openCV 의 성능 향상을 위한 아키텍처 연구

조영필, 허인구, 김용주, 백윤흥
서울대학교 전기정보공학부
e-mail : ypcho@sor.snu.ac.kr

A Study on Architecture Improving Performance of openCV

Yeongpil Cho, Ingoo Heo, Yongjoo Kim, Yunheung Paek
School of Electrical and Computer Engineering, Seoul National University

요 약

최근 컴퓨터 비전의 활용 영역이 증가함에 따라 컴퓨터 비전의 대표적인 라이브러리인 openCV[1]의 사용 또한 증가하는 추세이다. openCV 에는 컴퓨터 비전 알고리즘의 특성상 massive 한 연산을 수행해야 하는 부분이 상당수 존재한다. 본 논문은 이러한 연산량의 부담을 줄임으로써 openCV 의 성능 향상을 위한 아키텍처를 연구한다. openCV 의 massive 한 연산은 라이브러리 함수에 있는 내부 반복문에서 발생하기 때문에, 본 논문은 반복문의 특성을 분석하고 이를 가속할 수 있는 아키텍처가 무엇인지 연구한다. 결론적으로 반복문의 각 iteration 이 독립적일 경우에는 SIMD (Single Instruction Multiple Data)와 SIMT (Single Instruction Multiple Thread)이 적합하며 반복문의 각 iteration 이 의존적일 경우에는 MIMD (Multiple Instruction Multiple Data)를 바탕으로 하는 파이프라인 아키텍처가 적합하다.

1. 서론

최근 컴퓨터 비전의 활용 영역이 급격히 증가하고 있다. 무인 감시 시스템, 의료 영상 분석 등의 전통적인 영역과 증강 현실과 같은 신규 서비스 영역, 또한 닌텐도 Wii 와 마이크로소프트 Kinect 등과 같은 엔터테인먼트 분야에서도 두각을 나타내고 있다. 이렇게 컴퓨터 비전이 두루 활용됨에 따라 관련 라이브러리 개발의 필요성이 증가하였다. 이에, 2006 년 인텔에서는 오픈 컴퓨터 비전 라이브러리인 openCV 를 배포하게 되었다. 배포가 시작된 이래로 openCV 는 최적화된 코드와 편리한 사용방법으로 말미암아 컴퓨터 비전 연구와 상용 어플리케이션 개발에 널리 사용되었다. 현재 배포중인 openCV 의 최신 버전은 2.3.1 이며 유지보수는 로봇 연구 기관인 Willow Garage 에서 담당하고 있다. openCV 에는 컴퓨터 비전 알고리즘의 특성상 massive 한 연산 수행하는 부분이 상당수 있다. 이들 연산은 라이브러리 함수의 내부 반복문에서 실행되기 때문에, 결론적으로 openCV 의 성능을 향상하기 위해선 내부 반복문을 가속할 필요성이 있다.

반복문을 가속하기 위한 아키텍처로는 SIMD, SIMT, MIMD 가 있으며 이들 아키텍처는 ILP (Instruction Level Parallelism)을 바탕으로 LLP (반복문 Level Parallelism)를 구현함으로써 반복문을 가속한다. SIMD 는 다수의 프로세싱 유닛이 동일한 명령어를 서로 다른 입력 데이터에 대해 병렬적으로 실행함으로써 반복문을 가속한다. SIMT 는 GPGPU (General Purpose computation on Graphics Processing Units)에서 제안된 것으로 SIMD 에 스레드 개념과 재빠른 스레드 전환 기능이 추가된 아키텍처이다. 마지막으로 MIMD 는 슈퍼컴퓨터의 초기 아키텍처로, 다수의 프로세싱 유닛이 서로 다른 명령어를 서로 다른 입력 데이터에 대해 병렬적으로 실행하는 개념이며, 구조상 MIMD 는 SIMD 와 SIMT 기능을 모두 포함하게 된다.

반복문을 가속하는데 있어 SIMD, SIMT, MIMD 중 어떠한 아키텍처를 사용할지는 반복문의 특성에 기인한다. 가령, RGB 이미지를 GRAY 이미지로 변환하는 함수처럼 내부 반복문의 iteration 들이 독립적일 경우에는 각 iteration 을 병렬적으로 실행할 수 있기 때문에 SIMD, SIMT, MIMD 를 모두 반복문 가속에 사용할 수 있다. 반면, 이미지의 픽셀 히스토그램 연산 함수와 같이 각각의 픽셀값을 누적해야 하는 등 내부 반복문의 iteration 들이 의존적일 경우에는 매 iteration 을 순차적으로 수행해야 하기 때문에 SIMD 와 SIMT 를 기반으로는 반복문을 가속할 수 없게 된다.

이처럼 반복문의 특성에 기인하여 반복문을 가속할 수 있는 아키텍처가 달라지기 때문에, openCV 에 적합한 아키텍처를 찾기 위해선 1) openCV 의 주요 함수를 분석하고, 2)각 함수 속 내부 반복문의 특성을 살펴보고, 3)해당 반복문을 가속할 수 있는 아키텍처를 찾아야 한다.

이에, 본 논문의 전체적 흐름은 2 장에서 openCV 의 전체적 구성을 살펴보고, 3 장에서 openCV 의 주요 함수 분석 및 이에 적합한 아키텍처를 찾은 후, 4 장에서 결론을 맺도록 한다.

2. openCV 의 구성

openCV 의 전체적인 구조는 그림 1 과 같다. 이 중

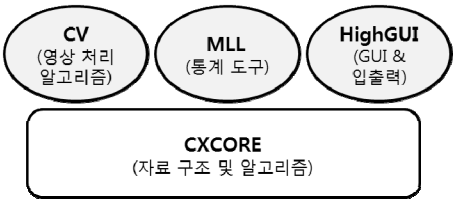

그림 1 openCV 의 구조

openCV 연산의 대부분을 차지하는 것은 실제적으로 영상 처리를 담당하는 CV 영역이다. CV 영역의 코드를 살펴보면 openCV 자체적인 최적화 흔적을 찾아볼 수 있다. 대부분의 반복문은 2~4 회 중첩의 loop unrolling 이 되어있으며 SIMD 로 대체 가능한 연산의 경우에는 인텔의 MMX (Multi Media Extension)와 SSE (Streaming SIMD Extensions) 명령어를 사용하도록 되어 있다.

3. openCV 의 주요 함수 분석 및 이에 적합한 아키텍처

[3]를 참조하면 openCV 에서 사용되는 주요 함수가 실행시간과 함께 나열되어 있다. 분석 대상 함수를

$$\text{Speedup}_{overall} = \text{Execution time}_{old} / \text{Execution time}_{new} = 1 / [(1-\text{Fraction}_{enhanced}) + \text{Fraction}_{enhanced}/\text{Speedup}_{enhanced}]$$

그림 2 Amdahl’s Law

선택하기 위해 Amdahl’s Law 에 따라 실행시간이 긴 분석 대상 함수를 표 1 과 같이 선택하였다.

표 1 분석 대상 함수

함수명	실행시간(Core2)
cvCvtColor (RGB to YCrCb)	2.130
cvFilter2D	1.524
cvIntegral	2.498

3.1. cvCvtColor (RGB to YCrCb)

cvCvtColor 함수는 이미지의 컬러공간을 변환하는 함수이다. RGB, Gray Scale, YCrCb, HSV 등 다양한 컬러공간간 상호 변환가능하며 본 논문에서는 RGB 에서 YCrCb 로 변환하는 경우를 분석하였다.

그림 3 은 이 함수에서 가장 핵심이 되는 커널 반복문을 간략히 표현한 것이다. openCV 에서는 최적화를 위해 픽셀의 비트크기에 따라 구현을 달리하여 비트크기가 8 비트를 초과할 경우에는 그림 3-(a)와 같이 일반적인 컬러공간 변환 공식을 사용하였으며 비트크기가 8 비트일 경우에는 그림 3-(b)처럼 미리 계산된 컬러공간 LUT (Look Up Table)을 사용하여 별로의 산술 연산 없이 테이블 매핑만으로 컬러공간 변환을 구현하였다.

```
for(int i = 0; i < n; i++, src += scn)
    dst[i] = saturate_cast<_Tp>(src[0]*cb
           + src[1]*cg + src[2]*cr);
```
(a) 변환 공식 사용

```
for(int i = 0; i < n; i++, src += scn)
    dst[i] = (uchar)(_tab[src[0]] +
           _tab[src[1]+256] + _tab[src[2]
           +512]) >> yuv_shift);
```
(b) LUT 사용

그림 3 cvCvtColor 함수의 커널

이처럼 픽셀의 비트크기에 따라 컬러 공간을 변환하는 방법이 달라지기 때문에 이 함수에 적합한 아키텍처 또한 변환 방법에 따라 달라진다. 먼저 변환 공식을 사용하는 경우에 반복문의 각 iteration 이 독립적이며 동일한 연산을 수행하기 때문에 SIMD 와 SIMT 아키텍처가 적합하다. 반면에 LUT 를 사용하는 경우에는 단순히 메모리를 엑세스 하는 것에 불과하므로 메모리 대역폭이 충분하다면 어떠한 아키텍처를 사용해도 무방하다.

3.2. cvFilter2D

필터링은 컴퓨터 비전에서 가장 많이 쓰이는 연산 중 하나이다. 필터링은 이미지와 n*n 의 필터 행렬의 convolution 연산을 통해 이루어지며, 이때 convolution 연산의 복잡도를 줄이기 위해 주파수 영역에서 필터링을 진행하기도 한다. openCV 에서도 필터 행렬 원소의 개수가 50 을 초과할 경우 FFT (Fast Fourier Transform)을 통해 주파수 영역에서 필터링을 진행하게 된다. 하지만 많은 경우에 5x5 의 필터 행렬을 사용하므로 본 논문에서는 공간 영역에서 필터링이 이루어 지는 경우를 분석하였다.

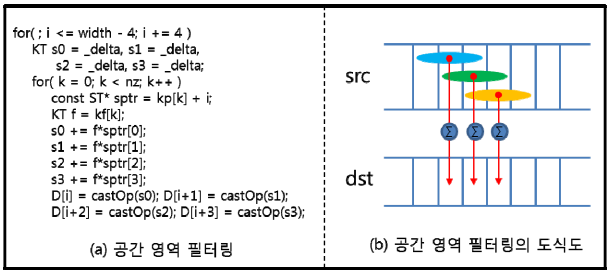

그림 4 cvCvtColor 함수의 커널

그림 4-(a)에 cvFilter2D 함수의 커널 반복문이 간략하게 나타나 있다. i-반복문과 k-반복문이 중첩된 형태를 가지며 이를 그림 4-(b)의 도식도를 통해 살펴보면, k-반복문은 convolution 연산을 담당하고 i-반복문은 convolution 연산을 반복적으로 수행하도록 함을 확인할 수 있다. 결과적으로 i-반복문은 각 iteration 이 독립적이며 k-반복문은 각 iteration 이 의존적임을 알 수 있다.

이러한 특성에 의거했을 때 k-반복문은 SIMD 와 SIMT 아키텍처에 적합하지 않다. 반면에 i-반복문은

각 iteration 이 독립적이므로 SIMD 와 SIMT 를 적용할 수 있을 것처럼 보인다. 하지만, i-반복문의 iteration 이 반복문형태를 띄므로 i-반복문은 SIMD 가 아닌 SIMT 아키텍처를 통해 가속할 수 있음을 알 수 있다.

3.3. cvIntegral

이미지 integral 은 얼굴 인식 등에서 이미지의 부분합을 빠르게 계산하기 위해 사용되는 알고리즘이다.

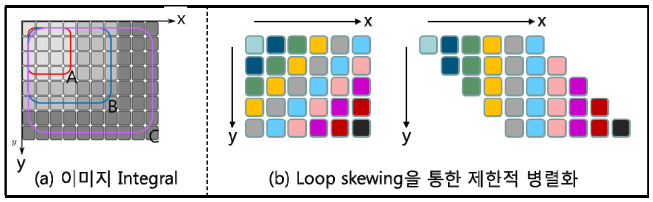

그림 5 Integral 알고리즘

그림 5-(a)에서 A 에는 이미지의 0,0 에서 A 까지의 합이 저장되며 B 에는 0,0 에서 B 까지의 합이 저장된다. 이때 이미지의 A 부터 B 까지의 부분합은 B-A 를 계산함으로써 빠르게 구할 수 있다.

이처럼 각 픽셀의 값을 누적해야 하는 integral 알고리즘의 특성상 SIMD 와 SIMT 를 통해 내부 반복문을 가속할 수 없다. 그렇기 때문에 그림 5-(b)와 같이 코딩 기술 중 하나인 Loop Skewing 을 통해 integral 연산을 가속 할 수 있다. 일반적으로 반복횟수가 n 인 외부 반복문과 반복횟수가 m 인 내부 반복문이 중첩된 임의의 반복문이 있을 경우 복잡도는 O(n*m)이 된다. 이때, 내부 반복문을 SIMD 를 통해 가속할 경우 복잡도는 O(n)로 감소하며, Loop Skewing 을 통해 전체 반복문을 가속할 경우 복잡도는 O(2n + c)가 되어 결국 내부 반복문을 SIMD 로 가속할 경우와 같은 O(n)의 복잡도를 가지게 된다. 물론 Loop Skewing 의 경우 연산이 복잡하고 코드 크기가 증가함에 따라 c 값이 커지게 되므로 실질적으로 SIMD 를 사용한 경우보다 빠를 순 없다.

cvIntegral 과 같이 SIMD 와 SIMT 로써 가속할 수 없는 경우에도 MIMD 로 가속을 할 수 있다. 방법은 modulo 스케줄링을 반복문에 적용하고 그 결과를 MIMD 에 매핑하는 것이다. 이때 하나의 반복문만 매핑하는 것이 아니라 다수의 연속적인 반복문을 MIMD 에 매핑할 수 있기 때문에 연속적인 반복문들이 파이프라이닝되는 효과를 얻게 된다.[4] 그림 6-(a)에 임의의 반복문의 DFG (Data Flow Graph)가 나타나 있다. 이를 4x2 MIMD 아키텍처에 modulo 스케줄링을 통해 매핑하였다(II=2). 이때 5 개의 순차적인 반복문을 파이프라이닝 하여 MIMD 에 매핑할 수 있으며, 그 결과는 그림 6-(b)와 같다. 그림에서 회색, 하늘색, 붉은색, 보라색, 하얀색으로 표시된 매핑이 각각 하나의 반복문을 나타낸다. 이처럼 5 개의 반복문을 MIMD 에 파이프라이닝하여 매핑할 경우 MIMD 에 매핑하지 않은 경우에 비해 5 배의 성능향상을 얻을 수 있다.

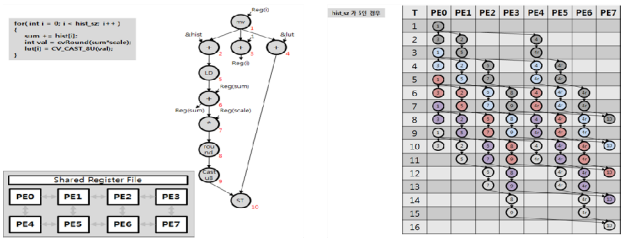

그림 6 MIMD 를 통한 반복문의 가속

4. 결론

openCV 는 각 함수의 형태에 따라 SIMD, SIMT, MIMD, 혹은 일반 RISC, CISC 아키텍처에서 최적의 성능을 보이게 된다. RISC 와 CISC 를 제외하면 SIMD 와 SIMT 는 모두 MIMD 를 통해 구현이 가능하다. 다만 이를 위한 오버헤드가 추가됨으로 MIMD 를 SIMD, SIMT 화 한다 하더라도 순수 SIMD, SIMT 에 비해선 성능감소가 있게 된다. 그러므로 openCV 를 지원하기 위한 아키텍처라 할지라도 확실한 목적성이 있고 또, 사용된 함수들를 SIMD, SIMT 로 가속할 수 있을 경우에는 SIMD, SIMT 위주의 아키텍처를 구성하는 것이 효율적이다. 반면에 openCV 를 전반적으로 지원해야 하거나 확실한 목적성이 있을 지라도 사용된 함수들을 SIMT, SIMT 로 가속할 수 없을 경우에는 MIMD 를 위주로 하여 파이프라이닝 아키텍처를 구성하는 것이 전체적인 컴퓨터 비전처리의 성능 향상에 효율적이다.

참고문헌

[1] http://opencv.willowgarage.com/wiki/
[2] Shorin Kyo and Shin'ichiro Okazaki, "IMAPCAR: A 100 GOPS In-Vehicle Vision Processor Based on 128 Ring Connected Four-Way VLIW Processing Elements", JOURNAL OF SIGNAL PROCESSING SYSTEMS Volume 62, Number 1, 5-16
[3] Hiroki Sugano and Ryusuke Miyamoto, "OPENCV IMPLEMENTATION OPTIMIZED FOR A CELL BROADBAND ENGINE PROCESSOR", Digital Signal Processing Workshop and 5th IEEE Signal Processing Education Workshop, 2009. DSP/SPE 2009. IEEE 13th
[4] B. Mei, S. Vernalde, D. Verkest, H. De Man and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling", Computers and Digital Techniques, IEE Proceedings - 22 Sept. 2003, Volume 150 Issue:5, 255-61

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2011-0000975), 2011 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No.2011-0018609) 및 IDEC 의 지원을 받아 수행되었습니다.