Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

AREA-TIME COMPLEXITY OF MATRIX-VECTOR MULTIPLICATION

B. Codenotti, F. Romani, G. Lotti

Nota interna B86-02
Gennaio 1986

AREA-TIME COMPLEXITY OF MATRIX-VECTOR MULTIPLICATION.

B. CODENOTTI and F. ROMANI
Istituto di Elaborazione dell'Informazione - CNR,
Via S.Maria 46, PISA, ITALY.

G.LOTTI
Dipartimento di Informatica, Universita' di Pisa,
Corso Italia 40, PISA, ITALY.

ABSTRACT.

It is studied the matrix-vector multiplication problem in VLSI. Both lower and upper bounds are derived which depend on the chosen model, obtained by different I/O conventions.

The problem of multiplying a sparse matrix by a vector is taken into consideration as well.

The results obtained for matrix-vector product are applied to study the VLSI complexity of other numerical problems.

KEY WORDS. VLSI Models, Area-Time Complexity, Sparse Matrices, Matrix-Vector Product, Iterative Methods.

1. INTRODUCTION

The area-time complexity of some problems can be deeply influenced by the I/O conventions, as shown by Ja' Ja' [6] for graph problems. In this paper we consider matrix-vector multiplication, deriving both lower and upper bounds depending on the chosen I/O conventions. We also will take into consideration the problem of multiplying a sparse matrix by a vector.

The computational complexity of matrix-vector multiplication in VLSI models induces complexity results for other important numerical problems such as eigenvalues computations, operations concerning polynomials, solution of linear systems by iterative methods. In the following, we will deserve a special attention to the last problem.

We turn now to the VLSI models of computation here considered. Following Ja' Ja', we can assume, without loss of generality, the chip to be a rectangle, whose area A is determined by gates connected by wires. Wires have minimal width $1>0$, and each gate has an area (1^2). A constant number of wires can intersect at any point. Moreover the chip is where- and when-oblivious, i.e. the position and the time for which data are available are independent of the values of the input bits. The propagation of a bit along a wire takes a unit time, independent of the wire length.

Several possibilities are considered for the I/O conventions:

G) I/O ports lie anywhere on the chip.

B) all I/O ports lie on the boundary of the chip.

P) pipelining is allowed for I/O ports.

N) no pipelining is allowed for I/O ports.

These alternatives allow defining four different models (denoted as GP, GN, BP, BN), which will be used in the following.

In numerical computations data are grouped into words of fixed length d. The value d is often independent of the size of the problem; in this case the asymptotic complexity of the problem is the same both in word-based models and in bit-based models. Sometimes, error analysis considerations suggest to use an word length depending on the size of the problem. This leads to different asymptotic complexity results in the two types of models. We will state our results by assuming the former situation to hold.

The following quantities will be used in deriving lower bounds:

ϕ_i, ϕ_o the input and output information flow, respectively,

C_s the sequential complexity of the problem,

I the number of input data-words,

R the number of internal data-words (constants of the problem).

For a specific problem area-time lower bounds depend on the used model.

In the model GP the following lower bounds hold:

$$AT^2 = \Omega(\phi_i^2), \quad AT^2 = \Omega(\phi_o^2), \quad AT > C_s, \quad A > R.$$

In the model BP all the bounds for GP hold, together with

$$AT^2 = \Omega(\phi_i I).$$

In the model GN all the bounds for GP hold, together with

$$A > I$$

In the model BN all the bounds for GP, GN, BP hold, together with

$$AT = \Omega(\phi_i I).$$

In the next section we will discuss the problem of multiplying a matrix by a vector. In section 3 we will discuss dense matrix by vector multiplication; in section 4 the "sparse" case will be considered. In section 5 the results obtained for matrix-vector product are applied to study the VLSI complexity of other numerical problems. We will give a special attention to the solution of linear systems by iterative methods.

2. THE MATRIX VECTOR MULTIPLICATION PROBLEM

Let $A = (a_{ij})$ be an nxn real matrix and let $\underline{x} = (x_j)$ be an n-vector. The product $A\underline{x}$ is the n-vector $\underline{y}$ whose components are

$$y_i = \sum_{j=1}^{n} a_{ij} x_j .$$

When solving this problem in VLSI models there are two main

possibilities, namely:

a) the matrix A is resident into the circuit, i.e. its entries are either constant and wired on the circuit or inserted in a preprocessing phase. The input data are the entries of $\underline{x}$.

b) The input data are the entries both of A and $\underline{x}$.

The first case occurs when the matrix A is related to a given transform (e.g. Discrete Fourier Transform) or the same matrix has to be multiplied by several different vectors as in iterative methods for the solution of linear systems.

In this section we present a general technique to obtain good upper bounds, namely we show two ways of aggregating basic matrix-vector multipliers, and we discuss the attained performances. These results will be applied either to "dense" or to "sparse" matrix-vector multiplication, in the following sections.

The technique used in the following consists in partitioning the matrix into $(n/k)\times(n/k)$ blocks of size k and performing the nxn matrix-vector product by using simpler modules for the multiplications of order k.

1) LINEAR ARRAY

Let us consider a linear array of n/k modules, each capable of performing kxk matrix-vector multiplication (see Fig. 1). The basic module has area a(k), time t(k), and period $p(k) \leq t(k)$, i.e. a matrix-vector product computation can start every p(k) units of time. Vector $\underline{x}$ enters via b input paths on

FIG. 1

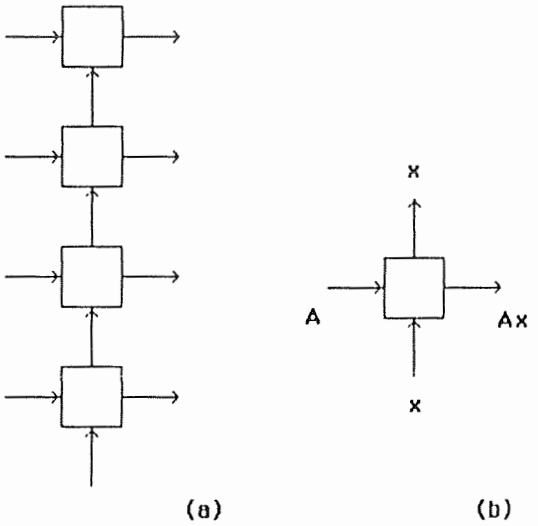


Fig.1. (a) linear array of modules; (b) single module.

the boundary and it is transmitted to neighbouring modules. Analogously vector $\underline{y}$ exits via b output paths on the boundary. The matrix enters via c input paths. The perfomance of this design is

$$A(n) = a(k)\ n/k;$$

$$T(n) = \max(p(k),\ k^2/c)\ n/k + t(k).$$

Note that when the linear array is used in a modular structure, the output paths can be required to be as many as the input paths. In this case the resulting vector y can exit from the b top output paths in time $O(n)$.

2) RECURSIVE AGGREGATION

Another type of aggregation is the recursive design of Fig. 2, which can be derived by a mesh of trees.

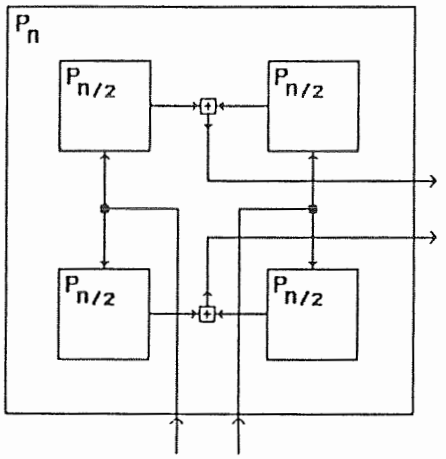
Assume n to be a power of two. Module P_n performs nxn matrix vector multiplication using four $P_{n/2}$ modules. The P_k module is capable of performing kxk matrix-vector multiplication in time $t(k)$. The latter module has heigth $h(k)$ and length $l(k)$, i.e. area $a(k) = l(k)\ h(k)$; vector $\underline{x}$ enters via b $(b \le l(k),\ b \le h(k))$ input paths on the boundary. Analogously vector $\underline{y}$ exits via b output paths on the boundary.

The following recurrence relations can be estabilished for the heigth $H(n)$ and the length $L(n)$ of the resulting network

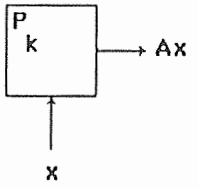
$$H(n) = 2\ H(n/2) + b\ n/k;$$

$$H(k) = h(k);$$

FIG. 2



(a)



(b)

Fig.2. (a) recursive aggregation of modules; (b) single module.

$L(n) = 2\,L(n/2) + b\,n/k;$

$L(k) = l(k).$

Whence

$H(n) = n/k\ h(k) + b\ n/k \log(n/k);$

$L(n) = n/k\ l(k) + b\ n/k \log(n/k);$

$A(n) = O(n^2/k^2\ a(k) + b^2\ n^2/k^2 \log^2(n/k)).$

Moreover

$T(n) = \log(n/k) + t(k).$

3. THE DENSE CASE

In this section we present some results concerning the area-time complexity of dense matrix-vector multiplication, in the models considered in section 1.

For dense matrix-vector multiplication the following quantities can be used in deriving lower bounds:

$\phi_i = \Omega(n), \qquad C_s = \Omega(n^2).$

Moreover

$I = n, \quad R = n^2,$ if the matrix is resident,

$I = n^2, \quad R = 0,$ otherwise.

Then the following lower bounds can be stated:

a) resident matrix case.

In all models we have $A = \Omega(n^2)$.

b) non-resident matrix case.

In model GP, $AT = \Omega(n^2)$.

In model BP, $AT^2 = \Omega(n^3)$.

In model GN, $A = \Omega(n^2)$.

In model BN, $AT = \Omega(n^3)$.

Consider now the upper bounds which can be attained with specific networks. The performances which meet the lower bound are marked with "OPTIMAL", the performances which meet the lower bound up to logarithmic factors are marked with "O.T.L.F.".

1) BASIC MODULE

The first design is the basic module of fig. 3. It consists of an aritnmetic processor and a finite addressable memory. The module performs the straigthforward algorithm with the following performances.

a) resident matrix case.

In all models $A(n) = O(n^2), \quad T(n) = O(n^2)$.

b) non-resident matrix case.

In model GP,BP $A(n) = O(n), \quad T(n) = O(n^2)$.

In model GN $A(n) = O(n^2), \quad T(n) = O(n^2)$.

In model BN $A(1) = O(1), \quad T(1) = O(1),$

since only the case n=1 is significative.

2) BASIC MODULE + RECURSIVE AGGREGATION

The latter design is clearly not optimal, but it can be used to perform kxk matrix-vector multiplications in a mesh of tree, used to perform nxn matrix-vector multiplication.

The area and the time of the above described network are:

$A = O(n^2\ A(k)/k^2 + n^2/k^2 \log^2(n/k)),$

and

FIG. 3

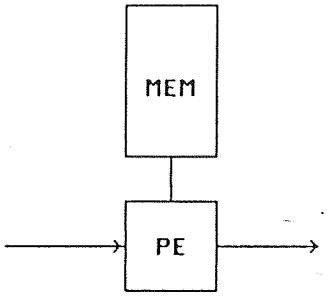


Fig.3. Basic module.

$T = O(\log(n/k) + k^2)$.

Let us study the performance of this design in the models considered in this paper.

a) resident matrix case.

Models GP, BP.

Let $k \in (\Omega(\sqrt{\log n}), O(\log n))$. Then

$A = O(n^2/T \log^2 n/T)$,

$T \in (\Omega(\log n), O(\log^2 n))$ (O.T.L.F).

For $T = \Omega(\log^2 n)$ no saving in the area can be obtained by slowing down the computation.

Models GN, BN.

Only the choice k=1 is significative. Hence

$A = O(n^2 \log^2 n)$, $T = O(\log n)$ (O.T.L.F).

b) non-resident matrix case.

In GP,BP models better bounds are produced by other designs, to be described later.

Model GN.

Only the choice k=1 is significative. Hence

$A = O(n^2 \log^2 n)$, $T = O(\log n)$ (O.T.L.F).

Here the memory cells of the resident case are substituted with input ports.

Model BN.

Only the choice k=1 is significative. The circuit has to be stretched in order to insert n^2 wires connecting the boundary to the processors, (see [4,5]), then

$A = O(n^3 \log^2 n)$, $T = O(\log n)$ (O.T.L.F).

3) LINEAR ARRAY OF ELEMENTARY PROCESSORS

The linear array can be built with elementary processors for the scalar multiplication. In this case c=b=1, and the performance is:

a) resident matrix case: NOT APPLICABLE.

b) non-resident matrix case.

In models GP,BP, we have A = O(n); T = O(n) (OPTIMAL). This design is the classical systolic array [3].

4) LINEAR ARRAY + RECURSIVE AGGREGATION

The linear array can be used to perform the kxk matrix-vector multiplications in the mesh of tree.

We obtain:

a) resident matrix case: NOT APPLICABLE.

b) non-resident matrix case.

In model GP,

the area and the time for the nxn matrix-vector multiplication are respectively

$A = O(n^2/k + n^2/k^2 \log^2(n/k))$,

$T = \log(n/k) + k$.

Let $k=\Omega(\log n)$. Then $T=k$ and

$A = O(n^2/T^2 \log^2 n/T)$ for $T\in(\Omega(\log n), O(\log^2 n))$, (O.T.L.F).

$A = O(n^2/T)$ for $T\in(\Omega(\log^2 n), O(n))$, (OPTIMAL).

In the case of boundary chips the circuit is stretched in order to insert the wires which transfer the matrix from the

boundary to the elementary processors. For the nxn problem n^2/k paths have to be inserted. Therefore the heigth of the circuit is:

$H(n) = 2H(n/2) + O(n^2/k)$, $H(k) = k$, whence

$H(n) = O(n + n \log n/k)$, and from

$L(n) = O(n/k + n/k \log n/k)$ we obtain

$A(n) = O(n^2/k + n^3/k^2 \log^2(n/k))$. Then the performance in model BP is:

$A = O(n^3/T^2 \log^2(n/T))$ for $T\in(\Omega(\log n), O(n))$, (O.T.L.F).

The results produced for the "dense" case are summarized in table I and plotted in logarithmic scale in fig.4.

4. THE SPARSE CASE

In this section we present some results concerning the area-time complexity of matrix-vector multiplication, when the matrix is sparse.

Large linear systems with sparse coefficient matrix arise frequently in solving many problems of applied mathematics. Sometimes the coefficient matrix has a strong structure which can be fully exploited in devising solution algorithms. On the other hand, if A is a general sparse matrix with weak structure properties, the algorithms suited for dense matrices are very inefficient and "ad hoc" algorithms cannot be derived. Note that iterative methods for the solution of linear systems have to perform, as main steps, matrix-vector products. The main

advantage of using iterative methods with a sparse iteration matrix is that the storage can be bounded by the nonzero elements of the iteration matrix. Note that area is the critical resource when dealing with sparse matrices, since their size usually is very large. Our results will lead to a number of nearly optimal VLSI designs for the solution of large sparse linear systems by iterative methods.

Given an nxn matrix A, let M(n) be the number of its nonzero entries. Let us partition A into $(n/k)x(n/k)$ blocks of size k. We denote with M(k) the maximum number of nonzero entries of the kxk blocks. We define three types of sparsity:

simple-sparsity if $M(n) = O(n)$,

weak-sparsity if $M(n) = O(n)$ and there exists a constant c not depending on n such that $M(k) \leq ck$, $1<k<n$,

strong-sparsity if $M(n) = O(n)$ and there exists a constant c not depending on n such that $M(k) \leq c$, $1<k<\sqrt{n}$.

An example of weak sparsity are band matrices, e.g. those arising from the discretization of partial differential equations. Examples of strong sparsity are some permutation matrices and some matrices arising from network problems. To exploit sparsity we will use the technique of partitioning the matrix into submatrices, and only pipelined models (GP and BP) will be considered in the following.

For sparse matrix-vector multiplication the following quantities can be used in deriving lower bounds:

$$\phi_i = \Omega(n), \qquad C_s = \Omega(n).$$

Moreover

$I = n$, $R = M(n)$, if the matrix is resident,

$I = n$, $R = 0$, otherwise.

Then the following lower bounds can be stated:

a) resident matrix case,

models GP, BP, $AT^2 = \Omega(n^2)$, $A = \Omega(n)$,

b) non-resident matrix case,

model GP, BP $AT^2 = \Omega(n^2)$.

Now we present the results on the front of upper bounds.

1) BASIC MODULE

The first design is the basic module of fig. 3. In the memory the $M(n)=O(n)$ nonzero elements of A are stored together with their indexes. The area sufficient for the computation is $A=O(M(n))$. The module performs the multiplication in two phases:

1) compute the products $a_{ij} x_j$, $a_{ij} \neq 0$, $j=1,2,\ldots n$;

2) compute the sums $\sum_{\substack{j=1 \\ a_{ij} \neq 0}}^{n} a_{ij} x_j$, $i=1,2,\ldots n$.

Each phase can be completed in time $T=O(n+M(n))$.

This module can be used for all the types of sparse matrices, producing

$$A(n) = O(n), \qquad T(n) = O(n),$$

in both GP and BP models and with resident or non-resident matrix.

TABLE I

DENSE MATRICES

MODEL	LOWER BOUND	UPPER BOUND	DESIGN	OPTIMALITY
a) Resident matrix				
GP, BP	$A = \Omega(n^2)$	$A = O(n^2/T \log^2 n/T)$, $T \in (\Omega(\log n), O(\log^2 n))$	(2)	O.T.L.F.
GN, BN	$A = \Omega(n^2)$	$A = O(n^2 \log^2 n)$, $T = O(\log n)$	(2)	O.T.L.F.
b) Non-resident matrix				
		$A = O(n)$, $T = O(n)$	(3)	OPTIMAL
GP	$AT = \Omega(n^2)$	$A = O(n^2/T^2 \log^2 n/T)$, $T \in (\Omega(\log n), o(\log^2 n))$	(4)	O.T.L.F.
		$A = O(n^2/T)$, $T \in (\Omega(\log^2 n), O(n))$	(4)	OPTIMAL
		$A = O(n)$, $T = O(n)$	(3)	OPTIMAL
BP	$AT^2 = \Omega(n^3)$	$A = O(n^3/T^2 \log^2 n/T)$, $T \in (\Omega(\log n), o(n))$	(4)	O.T.L.F.
GN	$A = \Omega(n^2)$	$A = O(n^2 \log^2 n)$, $T = O(\log n)$	(2)	O.T.L.F.
BN	$AT = \Omega(n^3)$	$A = O(n^3 \log^2 n)$, $T = O(\log n)$	(2)	O.T.L.F.

FIG. 4

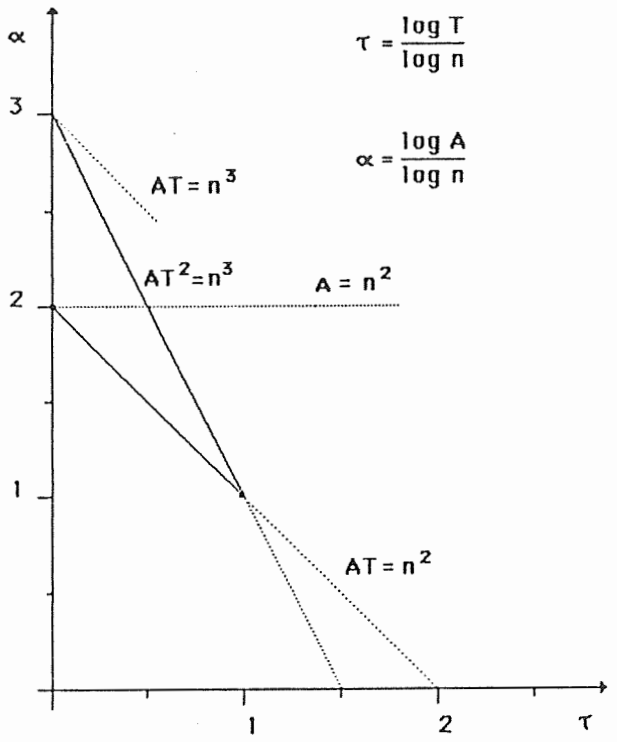


Fig.4. Upper and lower bounds for the case of dense matrices plotted in log-log scale. The dotted lines denote lower bounds. The evidenced points and the solid lines denote upper bounds. Logarithmic factors disappear in this presentation.

Now we distinguish the cases of weak and strong sparsity.

COMPLEXITY RESULTS FOR WEAKLY-SPARSE MATRICES

2) BASIC MODULE + RECURSIVE AGGREGATION

The basic module can be used to perform the kxk matrix-vector multiplications in the mesh of trees. The total area is:

$$A = O(n^2/k + n^2/k^2 \log^2(n/k)),$$

and the time is

$$T = O(\log(n/k) + k).$$

Then

$A = O(n^2/T^2 \log^2 n/T)$, $T \in (\Omega(\log n), O(\log^2 n))$, (O.T.L.P),

$A = O(n^2/T)$, $T \in (\Omega(\log^2 n), O(n))$, (O.T.L.P).

This design can be used in the resident matrix case with models GP, BP, and in the non-resident matrix case with GP model only.

3) GRID OF BASIC MODULES

In fig. 5 another structure is shown, which performs block matrix-vector products. The design consists of a mesh of processors of the type shown in fig.3. Each processor performs the multiplication of a kxk submatrix by a k-vector and accumulates partial results, with $k=O(\sqrt{n})$. The area of a processor is $O(\sqrt{n})$. The time sufficient to perform kxk matrix-vector multiplication is $O(\sqrt{n})$ and the same time is sufficient for data transfer. The whole network attains the bounds $A=O(n \sqrt{n})$ and $T=O(\sqrt{n})$.

FIG. 5

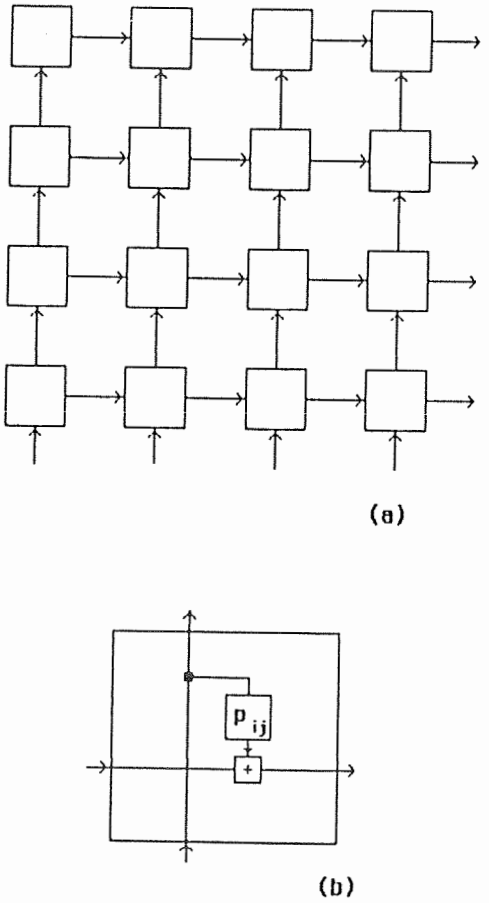


Fig.5. (a) grid of modules; (b) single module.

This design can be used in the resident matrix case with models GP, BP, and in the non-resident matrix case with GP model only.

The results obtained for the "weakly-sparse" case are summarized in table II and plotted in logarithmic scale in fig.6.

COMPLEXITY RESULTS FOR STRONGLY-SPARSE MATRICES

2) BASIC MODULE + RECURSIVE AGGREGATION

In this case, if the block size k is $O(\sqrt{n})$ the area of the basic modules is $O(1)$. The total area is:

$$A = O(n^2/k^2 + n^2/k^2 \log^2(n/k))$$

and the time is

$$T = O(\log(n/k) + k).$$

Then

$$A = O(n^2/T^2 \log^2 n/T), \quad T \in (\Omega(\log n), O(\sqrt{n})), \quad (O.T.L.F)$$

This design can be used in the resident matrix case with models GP, BP, and in the non-resident matrix case with GP model only.

3) GRID OF BASIC MODULES

In this case the area of the basic module is $O(1)$. Then the network attains the performance $A=O(n)$ and $T=O(\sqrt{n})$.

This design can be used in the resident matrix case with models GP, BP, and in the non-resident matrix case with GP model only.

4) LINEAR ARRAY OF ELEMENTARY PROCESSORS

The linear array structure presented in section 2 can be formed with basic modules of area $O(1)$ and period $p(k)=O(k)$, with $k=O(\sqrt{n})$. Moreover $c=b=1$, so that

$$A = O(\sqrt{n}); \quad T = O(n).$$

This design can be used in the non-resident matrix case with GP, BP models.

The results obtained for the "strongly-sparse" case are summarized in table III and plotted in logarithmic scale in fig.7.

5. APPLICATIONS

In this section we show that efficent implementation of matrix-vector multipliers leads to efficient designs for other important numerical problems.

It is easy to see that, in any model, the VLSI complexity of sparse matrix by vector product and of the computation of one step of an iterative method for the solution of linear systems, are of the same order.

Assume first that a network with area A, time T, and b input gates computes the product of an nxn sparse matrix by an n-vector. It is easy to see that the area A^* and the time T^* sufficient to perform one step of an iterative method, are given by:

$$A^* = O(\sqrt{A} + b)^2 + n = O(A),$$

FIG. 6

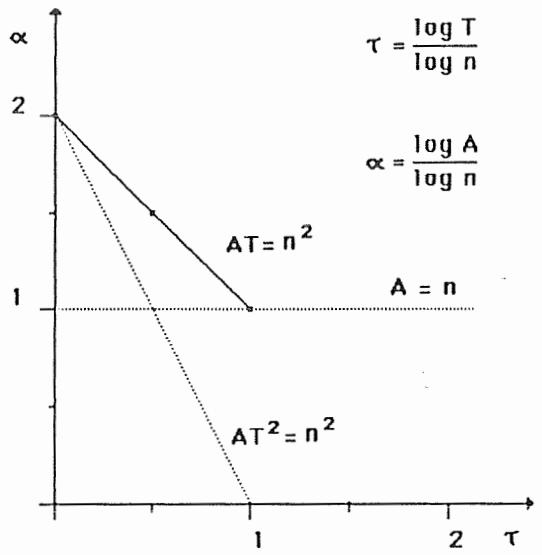


Fig.6. Upper and lower bounds for the case of weakly-sparse matrices plotted in log-log scale.

TABLE II

WEAKLY-SPARSE MATRICES

MODEL	LOWER BOUND	UPPER BOUND	DESIGN	OPTIMALITY
a) Resident matrix				
		$A = O(n)$, $T = O(n)$	(1)	
GP, BP	$AT^2 = \Omega(n^2)$, $A = \Omega(n)$	$A = O(n^2/T^2 \log^2 n/T)$, $T \in (\Omega(\log n), o(\log^2 n))$	(2)	O.T.L.F.
		$A = O(n^2/T)$, $T \in (\Omega(\log^2 n), O(n))$	(2)	
		$A = O(n^{3/2})$, $T = O(n)$	(3)	
b) Non-resident matrix				
		$A = O(n)$, $T = O(n)$	(1)	
GP	$AT^2 = \Omega(n^2)$	$A = O(n^2/T^2 \log^2 n/T)$, $T \in (\Omega(\log n), o(\log^2 n))$	(2)	O.T.L.F.
		$A = O(n^2/T)$, $T \in (\Omega(\log^2 n), O(n))$	(2)	
		$A = O(n^{3/2})$, $T = O(n)$	(3)	
BP	$AT^2 = \Omega(n^2)$,	$A = O(n)$, $T = O(n)$	(1)	

FIG. 7

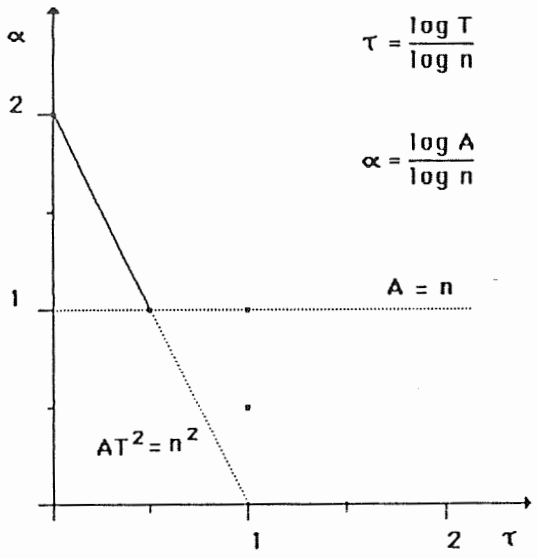


Fig.7. Upper and lower bounds for the case of strongly-sparse matrices plotted in log-log scale.

TABLE III

STRONGLY-SPARSE MATRICES

MODEL	LOWER BOUND	UPPER BOUND	DESIGN	OPTIMALITY
a) Resident matrix				
		$A = O(n),\ T = O(n)$	(1)	
GP, BP	$AT^2 = \Omega(n^2),$ $A = \Omega(n)$	$A = O(n^2/T^2 \log^2 n/T),$ $T \in (\Omega(\log n), o(n^{1/2}))$	(2)	O.T.L.F.
		$A = O(n),\ T = O(n^{1/2})$	(3)	OPTIMAL
b) Non-resident matrix				
		$A = O(n),\ T = O(n)$	(1)	
GP	$AT^2 = \Omega(n^2),$	$A = O(n^2/T^2 \log^2 n/T),$ $T \in (\Omega(\log n), o(n^{1/2}))$	(2)	O.T.L.F.
		$A = O(n),\ T = O(n^{1/2})$	(3)	OPTIMAL
		$A = O(n^{1/2}),\ T = O(n)$	(4)	
BP	$AT^2 = \Omega(n^2),$	$A = O(n),\ T = O(n)$	(1)	
		$A = O(n^{1/2}),\ T = O(n)$	(4)	

$$T^* = O(T + n/b) = O(T).$$

The above observation show that iterative methods for the solution of linear systems can be performed with a VLSI design for matrix-vector product, with the matrix resident into the circuit.

Moreover it is worth noting that eigenvalues computation by Lanczos methods [1], operations concerning polynomials, iterative methods for unconstrained optimization [2], can be efficiently implemented in VLSI, by using the designs for matrix-vector multiplication presented in this paper.

REFERENCES

[1] J. Cullum and R.A. Willoughby, Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol.1 and 2, Progress in Scientific Computing Series, Birkhauser, Boston, 1984.

[2] R. Fletcher, Pratical Methods of Optimization, Wiley, 1980.

[3] H.T. Kung and C.E. Leiserson, Algorithms of VLSI Processor Arrays, in: C.A. Mead and L.A. Conway, eds., Introduction to VLSI Systems (Addison-Wesley, Reading, MA, 1980) pp. 271-292.

[4] F.T. Leigthon, New Lower Bound Tecniques for VLSI. Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science (1981), pp. 1-12.

[5] C.E. Leiserson, Area-efficient graph layouts (for VLSI). Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (1980), pp. 270-281.

[6] J. Ja'Ja', The VLSI Complexity of Selected Graph Problems, J. Assoc. Comput. Mach., 31 (1984), pp. 377-391.