CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

Thread and GPU

쓰레드와 GPU

내용 contents

- **CUDA hardware**
- **Transparent Scalability**
  - thread block scheduling
- **Thread and Warp**
- **CUDA thread scheduling**
  - warp scheduling
- **Overall execution model**

- **실습: warp-lane.cu – warp ID, lane ID check**

Underlying CUDA hardware

- **NVIDIA GeForce 8 series (shortly, G80)**
  - first released in 2006
  - includes GeForce 8800, 8600, 8400 and more
  - **unified shader architecture**
    - graphics pipeline 과 CUDA computing device 가 **혼합**
- **현재의 CUDA hardware**
  - **보다 단순**해 지고, computing device 역할이 강조됨
  - 여전히 graphics card 로서의 역할도 수행
    - 예: texture unit

NVIDIA Tesla GP100 GPU Hardware

- **1 GPU = 6 GPCs (graphics processing cluster)**
  - 1 GPC = 10 Pascal SMs → totally, 60 SMs
- **SM = streaming multiprocessor**
  - the fundamental processing unit

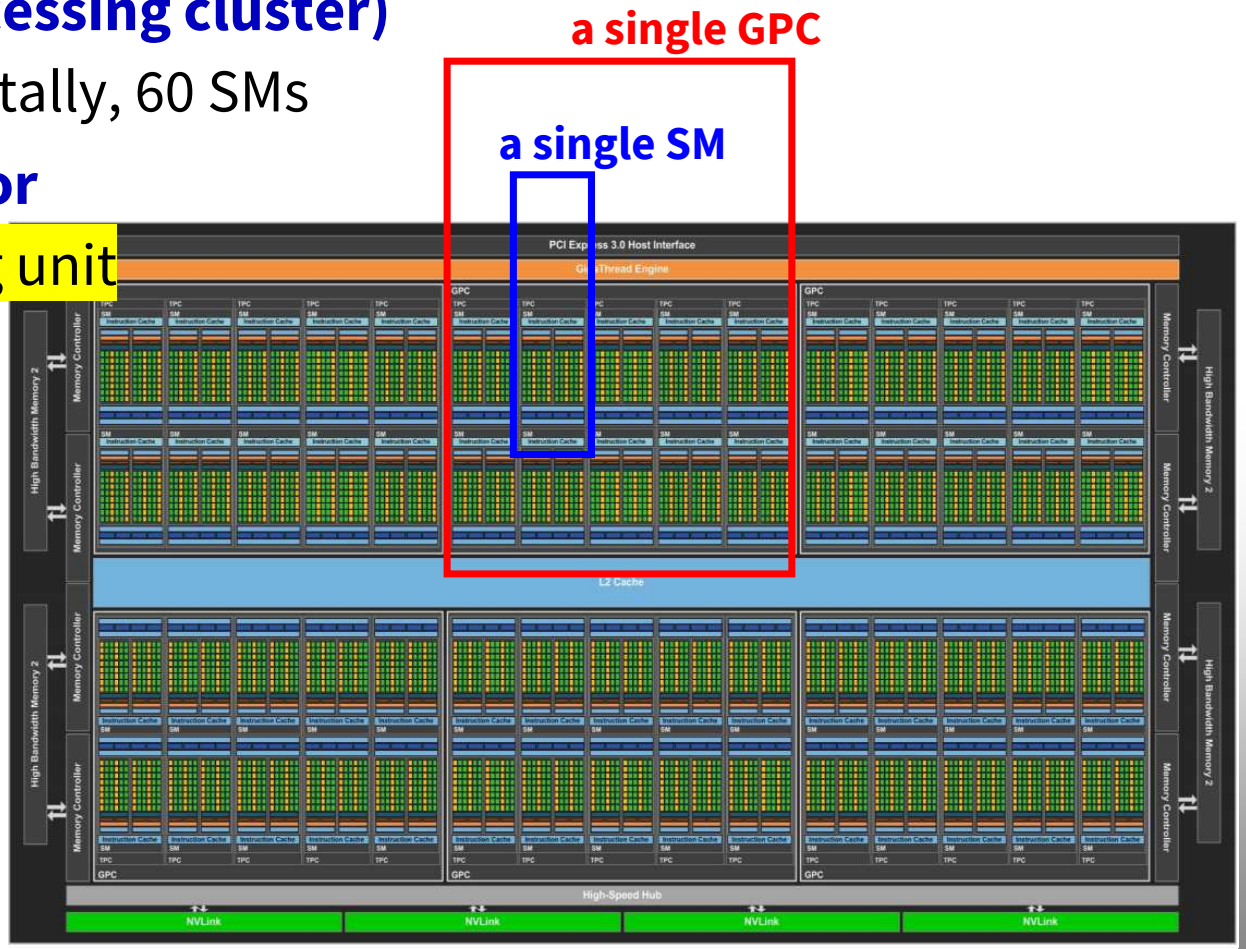


Figure 7. Pascal GP100 Full GPU with 60 SM Units, from "NVIDIA Tesla P100 White Paper", https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

NVIDIA Tesla GP100 GPU Hardware 계속

- **SM : streaming multiprocessor**
  - 최신 구조는 SM 내에 **독립 unit**
- **1 unit = 32 SP + 16 DP + 8 SFU + 2 Tex**
- **SP (streaming processor) = FP32 core**
  - Scalar ALU for a single CUDA thread
- **DP (double precision) = FP64 core**
- **SFU (special function unit)**
  - for complex math functions: sin, cos, square root, ...
  - execute one special instruction per thread, per clock
- **Tex (texture processor)**
  - for graphics purpose

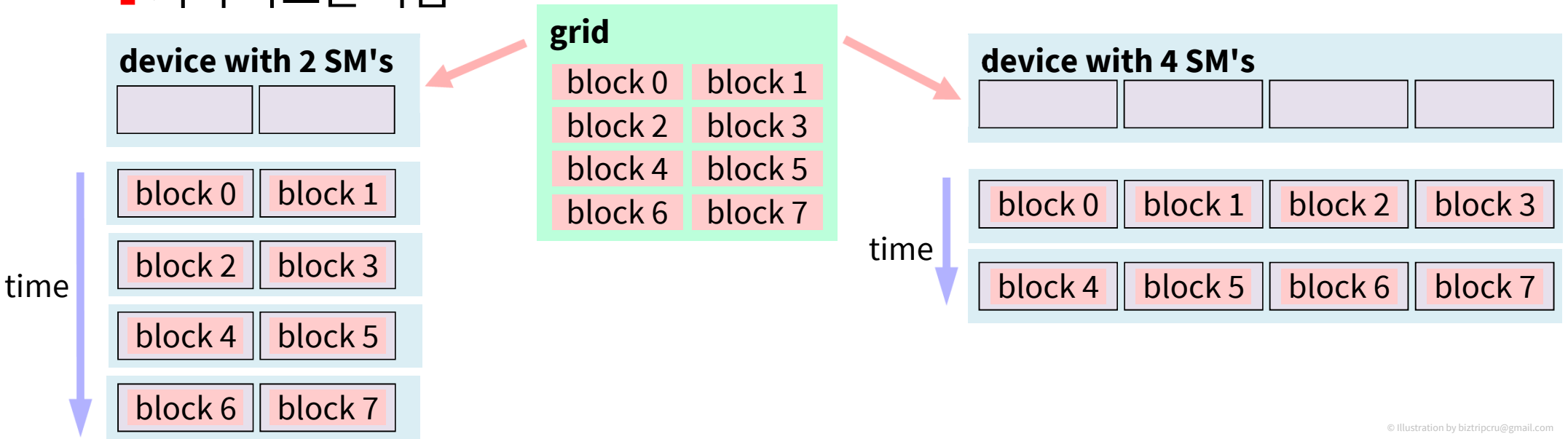
Figure 8. Pascal GP100 SM Unit, from "NVIDIA Tesla P100 White Paper", https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Transparent Scalability 확장성

- **Scalability Issue:**
  - SM is the fundamental processing unit.
  - **CUDA device 는 매우 다양할 수 있음**
    - 저가 스마트폰, 태블릿 : 1~4개의 SM 만 있는 저가 GPU 채용
    - 일반적인 PC : 6 ~ 200 개 정도의 SM을 가진 고가 GPU
    - high-end 워크스테이션 : 1000+ 개의 SM 가능
- **CUDA가 제시한 해결책 solution**
  - **thread block 개념의 도입**
  - SM 1개가 thread block 1개를 처리
  - **grid – block – thread** 의 계층 구조 hierarchy가 필요한 이유

Transparent Scalability 확장성

- **같은 CUDA 프로그램을 다양한 CUDA device 에서 실행 가능**
  - thread block 들이 SM 에 자유롭게 assign 되어서 처리되는 구조
  - Each block can execute **in any order** relative to other blocks.
  - 처리 속도는 다름



NVIDIA Tesla GP100 GPU Hardware 계속

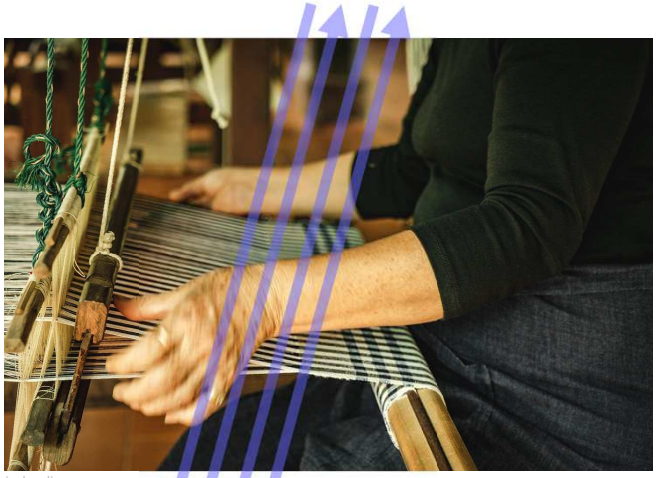
- **Streaming Multiprocessor (SM)**
- **1개의 CU (control unit)**
  - 32 core → 물리적으로 physically **32 thread 가 동시 실행**
  - 1개의 **warp scheduler** …
  - 32 thread → **같은 instruction** 을 동시 실행
- **time sharing 관점**
  - SM 1개는 2048+ thread 를 **동시 관리**
  - memory의 느린 반응 속도 해결

Figure 8. Pascal GP100 SM Unit, from "NVIDIA Tesla P100 White Paper", https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Thread and Warp

- **a single thread**
  - 독립적 실행의 단위

weft thread : warp threads 사이를 관통해서 엮이는 실 thread 1줄



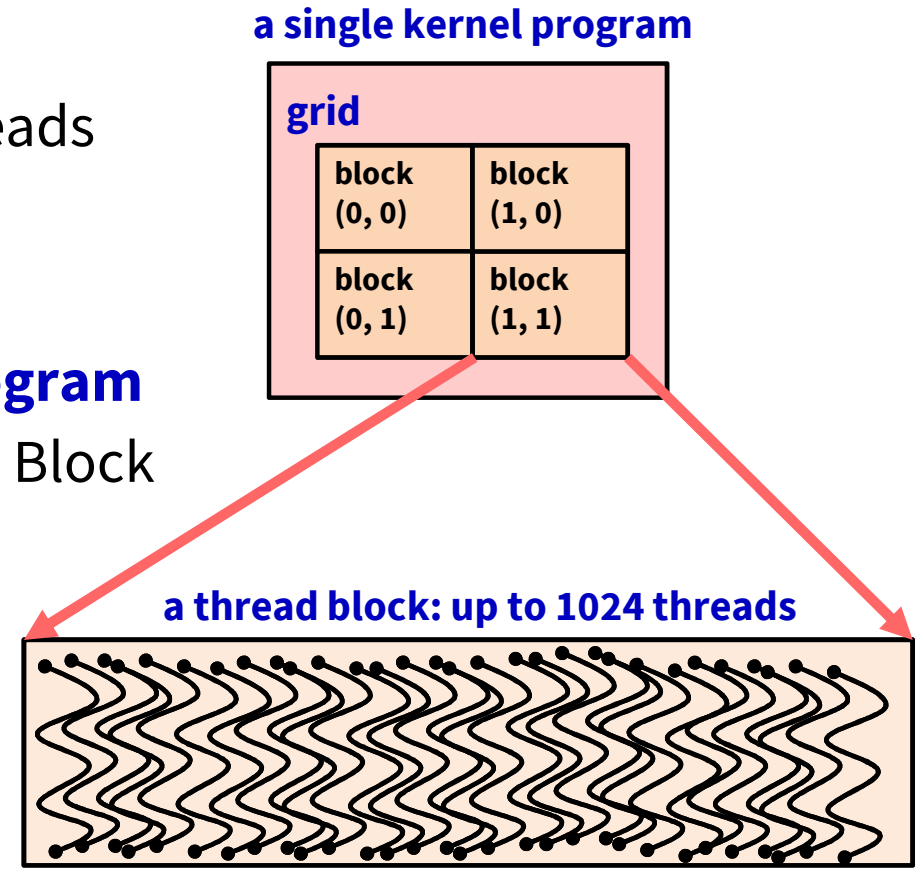
pixabay license
https://pixabay.com/ko/photos/%EC%A7%81%EB%AC%BC-%ED%95%B8%EB%93%9C-%EC%B9%B4%ED%8E%AB-%EB%B2%A0%ED%8B%80-4079366/

warp threads : 평행하게 함께 움직이는 여러 개의 실 thread

- **warp = 32 threads**
  - 물리적으로 (진짜) 동시 실행되는 thread 그룹
- **lane**
  - warp 내에서 각 thread의 index
  - 0 ~ 31

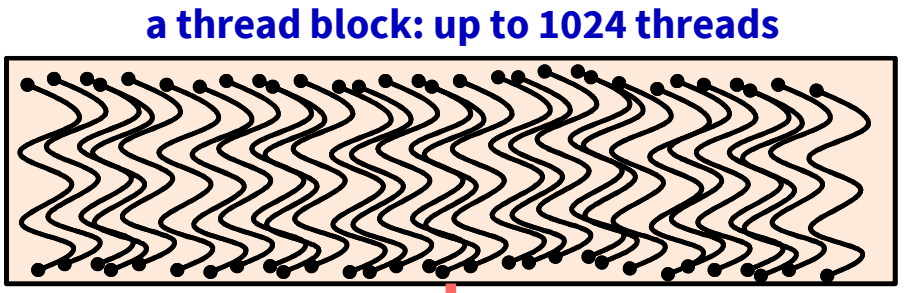
CUDA Thread Block

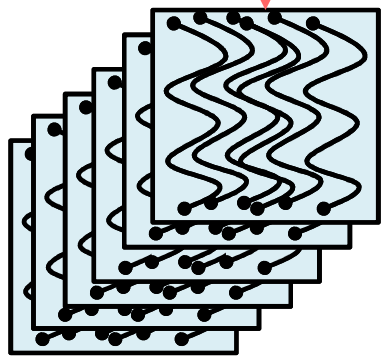
- **Programmer declares (thread) Block:**
  - Block can have 1 to 1024 concurrent threads
  - Block shapes 1D, 2D, or 3D
- **A single kernel program !**
  - All threads execute **the same kernel program**
  - Threads have **thread ID numbers** within Block
  - Thread program uses thread ID to select work and address shared data

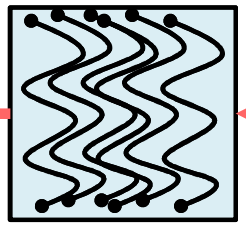


Thread Scheduling/Execution

- **Threads run concurrently**
  - SM assigns/maintains thread ID's
  - SM manages/schedules thread execution
- **Each Thread Blocks is divided in 32-thread Warps**
  - This is an implementation decision, not part of the CUDA programming model.







1 unit = 32 cores
→ parallel execution !

Figure 8. Pascal GP100 SM Unit, from "NVIDIA Tesla P100 White Paper",
https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Thread Scheduling/Execution

- **Warps are scheduling units in SM**
- **예제: an example scenario**
  - 3 blocks are assigned to an SM
  - each block has 256 threads
- **how many warps?**
  - each block has 256 / 32 = 8 warps
  - SM has 3 * 8 = 24 warps
  - At any point in time, **only one** of the 24 warps will be selected for instruction fetch and execution.

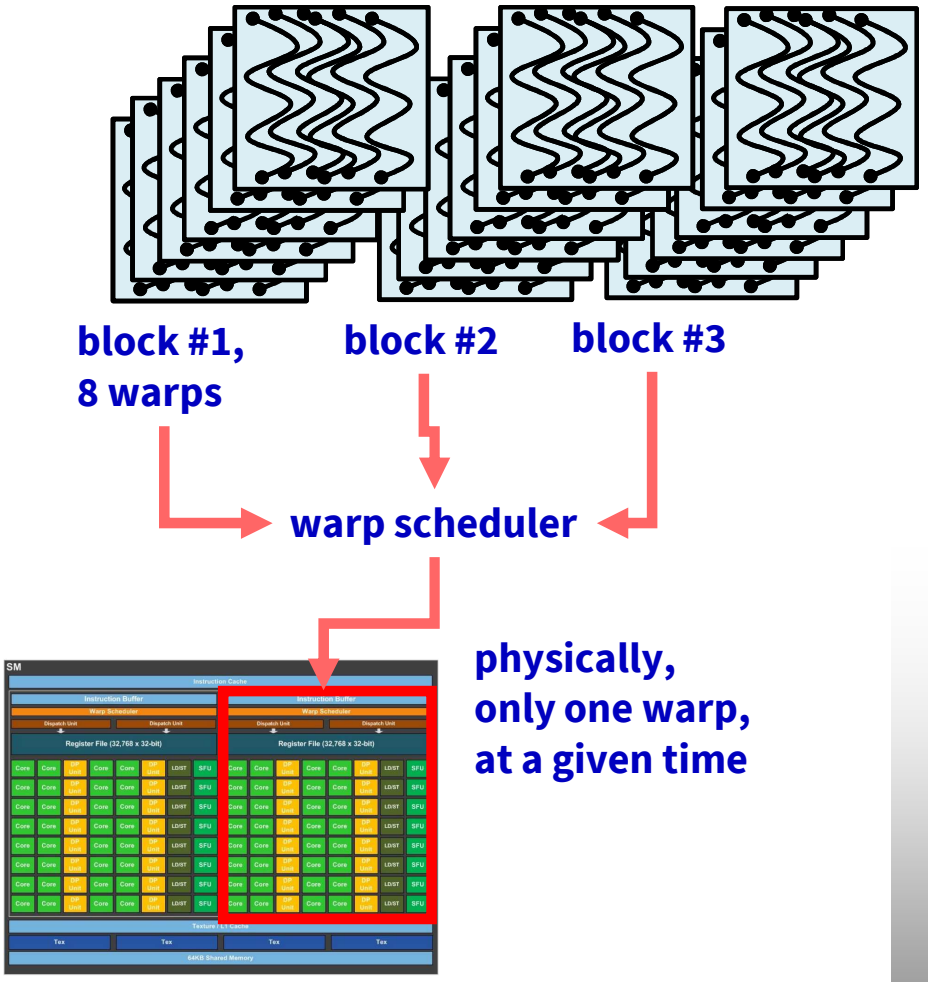
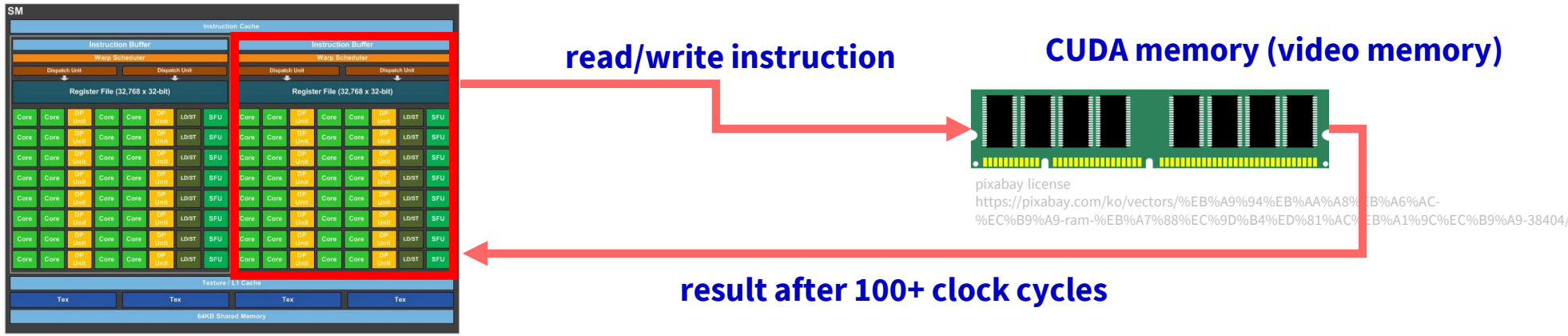


Figure 8. Pascal GP100 SM Unit, from "NVIDIA Tesla P100 White Paper",
https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

SM Warp Scheduling

- **All threads in a Warp execute the same instruction when selected**
  - only **one control logic for an SM**
  - **SPMD model** : single-program, multiple-data
- **memory access → 속도 지연 latency problem → scheduling required !**

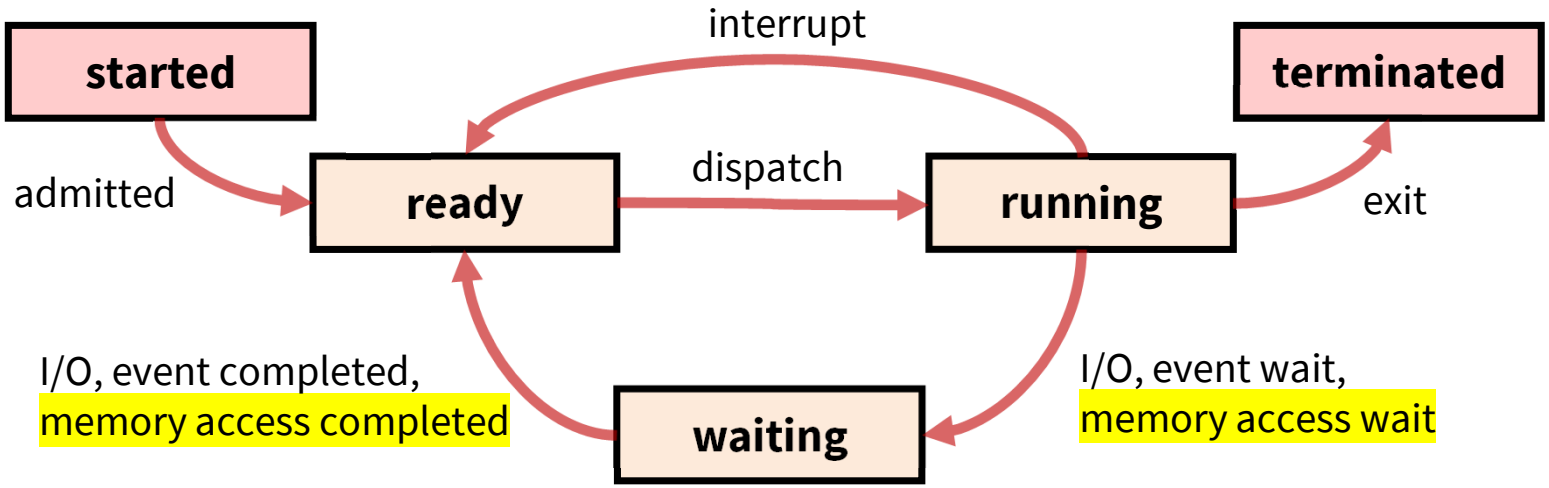


pixabay license
https://pixabay.com/ko/vectors/%EB%A9%94%EB%AA%A8%EB%A6%AC-%EC%B9%A9-ram-%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%EC%B9%A9-38404/

Figure 8. Pascal GP100 SM Unit, from "NVIDIA Tesla P100 White Paper",
https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Time Sharing

- time sharing 시의 process / thread 실행
- 목표: processor 가 쉬지 않도록 한다 → 최고 효율 달성

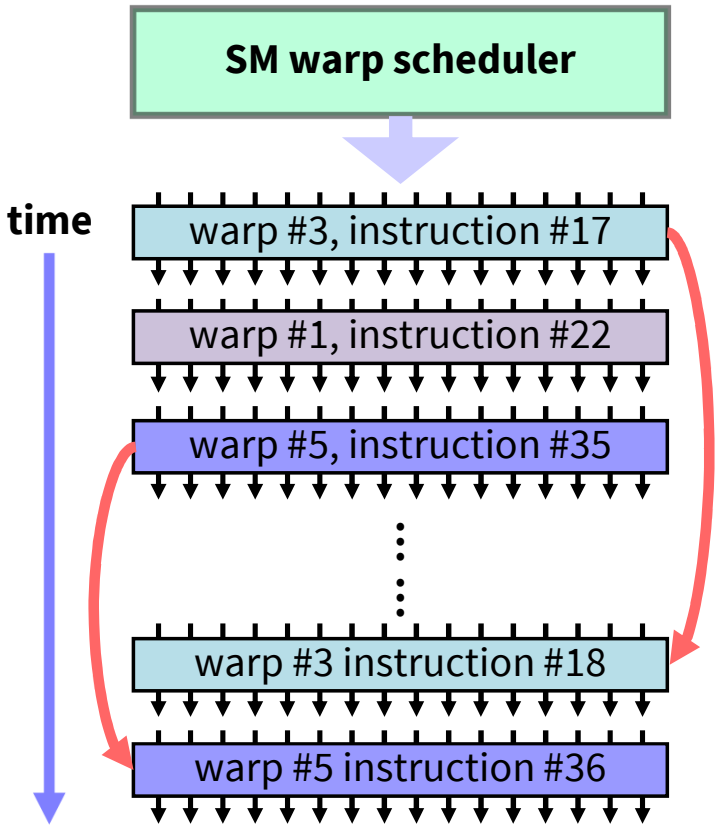


SM Warp Scheduling

- **SM hardware implements zero-overhead warp scheduling**
  - 충분히 많은 register를 확보 : 65,536+ registers for an SM
  - warp 간의 전환 transition 에 (almost) **zero-cost** !
- **score-boarding : All register operands of all instructions in the Instruction Buffer are score-boarded**
  - memory read/write instruction → 해당 warp는 waiting 으로
  - instruction의 operand 가 모두 ready → 해당 warp가 ready
- **warp scheduler**
  - 우선 순위에 따라, ready 상태의 warp 중 1개를 pick up

Warp Scheduling Example

- **an SM = 32 SP's**
- **clock cycles:**
  - 1 clock cycle for register instructions
  - 100 clock cycle for memory access
- **a typical CUDA program**
  - memory access for every 4 instructions
  - 100 / 4 = 25 warps are needed for a good performance



2 Levels of Parallelism

- **A kernel program → a grid**
- **grid → thread blocks → SMs**
  - parallel execution !
- **each SM : warps → score-boarding**
  - parallel execution !
- **warp / block 의 종료 시,**
  - 다음 warp / block 을 pick up

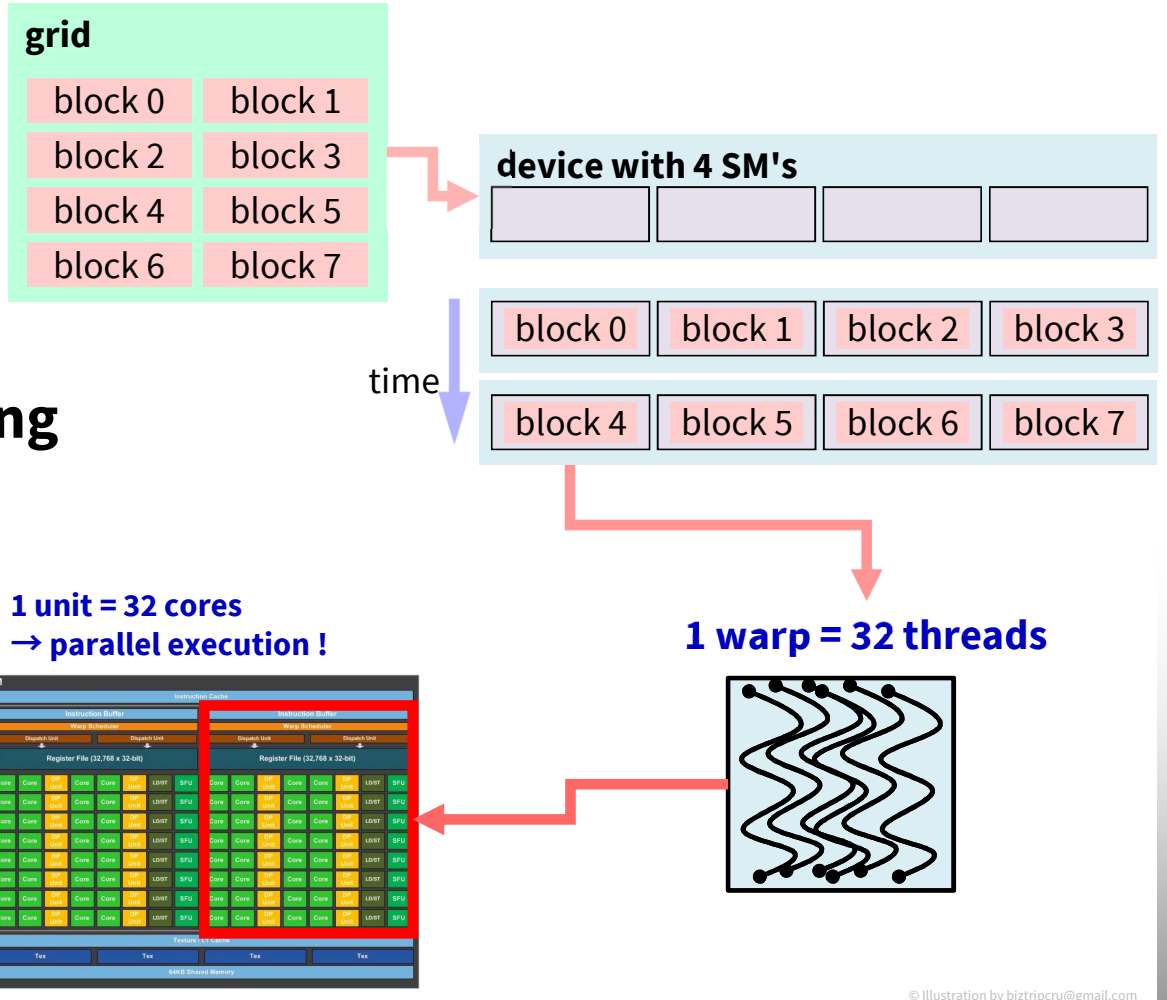


Figure 8. Pascal GP100 SM Unit, from "NVIDIA Tesla P100 White Paper",
https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Resource Restrictions

- **자원 제약에 대한 고려가 필요**
- **예: 특정 SM 에서 다음 제약 가능:**
  - SM 의 thread block 은 32개까지 가능
  - SM 전체의 thread 는 2,048개까지 가능
- **block granularity** 세분성
  - 128 threads/block * 16 blocks
  - 256 threads/block * 8 blocks
  - 512 threads/block * 4 blocks
- **자원 최적화 문제: best performance를 찾아야**

warp id, lane id

- **1 warp = 32 threads**
  - lane = warp 내에서 각 thread의 index (0 ~ 31)
- **warp id : SM 내에서, 특정 warp 의 ID number**
  - SM 단위로 관리 → globally unique 하지 않음 (다른 SM 에 동일 warp id 가능)
  - warp id 를 가져오는 함수는 없음
  - GPU assembly instruction 으로 체크 가능
- **lane id : 1개 warp 내에서, 자신의 lane id**
  - 마찬가지로, lane id를 가져오는 함수는 없음
  - GPU assembly instruction 으로 체크 가능

warp-lane.cu

```
#include "./common.cpp"

__device__ unsigned warp_id(void) {
  // this is not equal to threadIdx.x / 32
  unsigned ret;
  asm volatile ("mov.u32 %0, %warpid;" : "=r"(ret));
  return ret;
}

__device__ unsigned lane_id(void) {
  unsigned ret;
  asm volatile ("mov.u32 %0, %laneid;" : "=r"(ret));
  return ret;
}
```

warp-lane.cu 계속

```
__global__ void kernel_warp_lane( void ) {
   unsigned warpid = warp_id();
   unsigned laneid = lane_id();
   if (warpid == 0) {
      printf("lane=%2u threadIdx.x=%2d threadIdx.y=%2d blockIdx.x=%2d blockIdx.y=%2d\n",
          laneid, threadIdx.x, threadIdx.y, blockIdx.x, blockIdx.y);
   }
}

int main( void ) {
   dim3 dimBlock(16, 16, 1);
   dim3 dimGrid(2, 2, 1);
   kernel_warp_lane <<< dimGrid, dimBlock>>>();
   cudaDeviceSynchronize();
   CUDA_CHECK_ERROR();
   return 0;
}
```

warp-lane.cu 실행 결과

```
linux/cuda-work > ./15a-warp-lane.exe
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 1 threadIdx.x= 1 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 2 threadIdx.x= 2 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 3 threadIdx.x= 3 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 4 threadIdx.x= 4 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 5 threadIdx.x= 5 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 6 threadIdx.x= 6 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 7 threadIdx.x= 7 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 8 threadIdx.x= 8 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 9 threadIdx.x= 9 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=10 threadIdx.x=10 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=11 threadIdx.x=11 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=12 threadIdx.x=12 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=13 threadIdx.x=13 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=14 threadIdx.x=14 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=15 threadIdx.x=15 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=16 threadIdx.x= 0 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=17 threadIdx.x= 1 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=18 threadIdx.x= 2 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=19 threadIdx.x= 3 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=20 threadIdx.x= 4 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=21 threadIdx.x= 5 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=22 threadIdx.x= 6 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=23 threadIdx.x= 7 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=24 threadIdx.x= 8 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=25 threadIdx.x= 9 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=26 threadIdx.x=10 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=27 threadIdx.x=11 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=28 threadIdx.x=12 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=29 threadIdx.x=13 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=30 threadIdx.x=14 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=31 threadIdx.x=15 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 1
lane= 1 threadIdx.x= 1 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 1
lane= 2 threadIdx.x= 2 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 1
lane= 3 threadIdx.x= 3 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 1
```

warp-lane.cu 실행 결과

- **전체 실행 결과**

```
linux/cuda-work > ./15a-warp-lane.exe
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
…
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 1
…
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 0 blockIdx.y= 1
…
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 0 blockIdx.y= 0
…
linux/cuda-work >
```

- **block 실행 순서가 정해져 있지 않음 !**

lane, warp id 부여 규칙?

- lane id 부여 규칙은?
- warp id 부여 규칙은?
  - **system dependent**
  - 알고 있을 필요 없음

- CUDA 처리는 **thread block 단위**
  - **block 실행 순서도 정해지지 않음**
  - 그 이하는 system 에서 처리

```
linux/cuda-work > ./15a-warp-lane.exe
lane= 0 threadIdx.x= 0 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 1 threadIdx.x= 1 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 2 threadIdx.x= 2 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 3 threadIdx.x= 3 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 4 threadIdx.x= 4 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 5 threadIdx.x= 5 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 6 threadIdx.x= 6 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 7 threadIdx.x= 7 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 8 threadIdx.x= 8 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane= 9 threadIdx.x= 9 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=10 threadIdx.x=10 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=11 threadIdx.x=11 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=12 threadIdx.x=12 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=13 threadIdx.x=13 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=14 threadIdx.x=14 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=15 threadIdx.x=15 threadIdx.y= 0 blockIdx.x= 1 blockIdx.y= 0
lane=16 threadIdx.x= 0 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=17 threadIdx.x= 1 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=18 threadIdx.x= 2 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=19 threadIdx.x= 3 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=20 threadIdx.x= 4 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=21 threadIdx.x= 5 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=22 threadIdx.x= 6 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=23 threadIdx.x= 7 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=24 threadIdx.x= 8 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=25 threadIdx.x= 9 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=26 threadIdx.x=10 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=27 threadIdx.x=11 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=28 threadIdx.x=12 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=29 threadIdx.x=13 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=30 threadIdx.x=14 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
lane=31 threadIdx.x=15 threadIdx.y= 1 blockIdx.x= 1 blockIdx.y= 0
```

내용 contents

- **CUDA hardware**
- **Transparent Scalability**
  - thread block scheduling
- **Thread and Warp**
- **CUDA thread scheduling**
  - warp scheduling
- **Overall execution model**

- **실습: warp-lane.cu – warp ID, lane ID check**

Thread and GPU

쓰레드와 GPU

폰트 끝단 일치 → 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
본고딕 Noto Sans KR

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
Source Sans Pro

Mathematical Notations $O(n \log n)$
Source Serif Pro