Chapter • 1

왜 CUDA인가? 왜 지금인가?

그리 오래되지 않은 과거에 병렬 컴퓨팅은 "특수한" 작업으로 여겨졌고 컴퓨터 과학에서는 특수 분야로 여겨지던 때가 있었다. 이러한 인식은 최근 몇 년 동안 완전히 바뀌었다. 컴퓨팅 세계에서, 소수만 추구했던 시점에서 장차 프로그래머가 되려고 하는 거의 모든 사람이 최대한 효과적인 결과물을 개발하기 위해 병렬 프로그래밍을 학습해야 하는 시점에 다다랐다. 어쩌면 여러분은 오늘날 컴퓨팅 세계에서의 병렬 프로그래밍의 중요성과 앞으로 그것이 얼마나 큰 활약을 할지를 확신하지 못한 채 이 책을 선택했을지도 모른다. 이 장에서는 프로그래머들이 작성한 소프트웨어를 열심히 구동하는 최근 하드웨어의 동향에 대해 보여줄 것이다. 그렇게 함으로써 우리는 여러분에게 병렬 컴퓨팅 혁명이 이미 발생했음을 확신시켜주고, CUDA C를 배움으로써 여러분이 CPU와 GPU를 포함한 헤테로지니어스 플랫폼에서 고성능 어플리케이션을 작성할 수 있는 좋은 위치에 있기를 희망한다.

1.1 이번 장의 목표

이 장을 통해 여러분은 다음 사항들을 학습할 것이다:

- 나날이 증가하는 병렬 컴퓨팅의 역할의 중요성에 대해 배운다.
- 병렬 컴퓨팅의 간략한 역사와 CUDA에 대해 배운다.
- CUDA C를 이용하는 일부 성공적인 어플리케이션에 대해 알아본다.

1.2 병렬 프로세싱의 시대

최근 몇 년 동안 컴퓨팅 산업에서 광범위한 것들이 병렬 컴퓨팅으로 바뀌었다. 2010년 이후의 모든 컴퓨터는 멀티코어 CPU를 장착할 것이다. 저가의 넷북부터 8개, 16개 코어의 워크스테이션에 이르는 컴퓨터들의 도입으로 병렬 컴퓨팅은 더 이상 특수한 슈퍼컴퓨터나 중앙컴퓨터에 국한되지 않을 것이다. 게다가, 휴대폰이나 휴대용 음악 재생기와 같은 전자 장치는 이전 모델들보다 더 나은 기능을 제공하기 위해 병렬 컴퓨팅 능력을 포함하기 시작했다.

갈수록 사용자들의 눈높이가 높아지기 때문에 소프트웨어 개발자들은 이들을 위한 기발하고 풍부한 사용자 경험을 제공하기 위해서 갖가지 병렬 컴퓨팅 플랫폼과 기술들을 더 많이 처리해야 할 것이다. 명령 프롬프트의 시대는 끝났으며, 이제 멀티스레드 기반의 그래픽 인터페이스가 사용된다. 단지 전화만 할 수 있었던 휴대폰은 사라지고, 음악을 재생하고 웹 검색을 하며 GPS 서비스를 동시에 제공할 수 있는 전화기들이 보급된다.

1.2.1 중앙 처리 장치

30년 동안, 소비자 컴퓨팅 장치의 성능 향상을 위한 주요한 방법 중 하나는 프로세서의 클럭 속도를 높이는 것이었다. 1980년대 초반에 최초의 개인용 컴퓨터가 등장했을

때 소비자 중앙 처리 장치(CPU)는 약 1MHz의 내부 클럭 속도로 작동하였다. 약 30년 후, 대부분의 데스크톱 프로세서들은 최초의 개인용 컴퓨터의 클럭보다 거의 1000배나 빠른 1GHz에서 4GHz 사이의 클럭 속도를 가지고 있다. CPU 클럭 속도의 증가가 컴퓨팅 성능을 증가시킨 유일한 수단은 분명 아니지만, 그것은 항상 성능 향상을 위한 신뢰할만한 원천이었다.

그러나 최근 제조업체들은 이러한 증가된 계산 능력의 전통적인 수단에 대한 대안방안들을 물색해야만 했다. 직접 회로의 제조에 있어서 여러 가지 근본적인 제한으로, 기존의 구성에서 추가적인 출력을 추출하기 위한 수단으로 상승중인 프로세서 클럭 속도에 의존하는 것은 더 이상 실현 불가능했기 때문이다. 전력과 열의 제한뿐만 아니라 물리적인 트랜지스터 크기도 급속도로 한계에 도달했기 때문에 연구원들과 제조업체들은 다른 곳에서 방안을 찾기 시작했다.

소비자 컴퓨팅 세계와 동떨어진 슈퍼컴퓨터들은 유사한 방법으로 수십 년 동안 막대한 성능 향상을 이루었다. 슈퍼컴퓨터에서 사용된 프로세서의 성능은 개인용 컴퓨터 CPU와 비슷하게 천문학적으로 향상했다. 하지만 슈퍼컴퓨터 제조업체들은 이러한 단일 프로세서 성능의 극적인 향상뿐만 아니라 프로세서의 개수 또한 꾸준히 증가시킴으로써 성능에서의 막대한 도약을 이끌어 낼 수 있었다. 가장 빠른 슈퍼컴퓨터가 수만 또는 방대한 수의 프로세서 코어들을 동시에 가동하는 것은 드문 일이 아니다.

개인용 컴퓨터의 연산 능력 향상을 위한 연구에서 슈퍼컴퓨터의 성능 향상으로부터 좋은 영감을 얻을 수 있었다 – 오로지 단일 프로세싱 코어의 성능 향상만 고려하기 보다는 하나의 개인용 컴퓨터에 하나 이상의 코어들을 장착하는 것이 어떠한가? 이 방법으로 프로세서 클럭 속도를 계속 증가할 필요 없이 개인용 컴퓨터의 성능을 향상시킬 수 있을 것이다.

2005년에는 시장 경쟁이 갈수록 치열해지는 가운데 소수의 대안 방안에 직면한 선두의 CPU 제조업체들이 하나 대신 두 개의 컴퓨팅 코어를 탑재한 프로세서들을 내놓기 시작했다. 그 후로 세 개, 네 개, 여섯 개 그리고 여덟 개 코어의 중앙 처리 장치들을 출시하면서 이러한 개발은 지속되었다. 가끔 멀티코어 혁명이라고 언급되면서, 이러한 추세는 소비자 컴퓨팅 시장의 혁명에 있어서 거대한 변화의 전조가 되었다.

오늘날, 단일 컴퓨팅 코어가 탑재된 데스크톱 컴퓨터를 구입하는 것이 상대적으로 쉽

지 않다. 심지어 저렴하고 저출력의 중앙 처리 장치도 하나의 칩에 둘 또는 그 이상의 코어들을 탑재하고 있다. 선두의 CPU 제조업체들은 이미 12, 16 코어의 CPU들을 출시할 계획을 발표했으며, 이는 병렬 컴퓨팅이 적절한 시기에 도달해 있음을 더욱 확신시켜준다.

1.3 GPU 컴퓨팅의 도래

중앙 처리 장치의 전통적인 데이터 처리 파이프라인과 비교해보면, 그래픽 처리 장치(GPU) 상에서 범용 계산을 수행하는 것은 하나의 새로운 개념이다. 사실 일반적으로 GPU 자체를 컴퓨팅 분야와 비교했을 때는 상대적으로 새롭지만, 그래픽스 프로세서의 컴퓨팅에 대한 발상은 여러분이 생각한 것처럼 새롭지 않다.

1.3.1 GPU의 간략한 역사

우리는 이미 CPU의 클럭 속도와 코어 개수에 대해 살펴보았다. 그러던 중 그래픽스 처리 기술에는 극적인 혁명이 있었다. 1980년대 후반과 1990년대 초반에 Microsoft 윈도우즈(Microsoft Windows)와 같은 그래픽 기반의 운영체제들의 인기 상승은 새로운 종류의 프로세서 시장을 창출했다. 1990년대 초창기에 사용자들은 개인 컴퓨터를 위한 2D 디스플레이 가속기를 구입하기 시작했다. 이러한 디스플레이 가속기들은 그래픽 기반의 운영체제 화면 정보와 사용성을 보조하기 위해 하드웨어 비트맵 연산을 제공하였다.

같은 시점에 1980년대 전문직의 컴퓨팅 세계에서 실리콘 그래픽스(Silicon Graphics)라는 이름의 한 회사는 다양한 시장에서 3차원 그래픽스의 사용을 대중화시켰는데, 이 3차원 그래픽스는 정부 및 국방부의 어플리케이션들과 과학 기술에서 시각적 표현을 위해 사용되었다. 또한 실리콘 그래픽스는 3차원 그래픽스 툴들을 영화의 효과를 제작하기 위한 도구로도 제공하였다. 1992년도에 실리콘 그래픽스는 OpenGL 라이브러리를 배포함으로써 하드웨어에 대한 프로그래밍 인터페이스를 공개했다. 실리콘 그래픽스

는 3D 그래픽스 어플리케이션을 작성하기 위해 플랫폼 독립적인 표준 그래픽 라이브러리로서 OpenGL이 사용되길 의도했다. 병렬 처리 및 CPU와 마찬가지로, 그것은 단지 소비자 어플리케이션에 대한 그들만의 기술 방식이 결정되기 전의 하나의 시간 문제였을 것이다.

1990년대 중반쯤에는 상당히 중요한 두 개발 단계를 거치면서 3D 그래픽스를 이용하는 소비자 어플리케이션의 요구가 급속히 증가했다. 첫째로, 둠(Doom), 듀크 뉴켐 3D(Duke Nukem3D) 그리고 퀘이크(Quake)와 같은 사용자들을 에워싸는 듯한 1인칭 게임들의 발매는 PC 게이밍을 위한 더욱 실사 같은 3D 환경들을 계속 제작하기 위한 연구를 점화하는 데 도움이 되었다. 결국 3D 그래픽스가 거의 모든 컴퓨터 게임에 적용되었기는 하지만, 초창기 1인칭 슈팅 장르의 인기가 소비자 컴퓨팅에서 3D 그래픽스를 적용하는 매우 큰 계기가 되었을 것이다. 동시에 NVIDIA, ATI 테크놀로지스(ATI Technologies), 3dfx 인터렉티브(3dfx Interactive)와 같은 회사들은 전 세계의 이목을 끌기에 충분한 그래픽 가속기들을 발매하기 시작했다. 이러한 개발들이 3D 그래픽스를 미래의 매우 중요한 하나의 기술로 결속시켰다.

NVIDIA GeForce 256의 발매는 소비자 그래픽스 하드웨어의 성능을 더욱 부추겼다. 최초로 변환과 광원 연산이 그래픽스 프로세서 상에서 직접 수행될 수 있었다. 이로 인해 시각적으로 흥미로운 어플리케이션들의 가능성이 한층 향상되었다. 변환과 광원이 OpenGL 그래픽스 파이프라인의 일부분이 됨으로써, GeForce 256은 점차 그래픽스 파이프라인의 더 많은 부분들이 그래픽스 프로세서 상에서 직접 구현되는 진보의 시작의 거점이 되었다.

병렬 컴퓨팅의 관점에서, 2001년 NVIDIA GeForce 3 시리즈의 발매는 GPU 기술에서의 가장 중요한 돌파구이었음이 틀림없다. GeForce 3 시리즈는 Microsoft의 새 DirectX 8.0 표준을 구현하기 위한 컴퓨팅 산업에서의 첫 번째 칩이었다. 이 표준은 하드웨어가 프로그래밍이 가능한 버텍스와 프로그래밍이 가능한 픽셀 세이딩 단계들을 모두 포함하길 요구했다. 처음으로 개발자들은 GPU 상에서 수행될 수 있는 정밀한 계산들의 일부를 조작할 수 있었다.

1.3.2 초창기의 GPU 컴퓨팅

프로그래밍이 가능한 파이프라인을 탑재한 GPU들이 발매되자 OpenGL 또는 DirectX를 이용하여 단순 렌더링 목적 이상으로 그래픽스 하드웨어를 사용할 가능성이 더 많아졌으며 이는 많은 연구가들의 관심의 대상이 되었다. 초창기 시절 GPU 컴퓨팅의 일반적인 접근법은 특히 일방적이었다. 왜냐하면 OpenGL이나 DirectX와 같은 표준 그래픽스 API들이 GPU와 상호작용하는 유일한 방법이었으며, GPU에서 임의의 계산을 수행하려고 시도하는 것은 그래픽스 API로 프로그래밍 하기엔 제한이 있었기 때문이다. 이 때문에, 연구가들은 전통적인 렌더링을 수행하는 GPU에 그들의 문제를 맞추려고 노력함으로써 그래픽스 API들을 통한 범용의 계산을 강구했다.

기본적으로 2000년대 초반의 GPU들은 픽셀 셰이더(Pixel Shader)로 알려진 프로그래밍이 가능한 산술 연산 장치를 사용하여 스크린 상의 모든 픽셀의 색상을 생성할 수 있도록 설계되었다. 보통 픽셀 셰이더는 스크린 상의 (x, y) 위치를 이용할 뿐만 아니라, 최종 색상을 계산하는 데 있어서 여러 입력 정보들을 결합하기 위한 추가적인 정보들을 이용한다. 추가적인 정보는 입력된 색상 값, 텍스처(texture) 좌표 또는 작동 중에 셰이더로 전달될 수 있는 다른 어떤 속성들이 될 수 있다. 그러나 프로그래머들은 입력된 색상 값들과 텍스처들에 대해 수행되는 산술 연산을 완전히 조작할 수 있었으며, 연구가들은 이러한 입력 "colors" 값들이 실제로 **어떠한 데이터**도 될 수 있다는 것을 알 수 있었다.

따라서 만약 입력 값이 실제로 색상이 아닌 다른 것을 의미하는 수치적인 데이터이면, 프로그래머들은 이러한 데이터들을 이용하여 임의의 계산을 수행하도록 픽셀 셰이더를 프로그래밍 할 수 있었다. 프로그래머들이 그들의 입력값들에 대해서 GPU가 계산하도록 명령을 내리면 어떠한 계산 결과가 되든지 간에 그 결과들은 최종 픽셀 "color"로서 다시 GPU로 전달되었다. 연구가들은 이 데이터를 다시 읽을 수 있었고 GPU는 결코 더 똑똑해 질 수 없었다. 실질적으로 GPU는 비렌더링(nonredering) 작업을 하나의 일반 렌더링처럼 수행하도록 처리되고 있었다. 이러한 속임수는 매우 기발했지만 유연하지는 못했다.

GPU의 높은 산술 처리 능력으로 이러한 실험의 초기 결과는 GPU 컴퓨팅에 대한 밝은 미래의 조짐을 보여주었다. 하지만 일부 비판적인 개발자들 무리에게는 이 프로그래밍

모델이 정형화되기에는 너무 제한이 많아 보였다. 프로그램들은 오직 소수의 입력 색상 값들과 소수의 텍스처 구성 단위로만 입력 데이터를 받을 수 있었기 때문에 자원이 넉넉하지 못한 제한사항들이 있었다. 프로그래머들이 어떻게 그리고 어디서 결과값들을 메모리에 기록할 수 있는지에 대한 심각한 제한도 있었다. 그래서 산재된 메모리 중 임의의 위치에 데이터를 기록해야 하는 알고리즘은 GPU에서 수행될 수 없었다. 게다가, 만약 여러분이 어떤 식으로라도 부동소수점 데이터를 처리한다고 하면 특정 GPU가 부동 소수점 데이터를 어떻게 처리하는지 예측하는 것은 거의 불가능했다. 그래서 대부분의 과학적 계산에서는 GPU를 사용할 수 없었다. 마지막으로, 프로그램이 종료하는데 실패하거나 장비를 단순히 멈추게 하는 부정확한 결과를 불가피하게 계산했을 때 GPU에서 실행되는 코드를 디버깅할 수 있는 좋은 방법이 존재하지 않았다.

제한사항들이 충분히 엄격하지 않았을지라도, OpenGL과 DirectX는 GPU와 상호작용할 수 있는 유일한 수단이었기 때문에 범용 계산을 수행하기 위해 GPU를 사용하길 원하는 이들은 여전히 OpenGL과 DirectX를 배우길 희망했다. 이것은 OpenGL과 DirectX 함수들을 호출하여 그래픽스 텍스처들에 데이터를 저장하고 계산을 수행하는 것을 의미할 뿐만 아니라, 셰이딩 언어(shading language)로 알려진 특별한 그래픽스 프로그래밍 언어들을 이용하여 계산 프로그램을 작성해야 함을 의미했다. 광범위하게 사용되기에는 너무 많은 난관이 있는 GPU의 연산 능력을 이용하기 위해서 연구가들은 몇몇 자원과 프로그래밍의 제약 모두를 극복해야 할 뿐만 아니라, 컴퓨터 그래픽스와 셰이딩 언어들을 배워야만 했다.

1.4 CUDA

GeForce 3 시리즈가 출시된 지 5년도 채 되지 않아서 GPU 컴퓨팅은 전성기를 맞이하고 있었다. 2006년 11월, 컴퓨팅 칩 산업에서 첫 번째 DirectX 10 GPU인 GeForce 8800 GTX가 베일에서 드러났다. 또한 GeForce 8800 GTX는 NVIDIA의 CUDA 아키텍처 기반의 첫 번째 GPU였다. 이 아키텍처는 GPU 컴퓨팅을 위해 엄격히 설계된 몇 개의 새로운 요소들을 포함했고, 유용한 범용 계산에 있어서 이전 그래픽스 프로세서

들이 가진 제한사항들을 완화하는 것을 목표로 하고 있었다.

1.4.1 CUDA 아키텍처란 무엇인가?

컴퓨팅 자원들을 버텍스와 픽셀 셰이더로 분할했던 이전 세대와는 다르게, CUDA 아키텍처는 통합된 셰이더 파이프라인을 포함한다. 이 파이프라인은 범용 목적의 계산을 수행하는 프로그램이 칩에 있는 모든 산술 논리 연산 장치(ALU)를 통제하도록 허용한다. NVIDIA는 범용의 목적으로 이 새로운 구성의 그래픽스 프로세서들을 설계했기 때문에 이들의 ALU은 단일 정밀도 부동 소수점 연산이 IEEE 요구사항과 부합하도록 제작되었고, 특별히 그래픽스를 위한 것보다는 일반적인 계산 목적에 맞춰진 명령셋을 사용하도록 설계되었다. 게다가, GPU의 실행 유닛(execution unit)들은 메모리에 대한 임의의 읽기, 쓰기가 가능했을 뿐만 아니라 공유 메모리로 알려진 소프트웨어에 의해 관리되는 캐시에 대한 접근도 가능했다. 고전 그래픽스 작업을 잘 수행할 뿐만 아니라 계산에서도 탁월한 GPU를 제작하기 위해 CUDA 아키텍처에 이런 모든 특징이 추가되었다.

1.4.2 CUDA 아키텍처의 사용

그렇지만 계산 및 그래픽스 모두를 위한 제품을 소비자들에게 제공하기 위한 NVIDIA의 노력은 CUDA 아키텍처가 결합된 하드웨어의 생산에서 멈출 수는 없었다. NVIDIA가 계산을 용이하게 하기 위한 많은 기능들을 그들의 칩에 추가한 것과 상관없이, OpenGL과 DirectX를 이용하지 않고서는 이러한 기능들에 접근할 수 있는 방법은 여전히 없었다. 이것은 사용자들이 그들의 계산을 그래픽스 문제로 위장해야 하고, OpenGL의 GLSL 또는 Microsoft의 HLSL과 같은 그래픽스에 중점을 둔 셰이딩 언어를 이용한 프로그래밍을 계속해야 함을 의미했다.

최대한 많은 개발자들을 받아들이기 위해 NVIDIA는 산업 표준 C 언어를 채택했고 CUDA 아키텍처의 일부 특정 기능들을 이용하기 위한 소수의 관련 키워드들을 추가했다. GeForce 8800 GTX를 출시한지 몇 달 후, NVIDIA는 CUDA C 언어를 위한 대중적인 컴파일러를 제작하였다. 그리고 한 GPU 회사에 의해 CUDA C는 그 컴파일러와

더불어 GPU에서 범용 계산이 가능하도록 특별히 제작된 첫 번째 언어가 되었다.

GPU 코드를 작성하기 위한 하나의 언어를 창조하는 것 외에 NVIDIA는 CUDA 아키텍처의 대규모 연산 능력을 활용하기 위해 전문 하드웨어 드라이버를 제공하였다. 이제 사용자들은 OpenGL이나 DirectX의 그래픽스 프로그래밍 인터페이스들에 대한 어떠한 지식도 필요 없게 되었고, 그들의 문제를 컴퓨터 그래픽스 작업에 맞춰 작성할 필요도 없게 되었다

1.5 CUDA를 이용한 어플리케이션

2007년 초, CUDA C가 등장한 이래로 다양한 산업 및 응용 분야에서는 CUDA C를 채택하여 어플리케이션을 제작하여 커다란 성공을 누렸다. 이전의 최신식 구현 방안들보다 몇 자리 높은 수치의 성능 향상도 이러한 CUDA C의 이점에 해당된다. 더욱이, NVIDIA 그래픽스 프로세서로 처리되는 어플리케이션들은 가격대비 우수한 성능향상이 있으며 전통적인 중앙 처리 기술 전용으로 구현한 것보다도 와트(watt) 대비 우수한 성능향상을 갖는다. 다음은 CUDA C와 CUDA 아키텍처를 적용한 이들의 몇 가지 성공적인 사례다.

1.5.1 의학 화상(MEDICAL IMAGING)

과거 20년 동안, 비극적인 유방암에 걸린 사람들의 수가 극적으로 증가했다. 마찬가지로, 최근 몇 년 동안 많은 이들의 지칠지 모르는 노력에 크게 힘입어 이러한 끔직한 질병에 대한 예방, 치료법에 대한 관심과 연구활동 또한 증가하였다. 궁극적으로 방사선 치료와 화학 요법, 수술에 대한 영구적인 기억, 치료 실패시의 치명적인 결과 등의 부작용으로부터 고통을 막기 위해 모든 유방암은 초기에 잡아야 한다. 결국 연구가들은 초기 유방암의 증세와 빠르고 정확하고 최소한의 외과적 방법을 찾기 위해 함께 노력하고 있다.

초기 유방암 발견을 위한 현존하는 최고의 기술들 중 하나인 유방조영술은 몇 가지 중요한 제한사항들을 가지고 있다. 둘 이상의 영상을 촬영해야 하고 잠재적 종양을 발견하기 위해 숙달된 의사가 필름을 개발하고 읽어야 한다. 게다가 이 엑스레이(X-Ray) 절차는 엑스레이가 환자의 흉부에 반복해서 방출될 시의 모든 위험을 가지고 있다. 세심한 검토 후, 의사들은 암의 가능성을 제거하기 위해 특별 화상 –심지어 조직 검사– 을 종종 더 요구한다. 이러한 검사가 실패하면 고비용의 추가 작업들이 필요하며, 최종 결론이 도출되기 전까지 환자들에게 과도한 스트레스를 유발한다.

초음파 화상은 엑스레이 화상보다 안전하다. 그래서 의사들은 종종 유방조영술과 함께 초음파 화상을 이용하여 유방암 검사 및 진단을 보조한다. 그러나 종래의 가슴 초음파 화상 역시 여러 제한사항들을 가지고 있다. 그 결과, 테크니스캔 메디컬 시스템(TechniScan Medical Systems) 사가 탄생했다. 테크니스캔은 유망한 3차원 초음파 화상 방법을 개발했다. 그러나 그 해결책은 단순한 한 가지 이유로 현실적이지 못했다. 바로 계산의 제한이었다. 수집한 초음파 데이터를 단순히 입력하고 3차원 이미지로 바꾸는 작업은 계산하는 데 많은 시간이 소요되었고, 실제로 사용하기에는 비용이 너무 비쌌다.

테크니스캔은 CUDA C 프로그래밍 언어와 함께 CUDA 아키텍처를 기반으로 한 NVIDIA의 최초의 GPU를 도입하여 설립자들의 꿈을 현실로 이룰 수 있는 하나의 플랫폼을 가질 수 있었다. 이름에서 나타나듯이, 스바라(Svara) 초음파 화상 시스템은 환자의 가슴을 이미지화하기 위해 초음파 파장을 사용한다. 테크니스캔의 스바라 시스템은 15분 동안 스캔하여 생성한 35GB 데이터를 가공하기 위해 두 개의 NVIDIA 테슬라(Tesla) C1060 프로세서를 이용하고 있다. 테슬라 C1060의 계산덕분에 의사는 고도로 섬세한 여성의 3차원 가슴이미지를 20분 내로 다룰 수가 있다. 테크니스캔은 2010년에 시작된 그들의 스바라 시스템이 널리 보급되길 기대하고 있다.

1.5.2 컴퓨터를 이용한 유체동역학

수년 동안 고성능의 회전자와 날개를 설계하는 일은 일종의 마술로 여겨졌다. 공기와 유체는 이러한 장비들 주위로 놀라울 정도로 복잡하게 움직였으며, 단순한 공식으로는 실질적인 견본이 될 수가 없었다. 현실적으로 정확한 시뮬레이션들은 계산을 수행하는

데 비용이 너무 많이 들었다. 유일하게 세상에서 가장 큰 슈퍼컴퓨터들만이 설계한 것들을 개발하고 입증할 수 있는 정교한 수치 모델 수준의 계산 자원들을 제공할 수 있었다. 그러나 그러한 기계에 접근할 수 있는 사람의 수가 적었기 때문에, 그러한 기계의 설계 혁신은 계속 침체되었다.

찰스 배비지(Charles Babbage)에 의해 위대한 전통이 시작된 캠브릿지(Cambridge) 대학은 고급 병렬 컴퓨팅에 대한 활발한 연구가 진행중인 병렬 컴퓨팅의 고향이다. "매니 코어 그룹(many-core group)"의 그라함 풀란(Graham Pullan) 박사와 토비아스 브랜빅(Tobias Brandvik) 박사는 컴퓨터를 이용한 유체 동역학을 전례 없는 수준까지 향상시키기 위해 NVIDIA CUDA 아키텍처의 잠재성을 정확히 찾아냈다. 그들의 첫 연구는 GPU를 이용하는 개인 단말기에서도 허용 수준의 성능이 산출될 수 있음을 보여주었다. 그 후, 하나의 작은 GPU 클러스터를 사용하는 것으로 그보다 훨씬 더 많은 비용이 드는 슈퍼컴퓨터의 능력을 쉽게 능가할 수 있었고, NVIDIA GPU의 능력이 해결해야 할 문제들을 소화하는 데 매우 적합하다는 것을 입증하였다.

CUDA C를 통한 막대한 성능 향상은 캠브릿지의 연구원들에게는 그들의 슈퍼컴퓨팅 제원을 단순히 증가시키는 것 이상을 의미한다. 방대한 양을 처리할 수 있는 저가 GPU의 계산 능력은 캠브릿지 연구원들이 실험을 빨리 수행할 수 있는 기회를 제공하였다. 단 몇 초 만에 실험 결과를 도출함으로써 돌파구에 도달하기 위해 연구원들이 의지하는 피드백 과정이 간소화되었다. 그 결과, 연구원들은 GPU 클러스터를 이용함으로써 그들의 연구 방식을 완전히 탈바꿈하였다. 대부분의 상호작용하는 시뮬레이션들은 이전의 억압된 연구 분야에서 혁신과 창조를 위한 새로운 기회를 불러일으켰다.

1.5.3 환경 과학

세계 경제 산업은 급속도로 성장하고 있으며 이에 대한 결과로 환경 관련 소비자 상품이 점점 더 필요해졌다. 산업 생산의 성공적인 진전으로 기후의 변화, 연료비의 상승, 공기와 물의 오염 증가로 인한 부차적인 피해의 우려가 커졌으며, 이는 극심한 구호를 야기했다. 세제와 청정제는 오랫동안 가장 유용한 생필품이었으며, 잦은 사용시 잠재적으로 재앙을 초래하는 좋지 못한 소비재였다. 그 결과, 많은 과학자들은 효능을 유지하면서도 세제처럼 환경에 끼치는 영향을 줄이는 방법을 탐구하기 시작했다. 그러나

이를 성공하는 것은 까다로운 문제가 될 수 있었다.

청정제의 핵심 요소는 표면 활성제로 알려져 있다. 표면 활성제 분자들은 세탁 능력 및 세제와 샴푸의 질감을 결정한다. 하지만 이들은 흔히 청정제들 중 가장 큰 환경 파괴 요소의 원인이기도 하다. 표면 활성제들이 더러운 때와 함께 씻기는 것처럼, 이러한 분자들은 때에 들러붙은 후 물과 섞인다. 자고로, 새로운 표면 활성제의 청정 수준을 측정하기 위해서는 물질들과 청정될 불순물들의 혼합물을 많이 갖춘 대규모 실험실에서의 실험이 필요하다. 당연히 이러한 과정은 매우 느리고 비용이 비싸다.

템플(Temple) 대학은 표면 활성제의 분자 시뮬레이션을 이용하기 위해 일류업계인 프록터앤겜블(Proctor & Gamble) 사와 함께 일했으며 표면 활성제는 때와 물 그리고 다른 물질들과 함께 상호작용한다. 컴퓨터 시뮬레이션을 도입함으로써 전통적인 연구의 처리 속도가 빨라졌고, 과거의 실질적인 실험보다 훨씬 더 폭넓은 실험이 가능했다. 템플 대학의 연구원들은 자원부 산하 아메스(Ames) 연구소에 의해 제작된, GPU 가속을 받은 고도로 최적화된 객체지향 다입자 동역학(Highly Optimized Object-oriented Many-particle Dynamics, HOOMD) 시뮬레이션 소프트웨어를 사용했다. 그들의 시뮬레이션을 두 개의 NVIDIA 테슬라 GPU로 분담시킴으로써, 크레이(Cray) XT3의 128개 CPU 코어들 및 IBM 블루진(BlueGene)/L 장비의 1024개 CPU들과 동등한 성능을 발휘시킬 수 있었다. 그들은 이미 해결책으로 테슬라 GPU의 개수를 늘림으로써 기존의 플랫폼보다 16배나 월등한 성능의 표면 활성제 상호작용 시뮬레이션을 가동하고 있다. NVIDIA의 CUDA가 그러한 종합적인 시뮬레이션을 완료하는 데 걸리는 시간을 몇 주로부터 단 몇 시간으로 단축시켰기 때문에, 앞으로도 효과는 친환경적인 제품들의 극적인 증가에 도움을 줄 것이다.

1.6 요약

컴퓨팅 산업은 병렬 컴퓨팅의 최절정에 도달했으며 NVIDIA의 CUDA C는 지금까지 병렬 컴퓨팅을 위해 설계된 가장 성공적인 언어들 중 하나다. 이 책을 통해 여러분은 CUDA C를 이용한 여러분만의 코드를 작성하는 방법을 배울 수 있을 것이고, 또한 C

언어의 특별 확장 문법들과 NVIDIA가 GPU 컴퓨팅을 위해 제작한 어플리케이션 프로그래밍 인터페이스들을 익힐 수 있을 것이다. 우리는 여러분이 OpenGL이나 DirectX를 알고 싶어하지 않고, 컴퓨터 그래픽스의 어떠한 배경지식도 배우고 싶어하지 않을 것이라고 가정한다.

여기서 C 프로그래밍의 기초에 대해서는 다루지 않을 것이다. 따라서 컴퓨터 프로그래밍을 처음 접하는 사람들에게는 이 책을 추천하지 않는다. 여러분이 병렬 프로그래밍에 대한 경험이 있을 것이라고 기대하지는 않지만, 병렬 프로그래밍에 익숙하다면 도움은 될 것이다. 여러분이 이해해야 할 병렬 프로그래밍에 관련된 전문 용어 및 개념들은 본문에서 설명할 것이다. 사실, 전통적인 병렬 프로그래밍에서 여러분은 GPU 컴퓨팅의 잘못된 판정을 추정해야 하는 경우도 있을 것이다. 따라서 실제로 이 책을 통해 그것을 확인하기 위해서는 보통 수준의 C 나 C++ 프로그래밍 경험만 있으면 된다.

다음 장에서는 GPU 컴퓨팅을 시작하기 위해 반드시 필요한 하드웨어와 소프트웨어를 갖출 수 있도록 여러분의 장비를 설정하는 방법을 알려줄 것이다. 그러한 설정을 마치면 여러분은 CUDA C를 이용한 프로그래밍을 할 준비가 끝난 것이다. 만약 CUDA C에 대한 경험이 있거나 여러분의 시스템이 CUDA C를 이용한 개발 준비를 제대로 갖추고 있다면 3장으로 건너뛰어도 좋다.