論文 91-28B-5-4

# 다중 프로세서 시스템을 이용한 디지탈 필터링 알고리듬의 효율적 구현

# (An Efficient Multiprocessor Implementation of Digital Filtering Algorithms)

成 元 鎔*

(Won Yong Sung)

**要 約**

다중프로세서 시스템을 이용하여 디지탈 필터링 알고리즘을 효율적으로 실시간 구현하는 방법이 연구되었다. 디지탈 시그날 프로세서를 이용하여 알고리듬을 소프트웨어로 구현하므로, 고속의 복잡한 신호처리 시스템을 빨리 개발할 수 있다. 각 프로세서간의 상호연결망을 간략화하고 통신회수를 줄이기 위하여서, 입력 데이타를 블럭(block)으로 나눈후 각 프로세서에 할당하여 동시에 처리하게하는 병렬 블럭처리 방법을 이용하였다. 이 시스템에서 프로세서들의 운영(scheduling)은 동시에 처리되는 블럭간의 의존관계에 의해 제약되는데, 이 관계를 dependence graph를 이용하여 쉽게 분석하는 방법이 개발되었다. 그리고 프로세서간의 의존시간을 줄여서 대형 다중프로세서(multiprocessor) 시스템을 구현할 수 있는 방법이 연구되었다. FIR, 순환(recursive), 그리고 적응(adaptive) 필터들을 구현하는 방법과 결과가 제시되었다.

**Abstract**

An efficient real-time implementation of digital filtering algorithms using a multiprocessor system in a ring network is investigated. The development time and cost for implementing a high speed signal processing system can be considerably reduced because algorithms are implemented in software using commercially available digital signal processors. This method is based on a parallel block processing approach, where a continuously supplied input data is divided into blocks, and the blocks are processed concurrently by being assigned to each processor in the system. This approach not only requires a simple interconnection network but also reduces the number of communications among the processors very much. The data dependency of the blocks to be processed concurrently brings on dependency problems between the processors in the system. A systematic scheduling method has been developed by using a dependence graph for the analysis of the dependency relation. To increase the number of processors which can be used efficiently, the methods for solving dependency problems between the processors are investigated. Implementation procedures and results for FIR, recursive (IIR), and adaptive filtering algorithms are illustrated.

*正會員, 서울大學校 半導體共同研究所 및 制御計測工學科

(ISRC and Dept. of Cont. & Instr., Seoul Nat'l Univ.)

接受日字 : 1991年 2月 6日

## I. 서　론

단일 칩(chip) 디지탈 시그날 프로세서가 값싸게 공급됨에 따라 다중 프로세서 시스템을 이용하여 고속 신호처리 시스템을 구현하는 방법이 점점 유망 해지고 있다.[1][4] 이 방법은 디지탈 시그날 프로세서 개발 시스템들이 제공되고 있고, 또 알고리듬이 소프트웨어로 구현되므로, 전용 VLSI를 이용하여 개발하는 방법보다 시간과 비용의 면에서 유리하다. 그런데 디지탈 시그날 프로세서는 프로그램과 데이타용으로 비교적 큰 양의 메모리를 내장할 수 있는 이점이 있으나, 프로세서간의 통신이 프로그램에 의해 제어되므로 비 동기적(asynchronous) 이고 상당히 느리다는 약점이 있다. 따라서 어떤 알고리듬을 몇개의 부기능(sub-function)으로 나눈후, 각각을 디지탈 시그날프로세서로 구현하는 방법으로는 통신시간의 과다한 overhead 때문에 효율적인 시스템 구성이 어려운 경우가 많다. 따라서 하나의 디지탈 시그날 프로세서 내에 한 블럭의 데이타를 충분히 저장할 수 있다는 장점을 이용하여 프로세서간의 통신 회수가 적은 새로운 구현 방법을 개발할 필요가 있다. 그리고 FFT (Fast Fourier Transform)나 디지탈 필터링과 같이 상이한 구조(structure)를 가지는 여러가지의 디지탈 신호처리 알고리듬을 하나의 다중프로세서 시스템내에 함께 구현하는 것이 가능하다면 시스템의 쓸모가 더욱 클 것이다.

이와 같은 필요성 때문에 시스템의 구조(architecture) 및 운영법(scheduling)의 설계에 병렬 블럭처리(parallel block processing) 방법이 이용되었다. 이 방법은 연속된 입력데이타를 일정길이의 블럭(block)으로 나눈 후, 이들을 각각의 프로세서에 할당하여 처리하게 하는 방법이다. 즉, 제안된 시스템은 그림 1에 보이는 바와같이 모든 프로세서들이 A/D나 D/A 콘버터와 같은 I/O에 병렬로 연결되고, 또 각 프로세서들이 이웃한 프로세서들과 환상(ring) 연결망으로 연결되는 비교적 간단한 형태이다. I/O가 모든 프로세서에 병렬로 연결되어 있기 때문에, 각 프로세서들은 할당된 블럭의 데이타를 차례로 입·출력 버스를 통하여서 직접 보내고 받는다. 그리고 이 시스템에서 각 프로세서는 주어진 블럭의 처리를 모두 담당하기 때문에 중간 결과가 프로세서 상호간을 이동하지는 않는다. 단지 초기조건과 같은 블럭 경계면의 값을 주고 받기 위하여서 상호 연결망이 있고, 따라서 통신회수가 대단히 적어진다. 어떤 프로세서에 데이타를 주는 프로세서를 선처리기(pre-processor), 유사하게 데이타를 받는 프로세서를 후처리기(post-processor)라고 부를 것이다. 예를들면 그림 1에서 프로세서 #P는 프로세서 #1의 선처리기(preprocessor)이며, 프로세서 #2는 프로세서 #1의 후처리기(postprocessor)이다.

그림 2에 이 시스템에서 각 프로세서들이 운영되는 시간표(timing diagram)의 예가 나타나 있다. 즉 프로세서 #1이 시간 0에서 첫번째 블럭의 처리를 시작하였다고 가정하면, 프로세서 #2는 약간의 딜림 시간후, 즉 시간 $T_{SK}$(skew time)에 두번째 블럭의 처리를 시작한다. 마찬가지 방법으로 프로세서 #P는 그 블럭의 처리를 시간 $(P-1)T_{SK}$에서 시작한다. 한편, 뒤 따라서 시간 $P \cdot T_{SK}$에서 프로세서 #1은 다시 $(P+1)$번째 블럭의 처리를 시작하여야 한다. 비록 각

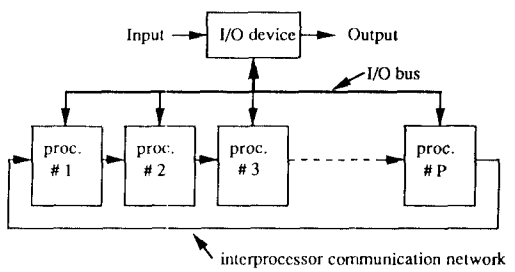


그림 1. 제안된 다중프로세서 시스템의 구조
Fig. 1. The proposed multiprocessor architecture in a ring network.

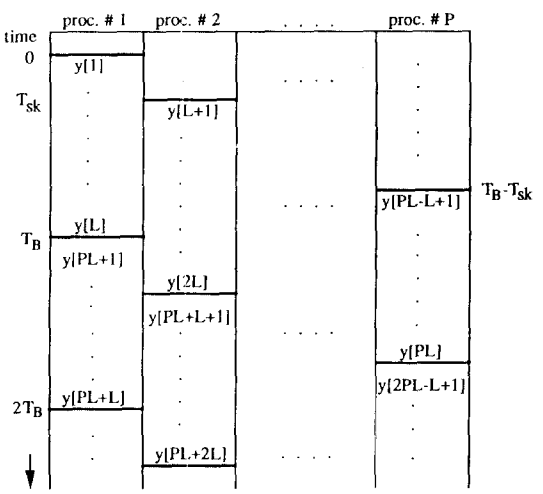


그림 2. 병렬 블럭처리 방식에서 각 프로세서의 운용시간표
Fig. 2. Scheduling of processors in the parallel block processing scheme.

각의 블럭을 맡겨진 프로세서를 이용하여 독립적으로 처리할 수 있다면 가장 바람직하겠으나, 각 블럭을 처리하는데 앞의 블럭으로 부터 초기조건 등의 데이타를 받아야 하는데, 이것이 프로세서간에 의존관계(dependency relation)를 만든다. 따라서 각 프로세서의 운영(scheduling)은 프로세서간의 의존관계에 의하여 지배를 받게된다. 그림 2의 운영시간표에서 보는 바와 같이, 하나의 데이타 블럭을 어떤 프로세서가 처리하도록 주어지는 시간 $T_B$는 $P \cdot T_{SK}$가 된다. 따라서 처리속도(throughput)를 올리기 위해서 프로세서의 수 P를 증가시키면, 프로세서 사이의 밀림 시간 $T_{SK}$가 줄어든다. 한편 각 블럭의 data를 처리하는데는 선처리기(preprocessor)에서 넘어오는 초기조건 등을 필요로 하는 경우가 많은데, 밀림 시간 $T_{SK}$가 적어지면 이 초기조건을 미처 받지 못하게 된다. 따라서 $T_{SK}$의 최소값 $T_{SK,min}$은 프로세서간의 의존시간 $T_d$에 의해 제한 되므로, 이용될 수 있는 최대 프로세서의 수 $P_{max}$는 그림 2에서 다음과 같이 유도된다.

$$P_{max} = \left\lfloor \frac{T_B}{T_d} \right\rfloor \qquad (1)$$

단 $\lfloor x \rfloor$는 x보다는 작거나 같은 최대 정수를 나타낸다. 각 블럭간, 또는 이들을 처리하는 각 프로세서 사이의 의존관계를 나타내기 위하여 dependence graph가 사용되었다. 프로세서의 갯수가 증가하면서 dependence graph는 더욱 복잡해 지지만 실시간 디지탈 신호처리 알고리듬의 시간이동불변(shift invariant)특성을 이용하여 단순화시킬수 있기 때문에 개발된 다중프로세서 시스템의 운영(scheduling) 방법은 단일 프로세서의 경우보다 크게 더 복잡하지 않다.

식 (1)에 보는 바와 같이 이 시스템에 사용될 수있는 프로세서의 수는 각 블록간의 의존시간에 의해 제한되기 때문에, 일부의 알고리즘은 의존시간을 줄이지 않으면 필요로 하는 수 만큼의 프로세서를 사용할 수 없다. 따라서 두가지의 의존시간을 줄이는 방법이 개발되었는데, 하나는 분해(decomposition) 방법이고 다른 하나는 전방예측(look-ahead) 방법이다. 분해(decomposition) 방법은 복잡한 알고리듬을 몇개의 작은 알고리즘으로 분해하여서 전체의 의존시간을 줄이는 방법이다. 전방예측(look-ahead) 방법은 독립된 계산식을 사용하여 다음 블럭의 초기 조건을 미리 계산해내는 방법이다.

본 논문의 개요는 다음과 같다. 제 2절에서 단일 프로세서 시스템과 멀티프로세서 시스템의 실행시간 모형이 제시되었고, 제 3절에서 프로세서 사이의 의존관계를 고려한 운영 방법이 개발되었고, 제 4절에서는 의존시간을 줄이는 방법이 연구되었다. 제 5절에서는 연구된 방법들을 이용하여 FIR, 순환(recursive) 및 적응(adaptive) 필터의 구현방법과 성능등이 제시되었다. 제 6절에서는 부동소수점(floating-point) 디지탈 시그날 프로세서 AT & T DSP32를이용한 하드웨어 구현 내용이 설명되었다.

## Ⅱ. 실행시간 모형과 성능 척도

이 장에서는 개발된 방법의 정량적 평가를 위하여 시스템의 실행시간에 대한 모형을 제시하였다. 어떤 알고리듬을 디지탈 시그날 프로세서로 구현할 때, 그 실행시간은 크게 세부분-즉 산술(arithmetic), 입·출력(I/O) 및 프로세서간의 통신(interprocessor communication) 시간으로 나눌 수 있다. 각 동작에 대한 단위시간을 각각 $C_a, C_{io}, C_c$라고 표기한다. 여기서 $C_a$, 즉 산술연산을 위한 단위시간은 단순히 연산회로의 cycle time을 의미하지 않고 하나의 산술 연산에 수반되는 데이타 가져오기나 제어명령과 같은 모든 부가시간들을 포함한다. 한개의 입력 sample을 처리하기 위한 산술, 입·출력, 통신 횟수를 각각 $N_a, N_{io}, N_c$로 표기한다. 그러면 한 샘플당의 실행시간은 다음과 같이 표시된다.

$$t = \frac{T_B}{L} = C_a N_a + C_{io} N_{io} + C_c N_c \qquad (2)$$

여기서, $T_B$는 한 블럭의 처리시간이고, L은 한 블럭의 길이이다.

위의 실행시간 모형은 다중프로세서(multiprocessor) 시스템의 경우로 쉽게 확장할 수 있다. 우선 다중프로세서(multiprocessor) 시스템 내의 각 프로세서와 어떤 단일처리 시스템의 프로세서가 같은 값의 $C_a$와 $C_{io}$를 갖는다고 가정하자. 그러면 그 단일처리 시스템과 다중 프로세서(multiprocessor) 시스템에서의 알고리즘의 수행시간은 식(3)처럼 표시된다. 여기서 첨자 "s"는 단일처리 시스템을 "m"은 다중 프로세서(multiprocessor) 시스템을 나타낸다. 예를 들어, $N_{a,s}$는 단일처리 시스템에서의 산술 연산 갯수를 나타내며, $N_{a,m}$은 다중 프로세서(multiprocessor) 시스템용 병렬 처리 알고리듬에서의 매 샘플당 연산 갯수를 표시한다. 많은 경우 병렬 처리 알고리듬은 단일 프로세서 시스템용의 순차(sequential) 알고리듬에 비해, 비록 다중처리를 하므로 시간은 짧아질지라도, 전체적음로는 더 많은 산술 연산 회수를 필요로 한다. 결과적으로 $N_{a,m}$은 $N_{a,s}$보다 클 수 있다. 두 실행시간에 대한 수식은

$$t_s = C_a \cdot N_{a,s} + C_{io} \cdot N_{io,s}$$
$$t_m = C_a \cdot N_{a,m} + C_{io} \cdot N_{io,m} + C_c \cdot N_{c,m} \quad (3)$$

이다. 위의 식에서 $t_m$의 표현에는 아직 여러개의 프로세서를 사용한 시간절감의 효과는 고려되지 않았다. 단일프로세서 시스템의 경우 실행시간은 보통 연산 처리시간에 의해 좌우되며 프로세서 사이의 통신시간은 무관하다. 그러나 다중 프로세서(multiprocessor) 시스템에서는 프로세서간의 통신시간이 중요해 질 수 있다.

본 다중 프로세서(multiprocessor) 시스템에서의 구현방법을 평가하기 위하여 세가지 성능척도, 즉 일정시간내의 처리속도(throughput), 속도개선(speed-up) 효율(efficiency)이 사용되었다.[5,6] 제안된 구현 방식에서는 만일 시스템 구현에 사용된 프로세서의 갯수가 식(1)에서 정의된 $P_{max}$보다 크지 않다면, 모든 프로세서들을 100% 사용할 수 있다. 이 경우 처리속도(throughput)는 샘플당 실행시간을 프로세서 갯수로 나눈 수의 역수가 된다. 따라서, P개의 프로세서를 사용했을 경우의 처리속도(throughput) R(P)는 다음 식과 같이 표시할 수 있다.

$$R(P) = \frac{P}{C_a N_{a,m} + C_{io} N_{io,m} + C_c N_{c,m}} = \frac{PL}{T_B} \quad (4)$$

속도 개선(speed-up)은 다중 프로세서 시스템의 처리속도(throughput)와 기준이 되는 단일 프로세서 시스템의 처리속도(throughput)의 비로 정의한다. 그러므로 다중 프로세서(multiprocessor) 시스템을 이용한 속도 개선은 다음과 같다.

$$S(P) = \frac{P(C_a N_{a,s} + C_{io} N_{io,s})}{C_a N_{a,m} + C_{io} N_{io,m} + C_c N_{c,m}} \quad (5)$$

전체적인 시스템 효율은 프로세서 갯수에 대한 속도 개선의 비로 정의된다.

$$E(P) = \frac{C_a N_{a,s} + C_{io} N_{io,s}}{C_a N_{a,m} + C_{io} N_{io,m} + C_c N_{c,m}} \quad (6)$$

위 식은 통신 시간의 절감이 다중 처리 시스템의 효율이나 속도개선을 위해서 실제적으로 필요함을 보여준다.

## Ⅲ. 시스템의 운영 (scheduling)

본 시스템내에서 각 프로세서는 동일한 프로그램을 내장하고 있고, 또 주어진 데이타 블럭에 대하여 같은 동작을 실행하므로 비록 운영시간이 그림 3에서 보이는 바와 같이 조금씩 뒤로 밀려 있기는 하지만, 이 시스템은 Skewed Single Instruction Multiple Data(SSIMD) 형의 컴퓨터에 속한다.[7] 프로세서의 운영은 입·출력 동작과 프로세서 사이의 통신 동작에 기인하는 의존관계에 의하여 제한이 된다. 즉 각 프로세서들은 공통의 입·출력 장치를 공유하고 있으므로 각 프로세서의 입·출력은 반드시 한 프로세서씩 순차적이어야 하는데, 이 때문에 입·출력에 의한 의존관계가 발생한다. 또한 각 프로세서는 주어진 블럭을 처리하기 위하여 먼저번 블럭에서 초기조건을 받아야하는데, 이경우 선처리기(preprocessor)가 초기조건을 보내기 전까지는 받을 수 없으므로 프로세서 사이의 통신 동작 또한 의존 관계를 만든다. 대부분의 신호처리 알고리듬은 여러개의통신동작을 필요로하므로, 복수의 의존관계가 발생한다. 따라서 가장 중요한 의존관계(critical dependency relation)를 찾아내는 방법이 필요한데 이를 위하여 dependence graph가 사용되었다.[8]


그림 3. $PR_p(i)$가 $PR_q(j)$에 의존적임을 나타내는 의존표

Fig. 3. A precedence graph showing that $PR_p(i)$ is dependent on $PR_q(j)$.

만일, p번째 프로세서의 i번째 procedure $PR_p(i)$가 q번째 프로세서의 j번째 procedure $PR_q(i)$에 의존한다면, dependence graph는그림 3과 같이 표시된다. 각 점(node)은 한 procedure의 실행을 나타내며, 각 가지(branch)는 화살표 방향으로의 의존관계를나타낸다. 따라서 그림 3에서 $PR_p(i)$, $PR_q(\bar{o})$가 완료되지는 실행될 수 없다.

어떤 알고리듬을 구성하는 procedure들이 그림 4와 같은 경우의 dependence graph를 생각해 보자. 이 알고리듬에서 각 프로세서는 procedure #1에서 한 블럭의 입력 데이타를 받아 초기 조건에 무관한 중간결과 z[j]를 procedure #2에서 계산한다. 그리고 procedure #3에서초기조건 $S_{p-1}[j]$를 선처리기(preprocessor)로 부터 받고, 다음번 블럭의 초기조건 $S_p[j]$를 procedure #4에서 계산하여, procedure #5에서 후처리기(postprocessor)로 보낸다. procedure #6에서는 출력 데이타를 계산하여, 출력장치로 procedure #7에서 보낸다.

이 알고리듬의 dependence graph가 그림 5에 나타나 있는데, 프로세서의 수에 비례해서 복잡도가 증가하기 때문에 분석하기가 어렵다. 그런데 각 세로줄의 procedure들은 서로다른 프로세서에서 수행되

Procedure (1): x[1], . . . , x[L] 을 I/O 에서 받는다.
Procedure (2): 중간결과 z[i] = $f_z$(x[1], · · ,x[L]), for i = 1 to L, 를 계산한다.
Procedure (3): 초기조건 $S_{p-1}$[j], for j = 1 to M, 를 받는다.
Procedure (4): 후처리기(post processor) 를 위한 초기조건 $S_p$[j] = $f_s$(x,z,$S_{p-1}$), for j = 1 to M, 을 계산한다.
Procedure (5): $S_p$[j], for j = 1 to M, 를 후처리기로 보낸다.
Procedure (6): 출력 y[i] = $f_y$(x,z,$S_{p-1}$), for i = 1 to L, 을 계산한다.
Procedure (7): 출력 y[i], i = 1 to L, 을 I/O 에 보낸다.

그림 4. 간단한 신호처리 알고리듬에서의 procedure 의 예
Fig. 4. An example of procedures for a signal processing algorithm.

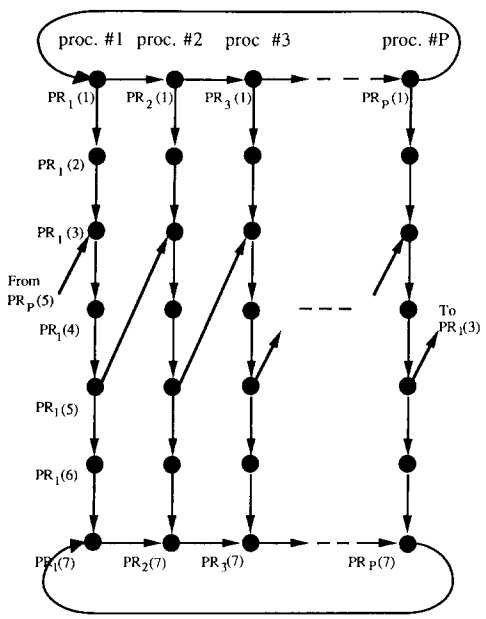


그림 5. 제안된 다중프로세서 시스템에서의 의존표의 예
Fig. 5. A multiprocessor precedence graph.

지만 모두 같은 것 들이다., 예를들면, $PR_p$(i)와 $PR_q$(i)는 동일한 procedure PR(i)이다. 디지탈 신호처리 알고리즘의 시간이동불변(shift invariance)특성이 이들을 동일한 procedure가 되게한다.[7,9] 따라서 이러한 성질을 이용하면 그림 5의 dependence graph를 대폭적으로 간략화시킬 수 있다. 즉 프로세서 하나의 procedure만 남기는 대신 $PR_{p-1}$(i)에서 $PR_p$(j)로 가는 dependence graph를 PR(i)에서 PR(j)로 가는 dependence graph로 바꾼다. 이 결과로 간략화된 dependence graph가 그림 6에 보이는데, 여기서 의존관계는 궤환 고리(feedback loop)로 표시됨을 알 수 있다. 이 결과로 본 멀티프로세서 시스템의 운영(scheduling)이 단일 프로세서 시스템의 운영처럼 간략화 되었음을 알 수 있다. 그림 6에서 procedure #1과 #7의 궤환고리는 I/O의 공유때문에 생기는 의존관계를 나타내고, procedure #3,4,5 의 궤환고리는 프로세서간의 통신에 기인하는 의존관계를 나타낸다.

입력 데이타가 샘플링 주파수 R의 비율로 공급된다면, 길이가 L인 한 블럭의 데이타를 모으는데 걸리는 시간은 L/R이고 따라서 p번째 프로세서는 (p-1)번째 프로세서보다 그만한 시간만큼 늦게 입력 데이타를 받게된다. 그렇기 때문에 입력이나 출력, 즉 그림 6의 loop #1과 #7의 loop delay는 L/R이 된다. 한편 프로세서간의 통신에 기인하는 loop의 경우, loop delay는 loop내의 모든 procedure가 요구하는 시간의 합이며, 그것은 간략화된 dependence graph 로부터 쉽게 계산가능하다. 이때 가장 긴 loop delay를 갖는 것이 critical loop가 되고, 프로세서간의 의존

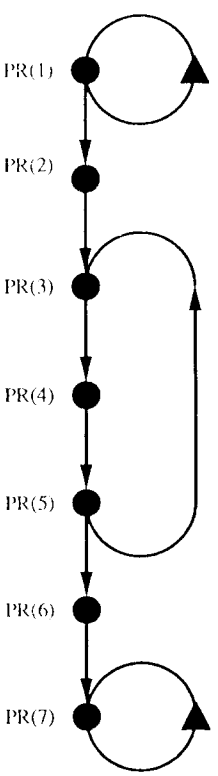


그림 6. 간략화된 의존표
Fig. 6. A reduced precedence graph.

시간 $T_d$를 결정한다.

그림 4의 알고리듬과 그림 6의 dependence graph를 이용하여 두가지 설계 예에서 시스템의 운용을 살펴보자. Procedure i의 수행에 필요한 시간을 T(i)라고 표기한다.

경우 1

여기서는 어떤 처리속도(throughput) R을 얻을 수 있는 시스템을 구현하고자 할때, 그때 필요한 프로세서의 최소갯수 $P_{mim}$과 이때 필요한 프로세서 사이의 밀림시간 $T_{sk}$를 찾는 것이다. 이경우 다음 관계식을 쉽게 세울 수 있다.

(i) 입력데이타의 공급속도가 R이기 때문에, 각 프로세서들은 밀림시간 $T_{sk}$동안에 한 블럭의 데이타를 저장할 수 있어야 한다. 이는 procedure #1이 critical loop임을 의미한다. 따라서

$$T_{sk}=\frac{L}{R} \tag{7}$$

(ii) 다른 loop들의 지연은 위의 밀림시간($T_{sk}$)보다 작거나 같아야 한다. 따라서 다음식이 성립한다.

$$T(3)+T(4)+T(5)\leqq T_{sk} \tag{8}$$

때로 두 프로세서 사이에서 초기조건을 보내는 시간인 procedure #5가 그것을 받는 시간인 procedure #3과 겹쳐서 지연이 짧아질 수도 있으나, 최대의 지연을 생각하면 시간을 위식과 같이 더해야 한다.

(iii) 주어진 처리속도(throughput) R을 얻기위해 요구되는 최소 프로세서의 수 $P_{min}$는 다음의 식에 의해 결정된다.

$$P_{mim}=\left\lceil \frac{T_B}{T_{sk}} \right\rceil = \left\lceil \frac{T_B R}{L} \right\rceil \tag{9}$$

단 $\lceil x \rceil$는 x보다 크거나 같은 최소 정수이다.

경우 2

여기서의 문제덕 입력데이타가 무제한의 속도로 공급될 수 있을때, 시스템의 최대 처리속도(throughput) $R_{max}$를 찾고, 그 경우에 최소로 필요한 프로세서의 갯수 $P_{mim}$을 결정하는 것이다.

(i) 입력데이타가 무제한의 속도로 공급될 수 있기 때문에 I/O의 의존시간을 의미하는 procedure #1의 loop delay는 문제가 안된다. 따라서 critical loop는 processor 사이의 통신 동작에 의해 형성된다.

$$T_{sk}=T(3)+T(4)+T(5) \tag{10}$$

(ii) 최대 throughput $R_{max}$는 다음과 같이 결정된다.

$$R_{max}=\frac{L}{T_{sk}} \tag{11}$$

(iii) 위의 throughput을 얻기 위한 최소 프로세서의 갯수 $P_{mim}$은

$$P_{min}=\left\lceil \frac{T_B}{T_{sk}} \right\rceil = \left\lceil \frac{T_B}{T(3)+T(4)+T(5)} \right\rceil \tag{12}$$

이다.

위의 결과가 가르쳐 주는 바는, 아무리 프로세서의 수를 늘려도 프로세서 사이의 통신에 기인하는 의존시간이 줄지 않으면 높은 처리속도(throughput)를 얻을 수 없다는 것이다. 본 제시된 병렬 블럭처리 방법을 이용하여 DFT(FFT)나 FIR 필터를 구현하는 경우에는 프로세서 사이의 통신이 dependence graph에서 궤환고리(feedback loop)를 형성하지 않으므로 매우 높은 처리속도(throughput)를 얻는 것이 가능하다. 그러나 순환(recursive)이나 적응(adaptive) 이나 적응(adaptive) 필터의 경우와 같이 알고리듬에 궤환(feedback)이 있는 경우는 각 프로세서 사이에 매우 긴 의존시간을 갖게되고, 따라서 이를 줄이는 방법을 채용하지 않고서는 높은 처리속도(throughput)를 얻을 수 없다. 다음 절에 궤환(feedback)이 있는 알고리듬에서 의존시간을 줄이는 법이 제시되었다.

## Ⅳ. 의존시간을 줄이는 방법

식(1)에서 보는 바와 같이 시스템내에서 쓰일 수 있는 프로세서의 갯수는 그들간의 의존시간의 길이, 즉 $T_d$, 에 반비례한다. 특히 궤환(feedback) 있는 알고리듬에서는 의존시간이 대단히 길다. 예를들면 1차의 순환(recursive)필터링 알고리듬에서 다음 프로세서를 위한 초기조건, 즉 y[L], 은 어떤 프로세서가 한 블럭의 입력 x[1], …, x[L]과 선처리기(preprocessor)로 부터 초기조건 y[0]를 받아 그 블럭의 출력 y[1], …, y[L]을 순서적으로 모두 계산한 뒤에야 얻어진다. 이때 의존시간은 초기조건 y[0]를 선처리로 부터 받아서 y[L]을 후처리기로 보내는데 걸리는 시간이다. 따라서 이 경우 의존시간의 길이는 거의 한 블럭의 처리시간 $T_B$와 같아지고, 프로세서를 한개나 또는 두개이상 쓸 수 없다. 이 문제를 극복하기 위하여 다음 두가지 방법이 연구되었다.

1. 분해(decomposition) 방법

이 방법에서는 알고리듬을 덜 복잡한 몇개의 과정(procedure)으로 나누어서 의존시간을 줄인다. Dependence graph에서는 어떤 긴 loop가 여러개의 작은 loop로 변환되고, 따라서 critical loop가 짧아진다. 일예로 고차의 순환(recursive)디지탈 필터를 direct form 대신 병렬(parallel) 또는 직렬(cascade)

형태로 구현되는 방법을 들 수 있다. 이 경우 전체 의존시간은 1차 또는 2차 한 단의 의존시간에 의해 결정되고 따라서 필터를 저차(low order)의 형태로 나눈 만큼 의존시간이 줄어든다. 이 방법은 적응 격자(adaptive lattice)필터등과 같이 분해된 구조를 갖는 필터의 구현에 쉽게 적용될 수 있다. 이 방법의 한가지 제한점은 필터의 차수가 클 때에만 적용의 성과가 크다는 것이다.

2. 전방예측(look-ahead)방법

이 방법에서는 다음 processor를 위한 초기조건을 별도의 식을 사용하여 미리 계산함으로써 의존시간을 줄이는 방법이다. 즉 look-ahead carry generation adder에서는 입력을 중간형태의 신호인 propagation과 generation으로 미리 바꾼후 입력 carry $C_0$와 조합하여 임의의 단의 carry 출력 $C_n$을 계산하기 때문에 carry 전달에의한 지연시간을 줄인다. 마찬가지로 본 제시된 방법에서는 입력 데이타가 들어오면, 그 입력 데이타만 사용하여 중간결과를 미리 만든후, 선처리기(preprocessor)에서 얻어진 초기조건과 결합하여 후처리기(postprocessor)을 위한 초기조건을 우선적으로 계산하여 보낸다. 이 과정을 이용하면 중간결과를 만들지 않았던 경우보다 초기조건 지연시간이 대폭 줄어든다. 이 방법은 비록 추가계산을 필요로 하지만, 분해(decomposition)방법을 적용할 수 없었던 1차 또는 2차의 순환(recursive)필터 알고리듬에도 적용이 가능하다.

순환(recursive)이나 적응(adaptive)필터에 전방예측(look-ahead)방법은 다음과 같은 과정을 거쳐서 적용될 수 있다. 즉 이러한 필터들은 선형 순환식(linear recursive equation)으로 나타낼 수 있는데, 선형 순환식에서는 특성해(particular solution)와 과도해(transient solution)두 값을 더함으로써 최종 값(solution)을 얻을 수 있다. 이때 특성해(particular solution)는 초기조건 y[0]에는 상관없이 입력데이타, x[n], 만으로 구할 수 있으므로, 프로세서간의 통신에 의한 의존관계를 발생시키지 않고 계산이 가능하다. 한편 특성해(particular solution)가 미리 계산된 후, 선처리기(preprocessor)로 부터 초기조건을 받으면 후처리기(postprocessor)를 위한 초기조건을 즉시 계산할 수 있는데, 이에 의하여 의존시간이 크게 줄어든다.

초기조건 y[0]를 가지는 식(13)의 1차 순환식(recursive equation)의 의존시간(dependency time)을 전방예측(look-ahead)방법을 이용하여 줄이는 방법을 한 예로써 설명하면 다음과 같다.

$$y[n]=a_1 y[n-1]+x[n], \quad n=1, 2, 3, \cdots \qquad (13)$$

우선 특성해(particular solution) $y_P[n]$, $n=1, 2, \cdots$, L은 처음에 위식의 초기조건 y[0]를 0으로 생각한 후 y[n]을 구함으로써 얻어진다. 따라서 이 과정은 선처리기(preprocessor)로 부터의 초기조건에 의존하지 않고 계산될 수 있다. 다음에 후처리기(postprocessor)를 위한 초기조건 y[L]은 다음식을 이용하여 즉시 계산될 수 있다.

$$y[L]=y_P[L]+y_H[L]=y_P[L]+a_1^L y[0] \qquad (14)$$

즉 선처리기(preprocessor)에 의존할 필요가 없는 특성해(particular solution) $y_P[n]$을 우선 계산한후, 선처리기(preprocessor)에서 초기조건, y[0]을 받으면, 후처리기(postprocessor)를 위한 초기조건, y[L]은 단 몇 step만에 위의 식을 이용하여 계산할 수 있다. 여기서 초기조건을 주고받는 시간과 초기조건을 계산하는 식(14)에 걸리는 시간의 합이 의존시간, $T_d$이 되는데, 이는 $2C_a+2C_c$이다. 한편 한 블럭의 출력을 특성해(particular solution)와 과도해(transient solution)로 나누어서 계산하는데 필요한 시간 $T_B$는 $4LC_a+2LC_{io}+2C_c$이다. 따라서 1차 순환식(recursive equation)에 전방예측(look-ahead)방법을 적용했을 때의 $P_{max}$는 다음과 같다.

$$P_{max,look-ahead}=\left\lfloor \frac{4LC_a+2LC_{io}+2C_c}{2C_a+2C_c} \right\rfloor \qquad (15)$$

위의 식에서 $P_{max}$는 한 block의 길이 L에 비례하므로 상당히 큰 값이 될 수 있다. 일예로 $L=256$, $C_a=1$, $C_{io}=2$, $C_c=4$일때 $P_{max}$는 205가 된다.

본 전방예측 방법의 약점은 초기조건을 미리계산하기 위해서 추가의 계산이 필요하다는 점이다. 그러나, 이 약점은 본 방법에 의하여 대량의 다중처리가 가능하기 때문에 쉽게 덮어질 수 있다.

## V. 디지탈 필터링 알고리듬의 구현

본 제시된 방법을 이용하여 FIR, 순환(recursive) 그리고 적응(adaptive)필터링 알고리듬이 구현되었으며, 특히 사용가능한 프로세서의 수를 늘이기 위해 의존시간을 줄이는 방법이 적용되었다. 사용될 수 있는 최대 프로세서 수 $P_{max}$, 시스템의 처리속도(throughput), 그리고 전체적인 효율이 제시된 실행시간 모형에 기초하여 구해졌다.

1. FIR 필터링

사용된 FIR필터링 식은 다음과 같다.

$$y[n]=\sum_{i=0}^{M-1} h_i\, x[n-i], \qquad n=1,2,\cdots,L \qquad (16)$$

여기서 x[n]과 y[n]은 각각 입·출력 신호이고, M은 필터의 길이이며, 그리고 필터계수는 $h_i$로 주어졌다. 블럭의 길이 L은 필터의 길이 M보다 매우 큰 것으로 가정한다. 이 알고리듬에는 궤환(feedback)이 없으므로 해당 dependence graph는 입·출력 동작을 제외하면 궤환고리(feedback loop)를 갖지 않는다. 이경우 블럭 경계면에서 필터 입력의 중첩(overlap)때문에 선처리기(preprocessor)로 부터 M−1샘플을 받아야 하고, 마찬가지로 현 block의 입력 데이타 중의 마지막 M−1개의 입력 샘플을 후처리기(postprocessor)로 보내야 한다. 그러나 선처리기(preprocessor)에서 data를 받기 전에 후처리기(postprocessor)에 데이타를 보낼 수 있으므로, 프로세서 사이에 통신동작은 이루어지나 dependence graph 에서 궤환고리(feedback loop)를 형성하지는 않는다. 따라서 $P_{max}$에 대한 제약은 없다. 한 블럭에 대한 처리시간 $T_B$는 $\{2MLC_a+2LC_{io}+2(M-1)C_c\}$이다. 그러므로 주어진 처리속도(throughput) R을 얻기위해 사용되어야 하는 프로세서 갯수는 식(9)로부터 유도될 수 있으며 다음과 같다.

$$P_{min,FIR}=\left\lceil R\left\{2MC_a+C_{io}+\frac{2(M-1)C_c}{L}\right\}\right\rceil \qquad (17)$$

P개의 프로세서를 사용할 때의 처리속도(throughput)는 다음과 같이 표시될 수 있다.

$$R(P)_{FIR}=\frac{P}{2MC_a+2C_{io}+\dfrac{2(M-1)C_c}{L}} \qquad (18)$$

이 알고리듬의 효율은 식(16)으로부터 구할 수 있으며 식(19)와 같다. 프로세서 사이의 통신 시간이 한 블럭의 길이 L로 나뉘어지므로, 블럭길이가 증가함에 따라 효율이 높아짐을 알 수 있다.

$$\mathbf{E}(P)_{FIR}=\frac{2MC_a+2C_{io}}{2MC_a+2C_{io}+\dfrac{2(M-1)C_c}{L}} \qquad (19)$$

예를 들어, M=32, $C_a$=1, $C_{io}$=2, $C_c$=4인 경우, 효율은 99%이상이 된다. 비록 매우 느린 통신 경로를 사용한다 하더라도 효율이 크게 저하되지는 않는데, 예를 들어 $C_c$가 20인 때에도 효율은 96%이상이다. 또한 일반적인 멀티프로세서 시스템은 프로세서의 수가 증가할수록 효율이 떨어지나, 본 방법에서는 효율이 프로세서 갯수에 무관하다.

Indirect convolution 방법을 이용한 FIR 필터링 알고리듬도 제안된 방법으로 구현될 수 있다. 이 방법은 필터 길이가 길 때 연산횟수가 적은 장점이 있으나, 내부에 butterfly 형태의 데이타 교환을 필요로 한다. 그러나 제안된 시스템에서는 한 데이타 블럭이 한 프로세서에 의해 처리되므로, butterfly interconnection을 필요로하지 않는다.

2. 순환(recursive)필터링

아래와 같은 M차 IIR 필터링 식은 한 블럭의 입력 데이타, x[1], ⋯, x[L]외에 전 블럭의 마지막 M출력샘플 y[0], y[−1], ⋯, y[−M+1]과 마지막M 입력샘플 x[0], x[−1], ⋯, x[−M+1]을 필요로 한다.

$$y[n]=\sum_{i=1}^{M} a_i y[n-i]+\sum_{j=0}^{M} b_j\, x[n-j] \quad n=1,2,\cdots,L \qquad (20)$$

여기서 필요한 M개의 입력 샘플은 FIR 필터링에서와 같은 방법으로 의존관계를 발생시키지 않고 얻어질 수 있다. 그러나 출력 샘플은 순환적으로 계산되므로, 먼저번 블록의 마지막 M개의 출력 샘플, y[0], y[−1], ⋯, y[−M+1]은 전체 출력 샘플, y[−L+1], ⋯, y[0]가 전부 계산되기 전까지는 얻어질 수 없다. 따라서 비록 전달함수(transfer function)의 분자 부분이 독립적으로 미리 계산되더라도 위와 같은 형태의 IIR 필터링을 구현하는데 허용되는 최대 프로세서 갯수는 두개정도로 매우 제한된다.

프로세서의 수를 늘리기 위해서 분해(decomposition)방법을 적용할 수 있는데, 이는 direct form 형태의 IIR filter를 2차의 직렬(cascade form) 형태로 바꿈으로써 가능하다. 같은 차수의 pole과 zero를 갖는 전달함수를 가정하면 2차의 직렬형태를 사용한 경우 $P_{max}$는 근사적으로 필터 차수와 같아진다. 2차 회귀식은 10개의 산술연산을 사용하기 때문에 M차 필터의 산술연산 갯수는 5M이다. 필터 차수를 짝수라고 가정하면, 이 방법의 효율은 다음과 같이 표현될 수 있다.

$$\mathbf{E}(P)_{IIR,decomp.}=\frac{5MC_a+2C_{io}}{5MC_a+2C_{io}+\dfrac{4MC_c}{L}} \qquad (21)$$

L=256, M=5, $C_a$=1, $C_{io}$=2, $C_c$=4와 같은 경우에 효율은 99% 정도이다.

한편 사용가능한 프로세서의 수를 대폭 늘리기 위해서 전방예측(look-ahead)방법을 적용할 수 있다. 이 방법은 초기조건, y[0], y[−1], ⋯, y[−1], ⋯, y[−M+1], 이 "0"임을 가정하고, 일반적인 순환필터링 식인 식(20)을 사용하여 우선 특성해(particular solution)를 계산한다. 여기서는 한 블럭을 계산하기 위해 덧

셈과 곱셈을 포함하여 약 $2(2M+1)L$번의 산술연산이 필요하다. 그 다음 단계에서, 현재의 프로세서는 선처리기(preprocessor)로 부터 초기조건을 받고 다음 식을 이용해 후처리기(postprocessor)를 위한 초기조건을 계산한다.

$$\begin{bmatrix} y[L] \\ y[L-1] \\ \cdot \\ \cdot \\ y[L-M+1] \end{bmatrix} = \begin{bmatrix} y_p[L] \\ y_p[L-1] \\ \cdot \\ \cdot \\ y_p[L-M+1] \end{bmatrix} + \begin{bmatrix} W[L] & & \\ W[L-1] & & \\ \cdot & & \\ \cdot & & \\ W[L-M+1] & \cdots & y[-M+1] \end{bmatrix} \begin{bmatrix} y[0] \\ y[-1] \\ \cdot \\ \cdot \\ y[-M+1] \end{bmatrix} \quad (22)$$

가중함수(weighting function) $[W[L], W[L-1], \cdots, W[L-M+1]]^T$는 $M\times M$의 행렬으로서 식(23)을 사용해 필터 설계단계에서 계산할 수 있다.[12][14]

$$\mathbf{W}[n] = \mathbf{C}\cdot\mathbf{A}^n, \qquad \text{for } n=1,2,\cdots,L \quad (23)$$

위 식에서 M차의 벡터 $\mathbf{C}$와 $M\times M$의 행렬 $\mathbf{A}$는 다음과 같다.

$$\mathbf{C}=[1\ 0\ 0\cdots 0]^T, \quad \mathbf{A}[n]=\begin{bmatrix} a_1 & a_2 & \cdots & a_M \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & \cdots & 1\ 0 \end{bmatrix}$$

이 단계에서 의존시간은 $2M^2$번의 산술연산과 $2M$ 샘플의 통신 시간을 합한 것이 된다. 나머지 출력샘플에 대한 특성해(transient solution)는 다음식을 이용해서 순환적으로 계산된다.

$$y_t[n]=\sum_{i=1}^{M} a_i y_t[n-i], \quad \text{for } n=1,2,\cdots,L-M \quad (24)$$

단, $y_t[-i]=y[-i]$, $i=0,1,2,\cdots,M-1$이다. 실제로 모든 특성해(transient solution)는 초기조건을 가중함수(weighting function)와 곱함으로써 계산할 수 있으나 이 경우 가중함수로 $L\times M$의 행렬을 필요로 한다. 그러므로 다른 출력 샘플들에 대한 특성해는 순환적으로 계산하는 것이 간단하다.

특성해 계산과정에서의 산술연산 횟수는 약 $2LM$이다. 따라서 특성해와 과도해를 포함한 전체 산술연산 횟수는 약 $2(3M+1)L$이며, 하나의 데이타 블럭의 실행시간 $T_B$는 $\{2(3M+1)LC_a+4MC_a+2LC_{io}\}$로 표시된다. 의존시간 $T_d$는 $2M^2C_a+2MC_c$이다. 결과적으로 식(1)을 사용해서 얻을 수 있는 $P_{max,IIR,lookahead}$는 $P_{max,IIR,decomp}$에 비해 매우 크다. 예를 들어 L이 256, M이 5, $C_a$가 1, $C_{io}$가 2, $C_c$가 4이면 $P_{max,IIR,lookahead}$는 103이다. $P_{max}$는 블럭길이 L이 증가함에 따라 커지고, 분해(decomposition) 방법을 함께 사용함으로써 더욱 증가시킬 수 있다. 전방예측(look-ahead)방법을 사용할 때의 효율은 다음과 같다.

$$\mathbf{E}(P)=\frac{2(2M+1)C_a+2C_{io}}{2(3M+1)C_a+2C_{io}+\dfrac{4MC_c}{L}} \quad (25)$$

전방예측(look-ahead)방법을 적용시 샘플당 전체 산술연산 횟수가 $2(3M+1)$이므로 일반적인 계산 횟수인 $2(2M+1)$보다 50%이상 많아진다. 따라서 이 방법의 효율은 67% 정도로 제한된다. 예를 들어, M이 5, L이 256, $C_a$가 1, $C_{io}$가 2, $C_c$가 4이면 효율은 71%이다. 그러나 사용되는 프로세서의 갯수가 많아져도 그 효율이 유지된다는 점은 주목할 만하다. 속도개선비(spped-up)는 단순히 프로세서 갯수를 효율과 곱함으로써 계산할 수 있다. 시스템의 효율이 프로세서 갯수와 무관하기 때문에, 처리속도(throughput)는 프로세서의 갯수가 증가함에 따라 거의 직선적으로 증가한다.

### 3. 적응(adaptive) 필터링

가장 널리 쓰이는 adaptive transversal 필터 구조는 global feedback을 사용하므로 분해(decomposition) 방법을 적용하기 힘들다. 한편 adaptive transversal filter를 시변(time-varying)계수를 갖는 회귀식(recursive equation)으로 표시한 후, 전방예측(look-ahead) 방법을 적용할 수 있으나 이에는 $O(M^2)$의 계산량이 필요하다.[13]-[15] 즉 순환식(recursive equation)의 병렬 계산에 기초한 전방예측(look-ahead)방법은, 시변(time-varying)계수를 갖는 경우 필터의 차수가 1이 아니면 매우 비효율적이다. 따라서 adaptive transversal filter보다는 adaptive lattice filter 처럼 분해된 구조(decomposed structure)를 갖는 filter 구조가 분해(decomposition)방법뿐만 아니라 전방예측(look-ahead)방법을 적용하는 데도 더 적합하다. 즉 decomposed filter에서 각 단은 단순한 1차 adaptive필터이므로 전방예측(look-ahead)방법이 $O(1^2)$, 즉 매우 적은, 계산량으로 쉽게 구현될 수 있다. 이 장에서는, 그림 7에서 보이는 adaptive lattice filter의 구현이 논의된다. 이 필터에 관계된 식들은 다음과 같다.

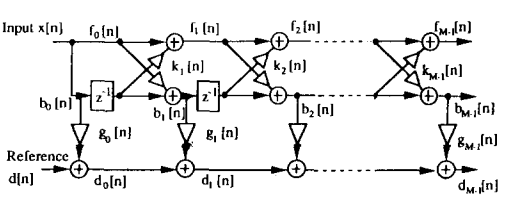


그림 7. 적응격자 여파기
Fig. 7. Adaptive lattice joint process filter.

$$C_m[n]=\gamma C_m[n-1]+2f_{m-1}[n]b_{m-1}[n-1]$$
$$D_m[n]=\gamma D_m[n-1]+(f_{m-1}[n])^2+(b_{m-1}[n-1])^2$$
$$k_m[n+1]=-C_m[n]/D_m[n]$$
$$f_m[n]=f_{m-1}[n]+k_m[n]b_{m-1}[n-1]$$
$$b_m[n]=k_m[n]f_{m-1}[n]+b_{m-1}[n-1]$$
$$d_m[n]=d_{m-1}[n]-g_m[n]b_m[n]$$
$$g_m[n+1]=g_m[n]+\delta_m d_m[n]b_m[n] \qquad (26)$$

독립적으로 제어되는 각 단의 직렬연결인 adaptive lattice filter의 dependence graph는 그림 8과 같다. 여기서 critical loop는 $d_m[n]$과 $g_m[n]$을 계산하기 위한 procedure에 의해 형성된다. 즉 critical loop에는 5L번의 산술연산과 2번의 통신동작이 포함된다. 한편, 한 블럭의 데이타에 대해 매단(stage)당 약 19L번의 산술연산과 6번의 통신동작이 필요하다. 따라서 매 블럭마다 2L번의 입출력 동작이필요하다고 가정하면, 분해(decomposition)방법을 사용한 경우 최대 프로세서 수 $P_{max}$는 다음과 같다.

$$P_{max,ALTC,decomp}=\left\lfloor \frac{M(19LC_a+6C_c)+2LC_{io}}{5LC_a+2C_c} \right\rfloor \qquad (27)$$

식(4)의 $T_B$대신 $M(19LC_a+6C_c)+2LC_{io}$를 대입함으로써, 이 시스템의 처리속도(throughput)를 계산할 수 있다. 이 방법의 효율은 다음과 같다.

$$\mathbf{E}(P)_{ALTC,decomp.}=\frac{19MC_a+2C_{io}}{M(19C_a+\frac{6C_c}{L})+2C_{io}} \qquad (28)$$

여기에서도 통신시간이 블럭의 길이L로 나눠지므로 효율이 매우 높아져서, 대개 99% 이상의 값을 갖게된다.

Dependence graph에서 $C_m[n]$, $D_m[n]$과 $g_m[n]$을 구하는 loop에 대해 전방예측(look-ahead)방법을 적용함으로써 사용가능한 프로세서의 갯수를 더욱 증가시킬 수 있다. 즉 식(26)에서 $C_m[n]$과 $D_m[n]$을 구하기 위한 계산 절차는 상수의 계수를 갖는 1차의 순환식이다. 따라서 순환필터링에 쓰인 분해(decomposition)방법이 적용될 수 있다. 한편 $g_m[n]$을 위한 계산 절차는 다음 식에서 보는 것처럼 1차의 시변순환식(time-varying recursive equaion)이 된다.

$$\begin{aligned} g_m(n+1) &= g_m(n)+\delta_m d_m(n)b_m(n) \\ &= g_m(n)+\delta_m\{d_{m-1}(n)-g_m(n)b_m(n)\}b_m(n) \\ &= \{1-\delta_m[b_m(n)]^2\}g_m(n)+\delta_m b_m(n)d_{m-1}(n) \\ &= \alpha_m(n)g_m(n)+\beta_m(n) \end{aligned} \qquad (29)$$

한편,

$$\alpha_m(n)=1-\delta_m[b_m(n)]^2$$
$$\beta_m(n)=\delta_m b_m(n)d_{m-1}(n)$$

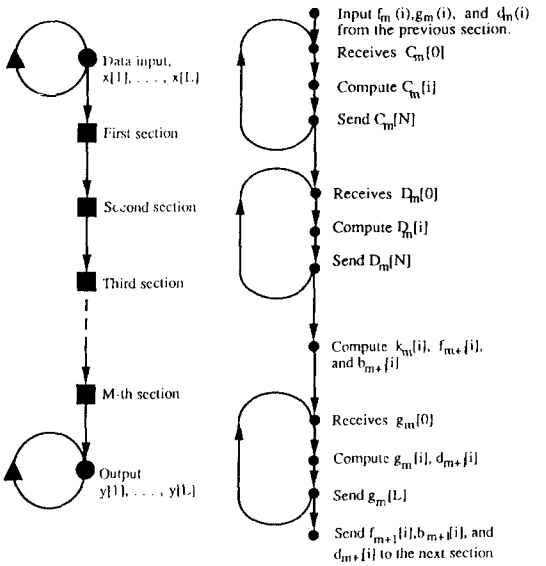


그림 8. Adaptive lattice filter의 의존표
(a) 각 section은 사각형으로 표시되었다
(b) 각 section의 의존표

Fig. 8. A precedence graph for adaptive lattice filtering.
(a) each section is represented by a box,
(b) a precedence graph for each section.

따라서 전방예측(look-ahead)방법이 적용될 수 있으나, 이때 가중함수(weighting function)는 시변(time-varying)계수로 구성되어 있기 때문에 각 블럭에서 식(30)을 이용하여 매번 계산되어야 한다.

$$W[n]=\prod_{i=1}^{n} a[i]=a[n]\ W[n-1],\quad n=1,2,\cdots,L \qquad (30)$$

이 방법을 사용한 adaptive lattice filter 한 단의 총 계산 횟수는 매 샘플당 약 25산술연산이며, 이는 이 방법을 사용치 않은 알고리듬에 비해 30% 정도 많은 것이다. 그러나, critical loop의 길이가 대폭 줄어듬으로 최대 프로세서 갯수는 다음과 같이 블럭 길이에 거의 비례하게 된다.

$$P_{max,ALTC,look-ahead}=\left\lfloor \frac{M(25LC_a+6C_c)+2LC_{io}}{2C_a+2C_c} \right\rfloor \qquad (31)$$

예를 들어 필터 차수가 10, $C_a$가 1, $C_{io}$는 2, $C_c$는 4 그리고 L이 256이면 $P_{max}$는 6500이상이다. 이 방법의 효율은 다음과 같다.

$$\mathbf{E}(P)_{ALTC,look-ahead}=\frac{19MC_a+2C_{to}}{25MC_a+\frac{6MC_c}{L}+2C_{to}} \quad (32)$$

효율은 위와 동일한 계수를 가정하였을 경우 약76 %이상이다.

지금까지 설명된 것처럼, FIR, 순환(recursive), 적응(adaptive lattice)필터가 본 다중프로세서(multiprocessor)시스템에서 성공적으로 구현되었으나, 알고리듬에 global feedback을 내장한 adaptive transversal filter의 경우 구현에 문제점이 있었다. 이것은 단일 프로세서 시스템을 이용한 구현의 경우 연산회수의 절약이 가장 중요한 문제이나, 다중 프로세서(multiprocessor)시스템 환경에서는 필터의 구조가 modular하고 지역궤환(local feedback)을 이용해서 설계되는 것이 더 중요함을 의미한다.

## Ⅵ. 시스템 구현 결과

제안된 구조에 기초한 다중 프로세서(multiprocessor)시스템은 AT&T의 DSP32 디지탈 시그날 프로세서 칩[11][16]을 사용하여 구성되었으며, 이제 이 시스템의 기본 특징을 기술하고 실제적인 구현 결과를 제시하려 한다.

### 1. 기본 구성

본 시스템은 VME Bus를 이용하였으며, 입출력 시스템과 링(ring)형태로 연결된 DSP32 프로세서 보드로 구성되어 있다. 입출력 시스템은 콘트롤러 보드와 A/D 및 D/A 콘버터 보드로 이루어졌다. 콘트롤러는 MVME105 단일 보드 컴퓨터로서 MC68010 마이크로 프로세서와 512K 바이트의 RAM, 상주 모니터 그리고 소프트웨어 개발용 디버거를 포함한다. A/D 콘버터 보드, MPV952는 12비트 정밀도와 최대 330KHz의 샘플링 주파수를 갖는다. 콘트롤러 보드로부터 각각의 DSP32 프로세서 보드로 코드를 보낼 수 있으며 또한 각각을 독립적으로 시작 또는 정지시킬 수 있다. 한편, 콘트롤러는 A/D 변환기로부터 12비트 디지탈 데이타를 받아 4개의 "0"을 붙여 해당되는 순서의 DSP32로 보낸다. 앞에서 설명된 것처럼, 데이타는 블럭 형태로 각 처리 node로 보내지거나 받아들여진다.

현재 이 시스템은 8개의 처리 node를 갖고 있으나, 32처리 node로 확장 가능한데, 이것은 VME case의 사용 가능한 공간에 의한 제약 때문이다. 각 처리 node는 좌우의 인접한 처리 node와 통신할 수 있으며 dual-port 메모리를 통하여 통신이 이루어진다.

그림 9는 한 DSP 보드의 구성을 보여준다. 하나의 보드는 두개의 처리 node와 VME 버스 인터페이스로 구성된다. 매 처리 node는 16MHz로 동작하는 DSP32 디지탈 시그날 프로세서 칩을 가지고 있는데, 이 프로세서는 4개의 accumulator를 갖는 32비트 부동 소수점(floating-point)연산 유닛과 21개의 레지스터를 갖는 16비트 제어 연산 유닛 1K×32비트 내장 RAM을 가지고 있다.

또한 각 처리 node는 프로그램과 데이타 메모리용으로 8K×32비트의 고속 static RAM을 가지고 있으며, 인접 처리 node 간의 통신을 위해서 1K×32비트의 듀얼포트 RAM을 갖는다. 또한 I/O를 위해서 VME 버스에 한쪽이 연결된 1K×16비트 듀얼 포트 RAM을 갖고 있다. 어느 프로세서에서 다른 프로세서로 데이타를 이동시킬 때에는 그 프로세서에게 듀얼 포트 메모리로부터 읽어갈 수 있는 데이타가 있음을 알려줄 필요가 있다. 이 목적으로 각 듀얼 포트 RAM에 flag의 상태결정과 읽기를 가능하게 하는 회로가 추가되었다.

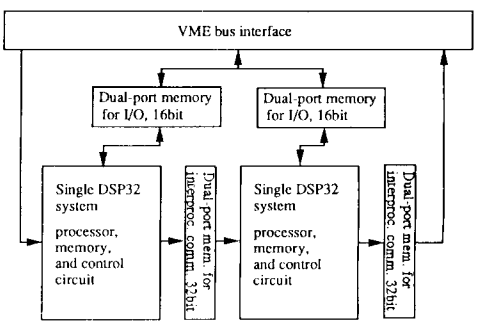


**그림 9.** 각 board의 block diagram
**Fig. 9.** Block diagram of the board.

### 2. 개발 환경

각 node에서 실행될 프로그램은 AT&T DSP32 소프트웨어 개발 도구를 사용하여 개발되었다. 이 도구는 SUN/3 workstation에서 사용되었는데, 어셈블러, 링커, 시뮬레이터, 라이브러리 그리고 여러 유틸리티들을 포함한다. 추가적인 소프트웨어가 MC68010 제어기를 위해 개발되었는데, C언어와 MC68000 어셈블리 코드를 혼합하여 프로그램을 작성하였다.

### 3. 알고리듬 구현 결과

순환(recursive)필터를 실시간으로 구현함으로써 본 시스템의 처리속도(throughput)를 측정하였다. 전

방예측(look-ahead) 방법을 이용해서 구현한 필터링 프로그램의 블럭도는 그림10에 있다. 비록 32비트 듀얼 포트 RAM이 하드웨어로 제공되었지만, 이 구현에는 프로세서 사이의 통신용으로 단지 8비트 듀얼 포트 RAM만을 사용했다. 정수(integer) 16비트 데이타 형식이 입·출력과 프로세서 사이의 통신용으로 사용되었다. 한 블럭의 데이타를 처리하기 위한 사이클 수는 아래 식(33)에 나타나 있다. 단 M은 필터차수, L은 블럭 길이이며, 각 행의 두번째 항은 앞에서 사용된 모델을 나타내고, 세번째 항은 프로그램을 분석해서 얻은 값이다.

$$\begin{aligned}&\text{연산동작에 필요한 clock수} = 2(3M+1)LC_a = (5M+18)L\\&\text{I/O 동작에 필요한 clock수} = 2LC_{io} = 10L\\&\text{통신동작에 필요한 clock수} = 2MC_c = 18M+20\\&\text{초기화에 필요한 clock수} = 37\end{aligned} \tag{33}$$

위의 두번째와 세번째항을 비교하여 $C_a$, $C_c$및 $C_{io}$를 계산할 수 있는데, 필터 차수에 따라 $C_a$가 1과 2 사이, 약 1.5이며, $C_{io}$는 5, $C_c$는 약 11임을 알 수 있다. 프로그램 초기화(initialization)의 시간은 크지 않기 때문에 이 모델에 반영되지 않았다.

본 시스템의 처리속도(throughput) 측정을 위해 전방예측(look-ahead) 방법에 기초한 8차 순환 필터의 실시간 구현이 수행되었다. 총 5개의 프로세서를 사용하여 매초당 245KHz의 샘플을 처리할 수 있었다. 시스템내의 8개의 프로세서를 모두 사용할경우, I/O속도의 제한(330KHz)에 이르렀다. 전용 I/O 보드를 만들고, 프로세서의 수를 증가시키고, 또 DSP32보다 두배이상 빠른 DSP32C를 사용한다면, 수 MHz대의 신호처리가 가능하다.

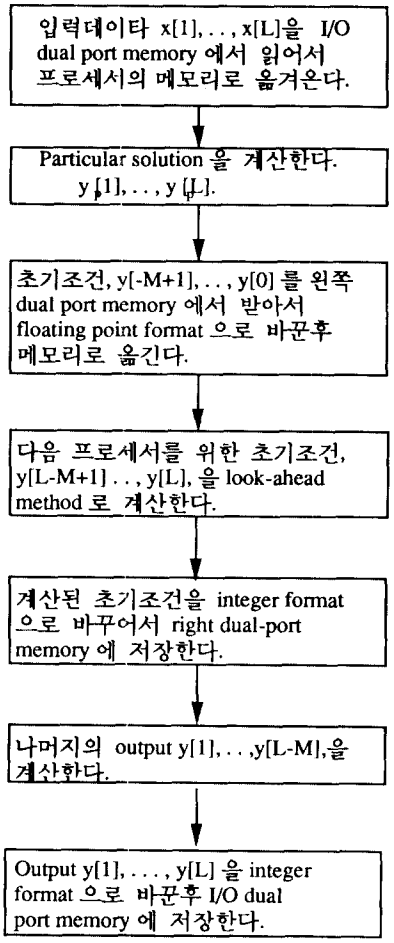


그림10. IIR filtering을 위한 프로그램의 블록 다이아그램

Fig. 10. Block diagram of the implemented program for IIR filtering.

## Ⅶ. 결 론

대규모의 다중처리(multiprocessing), 지역처리(local processing), 처리소자의 비동기적 동작등의 개념들이 다음에 설명된 방법으로 잘 융합되어서 효과적인 시스템이 구현되었다.

(1) 프로세서들 사이에 작업을 나누기 위하여 입력 데이타를 분할하는, 데이타 수준 병렬화(data level parallelism)를 채택하였다. 이 방식은 종전의 제어 수준병렬화(control-level parallelism) 방법에 비해 큰 규모의 다중처리를 가능케 할 뿐 아니라 시스템의 스케줄링을 간단하게 한다.

(2) 프로세서의 동작을 지역화시키기 위해 블럭 처리 방식이 채택되었다. 비록 이 방식은 각 프로세서에 더 많은 메모리를 요구하나, 다중 프로세서(multiprocessor) 시스템 구현에서 더 중요한 제약이 되는 통신시간을 매우 크게 줄였다.

(3) 프로세서들의 비동기적 동작은 공통의 global클럭을 제거함으로써 전체 속도를 증가시킬 뿐 아니라 알고리듬의 프로그래밍도 간단화 시켰다.

제안된 다중 프로세서(multiprocessor) 시스템 구현 방법의 장점은 다음과 같이 요약된다.

(1) 구현될 알고리듬에 관계없이 링(ring)과 같은 간단한 연결망만이 필요하다. 이는 시스템 내에서 프로세서 갯수를 쉽게 증가시킬 수 있게 할 뿐 아니라, adaptive 필터링이나 FFT 등과 같이 다른 구조를 갖는 여러 알고리듬의 혼합된 구현에 유리하다.

(2) 모든 프로세서가 같은 프로그램을 필요로 하므로 프로그래밍과 스케줄링이 단일처리 시스템만큼

쉬우며, 각 프로세서들은 할당된 데이타 블럭에 대해서는 마치 독립된 단일 프로세서처럼 동작한다.

(3) 시스템의 수행량은 프로세서의 갯수에 거의 직선적으로 증가한다. 또한 프로그램의 변경없이 처리속도(thoughput)와 프로세서 갯수간의 트레이드-오프(trade-off)를 할 수 있다. 따라서 더욱 빠른 프로세서가 공급되면 프로세서의 갯수를 줄일 수 있다. 또한 이 특징은 고장에 강한 시스템 구현에 이용될 수도 있다.[14]

## 감사의 글

이 연구는 본인이 미국 University of California, Santa Barbara에서 공부하는 동안에 시작되었다. 한국에서는 반도체공동연구소, 제어계측 신기술 연구센터(ERC-ACI)와 자동화연구소내 고속프로세싱실의 프로젝트로 수행되고 있다. Sanjit Mitra 교수와 동료 Branko Jeren 박사의 도움에 감사드린다.

## 參 考 文 獻

[1] AT&T Technologies, *WE DSP32 Digital Signal Processor Information Manual,* 1986.

[2] NEC Electronics Inc., *μPD77230 Advanced Signal Processor,* February 1986.

[3] Texas Instruments Inc., *TMS 32020 User's Guide,* 1986.

[4] Motorola Inc., *DSP56000 Digital Signal Processor User's Manual,* 1986

[5] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing,* McGraw-Hill, New York, 1984.

[6] R.W. Hockney and C.R. Jesshop, *Parallel Computers,* Adam Hilger Ltd, Bristll, 1981.

[7] D.A. Schwartz, *Synchronous Multiprocessor Realizations of Shift-Invariant Flow Graphs,* Ph.D. Dissertation, Georgia Institute of Technology, July 1985.

[8] R.E. Crochiere and A.V. Oppenheim, "Analysis of linear digital networks," *Procedings of the IEEE* vol. 63, pp. 581-595, April 1975.

[9] D.A. Schwartz and T.P. Barnwell III, "Cyclo-static multiprocessor scheduling for the optimal realization of shift-invariant flow graphs," *Proc. IEEE-International Conference on Acoustics, Speech, and Signal Processing,* Tampa, Florida, pp. 1384-1387, April 1985.

[10] H.H. Loomis, Jr. and B. Sinha, "High-speed recursive digital filter realization," Circuits, systems and signal processing, vol. 3, no. 3, pp. 267-294, 1984.

[11] H.H. Lu, E.A. Lee, and D.G. Messerschmitt, "Fast recursive filtering with multiple slow processing elements," *IEEE Trans. on Circuits and Systems,* vol. CAS-32, pp. 1119-1129, November 1985.

[12] W. Sung and S.K. Mitra, "Efficient multi-processor implementation of recursive digital filters," *Proc. IEEE-International Conference on Acoustics, Speech, and Signal Processing,* Tokyo, Japan, pp. 257-260, April 1986.

[13] W. Sung and S.K. Mitra, "Fast adaptive filtering using vector-processors and multi-processors," *Proc. IEEE Academia Sinica Workshop on Acoustics, Speech, and Signal Processing,* Beijing, China, April 1986.

[14] W. Sung, Vector and Multiprocessor Implementation of Digital Filtering Algorithms, Ph.D. dissertation, Department of Electrical and Computer Engineering, University of California, Santa Barbara, July 1987.

[15] D.D. Gajski, "An algorithm for solving linear recurrence systems on parallel and pipelined machines," *IEEE Trans. on Computers* vol. C-30, pp. 190-206, March 1981.

[16] W.P. Hays and et. al., "A 32-bit VLSI digital signal processor," *IEEE J. of Solid-State Circuits,* vol. SC-20, no. 5, pp. 998-1004, October 1985.

[17] K.O. Kjolas, *Multi-processor Implementation or Recursive Digital Filters, Using the DSP32 simulator,* Internal Report, Signal Processing Laboratory, University of California, Santa Barbara, 1986.

[18] J.P. Brafman, J. Szczupak, and S.K. Mitra, "An approach to the implementation of digital filters using microprocessors," *IEEE Trans. Acoustics, Speech, and Signal Processing,* vol. ASSP-26, no. 5, pp. 442-446, October 1978.

[19] W. Gass, R. Tarrant, and G. Doddington, "A parallel signal processor system," *Proc. IEEE-International Conference on Acoustics, Speech, and Signal Processing,* Tokyo, Japan, pp. 2887-2890, April 1986.

## 著 者 紹 介

**成 元 鎔**(正會員)

1955年 4月 14日生. 1978年 2月 서울대학교 전자공학과 졸업(공학사). 1980年 2月 한국과학원 전기 및 전자공학과 졸업(공학석사). 1987年 7月 미국 University of California, Santa Barbara 전기 및 컴퓨터공학과 졸업(박사). 1980年 2月～1983年 7月 (주)금성사 중앙연구소. 연구원. 1989年 2月～현재 서울대학교 반도체공동연구소 및 제어계측 공학과 조교수. 반도체공동연구소 시스템 연구실장과 자동화시스템 공동연구소 고속 프로세싱 연구실장을 맡고 있음. 주관심분야는 병렬처리 컴퓨터와 VLSI를 이용한 고속신호 처리등임.