병렬 영상처리 기반의 고속 머신 비전 기술동향

본 고에서는 병렬 영상처리를 이용한 고속 머신 비전(Machine Vision) 기술의 동향에 관해 다룬다. 머신 비전에서 사용되는 대표적인 고속 상용 영상처리 라이브러리인 MIL, HALCON, IPP에 대해 소개하고 현재 활발히 연구되고 있는 SSE, OpenMP, CUDA와 같은 병렬 처리 기술에 대하여 알아 본다. 이러한 병렬 처리 기술을 실제 영상처리 알고리즘에 적용하여 그 성능을 고속 상용 영상처리 라이브러리의 성능과 비교하고 소개된 병렬 처리 기술을 실제 PCB 기판 자동검사와 같은 머신 비전에 적용한 연구사례에 대해서 알아본다.

■ 박은수, 최학남, 김준철, 정용한, 김학일*
(인하대학교)

1. 서론

25년 전부터 제조업 분야에서는 높은 품질과 비용 절감 이라는 목표 달성을 위해 머신 비전 시스템을 도입하였다. 최근의 향상된 광학 기술과 컴퓨터 비전 기술 그리고 제어 기술은 과거의 이러한 목표를 현실화 시켜주고 있다. 광학 기술과 제어 기술의 발전은 머신 비전 시스템에서 더 높은 고 해상도의 이미지를 얻을 수 있게 해주었고 이를 처리 하기 위한 컴퓨터 비전 기술은 더욱 빠른 처리 속도를 요구 하게 되었다. 또한 제조업 분야에서 현재의 머신 비전 시스템은 더 높은 품질을 위해서 컬러 정보와 3차원 정보를 이용하는 추세여서 처리해야 할 데이터는 지속적으로 증가하고 있는 중이다. 대용량 영상 데이터를 고속으로 처리하는 것은 생산력과 기술력 증가로 이어져 머신 비전 업계의 경쟁력을 확보하는 중요한 수단이 되고 있다.

머신 비전 시스템에서 사용되는 영상처리 기술의 처리 속도는 빠른 연산 속도를 갖는 프로세서와 이를 병렬화 함으로써 실현 될 수 있다. 대표적인 CPU (Central Processing Unit) 제조 회사인 인텔이나 AMD의 프로세서는 다수의 데이터를 한번의 연산으로 처리 할 수 있는 벡터 연산을 지원한다. 출시되는 제품은 이러한 프로세서를 여러 개 모은 멀티 코어 시스템이기에 이를 머신 비전 기술에 효과적으로 활용하기 위해서는 벡터 연산과 멀티 코어 시스템에 적합한 프로그래밍 기술이 필요하다. 또 다른 대안으로 현재까지 활발히 연구 되고 있는 GPGPU (General Purpose Graphics Processing Unit) 기술을 활용할 수 있다. 이는 수백 개의 프로세서를 내장하고 있는 GPU의 연산능력을 일반적인 연산에 활용하는 기술로 현재 CUDA (Compute Unified Device Architecture)를 활용한 기술이 활발히 연구되고 있다.

본고에서는 현재 활발히 연구 되고 있는 병렬화 기술과 이를 영상처리 알고리즘에 적용한 머신 비전 기술의 개발 동향에 관해 살핀다. II장에서는 머신 비전에서 대표적으로 사용되고 있는 고속 상용 영상처리 라이브러리에 대해 소개하고, III장에서는 현재 활발히 연구 되고 있으며 고속 영상처리 라이브러리의 성능을 실현 시켜 주는 대표적인 병렬화 기술을 소개한다. IV장에서는 III장에서 소개한 병렬화 기술을 적용한 영상처리 알고리즘의 성능과 고속 상용 영상처리 라이브러리와의 성능을 비교하고, V장에서는 이러한 병렬화 기술을 실제 머신 비전에 적

용한 연구 사례에 대하여 설명하며 VI장에서 결론을 맺는다.

2. 머신 비전 라이브러리 소개

2.1 MIL (Matrox Imaging Library)

캐나다 Matrox사의 MIL®은 카메라와 프레임 그레버(frame grabber)를 통하여 획득된 영상을 제어하고 프로세싱 라이브러리를 사용하여 고속으로 영상을 처리 할 수 있는 머신 비전용 개발 툴이다. MIL은 이미지의 획득과 전송 및 출력, 처리 및 분석 등을 쉽게 할 수 있도록 개발 되어 있으며, 9.0버전부터 64비트 환경을 지원하고 있다. MIL은 그림 1과 같이 영상획득, 출력, 전송, 기본적인 데이터 조작 등을 할 수 있는 MIL-Lite®의 기능을 포함하면서 확대된 영상처리와 분석 등을 가능하게 한다. 영상처리는 점 대 점 (point-to-point), 공간적 필터(spatial filter), 형태론적 (morphological), 기하학적(geometric) 영상 처리 및 고속 푸리에 변환(fast fourier transform) 등의 다양한 기능을 포함 하고 있다. MIL에서 가능한 기본적인 기능들은 다음과 같다[1-2].

- 흑백 영상이나 컬러 영상을 16 비트 까지 획득 가능
- 1, 8, 16, 32 비트 정수나 부동 소수점(floating point) 영상 처리 가능
- 컬러 이미지의 각 채널에 따른 처리 가능
- 1, 8, 16 비트 흑백 영상이나 컬러 영상 출력 가능

2.2 HALCON

독일 MVTec사에서 개발한 HALCON 라이브러리는 FPD(Flat Panel Display)/반도체 기타 자동화 장비 등의 머신 비전과 의료 영상(medical imaging) 및 영상 처리 분야에서 사용되는 패턴 매칭, 기하학적 외곽선 검출, blob 분석, OCR, 코드 리더(Code Reader), 2D/3D Calibration, 영상획득, 영상 압축 및 저장 등의 함수를 제공하는 영상처리 라이브러리 이다. 또한 HACON 라이브러리는 Visual C++/Visual Basic/Delphi 등의 다양한 프로그래밍 언어와의 인터페이스를 통하여 1300개가 넘는 HALCON 연산자들에 대한 강력한 사용 환경을 제공한다. HALCON 라이브러리의 특징을 요약하면 다음과 같다.

- 머신 비전을 위한 실시간 이미지 프로세싱
- 회전과 부분적으로 가리워진 물체를 인식하는 강인한 패턴 매칭
- 50개 이상의 모양(shape)과 그레이 값(gray value)의 특징을 분석하는 blob 분석
- 멀티프로세서와 멀티코어 컴퓨터의 지원
- 50개가 넘는 프레임 그레버와 수백 개의 산업용 카메라 지원
- 1/50의 pixel 정확도를 갖는 선, 원, 타원형 검색
- 빠른 형태론적 영상처리 알고리즘
- 컬러 영상처리 알고리즘
- 대용량 이미지 처리
- SSE (Streaming SIMD Extensions) 기술을 이용한 높은 성능

HALCON은 그림 2와 같은 유연성 있는 구조를 갖고 있어 다른 운영체제로의 이식, 또는 새로운 프로그래밍 환경과의 통합과 같은 상위 개발 환경과의 호환성을 보장한다. 이러한 구조로 인하여 사용자는 통합된 추가적인 이미지 획득 장치에 관한 새로운 인터페이스를 개발할 수 있다[3-4].

2.3 IPP (Integrated Performance Primitives)

IPP는 인텔에서 출시한 라이브러리로 SSE기술을 이용하여 개발 되었다. IPP는 신호처리, 이미지 처리, 행렬 처리, 3D 데이터 처리 등에 대한 천여 개의 다양한 함수를 제공하고 있다. IPP는 인텔 프로세서에 최적화 되어 있고, 구 버전과 신 버전 사이의 호환성을 유지하며, 최소의 코드와 메모리 사용으로 고속 처리를 보장한다. 또한 기존의 기타 라이브러리에 비해 가격이 싸

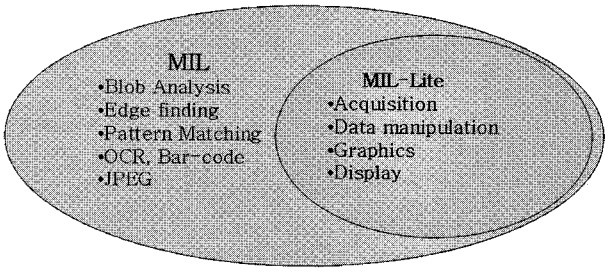


그림 1. MIL 및 MIL-Lite의 주요 구성

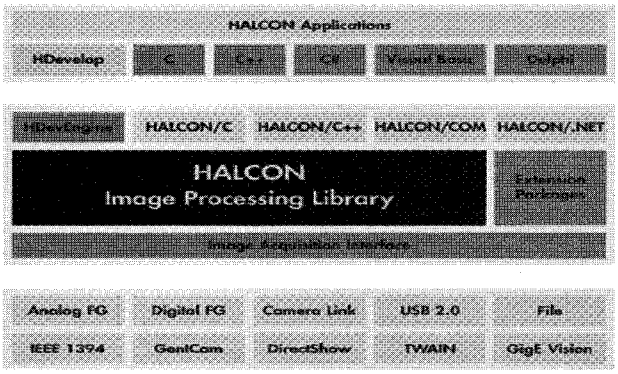


그림 2. HALCON의 기본 구조[3]

고, IPP 를 이용해 개발한 제품에 대한 로열티도 따로 받지 않으므로 가격과 성능을 모두 고려한 라이브러리라고 할 수 있다.

IPP 라이브러리를 사용 했을 때의 가장 큰 이점은 멀티코어 프로세서의 지원과 성능 최적화된 함수를 제공한다는 점이다. IPP 함수는 SSE와 같은 프로세서 가용 기능과 기타 최적화된 명령 집합에 기초한 로우 레벨(low level) 최적화 함수 알고리즘을 일치시킴으로써 최적화된 컴파일러 자체가 제공 할 수 잇는 것 이상의 성능을 제공하도록 고안되었다. 다음은 IPP 라이브러리 함수가 제공하는 기능이다[5].

- 이미지 프로세싱
- 비디오 코딩
- 데이터 압축
- 컴퓨터 비전
- 오디오 코딩
- 암호화
- 광선 추적/ 렌더링
- 신호 처리

3. 머신 비전에 적용되는 대표적인 병렬 처리 기술

3.1 SSE (Streaming SIMD Extensions)

SIMD (Single Instruction Multiple Data)는 그림 3과 같이 하나의 명령어로 다수의 데이터를 처리 할 수 있는 프로그래밍 구조를 말한다. 현재 대표적인 프로세서인 인텔과 AMD의 CPU는 SIMD 연산에만 사용되는 16개의 128비트 레지스터와 SSE로 불리는 SIMD 명령어 집합을 보유하고 있다[6].

SSE는 프로그램의 복잡성을 줄여주며 다양한 연산을 수행하는 함수를 추가하여 SSE2, SSE3, SSSE3 (Supplemental SSE3), SSE4의 형태로 지속적으로 발전하였고 현재도 보다 높은 효율의 SIMD 연산을 위한 개발이 진행 중이다. 그림 4는 SSE의 발전 과정을 나타낸다.

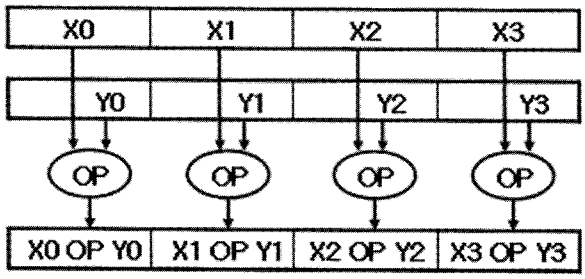


그림 3. SIMD 연산구조[6]

3.1.1 SSE 코딩 기술

SSE 명령어를 이용한 SIMD 기술은 새로운 방법의 코딩 알고리즘을 필요로 한다. SIMD 연산의 이점을 충분히 활용하기 위해선 128비트 레지스터 (인텔에서 XMM 레지스터로 칭함)의 크기에 맞게 데이터들을 정렬하고 벡터화 시켜야 한다. SSE 명령어를 이용하여 프로그램을 작성하는 방법은 다음과 같이 4가지로 나뉜다[6].

- 어셈블리 코드로 작성하는 방법
- Intrinsics 코드로 작성하는 방법
- 클래스를 활용하는 방법 (C/C++ 코딩 형태)
- 자동 벡터화를 사용하는 방법

그림 5는 SSE의 구현 방법에 따른 어플리케이션의 성능 대비 코딩의 편리성과 이식성의 상관 관계를 보여준다.

컴파일러의 성능이 좋아지면서 자동 벡터화를 사용하는 방법이 많이 늘어나고 있는 추세이다. 그러나 자동 벡터화를 사용

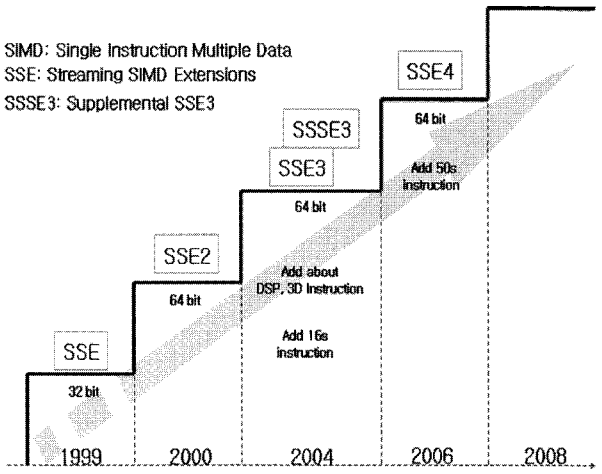


그림 4. SSE의 발전 과정

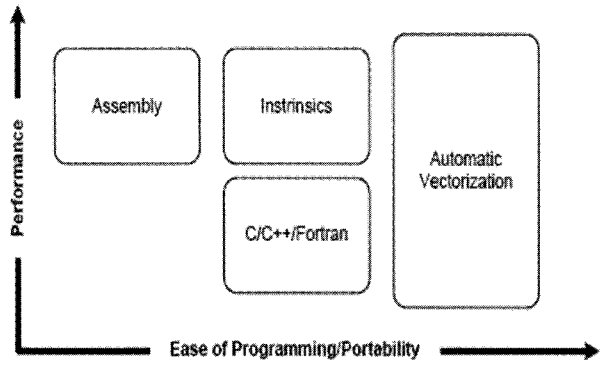


그림 5. 성능 대비 코딩의 이식성과 프로그래밍의 편리성

할 경우 그림 5에서 볼 수 있는 것처럼 성능 변화의 범위가 넓게 나타난다. 이는 데이터 참조의 의존성이 높거나 루프구조가 복잡한 알고리즘의 경우 컴파일러를 이용한 자동 벡터화가 활성화 되지 않음을 보여준다. SSE를 이용한 효율적인 영상처리 알고리즘의 구현방법은 [6]에서 찾아 볼 수 있다.

3.2 OpenMP

OpenMP는 공유메모리 환경에서 다중 스레드(thread) 병렬 프로그램을 작성하기 위한 응용 프로그램 인터페이스이다. MPI (Message Passing Interface)가 분산 메모리 병렬처리 시스템의 표준인 것처럼 OpenMP는 공유 메모리 병렬 컴퓨팅의 사실상 표준이며 현재 많은 업체들이 표준화를 지원하고 있다.

3.2.1 OpenMP의 구성

OpenMP는 그림 6과 같이 컴파일러 지시어(directives), 실행시간 라이브러리(runtime library), 환경변수(environment variables)로 구성 된다[7].

- 컴파일러 지시어: 기본적인 의미에서 OpenMP는 공유메모리 병렬성을 표현하기 위한 컴파일러 지시어들의 집합이다. 컴파일러 지시어 집합은 다시 병렬성 표현을 위한 제어 구조, 스레드들 간의 통신을 위한 데이터 환경 구문, 다중 스레드의 실행을 조율하는 동기화 구문의 세 부분으로 구성된다.
- 실행시간 라이브러리: 병렬 실행에 참여하는 스레드 개수, 스레드 번호 등과 같은 병렬 인수들의 설정과 조회를 가능하게 한다.
- 환경 변수: 병렬 실행에 참여하는 스레드 개수의 지정과 같은 실행 시스템의 병렬 인수들을 정의하는데 이용한다.

3.2.2 OpenMP의 프로그래밍 모델

OpenMP는 스레드를 기반으로 하는 공유 메모리 프로그래밍 모델이며 프로그램의 병렬 실행을 위해 Fork-Join 모델을 사용한다. Fork-Join 모델은 그림 7과 같이 병렬 구간에서 팀 스레드를 생성하거나, 팀 스레드를 동기화 시켜 작업을 실행하고, 직렬 구간에서 팀 스레드의 작업 종료를 병합하는 스레드 모델이다.

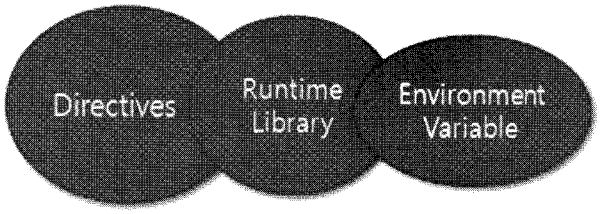


그림 6. OpenMP의 구성

OpenMP를 이용한 병렬 프로그래밍은 순차 코드에 필요한 부분만 지시어를 삽입하는 컴파일러 지시어 기반으로 구현된다. 사용자는 병렬화가 필요한 부분에 적절한 OpenMP 지시어를 삽입하고 컴파일러는 삽입된 지시어를 참고하여 다중 스레드 코드를 생성한다. 삽입된 지시어가 컴파일러에 의해 인식되어 병렬화 코드가 생성되는 것이므로 반드시 OpenMP를 지원하는 컴파일러가 필요하다. 예를 들어 영상처리를 위한 프로그램들은 실질적인 CPU 시간을 do (또는 for) 루프를 실행하기 위해 소비한다. 사용자는 루프 실행을 여러 프로세서 들에게 할당 시켜 동시에 작업하도록 하는 루프 수준 병렬성을 이용하여 이러한 프로그램들의 실행 시간을 줄일 수 있다. 병렬화 할 수 있는 루프들에 대해 OpenMP는 컴파일러 지시어를 발견하는 즉시 루프를 병렬화 할 수 있도록 하는 간단한 지시어를 제공한다[7].

3.3 CUDA

GPU는 그래픽 렌더링 작업을 가속화 하기 위해 만들어졌다. 현재의 GPU 하드웨어 구조는 수백 개의 코어를 지니고 있는 병렬 구조로 그래픽 분야뿐 아니라 다른 높은 계산량을 갖는 분야

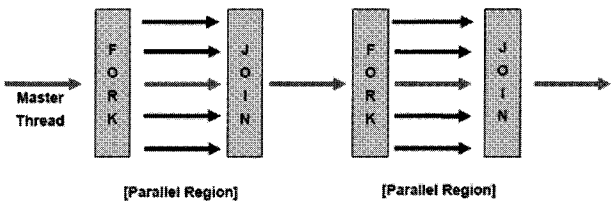


그림 7. Fork-Join 모델

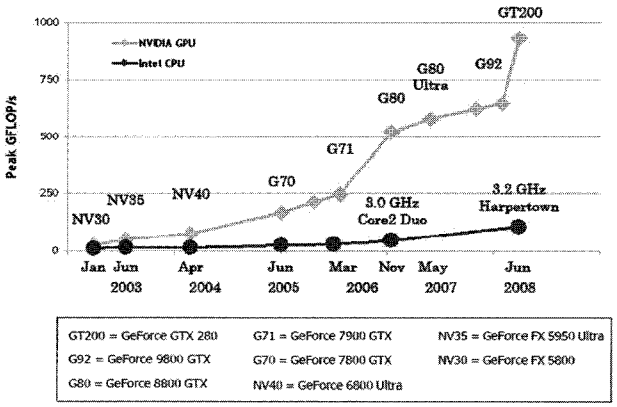


그림 8. CPU와 GPU의 계산 속도

에서도 이용되고 있다. 이러한 하드웨어 구조로 인하여 그림 8과 같이 GPU의 계산 속도는 CPU를 앞서고 있으며 지속적으로 발전하고 있다[8].

CUDA는 강력한 성능의 GPU를 활용하기 위해 설계된 하드웨어와 소프트웨어 구조를 말한다. CUDA를 사용 할 경우 그래픽스 관련 지식이 없이도 친숙한 C언어를 통해 프로그래밍 할 수 있기 때문에 프로그램 작성이 상대적으로 쉽다.

3.3.1 CUDA 프로그래밍 모델

CUDA는 프로그래머에게 병렬 프로그래밍 언어의 구조에 중점을 두지 않고 병렬 알고리즘 개발에 중점을 둘 수 있도록 돕는다. CUDA는 병렬 처리의 대상으로 C 함수를 정의하는 방법과 같이 커널 (kernel)을 정의하며 이렇게 정의 된 커널은 CUDA 스레드를 이용하여 병렬처리 된다. 하나의 커널은 그림 9와 같이 __global__ 이라는 선언문을 사용하여 정의되며 이 커널에 사용될 스레드의 숫자는 <<<…>>> 문장을 이용하여 표시 할 수 있다. 커널을 수행하는 스레드는 고유의 스레드 ID가 주어진다. 그림 9는 크기가 N x N인 두 개의 메트릭스 A, B의 합을 구하고 그 결과를 C에 저장하는 과정을 나타낸다. 그림 9의 threadIdx 내장 변수는 3개의 변수를 갖는 벡터로 구성되어 있기 때문에 1D, 2D, 3D의 스레드 블록이 구성될 수 있다. 다차원의 스레드일 경우 그 스레드의 ID는 다음과 같이 구할 수 있다: 스레드가 2D 블록 (Dx, Dy)를 구성 할 경우 스레드 인덱스 (x, y)의 ID는 (x+yDx)가 되며 3D 블록 (Dx, Dy, Dz)를 구성 할 경우 스레드 인덱스 (x, y, z)의 ID는 (x+yDx+zDxDy)가 된다.

커널은 위와 같은 방식으로 스레드 블록으로 나뉘어져 처리되기 때문에 커널을 처리하는 총 스레드의 숫자는 [블록당 스레드 숫자 x 블록 숫자] 가 된다. 또한 그림 10과 같이 커널을 구성하는 블록을 그리드(grid)라고 부르며 1D와 2D 로 구성 할 수 있다.

그리드의 구성은 그림 9의 <<<…>>> 문장의 첫 번째 매개 변수로 명시할 수 있으며, 커널 내에서 내장 변수인 blockIdx 로 접근 가능하고 그리드의 차원은 내장 변수 blockDim 으로 접근 할 수 있다. 그림 9의 메인 함수 내의 dimBlock은 블록의 크기를 16 x 16, 즉 스레드를 256개 생성한다는 뜻이다. dim3 타입은 3개의 unsigned int 변수를 갖는 데이터 타입이다. 3개 중 설정하지 않은 변수는 자동으로 1로 설정된다. 그림 9의 dimGrid에서 그리드를 생성하는 블록을 하나의 스레드가 하나의 메트릭스 요소를 처리 할 수 있도록 설정 하였다. 스레드 블록은 독립적으로 실행되며 이러한 스케쥴링을 프로그래머가 직접 작성하기 때문에 확장 가능한 코드 작성이 가능하게 되는 것이다[8].

3.3.2 호스트와 디바이스 (Host and Device)

그림 11에서 보여지는 것처럼 CUDA 스레드들은 C 프로그램을 동작시키는 호스트(CPU)와 물리적으로 나뉘어진 디바이스(GPU) 에서 실행된다. 디바이스는 자체적인 DRAM을 보유하고 있다. 디바이스 메모리는 전역(global), 상수(constant), 텍스쳐(texture) 메모리로 나뉘어지며 이 들 메모리의 할당과 해제를 위해서는 CUDA 명령어를 이용해야 한다. 그림 11은 호스트와 디바이스가 어떻게 작동되는지 설명한다. 순차코드는 호스트에

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x,
                 (N + dimBlock.y - 1) / dimBlock.y);
    MatAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

그림 9. __global__ 선언문과 〈〈〈…〉〉〉 문장

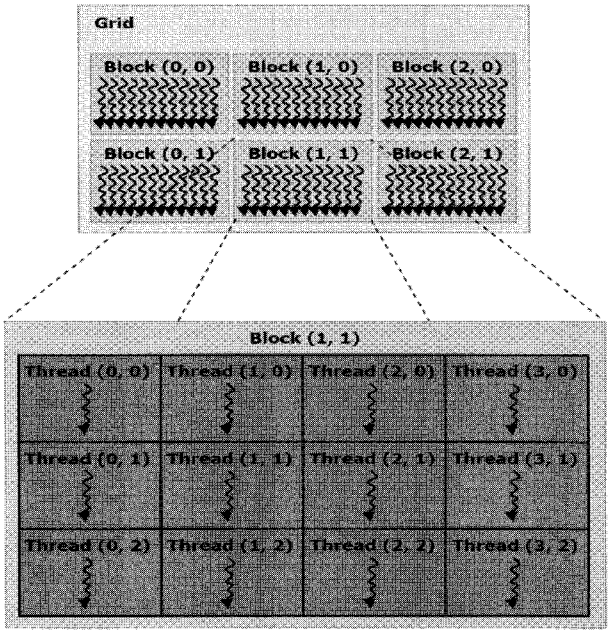


그림 10. 그리드를 구성하는 스레드 블록

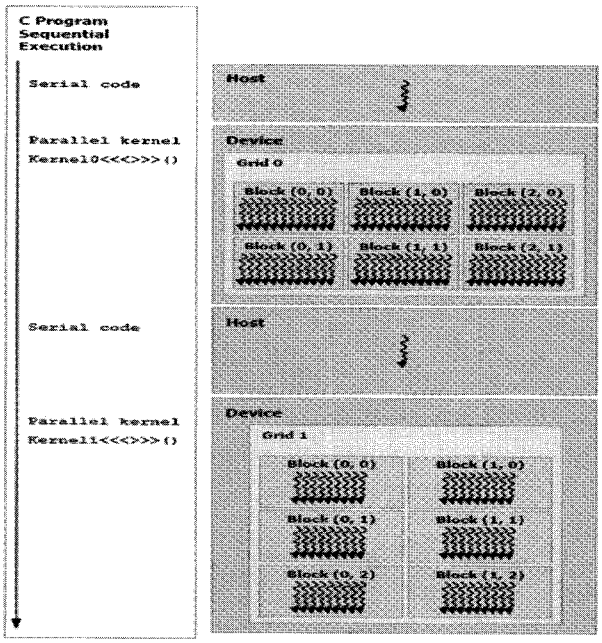


그림 11. CUDA응용 프로그램의 실행 과정

서 실행되고 병렬로 실행되는 커널 부분은 디바이스에서 실행된다[8].

4. 병렬 영상처리 기술의 성능

본 장에서는 대표적인 상용 영상처리 라이브러리의 성능과 앞 절에서 설명한 병렬화 방법을 이용하여 직접 구현한 대표적 영상처리 알고리즘에 대한 성능 평가를 설명한다. 머신 비전에 적용 되는 영상처리 알고리즘들과 고속화가 필요한 부분은 어플리케이션에 상당 부분 의존적이기 때문에 목적 어플리케이션에 효율적인 알고리즘을 개발하고 이를 최적으로 확장하기 위해서는 라이브러리에 의존적인 방법보다는 직접 알고리즘을 구현 하여 고속화 할 수 있는 방법이 효율적이다.

먼저 단일 프로세서에서 수행 될 수 있는 데이터 병렬화 방법인 SIMD를 이용하여 구현한 결과를 상용 영상처리 라이브러리와 비교해 보고 멀티 프로세서 병렬화 방식인 OpenMP 방법과 GPU를 활용한 CUDA를 비교한다.

성능 평가를 위해서 본 실험에 사용된 영상 처리 알고리즘은 전처리 과정에서 많이 사용되는 평균화 필터(mean filter), 형태론 영상처리 방식인 팽창(dilation)과 침식(erosion), 그리고 소벨(Sobel) 수평 방향 외곽선 검출자를 선정하였다. 실험에 사용된 프로세서는 인텔 코어 2 듀오 데스크탑 프로세서로 clock speed

표 1. SSE 방법과 상용 라이브러리의 성능 비교

Method / Function	Seq.	MIL 8.0	HALCON 8.0	IPP 5.1	SSE
Mean	117.29	37.39	71.05	7.64	24.08
Dilation	70.40	15.05	6.30	11.60	10.10
Erosion	56.62	13.85	6.26	11.49	10.47
Sobel Horizontal	168.85	30.12	91.43	14.81	14.32

는 2.40GHz, 4MB 크기의 L2 캐쉬를 갖고 있다. 영상처리 알고리즘은 4057 x 4048 크기의 PCB (Printed Circuit Board) 이미지에 적용되었다.

4.1 데이터 병렬화 방법의 성능 비교

실험에 사용된 상용 영상 처리 라이브러리인 MIL, HALCON, IPP의 함수는 내부적으로 앞 장에서 설명한 SIMD 방식을 이용하여 구현되어 있다. SSE 명령어를 사용하여 직접 구현한 방법과의 처리 속도 비교를 ms 단위로 표 1에 나타내었다.

표 1의 방법 중 Seq.는 일반적인 순차처리 방식으로 처리 된 결과이다. 일반적인 방법보다는 상용 영상처리 라이브러리 그리고 SSE로 직접 구현 한 방법이 훨씬 빠른 처리 속도를 보이는 것을 볼 수 있다. 또한 상용 라이브러리 중 IPP가 뛰어난 성능을 보임을 확인 할 수 있다. HALCON의 경우 팽창과 침식 연산에서 다른 라이브러리에 비해 빠른 성능을 보임을 확인 할 수 있는데 이 두 알고리즘은 머신 비전에서 검사 대상 영상에 빈번하게 쓰이는 알고리즘이다.

직접 구현 한 SSE 방법은 SIMD 연산에 적합하도록 최적화하였다[6]. 표 1을 통해 SIMD 방식을 SSE를 통하여 직접 구현하는 것은 목적 어플리케이션에 적합한 확장성을 얻을 수 있으며 처리 속도도 상용 라이브러리 보다 좋거나 비슷함을 확인 할 수 있다.

4.2 멀티 프로세서 병렬화 방법의 성능 비교

앞서 설명한 데이터 병렬화 방법은 멀티 프로세서 환경에서 각각의 프로세서로 할당되어 더 높은 성능을 얻을 수 있다. 본 실험은 일반적인 순차코드로 작성된 영상처리 알고리즘이 멀티 프로세서 환경에서 얼마나 높은 성능을 얻게 되는지 OpenMP를 이용한 방법과 GPU에서 동작하는 CUDA를 사용한 방법을 통해 비교해 본다. 표 2에 순차코드가 OpenMP와 CUDA

표 2. OpenMP와 CUDA의 성능 비교

Function \ Method	Seq.	OpenMP	CUDA 1.1
Mean	117.29	78.07	5.41
Dilation	70.40	36.40	3.69
Erosion	56.62	36.16	3.71
Sobel Horizontal	168.85	112.41	4.75

를 통해 얻게 되는 처리 속도를 ms 단위로 나타내었다.

표 2를 보면 듀얼 코어를 사용한 OpenMP의 처리 속도는 병렬화 과정에서 발생하는 오버헤드로 인해 2배에 미치지 못한다. CUDA를 이용한 영상처리 알고리즘의 속도는 오직 커널 함수만의 시간을 측정한 것으로 OpenMP에 비해 매우 빠른 속도를 보인다. 그러나 GPU로 메모리를 복사하는 과정과 처리 결과를 다시 CPU로 복사하는 과정이 필요하기 때문에 실제 구현 시에는 반드시 이 비용을 고려하여야 한다.

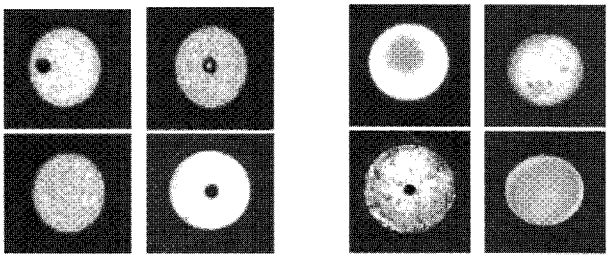
5. 최근의 병렬 영상처리 기술 적용 사례

본 절에서는 앞서 설명한 병렬 처리 기술을 실제 머신 비전 어플리케이션에 활용한 인하대학교와 포항공대의 연구 사례[9-12]를 소개한다.

5.1 GPU를 이용한 금속 패드 변색 분류 알고리즘[9]

본 연구에서는 AFVI (Automated Final Vision Inspection) 시스템에서 사용되는 금속 패드의 변색을 분류하는 알고리즘을 CUDA를 통해서 개발 하였다. 그림 12 (a)에 양품으로 판정되는 거칠기 영상과 (b)에 불량으로 판정되는 변색 영상을 나타내었다.

위와 같이 눈으로 구분하기 힘든 거칠기와 변색을 구분하기 위해서 Gabor 필터를 적용하여 텍스쳐 정보를 얻어낸다. Gabor 필터는 조명의 영향을 적게 받고 회전에 강인한 특징이 있지만 계산량이 많은 단점이 있어 이를 해결하고자 GPU의 높은 처리 속도를 이용한 것이다. 텍스쳐 정보를 추출하기 위한 처리 시간은 그림 13과 같다. OpenMP를 통해 듀얼 코어에서 처리 된 시간을 함께 표시해 두었으며 GPU의 전역(global) 메모리만을 사용했을 때와 공유(shared) 메모리를 동시에 사용 했을 때의 결과를 함께 표시하였다.



(a) (b)

그림 12. 거칠기 영상 (a)과 변색 영상 (b)

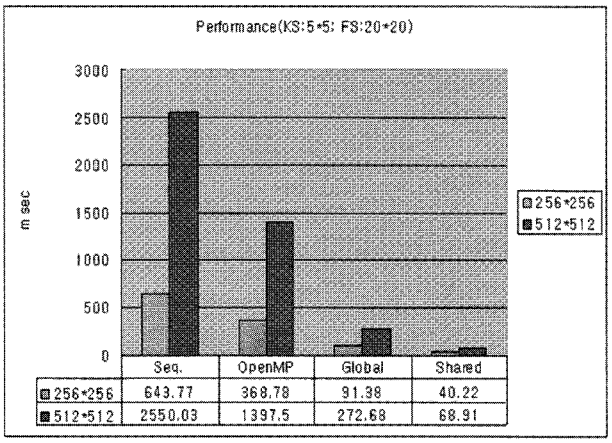


그림 13. 텍스쳐 정보 추출에 걸린 시간

본 연구에서는 제안하는 방법을 20장의 실험 영상에 적용하였으며 그 결과 정확하게 모두 변색을 분류해 냈음을 증명하였다.

5.2 고속 SIFT 방법을 이용한 PCB 영상의 정렬 알고리즘[10]

본 연구에서는 물체 인식 방법에 널리 사용되는 SIFT (Scale-Invariant Feature Transform)을 사용하여 PCB 영상을 정렬하는데 이용한다. 단순 SIFT를 영상의 PCB 영상의 정렬에 사용하게 될 경우 템플릿으로 사용되는 영상과 카메라로부터 얻어진 목표 영상간의 매칭되는 점이 다르게 나타나는 경우가 있기 때문에 그림 14와 같은 방법을 제안하였다.

제안하는 방법은 잡음에 대하여 강인한 성능을 보였다. 또한 기존의 템플릿의 상관관계를 이용한 매칭의 경우 회전 정보를 확인하기 위하여 많은 수행시간이 요구 되지만, 제안된 방법은 테스트 영상의 중심점과 회전량을 짧은 시간에 추출할 수 있었다. 이 방법은 현재 [11]에 제시 된 방법으로 CPU와 GPU를 이용

하여 가속화되고 있으며 더 높은 정확도를 위한 연구가 진행 중이다.

5.3 CUDA를 활용한 TFT-LCD 검사 장비[12]

본 연구에서는 TFT-LCD 표면의 불량을 검출하는데 GPU를 사용하였다. 라인 스캔 카메라를 사용 하여 얻어진 TFT-LCD 이미지는 대용량이기 때문에 생산공정에 적용하기 위해서는 빠른 영상 처리 시간을 요구한다. TFT-LCD 영상은 그림 15와 같이 반복된 패턴을 갖고 있기 때문에 인접 패턴과의 영상차를 이용하여 쉽게 불량을 검출해 낼 수 있다.

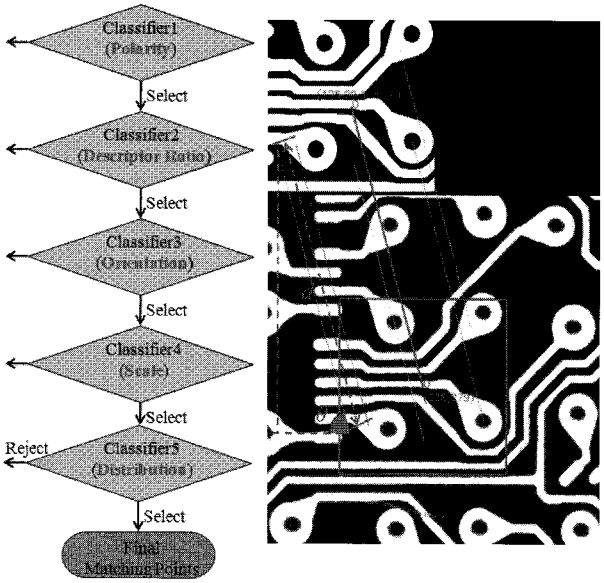


그림 14. 제안하는 SIFT 기반의 PCB 영상 정렬 방법

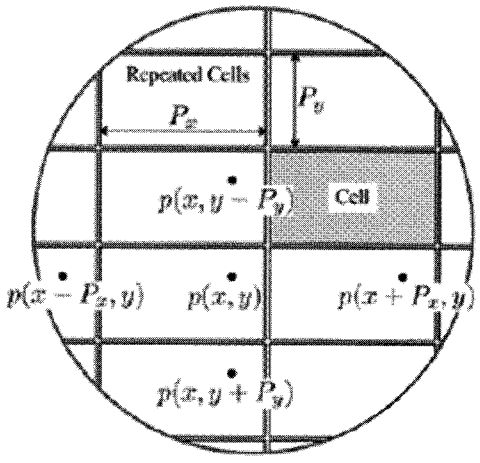


그림 15. TFT-LCD 검출에 사용하는 변수들

실험에 사용된 GPU는 Geforce 8800 ultra와 Quadro FX 5600이고 CUDA를 사용하여 구현되었다. 실험결과는 인텔의 듀얼 코어에서 사용된 검사 결과와 처리 속도를 비교 하였으며 그 결과는 그림 16과 같고 검출된 불량은 그림 17에 나타내었다. 그림 16을 보면 알 수 있듯이 CUDA를 활용한 방법은 CPU의 듀얼 코어를 사용한 방법보다 대략 10배 정도 빠른 성능을 보이는 것을 확인 할 수 있다. 그림 17 (b)의 검은색으로 나타난 부분은 TFT-LCD의 불량영역을 나타낸다.

6. 결론

본 고에서는 병렬 영상처리를 이용한 고속 머신 비전 기술의 동향에 관해 다루었다. 머신 비전에서 대표적으로 사용되는 고속 상용 라이브러리에 대해 알아보았고 병렬성을 효율적으로 달성하기 위하여 현재 활발히 연구되고 있는 SSE, OpenMP, CUDA에 관하여 설명하였다. 이러한 병렬 처리기술을 영상처리 알고리즘에 직접 적용하여 수행한 처리 시간을 고속 영상처리 라이브러리의 처리 속도와 비교한 결과 현재의 병렬처리 기술을 효과적으로 이용하면 고속 상용 라이브러리의 이용 없이

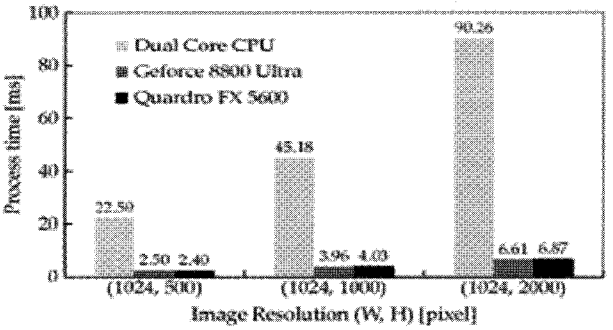


그림 16. CPU와 GPU에서의 처리 속도 비교

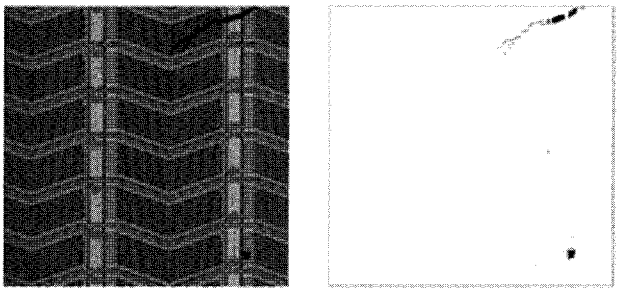


그림 17. TFT-LCD 이미지(a) 와 검출된 불량 영역(b)

도 높은 처리 속도를 얻을 수 있음을 확인 할 수 있었다. 그리고 이러한 병렬 영상처리 방법을 머신 비전에 적용한 연구 사례를 국내 대학 연구실에서의 연구결과를 통해서 알아 보았다.

현재 머신 비전에서 가장 활발히 연구되고 있는 병렬 영상 처리 기술은 GPGPU 기술이다. 머신 비전에서 사용되는 영상 데이터는 점차 대용량화 되어 가고 있어 이를 효율적이고 빠르게 처리하기 위한 GPGPU 기술은 한동안 지속적으로 발전할 것으로 보인다.

참고문헌

[1] Matrox, "Matrox Image Library 8.0 user guide", June 2005.

[2] http://www. Matrox.com

[3] MVTec Software GmbH, "HALCON/HDevelop Reference Manual", May 2007.

[4] MVTec Software GmbH, "HALCON - QuickGuide", June 2007.

[5] Intel, "Intel Integrated Performance Primitives for Windows OS on Intel64 architecture", March 2009.

[6] 박은수, 최학남, 김준철, 임유청, 김학일, "SSE 명령어를 이용한 영상의 고속 전처리 알고리즘", 전자공학회 논문지, vol. 4, no. 2, pp. 65-77, March 2009.

[7] 한국과학 기술 정보원, "OpenMP를 이용한 병렬 프로그래밍", 2002.

[8] NVidia, "NVIDIA CUDA Programming Guide", July 2009.

[9] 최학남, 박은수, 김준철, 김학일, "GPU를 이용한 Gabor Texture 특징점 기반의 금속 패드 변색 분류 알고리즘", 제어 · 로봇 · 시스템학회 논문지, vol. 15, no. 8, pp.778~785, June 2009.

[10] 김준철, 홍성욱, 최학남, 김학일, "SIFT를 기반으로 하는 PCB 영상의 정렬 알고리즘 개발", 대한전자 공학회 하계 학술대회, vol. 31, no. 1, July 2009.

[11] 김준철, 정용한, 박은수, 최학남, 김학일, 허욱렬, "CPU와 GPU의 병렬 처리를 이용한 고속 물체 인식 알고리즘 구현", 제어 · 로봇 · 시스템학회 논문지, vol. 11, no. 5, pp. 487-494, May 2009.

[12] C. H. Lee, C. Jeong, M.-S. Jang, and P.G. Park, "Implementation of TFT Inspection System using the Common Unified Device Architecture (CUDA) on Modern Graphics Hardware", *Proc of the 10th Control, Automation, Robotics and Vision*, pp.1899-1902, Dec 2008.

저자약력

박 은 수

· 2007년 인하대학교 정보통신공학부 학사졸업.
· 2007년~현재 인하대학교 정보공학과 석사과정.
· 관심분야 : 컴퓨터비전, 머신비전, 병렬 영상처리.

최 학 남

· 2004년 연변대학 응용수학과 졸업.
· 2007년 상명대학교 컴퓨터과학과 졸업.
· 2007년~현재 인하대학교 정보공학과 박사과정.
· 관심분야 : 의료영상처리, 머신비전, 패턴인식, 병렬 영상처리.

김 준 철

· 2008년 인하대학교 정보통신공학부 학사 졸업.
· 2008년~현재 인하대학교 정보공학과 석사과정.
· 관심분야 : 컴퓨터 비전, 로봇비전, 병렬 영상처리.

정 용 한

· 2008년 인하대학교 정보통신공학부 학사 졸업.
· 2008년~현재 인하대학교 정보공학과 석사과정.
· 관심분야 : 컴퓨터 비전, 로봇비전, 병렬 영상처리.

김 학 일

· 1983년 서울대학교 제어계측공학과 학사 졸업.
· 1985년 (미) 퍼듀대학교 전기컴퓨터공학과 석사 졸업.
· 1990년 (미) 퍼듀대학교 전기 컴퓨터공학과 박사 졸업.
· 1990년 9월~현재 인하대학교 공과대학 교수.
· 2001년 2월~현재 한국생체인식포럼 부의장.
· 2002년 1월~현재 한국정보보호학회 바이오인증연구회위원장.
· 2003년 3월~현재 ISO/ IEC JTC1/SC37 (생체인식), WG5(성능평가) Rapporteur Group.
· 2005년 4월~현재 ITU-T SG17 Q.9 (Telebiometircs) Rapporteur.
· 관심분야 : 바이오인식, 컴퓨터비전, 패턴인식 및 병렬 영상처리.