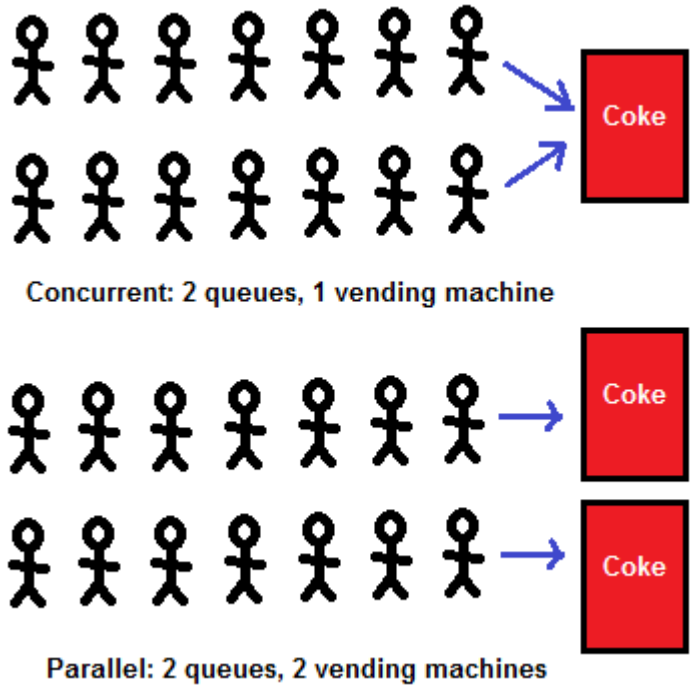
Multiple GPUs with CUDA(1)

https://youtu.be/mn_yGzH5yZA

Implementation Strategies

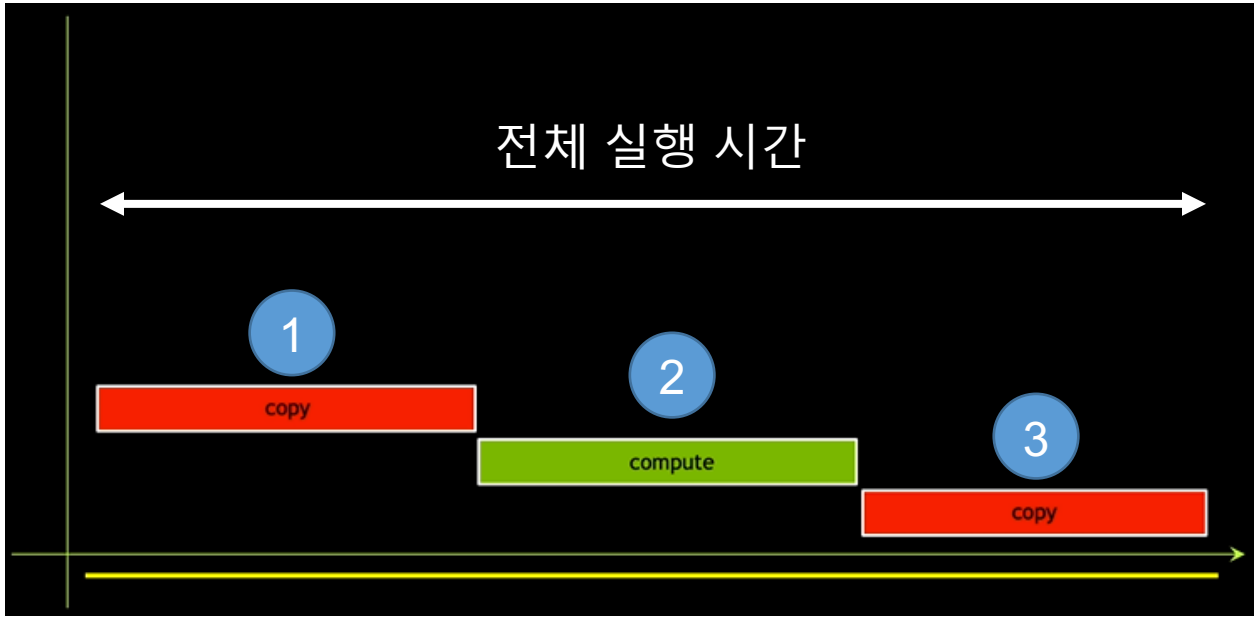
- Concurrency와 Parallelism의 차이
  - Concurrency – 동시성
    - 동시에 실행되는 것처럼 보이는 것
    - 논리적인 개념
    - 싱글코어, 멀티코어에서 가능
  - Parallelism – 병렬성
    - 실제로 동시에 실행되는 것
    - 물리적인 개념
    - 멀티코에어서만 가능



Implementation Strategies

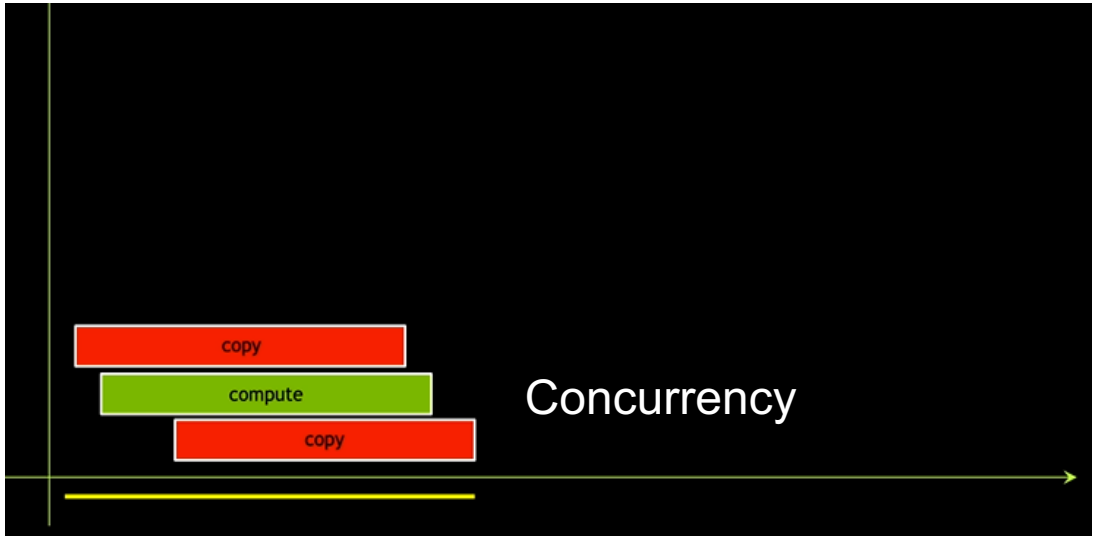
- 일반적으로 GPU Programming은 3가지 단계로 동작
  1. Transfer data to GPU device
  2. Perform computation on GPU device
  3. Transfer data back to the host

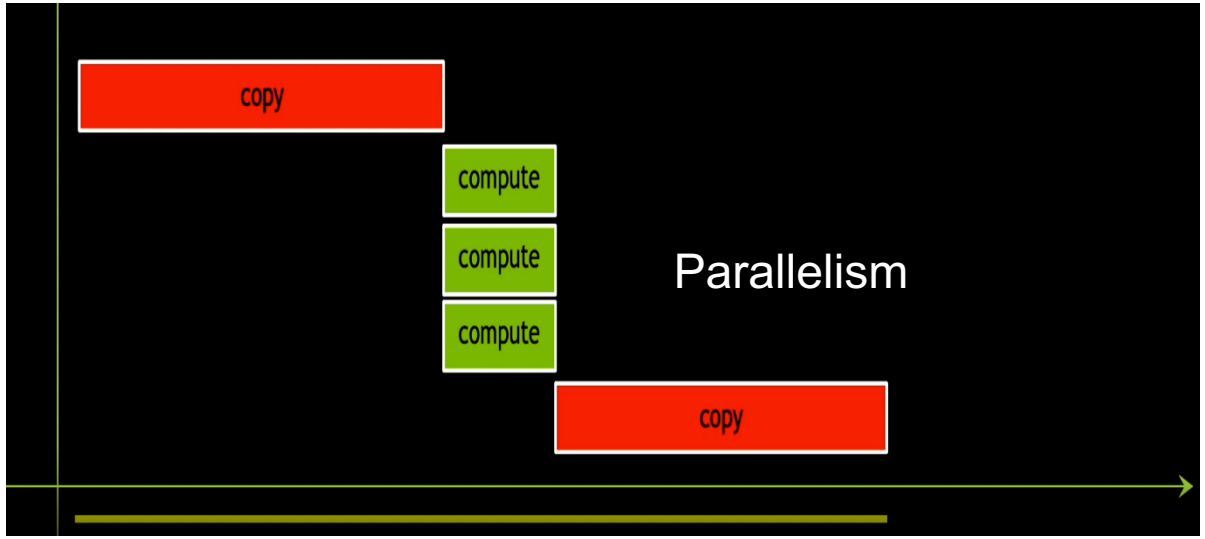
- 실행 시간은 보통 데이터가 복사되는 시점부터 다시 복사하는 과정 모두를 포함



Implementation Strategies

- 실행 시간을 단축 시킬 수 있는 방법
  1. 메모리 전송과 연산을 오버랩 하는 방법
     - Concureency 동시성을 활용
  2. 연산만을 오버랩 하는 방법
     - Parallelism 병렬성을 활용



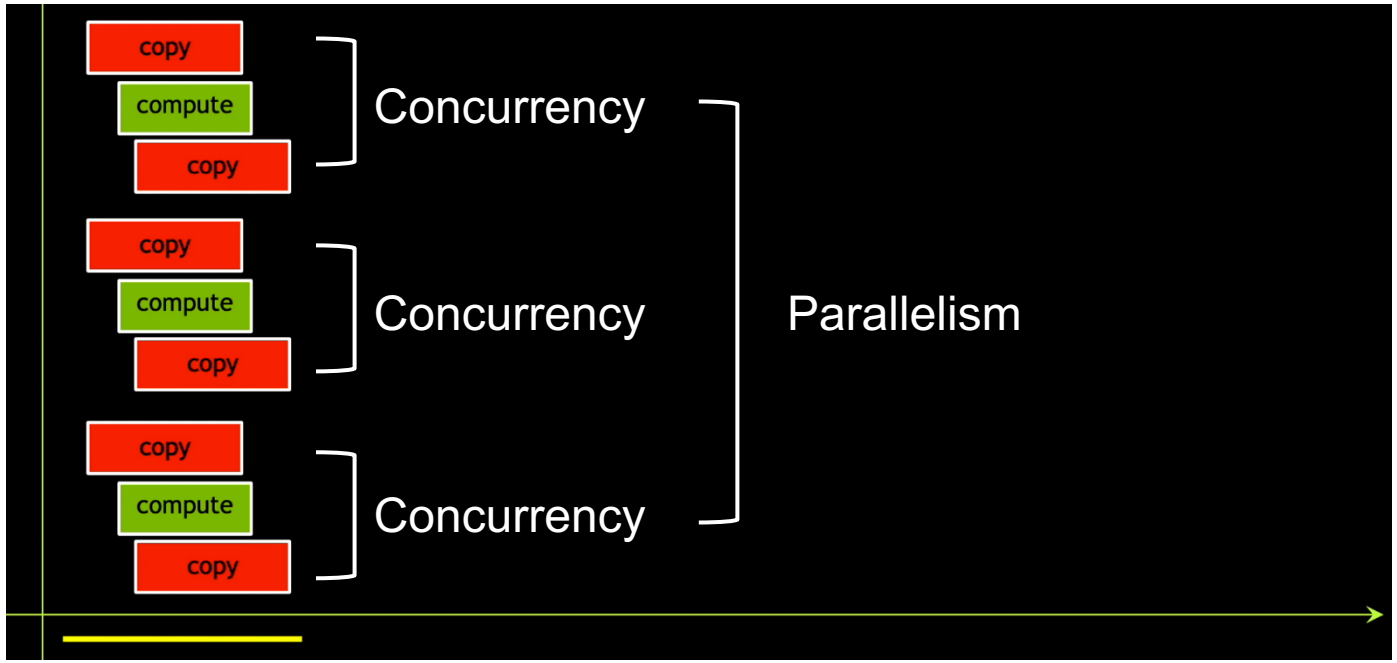


Implementation Strategies

- 두 방법을 조합하게 되면 실행 시간을 훨씬 더 단축시킬 수 있음

- 이때 데이터를 적절하게 분배되어야 함.

- 암호 구현 관점에서 보았을 때, 연산되는 값이 서로에게 영향을 주지 않는다면 적용 가능

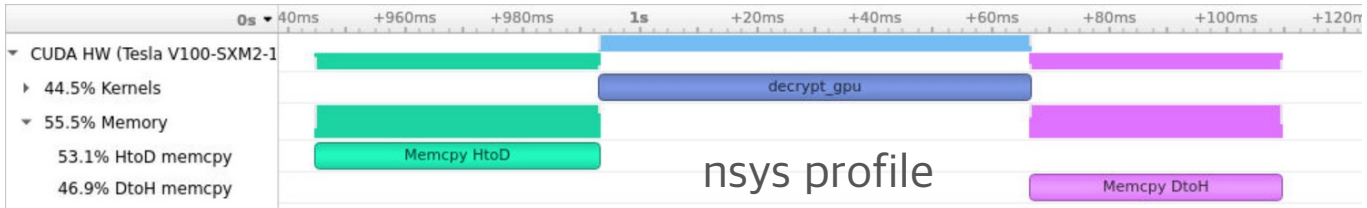


Test Code 리뷰

```
Timer timer, overall;

uint64_t *data_cpu, *data_gpu;

timer.start();
cudaMallocHost(&data_cpu, sizeof(uint64_t) * num_entries);
cudaMalloc(&data_gpu, sizeof(uint64_t) * num_entries);
timer.stop("allcate memory");

timer.start();
// encrypt data
encrypt_cpu(data_cpu, num_entries, num_iters, openmp);
timer.stop("encrypt data on CPU");

overall.start();
timer.start();
// Data copy from CPU to GPU
cudaMemcpy(data_gpu, data_cpu, sizeof(uint64_t) * num_entries, cudaMemcpyHostToDevice);
timer.stop("copy data from CPU to GPU");

timer.start();
// Decrypt data on GPU(s).
decrypt_gpu<<<80 * 32, 64>>>(data_gpu, num_entries, num_iters);
timer.stop("decrypt data on GPU");

timer.start();
// Copy data from GPU to CPU
cudaMemcpy(data_cpu, data_gpu, sizeof(uint64_t) * num_entries, cudaMemcpyDeviceToHost);
timer.stop("copy data from GPU to CPU");

// Stop timer for total time on GPU(s).
overall.stop("total time on GPU");
```

```
TIMING: 269.702 ms (allcate memory)
TIMING: 14200.5 ms (encrypt data on CPU)
TIMING: 48.3961 ms (copy data from CPU to GPU)
TIMING: 71.6235 ms (decrypt data on GPU)
TIMING: 42.8299 ms (copy data from GPU to CPU)
TIMING: 163.035 ms (total time on GPU)
STATUS: test passed
TIMING: 8.90096 ms (checking result on CPU)
TIMING: 84.1282 ms (free memory)
```

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum          Name
 -------  ---------------  ---------  -----------  ---------  ---------  --------------------
    41.6        371714936          2  185857468.0       2291  371712645  cudaEventCreate
    30.6        273186824          1  273186824.0  273186824  273186824  cudaHostAlloc
    18.4        164427229          2   82213614.5   48418619  116008610  cudaMemcpy
     9.3         82771586          1   82771586.0   82771586   82771586  cudaFreeHost
     0.1          1066987          1    1066987.0    1066987    1066987  cudaMalloc
     0.1           900426          1     900426.0     900426     900426  cudaFree
     0.0            43019          1      43019.0      43019      43019  cudaLaunchKernel
     0.0            28532          2      14266.0       7081      21451  cudaEventRecord
     0.0             8092          1       8092.0       8092       8092  cudaEventSynchronize
     0.0             6113          2       3056.5       1750       4363  cudaEventDestroy


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances  Average   Minimum   Maximum                                                Name
 -------  ---------------  ---------  --------  --------  --------  ----------------------------------------------------------------------------
   100.0         73201686          1  73201686.0  73201686  73201686  decrypt_gpu(unsigned long*, unsigned long, unsigned long)


CUDA Memory Operation Statistics (by time):

 Time(%)  Total Time (ns)  Operations  Average   Minimum  Maximum      Operation
 -------  ---------------  ----------  --------  -------  --------  ------------------
    53.1         48375502           1  48375502.0  48375502  48375502  [CUDA memcpy HtoD]
    46.9         42793639           1  42793639.0  42793639  42793639  [CUDA memcpy DtoH]


CUDA Memory Operation Statistics (by size in KiB):

    Total      Operations   Average    Minimum    Maximum      Operation
-----------  ----------  -----------  ---------  ---------  ------------------
524288.000           1  524288.000  524288.000  524288.000  [CUDA memcpy DtoH]
524288.000           1  524288.000  524288.000  524288.000  [CUDA memcpy HtoD]
```



nsys profile

출처 : https://junstar92.tistory.com/432

Test Code 리뷰

```cpp
int num_gpus;
cudaGetDeviceCount(&num_gpus);

int device;
cudaGetDevice(&device); // 'device' is now a 0-based index of the current GPU.

printf("GPU num : %d₩ncurrent GPU : %d ₩n", num_gpus, device);
```

```
Microsoft Visual Stu
GPU num : 1
current GPU : 0
```

메모리 할당

```cpp
timer.start();
cudaMallocHost(&data_cpu, sizeof(uint64_t) * num_entries);
cudaMalloc(&data_gpu, sizeof(uint64_t) * num_entries);
timer.stop("allcate memory");
```

```cpp
const int num_gpus;
cudaGetDeviceCount(&num_gpus);

const uint64_t num_entries = 1UL << 26;
const uint64_t chunk_size = sdiv(num_entries, num_gpus);

uint64_t *data_gpu[num_gpus]; // One pointer for each GPU.

for (int gpu = 0; gpu < num_gpus; gpu++) {
    cudaSetDevice(gpu);

    const uint64_t lower = chunk_size * size;
    const uint64_t upper = min(lower + chunk_size, num_entries);
    const uint64_t width = upper - lower;

    // Allocate chunk of data for current GPU.
    cudaMalloc(&data_gpu[gpu], sizeof(uint64_t) * width);
}
```

메모리 복사

```cpp
timer.start();
// Copy data from GPU to CPU
cudaMemcpy(data_cpu, data_gpu, sizeof(uint64_t) * num_entries, cudaMemcpyDeviceToHost);
timer.stop("copy data from GPU to CPU");
```

```cpp
// Assume data has been allocated on host and for each GPU
for (int gpu = 0; gpu < num_gpus; gpu++) {
    cudaSetDevice(gpu);

    const uint64_t lower = chunk_size * gpu;
    const uint64_t upper = min(lower + chunk_size, num_entries);
    const uint64_t width = upper - lower;

    // Note use of 'cudaMemcpy' and not 'cudaMemcpyAsync' since we are not
    // presently using non-default streams.
    cudaMemcpy(data_gpu[gpu], data_cpu + lower, sizeof(uint64_t) * width,
        cudaMemcpyHostToDevice); // .. or cudaMemcpyDeviceToHost
}
```

```cpp
timer.start();
// Decrypt data on GPU(s).
decrypt_gpu<<<80 * 32, 64>>>(data_gpu, num_entries, num_iters);
timer.stop("decrypt data on GPU");
```

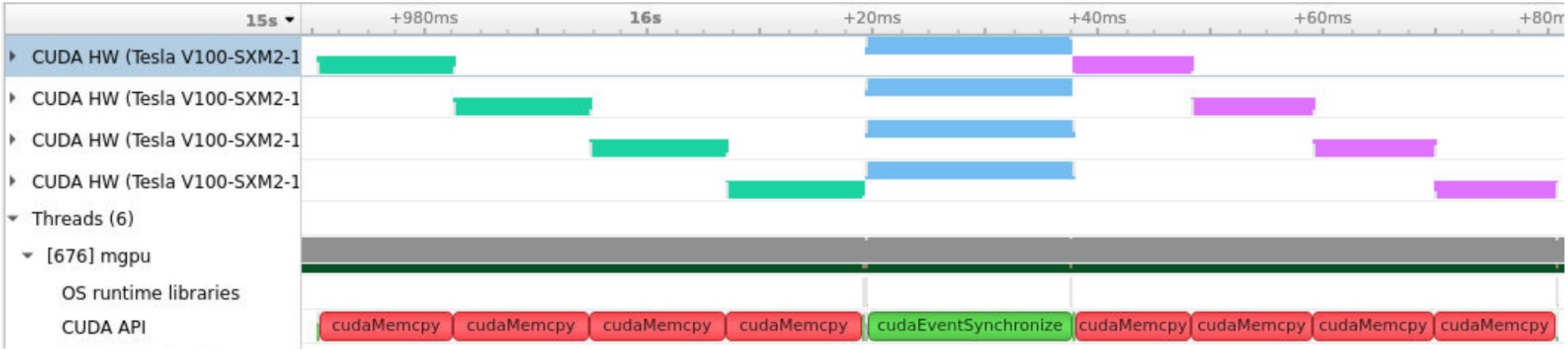
커널함수 동작

```cpp
// Assume data has been allocated on host and for each GPU
for (int gpu = 0; gpu < num_gpus; gpu++) {
    cudaSetDevice(gpu);

    const uint64_t lower = chunk_size * gpu;
    const uint64_t upper = min(lower + chunk_size, num_entries);
    const uint64_t width = upper - lower;

    // Pass chunk of data for current GPU to work on.
    kernel<<<grid, block>>>(data_gpu[gpu], width);
}
```

Test Code 리뷰

```
int num_gpus;
cudaGetDeviceCount(&num_gpus);

int device;
cudaGetDevice(&device); // 'device' is now a 0-based index of the current GPU.

printf("GPU num : %d₩ncurrent GPU : %d ₩n", num_gpus, device);
```

```
// Assume data has been allocated on host and for each GPU
for (int gpu = 0; gpu < num_gpus; gpu++) {
    cudaSetDevice(gpu);

    const uint64_t lower = chunk_size * gpu;
    const uint64_t upper = min(lower + chunk_size, num_entries;
    const uint64_t width = upper - lower;

    // Pass chunk of data for current GPU to work on.
    kernel<<<grid, block>>>(data_gpu[gpu], width);
}
```



Test Code 리뷰

단일 GPU 결과

```
TIMING: 48.3961 ms (copy data from CPU to GPU)
TIMING: 71.6235 ms (decrypt data on GPU)
TIMING: 42.8299 ms (copy data from GPU to CPU)
TIMING: 163.035 ms (total time on GPU)
STATUS: test passed
TIMING: 8.90096 ms (checking result on CPU)
TIMING: 84.1282 ms (free memory)
```

다중 GPU 결과

```
TIMING: 48.6154 ms (copy data from CPU to GPU)
TIMING: 18.3179 ms (total kernel execution on GPU)
TIMING: 42.9864 ms (copy data from GPU to CPU)
TIMING: 110.158 ms (total time on GPU)
STATUS: test passed
TIMING: 9.14307 ms (checking result on CPU)
TIMING: 86.6727 ms (free memory)
```

71ms 에서 18ms 로 약 4배(늘어난 GPU수 만큼) 빨라짐

감 사 합 니 다