Computers Math. Applic. Vol. 21, No. 2-3, pp. 103–109, 1991
Printed in Great Britain. All rights reserved

0097-4943/91 $3.00 + 0.00
Copyright© 1991 Pergamon Press plc

GENERATING SUBSETS ON A SYSTOLIC ARRAY

CHAU-JY LIN
Department of Applied Mathematics, National Chiao Tung University,
Hsinchu, Taiwan, 30050, Republic of China

(Received February, 1990 and in revised form May, 1990)

Abstract—Given n elements and an arbitrary integer m for $m \leq n$, a systolic algorithm for generating all r–subsets (subsets containing r elements) with $1 \leq r \leq m$ in lexicographic order is presented. The computational model used is a linear systolic array consisting of m identical processing elements with a simple structure. One subset is produced at a time step. The elapsed time within a time step is independent of integers r, n and m. The design process of systolic array and the verification of systolic algorithm are considered in detail.

1. INTRODUCTION

There are many problems in science and engineering which require a computer having high computation speed and being able to solve them in real–time. Using parallel computers is a way to achieve higher computing speeds. This appealing approach has greatly increased interest in the area of design and analysis of parallel algorithms. Systolic arrays were introduced by H. T. Kung [4] and his colleagues in Carnegie–Mellon University. It is specified by the timing of data movement and interconnection of processing elements (PEs) such that the movement of data is simple, regular and uniform. These systolic arrays are made up of identical PEs that operate synchronously. Thus it is suitable for VLSI implementation. The parallel algorithms which can be executed on systolic arrays are called *systolic algorithms*. To solve a problem with a systolic algorithm, we have to do the following three things: (1) to determine the topology of a systolic array, (2) to propose a design strategy for deriving a systolic algorithm, and (3) to prove the correctness of a systolic algorithm.

Given n elements and an integer m with $m \leq n$, many algorithms for generating m–subsets (subsets containing m elements) have been proposed, see [1–3]. Two sequential algorithms for generating all of r–subsets (for $1 \leq r \leq m \leq n$) in lexicographic order are presented in [5,6], respectively. In this paper we present a systolic algorithm to generate all of r–subsets in lexicographic order. The computational model used is a linear systolic array consisting of m identical PEs with a simple structure. All PEs perform the same program in an arbitrary time step. The elapsed time for producing a subset is constant.

A parallel algorithm to generate $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ m–subsets was presented in [1]. This algorithm uses arbitrary k processors for $1 \leq k \leq \binom{n}{m}$. For each processor PE($i$), all it needs to do is: (a) to evaluate the index j of a specified m–subset, say Q_i, (b) to obtain this Q_i by applying an inverse ranking function on j, and (c) to generate sequentially an interval of subsets starting with Q_i. This design idea can be applied to generate all subsets of n elements when the sequential algorithm used in (c) is one of the algorithms presented in [5,6]. Here we design a parallel algorithm to generate r–subsets under a different design consideration. There are m identical PEs denoted by PE(i) for $1 \leq i \leq m$ to be used. The responsibility of PE(i) is to evaluate the ith component of each subset (including the empty element). The m components of an arbitrary subset are coming out simultaneously.

I would like to thank the referees for their comments. This research was supported by National Science Council of Republic of China under Grant NSC 79–0208–M009–22.

2. AN OVERVIEW OF SYSTOLIC ARRAYS

A systolic array can be viewed as a network composed of a few types of computational PEs. In our systolic array, its data communication is only allowed between two adjacent PEs because it has no shared memory and prohibits the behavior of data broadcasting. That is, let PE1 and PE2 be two PEs in an existing systolic array, if it is necessary to send data from PE1 to PE2 then there exists a communication link (say e–link) from PE1 to PE2. We call such an e–link an ***input link*** of PE2 and an ***output link*** of PE1. We also write e_{in} and e_{out} to denote an input value of PE2 and an output value of PE1 via an e–link, respectively. We assume that each PE can perform the following three tasks (phases):

(1) to receive data from its input links (read phase),

(2) to execute the loop of a systolic algorithm once (computation phase),

(3) to send data to its output links (write phase).

The maximal time (considering all PEs) to do the above three tasks is called a *time step*. Moreover, if an e–link is labeled with δ delays (denoted by δD) for δ a positive integer, it means that when PE1 sends its e_{out} at the time step t_1, then such e_{out} is the e_{in} of PE2 at the time step $t_1 + \delta$. For the sake of convenience, we also use e_{in}, e_{out} as the names of variables in our systolic algorithm. The symbol δD on a link will be omitted if $\delta = 1$. We will show that each communication link used in our systolic array has $\delta = 1$.

3. THE DESIGN PROCESS FOR GENERATING SUBSETS

Without loss of generality, the given n elements are denoted by $1, 2, \ldots, n$. We write *sublex* to denote the set which contains all of r-subsets of $\{1, 2, \ldots, n\}$ (for $1 \le r \le m \le n$) in lexicographical order. We use an xy–plane with integer coordinates to describe the design consideration of our systolic array. The x–axis is the index of PEs and the y–axis the time step as shown in Figure 1. Since we require the components of a subset to be come out at the same time step, and there are m components in a subset (the empty component will be considered as blank), the number of PEs used is m at any time step. The PEs appeared in Figure 1 can be referred to as PE(i,t) for $1 \le i \le m$ and $1 \le t$. Figure 1 is constructed as follows.

(1) The ith component of any subset is not greater than n. An internal register R in PE(i,t) is used to store this n because we need to test whether the ith component is n.

(2) A c–link in the direction from PE(i,t) to PE$(i,t+1)$ is used to transfer the ith component of a subset which is produced at the time step t.

(3) A y–link in the direction from PE(i,t) to PE$(i+1,t+1)$ is used to indicate whether the c_{out} of PE(i,t) will influence the c_{out} of PE$(i+1,t+1)$.

(4) A d–link in the direction from PE(i,t) to PE$(i+1,t+1)$ is used to transfer the c_{out} of PE(i,t) to PE$(i+1,t+1)$.

(5) An x–link in the direction from PE(i,t) to PE$(i-1,t+1)$ is used to indicate whether the c_{out} of PE(i,t) is equal to n.

(6) A register F in PE(i,t) is used to determine the y_{out} of PE(i,t).

We project Figure 1 along y–axis to obtain a linear systolic array as shown in Figure 2, where c, d, x, y are the aforementioned communication links and o_i the output terminal which is used to output the c_{out} if necessary. Each link has exactly one delay, so the delay symbol "1D" is omitted. These PEs in Figure 2 are numbered from 1 to m and referred to as PE(i) for $1 \le i \le m$. Each PE(i) is responsible for generating the ith component of any subset in *sublex*. The specification of PE(i) is depicted in Figure 3, where R and F are two internal registers.

4. THE SYSTOLIC ALGORITHM

At any time step, during the execution of our logarithm, the values transmitted on links and the contents stored in registers of PE(i) for $1 \le i \le m$ are determined as below.

(1) On the c–link; according to y_{in}, F, and x_{in} there are four cases to assign a value to c_{out}. (α): When $y_{in} = 1$, we assign $d_{in} + 1$ to c_{out}. (β): When $y_{in} = 0$ and $F = 1$, c_{out} is an empty element which is denoted by the symbol "^". (γ): When $y_{in} = 0$, $F = 0$ and $x_{in} = 1$, we assign $c_{in} + 1$ to c_{out}. (δ): When $y_{in} = 0$, $F = 0$ and $x_{in} = 0$, c_{out} has the same value as c_{in}. These

four cases to evaluate c_{out} will appear in our algorithm with procedure names: ***adding-one-din***, ***empty-element***, ***adding-one-cin***, and ***preserving-cin***, respectively.

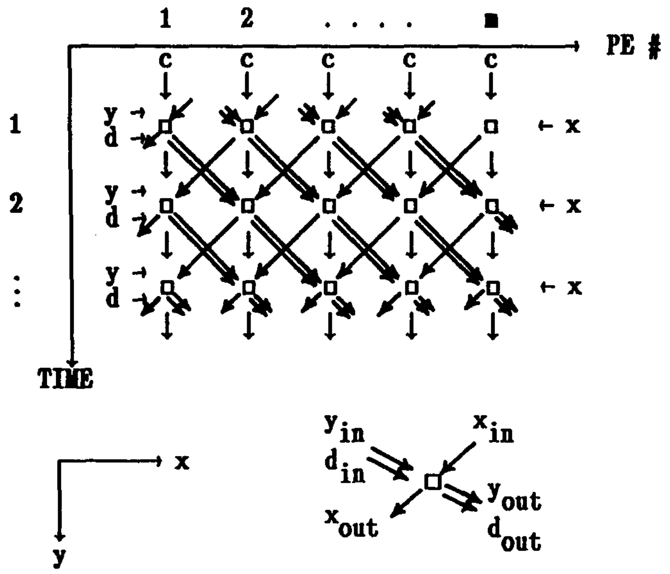

Figure 1. Directed graph for generating r–subsets.

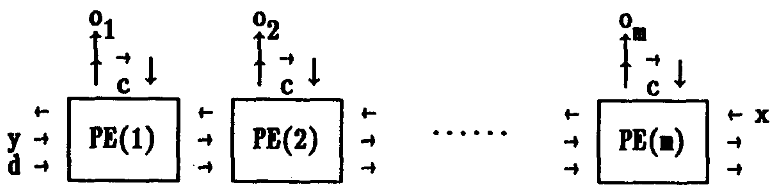

Figure 2. Computational model for generating r–subsets.

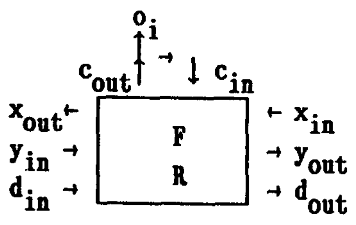

Figure 3. The specification of PE.

(2) On the d–link; d_{out} has the same value as c_{out}.

(3) On the x–link; x_{out} is determined by c_{out}. If $c_{out} = n$ then $x_{out} = 1$ else $x_{out} = 0$.

(4) On the y–link; y_{out} is determined by y_{in}, d_{in}, F, x_{in}, and c_{in}. If ($y_{in} = 1$, $d_{in} < n - 1$) or ($y_{in} = 0$, $F = 0$, $x_{in} = 1$, $c_{in} < n - 1$) then $y_{out} = 1$ else $y_{out} = 0$.

(5) The content of register F is also determined by y_{in}, d_{in}, F, x_{in}, and c_{in}. (α): If ($y_{in} = 1$, $d_{in} < n-1$) then $F = 0$; (β): If ($y_{in} = 1$, $d_{in} = n-1$) or ($y_{in} = 0$, $F = 0$, $x_{in} = 1$, $c_{in} = n-1$) then $F = 1$; (γ): Otherwise F preserves its previous value.

(6) Register R always contains the number n.

The actions of PEs to set $y_{out} = 1$ and $F = 1$ will appear in our algorithm with names *increasing-one-PE* and *decreasing-one-PE*, respectively. In fact, when PE(i) receives $y_{in} = 1$, it means that PE(i) has $c_{in} =\hat{}$ and will assign a nonempty element, $d_{in} + 1$, to its c_{out}. When PE(i) sets its $F = 1$, it means that PE(i) has $c_{out} = n$ and will assign an empty element to its c_{out} at the next time step.

Since systolic arrays are always attached to a host computer through an interface, the signal used to stop the execution of our systolic algorithm can be sent by the interface when PE(1) has sent its $x_{out} = 1$. The systolic algorithm for generating *sublex* with the name *generating-subsets(n,m)* is listed as follows. Note that the elapsed time within a time step is independent of integers r, n and m.

ALGORITHM 1. *generating-subsets(n,m)*
[Initial state]
Set $y_{in} = 1$ for PE(1) and $y_{in} = 0$ for PE(i) $2 \le i \le m$. Set $c_{in} =\hat{}$, $d_{in} = 0$, $R = n$ and $F = 1$ for PE(i) $1 \le i \le m$. Set $x_{in} = 1$ for PE(m) and $x_{in} = 0$ for PE(i), $1 \le i \le m-1$. Further, set $y_{in} = 0$, $d_{in} = 0$ for PE(1) and $x_{in} = 1$ for PE(m) at the time step $t > 1$.
[Executive state]

```
begin
   repeat /*do parallel for all PEs*/
      if $y_{in} = 1$ then adding-one-din
            else if $F = 1$ then empty-element
                  else if $x_{in} = 1$ then adding-one-cin else preserving-cin
   until $x_{out} = 1$ of PE(1) is recognized by host computer
end.
```

increasing-one-PE ≡ **begin** $y_{out}:=1$; $x_{out}:=0$; $F:=0$ **end.**

decreasing-one-PE ≡ **begin** $y_{out}:=0$; $x_{out}:=1$; $F:=1$ **end.**

adding-one-din ≡ **begin** $c_{out}:=d_{in}+1$; $d_{out}:=d_{in}+1$; **if** $d_{in} < n-1$ **then** *increasing-one*-PE **else** *decreasing-one*-PE **end.**

empty-element ≡ **begin** $c_{out}:=\hat{}$; $d_{out}:=\hat{}$; $y_{out}:=0$; $x_{out}:=0$ **end.**

adding-one-cin ≡ **begin** $c_{out}:=c_{in}+1$; $d_{out}:=c_{in}+1$; **if** $c_{in} < n-1$ **then** *increasing-one*-PE **else** *decreasing-one*-PE **end.**

preserving-cin ≡ **begin** $c_{out}:=c_{in}$; $d_{out}:=c_{in}$; $y_{out}:=0$; $x_{out}:=0$ **end.**

An example with $n = 4$, $m = 3$ for illustrating the execution of *generating-subsets(n,m)* is given in Table 1. It contains all r-subsets of $\{1,2,3,4\}$ for $1 \le r \le 3$. The values of y_{in}, y_{out}, x_{in}, x_{out}, d_{in}, d_{out}, c_{in}, c_{out}, F and R are located at their corresponding positions as shown in Figures 1 and 2, where arrows are omitted for saving the space of Table 1.

5. THE PROOF OF CORRECTNESS

In what follows, we write PE(i)[$y_{in} = 1$, $d_{in} = 2$, $c_{out} = c_{in} + 1, \ldots$]$t = t_0$ to mean that PE(i) has values $y_{in} = 1$, $d_{in} = 2$, $c_{out} = c_{in} + 1$ and so on at time step t_0. The symbol "$A \Rightarrow B$" is used to mean that statement A implies statement B. From the previous example, we observe that there are three main processes involved in *generating-subsets(m,n)* to generate *sublex*.

PROCESS-1. There exists an integer α with the time step t_α such that PE(α)[$y_{in} = 1$]$t = t_\alpha$.The procedure *adding-one-din* implies PE(α)[$c_{out} = d_{in}+1$]$t = t_\alpha$, and *preserving-cin* implies PE(i)[$c_{out} = c_{in}$]$t = t_\alpha$ for $1 \le i \le m$ and $i \ne \alpha$.If PE(α)[$d_{in} = n-1$]$t = t_\alpha$, we have PE(α)[$c_{out} = n$, $y_{out} = 0$, $F = 1$, $x_{out} = 1$]$t = t_\alpha$ by *decreasing-one*-PE. In this case, we go to PROCESS-3. If PE(α)[$d_{in} < n-1$]$t = t_\alpha$, *increasing-one*-PE implies P(α)[$c_{out} < n$, $y_{out} = 1, F = 0$]$t = t_\alpha$. In this case, if $\alpha < m$ we go to PROCESS-1 for PE($\alpha + 1$)[$y_{in} = 1$]$t = t_\alpha + 1$; otherwise, we go to PROCESS-2 because we have PE(m)[$y_{in} = 0$, $F = 0$, $x_{in} = 1$]$t = t_\alpha + 1$.

Table 1. Illustrative example for n=4, m=3.

T \ P	PE(1)	PE(2)	PE(3)	OUT
0	^ ^ ^ 1 0 1 ^ 0 4 ^	^ ^ ^ 1 0 0 ^ 0 4 ^	^ ^ ^ 1 1 0 ^ 0 4 ^	
1	1 ^ 0 0 0 1 1 0 4 1	^ ^ ^ 1 0 0 0 0 4 ^	^ ^ ^ 1 1 0 0 0 4 ^	1
2	1 1 0 0 0 0 0 0 4 1	2 ^ 0 0 0 1 1 1 4 2	^ ^ 0 1 1 0 0 ^ 4 ^	12
3	1 1 0 0 0 0 0 0 4 1	2 2 0 0 0 0 0 1 4 2	3 ^ 0 0 1 1 1 2 4 3	123
4	1 1 0 0 0 0 0 0 4 1	2 2 0 0 0 0 0 1 4 2	4 3 1 1 1 0 1 2 4 4	124
5	1 1 0 0 0 0 0 0 4 1	3 2 0 0 1 0 1 1 4 3	^ 4 0 1 1 0 0 2 4 ^	13
6	1 1 0 0 0 0 0 0 4 1	3 3 0 0 0 0 0 1 4 3	4 ^ 1 1 1 1 1 3 4 4	134
7	1 1 0 0 0 0 0 0 4 1	4 3 1 1 1 0 0 1 4 4	^ 4 0 1 1 0 0 3 4 ^	14
8	2 1 0 0 1 0 1 0 4 2	^ 4 0 1 0 0 0 1 4 ^	^ ^ 0 1 1 0 0 4 4 ^	2
9	2 2 0 0 0 0 0 0 4 2	3 ^ 0 0 0 1 1 2 4 3	^ ^ 0 1 1 0 0 ^ 4 ^	23
10	2 2 0 0 0 0 0 0 4 2	3 3 0 0 0 0 0 2 4 3	4 ^ 1 1 1 1 1 3 4 4	234
11	2 2 0 0 0 0 0 0 4 2	4 3 1 1 1 0 0 2 4 4	^ 4 0 1 1 0 0 3 4 ^	24
12	3 2 0 0 1 0 1 0 4 3	^ 4 0 1 0 0 0 0 4 ^	^ ^ 0 1 1 0 0 4 4 ^	3
13	3 3 0 0 0 0 0 0 4 3	4 ^ 1 1 0 1 0 3 4 4	^ ^ 0 1 1 0 0 ^ 4 ^	34
14	4 3 1 1 1 0 0 0 4 4	^ 4 0 1 0 0 0 3 4 ^	^ ^ 0 1 1 0 0 4 4 ^	4

PROCESS–2. There exists an integer β with the time step t_β such that PE(β)[$y_{in} = 0$, $F = 0$, $x_{in} = 1$]$t = t_\beta$. The procedure *adding-one-cin* implies PE(β)[$c_{out} = c_{in} + 1$]$t = t_\beta$. If PE(β)[$c_{in} = n - 1$]$t = t_\beta$, we have PE(β)[$c_{out} = n$, $F = 1$, $x_{out} = 1$]$t = t_\beta$ by *decreasing-one-*PE. In this case, we go to PROCESS–3. If PE(β)[$c_{in} < n - 1$]$t = t_\beta$, *increasing-one*-PE implies PE(β) [$c_{out} = c_{in} + 1 < n$, $F = 0$, $y_{out} = 1$, $x_{out} = 0$]$t = t_\beta$. In this case, if $\beta = m$ we go to PROCESS–2 for PE(m)[$y_{in} = 0$, $F = 0$, $x_{in} = 1$]$t = t_\beta + 1$; otherwise, we go to PROCESS–1 because of PE($\beta + 1$)[$y_{in} = 1$]$t = t_\beta + 1$.

PROCESS–3. There exists an integer γ with the time step t_γ such that PE(γ)[$c_{out} = n$]$t = t_\gamma$. The procedure *decreasing-one*-PE implies PE(γ)[$y_{out} = 0$, $F = 1$, $x_{out} = 1$]$t = t_\gamma$. This implies that PE(γ) will generate its c_{out} =^ at the next time step $t_\gamma + 1$ from the execution of *empty-element*. If $\gamma = 1$ then the execution of *generating-subsets(n,m)* stops at the time step $t_\gamma + 1$ because PE(1) sends the message $x_{out} = 1$ at the time step t_γ and this message is recognized by host computer at the time step $t_\gamma + 1$. If $\gamma > 1$ we go to PROCESS–2 because of PE($\gamma - 1$)[$y_{in} = 0$, $F = 0$, $x_{in} = 1$]$t = t_\gamma + 1$.

Now, we want to show that *generating-subsets(n,m)* is correct for generating all subsets in *sublex*, which we state as a theorem.

THEOREM. The algorithm *generating-subsets(n,m)* is correct to generate all of the r–subsets of $\{1, 2, \ldots, n\}$ for $1 \leq r \leq m \leq n$ in lexicographical order.

PROOF. We prove this theorem by induction on time step t.

[Basis]. For $t = 1$, at the beginning of the execution of *generating-subsets(n,m)*, we have PE(1)$[y_{in} = 1,\ d_{in} = 0]t = 1$ and PE(i)$[y_{in} = 0,\ F = 1]t = 1$ for $2 \le i \le m$. PROCESS-1 shows that the first subset is $\{1\}$ for PE(1) performing ***adding-one-din*** and the other PEs performing ***empty-element***.

[Assumption]. Suppose that the kth subset $A = \{a_1, a_2, \ldots, a_\alpha\}$ in ***sublex*** is generated in the time step $t = k$.

[Induction]. For $t = k+1$, let $B = \{b_1, b_2, \ldots, b_\beta\}$ be the subset generated by the algorithm *generating-subsets(n,m)* at the time step $k+1$. We want to claim that B belongs to ***sublex*** with the index $k+1$. We divide the proof into four cases according to the values of α and a_α.

Case (1): $\alpha = m$ and $a_m = n$.

$$PE(i)[c_{out} = a_i \ne n]t = k,\ 1 \le i \le m-1;\ PE(m)[c_{out} = n]t = k.$$
$$\Rightarrow PE(i)[y_{out} = 0,\ F = 0,\ x_{out} = 0]t = k,\ 1 \le i \le m-1;$$
$$PE(m)[y_{out} = 0,\ F = 1,\ x_{out} = 1]t = k.$$
$$\Rightarrow PE(i)[y_{in} = 0,\ F = 0,\ x_{in} = 0,\ c_{in} = a_i,\ c_{out} = c_{in} = a_i]t = k+1,\ 1 \le i \le m-2;$$
$$PE(m-1)[y_{in} = 0,\ F = 0,\ x_{in} = 1,\ c_{in} = a_{m-1},\ c_{out} = c_{in} + 1 = a_{m-1} + 1]t = k+1;$$
$$PE(m)[y_{in} = 0,\ F = 1,\ c_{out} = \hat{}\ ]t = k+1.$$

In this case, we have $\beta = m - 1$ and $B = \{a_1, a_2, \ldots, a_{m-2}, a_{m-1} + 1\}$.

Case (2): $\alpha = m$ and $a_m \ne n$.

$$PE(i)[c_{out} = a_i \ne n]t = k,\ 1 \le i \le m.$$
$$\Rightarrow PE(i)[y_{out} = 0,\ F = 0,\ x_{out} = 0]t = k,\ 1 \le i \le m-1;$$
$$PE(m)[y_{out} = 1,\ F = 0,\ x_{out} = 0]t = k.$$
$$\Rightarrow PE(i)[y_{in} = 0,\ F = 0,\ x_{in} = 0,\ c_{in} = a_i,\ c_{out} = c_{in} = a_i]t = k+1,\ 1 \le i \le m-1;$$
$$PE(m)[y_{in} = 0,\ F = 0,\ x_{in} = 1,\ c_{in} = a_m,\ c_{out} = c_{in} + 1 = a_m + 1]t = k+1.$$

In this case, we have $\beta = m$ and $B = \{a_1, a_2, \ldots, a_{m-1}, a_m + 1\}$.

Case (3): $\alpha \ne m$ and $a_\alpha = n$.

$$PE(i)[c_{out} = a_i \ne n]t = k,\ 1 \le i \le \alpha - 1;\ PE(\alpha)[c_{out} = a_\alpha = n]t = k;$$
$$PE(i)[c_{out} = \hat{}\ ]t = k,\ \alpha + 1 \le i \le m.$$
$$\Rightarrow PE(i)[y_{out} = 0,\ F = 0,\ x_{out} = 0]t = k,\ 1 \le i \le \alpha - 1;$$
$$PE(\alpha)[y_{out} = 0,\ F = 1,\ x_{out} = 1]t = k;$$
$$PE(i)[y_{out} = 0,\ F = 1,\ x_{out} = 0]t = k,\ \alpha + 1 \le i \le m.$$
$$\Rightarrow PE(i)[y_{in} = 0,\ F = 0,\ x_{in} = 0,\ c_{out} = a_i,\ c_{in} = c_{out} = a_i]t = k+1,\ 1 \le i \le \alpha - 2;$$
$$PE(\alpha - 1)[y_{in} = 0,\ F = 0,\ x_{in} = 1,\ c_{in} = a_{\alpha-1},\ c_{out} = c_{in} + 1 = a_{\alpha-1} + 1]t = k+1;$$
$$PE(\alpha)[y_{in} = 0,\ F = 1,\ c_{out} = \hat{}\ ]t = k+1;$$
$$PE(i)[y_{in} = 0,\ F = 1,\ c_{out} = \hat{}\ ]t = k+1,\ \alpha + 1 \le i \le m.$$

In this case, we have $\beta = \alpha - 1$ and $B = \{a_1, a_2, \ldots, a_{\alpha-2}, a_{\alpha-1} + 1\}$.

Case (4): $\alpha \ne m$ and $a_\alpha \ne n$.

$$PE(i)[c_{out} = a_i \ne n]t = k,\ 1 \le i \le \alpha;\ PE(i)[c_{out} = \hat{}\ ]t = k,\ \alpha + 1 \le i \le m.$$
$$\Rightarrow PE(i)[y_{out} = 0,\ F = 0,\ x_{out} = 0]t = k,\ 1 \le i \le \alpha - 1;$$
$$PE(\alpha)[y_{out} = 1,\ F = 0,\ x_{out} = 0,\ d_{out} = a_\alpha]t = k;$$
$$PE(i)[y_{out} = 0,\ F = 1,\ x_{out} = 0]t = k,\ \alpha + 1 \le i \le m.$$
$$\Rightarrow PE(i)[y_{in} = 0,\ F = 0,\ x_{in} = 0,\ c_{in} = a_i,\ c_{out} = c_{in} = a_i]t = k+1, 1 \le i \le \alpha;$$
$$PE(\alpha + 1)[y_{in} = 1,\ d_{in} = a_\alpha,\ c_{out} = d_{in} + 1 = a_\alpha + 1]t = k+1;$$
$$PE(i)[y_{in} = 0,\ F = 1,\ c_{out} = \hat{}\ ]t = k+1,\ \alpha + 2 \le i \le m.$$

In this case, we have $\beta = \alpha + 1$ and $B = \{a_1, a_2, \ldots, a_\alpha, a_\alpha + 1\}$.

Therefore the subset B generated by *generating-subsets(n,m)* has the index $k+1$ in ***sublex***. This completes the proof of the THEOREM. ∎

6. CONCLUSIONS

Although the rapid advance of VLSI technology has made the construction of a parallel computer more feasible than ever before, it is still difficult to construct most parallel computers due to their complex structure. In this paper, we present a new parallel algorithm to generate subsets on a linear systolic array. The design of a systolic array and an algorithm is considered in detail. Since all PEs presented in our systolic array have identical stucture and perform the same program, this systolic array is very suitable for VLSI implementation. Note that we can modify our systolic array so that it gives the subsets suceeding to a given subset Y. This is done by loading adequate values from the components of Y into the initial state of the algorithm *generating-subsets(n,m)*. Moreover, for $1 \leq i \leq m$, at the time step that PE(1) has just assigned $n - i + 1$ to its c_{out} (if $m < n$, it is done by *adding-one-cin*; if $m = n$, it is also done by *adding-one-cin* except at the time step $t = 1$ which is done by *adding-one-din*), PE(1) sends a special stop-signal to PE(i). Then after PE(i) receives this signal, PE(i) will stop its executive state at the next time step.

We hope that our design consideration can be used to design new systolic algorithms for other problems in the fields of numerical methods, computational geometry and graph theory. For example, the solution of simultaneous linear equations of Gauss-Jordan elimination with pivoting, the generation of $m!$ permutations, the convex hull and planarity testing problems etc. Furthermore, we are interested in investigating those systolic arrays in which the storage of a PE and the elapsed time of a time step are independent of the problem size.

REFERENCES

1. S. G. Akl, Adaptive and Optimal Parallel Algorithms for Enumerating Permutations and Combinations, *Computer Journal* **30**, 433–436 (1987).
2. B. Chan and S. G. Akl, Generating Combinations in Parallel, *BIT* **26**, 2–6 (1986).
3. G. H. Chen and M. S. Chern, Parallel Generation of Permutations and Combinations, *BIT* 26, 277–282 (1986).
4. H. T. Kung, Why Systolic Architecture?, *IEEE Computer* **15**, 37–46 (1982).
5. I. Semba, An Efficient Algorithm for Generating All k-subsets ($1 \leq k \leq m \leq n$) of the Set $\{1, 2, \ldots, n\}$ in Lexicographical Order, *Journal of Algorithms* **5**, 281–283 (1984).
6. I. Stojmenovic and M. Miyakawa, Applications of a Subset-Generating Algorithm to Base Enumeration, Knapsack and Minimal Covering Problems, *Computer Journal* **32**, 65–70 (1988).