Polyadic-Nonserial 동적 프로그래밍 처리를 위한 시스토릭 어레이의 설계 및 효율적인 운영

(A Design and the Efficient Operation of Systolic Array for Polyadic-Nonserial Dynamic Programming Processing)

禹 鍾 鎬*, 安 光 善**

(Chong Ho Woo and Kwang Sun An)

要 約

본 논문에서는 Polyadic-nonserial DP 문제를 처리하기 위한 시스토릭 어레이를 설계하고, 성능 평가와 효율적인 운영방법을 제안한다. 알고리듬을 단계적으로 변형하여 데이타 의존에서 broadcasting과 원격 통신패스를 제거하여 모든 통신을 local로 하였다. 이것을 Dan I. Moldovan의 방법을 이용하여 시스토릭 어레이에 mapping 하였다. 설계된 어레이는 동형(homogenous)이고 문제의 크기가 n인 경우 필요한 처리요소의 수는 $n(n+1)/2$이고 처리시간은 2n 단위시간이 소요된다. 처리할 문제가 많을 경우 초기값을 연속으로 입력하여 어레이의 효율을 증가시키고 이때 입력제한시간은 $\lfloor n/2 \rfloor + 1$이고 속도 향상은 약 4배로 된다. 각각의 경우에 대한 처리기의 이용율을 구하였다.

Abstract

In this paper, a systolic array for polyadic-nonserial DP problms is designed, the performance is analyzed and the efficient operating method is proposed. The algorithm is transformed to remove the broadcasting and global communication paths in the data dependece step by step. The transformed algorithm is mapping to the systolic array using the method proposed by D. I. Moldovan. The designed array is homogenous, has the processing elements of $(n+1)/2$ and 2n computation time (n is the size of problm). In case of being many problms to process, the efficiency of array can be upward by inputing the problms successively. The interval between the initiations of two successive problem instances is $\lfloor n/2 \rfloor + 1$ and the speed-up is about 4. The processor utilizations of each case are calculated.

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

**正會員, 慶北大學校 電子計算氣工學科
(Dept. of Computer Eng., Kyungpook Nat'l Univ.)

接受日字 : 1989年 5月 31日

I. 서 론

동적 프로그래밍(dynamic programming)은 일련의 판단들을 최적화하는 기법으로 여러분야에 널리 적용된다. R. Bellman에 의해서 처음 이 기법이 제안되어 여러 방향으로 발전되어 왔다. (discrete vs. continuous state, finite vs. infinite number of states,

deterministic vs. stocastic system)[1]. DP문제는 함수식(functional equation)의 형태에 따라 monadic-serial, monadic-nonserial, polyadic-serial, polyadic-nonserial로 분류된다.[2] K. Q. Brown은 단일 source 최단 패스 문제, 최적 parenthesization 문제, 최적분할 문제, 최적 matching 문제, N-P combinatorial 문제로 분류하였고 여기서 최적 parenthesization 문제는 polyadic-nonserial 문제이다. 이것은$0(n^3)$의 처리시간을 가지고 context-free 언어인식, 최적 searching 문제, 일련의 행열곱의 최적화등이 여기에 속한다.[3]

VLSI 기술의 진보와 더불어 파이프라인 기법과 병열처리 기법을 이용한 시스토릭 어레이의 개념이 제안되어[4] 알고리듬, 설계 방법 및 관련 소프트웨어 등 많은 연구가 되어 왔다.[5~12] 데이타 의존 관계가 규칙적이고 국소적인 알고리듬에 널리 이용된다. 특히 신호처리, 행열계산, 동적 프로그래밍문제, 관계 데이타 베이스 연산등 알고리듬이 순환식(recursive equation)으로 표현되는 경우 시스토릭 어레이로 구성하여 처리속도를 크게 향상시킬 수 있다.[5]

polyadic-nonserial DP 문제 처리를 위한 시스토릭 어레이는 L. J. Guibas 등이 제안하였으나 설계과정이 체계적이지 않다.[13] K. H. Chu 등은 contextfree 언어 인식을 위한 어레이를 설계했으나 Guibas가 제안한 어레이를 이용하였다.[14] B. W. Wah 등은polyadic-nonserial 문제를 polyadic-serial 문제로 변형하여 AND/OR 그래프를 이용하여 시스토릭 어레이에 mapping 시켰으나 중요한 동기화와 타이밍 문제를 다루지 않았다.[2] M. C. Chen은 'Crystal' 병렬언어를 이용하여 시스토릭 구조를 합성하는 방법을 제시하였고 이 과정에서 이 문제를 처리하는 시스토릭 어레이를 설계했으나 상세한 성능분석을 하지않았다.[9]

본 논문에서는 알고리듬을 체계적이고 단계적으로 변형하여 시스토릭 구조에 적합하도록 broadcasting과 원거리 통신패스를 제거한 후 C. I. Moldovan의 방법을 이용하여 변형된 알고리듬을 시스토릭 어레이에 mapping 시켰고, 타이밍 문제를 다루었다. 처리요소의 간략화를 위한 제어신호 전송 방법을 제안하고 설계한 어레이의 성능을 분석하였다. 연속으로 문제를 처리할 경우 초기값의 입력제한 시간과 이경우 처리기의 이용율을 계산했다.

Ⅱ. Polyadic-nonserial 동적 프로그래밍 알고리듬

DP 문제는 순환함수식(recursive functional equation)으로 표현되고, 식의 왼편은 함수명을, 오른편은 비용함수의 값을 구하는 식을 나타낸다. 비용함수는 최적의 원리(principle of optimlity)가 만족하도록 단조(monotonic)가 되어야 한다. DP 문제는 함수식의 형태와 순환의 성질에따라 여러가지로 분류된다. polyadic-nonserial DP 문제는 함수식의 형태가 다음과 같다.

$$f(X) \underset{1\le i\le n}{\Phi} h_i\ (X_i) \qquad (1)$$

여기서 $X=\{X_1, X_2, \cdots, X_n\}$ 이고 $X_i \subset X$, Φ는 h_i에 대해서 단조함수이다. 함수식을 알고리듬으로 표현하면 다음과 같다.

```
Algorithm for polyadic-nonserial DP Problm
  Input: a string of the initial values CO_i, 1≦i≦n-1
  Output: C_{i,j}
  begin
    for i=1 to n-1 do
      C_{i,i+1} ← CO_i
    for d=2 to n-1 do
      for i=1 to n-d do
        begin
          j← d+i
          C_{i,j} ← 0
          for k=i+1 to j-1 do
            C_{i,j} ← g(h(c_{i,k}, c_{k,j}))
        end
  end
```

여기서 CO_i는 문제의 초기값이고, $C_{i,j}$는 비용함수이다. 함수 g는 일반 DP 문제에서 min, max, ∪, ∩ 등이 되고 h는 문제에따라 간단한 계산을 수행하는 함수이다. $C_{i,j} \leftarrow 0$는 g의 계산을 위한 초기값을 설정하는 것으로 g함수에 따라 ∞, 0, e(empty set) 등이 된다. 위의 과정을 병렬 알고리듬으로 표현하면 다음과 같다.

```
Parallel Algorithm for Polyadic-nonserial DP Problm
  begin
    for all 1≦i≦n-1 do in parallel
      C_{i,i+1} ← CO_i
    all for
    for all 1≦i<j≦n do in parallel
      C_{i,j} ← 0
      for all i<k<j do in parallel
      C_{i,j} ← g(h(C_{i,k}, C_{k,j}))
    all for
    all for
  end
```

위의 알고리듬에서 데이타 의존 관계를 그래프로 나타내면 그림 1 과 같다.

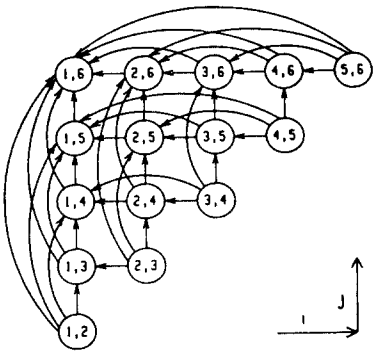


그림 1. Polyadic-Nonserial DP 문제의 데이타 의존 그리프(n= 6)

Fig. 1. The dependence graph of polyadic-nonserial DP (n=6).

Ⅲ. 알고리듬의 변형

그림 1 의 DAG (directed acyclic graph)에서 node는 처리를 나타내고 arc는 데이타 의존 관계를 나타낸다. VLSI 시스토릭 구조에 적합하도록 각 node에서 fan-in, fan-out가 일정한 값을 갖도록 알고리듬을 변형한다. fan-out는 위의 알고리듬에서 $c_{i,j}$ 가 계산되는 식의 우측 함수에서 k의 변화에 따라 결정되는 함수 h에서 $c_{i,k}$가 $k \leq j \leq n$의 범위에서 node i, j에 배정될 변수를 $a_{i,j,k}$로 $c_{k,j}$가 $1 \leq i \leq k$ 범위에서 node i, j에 배정될 변수를 $b_{i,j,k}$로 두어, $k=j$ 이면 $a_{i,j,k} \leftarrow c_{i,k}$로, $k=i$면 $b_{i,j,k} \leftarrow c_{k,j}$로 하고 이외의 조건에서는 $a_{i,j,k} \leftarrow a_{i,j-1,k}$, $b_{i,j,k} \leftarrow b_{i+1,j,k}$로 되도록 변형하면 모든 node의 fan-out는 1 이 된다.

함수 g의 연산에서 결합율(associative law)이 만족하면 다음의 간단한 평제를 이용하여 fan-in을 줄일 수 있고, 대부분의 poliadic-nonserial 동적 프로그래밍 문제의 함수 g에서 연산(max, min, ∪, ∩)은 결합율을 만족한다.

〔명제〕 연산 ⊕가 $u<1<v$ 범위의 v-u-1개의 인수 $x(1)$에 대해서 $z=\oplus\, x(1)$이 결합율(associative law)을 만족하면 $u<1<v$
다음과 같이 둘 수 있다.

$1=m$ 이면 $z(1)=x(m)\oplus x(u+v-m)$
$m<1<v$면 $z(1)=z(1-1)\oplus x(1)\oplus x(u+v-1)$
$1=v$ 이면 $z(1)=z(1-1)$
$z=z(v)$
여기서 $m=\lceil (v+u)/2 \rceil$ 이다.

명제를 앞의 알고리듬에 적용하고 $c_{i,j}$를 $C_{i,j,k}$ 로 인덱스를 확장하면 알고리듬은 다음과 같이 변형되고 fan-out, fan-in이 상수로 한정된다.

```
Algorithm with the constant fan-in and fan-out
 begin
  for all 1≦i<j≦n do in parallel
   c_{i,j,j} ← 0
   if j−i= 1 then
    c_{i,j,j} ← co_i
   else
    for all i≦k≦j do in parallel
    if k=j then
     a_{i,j,k} ← c_{i,j,k},  c_{i,j,k} ← c_{i,j,k-1}
    else
     a_{i,j,k} ← a_{i,j-1,k}
    end if
    if i=k  then
     b_{i,j,k} ← c_{k,j,k}
    else
     b_{i,j,k} ← b_{i+1,j,k}
    end if
    if k=⌈(i+j)/2⌉ then
     c_{i,j,k} ← g(h1(a_{i,j,k}, b_{i,j,k}),
                  h2(a_{i,j,i+j-k}, b_{i,j,i+j-k}))
    else
     if ⌈(i+j)/2⌉<k<j then
      c_{i,j,k} ← g(c_{i,j,k-1}, h1(a_{i,j,k}, b_{i,j,k}),
                  h2(a_{i,j,i+j-k}, b_{i,j,i+j-k}))
     end if
    end if
   all for
  end if
 all for
end
```

위의 알고리듬에는 원거리 통신 패스가 존재하여 시스토릭 구조에 적합하지 않으므로 이것들을 local로 만들어야 한다. 데이타 의존관계에 의해서 $h(a_{i,j,k}, b_{i,j,k})$는 $\max(k-i, j-k)$ 단위 시간전에는 계산될 수 없고, $k=\lfloor (i+j)/2 \rfloor$ 혹은 $k=\lceil (i+j)/2 \rceil$ (범위 $i<k<j$의 중간)에서 가장 먼저 수행된다. 식에서 $a_{i,j,k}$와 $a_{i,i+j-k}$가 존재하므로 $i<k<j$ 범위에서 통신 패스의 길이는 $2(|k-(i+j)/2|)$가 되고 n이 증가함에 따라

이 패스의 길이도 커진다. 여기서 $k=(i+j)/2$로 두면 $a_{i,j,k}$와 $a_{i,j,i+j-k}$는 하나로 되어 통신 패스가 '0'이 된다. 인덱스(i, j, k)에 대해서 기하적으로 고려하면 평면 $k=(i+j)/2$에 대해서 $a_{i,j,k}$와 $b_{i,j,k}$는 각각 $a_{i,j,i+j-k}$와 $b_{i,j,i+j-k}$에 대칭이 된다. $i\leqq k\leqq j$ 범위에서 데이타 a와 b가 $i\leqq k\leqq\lfloor(i+j)/2\rfloor$와 $\lceil(i+j)/2\rceil\leqq k\leqq j$의 범위로 대칭으로 반분되어 있다. 이 두 범위를 $\lceil(i+j)/2\rceil\leqq k\leqq j$로 축소하고 $i\leqq k\leqq\lfloor(i+j)/2\rfloor$에 존재하는 $a_{i,j,k}$, $b_{i,j,k}$를 각각 $d_{i,j,i+j-k}$, $e_{i,j,i+j-k}$로 두면 모든 통신패스가 local로 된다. 한편 $j-i$가 짝수이고 $k=\lfloor(i+j)/2\rfloor$이면 인덱스(i, j, k)의 데이타는 $i\leqq k\leqq\lfloor(i+j)/2\rfloor$에 존재하고 인덱스$(i, j-1, k)$의 데이타는 $\lceil(i+j)/2\rceil\leqq k\leqq j$에 존재하므로 $d_{i,j,k}\leftarrow a_{i,j-1,k}$로 두고, 한편 $k=\lceil(i+j)/2\rceil$이면 인덱스(i, j, k)의 데이타는 $\lceil(i+j)/2\rceil\leqq k\leqq j$에 인덱스$(i+1, j, k)$의 데이타는 $i\leqq k\leqq\lfloor(i+j)/2\rfloor$에 존재하므로 $b_{i,j,k}\leftarrow e_{i+1,j,k+1}$로 둔다. 식을 변형하는 과정에서 $i+j-k$를 k'로 두고 다시 k'를 k로 두어 정리하면 알고리듬은 다음과 같이 변형된다.

Algorithm with the constant fan-in, fan-out and localized path

```
begin
  for all 1≦i≦j≦n do in parallel
    c_{i,j,j} ← o
    if j−i=1 then
      c_{i,j,k} ← co_i
    else
      for all ⌈(i+j)/2⌉≦k≦j do in parallel
        if j=k then
          a_{i,j,k} ← c_{i,k,k}, e_{i,j,k} ← c_{i,j,j}, c_{i,j,k} ← c_{i,j,k-1}
        else
          a_{i,j,k} ← a_{i,j-1,k}, e_{i,j,k} ← e_{i+1,j,k+1}
          if ⌈(i+j)/2⌉=k then
            if even(j−i) then
              d_{i,j,k} ← a_{i,j-1,k}, b_{i,j,k} ← e_{i+1,j,k+1}
            else
              d_{i,j,k} ← d_{i,j-1,k-1}, b_{i,j,k} ← b_{i+1,j,k}
            end if if
            c_{i,j,k} ← g(h1(a_{i,j,k}, b_{i,j,k}), h2(d_{i,j,k}, e_{i,j,k}))
          else
            d_{i,j,k} ← d_{i,j-1,k-1}, b_{i,j,k} ← b_{i+1,j,k}
            c_{i,j,k} ← g(c_{i,j,k-1}, h1(a_{i,j,k}, b_{i,j,k}), h2(d_{i,j,k}, e_{i,j,k}))
          end if
        end if
      all for
    end if
  all for
end
```

위의 알고리듬은 fan-out, fan-in이 일정하고, 모든 통신 패스가 local이다.

Ⅳ. 시스토릭 어레레이의 구성

알고리듬을 시스토릭 어레이에 mapping 하는 기법에 관해서 많은 연구가 있어 왔다. 본 연구에서는 Moldovan의 방법을 이용한다.

알고리듬에서 k인덱스를 시간 t인덱스로 i와 j인덱스를 x, y 어레이 공간으로 mapping한다. 각 데이타에 대한 데이타 의존 벡타 $d_i=[k, i, j]^T$를 구하면 a는 $d_1=[0,0,1]^T$, e는 $d_2=[-1,-1,0]^T$, b는 $d_3=[0,-1,0]^T$, d는 $d_4=[1,0,1]^T$, c는 $d_5=[1,0,0]^T$로 된다. 기본 통신패스(primitive communication path) $p_i=[x, y]^T$는 a와 d는 $[1,0]^T$로, e와 b는 $[0,-1]^T$로, c는 $[0,0]^T$ 패스를 이용한다. 데이타 의존 행열 D, 원시 통신패스 행열 P, 패스 이용 행열 K를 구하면 다음과 같다.

$$D=\begin{bmatrix}0&-1&0&1&1\\0&-1&-10&0\\1&0&0&1&0\end{bmatrix},\ P=\begin{bmatrix}0&1&0\\0&0&-1\end{bmatrix},$$

$$K=\begin{bmatrix}0&0&0&0&1\\1&0&0&1&0\\0&1&1&0&0\end{bmatrix} \tag{1}$$

변형(transformation) 행열 $T=[\Pi, S]^T$에서 Π는 최소의 시간이 소요되는 조건으로 구하면 $\Pi=[1,-2,1]$로 된다. $SD=PK$로 두고 S, T 및 T의 역행열을 구하면 다음과 같다.

$$S=\begin{bmatrix}0&0&1\\0&1&0\end{bmatrix},\ T=\begin{bmatrix}1&-2&1\\0&0&1\\0&1&0\end{bmatrix}$$

$$T^{-1}=\begin{bmatrix}1&-1&2\\0&0&1\\0&1&0\end{bmatrix} \tag{2}$$

그래서

$$\begin{bmatrix}t\\x\\y\end{bmatrix}=T\cdot\begin{bmatrix}k\\i\\j\end{bmatrix}=\begin{bmatrix}t-2i+j\\j\\i\end{bmatrix}$$

$$\begin{bmatrix}k\\i\\j\end{bmatrix}=T^{-1}\begin{bmatrix}t\\x\\y\end{bmatrix}=\begin{bmatrix}t-x+2y\\y\\x\end{bmatrix} \tag{3}$$

이 된다.

P행열의 요소와 K의 요소의 선택의 경우에 따라 많은 S가 구해질 수 있고 통신 패스와 계산시간에 있어서 가장 적합한 것의 행열 P와 K의 요소를 선택한다.

앞의 알고리듬에서 익덱스$[i, j, k]^T$를 $[x, y, t]^T$로 하고, $z=x-y$로 두어, 식의 좌측편의 인덱스는 x, y, t로 두고 데이타 의존 행렬이 $D_1=T\cdot D$가 되도록 한다. 연속입력이 가능하도록 $2z=t$ 조건에서 동작이 완료된 후 $C_{x,y,t} \leftarrow 0$이 되게 한다. 조건식도 x, y, t의 함수(즉 z와 t의 함수)가 되게 한다. 결과의 시스토릭 알고리듬은 다음과 같다.

```
Systolic Algorithm for Polyadic-Nonserial
  begin
    for all 1≦y<x≦n do in parallel
      if x−y=1 then
        c_{x,y,t} ← co_t
      end if
      for all ⌈3z/2⌉≦t≦2z do in parallel
        if 2z=t then
          a_{x,y,t} ← c_{x,y,t-1}, e_{x,y,t} ← c_{x,y,t-1}, c_{x,y,t} ← 0
        else
          a_{x,y,t} ← a_{x-1,y,t-1}, e_{x,y,t} ← e_{x,y+1,t-1}
          if t=⌈3z/2⌉ then
            if even(z) then
              d_{x,y,t} ← a_{x-1,y,t-1}, b_{x,y,t} ← e_{x,y+1,t-1}
            else
              d_{x,y,t} ← d_{x-1,y,t-2}, b_{x,y,t} ← b_{x,y+1,t-2}
            end if
            c_{x,y,t} ← g(h1(a_{x,y,t}, b_{x,y,t}), h2(d_{x,y,t}, e_{x,y,t}))
          else
            d_{x,y,t} ← d_{x-1,y,t-2}, b_{x,y,t} ← b_{x,y+1,t-2}
            c_{x,y,t} ← g(c_{x,y,t-1}, h1(a_{x,y,t}, b_{x,y,t}),
                  h2(d_{x,y,t}, e_{x,y,t}))
          end if
        end if
      all for
    all for
  end
```

위의 알고리듬에서 어레이의 처리요소를 설계할 때 조건문은 처리요소의 제어 논리에 해당하고 조건식에 시간 t와 위치 $z=x-y$를 포함하고 있으므로 시간을 카운터하는 회로와 위치를 보관하는 기억회로가 필요하다. VLSI 시스토릭 구조에서 이러한 현상은 처리요소를 복잡하게하고 어레이의 규모가 처리 알고리듬보다 작을 경우 알고리듬 분할을 어렵게 한다. 이 문제를 해결하기 위하여 인덱스 t를 없애고 위치를 나타내는 z를 없애기 위하여 각 처리요소에 제어 변수를 두어 이웃처리 요소로 옮겨가도록 한다. 즉 알고리듬의 배정문에서 인덱스 t를 없애고 a, e에 대해서 각각 하나의 레지스터를 두고 한단위 시간에 d와 b에 대해서 각각 두개의 레지스터로 연결하여 두 단위시간에 이웃 레지스터로 파이프라인 형태로 옮겨가도록 한다.

조건식에서 위치와 시간에 관련되는 변수를 제거하기 위하여 제어 변수를 두어 이웃 처리요소로 옮긴다. 시간 t가 증가함에 따라 $z=x-y$가 증가되는 방향으로 제어가 이동되므로 제어 변수들이 통신패스를 $[x, y]=[1,0]$로 사용하는 것과 $[0,-1]$을 사용하는 것이 같은 결과를 가진다. 본 논문에서는 모든 제어신호가 $[0,-1]$패스를 이용하는 것으로 가정한다.

시간 t와 처리요소의 관계는 조건식$\lceil 3z/2\rceil \leqq t \leqq 2z$에서 각 처리기의 위치와 동작되는 시간범위로 표시될 수 있다. $\lceil 3z/2\rceil=t$와 관련되는 제어변수를 $m2_{x,y}$로 두고 셋 단위시간마다 두개의 처리요소를 지나도록 하고, $t=2z$와 관련되는 제어변수를 $m3_{x,y}$로 두어 두 단위시간마다 이웃 처리기로 전달되게 한다. 처리기에서 $m2_{x,y}$가 한 단위시간 동안만 '1' 상태로 있고 그 조건에 따라 동작되고 $m3_{x,y}$가 '1'의 상태로 되기전까지 $\lceil 3z/2<t<2z\rceil$의 조건에서 동작된 후, 한 단위시간 동안 $m\ 3_{x,y}$가 '1'의 상태로 있고 $t=2z$의 조건에 따라 동작된다. 이 후 '$c_{x,y,t} \leftarrow 0$'로 한다. 그러면 m2에서 m3까지 존재하는 처리기만 동작되고 이 외의 처리기는 아무런 동작을 수행하지 않는다. 이 때 동작되는 처리기의 집합은 z의 값에 따라 밴드(band) 모양으로 되며 이것을 이용밴드라 한다. m2가 m3보다 전달속도가 빠르므로 시간이 경과되면서 이 밴드가 커지면서 z가 증가하는 방향으로 옮겨가고 어레이에서 없어지면서 처리가 완료된다.

어레이를 구성하는 처리요소를 모두 같게 하여 어레이를 동형(homogeneous)으로 하고 처리기의 위치 즉 z가 짝수인 조건을 인식할 수 있게 $m1_{x,y}$ 제어변수를 두어 한 단위시간마다 이웃 처리기로 이동시킨다. 이것은 한번 결정되면 그 값으로 고정된다. 한편 어레이가 동작을 시작하기전에는 처리요소에 있는 모든 레지스터는 '0'으로 클리어 된다고 가정한다.

이러한 모든 조건들은 하드웨어로 쉽게 구현이 가능하고 설계한 처리요소는 그림 2와 같다. 여기서 'Condition 1'은 $\lceil 3z/2\rceil=t$의 조건에서, 'Condition 2'는 $\lceil 3z/2\rceil<t<2z$의 조건에서, 'Condition 3'은 $z=2t$의 조건에서 수행하는 동작들을 나타낸다. Fg는 처리요소가 현재 $\lceil 3z/2\rceil<t<2z$의 조건에 있는 여부를 표현하기위한 한 비트의 기억회로이다.

Procedure of Processing Element located at (x, y)

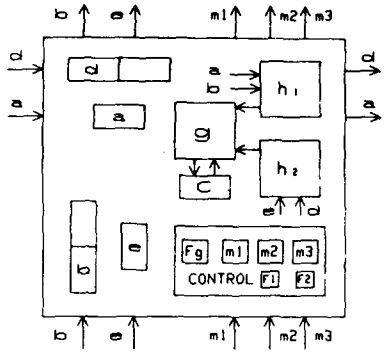


그림 2. 처리요소
Fig. 2. The processing element.

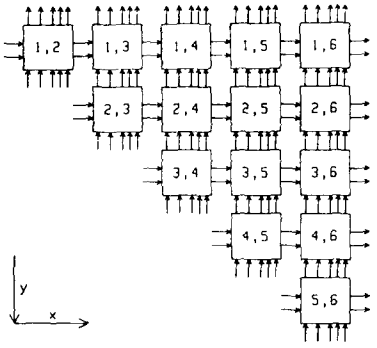


그림 3. 설계한 시스토릭 어레이(n= 6)
Fig. 3. The designed systolic array (n=6).

if $m1_{x,y+1}=0$ then $m1_{x,y}\leftarrow 1$ else $m1_{x,y}\leftarrow 0$ end if
if $F1_{x,y}=1$ and $F2_{x,y}=1$ then
Input initial values (z= 1)
$F1_{x,y}\leftarrow 0$, $F2_{x,y}\leftarrow 0$, $m2_{x,y}\leftarrow 1$, $m3_{x,y}\leftarrow 1$
else
if $m1_{x,y}=1$ then
if $m2_{x,y+1}=1$ then
Even of Condition 1, $m2_{x,y}\leftarrow 1$, $Fg_{x,y}\leftarrow 1$
else $m2_{x,y}\leftarrow 0$ end if
else
if $m2_{x,y+1}=1$ then
$F1_{x,y}\leftarrow 1$
else
if $F1_{x,y}=1$ then
Odd of Condition 2, $F1_{x,y}\leftarrow 0$, $Fg_{x,y}\leftarrow 1$,
$m2_{x,y}\leftarrow 1$
else $m2_{x,y}\leftarrow 0$ end if
end if
end if
if $F2_{x,y}=1$ then
Condition 3, $F2_{x,y}\leftarrow 0$, $Fg_{x,y}\leftarrow 0$, $m3_{x,y}\leftarrow 1$
else
$m3_{x,y}\leftarrow 0$
if $m3_{x,y+1}=1$ then $F2_{x,y}\leftarrow 1$ end if
if $Fg_{x,y}=1$ then Condition 2 end if
end if
end if

설계한 처리요소들로 어레이를 구성하면 그림 3과 같다.

그림 3에서 z= 1이 되는 처리요소들로 초기값과 제어변수가 입력되고 m1은 항상 '1'로 고정되고, 초기값을 입력할 경우 m2, m3는 한 단위시간 동안만 '1'로 두고 이 때 초기값 CO_i가 어레이에 입력되게 한다.

V. 성능분석 및 효율적인 운영

시스토릭 어레이를 평가하는 기준은 여러가지가 있고 여기서는 계산시간(computation time), 어레이의 크기(array size) 즉 처리요소의 수, 처리기의 이용율(processor utilization)을 다룬다.

설계된 시스토릭 어레이에서 계산시간은 다음 식과 같다.

$$T=\lceil (\max \Pi(J_1-J_2)+1)/(\min \Pi d_i)\rceil \qquad (1)$$

여기서 Π는 설계과정에서 구해진 $[1,-2,1]$ J_1, J_2는 알고리듬의 인덱스 공간에서 임의의 두 요소이다. $\min(\Pi\cdot di)=1$ 이고 문제의 크기가 n이면 $\max(\Pi\cdot(J_1-J_2))+1=\lceil 1,-2,1\rceil\cdot[n-1,0,n-2]^T+1=2n-2$ 가 되어 $T=2n-2$ 이다. 본 논문의 전개과정에서 문제의 크기와 어레이의 규모를 n−1로 하였고 문제의 크기가 n이면

$$T=2n \qquad (2)$$

이다

필요한 처리요소의 수는 $N=n(n-1)/2$이 되고 문제의 크기가 n이면

$$N=n(n+1)/2 \qquad (3)$$

이다.

처리기의 이용율은 계산시간동안 전체 처리요소의 수에 대한 사용된 처리기의 비율로 다음 식과 같다. 여기서 p(t)는 t번째 단위시간에서 이용된 처리기의 수이다.

$$\mu=\sum_{1\le t\le T} p(t)/N\cdot T \qquad (4)$$

여기서 $\Sigma p(t)$는 각 단위시간마다 이용된 처리요소의 수를 $1\le t\le T$ 범위에서 모두 더한 것으로 각 위치의 처리요소가 이용된 시간을 모두 더한 것과 같다. 각 처리요소에서 이용된 시간은 알고리듬의 조건식에서 $\lceil 3z/2\rceil\le t\le 2z$이고, $z=x-y$로 $1\le y<x\le n$의 주어진 범위에서 $1\le z\le n-1$의 범위에 있고 z의 위치에 있는 처리요소의 수는 $n-z$이므로 다음 식이 성립된다.

$$\begin{aligned}\sum_{1\le z\le n-1} p(z) &= \sum_{1\le z\le n-1} (2z-\lceil 3z/2\rceil+1)\cdot(n-z)\\ &= \sum_{1\le z\le n-1} (\lfloor z/2\rfloor+1)\cdot(n-z)\\ &= n(n-1)/2+\sum_{1\le z\le n-1} (\lfloor z/2\rfloor\cdot(n-z) \qquad (5)\end{aligned}$$

그래서 이용율 μ는 다음과 같다.

$$\mu=[n(n-1)/2+\sum_{1\le z\le n-1} ([z/2]\cdot(n-z))]/[(n(n-1)/2)\cdot 2(n-1)] \qquad (6)$$

여기서 문제의 크기를 n으로 하면 $1\le z\le n$의 범위로 하고 n을 n+1로 두고 식을 전개하면

$$\mu_1=[n(n+1)/2+\sum_{1\le z\le n} (\lfloor z/2\rfloor\cdot(n+1-z))]/[(n(n+1)/2)\cdot 2n] \qquad (7)$$

이 된다.

많은 문제를 처리할 경우 하나의 문제가 완료된 후 다음 문제를 처리하지 않고 연속해서 입력하여 처리기의 이용율을 높이고 평균 처리시간을 단축시킬 수 있다. 이 때 하나의 초기값을 입력한 후 다음 초기값을 입력할 때 정확한 지연시간을 두어야한다. 지연시간은 앞에서 설명한 처리요소의 동작밴드에 의존되며 다음과 같다.

$$\Delta t\ge 2z-\lceil 3z/2\rceil+1=\lfloor z/2\rfloor+1 \qquad (8)$$

어레이의 크기가 n일 경우 최적의 지연시간은 다음과 같다.

$$\Delta t=\lceil n/2\rceil+1 \qquad (9)$$

즉 문제가 커지면 지연시간이 커지고 이것을 블럭 파이프라인(block pipeline) 시간이라 한다.

문제가 연속해서 입력될 경우 처리기의 이용율은 처리한 문제의 수가 증가함에 따라 증가하고 다음과 같다.

$$\mu_2=M\cdot P(n)/(T_1\cdot N) \qquad (10)$$

여기서 M은 처리한 문제의 수이고, 이때 처리시간은

$$T_1=(M-1)\Delta t+2n$$

이고 P(n)은 한 문제를 처리할 때 사용하는 처리기의 수로서

$$P(n)=\sum_{1\le t\le 2n}\Big(\sum_{\lceil t/2\rceil\le z\le \lfloor 2t/3\rfloor\le n}(n-z+1)\Big) \qquad (11)$$

이다. 그리고 처리한 문제수가 매우 클때 이용율은

$$\begin{aligned}\mu^{\infty} &= M\cdot P(n)/((M-1)\Delta t+2n)n(n+1)/2)\\ &= P(n)/(((1-1/M)\Delta t+2n/M)n(n+1)/2)\\ &\cong 2P(n)/(\Delta t\cdot n(n+1)) \qquad (12)\end{aligned}$$

연속으로 입력할 경우 처리속도의 향상은

$$S=2n/\Delta t=2n/(\lfloor n/2\rfloor+1)\cong 4 \qquad (13)$$

이다.

다음 표는 문제의 크기 n과 처리시간 T의 변화에 대한 처리기의 이용율 나타낸 것이다.

표 1. n과 T에 대한 처리기의 이용율
Table 1. Processor utilization to n, T.

T	μ_1			μ_2		
	n=5	n=20	n=100	n=5	n=20	n=100
10	.18667	.10619	.02501	.42667	.10619	.02501
50	.18667	.10933	.07549	.58533	.30848	.07549
100	.18667	.11548	.10911	.60267	.35343	.14556
200	.18667	.11012	.08874	.61300	.37702	.24145
1000	.18667	.11012	.08874	.62027	.39580	.32671
∞	.18667	.11012	.08874	.62222	.40043	.34075

표에서 문제의 크기 즉 어레이의 규모가 커지면 이용율은 감소하고, 처리시간이 경과되면서 역시 이용율도 증가하고 일정한 값에 접근한다. $\mu=4\mu_1$이고 연속으로 문제를 입력할 경우 이용율은 4배로증가하며 처리속도 향상과 같다.

Ⅵ. 결 론

본 논문에서는 polyadic-nonserial D.P. 문제를 처리하기위한 시스토릭 어레이를 체계적으로 설계 및

분석하였고, 문제를 연속으로 처리할 경우 초기값의 입력 제한시간을 구하여 처리기의 이용율을 극대화하는 방법을 제시하였다.

알고리듬 변형을 단계적으로 적용하여 처리기의 fan-in, fan-out의 수를 고정시키고 원거리 패스를 제거하여 모든 통신패스를 local로 하였다. 설계된 어레이에서 처리시간은 2n, 처리요소 수는 $n(n+1)/2$ 이고, 이용율은 n이 증가함에따라 감소한다. 연속으로 입력하여 처리할 경우 입력 제한시간은 $\lfloor n/2 \rfloor + 1$ 이 된다. 이때 평균 처리속도는 4 배로 향상되며 처리기의 이용율도 같은 비율로 증가된다.

본 연구의 결과는 context-free언어의 인식, 음성인식, 일련의 행열곱을 효과적으로 처리하는 문제, 최적 searching 문제 등에 이용될 수 있고 특히 실시간 처리로 응답시간이 중요한 문제로 고려될 때 유용하다.

polyadic-nonserial DP 문제를 일차원 선형 어레이로 mapping, 알고리듬의 분할은 다음 연구과제로 남겨둔다.

參 考 文 獻

[1] M.E. Thomas, "A survey of the state of the art in dynamic programming," *AIIE Trans.*, vol. 8, no. 1, pp. 59-69, March 1976.

[2] U. Bertele and F. Brioschi, Nonserial Dynamic Programming, Academic Press, New York, 1972.

[3] K.Q. Brown, Dynamic Programming in Computer Science, Tech. Report CMU-CS-79-106, Carnegie-Mellon Univ., 1979.

[4] H.T. Kung and C.E. Leiserson, "Systolic array (for VLSI)," symposium on Sparse Matrix Computations, pp. 256-282, Nov. 1978.

[5] H.T. Kung, "Why systolic architectures?", *IEEE Computer*, pp. 37-46, Jan. 1982.

[6] D.I. Moldovan, "On the design of algorithms for VLSI systolic arrays," *Proc. of the IEEE*, vol. 71, no. 1, pp. 113-120, Jan. 1983.

[7] D.I. Moldovan and J.A. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Trans. on Computers*, vol. C-35, no. 1, pp. 1-12, Jan. 1986.

[8] P.R. Cappello and K. Steiglitz, "Unifying VLSI array designs with geometric translations," *IEEE Proc. of International Conf. on Parallel Processing*, pp. 448-457, 1983.

[9] M.C. Chen, "A design methodology for synthesizing parallel algorithms and architectures," Journal of Parallel and Distributed Computing, vol. 2, pp. 461-491, 1986.

[10] G.J. Li and B.W. Wah, "The design of optimal systolic arrays," *IEEE Trans. on Computers*, vol. C-34, no. 1, pp. 66-77, 1985.

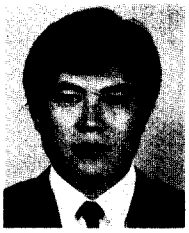
[11] O.H. Ibarra, S.M. Kim and M.A. Palis, "Designing systolic algoriithms using sequential machines," *IEEE Trans. on Computers*, vol. c-35, no. 6, pp. 531-542, Jun. 1986.

[12] S.Y. Kung, VLSI Array Processors, Prentice Hall, 1988.

[13] L.J. Guibas, H.T. Kung and C.D. Thompson, "Direct VLSI implementation of combinatorial algorithms," CALTECH Conf. on VLSI, pp. 509-525, Jan. 1979.

[14] K.H. Chu and K.S. Fu, "VLSI architectures for high speed recognition of context-free languages," *Proc. of 9th Symposium on C.A.* pp. 43-49, April 1982. *

著 者 紹 介



禹 鍾 鎬(正會員)

1954年 7月 9日生. 1978年 2月 경북대학교 전자공학과(전자 계산기공학) 졸업. 1981年 2月 경북대학교 대학원 전자공학과(전산공학전공) 석사학위 취득. 1986年 경북대학교 대학원 전자공학과 박사과정 수료. 1978年～1981年 2月 경남공업 전문대학 전자계산과 전임강사. 1981年 3月～현재 부산수산대학 전자공학과 부교수. 1987年 8月～1988年 8月 미국 렌슬러 공대(RPI) 전자계산학과 객원연구원. 주관심분야는 병열처리, VLSI 계산, cellular 오토마타 등임.

安 光 善 (正會員) 第25卷 第7號 參照

현재 경북대학교 전자계산기공학과 교수.