2015년 2월 전자공학회 논문지 제 52권 제 2 호
Journal of The Institute of Electronics and Information Engineers Vol. 52, NO. 2, February 2015 http://dx.doi.org/10.5573/ieie.2015.52.2.097

논문 2015-52-2-12

SIMD 구조를 갖는 프로세서에서 FFT 연산 가속화

(Acceleration of FFT on a SIMD Processor)

이 주 영*, 홍 용 근*, 이 현 석**

(Juyeong Lee, Yong-guen Hong, and Hyunseok Lee©)

요 약

이 논문은 SIMD 구조를 갖는 프로세서에서 FFT 연산을 효과적으로 처리하는 방법에 대한 것이다. FFT는 디지털 신호처리 분야에서 널리 사용되는 범용 알고리즘으로 이의 효과적인 처리는 성능 향상에 있어서 매우 중요하다. Bruun 알고리즘은 반복적인 인수분해를 통해 구현되는 FFT 알고리즘으로, 널리 사용되는 Cooley-Tukey 알고리즘에 비해 복소수 곱셈이 아닌 실수 곱셈으로 대부분의 동작을 수행하는 장점을 가지고 있으나, SIMD 프로세서에서 구현하는 데는 벡터 데이터의 정렬 형태가 복잡하고 연산에 필요한 계수들을 저장할 메모리를 더 필요로 하는 단점이 있다. 실험 결과에 따르면 길이 1024인 FFT 연산을 SIMD 프로세서에서 수행하는데 있어서 Bruun 알고리즘은 Cooley-Tukey 알고리즘에 비해서 약 1.2배의 더 높은 처리 성능을 보이지만, 약 4 배 더 큰 데이터 메모리를 필요로 한다. 따라서 데이터 메모리에 대한 제약이 큰 경우가 아니라면 SIMD 프로세서에서 Bruun 알고리즘이 FFT 연산에 적합하다.

Abstract

This paper discusses the implementation of Bruun's FFT on a SIMD processor. FFT is an algorithm used in digital signal processing area and its effective processing is important in the enhancement of signal processing performance. Bruun's FFT algorithm is one of fast Fourier transform algorithms based on recursive factorization. Compared to popular Cooley-Tukey algorithm, it is advantageous in computations because most of its operations are based on real number multiplications instead of complex ones. However it shows more complicated data alignment patterns and requires a larger memory for storing coefficient data in its implementation on a SIMD processor. According to our experiment result, in the processing of the FFT with 1024 complex input data on a SIMD processor, The Bruun's algorithm shows approximately 1.2 times higher throughput but uses approximately 4 times more memory (20 Kbyte) than the Cooley-Tukey algorithm. Therefore, in the case with loose constraints on silicon area, the Bruun's algorithm is proper for the processing of FFT on a SIMD processor.

Keywords: FFT, Bruun's FFT, SIMD Processor

Ⅰ. 서 론

디지털 신호처리 기술은 무선통신, 멀티미디어, 영상처리 등 많은 분야에 걸쳐 활용되고 있으며, 그 중요성은 점점 더 높아지고 있다. FFT (Fast Fourier transform)는 이산 신호의 주파수 분석을 위한 디지털 신호처리 알고리즘 가운데 하나로, 스펙트럼 분석기, OFDM 변복조기, MRI와 같은 전자기기에 사용된다.

FFT는 Cooley-Tukey 알고리즘[1], Prime-factor 알고리즘[2], Bruun 알고리즘[3], Rader 알고리즘[4]과 같은 여러 방법으로 구현이 가능하지만 이들 가운데 Cooley-Tukey 알고리즘이 가장 널리 사용되고 있다.

FFT 알고리즘 그 높은 연산량 때문에 ASIC (Application specific integrated circuit) 형태로 구현되는 경우가 많았지만, FFT 알고리즘을 사용하는 규격의

* 학생회원, ** 정회원, 광운대학교 전자통신공학과
(Dept. of Electronics and Communications Engineering, Kwangwoon Univ.)
© Corresponding Author(E-mail: hyunseok@kw.ac.kr)
접수일자: 2014년07월31일, 수정일자: 2014년12월26일
게재확정: 2015년02월04일

종류가 다양해지면서, 개발 비용이 높고 기간이 긴 ASIC 대신 프로그램 가능한 DSP (digital signal processor)를 사용하는 경우가 늘고 있다. DSP는 신호처리 연산에 특화된 연산을 다수 포함하고 있으며 그 구조도 VLIW (Very long instruction word), MIMD (Multiple instruction multiple data), SIMD (Single instruction multiple data)와 같은 다양한 구조가 시도되고 있다. DSP는 ASIC에 비하여 그 처리 성능이 상대적으로 낮기 때문에 DSP의 FFT 알고리즘 처리 성능을 높이는 연구가 계속되어 왔다. 본 논문은 벡터 형태의 연산에 적합한 SIMD 구조의 DSP를 이용해서 FFT 연산을 효과적으로 처리하는 방법에 대해 논한다.

지금까지 SIMD 구조의 DSP에서는 Cooley-Tukey 알고리즘이 널리 사용되어 왔다. [5]에서는 Intel 프로세서의 SIMD 확장 명령어를 활용하는 Cooley-Tukey 알고리즘 기반의 고성능 FFT 라이브러리를 제안하였다. [6][7]에서는 Cooley-Tukey 알고리즘 처리에 필요한 데이터 정렬 동작을 효과적으로 처리하는 ASIP (Application specific instruction-set processor)을 설계하고, 이 것을 집적회로로 구현한 결과를 보였다. 이외에도 SIMD구조의 프로세서에서 Cooley-Tukey FFT 연산의 효율성을 높이기 위해 데이터 처리 시간이 가장 빠른 벡터 레지스터를 효율적으로 사용하는 방법에 대한 연구가 진행되었다[8].

다른 주요한 FFT 알고리즘인 Bruun 알고리즘의 경우 ASIC 형태의 구현이 주를 이루고 있다, [9]에서는 ASIC으로 구현하였을 때, Bruun 알고리즘의 성능이 Cooley-Tukey 알고리즘에 비해 약 7% 높음을 보인다. [10]에서는 perfect shuffle network[11]를 이용하여 Bruun 알고리즘을 수행하는 ASIC 구조를 제안하였다.

이 논문에서는 이전의 연구들과는 달리 SIMD 구조의 프로세서에서 Bruun 알고리즘을 구현하여 성능을 높이는 방안에 대해 논한다. Bruun 알고리즘은 실수 곱셈이 주를 이루므로 Cooley-Tukey 알고리즘에 비해 곱셈 연산 횟수가 적은 특징을 나타낸다. 이와 같은 특성은 다수의 연산장치를 동시에 사용하는 SIMD 구조에 적합하다. 그러나 Bruun 알고리즘은 연산에 필요한 계수 생성 방식이 복잡하고 데이터 정렬 형태가 복잡하여 병렬처리가 어렵다. 따라서 순차적으로 작업을 처리하는 구조에서 Bruun 알고리즘이 효율적이라고 알려져 왔다[12]. 본 논문에서는 Bruun 알고리즘을 SIMD 구조의 프로세서에 적용하는데 있어 주요한 제약요소인 높은 계수 생성과 데이터 정렬의 복잡도를 메모리 추가 사용으로 해결하고자 한다. 연산에 필요한 계수와 데이터 정렬 형태는 메모리에 사전 계산하여 저장할 수 있다. ASIC은 제한된 용도를 목적으로 사용되므로 메모리의 크기를 늘리는 것이 어렵지만, DSP는 그렇지 않다. 이는 DSP가 여러 종류의 신호처리 알고리즘을 처리해야하기 때문이다. 알고리즘의 종류에 따라 메모리를 많이 필요로 하는 경우가 존재하므로, DSP는 가능한 범위 내에서 충분한 크기의 메모리를 가지도록 설계되며 그 크기는 수 십 Kbyte에 이른다.

이 논문에서는 성능 평가를 위해 동일한 구조의 실험용 SIMD 프로세서에서 Cooley-Tukey 알고리즘과 Bruun 알고리즘을 구현한 결과를 비교한다. 실험용 SIMD 프로세서는 C++ 언어 기반의 이산 사건 모사 라이브러리인 SSIM을 이용하여 구현하였다.[13]

모사 장치상의 실험 결과에 따르면 Bruun 알고리즘이 Cooley-Tukey 알고리즘에 비하여 SIMD 구조의 프로세서에서 더 많은 메모리를 필요로 하지만, 단위 시간 당 더 많은 입력 데이터를 처리할 수 있다. 예를 들면 <그림 6>과 <그림 7>에서 보인 것과 같이 1024 개의 데이터를 처리할 때 Bruun 알고리즘은 약 4배 정도의 데이터 메모리를 더 사용하지만, 그 처리 시간은 20% 정도 더 짧다. 이 조건에서 Bruun 알고리즘이 사용하는 데이터 메모리의 크기는 약 20Kbyte로, 통상의 DSP들이 가지고 있는 Cache의 크기보다 작으므로 구현이 타당성을 가진다.

본 논문은 다음과 같은 형태로 기술된다. Ⅱ.1에서는 Cooley-Tukey 알고리즘과 Bruun 알고리즘의 연산 특성을 비교한다. Ⅱ.2에서는 FFT 알고리즘을 구현하는 SIMD 프로세서의 구조에 대해서 설명한다. Ⅱ.3에서는 두 FFT 알고리즘을 SIMD 프로세서에 구현하는 과정을 상세히 설명한다. Ⅲ에서는 설계된 내용을 기반으로 실험 결과를 보여주고 Ⅳ에서 결론을 맺는다.

Ⅱ. 본 론

1. Cooley-Tukey FFT와 Bruun FFT의 비교

1) FFT 알고리즘의 병렬처리 특성

Cooley-Tukey 알고리즘과 Bruun 알고리즘 모두 한

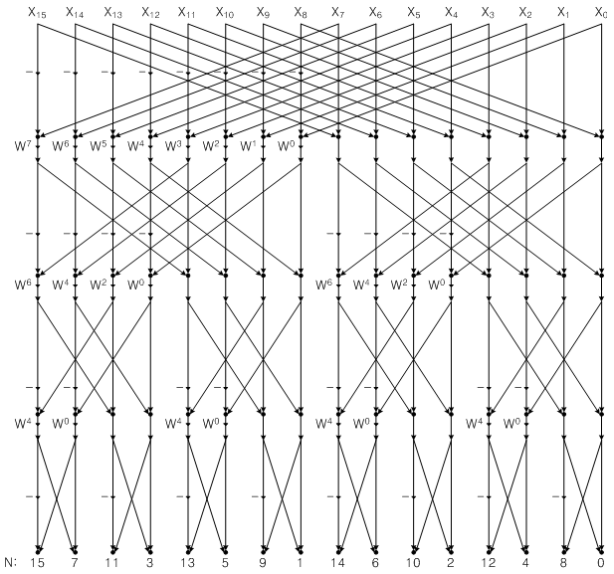

그림 1. 길이 16인 Cooley-Tukey FFT의 신호 흐름도
Fig. 1. Signal flow graph of Cooley-Tukey FFT, N=16.

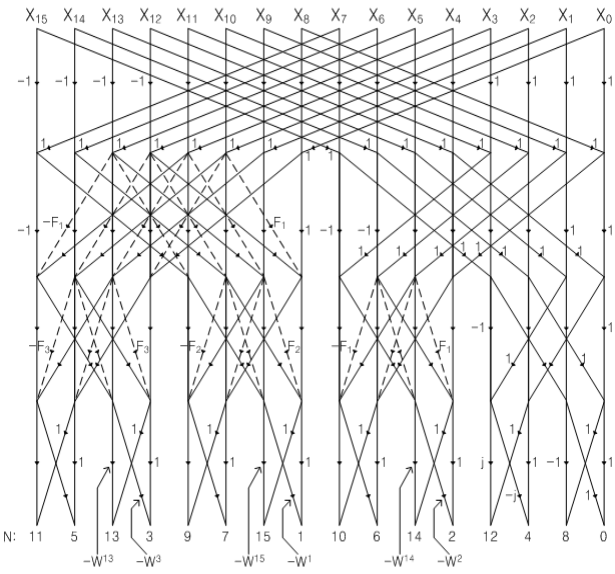

그림 2. 길이 16인 Bruun FFT의 신호 흐름도
Fig. 2. Signal flow graph of Bruun's FFT, N=16.

단에서 수행되는 연산들을 병렬 처리할 수 있다. 이는 동일한 단에서 수행되는 연산에서 입력으로 사용하는 모든 데이터들 사이에는 상호 의존성이 없기 때문이다. 그러나 한 단이 연산을 수행하기 위해서는 이전 단의 연산 결과가 입력으로 필요하므로 FFT 연산의 단들 사이에는 종속성이 존재한다. 두 FFT 알고리즘에서 서로 인접한 두 단의 연산 사이에는 데이터의 정렬 과정이 필요하다. 예를 들어 <그림 1>에서 첫 번째 단에서 (두 번째 줄) X_7에 대한 연산 결과는 다음 단에서 X_3의 위치에서 수행되는 연산의 입력으로 사용된다. 따라서 병렬처리 하드웨어에서는 이전 단의 결과를 다음 단의 입력으로 사용하기 위해 재정렬 과정이 필요하다.

2) FFT에 필요한 연산의 비교

Bruun 알고리즘을 수행하는데 필요한 곱셈 연산의 횟수는 Cooley-Tukey 알고리즘의 약 2/3이며, 이는 Bruun 알고리즘이 중간 단들의 연산에 실수 계수를 사용하기 때문이다. 반대로 Cooley-Tukey 알고리즘은 모든 단에서 복소수 계수를 사용한다.

두 알고리즘의 또 다른 차이는 입력 데이터의 수다. Cooley-Tukey 알고리즘에서 사용되는 모든 연산은 2개의 입력 데이터를 요구하지만, Bruun 알고리즘은 부분적으로 3개의 입력 데이터를 동시에 요구하는 연산을 사용한다. 예를 들면 <그림 2>에서 점선에 해당하는 데이터를 입력으로 사용하는 단에서는 3개의 입력데이터를 처리해야 한다. 이와 같은 불규칙적인 연산 형태는 SIMD 프로세서에서 구현 과정에서 조건부 분기문의 수를 늘려 프로세서의 처리 성능이 낮아지게 만들 수 있다.

3) 벡터 데이터 정렬 형태 비교

Bruun 알고리즘에서 수행되는 데이터 정렬 횟수가 Cooley-Tukey 알고리즘에 비해 더 많으며, 그 형태도 더 복잡하여 SIMD 프로세서에서 데이터 정렬을 위한 네트워크의 복잡도가 증가한다. Bruun 알고리즘은 Cooley-Tukey 알고리즘의 데이터 정렬 형태에 추가해서, <그림 2>의 점선과 같이 한 단의 연산 결과에 실수 계수를 곱한 값을 다음 단의 입력으로 제공하기 위한 데이터 정렬 형태가 추가로 사용된다[3].

$$Num_S^{CT} = Nlog_2^N \tag{1}$$

$$Num_S^B = 2Nlog_2^N + N\cdot\left(\frac{1}{2}\right)^{\log_2^N - 2} - 3N \tag{2}$$

위의 식들은 길이가 N인 데이터를 Cooley-Tukey 알고리즘과 (식 (1)) Bruun 알고리즘을 (식 (2)) 이용하여 처리할 때 필요한 데이터 정렬 횟수를 나타낸 것이다. 이 식들로부터 Bruun 알고리즘의 데이터 정렬 횟수가 더 많음을 알 수 있다. 그러나 이 식들은 신호 흐름도 상에서 비교한 데이터 정렬 횟수 비교이며, SIMD 구조

표 1. 길이가 16인 Bruun FFT 알고리즘에 사용되는 실수 필터 계수의 생성 예
Fig. 1. Example of real filter coefficients for the Bruun's FFT, N=16.

Coefficient of 2nd stage	Coefficient of 3rd stage
0	0
	0
0	$+\sqrt{2}=F_1$
	$-\sqrt{2}=-F_1$
$+\sqrt{2}=F_1$	$+\sqrt{2+\sqrt{2}}=F_2$
	$-\sqrt{2+\sqrt{2}}=-F_2$
$-\sqrt{2}=-F_1$	$+\sqrt{2-\sqrt{2}}=F_3$
	$-\sqrt{2-\sqrt{2}}=-F_3$

를 갖는 프로세서에서 Cooley-Tukey 알고리즘과 Bruun 알고리즘이 수행하는 벡터 단위의 데이터 정렬 횟수에 대한 비교가 추가로 필요하다. 이에 대한 내용은 <그림 8>에 보인 실험 결과에 포함되어 있다.

4) 필터 계수 생성 방법의 비교

FFT 알고리즘에서는 연산에 사용되는 필터 계수가 많기 때문에 메모리 사용을 줄이기 위해서 초기 값을 이용하여 이후에 필요한 필터 계수들을 생성하는 방법이 선호된다. 그러나 Bruun 알고리즘은 <표 1>에서 보인 것과 같이 이전 단의 계수 값을 이용해서 다음 단의 실수 필터 계수를 생성하려면 제곱근 연산을 수행해야 한다. 이와 같은 특징은 SIMD 구조의 프로세서를 구현할 때 하드웨어의 복잡도를 급격히 증가시킨다. 이런 문제를 해결하기 위해 본 논문에서는 Bruun 알고리즘의 연산에 사용되는 전체 필터 계수를 미리 계산하여 메모리에 저장하는 방법을 사용한다.

Cooley-Tukey 알고리즘에서는 현재 단의 연산에서 사용한 복소수 필터 계수와 실수 곱셈기를 이용하여 다음 단의 연산에 사용할 필터 계수를 쉽게 생성할 수 있다. 예를 들어, SIMD 데이터패스의 폭이 2인 프로세서가 <그림 1>의 Cooley-Tukey 알고리즘을 수행한다고 가정하면, 첫 번째 단에서 W^0, W^1을 이용한 연산 이후 W^2와 W^3을 이용한 연산을 수행해야 하는데, W^0, W^1에 W^2를 곱하면 다음 SIMD 연산을 위한 $W^2(=W^{(0+2)})$와 $W^3(=W^{(1+2)})$을 생성할 수 있다. 이 특성은 동일한 단의 연산이 종료될 때까지 동일하게 적용된다.

2. 실험용 SIMD 프로세서

두 FFT 알고리즘을 SIMD 구조의 프로세서에서 구현하였을 때의 성능과 효율성을 비교하기 위해 <그림 3>과 같은 구조의 실험용 SIMD 프로세서를 기준 하드웨어로 가정하였다. SIMD 프로세서는 신호처리 알고리즘의 대부분을 차지하는 벡터 연산을 효과적으로 처리할 수 있기 때문에 디지털 신호처리 분야에 널리 사용되고 있다[14~16].

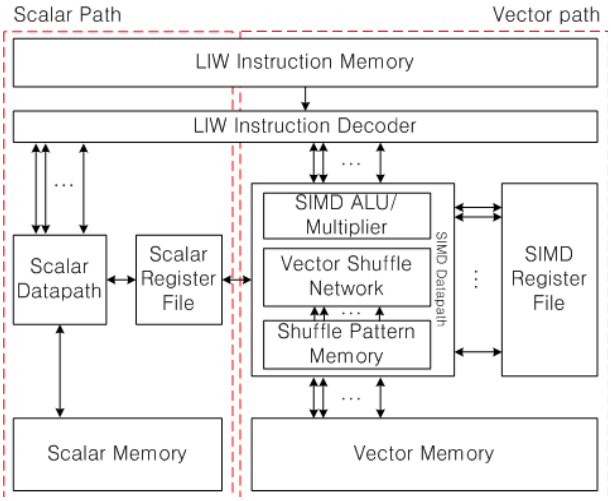

그림 3. 실험용 SIMD 프로세서의 구조
Fig. 3. Architecture of experimental SIMD processor.

1) 구조

실험용 SIMD 프로세서는 SIMD 데이터패스, 스칼라 데이터패스, 제어장치, 명령어 메모리, 벡터 메모리, 스칼라 메모리 등 6가지 장치로 구성되어 있다. SIMD 데이터패스는 벡터 연산을 담당하며 스칼라 데이터패스는 병렬처리가 어려운 순차 연산과 프로그램의 흐름 제어 동작을 수행한다. 명령어 메모리는 응용프로그램을 저장하고, SIMD 메모리는 SIMD 데이터패스의 데이터를 저장하고 공급하며, 스칼라 메모리는 스칼라 데이터패스에서 사용할 데이터를 저장하고 공급한다.

SIMD 데이터패스는 SIMD 레지스터 파일, 32개의 연산장치로 구성되는 SIMD 연산장치, 벡터 조건 메모리, 벡터 데이터 정렬 장치, 벡터 정렬 메모리로 구성되어 있다. 위 구성 요소들은 병렬로 동작하며 파이프라인을 형성한다. SIMD 연산장치와 벡터 조건 메모리는 논리/산술 연산에 사용된다. SIMD 레지스터 파일은 SIMD 연산장치에 입력 데이터를 공급하고, 연산 결과를 임시 저장하는 역할을 담당한다. 벡터 데이터 정렬

장치와 벡터 정렬 메모리는 벡터 형태로 정렬된 입력 데이터의 순서를 짧은 시간에 재정렬 하는데 사용된다. FFT 알고리즘의 경우 한 단의 출력 결과를 다음 단의 입력으로 제공하기에 앞서 벡터 데이터의 순서를 변경할 때 이 장치를 사용한다.

2) 명령어

<표 2>에서 보인 것과 같이 실험용 SIMD 프로세서에 구현된 명령어들은 크게 벡터 정렬 명령어군, 벡터 연산 명령어군, 벡터 메모리 명령어군 세 종류로 구분될 수 있다.

벡터 정렬 명령어군에는 vshuf 명령어가 속한다. 이 명령어는 특수 목적 메모리인 벡터 정렬 메모리에 저장된 내용에 따라 벡터 데이터 정렬 형태가 정해진다. 따라서 이 메모리의 내용을 변경하여 벡터 데이터의 정렬 형태를 자유롭게 변경할 수 있다. 벡터 정렬 메모리에는 다수의 데이터 정렬 형태가 저장되는데, 스칼라 레지스터 r15를 이용하여 이들 가운데 한 데이터 정렬 형태를 선택하여 SIMD 연산장치에 제공할 수 있다.

벡터 연산 명령어군에는 일반적인 실수 덧셈, 뺄셈, 곱셈, 쉬프트 명령어가 포함된다. 특징적인 명령어는 vcadd로 특수 메모리인 벡터 조건 메모리의 값에 따라 선택적으로 덧셈 혹은 뺄셈 동작을 수행한다. 벡터 조건 메모리에 사전에 계산된 조건 값을 적정하게 설정하면 조건 비교 명령이 필요하여 병렬화하기 어려운 연산을 효과적으로 SIMD 프로세서에서 처리할 수 있다. 이 방법은 연산 시간을 단축시키지만 그 대가로 조건 값을 저장하기 위한 메모리를 더 필요로 한다. 벡터 조건 메모리에도 다수의 조건 값이 저장되는데 스칼라 레지스터인 r14를 이용하여 이들 가운데 하나를 선택한다.

벡터 메모리 명령어군은 SIMD 레지스터 파일에 저장된 벡터 데이터를 SIMD 메모리에 저장하는 vstr 명령어와, SIMD 메모리의 데이터를 SIMD 레지스터 파일로 읽어 저장하는 vld 명령어로 구성된다.

3. 실험용 SIMD 프로세서에서 FFT 구현

1) 벡터 데이터 정렬의 구현

<그림 4>는 Bruun 알고리즘의 데이터 정렬동작을 실험용 SIMD 프로세서에서 구현된 어셈블리 명령어로 구현한 예이다. 이 예에서 왼쪽 명령어는 SIMD 데이터패스에서 실행되며 오른쪽 명령어는 스칼라 데이터패스에서 실행된다. 이 예제에서 r1과 r2는 Bruun 알고리즘에서 나타나는 2가지 데이터 정렬 형태가 저장된 벡터 정렬 메모리의 주소를 가리키는데 사용된다. r1은 <그림 2>의 Bruun 알고리즘 신호 흐름도에서 실선으로 표현된 벡터 데이터 정렬 형태를 가리키며, r2는 점선으로 표현된 벡터 데이터 정렬 형태를 나타내고 있다. 이 예

```
vld VR0 [r0]          | nop
vld VR1 [r0, 64]      | mov r1 r15
vshuf VR2 VR0         | nop
vshuf VR3 VR1         | mov r2 r15
vshuf VR4 VR0         | nop
vshuf VR5 VR1         | nop
```

그림 4. Bruun 알고리즘의 벡터 데이터 정렬을 위한 어셈블리 프로그램
Fig. 4. Assembly code for vector data alignment in Bruun's FFT algorithm.

표 2. SIMD 데이터패스를 제어하는 실험용 SIMD 프로세서의 명령어들
Table 2. Instructions of the experimental SIMD process that control its SIMD datapath.

Category	Instruction	Description
Vector shuffle	vshuf VR1 VR0	v. shuffling: VR1[i] = VR0[suffle_mem[r15][i]]
Vector arithmetic	vmult VR2 VR0 VR1	v. multiplication: VR2[i] = VR0[i] * VR1[i]
	vcadd VR2 VR0 VR1	v.conditional add: VR2[i] = con_mem[r14][i] ? (VR0[i]+VR1[i]):(VR0[i]-VR1[i])
	vlsl VR1 VR0 r0	v. logical shift left: VR1[i] = VR0[i] << r0
	vlsr VR1 VR0 r0	v. logical shift right: VR1[i] = VR0[i] >> r0
	vadd VR2 VR0 VR1	v. addition: VR2[i] = VR0[i] + VR1[i]
	vsub VR2 VR0 VR1	v. subtraction: VR2[i] = VR0[i] - VR1[i]
Vector memory	vld VR0 [rn, imm]	v. load: VR0 = vmem[rn + imm]
	vstr VR0 [rn, imm]	v. store: vmem[rn + imm] = VR0

에서 r1과 r2 값이 r15에 옮겨져서 필요한 정렬 형태가 벡터 정렬 회로에 공급되도록 한다. 정렬된 실수부와 (VR0) 허수부는(VR1)는 VR2~VR5에 저장된다.

2) 조건부 연산

<그림 5>는 Bruun 알고리즘의 조건부 연산을 이용하여 butterfly 연산을 구현한 예를 보여준다. 이 예에서 r13은 입력 데이터가 저장된 주소를, r7과 r8은 연산 결과를 저장할 주소를 각 각 나타낸다. r14는 스칼라 데이터패스에서 "벡터 조건 메모리"에서 적정한 조건데이터를 읽는데 필요한 주소를 담고 있다. 입력 데이터는 vcadd, vmult, vadd 연산들을 이용해 처리된다.

```
vlw VR6 [r13, 4]          | mov r3 r14
vcadd VR7 VR2 VR0         | nop
vcadd VR8 VR3 VR1         | nop
vmult VR9 VR4 VR6         | nop
vmult VR10 VR5 VR6        | nop
vadd VR7 VR7 VR9          | nop
vadd VR8 VR8 VR10         | nop
vstr VR7 [r0]             | nop
vstr VR8 [r0, 64]         | nop
```

그림 5. Bruun 알고리즘의 butterfly 연산을 위한 어셈블리 프로그램

Fig. 5. Assembly code for butterfly computation in Bruun's FFT algorithm.

3) 두 FFT 알고리즘 구현에 필요한 메모리 크기

두 FFT 알고리즘의 구현 과정에서 i) 벡터 정렬 정보, ii) 조건부 연산 조건 정보, iii) 계수 정보의 저장에 데이터 메모리가 필요하다.

실험용 SIMD 프로세서에서 수행되는 알고리즘의 종류에 따라 필요한 벡터 정렬 형태가 다르므로 벡터 정렬 메모리에 저장되는 정보의 크기도 달라진다.

$$M_S^{CT} = B \times W log_2 W \tag{3}$$

$$M_S^B = B \times (2W \log_2 W + 2W) \tag{4}$$

식 (3)은 Cooley-Tukey 알고리즘을 위해 필요한 정렬 형태 메모리의 크기를 나타낸다. $W\log_2 W$ 항은 butterfly 연산을 위한 SIMD 데이터 정렬 형태의 개수로, $\log_2 W$ 개의 단 동안 W개의 SIMD 데이터를 재배열하기 위해 필요한 벡터 데이터 정렬 장치 설정 값의 수를 의미한다. B 항은 데이터의 비트(bit) 수를 나타낸다.

식 (4)는 Bruun 알고리즘에서 벡터 데이터 정렬 형태를 표현하는데 필요한 메모리 크기를 나타낸다. 이 식에서 $2W\log_2 W$ 항은 butterfly 연산에 필요한 데이터 정렬 형태의 수이다. 2W 항은 FFT의 길이가 SIMD의 폭보다 길 때 필터 계수 연산에 필요한 데이터 정렬 형태를 나타낸다.

$$M_C = W \log_2 W \tag{5}$$

식 (5)는 두 FFT 알고리즘의 수행에 사용되는 조건부 연산에 필요한 조건 정보의 크기를 나타낸다. 첫 번째 단부터 ($\log_2 N$-$\log_2 W$) 번째 단까지 수행하는 조건부 연산은 덧셈과 뺄셈 연산만으로 수행이 가능하다. 예를 들어 <그림 2>에서 보인 Bruun 알고리즘의 연산을 SIMD 폭이 8인 프로세서에서 처리한다고 가정하면, 첫 번째 단에서 0~7번째 연산은 덧셈이며, 8~15 번째 연산은 뺄셈이다. 이는 SIMD 덧셈 명령어와 SIMD 뺄셈 명령어로 처리가 가능하다. 이 특징은 두 알고리즘에 동일하게 적용된다. 따라서 마지막 $\log_2 W$ 개의 단을 위한 조건부 연산에 대해서만 조건 설정 값이 필요하다.

식 (6)과 (7)은 두 알고리즘의 처리에 필요한 필터 계수용 메모리의 크기를 계산한 결과를 나타낸다.

$$M_{Coef}^{CT} = 32 \cdot (N + W \cdot (\log_2 W - 1)) \tag{6}$$

$$M_{Coef}^{B} = 16 \cdot (2N + N(\log_2 N - 2) - \sum_{k=1}^{\log_2 N/W} \frac{N}{2^k}) \tag{7}$$

4) 전체 메모리의 크기 비교

<그림 6>은 실험용 SIMD 프로세서에서 두 FFT 알고리즘을 구현할 때 필요한 메모리의 총 합을 보여준다. 이 그림에서 X-축은 처리되는 FFT 입력 데이터의 길이이며, Y-축은 필요한 메모리의 크기를 비트 단위로 나타낸 것이다.

이 그림을 살펴보면 FFT의 길이가 32일 때 Cooley-Tukey 알고리즘과 Bruun 알고리즘을 수행하는데 필요한 메모리의 크기는 비슷하다. 그러나 FFT의 길이가 길어질수록 Cooley-Tukey 알고리즘의 수행에 필요한 메모리의 크기는 선형적으로 증가하는 반면,

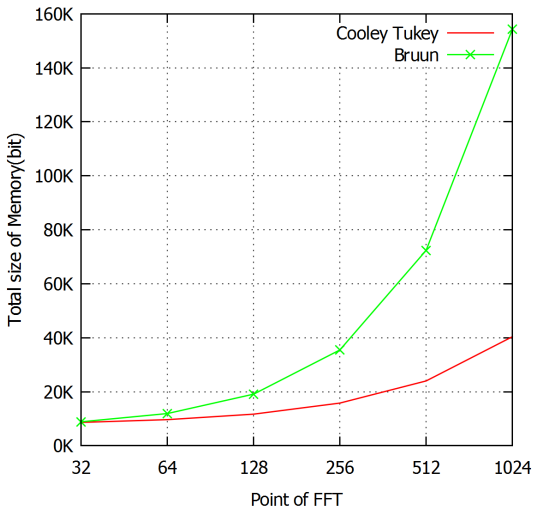

그림 6. Bruun 알고리즘과 Cooley-Tukey 알고리즘의 수행에 필요한 메모리의 크기 분석결과
Fig. 6. Total size of memory for Bruun FFT and Cooley-Tukey FFT algorithms.

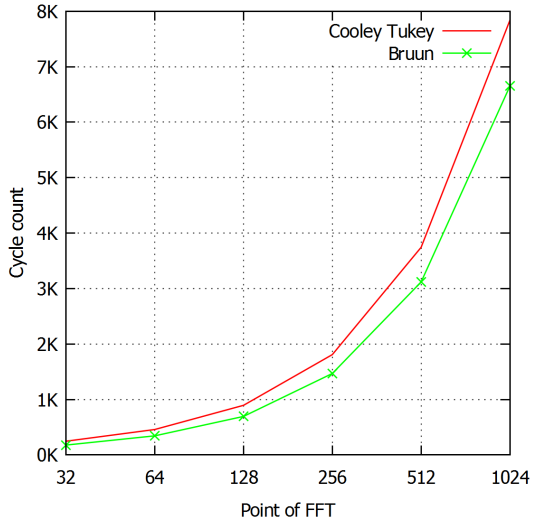

그림 7. Bruun 알고리즘과 Cooley-Tukey 알고리즘의 처리 시간 비교
Fig. 7. Comparison of processing time between Bruun FFT and Cooley-Tukey FFT algorithms.

Bruun 알고리즘은 지수함수 형태로 증가한다. 이와 같은 차이는 계수 저장에 필요한 메모리의 크기가 Cooley-Tukey 알고리즘의 경우 SIMD의 폭을 나타내는 W의 함수인 $W\log_2 W$로 표현되는데 반해, Bruun 알고리즘의 경우 FFT의 길이를 나타내는 N의 함수 $N\log_2 N$으로 표현되기 때문이다. 따라서 두 FFT 알고리즘이 필요로 하는 메모리 크기의 차이는 FFT의 길이가 증가함에 따라 점차 증가하며, 길이가 1024일 때 Bruun 알고리즘이 Cooley-Tukey 알고리즘에 비해 약 4배의 메모리를 더 필요로 하게 된다.

Ⅲ. 실 험

이 장에서는 Bruun 알고리즘과 Cooley-Tukey 알고리즘을 동일한 조건의 SIMD 프로세서에서 수행한 결과를 서로 비교한다. 이를 위해 실험용 SIMD 프로세서를 이산 사건 시뮬레이션 라이브러리인 SSIM을 이용하여 모사하였다.[13] FFT 프로그램은 <표 2>에서 보인 SIMD 어셈블리 명령어와 스칼라 어셈블리 명령어를 이용하여 구현하였다.

구현된 FFT 알고리즘은 32, 64, 128, 512, 1024 개의 복소수 데이터를 입력으로 하여 구동되었고 <그림 7>에서 보인 것과 같이 그 처리 시간이 사이클 단위로 측정되었다. <그림 8>은 더 나아가서 Bruun 알고리즘의 처리 시간을 Cooley-Tukey 알고리즘의 경우를 기준으로 하여 그 비율을 비교한 것이다. 또한, <그림 9>는 Bruun 알고리즘과 Cooley-Tukey 알고리즘을 수행하기 위한 사용된 SIMD 명령어의 빈도수를 측정한 것이다.

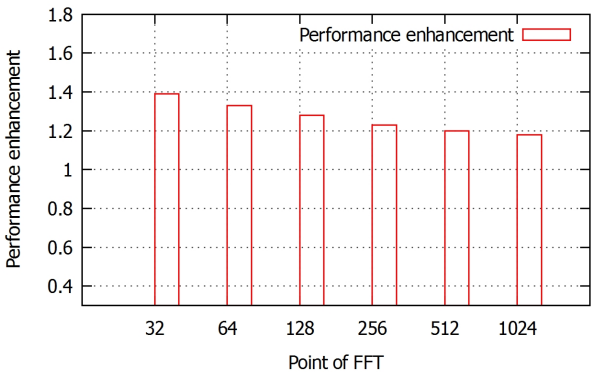

그림 8. Cooley-Tukey 알고리즘에 대비한 Bruun 알고리즘의 성능 향상도 비교
Fig. 8. Comparison of performance enhancement of Bruun FFT versus Cooley-Tukey FFT algorithms.

<그림 7>에서 길이가 32인 FFT에서 Cooley-Tukey 알고리즘의 실행 사이클 수는 250인 반면, Bruun 알고리즘의 실행 사이클 수는 180이다. 이를 <그림 8>의 비율로 따져보면 약 40%의 성능 향상이 이루어진 것이다. 이와 같은 결과는 <그림 9>에서 보인 것과 같이 Bruun 알고리즘에서 사용하는 실수 곱셈을 포함한 벡터 연산 명령어와 벡터 메모리 명령어의 수가 Cooley-Tukey 알고리즘에 비해 상대적으로 적기 때문

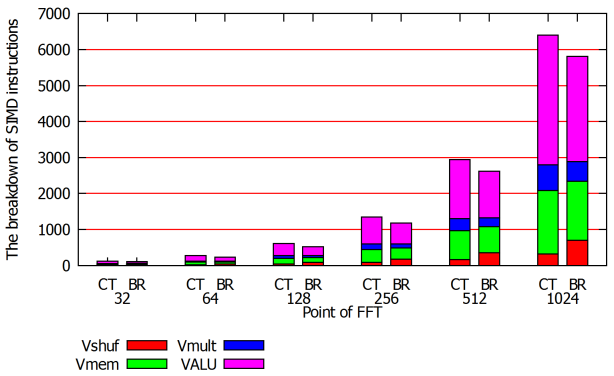

그림 9. Cooley-Tukey 알고리즘과 Bruun 알고리즘 수행을 위한 SIMD 명령어의 사용 횟수

Fig. 9. Breakdown of SIMD instructions for Cooley Tukey FFT and Bruun FFT algorithms.

이다. 두 FFT 알고리즘의 실행 사이클 수 차이는 FFT의 길이에 비례하여 증가하며, 길이 1024 인 FFT에서는 Bruun 알고리즘이 Cooley-Tukey 알고리즘에 비해 약 1200 사이클 빨리 처리함 알 수 있다. 이 수치를 <그림 8>의 비율로 따져보면 약 20%의 성능 향상이 이루어진 것이다. 따라서 FFT의 길이가 길어질수록 그 향상의 정도가 낮아짐을 알 수 있다.

<그림 6>에서 보인 분석 결과와 <그림 7>과 <그림 8>에서 보인 실험 결과를 종합하면, 구현된 Bruun 알고리즘은 Cooley-Tukey 알고리즘에 비하여 데이터 메모리를 더 필요로 하지만 그 처리 성능은 더 높음을 알 수 있다. Bruun 알고리즘이 필요로 하는 메모리의 크기는 이동통신 시스템 등에서 많이 쓰이는 1024 개의 복소수 입력데이터를 처리하는 경우 약 20Kbyte (=160Kbit/8) 이다. 이 크기의 데이터 메모리는 지금의 반도체 기술을 활용하면 고속의 Onchip 메모리 형태로 구현하는 것이 가능하다. 따라서 SIMD 구조의 프로세서에서 하드웨어 추가 없이 Bruun 알고리즘을 적용하여 FFT 처리 성능을 향상시킬 수 있다.

Ⅳ. 결 론

본 논문에서는 Bruun 알고리즘을 SIMD 구조를 갖는 프로세서에서 구현하기 위한 방법과 그 결과에 대해 살펴보았다. Bruun 알고리즘은 복잡한 복소수 곱셈 대신 간단한 실수 곱셈으로 FFT 동작을 처리하기 위해 고안된 알고리즘이다. Bruun 알고리즘을 순차적으로 작업을 진행시키는 스칼라 프로세서에서 활용할 경우 감소된 복소수 곱셈의 수 때문에 Cooley-Tukey 알고리즘에 비해 성능이 향상되는 이점이 있으나, SIMD 구조를 갖는 프로세서에서 수행시킬 경우는 그 효율성이 입증되지 않았다. 본 논문에서는 SIMD 구조를 갖는 프로세서에서 Bruun 알고리즘을 수행하기 위한 프로그래밍 방법을 제안하고, 동일한 조건에서 Cooley-Tukey 알고리즘을 수행한 결과와 비교하였다. 결과를 살펴보면 1024 개의 복소수 입력 데이터에 대한 FFT 연산에서 Bruun 알고리즘은 Cooley-Tukey 알고리즘에 비하여 약 4 배의 (약 20Kbyte)의 메모리를 필요로 하지만 처리 성능은 약 20% 높다. 따라서 사용하려는 SIMD 프로세서가 충분한 메모리를 가진다면 Bruun 알고리즘을 적용하여 성능 향상을 도모할 수 있다.

REFERENCES

[1] James W. Cooley and John W. Tukey, An Algorithm for the Machine Calculation of Complex Fourier Series, Mathematics of computation 19.90, pp.297-301, 1965.

[2] S. C. Chan and K. L. Ho, On Indexing the Prime Factor Fast Fourier Transform Algorithm, IEEE Transactions on Circuits and Systems, Vol. 38, No, 8, pp.951-953, 1991.

[3] Georg Bruun, z-Transform DFT Filters and FFT's, IEEE Transactions on Acoustics, Speech, And Signal Processing, Vol. 26, NO. 1, February, 1978.

[4] Rader, C.M., Discrete Fourier transforms when the number of data samples is prime, IEEE, Proceedings letters, No. 56, pp.1107-1108, 1968.

[5] Wang Xu, Zhang Yan and Ding Shunying, A High Performance FFT Library with Single Instruction Multiple Data(SIMD) Architecture, IEEE, International Conference on ICECC, pp.630-633, September, 2011.

[6] Ting Chen, Hengzhu Liu and Botao Zhang, A scalable, fixed-shuffling, parallel FFT butterfly processing architecture for SDR environment, IEICE Electronics Express, Vol.11, No.2, pp.1-9, 2014.

[7] T. Chen, X. Pan, H. Liu and T. Wu, Rapid Prototype and Implementation of a High-Throughput and Flexible FFT ASIP Based on LISA 2.0, IEEE, 15th International Symposium on ISQED, 2014.

[8] F. Yu, R. GE and Z. Wang, Efficient Utilization

of Vector Registers to Improve FFT Performance on SIMD Microprocessors, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E960A, No.7, July, 2013.
[9] Mittal Shashank. Efficient and High-Speed FFT Architectures for Software Defined Radio, Master Thesis. International Institute of Information Technology Hyderabad, INDIA, 2009.
[10] Yuhang Wu, New FFT Structures Based on the Bruun Algorithm, IEEE Transactions On Acoustics, Speech. And Signal Processing, Vol. 38. No. 1, pp.188-191, January, 1990.
[11] Harold S. Stone, Parallel processing with the perfect shuffle, IEEE Transactions on Computers, Vol. 20, No. 2 pp.153-161, 1971.
[12] Mittal, S., Area Efficient High Speed Architecture of Bruun's FFT for Software Defined Radio, IEEE, GLOBECOM '07, Global Telecommunications Conference, 2007.
[13] C. Antonio, SSim - A Simple Discrete-Event Simulation Library(2012), Retrieved Febuary, 2012, from http://www.inf.usi.ch/carzaniga/ssim/index.html
[14] Sehoon Yoo, A Reconfigurable Parallel Processor for Efficient Processing of Mobile Multimedia, Journal of the Institute of Electronics Engineers of Korea SD, Vol. 44, No. 10, pp.23-32, 2007.
[15] Kyeong-Seob Kim, Yun-Sub Lee, Byung-Cheol Yu, Control Unit Design and Implementation for SIMD Programmable Unified Shader, Journal of the Institute of Electronics Engineers of Korea SD, Vol. 48, No. 7, pp.37-47, 2011.
[16] Hillery C. Hunter, A new look at exploiting data parallelism in embedded systems, CASES '03 Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems, pp.159-169, 2003.

저 자 소 개

이 주 영(학생회원)
2013년 광운대학교 전자통신 공학과 공학사.
<주관심분야 : 프로세서, VLSI>

이 현 석(정회원)
1992년 KAIST 전기및전자공학과 공학사.
1995년 POSTECH 전자전기과 공학석사.
2007년 Univ. of Michigan, Ann Arbor, CSE, Ph.D.
1992년～2008년 삼성전자 통신연구소 수석연구원
2008년～현재 광운대학교 전자통신공학과 부교수
<주관심분야 : 유선통신망, 무선통신망, 프로세서, VLSI>

홍 용 근(학생회원)
2014년 광운대학교 전자통신 공학과 공학사.
<주관심분야: 프로세서, VLSI>