Journal of The Korea Society of Computer and Information
Vol. 18, No. 9, September 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.9.001

매니코어 프로세서를 이용한 SIFT 알고리즘 병렬구현 및 성능분석

김 재 영*, 손 동 구*, 김 종 면*, 전 희 성*

Parallel Implementation and Performance Evaluation of the SIFT Algorithm Using a Many-Core Processor

Jae-Young Kim*, Dong-Koo Son*, Jong-Myon Kim*, Heesung Jun*

요 약

본 논문에서는 대표적인 특징점 추출 알고리즘인 SIFT(Scale-Invariant Feature Transform)를 매니코어 프로세서를 이용하여 병렬 구현하고, 이를 실행 시간, 시스템 이용률, 에너지 효율 및 시스템 면적 효율 측면에서 분석하였다. 또한 기존의 고성능 CPU와 GPU(Graphics Processing Unit)와의 성능 비교를 통해 제안하는 매니코어의 잠재가능성을 입증하였다. 모의실험 결과, 매니코어를 이용한 SIFT 알고리즘 구현 결과는 기존의 OpenCV 구현 결과와 정확도면에서 동일하였고, 매니코어 구현은 고성능 CPU 및 GPU 구현보다 실행시간 측면에서 우수하였다. 또한 본 논문에서는 SIFT알고리즘의 옥타브 크기에 따른 에너지 효율 및 시스템 면적 효율을 분석하여 최적의 모델을 제시하였다.

▸ Keywords : 매니코어 프로세서, SIFT, 병렬처리, 그래픽 처리 유닛, 에너지 효율, 시스템 면적 효율

Abstract

In this paper, we implement the SIFT(Scale-Invariant Feature Transform) algorithm for feature point extraction using a many-core processor, and analyze the performance, area efficiency, and system area efficiency of the many-core processor. In addition, we demonstrate the potential of the proposed many-core processor by comparing the performance of the many-core processor with that of high-performance CPU and GPU(Graphics Processing Unit). Experimental results indicate that the accuracy result of the SIFT algorithm using the many-core processor was same as that of OpenCV. In addition, the many-core processor outperforms CPU and GPU in terms of execution time. Moreover, this paper proposed an optimal model of the SIFT algorithm on the many-core

•제1저자 : 김재영 •교신저자 : 전희성
•투고일 : 2013. 3. 20, 심사일 : 2013. 4. 5, 게재확정일 : 2013. 5. 18.
* 울산대학교 전기공학부(School of Electrical Engineering, University of Ulsan)
※ 이 논문은 2011년 울산대학교 연구비에 의하여 연구되었음.

processor by analyzing energy efficiency and area efficiency for different octave sizes.

▸ Keywords : Many-core processor, SIFT, parallel processing, graphics processing unit, energy efficiency, system area efficiency

I. 서 론

SIFT(Scale-Invariant Feature Transform) 알고리즘[1]은 입력된 영상에서 특징점에 대한 벡터형식의 데이터를 추출하는 알고리즘으로, 비교영상의 크기 변화 및 회전과 같은 변형이 일어나는 환경에서도 사용할 수 있는 알고리즘이다. 따라서, SIFT 기반 특징점 추출 기법들이 이미지 매칭, 물체 인식, 이미지 스티칭 등의 분야에서 활발한 연구되어 왔다. 하지만 SIFT알고리즘은 반복적인 연산과정이 많아 휴대용 임베디드 시스템 환경에서는 실시간 성능을 만족하기가 어렵다. 또한 휴대용 임베디드 시스템은 베터리 사용량에 민감하기 때문에 저전력을 만족해야 한다. 따라서 고성능 저전력 SIFT 알고리즘에 대한 연구가 활발히 진행되고 있다[2-3].

이러한 휴대용 기기에서 요구되는 고성능 및 저전력을 만족시키기 위한 대안 가운데 하나로 SIMD(Single Instruction Multiple Data) 기반 매니코어 프로세서가 유망하다. 명령어 레벨(instruction-level)이나 스레드 레벨(thread-level)로 동작하는 다중 슈퍼스칼라 프로세서들은 실리콘 면적을 멀티포트 레지스터 파일(multiported register file), 캐쉬 (cache), 파이프라인 (deep pipelined) 기능 유닛 들을 위해 사용한다. 반면 SIMD 기반 매니코어 프로세서는 수백, 수천 개의 저비용 프로세싱 엘리먼트(Processing Element, PE)를 이용하여 고성능뿐만 아니라 저전력도 만족시킨다[4]. 특히 SIMD 기반 매니코어 프로세서는 지역성(locality)이나 규칙성(regularity)이 있는 2차원 패턴 이미지나 비디오 픽셀 처리에 더욱 효과적인 구조이다.

본 논문에서는 특징점 추출을 위한 SIFT 알고리즘을 고속으로 처리하기 위해 SIMD기반의 매니코어 프로세서를 제안하고, 이를 이용하여 SIFT알고리즘을 구현하고 성능, 에너지 효율 및 시스템 면적 효율을 분석하였다. 또한 SIMD기반의 매니코어 프로세서의 성능 및 효율을 검증하기 위해 범용 고성능 CPU와 범용 그래픽 처리장치(GPGPU, General-Purpose Graphics Processing Unit)에서 구현한 SIFT 알고리즘과의 성능을 비교하였다.

본 논문의 구성은 다음과 같다. 2장에서는 SIFT 알고리즘을 설명하고, 3장에서는 실험에서 사용되는 매니코어, CPU 및 GPU에 대한 사양과 실험 방법론에 대해 기술한다. 4장에서는 매니코어를 이용한 SIFT 알고리즘의 병렬 구현에 대해 설명하고, 5장에서는 매니코어의 실험 결과를 분석하고, 동시에 기존 고성능 CPU와 GPU와의 성능을 비교한다. 마지막으로 6장에서 본 논문의 결론을 맺는다.

II. SIFT 알고리즘

SIFT 알고리즘은 다음과 같이 4단계로 구분할 수 있다[1].

- 스케일 공간 극값 특징점 추출 : 모든 스케일의 DoG(Difference of Gaussian) 피라미드에서 극값을 갖는 특징점 후보들을 추출한다. 스케일변화와 회전에 강한 요소이다.
- 특징점 위치 측정 : 추출한 특징점 후보 중에서 매칭에 있어서 안정적이지 못한 코너에 가깝거나 이웃 픽셀들과 대비가 낮은 특징점들을 제거한 후, 각 특징점의 위치를 원본 이미지의 크기에 맞게 보간한다.
- 방향 설정 : 그레디언트 방향을 기반으로 각 특징점에서 하나의 방향성분을 구한다. 회전과 평행 변환 등에 강한 요소이다.
- 특징점 기술자 : 각 특징점 위치에서 주변 픽셀 정보들을 통해 그레디언트 히스토그램을 생성하여 기술자를 생성한다.

본 논문에서는 위 4단계 중 첫번째 단계인 스케일 공간 극값 특징점 추출 단계를 SIMD기반의 매니코어 프로세서를 이용하여 구현하여 이를 분석하였다.

스케일 변화에 강한 특징점 후보들을 추출하기 위해서는 먼저 옥타브와 레이어로 구성되는 스케일 공간 피라미드를 생성해야 한다[5]. 각 옥타브는 여러 개의 레이어로 구성되어 있으며, 각 레이어들은 원본 계조영상에 가우시안 커널을 이용해 컨벌루션을 반복하여 얻을 수 있다. 첫 번째 옥타브는 원본 계조 영상을 통해 만들어지고, 두 번째 옥타브는 첫 번

째 옥타브의 가장 많이 컨벌루션된 레이어를 첫 번째 레이어로 사용한다. 이 때 첫 번째 레이어의 크기는 이전 옥타브 영상의 가로 세로 크기를 2로 나눈 크기이다. 이러한 작업은 옥타브의 이미지 크기가 특정 크기에 도달할 때까지 반복된다.

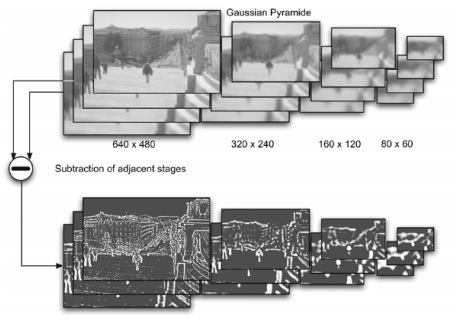


그림 1. 스케일 공간 피라미드와 DoG피라미드
Fig. 1. Scale-space pyramids and DoG pyramids

옥타브 피라미드가 완성되면 DoG피라미드를 생성한다. DoG 피라미드는 앞서 생성한 스케일 공간 피라미드의 각 옥타브에서 인접한 레이어 이미지들의 차이를 계산하여 생성할 수 있다. 그림 1은 스케일 공간 피라미드를 이용하여 DoG 피라미드를 만드는 과정을 보여준다. 특징점 후보들을 구하기 위해서는 마지막으로 생성된 DoG 피라미드를 사용한다. 먼저 같은 레이어 상의 이웃한 8개의 픽셀 값을 현재 픽셀 값과 비교하여 가장 크거나 가장 작을 경우 특징점 후보의 대상이 된다. 다음으로 현재 레이어와 인접한 두 레이어의 같은 위치에 있는 각각 9개의 픽셀들과 비교하여 가장 크거나 작으면 특징점 후보가 된다. 다시 말해 하나의 픽셀에 대해 26개의 이웃한 픽셀들을 비교하는 것이다. 이러한 과정은 DoG 피라미드 각 옥타브의 첫 레이어와 마지막 레이어를 제외한 모든 레이어에서 수행된다. 그림 2는 특징점 후보를 구할 때 사용되는 인접한 26개의 픽셀을 보여준다.

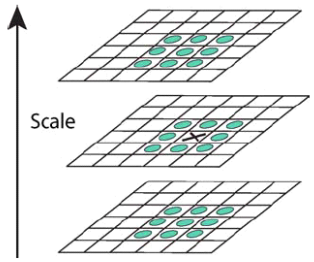


그림 2. 특징점 후보 추출을 위한 극값 계산
Fig. 2. Calculation of the extreme value for extracting candidate of feature points

III. 실험 환경

1. SIMD 기반 매니코어 프로세서 구조

그림 3은 SIMD기반의 매니코어 프로세서 아키텍처의 블록 다이어그램을 보여준다[6]. SIMD 기반 매니코어 프로세서는 여러 개의 프로세싱 엘리먼트(Processing Element, PE)가 2차원 배열 형태로 구성되어 있다. 이웃한 PE들은 매쉬 구조로 연결되어 있으며, 어레이 컨트롤 유닛(Array Control Unit, ACU)은 이들의 통신을 제어한다. 각 PE는 다음과 같은 RISC (Reduced Instruction Set Computer) 특징을 가지고 있다.

- ☐ 32비트 폭의 256개 워드로 구성된 로컬 메모리
- ☐ 32비트 폭의 16개 3포트 범용 레지스터
- ☐ 기본적인 산술/논리 연산을 수행하는 ALU
- ☐ 64비트 곱셈 및 누산기(multiply accumulator)
- ☐ 멀티 비트 산술/논리 시프트 연산을 수행하는 배럴 시프트 (Barrel Shifter)
- ☐ 지역 정보를 이용해 각 PE들을 활성 및 비활성 시키는 Sleep 유닛
- ☐ 이웃하는 PE들과 데이터 통신을 위한 NEWS (north-east-west-south) 네트워크 및 직렬 I/O유닛

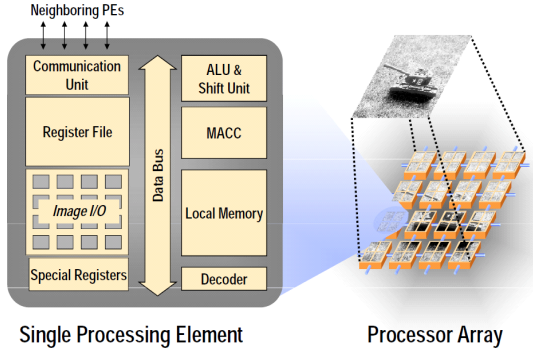


그림 3. SIMD 기반 매니코어 프로세서 아키텍처
Fig. 3. SIMD-based many-core processor architecture

2. 실험 방법론 및 성능 평가 지표

그림 4는 SIMD 매니코어 프로세서의 성능, 시스템 면적 효율 및 에너지 효율을 평가하기 위한 실험 방법론을 보여준다. 실험 방법론은 애플리케이션, 아키텍처 및 테크놀로지 레

벨로 구성되어 있다. 애플리케이션 레벨에서는 매니코어 프로세서용 시뮬레이터를 이용하여 특징 추출을 위한 각 알고리즘을 병렬 구현하며, 이를 통해 실행 사이클 개수, 동적 명령어 빈도, 시스템 이용률 등의 데이터를 추출한다. 아키텍처 레벨에서는 아키텍처 모델링 툴을 사용하여 모델링된 아키텍처의 디자인 변수들을 계산한다[7]. 테크놀로지 레벨에서는 아키텍처 레벨에서 구해진 디자인 변수들을 Generic System Simulator(GENESYS)의 입력으로 사용하여 아키텍처 모델들의 사이클 시간(cycle time), 와이어 지연(wire latency), 전력(power), 클록 주파수(clock frequency)를 계산한다[8]. GENESYS는 매니코어 및 매니 클러스터 등의 다양한 시스템 구조를 모델링하기 위한 테크놀로지 모델링 툴로 각 코어들은 동기 ASIC 칩으로 표현되는 계층적 모델로 구성된다. 이러한 계층적 모델은 기본요소, 재료, 디바이스, 회로, 시스템 등 5-레벨로 이루어져 있다. 처음 세 가지 레벨은 재료특성과 스위칭 디바이스 특성의 물리적 효과를 모델링하며, 회로 레벨은 신호 지연, 동적 및 정적 에너지와 같은 게이트의 특징을 모델링한다. 시스템 레벨은 싱글 ASIC 칩을 묘사하기 위한 아키텍처, 연결구조, 패키징 상세 정보를 포함한다. 마지막으로 위의 각 레벨에서 구해진 파라미터들을 입력으로 사용하는 디자인 공간 탐색기(design space explorer)는 PE 구조 별 실행시간, 시스템 면적 효율 및 에너지 효율을 측정한다.

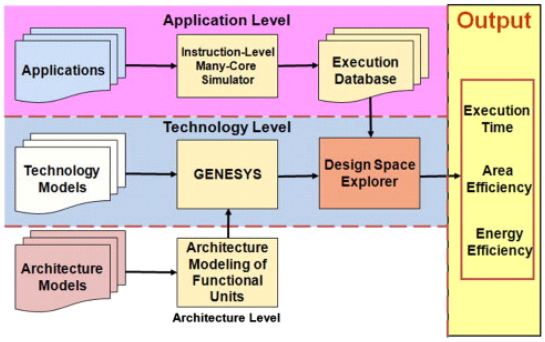


그림 4. 매니코어 프로세서를 위한 실험 방법론
Fig. 4. Simulation methodology for a many-core processor

표 1은 매니코어 프로세서의 성능 평가를 위한 세 가지 지표(실행시간, 에너지 효율, 면적 효율)를 보여준다[9]. 실행시간 (execution time)은 래스터화 알고리즘이 수행된 시간을, 에너지 효율 (energy efficiency)은 단위 에너지당 소비된 명령어 개수(Giga-operations/Joule)를 나타내고, 시스템 면적 효율 (area efficiency)은 단위 시스템 면적당 소비된 명령어 개수를 나타낸다.

표 1. 성능평가 지표 요약
Table 1. Summary of performance evaluation metrics

실행 시간	$t_{exec} = C/f_{clk}$
에너지 효율	$\eta_E = \frac{1}{t_{exec} \times Energy}\left[\frac{1}{s \times Joules}\right]$
면적 효율	$\eta_A = \frac{1}{t_{exec} \times Area}\left[\frac{1}{s \times mm^2}\right]$
C : 사이클 개수, f_{clk} : 클럭 주파수	

3. 실험 시스템 사양

표 2는 SIMD 기반의 매니코어 프로세서의 시스템 파라미터를 보여준다.

표 2. 매니코어 프로세서 시스템 파라미터
Table 2. Many-core system parameters

내용	사양
PE 개수	1024개
하나의 레지스터 크기	32Bit
일반 레지스터 개수	16개
검출 레지스터 개수	64개
로컬 메모리 크기	1KB(1024Byte)
내부 연결 구조	Mesh
클럭 주파수	93.43Mhz
영상 크기	256x256

표 3은 매니코어의 성능을 비교하기 위해 선택한 기존 고성능 CPU와 GPU의 사양을 보여준다. CPU는 Intel의 E8500 듀얼 코어를, GPU는 Nvidia의 Geforce 8600GTS를 사용하였다.

표 3. 매니코어와의 성능비교를 위해 선택한 CPU와 GPU 사양
Table 3. CPU and GPU specifications for the performance comparison with the many-core processor

CPU	사양
코어 클럭 주파수	3.16GHz
코어 개수	2

GPU	사양
코어 클럭 주파수	675Mhz
섀더 클럭 주파수	1450MHz
로컬 메모리 크기	512MB
스트림 프로세서 개수	32

IV. 매니코어를 이용한 SIFT알고리즘의 병렬 구현

1. 가우시안 커널

아래 식은 SIFT에 사용되는 각 레이어의 2차원 가우시안 커널을 구하는 식이다[1]. k는 레이어의 단계를 나타내며 $G(x,y,k\sigma)$가 0이 되지 않는 범위가 커널의 크기이다.

$$G(x,y,\sqrt{2^k}\sigma)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}\,|\,1\leq k\leq n \quad (1)$$

x, y : 픽셀의 좌표
n : 총 레이어의 수

가우시안 컨벌루션을 실수연산으로 수행하면 많은 시간이 소모된다. 가우시안 컨벌루션의 연산 시간을 줄이기 위해 커널을 정수로 표현하였다. 아래의 식은 실수형 커널을 정수형 커널로 변환하는 식이다. $I(x,y)$는 x,y좌표에 있는 픽셀 값을 나타내고 $L(x,y,k\sigma)$는 컨벌루션의 결과 값이다. 여기서 p는 정밀도를 나타내며 처리과정에서 오버플로가 발생하지 않는 최대값인 2^{22}으로 정하였다. 가우시안 커널의 모든 값들의 합은 1이므로 컨벌루션한 결과를 p로 나누면 원래의 컨벌루션된 픽셀 값을 얻을 수 있다.

$$G_i(x,y,k\sigma)=[pG(x,y,k\sigma)]$$
$$L(x,y,k\sigma)=\left[\frac{G_i(x,y,k\sigma)*I(x,y)}{p}\right] \quad (2)$$

가우시안 커널은 시그마 값에 따라 커널의 크기와 커널의 값이 달라진다. 본 구현에서는 커널 생성에 소모되는 시간을 줄이기 위해 커널을 동적으로 생성하지 않고 참고 논문[1]의 실험 결과에서 가장 좋은 결과를 보여주었던 시그마 값($\sigma=1.6$)과 증가 값($k=\sqrt{2}$)으로 구한 정수형 커널을 정적으로 구현하였다.

2. 스케일 공간 피라미드

가우시안 컨벌루션은 가우시안 커널 크기만큼의 가장 이웃한 픽셀들의 정보가 필요하다. 가우시안 컨벌루션을 매니코어 프로세서 상에서 구현할 때 가장 큰 문제는 PE간의 컨벌루션에 필요한 이미지 데이터 공유이다. 이를 해결하기 위해서 필요한 이웃 픽셀의 개수에 따라 이웃 PE들은 처리중인 PE로 필요한 이미지 데이터를 전달해 주어야 한다. 전달된 이미지 데이터는 현재 PE의 이미지 데이터와 함께 재배치되고 최종적으로 컨벌루션이 수행된다. 그림 5는 이러한 방법을 통해 가로방향으로 컨벌루션되는 것을 보여준다. SIFT에서 커널의 최대 크기는 27이므로 세로방향과 가로방향에 대해 4개의 이웃 PE들의 이미지 데이터가 필요하다. 컨벌루션된 결과는 다음 레이어를 생성하기 위해 로컬 메모리에 저장된다. 256x256이미지의 각 레이어 이미지는 1024개 PE들의 병렬처리로 각 방향으로 4번의 NEWS통신, 한 번의 재배치 및 가로방향과 세로방향 각각 64번의 컨벌루션 연산으로 구할 수 있다. 다음 옥타브의 첫 레이어는 바로 이전 옥타브 마지막 레이어의 짝수 행과 열에 위치한 픽셀들로 재배치하여 생성한다. 본 구현에서는 이와 같은 방식으로 4개의 옥타브를 생성하였다.

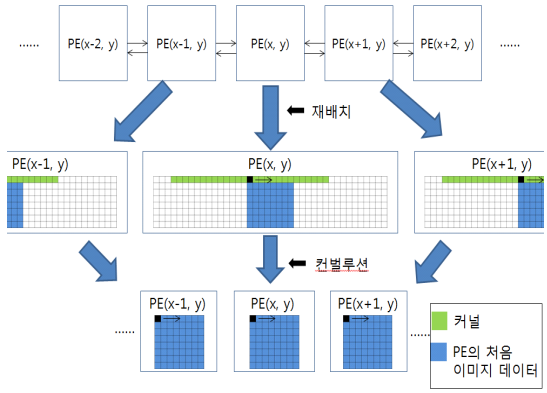


그림 5. 가로방향 가우시안 컨벌루션 수행 과정
Fig. 5. The process of performing a Gaussian convolution in the horizontal direction

3. 특징점 추출

SIFT에서 특징점은 스케일 공간에서의 극값을 나타낸다. 앞서 생성한 DoG를 사용하여 그림 2에서 설명하였듯이 스케일 공간의 이웃 픽셀 값들과 비교하여 극값인지 아닌지를 판별한다. 매니코어 프로세서 상에서 이러한 비교는 PE의 활성

화/비활성화 상태로 구분한다. 먼저 현재 픽셀의 절대값이 특정 임계값을 넘지 못할 경우 해당 픽셀이 위치한 PE는 비활성화 상태가 된다. 다음으로 현재 픽셀이 위치한 DoG 레이어에서 모든 이웃 픽셀보다 가장 크거나 가장 작지 않으면 비활성화 상태가 된다. 마지막으로 다음 스케일의 레이어와 이전 스케일의 레이어 이웃과 모두 비교하여 가장 크거나 작지 않으면 비활성화 상태가 된다. 이 때 활성화 상태의 PE에서 처리중인 픽셀은 특징점이 된다. 이와 같은 작업을 각 PE에 샘플링된 픽셀 수만큼 반복한다.

4. 로컬 메모리 사용 효율화

로컬 메모리 공간을 최소화하기 위해 SIFT 수행 시 불필요한 데이터를 저장하고 있는 메모리 공간을 재사용하였다. 각 옥타브의 레이어 이미지들은 DoG 이미지가 구해진 시점에서 더 이상 사용되지 않는다. 2개의 레이어가 구해지면 하나의 DoG 이미지를 구할 수 있고 첫 번째 레이어는 더 이상 사용되지 않으므로 DoG이미지를 첫 번째 레이어가 저장된 메모리 공간에 저장할 수 있다. 또한 각 옥타브들의 특징점 추출은 독립적으로 수행되기 때문에 하나의 옥타브가 완성되면 특징점을 먼저 추출하고 다음 옥타브를 이전 옥타브의 메모리 공간에 저장할 수 있다.

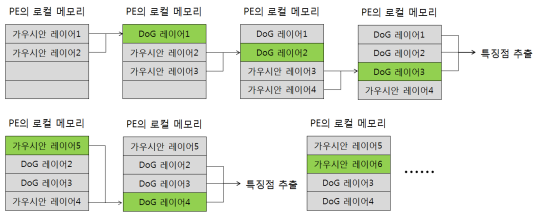


그림 6. 각 옥타브에서의 메모리 사용
Fig. 6. Memory usage for each octave

그림 7은 위에서 설명한 방법을 바탕으로 256x256이미지의 SIFT 특징점 추출을 구현하였을 때 1KB 로컬 메모리의 요소별 사용 분포를 나타낸다. 가우시안 레이어 1-6의 커널 크기는 각각 13Byte, 11Byte, 13Byte, 17Byte, 21Byte, 27Byte이고, 하나의 레이어 이미지 크기는 64Byte이고, 레이어는 총 4개가 있다. Rearrangement 공간은 컨벌루션할 때 재배치 시 사용되는 로컬메모리 크기로 320Byte를 차지한다. Output은 출력 결과를 확인하기 위해 사용되는 출력 이미지 데이터의 저장 공간으로 결과를 확인할 필요가 없다면 사용되지 않는다. 흰색부분은 사용되지 않는 로컬메모리 공간을 나타낸다.

아래의 식은 이미지와 PE개수, 커널 크기에 따른 사용 메모리 공간의 크기를 구하는 공식이다. s_{img}는 이미지 크기, n_{pe}는 PE개수, s_{ki}는 i번째 가우시안 커널 크기이다. m_{pyr}은 피라미드 생성, $m_{\ker}$은 커널 생성, m_{output}은 결과 이미지 출력, m_{rearr}는 재배치에 필요한 메모리 크기이다.

$$m_{pyr} = \frac{s_{img}}{n_{pe}} \times 4,\ m_{output} = \frac{s_{img}}{n_{pe}},\ m_{\ker} = \sum_{i=0}^{N} s_{ki}$$
$$m_{rearr} = \max\left(\frac{s_{img_w}}{n_{pe_w}}/\max(s_{ki}) * \frac{s_{img}}{n_{pe}}, \frac{s_{img_h}}{n_{pe_h}}/\max(s_{ki}) * \frac{s_{img}}{n_{pe}}\right) \quad (3)$$

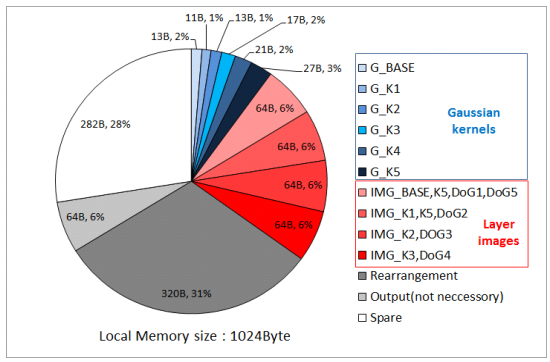


그림 7. 로컬 메모리의 사용 분포
Fig. 7. Distribution of the local memory usage

V. 실험 결과 및 분석

1. SIFT 알고리즘 병렬 구현의 정확성 분석

SIMD기반 매니코어 프로세서를 이용하여 구현한 SIFT 알고리즘은 OpenCV[10]의 코드를 참조하여 구현하였다. 그

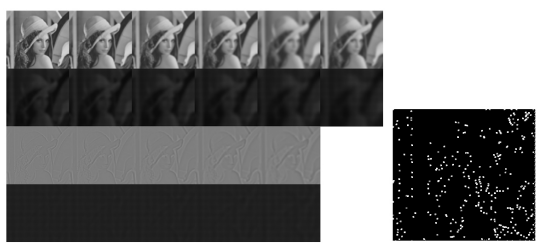


그림 8. 가우시안 피라미드와 특징점 추출 결과
Fig. 8. Gaussian pyramid and the result of feature point extraction

림 8은 매니코어 프로세서에서 구현한 옥타브1과 옥타브2의 가우시안 피라미드, DoG 피라미드 및 특징점 추출 결과를 보여준다. 옥타브2는 옥타브1에서 짝수 픽셀만 샘플링하여 출력한 결과이다. 특징점 추출 결과는 검은바탕의 이미지에 특징점 위치를 흰점으로 표시한 것이다.

표 4는 OpenCV의 SIFT에서 추출한 특징점 수와 매니코어를 이용하여 추출한 특징점 중 위치가 같은 특징점의 수를 보여준다.

표 4. 매니코어와 OpenCV와의 특징점 추출 비교
Table 4. Comparison of feature point extraction using many-core and OpenCV

	이미지1	이미지2	이미지3	이미지4
공통 특징점	23	33	35	54
OpenCV의 특징점	41	56	83	161
매니코어를 이용한 특징점	108	256	180	305

2옥타브부터는 SIFT의 2단계 알고리즘이 적용되어야 올바른 특징점이 만들어지므로 1옥타브에서만 실험하였다. OpenCV에서의 추출 결과는 2단계(특징점 필터링)까지 적용한 것이고 매니코어의 추출 결과는 1단계만 적용하였기 때문에 매니코어 구현의 특징점이 더 많다. 또한 OpenCV의 실수형 정밀도(precision)는 매니코어에서 구현한 정밀도와 다르기 때문에 두 특징점의 집합은 서로 다르고 공통된 특징점 또한 이미지에 따라 추출되는 특징점이 다르기 때문에 다르게 나타난다.

2. 옥타브 별 실행시간 비교

표 5는 93.43Mhz 클럭 주파수에서 동작하는 1024개 PE구조를 사용하여 각 옥타브 별로 성능과 전체성능을 보여준다. 옥타브1에서의 처리 이미지 크기가 가장 크기 때문에 컨볼루션을 수행해야 하는 전체 화소 수가 많아 실행시간이 많이 걸리는 것을 알 수 있다. 구현한 알고리즘에서 전체 옥타브 연산에 수행되는 실행시간이 2.66ms 이므로 초당 30프레임의 실시간 요구조건을 만족함을 알 수 있다.

3. 시스템 이용률 분석

시스템 이용률(System utilization)은 SIFT 알고리즘 수행동안 전체 PE에 대한 활성PE의 비율을 의미한다. 그림 9는 이미지의 특징점 개수에 따른 시스템 이용률의 변화를 보여준다. 피라미드 생성이 끝난 후 특징점 추출 단계에서 비교연산으로 PE의 활성화상태를 선택한다. 따라서 그림 9의 결과에서 특징점의 개수가 많을수록 시스템 이용률이 높다. 하지만 이미지4의 경우 이미지3보다 특징점 추출의 첫 번째 과정에서 더 많은 PE들이 비활성화 상태에서 수행되기 때문에 시스템 이용률이 이미지3보다 낮아졌다. 첫 번째 과정의 비활성화 상태 빈도는 이미지의 주파수에 영향을 받는다. 이는 이미지 주파수가 높으면 높을수록 첫 번째 과정에서 특징점 후보가 많아지고, 상대적으로 PE의 활성화 상태가 더 자주 발

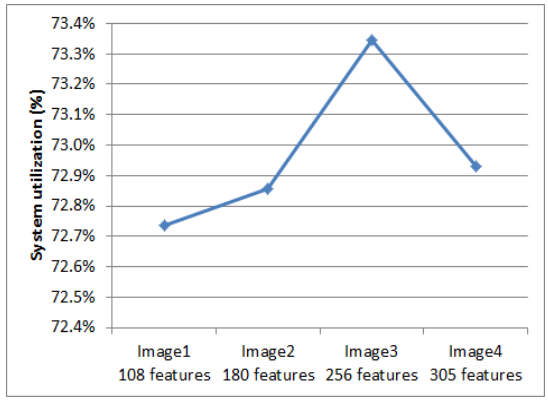


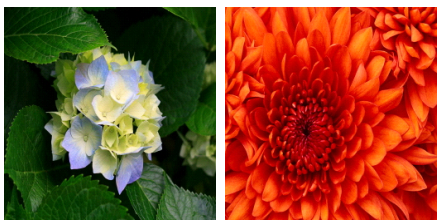
그림 9. 실험 이미지별 시스템 이용률
Fig.9. System utilization for each test image

표 5. 1024개 PE 구조에서의 시스템에서의 옥타브별 성능
Table 5. Performance of the system in each octave from number of 1024 PE structure

옥타브 수	Total cycle (cycles)	Vector instruction	Scalar instruction	System Utilization (%)	Sustained throughput (Gops/sec)	Execution time (ms)
Octave1	186458	148394	38064	72.36	55.1	2
Octave2	46477	37008	9469	71.85	54.74	0.5
Octave3	14428	11333	3095	77.02	57.88	0.16
Octave4	5732	4420	1312	85.27	62.91	0.06
Total	250958	199429	51529	72.74	55.3	2.66

생한다. 그림 10은 실험에 사용된 이미지이다.

〈Image 1〉 〈Image 2〉



〈Image 3〉 〈Image 4〉

그림 10. 실험에 사용한 이미지
Fig. 10. Test images used in the experiment

4. 에너지 효율 분석

에너지 효율은 단위 에너지의 처리량을 나타낸다. 에너지 효율이 높으면 같은 연산을 수행할 때의 전력 소모를 줄일 수 있다. 그림 11은 옥타브 별로 각 레이어의 에너지 효율을 보여준다. 가로축은 옥타브별 각 레이어의 크기를 나타내고, 세로축은 에너지효율(Gops/Joule)을 나타낸다. 레이어1은 첫 옥타브에서만 컨벌루션을 하기 때문에 옥타브2부터는 표시하지 않았다. 옥타브4에서 평균적으로 에너지 효율이 가장 높다. 이는 옥타브4에서 처리해야 할 이미지 크기가 가장 작기 때문이다. 또한 레이어 별 에너지효율 비교에서는 가우시안 커널의 크기가 가장 작은 레이어2에서 에너지 효율이 가장 높다.

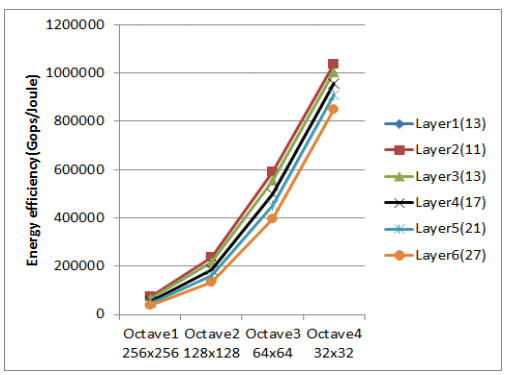


그림 11.옥타브별 각 레이어의 에너지 효율
Fig. 11. Energy efficiency of each layer for different octaves

5. 면적 효율 분석

면적 효율은 시스템의 단위 면적에서 수행할 수 있는 작업량을 의미한다. 그림 12는 옥타브가 차례로 수행될 때 면적 효율의 변화를 보여준다. 이미지의 특징점 추출 개수에 따라 면적 효율이 달라지는데 이미지1은 256개, 이미지2는 108개, 이미지3은 180개의 특징점이 추출되었다. 따라서 발견되는 특징점 수가 많은 이미지일수록 공간 효율이 높다. 또한 특이한 점은 옥타브2가 처리될 때 모든 이미지에서 공간 효율이 모두 가장 낮다. 이는 옥타브2에서 옥타브1보다 더 많은 PE간 통신이 발생하기 때문이다.

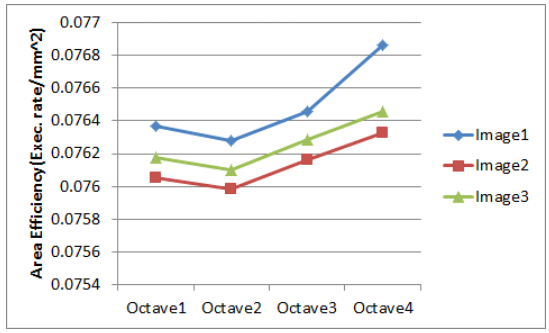


그림 12. 옥타브별 면적 효율 변화
Fig. 12. Area efficiency for different octaves

6. 시스템별 실행시간 비교 및 분석

[11]의 실험결과에 의하면 CPU의 SSE(Streaming SIMD extensions) 명령어로 최적화한 SIFT에서 가장 많은 연산량을 필요로 하는 단계는 1단계(피라미드 생성, 스케일 공간 극값 추출)로 전체 실행시간의 48%를 차지한다. 따라서 1단계의 실행시간만 줄여도 SIFT전체의 성능향상에 크게 도움이 된다. 비교대상으로 사용한 SIFT 알고리즘은 SiftGPU-V400[12]을 사용하였으며 커맨드 라인 인수를 이용해 옥타브 수와 DoG 레이어 수 및 CUDA 사용 유무 등을 조절할 수 있다.

그림 13은 각각의 시스템에서 실행시간을 비교한 그래프이다. 막대그래프에서 밝은 회색부분은 가우시안 및 DoG 피라미드 생성시간을 나타내고, 진한 회색부분은 특징점 추출시간을 나타낸다.

CPU에서의 총 실행시간은 22.86ms이고 CUDA에서의 총 실행시간은 11.06ms이다. SIMD기반의 매니코어 프로세서의 실행시간은 2.66ms로 가장 빠르다.

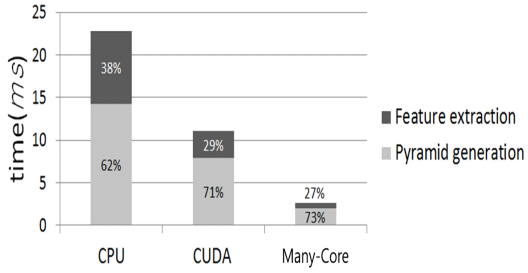


그림 13. 시스템 별 성능 비교
Fig. 13. Performance comparison of each system

그림 14는 각 구현 요소별로 전체 실행시간에서 차지하는 비중을 보여 준다. 컨벌루션이 58.05%로 가장 비중이 크다. 다음으로 비교연산이 많은 특징점 추출이 차지하는 비중은 26.44%이다. 매니코어 프로세서에서 로컬 이미지 데이터의 공유로 인해 수행해야하는 요소인 재배치는 전체 실행시간에서 3번째로 큰 비중을 차지한다.

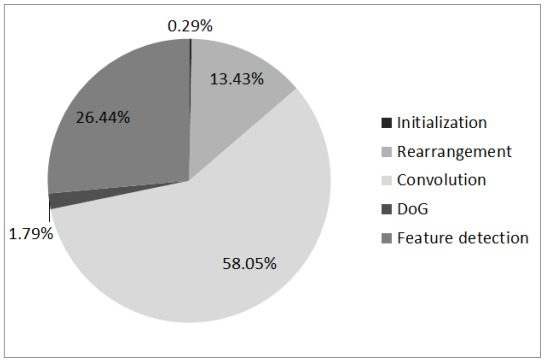


그림 14. 구현 요소별 실행시간 비율
Fig. 14. Percentage of the execution time of each implementation element

VI. 결 론

본 논문에서는 SIFT 알고리즘의 고속처리를 위해 SIMD 기반의 매니코어 프로세서를 제안하였다. 제안한 매니코어 프로세서를 이용하여 특징점 추출을 위한 SIFT알고리즘을 구현하고, 성능, 에너지 효율 및 시스템 면적 효율을 분석하였다. 또한 매니코어 프로세서를 이용하여 구현한 알고리즘을 기존의 고성능 CPU와 GPU를 이용하여 구현한 SIFT 알고리즘과의 성능을 비교하였다. 모의실험 결과, 매니코어는 CPU와의 성능비교에서 415%의 성능 향상을, GPU와의 비교에서는 859%의 성능향상을 보였다. 이러한 결과는 제안하는 SIMD기반 매니코어 프로세서가 SIFT 알고리즘 연산장치로서의 잠재가능성을 보여준다.

참고문헌

[1] D.D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints." International Journal of Computer Vision, Vol. 60, No. 2, pp. 91-110, Nov. 2004.

[2] G. Hua, Y. Fu, M. Turk, M. Pollefeys, Z. Zhang, "Introduction to the Special Issue on Mobile Vision." International Journal of Computer Vision, Vol.96, No.3, pp. 277-279, Feb. 2012.

[3] V. Chandrasekhar, G. Takacs, D. M. Chen, S. S. Tsai, R. Grzeszczuk, B. Griod, "CHoG: Compressed histogram of gradients A Low Bit-Rate Feature Descriptor." in IEEE Conf. on Computer Vision and Pattern Recognition, pp. 2504-2511, 2009.

[4] D. Genbrugge and L. Eeckhout, "Chip Multiprocessor Design Space Exploration through Statistical Simulation." IEEE Transactions on Computers Vol.58, No.12, pp.1668-1681, Dec. 2009.

[5] A. P. Witkin, "Scale-space filtering." in International Joint Conference of Artificial Intelligence, pp. 1019-1022, 1983.

[6] B.-K. Choi, C.-H. Kim, J.-M. Kim, "Implementation of SIMD-based Many-Core Processor for Efficient Image Data Processing." Journal of The Korea Society of Computer Information, Vol. 16, No. 1, pp. 1-9, Jan. 2011.

[7] S. M. Chai, T. Taha, D. S. Wills, J. D. Meindl, "Heterogeneous architecture models for interconnect-motivated system design." IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.8, No.6, pp. 660-670, Dec. 2000.

[8] S. Nugent, D. S. Willis, J. D. Meindl, "A hierarchical block-based modeling methodology for SoC in GENESYS." in 15th Annual IEEE

International ASIC/SOC Conference, pp. 239-243, Sept. 2002.
[9] Y.-M. Kim, C.-H. Hwang, C.-H. Kim, and J.-M. Kim, "Hardware Design and Implementation of a Parallel Processor for High-Performance Multimedia Processing," Journal of The Korea Society of Computer Information, Vol. 16, No. 5, pp. 1-11, 2011.
[10] OpenCV (Open Source Computer Vision), http://opencv.org/
[11] S. Heymann K. Muller, A. Somolic, "SIFT Implementation and Optimization for General-Purpose GPU", in 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Jan. 2007.
[12] Changchang Wu, "SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)", http://cs.unc.edu/~ccwu/siftgpu/

저 자 소 개

김 재 영

2012: 울산대학교
컴퓨터정보통신공학부 공학사
현 재: 울산대학교 대학원
전기전자컴퓨터공학과 석사과정
관심분야: 디지털영상처리,
GPU 컴퓨팅
Email : kjy7097@naver.com

손 동 구

2013: 울산대학교
컴퓨터정보통신공학부 공학사
현 재: 울산대학교 대학원
전기전자컴퓨터공학과 석사과정
관심분야: GPU 컴퓨팅,
컴퓨터 그래픽스,
병렬 프로그래밍,
임베디드시스템
Email : dongkoo88@gmail.com

김 종 면

1989: 명지대학교
전기공학과 공학사
2000: University of Florida
전기컴퓨터공학과 공학석사
2004: Georgia Tech.
전기컴퓨터공학과 공학박사
현 재: 울산대학교 전기공학부 교수
관심분야: 컴퓨터구조, 병렬프로세서,
멀티미디어 신호처리,
SoC설계
Email : jongmyon.kim@gmail.com

전 희 성

1981년 : 서울대학교
전기공학과 공학사
1983년 : 서울대학교 대학원
전기공학과 공학석사
1992년 : 미국 Rutgers - The
State University of New
Jersey
컴퓨터공학과 공학박사
현 재: 울산대학교 전기공학부 교수
관심분야 : 디지털영상처리,
GPU 컴퓨팅, 컴퓨터비전,
증강현실
Email : hsjun@ulsan.ac.kr