論文93-30A-3-7

Polyadic-nonserial 동적 프로그래밍을 위한 문제크기에 독립적인 시스토릭 어레이의 설계

(Design of Problem Size-Independent Systolic Array for Polyadic-Nonserial Dynamic Programming)

禹 鐘 鎬*, 申 東 錫*, 鄭 信 一*, 權 大 亨*

(Chong Ho Woo, Dong Suk Shin, Shin Il Jeong and Dae Hyung Kwon)

要 約

시스토릭 어레이의 실제 응용에 있어서 문제의 크기(n)가 어레이의 크기(M)보다 큰 경우가 흔히 있다. 이 경우에는 문제가 처리되기 전에 사용할 어레이 크기에 맞게 분할 되어야 한다. 본 논문에서는 동적 프로그래밍의 문제분할 방법과 이에 적합한 이차원 시스토릭 어레이를 제안하였다. 분할된 문제를 처리하기 위하여 두가지 형태의 어레이 를 사용하였으며, 중간 계산결과를 저장하고 정확한 위치와 시간에 데이타를 어레이로 다시 입력하기 위하여 큐를 설계하였다. 어레이에서 필요한 처리요소의 수는 M(3M+1)/2, 큐의 수는 4M, 처리 시간은 2(M+1)+(n+10M+3)(n/M)(n/M-1)/6 이다.

Abstract

In many practical applications of systolic array, it is common that the problem size(n) is larger than the array size(M). In this case, the problem has to be partitioned into block to fit into the array before it is processed. This paper presents a problem partition method for dynamic programming and 2-dimensional systolic array suitable for it. Designed array has two types of array configur- ation for processing the partitioned problem. The queue is designed for storing and recirculating the intermediate results in the correct location and time. The number of processing elements and queues required are M(3M+1)/2, 4M respectively. The total processing time is 2(M+1)+(n+10M+3)(n/M)(n/M-1)/6.

Ⅰ. 서 론

동적 프로그래밍은 Bellman [1] 에 의해서 처음 제안되었으며, Context-free 언어의 인식, 행렬곱의 최적화, 이진트리 탐색의 최적화 등 일련의 판단을 최적화하는 기법으로 $O(n^3)$의 계산 복잡도를 갖는다. [2] 이것을 이차원 시스토릭 어레이로 구현하면 계산 복잡도가 O(n)으로 되어 실시간 처리가 가능하다. [3,4,5]

시스토릭 어레이에서 문제의 분할은 처리할 문제의 크기가 그 문제를 처리하기 위한 어레이의 크기보다 클때 요구된다. 시스토릭 어레이의 실제 응용에서는 문제의 크기가 어레이의 크기보다 큰 경우가 흔히 발생된다. [6] 그러므로 문제내에 주어지는 데이타는 어

*正會員, 釜山水産大學校 電子工學科
(Dept. of Elec. Eng., Nat'l Pusan Fisheries Univ.)

(※이 논문은 1991년도 교육부지원 한국학술진흥재단의 지방대학육성과제 학술연구조성비에 의하여 연구되었음.)
接受日字: 1992年 10月 23日

레이내에서 적합한 데이타 블럭(block)으로 분할되어 처리된다. Moldovan [7] 등은 알고리듬의 인덱스 집합을 어레이의 크기로 분할하고, 분할된 블럭의 의존관계에 따라 순차적으로 처리하는 방법을 제안하였으나 어레이에서 데이타의 흐름이 같은 속도가 될 때에만 적용가능하다. 또한 처리 과정에서 어레이로 부터 출력되는 중간 계산결과를 저장하기 위한 큐의 설계는 다루지 않았다. Navarro [8] 등은 행렬 곱과 LU-decomposition등의 행렬 문제들을 제한된 크기의 밴드(band)행렬로 전체 문제를 부문제(subproblem)로 변환하여 전체 문제를 처리하는 방법을 제안하였다. 그러나 이 방법들은 동적 프로그래밍 문제에는 적용될 수 없다. 왜냐하면 동적 프로그래밍에서는 인덱스 공간에서 각 계산점(computation point)의 계산은 어레이의 위치와 시간에 따라 다르며, 데이타의 흐름속도도 일정하지 않고, 또한 이차원 어레이로 구성하였을 경우 어레이의 각 처리요소들은 서로 다른 제어 변수에 의해 동작되기 때문이다.

본 논문에서는 동적 프로그래밍 처리를 위한 시스토릭 어레이에서 문제의 분할방법을 제안하고 평가하였다. 먼저 공간분할을 위하여 LPGS(Locally Parallel Globally Sequencial) 방법을 사용하여 문제의 블럭들을 어레이로 사상(mapping)하였고, 블럭간의 데이타 의존관계에 따라서 블럭의 처리순서를 결정하여, 각 계산점들을 처리요소로 시간사상하였으며, 어레이에서 출력되는 중간계산결과를 저장하기 위한 큐(queue)를 설계하였다.

Ⅱ. 동적 프로그래밍을 위한 이차원 시스토릭 어레이

동적 프로그래밍 처리를 위한 이차원 시스토릭 알고리듬은 그림 1과 같다. [9] 여기서 x,y는 처리요소의 위치를 나타내는 좌표이고 z=x-y이다.

시간 t가 증가함에 따라 z가 증가되는 방향으로 제어가 이동되므로 제어변수들의 통신패스방향으로 [x,y] [1,0] 혹은 [0,-1] 을 사용할 수 있다. 본 논문에서는 제어변수들의 통신패스를 [0,-1] 방향으로 선택한다.

알고리듬에서 조건식 [3z/2] =t와 관련되는 제어변수를 $m2_{x,y}$로 두어 세단위시간 동안 두개의 처리요소를 지나도록 하고, t=2z와 관련되는 제어 변수를 $m3_{x,y}$로 두어 두 단위 시간에 이웃한 처리요소로 전달되게 한다. even(z)의 조건을 인식하기 위하여 제어변수를 $m1_{x,y}$로 두어 한 단위시간 마다 이웃한 처리요소로 이동시킨다. 그리고, $m2_{x,y}$와 $m3_{x,y}$의 정확한 이동을 위하여 한 비트의 기억회로 $f1_{x,y}$와 f2x,y를 사용한다.

처리요소는 $m2_{x,y}$가 '1' 상태일 때 t= [3z/2] 조건에 따라 동작되고, $m3_{x,y}$가 '1' 상태로 되기전까지 [3z/2] ⟨t⟨2z인 조건에서 동작된다. 그리고 $m3_{x,y}$가 '1' 상태일 때 t=2z의 조건에 따라 동작된다. 또한 한 비트의 기억회로 $fg_{x,y}$를 두어 [3z/2] ⟨t⟨2z인 조건에서의 동작을 제어한다. $fg_{x,y}$는 $m2_{x,y}$가 '1' 상태로 된 직후 '1'로 세트되고, $m3_{x,y}$가 '1' 상태로 되기 직전에 클리어 된다. 위와 같은 동작을 하기 위하여 (x, y)위치에서 각 제어변수들의 제어논리는 다음과 같다.

```
begin
  for all 1≤y≤x≤n do in parallel
   if z=1 then
     cx,y,t=C0,
   else
     for all [3z/2] ≤t ≤2z do in parallel
       ax,y,t=ax-1,y,t-1
       ex,y,t=ex,y+1,t-1
       bx,y,t=bx,y+1,t-2
       dx,y,t=dx-1,y,t-2
       if t=2z then
         ax,y,t=cx,y,t
         ex,y,t=cx,y,t
         cx,y,t=cx,y,t-1
       else
        if t=⌈3z/2⌉ then
          if even(z) then
            bx,y,t=ex,y+1,t-1
            dx,y,t=ax-1,y,t-1
          end if
          cx,y,t=g(h(ax,y,t bx,y,t),h(dx,y,t ex,y,t))
        else
          cx,y,t=g(cx,y,t-1, h(ax,y,t bx,y,t), h(dx,y,t ex,y,t))
        end if
       end if
     all for
   end if
  all for
end
```

그림 1. 동적 프로그래밍을 위한 이차원 시스토릭 알고리듬

Fig. 1. Systolic algorithm for dynamic programming.

$$m1_{x,y,t+1} = \overline{m1_{x,y+1,t}}$$

$$m2_{x,y,t+1} = \{(m2_{x,y+1,t} \wedge m1_{x,y+1,t}) \vee f1_{x,y,t}\} \wedge \overline{m2_{x,y,t}}$$

$$m3_{x,y,t+1} = f2_{x,y,t} \wedge \overline{m3_{x,y,t}}$$

$$f1_{x,y,t+1} = m2_{x,y+1,t} \wedge m1_{x,y+1,t} \wedge \overline{f1_{x,y,t}}$$

$$f2_{x,y,t+1} = m3_{x,y+1,t} \wedge \overline{f2_{x,y,t}}$$

$$fg_{x,y,t+1} = (m2_{x,y,t} \vee fg_{x,y,t}) \wedge \overline{f2_{x,y,t}}$$

여기서, ∧와 ∨는 각각 논리적(AND)과 논리합(OR)을 나타내며 ¯는 논리부정(NOT)이다. 또한 a, e는 한 단위시간전의 이웃 처리요소의 값을 이용하므로 하나의 레지스터를 두어 이웃 처리요소로 옮겨가도록 하고, b,d는 두 단위시간 전의 값을 이용하므로 두개의 레지스터를 두어 파이프라인 형태로 이웃 처리요소로 옮겨 가도록 한다. 그리고 g, h_1 및 h_2는 각 함수를 이행하는 조합논리로 구성되며, a를 비롯

한 다른 요소들은 모두 레지스터 및 플립플롭으로 구성된다. 동적 프로그래밍을 위한 이차원 시스토릭 어레이는 삼각형의 구성형태를 가지며, 그 처리요소는 그림 2와 같다. [9]

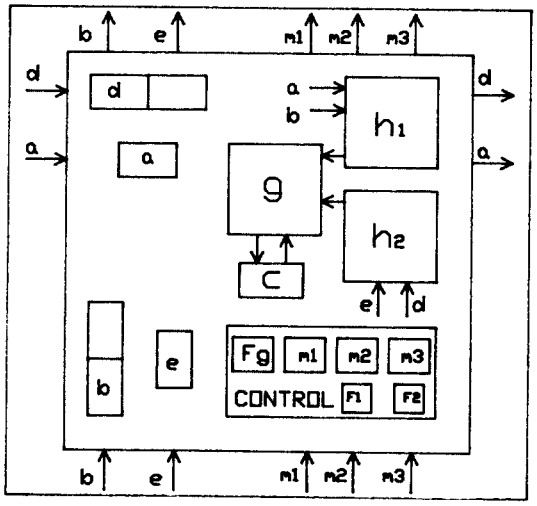


그림 2. 동적 프로그래밍을 위한 이차원 시스토릭 어레이의 처리요소

Fig. 2. The processing element of 2-dimensional systolic array for dynamic programming.

Ⅲ. 문제의 분할

1. 문제의 분할 및 공간사상

문제를 분할하는 방식은 크게 LSGP(Locally Sequential Grobally Parallel)와 LPGS (Locally Parallel Grobally Sequential) 로 분류된다. [6] 본 논문에서는 LPGS 방식으로 데이타와 제어 신호의 흐름과 어레이의 구성 형태를 고려하여 그림 3과 같은 형태로 문제를 분할하였다. 분할된 문제를 효과적으로 처리하기 위하여 어레이는 그림 4와 같이 두가지 형태의 어레이로 구성된다. 그림 4에서 어레이의 처리요소들은 그림 2의 처리요소를 그대로 사용한다. LPGS 방식으로 문제를 분할한 각 블럭을 어레이로 사상하고, 각 블럭의 처리 순서를 블럭간의 의존 관계에 따라 그림 3과 같이 정한다.

문제의 인덱스 집합을 공간적으로 분할하기 위하여 이차원 어레이 설계 과정에서 구한 식(1)의 공간 변환 S를 이용한다. [9]

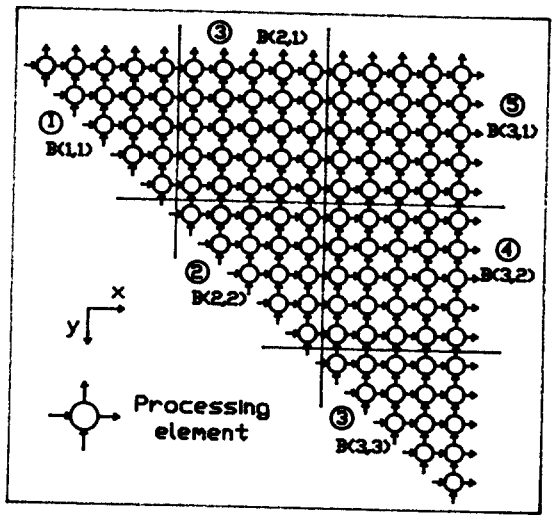


그림 3. 동적 프로그래밍 문제의 분할(n=16, M=5)

Fig. 3. The partition of problem for dynamic programming(n=16, M=5).

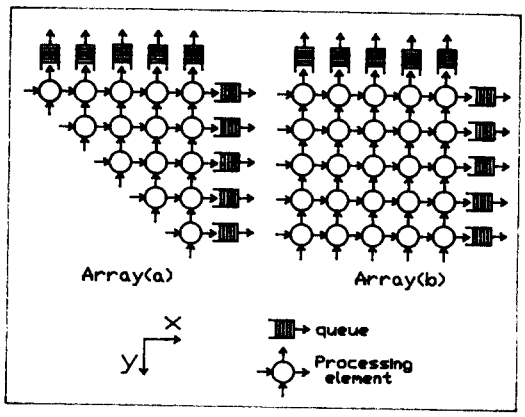


그림 4. 분할된 문제를 처리하기 위한 어레이 구성 (M=5)

Fig. 4. The array configuration to process partitioned problem(M=5).

$$S = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} 0\ 0\ 1 \\ 0\ 1\ 0 \end{bmatrix} \quad (1)$$

식(1)은 S_1 [0 0 1] 과 S_2 [0 1 0] 를 어레이의 크기와 같은 블럭으로 분할하고, 이들 블럭을 B(b_1,

b_2)로 나타낸다. 식(1)으로 부터 문제의 인덱스 집합 J (k,i,j)에서각 계산점이 속하는 블럭의 위치 B(b_1, b_2)가 식(2), (3)에서 구해지고, 각 계산점이 실제 어레이로 사상되는 처리요소의 위치는 식(4), (5)와 같다.

$$b_1 = \lfloor (S_1 \cdot \bar{j} - 1) / M \rfloor \quad (2)$$

$$b_2 = \lfloor (S_2 \cdot \bar{j}) / M \rfloor \quad (3)$$

$$\hat{x} = (S_1 \cdot \bar{j} - 2) \bmod M \quad (4)$$

$$\hat{y} = (S_2 \cdot \bar{j} - 1) \bmod M \quad (5)$$

여기서, j는 알고리듬의 인덱스 포인트이고, x, y는 인덱스 포인트가 어레이의 처리요소에 할당된 위치를 나타내며 M은 어레이의 크기이다. 그리고, 각각의 블럭을 처리하는 어레이의 형태는 식(2), (3)에서 구한 b_1, b_2에 의해서 결정된다. 즉, 각 블럭에서 b_1=b_2면 그 블럭은 그림 3의 (a)형태의 어레이에서 처리되고, $b_1 \neq b_2$면 (b)형태의 어레이에서 처리된다.

2. 시간사상

분할된 블럭 B(b_1,b_2)에서 처리시간은 b_1=b_2인 경우와 $b_1 \neq b_2$일 경우에 대해서 각각 구한다.

1) b_1=b_2인 경우

이러한 형태의 블럭에서의 처리시간은 분할되기전 어레이에서의 처리시간과 같다. 왜냐하면 어레이의 형태가 원래의 어레이 형태이고 다른 출력 데이타에 의존되지 않고 단지 초기값에 의해서 계산되기 때문이다. 따라서 블럭 B(b1,b2)의 처리시간은 식(6)과 같다 [9]

$$t_a = \left\lceil (\max(\Pi \cdot (\hat{j}_1 - \hat{j}_2)) + 1) / \min(\Pi \cdot d_i) \right\rceil \quad (6)$$
$$= [1 - 2 \;\; 1][M + 1 - 1,\; 0,\; M + 1 - 2]^T + 1 = 2M$$

여기서, j1, j2= [k i j] $^T \in$B(b_1,b_2)이다.

2) b1≠b2인 경우

블럭 B(b_1,b_2)의 처리시간은 어레이에서 출력되는 모든 제어변수들의 변화를 고려하여야 한다. 제어 변수 m1$_{x,y}$는 1 단위시간 후에 이웃한 처리요소로 이동되며, 각 블럭의 좌측 상단 처리요소에서 가장 먼저 출력되어 좌측하단 처리요소로 입력된다. 이때 블럭B (b_1,b_2)의 좌측하단 처리요소로 m1$_{x,y}$가 도달하기 위해서 M(b_1-b_2-1)+1 단위시간이 요구된다. 블럭의 처리시간은 크기 b_1M+1인 문제를 처리하기위해 요구되는 시간에서 B(b_1,b_2)의 좌측하단 처리요소로 제어변수가 이동하는데 소요되는 시간을 감산한 것이다. 따라서 $b_1 \neq b_2$인 블럭의 처리시간은 다음과 같다.

$$t_b = \left\lceil (\max(\Pi \cdot (\hat{j}_1 - \hat{j}_2)) + 1) / \min(\Pi \cdot d_i) \right\rceil - (M(b_1 - b_2 - 1) + 1)$$
$$= [1 - 2 \;\; 1] \cdot \left(\begin{bmatrix} b_1 M + 1 \\ (b_2 - 1)M + 1 \\ b_1 M + 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \right) + 1 - (M(b_1 - b_2 - 1) + 1)$$
$$= [1 - 2 \;\; 1][b_1 M,\; (b_2 - 1)M,\; b_1 M - 1]^T + 1 - (M(b_1 - b_2 - 1) + 1)$$
$$= M(b_1 - b_2 + 3) - 1) \quad (7)$$

여기서 j_1= [k i j] $^T \in$B(b_1,b_2)이고 j_2= [k i j] ^{T} $\in$J 이다.

그리고, 우측상단 처리요소 내의 a와 e 및 b와 d 레지스터의 내용을 큐에 저장하기 위해서 2 단위시간이 더 소요되기 때문에 식(6)과 (7)은 각각 식(8), (9)와 같다.

$$t_a = 2(M\ 1) \quad (8)$$

$$t_b = M(b_1\ b_2\ 3)\ 1 \quad (9)$$

각 블럭의 계산시간을 이용하여, 문제의 인덱스 공간의 계산점들의 계산시각은 B(b_1,b_2) 이전에 처리된 모든 블럭의 계산시간의 총합에 B(b_1,b_2)에 존재하는 각 계산점의 계산시각을 합해서 구한다. 그리고 (b)형태의 어레이는 (a)형태의 어레이로 사상되는 두번째 블럭의 처리가 시작된지 2단위시간 후에 동작하므로 B(b_1,b_2)에 존재하는 계산점의 계산시각은 b_1=b_2 인 경우 식(10)과 같고 $b_1 \neq b_2$인 경우 식(11)과 같다.

$$t = \Pi \cdot \bar{j} + \sum_{1 \le i \le b_1} 2(M + 1) \quad (10)$$

$$t = \Pi \cdot \bar{j} - \{M(b_1 - b_2 - 1) + 1\} + \sum_{2 \le i \le b_1 - 1} \left[\sum_{1 \le j \le i - 1} \{M(i - j + 3) + 1\} \right]$$
$$+ \sum_{b_2 + 1 \le k \le b_1 - 1} \{M(b_1 - k + 3) + 1\} + 2(M + 2) \quad (11)$$

여기서, j [k,i,j] $^T \in$B(b_1,b_2)이다.

3. 큐의 설계

문제를 분할하여 처리할 경우 중간계산결과를 저장

하고 정확한 위치와 시간에 데이타를 어레이로 재입력하기 위하여 큐가 요구된다. 어레이로부터 출력되는 데이타와 제어신호를 언제 어느 큐에 저장하고, 어떻게 제어하여 정확한 시간과 위치로 큐의 내용을 출력시키는 문제가 큐의 설계에서 고려되어야 한다. 그리고 어레이에서 시간이 증가됨에 따라 어레이의 동작밴드의 범위가 증가되기 때문에 중간계산결과를 저장하는 시간과 저장해야 할 데이타와 제어신호의 수는 블럭의 위치 b1과 b2에 의존된다.

어레이의 크기가 M인 경우 그림 4의 (a), (b)형태의 어레이에서 각각 2M개의 큐가 요구되므로 필요한 전체 큐의 수는 4M개이다. 그림 4의 (a)형태의 어레이에 있는 큐는 해당 블럭의 처리가 시작된 후 1단위 시간 후에 어레이로부터 출력되는 데이타와 제어신호를 저장한다. 그러나 (b)형태의 어레이는 M단위시간 후에 좌측상단 처리요소에서 데이타와 제어신호들이 출력되기 때문에 M+1시각에 데이타와 제어신호를 저장한다. 따라서 식(8)과 (9)로부터 각 블럭이 처리될 때 큐로의 데이타 입력시간 q_i는 식(12)와 같다.

$$q_i = M \cdot (b_1 - b_2 + 3) + 1 - M$$
$$= M(b_1 - b_2 + 2) + 1 \qquad (12)$$

큐에 저장된 데이타는 그림 3의 (b)형태의 어레이에서 이용되며, $b_2 = b_1 - 1$이면 (a)형태의 어레이에 있는 큐의 내용이 입력되고, $b_2 < b_1 - 1$일 경우에는 (b)형태의 어레이에 있는 큐에 저장된 값이 입력된다. 그리고, 각 어레이는 제어신호에 의해 동작되기 때문에 $B(b_1, b_2)$가 처리되는 경우에는 $B(b_1 - 1, b_2)$에서 출력되는 데이타 및 $B(b_1, b_2 + 1)$에서 출력되는 데이타와 제어신호들이 필요하게 된다. 그러므로 블럭 $B(b_1, b_2)$가 처리될 때 큐로부터 어레이로의 데이타 출력시간 q_0는 $B(b_1 - 1, b_2)$ 또는 $B(b_1, b_2 + 1)$에서 큐에 데이타를 저장하는 시간과 같으므로 식(12)로부터 다음 식과 같다.

$$q_o = M(b_1 - b_2 + 1) + 1 \qquad (13)$$

위의 결과로부터 큐를 설계할 경우, 큐에 데이타를 저장하고 큐에 저장된 데이타를 어레이로 출력하는 정확한 시간을 구하기 위하여 각 어레이 형태에 카운터가 필요하다. 그림 4의 어레이는 상호 독립적으로 동작하고 (a)형태의 어레이에서 출력되는 중간계산결과를 저장하고 있는 큐의 내용은 (b)형태의 어레이로 입력된다. 그래서 큐로의 입력 및 큐에서 어레이로의 출력이 동시에 발생하는 경우와 큐로의 입력만 발생하는 경우가 있다. 그러므로 (a)형태의 어레이에서 출력되는 중간계산결과를 저장하기 위한 큐는 그 길이가 문제의 크기에 따라 불규칙적으로 변하게 된다. 이를 해결하기 위하여 전체 처리시간에 영향을 미치지 않는 범위에서 (a)형태의 어레이로 사상되는 블럭의 처리 시작시간을 지연시킨다. 이 때의 지연시간 t_d는 $B(b_1 - 1, b_2 - 2)$의 처리가 완료된 후부터 $B(b_1 - 1, 1)$의 처리가 완료될때까지의 시간으로 식(14)와 같다.

$$t = \sum_{1 \le i \le b_1 - 2} \{M(b_1 - i + 2) + 1\} - 2(M + 1)) \qquad (14)$$

그림 4의 어레이 형태에서 y-축 방향으로 출력되는 데이타와 제어신호를 저장하기 위해서 필요한 큐의 최대 길이는 블럭의 계산순서에 따라 큐에 저장되어 있는 블럭 $B(b_1, b_2)$의 중간계산결과는 블럭 $B(b_1, b_2 - 1)$의 처리를 위해 곧바로 어레이로 입력되기 때문에 간단하게 결정된다. 그러나 x-축 방향으로 출력되는 중간계산결과를 저장하는 큐의 길이는 간단하게 결정될 수 없다.

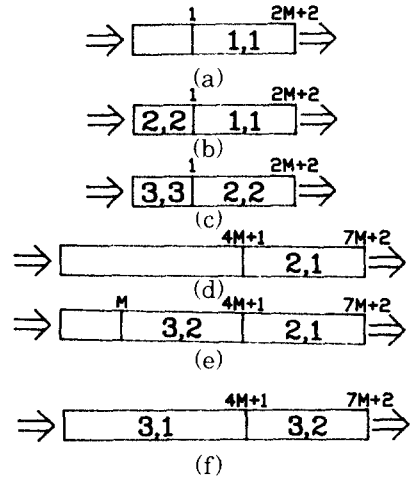


그림 5. 블럭의 중간계산결과를 저장하는 큐의 스냅-샷
(a)블럭 B(1,1), (b)블럭 B(2,2), (c)블럭 B(3,3), (d)블럭 B(2,1), (e)블럭 B(3,2), (f)블럭 B(3,1)

Fig. 5. The snap-shuts of the queue which stored the intermediate results of the block(n=4M).
(a) block B(1,1), (b) block B(2,2), (c) block B(3,3), (d) block B(2,1), (e) block B(3,2), (f) block B(3,1).

그림 5는 (a) 및 (b)형태의 어레이에서 각 블럭의 처리후 큐의 내용을 개략적으로 나타낸 것이다. 그림 5의 (a)는 블럭 B(1,1)이 처리된후 큐의 내용을 나타내며 (b)는 (b)형태의 어레이에서 블럭 B(2,1)의 처리가 시작될 때 블럭 B(1,1) 및 B(2,2)의 처리과정에서 출력되는 중간계산결과가 저장된 형태를 나타낸다. (c)는 블럭 B(3,2)의 처리가 시작될 때의 큐의 내용을 나타낸다. b1 b2인 블럭을 처리할 때에는 어레이에서 출력되는 데이타와 제어신호를 저장해야 하는 시간이 블럭 수에 관계없이 일정하기 때문에 큐의 길이는 고정된다.

그림 5의 (d), (e), (f)는 각각 블럭 B(2,1), B(3,2), B(3,1)의 처리가 완료된 후 큐의 내용을 나타낸다. $b_1 \neq b_2$인 블럭이 (a)형태의 어레이에서 처리될 때 x-방향으로 출력되는 중간계산결과를 저장하기 위한 큐는 그림 3의 블럭의 처리순서에 의해서 $1 \leq \alpha \leq b_1-2$인 $B(b_1-1, \alpha)$와 $b_1-1 \leq \beta \leq b_2+1$인 $B(b_1, \beta)$에서 출력되는 모든 데이타를 저장하여야 한다. 따라서 (a)형태의 어레이에서 요구되는 큐의 길이 q_{a1}은 식(15)와 같으며 (b)형태의 어레이에서 요구되는 큐의 길이 q_{b1}는 식(16)과 같다.

$$q_{a1} = \begin{cases} 2(M+1) & \text{, for x - direction} \\ 1 & \text{, for y - direction} \end{cases} \qquad (15)$$

$$q_{b1} = \begin{cases} \sum_{1 \le i \le n/M-2} \{M(n/M - i + 2) + 1\} \\ \quad = (n+3M+2)(n/M-2) & \text{, for x - direction} \\ M(n/M - 2 + 2) + 1 = n + 1 & \text{, for y - direction} \end{cases} \qquad (16)$$

4. 문제분할의 절차

제안한 문제분할의 절차는 다음과 같다.

[단계1] 이차원 어레이 구성을 위한 최적의 공간변환 S를 이용하여 문제를 어레이 어레이의 크기로 분할한다.

[단계2] 데이타 의존 및 제어신호의 흐름을 고려하여 분할된 블럭의 계산 순서를 결정한다.

[단계3] 블럭내의 각 계산점을 어레이의 처리요소에 할당한다.

[단계4] 시간변환 를 이용하여 처리요소에 할당된 각 계산점의 계산시각을 결정한다.

그림 6은 공간사상 S를 이용하여 어레이에서 처리될 수 있는 크기로 문제를 분할하고, 분할된 블럭의 계산순서를 결정한다.

```
L1:  procedure FIND-B(I,S,M,n)
       // I is index set, S is optimal space transformation //
       // M is array size, n is problem size //
L2:    initialization;
L3:    read S              // S=[S1 S2]T //
L4:    compose B           // B=(b1,b2,i,j,k) //
L5:    for i=1 to n do
L6:      for j=i to n do
L7:        for k=(i+j)/2 to j do
L8:          compose J               // J=[k i j]T //
L9:          b1 ← (S1·J-1)/M
L10:         b2 ← (S2·J)/M
L11:         B ← (b1,b2,i,j,k)
L12:         print(B)                // output B //
L13:       repeat
L14:     repeat
L15:   repeat
L16: end FIND-B
```

그림 6. 문제의 인덱스 공간에서 블럭의 인덱스 집합을 구하는 알고리듬

Fig. 6. The algorithm for finding the index set of blocks in the problems index space.

```
L1:  procedure ALLOCATE-L(I,S,M,n)
       // I is index set, S is optimal space transformation //
       // M is array size, n is problem size //
L2:    initialization;
L3:    read S              // S=[S1 S2]T //
L4:    compose L           // L=(type,x,y,i,j,k) //
L5:    for i=1 to n do
L6:      for j=i to n do
L7:        for k=(i+j)/2 to j do
L8:          compose J               // J=[k i j]T //
L9:          x ← (S1·J-2) mod M
L10:         y ← (S2·J-1) mod M
L11:         if b1=b2 then           // b1,b2 is block number of J //
L12:            type ← a             // a denote (a)-type of array //
L13:         else
L14:            type ← b             // b denote (b)-type of array //
L15:         endif
L16:         L ← (type,x,y,i,j,k)
L17:         print(L)                // output L //
L18:       repeat
L19:     repeat
L20:   repeat
L21: end ALLOCATE-L
```

그림 7. 블럭의 인덱스 점들을 어레이의 처리요소들로 할당하는 알고리듬

Fig. 7. The algorithm for allocating the index points of the block to PEs of array.

```
L1:  procedure SCHEDULE-T(I,Π,S,M,n)
       // I is index set, Π is optimal time transformation, //
       // S is optimal space transformation, M is array size,//
       // n is problem size //
L2:    initialization;
L3:    read S        // S=[S1 S2]T //
L4:    compose T     // T=(t,type,i,j,k) //
L5:    for i=1 to n do
L6:      for j=i to n do
L7:        for k=(i+j)/2 to j do
L8:          compose J              // J=[k i j]T //
L9:          b1 ← (S1·J-1)/M
L10:         b2 ← (S2·J)/M
L11:         if b1=b2 then
L12:           type ← a
L13:           if b1≤2 then
L14:             t ← Π·J + Σ_l(2(M+1))        // 1≤l≤b1-1 //
L15:
L16:           else
L17:             ta ← Σ_h{M(b1-h+2)+1} -2(M+1)
                                              // 1≤h≤b1-2 //
L18:             t ← Π·J + Σ_l(2(M+1)) + Σta
                   // 1≤l<b1-1, Σta is sum of delay time //
L19:           endif
L20:         else
L21:           type ← b
L22:           t ← Π·J - {M(b1-b2-1)+1} + Σ_l{Σ_m(M(l-m+3)+1)
                 + Σ_h{M(b1-h+3)+1} +2(M+2)
                 // 2≤l≤b1-1, 1≤m≤l-1, b2+1≤h≤b1-1 //
L23:         endif
L24:         T ← (t,type,i,j,k)
L25:         print(T)              // output T //
L26:       repeat
L27:     repeat
L28:   repeat
L29: end SCHEDULE-T
```

그림 8. 인덱스 점들의 시간 스케줄링 알고리듬

Fig. 8. The algorithm for time scheduling the index points.

그림 7은 블럭내의 각 계산점을 처리요소로 할당하는 알고리듬이다. 그림 8은 각 블럭의 처리시간과 (a)형태의 어레이에서의 계산지연시간을 이용하여 각 계산점이 처리요소에서 처리되는 시각을 결정한다. L14부터 L18까지는 지연시간 t_d를 계산하여 b_1 b_2인 계산점 j=(i,j,k)에 대한 계산시각을 결정하고, L22는 $b_1 \neq b_2$인 계산점 j=(i,j,k)의 계산시각을 구한다.

Ⅳ. 결과 및 고찰

문제를 분할하여 처리할 경우 전체처리시간은 각 블럭의 처리시간의 합과 같다. 어레이(b)는 어레이(a)로 사상되는 첫번째 블럭이 처리된 후 2단위시간 후에 동작되고, 어레이(b)로 사상되는 마지막 블럭에서는 어레이에서 출력되는 중간계산결과를 저장하지 않는다. 또한 그림 3의 어레이(a)와 (b)는 서로 독립적으로 동작되기 때문에 전체처리시간 t_T는 다음 식과 같다.

$$t_T = 2(M+1)+2+\sum_{i=2}^{n/M}\left[\sum_{j=i-1}^{1}\left[M\cdot(i-j+3)+1\right]\right]-2$$
$$= 2(M+1)+(n/M)(n/M-1)(n+10M+3)/6 \quad (18)$$

필요한 처리요소의 수 N은 다음과 같다.

$$N=M(M+1)/2+M^2=M(3M+1)/2 \quad (19)$$

처리요소의 이용율 μ는 계산시간 동안 전체 처리요소의 수에 대한 사용된 처리 요소의 비율로써 다음 식과 같다.

$$\mu = \sum_{1\le t\le T} p(t)/(N\cdot T) \quad (20)$$

여기서, p(t)는 t번째 단위시간에 이용된 처리요소의 수이다. $\Sigma p(t)$는 처리시간 동안 이용된 처리요소의 총 수로써 분할전 이차원 어레이에서의 수와 같다. 분할전 이차원 어레이에서 이용된 처리요소의 수는 각 위치의 처리요소가 이용된 시간을 모두 더한 것과 같다. 각 처리요소에서 이용된 시간은 다음과 같이 표현된다.

$$\sum_{1\le t\le T} p(t) = \sum_{1\le z\le n-1} p(z) = \sum_{1\le z\le n-1}\left\{(2z-\lceil 3z/2\rceil+1)\cdot(n-z)\right\}$$
$$= \sum_{1\le z\le n-1}\left\{(\lfloor z/2\rfloor+1)\cdot(n-z)\right\} \quad (21)$$

식(21)에서 $1\le z\le n$의 범위로 하면

$$\sum_{1\le t\le T} p(t) = \sum_{1\le z\le n} p(z) = \sum_{1\le z\le n}\left\{(2z-\lceil 3z/2\rceil+1)\cdot(n+1-z)\right\}$$
$$= \sum_{1\le z\le n}\left\{(\lfloor z/2\rfloor+1)\cdot(n+1-z)\right\} \quad (22)$$

이다.

그림 9는 문제의 크기(n)와 어레이 크기(M)의 변화에 대한 처리요소의 이용율을 나타낸다. 처리요소의 이용율은 문제를 분할하여 처리할 때와 어레이의 크기가 작을 경우 더 큰 이용율을 나타낸다.

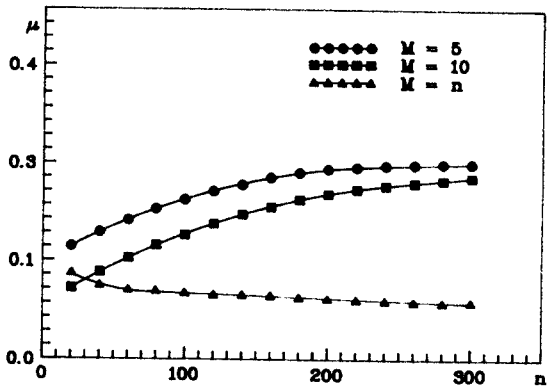


그림 9. 처리요소의 이용율
Fig. 9. The utilization of processing element.

블럭의 처리과정에서 (b)형태의 어레이로부터 출력되는 중간계산결과를 저장하기 위한 큐는 식(16)에 의해서 문제의 크기에 따라 증가한다. 따라서 실제 응용에서는 처리 할 문제의 범위와 제작할 어레이의 크기에 따라 (b)형태 어레이의 큐 길이를 결정하여야 한다.

Ⅴ. 결 론

동적 프로그래밍 문제 처리를 위한 이차원 시스토릭 어레이에서 문제분할 방법과 이에 적합한 이차원 어레이의 구성을 제안 평가하였다. 분할된 문제를 처리하기 위하여 두 가지 형태의 어레이를 사용하였으며 큐를 설계하였다.

문제를 분할하기 위하여 이차원 어레이 구성형태를 고려하여 이차원 어레이 구성을 위한 공간변환 S를 이용하여 제한된 크기의 어레이로 분할하고 이들 분할된 어레이에 문제 분할을 적용하였다. 즉 분할된 어레이로 사상되는 각 계산점들의 집합을 구함으로써 문제를 공간적으로 분할하였다. 이차원 어레이의 선형시간변환 Π를 이용하여 분할된 블럭내의 각 계산

점을 시간 스케쥴링하였으며, 처리과정에서 어레이로부터 출력되는 중간계산결과를 저장하기 위한 큐를 설계하였고, 큐로의 데이타 입 출력시간을 구하였다. 그리고 어레이와 큐의 정확한 동작을 위하여 각 어레이형태에 카운터를 사용하였다.

문제를 분할하여 처리할 경우 처리시간은 2(M+1)+(n+10M+3)(n/M)(n/M-1)/6 단위시간이 소요되며 M(3M+1)/2개의 처리요소가 요구된다. 또한 처리요소의 이용율은 분할하기 전의 이차원 어레이에서 보다 증가하고 어레이의 크기가 작을수록 처리요소의 이용율은 높아진다.

본 논문에서 제안한 이차원 시스토릭 어레이는 Context-free 언어 인식, 행렬곱의 최적화, 패턴인식 등과 같은 동적 프로그래밍의 실시간 처리에서 문제의 크기가 가변적인 경우에 이용될 수 있다.

參 考 文 獻

[1] R. Bellman, Dynamic programming, Princeton Univ. Press, Princeton, New Jersey, 1957.

[2] K.Q. Brown, " Dynamic programming in Computer Science ," Carnegie Mellon Univ., Tech. Report, CS-79-106, Feb. 1979.

[3] M.C.Chen, "Synthesizing VLSI architecture: Dynamic programming solver", *IEEE Int'l Conf.* on Parallel Processing, pp.776-784, Aug. 1986.

[4] G.Lee and B.W.Wah, "Systolic processing for dynamic programming problem", *Proc. of Int'l Conf.* on Parallel Processing, pp.434-441, Aug. 1985.

[5] 우종호, 안광선, "Context-free 언어 인식을 위한 일차원 시스토릭 어레 이의 설계," 전자공학회논문지, 제27권 제1호, pp. 30-36, 1990年.1月.

[6] S.Y. Kung, VLSI array processors, Prentice Hall, 1988.

[7] D. I. Moldovan, 'Modern parallel processing , Univ. of Southern California, pp.107-123, Jan, 1986.

[8] J.J. Navarro, J.M. Llaberia and M. Valero, "Solving matrix problems with no size restriction on a systolic array processor," *Proc. of, the ICPP*, pp.676-683, 1986.

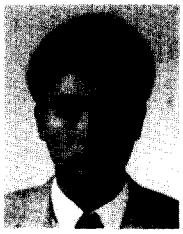
[9] 우종호, 안광선, " Polyadic-nonserial 동적 프로그래밍 처리를 위한시스 토릭 어레이의 설계 및 효율적인 운영", 전자공학회논문지, 제26권 제8 호, pp.51-59, 1989年 8月.

[10] 권대형, 신동석, 우종호, "동적 프로그래밍 처리를 위한 시스토릭 어레 이에서 문제 분할에 관한 연구, "정보과학회 추계학술발표논문집, vol. 18,no2,pp.729-732,1991.

著 者 紹 介

禹 鐘 鎬(正會員) 第27卷 第1號 參照
현재 부산수산대학교 전자공학과 부교수.

•



申 東 錫(正會員)
1961年 12月 16日生. 1985年 부산수산대학교 전자통신공학과 졸업. 1987年 부산수산대학교 전자통신공학과(석사). 1992年 2月 부산수산대학교 전자통신공학과 박사과정 수료. 1992년 ~ 현재 동명전문대학 전자계산과 전임강사. 주관심분야는 병렬처리, 컴퓨터구조, VLSI 등임.

鄭 信 一(正會員) 第26券 第8號 參照
현재 부산수산대학교 정보통신공학과 부교수.

•

權 大 亨(正會員)
1969年 12월 15日生. 1990年 부산수산대학교 전자공학과 졸업. 1992年 2月 부산수산대학교 전자통신공학과(석사). 1992年 ~ 현재 콤텍시스템 기술연구소 연구원. 주관심분야는 병렬처리 알고리듬, VLSI 등임.