Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

VLSI Designs for the Solution of Linear Systems by Montecarlo Methods

M.Bonamni , B.Codenotti

Nota Interna B4-31

Dicembre 1987

VLSI DESIGNS FOR THE SOLUTION OF LINEAR SYSTEMS BY MONTECARLO METHODS

M.BONANNI and B.CODENOTTI
Istituto di Elaborazione dell'Informazione
Via S.Maria, 46 - 56100-PISA (Italy).

ABSTRACT

This paper presents VLSI networks for the solution of linear systems, which are based on the application of Monte Carlo methods.

We obtain $area \times (time)^2$ performances improving the results attained by the best known VLSI linear system solvers, at the price of introducing, in the result, a "probabilistic error", which can be evaluated in terms of Chebychev's inequality.

Key Words: Area-Time Complexity, Layout, VLSI, Monte Carlo Methods, Random Numbers, Markov Chain.

1. Introduction

We here consider the problem of solving linear systems in VLSI by using Monte Carlo Methods, with the goal of speeding up the computation, at the price of introducing a probabilistic error in the result.

The VLSI solution to a computational problem consists of the conception of a layout, and of an algorithm running on it. The evaluation of the obtained design can be performed by exploring the tradeoff between the area of the layout, and the computation time of the algorithm.

A computational model for VLSI has been proposed by several authors [see for example 2,5,6, 9], with the goal of providing a base for the evualuation of given designs.

In this paper we will show several VLSI networks for linear systems solution by Monte Carlo Methods, either in the general case, or in the case of linear systems with special coefficient matrix.

More precisely, we will show an approach yielding the solution of the problem by means of architectures closely related to the mesh of trees, on which it seems quite natural to implement Monte Carlo algorithms.

It is well known that the designer of a VLSI nxn linear system solver has to face the lower bound $AT^2 = \Omega(n^4)$, which has been estabilished by Savage in [5].

In [1] an optimal systolic design based on Givens' algorithm is presented, attaining the performance $A=O(n^2)$ and $T=O(n)$.

Here we give a first look to the implementation of Monte Carlo methods, oriented towards the solution of linear systems, the

inversion of a nonsingular matrix, the computation of only one entry either of the solution of a system, or of the inverse of a matrix.

The rest of this paper is organized as follows.

In section 2, we first present some algorithms - to be used next - for the generation of random numbers (RNGs, from now); then we describe some general features of Montecarlo methods (referred to as MC methods, from now), and we finally show some algorithms for the solution of linear systems.

Section 3 is dedicated to the description of VLSI circuits implementing Monte Carlo Methods for the solution of linear systems.

In section 4, we finally show some tables concerning the area-time performance of the designs proposed in the paper, and we analyze relations with other work in the field.

The following notations will be used in the paper.

E[X] denotes the expected value of a random variable X;

V[X] denotes the variance of a random variable X;

||B|| denote the infinite norm of a matrix B.

2. Monte Carlo Methods for the Solution of Linear Systems and for Matrix Inversion.

We first recall some results concerning the automatic generation of sequences of random numbers.

We mentioned - and it will be clearer later - that such problem is central when implementing Monte Carlo algorithms, since they

consist of random sampling of quantities by means of independent trials.

We assume the set of sampling to be the interval I=[0 1], and we will show later how to generalize this assumption.

We need an impartial procedure to produce a sequence of random numbers on I. Hence the following questions arise:

which is a concrete definition of impartiality?

how to treat and explain the concept of casuality?

Hystorically, the first RNGs were based on empirical procedures, whose results were collected into tables.

Such generators had high costs in terms of storage requirements, so that it became necessary to produce sequences with the property of being not "exactly random", but easy to compute and able to pass a given set of randomness tests. Such sequences has been called pseudorandom numbers.

One of the main features of RNGs is their period, i.e. the lenght of the produced sequence without repetitions.

We present two algorithms, belonging to two different classes of methods, namely the "congruential", and the "middle square" approaches.

Congruential methods [see 8] have the following general form:

Given a, U_0, M,

compute:

$$\begin{cases} U_{k+1} = a\,U_k \bmod M \\ X_{k+1} = U_{k+1}/M \end{cases} \quad , \; k=0,1,\ldots,n-1,$$

where X_i, $i=1,2,\ldots,n$, is the pseudorandom sequence.

Middle Square methods [see 8] are constructed by means of the following relations:

$$X_{k+1} = \{ [X_k^2 M^{3/2}] / M \} ,$$

where {a} and [a] denote the fractionar and the integer part of a real number a, respectively.

We now show a congruential algorithm (RND1), and a middle square algorithm (RND2), described in Pascal-like notation.

```
RND1
(init c:=1)
BEGIN
  b:=663608941 * c (mod 2^d);
  RETURN u:=c/2^d ;
  c:=b
END.
```

The period of the sequence results to be 2^{d-2}.

```
RND2
(init u:=0.v_1 v_2 ... v_m)
BEGIN
  a:=u^2; (a=0.r_1 r_2 ... r_2m);
  RETURN u:= 0. r_m/2+1 r_m/2+2 ... r_3m/2 ;
END.
```

From now, the assignment "u:=RND" means that u is given a random number in I, by using either algorithm RND1 or RND2.

The generation of uniform random numbers on an interval [a b] can be performed by using RND1 or RND2, according to algorithm RNDG.

```
RNDR (b,a)
BEGIN
  u:=RND;
  RETURN y:=u(b-a) + a;
END.
```

We now give a description of some algorithms for the solution of linear systems (and for matrix inversion), based on the Monte Carlo approach.

It is first necessary to introduce the techniques to be used in order to simulate paths on Markov chains.

The basic idea is the following:

let us consider the transition matrix of a given Markov chain, $P=(p_{i,j})$; for any row of such matrix, the sum of the (nonnegative) entries is equal to one; the value $p_{i,j}$ corresponds to the probability of going, from the state "i" to the state "j".

Therefore, the entry $p_{i,j}$ can be associated to a segment of lenght $p_{i,j}$ in [0 1], and the transition "i to j" is allowed when a RNG returns a random number corresponding to the segment.

It is self evident that a given segment is chosen with probability increasing with its lenght.

Consider now the problem of solving linear systems, following [7].

Let $A=(a_{i,j})$ be an nxn nonsingular matrix, and let $q=(q_i)$ be an n-vector. Moreover, we denote with I the identity matrix.

Assume that

$A = I - B$, where $\| B \| < 1$.

Then the linear system

$$A x = q ,$$

can be written as:

$$x = B x + q,$$

from which is easy to derive the iterative method:

$$\begin{cases} x^{i+1} = B x^i + q , & i=1,2,... \\ x^0 = 0. \end{cases}$$

We have:

$$x^1 = q \ ,$$

$$x^2 = B\,q + q,$$

. . . . .

$$x^k = B^k q + B^{k-1} q + \ldots + B\,q + q,$$

. . . . .

The m-th entry of the solution x, can be written as:

$$x_m = \sum_{r=1,2,\ldots} \sum_{i_1\, i_2 \ldots i_r} b_{m i_1}\, b_{i_1 i_2} \ldots b_{i_{r-1} i_r} q_{i_r} \ .$$

Let us denote with x_m^R the m-th entry of the vector x^R, i.e.

$$x_m^R = \sum_{r<R} \sum_{i_1\, i_2 \ldots i_r} b_{m i_1}\, b_{i_1 i_2} \ldots b_{i_{r-1} i_r} q_{i_r} \ .$$

Following [7], we now choose

$$b_{i,j} = f_{i,j}\, p_{i,j} \ , \text{ with } 0<p_{i,j}<1, \ i,j=1,2,\ldots,n,$$

and $$q_i = g_i\, p_i \ , \quad \text{with } 0<p_i<1, \quad i=1,2,\ldots,n,$$

in order to satisfy

$$\sum_{j=1}^{n} p_{i,j} + p_i = 1 \ , \ i=1,2,\ldots,n.$$

Then x_m^R can be written as:

$$x_m^R = \sum_{r<R} \sum_{i_1 \ldots i_r} f_{m i_1} \ldots f_{i_{r-1} i_r}\, g_{i_r}\, p_{i_1 i_2} \ldots p_{i_{r-1} i_r}\, p_{i_r} \ .$$

We now construct a Markov chain with (n+1)x(n+1) transition matrix P defined as:

$$P = \begin{pmatrix} p_{11} & p_{12} & \dots & p_{1n} & p_1 \\ p_{21} & p_{22} & \dots & p_{2n} & p_2 \\ . & . & \dots & . & . \\ p_{n1} & p_{n2} & \dots & . & p_n \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} .$$

The sequence

$$m \longrightarrow i_1 \longrightarrow i_2 \longrightarrow \dots \longrightarrow i_r \longrightarrow n+1,$$

with $i_j \neq n+1$, $j=1,2,\dots,r$, is a path with initial state m.

From the Markov property, we have:

$$P(m,i_1,\dots,i_r,n+1) = p_{mi_1} p_{i_1 i_2} \dots p_{i_{r-1} i_r} p_{i_r} .$$

Let now define a random variable $Y_m^{(R)}$ as:

$$Y_m^{(R)}(m, i1, \dots, ir, n+1) = f_{mi_1} f_{i_1 i_2} \dots f_{i_{r-1} i_r} g_{i_r}, \quad r<R,$$

$$Y_m^{(R)}(m, i1, \dots, ir) = f_{mi_1} f_{i_1 i_2} \dots f_{i_{r-1} i_r} q_{i_r}, \quad r=R$$

It readily follows that

$$E[Y_m^{(R)}] = x_m^{(R)}.$$

In the following, an algorithm for the evaluation of $x_m^{(R)}$ is presented.

```
SOLVE1
  BEGIN
    <compute f_ij , p_ij , p_i , g_i>;
    <compute n paths w_1 , w_2 , ... , w_n>;
    U:=0;
    FOR h:=1 TO N DO
          BEGIN
            <update r,i_1,...,i_r>;
            V:= f(m,i_1);
            FOR j:=1 TO r-1 DO
                  V:=V*f(i_j,i_j+1);
            IF r<R THEN V:=V*g_i
                   ELSE V:=V*q_i;
            U:=U+V;
```

```
            END;
      U:=U/N
      RETURN U;
    END.
```

A simple modification of SOLVE1 leads to INV1, which can be used to compute the inverse of a given matrix, and requiring the same assumptions on A than SOLVE1.

Another approach has been presented in [7], where ergodic Markov chains are used to compute random paths. The corresponding algorithms algorithms for linear systems solution will be here called SOLVE2 and SOLVE3, while the algorithms for matrix inversion will be called INV2 and INV3. SOLVE2 and SOLVE3 (as well as INV2 and INV3) differ for the choice of the starting point of each path, which is made "a priori" for SOLVE2 (INV2), and "random" for SOLVE3 (INV3). Details of these algorithms will be given in section 3, when necessary.

It is worth pointing out that all the above described methods require a special matrix structure. In the following, we recall the steps of another algorithm [see 7], which can be applied to arbitrary nonsingular matrices.

In the algorithm, we will refer to a function V(x) which is defined as:

$$V(x) = \sum_{iJ} (a_{i,j}x_j - q_i)^2.$$

```
RISGEN1 (computation of x_m)
  BEGIN
    M:=0;
    FOR i:=1 TO N DO
        BEGIN
          <construct, at random, an n-vector Y(i)>;
          IF V(Y(i))<=c THEN
```

```
                BEGIN
                  M:=M+1;
                  <hold the vector Y(i)>
                END
        END;
    <sort the m-th entries of the vectors which have been held
     and put them into the vector POS>;
    RETURN U:=POS[M/2]
END.
```

A variant of RISGEN1 is shown in [7], where all the computed n-vectors are used in order to evaluate the solution. The algorithm corresponding to this approach is called RISGEN2.

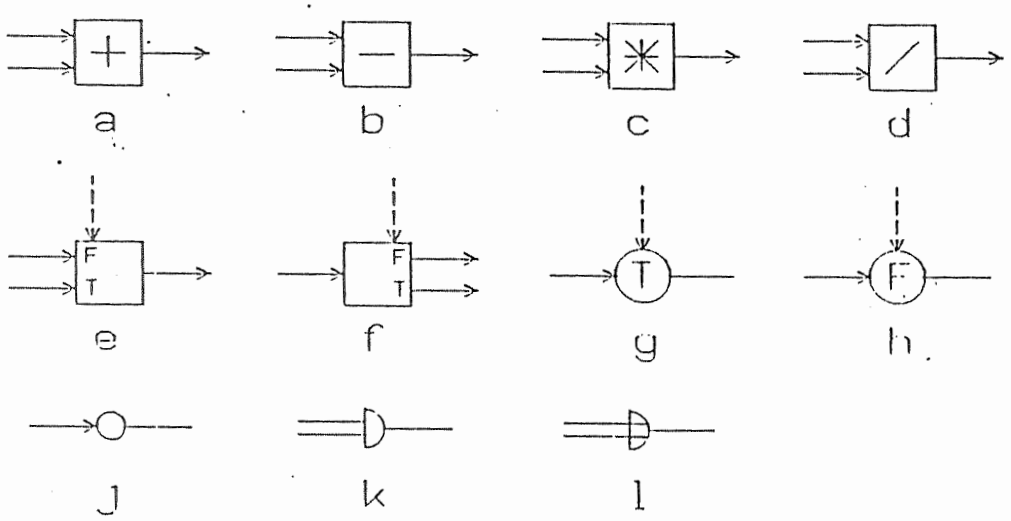
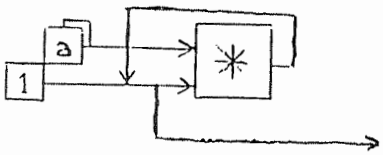


Fig.1



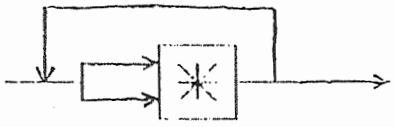


Fig.2

3. Monte Carlo Methods for the Solution of Linear Systems and for Matrix Inversion.

In this section, some VLSI architectures are presented, which are derived from the algorithms presented in section 2.

In fig.1 we describe some basic modules to be used here and later. (In the figures, here and in the following, dotted lines denote wires carrying one bit of information; continuous lines denote wires carrying d bits of information, where d is the number of digits of the arithmetic).

Some of them perform trivial arithmetic or logical computations. A discussion is worth for what concerns modules SWITCH, MERGE, and gates TRUE and FALSE.

Module SWITCH send its input only to one output wire, depending on a logical value (dotted line) in input.

Module MERGE send to its output wire one of its two input, depending on a logical value in input.

TRUE (FALSE) gates send their input to the output, if the logical input is 1 (0).

For what concerns the logical modules OR, and NOT, we will use for simplicity the same notation in the case of wires carrying one bit, and d bits.

We now present circuits implementing uniform RNGs.

Fig.2 shows the trivial implementation of RND1, RND2, and RNDG, whose area-time cost id dominated by the multiplier.

Therefore the resulting performance is [see 2]:

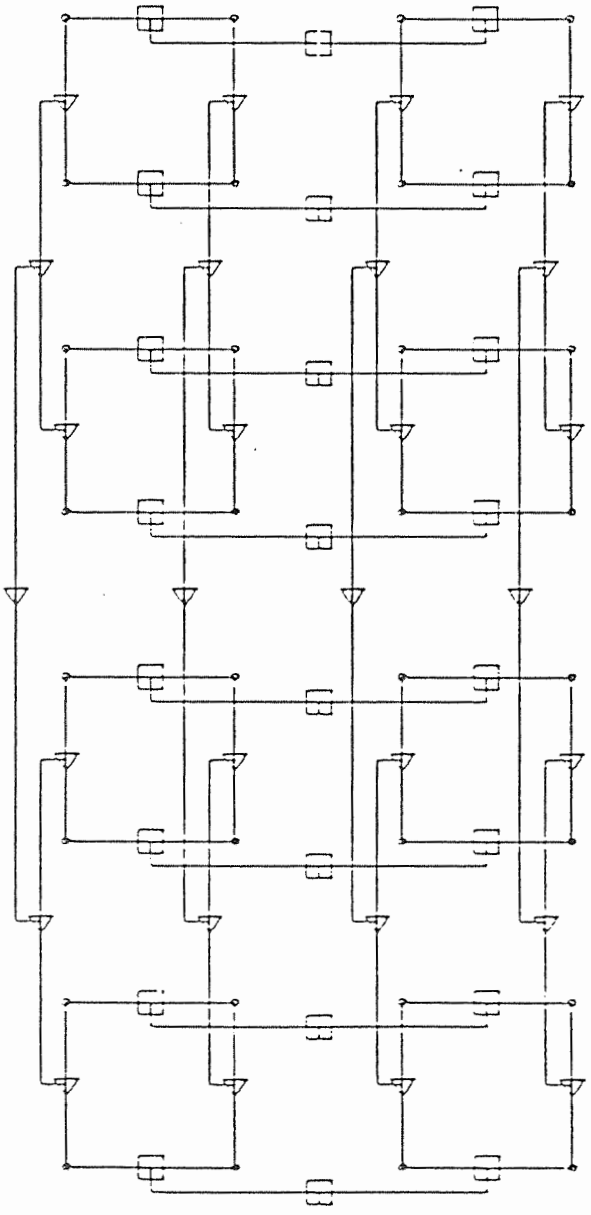
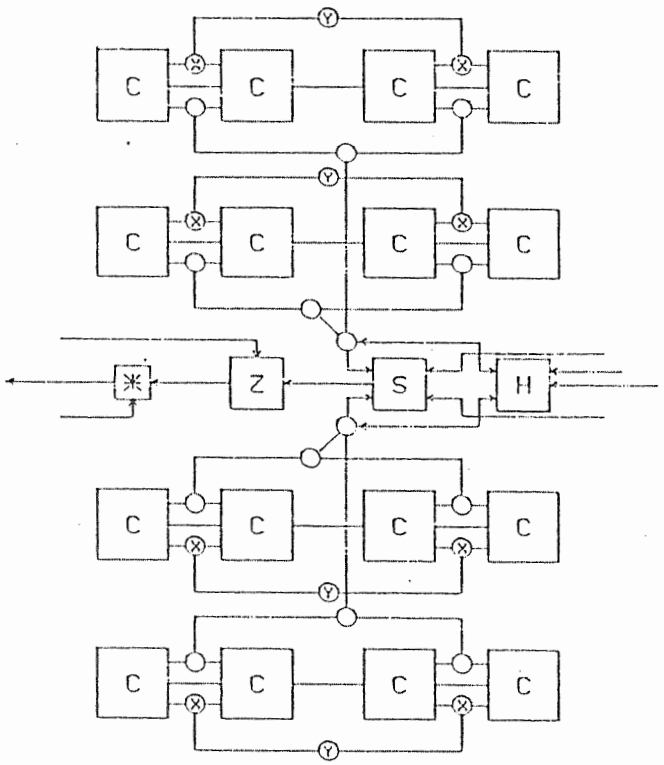
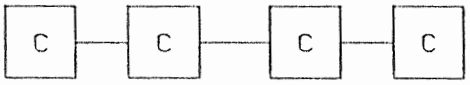
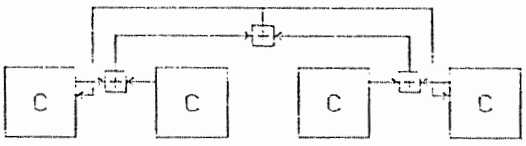


Fig.3







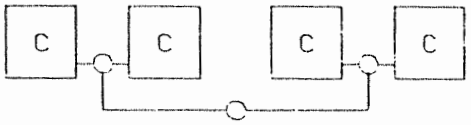
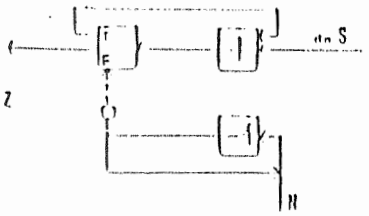
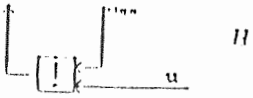
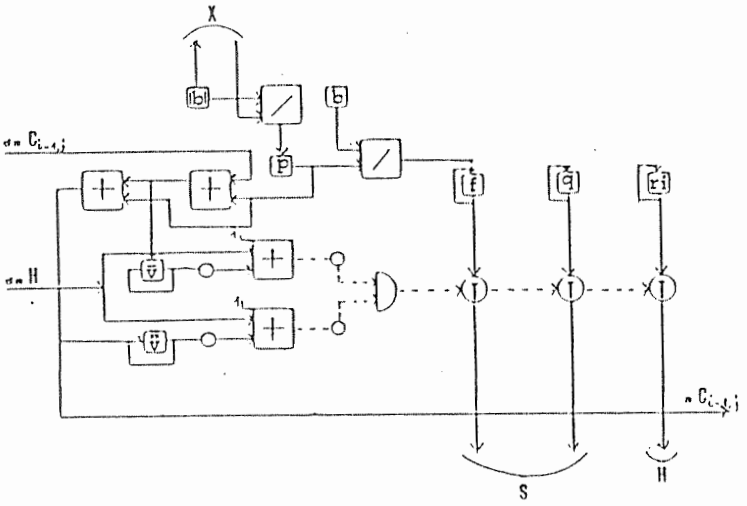


Fig.4







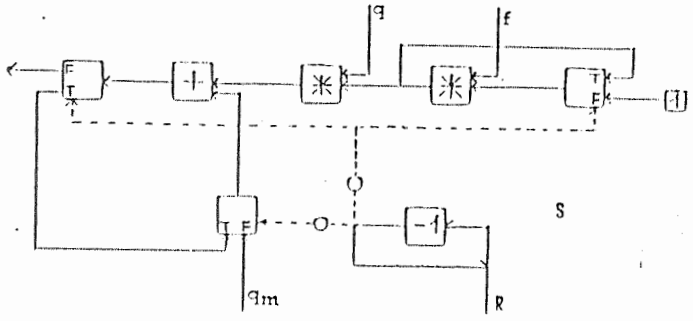


Fig.5

$AT^2=O(d^2)$, for any $T\in[\log d, \sqrt{d}]$.

Circuits corresponding to one of those illustrated in fig.2 will be denoted by RND in the following.

We present some VLSI architectures, which are obtained by minor modification of mesh-of-trees, on which it is easy to run the Monte Carlo algorithms for the solution of linear systems presented in section 2. Some of the results lead to an improved area-time performance, with respect to solutions derived from deterministic algorithms, at the price of introducing a probabilistic error in the result. An analysis of the results shown in this section, is given in the next section.

The basic architecture to be adopted is described by fig.3.

Such architecture has area $A=O(n^2\log^2 n)$, where n^2 is the number of nodes.

The VLSI architecture proposed to implement SOLVE2 (see section 2) is described in fig.4a, and it consists of a minor modification of a mesh-of-trees. Modules denoted by C send to module S the values f and q, and to module H the information concerning the row of the matrix, needed in order to simulate a path on a Markov chain, according to the arguments of section 3.

Module S send its output to module Z, which sums N quantities received, before sending the result to a multiplier, which computes x_m.

Fig.4b shows details of the communication network among modules of type C.

Fig.5 illustrates the structure of modules Z, H, C, S.

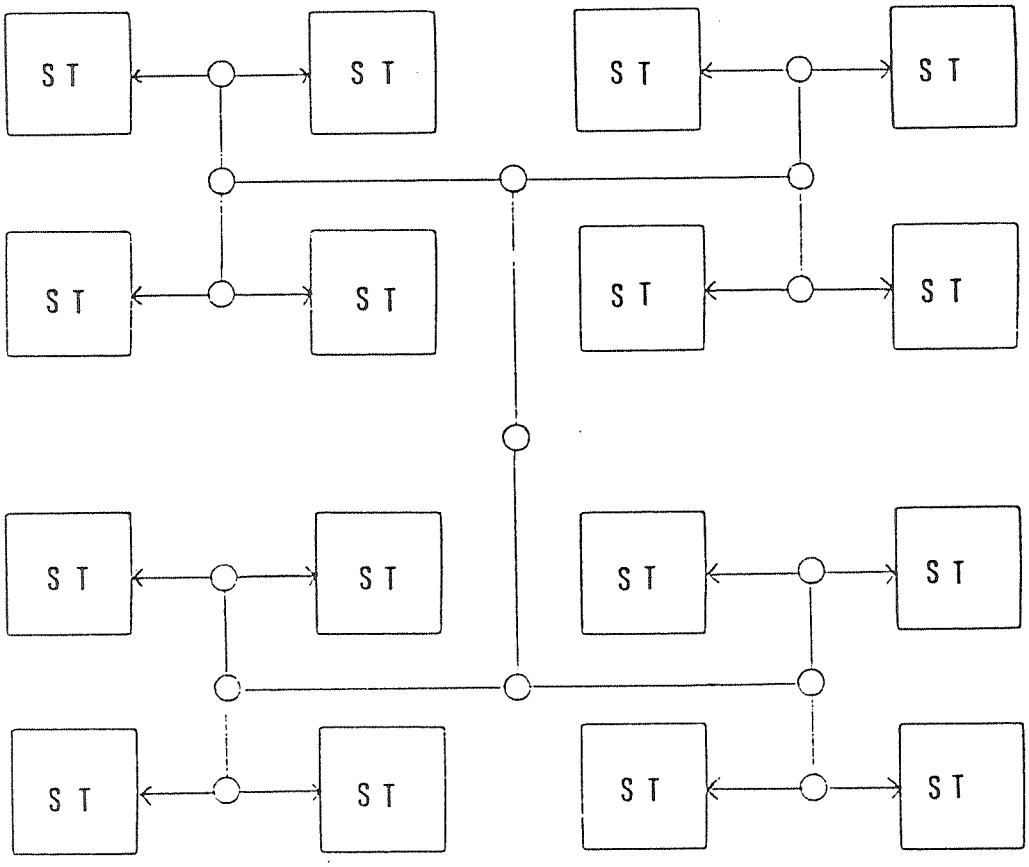
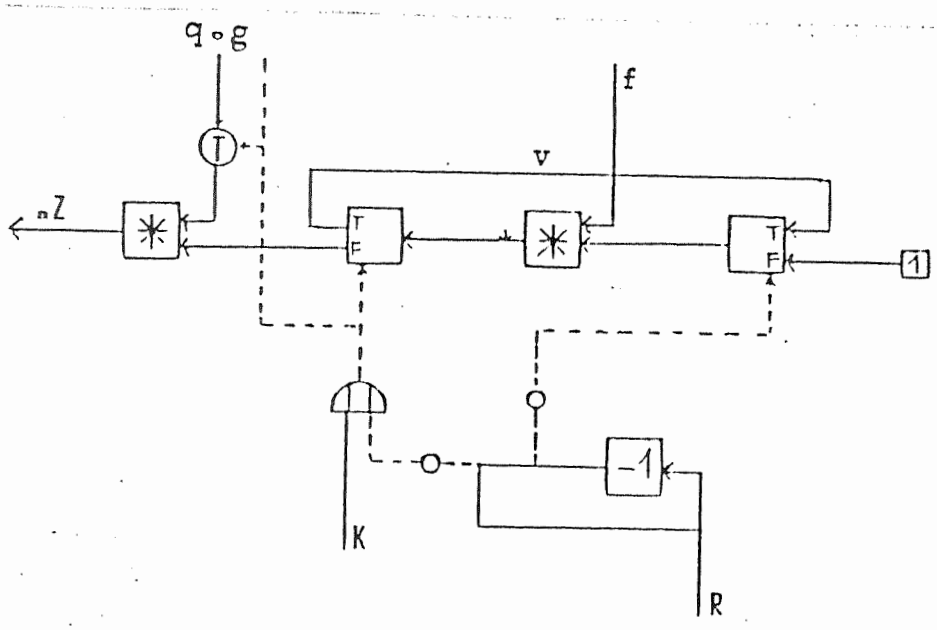


Fig.6



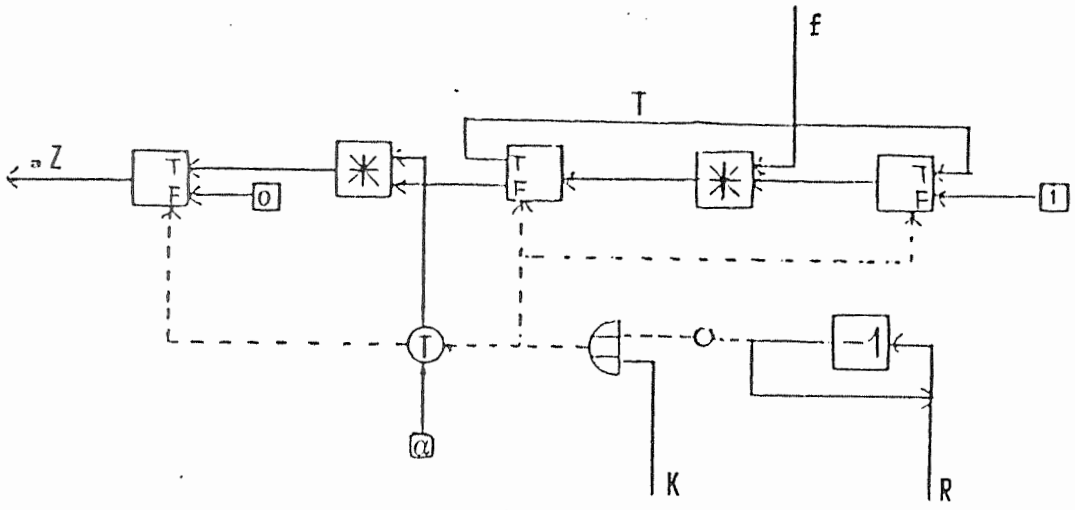


Fig.7

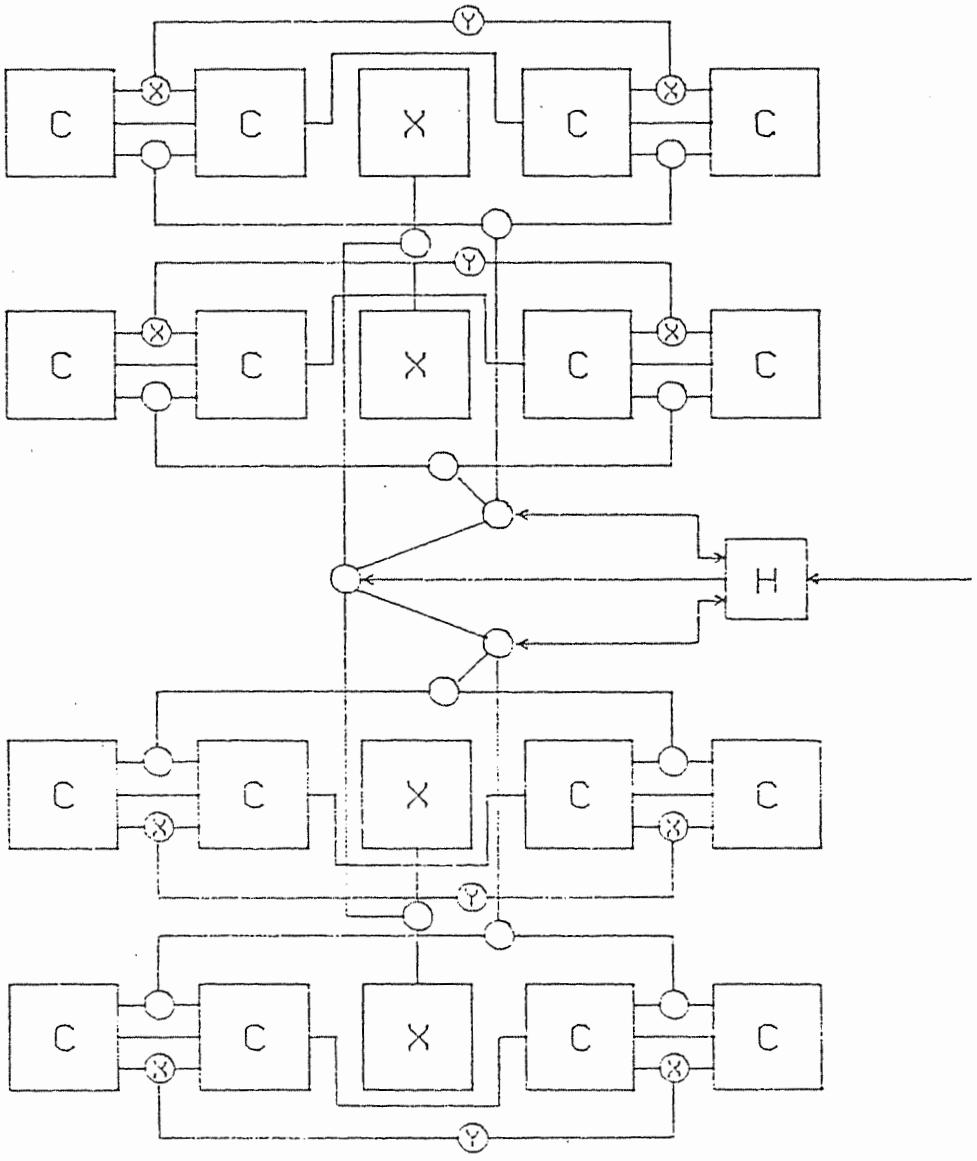
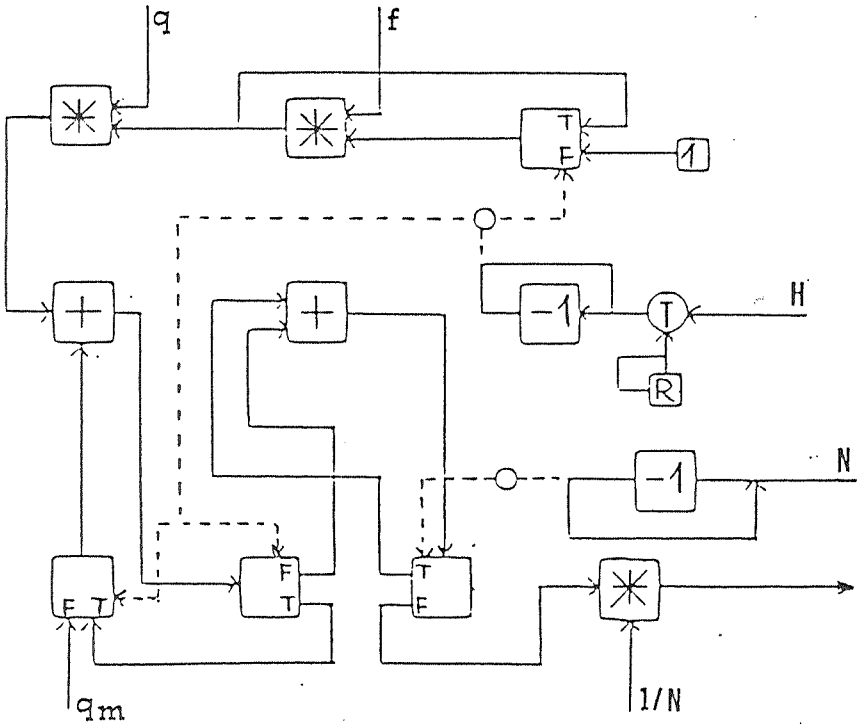


Fig.8



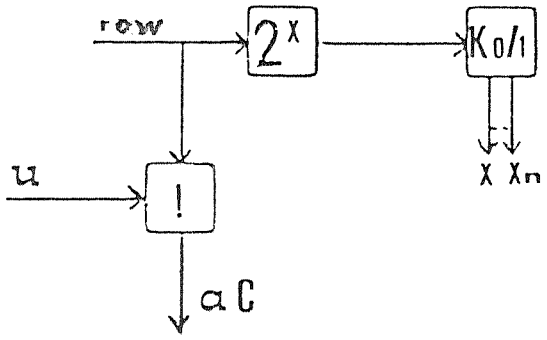


Fig.9

The network shown in fig.4 will be called P1 mesh-of-trees. It has area $A=O(n^2 \log n)$, as one can readily prove.

For what concerns the computation time, we have:

$$T=O(NR\log n),$$

where N is number of paths on the Markov chain, and R is the maximum lenght of the chains. In the next section, we will summarize the area-time bounds obtained as functions of n. For this purpose, we will bound N and R in terms of n.

The computation of all the entries of the solution x can be performed by using a binary tree (see fig.6), in which each one of the n modules (ST in the figure) computes one entry.

The modifications to the structure of figg.4 and 5 are illustrated in fig.7.

The performance of this solution depends on the chosen I/O conventions,as well as on the VLSI model of computation adopted. In the following section, we will show that the multilective model [see 6 for a rigorous definition] allows improving the performance obtained correspondently to the semelective model [see for example 6,8].

In fig.8, another VLSI scheme for the implementation either of SOLVE3, or of INV2, INV3 (up to minor modifications) is presented.

Such scheme is called P2 mesh-of trees, and has area $A=O(n^2 \log n)$.

Modules denoted by X (see fig.9a for their description) perform the computations of modules S, Z, and ST (see figg.5 and 6), in order to evaluate simultaneously several entries of the solution. Module H has to be modified, according to fig.9b.

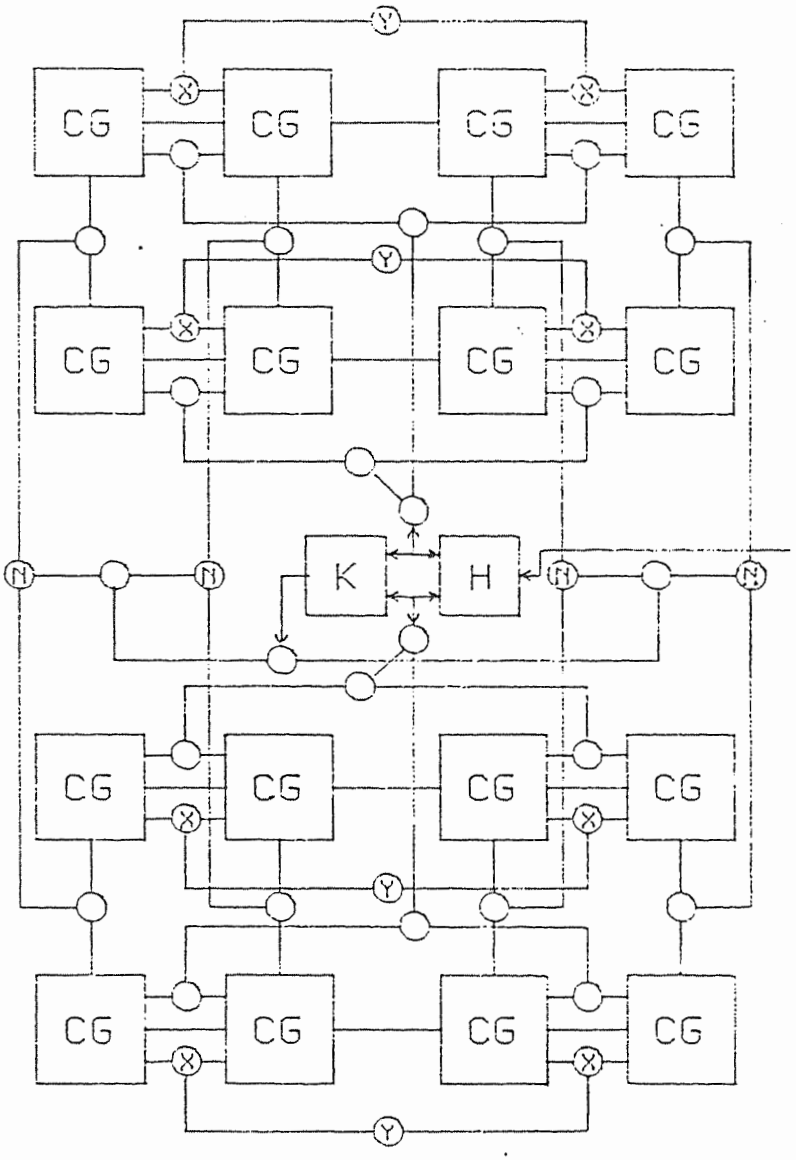


Fig.10

A mesh-of-tree can be used in order to implement INV3, as shown in fig.10. Modules CG substitute both C and G, and they will hold the result at the end of the computation.

Details concerning the area-time performance of all these schemes will be given in section 4.

For what concerns the implementation of algorithm RISGEN1 (section 2), we use the network described in fig.11, which essentially consists of a structure performing matrix-vector product, which is the main step of the algorithm.

In fig.12, some details of the design are illustrated.

4. Conclusions.

Some networks for the VLSI solution of linear systems have been presented, either in the semelective or in the multilective model.

In table I such results are shown, as a function of the size of the problem, and of quantities (N,S) typical of Monte Carlo methods. In some cases, such quantities can be upper bounded as functions of the size of the problem, by means of extimates of the variance of the opportune random variable. Table II contains the results, given as functions only of the size of the problem.

It is now worth noting that the latter table presents "pessimistic" complexity bounds, since the evaluation of the variance often results not to be sharp.

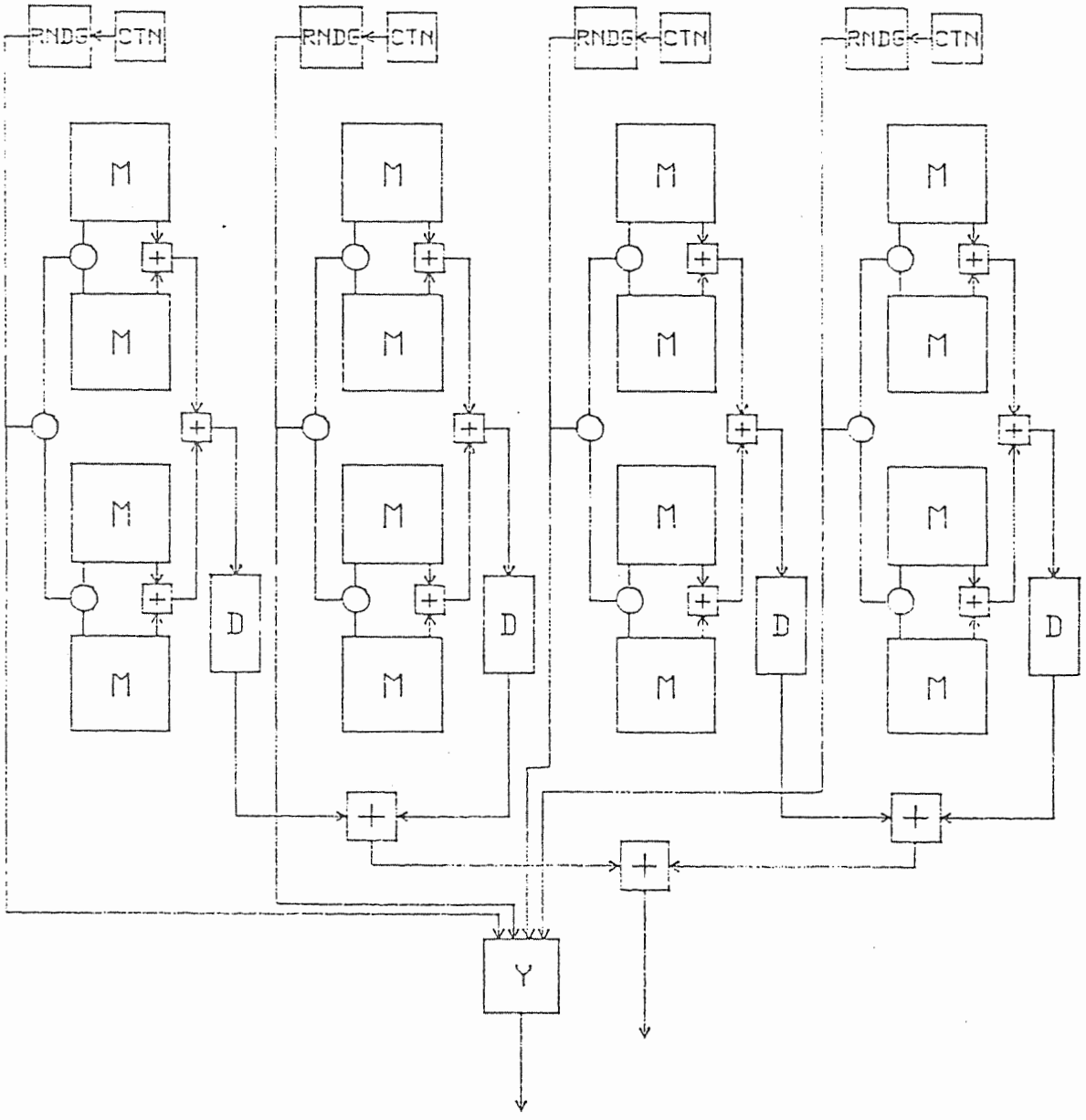


Fig.11

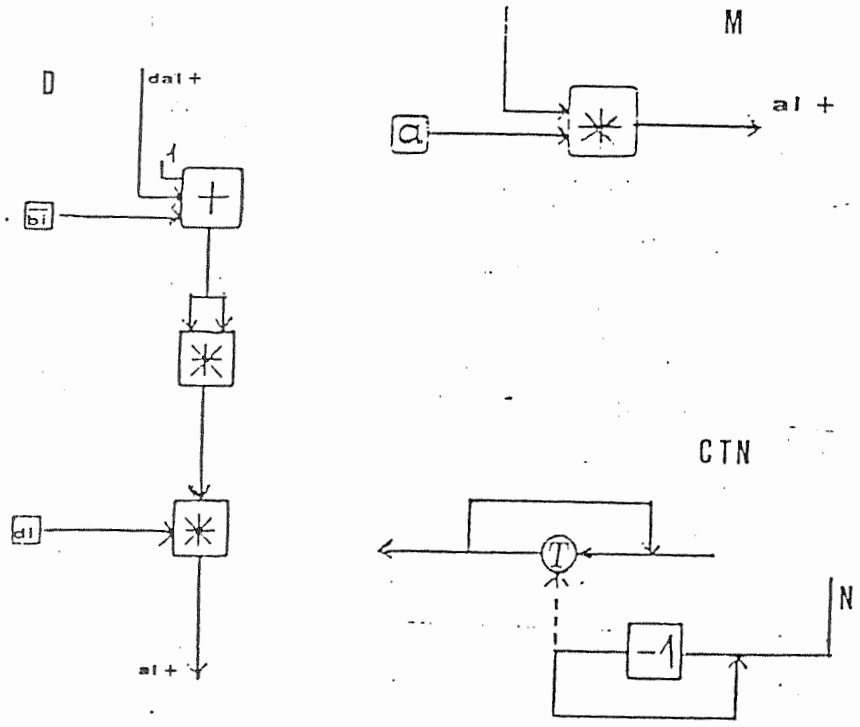


Fig.12

Finally, table III presents a comparison between the best results due to the Monte Carlo methods presented in this paper, and known results derived from deterministic algorithms.

When analyzing the tables, it is worth taking into account, that the behaviour of Monte Carlo solvers has been studied (and can be studied only) in terms of the Chebychev's inequality, i.e. up to an arbitrarily small probabilistic error.

References

[1] Bojanczik, A.,Brent, R.P., and Kung,H.T., Numerically Stable Solution of Dense Systems of Linear Equations using Mesh-Connected Processors, SIAM J. Scie, Stat. Comput. 5, 95-104 (1984).

[2] Brent, R.P., and Kung,H.T., The Area-Time Complexity of Binary Multiplication, J. Ass. Comput.Mach. Vol.28, 521-534 (1981).

[3] Codenotti, B., Lotti, G., and Romani, F., VLSI Implementation of Iterative Methods for the Solution of Linear Systems, Integration 3, 211-221 (1985).

[4] Halton, J.H., A Restrospective and Prospective Survey of the Monte Carlo Method , SIAM Rev. 12, 1-63 (1970).

[5] Savage, J.E., Area-Time Tradeoffs for Matrix Multiplication and Related Problems in VLSI, J. Comput. and Sist. Science 22, 230-242 (1981).

[6] Savage, J.E., The Performance of Multilective VLSI Algorithms, J. Comput. and Sist. Science 29, 243-273 (1984).

[7] Shreider, Y.A., The Monte Carlo Method, Pergamon Press (1966).

[8] Thompson, C.D., Area-Time Complexity for VLSI, Proc. 11th ACM Symp. Theory of Comput., 81-88 (1979).

[9] Vuillemin, J, A Combinatorial Limit to the Computing Power of VLSI Circuits, Proc 21th IEEE Symp. Found.of Comput. Sci.,294-300 (1980).

Table I

circuit	AT^2	function
SOLVE1	$Nn^2\log^3 n$	x_m
SOLVE1	$N^2n^3\log^3 n$	x
SOLVE2	$Nn^2\log^3 n$	x_m
SOLVE2	$N\,n^3\log^3 n$	x
INV1	$n^2\log^3 n$	$A^{-1}(i,j)$
INV2	$n^3\log^3 n$	A^{-1}

Table II

circuit	AT^2	function
SOLVE1	$n^2 \log^3 n$	x_m
SOLVE1	$n^4 \log^3 n$	x
SOLVE2	$n^2 \log^3 n$	x_m
SOLVE2	$n^3 \log^3 n$	x
INV1	$n^2 \log^3 n$	$A^{-1}(i,j)$
INV2	$n^3 \log^3 n$	A^{-1}

Table III

circuit	AT^2	function
GIVENS	n^4	x
GAUSS-JORDAN	n^4	A^{-1}
SOLVE2	$n^2 \log^3 n$	x_m
SOLVE2	$n^3 \log^3 n$	x
INV1	$n^2 \log^3 n$	$A^{-1}(i,j)$
INV2	$n^3 \log^3 n$	A^{-1}