

기초부터 뽀개보자~ 잘 뽀개지는 쿠쿠다스

CUDA 프로그래밍 기초

2018

MODUCON

이성철, Ph.D

NAVER 쇼핑데이터개발, 비즈OCR

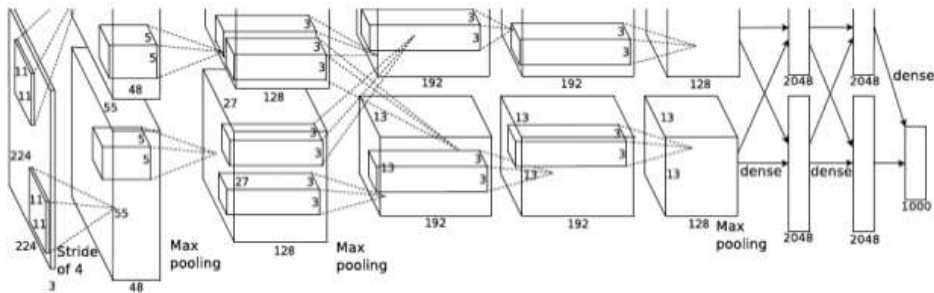
목차

- What is CUDA?
- 개발환경 설정
 - Visual Studio 2017
 - AWS
- CUDA by Example 따라 하기
 - Hello world
 - Vector Add
 - Dot Product
 - Histogram
- 병렬처리개요
 - 병행, 병렬, 분산
 - 병렬처리의 분류
 - CUDA 프로그래밍 모델
 - 기타 병렬처리 기술

What is CUDA?

4

- Similar framework to LeCun'98 but:
 - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
 - More data (10^6 vs. 10^3 images)
 - GPU implementation (50x speedup over CPU)
 - Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

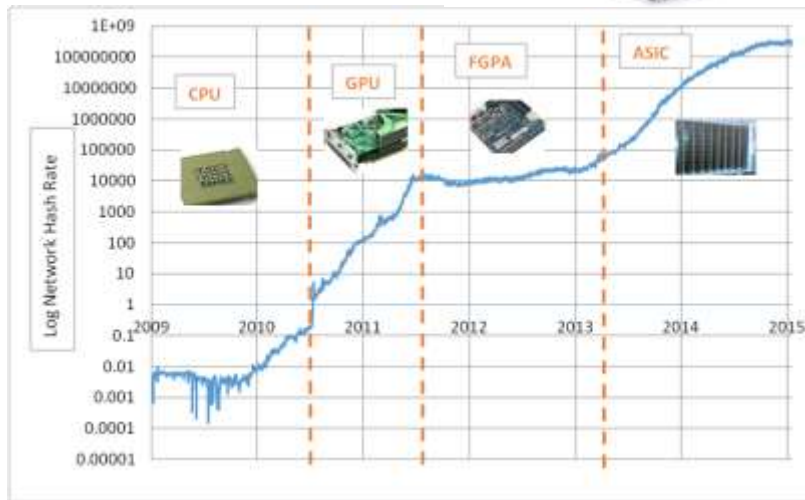
대부분 딥러닝 라이브러리는 학습시간을 줄이기 위해 CUDA 백엔드를 사용한다.

Date	Configuration	Caffe Performance (FPS)
11/2013	CPU	~1
11/2013	K40	~10
9/2014	K40 + cuDNN1	~14
7/2015	M40 + cuDNN3	~44
12/2015	M40 + cuDNN4	~54

AlexNet training throughput based on 20 iterations,
CPU: 1x E5-2680v3 12 Core 2.5GHz, 128GB System Memory, Ubuntu 14.04



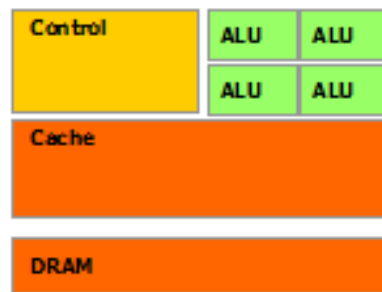
What is CUDA?



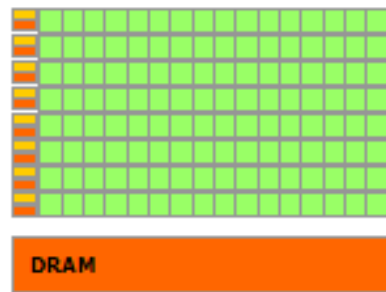
코인 마이닝에도 엄청 많은 GPU 장비가 소요 된다. 여기에 필요한 프로그래밍 언어가 CUDA 또는 OpenCL 이다.

What is CUDA?

- Compute Unified Device Architecture (CUDA)
 - 2007년 1.0 Release, 현재 10.0 Release
- Nvidia 의 General-Purpose Graphic Processing Unit (GPGPU) 기술
 - CPU에 비해 훨씬 많은 스레드를 동시에 처리할 수 있다.
 - 캐시와 공유 메모리를 지원한다.
 - C 언어와 유사하여 배우기 쉽다.

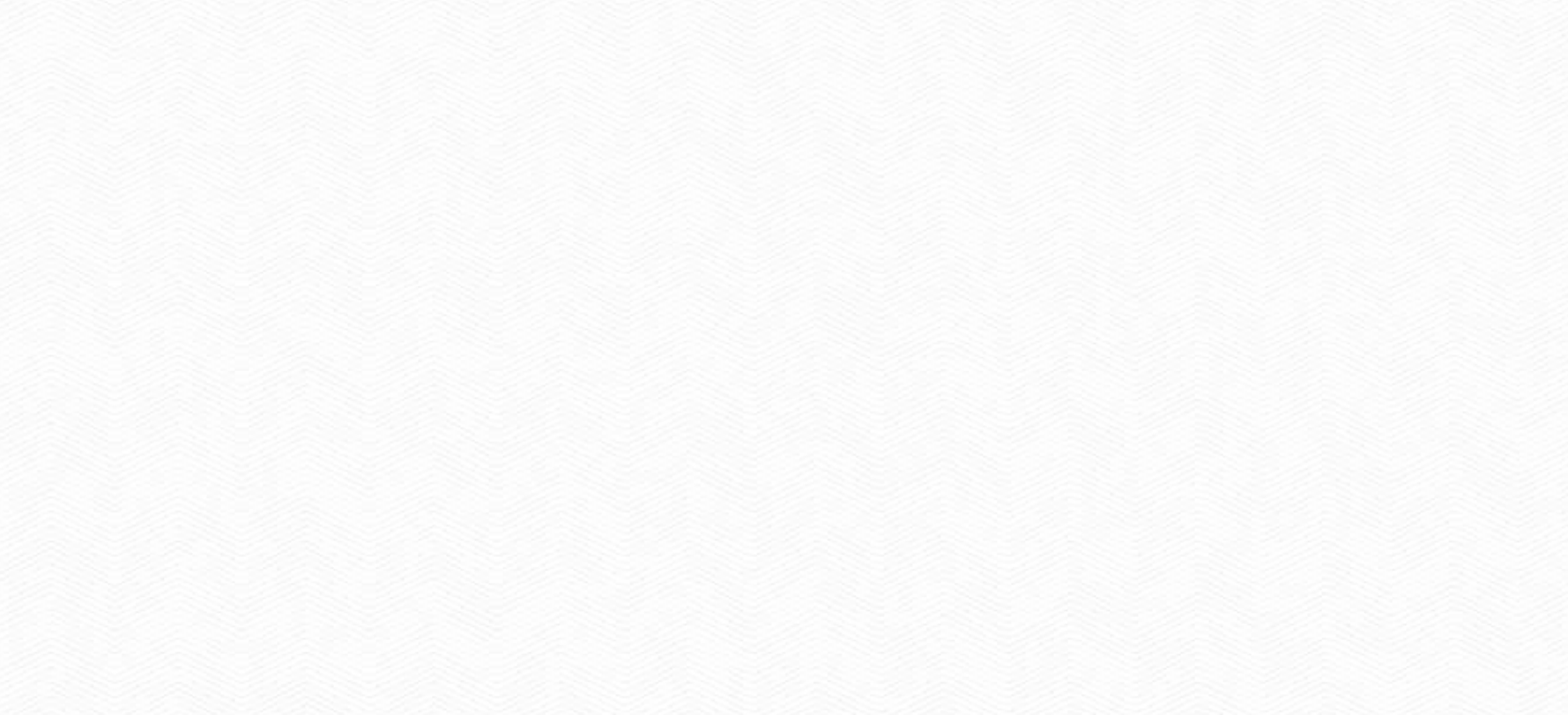


CPU



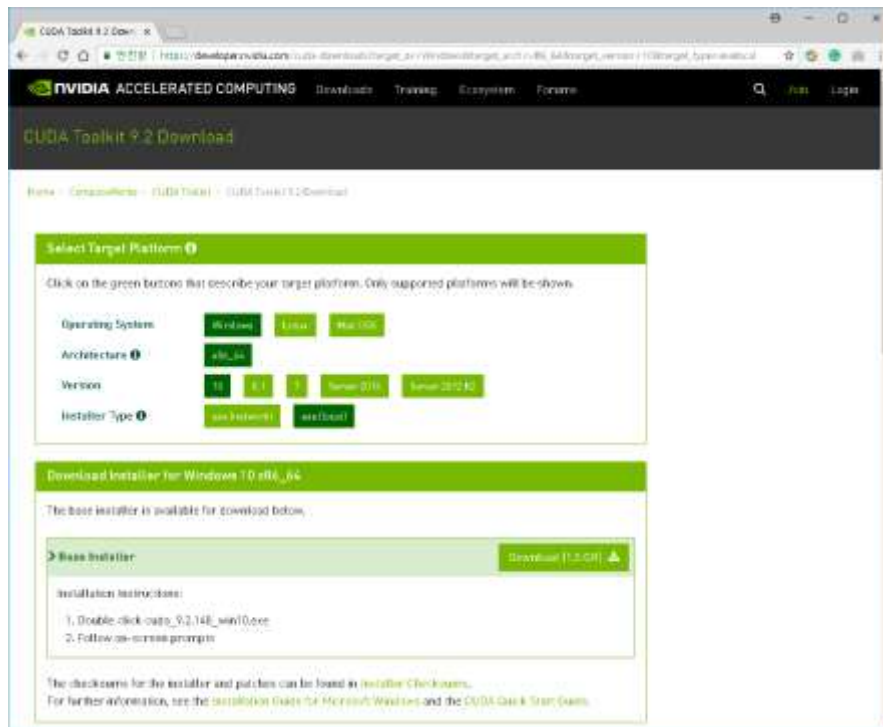
GPU

개발환경 설정

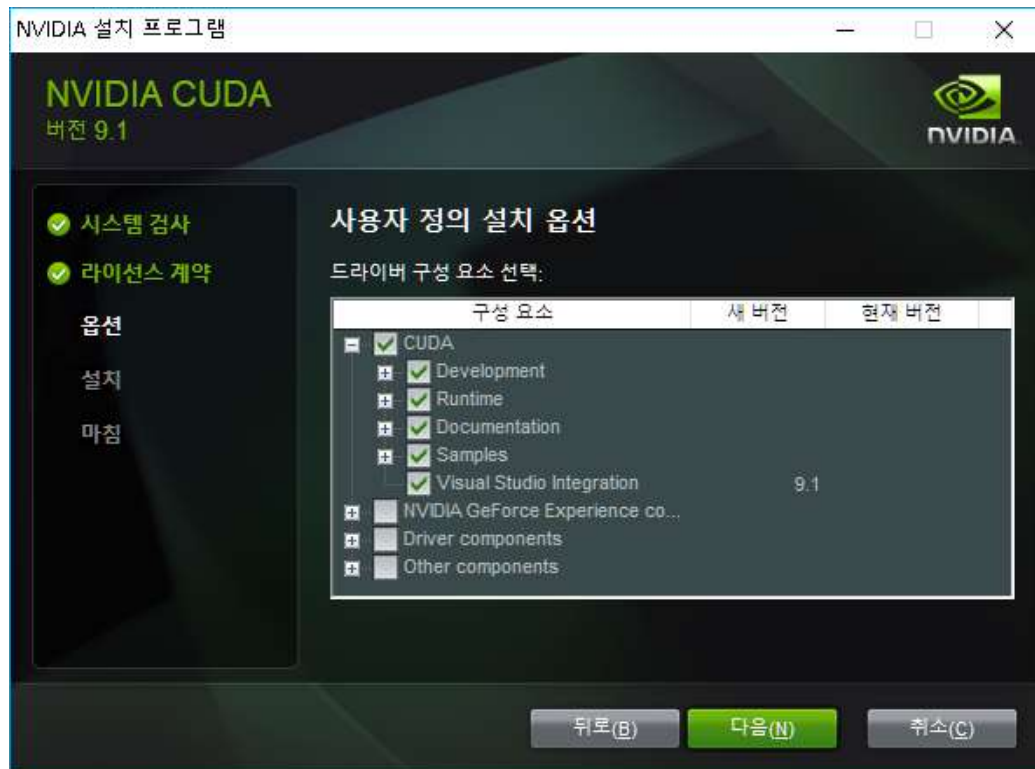


CUDA Toolkit 9.2 설치

- https://developer.nvidia.com/cuda-downloads?target_os=Windows&target_arch=x86_64



CUDA Toolkit 9.2 설치



GPU 드라이버 버전 확인

- 최신 드라이버로 업데이트 해야 함

Table 1. CUDA Toolkit and Compatible Driver Versions

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62

<https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html>

VS 2017 와 CUDA 9.2 호환성 문제 해결

- VS2017 와 CUDA 9.2 호환문제가 있음
- CUDA 설치 경로에서 host_config.h 편집
- "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.2\include\crt\host_config.h"
- _MSC_VER > 1915 로 바꿔 줘야 함 <http://kkokkal.tistory.com/1327>

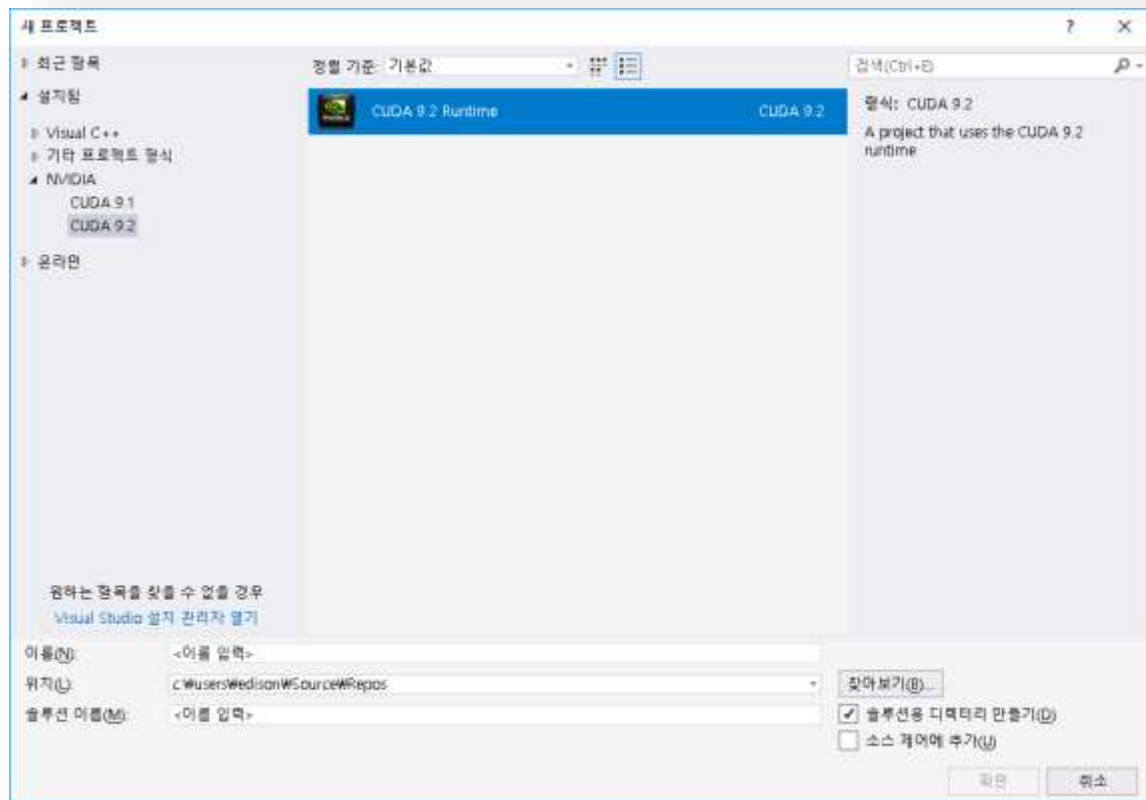
```
129 #if defined(_WIN32)
130
131 #if _MSC_VER < 1600 || _MSC_VER > 1915
132
133     #error -- unsupported Microsoft Visual Studio version! Only the versions 2012, 2013, 2015 and 2017 are supported!
134
135     #elif _MSC_VER == 1600 /* _MSC_VERION == 1600 */
136
137     #pragma message("support for Microsoft Visual Studio 2010 has been deprecated!")
138
139 #endif /* _MSC_VER < 1600 || _MSC_VER > 1800 || _MSC_VERSION == 1600 */
140
141 #endif /* _WIN32 */
```

2.2.1. CUDA Compilers

- The following compilers are supported as host compilers in `nvcc`:
 - Clang 6.0
 - Microsoft Visual Studio 2017* (RTW, Update 8 and later)
 - Xcode 9.4
 - XLC 16.1.x
 - ICC 18
 - PGI 18.x (with -std=c++14 mode)
 - *Starting with CUDA 10.0, *nvcc* supports all versions of Visual Studio 2017

CUDA 프로젝트 생성

- 새 프로젝트 생성
- CUDA 버전 9.2 선택



AWS Linux CUDA Toolkit 설정

앞부분의 Visual Studio 사용하지 않거나 GPU가 없으면 AWS 혹은 NCLOUD 를 고려할 수 있다.

- AWS Deep Learning AMI를 사용하면 이미 CUDA 9가 설치되어 있기 때문에 별도의 설정이 필요없이 nvcc 컴파일러로 CUDA 프로그램을 컴파일 할 수 있다.
- NCLOUD gpu server 에도 nvcc 컴파일러가 포함되어 있다.
- 다른 버전의 Linux를 사용할 때 <https://docs.nvidia.com/cuda/cuda-installation-guide-linux/>를 따라 CUDA Toolkit을 설치하면 된다.

AWS 사용 시 주의사항

AWS Support Dashboard x Billing Management Co x Billing Management Co x

안전함 | https://console.aws.amazon.com/billing/home?nc2=bi_m_bc#/paymenthistory/history/redirected

aws 서비스 리소스 그룹

edison 전역 치환

대시보드
청구서
비용 탐색기
예산
보고서
비용 할당 태그
결제 방법
결제 내역
통합 결제
기본 설정
크레딧
세금 설정

결제 내역

다음은 성공한 지불 거래 목록입니다.

CSV 다운로드 인쇄

시작: 2018-04-09 끝: 2018-07-09 필터 표시: 4

자물 날짜	인보이스/영수증 ID	결제 수단	거래 유형	결제 방법	거래 금액
2018-05-12	134737591	마지막 숫자: 2517	Charge	CreditCard	5.67 USD
2018-06-03	135483598	마지막 숫자: 2517	Charge	CreditCard	11.24 USD
2018-07-03	141517251	마지막 숫자: 2517	Charge	CreditCard	447.72 USD
2018-07-09	142725072	마지막 숫자: 2517	Refund	CreditCard	447.72 USD

AWS 사용 후 STOP을 하지 않으면 요금 폭탄 맞을 수 있다. $\pi\pi$ 사용 후 꼭 STOP을 눌러야 한다!



<http://m.kado.net/news/articleView.html?idxno=617258#Redyho>

폭탄 맞은 후 고객센터한테 문의하여 해결..

NCLOUD 사용 시 주의사항

상세 요금내역 조회

청구 월: 2018년 7월

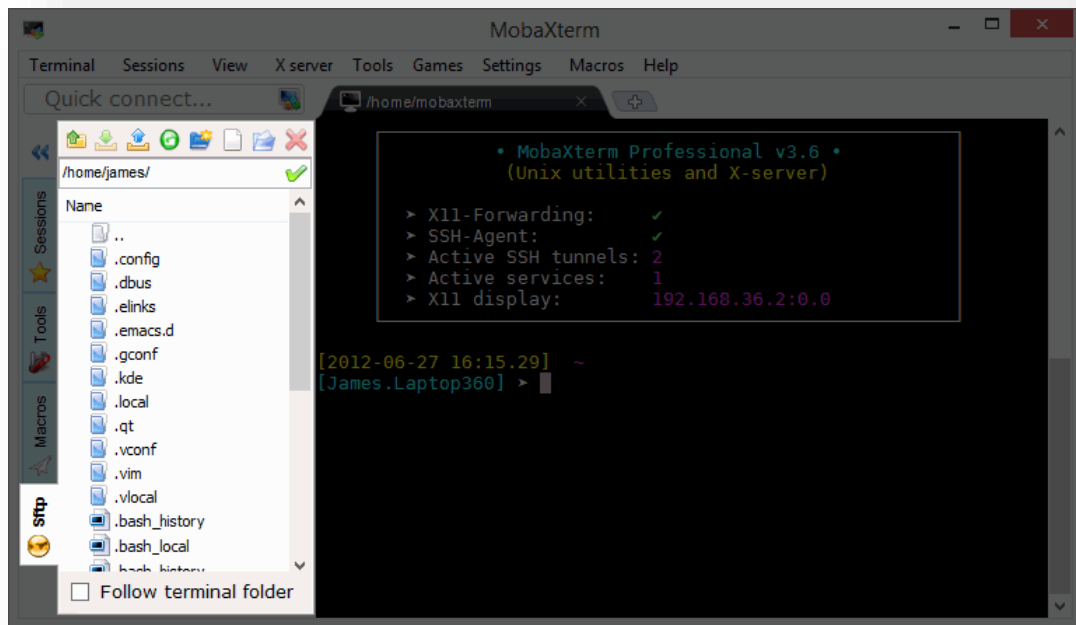
서비스 구분/지역		설비	요금제	이용요금	합산 금액			귀책금	납부 예상 요금
					프로덕션	안정	기타		
Server	한국	gcloud	종량제	1,191,420	0	0	0	0	1,191,420
	소계			1,191,420	0	0	0	0	1,191,420
	한국	106.10.45.14	종량제	4,030	0	0	0	0	4,030
공간 IP									4,030

NCLOUD는 정지해도 요금이 발생함. 사용 후 꼭
반납해야 합니다. TTT

답은 여전히 고객센터..

Window 용 터미널 프로그램

- MobaXterm 다운받고 설치
 - Windows에서 SSH와 SFTP 동시에 띄울 수 있어 아주 편리함
 - <https://mobaxterm.mobatek.net/>



EC2 Management Console

← → ↺ 🏠

Secure | https://ap-northeast-2.console.aws.amazon.com/ec2/v2/home?region=ap-northeast-2#Instances:sort=instanceId

☆

🔍 📄 📺 🔄

VA Ruby | Codecademy 🖐 Camera support - S DIGITS/GettingStart DL study DIGITS SMS 시작하기 - 시작하기 A basic Windows se Medialan - RTSP, R1 h.264 - Stream H26

aws

Services ▾ Resource Groups ▾ ⭐

🔔 edison ▾ Seoul ▾ Suppo

EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Launch Templates

Spot Requests

Reserved Instances

Dedicated Hosts

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

NETWORK & SECURITY

Security Groups

Launch Instance ▾

Connect

Actions ▾

🔍 Filter by tags and attributes or search by keyword

🔍 1 to 2 of 2

<input type="checkbox"/>	Name ▾	Instance ID ▴	Instance Type ▾	Availability Zone ▾	Instance State ▾	Status Checks ▾	Alarm Status	Public DNS (IP
<input checked="" type="checkbox"/>		i-05f2183e47b9ab3f5	p2.xlarge	ap-northeast-2c	🟢 running	✅ 2/2 checks passed	None	ec2-13-209-48-
<input type="checkbox"/>		i-08157ac7f069021af	t2.micro	ap-northeast-2c	🔴 stopped		None	

Instance: i-05f2183e47b9ab3f5

Public DNS: ec2-13-209-48-114.ap-northeast-2.compute.amazonaws.com

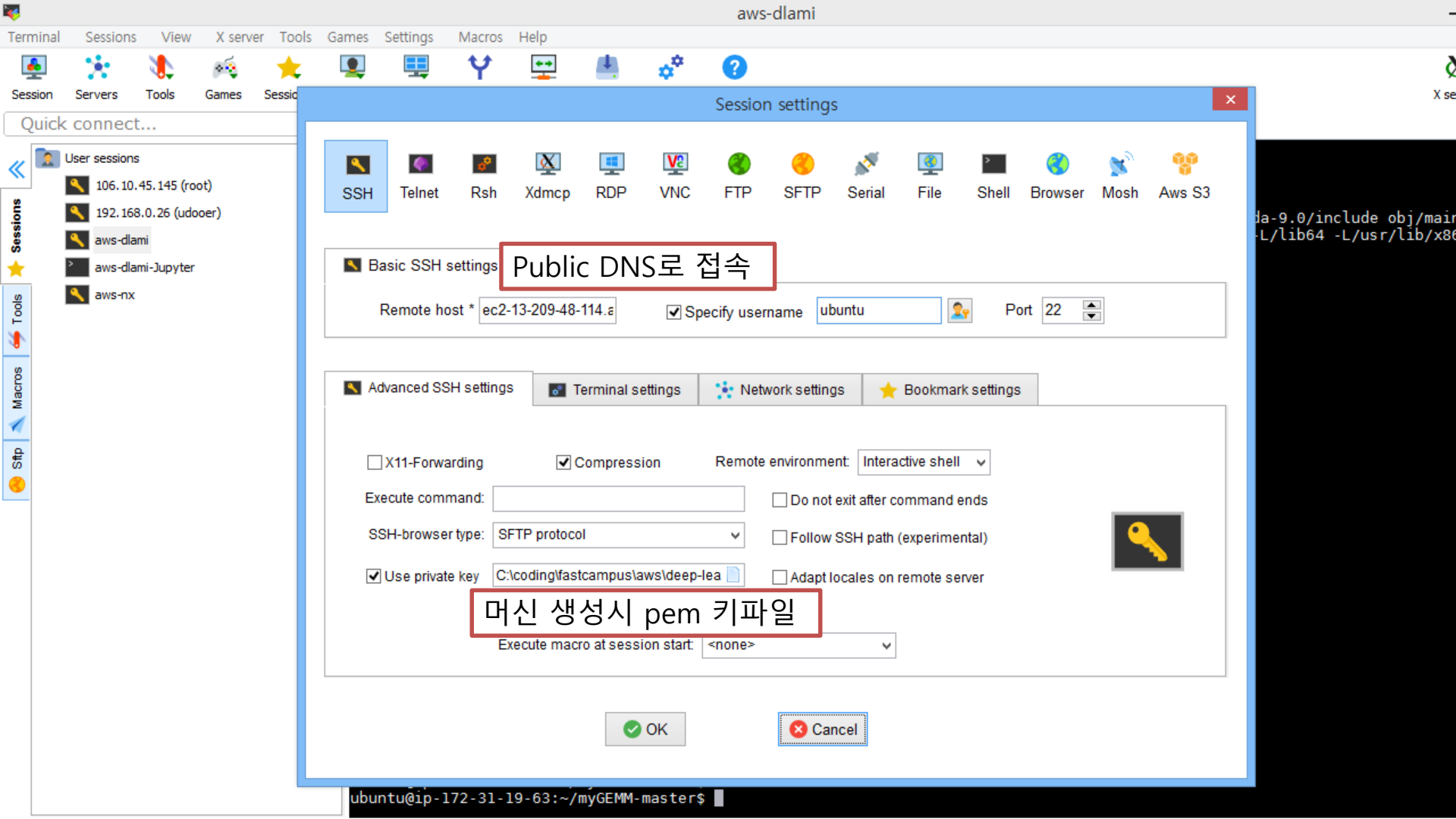
Description

Status Checks

Monitoring

Tags

Instance ID	i-05f2183e47b9ab3f5	Public DNS (IPv4)	ec2-13-209-48-114.ap-northeast-2.compute.amazonaws.com
Instance state	running	IPv4 Public IP	13.209.48.114
Instance type	p2.xlarge	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-19-63.ap-northeast-2.compute.internal
Availability zone	ap-northeast-2c	Private IPs	172.31.19.63
Security groups	Deep Learning AMI -Ubuntu--9-0-AutogenByAWSMP-2. view inbound rules.	Secondary private IPs	



aws-dlami

Terminal Sessions View X server Tools Games Settings Macros Help



Session Servers Tools Games Sessions

Quick connect...

User sessions

- 106.10.45.145 (root)
- 192.168.0.26 (udooer)
- aws-dlami
- aws-dlami-Jupyter
- aws-nx

Sessions

Tools

Macros

Stp

Session settings



Basic SSH settings

Public DNS로 접속

Remote host * ec2-13-209-48-114.a ☒ Specify username ubuntu Port 22

Advanced SSH settings

Terminal settings

Network settings

Bookmark settings

☐ X11-Forwarding ☒ Compression Remote environment: Interactive shell
Execute command: ☐ Do not exit after command ends
SSH-browser type: SFTP protocol ☐ Follow SSH path (experimental)
☒ Use private key C:\coding\fastcampus\aws\deep-lea ☐ Adapt locales on remote server

머신 생성시 pem 키파일

Execute macro at session start <none>

OK

Cancel

ubuntu@ip-172-31-19-63:~/myGEMM-master\$

Compiling the Code: Linux



nvcc <filename>.cu [-o <executable>]

- Builds release mode

nvcc -g <filename>.cu

- Builds debug (device) mode
- Can debug host code but not device code (runs on GPU)

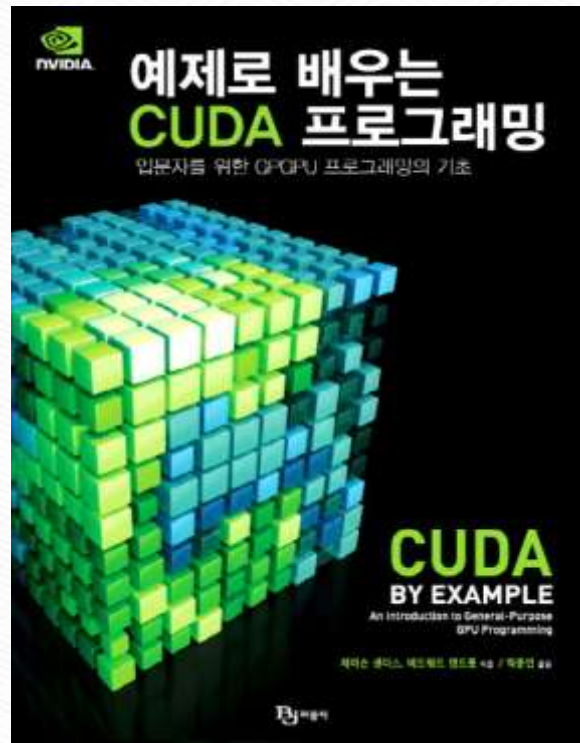
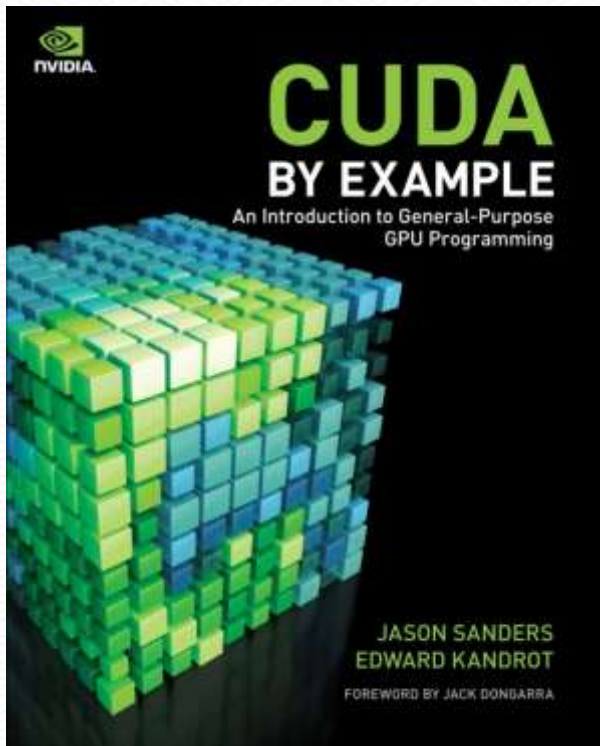
nvcc -deviceemu <filename>.cu

- Builds device emulation mode
- All code runs on CPU, but no debug symbols

nvcc -deviceemu -g <filename>.cu

- Builds debug device emulation mode
- All code runs on CPU, with debug symbols
- Debug using gdb or other linux debugger

CUDA by Example 따라 하기



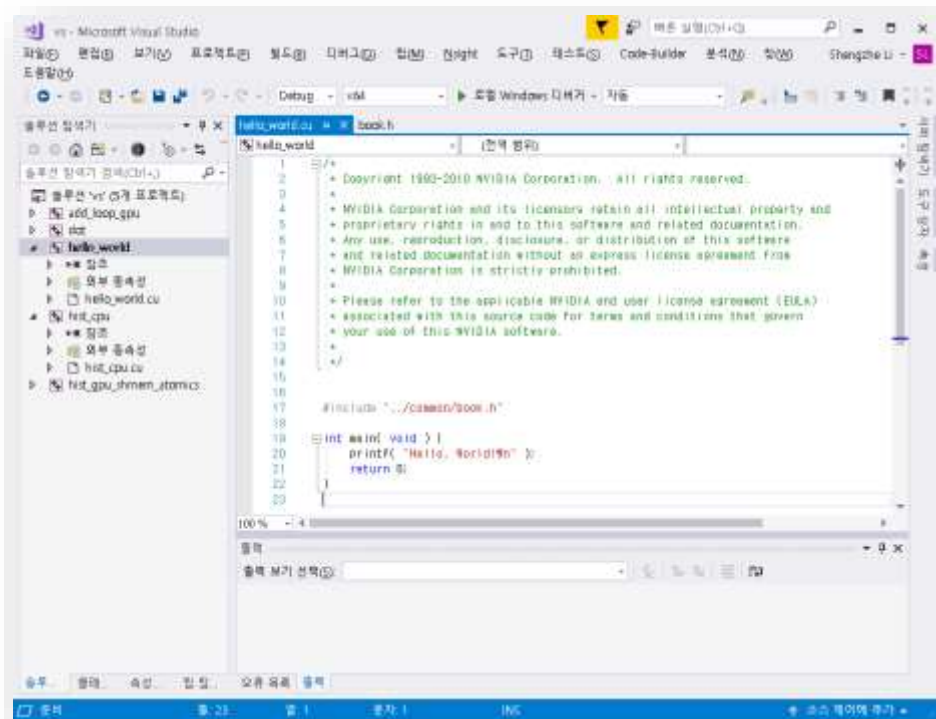
Source code at

https://github.com/lishengzhe/cuda_by_example_vs2017



<https://openwiki.kr/%EB%94%B0%EB%B4%89%EC%B6%A9>

Hello world



- 일반적인 c 프로그램과 동일 함
- 확장자가 .cu (hello_world.cu) 이고 nvcc 컴파일러로 컴파일
- book.h 에 일반적인 유틸리티 함수 포함
- nvcc 컴파일 환경설정이 잘 되었으면 바로 컴파일/실행 할 수 있음

Vector Add (Chapter 4)

```
#include "../common/book.h"

#define N 10

void add( int *a, int *b, int *c ) {
    int tid = 0;    // this is CPU zero, so we start at zero
    while (tid < N) {
        c[tid] = a[tid] + b[tid];
        tid += 1;    // we have one CPU, so we increment by one
    }
}
```

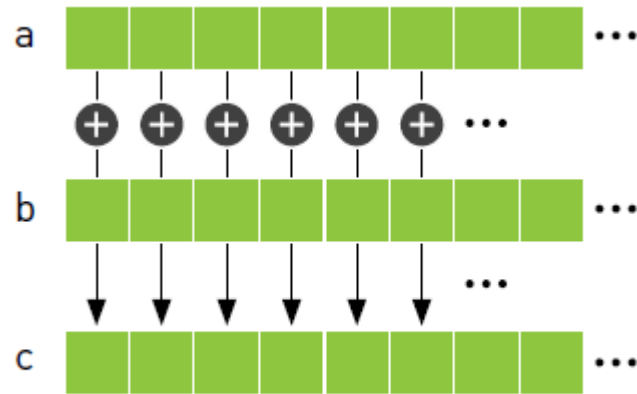


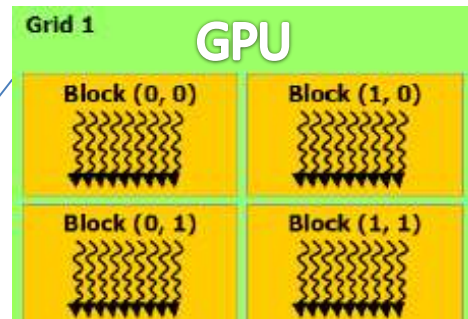
Figure 4.1 Summing two vectors

- Vector Add 의 C version 이다.
- While 문으로 단순히 $c[tid] = a[tid] + b[tid]$ 를 반복 계산한다.
- tid는 1씩 증가하므로 총 N번 루프가 필요하다.

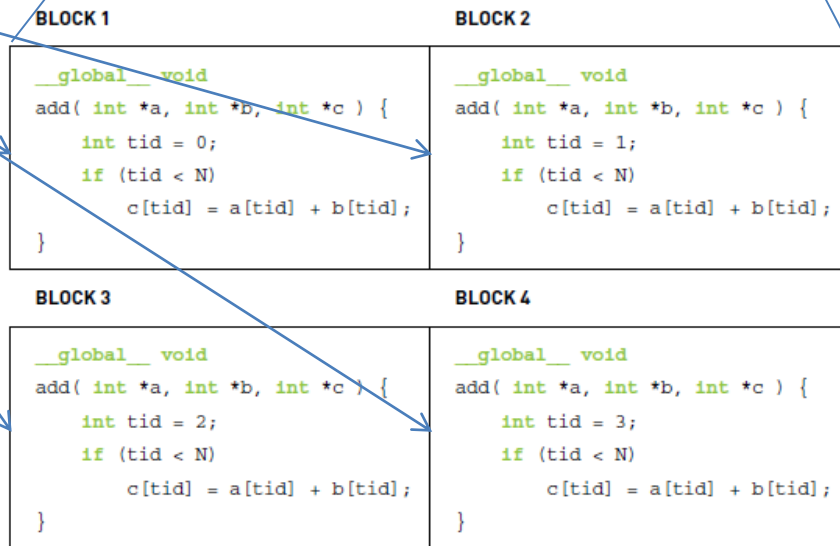
Vector Add – CUDA version (Chapter 4)

```
#define N 10

__global__ void add( int *a, int *b, int *c ) {
    int tid = blockIdx.x; // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
```



각 GPU block에 코드 복사함
동시에 blockIdx 인자 대입



- 커널 코드를 N개의 Block에 각각 복사하여 동시에 실행한다.
- 이때 blockIdx는 블록마다 다르다.
- Block수 가 10 일 때 한번 루프로 계산된다.

Vector Add – CUDA version (Chapter 4)

```
int main( void ) {
    int *a, *b, *c;
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the CPU
    a = (int*)malloc( N * sizeof(int) );
    b = (int*)malloc( N * sizeof(int) );
    c = (int*)malloc( N * sizeof(int) );

    // allocate the memory on the GPU
    HANDLE_ERROR( cudaMalloc( (void**)&dev_a, N * sizeof(int) ) );
    HANDLE_ERROR( cudaMalloc( (void**)&dev_b, N * sizeof(int) ) );
    HANDLE_ERROR( cudaMalloc( (void**)&dev_c, N * sizeof(int) ) );

    // fill the arrays 'a' and 'b' on the CPU
    for (int i=0; i<N; i++) {
        a[i] = i;
        b[i] = 2 * i;
    }

    // copy the arrays 'a' and 'b' to the GPU
    HANDLE_ERROR( cudaMemcpy( dev_a, a, N * sizeof(int),
                             cudaMemcpyHostToDevice ) );
    HANDLE_ERROR( cudaMemcpy( dev_b, b, N * sizeof(int),
                             cudaMemcpyHostToDevice ) );
}
```

호스트 메모리 할당

디바이스 메모리 할당

벡터 초기화

호스트 메모리를 디바이스에
복사

Vector Add – CUDA version (Chapter 4)

```
add<<<N,1>>>( dev_a, dev_b, dev_c );  
  
// copy the array 'c' back from the GPU to the CPU  
HANDLE_ERROR( cudaMemcpy( c, dev_c, N * sizeof(int),  
                          cudaMemcpyDeviceToHost ) );  
  
// display the results  
for (int i=0; i<N; i++) {  
    printf( "%d + %d = %d\n", a[i], b[i], c[i] );  
}  
  
// free the memory allocated on the GPU  
HANDLE_ERROR( cudaFree( dev_a ) );  
HANDLE_ERROR( cudaFree( dev_b ) );  
HANDLE_ERROR( cudaFree( dev_c ) );  
  
return 0;  
}
```

10개 Block,
Block당 1개 Thread로 커널 실행

디바이스에서 호스트로
메모리 복사

결과 검증

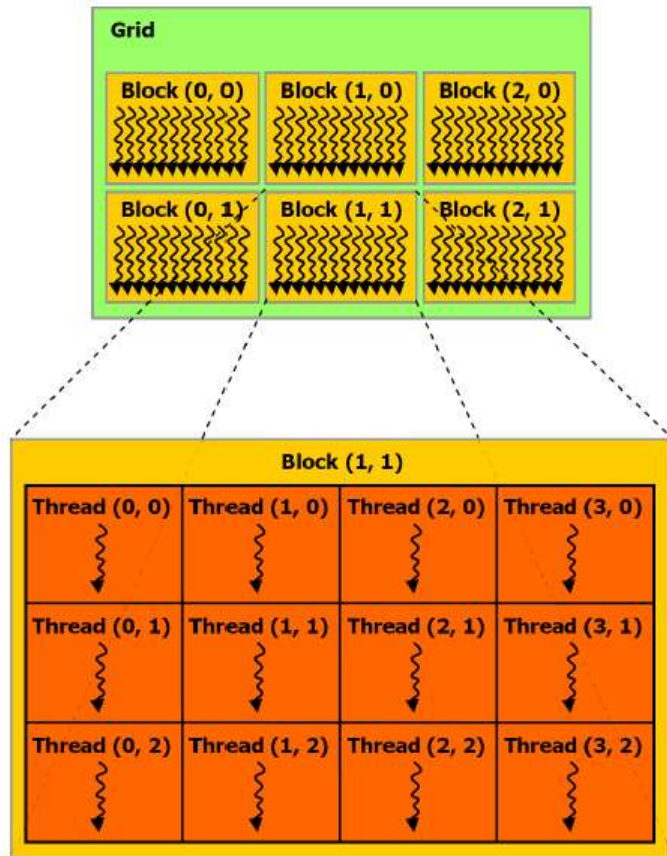


C:\WINDOWS\system32\cmd.exe
We did it!
계속하려면 아무 키나 누르십시오 . . .

Grid

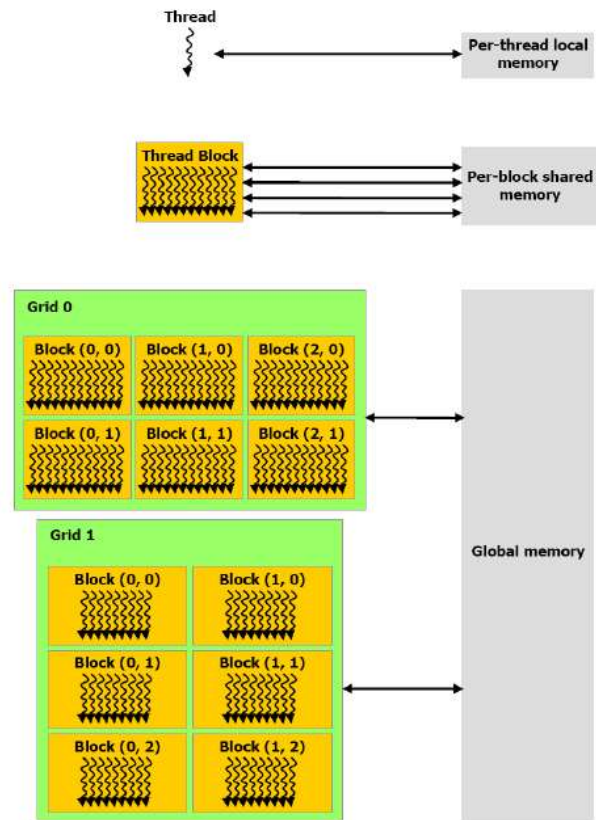
- C 함수는 N번의 연산을 N개 Thread에서 동시 실행
- 같은 Block에 있는 Thread는 협동하여 작업을 수행할 수 있으며 메모리를 공유할 수 있다.
(Shared memory)

* OpenCL 에서의 Global work size과 Local work size
에 해당되는 개념 임



Memory type

- Private local memory
 - Thread 내부에서 생성되고 Thread 사이에 공유할 수 없음
- Shared memory
 - Thread 에서 생성되고 같은 Block에서 공유 할 수 있음
- Global memory
 - 디바이스 전체에서 접근 할 수 있음
- Constant memory
 - Thread 내부에서 읽기 전용으로 접근 가능
- Texture memory
 - 디바이스 전체에서 접근 가능, text2D 함수로 접근



Dot Product (Chapter 5)

Equation 5.1

$$(x_1, x_2, x_3, x_4) \cdot (y_1, y_2, y_3, y_4) = x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4$$

- Vector Add와 달리 Dot Product 연산은 각 Element가 독립적이지 않다.
- Thread 별로 곱셈 연산을 병렬로 구하고 결과를 더해주는 연산이 필요하다.
- 같은 Block내의 Thread는 Shared memory를 사용할 수 있기 때문에 중간 곱셈 결과를 Shared memory 에 더해 주고 모든 Block이 계산이 끝날 때 Global memory에 최종 결과를 더해 준다.
- 모든 Block의 계산이 끝나야 하므로 __syncthreads()를 사용하게 된다.

Parallel Reduction

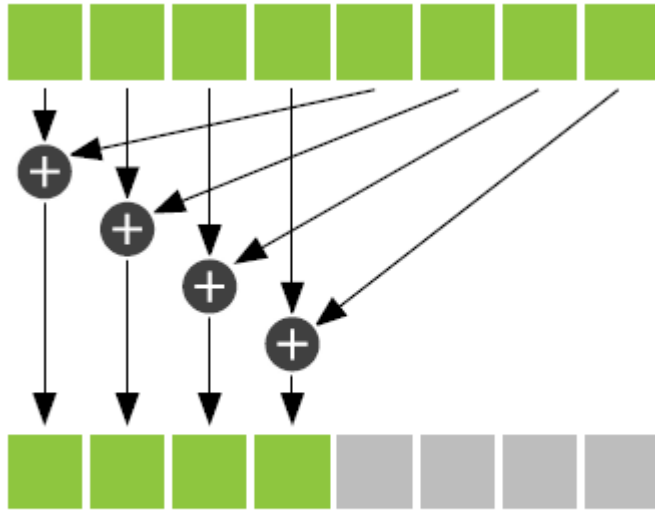


Figure 5.4 One step of a summation reduction

- For loop -> 8회
- Parallel reduction -> 3회

Dot Product (Chapter 5)

```
#include "../common/book.h"

#define imin(a,b) (a<b?a:b)

const int N = 33 * 1024;
const int threadsPerBlock = 256;
const int blocksPerGrid =
    imin( 32, (N+threadsPerBlock-1) / threadsPerBlock );
```

Block 당 Thread 개수를
지정하고 필요한 Block 개수를
계산 함

```

__global__ void dot( float *a, float *b, float *c ) {
    __shared__ float cache[threadsPerBlock];
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int cacheIndex = threadIdx.x;

    float temp = 0;
    while (tid < N) {
        temp += a[tid] * b[tid];
        tid += blockDim.x * gridDim.x;
    }

    // set the cache values
    cache[cacheIndex] = temp;

    // synchronize threads in this block
    __syncthreads();

    // for reductions, threadsPerBlock must be a power of 2
    // because of the following code
    int i = blockDim.x/2;
    while (i != 0) {
        if (cacheIndex < i)
            cache[cacheIndex] += cache[cacheIndex + i];
        __syncthreads();
        i /= 2;
    }

    if (cacheIndex == 0)
        c[blockIdx.x] = cache[0];
}

```

로컬 메모리 할당

글로벌 및 로컬 idx 가져오기

temp+=a[i]*b[i] 임시 결과
계산

로컬 메모리에 임시 결과
저장

로컬 메모리 동기화

Reduction 계산

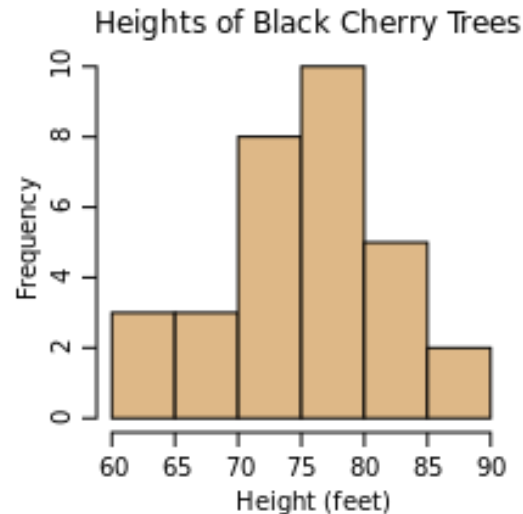
Histogram (Chapter 9)

- Histogram이란 단순히 관측값의 개수를 겹치지 않는 다양한 계급(상자라고도 한다)로 보내는(사상, mapping) 것이다. 즉, 일종의 사상이다. --- from wikipedia

N 을 모든 관측값의 수라 하고, n 을 상자 개수라 하면, 히스토그램 h_k 는 다음 조건을 만족한다:

$$N = \sum_{k=1}^n h_k$$

여기서 k 는 상자의 번호이다.



Histogram – C version (Chapter 9)

```
#include "../common/book.h"

#define SIZE    (100*1024*1024)

int main( void ) {
    unsigned char *buffer =
        (unsigned char*)big_random_block( SIZE );

    // capture the start time
    clock_t    start, stop;
    start = clock();

    unsigned int    histo[256];
    for (int i=0; i<256; i++)
        histo[i] = 0;

    for (int i=0; i<SIZE; i++)
        histo[buffer[i]]++;

    stop = clock();
    float    elapsedTime = (float)(stop - start) /
        (float)CLOCKS_PER_SEC * 1000.0f;
    printf( "Time to generate: %3.1f ms\n", elapsedTime );

    long histoCount = 0;
    for (int i=0; i<256; i++) {
        histoCount += histo[i];
    }
    printf( "Histogram Sum: %ld\n", histoCount );

    free( buffer );
    return 0;
}
```

- 256개 bin의 histogram 을 생성하고 SIZE 만큼 반복하여 해당되는 bin에 누적시킨다.

Histogram – CUDA version (Chapter 9)

```
--global-- void histo_kernel( unsigned char *buffer,
                             long size,
                             unsigned int *histo ) {

    // clear out the accumulation buffer called temp
    // since we are launched with 256 threads, it is easy
    // to clear that memory with one write per thread
    __shared__ unsigned int temp[256];
    temp[threadIdx.x] = 0;
    __syncthreads();

    // calculate the starting index and the offset to the next
    // block that each thread will be processing
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    int stride = blockDim.x * gridDim.x;
    while ( i < size ) {
        atomicAdd( &temp[buffer[i]], 1 );
        i += stride;
    }

    // sync the data from the above writes to shared memory
    // then add the shared memory values to the values from
    // the other thread blocks using global memory
    // atomic adds
    // same as before, since we have 256 threads, updating the
    // global histogram is just one write per thread!
    __syncthreads();
    atomicAdd( &(histo[threadIdx.x]), temp[threadIdx.x] );
}
```

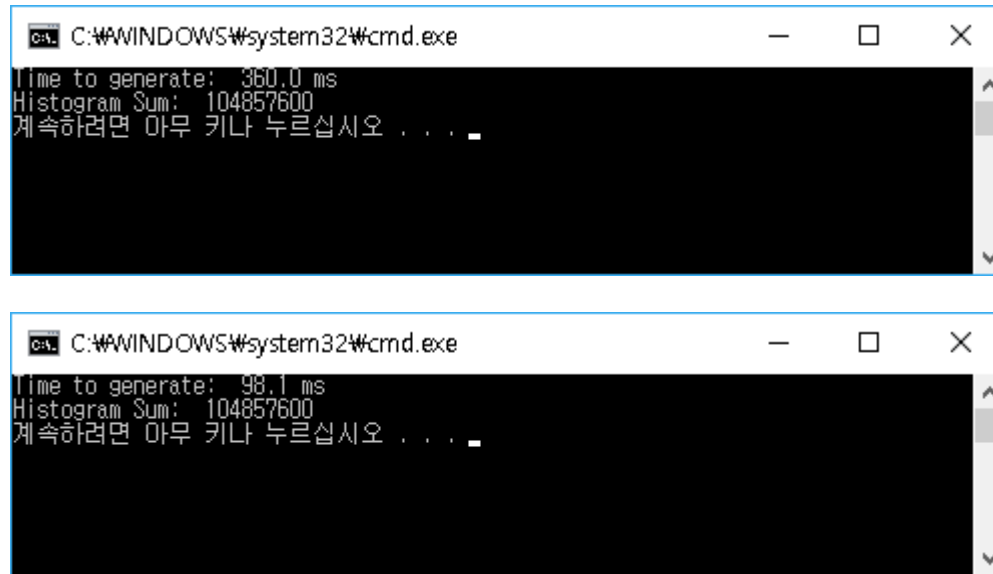
로컬 메모리 할당

로컬 histogram 구하기
atomic 연산

동기화

글로벌 histogram 결과 저장
atomic 연산

Histogram – CUDA version (Chapter 9)



```
CA: C:\WINDOWS\system32\cmd.exe
time to generate: 360.0 ms
Histogram Sum: 104857600
계속하려면 아무 키나 누르십시오 . . . . .

CA: C:\WINDOWS\system32\cmd.exe
time to generate: 98.1 ms
Histogram Sum: 104857600
계속하려면 아무 키나 누르십시오 . . . . .
```

- CPU 360ms vs. GPU 98ms

AWS에서 Histogram 예제 실행해보기

- 실습폴더를 aws 에 복사한 후 chapter 경로로 바꾼다.

```
ubuntu@ip-172-31-19-63:~$ cd cuda_by_example_cuda9.2_vs2017/
```

```
ubuntu@ip-172-31-19-63:~/cuda_by_example_cuda9.2_vs2017$ cd chapter09/
```

- nvcc 컴파일러로 CPU 코드 컴파일한다. -o는 출력파일명 이다.

```
ubuntu@ip-172-31-19-63:~/cuda_by_example_cuda9.2_vs2017/chapter09$ nvcc hist_cpu.cu -o hist_cpu
```

- ./hist_cpu 실행하여 CPU 버전의 Histogram 처리시간 체크한다.

```
ubuntu@ip-172-31-19-63:~/cuda_by_example_cuda9.2_vs2017/chapter09$ ./hist_cpu
```

```
Time to generate: 301.2 ms
```

```
Histogram Sum: 104857600
```

- 비슷한 방법으로 GPU 버전의 Histogram 처리시간 체크한다.

```
ubuntu@ip-172-31-19-63:~/cuda_by_example_cuda9.2_vs2017/chapter09$ nvcc hist_gpu_shmem_atomics.cu -o hist_gpu
```

```
ubuntu@ip-172-31-19-63:~/cuda_by_example_cuda9.2_vs2017/chapter09$ ./hist_gpu
```

```
Time to generate: 36.3 ms
```

```
Histogram Sum: 104857600
```


Further study

- Chapter 6. Constant Memory and Events
- Chapter 7. Texture Memory
- Chapter 9. Atomics
- Chapter 10. Streams
- Chapter 11. CUDA C on Multiple GPUs

Constant Memory

- 작고 빠르고 Read-only 한 메모리

- A single read from constant memory can be broadcast to other “nearby” threads, effectively saving up to 15 reads.
- Constant memory is cached, so consecutive reads of the same address will not incur any additional memory traffic.

```
HANDLE_ERROR( cudaMemcpyToSymbol( s, temp_s,  
                                   sizeof(Sphere) * SPHERES ) );
```

We use this special version of `cudaMemcpy()` when we copy from host memory to constant memory on the GPU. The only differences between `cudaMemcpyToSymbol()` and `cudaMemcpy()` using `cudaMemcpyHostToDevice` are that `cudaMemcpyToSymbol()` copies to constant memory and `cudaMemcpy()` copies to global memory.

Texture Memory

- 2차원 혹은 3차원 메모리
- X, Y 좌표로 직접 메모리 접근
- Offset 혹은 Index를 계산해서 메모리 접근할 필요가 없음

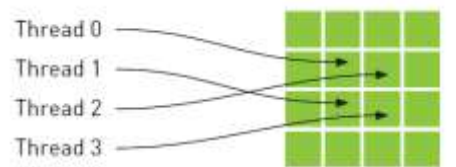


Figure 1.1 A mapping of threads into a two-dimensional region of memory

```
__global__ void copy_const_kernel( float *iptr,
                                   const float *cptr ) {
    // map from threadIdx/BlockIdx to pixel position
    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;
    int offset = x + y * blockDim.x * gridDim.x;

    if (cptr[offset] != 0) iptr[offset] = cptr[offset];
}
```

```
__global__ void copy_const_kernel( float *iptr ) {
    // map from threadIdx/BlockIdx to pixel position
    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;
    int offset = x + y * blockDim.x * gridDim.x;

    float c = tex2D(texConstSrc,x,y);
    if (c != 0)
        iptr[offset] = c;
}
```

Atomic

- 두개 이상 스레드가 동시에 메모리 Write 시도할 때 결과가 틀릴 수 있음
- 예를 들면 두 스레드 동시에 `x++` 할 때 한번만 카운트함
- `atomicAdd` 를 사용하면 여러 스레드에서 `write` 할 때 결과 보장함

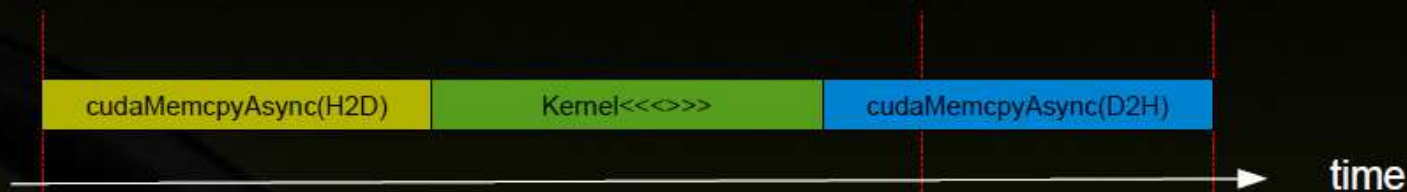
Table 9.3 Two threads incrementing the value in `x` with interleaved operations

STEP	EXAMPLE
Thread A reads the value in <code>x</code> .	A reads 7 from <code>x</code> .
Thread B reads the value in <code>x</code> .	B reads 7 from <code>x</code> .
Thread A adds 1 to the value it read.	A computes 8.
Thread B adds 1 to the value it read.	B computes 8.
Thread A writes the result back to <code>x</code> .	<code>x <- 8.</code>
Thread B writes the result back to <code>x</code> .	<code>x <- 8.</code>

Concurrency Example



- Serial



- Concurrent – overlap kernel and D2H copy



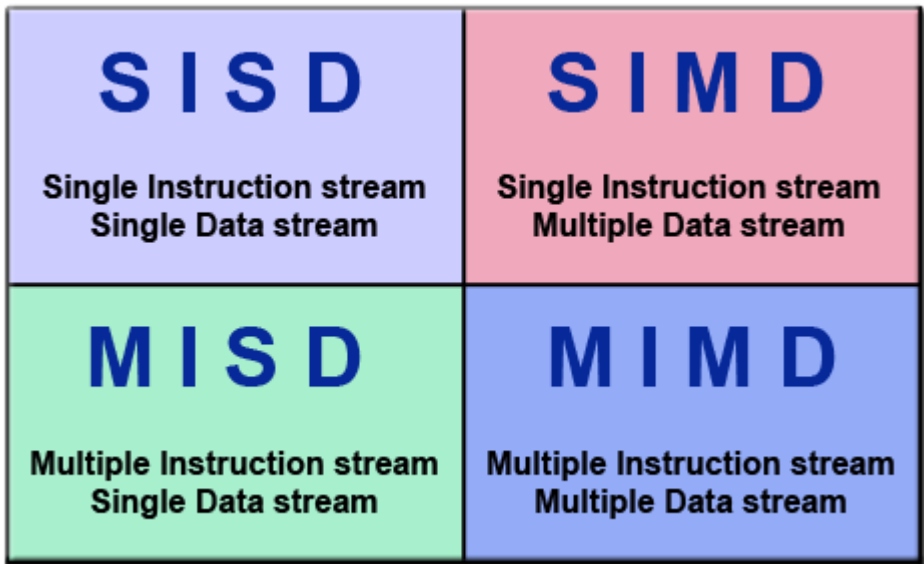
병렬처리개요

병행, 병렬, 분산

- **병행 컴퓨팅**(concurrent computing)에서 하나의 프로그램은 일정한 짧은 기간에 여러 개의 태스크가 동작하는 것을 의미 함
- **병렬 컴퓨팅**(parallel computing)에서 하나의 프로그램은 어떤 문제를 해결하기 위해 여러 개의 태스크가 밀접하게 연합하는 것을 의미 함
- **분산 컴퓨팅**(distributed computing)에서 하나의 프로그램은 어떤 문제를 해결하기 위해 다른 프로그램과의 연합이 필요 함
- *공유 메모리 프로그램을 병렬로 생각하고 분산 메모리 프로그램은 "분산"이라고 하기도 함*

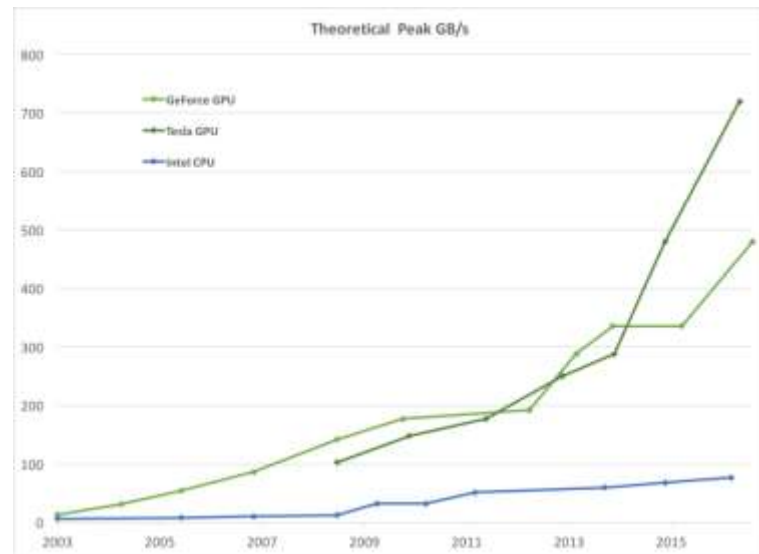
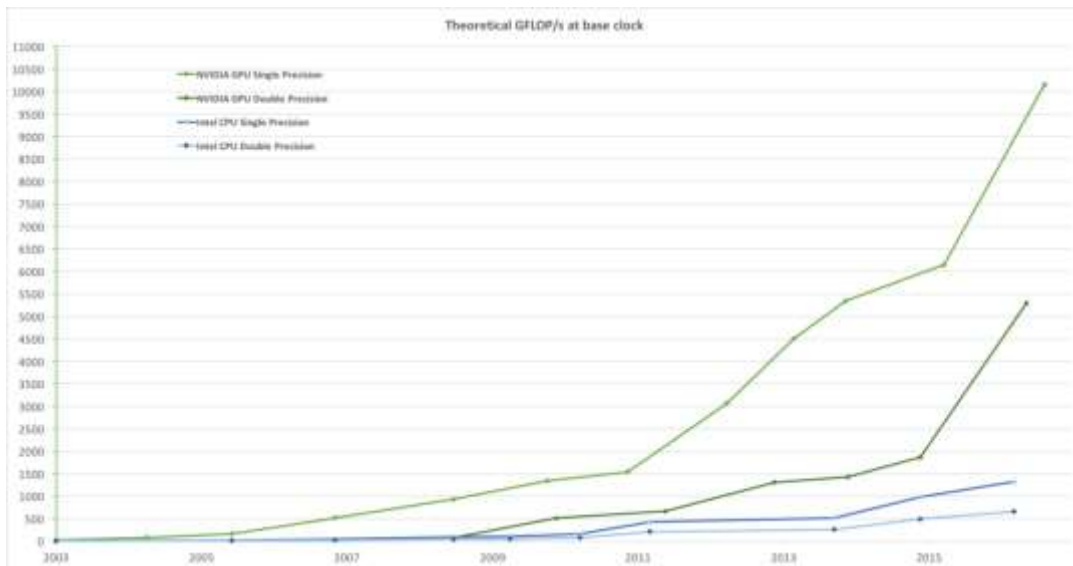
병렬처리 분류

- 대부분 Flynn의 방법을 사용하여 4가지로 구분 함
- SIMT: SIMD 과 Multithreading을 결합한 모델, GPU



GFLOP and MB

- Floating-Point Operations per Second : 연산 속도의 척도
- Memory Bandwidth : 메모리 접근 속도의 척도



CUDA 프로그래밍 모델

```
// 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 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

커널 함수: `__global__` 로
시작되고 디바이스에서
실행되는 코드

Grid 생성: `dim3` 형식으로
블록 개수 및 스레드 개수
정의 함

Kernel 실행: 위에서 정의된
MatAdd 커널을 호출 함
`<<<...>>>` 형식으로 호출 함

프로그램의 구조

```
__device__ int deviceSum ( ... )
{
    ...
}

__global__ void kernelBigSum ( ... )
{
    ...
    deviceSum ( ... );
}

__host__ int HostCallCUDA ( ... )
{
    dim3 block ( 16, 16, 1 );
    dim3 gird ( ... );
    ...
    kernelBigSum<<< grid, block >>> ( ... );
    return ...
}

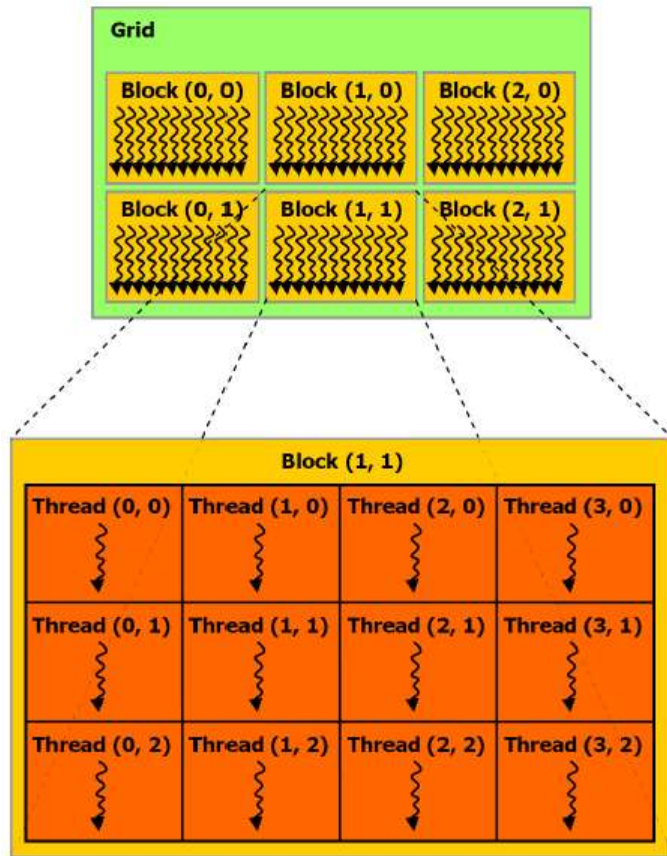
int main ( int argc, char *argv[] )
{
    ...
    HostCallCUDA ( ... );
    ...
    return 1;
}
```

- OpenCL과 비슷하게 Kernel 코드와 Host 코드로 구분되어 있다.
- `__device__` 함수
 - 커널에서 호출 함, Device에서 실행
- `__global__` 함수
 - 커널 함수, Host에서 호출 함, Device에서 실행
- `__host__` 함수
 - Host 함수, Host 에서 호출 함 , Host 에서 실행

그리드

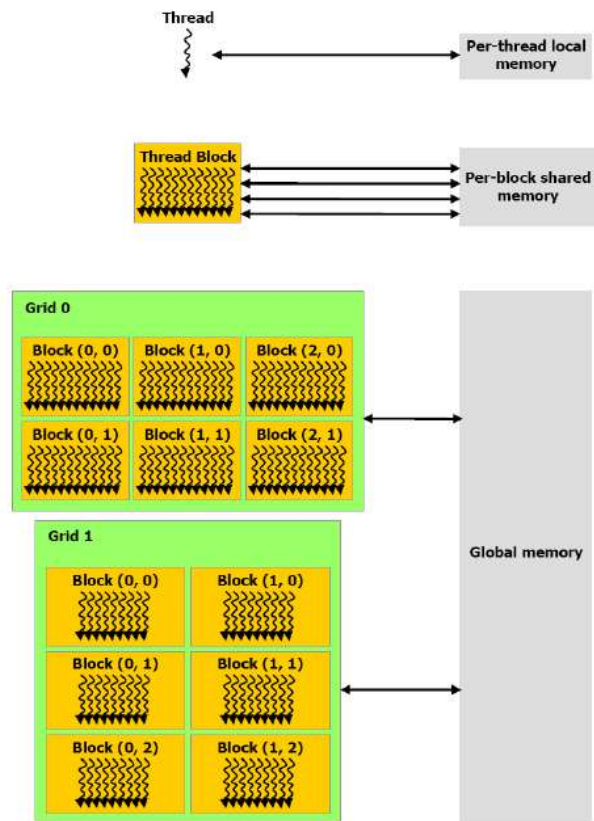
- C 함수는 N번의 연산을 N개 스레드에서 동시 실행
- 같은 Block에 있는 스레드는 협동하여 작업을 수행할 수 있으며 메모리를 공유할 수 있다.
(Shared memory)

* OpenCL 에서의 Global work size과 Local work size
에 해당되는 개념 임



메모리 유형

- Private local memory
 - 스레드 내부에서 생성되고 스레드 사이에 공유할 수 없음
- Shared memory
 - 스레드에서 생성되고 같은 블록에서 공유 할 수 있음
- Global memory
 - 디바이스 전체에서 접근 할 수 있음
- Constant memory
 - 스레드 내부에서 읽기 전용으로 접근 가능
- Texture memory
 - 디바이스 전체에서 접근 가능, text2D 함수로 접근



커널 함수

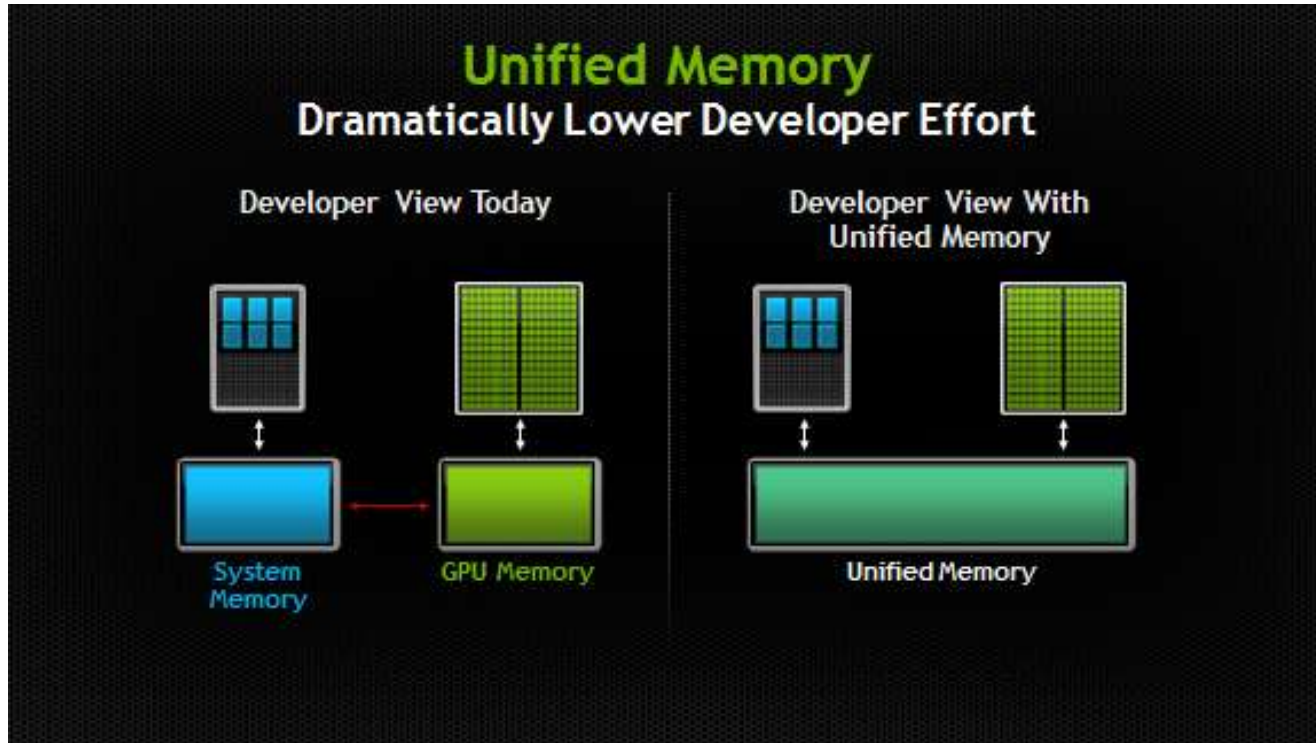
	Executed on the:	Only callable from the:
<code>__device__ float DeviceFunc()</code>	device	device
<code>__global__ void KernelFunc()</code>	device	host
<code>__host__ float HostFunc()</code>	host	host

- 함수 명 앞에 prefix 를 추가하여 Host 함수인지 Device 함수인지 구분한다.
- `__device__`, `__global__` 함수는 recursion 지원 불가
- `__device__`, `__global__` 함수는 변수 리스트가 일정해야 함

메모리 유형

	Resides in:	Has the lifetime of:	Is accessible from all the threads within:
<code>__device__</code>	global memory	application	grid
<code>__constant__</code>	constant memory	application	grid
<code>__shared__</code>	shared memory	block	block

Unified Memory in CUDA 6

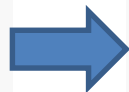


Unified Memory

```
// Allocate storage for struct and name
cudaMalloc(&d_elem, sizeof(dataElem));
cudaMalloc(&d_name, namelen);

// Copy up each piece separately, including new "name" pointer value
cudaMemcpy(d_elem, elem, sizeof(dataElem), cudaMemcpyHostToDevice);
cudaMemcpy(d_name, elem->name, namelen, cudaMemcpyHostToDevice);
cudaMemcpy(&(d_elem->name), &d_name, sizeof(char*), cudaMemcpyHostToDevice);

// Finally we can launch our kernel, but CPU & GPU use different copies of "elem"
Kernel<<< ... >>>(d_elem);
```



```
// Allocate Unified Memory -- accessible from CPU or GPU
float *x, *y;
cudaMallocManaged(&x, N*sizeof(float));
cudaMallocManaged(&y, N*sizeof(float));
...

// Free memory
cudaFree(x);
cudaFree(y);
```

기존에는 Device Memory 할당, Device 복사, 커널 실행 순서로 프로그래밍을 해야 하지만 Unified Memory를 사용하면 별도의 Device 복사가 필요 없이 Host 메모리 처럼 사용하면 된다.

<https://devblogs.nvidia.com/unified-memory-in-cuda-6/>
<https://devblogs.nvidia.com/even-easier-introduction-cuda/>

GPU Computing Applications

Libraries and Middleware

cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CUDA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica
-------------------	---------------------------------------	---------------	---------------	-----------------------------	------------------------	-----------------------

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)
---	-----	---------	----------------------------	---------------	------------------------------

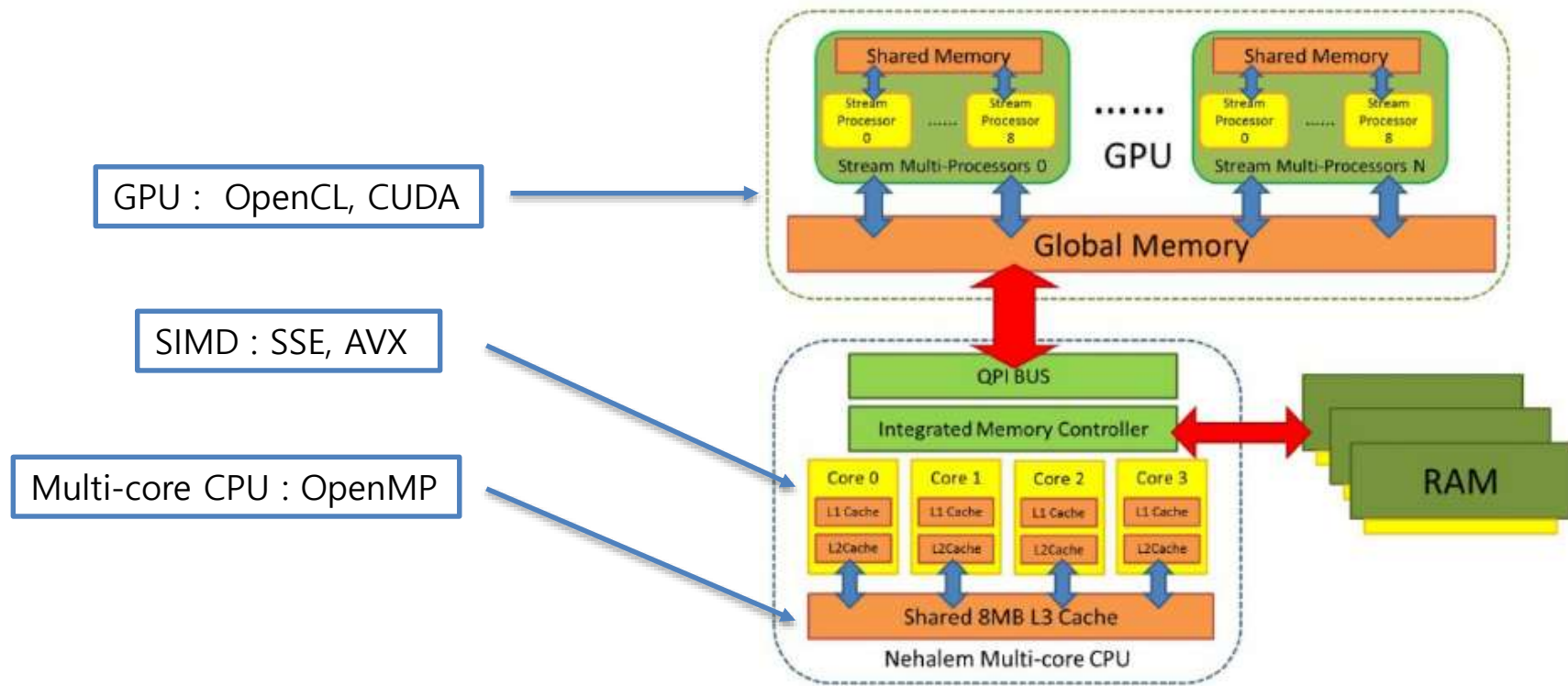


CUDA-Enabled NVIDIA GPUs

Volta Architecture (compute capabilities 7.x)				Tesla V Series
Pascal Architecture (compute capabilities 6.x)		GeForce 1000 Series	Quadro P Series	Tesla P Series
Maxwell Architecture (compute capabilities 5.x)	Tegra X1	GeForce 900 Series	Quadro M Series	Tesla M Series
Kepler Architecture (compute capabilities 3.x)	Tegra K1	GeForce 700 Series GeForce 600 Series	Quadro K Series	Tesla K Series



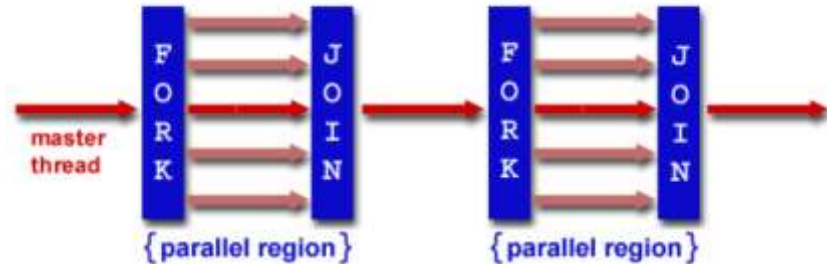
기타 병렬처리 기술



Parallel-META: A high-performance computational pipeline for metagenomic data analysis

OpenMP

- Open Multi-Processing (OpenMP)
- 공유 메모리 병렬처리를 위한 API
- 병렬코드의 블록만 설명하면 컴파일러가 처리
- #pragma 명령어를 코드에 추가하여 사용



```
#include <stdio.h>
#include <omp.h>

int main(int argc, char *argv[]) {
    int i;
    #pragma omp parallel for
    for (i = 0; i < 10; i++) {
        printf("[%d-%d] Hello World\n", omp_get_thread_num(), i);
    }
    return 0;
}
```

SSE/AVX

- Streaming SIMD Extensions (SSE) 및 Advanced Vector Extensions (AVX)
- x86 명령어 집합의 확장 SIMD명령어 집합
- SIMD 레지스터의 폭이 128비트에서 256비트로 확장
- AVX-512는 512bit 벡터 연산 가능 함

```
int main() {  
  
    __m256d veca = _mm256_setr_pd(6.0, 6.0, 6.0, 6.0);  
    __m256d vecb = _mm256_setr_pd(2.0, 2.0, 2.0, 2.0);  
    __m256d vecc = _mm256_setr_pd(7.0, 7.0, 7.0, 7.0);  
  
    __m256d result = _mm256_fmaddsub_pd(veca, vecb, vecc);  
  
    return 0;  
}
```

OpenCL

- Apple 이 개발하고 AMD, Intel, Nvidia 등과 공동으로 만든 병렬처리 표준 API
- GPU 뿐만 아니라 CPU, DSP, FPGA 에서도 사용할 수 있도록 설계
- 크로노스 그룹(Khronos Group)에서 관리하고 있음

```
__kernel void vecAdd(  __global float *A,
                      __global float *B,
                      __global float *C,
                      const unsigned int n)
{
    //Get our global thread ID
    int id = get_global_id(0);

    //Make sure we do not go out of bounds
    if (id < n)
        C[id] = A[id] + B[id];
}
```

참고자료

- <https://docs.nvidia.com/cuda/>
- <https://devblogs.nvidia.com/even-easier-introduction-cuda/>
- <https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c/>
- http://www.nvidia.co.kr/object/cuda_education_kr_old.html#1
- <http://blogs.nvidia.co.kr/2018/01/16/cuda-toolkit/>



I ♥
OPEN
SOURCE

Open your 'Hello world'

겨울방학을 오픈소스로 가득 채울 학생 개발자 찾습니다.

2018.12 ~ 2019.02.27

2. 이미지 유사도 측정하기 [참가신청](#)

! 본 주제로 참가시 프로젝트 제출 전 참가신청이 필요합니다. 12월 27일(목) 마감

Data set : 쇼핑 이미지 약 1만장 제공

[평가코드 및 세부사항 확인](#)

<http://d2campusfest.kr/6th/>

쇼핑 데이터 플랫폼 개발

#쇼핑 #데이터 #플랫폼 #개발

상시모집

모집 시

모집부문	역할	지원자격 및 우대요건
쇼핑 데이터 플랫폼 개발	<ul style="list-style-type: none"> - 쇼핑 데이터에 대한 수집 및 관리 개발 - 가격 비교를 위한 제반 개발 - 쇼핑 데이터의 검색 관련 개발 - 상품 정보에 새로운 가치를 부여할 수 있는 R&D 업무 	<p>[자격 요건]</p> <ul style="list-style-type: none"> - 기술 전파, 코드 리뷰 등 주변에 자극을 주시는 분 - '저 사람 덕분에 이런걸 알게 되다니 !' - 새로운 기술에 호기심이 많으신 분 (보는데 그치지 않고 직접 구현해보고 실험해보는 걸 좋아하시는 분) - 적극적으로 이슈 제기하고 능동적으로 해결하시는 분 - Open Source 기여 경험이 있으신 분 - Hadoop echo system 및 Kafka/Spark 등에 대한 경험이 있으신 분 - Elastic stack에 대한 경험이 있으신 분 <p>[우대 사항]</p> <ul style="list-style-type: none"> - 대규모 데이터 처리에 대한 경험을 보유하신 분 - 국내외 대형 커머스 업무 경험이 있으신 분 - 관련 커뮤니티 등에서 활발히 활동하시는 분 - Kafka 기반의 스트리밍 처리에 관심이 많으신 분 - Spring 기반 프레임워크 및 React/Node.JS/Mongo DB에 대해 능숙하신 분 - 기술 : Spring / React / Node.JS / Elastic Stack / Hadoop / Kafka / Spark / Docker / Oracle / Mongo DB / Redis 등



<https://recruit.navercorp.com/naver/job/list/developer>

부록

Compute Capability

OpenCL 용어 및 함수 비교

특 이 성 능 적 (비 전)	GPU	지문스
1.0	G80	지문스 8800 울트라, 지문스 8800 GTX, 지문스 8800 GTS(512K)
1.1	G92, G94, G96, G98, G98, G98	지문스 GTX 250, 지문스 9800 GX2, 지문스 9800 GTX, 지문스 9800 GT, 지문스 8800 GTS(G92), 지문스 8800 GT, 지문스 9600 GT, 지문스 9500 GT, 지문스 9400 GT, 지문스 8600 GTS, 지문스 8600 GT, 지문스 8500 GT, 지문스 9110M, 지문스 9200M GS, 지문스 9200M GS, 지문스 9100M G, 지문스 8400M GT, 지문스 GT 25M
1.2	GT218, GT216, GT215	지문스 GT 340M, 지문스 GT 330M, 지문스 GT 320M, 지문스 GT 315M, 지문스 GT 310M, 지문스 GT 240M, 지문스 GT 220M, 지문스 GT 210M, 지문스 GTX 360M, 지문스 GTX 350M, 지문스 GT 333M, 지문스 GT 330M, 지문스 GT 325M, 지문스 GT 340M, 지문스 GT 210M, 지문스 GT 310M, 지문스 GT 305M
1.3	GT200, GT200b	지문스 GTX 295, GTX 285, GTX 280, 지문스 GTX 275, 지문스 GTX 260
2.0	GF100, GF110	지문스 GTX 580, 지문스 GTX 580, 지문스 GTX 570, 지문스 GTX 480, 지문스 GTX 470, 지문스 GTX 465, 지문스 GTX 480M
2.1	GF104, GF106, GF108, GF114, GF116, GF117, GF119	지문스 GTX 560 Ti, 지문스 GTX 550 Ti, 지문스 GTX 460, 지문스 GTX 450, 지문스 GTX 450, 지문스 GTX 640(GDDR3), 지문스 GT 630, 지문스 GT 620, 지문스 GT 610, 지문스 GT 520, 지문스 GT 440, 지문스 GT 440P, 지문스 GT 430, 지문스 GT 430P, 지문스 GT 420P, 지문스 GTX 675M, 지문스 GTX 670M, 지문스 GT 655M, 지문스 GT 650M, 지문스 GT 625M, 지문스 GT 720M, 지문스 GT 610M, 지문스 710M, 지문스 610M, 지문스 610M, 지문스 GTX 590M, 지문스 GTX 570M, 지문스 GTX 560M, 지문스 GT 555M, 지문스 GT 550M, 지문스 GT 540M, 지문스 GT 525M, 지문스 GT 520M, 지문스 GT 520M, 지문스 GTX 485M, 지문스 GTX 470M, 지문스 GTX 460M, 지문스 GT 445M, 지문스 GT 435M, 지문스 GT 420M, 지문스 GT 415M, 지문스 710M, 지문스 410M
3.0	GK104, GK106, GK107	지문스 GTX 770, 지문스 GTX 780, 지문스 GT 740, 지문스 GTX 690, 지문스 GTX 680, 지문스 GTX 670, 지문스 GTX 660 Ti, 지문스 GTX 660, 지문스 GTX 660 Ti BOOST, 지문스 GTX 660 Ti, 지문스 GTX 660, 지문스 GTX 680M, 지문스 GTX 780M, 지문스 GTX 770M, 지문스 GTX 765M, 지문스 GTX 760M, 지문스 GTX 680M, 지문스 GTX 680M, 지문스 GTX 675M, 지문스 GTX 670M, 지문스 GTX 660M, 지문스 GT 750M, 지문스 GT 650M, 지문스 GT 745M, 지문스 GT 645M, 지문스 GT 740M, 지문스 GT 730M, 지문스 GT 640M, 지문스 GT 640M LE, 지문스 GT 735M, 지문스 GT 730M
3.2	GK204	
3.3	GK110, GK208	지문스 GTX 타이탄 Z, 지문스 GTX 타이탄 Black, 지문스 GTX 타이탄, 지문스 GTX 780 Ti, 지문스 GTX 780, 지문스 GT 640(GDDR5), 지문스 GT 630 v2, 지문스 GT 730, 지문스 GT 720, 지문스 GT 710, 지문스 GT 740M (B4H5, GDDR5)
3.7	GK210	
4.0	GM107, GM108	지문스 GTX 750 Ti, 지문스 GTX 750, 지문스 GTX 960M, 지문스 GTX 950M, 지문스 940M, 지문스 930M, 지문스 GTX 960M, 지문스 GTX 950M, 지문스 945M, 지문스 940M, 지문스 930M
5.2	GM200, GM204, GM206	지문스 GTX 맥스웰 X, 지문스 GTX 980 Ti, 지문스 GTX 980, 지문스 GTX 970, 지문스 GTX 960, 지문스 GTX 950, 지문스 GTX 750 SE, 지문스 GTX 980M, 지문스 GTX 970M, 지문스 GTX 965M
5.3	GM20B	
6.0	GP100	
6.1	GP102, GP104, GP106	엔비디아 타이탄 X, 지문스 GTX 1080, GTX 1070, GTX 1080
6.2		
7.0		
7.1		

본인의 GPU가 어떤 모델이며 Compute Capability 를 확인해야 기능이 지원되는지 알 수 있다.

GPU의 Compute Capability 확인 하는 방법
<https://ko.wikipedia.org/wiki/CUDA>

Feature Support	Compute Capability					
(Unlisted features are supported for all compute capabilities)	3.0	3.2	3.5, 3.7, 5.0, 5.2	5.3	6.x	7.x
Atomic functions operating on 32-bit integer values in global memory (Atomic Functions)	Yes					
atomicExch() operating on 32-bit floating point values in global memory (atomicExch())						
Atomic functions operating on 32-bit integer values in shared memory (Atomic Functions)						
atomicExch() operating on 32-bit floating point values in shared memory (atomicExch())						
Atomic functions operating on 64-bit integer values in global memory (Atomic Functions)						
Atomic functions operating on 64-bit integer values in shared memory (Atomic Functions)						
Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())	No					
Atomic addition operating on 64-bit floating point values in global memory and shared memory (atomicAdd())						
Warp vote and ballot functions (Warp Vote Functions)	Yes					
__threadfence_system() (Memory Fence Functions)						
__syncthreads_count(), __syncthreads_and(), __syncthreads_or() (Synchronization Functions)						
Surface functions (Surface Functions)						
3D grid of thread blocks						
Unified Memory Programming						
Funnel shift (see reference manual)	No	Yes				
Dynamic Parallelism	No		Yes			
Half-precision floating-point operations: addition, subtraction, multiplication, comparison, warp shuffle functions, conversion	No			Yes		
Tensor Core	No				Yes	

Concepts – cmp. with OpenCL

OpenCL parallelism concept	CUDA equivalent
Kernel	Kernel
Host program	Host program
NDRange (index space)	Grid
Work item	Thread
Work group	Block

CUDA와 OpenCL은 용어만 다를 뿐 프로그래밍 모델은 같다. 다만 OpenCL은 좀 더 범용으로 사용할 수 있도록 C 언어로 개발할 수 있고 CUDA는 전용 컴파일러를 사용하기 때문에 문법이 약간 다르다.
<<<>>> 커널 호출 같은 경우이다.

Dimensions and indices – cmp. with OpenCL

OpenCL API call	Explanation	CUDA equivalent
Get_global_id(0)	Global index of the work item in the x dimension	$\text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x}$
Get_local_id(0)	Local index of the work item within the work group in the x-dimension	threadIdx.x
Get_global_size(0)	Size of NDRange in the x-dimension	$\text{gridDim.x} * \text{blockDim.x}$
Get_local_size(0)	Size of each work group in the x-dimension	blockDim.x

Work Item은 Thread 에 해당되고 Work Group은 Block에 해당 된다.

Memory types – cmp. with OpenCL

OpenCL Memory types	CUDA Equivalent
Global Memory	Global Memory
Constant Memory	Constant Memory
Local Memory	Shared Memory
Private Memory / Register Memory	Local Memory / Register Memory

Kernel Functions – cmp. with OpenCL

OpenCL terminology	C for CUDA terminology
<code>__kernel</code> function (callable from device, including CPU device)	<code>__global__</code> function (callable from host, not callable from device)
No annotation necessary	<code>__device__</code> function (not callable from host)
<code>__constant</code> variable declaration	<code>__constant__</code> variable declaration
<code>__global</code> variable declaration	<code>__device__</code> variable declaration
<code>__local</code> variable declaration	<code>__shared__</code> variable declaration

Important API calls – cmp. with OpenCL

C for CUDA terminology	OpenCL terminology
cuInit()	No OpenCL initialization required
cuDeviceGet()	clGetContextInfo()
cuCtxCreate()	clCreateContextFromType()
No direct equivalent	clCreateCommandQueue()
cuModuleLoad() [requires pre-compiled binary]	clCreateProgramWithSource() or clCreateProgramWithBinary()
No direct equivalent. CUDA programs are compiled off-line	clBuildProgram()
cuModuleGetFunction()	clCreateKernel()
cuMemAlloc()	clCreateBuffer()
cuMemcpyHtoD()	clEnqueueWriteBuffer()
cuMemcpyDtoH()	clEnqueueReadBuffer()
cuFuncSetBlockShape()	No direct equivalent [functionality inclEnqueueNDRangeKernel()]
cuParamSeti()	clSetKernelArg()
cuParamSetSize()	No direct equivalent [functionality inclSetKernelArg()]
cuLaunchGrid()	clEnqueueNDRangeKernel()
cuMemFree()	clReleaseMemObj()