Compute Shader

쿠재아이
김재경

GPU의 발전

- 3차원 그래픽 처리를 빠르게 하기 위해 나온 GPU
- 처음에는 단순히 3차원 그래픽 처리만 했으나 점점 발전하다보니…

GPU의 발전

GPU를 CPU처럼 쓸 수 있겠는데?

GPU의 발전

그래서 나온 것이

GPGPU

(General Purpose Graphic Processing Unit)

GPGPU?

갑자기 왠 GPGPU?

GPGPU?

- Compute Shader는 Direct X에서 지원하는 GPGPU 기술이다
- 따라서 Compute Shader를 이해하려면 GPGPU를 이해해야 한다

GPGPU?

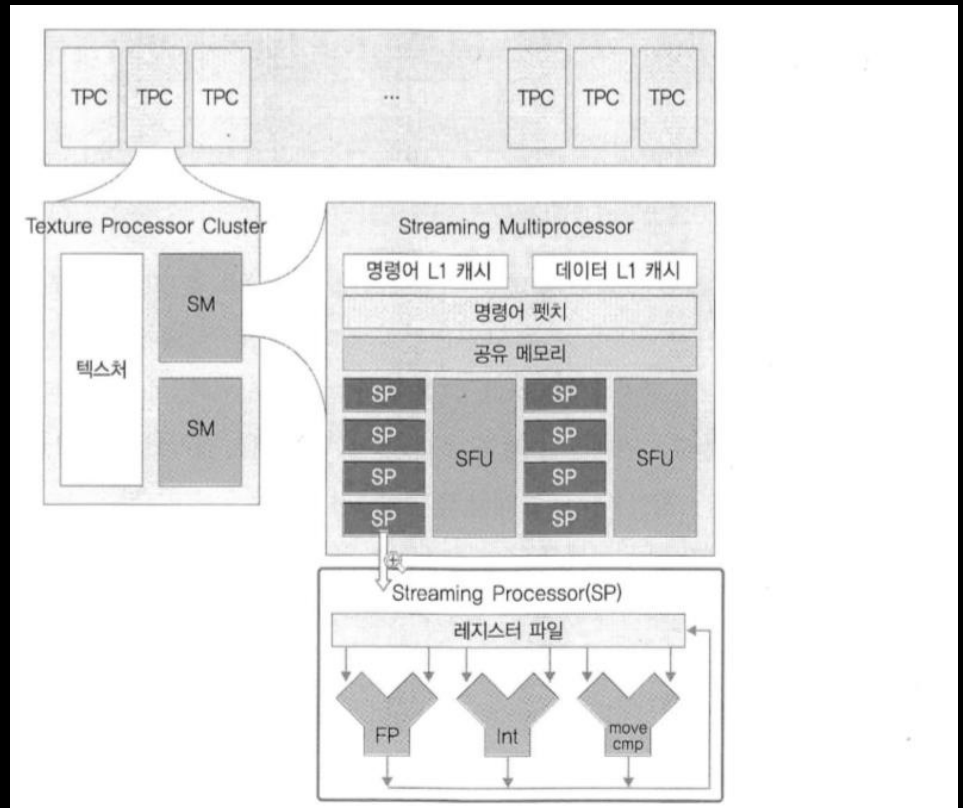


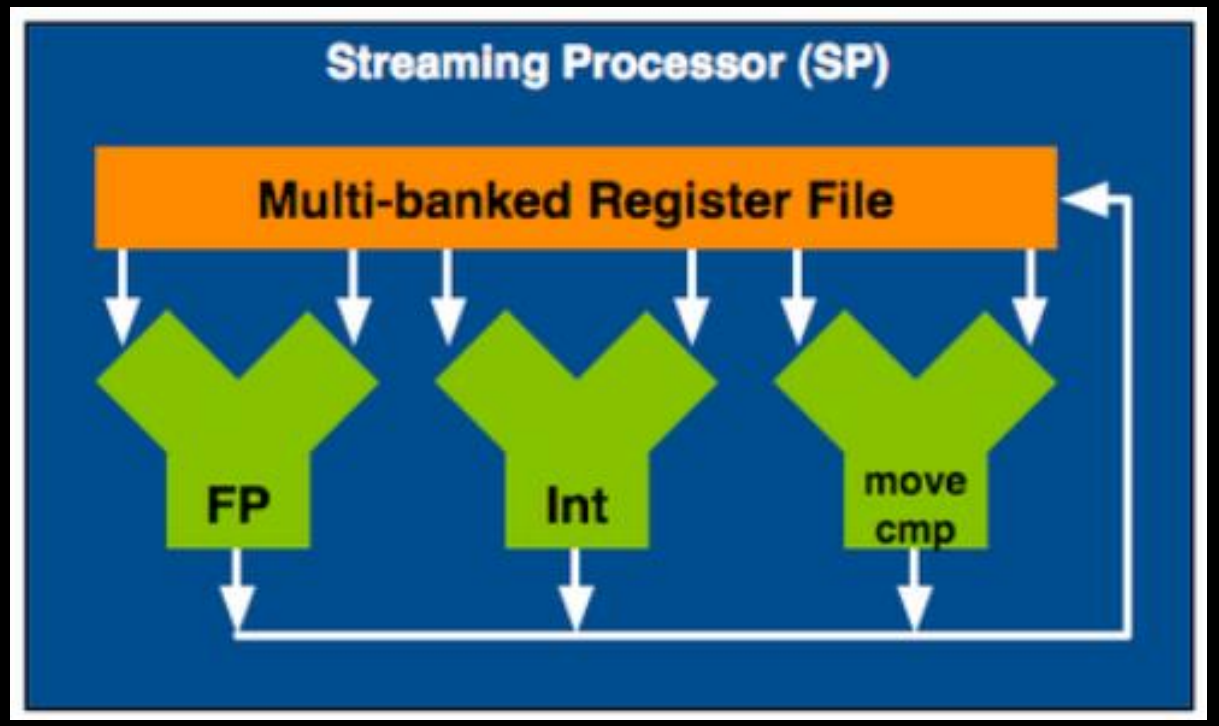
그림 11-5 nVidia GPU의 계층 구조 : SP가 모여서 SM을 이루고 SM이 모여서 TPC를 만들고, 이 TPC가 모여 GPU가 된다[3].

출처 : 멀티코어 CPU 이야기 198p

그리고 GPGPU를 이해하기 위해서는 GPU를 알아야 한다

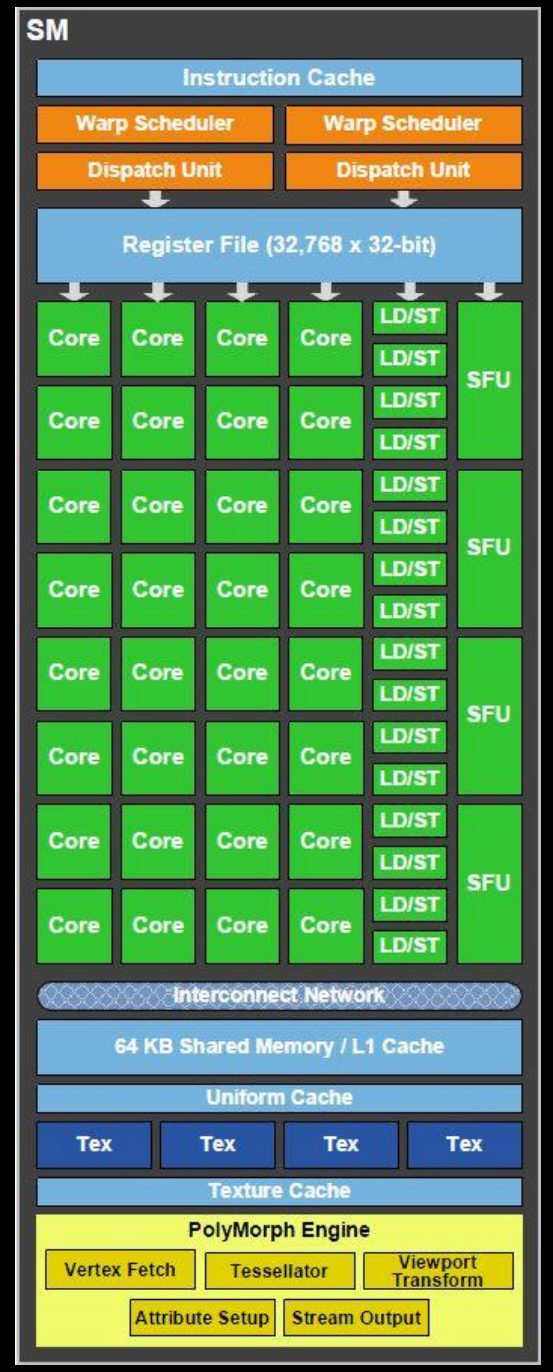
SP(Streaming Processor)

- GPU의 가장 기본적인 Unit
- 기본 연산(Thread)을 하는 모듈



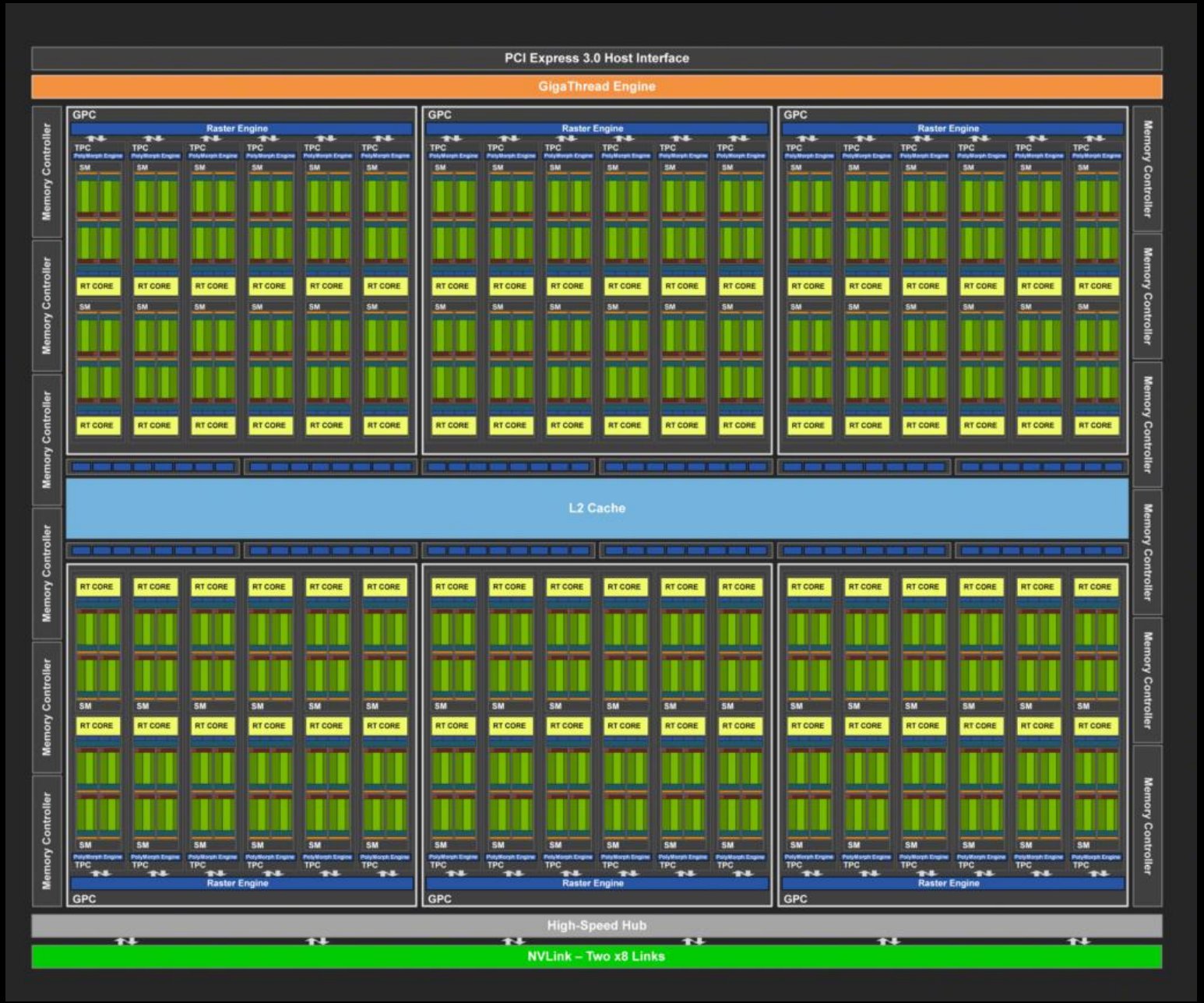
SM(Streaming Multiprocessor)

- SP들이 모인 것
- CPU의 Core에 해당
- 한 번에 수행되는 연산은 동일한 종류만 가능



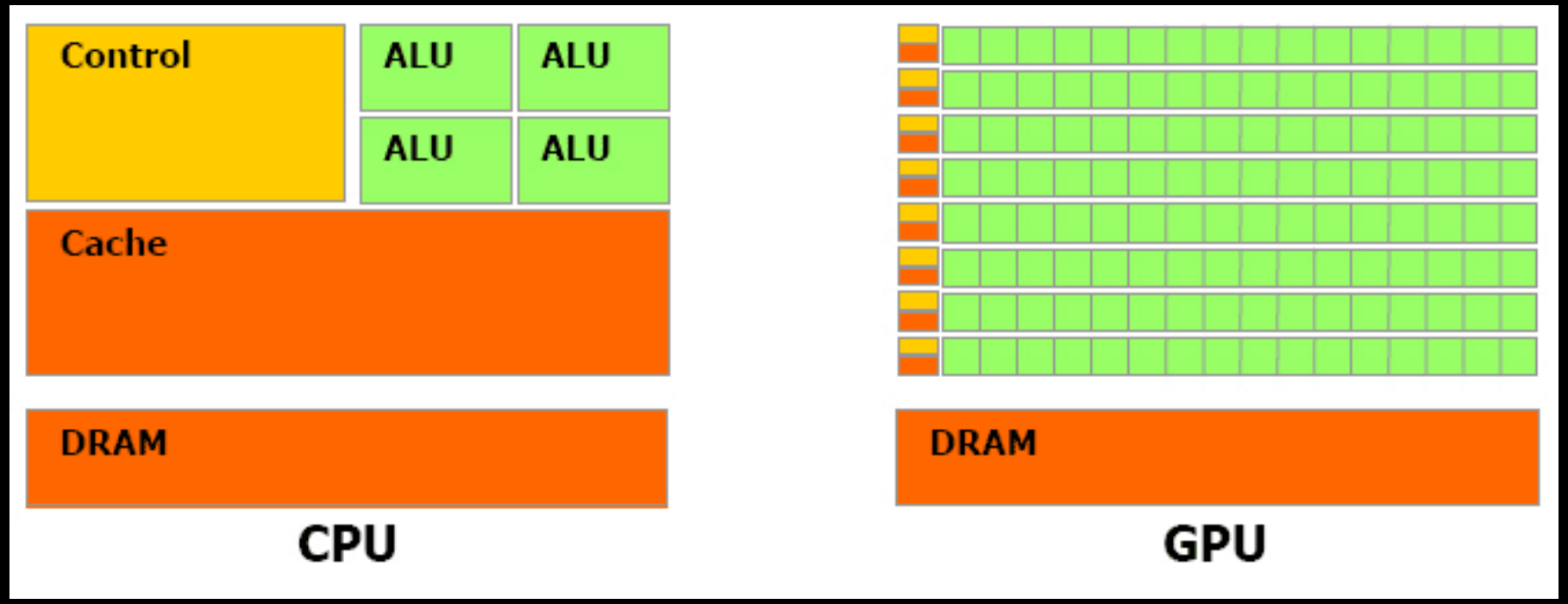
GPU

- SM들이 모인 것


PCI Express 3.0 Host Interface
GigaThread Engine
Memory Controller
GPC
Raster Engine
TPC
SM
RT CORE
L2 Cache
High-Speed Hub
NVLink – Two x8 Links

GPU

- 범용 목적이 아닌 그래픽 처리를 위해 만들어짐
- 그러기 위해 단순한 연산을 병렬로 처리하는 구조로 되어있다



GPU

- GPU 코어 1개는 CPU 코어 1개에 비해 연산 성능이 많이 떨어지지만 많은 수의 코어를 사용하면 병렬 산술 연산 성능은 CPU 성능을 능가한다

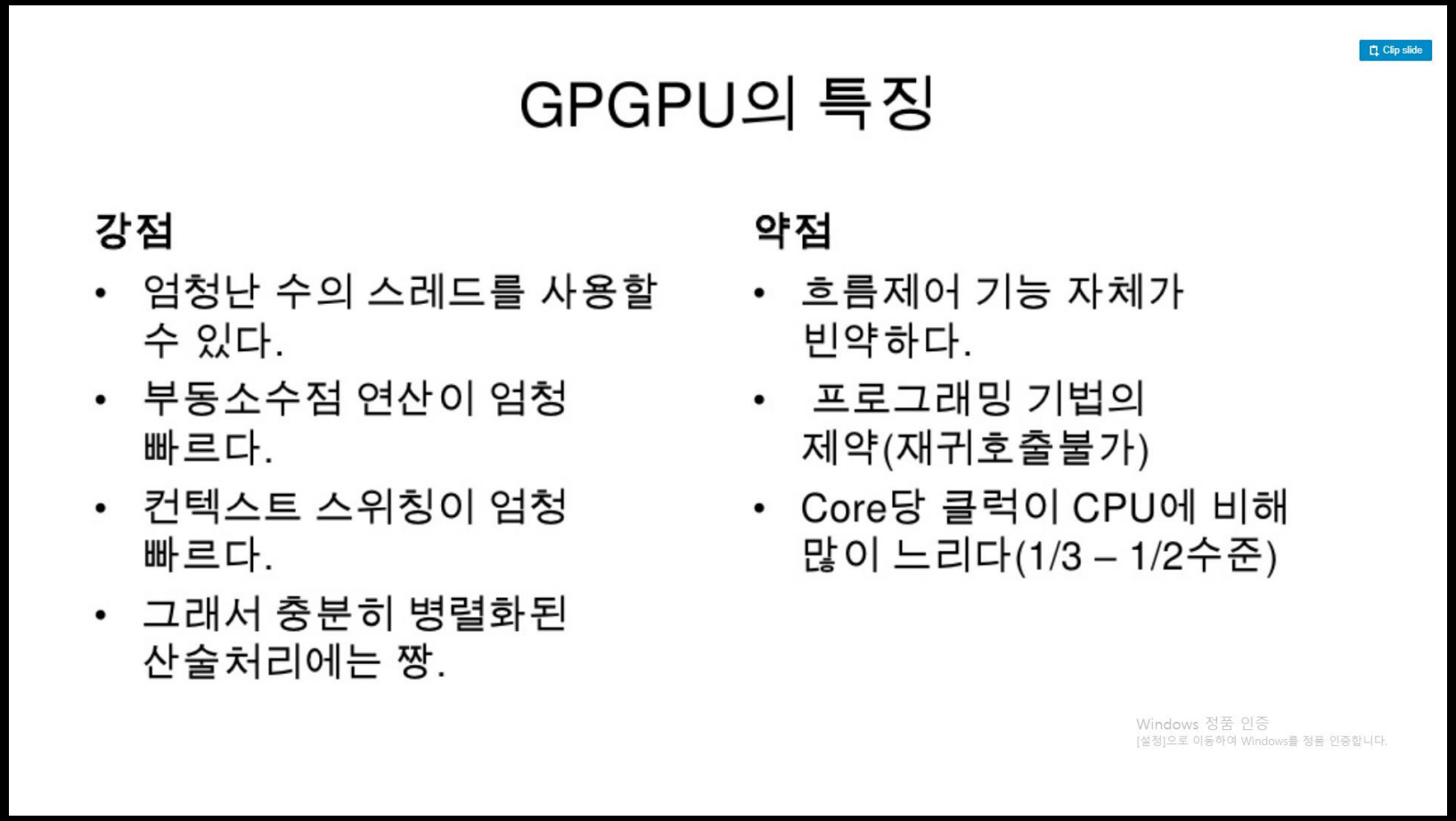
성능이 떨어지면 숫자로 압도한다

GPGPU

GPU의 병렬 산술 연산을 범용적으로 활용하는 것

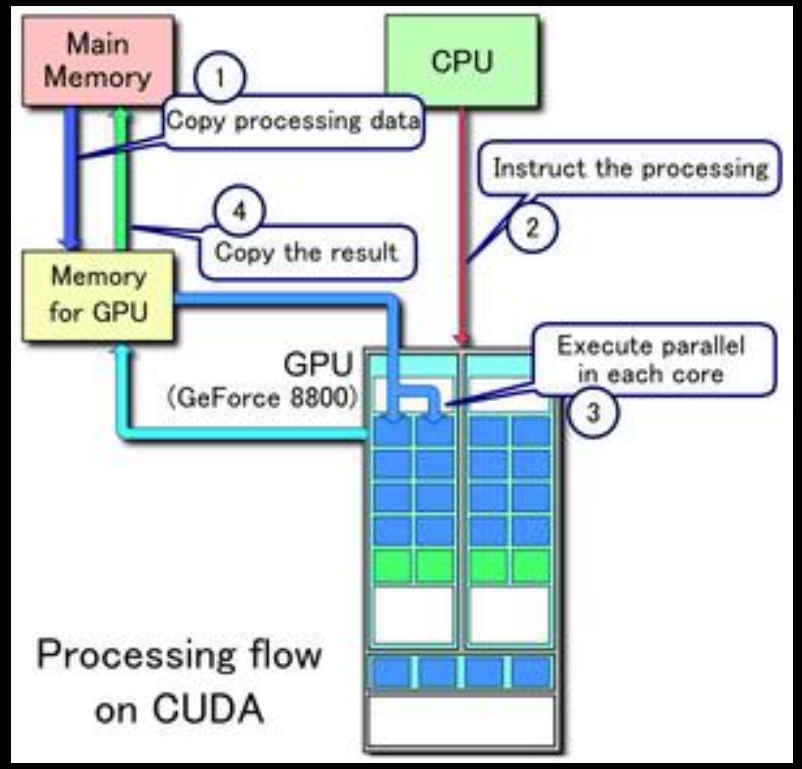
CPU보다 빠르니까…

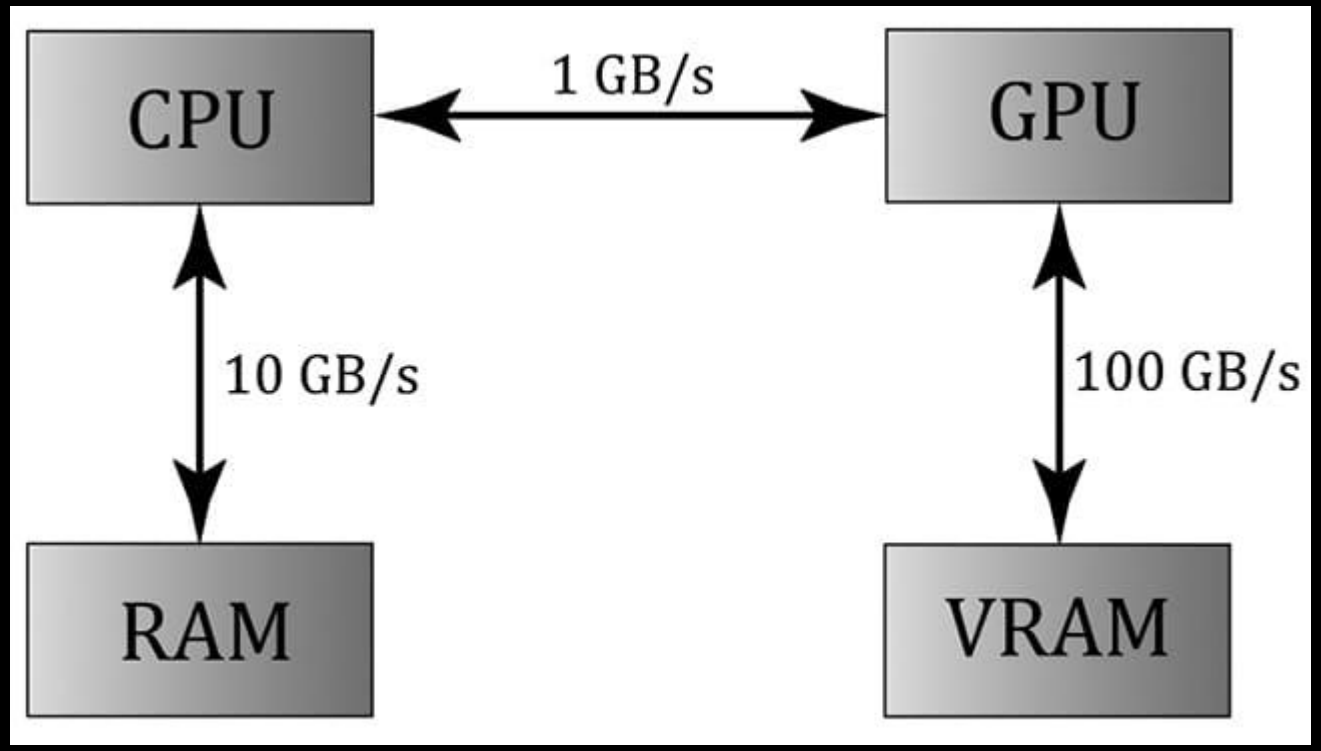
GPGPU 특징



출처 : GPGPU(CUDA)를 이용한 MMOG 캐릭터 충돌처리 16p

GPGPU 특징





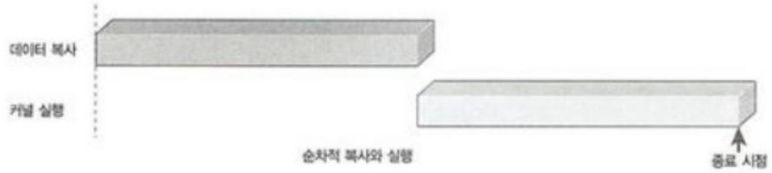
CPU와 GPU 사이의 메모리 전송이 병목이 될 수 있다

GPGPU 특징

CUDA Streaming

Clip slide

- 연산 속도에 비해 Host DRAM 과 GPU DRAM의 Data 전송속도가 느림
- 큰 Data를 가공할 경우 연산시간보다 Data 전송시간의 비중이 더 커짐

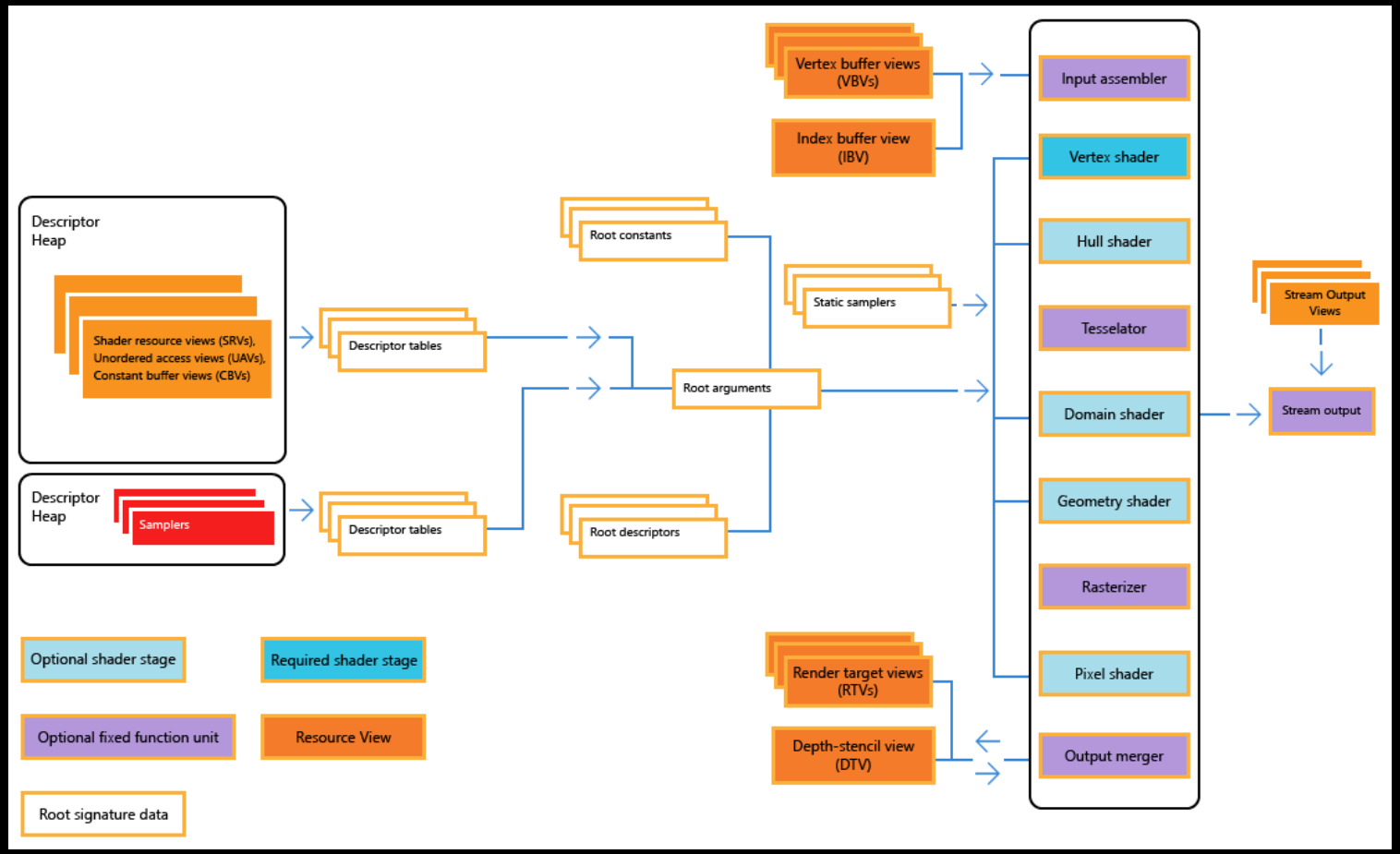


- Data 전송 및 연산시간을 작은 단위로 나눠서 순차적으로 진행

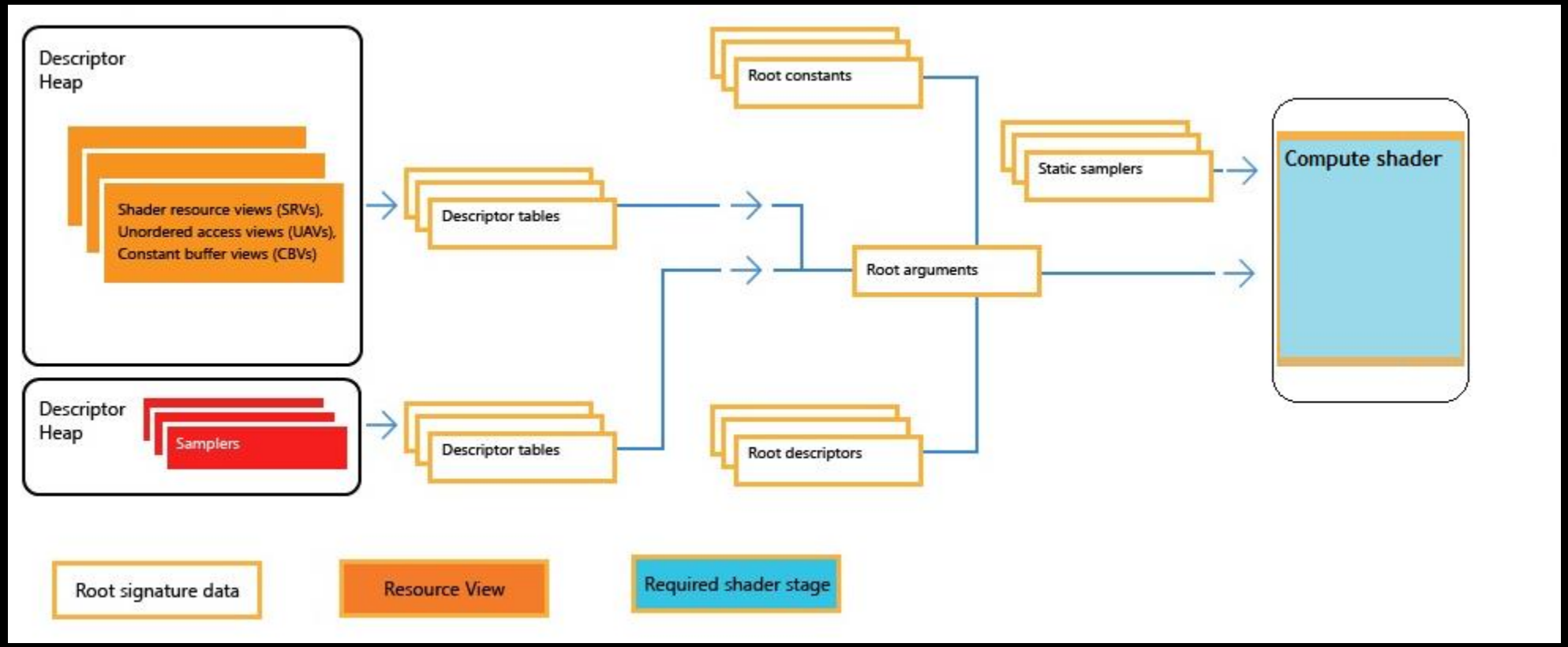
Windows 정품 인증
[설정]으로 이동하여 Windows를 정품 인증합니다.

출처 : 병렬프로그래밍과 Cuda 18p

Compute Shader



Compute Shader



Compute Shader는 렌더링 파이프라인의 일부가 아니라 개별적인 처리 단위이다

Compute Shader

- Thread : 기본 연산을 의미
- Thread Group : Thread들이 모인 것. SM에서 실행된다
- Numthreads : 1개 Thread Group의 Thread 개수. 256개를 권장

Compute Shader

```
struct Data
{
    float3 v1;
    float2 v2;
};

StructuredBuffer<Data> gInputA : register(t0);
StructuredBuffer<Data> gInputB : register(t1);
RWStructuredBuffer<Data> gOutput : register(u0);


[numthreads(32, 1, 1)]
void CS(int3 dtid : SV_DispatchThreadID)
{
    gOutput[dtid.x].v1 = gInputA[dtid.x].v1 + gInputB[dtid.x].v1;
    gOutput[dtid.x].v2 = gInputA[dtid.x].v2 + gInputB[dtid.x].v2;
}
```

Compute Shader

```
Texture2D gInputA;
Texture2D gInputB;
RWTexture2D<float4> gOutput;

[numthreads(16, 16, 1)]
void CS(int3 dtid : SV_DispatchThreadID)
{
    gOutput[dtid.xy] = gInputA[dtid.xy] + gInputB[dtid.xy];
}
```

Compute Shader

- 만약 Indexing이 필요 없다면

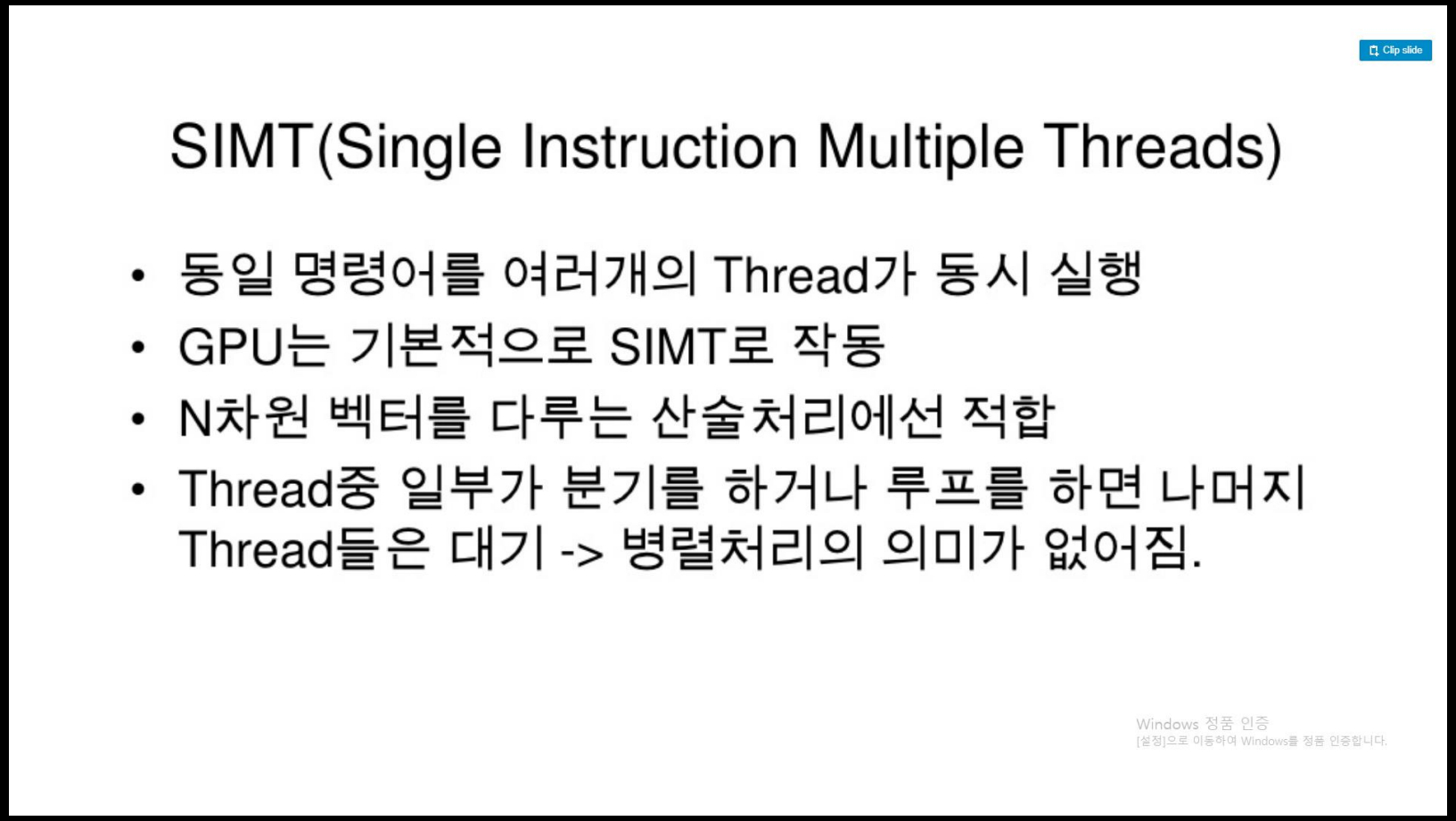
```
struct Particle
{
    float3 Position;
    float3 Velocity;
    float3 Acceleration;
};

float TimeStep = 1.0f / 60.0f;

ConsumeStructuredBuffer<Particle> gInput;
AppendStructuredBuffer<Particle> gOutput;

void CS()
{
    // 입력 버퍼에서 자료 원소 하나를 소비한다.
    Particle p = gInput.Consume();

    p.Velocity += p.Acceleration * TimeStep;
    p.Position += p.Velocity * TimeStep;

    // 정규화된 벡터를 출력 버퍼에 추가한다.
    gOutput.Append(p);
}
```

Compute Shader

- Thread Group Shared Memory
- Thread Group 안에서 공유되는 메모리로 최대 32KB
- 한 번 선언하면 Thread Group 마다 생성

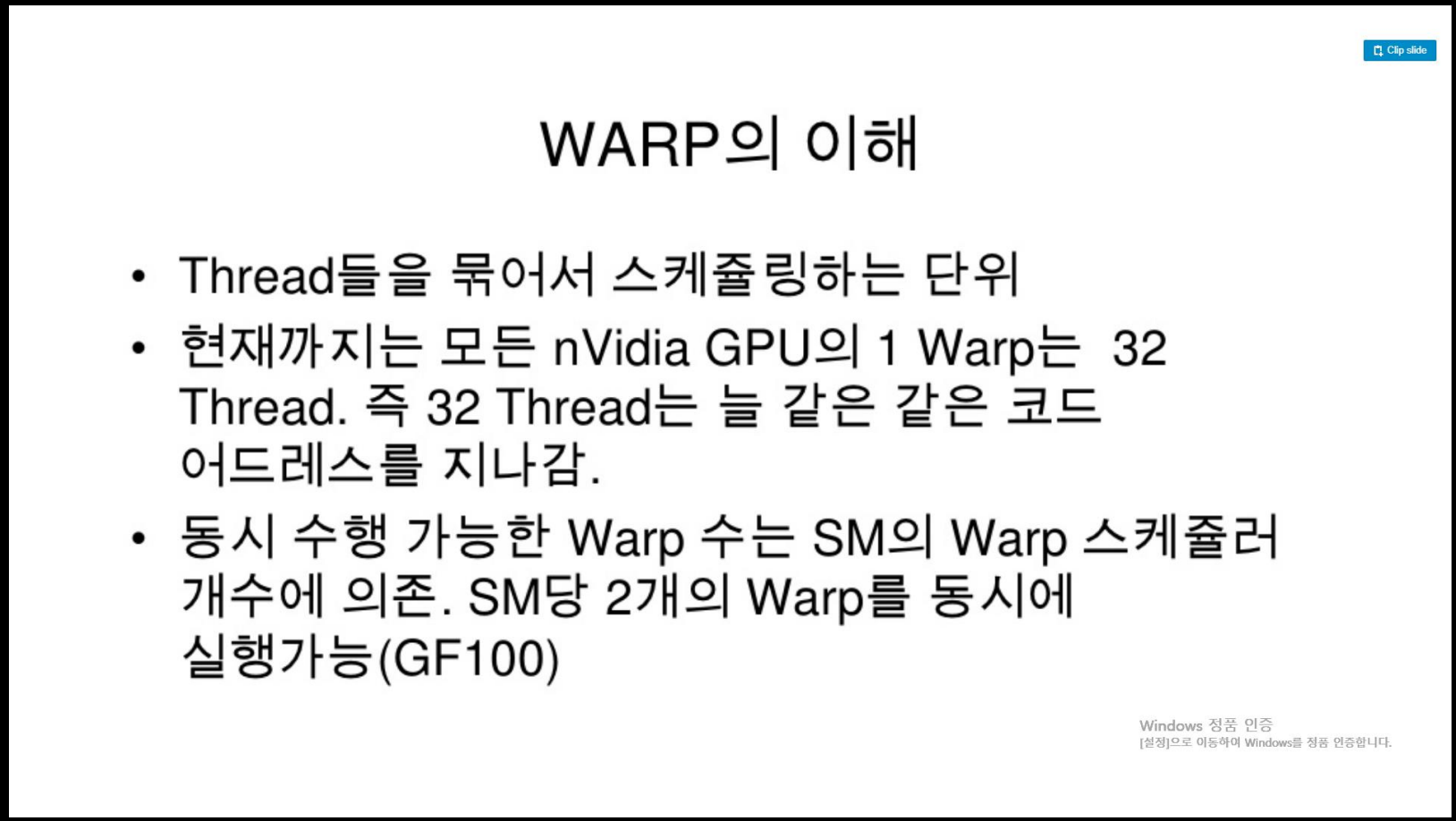
```hlsl
Texture2D gInput;
RWTexture2D<float4> gOutput;

groupshared float4 gCache[256];

[numthreads(256, 1, 1)]
void CS(int3 groupThreadID : SV_GroupThreadID,
        int3 dispatchTreadID : SV_DispatchThreadID)
{
    // 각 스레드는 텍스처에서 표본을 추출해서 공유 메모리에 저장한다.
    gCache[groupThreadID.x] = gInput[dispatchThreadID.xy];

    // 모든 스레드가 이 지점에 도달할 때까지 기다린다.
    GroupMemoryBarrierWithGroupSync();

    // 기타 작업...
    float4 left = gCache[groupThreadID.x - 1];
    float4 right = gCache[groupThreadID.x + 1];

    ...
}
```

SIMT



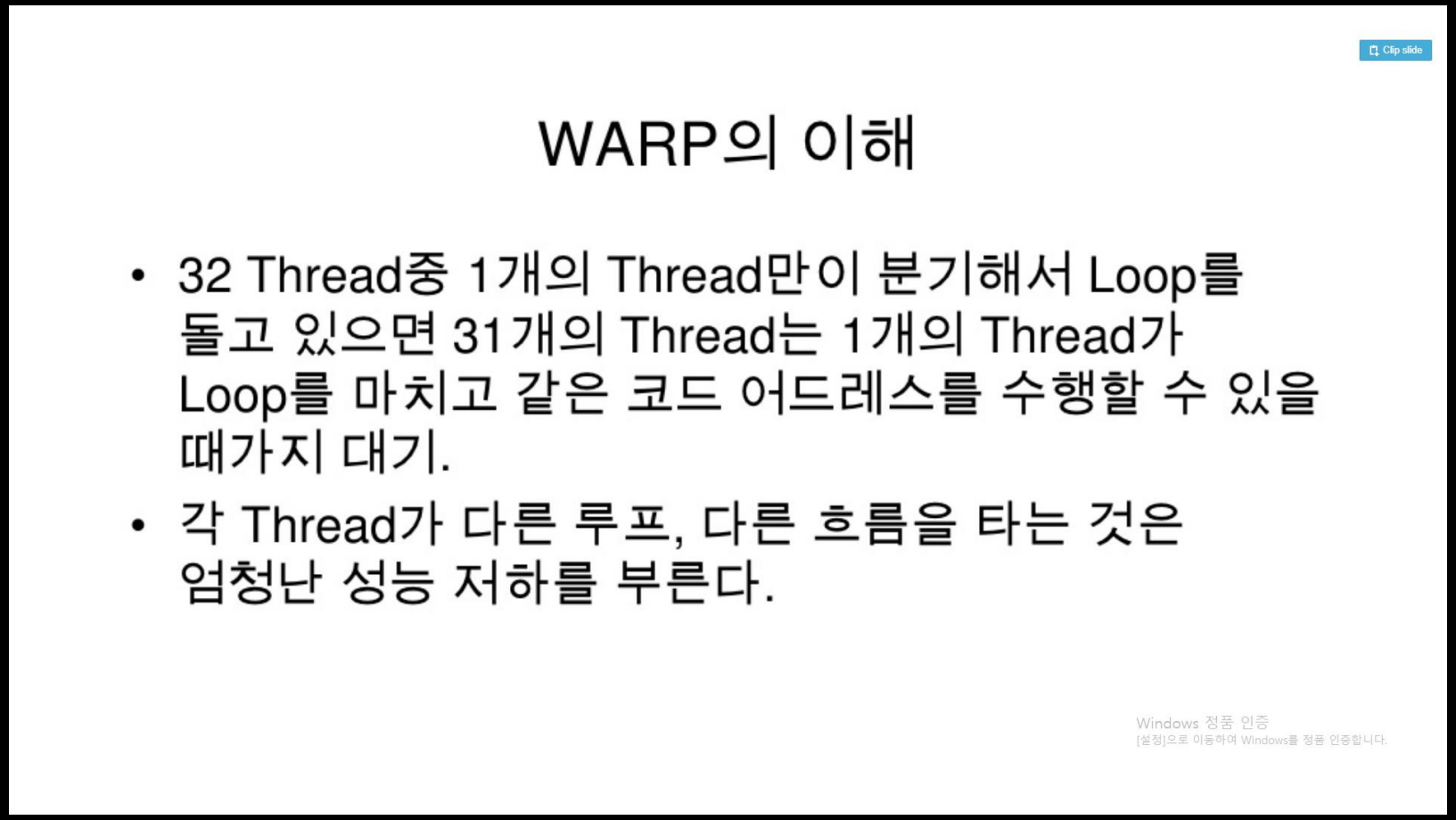
출처 : GPGPU(CUDA)를 이용한 MMOG 캐릭터 충돌처리 41p

Warp의 이해



출처 : GPGPU(CUDA)를 이용한 MMOG 캐릭터 충돌처리 42p

Warp의 이해



출처 : GPGPU(CUDA)를 이용한 MMOG 캐릭터 충돌처리 43p

Warp의 이해

Clip slide

Worst Case

```
DWORD ThreadIndexInBlock = threadIdx.y * blockDim.x + threadIdx.x;
DWORD UniqueThreadIndex = UniqueBlockIndex * ThreadNumPerBlock + ThreadIndexInBlock;

int CountList[32] = {0,1,2,3,...,31};

// 32개의 스레드가 동시에 루프를 돈다.
for (int i=0; i<CountList[UniqueThreadIndex]; i++)
{
    ...
}
// CountList[UniqueThreadIndex] == 0 인 스레드도 CountList[UniqueThreadIndex] == 31인 스레드를 대기
// 모든 스레드가 가장 오래 걸리는 스레드에 맞춰서 움직인다.즉
// 모든 스레드가 31번 루프를 도는 것과 같다.
```

Windows 정품 인증
[설정]으로 이동하여 Windows를 정품 인증합니다.

출처 : GPGPU(CUDA)를 이용한 MMOG 캐릭터 충돌처리 44p

참고자료

컴퓨트 쉐이더에서 SPIR까지
https://blog.naver.com/tigerjk0409/221225361148

Compute shader DX11
https://www.slideshare.net/leemwymw/compute-shader-dx11

Direct x 11 입문
https://www.slideshare.net/JinWooLee2/direct-x-11

DirectX11 - 계산 파이프라인란?
https://grandstayner.tistory.com/entry/DirectX11-%EA%B3%84%EC%82%B0-%ED%8C%8C%EC%9D%B4%ED%94%84%EB%9D%BC%EC%9D%B8

참고자료

DirectX 11 기초 (Tutorial) - 컴퓨트 셰이더

https://m.blog.naver.com/PostView.nhn?blogId=sorkelf&logNo=40172400532&proxyReferer=https%3A%2F%2Fwww.google.com%2F

GPGPU(CUDA)를 이용한 MMOG 캐릭터 충돌처리

https://www.slideshare.net/dgtman/gpgpummog

How does the GPU work? 1편

http://blog.naver.com/PostView.nhn?blogId=padagi20&logNo=221204107315&parentCategoryNo=&categoryNo=22&viewDate=&isShowPopularPosts=true&from=search

CUDA 프로그래밍 모델

http://haanjack.github.io/cuda/2016/03/27/cuda-prog-model.html

병렬프로그래밍과 Cuda

https://www.slideshare.net/seokjoonyun9/cuda-33834381

Q/A