ALGORITHM PARTITION FOR A FIXED-SIZE VLSI ARCHITECTURE USING SPACE-TIME DOMAIN EXPANSION†

H. D. Cheng and K. S. Fu
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Abstract

The space-time domain expansion method has recently been used to transform a computational task with a recursive formula into a VLSI architecture. In addition to its simplicity and completeness, an important advantage of this method is that it can easily solve the problem of partitioning an algorithm to fit a fixed size VLSI architecture. We propose a computational model and a partition rule which can be easily used to partition any recursive computation problem suited to the space-time domain expansion method so it can be solved on fixed-size VLSI architectures. Several examples, such as partitioned vector inner product, partitioned comparators in relational database management, partitioned matrix multiplications, and partitioned transitive closure computation, parallel recognition of general context-free languages, string matching and dynamic time-warp pattern-matching are used to illustrate the proposed method.

Index terms: Space-time domain expansion, recursive task, pipelining, multiprocessing, very large scale integration (VLSI), algorithm partition.

1. INTRODUCTION

There are many articles [3 - 12] about the design of special VLSI architectures for specific applications. There are also some works which attempt to develop general methods for designing VLSI architectures systematically [1,13-19]. Most of these methods assume no limitation on the size of VLSI architecture, or the number of processing units on the VLSI chip. Such an assumption appears to be unrealistic, since there are several factors (physical , technology and complexity limitations) that actually constrain the integration level of the silicon IC technology [20]. Even by the late 1980's, it could only be possible to fabricate 10^7 or 10^8 transistors on a monolithic chip [21], which may be much smaller than the sizes required by many large-scale computation problems. The assumption of no size limitation of VLSI architectures is also not very economical and convenient because we have to design VLSI architectures with different sizes for different computation problems even they may use the same algorithm. We have to find methods to partition computation problems so they can be solved on fixed-size VLSI architectures.

We will use 'algorithm partition' for solving a problem using fixed-size VLSI architectures and use 'recursive task' to represent a task with recursive mathematical formula in the following discussions.

Although algorithm partition becomes a key to extend the computational capability of VLSI architectures, so far only a few papers have paid attention to this problem [22, 28, 29]. In this paper we propose a partitioning method based on the space-time domain expansion. The application of the proposed method is illustrated by examples.

† This work was supported by the NSF Grant ECS 80-16580

2. SPACE-TIME DOMAIN EXPANSION AND COMPUTATIONAL MODEL

Let S represent space domain and T represent time domain. The space-time domain is then the set of $\{S,T\}$ which, in general, is a (n +1)-dimensional vector $\{x_1,x_2,\cdots,x_n \mid x_i \varepsilon S \;\; i=1,2,\cdots,n, \text{ and } t \varepsilon T\}$ But for the real world , we only concern the case of at most 4-dimensional vectors $\{x_1,x_2,x_3,t\}$

Along x_i , a k-space expansion means that the processing unit repeats uniformly k times along the x_i direction, as shown in Fig.1. The structure in Fig.1(b) is called the k-space expansion of the structure in Fig. 1(a) ; on the other hand the structure in Fig. 1(a) is called the k-space condensation of the structure in Fig. 1(b). The space-expansion (condensation) can be performed at several levels , each processing unit could be a gate, a processing element (PE) , a group of PEs, a processor , a group of processors or a processing system, etc. In this paper, we only consider processing unit consisting of a PE or a group of PEs.

A K-time expansion means that J events occur sequentially and each adjacent pair of the events has equal time interval (1 time unit) . Its configuration is shown in Fig.2. The structure in Fig.2(b) is called the K-time expansion of the structure in Fig 2(a); on other hand the structure in Fig. 2(a) is called the K-time condensation of the structure in Fig 2(b). The time-expansion (condensation) can also be performed at several levels, each event could be a datum, a block of data, a task, or a group of tasks, etc. We only consider an event as a block of data in this paper.

We propose the following rules for space-time domain expansion to design a VLSI architecture for solving a recursive task [1].

(1) Space-expansion rule

Input data of the Kth processing unit should spend K time units to reach this processing element to maintain the time consistency. The space expansion essentially uses the multiprocessing technique.

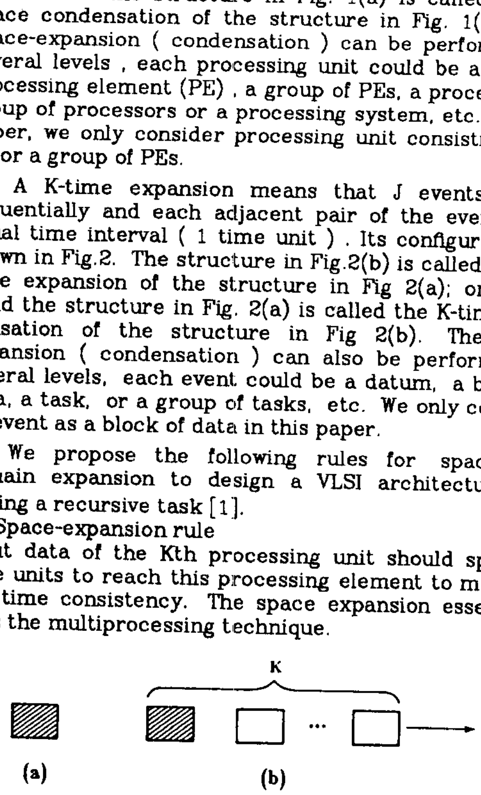


Figure 1. (a) Single processing unit, (b) K-space expansion

(2) Time-expansion rule
Input data of every processing unit must be expanded in time domain, the Jth data needs J steps (each needs one time unit) to reach the processing element to maintain the space consistency. The time expansion essentially uses the pipelining technique.
(3) Rule of mapping a recursive algorithm into a VLSI architecture
For the index (indices) in the recursive formula (or of the program loop) we can apply space and time expansions alternatively such that each index corresponds to one expansion and the processing unit performs the mathematical computation in the formula. In general, space and time expansions are applied alternately, can we obtain higher performance.

There are several measurements for a computational task. We will use problem size N which is the number of operations needed to solve the given task to measure a computational task. The computation model of a VLSI architecture obtained by space-time domain expansion can be described by the tuple $\{K_1, K_2, K_3, Q_1, \cdots, Q_l\}$. Here K_i denotes a K_i-space expansion along the x_i direction and Q_j denotes a Q_j-time expansion in the jth expansion. K_i equals to one if there is no space-expansion along x_i direction , and Q_j equals to one if there is no expansion in the jth time-expansion. The necessary condition for solving a recursive task by using the space-time domain expansion is :

$$\prod K_i \times \prod Q_j \geq N \qquad (1)$$

which indicates that the VLSI architecture based on the space-time expansion can perform a number of operations equal to the product of all the space-time domain expansions.

(4) Partitioning rule
If we make a k-space condensation along the x_i direction , then we have to also make a k-time expansion. Multi-dimensional condensations need multi-time expansions which may require some input data to be used repeatedly.

In the next section , we will use some examples to illustrate how to apply the proposed computation model and the partitioning rule to partition a given algorithm.

3. PARTITIONING ALGORITHMS USING THE PROPOSED METHOD

3.1. Partitioning Vector Inner Product

The inner product of two vectors can be expressed as :

$$C = \sum_{i=1}^{i=N} a(i) \times b(i) \qquad (2)$$

Its recursive formula is

$$C^0 = 0;$$
$$C^i = C^{(i-1)} + a(i) \times b(i) \qquad 1 \leq i \leq N;$$
$$C = C^N. \qquad (3)$$

We can solve it in three different ways by using different structures.

3.1.1. Using a single processing unit with feedback

The processing unit has the structure as shown in Fig.2(a). Since $K_1 = K_2 = K_3 = 1$, according Eq.(1), we obtain $Q_1 \geq N$. It means that a N-time expansion is needed, and the structure is shown in Fig.2(b).

3.1.2. Using an one-dimensional array of processing units without time-expansion

Since $Q_j = 1$ for *all* j and $K_2 = K_3 = 1$, according Eq.(1), we obtain
$K_1 \geq N$
It means that we need to make an N-space expansion along the x_1 direction. The structure is shown in Fig.2(c) and obtained by using the space-expansion rule, details can be found in [1].

For reducing the number of pins, we could use the structure in Fig. 2(d) and an asymmetric two-phase clock.

The structure in Fig.2(b) and the structure in Fig.2(c) can perform the same operation and require the same amount of time (N time units) to complete the task, but the structure in Fig.2(b) only needs one processing unit, because the time-expansion can improve the utilization of the system, in this case the utilization is 100%. If we want to compute multiple vector inner products, we can make one more time-expansion, then the structure in Fig.2(c) can gain the speedup, but the structure in Fig.2(b) can not. It is simply because of the fact that a space-expansion followed by a time-expansion can speed-up the computation.

3.1.3. Using a size-K VLSI one-dimensional array

The size K one-dimensional array can be considered as a K-space expansion of the structure in Fig.2(a) and a $\frac{N}{K}$-space condensation of the structure in Fig.2(c). The input data should be arranged according to the space-expansion rule. Since $K_2 = K_3 = 1$, following Eq.(1), we have to make a $\left\lceil \frac{N}{K} \right\rceil$-time expansion. The input data a(i) and b(i) become a(m,n) and b(m,n) with the following relations:

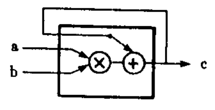
$$a(m,n) = a(i) ;$$
$$b(m,n) = b(i) ;$$
$$m = \left\lfloor \frac{i}{(K+1)} \right\rfloor + 1 ;$$
$$n = i - \left\lfloor \frac{i}{(K+1)} \right\rfloor \times K ; \qquad (4)$$

Since there is data-dependency between the operations of this problem, we need to add one more cell - an accumulator controlled by the signal sent from the host machine. The resulting structure is shown in Fig.2(e). The accumulator initially is set to zero and will be stimulated at (K+1)st time-unit by the input data and reset at the end of each $\left\lceil \frac{N}{K} \right\rceil$ accumulations which could be controlled by the host machine. The output has to be removed before the reset. If $N\ (mod\ K) \neq 0$, the remaining portion of the last row has to be filled with zeros. Using the structure in Fig.2(e), we can compute vector inner product of any size on a fixed-size VLSI linear array.

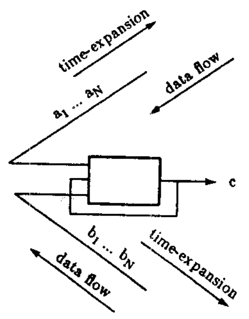
Consider the computation of M pairs of size N vector inner products,

$$C(i) = \sum_{j=1}^{j=N} a(i,j) \times b(i,j), \quad 1 \leq i \leq M \qquad (5)$$

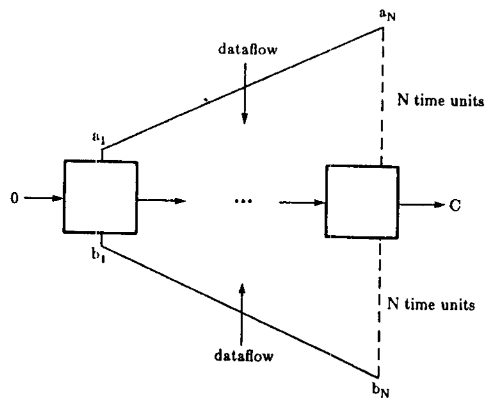
we can use one of the structures in Fig.2(b) and Fig.2(c) by performing one M-time expansion. We can also use the structure in Fig.2(e) by performing a M-time expansion, then the structure in Fig.2(f) is obtained. It needs $M \times \left\lceil \frac{N}{K} \right\rceil + K$ time units to perform the computations.



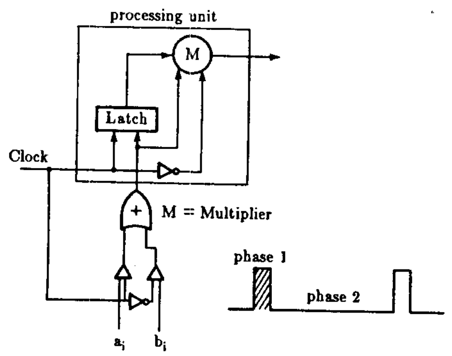
(a) Single processing unit with feedback,



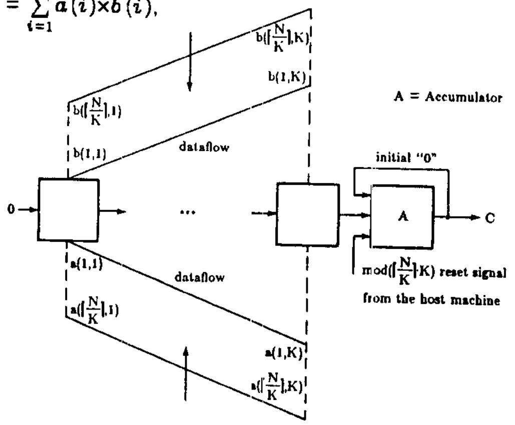
(b) N-time expansion of the structure in (a) for computing $C = \sum_{i=1}^{i=N} a(i) \times b(i)$,



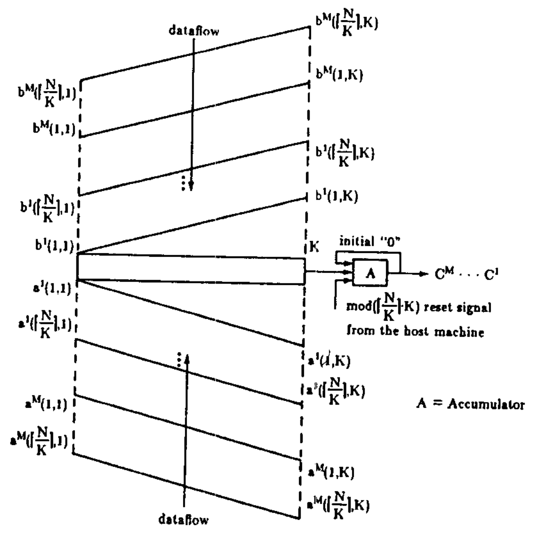
(c) N-space expansion of the structure in (a) for solving $C = \sum_{i=1}^{i=N} a(i) \times b(i)$,



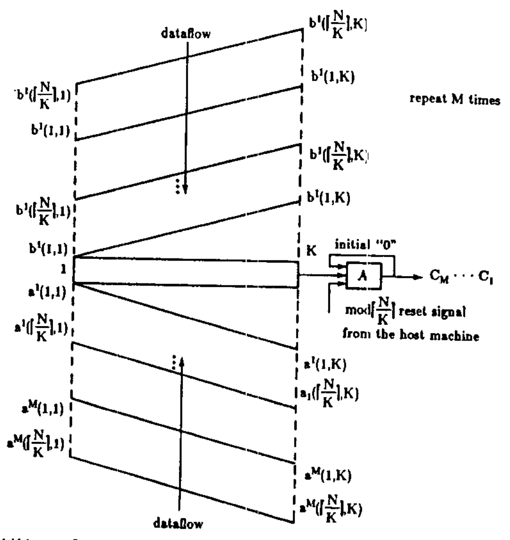
(d) The structure of the processing unit for reducing the number of input pins,


$b(\lceil \frac{N}{K} \rceil, K)$
$b(1,K)$
$b(\lceil \frac{N}{K} \rceil, 1)$
$b(1,1)$
dataflow
A = Accumulator
initial "0"
0
A
C
$a(1,1)$
dataflow
$\text{mod}(\lceil \frac{N}{K} \rceil K)$ reset signal from the host machine
$a(\lceil \frac{N}{K} \rceil, 1)$
$a(1,K)$
$a(\lceil \frac{N}{K} \rceil, K)$

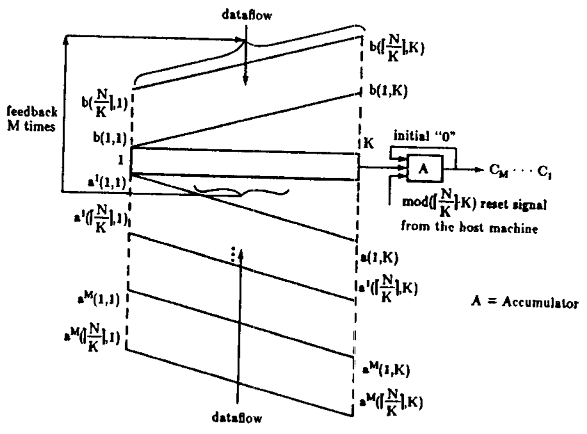
(e) partitioned vector inner product implementation for vectors with any size ,


dataflow
$b^M(\lceil \frac{N}{K} \rceil, K)$
$b^M(1,K)$
$b^M(\lceil \frac{N}{K} \rceil, 1)$
$b^M(1,1)$
$b^1(\lceil \frac{N}{K} \rceil, K)$
$b^1(1,K)$
$b^1(\lceil \frac{N}{K} \rceil, 1)$
$b^1(1,1)$
K
initial "0"
A
$C^M \cdots C^1$
$a^1(1,1)$
$a^1(\lceil \frac{N}{K} \rceil, 1)$
$\text{mod}(\lceil \frac{N}{K} \rceil K)$ reset signal from the host machine
$a^1(1,K)$
$a^1(\lceil \frac{N}{K} \rceil, K)$
A = Accumulator
$a^M(1,1)$
$a^M(\lceil \frac{N}{K} \rceil, 1)$
$a^M(1,K)$
$a^M(\lceil \frac{N}{K} \rceil, K)$
dataflow

(f) M-time expansion of the structure in (e),


dataflow
$b^1(\lceil \frac{N}{K} \rceil, K)$
$b^1(1,K)$
repeat M times
$b^1(\lceil \frac{N}{K} \rceil, 1)$
$b^1(1,1)$
$b^1(\lceil \frac{N}{K} \rceil, K)$
$b^1(1,K)$
$b^1(\lceil \frac{N}{K} \rceil, 1)$
$b^1(1,1)$
1
K
initial "0"
A
$C_M \cdots C_1$
$a^1(1,1)$
$a^1(\lceil \frac{N}{K} \rceil, 1)$
$\text{mod}\lceil \frac{N}{K} \rceil$ reset signal from the host machine
$a^1(1,K)$
$a_1(\lceil \frac{N}{K} \rceil, K)$
$a^M(1,1)$
$a^M(\lceil \frac{N}{K} \rceil, 1)$
$a^M(1,K)$
$a^M(\lceil \frac{N}{K} \rceil, K)$
dataflow

(g) Partitioned matrix-vector multiplication for matrix-vector pairs of any size ,


dataflow
$b(\lceil \frac{N}{K} \rceil, K)$
$b(1,K)$
feedback M times
$b(\lceil \frac{N}{K} \rceil, 1)$
$b(1,1)$
1
$a^1(1,1)$
K
initial "0"
A
$C_M \cdots C_1$
$\text{mod}(\lceil \frac{N}{K} \rceil K)$ reset signal from the host machine
$a^1(\lceil \frac{N}{K} \rceil, 1)$
$a(1,K)$
$a^1(\lceil \frac{N}{K} \rceil, K)$
A = Accumulator
$a^M(1,1)$
$a^M(\lceil \frac{N}{K} \rceil, 1)$
$a^M(1,K)$
$a^M(\lceil \frac{N}{K} \rceil, K)$
dataflow

(h) Another implementation of partitioned matrix-vector multiplier.

Figure 2.

Note that all the patterns of the input data will be repeated M times in a M-time expansion.

If the processing unit performs comparison instead of multiplication and logic AND instead of addition and accumulation and the additional processing unit is initially set to logic one, the structure in Fig.2(e) can be used as a comparator in relational database management to perform any size 'comparison' computations. Also after a time-expansion, it can be used for a sequence of comparisons.

3.2. Partitioning Matrix-Vector Multiplication

Consider a matrix A with size MxN and a vector B with size N The multiplication of A and B is a vector C with size M and is defined as :

$$C(i) = \sum_{j=1}^{j=N} a(i,j) \times b(j), \quad 1 \le i \le M \qquad (6)$$

Eq.(6) can be considered as a special case of Eq.(5) when $b(i,j) = b(j)$ for *all* i. This can be implemented by the architecture in Fig.2(g) or the architecture in Fig.2(h), no matter what sizes they are.

Since convolution and FFT (Fast Fourier Transform) have similar computational properties as matrix-vector multiplication, they can also be solved by the architecture in Fig.2(g) or the architecture in Fig.2(h), no matter what sizes they are. Certainly, we can also make another time-expansion to perform multiple matrix-vector multiplications, multiple convolutions, and multiple FFT.

3.3. Partitioning Matrix Multiplication

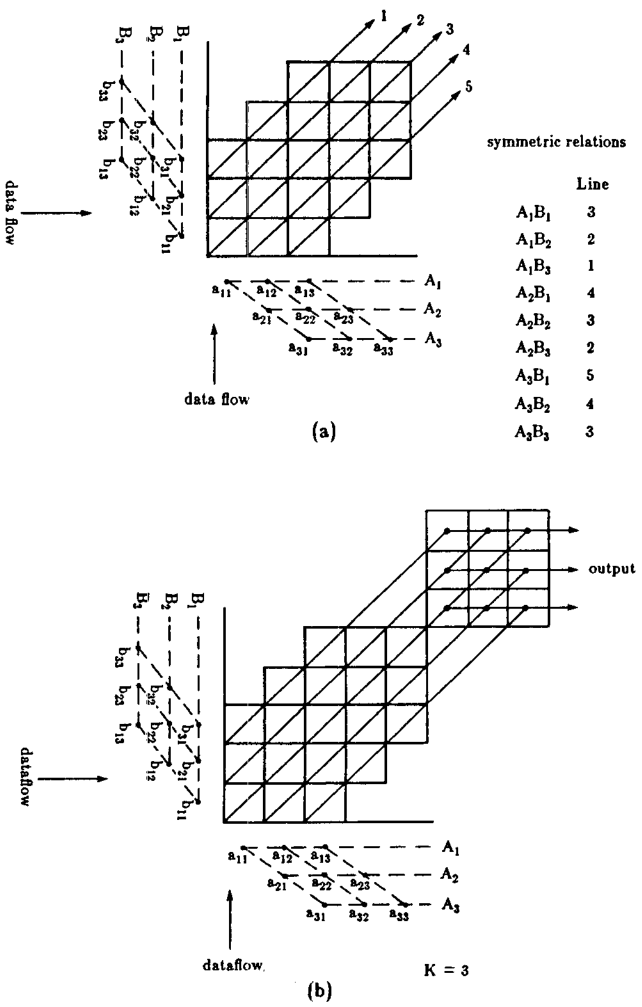
Without lossing generality, we can assume that two matrices A and B both have size MxM (actually it merely needs that they are compatible). The multiplication of two matrices is defined as :

$$C(i,j) = \sum_{k=1}^{k=M} a(i,k) \times b(k,j), \quad 1 \le i,j \le M \qquad (7)$$

Its recursive form is:

$$C^0 = 0 ;$$
$$C^k(i,j) = C^{(k-1)}(i,j) + a(i,k) \times b(k,j) ;$$
$$C(i,j) = C^M(i,j) \quad 1 \le i,j \le M . \qquad (8)$$

Using space-time domain expansion rules, we can obtain the MxM structure in Fig.3(a). It needs totally $3M^2 - 3M + 1$ processing units and $2M - 1$ time units for the computations. If we want to use a VLSI architecture with size KxK to solve this problem, according Eq.(1) and the partitioning rule, we need to make two $\left\lceil \frac{M}{K} \right\rceil$-time expansions, and one M-time expansion. The input matrices are partitioned into $\left\lceil \frac{M}{K} \right\rceil^2$ KxK submatrices. The submatrices of A and B matrices are the inputs to the KxK VLSI architecture sequentially according to the time-expansion rule. Since there is data-dependency in the matrix multiplications, except the KxK VLSI module based on the space-time domain expansion We have to add KxK processing units controlled by the signal sent from the host machine, as shown in Fig.3(b). Each processing unit has an accumulator and an one-time-unit delay and a Flip-Flop which stores the output of the accumulator or the output of Flip-Flop of the left-side processing unit. All the accumulators are initialized at zero and will start at 2Kth time-unit after inputing the



symmetric relations	Line
A_1B_1	3
A_1B_2	2
A_1B_3	1
A_2B_1	4
A_2B_2	3
A_2B_3	2
A_3B_1	5
A_3B_2	4
A_3B_3	3

Figure 3. (a) Full sized VLSI matrix multiplication architecture based on the space-time domain expansion approach, (b) size KxK partitioned matrix multiplier (module) and the VLSI structure for partitioned matrix multiplication.

first pair of submatrices. The accumulators perform the operations in a synchronous manner with the clock period K and will be reset at the end of each $\left\lceil \frac{M}{K} \right\rceil$ accumulations. The results have to be removed just before the resets. If we remove all the outputs simultaneously, we need 2K-1 output pins, it may be difficult when K is very large. To reduce the number of output pins, we can chain the Flip-Flops in the same row and form K output channels which require only K output pins. The data in the left-side Flip-Flop will be shifted into the right-side Flip-Flop in one time unit. Since the clock period is K, before the change of the accumulator occurs, it can shift K positions to the right and all accumulations can be moved out with only K output pins.

If $M \ (mod\ K) \ne 0$, it may be one of the following two cases:

(1) $K > M$
Input data need to be filled with zeros and only MxM outputs are needed.

(2) $K < M$
The submatrices in the last column and the last row have to be filled with zeros.

From the above discussion, we know that the KxK VLSI module consisting of ($4K^2-3K+1$) processing units as shown in Fig.3(b) can solve matrix multiplication of any size. It needs ($2K + K\times\left\lceil \frac{M}{K}\right\rceil^3$) time units to complete the computations When M>>K, $\frac{M^3}{K^2}$ time units are needed. For the structure in Fig.3(b), if we make a $\left\lceil\frac{M}{K}\right\rceil$-space expansion, according the partitioning rule, then we only need one $\left\lceil\frac{M}{K}\right\rceil$-time expansion and one M-time expansion. It needs $\left\lceil\frac{M}{K}\right\rceil$ KxK VLSI modules and ($M\times\left\lceil\frac{M}{K}\right\rceil + 2K$) time units to complete the computation ; when $M \gg K$, it becomes ($M\times\left\lceil\frac{M}{K}\right\rceil$) time units. For the obtaining structure, if we make another $\left\lceil\frac{M}{K}\right\rceil$-space expansion, then we only need one M-time expansion. It needs $\left\lceil\frac{M}{K}\right\rceil^2$ KxK VLSI modules and (M+2k) time units to complete the computation; when $M >>K$ it becomes M time units. Of cause we can make another time-expansion for the above structures to perform multiple pair of matrices multiplications, the details are omitted here.

3.4. Partitioning the Computation of Transitive Closure

The transitive closure problem has many practical applications such as process synchronization, data flow analysis of computer programs, and many graph-theoretic problems [10,23-25]. The fact of that the computation of transitive closure can be partitioned is also indicated in [10].

We use a boolean matrix A to represent a directed graph and let the nodes of the graph be 1,2,...,N. The element a_{ij} of the matrix A is 1 if and only if there is an arc from i to j, and 0 otherwise. The transitive closure of A representing a graph G is the matrix A^+ whose (i,j) entry is 1 if and only if there is a path of length zero or more from node i to node j , and 0 otherwise. Its recursive formula is:

$$a_{ij}^{(t+1)}=a_{ij}^t+a_{it}^t * a_{tj}^t, \quad 1 \le t \le N, \text{ for all } 1 \le i,j \le N. \quad (9)$$

Here + indicates logic OR operation and * indicates logic AND operation. There are several algorithms for computing the transitive closure. Here we consider the structure proposed by L.J.Guibas, et al [10,23] as shown in Fig.4(a). The algorithm is as follows:
Using a processor mesh with the connections between top and bottom and between left and right for feedbacking the data. At each processor A_{ij}, the following operations are performed:
1. $A_{ij} = A_{ij} + a_{it} * a'_{tj}$
2. When a_{ij} arrives at A_{ij} , set $a_{ij}=A_{ij}$, and the other $a_{ik's}(k \neq j)$ are unchanged. Similarly for a'_{ij}.
After three passes the transitive closure is obtained in A_{ij}. The time complexity for this algorithm is O(N).

If we want to use a VLSI architecture with size KxK to solve the transitive closure problem, according Eq.(1) and the partition rule, we need to make two $\left\lceil\frac{N}{K}\right\rceil$-time expansions, and one N-time expansion similar to the case of partitioning matrix multiplication. Each input matrix is partitioned into $\left\lceil\frac{N}{K}\right\rceil^2$ KxK submatrices. We number

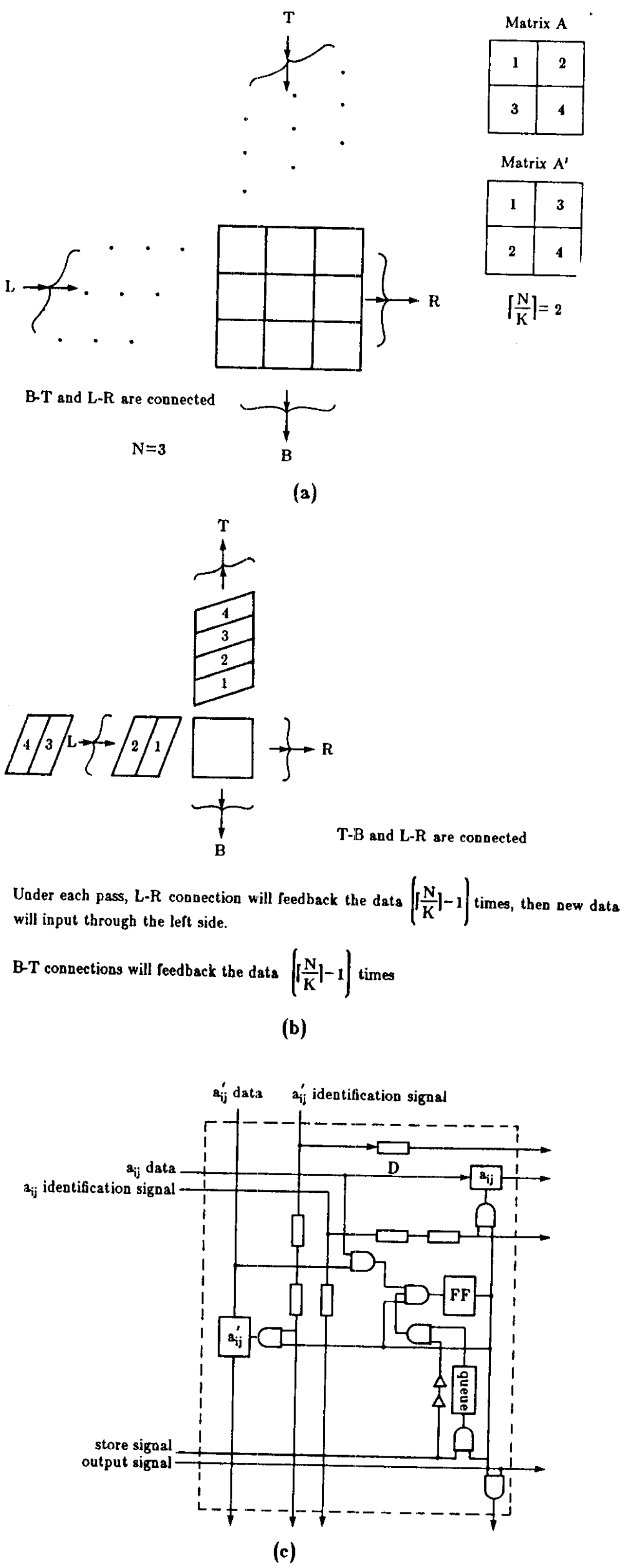

T
Matrix A
1
2
3
4
Matrix A'
1
3
2
4
$\lceil\frac{N}{K}\rceil = 2$
L
R
B-T and L-R are connected
N=3
B
(a)
T
4
3
2
1
4
3
L
2
1
R
B
T-B and L-R are connected
Under each pass, L-R connection will feedback the data $\left(\lceil\frac{N}{K}\rceil-1\right)$ times, then new data will input through the left side.
B-T connections will feedback the data $\left(\lceil\frac{N}{K}\rceil-1\right)$ times
(b)
a'_{ij} data
a'_{ij} identification signal
a_{ij} data
a_{ij} identification signal
D
a_{ij}
FF
a'_{ij}
queue
store signal
output signal
(c)

Figure 4. (a) The structure for computing the transitive closure, (b) VLSI architecture with size KxK for partitioning transitive closure computation, (c) The structure of each processing unit and the control mechanism.

the submatrices of the matrix A in a row-major manner and the submatrices of the matrix A' in a column-major manner. The submatrices will input to the given KxK VLSI structure sequentially according to the time expansion rule as shown in Fig.4(b). If $N \ (\ mod\ K\ ) \neq 0$, it may be one of the two cases as discussed in Section 3.3.

The structure of each processing unit and the control mechanism are shown in Fig.4(c) which are more complicated than those for matrix multiplications since the algorithm of the transitive closure is more complex than the one for matrix multiplication. Each processing unit has a local memory which could be a shift register with $\left\lceil \frac{N}{K} \right\rceil^2 +1$ bits forming a queue. The queue is initialized to zero. The host machine will send the identification signals for a_{ij}'s at the time unit according to the following formula:

$$\left\lceil \frac{N}{K} \right\rceil (j-1)K+(j-1)K+\left\lceil \frac{N}{K} \right\rceil^2 K(i-1)+1 \quad 1\leq i,j \leq \left\lceil \frac{N}{K} \right\rceil . \quad (10)$$

It will also send the identification signals for a'_{ij}'s at the time unit according to the following formula:

$$\left\lceil \frac{N}{K} \right\rceil (j-1)K+(i-1)K+\left\lceil \frac{N}{K} \right\rceil^2 K(i-1)+1 \quad 1\leq i,j \leq \left\lceil \frac{N}{K} \right\rceil . \quad (11)$$

After completing the third pass , the host machine sends the output signal which controls the contents of the accumulators moving downward one row for each time unit. It needs K output pins and its time complexity is $O(\frac{N^3}{K^2})$. Certainly, we can make a space expansion to speed up the computation. We make a $\left\lceil \frac{N}{K} \right\rceil$-space expansion of the structure in Fig.4(b) along the x_1 direction to speed up the computation. If we make another $\left\lceil \frac{N}{K} \right\rceil$-space expansion along the x_2 direction, we will obtain the structure similar to the one in Fig.4(a). (see [2] for the details.)

The structure in Fig.4(b) can also be used for matrix multiplication if each processing unit performs addition and product operation, and without the second step in the algorithm.

3.5. Partitioning recognition of general context-free languages and pattern-matching

Recognition of general context-free languages has been very important in language processing and syntactic pattern recognition. Many researchers have attempted to speed-up the recognition procedure. Recently, several VLSI implementations of context-free languages recognition have been proposed[8,9]. The main disadvantage of these proposed VLSI systems is that the size of the processor array is critical to the performance of the systems. That is, an $n \times n$ upper triangular VLSI array can only process strings with length no longer than n. It will have difficulty in recognizing general context-free languages[9]. We have proposed new algorithms for recognition of general context-free languages and the algorithm partition problem has been discussed by using the space-time domain expansion approach[26]. The details are omitted here. Since these algorithms essentially speed-up the dynamic programming procedure by using highly pipelining and parallelism of VLSI architecture, the proposed parallel algorithms and the algorithm partition methods may also be useful for other problems solvable by dynamic programming.

String-matching problem arises in a number of applications such as artificial intelligence, pattern recognition and information retrieval. We have proposed a VLSI architecture based on the space-time domain expansion approach which has a very natural and regular configuration. It can compute string distance and find the edit sequence. The algorithm partition problem is also solved by using the proposed computational model and partition rule[27]. The technique of dynamic time-warping has found extensive applications in speech recognition and image processing[27]. We have proposed a VLSI architecture for dynamic time-warp pattern-matching based on the space-time domain expansion approach which results in a very natural and regular configuration. It can perform dynamic time-warp pattern-matching and find the warp-path in a much more efficient manner by using extensive pipelining and parallelism techniques. By using the proposed computational model and partition rule, we also solve the algorithm partition problem of dynamic time-warp pattern-matching. (See [27] for the details.)

4. CONCLUDING REMARKS

We have proposed a partitioning method and a computational model based on the space-time domain expansion approach. Using the given rules and the model, we can partition a recursive computational problem of any size to match a fixed-size VLSI architecture. Partitioning vector inner product computations, partitioning matrix-vector multiplications, partitioning convolution computations, partitioning FFT, partitioning matrix multiplications, partitioning transitive closure computation partitioning recognition of general context-free languages, partitioning string-matching and partitioning dynamic time-warp pattern-matching are discussed to demonstrate the application of the proposed model and method. These problems can be computed on fixed size VLSI architectures. Since the partition problem is a major step in extending the computational capability of a VLSI architecture, the proposed method will certainly expand the general usage of VLSI systems.

REFERENCES

1. H. D. Cheng , W. C. Lin and K. S. Fu, " Space-Time Domain Expansion Approach to VLSI and Its Application to Hierarchical Scene Matching," accepted for publication in *IEEE Trans. Pattern Anal. Machine Intell.*, March 1985; (Summary published in the *Proc. Pattern Recognition Conf.*, Montreal, July 1984.)
2. H. D. Cheng and K. S. Fu " Algorithm partition for a fixed-size VLSI architecture using space-time domain expansion," School of Electrical Engineering, Purdue University, West Lafayette 47907, 1984.
3. H. T. Kung, et al, *VLSI systems and Computations* , Computer Science Press, 1981.
4. H. T. Kung, " Algorithms for VLSI processor arrays," in *Introduction to VLSI systems*, edited by C. Mead and L. Conway, Addison-Wesley, 1980.
5. H. T. Kung, " Let's design algorithms for VLSI systems," Proc. of the Caltech Conference on VLSI, Jan. 1979.
6. H. T. Kung, " Why systolic architectures ?," *Computer*, Jan. 1982.
7. M. J. Forster, H. T. Kung, " The design of special-purpose VLSI chips," *Computer* , Jan. 1980.
8. K. H. Chu and K. S. Fu, " VLSI architectures for high speed recognition of general context-free languages and finite-state languages," TR-EE 81-42, Nov. 1981. School of Electrical Engineering Purdue University.

9. Yetung Chiang and K. S. Fu, " A VLSI architecture for fast context-free language recognition (Earley's algorithm)," *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 3, PAMI-6, May 1984.
10. L. j. Guibas, H. T. Kung and C. D. Thompson, " Direct VLSI implementation of combinatorial algorithms," Proc. of the Caltech Conference on VLSI , Jan. 1979.
11. H. T. Kung, et al, " Systolic (VLSI) arrays for relational database operations," Carnegie-Mellon University, Oct. 1979.
12. C. E. Leiserson, " Systolic priority queues," Proc. of Caltech Conference on VLSI, Jan. 1979.
13. Lennart Johnsson, et al, " Towards a formal treatment of VLSI arrays," Proc. of the Caltech Conference on VLSI, 1981.
14. Lennart Johnsson, et al, " A mathematical approach to modeling the flow of data and control in computational networks," in *VLSI systems and computations* , edited by H. T. Kung, et al, Computer Science Press, 1981.
15. S. Y. Kung, et al, " Wavefront array processor : language, architectures and applications," *IEEE trans. on computer*, No. 11 , Nov. 1982.
16. M. C. Chen and C. A. Mead, " The semantics of a functional language for VLSI systems," Caltech report, March 1983.
17. M. C. Chen and C. A. Mead, " Concurrent algorithms as space-time recursion equations," Proc. USC Workshop on VLSI & Modern Signal Processing,
18. D. I. Moldovan, " On the design of algorithms for VLSI systolic arrays," *Proceedings of the IEEE*, Vol. 71, No. 1, January 1983.
19. P. R. Cappello and K. Steiglitz, " Unifying VLSI array designs with geometric transformations," *IEEE Proc. Parallel Processing* , 1983.
20. C. V. Ramamoorthy and Y. W. Ma, " Large scale computer systems," *Hardware and Software Concepts in VLSI*, edited by G. Rabbat, Van Nostrand Reinhold Company, 1983.
21. C. Mead, L. Conway, *Introduction to VLSI systems*, Addison-Wesley, 1980.
22. K. Hwang and Y. H. Cheng, " Partitioned matrix algorithms for VLSI arithmetic systems," *IEEE Trans. Comput.* Vol. c-31, No. 12, December 1982.
23. J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.
24. Warren, S. W., Jr.,"A modification of Warshall's algorithm for the transitive closure of binary relations," *Comm. ACM*, Vol.18, No.4, April 1975.
25. Warshall, S., "A theorem on boolean matrices," *Journ. ACM*, Vol.9, No.1, Jan. 1972.
26. H. D. Cheng and K. S. Fu, " Algorithm partition and parallel recognition of general context-free languages using fixed-size VLSI architecture," June 1984, School of Electrical Engineering Purdue University, West Lafayette IN 47907.
27. H. D. Cheng and K. S. Fu, " VLSI architectures for pattern matching using space-time domain expansion approach," October 1984, School of Electrical Engineering Purdue University, West Lafayette IN 47907.
28. D. I. Moldovan and J. A. B. Fortes, " Partitioning of algorithms for fixed size VLSI architectures," TR PPP-83-5, Dept. of EE Systems, USC, Los Angeles, 1983.
29. D. I. Moldovan, C. I. Wu and J. A. B. Fortes, " Mapping an arbitrarily large QR algorithm into a fixed size VLSI array," *Proc. of 1984 Int'l Conf. on Parallel Processing*, Aug. 1984.