral coordinates. The formulas given above for a_i, b_i are valid only for one of the lattices, say Γ_{11}.

For the other three lattices, the equations should be modified in order to take into account the vertical and horizontal translations of size $\delta(S_2)$.

$$
\begin{aligned}
a_{i(\Gamma_{12})} &= \left[\left(x_{i_1} + \delta(S_2)\right)/2\delta(S_2)\right], & b_{i(\Gamma_{12})} &= \left[x_{i_2}/2\delta(S_2)\right], \\
a_{i(\Gamma_{13})} &= \left[x_{i_1}/2\delta(S_2)\right], & b_{i(\Gamma_{13})} &= \left[\left(x_{i_2} + \delta(S_2)\right)/2\delta(S_2)\right], \\
a_{i(\Gamma_{14})} &= \left[\left(x_{i_1} + \delta(S_2)\right)/2\delta(S_2)\right], & b_{i(\Gamma_{14})} &= \left[\left(x_{i_2} + \delta(S_2)\right)/2\delta(S_2)\right].
\end{aligned} \tag{1}
$$

It is to be observed that the absolute values of these integral coordinates do not matter. What is important, for comparison and grouping purposes, is their relative value.

After obtaining $\delta(S_1)$ via the above sequence of operations, its value will serve to decompose the initial set into clusters, repeating the same process described for S_1 in order to define $\delta(S)$, the distance between the closest pair of points in S.

An important observation can be made: if $N(\Gamma_i)$ is the total number of distance computations for the separate clusters defined by Γ_i, in [23] it is proved that the expected value (with high probability) of $N(\Gamma_i)$ is $O(n)$. This being valid for S, is all the more valid for S_1.

We must mention that the value of $\delta(S)$ in the original versions of Rabin's and Weide's algorithms is exact, not an approximation or an estimate. (This is because the random sampling controls only the clustering process and because the four lattices ensure covering of all pairs.) Note that this feature, while valid in the cases noted, is not necessarily valid for all algorithms referred to as probabilistic.

In some of our proposed VLSI architectures, perfection in the determination of the closest pair is fully conserved. In others, it is traded for more compact implementations destined for applications where a high enough confidence level (but not 100%) is considered sufficient in finding the minimum distance.

3. A VLSI architecture for the implementation of Rabin's algorithm

In order to obtain the subsets of true (real) point coordinates S_1 and S_2 from the original set S, a pseudo-random sequence generator (PRG) could be used in the selection process. A possible block diagram for this preprocessing section would be the one described in Fig. 2.

In terms of implementation, the 3 memories might be RAMs or recirculating shift-registers. The latter solution would be more compatible with the systolic character of the main processing blocks.

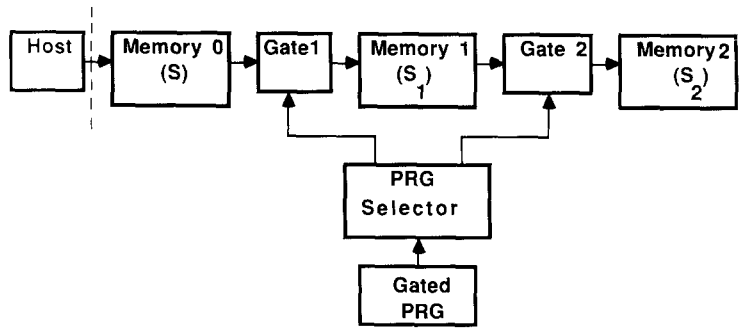


Fig. 2. Block diagram of the input processor for the implementation of Rabin's algorithm.

Linear-feedback shift registers [27] are deemed to be a good solution for a VLSI implementation of the PRG (Pseudo-Random Generator). They contain only shift-register cells and gates.

The sets of coordinates S_2, S_1, and S are provided to the computational parts of the processor in this sequence. The previous description of the algorithm should be sufficient to understand the block diagrams (in Fig. 3) leading sequentially to the nearest-pair distance computation.

The Integral-Coordinates Processor generates, based on the true (full) coordinates x_{i_1}, x_{i_2} of the points, the integral coordinates a_i, b_i, following the expressions indicated in equations (1). It is an arithmetic processor limited to some specific operations. As its implementation should not be a problem and it is not systolic, we will not detail it any further.

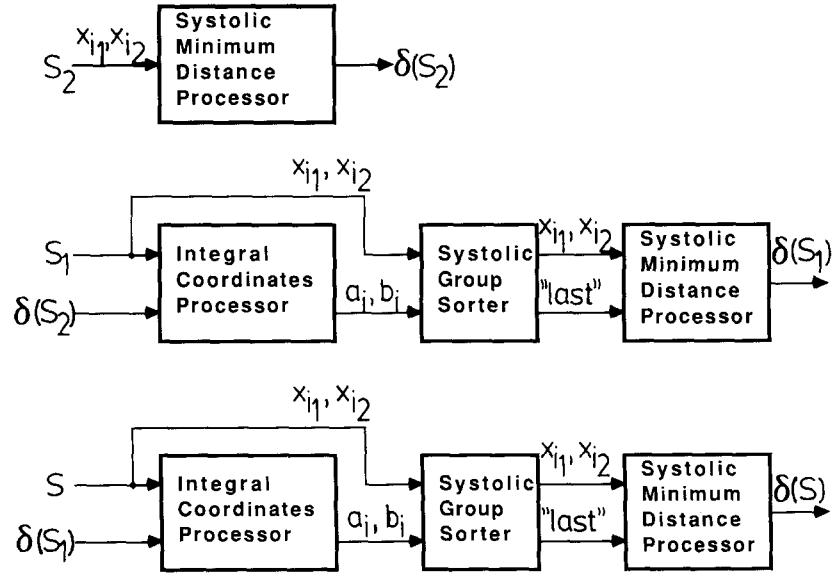


Fig. 3. Block diagram of the computational hardware.

An important comment to be made at this point is the fact that the integral coordinates (necessary for the task of point clustering) will have to travel throughout the final two systolic processing stages, in parallel with the corresponding true coordinates. In this way, after the minimum distance is finally detected, the coordinates of the corresponding points can be displayed, as well, for identification purposes.

In the following two sections of this paper, we will present the structure of the systolic group sorter and systolic minimum-distance processor, each of which contain several original contributions, and are used in implementing both Rabin's and Weide's algorithms.

4. The systolic group sorter

The clustering (grouping) of points located within the same square (cell) in a lattice will be done on the basis of their integral coordinates, rather than using their true (real) coordinates. Further it is assumed that the origin has been positioned appropriately to provide only positive coordinates. This is generally possible, since for our purposes only the relative coordinates matter. A bi-dimensional example is presented in Fig. 4.

Grouping is performed using a simultaneous-coincidence check on both a_i and b_i (the integral coordinates of a point). The operation is easily extended to an n-dimensional space.

The lattice cells have no preassigned order (or priority). Only belonging to a certain cell is important. Obviously, groups of points in different cells can have different numbers of members.

The reasons for choosing a hardware sorter have been exposed in the introduction.

A linear, bent, systolic array-processing structure (Fig. 5) has been chosen for its modularity, simplicity and locality of data transfers. The starting point has been the systolic ripple sorter described by Leiserson [16] in its original form, that

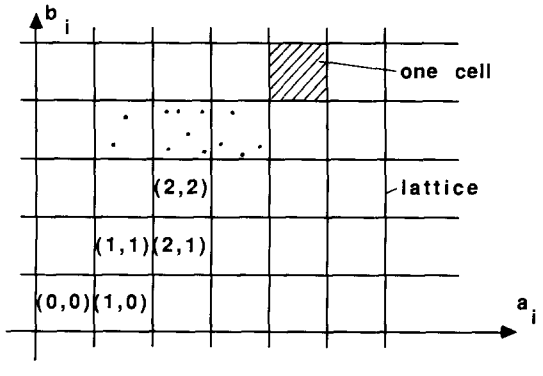


Fig. 4. Clusters and integral coordinates.

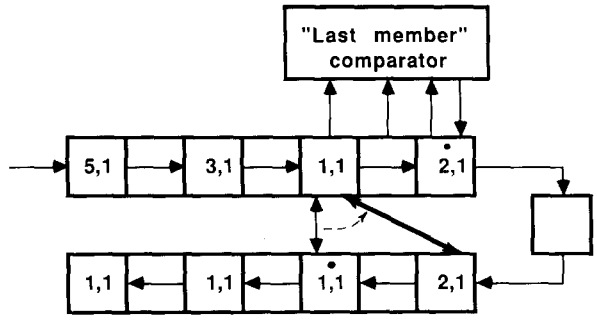


Fig. 5. The systolic group sorter.

of a priority queue, and to which Shin et al. [26] have brought contributions. However, in our case the design has been substantially modified. In [26] an order relationship (greater than) is the basis for the pure sorting process, while in our design new members (points) gradually join a cluster based on the value of associated 'keys' (or 'labels')—the integral coordinates.

A cluster collects its members (points and subclusters), starting from a nucleus (which could be represented initially by just a single point) which grows by accumulating members travelling through the array. This process can be likened to the growth of a snowball. Accordingly, snowball-array might be an appropriate name for this clustering (grouping) structure.

A marker (flag) bit for the 'last' point in the subcluster (as present in the systolic sorter) is generated, in our design, by a specialized comparator. A multiphase clock should be used to order the succession of basic operations in the array:

- shift (all groups);
- compare integral coordinates for grouping purposes;
- enable context exchange between PEs (processing elements) based on comparison results;
- reposition the 'last' flag.

Basically, the PEs contain shift-register cells, comparators and swap (exchange) circuits. At the array's right extremity, a separate comparator (Fig. 5) assigns the 'last-in-a-group' flag (represented by a dot), setting the corresponding bit carried by the current last member of a subcluster (corresponding to those points having the same integral coordinates).

In Fig. 5 we show a schematic representation of this array. In each PE, we have represented only the integral coordinates, because only these are significant to the grouping process. However, when a content exchange takes place, the true coordinates move together with the integral coordinates. All the connection links should be dimensioned accordingly.

The data (x_{i_1}, x_{i_2}, i.e. point coordinates) are input every other clock pulse. The reason for 'spaces' will be outlined later. The data move systolically to the right on the upper segment of the array, and to the left on the lower segment of the

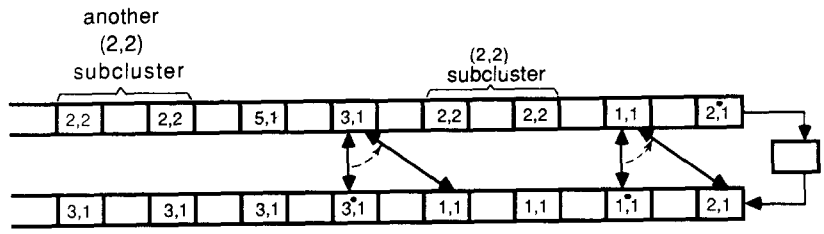


Fig. 6. Clustering of points in the systolic group sorter.

array. As described in [26], the processing is 'I/O overlapped', i.e. "upon completion of the input operation, the first sorted item is ready for output" [26].

The array is bent, to make possible coordinate-coincidence comparisons and point exchanges required to group together all members of a cluster. The idea is to add new members at the tail end of the group while it is travelling on the lower part of the array. The (integral) coordinates of a last member of a group are continously compared for coincidence (Fig. 6) with those of the point whose coordinates are located, at the time, in the PE just above the one containing the current last member. Upon coincidence, the coordinates of a nonmember (which immediately follows the current group) are exchanged with those of a future member, located just above the current last element and having, of course, the same integral coordinates. Now, the reasons for interspersing the data with spaces should be clear [12]: that is to avoid the lack of interaction between data in the two rows of the array which 'fly by' each other.

In Fig. 6, two (the rest are not enabled via the 'last' bit) such typical membership-test comparisons and exchanges are shown to be performed. Here a thin arrow indicates a coincidence test with the current last member of a subgroup, while a thick arrow indicates content exchange, and a dotted arrow that an exchange is enabled. It is seen that stopping the data shift periodically for some clock phases (while the comparisons, content exchanges and flag repositioning are done), will be sufficient to perform this sequence of operations and that only local communication links need be provided, a basic requirement for a systolic design. The corner cell should be provided with the same types of connections.

Schematically, this can be represented as in Fig. 7. It should be clear by now that the 'last' flag enables all membership (coincidence) tests and the associated exchanges. As such, the 'last member' comparator should indeed be located at the right end of the upper row.

The number of cells in each row should be at least n (n being the total number of points in the initial set S), or otherwise the members travelling on the lower row will not meet all their 'relatives' travelling on the upper row and will fail to enrol them at the tail of the group.

Because of the inherent parallelism of this structure, all points are allocated to the appropriate cluster after $2n$ clocks, when the last point leaves the array. As such, the process in $O(n)$ both in area and time.

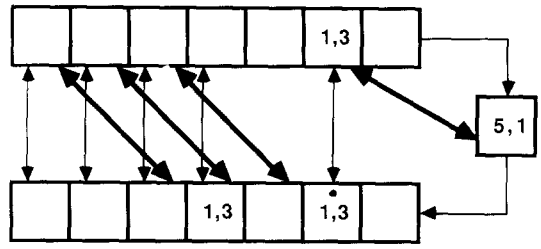


Fig. 7. Connections in the systolic group sorter.

A point that has not yet been mentioned is that in the process of movement to the left in the lower row of the array, a (sub)cluster loses some of its leading (front) members, as a result of exchanges occuring at the tail of the previous group. This, however, happens less and less frequently as it travels to the left and these losses are compensated for later in the process, when the lost members are all recollected at the tail of a fully assembled cluster.

5. The systolic minimum-distance processor

This processor is organized as a separate systolic array, accepting as input the true coordinates (and the 'last' flag) of the points clustered by the group sorter (Fig. 3). Its operation is based on the idea that not all distances between points belonging to the same cluster (lattice cell) are needed externally. Thus in running distance computations and comparisons in a systolic fashion, only relative minima need be kept, along with the associated point coordinates.

The distances from each point to all other points within the same cluster are computed, but in a parallel systolic fashion, so that even if in all clusters

$$\sum_{\substack{\text{all}\\ \text{clusters}}} \tfrac{1}{2} n_{\text{cluster}_i}\left(n_{\text{cluster}_i} - 1\right)$$

distance computations are needed for each decomposition, in a given lattice, finally only $O(n)$ time is necessary.

The linear, bent, shift register is again the basic component (Fig. 8). It supports the movement of the coordinates and of the 'last' bit. Other types of PEs are included as well: D computes the distance between two points (x_{i_1}, x_{i_2}) and (x_{j_1}, x_{j_2}), while $D_{\min}$ (essentially a comparator) transfers to its output the smallest distance from the two presented at its inputs. In this systolic array, all distances (minimal or not) are transferred together with the coordinates of the corresponding pair of points, for later reference. This is not explicitly presented in Fig. 8 for reasons of simplicity.

Data are input with alternating spaces, for the same reasons which apply in the systolic array described in the previous section. The $D_{\min}$-PEs are clocked too; they contain distance comparators and registers. All PEs work synchronously, on different clock phases. Note that particular care must be taken with the timing of feedback connections.

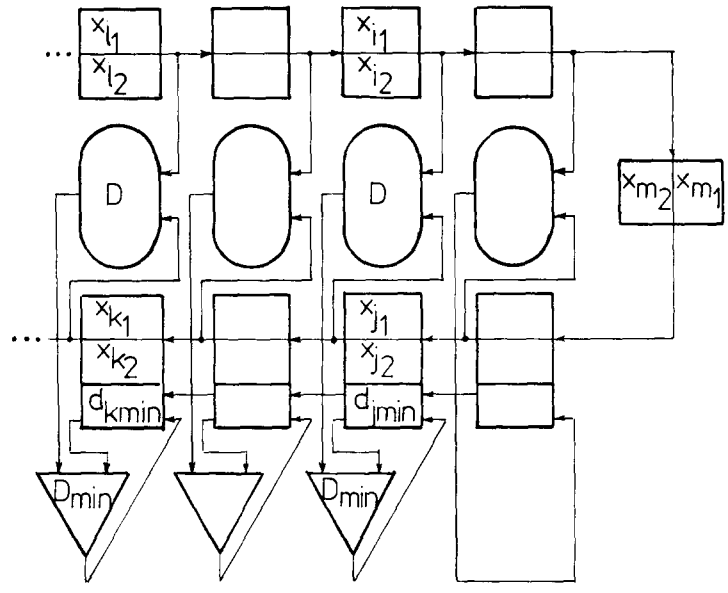


Fig. 8. Block diagram of the systolic minimum-distance processor.

This bent configuration ensures that each member of a cluster, moving on the lower section of the shift register, faces, step by step, all the other members of that particular cluster, travelling on the upper rail, interacting with them and thus generating the associated distances.

The $d_{\min}$ which are generated, as a result of the type of processing implied, are local (relative) minima and move in step with the associated coordinates in the lower part of the array.

A point (x_{j_1}, x_{j_2}) carries with it the minimal distance to all the points in its cluster with which it has already interacted. The distance between (x_{i_1}, x_{i_2}) and (x_{j_1}, x_{j_2}), computed by the D-processor, in Fig. 8, serves to update the minimal distance from (x_{j_1}, x_{j_2}) to the rest of the points in the clusters. If the new distance is smaller than the old value propagated through the $d_{\min}$ connections, the old is replaced.

Very importantly, the 'last' labels move together with point coordinates (Fig. 9) in order to prohibit interaction (distance computations) between points located in different lattice cells, by a selective enabling of D-processors which is described below. The observations made concerning the number of cells in each row apply for this array as well.

It can be seen that only points located close to the right end of the array interact, in a $\supset$-shaped area. The size of this area of interaction (defining the current cluster for which distances are being computed), varies with the current n_{cluster} (number of points which are members of a cluster).

At the extreme left of the lower row, a final comparator (not shown), having some memory registers, sequentially compares all $d_{j\,\min}$ serially coming to it as the smallest distances in each of the clusters, retrieving δ, i.e. $(d_{\min})_{\min}$ for the set of points in question (S_1 or S_2, in our case).

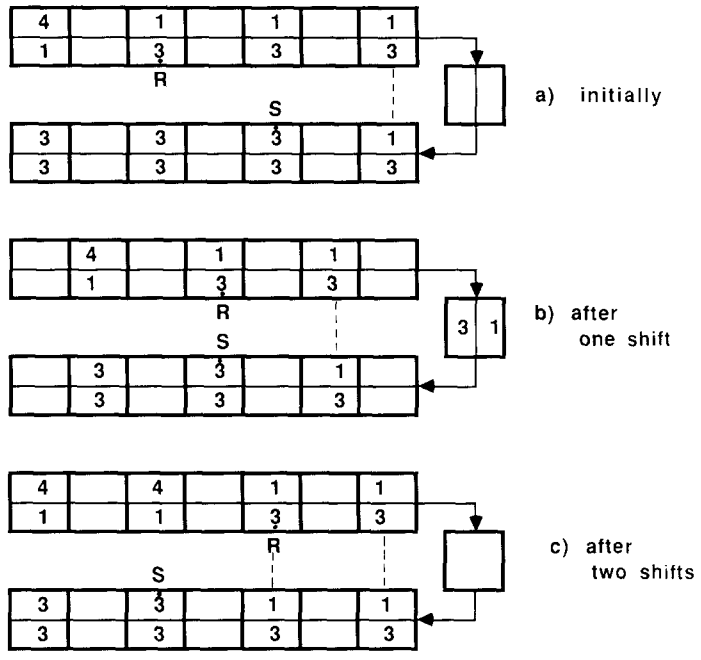


Fig. 9. Defining the active zone of interaction.

The 'last'-flag-detection process is the one defining which D-cells are active at a particular clock beat. Three cases are possible, depending upon the relative position of the 'last' point in the current group (the one temporarily located around the right end of array) and in the previous group (for which all distance computations have been already finished). The 'last' bits in the two groups are denoted by R (current group) and S (previous one). In Fig. 9, only the shift-register skeleton of the array is represented. Dotted lines indicate pairs of points in interaction (i.e. for which the D-PEs are enabled).

To label (in the example of Fig. 9) the components of a group, as they are input from the group-sorter systolic array, we have been using the integer coordinates, even if these coordinates are not carried or used in the actual distance (D) processor. The order of components of the group is not modified in the D-processor.

It should be mentioned that pairs of points fully located in either the upper or lower part of the shift-register part of the array do not interact.

As R and S travel (as 'last' bits) through the array, together with the true coordinates (which have not been explicitly represented), they are latched at the inputs of the D-type PEs, causing them to be enabled or disabled through a simple control function ($S = 1$, travelling on the lower row enables or sets the D-processors, while $R = 1$ moving on the upper row resets or disables them). In order for D_{min}-comparators to operate properly when the corresponding D-processor is disabled, the output will be forced to provide an 'out-of-range' value.

This structure too is $O(n)$ in area and time.

6. Weide's algorithm for the closest-pair problem and the VLSI architecture implementing it

While being less known, Weide's algorithm [29] is considerably simpler than Rabin's. As a result, the derived architecture proposed in this section is faster and allows for a simpler implementation than known solutions based on either Rabin's algorithm or purely deterministic algorithms.

Weide's algorithm also makes use of Yuval's [31] idea of considering clusters of points, and checking for the nearest pair only within these clusters.

A basic rectangular lattice Γ_1 with mesh size 3δ (and two replicas Γ_2, Γ_3 obtained by translation by δ, Fig. 10) is used to decompose S into subsets (clusters) of points. The random part of Weide's algorithm refers only to the process of defining δ, and thereby, implicitly, the mesh-size. As mentioned in the introduction to this paper, in contrast to Rabin's algorithm, which uses a hierarchical, two-level lattice to define the 'bins' in which the clusters are located. Weide's algorithm uses a *single mesh-size* for its lattices.

We would first of all like to mention that the definition of mesh-size is different in Weide's algorithm than in Rabin's. The unit length δ (which determines the mesh-size 3δ) is obtained as a minimum of n distances chosen at random between points in S (in fact, from a subset S_{nd}). The block diagram of the input processor (Fig. 11) is thus different from the one suggested for Rabin's algorithm (Fig. 2).

After the transfer from the host computer of point coordinates (Fig. 11) into Memory 0 (containing S), the subset S_{nd} is obtained by a sampling process controlled by a pseudo-random generator (PRG) and stored in Memory 1. To obtain n different distances, as anticoincidence circuit is used to avoid repetition of point pairs, under the control of a counter.

A Systolic Minimum-Distance Processor (such as the one described in Section 5) is used (Fig. 12) to obtain the unit length δ from the coordinates of the points in the sampled set S_{nd}.

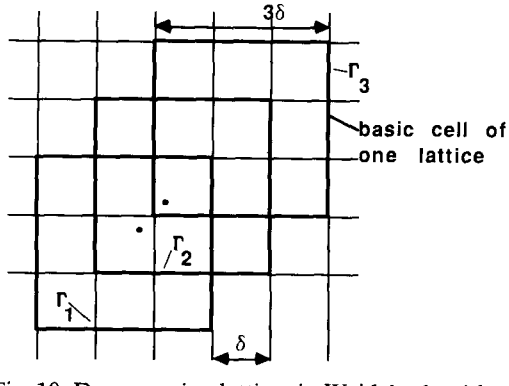


Fig. 10. Decomposing lattices in Weide's algorithm.

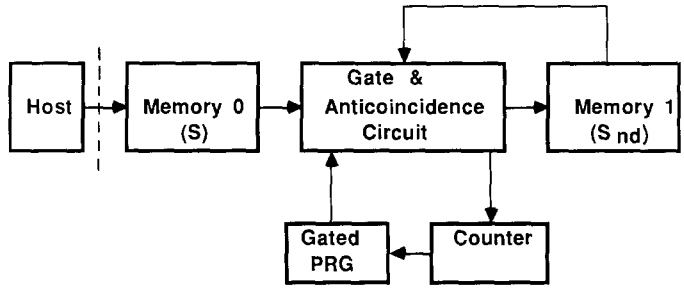


Fig. 11. Block diagram of the input processor for Weide's algorithm implementation.

The points in each cluster are distinguished (labelled) by their integral coordinates (Fig. 4), which are obtained using the Integral-Coordinates Processor (Fig. 12).

As mentioned in the description of Rabin's algorithm, this processor is an arithmetic processor limited to only a few operations. However, the formulas (equations (2)) used by this processor in order to derive, from the original (true) coordinates of a point x_{i_1}, x_{i_2}), the integral coordinates in a 3δ scale, (a_i, b_i), are different from those given for Rabin's algorithm (in equations (1)):

$$\begin{aligned} a_{i(\Gamma_1)} &= \left[x_{i_1}/3\delta\right], & b_{i(\Gamma_1)} &= \left[x_{i_2}/3\delta\right], \\ a_{i(\Gamma_2)} &= \left[(x_{i_1}+\delta)/3\delta\right], & b_{i(\Gamma_2)} &= \left[(x_{i_2}+\delta)/3\delta\right], \\ a_{i(\Gamma_3)} &= \left[(x_{i_1}+2\delta)/3\delta\right], & b_{i(\Gamma_3)} &= \left[(x_{i_2}+2\delta)/3\delta\right]. \end{aligned} \tag{2}$$

In equations (2), the expressions for the integral coordinates of a point in lattices Γ_2 and Γ_3 are obtained taking into account the horizontal and vertical translations of size δ, respectively 2δ, by which these lattices are obtained from the original lattice Γ_1. As was mentioned in Section 2, multiple lattices are necessary in order to deal with pairs of points located on both sides of one line in the basic grid, which has a mesh-size of 3δ in Weide's algorithm (as opposed to 2δ in Rabin's algorithm, but where the δ definitions differ in the two cases).

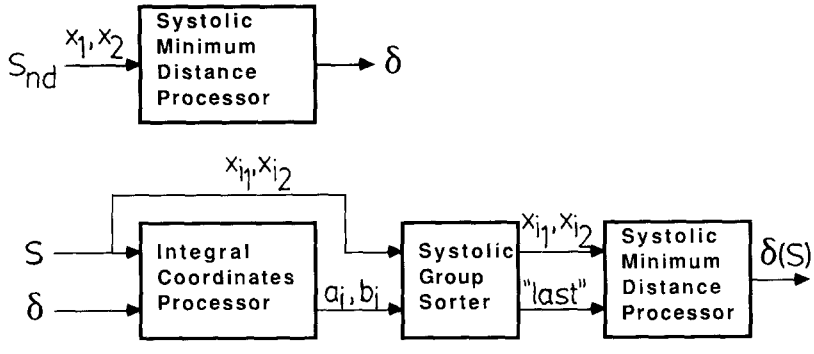


Fig. 12. Computational hardware-block diagram (for Weide's algorithm implementation).

The clustering process corresponding to the block diagram shown in Fig. 12 is implemented within a Systolic Group Sorter identical to the one proposed in Section 4.

After clustering, a Systolic Minimum-Distance Processor (similar to the one described in Section 5) compares all possible distance within all clusters. The process is repeated three times to cover all three partially overlapping grids. As mentioned previously, overlap is necessary in order to cover pairs of points which might fall only partially in one basic square ('bin') of a certain grid even if the points are very close to each other. The closest pair of points is thus detected.

The solution proposed for implementing Weide's algorithm is decidedly more compact than the one required by Rabin's algorithm. A quick comparison between the block diagrams described in Figs. 12 and 3 will make this quite clear.

7. Exact and approximate solutions to the closest-pair problem

As was already stated previously, the algorithms by Rabin and Weide indeed both provide the smallest distance in the set of points involved.

However, as a result of practical difficulties and limitations arising in the VLSI implementation of these algorithms, the exact value of the smallest distance might not be obtained with complete certitude, but only with a controllably high probability.

First of all, one has to consider that in the theoretical proofs of probabilistic algorithms, the existence of an ideal random generator is normally implied. However, in terms or practical implementation, one normally resorts to using pseudo-random generators with finite periods. As a result of this, an important design criterion will be to appropriately define the period length of the pseudo-random generator employed, in order to attain the exact result with a high confidence level (a given high probability).

Another design factor able to influence the probability (or eventual certitude) of indeed finding the closest pair of points (as opposed to just one of the smaller distances) is the size of the two systolic arrays proposed for inclusion in our design.

If, first of all, we must compromise on the size of the systolic group sorter by reducing it, (for reasons of area and cost overheads), then the process of clustering will still take place, but will be incomplete. If, for example, the size of the systolic array described in Fig. 6 is reduced from $2n$ cells to fewer, some members of a cluster travelling on the upper rail will never have the occasion to join their own cluster, travelling on the lower rail, for the simple reason that by the time a potential member is placed on the upper rail, the (sub)cluster has been removed from the lower end of the array. Thus, some larger clusters will be inevitably be fragmented, and we might end up either with a number of computations larger than $O(n)$, or a reduction in the probability of finding the smallest distance from 1 (certitude) to less than 1. This latter effect will be generated by the fact that subclusters still in the same square of the grid (having

the same integral coordinates) will not be united. As a result of this, distances between some points still having the same integral coordinates, but placed in different subclusters as a result of reducing the size of the grouper, will not be considered in the minimum distance computation. Thus, the probability of missing the smallest distance increases as the size of the systolic group sorter is reduced.

Extending the length of the systolic grouper to its ideal value ($2n$) should not be a problem, because the cells used are not very complicated (they perform only shift, comparison and exchange operations).

However, the size of the systolic minimum-distance processor is a more delicate problem. This processor contains (Fig. 8) at most n D(distance)-processing elements, each of which can be quite complicated, depending the definition of distance in use. They are basically arithmetic processors, and, by nature potentially more complicated than the shift-register cells and the simple comparator cells which are a small part of the systolic minimum-distance processor but dominate the systolic grouper.

First of all, the reader might have recognized that if the systolic minimum-distance processor has at least n D-processing elements, (which could be prohibitive in terms of area), we would obtain the spectacular result of finding the minimum distance without the use of any of the other proposed blocks and without using any probabilistic algorithms at all, and in $O(n)$ time too. And as a bonus, the result would be obtained as well with 100% confidence. The $O(n)$ time is justified simply by the existence of $O(n)$ D-processing elements computing in fact in parallel, while a total of $O(n^2)$ total distances must be calculated.

In other words, the proposed systolic minimum-distance processor maximized in effect in size to n D-processing elements, is a perfectly complete deterministic solution to the nearest-pair problem. It would still provide $O(n)$ response time for $O(n)$ area, and provide the exact smallest distance.

But in many applications, the designer is willing to exchange certitude for a given (high) confidence level in finding the smallest distance, if the implementation costs become considerably lower. In our case, a major contribution to the size of the VLSI chip computing the nearest-pair distance is brought by the systolic minimum-distance processor, for the reasons mentioned above. Thus we can radically reduce the overall size ot this processor, as shown in the following, from $O(n)$ to $O(\sqrt{n})$ and still obtain the exact value with a given high probability, by using the full probabilistic design presented in this paper (as shown in Fig. 3 for the Rabin-motivated design and in Fig. 12 for the Weide-motivated design), by including a 'shrunken' systolic minimum-distance processor.

This approach is justified by the fact that in both algorithms, it is guaranteed that on the average not more than $O(n)$ distance computations would result. This being valid for the whole set of points, it is all the more valid for a cluster located in any of the grid squares. Because all the distance computations should be performed within each cluster for both randomized algorithms, it is clear that the expected number of points present in a cluster, and leading to $O(n)$ distance computations, is $O(\sqrt{n})$. This is because $C^2_{\sqrt{n}} = \frac{1}{2}\sqrt{n}(\sqrt{n} - 1)$ is $O(n)$.

An expected number of points in each cluster of $\sqrt{n}$ can be processed with a high probability by only $O(\sqrt{n})$, say $\sqrt{n}$, D-cells in the systolic minimum-distance processor. When we say with high probability, we understand that some clusters might have more than $O(\sqrt{n})$ elements and thus would not be accomodated (but only with a small, controllable probability) by a systolic minimum-distance processor of size $\sqrt{n}$ only.

If this loss in the certitude of finding the smallest distance is unacceptable in a certain class of applications, we have two solutions, both $O(n)$ in area and average time but with 100% in the confidence level of finding the correct minimal distance:

- a probabilistic one, using the same algorithms (Rabin's and Weide's) but replacing the systolic form of the minimum-distance processor by a sequential form, described in the following;
- a deterministic solution, using exclusively the systolic minimum-distance processor of maximum size (containing n D-type processors) without any input pre-processor, as discussed earlier in this section.

The sequential minimum-distance processor would simply substitute the systolic minimum-distance processor in the block diagrams of Fig. 3 (for Rabin's algorithm) and Fig. 12 (for Weide's algorithm). It is a simple arithmetic uniprocessor, able also to scan sequentially all the clusters obtained at the output of the systolic group sorter. These clusters are initially stored in adjacent locations of a random-access memory (Fig. 13) in the same order as they are output by the systolic group sorter. The 'last' point in a cluster in this serial-storage scheme is again flagged. This helps the minimum-distance processor in detecting the end of a cluster. Using a simple program, the sequential minimum-distance processor

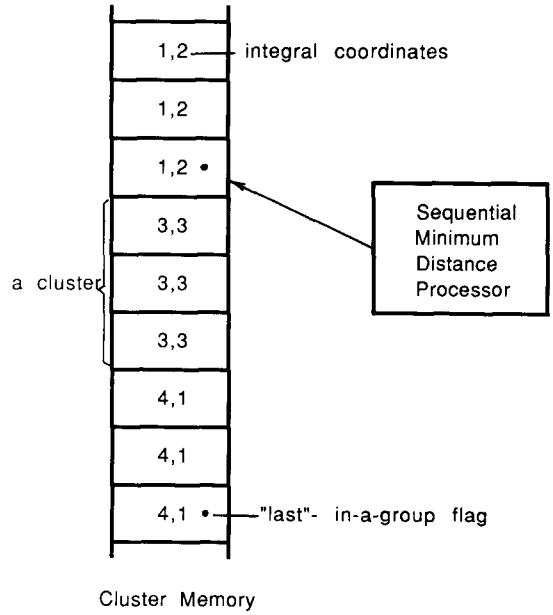


Fig. 13. Scanning the cluster memory using the sequential minimum-distance processor.

scans gradually all the point in a cluster, computing their distances to all subsequent points in the same cluster until the presence of a 'last' flag is detected. Using comparisons, only the current minimum-distance from all clusters already scanned is kept.

As previously mentioned, the grid definition in Rabin's and Weide's algorithms ensures that the expected total number of distances for all the clusters thus defined is O(n). At the same time, the sequential minimum-distance processor can be made to occupy considerably less area than its systolic counterpart of maximum size (n) as it contains one D-(distance)-processing element, while both have an O(n) response time. However, it is far less regular and modular in structure.

8. Using probabilistic systolic arrays in implementing the minimum-distance processor

Probabilistic systolic arrays have been introduced by us in an earlier paper [9]. While reviewing their principles, we will do so only to the extent that this is necessary to understand their use in the present paper.

While studying how to implement probabilistic algorithms in VLSI structures characterized by high regularity, modularity, and concurrency, such as systolic arrays, it occurred to us that it would be natural to try to implement direct randomization of systolic processing.

Systolic arrays [12] have seen increasingly widespread use as highly parallel computing organizations suitable for VLSI implementation of specialized processors for various applications.

In a majority of available designs, control of the movement of data is synchronous [17]. Characteristically, a multiphase clock is used to direct the alternation of incremental data transfer and processing steps. Thus, the clock line is the only acceptable global signal connection, all data transfers being of a local neighbour-to-neighbour kind.

Under these assumptions, data flow within a systolic array is in highly organized and predictable (deterministic) streams, where data interactions occur in the PEs (processing elements), 'while data items travel' through the array. For each data item, the inherent assumption is made that it can be localized at any moment in time and space. This means that we can specify the propitious moment at which to inject data into the array (from the interface with the host computer) and, as well, the exact operand to be processed by each PE at a particular time. Correspondingly, we are able to predict the full composition and timing of the stream of output results.

While this deterministic data-propagation and processing approach is quite appropriate in many applications, there exist applications where there is a need for a probabilistic (randomized) treatment of data.

Thus, when implementing a probabilistic algorithm, some random process, such as random sampling of points or distances, as in the algorithms used in this paper, must be added to an otherwise deterministic computation.

Therefore if large amounts of data are to be processed, (occuring in sets with a large number of points, such as in the context of the nearest-pair problem discussed in this paper) a systolic array will remain the VLSI solution of choice. This is especially true for reasons of intercompatibility, in the event that the rest of the VLSI chip under design is conceived as a systolic structure.

In [10], a proposal was made for systolic structures able to perform such random operations. There it was suggested that one could 'randomly modulate':
- the data streams,
- the functions performed by the PEs.

The first alternative only is used in the context of this paper. In the modified systolic array, the systolic rhythmicity of the data flow, and the functions of each PE, are conserved, but the operands are sampled from the initial data streams on a random basis. This process is equivalent to the modulation of the data stream with a binary random sequence.

The intended purpose of probabilistic systolic arrays in the context of this paper is to eliminate, especially for the Weide's algorithm implementation (the more compact one), the input processor (represented in Fig. 1), and to incorporate the sampling performed by it in the operation of the systolic minimum-distance processor which determines δ (Fig. 12), the basis distance defining the grid size of 3δ. We would like to remind the reader that δ is obtained as the smallest distance in a set of n distances sampled at random from the total set of C_n^2 distances between the pairs of points in the point set.

Sampling (or generating) only n of the possible distances can be implemented within the systolic minimum-distance processor (Fig. 8) in two ways:
- sampling of points (Fig. 14(a)),
- sampling of distances (Fig. 14(b)).

Sampling of points is accomplished by conditioning the input register with a pseudo-random binary sequence (Fig. 14(a)) which gates the coordinates of the input points into the input register (IR), preceding the first shift-register cell. For the value 0 in the pseudo-random sequence, the input register is loaded with some 'out-of-range' coordinate, which subsequently overranges the D-processors. As a result of comparisons, the corresponding distance values, which result in a value naturally larger than any real distance value, are dismissed.

For the value 1 in the pseudo-random sequence, the input coordinates are gated directly (undistorted) into IR. Subsequently, they generate 'in-range' distances, which enter the comparison process normally.

The pseudo-random sequence must be started in synchronism with the input of the coordinates of the n points. The duty factor (ratio of 1s in a full binary sequence of n bits) is defined by the necessity of generating only n distances by interaction on n points. As such, $n = \frac{1}{2}m(m-1)$, which gives m, the number of 1s in a period of the pseudo-random sequence. If m is increased too much, outside the range given by the above equation, the resulting grid might become too dense, leading to too many computations in Weide's algorithm.

Sampling of distances (Fig. 14(b)) requires the propagation of a binary pseudo-random sequence in a shift-register enabling the D (distance)-processors.

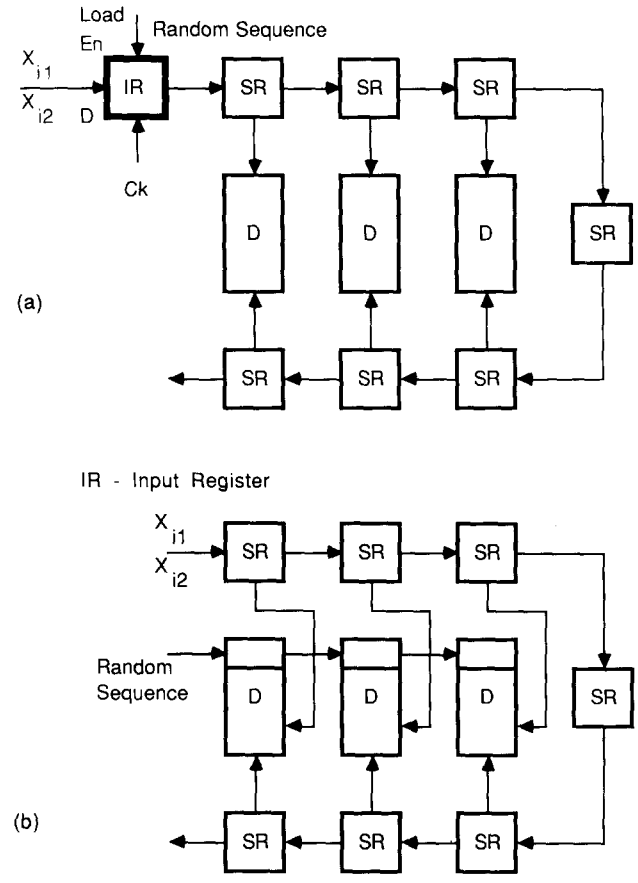


Fig. 14. Probabilistic systolic minimum-distance processor: (a) using sampling of points; (b) using sampling of distances.

For example, the presence of a 0 in the sequence will disable (overrange, in fact) the *D*-processors such that their current output will be too large to be considered in the comparisons leading to δ. At the same time, 1s will generate 'in-range' distances. The number of 1s in a period of the random sequence should be determined by the condition that while the input coordinates are already on the lower rail of the array, (that is, when distance computations really take place) n activations of the *D*-processors should occur. However, further discussion of this point is beyond our need in the present context.

Other envisaged applications for the probabilistic systolic arrays, beside the execution of randomized algorithms lie in the areas of data encryption and digital signal processing. However, they will not be explored further here.

9. Conclusions

In this paper, an attempt has been made to propose and advocate the use of probabilistic (i.e. randomized) algorithms in VLSI architectures. As examples of

the general approach, specific VLSI structures implementing randomized algorithms for computational geometry have been proposed and discussed in some detail.

Probabilistic algorithms are a topic of active research for computer scientists. A number of such algorithms are already available for various applications, and we expect to see the emergence of such algorithms designed specifically for easy VLSI implementation. Furthermore, we expect a gradual acceptance in VLSI design of those probabilistic algorithms which do not provide exact answers, but only estimates, valid to a controllable extent. Such acceptance depends on a compromise (probably application-specific) between chip area, time of computation, chip reliability and actual required accuracy.

In order to maximize the throughput of the proposed structures, new systolic designs have been introduced in this paper, including a systolic group sorter and a systolic minimum-distance processor.

They can have, we believe, multiple other uses in such areas as:
- data base applications,
- pattern clustering,
- computational geometry.

For direct implementation of randomization of data in systolic structures, we have briefly introduced probabilistic systolic arrays, in which data streams or operators are modulated by a random sequence.

Besides the use of such probabilistic systolic arrays for the case example in this paper, namely the solution of the nearest-pair problem in computational geometry using a probabilistic algorithm, we suggest the importance of further study of their use in data encryption and digital signal processing applications.

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms* (Addison-Wesley, Reading, MA, 1983).

[2] Bentley, J.L. and M.I. Shamos, Divide-and-conquer in multidimensional space, Proc. 8th Annual ACM Symp. Theory of Computing, 1976, pp. 220–230.

[3] Bentley, J.L. B.W. Weide and A.C. Yao, Optimal expected time algorithms for closest-point problems, Proc. 16th Allerton Confer. Commun., Control and Computers, 1978, pp. 843–851.

[4] Chazelle, B., Computational Geometry on a systolic chip, *IEEE Trans. Computers* **C-33** (9) (1984) 774–785.

[5] Fortune, S. and J.E. Hopcroft, A note on Rabin's nearest-neighbour algorithm, Cornell Computer Science Report TR78-340, Cornell University, Ithaca, NY, 1978.

[6] Fussell, D. and P. Varman, Fault-tolerant wafer-scale architectures for VLSI, Proc. 4th Internat. Symp. Computer Architecture (1982) 190–198.

[7] Guibas, L.J. and F.M. Liang, Systolic stacks, queues, and counters, in: Proc. Confer. Adv. Res. VLSI, MIT, Cambridge, MA, 1982, pp. 156–164.

[8] Hâncu, M. and K.C. Smith, Systolic arrays implement a probabilistic algorithm for the nearest-pair problem, *J. VLSI Comput. Syst.*, 1985.

[9] Hâncu, M. and K.C. Smith, A systolic implementation of Weide's probabilistic algorithm for the nearest-pair problem, Proc. 1985 Canadian Confer. VLSI, 1985, pp. 209–212.

[10] Hâncu, M. and K.C. Smith, Probabilistic systolic arrays, Proc. IEEE Confer. Computer Design: VLSI in Computers—ICCD'85, 1985, pp. 242–243.
[11] Knuth, D.E., *The Art of Computer Programming. Vol. 3. Sorting and Searching* (Addison-Wesley, Reading, MA, 1973).
[12] Kung, H.T. and C.E. Leiserson, Systolic arrays (for VLSI), in: I.S. Duff and G.H. Stewart, Eds., *Sparse Matrix Proceedings 1978* (SIAM, Philadelphia, PA, 1979) 256–282.
[13] Kung, H.T., Let's design algorithms for VLSI systems, Proc. Caltech Conf. on VLSI, 1979, pp. 65–90.
[14] Kung, S.-Y., H.J. Whitehouse and T. Kailath, Eds, *VLSI and Modern Signal Processing* (Prentice Hall, Englewood Cliffs, NJ, 1985).
[15] Lee, D.T. and F.P. Preparata, Computational geometry—a survey, *IEEE Trans. Comput.* **C-33** (12) (1984) 1072–1101.
[16] Leiserson, C.E., Systolic priority queues, Proc. Caltech Conference on VLSI, 1979, pp. 200–214.
[17] Leiserson, C.E., Systolic and semisystolic design, Proc. IEEE Internat. Confer. Computer Design: VLSI in Computers (ICCD'83), 1983, pp. 627–632.
[18] Liu, H.H. and K.S. Fu, architectures for minimum-distance classification, Proc. IEEE Internat. Confer. Computer Design: VLSI in Computers (ICCD'83), 1983, pp. 547–552.
[19] Mead, C.A. and L.A. Conway, *Introduction to VLSI Systems* (Addison-Wesley, Reading, MA, 1980).
[20] Moldovan, D.I., On the design of algorithms for VLSI systolic arrays, *Proc. IEEE* **7** (1) (1983) 113–120.
[21] Ni, L.N. and A.K. Jain, A VLSI systolic architecture for pattern clustering, Proc. 1983 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, 1985, pp. 110–117.
[22] Rabin, M.O., Probabilistic automata, *Information and Control* (1963) 230–245.
[23] Rabin, M.O., Probabilistic algorithms, in: *J.F. Traub, Ed., Algorithms and Complexity: New Directions and Recent Results* (Academic Press, New York, 1976) 21–19.
[24] Rabin, M.O., Complexity of computations, turing award lecture 1976, *Comm. ACM* **20** (9) (1977) 625–633.
[25] Shamos, M.I., Computational geometry, Ph.D. thesis, Yale Univ., New Haven, CT, 1978.
[26] Shin, H., A.J. Welch and M. Malek, I/O overlapped sorting schemes for VLSI, Proc. ICCD'83, IEEE Internat. Confer. Computer Design: VLSI in Computers, 1983, pp. 731–734.
[27] Tausworthe, R.C., Random numbers generated by linear recurrence modulo two, *Math. Comp.* **19** (1965) 201–209.
[28] Ullman, J.D., *Computational Aspects of VLSI* (Computer Science Press, Rockville, MA, 1984).
[29] Weide, B.W., Statistical methods in algorithm design and analysis, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1978, pp. 3.58–3.62.
[30] Weide, B.W., Personal communication, 1986.
[31] Yuval, G., Finding nearest neighbours, *Info. Process. Lett.* **5**(3) (1976) 63–65.