특집

영상처리와 컴퓨터비젼을 위한 GPGPU 활용

박인규·이만희 (인하대학교) Nitin Singhal (삼성전자)

I. 서 론

최근 GPU(Graphics Processing Unit)의 연산 성능은 눈부시게 발전해왔다. 연산속도의 급격한 증가와 더불어 GPU에서 쉽게 사용할 수 있는 고수준의 언어는 기존의 3차원 그래픽스뿐만 아니라 GPGPU(General Purpose GPU)라고 불리는 일반적인 용도로의 GPU 사용을 가능하게 하였다[1,2]. 뿐만 아니라 NVIDIA의 CUDA (Compute Unified Device Architecture)[3]와 같은 통합 컴퓨팅 프레임워크는 스레드 (thread) 레벨에서 사용자가 제어할 수 있는 병렬처리 방법을 제공하였고 전역 메모리 (global memory) 전체 공간에 대한 읽기와 쓰기를 자유롭게 사용할 수 있게 하였다. 또한 on-chip 공유 메모리 (shared memory)를 장착하여 캐쉬 사용과 유사한 방식의 고속 자료 접근과 스레드 사이의 효율적인 자료 교환이 가능하다. 따라서 기존의 shading 언어 기반의 GPGPU에 비해 프로그래밍의 자유도가 획기적으로 높아져 대부분의 병렬 알고리즘을 CUDA 기반으로 구현할 수 있다.

NVIDIA의 Cg나 DirectX의 HLSL(High Level Shading Language), 또는 OpenGL의 GLSL (OpenGL shading language)를 사용한 shader 기반의 GPGPU 활용은[2]에서 집중적으로 기술되었다. 최근 영상처리나 컴퓨터 비젼 분야의 GPGPU 관련 연구논문은 특정한 알고리즘을 병렬 기반의 GPU 구조에서 구현하는 것, 즉 매핑 (mapping)에 초점이 맞춰져있다[4]. 그러나 조직적인 분석과 GPGPU를 좀 더 일반적으로 사용하기 위한 길잡이는 아직 많이 부족한 현실이다. Ryoo[5]는 G80 GPU에서 최적화 하여 구현할 수 있는 몇몇의 유용한 이론들에 대하여 언급하였다.

GPGPU의 목적은 대용량 데이터 처리나 복잡한 알고리즘의 처리에서 CPU를 능가하여 눈부신 속도 향상을 얻는 것이다. 일반적으로 영상처리 및 컴퓨터비젼 알고리즘이 일반적으로 GPU의 대용량 병렬처리 구조에 적합하지만, 특정한 알고리즘의 고유한 특징과 GPU의 환경 사이에서 협력이 제대로 이루어 지지 않을 수 있기 때문에 기대한 만큼의 성능 향상을 얻지 못하는 경우도 많이 존재한다. 따라서 많은 시간이 소요되는 알고리즘의 구현 이전에 알고리즘 환경의 병렬화 가능성을 정확히 평가하는 것이 매우 중요하다.

본 기고에서는, 많은 코어를 갖는 대용량 병렬 GPU에서 비주얼 컴퓨팅 알고리즘을 디자인하고 구현하기 위한 몇몇 주요한 요소들에 대하여 조사한다. 본 기고에서는 몇몇 의미 있는 기준들을 정의하고 각각의 알고리즘이 정량적으로 측정될 수 있도록 분석한다. 컴퓨팅 플랫폼으로는 shading 언어에 비해 보다 다양한 알고리즘을 구현할 수 있는 CUDA를 선택하였다. 이러한 작업을 위하여 네 개의 주요한 분야를 선택하고 각각의 분야를 대표할만한 알고리즘들을 선정하였다. 각각의 알고리즘들은 모두 CUDA를 이용하여 GPU 상에서 구현되었고 일반적인 예측과 지침을 이끌어내기 위하여 측정된 기준들과 얻어진 결과들 사이의 관계를 집중적으로 분석한다.

II. GPU 상에서 병렬 구현을 기술하기 위한 기준

일반적으로 영상처리 및 컴퓨터비젼에서는 각각의 화소들이나 영상의 특징들 사이에서 독립적인 처리가 이루어지게 되고 이러한 과정은 GPU에서 SIMD 방식의 병렬 연산 방법을 이용하여 매우 효율적으로 구현될 수 있다. 이와 같은 경우 각각의 특정한 알고리즘의 병렬화 가능성을 기술하기 위한 의미 있는 기준을 설정할 수 있고 이러한 기준을 지속적으로 향상시키는 방향으로 병렬 알고리즘 설계와 최적화를 수행할 수 있다.

1. 병렬처리 속도 향상의 상한 (upper bound)

Amdahl's 법칙[6]은 병렬 처리에 의한 속도 향상의 상한선을 이론적으로 예측할 수 있다. 이에 대한 수식은 다음과 같다.

$$S \leq \frac{1}{f + \frac{1-f}{N}} \tag{1}$$

이 때, f는 전체 CPU 코드중 병렬화 후에도 CPU에서 처리해야 할 부분의 수행 시간의 비율을 의미하고 N은 프로세서의 개수를 의미한다.

2. 분기 다양성 (branching diversity)

NVIDIA의 CUDA 환경(G80 GPU 기준)에서 warp은 명령을 동시에 수행하고자 하는 32개의 스레드의 집합을 의미하고, 그 32개의 스레드가 동시에 정확히 같은 명령을 실행할 때 최대 효율을 얻을 수 있다. 그러나 if, for문 등의 흐름 제어 명령은 동일한 warp에 있는 스레드를 분기시킬 수 있으며 이 때 분기된 부분은 직렬 처리를 통해 수행하여야 하므로 병렬화의 효율에 좋지 않은 영향을 미친다. 따라서 동일한 warp에 있는 스레드 사이의 로드 밸런스의 관점에서 제어 흐름 명령에 의한 분기를 많고 적음을 하나의 기준으로 선정하였다. 우선 디바이스 에뮬레이션 모드 또는 CPU의 for 문 단위로 (주로 for 문의 내부가 병렬화 되므로) 하나의 warp에서 각각의 스레드의 수행 시간을 측정한 후 표준편차를 계산하였다. 이 때 표준편차의 값이 클수록 불균형이 심한 것을 의미한다.

3. 부동 소수점 연산의 수와 전역 메모리 접근 회수의 비율

CUDA 환경에서 멀티프로세서는 높은 메모리 지연(400~600 clock cycles)을 갖고 있다. 이

것은 전역 메모리에서의 읽기와 쓰기 동작에서 매우 오랜 시간을 소비한다는 것을 의미하고 이러한 부분은 전체 실행시간에 있어서 bottleneck으로 존재하게 된다. 만약 스레드 내부에서 부동소수점 연산이 충분히 독립적이라면 이와 같은 메모리 지연 문제는 숨겨질 수 있다. 따라서 부동소수점 연산의 비중을 연산의 수와 메모리 접근 사이의 비율로 추정할 수 있고 이 값이 클수록 GPU의 활용도가 높은 것을 의미한다.

4. 공유 메모리 접근 회수와 전역 메모리 접근 회수의 비율

CUDA 환경에서 on-chip 공유 메모리의 접근은 지역 메모리나 전역 메모리공간의 접근에 비하여 매우 빠르다. 하나의 warp에 존재하는 스레드에서는 레지스터에 접근하는 것만큼 빠르게 공유 메모리에 접근할 수 있다. 따라서 CUDA 커널 안에서 최대한으로 공유 메모리를 사용할 때 더 높은 성능을 기대할 수 있다. 벤치마크 도구를 이용하여 공유 메모리 접근과 전역 메모리 접근 비율을 계산할 수 있고 이것을 이용하여 메모리 접근의 효율을 정의할 수 있다.

5. GPU 점유율 (GPU occupancy)

동시에 수행할 수 있는 최대의 warp 개수와 현재 활성화 되어있는 warp 개수의 비율을 이용하여 멀티프로세서의 점유율을 계산할 수 있다. 이러한 점유율은 각각의 CUDA 스레드 블록 내에서 공유 메모리의 사용량과 레지스터의 사용량에 의해서 결정된다. 위의 정보는 nvcc(CUDA C compiler)의 특정한 옵션을 이용하여 추출할 수 있다.

6. 병렬 작업 의존도 (parallel task dependency)

전형적인 CUDA 프로그램의 경우 메모리 접근을 통하여 각각의 서로 다른 스레드에서 데이터를 공유하고자 할 경우 (즉, 스레드간 통신이 필요한 경우) 동기화 (synchronization) 명령이 필요로 하게 되고 이 때 스레드 블록의 병렬화는 깨지게 된다. 따라서 스레드 내부에서의 동기화 명령의 빈도를 이용하여 병렬 작업 의존도를 추측할 수 있다.

7. 소스 코드의 분량

CUDA는 커널 코드를 PTX(parallel thread execution)로 컴파일하고 이것은 GPU에서 수행되기 위하여 바이너리 코드로 다시 컴파일 된다. CUDA 컴파일러의 복잡도는 PTX의 연산수와 비례하게 되고 이것은 원본 코드의 라인 수에 의존적이다. 따라서 커널 코드의 라인수를 이용하여 코드의 복잡도를 계산할 수 있다.

III. 시험 알고리즘과 GPU상에서의 구현

1. 알고리즘 선정의 기준

영상처리 및 컴퓨터비젼은 각각의 분야에서 넓은 범위를 갖고 있다. 본 기고에서는 전반적인 연구 관심도를 기반으로 네 개의 주요한 영역(3차원 물체 복원, 특징 추출, 물체 인식, 영상 압축)을 선정하였다. 각각의 영역에서 특정한 알고리즘(다시점 스테레오 매칭, 선형 특징 추출, 물

〈표 1〉 GPU 상에서 설계되고 구현된 병렬 알고리즘들의 특징

	처리 영역	병렬 작업의 분량	병렬 스레드 간의 분기 다양성	부동 소수점 연산의 집중도	기타 특징
다시점 영상 스테레오 매칭	화소	많음	적음	낮음	-
선형 특징 추출	화소 / 특징	중간	적음(에지 검출) 많음(Linking, Fitting)	높음	불필요한 연산이 존재
물체 자세 추정	화소	많음	적음	낮음	3차원 범위 데이터 처리
JPEG2000 압축	화소 / bitplane	많음(DWT) 적음(EBCOT)	많음	높음(DWT) 높음(EBCOT)	-

체 자세 추정, JPEG2000 영상 압축)을 선택하여 GPU상에서 구현하고 분석하였다. 선택한 알고리즘들 사이의 고유한 구조와 특징들은 완전히 다르므로 GPU 상에서의 몇몇 주요한 디자인과 구현 논점으로 이용될 수 있다. <표 1>에 선택된 알고리즘들의 특징을 정리하였다.

2. Multiview Stereo Matching (MVS)

MVS에서 가장 연산량이 많은 부분은 입력 영상 사이에서 일어나는 많은 수의 지역 윈도우 매칭 부분이다. 각각의 윈도우 매칭은 독립적으로 수행되기 때문에 GPU의 활용성이 매우 크다.

각각의 입력 영상에서 올바른 화소의 깊이를 알아내기 위하여 bounding box로 표현된 관심 볼륨 (volume of interest) 내부에 존재하는 시점 방향의 3차원 후보 점들이 주변 영상들로 각각 투영 된다. 투영된 위치 주위의 지역적인 윈도우들은 NCC(normalized cross correlation)을 이용하여 기준 영상의 윈도우와 매칭 과정을 거치게 된다. 총 연산 복잡도는 $O(N^2WHL)$이고 이 때 N은 입력 영상의 수, W와 H는 영상의 가로와 세로 길이, 그리고 L은 정육면체 형태의 bounding box의 각 모서리의 길이를 의미한다. GPU상에서의 구현에서는 $W\times H$ 만큼의 스레드를 발생시켜 각각의 스레드에서 하나의 화소에 대한 깊이를 계산한다. 하나의 기준 영상에 대한 깊이 영상을 얻기 위하여 한 번의 커널 함수 호출이 일어나고 이 때 커널의 계산 복잡도는 $O(NL)$이 된다. 따라서 병렬 처리를 이용하여 모든 영상에 대한 깊이 영상을 계산하는 과정의 전체 복잡도는 커널을 N번 호출하는 것으로 생각할 수 있고 이를 고려하면 결국 $O(N^2L)$로 줄어들게 된다.

3. Linear Feature Extraction

선형 특징 추출을 위하여 우선 Canny 에지 (edge) 검출을 수행한 후 에지 세선화 (thinning) 과정을 거치게 된다. 그 후 각각의 에지 화소들을 3×3 크기의 주변 화소들을 이용하여 세 종류로 분류한 뒤 연결되어 있는 화소를 기준으로 좌우 방향으로 탐색을 수행하여 현재 화소가 포함되어 있는 에지의 시작과 끝 지점을 찾는다. 이러한 과정은 모든 화소에 대하여 스레드를 발생한 뒤 현재 화소가 에지 화소일 경우에만 동작을 하기 때문에 branching diversity가 증가하게 되고 같은 에지에 존재하는 화소들의 경우 동일한 과정을 반복적으로 수행하게 되어 불필요한 연산이 발생하게 된다.

에지의 연결 정보를 얻은 후, 각각의 에지에 대하여 반복적으로 line fitting을 수행한다. 우선

하나의 chain에서 시작점과 끝점을 연결한 직선을 생성하고 각각의 에지 화소들과 line segment 사이의 거리를 계산한다. 이 때 구해진 거리가 임계값 이상이 될 경우 최대의 거리를 갖는 화소에서 해당 에지를 두 개의 chain으로 나누고 위의 과정을 오차의 최대값이 임계값보다 작을 때까지 반복적으로 수행한다.

4. Object Pose Estimation

3차원 물체의 자세 추정을 위하여 지역적인 descriptor 기반의 방법 대신 GPU에서의 병렬 연산에 유용한 깊이 영상 기반의 알고리즘[7]을 사용하였다. 기존의 알고리즘의 경우 GLSL로 작성되었기 때문에 CUDA 플랫폼으로의 변환을 시도하였다.

위 알고리즘의 가장 중요한 연산 부분인 기준 영상과 입력 영상 사이의 화소당(per-pixel) 오차 계산을 GPU를 이용하여 수행하였다. 각각의 최적화 과정에서 연산 복잡도는 $O(NWH)$가 되고 이 때 N은 기준 자세의 개수를 의미하고 W와 H는 기준 영상의 가로와 세로 길이를 의미한다. 일반적으로 $(N, W, H)=(1024,64,64)$를 사용하였으며 4백만 개 이상의 병렬 스레드를 생성하여 연산을 수행하였다.

5. JPEG2000 Encoding

본 절에서는 JPEG2000의 DWT와 EBCOT Tier-1 부분을 CUDA 프로그래밍 모델을 이용하여 GPU로 구현하는 과정에 대하여 설명한다.

가. Parallel DWT

JPEG2000 영상 압축을 병렬화 하기 위하여 FBS(filter bank scheme)를 채용하였다. 2차원 DWT는 각각의 차원에서 1차원 변환으로 구분하여 적용될 수 있다. 수직방향의 1차원 DWT를 생각하였을 경우 열의 길이를 384개의 스레드로 할당하고 각각의 스레드는 실제 연산 이전에 우선적으로 텍스처 메모리에서 공유 메모리로 열 데이터를 가져온다. 각각의 출력 화소에서 화소의 위치를 이용하여 필터의 종류(로우 패스와 하이 패스)를 선택하고 현재의 출력 화소 위치는 중심의 기준 화소와 $\pm\lceil\frac{N_0}{2}\rceil$ 범위의 주변 픽셀들을 이용하여 공유 메모리 공간상의 입력 기준 위치 β로 매핑된다. 이 때 N_0는 필터 커널의 길이를 의미하고 기준 위치 β는 입력 신호의 길이 l에 의하여 식 2와 같이 정의된다.

$$\beta=\begin{cases}2n & \text{if } n<\frac{l}{2}\\ 2\left(n-\frac{l}{2}\right)-1 & \text{otherwise}\end{cases} \quad (2)$$

나. Parallel EBCOT Tier-1

JPEG2000의 EBCOT Tier-1 단계는 context 모델링과 연산 encoder의 두 개의 주요한 부분으로 구성된다. 본질적으로 위의 두 단계는 매우 반복적이고 순차적으로 진행된다. 그러나 코드블록 사이에 동기화 부분이 필요하지 않기 때문에 각각을 독립적인 코드블록으로 병렬화 하여 EBCOT Tier-1 단계를 CUDA 구조로 변환하였다. 각각의 코드블록은 CUDA 커널 내부에서 분리된 스레드의 입력으로 할당되었다.

Ⅳ. 실험 결과

모든 테스트 알고리즘은 우선 CPU 코드로 구

〈표 2〉

		이상적인 속도 향상	분기 다양성	FP 연산 vs GM 접근	SM vs GM 접근 비율	GPU 점유율	작업 의존도	소스 코드 길이	커널 코드 길이
MVS	TempleRing	15.23	0.8463	3.19	0.02	33%	2	282	159
선형특징 추출	Lena	15.23	0.00199	7.57	0.09	41.65%	36	738	556
	Baboon			7.46	0.09				
	Pepper			7.57	0.09				
	Airplane			7.55	0.09				
물체 자세 추정	NewElbow	N/A	0.0011	1.69	0	99.57%	6	1251	1010
JPEG2000 압축	Lena	3.27	0.0008	26.89	5.00	50% (DWT) 17% (EBCOT)	4 (DWT) 2 (EBCOT)	192 (DWT) 703 (EBCOT)	82 (DWT) 400 (EBCOT)
			14.5028	6.16	1.92				
	Baboon		0.0008	26.89	5.00				
			18.4455	6.09	1.89				
	Pepper		0.0008	26.89	5.00				
			15.4883	6.22	1.75				
	Airplane		0.0008	26.89	5.00				
			22.0353	6.19	2.05				

현되었고 CUDA를 이용하여 모두 병렬화 하였다. 이 때, CPU 코드는 최적화되지 않았지만 전반적인 성능 비교에는 문제가 없다고 볼 수 있다. 모든 실험은 Intel CPU(42.56 GFLOPS의 Q9450)와 128개의 코어를 갖는 NVIDIA G92 (GeForce 9800 GTX, 512MB) 그래픽카드를 이용하여 수행되었다.

우선 목표 알고리즘에 대하여 제안하는 기준들을 평가하였고 <표 2>는 그 결과들을 보여주고 있다. 이 때 별도의 디버깅 코드를 작성하여 부동 소수점 연산의 개수와 공유 메모리와 전역 메모리의 접근 횟수를 계산하였다. <표 2>에서 보는바와 같이 각각의 알고리즘들은 2장에서 정의한 기준들에 대하여 각기 다른 병렬 구현 특징들을 갖고 있는 것을 확인할 수 있다. Amdahl's 법칙을 이용하여 계산된 속도 향상의 경우 JPEG2000 압축을 제외하고 모두 15.23(무한대 대신 GPU와 CPU의 이론적인 최대 GFLOPS 비율을 제시함)의 값을 가져 100% 병렬화 되었음을 알 수 있다. 또한, JPEG2000의 경우 DWT와 EBCOT Tier-1 단계는 병렬화가 가능하였지만 bitstream I/O와 EBCOT Tier-2 과정은 여전히 CPU에서 순차적으로 진행되기 때문에 CPU와 상대적인 속도 향상이 감소하게 되었다.

<표 2>의 다른 정보들은 부동 소수점 연산의 비율과 메모리 지연을 표현하는 전역 메모리의 접근 비율을 나타낸다. 공유 메모리와 전역 메모리 접근 비율은 CUDA 커널에서의 공유 메모리 사용량의 범위를 결정한다. DWT와 EBCOT Tier-1의 경우 높은 공유 메모리 접근을 보여준다. GPU의 점유율 기준을 위하여 NVIDIA의 점유율 계산 형식을 이용하였고 EBCOT Tier-1의 경우를 제외하고 대부분 33%~99%의 활성 점유율을 보여줬다. EBCOT Tier-1의 경우 스레드의 개수가 코드블록의 개수에 의하여 제한되고 이것은 GPU의 최대 스레드의 개수보다 현저하게 작다. <표 2>에서 보는바와 같이 선형 특징 추출 알고리즘의 경우 높은 작업 의존도를 갖고 있고 이것은 해당 알고리즘을 병렬화 하는 것에 많은 어려움이 있는 것을 의미한다. 또한 선형 특

〈표 3〉

알고리즘	자료	수행 시간			CPU 대비 속도 향상		GPU Scalability
		CPU (Q9450)	GPU (G92)	GPU (GX200)	Q9450/G92	Q9450/GX200	G92/GX200
MVS	TempleRing (47 영상)	39,200	340	110	115.29x	356.36x	3.09x
선형 특징 추출	1280×1024	610	250.42	223.95	2.43x	2.72x	1.11x
	1200×1800	1,250	471.88	387.73	2.65x	3.22x	1.22x
	2288×1712	2,375	1018.98	789.16	2.33x	3.00x	1.29x
물체 자세 추정	NewElbow1024	49,450	1,700	430	29.08x	115x	3.95x
JPEG2000 압축	2288×1712 DWT	1,469	152	94	9.66x	15.63x	1.62x
	2288×1712 Tier-1	2,859	1,468	1,016	1.94x	2.81x	1.44x
	3024×2089 DWT	2,391	243	156	9.84x	15.33x	1.56x
	3024×2089 Tier1	4,109	2,062	1,531	1.99x	2.63x	1.34x

징 추출과 EBCOT Tier-1의 경우 매우 큰 커널 크기를 갖는 것을 알 수 있고 이것 역시 해당 알고리즘이 병렬화 되기 어렵다는 것을 의미한다.

정확히 측정된 기준들은 각각의 알고리즘과 GPU를 활용한 병렬화 사이의 협력 관계를 반영한다. 이러한 관계들은 <표 3>에서 보는바와 같이 실행시간 분석으로 실증될 수 있다. 각각 알고리즘의 수행속도 향상은 협력 관계에 따라 두 배(JPEG2000 Tier-1)에서 수백 배(다시점 스테레오 매칭)까지 다양하다. 이 때 GPU의 수행 시간은 커널의 수행 시간뿐만 아니라 메모리의 이동까지 모두 포함하고 있다. CPU 상에서의 실행은 최적화된 CPU 코드를 사용하지 않았으나, GPU 상에서의 구현에 의한 전반적인 수행 속도 향상을 비교하는 데 큰 문제는 없을 것으로 생각된다.

V. 영상처리와 컴퓨터비젼 분야의 응용을 위한 GPGPU 지침

본 기고에서는 측정 가능한 다양한 성능 특성들을 제시하고, 이를 이용하여 응용 프로그램의 GPU 구현의 효율성을 판단할 수 있는 객관적인 기준을 제시하였다. 본 장에서는 병렬 프로세서를 기반으로 이러한 응용 프로그램을 개발하는 과정에서, 일반적이지만 반드시 고려해야하는 유용한 지침들을 정리하였다.

1. GPU 컴퓨팅 플랫폼의 명확한 이해

구현하는 알고리즘에 대하여 성능 향상을 얻기 위하여 개발자들은 프로그래밍 모델뿐만 아니라 하드웨어 내부 구조의 기본 특징들에 대하여 반드시 이해하여야 한다. 대용량 멀티 스레딩을 통한 자료 처리에 초점을 맞출 경우 중요한 일 중의 하나가 될 것이다. 또 다른 중요한 관점은 일관적인 흐름 제어를 필요로 하는 warp의 구성이다. 하나의 멀티 프로세서가 제공하는 공유 메모리와 레지스터의 용량이 한정되어 있고 이는 그 멀티프로세서에서 실행되는 스레드들에서 공유되기 때문에 이러한 자원들을 효율적으

로 사용하여야 한다. GPU 점유율 은 병렬화된 작업을 최대로 수행할 수 있는 스레드 블록의 개수와 스레드당 사용하는 레지스터의 크기를 효율적으로 결정할 수 있도록 도와줄 것이다.

2. 적은 작업 의존도의 효율적인 매핑

개발자들은 CUDA 기반의 스레드 모델을 활용하기 위하여 효율적인 알고리즘-GPU 병렬화 매핑 방법을 반드시 찾아야 한다. 다시점 스테레오 매칭이나 JPEG2000의 압축과 같은 경우 매우 적은 작업 의존도를 요구한다. 그러나 물체 자세 추정이나 선형 특징 추출과 같이 좀 더 복잡하고 불규칙한 알고리즘의 경우 효율적으로 매핑하기 어렵다. 높은 단계의 데이터 병렬화를 제시하는 응용프로그램일 경우에도 매핑이 어려워질 가능성이 커진다.

3. GPU 메모리 체계의 효율적인 사용

개발자들은 반드시 GPU 메모리 체계의 장점을 최대한으로 이용하여야 한다. 본 기고에서는 부동 소수점 연산 대비 전역 메모리 접근 비율과 공유 메모리 접근 대비 전역 메모리 접근 비율의 두 가지 중요한 기준에 대하여 정의하였다. 4장에서 확인할 수 있듯이 이 두 가지의 기준에서 높은 점수를 얻을 경우 높은 성능향상을 나타내는 것을 알 수 있다.

4. 분기 다양성의 최소화

본 기고에서 정의한 분기 다양성은 개발자들이 현재 알고리즘을 GPU로의 이식이 가능한지 결정하는 과정에 도움을 줄 수 있는 도구로써 제공될 수 있다. 높은 분기 다양성은 속도 향상의 병렬화 성능에 매우 크게 영향을 끼치게 되고 이것은 반드시 피해야 한다.

VI. 결 론

본 기고에서는 GPU에서 영상처리 및 컴퓨터 비젼 알고리즘을 수행하기 위한 계획과 구현 쟁점에 대하여 알아보았다. 주요 응용 분야의 네 개의 대표적인 알고리즘을 선택하여 CPU와 GPU 상에서 구현하였고 각각의 알고리즘을 병렬화 하는데 특징을 구분하기 위하여 몇 가지 기준을 설정하였다. 실험 결과의 분석을 통하여 영상처리나 컴퓨터 비젼 분야에서 GPU를 사용하기 위한 일반적인 지침을 제시하였고 이것들은 다른 연구자들에게 참고 될 수 있을 것이다.

참고문헌

[1] General Purpose GPU Programming (GPGPU) Website, http://www.gpgpu.org.

[2] J. Owens et al., "A survey of general-purpose computation on graphics hardware," Computer Graphics Forum, Vol.26, No.1, pp.80-113, March, 2007.

[3] NVIDIA Corporation, Compute Unified Device Architecture (CUDA), http://developer.nvidia.com/object/cuda.html

[4] J.-M. Frahm, M. Pollefeys, and M. Shah (Co-Chairs), Proc. of CVPR Workshop on Visual Computer Vision on GPU's, June, 2008.

[5] S. Ryoo et al., "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," Proc. of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp.73-82, February, 2008.

[6] A. Grama, A. Gupta, G. Karypis, and V. Kumar, Introduction to Parallel Computing, 2nd Edition, Pearson Education LTD., 2003.

[7] M. Germann, M. D. Breitenstein, I. K. Park, and H. Pfister, "Automatic pose estimation for range images on the gpu," Proc. of International Conference on 3-D Digital Imaging and Modeling, pp.81-88, August, 2007.

[8] Information Technology - JPEG2000 Image Coding System, ISO/IEC International Standard 15444-1 and ITU Recommendation T.80, 2000.

[9] S. Mallat, "The theory for multiresolution signal decomposition: The Wavelet representation," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.11, No.7, pp.674-693, July, 1989.

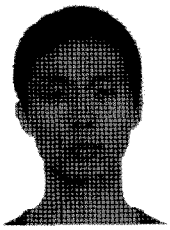
[10] D. Taubman, "High performance scalable image compression with EBCOT," IEEE Transactions on Image Processing, Vol.9, No.7, pp.1158-1170, July, 2000.

저자소개

박 인 규

1995년 2월 서울대학교 제어계측공학과 공학사
1997년 2월 서울대학교 제어계측공학과 공학석사
2001년 8월 서울대학교 전기컴퓨터공학부 공학박사
2001년 9월 ~ 2004년 3월 삼성종합기술원 멀티미디어 랩 전문연구원
2007년 1월 ~ 2008년 2월 Mitsubishi Electric Research Laboratories(MERL) 방문연구원
2004년 3월 ~ 현재 인하대학교 정보통신공학부 조교수

주관심 분야 : 컴퓨터 그래픽스 및 비젼 (영상기반 3차원 형상 모델링 및 렌더링, computational photography), GPGPU

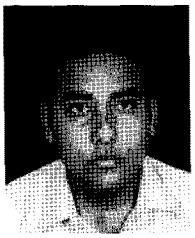


이 만 희

2006년 2월 인하대학교 컴퓨터공학과 공학사
2008년 8월 인하대학교 정보공학과 공학석사
2008년 9월 ~ 현재 인하대학교 정보공학과 박사과정
2007년 4월 ~ 2008년 2월 한국전자통신연구원(ETRI) 위촉연구원

주관심 분야 : GPGPU, 영상기반 모델링 및 렌더링

저자소개



Nitin Singhal

2006년 5월 인도 Indian Institute of Technology Guwahati (IITG), 전자통신공학과 공학사
2008년 8월 서울대학교 전기컴퓨터공학부 공학석사
2008년 9월 ~ 현재 삼성전자 DMC 총괄 통신연구소 연구원

주관심 분야 : 영상처리 및 GPGPU