report

2022_28379 협동과정 인공지능 전공 임성준

▼ 다섯 번째 과제 22.11.23 수

▼ GPU 정보 확인하기

(a) (6점) 다음 커맨드들의 의미와 실행 결과를 답하라. 결과가 너무 길면 적당히 앞부분만 잘라서 첨부.

- srun --partition=shpc22 --gres=gpu:4 nvidia-smi 의미
  - gpu장치, gpu drvier version, cuda version 등의 정보를 알 수 있습니다. 또한 gpu의 총 메모리 중에서 현재 사용되고 있는 메모리는 어느 정도 인지 현재 사용하고 있는 전력량은 얼마인지 모니터링 할 수 있습니다.
  - gpu/fan부분은 gpu number입니다.
  - temp는 온도로 일정 수준 이상이 되면 성능 저하가 발생합니다.
  - perf는 Performance mode로 p0 - p12까지 있는데 숫자가 적을 수록 성능이 높은 상태입니다.
  - pwr은 현재 사용량 / 최대 용량입니다.
  - memory-usage는 사용 중인 메모리량과 총 량을 의미합니다.
  - volatile GPU-Utill은 gpu 성능치를 의미합니다.
  - processes는 현재 사용 중인 프로세스가 나타납니다.
  - compute m은 compute mode가 나타납니다.
- 실행결과

```
shpc123@login0:~$ srun --partition=shpc22 --gres=gpu:4 nvidia-smi
Wed Nov 23 19:41:37 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.43.04    Driver Version: 515.43.04    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:25:00.0 Off |                  N/A |
| 30%   34C    P0   111W / 350W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce ...  Off  | 00000000:41:00.0 Off |                  N/A |
| 30%   33C    P0   116W / 350W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce ...  Off  | 00000000:A1:00.0 Off |                  N/A |
| 30%   32C    P0   115W / 350W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA GeForce ...  Off  | 00000000:C1:00.0 Off |                  N/A |
| 30%   31C    P0   102W / 350W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- srun --partition=shpc22 --gres=gpu:4 nvidia-smi -q 의미
  - GPU의 거의 모든 정보를 보여주는 옵션이다. 여기에서 grep 명령어를 통해 필요한 정보만을 볼 수 있다.
- 실행결과

```
shpc123@login0:~$ srun --partition=shpc22 --gres=gpu:4 nvidia-smi -q

==============NVSMI LOG==============

Timestamp                                 : Wed Nov 23 19:54:47 2022
Driver Version                            : 515.43.04
CUDA Version                              : 11.7

Attached GPUs                             : 4
GPU 00000000:25:00.0
    Product Name                          : NVIDIA GeForce RTX 3090
    Product Brand                         : GeForce
    Product Architecture                  : Ampere
    Display Mode                          : Disabled
    Display Active                        : Disabled
    Persistence Mode                      : Disabled
    MIG Mode
        Current                           : N/A
        Pending                           : N/A
    Accounting Mode                       : Disabled
    Accounting Mode Buffer Size           : 4000
    Driver Model
        Current                           : N/A
        Pending                           : N/A
    Serial Number                         : N/A
    GPU UUID                              : GPU-30af9669-2a60-b0b3-8d53-f5bf700635c7
```

- srun --partition=shpc22 --gres=gpu:4 clinfo 의미
  - opencl의 platform과 device의 정보를 보여준다. 여기서는 NVIDIA CUDA가 platform이고 NVIDIA GeForce RTX 3090이 device로 나타났다.
- 실행결과

```
shpc123@login0:~$ srun --partition=shpc22 --gres=gpu:4 clinfo
  Number of platforms                               1
    Platform Name                                   NVIDIA CUDA
    Platform Vendor                                 NVIDIA Corporation
    Platform Version                                OpenCL 3.0 CUDA 11.7.57
    Platform Profile                                FULL_PROFILE
    Platform Extensions                             cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_fp64 cl_khr_3d_image_writes cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll cl_nv_copy_opts cl_nv_create_buffer cl_khr_int64_base_atomics cl_khr_int64_extended_atomics cl_khr_device_uuid cl_khr_pci_bus_info cl_khr_external_semaphore cl_khr_external_memory cl_khr_external_semaphore_opaque_fd cl_khr_external_memory_opaque_fd
    Platform Host timer resolution                  0ns
    Platform Extensions function suffix             NV

    Platform Name                                   NVIDIA CUDA
  Number of devices                                 4
    Device Name                                     NVIDIA GeForce RTX 3090
    Device Vendor                                   NVIDIA Corporation
    Device Vendor ID                                0x10de
    Device Version                                  OpenCL 3.0 CUDA
    Driver Version                                  515.43.04
    Device OpenCL C Version                         OpenCL C 1.2
    Device Type                                     GPU
    Device Topology (NV)                            PCI-E, 25:00.0
    Device Profile                                  FULL_PROFILE
    Device Available                                Yes
    Compiler Available                              Yes
    Linker Available                                Yes
    Max compute units                               82
    Max clock frequency                             1695MHz
    Compute Capability (NV)                         8.6
    Device Partition                                (core)
      Max number of sub-devices                     1
      Supported partition types                     None
      Supported affinity domains                    (n/a)
```

(b) (6점) 실습 서버의 계산 노드에 장착된 GPU의 모델명과 노드당 장착된 GPU의 개수를 답하라.

- GPU 모델명: NVIDIA GeForce RTX 3090
- 개수: 4개

(c) (6점) 실습 서버의 계산 노드에 장착된 GPU 하나의 메모리 크기를 MiB 단위로 답하라.

- 24576MiB

(d) (6점) 실습 서버의 계산 노드에 장착된 GPU의 maximum power limit(W)과 maximum SM clock speed(Mhz)를 답하라.

- max power limit: 350.00 W
- max sm clock speed: 2100 Mhz

(e) (6점) 실습 서버의 계산 노드에 장착된 GPU를 OpenCL 을 이용해 사용할 때 Max work item dimension,Max work item size, Max work group size 를 답하라

- max work item dimensions: 3
- max work item size: 1024x1024x64
- max work group size: 1024

▼ Matrix Multiplication with OpenCL

두 FP32 행렬의 곱을 하나의 GPU를 이용해 계산하는 프로그램 matmul.c, kernel.cl 을 작성하라.

- 자신의 병렬화 방식에 대한 설명.
  - 첫 번째, global memory 접근을 최소화하였습니다.
    - 기존의 코드는 모든 thread가 C의 한 element를 계산할 때마다 global memory에 접근했습니다. global memory access에는 local memory에 비해 상대적으로 시간이 오래걸리므로 이를 개선하기 위해서 tiling방법을

선택했습니다. tiling방법은 subBlock을 이용해서 하나의 work-group내에서 작업하도록 합니다. work-group내의 thread는 local memory를 공유하기때문에 global memory에 접근하지 않고도 데이터를 연산할 수 있습니다. 이와 같은 방법을 통해서 tile 당 2번의 global memory 접근이 이루어지고 tile 내부에서는 local memory의 접근이 이루어집니다.

- 두 번째, local memory 접근을 최소화하였습니다.
  - gpu thread마다 할당되는 workload의 양을 늘리면 local memory에 접근하는 수도 줄어들게 됩니다. 위의 첫 번째 개선을 이루고 나서 추가적인 개선을 했습니다. 하나의 스레드가 연속적인 행들에 놓인 n개의 원소들에 대해서 계산하면 local memory 접근을 n배 줄일 수 있습니다. n개의 스레드가 해야하는 일을 1개의 스레드가 처리하기때문에 1번만 접근해도 됩니다. 이렇게 thread마다 할당되는 work-item을 늘리는 것은 vector data type을 적용해서 해결할 수 있습니다. float8을 이용하면 하나의 vector type에 8개의 single precision float이 들어가게 됩니다.

아래는 첫 번째와 두 번째 방법을 사용한 커널입니다.

```
#define NUM_WORK_ITEM 32
#define VECTOR_WIDTH 8

__kernel void sgemm(__global float8 *A, __global float8 *B, __global float8 *C, int M, int N, int K){
  const int row = get_local_id(0);
  const int col = get_local_id(1);
  const int global_row = NUM_WORK_ITEM * get_group_id(0) + row;
  const int global_col = (NUM_WORK_ITEM/VECTOR_WIDTH) * get_group_id(1) + col;

  __local float8 tileA[NUM_WORK_ITEM][NUM_WORK_ITEM/VECTOR_WIDTH];
  __local float8 tileB[NUM_WORK_ITEM][NUM_WORK_ITEM/VECTOR_WIDTH];

  float8 inter_value = { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };

  const int num_tiles = K/NUM_WORK_ITEM;

  for (int t = 0; t < num_tiles; t++){
    const int t_row = NUM_WORK_ITEM * t + row;
    const int t_col = (NUM_WORK_ITEM/VECTOR_WIDTH) * t + col;
    tileA[row][col] = A[global_row * (K/VECTOR_WIDTH) + t_col];
    tileB[row][col] = B[t_row * (N/VECTOR_WIDTH) + global_col];

    barrier(CLK_LOCAL_MEM_FENCE);

    float8 vectorA, vectorB;
    float valA;

    for(int k = 0; k < NUM_WORK_ITEM/VECTOR_WIDTH; k++){
      vectorA = tileA[row][k];
      for(int w = 0; w < VECTOR_WIDTH; w++){
        vectorB = tileB[VECTOR_WIDTH*k + w][col];

        switch(w){
          case 0: valA = vectorA.s0; break;
          case 1: valA = vectorA.s1; break;
          case 2: valA = vectorA.s2; break;
          case 3: valA = vectorA.s3; break;
          case 4: valA = vectorA.s4; break;
          case 5: valA = vectorA.s5; break;
          case 6: valA = vectorA.s6; break;
          case 7: valA = vectorA.s7; break;
        }

        inter_value += vectorB * valA;
      }
    }
    barrier(CLK_LOCAL_MEM_FENCE);
  }
  C[global_row*(N/VECTOR_WIDTH) + global_col] = inter_value;
}
```

- 세 번째, 임의의 dimension에도 행렬 곱이 가능하도록 zero-padding을 사용했습니다.
  - 위의 두 가지 방법을 통해서 performance 향상에는 도움이 됐지만 임의의 dimension을 가진 행렬에서는 행렬 곱이 작동하지 않았습니다. 따라서 이를 해결하고자 tile의 크기인 32를 기준으로 32배수가 되도록 만들어주었습니다. 예를 들어서, 행의 크기가 31이라면 32로 만들어주고 열의 크기가 1022라면 1024로 만들어주었습니다. kernel이 아닌 host source code에서 패딩처리를 해주었습니다. buffer 생성 이전 그리고 host to

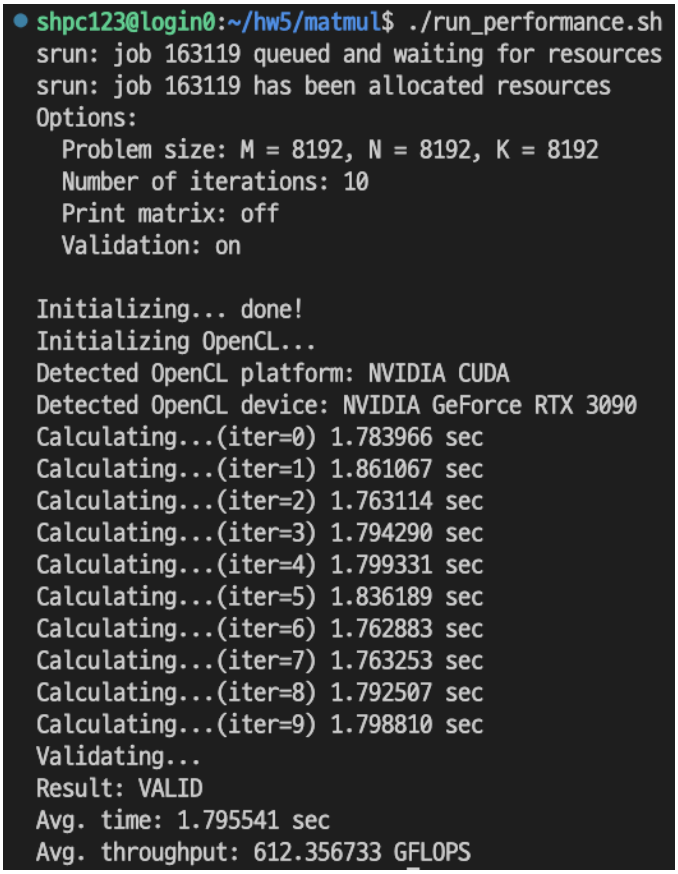
device 전에 zero-padding한 A, B를 통해 행렬 곱을 진행하고 결과 행렬은 padding을 제거하여 C를 결과로 얻을 수 있었습니다.

아래는 세 번째 방법을 이용한 Host 코드입니다.

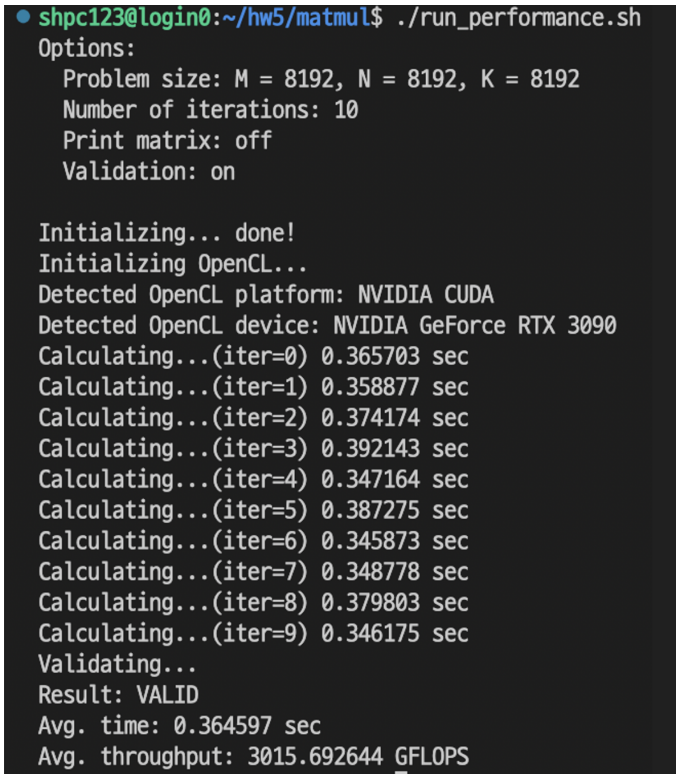
```
int p_M = M;
int p_N = N;
int p_K = K;

if(M%32 != 0){
  r = 32 - (M-(32*(M/32)));
  M = M + r;
}

if(N%32 != 0){
  r = 32 - (N-(32*(N/32)));
  N = N + r;
}

if(K%32 != 0){
  r = 32 - (K-(32*(K/32)));
  K = K + r;
}

float *A_PAD;
float *B_PAD;
float *C_PAD;
alloc_mat(&C_PAD, M, N);
zero_mat(C_PAD, M, N);

if(M%32 != 0 || N%32 !=0 || K%32 != 0 || M != N || M != K || N != K){
  alloc_mat(&A_PAD, M, K);
  alloc_mat(&B_PAD, K, N);
  zero_mat(A_PAD, M, K);
  zero_mat(B_PAD, K, N);

  for(int i = 0; i < M; i++){
      for(int j = 0; j < K; j++){
        float value;
        if(i < p_M && j < p_K){
          value = A[i*p_K + j];
        }else{
          value = 0;
        }
        A_PAD[i*K + j] = value;
      }
  }
  for(int i = 0; i < K; i++){
      for(int j = 0; j < N; j++){
        float value;
        if(i < p_K && j < p_N){
          value = B[i*p_N + j];
        }else{
          value = 0;
        }
        B_PAD[i*N + j] = value;
      }
  }
}else{
  A_PAD= (float*)A;
  B_PAD= (float*)B;
}
```

- 뼈대 코드 matmul.c의 각 부분에 대한 설명. matmul initialize, matmul, matmul finalize 함수 각각에서 사용하는 OpenCL API 및 각 API에 대한 간략한 설명. (API 당 한문장이면 충분).
    - matmul initialize:
        - clGetPlatformIDs: 사용가능한 플랫폼을 리스트로 반환합니다. (컴퓨터마다 내장된 cpu, gpu의 제조사)
        - clGetDeviceIDs: 해당 플랫폼에서 사용가능한 device를 반환합니다. 이번 과제에서는 gpu를 가져왔습니다.
        - clCreateContext: 커널이 실행되는 환경으로 디바이스에 메모리객체와 명령큐를 연결시켜주는 역할을 하는 context를 생성합니다.
        - clCreateCommandQueue: 호스트에서 디바이스 별로 생성되는 command queue를 생성합니다.
        - clCreateProgramWithSource: 프로그램 객체를 생성합니다. 소스코드를 로드합니다.
        - clCreateKernel: 커널 객체를 반환합니다.
        - clCreateBuffer: 버퍼 객체를 생성합니다.
    - matmul
        - clEnqueueWriteBuffer: host에 있는 것을 device로 이동시킵니다.
        - clFinish: command queue에 있는 명령어가 모두 완료될때까지 block합니다.
        - clSetKernelArg: 커널의 argument를 set합니다.
        - clEnqueueNDRangeKernel: 디바이스에서 커널을 실행시키기 위해서 명령어를 enqueue합니다.
        - clEnqueueReadBuffer: device로부터 읽어옵니다.
    - matmul finalize
        - 없음.

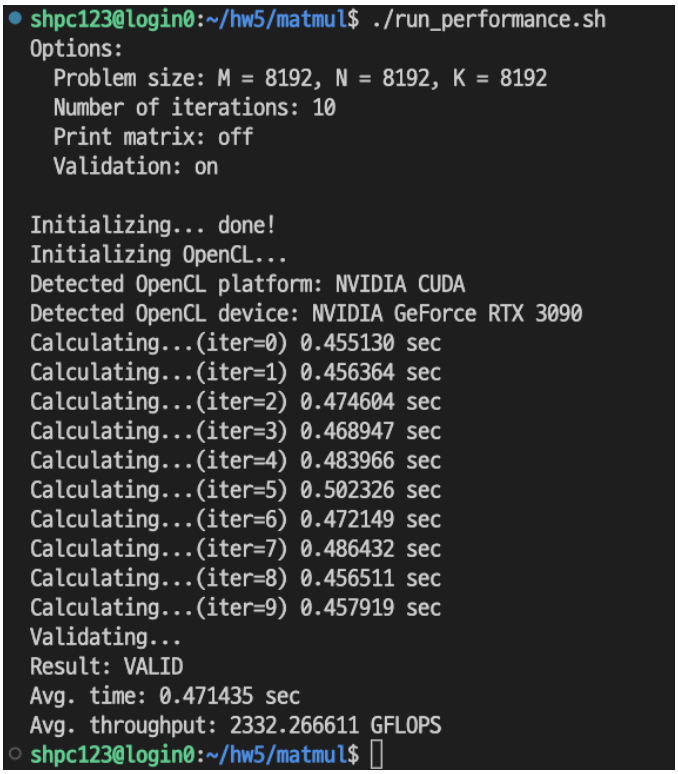
- 자신이 적용한 코드 최적화 방식을 분류하고, 각각에 대한 성능 실험 결과. (Matrix multiplication은 향후 프로젝트 진행에 있어 핵심적으로 사용될 예정이므로 해당 실험을 적극적으로 해보길 권장함.)
  - naive 구현
    - 시간 초과하였습니다.
  - global memory 접근 최소화
    - 612 GFLOPS



  - local memory 접근 최소화: 가장 높은 성능을 보입니다.
    - 3015 GFLOPS



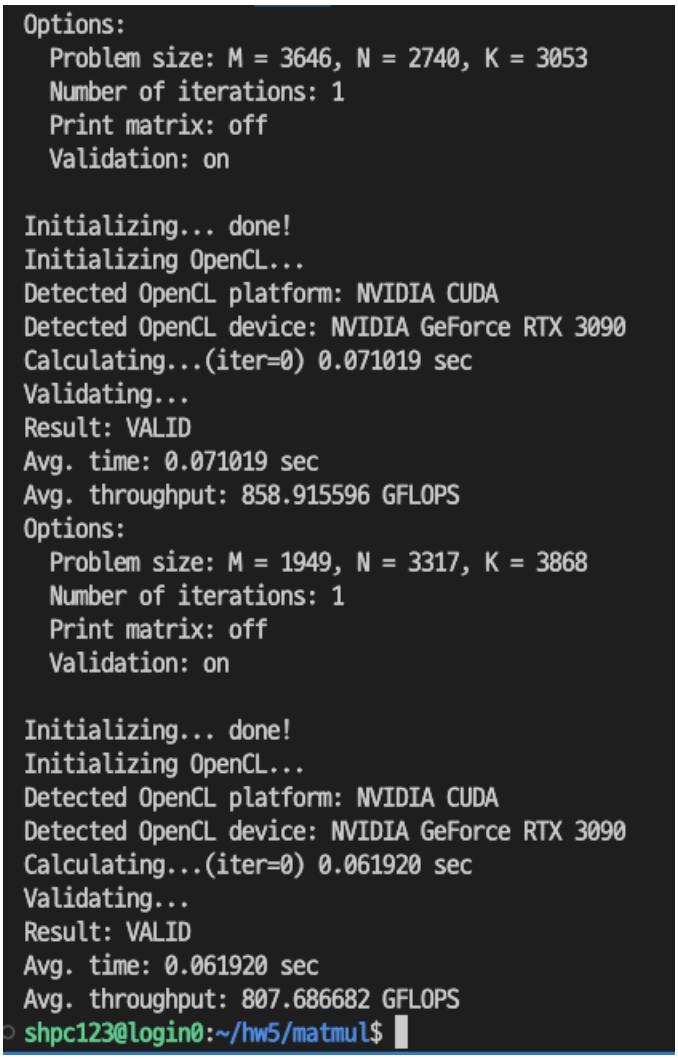
- zero-padding: 임의의 행렬을 처리하기 위해 패딩하는 과정에서 성능이 떨어졌습니다.
  - 2332 GFLOPS



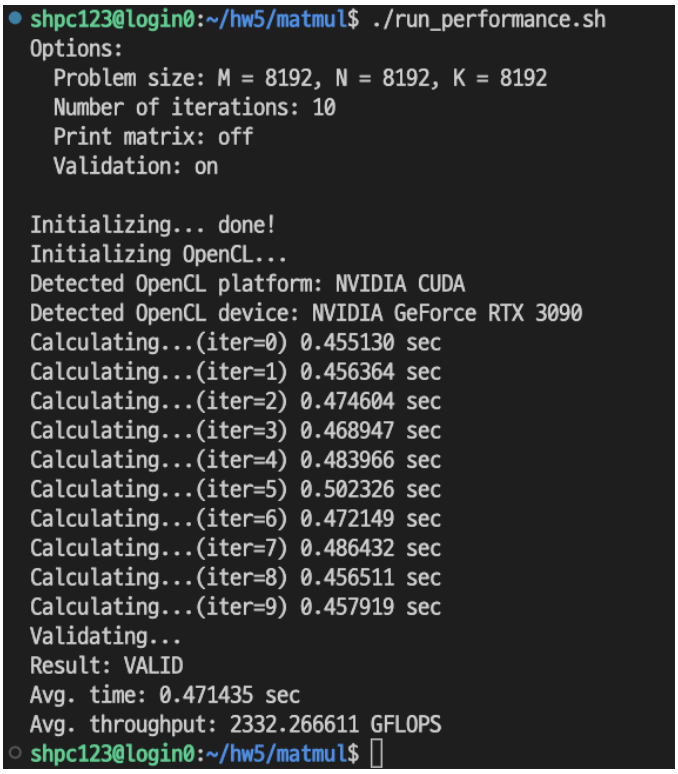
- validation 결과: 모든 case를 통과했습니다.

```
shpc123@login0:~/hw5/matmul$ ./run_validation.sh
Options:
  Problem size: M = 831, N = 538, K = 2304
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.007629 sec
Validating...
Result: VALID
Avg. time: 0.007629 sec
Avg. throughput: 270.032218 GFLOPS
Options:
  Problem size: M = 3305, N = 1864, K = 3494
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.059040 sec
Validating...
Result: VALID
Avg. time: 0.059040 sec
Avg. throughput: 729.165672 GFLOPS
Options:
  Problem size: M = 618, N = 3102, K = 1695
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.016520 sec
Validating...
Result: VALID
Avg. time: 0.016520 sec
Avg. throughput: 393.397599 GFLOPS
Options:
  Problem size: M = 1876, N = 3453, K = 3590
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.060483 sec
Validating...
Result: VALID
Avg. time: 0.060483 sec
Avg. throughput: 768.986549 GFLOPS
```

```
Options:
  Problem size: M = 1228, N = 2266, K = 1552
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.018259 sec
Validating...
Result: VALID
Avg. time: 0.018259 sec
Avg. throughput: 473.042583 GFLOPS
Options:
  Problem size: M = 347, N = 171, K = 688
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.001065 sec
Validating...
Result: VALID
Avg. time: 0.001065 sec
Avg. throughput: 76.662862 GFLOPS
Options:
  Problem size: M = 3583, N = 962, K = 765
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.016636 sec
Validating...
Result: VALID
Avg. time: 0.016636 sec
Avg. throughput: 316.995879 GFLOPS
Options:
  Problem size: M = 2962, N = 373, K = 1957
  Number of iterations: 1
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.015520 sec
Validating...
Result: VALID
Avg. time: 0.015520 sec
Avg. throughput: 278.629919 GFLOPS
```



- performance 결과: 1개의 gpu device를 사용해서 2000GFLOPS를 넘겼습니다.


shpc123@login0:~/hw5/matmul$ ./run_performance.sh
Options:
Problem size: M = 8192, N = 8192, K = 8192
Number of iterations: 10
Print matrix: off
Validation: on
Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.455130 sec
Calculating...(iter=1) 0.456364 sec
Calculating...(iter=2) 0.474604 sec
Calculating...(iter=3) 0.468947 sec
Calculating...(iter=4) 0.483966 sec
Calculating...(iter=5) 0.502326 sec
Calculating...(iter=6) 0.472149 sec
Calculating...(iter=7) 0.486432 sec
Calculating...(iter=8) 0.456511 sec
Calculating...(iter=9) 0.457919 sec
Validating...
Result: VALID
Avg. time: 0.471435 sec
Avg. throughput: 2332.266611 GFLOPS
shpc123@login0:~/hw5/matmul$