SSE 명령어를 이용한 고속 Sobel Edge Detector 구현

박은수, 최학남, 김준철, 김학일
인하대학교 정보공학과

Implementation of Fast Sobel Edge Detector Using SSE Instructions

Eunsoo Park, Xuenan Cui, Junchul Kim, Hakil Kim
School of Information Engineering, Inha University

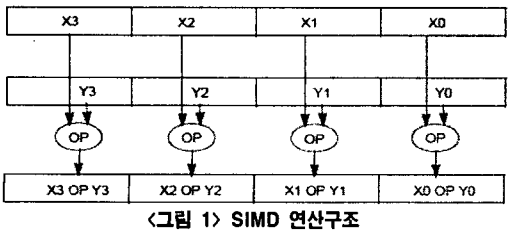
Abstract - 본 논문에서는 SSE(Streaming Sigle instruction multiple data Extensions)명령어를 이용한 고속 Sobel edge detection 알고리즘을 제안한다. SSE를 지원하는 CPU는 128bit의 SSE 레지스터를 보유하고 있으며 이에 속한 데이터는 한 번에 병렬적으로 처리 될 수 있다. 실험은 Sobel 연산에 대하여 순차처리 코딩, 이미지 처리 라이브러리인 OpenCV, MIL 8.0, IPP 5.2를 이용한 코딩, shift 알고리즘을 사용한 SSE 코딩, 제안하는 방법을 이용한 SSE 프로그램 코딩에 대해 각각의 수행 시간을 측정하고 이를 비교하였다. 실험결과 제안하는 방법은 순차코딩에 비해 약 12배, OpenCV에 비해 13배, MIL에 비해 2배 정도 빨랐으며, IPP에 대해선 약간 빠른 성능 향상을 보였다. 또한 일반적인 shift를 이용한 방법보다 제안하는 방법은 대략 1.5배 정도의 성능 향상이 있었다. 이를 통해 제안하는 방법은 라이브러리를 구입하는 비용을 들이지 않으며 추가적 하드웨어의 구입 없이도 PC에서 빠른 Sobel 연산을 수행 할 수 있음을 보였다.

1. 서 론

대부분의 이미지 처리를 위한 프로그램 코딩은 루프 구조를 이용하여 반복적인 명령을 수행하기 때문에 시간 지역성과 공간 지역성을 갖고 있다 [1]. 이런 지역성을 효과적으로 처리하기 위해 인텔에서는 처음으로 SIMD (Single Instruction Multiple Data) 처리를 가능하게 하는 MMX 기술을 선보였다. SIMD는 그림 1과 같이 하나의 명령어를 이용하여 다수의 데이터를 처리할 수 있는 기술이다. MMX 기술은 64bit 크기의 레지스터를 보유하고 있는데 이에 속한 integer 데이터들은 병렬적으로 SIMD 연산이 가능하게 된다. 후에 Intel은 64bit MMX 레지스터의 부족한 크기를 확장하고 integer뿐만 아니라 floating point 처리를 가능하게 하기위해 펜티엄III 프로세서를 시작으로 8개의 128bit SSE 레지스터와 명령어를 추가하였다. SSE는 프로그래머의 요구에 맞추어 SIMD 연산에만 전념하는 명령어 set을 지속적으로 추가하게 된다. SSE는 CPU의 발전과 함께 SSE2, SSE3, SSSE3 (Supplemental SSE3)시리즈를 선보였고 현재 SSE4가 개발되었으며 곧 상용화 예정이다. 이 SSE4는 더 이상 MMX 레지스터를 지원하지 않는다. 또한 8개의 SSE 레지스터는 CPU가 x64로 발전되면서 16개로 늘어나 더 큰 확장성을 보유하게 되었다 [2-3].

이러한 발전과 함께 SIMD 구조를 통한 프로그램 성능 향상에 관한 기존의 연구가 많이 존재하였다 [4-5]. 그러나 기존 SIMD 구조를 이용한 프로그램의 성능 평가에 대한 연구는 기본적인 순차처리에 대한 처리속도의 비교만을 제시하여 실제 application에서 가장 많이 사용되는 이미지 처리 라이브러리들 간의 비교 평가가 힘들며, 현재 CPU는 그 연구 당시 보다 높은 처리 속도를 갖고 있기 때문에 개선된 하드웨어 환경에서의 성능평가에 대한 비교가 필요하다. 또한 이미지 전처리에 사용되어 활용빈도가 높은 Sobel 연산과 같은 알고리즘에 대해서 적절한 SIMD 구조를 위한 알고리즘을 제시하지 못하고 있는 실정이다.

본 논문에서는 이미지의 전처리 과정에서 많이 사용되는 Sobel edge detection 알고리즘이 SIMD 구조를 갖고 있는 CPU에서 어떻게 효과적으로 처리 될 수 있는지 보인다. 실험은 SSE 명령어를 사용하지 않은 일반적인 순차처리 방법, 무료로 배포되는 이미지 처리 라이브러리인 OpenCV, 유료로 제공 되는 라이브러리 MIL 8.0과 IPP 5.2, 일반적인 filter 처리 방식인 shift를 이용한 SIMD 코딩, 제안하는 알고리즘을 이용한 SIMD 코딩 방법들을 이용하여 각각 구현 하고 이의 수행 시간을 측정하였다. 수행시간 비교 결과 제안하는 방법이 가장 빠르고 효과적으로 Sobel 알고리즘을 수행할 수 있음을 확인하였다.



<그림 1> SIMD 연산구조

2. 본 론

2.1 Sobel Edge Detection

Sobel 연산자는 이미지 프로세싱에서 이미지 edge를 검출하는데 사용된다. Sobel 연산의 결과는 수직방향의 기울기와 수평방향 기울기를 구한 후 이 기울기의 크기를 구함으로써 얻어질 수 있다. 수학적으로 수직과 수평방향의 기울기는 수식 1과 같이 표현 할 수 있다. 여기서 A는 처리해야할 이미지를 나타내고 G_x와 G_y는 각각 수직 방향과 수평 방향의 기울기를 나타낸다. G_x와 G_y를 이용하여 edge의 크기인 G를 수식 2와 같이 얻을 수 있다. 실제 구현에서는 수식 2를 구하는 것은 계산적으로 복잡하고 수행 시간이 오래 걸리기 때문에 수식 3과 같이 절대 값을 이용하는 방법을 주로 사용하게 된다 [4].

본 논문에서는 Sobel 알고리즘의 구현을 $|Gy|$, 즉 수평방향의 기울기를 구하는 것으로 한정하였다. 그 이유는 실험에 사용되는 라이브러리들 중에는 최종 Sobel 결과인 G를 구하는 함수가 존재하지 않고 각 방향 $|Gx|$와 $|Gy|$의 연산만을 지원하는 것이 있다. 따라서 G를 구하는 과정은 프로그래머가 따로 작성해줘야 한다. 이는 G연산에 대해 프로그래머의 개입으로 인해 정확한 라이브러리의 성능측정을 어렵게 하는 요인이 된다.

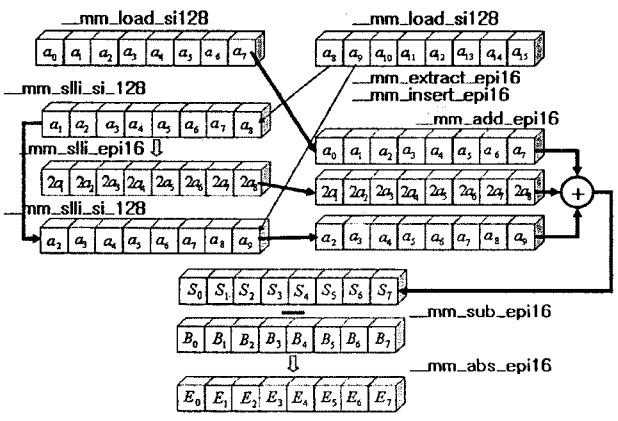
$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad and \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{1}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{2}$$

$$G = |G_x| + |G_y| \tag{3}$$

2.2.1 Shift 알고리즘을 이용한 SIMD 코딩

SIMD코딩을 하기 위해선 이미지는 128bit boundary로 정렬되어 있어야 한다. 정렬되지 않은 데이터구조는 정렬된 데이터에 비해 빠른 수행성능을 얻을 수 없기 때문이다 [5]. 실험에 사용된 대용량 이미지는 8bit gray scale로 이미지 크기는 4048×4057이다. Sobel 연산으로 수평방향의 edge 성분을 얻기 위해서 먼저 128bit의 이미지 데이터를 읽어 들여야 한다. 이미지의 각 픽셀은 8bit 크기로 나타낼 수 있는 색상의 범위인 [0, 255]사이의 값이다. 이러한 픽셀에 덧셈이나 곱셈 연산을 하게 되면 8bit으로 표현할 수 있는 범위를 넘기 때문에 8bit 픽셀의 데이터 형을 16bit크기의 데이터 형태로 바꿔줘야 한다. 전체적인 Sobel 알고리즘 처리 과정은 그림 2와 같다. 그림 2의 상단은 128bit 크기 데이터 2개를 불러 온 모습을 나타낸다.



<그림 2> Shift 방법을 이용한 SIMD Sobel 알고리즘

이렇게 불러 온 데이터는 16bit크기의 이미지 픽셀 8개를 포함하고 있다. 이 데이터는 SSE intrinsic 명령어 __mm_slli_128을 이용하여 한 픽셀 크기만큼 왼쪽으로 shift 될 수 있고 이렇게 shift된 SSE 레지스터는 왼쪽으로 1bit shift를 통해서 전체적으로 2를 곱한 결과를 산출하게 된다. 그림 2의 중단에 a라는 인덱스 앞에 2라는 계수를 갖고 있는 데이터가 shift연산의 결과이다. 앞에서 __mm_slli_128 명령어를 이용해서 한 픽셀만큼 왼쪽으로 shift된 결과에 다시 똑 같은 명령어를 적용하여 2칸 shift된 SSE 데이터를 만들 수 있다. 이는 a_2로 시작되는 데이터이다. 이 세 개의 결과를 __mm_add_epi16 명령어를 이용하여 덧셈을 수행하면 수식 1에서 수평 방향 Sobel mask의 상단 +1 ,+2 ,+1에 의해 계산되어지는 결과 값을 얻을 수 있다. 이는 한번에 8개의 픽셀을 계산하게 된 경우이다. 위와 같은 과정을 수평방향 Sobel mask 하단의 결과를 얻기 위해서 반복한다. 반복에 의한 결과는 그림 2에서 인덱스 B로 나타내었다. 이 두 개의 결과를 __mm_sub_epi16을 이용하여 뺄셈을 수행한 후 __mm_abs_epi16을 이용하여 절대 값을 취하면 수평 방향 Sobel edge detection 알고리즘을 완성하게 된다.

2.2.2 개선된 알고리즘을 이용한 SIMD 코딩

제안하는 SIMD 수평방향 Sobel edge detection 알고리즘은 많은 수의 SSSE3 명령어를 사용한다. SSSE3는 프로그래머가 쉽게 SIMD 알고리즘을 구현 할 수 있도록 하며 동시에 프로그램 속도를 향상 시킬 수 있는 명령어가 많이 추가 되었다. 전체적인 개선된 SIMD Sobel 알고리즘에 대한 설명은 그림 3에 나타내었다.

제안하는 수평방향 Sobel edge detection 알고리즘은 128bit 크기의 정렬된 이미지 데이터 3개를 로드한다. 이 128bit SSE 레지스터에는 8bit 크기의 이미지 픽셀 16개가 포함된다. 16개의 픽셀을 한 번에 로드하면 8개의 픽셀을 로드할 때 보다 메모리 접근 수가 줄어들기 때문에 프로그램 수행 시간을 많이 줄 일 수 있게 된다. 이 로드된 데이터와 계수 1,2,..1,2를 갖는 SSE 레지스터를 multiply and add한다. Multiply and add 명령어는 __mm_maddubs_epi16 으로 이 수행결과는 그림 3에 가장 길게 표현된 SSE 레지스터이다. 그림과 같이 16개의 데이터에 multiply and add 명령을 수행 한 결과는 8개의 16bit 크기의 데이터를 만들어낸다. 이제 나머지 연산을 위하여 a_{17}로 시작하는 두 번째 이미지 데이터 16개를 로드 한 후 __mm_alignr_epi8 란 명령어를 사용하여 a_1으로 시작하는 처음 로드 된 데이터의 한 픽셀크기만큼 shift된 형태의 레지스터를 생성한다. 이 SSSE3 명령어는 앞선 shift연산 보다 빠르게 작동한다. 이렇게 완성된 데이터에 계수 0,1,0,1...,0,1을 차례로 갖는 데이터와 multiply and add를 수행하여 처음 로드된 데이터의 홀수의 인덱스를 갖는 a_3로 시작하는 데이터를 만들 수 있다. 이렇게 홀수의 인덱스를 갖는 데이터와 앞서 임시변수에 저장한 데이터를 더하면 홀수 인덱스의 Sobel mask 연산의 상단 부분을 만들어 낼 수 있다. 짝수의 인덱스를 갖는 Sobel mask 상단 부분을 얻기 위해선 그림 3의 박스에서 설명한 것처럼 계수 데이터와 추가로 로드되는 데이터를 바꾸어 주고 위와 같은 알고리즘을 반복 적용한다면 쉽게 얻어 낼 수 있다.

이렇게 생성된 SIMD 레지스터 S는 8개의 짝수 인덱스와 홀수 인덱스를 갖는 2개의 데이터로 나뉘어져 있다. 수평방향 Sobel mask 하단의 결과를 얻기 위해선 하단 이미지 픽셀에 대하여 위와 똑같은 알고리즘을 적용한다. 이렇게 얻어진 Sobel mask하단의 연산 결과는 그림 3에서 인덱스 B로 표현 하였다. 두 번째 박스안의 인덱스 B는 위의 알고리즘을 그대로 반복하면 얻을 수 있다. 이제 S와 B의 차를 구하고 그 값에 절대치를 취하면 최종 수평방향 Sobel edge detection 알고리즘을 완성할 수 있다. SSSE3 명령어 __mm_abs_epi16 으로 절대치 값을 얻고 두 데이터를 __mm_packus_epi16 명령어를 이용하여 합한 후 뒤 섞여 있는 픽셀들을 명령어 __mm_shuffle_epi8 을 이용하여 순서대로 정렬 하면 최종 수평방향 Sobel edge detection 결과를 얻을 수 있게 된다.

2.2.3 수행 시간 비교

6개의 방법으로 구현한 수평방향 Sobel edge detection 알고리즘의 수행 시간을 표 1을 통해 나타내었다. 실험에 사용된 CPU는 Intel Core2 Duo Desktop Processor이며 CPU의 속도는 2.40GHz이다. 표의 SIMD 1은 shift 방법을 사용한 알고리즘이고 SIMD 2는 보다 개선된 방법이다. MIL과 IPP 라이브러리는 내부적으로 SSE 레지스터를 사용하여 만들어진 것이기 때문에 빠른 성능을 보인다. 표를 보면 제안하는 알고리즘은 순차처리에 비해 약 12배 정도 빠르며 이미지처리 라이브러리인 OpenCV에 비해 13배, MIL에 비해 약 2배 정도 빠르며, IPP에 대해선 대략 큰 차이를 보이지 못했지만 약간 빠른 성능을 보였다.

<표 1> 수행 시간 비교

	순차처리	OpenCV	MIL 8.0	IPP 5.2	SIMD 1	SIMD 2
수행시간 (ms)	168.89	192.06	30.12	14.81	23.37	14.32

3. 결 론

본 논문에서는 수평방향의 Sobel edge detection 알고리즘을 SSE 명령어를 사용하여 코딩하였다. SSE 명령어를 이용하여 shift를 사용한 알고리즘과 이 보다 개선된 방법 2가지 알고리즘을 제시하였으며 이 알고리즘에 대한 수행 시간을 순차처리, 이미지 처리 라이브러리인 OpenCV, MIL 8.0, IPP 5.2 와 비교하였다. 비교 결과 제안하는 방법은 가장 빠르게 수평방향 Sobel 알고리즘을 수행 할 수 있었다. 또한 일반적으로 많이 사용되는 shift를 이용한 Sobel edge detection 보다 multiply and add를 사용한 알고리즘이 더욱 빠른 성능을 보이는 것을 통해 최상의 성능을 얻기 위해선 새로운 알고리즘의 개발이 필요함을 알 수 있었다.

본 논문은 현재 많이 사용되는 빠른 속도의 CPU를 사용하여 실험함으로과거 하드웨어의 낮은 사양에서의 SSE 성능평가에 대한 최신의 추가적인 데이터를 제시하였으며, 라이브러리와의 비교를 통하여 프로그래머가 직접 SSE 코딩을 하였을 때의 이점을 확인 할 수 있도록 하였다. 또한 라이브러리를 사용하지 않고 보다 빠른 SSE Sobel edge detection 알고리즘을 구현함으로써 이미지 처리 라이브러리와 추가적인 하드웨어의 구입 없이도 빠르게 PC에서 Sobel edge detection을 구현 할 수 있는 방법을 제시하였다.

[참 고 문 헌]

[1] Randle Hyde, "WRITE GREAT CODE", No Starch Press, 2004

[2] Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture", Intel corporation, 2007

[3] Intel, "Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2: Basic Architecture", Intel corporation, 2007

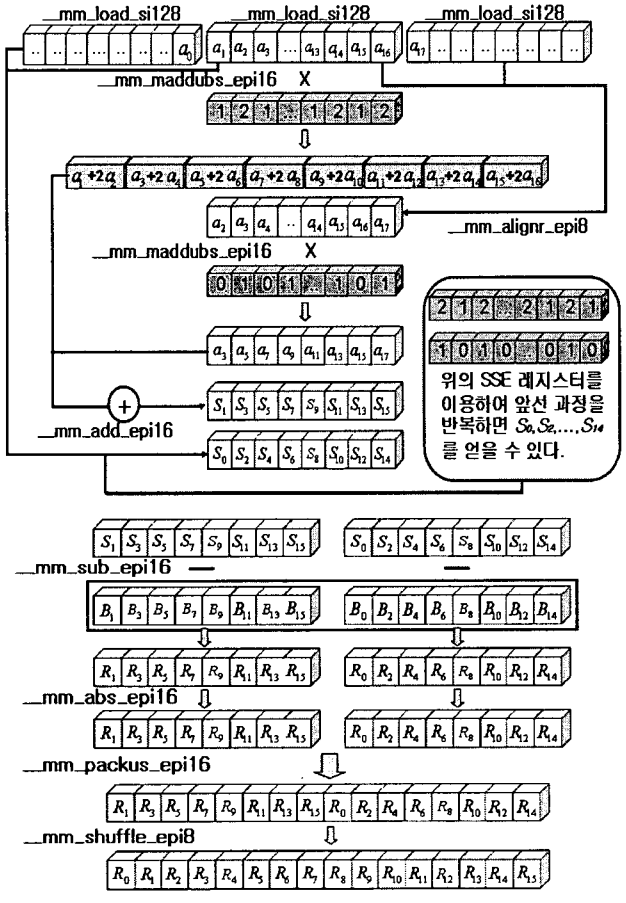
[4] Johan Skoglund and Michael Felsberg, "Fast image processing using SSE2", Proceedings of the SSBA Symposium on Image Analysis, 2005

[5] J´er˜ome Landr´e and Fr´ed´eric Truchetet, "Optimizing signal and image processing applications using Intel libraries", Proceedings of SPIE, Volume 6356, 2007

[5] Wikipedia, "http://en.wikipedia.org/wiki/Sobel_operator"

[6] Asadollah Shahbahrami. Ben Juurlink, Stamatis Vassiliadis, "Performance Impact of Misaligned Accessesin SIMD Extensions", ProRISC 2006, pp. 334-342, 2006

[7] Asadollah Shahbahrami Ben Juurlink Stamatis Vassiliadis, "Efficient Vectorization of the FIR Filter", ProRisc 2005, pp. 432-437, 2005



<그림 3> 제안하는 방법을 이용한 SIMD Sobel 알고리즘