opencl 번역

1. Introduction

최신 프로세서 아키텍처는 성능 향상을위해 병렬 처리를 채택했습니다. 고정 된 전력 범위에서 높은 클럭 속도로 기술적 문제에 직면 한 현재 중앙 처리 장치 (CPU)는 다중 코어를 추가하여 성능을 향상시킵니다. 그래픽 처리 장치 (GPU)는 고정 기능 렌더링 장치에서 프로그래머블 병렬 프로세서로 진화했습니다. 오늘날의 컴퓨터 시스템은 고도의 병렬 CPU, GPU 및 다른 유형의 프로세서를 포함하기 때문에 소프트웨어 개발자가 이러한 기종 처리 플랫폼을 최대한 활용할 수 있도록하는 것이 중요합니다.

이기종 병렬 처리 플랫폼을위한 애플리케이션을 개발하는 것은 멀티 코어 CPU와 GPU에 대한 전통적인 프로그래밍 방식이 매우 다르기 때문에 어려운 일입니다. CPU 기반 병렬 프로그래밍 모델은 일반적으로 표준을 기반으로하지만 일반적으로 공유 주소 공간을 가정하며 벡터 작업을 포함하지 않습니다. 범용 GPU 프로그래밍 모델은 복잡한 메모리 계층 및 벡터 연산을 처리하지만 플랫폼, 벤더 또는 하드웨어에 따라 다릅니다. 이러한 제한으로 인해 개발자는 하나의 다중 플랫폼 소스 코드 기반에서 이기종 CPU, GPU 및 기타 유형의 프로세서의 컴퓨팅 성능에 액세스하는 것을 어렵게 만듭니다. 그 어느 때보 다 소프트웨어 개발자가 고성능 컴퓨팅 서버에서 데스크톱 컴퓨터 시스템, 핸드 헬드 장치에 이르기까지 다양한 병렬 CPU, GPU 및 DSP와 같은 다른 프로세서 및 Cell/B.E. 프로세서 등 다양한 이기종 프로세싱 플랫폼을 효과적으로 활용할 필요가 있습니다 .

OpenCL (Open Computing Language)은 CPU, GPU 및 기타 프로세서 전반에 걸친 범용 병렬 프로그래밍을위한 개방형 으로 로열티가없는 표준이다. 소프트웨어 개발자가 이러한 이기종 프로세싱 플랫폼의 성능에 이식 가능하고 효율적으로 액세스 할 수있게 해줍니다.

OpenCL는 임베디드 및 소비자 용 소프트웨어에서 HPC 솔루션에 이르는 다양한 애플리케이션을 저수준의 고성능 휴대용 추상화를 통해 지원합니다. 효율적인 close-to-the-metal programming interface 를 생성함으로써 OpenCL은 플랫폼 독립적 인 도구, middleware 및 layer of a parallel computing ecosystem of platform-independent tools (응용 프로그램의 병렬 컴퓨팅 에코 시스템의 기초 계층)을 형성합니다.
OpenCL은 일반 병렬 계산 알고리즘과 그래픽 렌더링 파이프 라인을 결합한 새로운 대화 형 그래픽 응용 프로그램에서 점점 더 중요한 역할을 수행하는 데 특히 적합합니다.

OpenCL은 이기종 프로세서에서 병렬 계산을 조정하는 API로 구성됩니다. 잘 알려진 계산 환경을 갖춘 교차 플랫폼 프로그래밍 언어입니다. OpenCL 표준 :

○데이터 및 작업 기반 병렬 프로그래밍 모델 모두 지원
○병렬 처리를위한 확장과 함께 ISO C99의 하위 집합을 활용합니다.
○IEEE 754를 기반으로 일관된 수치 요구 사항을 정의합니다.
○핸드 헬드 및 임베디드 장치에 대한 구성 프로파일을 정의합니다.
○OpenGL, OpenGL ES 및 기타 그래픽 API와의 효율적인 상호 운용

이 문서는 OpenCL의 기본 개념과 아키텍처에 대한 개요와 그 실행 모델, 메모리 모델 및 동기화 지원에 대한 자세한 설명으로 시작됩니다. 그런 다음 OpenCL 플랫폼 및 런타임 API에 대해 설명하고 OpenCL C 프로그래밍 언어에 대한 자세한 설명이 이어집니다.

OpenCL에서 작성된 예제 계산 사례 및 사례를 설명하는 몇 가지 예가 제공됩니다. 사양은 OpenCL 호환 구현이 지원해야하는 핵심 사양으로 나뉩니다. 핸드 헬드 및 임베디드 장치에 대한 OpenCL 컴플라이언스 요구 사항을 완화하는 핸드 헬드 / 임베디드 프로파일 OpenCL 사양의 이후 개정에서 핵심 사양으로 이동할 가능성이있는 옵션 확장 세트를 제공합니다.

2. Glossary

Application : 호스트에서 실행중인 프로그램과 OpenCL 장치의 조합입니다.

Blocking and Non-Blocking Enqueue API calls: 비 블로킹 enqueue API 호출은 명령 대기열에 명령을 놓고 즉시 호스트로 리턴합니다. 블로킹 모드 enqueue API 호출은 명령이 완료 될 때까지 호스트로 돌아 가지 않습니다.

Barrier: 명령 대기열 장벽과 작업 그룹 장벽이라는 두 가지 유형의 장벽이 있습니다.

○ OpenCL API는 명령 대기열 장벽 명령을 대기열에 넣는 기능을 제공합니다.
barrier 명령은 명령 대기열안에 있는 대기열에 포함 된 모든 명령이 실행을 시작하기 전에 명령 대기열에 대해 이전 대기열에 있던 모든 명령이 실행을 완료했는지 확인합니다.

○OpenCL C 프로그래밍 언어는 내장 된 작업 그룹 장벽 기능을 제공합니다.
이 장벽 내장 함수는 장치에서 실행중인 커널이 커널을 실행하는 작업 그룹에서 작업 항목 간의 동기화를 수행하는 데 사용할 수 있습니다. 작업 그룹의 모든 작업 항목은 장벽을 넘어서 실행을 계속하기 전에 장벽 구조를 실행해야합니다.

Buffer Object: 선형 바이트 집합을 저장하는 메모리 객체입니다. 버퍼 객체는 장치에서 실행중인 커널의 포인터를 사용하여 액세스 할 수 있습니다. 버퍼 객체는 OpenCL API 호출을 사용하여 호스트에서 조작 할 수 있습니다. 버퍼 객체는 다음 정보를 캡슐화합니다.

○크기 (바이트).
○사용 정보 및 할당 할 영역을 설명하는 등록 정보.
○버퍼 데이터.

Command : 실행을 위해 명령 대기열에 제출되는 OpenCL 작업입니다. 예를 들어 OpenCL 명령은 컴퓨팅 장치에서 실행하기 위해 커널을 실행하고 메모리 개체를 조작합니다.

Command-queue : 특정 장치에서 실행될 명령을 보유하고있는 개체.

명령 대기열은 컨텍스트의 특정 장치에 작성됩니다. 명령 대기열에 대한 명령은 순서대로 대기열에 있지만 순서대로 또는 순서없이 실행될 수 있습니다. In-order Execution 및 Out-of-Order 실행을 참조하십시오.

Command-queue Barrier. 배리어를 참조하십시오.

Compute Device Memory : 컴퓨팅 장치에 연결된 하나 이상의 메모리를 나타냅니다.

Compute Unit: OpenCL 장치에는 하나 이상의 계산 단위가 있습니다. 작업 그룹은 단일 계산 단위에서 실행됩니다. 계산 단위는 하나 이상의 처리 요소와 로컬 메모리로 구성됩니다. 컴퓨팅 유닛은 또한 프로세싱 요소에 의해 액세스 될 수있는 전용 텍스처 필터 유닛을 포함 할 수있다.

동시성 (Concurrency) : 시스템 한 세트의 작업이 활성 상태를 유지하고 동시에 진행될 수있는 시스템의 속성. 프로그래머는 프로그램을 실행할 때 동시 실행을 활용하려면 문제의 동시성을 식별하고 소스 코드 내에서 노출 한 다음 동시성을 지원하는 표기법을 사용해야합니다.

상수 메모리 (Constant Memory) : 전역 메모리의 한 영역으로, 커널을 실행하는 동안 일정하게 유지됩니다. 호스트는 상수 메모리에 배치 된 메모리 객체를 할당하고 초기화합니다.

Context: 커널이 실행되는 환경과 동기화 및 메모리 관리가 정의되는 도메인입니다. 컨텍스트에는 장치 세트, 해당 장치에 액세스 할 수있는 메모리, 해당 메모리 속성 및 커널 실행 또는 메모리 개체 작업을 예약하는 데 사용되는 하나 이상의 명령 대기열이 포함됩니다.

Data Parallel Programming Model:전통적으로 이 용어는 데이터 구조 집합 내의 여러 요소에 적용되는 단일 프로그램의 명령으로 동시성을 표현하는 프로그래밍 모델을 나타냅니다. 이 용어는 OpenCL에서 단일 프로그램 명령어 집합이 인덱스의 추상 도메인 내의 각 지점에 동시에 적용되는 모델을 나타 내기 위해 일반화되었습니다.

Device: 장치는 계산 단위 모음입니다. 명령 대기열은 장치에 명령을 대기 행렬에 넣는 데 사용됩니다. 명령의 예로는 커널 실행 또는 메모리 오브젝트 읽기 및 쓰기가 있습니다. OpenCL 장치는 일반적으로 GPU, 멀티 코어 CPU 및 DSP와 같은 다른 프로세서 및 Cell/B.E. 프로세서에 해당합니다.

Event Object: 이벤트 객체는 명령과 같은 작업의 상태를 캡슐화합니다. 컨텍스트에서 작업을 동기화하는 데 사용할 수 있습니다.

Event Wait List: 이벤트 대기 목록은 특정 명령 실행이 시작될 때를 제어하는 데 사용할 수있는 이벤트 개체의 목록입니다.

프레임 워크 (Framework) : 소프트웨어 개발 및 실행을 지원하는 컴포넌트 세트를 포함하는 소프트웨어 시스템. 프레임 워크는 일반적으로 라이브러리, API, 런타임 시스템, 컴파일러 등을 포함합니다.

Global ID: 글로벌 ID는 작업 항목을 고유하게 식별하는 데 사용되며 커널을 실행할 때 지정된 전역 작업 항목 수에서 파생됩니다. 글로벌 ID는 (0, 0, ... 0)에서 시작하는 N 차원 값입니다. 지역 ID를 참조하십시오.

Global Memory: 컨텍스트에서 실행되는 모든 작업 항목에 액세스 할 수있는 메모리 영역입니다. 읽기, 쓰기 및 맵과 같은 명령을 사용하여 호스트에 액세스 할 수 있습니다.

Handle: OpenCL에서 할당 한 객체를 참조하는 불투명 유형입니다. 객체에 대한 모든 작업은 해당 객체의 핸들을 참조하여 발생합니다.

Host: 호스트는 OpenCL API를 사용하여 컨텍스트와 상호 작용합니다.

Host pointer: 호스트의 가상 주소 공간에있는 메모리에 대한 포인터입니다.

Illegal : 명시 적으로 허용되지 않고 OpenCL에서 오류로보고되는 시스템의 동작입니다.

Image Object: 2 차원 또는 3 차원 구조 배열을 저장하는 메모리 객체입니다.
이미지 데이터는 읽기 및 쓰기 기능으로 만 액세스 할 수 있습니다. 읽기 함수는 샘플러를 사용합니다.

image 객체는 다음 정보를 캡슐화합니다.
○이미지의 크기.
○이미지의 각 요소에 대한 설명입니다.
○사용 정보 및 할당 할 영역을 설명하는 등록 정보.
○이미지 데이터.

이미지의 요소는 사전 정의 된 이미지 형식 목록에서 선택됩니다.

Implementation Defined: OpenCL의 준수 구현간에 명시 적으로 허용되는 동작입니다. OpenCL 구현자는 구현 정의 동작을 문서화해야합니다.

In-order Execution: 명령 대기열의 명령이 제출 순서대로 실행되어 각 명령이 완료되기 전에 실행되어 다음 명령이 시작되는 OpenCL 실행 모델입니다. out-of-order 실행을 참조하십시오.
Kernel: 커널은 프로그램에서 선언되고 OpenCL 장치에서 실행되는 함수입니다. 커널은 프로그램에 정의 된 모든 함수에 적용되는 __kernel 한정자로 식별됩니다.

Kernel Object: 커널 객체는 프로그램에서 선언 된 특정 __kernel 함수와 __kernel 함수를 실행할 때 사용할 인수 값을 캡슐화합니다

Local ID: 로컬 ID는 커널을 실행중인 주어진 작업 그룹 내에서 고유 한 작업 항목 ID를 지정합니다. 지역 ID는 (0, 0, ... 0)에서 시작하는 N 차원 값입니다. 글로벌 ID를 참조하십시오.

Local Memory: 작업 그룹과 연관되고 해당 작업 그룹의 작업 항목 만 액세스 할 수있는 메모리 영역입니다.

마커 (Marker) : 명령 대기열에 대기중인 명령. 명령 대기열의 마커 앞에 대기중인 모든 명령에 태그를다는 데 사용할 수 있습니다. marker 명령은 응용 프로그램이 마커 이벤트에 대한 대기를 대기시키기 위해 사용할 수있는 이벤트를 반환합니다. 즉 마커 명령이 완료되기 전에 대기 된 모든 명령을 기다립니다.

Memory Objects: 메모리 객체는 전역 메모리의 참조 카운트 영역에 대한 핸들입니다. 또한 버퍼 객체 및 이미지 객체를 참조하십시오.

Memory Regions (or Pools): OpenCL의 고유 한 주소 공간. OpenCL은 논리적으로 구별되는 것으로 취급하지만 메모리 영역은 실제 메모리에서 겹칠 수 있습니다. 메모리 영역은 private, local, constant 및 global로 표시됩니다.

Object: 객체는 OpenCL API에 의해 조작 될 수있는 자원의 추상 표현입니다. 예제에는 프로그램 개체, 커널 개체 및 메모리 개체가 포함됩니다.

Out-of-Order Execution : 작업 큐에 배치 된 명령이 이벤트 대기 목록 및 명령 대기열 장벽에 의해 부과 된 제약 조건과 일치하는 순서로 실행을 시작하고 완료 할 수있는 실행 모델입니다. 순서 실행을 참조하십시오.

Platform: 호스트에 OpenCL 프레임 워크가 관리하는 장치 모음이 포함되어있어 응용 프로그램이 자원을 공유하고 플랫폼의 장치에서 커널을 실행할 수 있습니다.

Private Memory: 작업 항목 전용 메모리 영역. 한 작업 항목의 개인 메모리에 정의 된 변수는 다른 작업 항목에 표시되지 않습니다.

Processing Element: 가상 스칼라 프로세서. 작업 항목은 하나 이상의 처리 요소에서 실행될 수 있습니다.

Program: OpenCL 프로그램은 일련의 커널로 구성됩니다. 프로그램에는 __kernel 함수 및 상수 데이터에 의해 호출되는 보조 함수가 포함될 수도 있습니다

Program Object: 프로그램 개체는 다음 정보를 캡슐화합니다.

○연관된 컨텍스트에 대한 참조입니다.

○프로그램 소스 또는 바이너리.

○최신으로 성공적으로 빌드 된 프로그램 실행 파일, 프로그램 실행 파일이 빌드 된 장치 목록, 사용 된 빌드 옵션 및 빌드 로그.

○현재 접속 된 커널 오브젝트의 수.

Reference Count: OpenCL 개체의 수명은 참조 수 (개체에 대한 참조 수의 내부 수)에 의해 결정됩니다. OpenCL에서 객체를 만들면 참조 카운트가 1로 설정됩니다.
이후에 적절한 retain API (예 : clRetainContext,clRetainCommandQueue)를 호출하면 참조 횟수가 증가합니다. 마지막 개정 날짜로의 호출 : 6/1/11 Page 18 적절한 릴리스 API (예 : clReleaseContext, clReleaseCommandQueue)는 참조 횟수를 감소시킵니다. 참조 횟수가 0이되면 개체의 리소스는 OpenCL에 의해 할당이 해제됩니다.

Relaxed Consistency (Relaxed Consistency) : 장벽이나 다른 명시적인 동기화 지점을 제외하고는 다른 작업 항목이나 명령에서 볼 수있는 메모리의 내용이 다를 수있는 메모리 일관성 모델입니다.

Resource: OpenCL에 의해 정의 된 객체의 클래스. 자원의 인스턴스는 객체입니다. 가장 일반적인 리소스는 컨텍스트, 명령 대기열, 프로그램 개체, 커널 개체 및 메모리 개체입니다. 전산 자원은 프로그램 카운터를 발전시키는 활동에 참여하는 하드웨어 요소입니다. 예를 들면 호스트, 장치, 계산 단위 및 처리 요소가 포함됩니다.

Retain, Release : OpenCL 객체를 사용하여 참조 카운트를 증가 (유지) 및 감소 (해제)하는 동작입니다. 이것은이 객체를 사용하는 모든 인스턴스가 완료되기 전에 시스템이 객체를 제거하지 않도록하는 기능을 유지하는 책입니다. 참조 횟수를 참조하십시오.

샘플러 (Sampler) : 커널에서 이미지를 읽을 때 이미지를 샘플링하는 방법을 설명하는 객체. 이미지 읽기 함수는 샘플러를 인수로 사용합니다. 샘플러는, 이미지 주소 지정 모드, 즉 범위 외의 이미지 좌표의 처리 방법, 필터 모드, 및 입력 이미지 좌표가 정규화 또는 비 표준화 된 값일지 어떨지를 지정합니다.

SIMD : 단일 명령어 다중 데이터. 커널이 각각 자체 데이터와 공유 프로그램 카운터를 가진 다중 처리 요소에서 동시에 실행되는 프로그래밍 모델. 모든 처리 요소는 엄격하게 동일한 명령어 세트를 실행합니다.

SPMD : 단일 프로그램 다중 데이터. 커널이 각각 자체 데이터와 자체 프로그램 카운터를 가진 다중 처리 요소에서 동시에 실행되는 프로그래밍 모델. 따라서 모든 계산 리소스가 동일한 커널을 실행하지만 자신의 명령 카운터를 유지하고 커널의 분기로 인해 실제 명령 시퀀스는 처리 요소 세트 전체에서 상당히 다를 수 있습니다.

작업 병렬 프로그래밍 모델 (Task Parallel Programming Model) : 작업이 하나의 작업 그룹 (크기가 하나 인 그룹)에서 실행되는 여러 동시 작업의 관점에서 계산이 표시되는 프로그래밍 모델입니다. 동시 작업은 다른 커널을 실행할 수 있습니다.

Thread-safe: OpenCL API 호출은 OpenCL에 의해 관리되는 내부 상태가 여러 호스트 스레드에 의해 동시에 호출 될 때 일관성이 유지되면 스레드로부터 안전하다고 간주됩니다. 스레드로부터 안전한 OpenCL API 호출은 응용 프로그램이 여러 호스트 스레드에서 이러한 함수를 호출 할 수있게합니다. 즉, 호스트 스레드간에 상호 배제를 구현할 필요가 없으므로 다시 침입자가 안전합니다. 최종 수정 날짜 : 2011 년 6 월 1 일 Page 19

Undefined : OpenCL API 호출의 동작, OpenCL에서 명시 적으로 정의하지 않은 커널 내에서 사용되는 기본 제공 함수 또는 커널 실행. 부적합한 구현체는 OpenCL에서 정의되지 않은 구조체가 발생할 때 발생하는 것을 지정하지 않아도됩니다.

Work-group: 단일 계산 단위에서 실행되는 관련 작업 항목 모음입니다. 그룹의 작업 항목은 동일한 커널을 실행하고 로컬 메모리 및 작업 그룹 장벽을 공유합니다.

Work-group Barrier. 배리어를 참조하십시오.

작업 항목 (Work-item) : 명령에 의해 장치에서 호출 된 커널의 병렬 실행 모음 중 하나입니다. 작업 항목은 계산 단위에서 실행되는 작업 그룹의 일부로 하나 이상의 처리 요소에 의해 실행됩니다. 작업 항목은 해당 글로벌 ID 및 로컬 ID로 컬렉션 내의 다른 실행과 구별됩니다.

2.1 OpenCL Class Diagram

그림 2.1은 OpenCL 사양을 Unified Modeling Language1 (UML) 표기법을 사용하는 클래스 다이어그램으로 설명합니다. 이 다이어그램은 클래스와 그 관계 인 노드와 에지를 보여줍니다. 단순화로서 클래스 만 표시하고 속성이나 조작은 표시하지 않습니다. 추상 클래스에는 "{abstract}"주석이 붙습니다. 관계에 관해서는 집계 (솔리드 다이아몬드로 주석 처리), 연관성 (주석 없음) 및 상속 (열린 화살촉으로 주석 처리 됨)을 보여줍니다. 관계의 카디널리티가 각 끝에 표시됩니다. 카디널리티가 "*"이면 "많음"을 나타내며 "1"의 카디널리티는 "하나만"을 나타내며 "0.1"의 ardinality는 "선택적으로 하나"를 나타내며 "1 .. *"의 카디널리티를 나타냅니다. "하나 이상"을 나타냅니다. 관계의 탐색 가능성은 규칙적인 화살촉을 사용하여 표시됩니다.

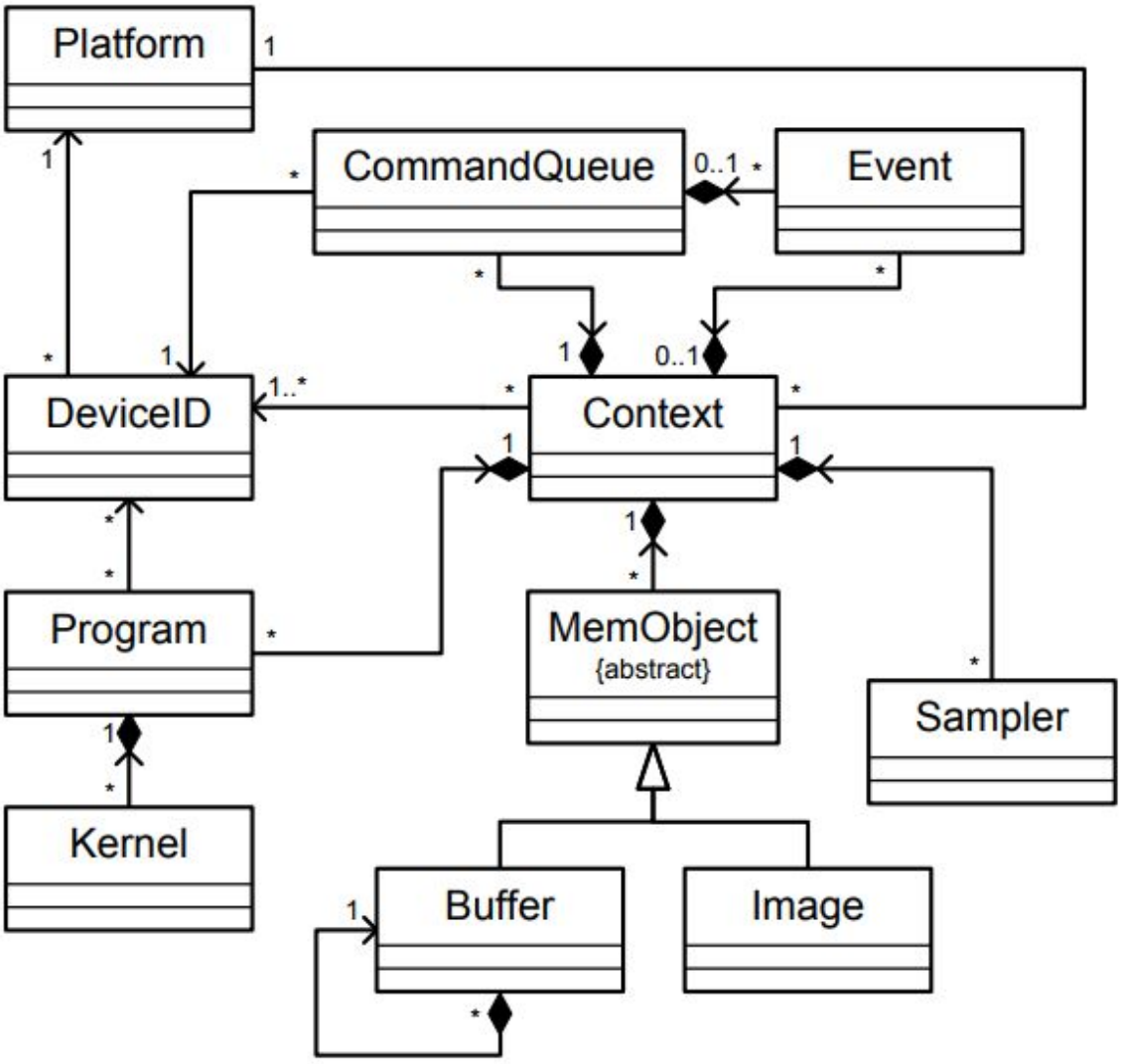

Figure 2.1 - *OpenCL UML Class Diagram*

3. The OpenCL Architecture

OpenCL은 단일 플랫폼으로 구성된 CPU, GPU 및 기타 개별 컴퓨팅 장치의 이기종 컬렉션을 프로그래밍하기위한 개방형 업계 표준입니다. 그것은 하나의 언어 이상입니다.

OpenCL은 병렬 프로그래밍을위한 프레임 워크이며 소프트웨어 개발을 지원하는 언어, API, 라이브러리 및 런타임 시스템을 포함합니다. 예를 들어 OpenCL을 사용하면 프로그래머는 OpenGL이나 DirectX와 같은 3D 그래픽 API에 알고리즘을 매핑하지 않고도 GPU에서 실행되는 범용 프로그램을 작성할 수 있습니다.

OpenCL의 목표는 이식성 있고 효율적인 코드를 작성하고자하는 전문 프로그래머입니다. 여기에는 라이브러리 작성자, 미들웨어 공급 업체 및 성능 지향 응용 프로그램 프로그래머가 포함됩니다. 따라서 OpenCL은 프로그래밍을 지원하는 프레임 워크와 기본 하드웨어의 많은 세부 사항을 드러내는 저수준 하드웨어 추상화를 제공합니다.

OpenCL의 핵심 아이디어를 설명하기 위해 다음과 같은 계층 구조를 사용합니다.
○플랫폼 모델
○메모리 모델
○실행 모델
○프로그래밍 모델

3.1 Platform Model

OpenCL 용 플랫폼 모델은 그림 3.1에 정의되어 있습니다. 이 모델은 하나 이상의 OpenCL 장치에 연결된 호스트로 구성됩니다. OpenCL 장치는 하나 이상의 처리 장치 (CU)로 나뉘며 하나 이상의 처리 요소 (PE)로 나뉩니다.
장치의 계산은 처리 요소 내에서 발생합니다.
OpenCL 응용 프로그램은 호스트 플랫폼 고유의 모델에 따라 호스트에서 실행됩니다.
OpenCL 응용 프로그램은 호스트의 명령을 제출하여 장치 내의 처리 요소에 대한 계산을 실행합니다. 연산 장치 내의 처리 요소는 단일 명령 스트림을 SIMD 단위 (단일 명령 스트림으로 잠금 단계로 실행) 또는 SPMD 단위 (각 PE는 자체 프로그램 카운터를 유지 관리)로 실행합니다.

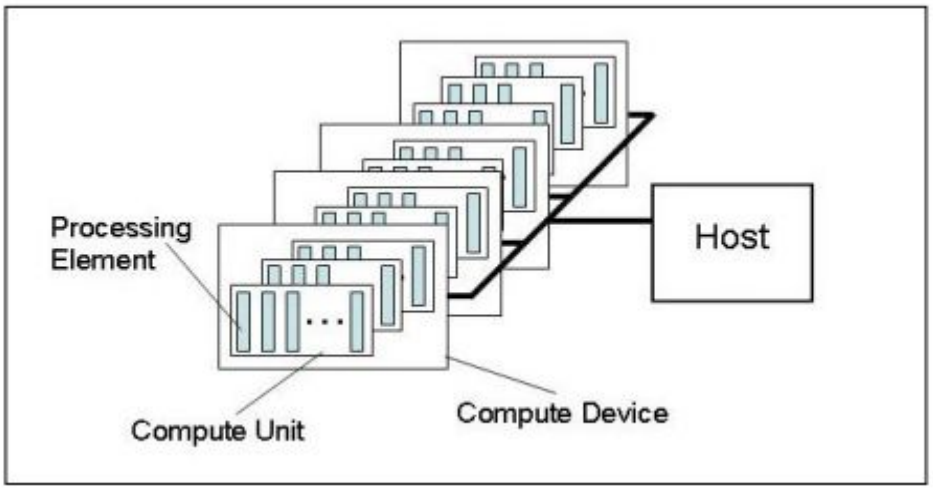

그림 3.1 : 플랫폼 모델 ... 하나의 호스트와 하나 이상의 컴퓨팅 장치 각각에 하나 이상의 처리 요소가있는 하나 이상의 계산 단위가 있습니다.

3.1.1 Platform Mixed Version Support(플랫폼 혼합 버전 지원)

OpenCL은 단일 플랫폼에서 다른 기능을 가진 장치를 지원하도록 설계되었습니다. 여기에는 다른 버전의 OpenCL 사양을 준수하는 장치가 포함됩니다. OpenCL 시스템에는 플랫폼 버전, 장치 버전 및 장치에서 지원되는 OpenCL C 언어의 버전 등 세 가지 중요한 버전 식별자가 고려됩니다.

플랫폼 버전은 지원되는 OpenCL 런타임의 버전을 나타냅니다. 여기에는 호스트가 컨텍스트, 메모리 개체, 장치 및 명령 대기열과 같은 OpenCL 런타임과 상호 작용하는 데 사용할 수있는 모든 API가 포함됩니다.

장치 버전은 clGetDeviceInfo에서 반환 한 장치 정보로 표시되는 런타임 및 컴파일러와 별도로 장치 기능을 나타냅니다. 장치 버전과 관련된 속성의 예는 자원 제한 및 확장 기능입니다. 반환 된 버전은 장치가 준수하지만 플랫폼 버전보다 높은 OpenCL 사양의 가장 높은 버전에 해당합니다

장치의 언어 버전은 개발자가 특정 장치에서 지원되는 것으로 가정 할 수있는 OpenCL 프로그래밍 언어 기능을 나타냅니다. 보고 된 버전이 지원되는 언어의 가장 높은 버전입니다.

OpenCL C는 이전 버전과의 호환성을 위해 설계되었으므로 장치가 단일 언어 버전 이상을 지원하지 않아도됩니다. 여러 언어 버전이 지원되는 경우 컴파일러는 기본적으로 장치에 지원되는 가장 높은 언어 버전을 사용합니다.
언어 버전은 플랫폼 버전보다 높지 않지만 장치 버전을 초과 할 수 있습니다 (5.6.3.5 절 참조).

3.2 Execution Model

OpenCL 프로그램의 실행은 하나 이상의 OpenCL 장치에서 실행되는 커널과 호스트에서 실행되는 호스트 프로그램의 두 부분으로 이루어집니다. 호스트 프로그램은 커널의 컨텍스트를 정의하고 실행을 관리합니다.

OpenCL 실행 모델의 핵심은 커널이 실행되는 방식에 따라 정의됩니다. 호스트가 실행을 위해 커널을 제출하면 색인 공간이 정의됩니다. 커널의 인스턴스는이 인덱스 공간의 각 포인트에 대해 실행됩니다. 이 커널 인스턴스는 작업 항목이라고하며 작업 공간 항목의 글로벌 ID를 제공하는 인덱스 공간의 해당 지점으로 식별됩니다. 각 작업 항목은 동일한 코드를 실행하지만 코드 및 작업중인 데이터를 통한 특정 실행 경로는 작업 항목마다 다를 수 있습니다.

작업 항목은 작업 그룹으로 구성됩니다. 작업 그룹은 인덱스 공간을 좀 더 거칠게 분해합니다. 작업 그룹에는 작업 항목에 사용 된 색인 공간과 동일한 차원으로 고유 한 작업 그룹 ID가 지정됩니다. 작업 항목에는 작업 그룹 내에서 고유 한 로컬 ID가 할당되므로 단일 작업 항목이 글로벌 ID 또는 로컬 ID와 작업 그룹 ID의 조합으로 고유하게 식별 될 수 있습니다. 주어진 작업 그룹의 작업 항목은 단일 계산 단위의 처리 요소에서 동시에 실행됩니다.

OpenCL에서 지원되는 인덱스 공간을 NDRange라고합니다. NDRange는 N 차원 색인 공간이며 N은 1, 2 또는 3입니다. NDRange는 오프셋 인덱스 F (기본값은 0)에서 시작하는 각 차원에서 인덱스 공간의 범위를 지정하는 길이 N의 정수 배열로 정의됩니다. 각 작업 항목의 전역 ID와 로컬 ID는 N 차원 튜플입니다. 전역 ID 구성 요소는 F에서 F까지의 범위에있는 값과 해당 차원의 요소 수에서 1을 뺀 값입니다.

작업 그룹에는 작업 항목 글로벌 ID에 사용되는 것과 유사한 접근법을 사용하여 ID가 할당됩니다.
길이 N의 h 열은 각 차원의 작업 그룹 수를 정의합니다. 작업 항목은 작업 그룹에 할당되고 0부터 해당 차원의 작업 그룹 크기를 뺀 범위의 구성 요소가있는 로컬 ID가 지정됩니다.
그러므로 작업 그룹 ID와 작업 그룹 내의 로컬 ID의 조합은 작업 항목을 고유하게 정의합니다.
글로벌 인덱스의 관점에서, 그리고 작업 그룹 인덱스와 작업 그룹 내의 로컬 인덱스의 관점에서 볼 때, 각 작업 항목은 두 가지 방법으로 식별 할 수 있습니다.

예를 들어 그림 3.2의 2 차원 색인 공간을 고려하십시오. 우리는 작업 항목 (Gx, Gy), 각 작업 그룹 (Sx, Sy)의 크기 및 글로벌 ID 오프셋 (Fx, Fy)에 대한 색인 공간을 입력한다.

글로벌 인덱스는 Gx 인덱스 공간에서 Gx를 정의합니다. 여기서 작업 항목의 총 수는 Gx와 Gy의 곱입니다. 로컬 인덱스는 단일 작업 그룹의 작업 항목 수가 Sx와 Sy의 곱인 Sy 인덱스 공간에서 Sx를 정의합니다. 각 작업 그룹의 크기와 전체 작업 항목 수를 감안할 때 작업 그룹 수를 계산할 수 있습니다.

2 차원 색인 공간은 작업 그룹을 고유하게 식별하는 데 사용됩니다. 각 작업 항목은 글로벌 ID (gx, gy) 또는 작업 그룹 ID (wx, wy), 각 작업 그룹 (Sx, Sy)의 크기 및 로컬 ID (sx, 작업 그룹의 수는 다음과 같이 계산할 수 있습니다.
(gx, gy) = (wx * Sx + sx + Fx, wy * Sy + sy + Fy)
작업 그룹 수는 다음과 같이 계산할 수 있습니다.

(Wx, Wy) = (Gx / Sx, Gy / Sy)
전역 ID와 작업 그룹 크기가 주어지면 작업 항목의 작업 그룹 ID는 다음과 같이 계산됩니다.
(wx, wy) = ((gx – sx – Fx) / Sx, (gy – sy – Fy) / Sy)

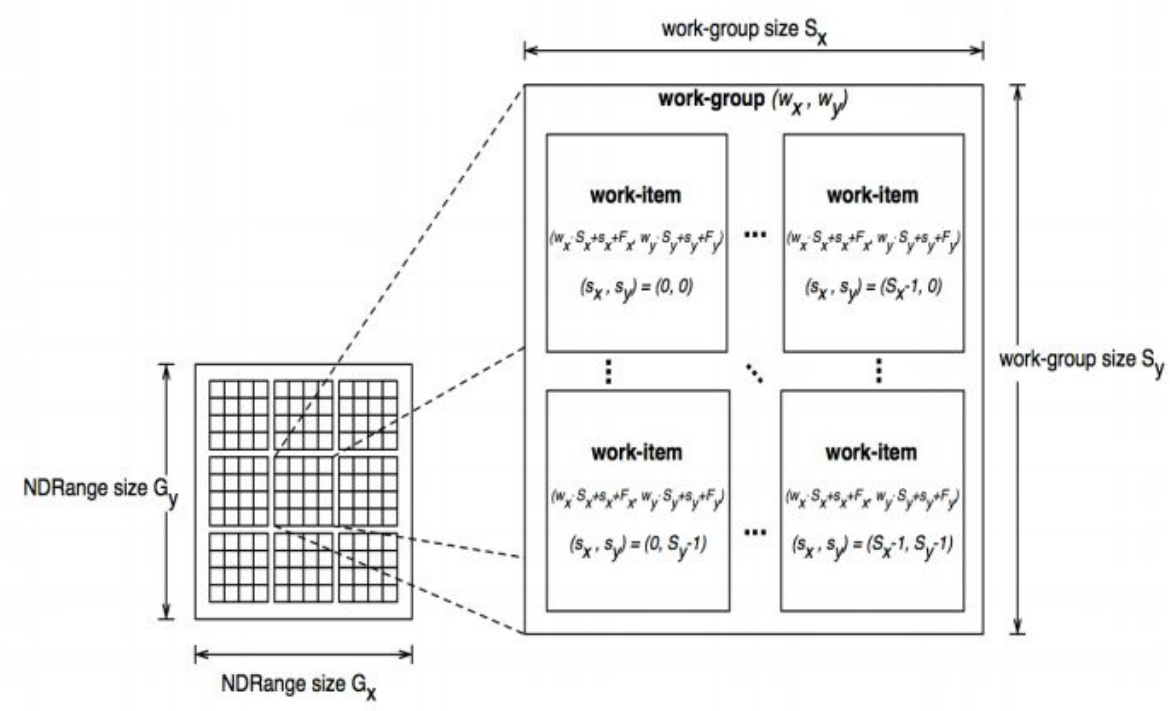

그림 3.2 작업 항목, 전역 ID 및 작업 그룹 및 로컬 ID 쌍에 매핑하는 NDRange 인덱스 공간의 예.

광범위한 실행 모델을이 실행 모델에 매핑 할 수 있습니다. 우리는 OpenCL 내에서이 두 모델을 명시 적으로 지원합니다. 데이터 병렬 프로그래밍 모델 및 작업 병렬 프로그래밍 모델.

3.2.1 Execution Model: Context and Command Queues

호스트는 커널 실행을위한 컨텍스트를 정의합니다. 컨텍스트에는 다음 리소스가 포함됩니다.
1. Devices: 호스트에서 사용할 OpenCL 장치 모음입니다.
2. Kernels: OpenCL 장치에서 실행되는 OpenCL 함수.
3. 프로그램 객체 (Program Objects) : 커널을 구현하는 프로그램 소스와 실행 파일.
4. 메모리 객체 (Memory Objects) : 호스트와 OpenCL 디바이스가 볼 수있는 일련의 메모리 객체. 메모리 객체는 커널의 인스턴스가 조작 할 수있는 값을 포함합니다.

컨텍스트는 OpenCL API의 함수를 사용하여 호스트가 만들고 조작합니다. 호스트는 커맨드 - 큐 (command-queue) 라 불리는 데이터 구조를 생성하여 디바이스상의 커널의 실행을 조정한다. 호스트는 커맨드 큐에 커맨드를 배치하고, 컨텍스트 내의 디바이스 상에 스케줄링한다. 여기에는 다음이 포함됩니다.

○Kernel execution commands : 장치의 처리 요소에서 커널을 실행합니다.
○Memory commands : 메모리 개체간에 데이터를 전송하거나 호스트 개체의 메모리 개체를 매핑 및 매핑 해제합니다.
○Synchronization commands : 명령 실행 순서를 제한합니다.

명령 대기열은 장치에서 실행하기위한 명령을 예약합니다. 이들은 호스트와 장치 사이에서 비동기 적으로 실행됩니다. 명령은 두 가지 모드 중 하나에서 서로에 대해 실행됩니다.

○In-order Execution : 명령은 명령 대기열에 나타나는 순서대로 시작되고 순서대로 완료됩니다. 즉, 다음 명령이 시작되기 전에 대기열의 이전 명령이 완료됩니다. 이것은 큐에있는 명령의 실행 순서를 직렬화합니다.

○Out-of-order Execution : 명령은 순서대로 실행되지만 명령을 실행하기 전에 완료 될 때까지 기다리지 마십시오. 모든 주문 제약 조건은 명시적인 동기화 명령을 통해 프로그래머에 의해 시행됩니다.

대기열에 제출 된 커널 실행 및 메모리 명령은 이벤트 객체를 생성합니다. 이것들은 명령 들간의 실행을 제어하고 호스트와 장치 사이의 실행을 조정하는데 사용됩니다.
여러 대기열을 단일 컨텍스트와 연관시킬 수 있습니다. 이러한 큐는 OpenCL 내에서 명시 적 메커니즘없이 동시 적으로 독립적으로 실행되어 이들을 동기화합니다.

3.2.2 Execution Model: Categories of Kernels

OpenCL 실행 모델은 커널의 두 가지 범주를 지원합니다.

○**OpenCL 커널**은 OpenCL C 프로그래밍 언어로 작성되고 OpenCL 컴파일러로 컴파일됩니다. 모든 OpenCL 구현은 OpenCL 커널을 지원합니다. 구현은 OpenCL 커널을 만드는 다른 메커니즘을 제공 할 수 있습니다.

○**Native kernels**은 호스트 함수 포인터를 통해 액세스됩니다. 네이티브 커널은 장치의 OpenCL 커널과 함께 실행을 위해 대기하고 OpenCL 커널과 메모리 개체를 공유합니다. 예를 들어, 이러한 원시 커널은 응용 프로그램 코드에 정의 된 함수이거나 라이브러리에서 내 보낸 함수 일 수 있습니다. 네이티브 커널을 실행하는 기능은 OpenCL의 선택적 기능이며 네이티브 커널의 의미는 구현에 따라 정의됩니다. OpenCL API에는 장치의 기능을 쿼리하고이 기능이 지원되는지 여부를 결정하는 기능이 있습니다.

3.3 Memory Model

커널을 실행하는 작업 항목은 4 개의 개별 메모리 영역에 액세스 할 수 있습니다.

○**글로벌 메모리**. 이 메모리 영역은 모든 작업 그룹의 모든 작업 항목에 대한 읽기 / 쓰기 액세스를 허용합니다. 작업 항목은 메모리 개체의 모든 요소를 읽고 쓸 수 있습니다. 전역 메모리에 대한 읽기 및 쓰기는 장치의 기능에 따라 캐시 될 수 있습니다.

○**상수 메모리 (Constant Memory)** : 전역 메모리의 한 영역으로, 커널을 실행하는 동안 일정하게 유지됩니다. 호스트는 상수 메모리에 배치 된 메모리 객체를 할당하고 초기화합니다.

○**Local Memory** : 작업 그룹의 로컬 메모리 영역입니다. 이 메모리 영역은 해당 작업 그룹의 모든 작업 항목이 공유하는 변수를 할당하는 데 사용할 수 있습니다. OpenCL 장치의 전용 메모리 영역으로 구현 될 수 있습니다. 대안으로, 로컬 메모리 영역은 글로벌 메모리의 섹션들 상에 매핑 될 수있다.

○**Private Memory** : 작업 항목 전용 메모리 영역. 한 작업 항목의 개인 메모리에 정의 된 변수는 다른 작업 항목에 표시되지 않습니다.

표 3.1은 커널 또는 호스트가 메모리 영역, 할당 유형 (정적 즉 컴파일 시간 대 동적 즉 런타임)과 허용되는 액세스 유형 즉 커널 또는 호스트가 읽을 수 있는지 또는 쓰는지 여부를 할당 할 수 있는지 여부를 설명합니다 메모리 영역.

	Global	**Constant**	**Local**	**Private**
Host	Dynamic allocation Read / Write access	Dynamic allocation Read / Write access	Dynamic allocation No access	No allocation No access
Kernel	No allocation Read / Write access	Static allocation Read-only access	Static allocation Read / Write access	Static allocation Read / Write access

표 3.1 메모리 영역 - 할당 및 메모리 액세스 기능

메모리 영역과 플랫폼 모델과의 관계는 그림 3.3에 설명되어 있습니다.

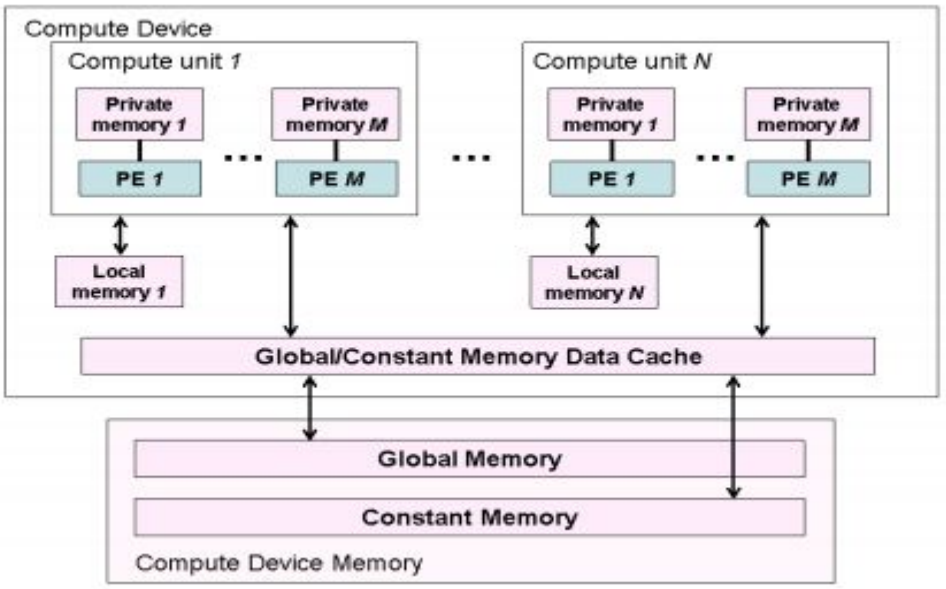

그림 3.3 : 처리 요소 (PE), 연산 장치 및 장치를 사용한 개념적 OpenCL 장치 아키텍처. 호스트가 표시되지 않습니다.

호스트에서 실행되는 응용 프로그램은 OpenCL API를 사용하여 전역 메모리에 메모리 객체를 만들고 이러한 메모리 객체에서 작동하는 메모리 명령 (3.2.1 절에서 설명 함)을 대기열에 추가합니다.

호스트 및 OpenCL 장치 메모리 모델은 대부분 서로 독립적입니다.
이것은 호스트가 OpenCL 외부에서 정의된다는 점에서 필수적입니다. 그러나 때때로 그들은 상호 작용할 필요가 있습니다. 이 상호 작용은 명시 적으로 데이터를 복사하거나 메모리 객체의 영역을 매핑 및 매핑 해제하는 두 가지 방법 중 하나로 발생합니다.

데이터를 명시 적으로 복사하기 위해 호스트는 메모리 개체와 호스트 메모리간에 데이터를 전송하는 명령을 대기열에 넣습니다. 이러한 메모리 전송 명령은 차단 또는 비 차단 일 수 있습니다.
호스트의 연결된 메모리 리소스를 안전하게 다시 사용할 수있게되면 차단 메모리 전송을위한 OpenCL 함수 호출이 반환됩니다. 논 블로킹 메모리 전송의 경우, OpenCL 함수 호출은 호스트 메모리의 사용 여부에 관계없이 명령이 대기열에 들어가자 마자 반환됩니다.

호스트와 OpenCL 메모리 객체 간의 상호 작용에 대한 매핑 / 매핑 해제 방법을 통해 호스트는 메모리 객체의 영역을 주소 공간에 매핑 할 수 있습니다. 메모리 맵 명령이 차단 또는 비 차단 일 수 있습니다. 메모리 객체의 영역이 매핑되면 호스트는이 영역을 읽거나 쓸 수 있습니다. 호스트는이 맵핑 된 region에 대한 액세스 (읽기 및 / 또는 쓰기)가 완료되면 region의 맵핑을 해제합니다.

3.3.1 Memory Consistency

OpenCL은 편안한 일관성 메모리 모델을 사용합니다. 즉, 작업 항목에 보이는 메모리의 상태가 항상 작업 항목의 모음을 통해 일관성을 유지한다고 보장 할 수는 없습니다.

work-item memory 에는로드 / 저장 일관성이 있습니다. 로컬 메모리는work-group barrier 에있는 single work-group 의 작업 항목에서 일관 적입니다. 전역 메모리는 work-group barrier 에서 single work-group 의 work-item간에 일관성이 있지만 커널을 실행하는 여러 작업 그룹간에 메모리 일관성을 보장 할 수는 없습니다. 대기열에 포함 된 명령간에 공유되는 메모리 객체의 메모리 일관성은 동기화 지점에서 시행됩니다.

3.4 Programming Model

OpenCL 실행 모델은 데이터 병렬 및 작업 병렬 프로그래밍 모델과이 두 모델의 하이브리드를 지원합니다. OpenCL의 설계를 주도하는 기본 모델은 데이터 병렬입니다.

3.4.1 Data Parallel Programming Model

데이터 병렬 프로그래밍 모델은 메모리 객체 다수의 엘리먼트에 적용되는 명령들 시퀀스 관점에서 계산을 정의한다. OpenCL 실행 모델과 연관된 인덱스 공간은 작업 항목과 데이터가 작업 항목에 매핑되는 방법을 정의합니다. 엄밀히 말하면 데이터 병렬 모델에서는 커널이 병렬로 실행될 수있는 메모리 개체의 작업 항목과 요소간에 일대일 매핑이 있습니다. OpenCL은 엄격한 일대일 매핑이 요구되지 않는 데이터 병렬 프로그래밍 모델의 느슨한 버전을 구현합니다.

OpenCL은 계층 적 데이터 병렬 프로그래밍 모델을 제공합니다. 계층 적 하위 구분을 지정하는 두 가지 방법이 있습니다. 명시 적 모델에서 프로그래머는 병렬로 실행할 작업 항목의 총 수와 작업 항목이 작업 그룹간에 어떻게 나누어 지는지 정의합니다. 암시 적 모델에서 프로그래머는 병렬로 실행할 작업 항목의 총 개수 만 지정하고 작업 그룹으로의 분할은 OpenCL 구현으로 관리합니다.

3.4.2 Task Parallel Programming Model

OpenCL 태스크 병렬 프로그래밍 모델은 커널의 단일 인스턴스가 인덱스 공간과 독립적으로 실행되는 모델을 정의합니다. 논리적으로는 하나의 작업 항목을 포함하는 작업 그룹이있는 계산 단위에서 커널을 실행하는 것과 같습니다. 이 모델에서 사용자는 다음과 같은 방법으로 병렬 처리를 표현합니다.

- 장치에 의해 구현 된 벡터 데이터 유형을 사용하여,
- 여러 작업 대기열에 넣기 및 / 또는
- OpenCL에 직교하는 프로그래밍 모델을 사용하여 개발 된 네이티브 커널을 대기열에 넣습니다.

3.4.3 Synchronization

OpenCL에는 두 가지 동기화 영역이 있습니다.
- a single work-group의 Work-items
- 단일 컨텍스트에서 명령 대기열에 대기중인 명령(Commands enqueued to command-queue(s) in a single context)
단일 작업 그룹의 작업 항목 간 동기화는 작업 그룹 장벽을 사용하여 수행됩니다. 작업 그룹의 모든 작업 항목은 장벽을 넘어서서 실행을 계속하기 전에 장벽을 실행해야합니다. 작업 그룹 장벽은 커널을 실행하는 작업 그룹의 모든 작업 항목이나 모든 작업 그룹에서 발생해야합니다. 작업 그룹 간의 동기화를위한 메커니즘은 없습니다.

명령 대기열의 명령 간 동기화 지점은 다음과 같습니다.

- Command-queue barrier. 명령 대기열 장벽은 이전에 대기중인 모든 명령의 실행이 완료되고 메모리 개체에 대한 결과 업데이트가 실행을 시작하기 전에 대기열에 들어간 명령에 대해 표시되도록합니다. 이 장벽은 단일 명령 대기열의 명령간에 동기화하는 데에만 사용할 수 있습니다.

- Waiting on an event. 모든 명령을 대기열에 추가하는 모든 OpenCL API 함수는 업데이트하는 명령 및 메모리 객체를 식별하는 이벤트를 반환합니다. 해당 이벤트를 기다리는 후속 명령은 해당 메모리 개체에 대한 업데이트가 명령 실행을 시작하기 전에 볼 수 있음을 보장합니다.

3.5 Memory Objects

메모리 객체는 버퍼 객체와 이미지 객체의 두 가지 유형으로 분류됩니다. 버퍼 객체는 요소의 1 차원 컬렉션을 저장하지만 이미지 객체는 2 차원 또는 3 차원 텍스처, 프레임 버퍼 또는 이미지를 저장하는 데 사용됩니다.

버퍼 객체의 요소는 스칼라 데이터 유형 (예 : int, float), 벡터 데이터 유형 또는 사용자 정의 구조가 될 수 있습니다. 이미지 객체는 텍스처 또는 프레임 버퍼로 사용할 수있는 버퍼를 나타내는 데 사용됩니다. 이미지 객체의 요소는 사전 정의 된 이미지 형식 목록에서 선택됩니다. 메모리 객체의 최소 요소 수는 1입니다.

버퍼와 이미지 객체의 근본적인 차이점은 다음과 같습니다.

버퍼의 요소는 순차적으로 저장되며 장치에서 실행되는 커널에 의해 포인터를 사용하여 액세스 할 수 있습니다. 이미지의 요소는 사용자에게 불투명 한 형식으로 저장되며 포인터를 사용하여 직접 액세스 할 수 없습니다. 내장 함수는 커널이 이미지를 읽거나 쓸 수있게 해주는 OpenCL C 프로그래밍 언어에 의해 제공됩니다.

버퍼 객체의 경우 데이터는 커널이 액세스하는 것과 동일한 형식으로 저장되지만 이미지 객체의 경우 이미지 요소를 저장하는 데 사용되는 데이터 형식이 커널 내부에서 사용되는 데이터 형식과 다를 수 있습니다 . 이미지 요소는 커널에서 항상 4 성분 벡터 (각 구성 요소는 부동 또는 부호가 있거나 부호없는 정수일 수 있음)입니다.

이미지에서 읽는 내장 함수는 이미지 요소를 저장된 형식에서 4 성분 벡터로 변환합니다. 마찬가지로, 이미지에 작성하는 내장 함수는 이미지 요소를 4 개 요소 벡터에서 4 개 8 비트 요소와 같이 지정된 적절한 이미지 형식으로 변환합니다.

메모리 객체는 cl_mem 객체로 설명됩니다. 커널은 메모리 객체를 입력으로 가져 와서 하나 이상의 메모리 객체에 출력합니다.

3.6 The OpenCL Framework

OpenCL 프레임 워크를 사용하면 응용 프로그램에서 호스트와 하나 이상의 OpenCL 장치를 단일 이기종 병렬 컴퓨터 시스템으로 사용할 수 있습니다. 프레임 워크는 다음 구성 요소를 포함합니다.

○**OpenCL Platform layer** : 플랫폼 계층을 통해 호스트 프로그램은 OpenCL 장치와 해당 기능을 검색하고 컨텍스트를 만들 수 있습니다.

○**OpenCL Runtime :** 런타임은 호스트 프로그램이 생성 된 컨텍스트를 조작 할 수있게합니다.

○**OpenCL Compiler** : OpenCL 컴파일러는 OpenCL 커널을 포함하는 프로그램 실행 파일을 만듭니다. 컴파일러에 의해 구현 된 OpenCL C 프로그래밍 언어는 병렬 처리를위한 확장과 함께 ISO C99 언어의 하위 집합을 지원합니다.

4. The OpenCL Platform Layer

이 절에서는 응용 프로그램이 OpenCL 장치와 장치 구성 정보를 쿼리하고 하나 이상의 장치를 사용하여 OpenCL 컨텍스트를 만들 수 있도록 플랫폼 별 기능을 구현하는 OpenCL 플랫폼 계층에 대해 설명합니다.

4.1 Querying Platform Info

사용 가능한 플랫폼 목록은 다음 함수를 사용하여 얻을 수 있습니다.

cl_int clGetPlatformIDs (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)

num_entries는 플랫폼에 추가 할 수있는 cl_platform_id 항목의 수입니다. 플랫폼이 NULL이 아닌 경우 num_entries는 0보다 커야합니다.

플랫폼은 발견 된 OpenCL 플랫폼 목록을 반환합니다. 플랫폼에서 반환 된 cl_platform_id 값은 특정 OpenCL 플랫폼을 식별하는 데 사용할 수 있습니다. platforms 인수가 NULL 인

경우이 인수는 무시됩니다. OpenCL platforms returned 의 수는 num_entries 또는 사용 가능한 OpenCL platforms available 의 수에 의해 지정된 값의 최소값입니다.

num_platforms는 사용 가능한 OpenCL 플랫폼 수를 반환합니다. num_platforms가 NULL이면이 인수는 무시됩니다.

clGetPlatformIDs 는 함수가 성공적으로 실행되면 CL_SUCCESS를 반환합니다. 그렇지 않으면 다음 오류 중 하나를 반환합니다.

○num_entries가 0이고 플랫폼이 NULL이 아니거나 num_platform 및 플랫폼이 모두 NULL 인 경우 CL_INVALID_VALUE

○CL_OUT_OF_HOST_MEMORY 호스트에서 OpenCL 구현에 필요한 리소스를 할당하지 못한 경우

The function

cl_int clGetPlatformInfo (cl_platform_id platform, cl_platform_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

OpenCL 플랫폼에 대한 특정 정보를 얻습니다. clGetPlatformInfo를 사용하여 쿼리 할 수있는 정보는 표 4.1에 지정되어 있습니다.

플랫폼은 clGetPlatformIDs에 의해 반환 된 플랫폼 ID를 참조하거나 NULL 일 수 있습니다.

platform가 NULL의 경우, 동작은 구현에 의해 정의됩니다.

param_name은 쿼리되는 플랫폼 정보를 식별하는 열거 형 상수입니다.

표 4.1에 지정된 다음 값 중 하나 일 수 있습니다.

param_value는 표 4.1에 지정된 param_name에 대한 적절한 값이 반환되는 메모리 위치에 대한 포인터입니다. param_value가 NULL이면 무시됩니다.

param_value_size는 param_value가 가리키는 메모리의 크기 (바이트)를 지정합니다. 이 바이트의 크기는 표 4.1에 지정된 반환 유형의 크기보다 커야합니다.

param_value_size_ret는 param_value에 의해 질의되는 데이터의 실제 크기 (바이트)를 반환합니다. param_value_size_ret가 NULL이면 무시됩니다.

cl_platform_info	Return Type	Description
CL_PLATFORM_PROFILE	char[][2]	OpenCL profile string. Returns the profile name supported by the implementation. The profile name returned can be one of the following strings: FULL_PROFILE – if the implementation supports the OpenCL specification (functionality defined as part of the core specification and does not require any extensions to be supported). EMBEDDED_PROFILE - if the implementation supports the OpenCL embedded profile. The embedded profile is defined to be a subset for each version of OpenCL. The embedded profile for OpenCL 1.1 is described in *section 10*.
CL_PLATFORM_VERSION	char[]	OpenCL version string. Returns the OpenCL version supported by the implementation. This version string has the following format: *OpenCL<space><major_version.minor_version><space><platform-specific information>* The *major_version.minor_version* value returned will be1.1.
CL_PLATFORM_NAME	char[]	Platform name string.
CL_PLATFORM_VENDOR	char[]	Platform vendor string.
CL_PLATFORM_EXTENSIONS	char[]	Returns a space separated list of extension names (the extension names themselves do not contain any spaces) supported by the platform. Extensions defined here must be supported by all devices associated with this platform.

Table 4.1. OpenCL Platform Queries

CL_PLATFORM_PROFILE- 기술 설명

OpenCL 프로필 문자열. 구현이 지원하는 프로파일 명을 돌려줍니다. 반환 된 프로필 이름은 다음 문자열 중 하나 일 수 있습니다.
FULL_PROFILE - 구현이 OpenCL 사양을 지원하는 경우 (코어 사양의 일부로 정의 된 기능이며 지원되는 확장 기능이 필요하지 않음)

FULL_PROFILE - 구현이 OpenCL 사양을 지원하는 경우 (코어 사양의 일부로 정의 된 기능이며 지원되는 확장 기능이 필요하지 않음)

EMBEDDED_PROFILE - 구현이 OpenCL 내장 프로파일을 지원하는 경우 삽입 된 프로파일은 OpenCL의 각 버전에 대한 하위 집합으로 정의됩니다. OpenCL 1.1에 대한 내장 프로파일은 섹션 10에 설명되어 있습니다.

CL_PLATFORM_VERSION 기술설명

OpenCL 버전 문자열. 구현이 지원하는 OpenCL 버전을 반환합니다. 이 버전 문자열의 형식은 다음과 같습니다.

OpenCL<space><major_version.min or _version><space><platform-specific information>
반환되는 major_version.minor_version 값은 1.1입니다.

CL_PLATFORM_NAME 기술설명
Platform name string

CL_PLATFORM_VENDOR기술설명
Platform vendor string.

CL_PLATFORM_EXTENSIONS 기술설명

플랫폼에 의해 지원되고있는 확장자 명의 공백 단락의리스트를 돌려줍니다 (확장자 그 자체는 스페이스를 포함하지 않는다). 여기에 정의 된 확장은이 플랫폼과 관련된 모든 장치에서 지원되어야합니다.

함수가 성공적으로 실행되면 **clGetPlatformInfo**는 CL_SUCCESS를 반환합니다. 그렇지 않으면 다음 오류(OpenCL 사양에는 API 호출에 의해 반환 된 오류 코드의 우선 순위가 설명되어 있지 않습니다.) 중 하나를 반환합니다 .

○플랫폼이 유효한 플랫폼이 아닌 경우 CL_INVALID_PLATFORM

○CL_INVALID_VALUE param_name이 지원되는 값 중 하나가 아닌 경우 또는 param_value_size로 지정된 바이트의 크기가 <표 4.1에 지정된 반환 유형의 크기이고 param_value가 NULL 값이 아닌 경우

○CL_OUT_OF_HOST_MEMORY 호스트에서 OpenCL 구현에 필요한 리소스를 할당하지 못한 경우

.

4.2 Querying Devices

플랫폼에서 사용 가능한 장치 목록은 다음 기능을 사용하여 얻을 수 있습니다.

cl_int **clGetDeviceIDs(clGetDeviceIDs는 플랫폼에있는 실제 물리적 장치의 전체 또는 일부를 반환 할 수 있으며 device_type과 일치합니다.)** (cl_platform_id platform,
cl_device_type device_type,
cl_uint num_entries,
cl_device_id *devices,
cl_uint *num_devices)

플랫폼은 clGetPlatformIDs에 의해 반환 된 플랫폼 ID를 참조하거나 NULL 일 수 있습니다. platform가 NULL의 경우, 동작은 구현에 의해 정의됩니다.
device_type은 OpenCL 장치의 유형을 식별하는 비트 필드입니다. device_type은 특정 OpenCL 장치 또는 사용 가능한 모든 OpenCL 장치를 쿼리하는 데 사용할 수 있습니다. device_type의 유효한 값은 표 4.2에 지정되어 있습니다.

cl_device_type	Description
CL_DEVICE_TYPE_CPU	An OpenCL device that is the host processor. The host processor runs the OpenCL implementations and is a single or multi-core CPU.
CL_DEVICE_TYPE_GPU	An OpenCL device that is a GPU. By this we mean that the device can also be used to accelerate a 3D API such as OpenGL or DirectX.
CL_DEVICE_TYPE_ACCELERATOR	Dedicated OpenCL accelerators (for example the IBM CELL Blade). These devices communicate with the host processor using a peripheral interconnect such as PCIe.
CL_DEVICE_TYPE_DEFAULT	The default OpenCL device in the system.
CL_DEVICE_TYPE_ALL	All OpenCL devices available in the system.

표 4.2. OpenCL 장치 범주 목록

num_entries는 장치에 추가 할 수있는 cl_device 항목의 수입니다. 장치가 NULL이 아닌 경우 num_entries는 0보다 커야합니다.

devices는 발견 된 OpenCL 장치 목록을 반환합니다. 장치에서 반환 된 cl_device_id 값은 특정 OpenCL 장치를 식별하는 데 사용할 수 있습니다. devices 인수가 NULL이면이 인수는 무시됩니다. 리턴 된 OpenCL 디바이스의 수는 num_entries가 지정한 값 또는 device_type과 일치하는 유형의 OpenCL 디바이스의 최소값입니다.

num_devices는 device_type과 일치하는 사용 가능한 OpenCL 장치의 수를 반환합니다. num_devices가 NULL이면이 인수는 무시됩니다.

clGetDeviceIDs는 함수가 성공적으로 실행되면 CL_SUCCESS를 반환합니다. 그렇지 않으면 다음 오류 중 하나를 반환합니다.

○플랫폼이 유효한 플랫폼이 아닌 경우 CL_INVALID_PLATFORM

○device_type이 유효한 값이 아닌 경우 CL_INVALID_DEVICE_TYPE

○num_entries가 0이고 장치가 NULL이 아니거나 num_devices와 장치가 모두 NULL 인 경우 CL_INVALID_VALUE

○device_type과 일치하는 OpenCL 장치가없는 경우 CL_DEVICE_NOT_FOUND입니다.

○장치에서 OpenCL 구현에 필요한 리소스를 할당하지 못한 경우 CL_OUT_OF_RESOURCES입니다.

○CL_OUT_OF_HOST_MEMORY 호스트에서 OpenCL 구현에 필요한 리소스를 할당하지 못한 경우