신기술 해설

GPU를 활용한 고성능 컴퓨팅

목 차

전진홍 · 이용우 · 이명호
(명지대학교)

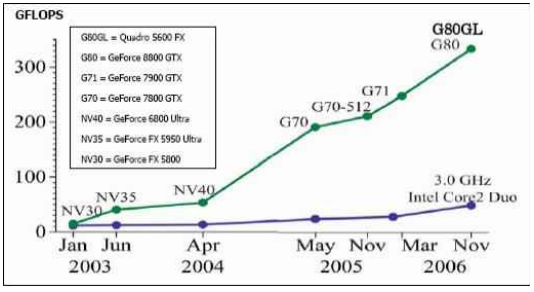
"예전에는 수퍼컴퓨터에서나 가능했던 고성능 컴퓨팅이 데스크탑에서도 가능하게 되었다"라는 말이 나올만큼 컴퓨팅 플랫폼의 획기적인 변화가 최근 일고 있다. 그 중심에는 3차원 그래픽 처리를 가속화하기 위하여 개발된 GPU의 혁명적인 성능 향상이 자리 잡고 있다. GPU는 그래픽 처리용 보조 프로세서로서 3차원 영상 처리 기능을 특화하기 위하여 개발되어왔다. 과연 이러한 GPU가 어떻게 Desktop Supercomputing을 가능하게 하는 것인지, 본 원고에서는 GPU의 발전 과정 및 GPU를 활용한 고성능 컴퓨팅의 예, 앞으로의 발전 전망 등을 살펴보도록 한다.

1. GPU 소개

GPU (Graphic Processing Unit)는 컴퓨터의 영상정보를 처리하거나 화면출력을 담당하는 그래픽 처리용의 보조 프로세서(Co-Processor)이다. 1990년대 후반 3D게임들과 DirectX의 등장으로 이전의 그래픽 카드에 없던 3D 그래픽 처리를 위한 가속 기능이 필요하게 되었고 이에 맞추어 GPU가 등장하게 되었다[1]. GPU의 등장으로 이전의 CPU에서 처리하던 영상 처리가 GPU로 옮겨짐으로써 CPU의 계산량 부담도 덜어주게 되었다. 최초의 GPU는 1999년 8월, 지금은 유명한 GeForce 시리즈의 최초 모델인 NVidia GeForce 256으로 등장하게 되었다. 그 이후 지난 10년간의 계속된 발전으로 현재는 거의 모든 Desptop 컴퓨터에 GPU가 사용되기에 이르렀다.

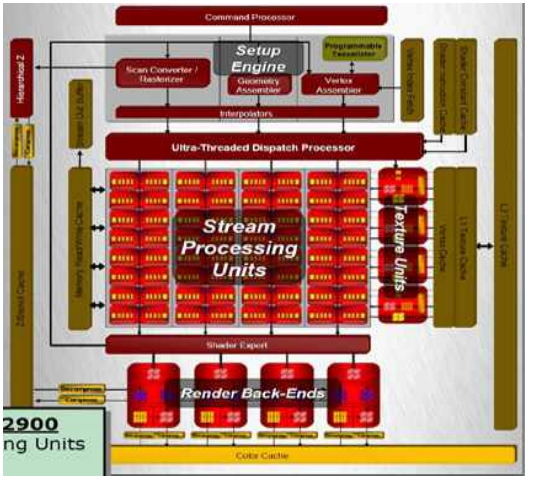
초기 GPU의 클럭 속도가 120MHz였던 반면 최근 GPU의 경우 NVidia GeForce 9 시리즈가 675MHz에 이를 만큼 그 성능이 크게 발전하였다. GPU의 성능 향상은 특히 부동 소수점 연산 능력에서 두드러지게 나타나는데, (그림 1)에서 보듯 2003년 이후 GPU의 부동 소수점 연산 능력은 최신의 마이크로프로세서를 능가하게 되었고 그 격차는 점점 확대되어 가고 있다. 또한 최근의 GPU에서는 Shader, Vertex, Pixel 처리부 등으로 나뉘어 있던 부분들이 하나로 통합되면서 Programmable한 프로세서로 대체되고 있다. 하나의 GPU 카드에 동일한 여러 처리 유닛들이 내장되어 On-Chip 병렬성을 활용한 높은 성능

구현이 가능하게 되었다. 이는 3D 영상 처리 응용 프로그램들이 SIMD (Single Instruction Multiple Data) 방식의 병렬 처리에 적합한 점에 기인한다. 이러한 Programmable 프로세서 기반의 GPU들의 등장은 이들이 그래픽 연산뿐 아니라 많은 계산량을 필요로 하는 고성능 컴퓨팅과 같은 보다 범용 응용 분야로 사용이 확대되는 계기를 제공하고 있다.



(그림 1) 마이크로프로세서와 GPU의 초당 부동 소수점 연산량 비교

GPU를 논하면서 AMD/ATI(그림 2)와 NVidia (그림 3)라는 두 제품군을 빼놓을 수 없다. 2008년 현재 그래픽 카드 시장을 양분 하고 있는 이 두 회사의 모델들은 비슷한 기능과 성능을 제공하며 경쟁중이다.

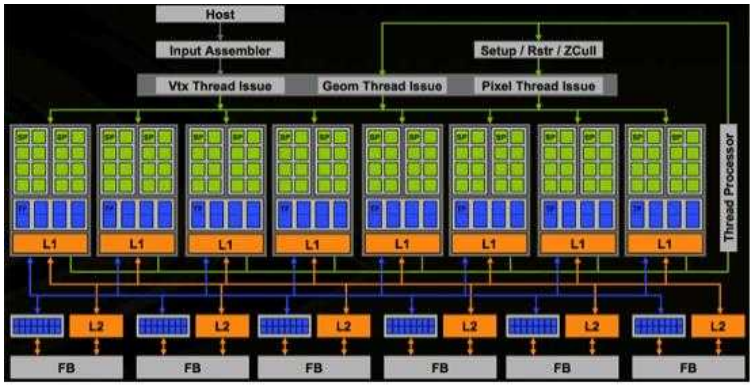


(그림 2) AMD Radeon HD4850 제품 구조

AMD/ATI는 55nm공정에 850Mhz (Shader 1050Mhz)의 클럭 속도를 갖는 480개의 스트림 처리 유닛들을 내장한 GPU를 2008년 7월에 선보였다. 이 GPU는 부동 소수점 연산 최대 성능이 1 Tflop에 이른다. 이 GPU는 256bit 대역폭의 메모리 버스를 통해 GDDR-5 메모리를 사용한다. 이 제품군은 ATI Power Play와 Avivo HD 기술을 접목하여 이미지와 동영상 재생 분야의 응용 프로그램 성능향상을 꾀하였다. 또한 AMD/ATI 제품에서는 멀티코어 CPU처럼 두 개의 GPU 코어를 하나의 칩에 집적하는 Cross-Fire라는 기술을 적용한 제품들도 선보이고 있다. CrossFire의 경우 경쟁사인 NVidia의 SLI 보다 비교적 성능이 떨어지지만, 메인 보드의 PCI-Express 슬롯을 필요로 하지 않아서 시스템 파워에 대한 부담이 비교적 적은 장점이 있다.

현재 AMD/ATI 계열 그래픽 카드의 경우 범용 응용 분야에 활용하기 위해서는 그래픽 관련 API를 사용해야한다. 하지만 Apple의 새로운 MAC OS 버전인 스노우 레오파드 상에서 실행 가능한 OpenCL을 내년에 출시할 계획으로 있어서 경쟁사인 NVidia의 CUDA와 이 분야에서도 경쟁을 하게 될 것으로 보인다. 뿐만 아니라 2007년 AMD가 ATI를 인수하면서 하나의 칩에 CPU와 Programmable GPU 코어가 함께 집적된 제품이 멀지 않은 미래에 출시되리라는 예상은 사용자들로부터 큰 기대를 모으게 하고 있다[1].

NVidia사는 65nm공정에 602Mhz (Shader 1296Mhz)의 클럭 속도를 갖고 240개의 스트림 처리 유닛들을 내장한 GPU를 2008년 6월에 출시하였다. 이 GPU는 512bit 대역폭의 메모리 버스를 통해 GDDR-3 메모리를 사용한다. 클럭 속도 및 처리 유닛의 개수는 AMD 제품보다 낮지만 Shader 클럭의 속도를 높임으로써 전체적으로 비슷한 성능을 보이고 있다. 또한 AMD의 CrossFire에 대항하는 SLI (Scalable Link Interface)를 이용하면 두 개의 그래픽 카드들을



(그림 3) NVidia G80 GPU의 구조

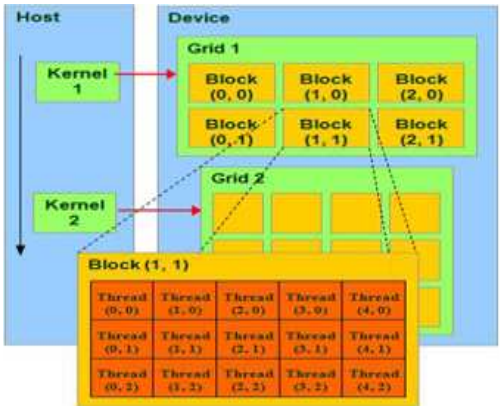
하나의 메인보드에 설치하여 2배의 성능을 이끌어 낼 수 있다. CrossFire와 비교 했을 때 추가로 SLI를 지원하는 보드가 필요하고 충분한 전력의 공급이 필요한 것이 사실이지만 성능에 있어서는 CrossFire보다 상위에 있다[2].

2007년 Tesla제품군을 발표하면서 그래픽 카드에 GPU를 위한 C-언어 개발환경을 통해 어떤 소프트웨어 개발자도 이용할 수 있는 CUDA (Compute Unified Device Architecture)라고 불리는 프로그래밍 환경을 제공하기 시작했다. CUDA는 디버거/프로파일러, 전용 드라이버 및 표준 라이브러리를 위한 C-컴파일러를 포함하고 있는 소프트웨어 개발 솔루션이다. CUDA는 대용량 데이터를 병렬 처리하기 위한 쓰레드 프로그램을 제작하는 표준 C 언어를 이용해 GPU에서의 병렬 컴퓨팅을 편리하게 해준다. CUDA는 윈도우즈 XP와 리눅스 환경을 지원한다. CUDA의 경우 그래픽 관련 API를 사용하지 않고 직접 GPU의 메모리에 접근이 가능하기 때문에 이전에 사용되던 GPU 환경보다 훨씬 쉽고 강력하게 그래픽 연산뿐만 아니라 일반 연산들도 실행 가능하도록 해준다. CUDA를 활용한 고성능 컴퓨팅은 그래픽 이외의 많은 분야에서 이미 좋은 성과를 거두고 있다[3]. CUDA를 활용한 고성능 컴퓨팅 방법론 및 예제들에 관한 자세한 사항은 2장과 3장에서 살펴보기로 한다.

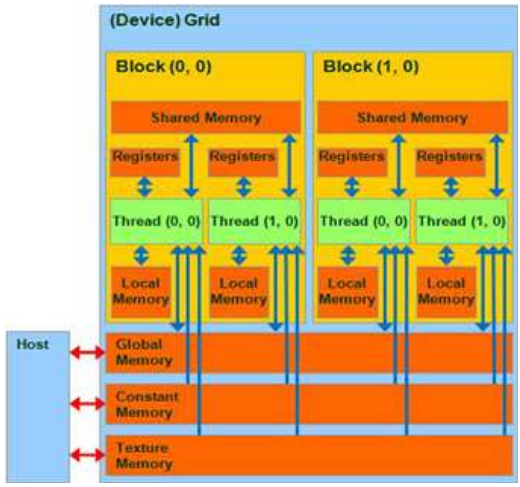
2. GPU의 범용화

GPU는 CPU와 달리 그래픽 처리에 특화된 칩이기 때문에 DirectX나 OpenGL과 같은 그래픽 API를 이용해 간접적으로 일반연산을 처리하는 방법을 사용해 왔다[4]. 이 방법은 하드웨어를 효율적으로 사용하지 못하고 프로그래머의 직관성을 해치는 등의 어려움을 야기해 왔다. 최근의 GPU들은 그래픽 특화된 칩의 모습에서 벗어나, Programmable한 여러 개의 동일한 처리 유닛들의 집합으로 구성되어 있다. 이러한 동일한 처리 유닛들을 사용하여 Shader 연산, Vertex 연산, Pixel 연산 등을 계산 할 수 있다는 의미이다. 이는 GPU가 그래픽 이외의 범용 응용 분야에 활용될 수 있는 길을 터 주었다. 이러한 GPU의 범용화를 촉진하기 위한 솔루션들이 개발 되고 있다. 여러 솔루션들 중 대표적인 것이 NVidia 사에서 개발된 C언어를 확장한 CUDA 개발 환경이다[3].

CUDA 개발 환경은 CUDA 전용 드라이버와 CUDA Toolkit으로 이루어져 있다. CUDA 전용 드라이버를 활용하면 Programmable한 여러 처리 유닛들(또는 쓰레드 프로세서들)에 소프트웨어 쓰레드를 분배하여 병렬 처리를 가능하게 해준다. Toolkit에 포함된 CUDA 컴파일러와 라이브러리는 C언어를 기반으로 한 환경이기 때문에



(그림 4) CUDA 프로그램의 구조



(그림 5) CUDA의 메모리 구조

일반 CPU 응용 프로그래머가 쉽게 다룰 수 있다는 장점이 있다. CUDA에서 제공하는 라이브러리들은 GPU의 메모리에 할당과 접근을 쉽게 해줄 뿐 아니라 Fast Fourier Transformation (FFT) 등의 함수들이 효율적으로 GPU 상에 실행될 수 있도록 하는데 큰 도움을 준다[5]. CUDA의 이러한 기능들은 기존 GPU가 가지고 있는 스트림 컴퓨팅의 한계를 뛰어넘는 탁월한 성능을 보여준다.

CUDA 프로그램은 커널이라고 불리는 GPU 프로그램과 GRID라고 불리는 커널을 수행하는 쓰레드 블록의 배열, 그리고 커널을 수행하고 공유 메모리를 통하여 대화 하는 SIMD 쓰레드의 그룹으로 이루어져 있다(그림 4 참조). CUDA 프로그래밍을 할 때 가장 많이 고려해야 하는 것이 있다면 메모리의 효율적인 사용과 관리이다. CPU가 사용하는 메모리 아키텍쳐와 GPU가 사용하는 메모리 아키텍처는 차이가 있기 때문에 그래픽 카드에서 사용하는 메모리 아키텍쳐에 대한 충분한 이해가 필수적이다. 그래픽 카드에서 사용하는 메모리는 각 쓰레드만 읽고 쓰기 (Read/Write: R/W)가 가능한 레지스터와 로컬 메모리, 쓰레드 블록에서 R/W 가능한 공유 (Shared) 메모리, 그리드에서 R/W가능한 글로벌 / 콘스탄트 / 텍스쳐 메모리로 구성되어 있다 (그림 5 참조). CUDA에서는 레지스터, 공유 메모리, 글로벌 메모리를 자원으로 활용하여 효율적인 애플리케이션을 작성할 수 있다.

글로벌 메모리는 GDDR off-chip 메모리로서 존재하며 호스트 메모리(CPU가 사용하는 메인 메모리)와 통신할 수 있고 256MB에서 크게는 4GB의 용량을 가진다. 프로그램에서 사용할 데이터를 호스트에서 복사하여 글로벌 메모리에 할당 한 다음 쓰레드 블록과 쓰레드에서 공유하여 계산할 수 있게 설계되었다. 공유 메모리는 쓰레드 블록 내에서 공유 가능한 메모리로 16KB의 크기와 레지스터와 거의 비슷한 속도를 가지며, CPU가 사용하는 캐시와 비슷한 역할을 한다. 하지만 CPU의 캐시처럼 OS가 자동으로 관리 하지 않고 프로그래머에 의해 할당되고 제어된다. 이 특성은 프로그램하기 어려운 환경을 제공하지만 프로그래머의 역량에 따라 각 블록에서 계산하는 데이터를 공유 메모리에서 처리하게 함으로써 엄청난 효율의 극대화를 가져 올 수 있다. 레지스터는 각 쓰레드 프로세서에서 커널이 수행될 때 실제로 명령어를 처리하기 위한 저장소로 CPU가 사용하는 레지스터와 비슷한 역할을 한다. CUDA에서는 파일 입출력에 관련

된 기능을 제공 하고 있지 않기 때문에 호스트에서 필요한 데이터를 읽어 들이고, 글로벌 메모리에 복사한 다음 공유 메모리에 프로그래머에 의해 나누어져 쓰레드 블록과 쓰레드에서 연산을 한다. 따라서 작은 데이터에 여러 가지 연산을 하는 프로그램보다는 SIMD 방식과 같이 같은 형식의 큰 데이터를 한꺼번에 처리하는데 강점을 보인다.

3. GPU를 활용한 고성능 컴퓨팅의 예

GPU를 활용한 고성능 컴퓨팅은 많은 성공 신화를 써가고 있다. 특히 NVidia의 CUDA를 사용한 성능 향상의 많은 예들이 최근 보고 된 바 있다. 이러한 예들은 매우 다양한 응용 분야로부터 나왔는데, Neural Network, 주조해석 시뮬레이션, H.264 비디오 인코딩, Molecular Dynamics Simulation, Medical Imaging 등을 들 수 있다. 그들 중 몇 가지 예를 살펴본다.

3.1 행렬 곱셈의 경우

우선 간단한 행렬 곱셈을 CPU만 사용하여 실행할 경우와 CPU와 GPU를 함께 사용한 경우의 성능 차이를 살펴본다. N * N 크기의 2차원 행렬 A, B를 곱하여 N * N 크기의 행렬 C 에 저장하는 코드를 예로 사용한다. GPU를 사용한 경우에는 N의 크기가 BlockSize * n으로 표현된다. 여기서 Blocksize는 GPU 상에서 한 기능 유닛이 한 번에 처리할 행렬의 Block 크기이다. CUDA를 사용하면 CUDA_Init 함수를 사용하여 GPU를 초기화하고 CUDA 라이브러리에 맞춰 연산할 수 있는 환경을 준비한다. 그리고 난후 라이브러리 함수를 호출하여 메인 메모리에 있는 데이터를 GPU의 메모리로 복사한다. 그리고 난후 GPU 상에서 행렬 곱셈 연산을 수행한다. 행렬 곱셈 계산에 사용되는 코드들은 아래의 (그림 6), (그림 7)과 같다.

```
#define BLOCK_SIZE n  // Block_Size
#define MAX_NUM i  // Block_Size * MAX_NUM = 행렬의 SIZE
  for (unsigned int i = 0; i < hA; ++i)
        for (unsigned int j = 0; j < wB; ++j) {
            double sum = 0;
            for (unsigned int k = 0; k < wA; ++k) {
                double a = A[i * wA + k];
                double b = B[k * wB + j];
                sum += a * b;
            }
            C[i * wB + j] = (float)sum;
        }
    }
```

(그림 6) CPU만 사용한 행렬 곱셈 코드

```
matrixMul( float* C, float* A, float* B, int wA, int wB)
{   //block id에 따라 전체 행렬의 구간을 나눔
    for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep) {
      //AS, BS라는 GPU에서 계산에 사용될 sub행렬을 shared memory영역에 할당.
       AS(ty, tx) = A[a + wA * ty + tx];
       BS(ty, tx) = B[b + wB * ty + tx];
       __syncthreads(); //동기화 작업
       for (int k = 0; k < BLOCK_SIZE; ++k) //각 GPU가 할당된 크기의 matrix를 연산함
           Csub += AS(ty, k) * BS(k, tx);
       __syncthreads(); //동기화
    }
    C[c + wB * ty + tx] = Csub; //main memory의 C에 계산된 결과를 저장
}
```

(그림 7) GPU를 활용하는 행렬 곱셈 코드

아래의 (그림 8)은 GPU를 사용한 경우의 행렬곱셈 시간과 CPU만 사용한 경우의 곱셈 시간을 비교한 결과를 보여준다. 실험에 사용한 CPU는 Intel Core2 Duo E6600 (2.4GHz * 2-코어)를 사용하였고, GPU는 GeForce9600 GT (512M RAM, 657MHz 클럭) 카드를 사용하여 테스트해보았다. Blocksize에 따라 성능 향상이 조금 다른 모양으로 나타나고 있지만, 가장 최적의 Blocksize인 16을 사용한 경우, CPU만 사용하는 경우와 비교하여 크게는 172배까지의 성능 향상을 보여준다. 일반적으로 데이터의 크기(x-축으로 표시된)가 커질수록 성능 향상도 커진다.

(그림 8) GPU를 사용한 경우의 성능 향상

3.2 Magnetic Resonance Imaging (MRI) 응용 프로그램의 경우[6]

MRI는 인체에 의료 기기를 삽입 하지 않고 인체 내부를 세밀하게 관찰할 수 있도록 하는 이미지 촬영 기술이다. MRI 장치에서 얻어진 데이터는 Fourier Transform을 통해 이미지로 변환 되는데 여기에서 행렬 곱셈을 수행하게 된다. 거의 대부분의 실행 시간이 이러한 행렬 곱셈에 소요된다. 이러한 행렬 곱셈 연산을 CUDA를 활용하여 GPU 상에서 병렬 처리 할 경우 163배까지 성능을 높일 수 있었다. (CPU는 2.66GHz Intel 쿼드코어(4MB L2캐시, 4GB DRAM), GPU는 NVidia Geforce 8800GTX.)

3.3 Neural Networks 응용 프로그램의 경우[7]

Neural Network은 인간의 두뇌 작용을 신경세포들 간의 연결 관계로 모델링 하여 인간의 학습을 컴퓨터에 적용하는 분야이다. 이 문제는 화상인식, 음성인식, 로봇제어 등 다양한 인공지능 분야에 적용 될 수 있다. 이 문제에서 각 노드의 내적 연산을 행렬의 곱 연산으로 변형 가능하며, 행렬의 곱은 CUDA를 사용하여 GPU에 효율적으로 구현할 수 있다. 따라서 큰 폭의 성능 향상을 관찰 할 수 있다.

3.4 비디오 인코딩의 경우[8]

미디어의 전성시대를 맞이한 지금의 추세에서 고용량 미디어의 압축 기술은 필수 불가결하다. 그러나 이러한 압축 알고리즘에서 Motion Estimation(ME)이 계산의 큰 부분을 차지한다. ME 연산은 높은 복잡도의 연산이라기보다 매우 많은 독립적인 단순 연산이기 때문에 SIMD 방식의 병렬 처리가 가능하여 비교적 쉽게 GPU를 활용하여 성능의 향상을 가져 올 수 있었다. ME 연산이 전체 인코딩 시간의 80%를 차지하는 Full Search의 경우 CPU만 사용하여 연산을 수행한 경우에 비해 36배의 성능 향상을 관찰할 수 있었다. (CPU는 Intel Q6600, DRAM = 4GB, GPU는 NVidia GeForce 8800GTS, CUDA 활용.)

4. 결론 및 향후 전망

최근 GPU의 혁신적인 성능 향상과 사용하기 쉬운 프로그래밍 환경의 개발로 GPU를 활용한 Desktop 고성능 컴퓨팅이 현실로 다가왔다. 이러한 발전은 전통적인 고성능 컴퓨팅 분야인 주조해석, 계산 유체 역학, 기상 예측 모델링, 등의 분야뿐만 아니라 비디오 인코딩, 금융공학, 등의 새로운 응용 분야로까지 확대되고 있다. Desktop 고성능 컴퓨팅은 마치 천문학에서 천체 망원경의 발명과 같이, 각각의 응용 분야에 새로운 지평을 열어줄 것으로 예상된다. 또한 이를 통하여 최신의 GPU를 탑재한 고성능 Desktop 컴퓨터의 보급이 크게 늘어 관련 업계의 매출 신장에도 크게 기여할 것이다.

차세대 Desktop 컴퓨터들은 이러한 고성능 GPU의 보급과 더불어 현재 발전이 가속화되고

있는 멀티-코어 프로세서들이 함께 탑재될 것이다. 다시 말해, 컴퓨터 내부에 두 종류의 고성능 병렬 컴퓨팅 칩들이 내장되게 될 것이다. 이러한 컴퓨터의 성능을 최대한 끌어내기 위해서는 두 종류의 이질적인 병렬 칩들의 사용을 최적화 할 수 있는 병렬화 및 관리 기법들의 개발이 필수적이다. 관련 기술들이 순조롭게 개발된다면, 지금보다 한 차원 더 높은 고성능을 구현함으로써 고성능 컴퓨팅의 지평을 확대하여 더 넓은 여러 응용 분야에 큰 파급 효과를 미치게 될 것이다.

참고문헌

[1] "AMD ATI RadeonTM HD 4800 Series", http://ati.amd.com/products/radeonhd4800/index.html

[2] "NVidia gtx280", http://kr.nvidia.com/object/geforce_family_kr.html

[3] "NVidia CUDA", http://developer.nvidia.com/object/cuda.html

[4] Matt Pharr et. al., "GPU Gems 2", Addison Wesley, 2004.

[5] "NVidia CUDA Programming Guide", http://kr.nvidia.com/object/cuda_develop_kr.html

[6] Sam S. Stone, Haoran Yi, Justin P. Haldar, Wen-mei W. Hwu, Bradley P. Sutton, and Zhi-Pei Liang, "How GPUs Can Improve the Quality of Magnetic Resonance Imaging", Urbana, IL, 2008.

[7] 정기철, "Implementation of Neural Networks using CUDA and OpenMP", 한국정보과학회 하계 학술대회, 2008.

[8] 이재규, "GPU를 이용한 비디오 인코딩", http://kr.nvidia.com/content/cudazone/download/showcase/kr/Voceweb-GPU-assisted-Video-Encoder.pdf, 2008.

저자약력

전 진 홍

2007년 명지대학교 컴퓨터소프트웨어학과 졸업 (학사)
현재 명지대학교 컴퓨터소프트웨어학과 대학원 재학(석사과정)
관심분야 : 병렬처리, GPU 컴퓨팅, 고성능 컴퓨팅
이 메 일 : hooch33@nate.com

이 용 우

2007년 명지대학교 컴퓨터소프트웨어학과 졸업 (학사)
현재 명지대학교 컴퓨터소프트웨어학과 대학원 재학(석사과정)
관심분야 : 병렬프로그래밍, 성능최적화
이 메 일: hizen01@hanmail.net

이 명 호

1986년 서울대학교 계산통계학과 (학사)
1988년 미국 University of Southern California 컴퓨터 과학과 (석사)
1999년 미국 University of Southern California 컴퓨터 공학과 (박사)
1999년~2003년 미국 Sun Microsystems, Inc. Scalable Systems Group, 책임 연구원
2003년~2005년 미국 Sun Microsystems, Inc. Scalable Systems Group, 수석 연구원
2004년~2008년 명지대학교 컴퓨터소프트웨어학과 조교수
2008년~현재 명지대학교 컴퓨터소프트웨어학과 부교수
관심분야 : 고성능 컴퓨팅, 병렬 알고리즘, 멀티코어 마이크로프로세서, 컴파일러
이 메 일 : myunghol@mju.ac.kr