Journal of Digital Contents Society Vol. 16 No. 5 Oct. 2015(pp. 805-813)
http://dx.doi.org/10.9728/dcs.2015.16.5.805

단일 명령 다중 스레드 병렬 플랫폼을 위한 무작위 부분적 Haar 웨이블릿 변환

박태정*

요 약

Compressive sensing 및 희소 복원 문제(sparse recovery problem)는 기존 디지털 기술의 한계를 극복할 수 있는 새로운 이론으로 많은 관심을 받고 있다. 그러나 신호 재구성에서 l1 norm 최적화 문제 해결에 많은 연산이 수행되며 따라서 병렬 처리 기법이 필요하다. 이 과정에서 무작위 행렬과 벡터 연산을 통한 변환 연산이 전체 과정 중에서 많은 부분을 차지하는데, 특히 원본 신호의 크기로 인해 이 과정에서 필요한 무작위 행렬을 메모리에 저장하기 곤란하며 계산 시 무작위 행렬의 절차적(procedural) 처리 방식이 필수적이다. 본 논문에서는 이 문제에 대한 해결책으로 단일 명령 다중 스레드(SIMT) 병렬 플랫폼 상에서 무작위 부분적 Haar 웨이블릿 변환을 절차적으로 계산할 수 있는 새로운 병렬 알고리듬을 제안한다.

키워드 : 절차적 Haar 웨이블릿, 압축 센싱, 병렬 처리, CUDA, GPU, 희소 신호 복원

Random Partial Haar Wavelet Transformation for Single Instruction Multiple Threads

Taejung Park*

Abstract

Many researchers expect the compressive sensing and sparse recovery problem can overcome the limitation of conventional digital techniques. However, these new approaches require to solve the l1 norm optimization problems when it comes to signal reconstruction. In the signal reconstruction process, the transform computation by multiplication of a random matrix and a vector consumes considerable computing power. To address this issue, parallel processing is applied to the optimization problems. In particular, due to huge size of original signal, it is hard to store the random matrix directly in memory, which makes one need to design a procedural approach in handling the random matrix. This paper presents a new parallel algorithm to calculate random partial Haar wavelet transform based on Single Instruction Multiple Threads (SIMT) platform.

Keywords : procedural Haar wavelet, compressive sensing, parallel processing, CUDA, GPU, sparse signal recovery

1. 서론

1.1 연구 배경

2006년에 발표된 compressed sensing (또는 compressive sensing) [1] 이론은 현대 디지털 기술의 이론적인 근간인 Nyquist-Shannon 표본화 정리(sampling theorem)에 의한 근원적인 한계를 극복할 수 있는 대안으로 많은 연구자들의 관심을 받고 있다. 이 이론은 먼저 가급적 높은 해상도로 표본을 수집한 후 불필요한 정보를 제거하는 과정, 즉, 압축을 수행하는 일

※ Corresponding Author : Taejung Park
Received : September 01, 2015
Revised : October 25, 2015
Accepted : October 31, 2015
* Dept. of Digital Media,
Duksung Women's University
Tel: +82-2-901-8339, Fax: +82-2-901-8646
email: tjpark@duksung.ac.kr
▣ This Research was supported by the Duksung Women's University Research Grants 2014 (No. 3000002224).

반적인 디지털 신호 처리에서의 신호 표본화(sampling)와 복원(recovery) 과정과는 달리 표본화와 동시에 무작위 표본화 작업을 통해 압축이 수행되고 복원 시에는 희소한 표본 정보만을 이용해서 원본 신호를 복원하는 과정을 거친다. 이러한 과정으로 수행되는 Compressive sensing 프레임워크에서는 매우 간단하고 빠르게 수행 가능한 표본화 과정과는 달리, 일반적인 디지털 신호 처리에 비해 신호 복원 과정에서 많은 연산이 필요하다. 이 신호 복원 문제는 벡터의 ℓ_0 norm [2] 최소화 문제로 정리되는데 이 최소화 문제는 NP-hard라고 알려져 있다. 그러나 ℓ_0 norm을 ℓ_1 norm으로 대체하면 볼록 최적화(convex optimization) 문제가 된다는 사실이 증명되었고 compressive sensing 이론 역시 대부분 ℓ_1 norm 최소화 문제를 다룬다. 그러나 ℓ_1 norm 최소화 문제는 컴퓨터 그래픽스를 포함한 많은 분야에서 다루는 에너지 최소화 문제인 ℓ_2 norm 최소화에 비해서 여전히 많은 연산이 필요하며 몇몇 연구자들이 병렬 처리 기법을 제안했다. 이러한 연구들 중에 Andrecut는 NVIDIA사의 GPU(Graphics Processing Unit)와 CUDA(Compute Unified Device Architecture) [3][4]을 이용한 방법을 제안했다. 이 방식에서는 CUDA의 일부로 제공되는 병렬 선형 방정식 계산 라이브러리인 CUBLAS를 활용한다. 그러나 광선 추적(ray-tracing)에서 compressive sensing을 적용한 Sen과 Darabi의 연구[5]에서 논의한 것처럼, 실제 적용에서는 Andrecut가 [6]에서 제안한 방식처럼 GPU 메모리 상에서 직접적으로 공간을 할당하고 무작위 행렬(random matrix)를 저장하는 방식은 행렬의 크기가 처리 불가능할 정도로 커야하기 때문에 GPU 메모리 상에 저장 자체가 불가능하고 절차적인 알고리듬을 적용할 필요가 있다. 그러나 Sen과 Darabi의 연구[5]에서는 구체적인 무작위 행렬의 절차적 알고리듬을 제시하지 않았다.

이러한 배경 하에서 이 논문은 SIMT (Single Instruction Multiple Threads) 병렬 프레임워크에서 희소 복원 문제를 해결하는 여러 알고리듬의 구현에서 가장 중요한 무작위 부분적 Haar 웨이블릿 변환을 수행하는 새로운 병렬 처리 알고리듬을 제안한다.

1.2 희소 복원 문제 (Sparse Recovery Problem)

원본 신호 벡터 $\mathbf{x} \in \mathbf{R}^M$의 원소 중에서 K개만 0이 아닐 때 이 벡터를 K-sparse라고 정의한다(단, K<<M). 이 경우, 이 희소(sparse) 신호 $\mathbf{x}$에 대해 다음과 같은 인코딩/디코딩이 가능하다.

○ 희소 신호 벡터 $\mathbf{x}$는 특정한 조건[7]을 준수하는 N x M 크기의 무작위 행렬 Ψ를 이용해서 보다 작은 차원의 벡터 $\mathbf{y} \in \mathbf{R}^N$로 인코딩할 수 있다(K<N<M).

$$\mathbf{y} = \Psi\mathbf{x} \in \mathbf{R}^N$$

이렇게 인코딩되어 크기가 축소된 신호 $\mathbf{y}$를 수신측에게 전달한다.

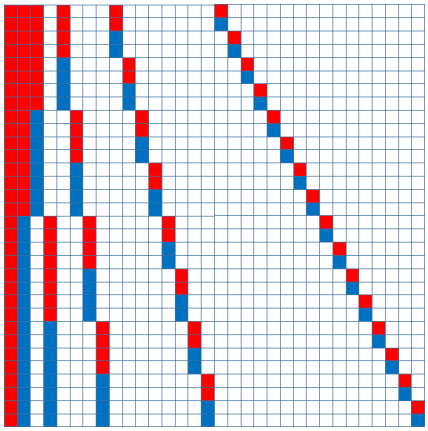
○ 수신측에는 인코딩 신호 $\mathbf{y}$가 전달되며 이미 무작위 행렬 Ψ를 파악하고 있다고 가정한다. $\mathbf{y}$와 Ψ로부터 원본 신호 벡터 $\mathbf{x}$를 복원하는 과정이 디코딩 과정에 해당되는데, 행렬 Ψ가 역행렬을 가지는 정방행렬이 아니기 때문에 일반적으로 선형대수에서 다루는 방식으로는 이 신호를 복원할 수 없고 보다 특수한 해법이 필요하다. 이 희소 복원 문제를 풀기 위한 대표적인 방법들 중 하나는 Matching Pursuit (MP) 기법으로, Andrecut가 [6]에서 정리한 유사 코드를 표 1에서 정리한다. 표에서 Ψ_i는 행렬 Ψ의 i 번째 열 벡터(column vector)를 의미하고 $\mathbf{x}_i$는 벡터 x의 i 번째 원소를 의미한다.

<표 1> 일반적인 Matching Pursuit Method. 자세한 내용은 [6] 참고

```
1. Initialize Variables:
   t = 0, x_0 = 0, r = y, set T
2. While ||r||_2 > ε||y||_2 and t < T, then repeat
   {
      Find i such that
            i = argmax_i |<r, Ψ_i>|
      x_i = x_i + <r, Ψ_i>
      r = r - <r, Ψ_i>Ψ_i
      t = t + 1
   }
3. Return x_i
```

<Table 1> Matching Pursuit (MP) Method. See [6] for details.

(그림 1) 32 x 32 Haar 웨이블릿 행렬



(Figure 1) 32 x 32 Haar Wavelet Matrix

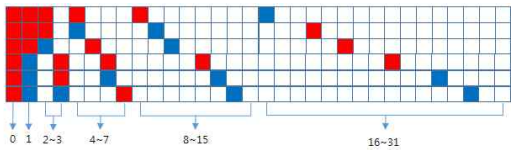
Sen과 Darabi는 compressive sensing 이론의 맥락 하에, 논의한 희소 신호 벡터의 인코딩/디코딩 과정을 레이트레이싱에 적용하면서, 렌더링 이미지의 모든 픽셀에서 광선을 추적하는 일반적인 방식과 달리, 무작위 가우시안 행렬에 측정 행렬(measure matrix)이라는 개념을 도입해서 NxM 크기의 무작위 생성 행렬 **Ψ**를 얻고 전체 픽셀 개수에 해당하는 M (즉, 신호 x의 차원) 번이 아닌 K 번만 광선 추적을 수행해서 전체 프로세스 시간을 단축했다(∵ K<<M)[5]. 이 방식에서 정방행렬 형태인 무작위 가우시안 행렬에 측정 행렬을 곱한 결과로 원래 정방행렬에서 무작위로 선택한 N 개 행을 가지는 행렬(즉, NxM)을 얻는다. Sen과 Darabi는 상당히 큰 M 때문에 무작위 행렬 전체를 메모리에 넣는 것이 비현실적이며 따라서 절차적(procedural) 알고리듬을 사용했다고 밝혔다(1024x1024 이미지의 경우 $M=2^{20}$). 그러나 이 논문에서는 MP가 아닌 ROMP(Regularized Orthogonal Matching Pursuit)로 구현했고 병렬 방식이 아니었으며 구체적인 알고리듬도 논문에서 언급하지 않았다.

본 논문에서는 GPU 병렬 프로그래밍 플랫폼으로 널리 사용되는 SIMT (Single Instruction Multiple Threads) 플랫폼을 위해, 정방행렬 형태의 Haar 웨이블릿 행렬에서 무작위 샘플링을 적용해서 무작위 행렬 **Ψ**를 절차적으로 처리해서 <표 1>에 제시한 MP 복원 방법 중 가장 연산량이 많은 웨이블릿 변환(즉, <**r**, **Ψ**$_T$>) 연산의 새로운 병렬 처리 알고리즘을 제안한다.

1.3 Single Instruction Multiple Threads (SIMT)

병렬 처리 기술의 범주에는 벡터 머신과 같이 단일 명령으로 여러 데이터를 처리하는 SIMD (Single Instruction Multiple Data), 여러 명령으로 여러 데이터를 처리하는 MIMD (Multiple Instructions Multiple Data) 등이 있다. NVIDIA사의 CUDA 프레임워크는 SIMD와 비슷하지만 최소 처리 단위인 스레드마다 별도의 레지스터와 명령 주소 카운터를 가지고 있다는 점에서 다른 SIMT (Single Instruction Multiple Threads) 방식을 적용한다[8]. 이 방식은 SIMD와 비슷하면서도 경우에 따라 MIMD의 특징을 활용할 수 있다는 점에서 장점을 가지고 있다. 이러한 SIMT 상에서 구축된 CUDA 라이브러리는 기본적으로 C 언어를 기초로 하는 프레임워크이며 최근 C++ 언어를 도입하고 더 나아가 Thrust라는 라이브러리를 통해 C++의 제너릭(generic) 프로그래밍 구현을 지원한다.

(그림 2) 무작위 측정을 통해 추출된 부분적 Haar 웨이블릿 행렬



(Figure 2) Randomly measured partial Haar wavelet matrix

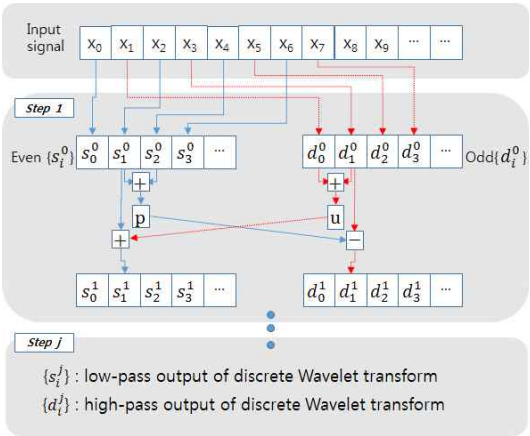
2. 부분적 Haar Wavelet 무작위 프로젝션

2.1 절차적 무작위 Haar Wavelet 행렬

제안하는 방법의 설명을 위해서 직관적으로 이해 가능한 32 x 32 크기의 일반적인 Haar

wavelet 행렬에서 무작위 행렬을 추출하는 예를 다룬다. (그림 1)에서는 32 x 32 Haar wavelet을 도식적으로 제시하고 있다. 이 그림

(그림 3) 일반적인 웨이블릿 변환을 위한 lifting 방법



(Figure 3) The lifting scheme of solving general wavelet transform

에서 빨간색 사각형은 +1, 파란색 사각형은 -1, 그리고 빈 사각형은 0을 의미하며 일반적으로 이 그림에서 세로줄에 해당하는 열 벡터는 normalize를 거쳐 행렬 내부의 각 값이 결정된다. 본 논문에서는 편의상 normalize를 하지 않은 상태로 설명한다. Haar 웨이블릿에 대한 자세한 내용은 [9]를 참고한다.

(그림 2)에서는 compressive sensing 이론에 의한 무작위 측정으로 추출한 부분적 Haar 웨이블릿 행렬, Ψ를 제시한다. 이 때 (그림 2)의 왼쪽에 세로로 제시된 숫자는 원래의 32 x 32 행렬에서의 행 번호(0부터 시작)를 의미한다. 즉, 이 예제에서는 원래 행렬에서 1, 6, 10, 16, 23, 27 번째 행이 무작위 측정에 의해서 선택되었다.

2.2 일반적 웨이블릿 변환 방법

일반적으로 절차적 웨이블릿 연산은 예측(prediction)과 갱신(update)라는 두 가지 과정을 반복해서 수행할 수 있다. (그림 3)에서는 일반적인 이산 웨이블릿 변환을 위한 lifting 방식의 개요를 제시한다[10]. 이 방식은 convolution 연산 없이 웨이블릿 변환을 수행하는 방식이며 (그림 3)에서 제시한 프로세스는 다음과 같이 k 번째 예측 및 갱신이라는 두 연산으로 다음과 같이 표현할 수 있다.

○ **예측 연산**

k 번째 반복 시 i 번째 원소 예측 오류 $\{d_i^k\}$

$$d_i^k = d_i^{k-1} - p(s_i^{k-1} + s_{i+1}^{k-1})$$

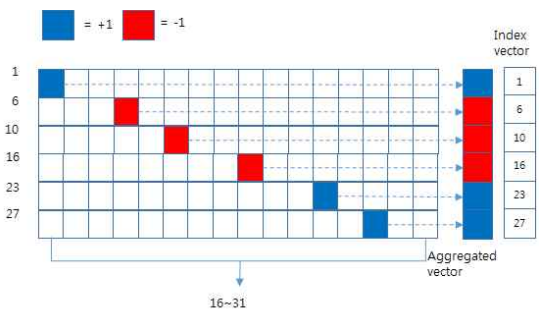
○ **갱신 연산**

k 번째 반복 시 i 번째 원소 갱신 $\{s_i^k\}$

$$s_i^k = s_i^{k-1} + u(d_{i-1}^k + d_i^k)$$

이 때 함수 p, u는 웨이블릿의 종류에 따라 결정된다. 이렇게 계산된 $\{d_i^k\}$와 $\{s_i^k\}$로 웨이블릿 행렬(명시적으로 행렬로 표현하지 못할 경우도 있으나)과 입력 신호 $\{x_i\}$와의 곱셈으로 계산한 변환 결과를 얻는다. 이 lifting 방식에서 주목할 만한 점은 $\{d_i^k\}$와 $\{s_i^k\}$의 계산이 대표적인 병렬 연산 알고리듬인 reduce 연산[11]과 매우 유사한 방식으로 진행된다는 사실이다. 그러나 (그림 3)의 방법은 (그림 1)과 같은 정방 행렬로 표현 가능한 일반적인 웨이블릿 행렬에만 적용 가능하며, 앞서 논의한 대로, 이 논문에서 다루는 (그림 2)와 같은 무작위 추출 행렬에 적용할 수 없다. 따라서 본 논문에서는 (그림 2)와 같은 무작위 추출 Haar 웨이블릿 행렬의 특징을 관찰함으로써 해결 방안을 제시하고자 한다.

(그림 4) 열 벡터 Ψ_{16} ~ Ψ_{31} 처리 방식

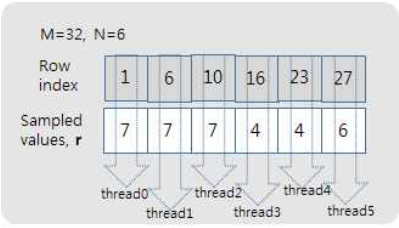


(Figure 4) Scheme for column vectors Ψ_{16} ~ Ψ_{31}

2.3 부분적 Haar 웨이블릿 행렬의 구조

<표 1> MP 알고리듬에서 가장 연산 집약적인 부분은 두 벡터의 내적 <**r**, Ψ_i>을 계산하는 곳이다. M x M 크기의 Haar 웨이블릿 행렬의 구조로 인해서 무작위로 N개의 행을 선택한 N x M 행렬의 열 벡터 Ψ_i는 행렬 내에서의 위치(i)에 따라서 다음과 같은 특징을 가진다.

(그림 5) 무작위 부분 Haar 웨이블릿 변환 예제



(Figure 5) Example for random partial Haar wavelet transform

○ $M/2 \le i < M$

(그림 1) 및 (그림 2)에서는 M = 32이기 때문에 16~31 번째 열 벡터에 해당된다. (그림 1)에서 볼 수 있듯이 원래 Haar 웨이블릿 행렬에서는 이 범위에 속하는 열 벡터들은 대부분의 원소가 0이고 +1(빨간색), -1(파란색) 원소를 하나 씩만 가진다. Sen과 Darabi가 [5]에서 제시한 compressive sensing 기법의 측정 행렬에서는 Poisson disc distribution[12]와 같이 무작위 샘플링 지점들 사이에서 균일한 간격이 보장되는 방식을 사용해서 (그림 2)와 같은 무작위 행렬을 추출하기 때문에 연속된 두 개의 행(row)이 선택될 확률이 0에 가깝다(또 다른 측면에서, [5]에서 논의한 대로 연속된 두 개의 행이 선택되도록 바로 이웃한 샘플들이 측정될 경우, compressive sensing의 복원 성능이 저하되기 때문에 이러한 상황을 적극적으로 피해야 한다). 따라서 (그림 2)에서 볼 수 있는 것처럼 이 범위에서의 열 벡터는 원소가 모두 0(세로 방향으로 모두 흰색)이거나 +1 또는 -1 하나만을 가진다고 높은 확률로 확신할 수 있다. 또한 (그림 1)에서 볼 수 있듯이 Haar 웨이블릿은 구조적으로 규칙성을 가지며 그 결과, 행과 열의 색인 번호만 있으면 행렬의 원소값을 쉽게 계산할 수 있다(예. Matlab 코드[13]). 따라서 연산 효율성을 높이기 위해 계산 과정에서 모든 원소가 0인 열 벡터를 효과적으로 회피하고 원소가 1개 있는 열 벡터((그림 2)에서 16, 19, 21, 24, 27, 29 번째 열 벡터, 즉, Ψ_{16}, Ψ_{19}, Ψ_{21}, Ψ_{24}, Ψ_{27}, Ψ_{29})만을 계산 시에 고려할 수 있어야 한다. 주목할 만한 특징은 이 범위에서 원소가 1개씩만 존재하는 열 벡터들을 비교해 보면 열 벡터 내에 원소가 0이 아닌 색인들이 서로 겹치지 않는다는 점이다(예를 들어 Ψ_{16}은 첫 번째 원소(파란색), Ψ_{19}는 두 번째 원소(빨간색)...). 따라서 별도의 벡터 하나를 할당하고 여기에 색인값을 저장하고 겹치지 않는 원소들을 한 벡터에 모두 저장할 수 있다(그림 4). 앞서 실제 적용 시 전체 신호 길이 M이 상당히 큰 수(예를 들어 512x512x512 복셀의 경우 $M=10^{27}$)가 된다고 논의한 바 있으며, 따라서 이 범위의 열 벡터들은 M/2개로 상당히 많지만 내적 <**r**, Ψ_i>을 계산하는 연산을 위해 M/2 회만큼의 많은 연산을 실행할 필요 없이 샘플링 횟수에 해당하는 길이 N (M>N)을 가지는 두 벡터의 각 원소별 곱셈 연산을 병렬로 한번만 수행하면 된다는 사실을 알 수 있다. Sen과 Darabi의 연구[5]에서 볼 수 있듯이 일반적으로 N은 매우 큰 M에 대해서도 현재의 컴퓨터 시스템이나 GPU의 메모리에서 다룰 수 있을 정도의 크기이기 때문에 앞서 논의한 내용이 실제 구현 측면에서 상당한 장점이 된다.

○ $M/2^2 \le i < M/2$

이 범위는 (그림 1) 및 (그림 2)에서는 8~15 번째 열 벡터에 해당된다. (그림 1)에서 볼 때 이 범위의 열 벡터들은 0이 아닌 원소를 4개씩 가지고 있다. 또한 (그림 4)에서처럼, 이 범위의 열 벡터들 역시 가로 방향으로 중복 없이 크기 N 벡터 하나에 모두 저장할 수 있다. 따라서 앞서 언급한 대로 이 범위에서 M/4 개의 열 벡터들에 대해 한 벡터에 모아서 병렬처리로 한 번에 내적 <**r**, Ψ_i>을 계산할 수 있다.

○ $M/2^q \le i < M/2^{q-1}$ $(i \neq 0,1,\ q \neq 1)$

위에서 논의한 구체적인 몇 단계에 대한 논의를 바탕으로 일반적인 q 단계에서의 특징을

유도할 수 있다. 일반적인 경우 Poisson disc distribution을 적용한다고 하더라도 첫 번째 단계에서와는 달리, (그림 2)에서 Ψ_2, Ψ_3, Ψ_6 등과 같은 열 벡터들은 0이 아닌 원소가 2개 이상 존재하기 때문에 이 단계만 보면, 가로 방향으로 중복 없이 크기 N 벡터 하나에 모두 저장이 불가능해 보인다. 그러나 전 단계에서

(그림 6) 1 단계 결과

Step n = 1
span = 2

row index	1	6	10	16	23	27
g	0	3	5	8	11	13
m	1	0	0	0	1	1
sub	-7	7	7	4	-4	-6
add	7	7	7	4	4	6
#	1	1	1	1	1	1
col index	16	19	21	24	27	29

(Figure 6) Step 1

의 계산 결과를 활용할 수 있기 때문에 일반적인 경우에도 하나의 벡터로 줄일 수 있고 지금까지와 동일한 병렬 연산을 수행할 수 있다. 자세한 내용은 2.4 절에서 구체적인 예를 통해 제시한다. 따라서 일반적인 경우에도 $M/2^q$ 번의 벡터 내적 $<\mathbf{r}, \Psi_i>$ 연산을 수행하지 않고 N개 벡터 두 개에 대해 각 원소들끼리의 곱셈 연산을 병렬로 수행할 수 있다. 이렇게 얻은 결과 벡터의 각 원소값은 별도로 저장한 색인 벡터(index vector)에 해당하는 내적 $<\mathbf{r}, \Psi_i>$ 값에 해당된다.

○ i = 0, 1

이 단계에서는 열 벡터 하나만 존재한다. i = 1일 때 행 색인 j ((그림 2, 3)에서 왼쪽에 표시한 번호)에 대해 j < M/2 일 때 원소값은 +1, j ≥ M/2에서는 -1이다. i = 0일 때는 모든 원소가 +1이다. $<\mathbf{r}, \Psi_0>$ 및 $<\mathbf{r}, \Psi_1>$의 계산에서는 이 단계만 볼 때에는 앞서 논의한 방식으로 계산이 힘들고 일반적인 벡터 내적 연산(즉, 각 벡터의 원소들끼리 곱한 후 곱한 값들을 합산)이 필요한 것처럼 보이나, 이전 단계에서 계산 후 기억한 값들을 활용함으로써 앞서 설명한 방식대로 일관성 있게 이 단계까지 계산이 가능하다. 자세한 내용은 3 장에서 예제를 통해 논의한다.

3. 제안하는 알고리듬 연산 결과

3.1 예제를 통한 구현 알고리듬 논의

(그림 5)에서는 제안하는 무작위 부분 Haar 웨이블릿 변환 알고리듬 설명을 위한 예제를 제시한다. 전체 신호 크기는 (그림 1) 및 (그림 2)에서 제시한 대로 M=32를 적용하고 샘플링 개수 N=6을 적용하며 샘플링된 값은 벡터 **r**로 구체적인 값을 제시했다. 예제에서 **r**의 원소들은 부분적으로 연속되는 정수로 설정되었는데 이 상황은 실제 0~255 사이의 8비트 정수로 표현되고 보통 선 또는 면으로 인식되는 부분은 동일한 색상값을 가지는 이미지의 RGB 채널들 중 한 채널의 상황을 가정한 것이다. 또한 row index 벡터에는 이 무작위 샘플링으로 선정된 행의 색인값이며 이 값들은 (그림 2)에서 왼쪽에 세로로 제시된 바 있다.

○ 초기화

연산이 시작되면 GPU 메모리 상에 생성된 벡터 **r** 및 row index의 각 원소별로 병렬 실행 단위 (예를 들어 CUDA의 경우 스레드) 하나 씩 할당되어 병렬 연산 구성을 준비한다. (그림 5)에서 스레드가 6개 할당되었음을 볼 수 있다.

매 단계별로 연산에 필요한 span이라고 하는 정수 변수가 하나 계산되어 각 스레드로 전달된다. 현재 단계를 n이라고 하면 $span=2^n$으로 계산하며 각 단계가 시작할 때 각 스레드는 자신이 맡은 row index 벡터의 원소값을 이용해서 다음과 같이 그룹 번호를 의미하는 g와 그룹 내 멤버 색인을 의미하는 m을 계산한다.

$$g = (row\ index) / span$$

$$m = (row\ index)\ \%\ span$$

(%는 modular 연산)

이 때 이 정보를 이용해서 원래 32x32 Haar 행렬에서의 열 색인(column index)을 구할 수 있다.

$$(column\ index) = M / span + g$$

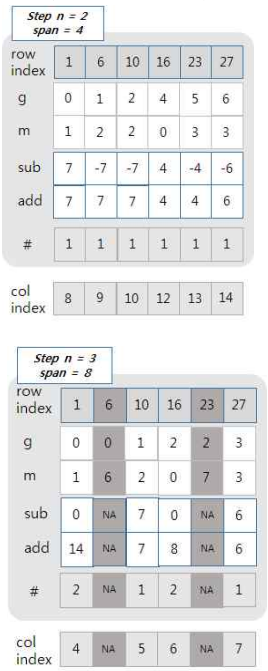
초기화 시에 크기 N=6인 두 개의 벡터 **sub**

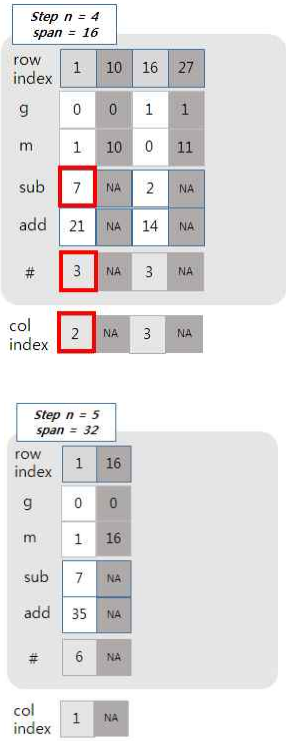
와 **add**를 GPU 메모리 상에 마련하고 **add** 벡터에 벡터 **r**을 복사한다. 마지막으로 normalize를 위해 0이 아닌 원소의 값을 계산하기 위한 같은 크기의 **#** 벡터를 생성하고 각 스레드별로 1로 초기화한다((그림 6) 참고).

○ **단계별 병렬 연산**

각 스레드는 자신이 맡은 j 번째 원소에서 g의 값과 자신의 바로 뒤에 있는 j+1 번째의 g 값을 비교한 후에 동일하면 (즉, 동일한 그룹에 속하면) 자신이 담당하는 add 벡터의 j 번째 원소의 값에 뒤에 있는 스레드가 맡고 있는 j+1번째 add 벡터 원소값을 더해서 저장하고 두 벡터 원소 값을 빼서 자신이 맡은 j 번째 **sub** 벡터 원소값에 저장한다. 이 때 인접한 바로 스레드의 g 벡터 원소값이 자신과 다를 경우에는 전후 벡터 값들이 0이라고 가정하고 위의 단계를 수행한다.

(그림 7) 2~5 단계 결과





(Figure 7) Step 2-5

이렇게 각 스레드별로 병렬 계산이 끝나면 (그림 6)에서 제시한 결과를 얻는다. 이 과정에서 # 벡터는 add 벡터 연산(즉, 이웃한 두 벡터 원소의 합계 계산) 조건이 충족되면 함께 합산해서 매 단계마다 갱신한다.

(그림 6)에서 제시한 1단계 계산 결과를 보면 col index 값을 통해 $<\mathbf{r}, \Psi_{16}>$, $<\mathbf{r}, \Psi_{19}>$, $<\mathbf{r}, \Psi_{21}>$, $<\mathbf{r}, \Psi_{24}>$, $<\mathbf{r}, \Psi_{27}>$, $<\mathbf{r}, \Psi_{29}>$의 결과가 계산되었음을 알 수 있고 아직 normalize 하지 않은 내적값은 sub에 저장되어 있다. 또한 # 벡터에서 갱신된 값은 0이 아닌 원소의 개수가 저장되기 때문에 normalize를 계산할 수 있다. 예를 들어 $<\mathbf{r}, \Psi_{16}>$값은 (그림 6)에서 굵은 빨간색 사각형으로 표시한 원소값들을 이용해서 다음과 같이 계산한다.

$$<\mathbf{r}, \Psi_{16}> = -7 / \sqrt{1} = -7$$

이 결과는 (그림 2)의 부분적 Haar 웨이블릿 행렬의 일반적인 연산 결과와 동일하다. 즉,

$$r = [777446]^T, \Psi_{16} = [-100000]^T$$
$$< r, \Psi_{16} > = 7 \times -1 = -7$$

(그림 7)에서는 나머지 2~5 단계의 수행 결과를 볼 수 있다. 특히 3, 4, 5 단계에서는 한 스레드가 앞쪽의 인접한 스레드의 벡터 g의 해당 요소값을 확인하고 자신과 동일하면 자신이 맡은 벡터 요소들을 모두 삭제(짙은 회색 및 NA로 표시)하고 다음 단계로 넘어 가게 된다. 이 처리는 2.3절에서 논의한 '가로 방향으로 중복 없이 크기 N 벡터 하나에 모두 저장 가능하도록' 하는 과정이다. 특히 이 과정은 현재 설명하는 알고리듬이 이진 트리의 리프 노드에서 루프로 진행하면서 병렬 연산을 수행하는 과정으로 설명할 수 있고, 따라서 이 삭제 과정은 두 개의 자식 노드 중 하나를 제거해서 부모 노드의 정보로 업데이트하는 과정으로도 설명할 수 있다. 앞서 언급한 대로 각 단계별로 <**r**, Ψ_1> 값을 계산할 수 있는데, 예를 들어 4 단계(n=4, span=16)에서는 (그림 7)의 결과를 통해 <**r**, Ψ_2>를 계산할 수 있다((그림 7)의 Step n=4 단계에서 굵은 빨간색 사각형 값 사용).

$$<\mathbf{r}, \Psi_2> = 7 / \sqrt{3}$$

마찬 가지로 이 단계에서 <**r**, Ψ_3>의 값은,

$$<\mathbf{r}, \Psi_3> = 2 / \sqrt{3}$$

일반적인 연산을 통해 확인하면,

$$r = [777446]^T,$$
$$\Psi_2 = [11-1000]^T / \sqrt{3},$$
$$\Psi_3 = [00011-1]^T / \sqrt{3},$$
$$< r, \Psi_2 > = (7+7-7) / \sqrt{3} = 7 / \sqrt{3}$$
$$< r, \Psi_3 > = (4+4-6) / \sqrt{3} = 2 / \sqrt{3}$$

결과가 동일함을 알 수 있다.

4. 결론 및 향후 계획

Compressive sensing, 희소 복원 기법 등에 적용 가능하고 SIMT 플랫폼에 기반한 새로운 병렬 무작위 부분적 Haar 웨이블릿 변환 기법을 제안한다. 이 변환 루틴은 Haar 웨이블릿 행렬의 규칙성을 활용해서 무작위로 추출될 때에도 매우 큰 신호 (즉, M이 매우 큰 경우) 까지도 절차적 방식으로 GPU 메모리에서 처리할 수 있다. 논의한 대로 제안하는 방법은 $\log_2^M$ 단계의 병렬 처리 루틴으로 구성되며 이러한 방식은 대표적인 병렬 알고리듬인 reduce 알고리즘[11]과 유사한 연산 구조를 가진다. 향후 제안하는 방법을 활용해 Matching Pursuit 등의 병렬 compressive sensing 희소 복원 애플리케이션을 구현할 예정이다.

References

[1] D.L. Donoho, "Compressed sensing," IEEE Transactions on Information Theory, Vol.52, No.4, pp.1289-1306, April 2006.

[2] Yonina C. Eldar and Gitta Kutyniok, "Compressed Sensing: Theory and Applications", 1st edition, Cambridge University Press, 2012

[3] https://developer.nvidia.com/cuda-zone

[4] T. Park, "CUDA-based Object Oriented Programming Techniques for Efficient Parallel Visualization of 3D Content", Journal of Digital Contents Society, Vol.13, No.2, pp. 169-176, 2012.

[5] P. Sen and S. Darabi, "Compressive Rendering: A Rendering Application of Compressed Sensing," IEEE Transactions on Visualization and Computer Graphics, Vol.17, No.4, pp.487-499, April 2011.

[6] M. Andrecut, "Fast GPU Implementation of Sparse Signal Recovery from Random Projections", http://arxiv.org/abs/0809.1833?context=q-bio

[7] E.J. Candes and M.B. Wakin, "An Introduction To Compressive Sampling," IEEE Signal Processing Magazine, Vol.25, No.2, pp.21-30, March 2008.

[8] J. Cheng, M. Grossman, and T. McKercher, "Professional CUDA C Programming", 1st Edition, Wrox, September 2014.

[9] G. Strang, "Computational Science and Engineering", 1st Edition, Wellesley-Cambridge Press, Nove

mber, 2007.

[10] C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, F. Tirado, "Parallel Implementation of the 2D Discrete Wavelet Transform on Graphics Processing Units: Filter Bank versus Lifting," IEEE Transactions on Parallel and Distributed Systems, Vol.19, No. 3, pp.299-310, March 2008.

[11] http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf

[12] https://www.jasondavies.com/poisson-disc/

[13] http://www.mathworks.com/matlabcentral/fileexchange/4619-haarmtx

박 태 정

1997년 : 서울대 전기공학부 (공학사)
1999년 : 서울대 전기공학부 대학원 (공학 석사, 반도체 전공)
2006년 : 서울대 전기컴퓨터공학부 대학원 (공학박사, 컴퓨터 그래픽스 전공)
2006년~2013년: 고려대학교 연구교수
2013년~현재 : 덕성여자대학교 디지털미디어학과 조교수
관심분야 : 컴퓨터그래픽스, 병렬처리, 게임 물리, 수치해석, 3차원 모델링