TA Session 02 – CUDA Review & Profiler

Jinpyo Kim
Dept. of Computer Science and Engineering, College of Engineering
Seoul National University
jinpyo@aces.snu.ac.kr

CUDA Programming Review

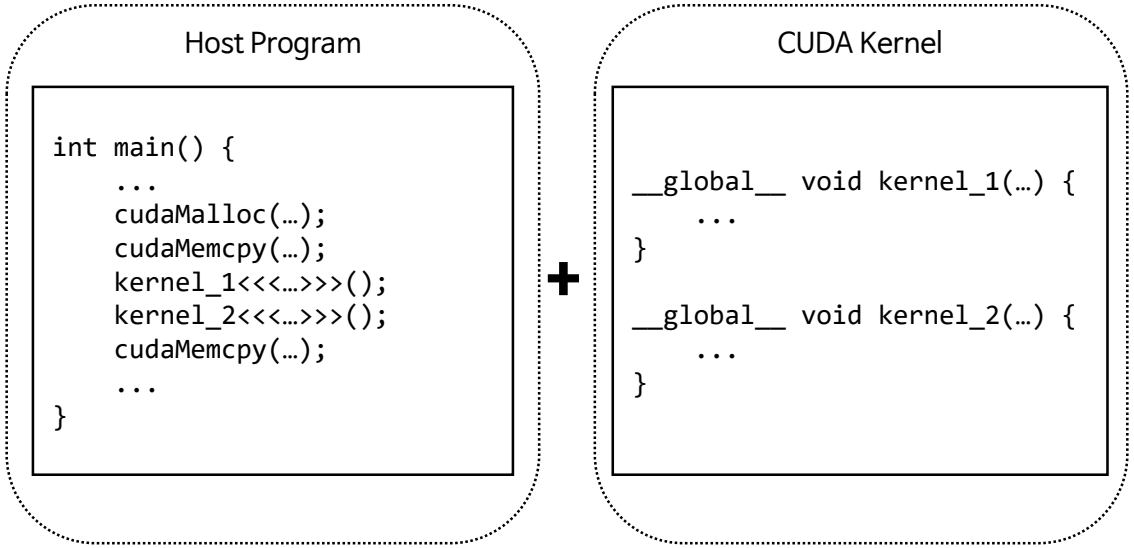
GPU 아키텍처

- GPU는 여러 Streaming Multiprocessors (SM)를 통해 연산을 수행
  - 예) V100의 경우 80개의 SM을 보유
- 각 SM 내에는 다양한 연산 유닛, 캐시, 워프 스케줄러 등이 있음

이미지 출처: NVIDIA V100 whitepaper

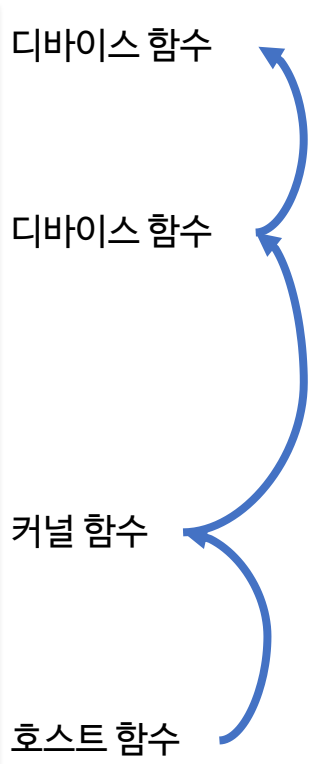
CUDA 프로그램

- 호스트 프로그램 + CUDA 커널
  - 호스트 프로그램(host program)은 호스트 프로세서(CPU)에서 실행
  - CUDA 커널은 디바이스(GPU)에서 실행

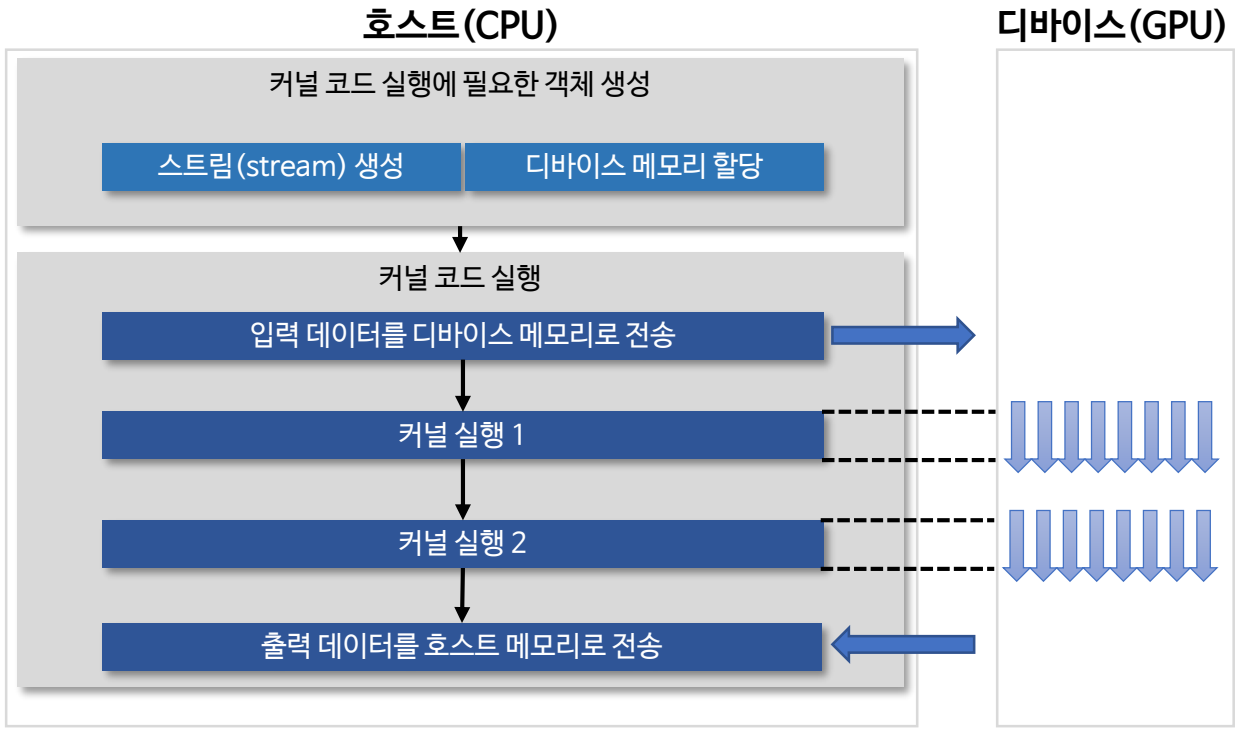


CUDA 프로그램 코드 예시

```
__device__ float foo(float A, float B) {
  ...
}

__device__ int bar(...) {
  int result;
  ... foo(A, B); ...
  return result;
}

__global__ void your_kernel(...) {
  ... bar(...) ...
}

// __host__ 생략 가능
int main() {
  ...your_kernel<<<...>>>(...) ...
}
```

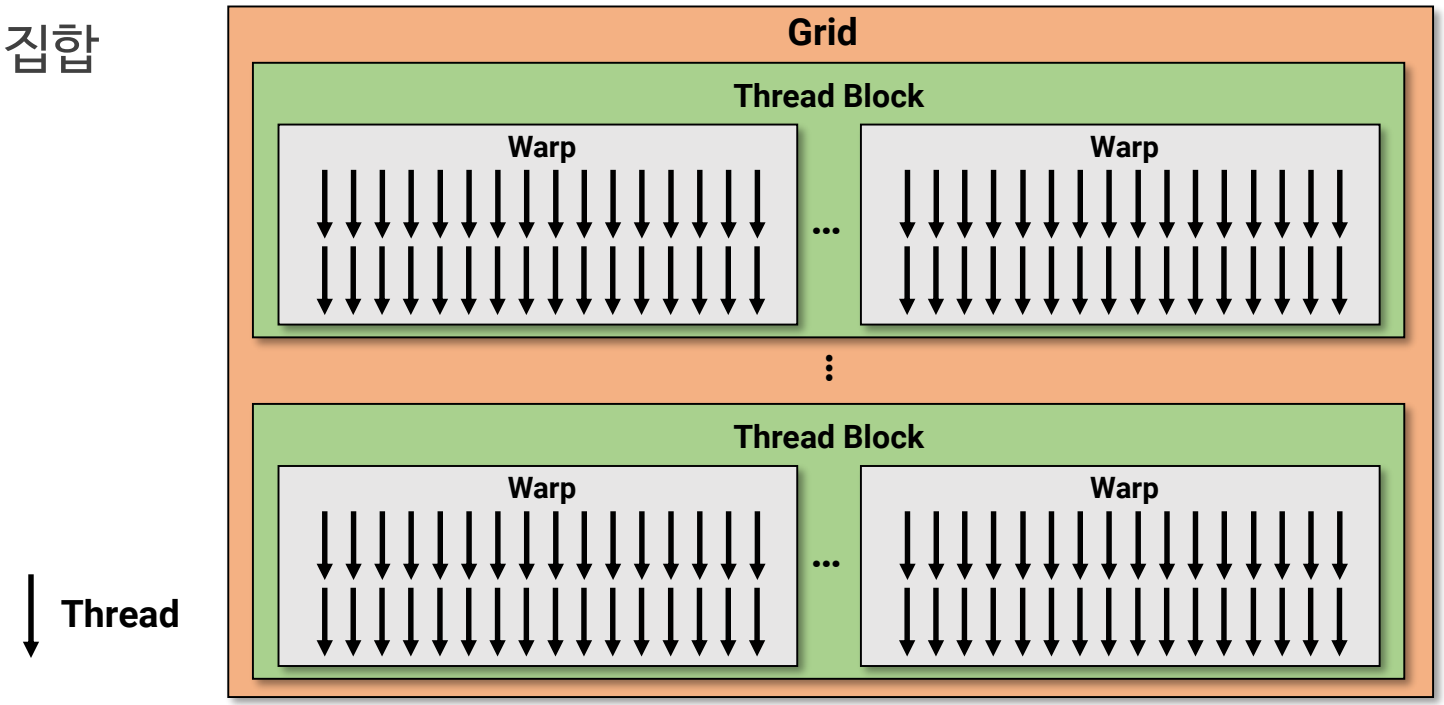


일반적인 CUDA 프로그램 실행 과정



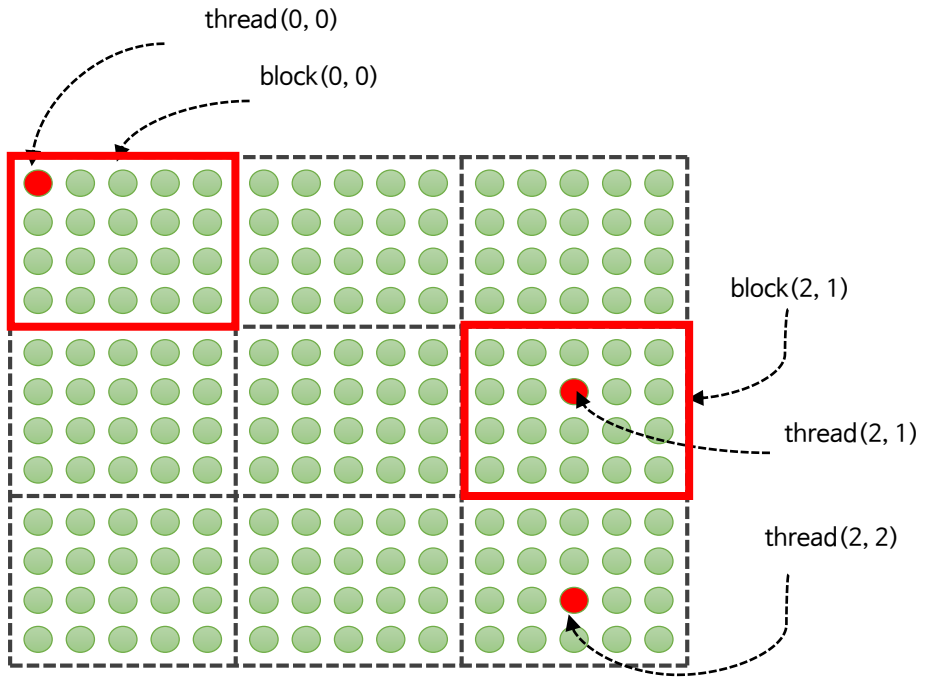
스레드 계층 구조

- 스레드(Thread): 연산을 수행하는 최소 단위 (CPU 스레드를 생각)
- 워프(Warp): 하드웨어 상에서 연산을 스케쥴링하는 최소 단위
  - 1워프=32스레드, 워프 내의 모든 스레드는 같은 인스트럭션 실행
- 스레드 블록(Thread Block): 여러 스레드의 묶음
- 그리드(Grid): 여러 스레드 블록의 집합

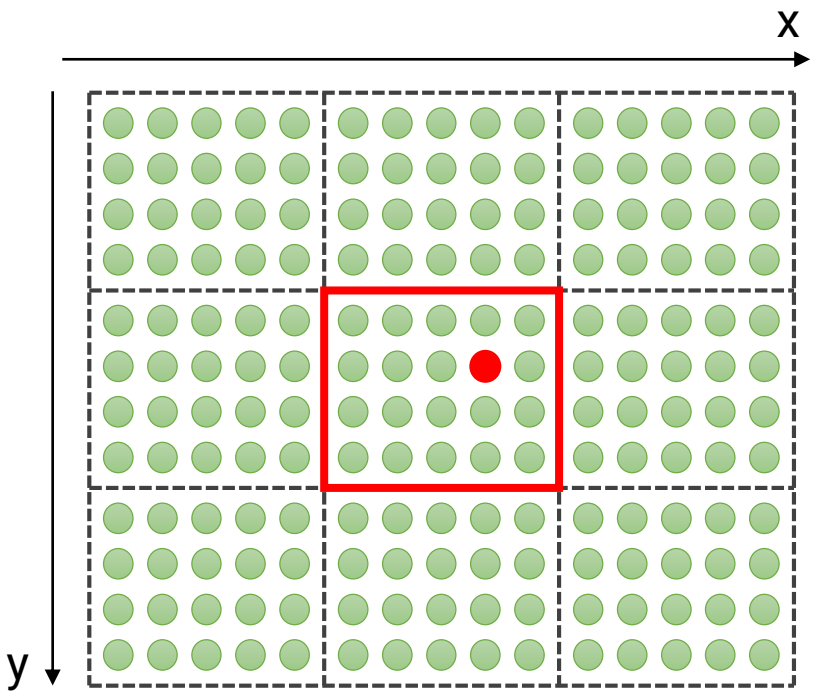


CUDA 커널 실행

- 인덱스 공간: 3차원의 좌표계를 설정해 모든 스레드에 고유한 ID를 부여
- 스레드 계층 구조를 고려해 3차원 인덱스 공간을 설정
  - `dim3 gridDim, blockDim`



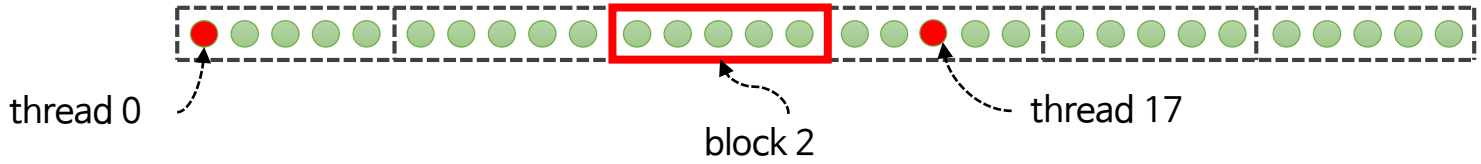
스레드 좌표 공간 구성 예시 1



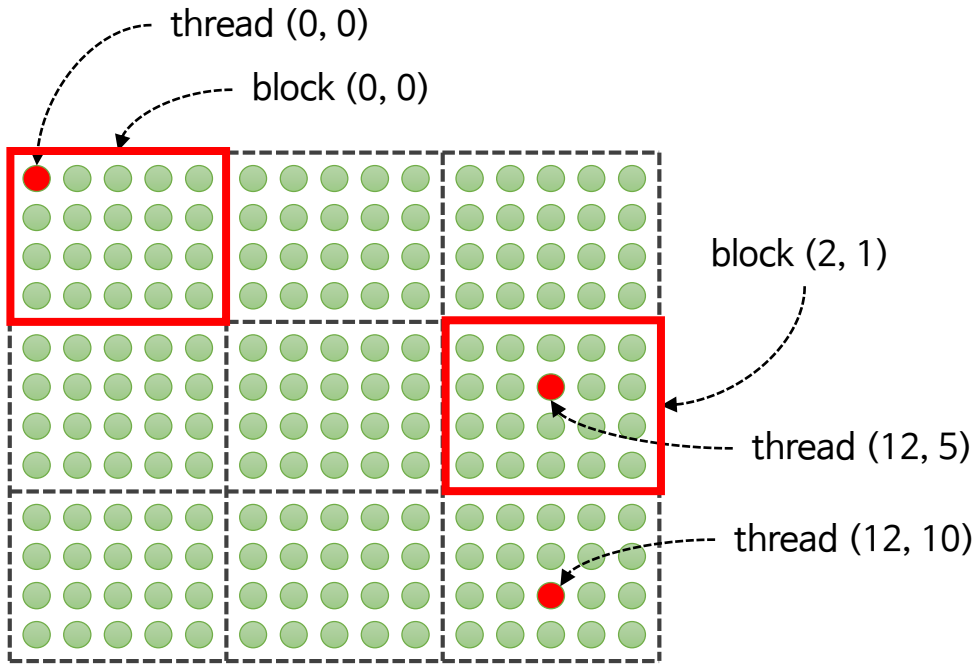
- 전체 크기: (15, 12)
- 블록 크기: (5, 4), `blockDim={5, 4};`
- 블록 개수: (3, 3), `gridDim={3, 3};`
- 빨간색 스레드
  - 스레드 ID (`threadIdx`): (3, 1)
    - 해당 블록 안에서의 위치
  - 블록 ID (`blockIdx`): (1, 1)
    - 스레드가 속한 블록의 위치
  - **글로벌 ID: (8, 5)**
    - 전체 좌표 공간 안에서의 위치
    - blockDim.x * blockIdx.x + threadIdx.x; 와 같이 계산

스레드 좌표 공간 구성 예시 2

- 1차원 - `blockDim={5}, gridDim={6}`



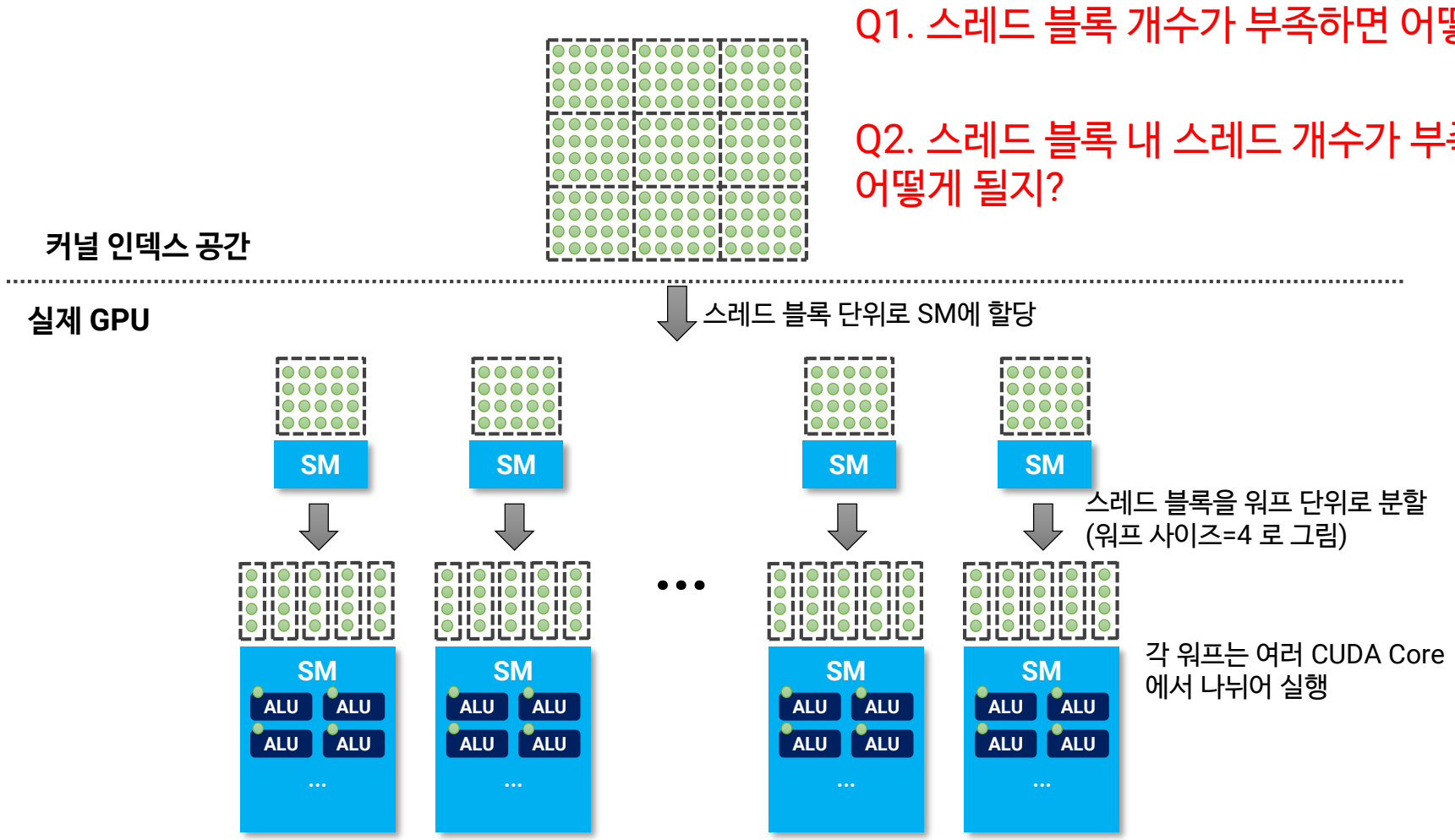
- 2차원 - `blockDim={5, 4}, gridDim={3, 3}`



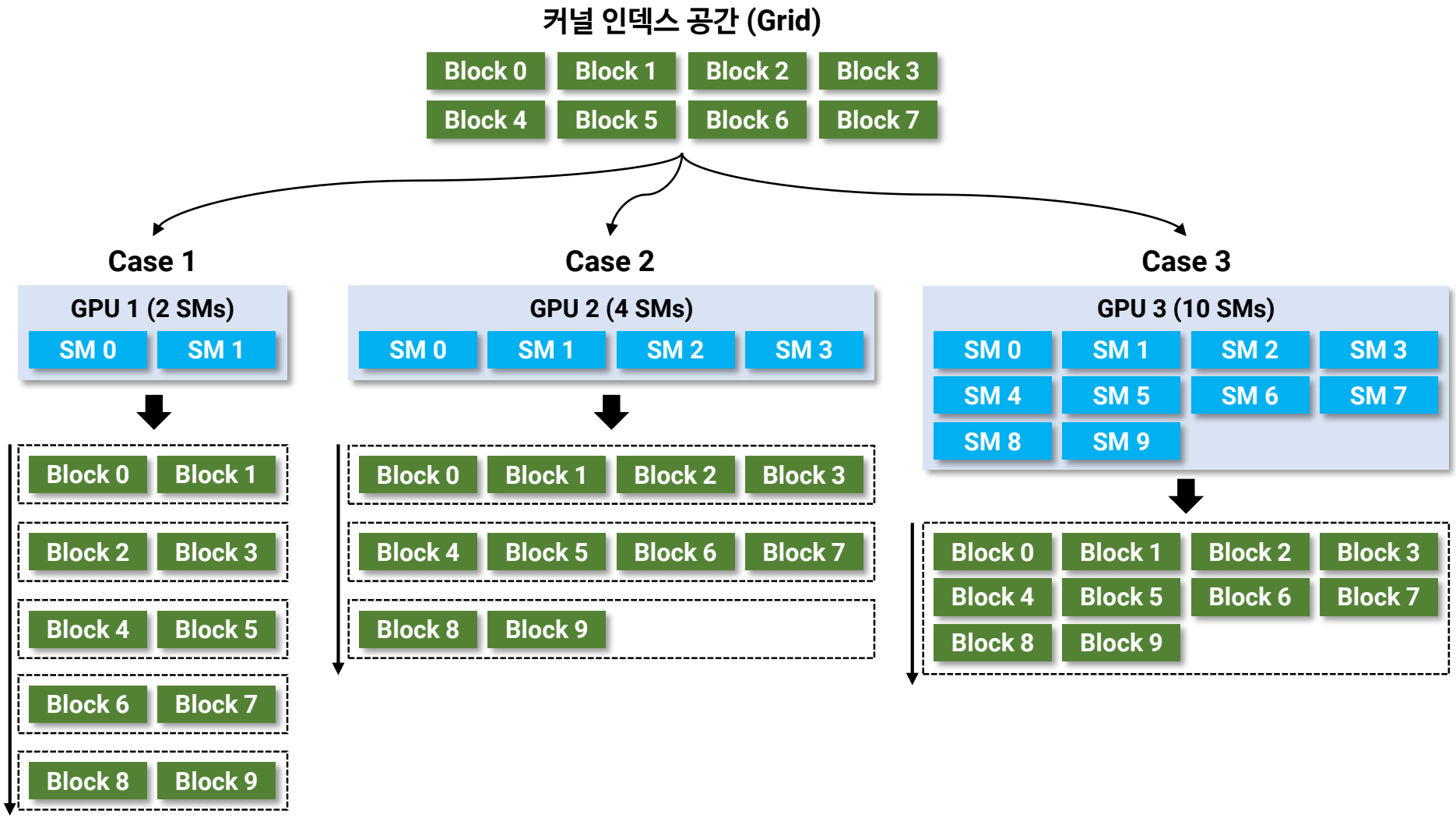
스레드 공간 구성 방법

1. 총 스레드 개수를 결정
   - 하나의 스레드가 하나의 출력 원소를 담당하도록 하는 것이 일반적
     - 길이가 N 인 벡터 덧셈 ⇒ N 개의 스레드 생성
     - 크기가 H x W 인 이미지에 필터 적용 ⇒ H x W 개의 스레드 생성
   - 하나의 스레드가 여러 출력 원소를 담당하는 것도 가능
   - 여러 스레드가 하나의 출력 원소를 담당하는 것은 구현이 어려움
2. 블록 크기를 결정
   - `blockDim`, 블록당 스레드 개수
   - 워프 크기(32)의 배수가 되도록: 128, 256, ..., 1024 (최대)
3. 그리드 크기를 결정
   - `gridDim`, 전체 블록 개수
   - 총 스레드 개수를 블록 크기로 나누어 결정

CUDA 커널 실행 과정


Q1. 스레드 블록 개수가 부족하면 어떻게 될지?
Q2. 스레드 블록 내 스레드 개수가 부족하면 어떻게 될지?
커널 인덱스 공간
실제 GPU
스레드 블록 단위로 SM에 할당
SM
스레드 블록을 워프 단위로 분할
(워프 사이즈=4 로 그림)
ALU
각 워프는 여러 CUDA Core
에서 나뉘어 실행

스레드 블록 스케줄링


커널 인덱스 공간 (Grid)
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Case 1
GPU 1 (2 SMs)
SM 0
SM 1
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 9
Case 2
GPU 2 (4 SMs)
SM 0
SM 1
SM 2
SM 3
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 9
Case 3
GPU 3 (10 SMs)
SM 0
SM 1
SM 2
SM 3
SM 4
SM 5
SM 6
SM 7
SM 8
SM 9
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 9

스레드 공간 구성 Rule of Thumb

- 충분한 수의 스레드 블록 및 스레드를 확보할 것
  - GPU의 HW 자원을 모두 활용하기 위해서는 많은 수의 스레드가 필요
  - 스레드 블록 개수는 적어도 SM 개수 이상이 되도록
  - 하나의 SM에서 여러 개의 스레드 블록이 동시에 실행될 수도 있음

- 스레드 블록의 크기는 warp의 크기(=32) 배수로 할 것
  - GPU상에서 모든 연산은 warp 단위로 실행됨
  - 스레드 블록 크기가 32의 배수가 아닐 경우 idle 스레드가 항상 존재하게 됨
  - 스레드 블록당 128-256 개 스레드가 좋은 출발점

스레드 공간 구성 Rule of Thumb (cont.)

- GPU 정보를 확인하고 스레드 공간을 구성할 것
  - SM 개수
  - 스레드 블록당 최대 스레드 개수
  - SM 당 연산 유닛 개수
  - ...

- 다양한 구성을 실험해보고 가장 좋은 성능을 보이는 것을 택할 것
  - 디바이스가 바뀌면 튜닝을 다시 해야 할 가능성이 높음
  - 같은 연산일지라도 입력 크기에 따라 최적의 스레드 공간 구성이 다를 수 있음

일반적인 프로그램 최적화 과정

- 프로그램 최적화는 다음 두 과정의 반복
  1. 병목 지점 (가장 오래 걸리는 부분) 파악
  2. 해당 부분을 병렬화/최적화

- 두 과정 모두 중요함
  - 병목 지점을 어떻게 파악할 것인지
  - 병목 지점이 병렬화/최적화 가능한 부분인지
  - 어느 수준까지 병렬화/최적화를 수행할 것인지
  - 어떻게 병렬화/최적화를 수행할 것인지

CUDA Profiler

Nsight Product Family

- **Nsight Systems**
  - CUDA **프로그램** (System-wide) 프로파일러
  - 프로그램의 전반적인 알고리즘을 최적화할 때 사용
- **Nsight Compute**
  - CUDA **커널** 프로파일러
  - 특정 CUDA 커널을 최적화할 때 사용
- Nsight Graphics
  - CUDA 그래픽 작업 프로파일러

Nsight Systems

- CUDA 프로그램 (System-wide) 프로파일러

- 최적화의 가장 첫 단계로 사용
  - Nsight Systems 로 프로그램 전반에 걸친 최적화를 진행하고 병목 지점을 파악한 이후
  - Nsight Compute 로 개별 CUDA 커널 최적화를 진행

- NVIDIA 공식 웹사이트에서 다운로드 & 설치
  - https://developer.nvidia.com/nsight-systems

Nsight Systems 사용법

- GPU 서버에서 프로파일링 실행 후 로컬에서 확인
  1. nsys CLI 를 이용해 GPU 서버에서 프로파일 데이터 수집
     - 프로파일 결과 데이터가 .nsys-rep 파일로 저장됨

```
$ nsys profile --cudabacktrace=all ./main
```

  2. 프로파일 데이터를 로컬로 다운받아 확인

Nsight Systems CLI - 사용법

```
$ nsys [command_switch] [optional command_switch_options] \
       [application] [optional application_options]
```

- `command_switch`: 프로파일러 기능 지정
  - `profile, launch, start, stop, ...`
  - 대부분 `profile` 커맨드로 충분

- `application`: 실행할 프로그램

- **Documentation**
  https://docs.nvidia.com/nsight-systems/UserGuide/index.html

Nsight Systems CLI - 주요 옵션

- `-e, --env-var`
  - 프로파일 대상 프로그램 실행 시 환경변수 설정
- `-y, --delay`
  - 프로그램 실행 이후 일정 시간 이후부터 프로파일링 시작
- `-t, --trace`
  - 프로파일링 대상 설정
  - cuda, cublas, cudnn, nvtx, mpi 등
- `-o, --output`
  - 프로파일링 결과 파일 이름
- `-f, --force-overwrite`
  - 프로파일링 결과 파일이 이미 있으면 덮어쓰기

Nsight Systems CLI - 사용 예시

기본 옵션으로 프로파일링

```
nsys profile ./main ...
```

환경 변수 설정 후 프로파일링 (-e 옵션)

```
nsys profile -e MYVAR=myvar1 ./main ...
```

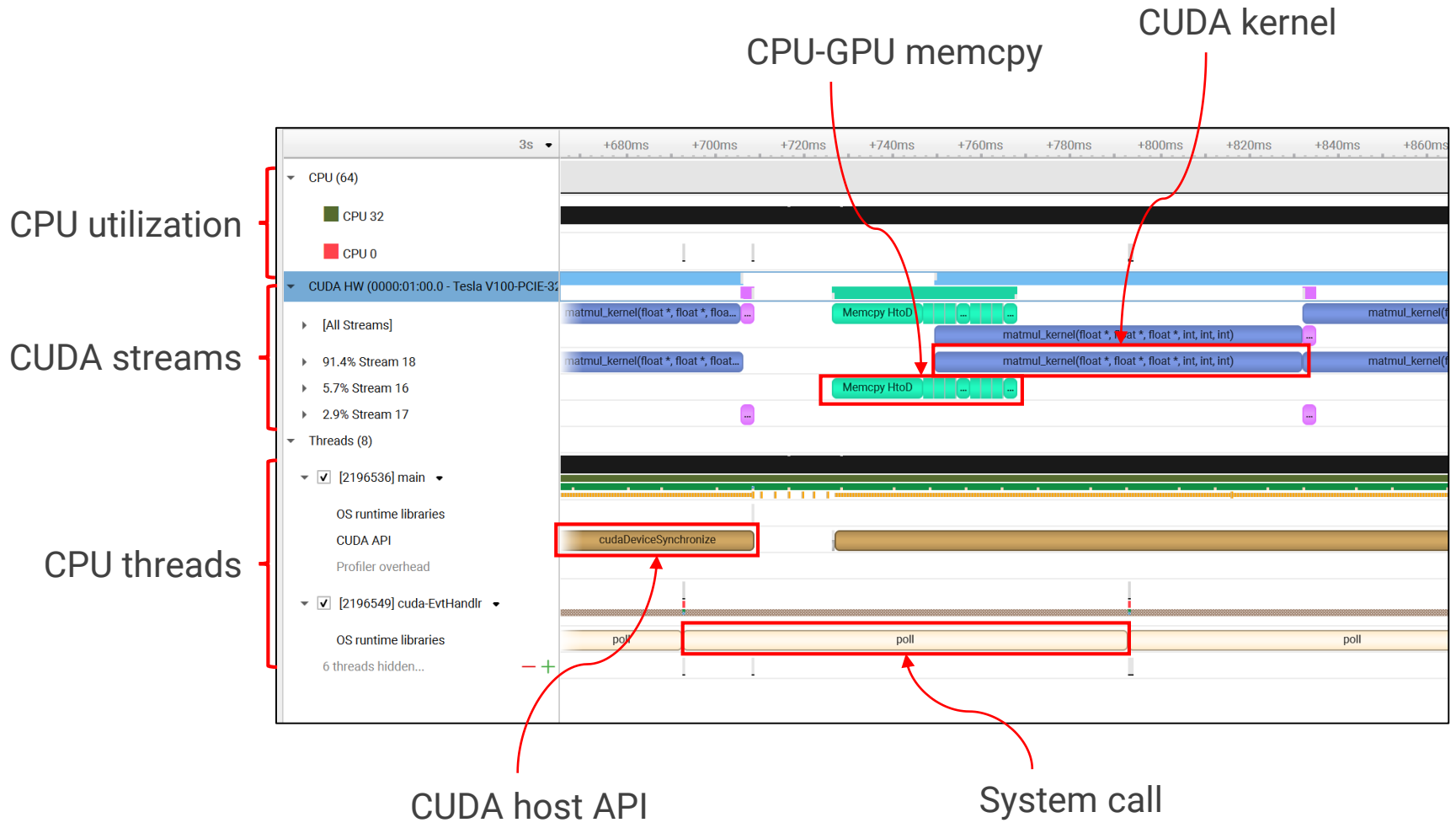
지연된 프로파일링 (--delay 옵션, 프로그램 실행 10초 후 프로파일 시작)

```
nsys profile --delay 10 ./main ...
```

일부 이벤트만 프로파일링 (--trace 옵션)

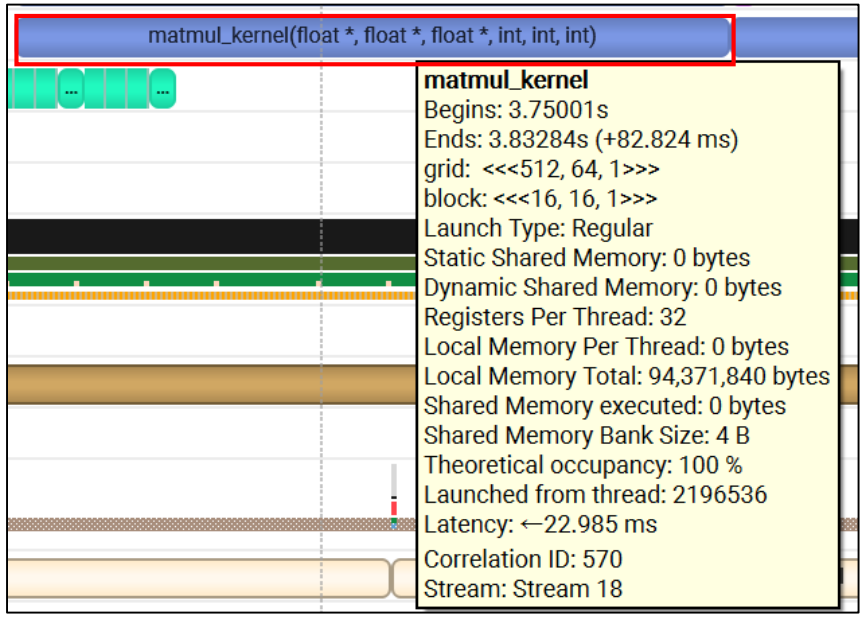
```
nsys profile --trace=cuda,cudnn,cublas ./main ...
```

Nsight Systems - 프로파일 결과 (cont'd)



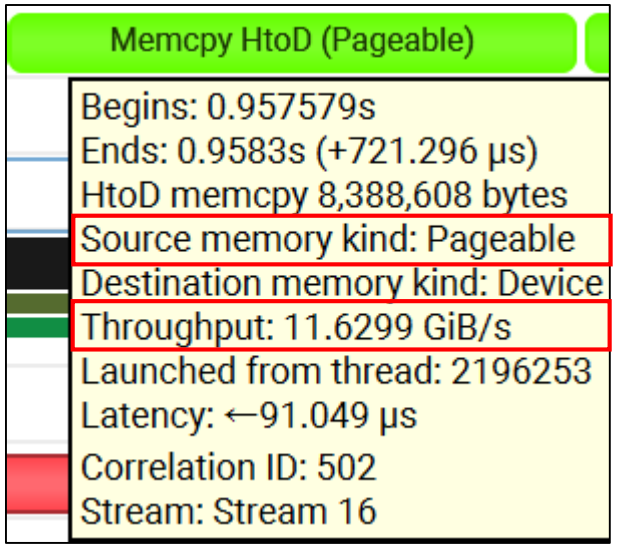
Nsight Systems - 프로파일 결과 (cont'd)

- 타임라인의 커널 부분에 마우스를 올려 상세 정보 확인 가능
- 의도한 대로 커널이 실행되었는 지 확인
  - 그리드 및 스레드 블록 크기
  - 공유 메모리 사용량
  - …



Nsight Systems - 프로파일 결과 (cont'd)

- 타임라인의 커널 부분에 마우스를 올려 상세 정보 확인 가능
- 호스트-디바이스 데이터 전송 대역폭이 정상인지 확인



Nsight Compute

- CUDA 커널 프로파일러
  - Nsight Systems 프로파일링 결과로 파악한 커널을 더욱 깊게 최적화할 때 사용

- 커널을 반복 실행해 다양한 HW performance counter 수집
  - 커널 코드에 자동으로 프로파일링 코드 삽입 (injection)
  - 커널 실행 직전의 메모리 상태를 저장해두고 반복 실행 시 복원
  - 프로파일링 시간이 길어질 수 있음, 원하는 부분만 프로파일링 하는 것을 추천

Nsight Compute 사용법

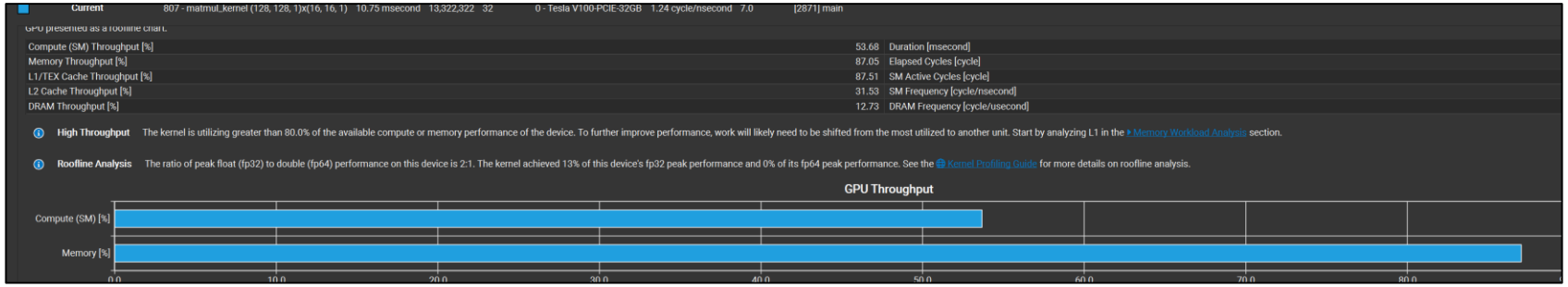
1. GPU 서버에서 프로파일링 실행 후 텍스트로 결과 확인

```
$ ncu --set full ./main
```

2. GPU 서버에서 프로파일링 실행 후 로컬에서 GUI로 확인
   1. ncu CLI 로 GPU 서버에서 프로파일 데이터 수집
      - 프로파일 결과 데이터가 .ncu-rep 파일로 저장됨

```
$ ncu -o ncu_report --set full ./main
```

   2. 프로파일 데이터를 로컬로 다운받아 확인



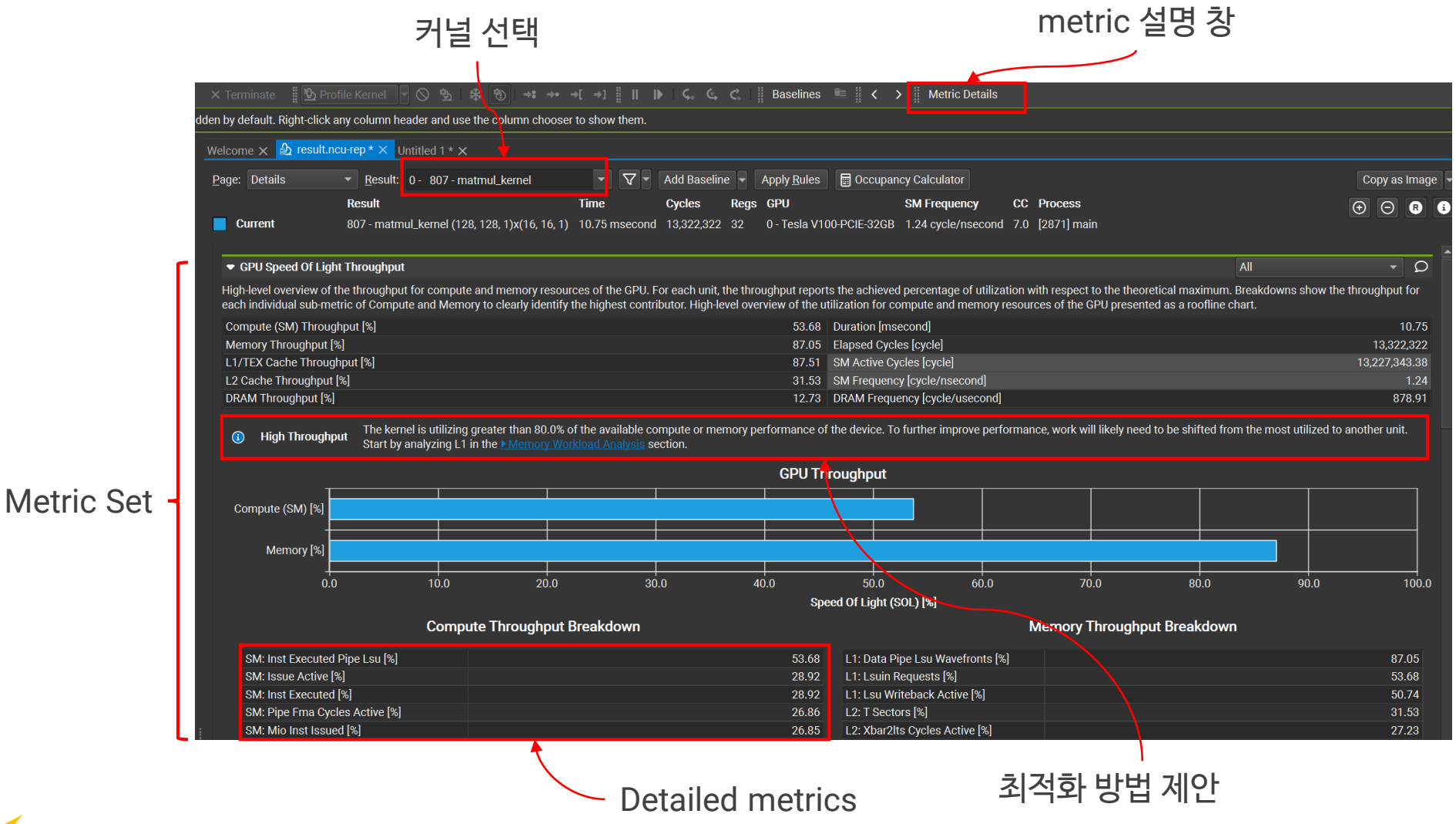
Nsight Compute - CLI 실행 결과 예시

```
kjp4155@login:~/matmul$ srun ncu --set full ./main -n 1 2048 2048 2048
Options:
  Problem size: M = 2048, N = 2048, K = 2048
  Number of iterations: 1
  Print matrix: off
  Validation: off

Initializing... ==PROF== Connected to process 2809 (/home/n0/kjp4155/matmul/main)
done!
Calculating...(iter=0) ==PROF== Profiling "matmul_kernel" - 0: 0%....50%....100% - 74 passes
4.580928 sec
Avg. time: 4.580928 sec
Avg. throughput: 3.750303 GFLOPS
==PROF== Disconnected from process 2809
[2809] main@127.0.0.1
  matmul_kernel(float *, float *, float *, int, int, int), 2023-Aug-13 19:04:59, Context 1, Stream 7
    Section: GPU Speed Of Light Throughput
    ----------------------- ----------------------------------------------------- -------------- ------------------------------
    DRAM Frequency                                                      cycle/usecond                         878.02
    SM Frequency                                                        cycle/nsecond                           1.24
    Elapsed Cycles                                                              cycle                     13,337,931
    Memory [%]                                                                      %                          86.93
    DRAM Throughput                                                                 %                          12.93
    Duration                                                                   msecond                          10.78
    L1/TEX Cache Throughput                                                         %                          87.65
    L2 Cache Throughput                                                             %                          31.23
    SM Active Cycles                                                            cycle                  13,205,601.81
    Compute (SM) [%]                                                                %                          53.60
    ----------------------- ----------------------------------------------------- -------------- ------------------------------
    INF   The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To
          further improve performance, work will likely need to be shifted from the most utilized to another unit.
          Start by analyzing L1 in the Memory Workload Analysis section.

    INF   The ratio of peak float (fp32) to double (fp64) performance on this device is 2:1. The kernel achieved 13% of
          this device's fp32 peak performance and 0% of its fp64 peak performance. See the Kernel Profiling Guide
          (https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#roofline) for more details on roofline
          analysis.
```

GPU performance metrics

Optimization suggestions

Nsight Compute - GUI 예시



Nsight Compute - Metric Sets

- GPU Speed of Light Throughput
  - 전반적인 GPU 자원 사용 효율 관련 metric 모음
- Compute Workload Anaysis
  - SM 들의 각 연산 유닛 사용량 관련 metric 모음
- Memory Workload Analysis
  - GPU 메모리 계층 구조의 각 부분 사용량 관련 metric 모음
- Scheduler Statistics
- Warp State Statistics
- Instruction Statistics
- ...