

論 文

Dataflow 연산에 의한 FFT 알고리즘의 구성

正會員 李 相 範* 正會員 朴 贊 政**

Structuring FFT Algorithm for Dataflow
Computation

Sang Burm RHEE * and Chan Jung PARK,* Regular Members.

요 약 Dataflow 컴퓨터는 프로그램이 고도의 병렬성을 갖고 수행될 수 있어 von-Neumann 기제 이상으로 계산처리 능력을 향상시키게 된다. 본 논문에서는 FFT Butterfly 알고리즘을 구성하여 dataflow 시뮬레이션을 통하여 수행하였다. 또한 이 알고리즘을 dataflow 연산으로 수행시킬 때에 프로그램 수행속도 증가비를 구하여 연산 속도를 향상시킬 수 있음을 보였다.

ABSTRACT Dataflow computers exhibit a high degree of parallelism which can not be obtained easily with the conventional von-Neumann architecture. Since many instructions are ready for execution simultaneously, concurrency can be easily achieved by the multiple processors modified the dataflow machine. This paper describes a FFT Butterfly algorithm for dataflow computation and evaluates the performance by the speed up factor of that algorithm through the simulation approach by the time-accelation method.

1. 서 론

최근 대규모 연산을 병렬로 처리하므로써 컴퓨터의 성능을 향상 시키기 위한 많은 연구가 진행되고 있다.

von-Neumann 컴퓨터 구조에서는 프로그램이 기억장치에 연속적으로 저장되어 있고, 이것이 수행될 때에는 프로그램 카운터가 지정한 기억장소로부터 순서대로 명령어를 추출(fetch)하여 수행되기 때문에 컴퓨터의 성능을 향상시키는데 많은 제한을 받게 된다⁽¹⁾ 이러한 제한을 극복하기 위한 한 방법으로 packet communication 기제조직을

바탕으로 하는 dataflow 구조가 연구되고 있다⁽²⁾ 이는 여러 개의 연산처리장치를 시간과 순서에 관계없이 각 장치들 사이에서 비동기적(asynchronous)으로 데이터가 도착 되자마자 병렬로 수행시켜 컴퓨터의 성능을 향상시킬 수 있다. 즉 그 기제어인 dataflow 프로그램 그래프에 의해 데이터를 병렬 처리하므로써 고도의 병렬성(high concurrency)을 제공하므로 대규모 연산이나 실시간 처리에 적합하다.

본 연구에서는 dataflow 구조 및 연산개념을 고찰하고 FFT(fast fourier transform) Butterfly 알고리즘을 구성하여, 이 알고리즘의 수행을 dataflow 컴퓨터 시뮬레이션을 통하여 실현하였다. 또한 이 알고리즘을 dataflow 연산으로 수행시킬 때, 프로그램 수행속도 증가비(speed up factor)를 구하여 연산속도를 향상시킬 수 있음을 보였다.

* ** 檀國大學校工科大學電子工學科
Dept. of Electronic Engineering Dankook University,
Seoul, 140 Korea.
論文番號 : 85-23(接受 1985. 6. 19)

2. Dataflow 구조 및 연산개념

Dataflow 컴퓨터를 구성하는 여러가지 방법중에서 MIT dataflow 컴퓨터는 packet communication 기체조직에 바탕을 두고 있다. Packet communication 기체조직은 각 장치들이 서로 독립적으로 packet 를 pipeline 형태로 단일 방향으로만 제공하는 편리한 방법이다⁽³⁾.

MIT dataflow 컴퓨터 구조는 그림 1 과 같다⁽⁴⁾.

그림 1 에서 기억장치(memory unit)는 여러개의 명령어 cell 로 구성되며, 이들은 각각 arbitration /distribution network 에 의하여 독립적으로 access/update 된다. 각 cell 에 저장되는 명령어 형식은 다음과 같이 여러 field로 구성된다.

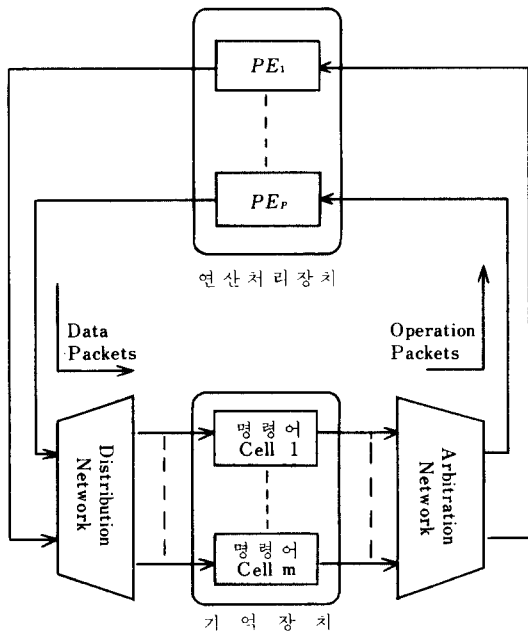


그림 1 MIT dataflow 컴퓨터구조
MIT dataflow computer architecture.

명령어 형식 : <operation code, operand, destination/port>

Arbitration network는 처리장치 (processing unit)의 PE(processing element)가 idle 상태인가를 판단하여 수행이 가능한 명령어 cell의 데이터를 access하여 operation packet <operation code,

operand, destination/port>를 만들어 처리장치로 보내고, 명령어 cell의 오퍼랜드를 리세트시킨다. 처리장치는 operation packet을 받아 operation code에 따라 연산처리하여 data packet <data, destination/port>를 distribution network에 보낸다. Distribution network는 data packet를 받아 destination/port가 지정하는 명령어 cell의 오퍼랜드에 데이터를 저장한다. Arbitration/distribution network는 packet들을 전달하는 일종의 스위칭 network이며 각 장치들 사이의 packet 전달은 비동기로 two-way handshake protocol에 의하여 이루어진다⁽⁵⁾.

일반적으로 널리 알려진 Dennis의 dataflow 제어 프로그램은 여러 기능을 수행하는 actor (instruction)와 데이터를 나타내는 token(argument), token의 흐름을 표시하는 단일 방향의 arc로 구성되는 dataflow 프로그램 그래프로서 나타낸다. 이 dataflow 프로그램 그래프의 수행은 각 actor에 필요한 token이 입력되자마자 이루어지는 수행조건(firing rule)을 갖게 된다⁽²⁾.

3. FFT Butterfly 알고리즘

일련의 $2^n = N$ 점을 갖는 유한기간함수 $x(n)$ 의 DFT(discrete fourier transform)는

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k=0, 1, \dots, N-1 \quad (1)$$

$$W_N = \exp(-j2\pi/N)$$

이다. N 이 2의 거듭제곱 즉 $N=2^q$ 인 경우 식(1)은 다음과 같다.

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^k + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^k$$

$$= G(k) + W_N^k H(k) \quad (2)$$

이 된다. 식(2)에서 각 항의 계산은 $N/2$ 점 DFT와 같다. 이와같이 재분할을 계속하면 그림 2와 같은 2점 DFT(Butterfly)를 얻을 수 있으며, 완전한 FFT의 계산은 2점 DFT의 반복(iterative) 연산에 의해 이루어질 수 있다. m 번째 stage에서의 2점 DFT 식은,

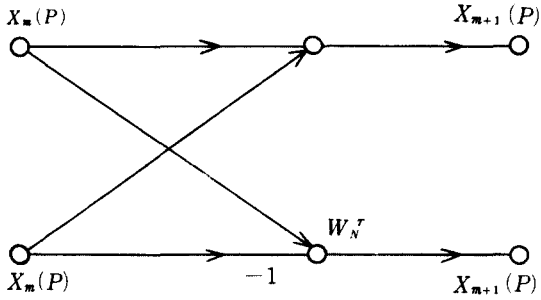


그림 2. Butterfly의 신호흐름선도
Flow graph of a Butterfly computation.

$$\begin{aligned} X_{m+1}(p) &= X_m(p) + W_N^r X_m(q) \\ X_{m+1}(q) &= X_m(p) - W_N^r X_m(q) \end{aligned} \quad (3)$$

으로 나타난다. 식(3)을 실수부와 허수부로 구별하여,

$$\begin{aligned} \text{Re}[X_m(p)] &= A, \quad \text{Im}[X_m(p)] = B, \\ \text{Re}[X_m(q)] &= C, \quad \text{Im}[X_m(q)] = D \\ \text{Re}[X_{m+1}(p)] &= A', \quad \text{Im}[X_{m+1}(p)] = B', \\ \text{Re}[X_{m+1}(q)] &= C', \quad \text{Im}[X_{m+1}(q)] = D' \end{aligned}$$

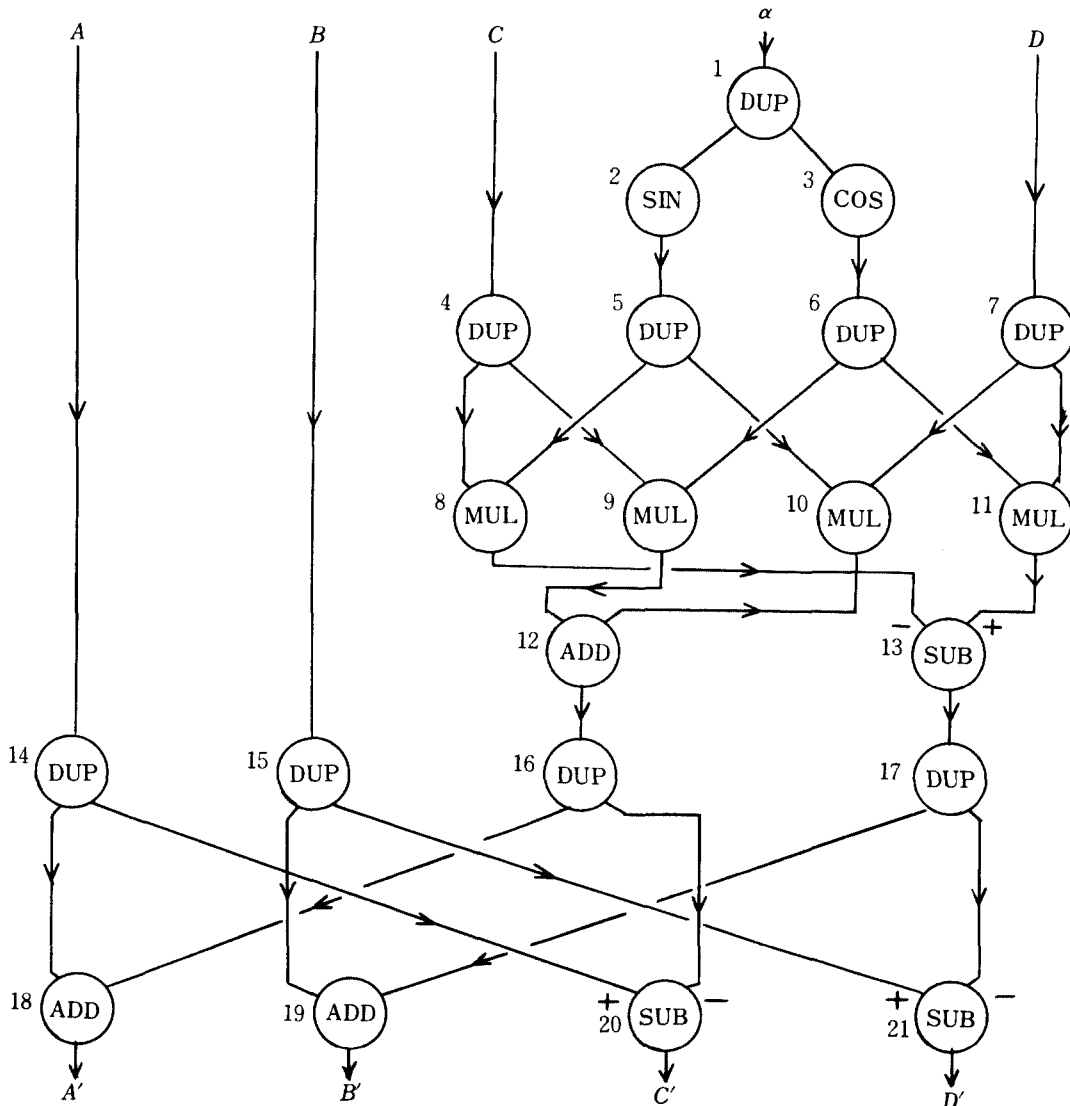


그림 3 Butterfly dataflow 프로그램 그래프,
Butterfly dataflow program graph.

$$R_e[W_N^r] = \cos \alpha, \quad I_m[W_N^r] = -\cos \alpha$$

라 놓으면,

$$\begin{aligned} A' &= A + C \cos \alpha + D \sin \alpha \\ B' &= B - C \sin \alpha + D \cos \alpha \\ C' &= A - C \cos \alpha - D \sin \alpha \\ D' &= B - D \cos \alpha + C \sin \alpha \end{aligned} \quad (4)$$

이 된다. 식(4)로 표시된 Butterfly 연산식을 dataflow 프로그램 그래프로 나타내면 그림 3 과 같다.

그림 3에 있는 프로그램 그래프는 ADD(addition), SUB(subtraction), MUL(multiplication), SIN(sine), COS(cosine), DUP(duplication)의 기능(function)을 수행하는 actor와 token, 단일 방향

표 1 명령어 cell
Initial instruction cell.

inst. cell	op code	operand 1			operand 2			destination 1		destination 2	
		perm	valid	value	perm	valid	value	addr.	port	addr.	port
1	DUP	0	1	α	0	1	0	2	1	3	1
2	SIN	0	0	0	0	1	0	5	1	0	0
3	COS	0	0	0	0	1	0	6	1	0	0
4	DUP	0	1	C	0	1	0	8	1	9	2
5	DUP	0	0	0	0	1	0	8	2	10	1
6	DUP	0	0	0	0	1	0	9	2	11	1
7	DUP	0	1	D	0	1	0	10	2	11	2
8	MUL	0	0	0	0	0	0	13	2	0	0
9	MUL	0	0	0	0	0	0	12	1	0	0
10	MUL	0	0	0	0	0	0	12	2	0	0
11	MUL	0	0	0	0	0	0	13	1	0	0
12	ADD	0	0	0	0	0	0	16	1	0	0
13	SUB	0	0	0	0	0	0	17	1	0	0
14	DUP	0	1	A	0	1	0	18	1	20	1
15	DUP	0	1	B	0	1	0	19	1	21	1
16	DUP	0	0	0	0	1	0	18	2	20	2
17	DUP	0	0	0	0	1	0	19	2	21	2
18	ADD	0	0	0	0	0	0	A'	1	0	0
19	ADD	0	0	0	0	0	0	B'	1	0	0
20	SUB	0	0	0	0	0	0	C'	1	0	0
21	SUB	0	0	0	0	0	0	D'	1	0	0

```

PROGRAM FFTdataflowSimulation;
BEGIN
  read << number of instruction cells & processors, instruction cells >>
  FOR processors := 1 TO number of processors
  DO BEGIN
    prog := save; ( save the initial state of instruction cells )
    running := true; ( if running is true, Program is executing )
    << initialize variables go(firable instr.), fetch(instr. fetch state),
      processor.ptime & paddr & pstate(end executiontime &
        executing instr.cell & processor executing state)>>
    WHILE running
    DO BEGIN
      FOR i := 1 to number of instruction cells
      BEGIN ( satisfy the firing rule )
        << check the firable state of instruction cell[i] >>
        IF << instruction cell[i] is firable >>
          THEN instruction cell[i].go := true;
      END;
      FOR i := 1 TO number of instruction cells
      BEGIN ( arbitration routing network )
        << check idle state of processors(exist executable processor) >>
        IF idle ( any one processor is idle )
          THEN BEGIN ( fetch the firable instr. cells & make op packets )
            WHILE lorun
            DO BEGIN
              FOR j := 1 to number of instruction cells
              with instruction cell[j]
              DO BEGIN
                IF go AND not fetch ( firable instruction )
                  THEN BEGIN
                    lorun := true;
                    k := j;
                    WHILE lorun DO WITH processor[k]
                    DO BEGIN
                      IF pstate ( k-th processor is busy )
                        THEN << jump next processor >>
                      ELSE BEGIN ( firable instruction fetch )
                        presenttime := presenttime + atime(
                          arbitration sw. time);
                        ptime := presenttime + executiontime;
                        paddr := j; ( firable inst. addr. )
                        pstate := true;
                        fetch := true; ( instr. cell[j] fetched )
                        lorun := false; ( jump next instr. cell )
                      END; ( ELSE ) END; ( WHILE ) END; ( IF go )
                    END; ( FOR ) END; ( WHILE ) END; ( IF idle )
                  END;
                << select the least ptime & paddr of processor >>
                << execution of least fetchtime processor & make data packets >>
                presenttime := least ptime;
                << reset instr. cell[execute].fetch & processor[execute].pstate >>
                presnttime := presnttime + dtime(distribution sw. time);
              BEGIN ( distribution routing network )
                FOR j := 1 to hiarc ( max. number of output arc )
                  << store result value & set valid flag in destination >>
                FOR j := 1 to maxin ( max. number of input arc )
                  DO BEGIN ( consume input tokens of executed actor )
                    IF instruction cell[execution].operand[j].peram = 0
                      THEN instruction cell[execution].operand[j].full := 0;
                  END;
                << IF all instruction cells executed, THEN running := False >>
                << select max ptime (program execution time) &
                  write the execution time, and result value >>
              END; ( WHILE running ) END; ( FOR processors := ? )
            END.
  
```

그림 4 시뮬레이션 알고리즘.
Simulation algorithm.

의 arc로 구성되어 있다. 이 프로그램이 각actor에 필요한 token들이 입력되자마자 수행될 수 있는 이상적인 기계에서 수행된다고 가정할 때, token A, B, C, D, α가 actor I₁₄, I₁₅, I₄, I₇, I₁에 각각 도달되면 I₁₄, I₁₅, I₄, I₇, I₁은 독립적으로 DUP를 병렬로 수행하게 된다. I₁에서의 출력 token은 I₂, I₃의 입력 token이 되므로 I₂, I₃는 각각 SIN, COS 연산을 수행하여 I₅, I₆의 입력 token을 공급하게 되며 I₅, I₆에서 DUP를 각각 수행하여 출력 token을 내보면 I₈, I₉, I₁₀, I₁₁은 필요한 입력 token이 모두 존재하므로 병렬로 MUL을 수행하게 된다. 같은 방법으로 I₁₂, I₁₃과 I₁₆, I₁₇이 수행된 다음에 I₁₈, I₁₉, I₂₀, I₂₁에 의하여 원하는 결과값인 A', B', C', D'를 얻을 수 있다.

MIT dataflow 컴퓨터 구조로 이와같은 프로그램 그래프를 수행하려면 명령어 cell에 표 1과 같이 명령어가 저장되어야 한다. 여기서 오퍼런드의 valid는 value가 있는 상태를 표시하고 perm은 수행된 후에 오퍼런드가 리셋되는 상태를 표시한다.

4. 성능해석 및 시뮬레이션 결과

Dataflow 컴퓨터 구조로 그림 3의 프로그램 그래프를 PE의 수에 따라 수행시켜 성능을 살펴보자. 컴퓨터의 성능은 프로그램을 수행하는 시간에 따라 결정될 수 있으므로, 프로그램 수행시간을 계산하기 위하여 다음값을 정의한다.

- p : 연산처리장치(PE)의 수
- α : 연산처리장치의 수행확률
- n : 수행될 actor의 총수
- t_i : i번째 actor의 수행시간
- t_a : arbitration network의 스위칭시간
- t_d : distribution network의 스위칭시간

이상과 같은 정의에 의해 프로그램의 수행시간 T_{ex}를 구해보자. 프로그램의 수행시간은 n개의 명령어를 기억장치에서 access하여 연산처리장치로 보내는 시간, 연산처리장치의 수행시간, 연산처리된 결과를 기억장치에 저장하는 시간의합

이 되므로 T_{ex}는 다음과 같이 표현된다.

$$T_{ex} = n(t_a + t_d) + \frac{\sum_{i=1}^n t_{i_i}}{\alpha \cdot P} \quad (5)$$

식(5)의 T_{ex}를 구하는 dataflow 연산 알고리즘은 시간가속법(time acceleration method)으로 시뮬레이션 하였으며⁽⁸⁾ PE(processing element)의 수가 1개인 경우부터 차례로 증가시키면서 수행하는 알고리즘은 그림 4와 같다. 그림 3에 있는 프로그램 그래프를 시뮬레이션 프로그램으로 수행시키기 위하여 각 actor들의 수행시간은, 비트슬라이스 마이크로 프로세서 AM2903의 마이크로 사이클 시간을 기준으로 하여⁽¹⁾,

$$t_f(\text{ADD}) = 184 \text{ ns}, \quad t_f(\text{SUB}) = 184 \text{ ns}, \\ t_f(\text{MUL}) = 200 \text{ ns}, \quad t_f(\text{DUP}) = 143 \text{ ns}$$

로 정하였으며, 고속 sine, cosine generator인 AM29526의 스위칭 시간을 기준으로 하여⁽¹⁾

$$t_f(\text{COS}) = 95 \text{ ns}, \quad t_f(\text{SIN}) = 95 \text{ ns}$$

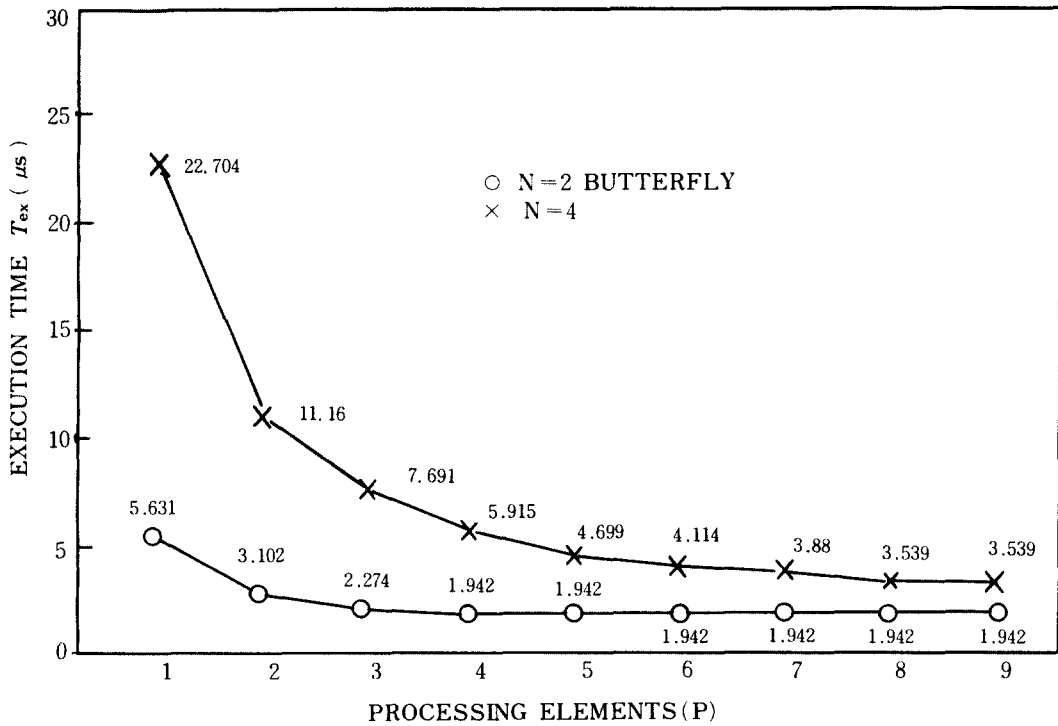
로 정하였다. 또한 arbitration/distribution network은 이상적인 routing network이라 가정하여 쌍방향 I/O ports with handshake 소자인 AM2950의 전달지연시간을 기준으로 하여⁽¹⁾

$$t_a = 50 \text{ ns}, \quad t_d = 50 \text{ ns}$$

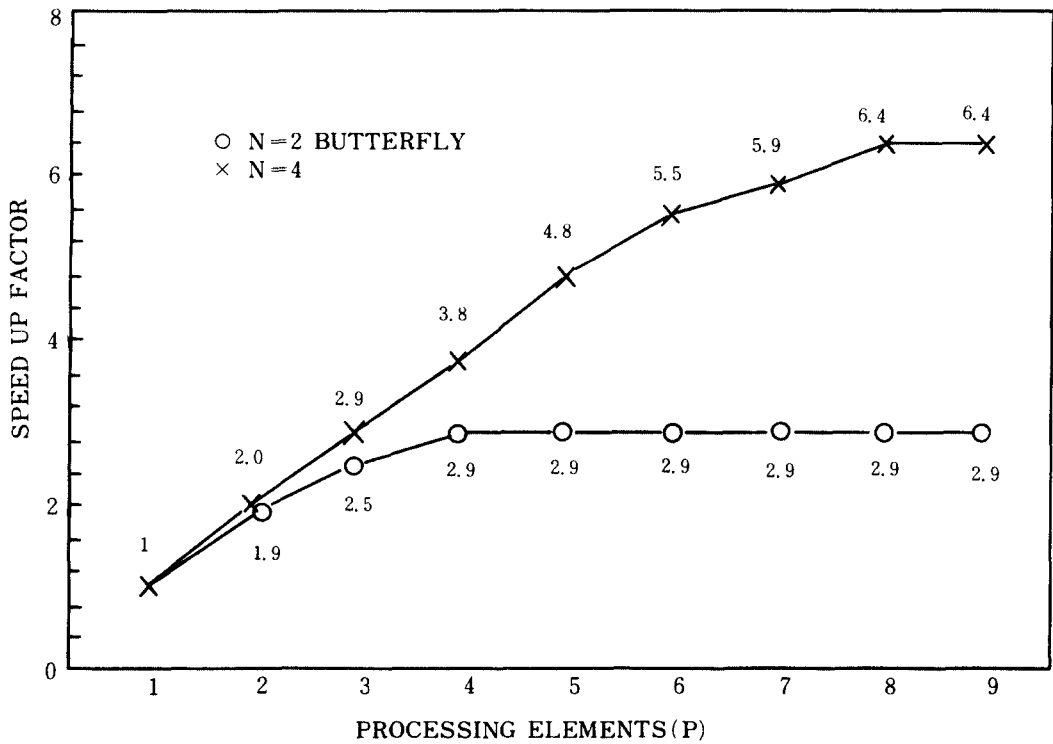
로 정하였다. Butterfly와 샘플점이 N=4인 경우에 연산처리장치의 수 P를 변화시켜가며 수행시킨 결과는 그림 5 (a)와 같다. 또한 P=1인 경우에 대한 수행시간 속도증가비 S(speed up factor)는,

$$S = \frac{\{T_{ex}\}_{P=1}}{\{T_{ex}\}_{P=k}}, \quad k = 1, 2, 3, \dots \quad (6)$$

에 의하여 그림 5 (b)와 같고, P의 변화에 따르는 최적의 수행시간 속도비는 Butterfly의 경우에는 P=4일 때 S=2.7이 되고, N=4인 경우에는 P=8일 때 S=6.4가 됨을 알 수 있다. 여기에서 샘플점이 N개인 경우에는 P=2N일 때



(a)



(b)

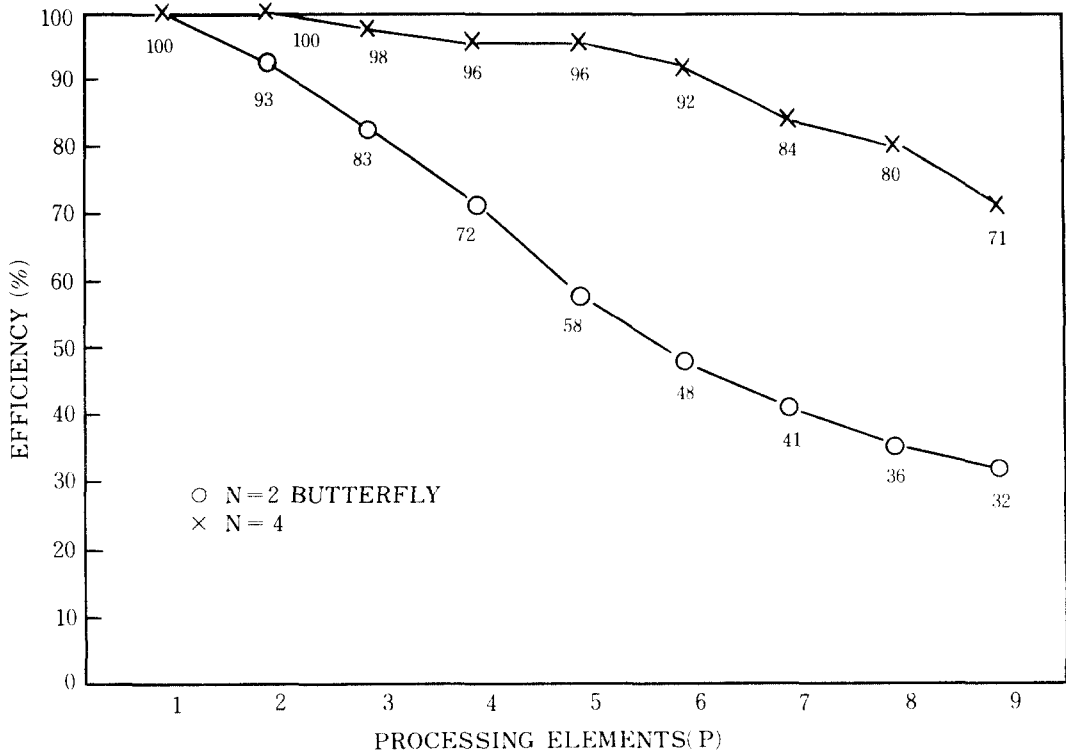


그림 5 (a) P의 변화에 따른 수행시간 (b) P의 변화에 따른 속도증가비 (c) 처리장치의 실효이용률

속도증가비 S 는 $2.7 \times \frac{N}{2}$ 보다 크게 증가되어 고도의 병렬성이 유지될 수 있음을 보였다. 또한 출력값을 구하는데 이용된 연산처리장치의 실효이용률 e 는

$$e = \frac{\sum_{i=1}^n (t_a + t_N + t_a)}{(\text{프로그램수행시간}) \times (\text{PE의 수})} \times 100 \quad (7)$$

로 정의할 수 있으며 이를 구하면 그림 5(c)와 같다. 따라서 수행해야할 actor가 많을 때, 즉 샘플점 N 이 크게 되면 실효이용률이 증가하므로 연산속도가 개선되어 dataflow 연산이 FFT와 같이 실시간 처리가 요구되는 연산에 적합함을 알 수 있다.

5. 결 론

Dataflow 컴퓨터는 프로그램이 고도의 병렬성

을 갖고 수행될 수 있어 von-Neumann 기계 이상으로 계산처리능력을 향상시키게 된다. 이 구조의 기계어 프로그램은 dataflow 프로그램 그래프에 의해 데이터를 병렬로 처리하므로서 대규모 연산이나 실시간 처리에 적합하다.

본 연구에서는 실시간 처리가 요구되는 FFT Butterfly 알고리즘을 dataflow 프로그램 그래프로 구성하였으며, 이 알고리즘을 dataflow 시뮬레이션 프로그램으로 실현하여 그 성능을 연산처리장치의 수에 따라 비교하였다. 즉 연산처리장치의 수가 한개인 경우에 비해 네개인 경우는 프로그램 수행시간 속도증가비가 2.7이 됨을 알 수 있었다. 또한 FFT 알고리즘의 실현에서 샘플점이 많은 경우에는 연산속도가 더욱 개선되어 dataflow 연산이 실시간 처리에 적합함을 알 수 있다.

본 논문은 1985년도 문교부
학술연구조성비에 의하여
연구 되었음.

참 고 문 헌

(1) Arbind & R.A. Iannucci, "A Critique of multiprocessing von-Neumann style," Proc. ACM SIGPLAN 1983 conf., pp. 426-436, Apr. 1983.
 (2) J. B. Dennis, "Data-flow suppercomputers," Computer 13 11 pp. 48-56, Nov. 1980.
 (3) P. C. Treleaven, et al. "Data-driven and demand-driven computer architecture," ACM Computing Surveys, vol. 14, no. 1, pp. 93-143, Mar. 1982.
 (4) A. L. Davis & M. Keller, "Dataflow program graph,"

IEEE Computer, pp. 26-41, Feb. 1982.
 (5) K. Hwang & F. A. Briggs, "Computer architecture and Parallel Processing," McGraw Hill, pp. 732-787, 1984.
 (6) I. Watson & J. Gurd, "A prototype dataflow computer with token labelling," AFIPS con. proc. NCC. N.Y., pp. 623-628, June 1979.
 (7) B. Gold & T. Bially, "Parallelism in Fast Fourier Transform hardware," IEEE Trans. on Audio and Elec., vol. AU-21, no. 1, Feb. 1973.
 (8) R. E. Bryant, "Simulation on a distribution system," CSG Memo 182 Lab. for Computer Science, MIT July 1979.
 (9) R. E. Bryant, "Simulation of packet communication architecture computer systems," MIT LCS TR-188, Nov. 1977.
 (10) B. P. Ziegler, "Theory of modelling and simulation," N. Y. Willey Int., 1976.
 (11) Am2900 Family Data Book, Bipolar Microprocessor Logic and Interface, AMD Inc, pp. 5-32~5-71, 5-321~5-335, 7-43~7-47, 1983.



李相範 (Sang Burm RHEE) 正會員
 1951年 2月 3日生
 1970年 3月~1974年 2月 : 연세대학교 공과대학 전자공학과 (공학사)
 1976年 9月~1978年 8月 : 서울대학교 대학원 전자공학과 (공학석사)
 1981年 3月~현재 : 연세대학교 대학원 전자공학과 (박사과정)
 1978年 9月~1979年 8月 : 아주공과대학 전자공학과 조교
 1983年 10月~1984年 12月 : 미국 Iowa 대학 객원교수
 1979年 9月~현재 : 단국대학교 공과대학 전자공학과 조교수



朴贊政 (Chan Jung PARK) 正會員
 1947年 1月 14日生
 1966年 3月~1970年 2月 : 서울대학교 사범대학 수학교육과 (이학사)
 1982年 3月~1984年 2月 : 단국대학교 대학원 전자공학과 (공학석사)
 1984年 3月~현재 : 단국대학교 대학원 전자공학과 (박사과정)