M3239.005400 데이터사이언스를 위한 컴퓨팅 2 (001)

Homework #4

Due : 2024/11/17 (Sun)

2024-28413 이정현

1. GPU 정보 확인하기

(a) 다음 커맨드들의 의미와 실행 결과를 답하라

- **srun --partition=class1 --gres=gpu:4 nvidia-smi**

  이 커맨드는 class1 파티션에서 4개의 GPU를 할당받은 후, nvidia-smi를 실행한다. nvidia-smi는 NVIDIA System Management Interface를 의미하고, GPU에 대한 다양한 정보를 얻고 관리 및 모니터링을 할 수 있는 도구이다. (Utilization, memory usage, temperature 등등의 정보 확인 가능) 아래 Figure 1 에서 실행 결과를 확인할 수 있고, 할당된 4개의 GPU 상태를 확인할 수 있다.

```
shpc106@elogin3:~$ srun --partition=class1 --gres=gpu:4 nvidia-smi
srun: job 980565 queued and waiting for resources
srun: job 980565 has been allocated resources
Tue Nov  5 12:40:37 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| 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 TITAN RTX    On   | 00000000:18:00.0 Off |                  N/A |
| 41%   33C    P8    30W / 280W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA TITAN RTX    On   | 00000000:3B:00.0 Off |                  N/A |
| 41%   27C    P8    13W / 280W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA TITAN RTX    On   | 00000000:86:00.0 Off |                  N/A |
| 41%   26C    P8     8W / 280W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA TITAN RTX    On   | 00000000:AF:00.0 Off |                  N/A |
| 41%   26C    P8     3W / 280W |      0MiB / 24576MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

Figure 1. <srun --partition=class1 --gres=gpu:4 nvidia-smi>

- **srun --partition=class1 --gres=gpu:4 nvidia-smi -q**

  이 커맨드는 class1 파티션에서 4개의 GPU를 할당받은 후, nvidia-smi -q를 실행한다. 앞서 살펴보았던 커맨드와 차이점은 -q로, q는 query를 의미한다. 이 커맨드를 실행하면 GPU들의 더 자세한 상태를 확인해볼 수 있다. 아래 Figure 2 에서 실행 결과를 확인할 수 있고, 4개의 GPU가 할당된 것이 확인 가능함과 동시에 세부 정보들을 볼 수 있다. 출력의 앞부분만 첨부하였다.

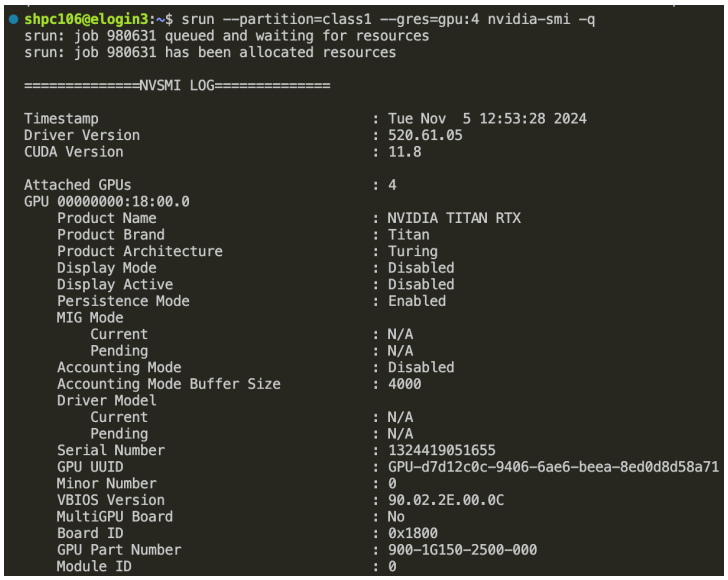
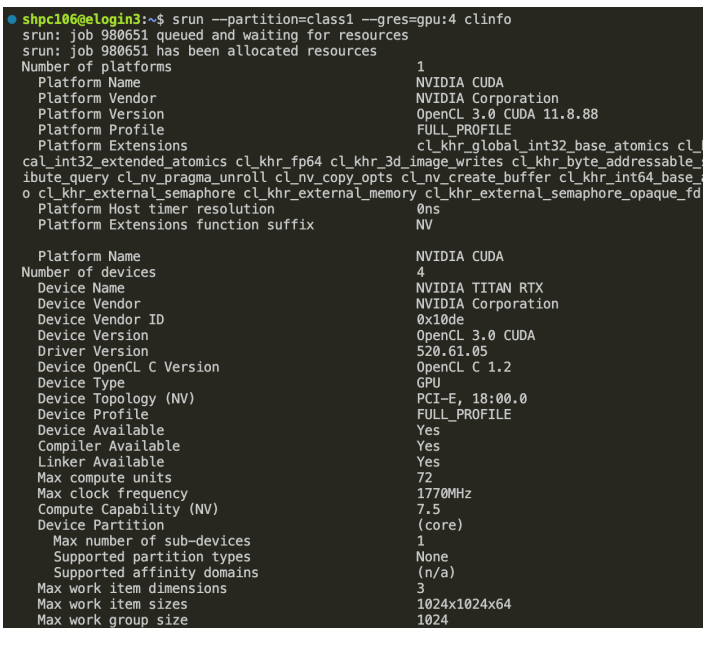


Figure 2. <srun --partition=class1 --gres=gpu:4 nvidia-smi -q>

- **srun --partition=class1 --gres=gpu:4 clinfo**

  이 커맨드는 class1 파티션에서 4개의 GPU를 할당받은 후, clinfo를 실행한다. clinfo를 사용하면 OpenCL을 지원하는 플랫폼과 장치들에 대한 세부 정보를 출력해서 확인할 수 있다. 아래 Figure 3 에서 실행 결과를 확인할 수 있고, CUDA platform이 지원된다는 것을 볼 수 있다. 이와 더불어 4개의 GPU가 할당된것이 확인되고, 각 GPU에 대한 세부 정보 또한 확인할 수 있다. 출력의 앞부분만 첨부하였다.

Figure 3. <srun --partition=class1 --gres=gpu:4 clinfo>



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

Answer: NVIDIA TITAN RTX, 노드 당 4개

(이전 문제 (a)의 출력들에서 이를 확인할 수 있다)

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

Answer: 24576 MiB

(nvidia-smi로 이를 확인할 수 있다)

(d) 실습 서버의 계산 노드에 장착된 GPU의 maximum power limit(W)과 maximum SM clock speed(MHz)를 답하라

Maximum power limit(W) : 280W

Maximum SM clock speed(MHz) : 2100MHz

(nvidia-smi, nvidia-smi -q 로 각각 확인할 수 있다)

(e) 실습 서버의 계산 노드에 장착된 GPU를 OpenCL을 이용해 사용할 때 Max work item dimension, Max work item size, Max work group size를 답하라

Max work item dimension : 3

Max work item size : 1024x1024x64

Max work group size : 1024

(clinfo로 확인할 수 있다)

2. Matrix Multiplication with OpenCL

-병렬화 방식에 대한 설명

이번 병렬화를 위해서는 크게 두 가지 방법을 사용하였다.

(1) Tiling (Block단위 연산)

Local memory를 선언한 후, 행렬들을 작은 블록으로 나누어서 할당해주었다. 하나의 work-group은 블록으로 나뉘어진 행렬의 연산을 담당하게 된다. 연산 전후로 barrier를 적절하게 사용함으로써 synchronization 문제를 없애주었다.

(2) 하나의 thread가 여러개의 원소 연산

하나의 thread(work-item)이 여러개의 원소들을 담당하도록 하였다.
NUM_ELEM_PER_THREAD (NEPT)라는 변수를 선언해서 NEPT x NEPT 크기의 sub-block을 만들어준 후, 각 work-group은 이 sub-block안에 속한 element들을 연산한다. 이는 thread의 개수를 줄이면서 data reuse를 늘릴 수 있기 때문에 효율적인 병렬화 방법이였다. 또한, sub-block 크기의 accumulate 행렬을 선언해서 중간 계산값들을 누적해서 저장해주고 output matrix C에 써주는 방법으로 구현하였다.

마지막으로는 padding을 사용해서 최적화를 마무리하였다.

```
//loop over all BLOCKS to compute C[globalRow, globalCol]
for(int t=0; t < (K + BLOCK_SIZE - 1)/ BLOCK_SIZE; t++){
  //load tiles of A and B to local memory
  for(int i=0; i < NUM_ELEM_PER_THREAD; i++){
    for(int j=0; j< NUM_ELEM_PER_THREAD; j++){
      if(globalRow + i < M && ((t * BLOCK_SIZE) + localCol + j) < K){
        A_block[localRow + i][localCol + j] = A[((globalRow + i) * K) + (t * BLOCK_SIZE) + localCol + j];
      }
      else{
        A_block[localRow + i][localCol + j] = 0.0f;
      }

      if(globalCol + j < N && ((t * BLOCK_SIZE) + localRow + i) < K){
        B_block[localRow + i][localCol + j] = B[(t * BLOCK_SIZE + localRow + i) * N + globalCol + j];
      }
      else{
        B_block[localRow + i][localCol + j] = 0.0f;
      }
    }
  } //end of tile loop
```

```
//compute on loaded tile
for (int k = 0; k < BLOCK_SIZE; k++) {
  for (int i = 0; i < NUM_ELEM_PER_THREAD; i++) {
    for (int j = 0; j < NUM_ELEM_PER_THREAD; j++) {
      acc[i][j] += A_block[localRow + i][k] * B_block[k][localCol + j];
    }
  }
}
```

<Local memory에 sub-block 단위로 데이터 로딩 및 연산 결과 acc 행렬에 저장>

-matmul.c의 각 부분에 대한 설명

(1) matmul_initialize에서 사용된 API List

-clGetPlatformIDs: 현재 system에서 사용할 수 있는 OpenCL platform ID를 구한다

-clGetDeviceIDs: 사용하는 OpenCL platform에 대한 device ID를 구한다

-clCreateContext: OpenCL context를 만든다

-clCreateCommandQueue: Context에 대한 command queue를 만든다. (수행될 커널들이 들어감)

-clCreateKernel: 프로그램에서 실행시키고자 하는 커널을 만든다

-clCreateBuffer: 메모리를 할당하기 위해서 Buffer를 만든다

(2) matmul에서 사용된 API List

-clEnqueueWriteBuffer: Host에서 Device로 데이터를 복사해준다

-clSetKernelArg: Kernel에 전달할 인자(argument)를 설정해준다

-clEnqueueNDRangeKernel: Kernel을 실행하기 위해서 command queue에 커맨드 추가

-clEnqueueReadBuffer: Device에서 Host로 데이터를 복사해준다

-clFinish: command queue안에 queue된 모든 커맨드들이 실행을 완료할때까지 host를 막는 synchronization function이다.

(3) matmul_finalize에서 사용된 API List

-clReleaseMemObject: 할당된 메모리 object를 해제한다

-clReleaseKernel: Kernel object를 해제한다

-clReleaseProgram: OpenCL program을 해제한다

-clReleaseCommandQueue: Command queue를 해제한다

-clReleaseContext : OpenCL context를 해제한다

-최적화 방식의 분류 및 각각에 대한 성능 실험 결과

이 부분에서 측정된 성능은 모두 **run_performance.sh에 대해** 측정되었다.

- 가장 단순한 GEMM kernel. **(52 GFLOPS)**

  별다른 방법을 적용하지 않은 가장 기본적인 행렬곱 연산을 구현하였다. Global work size는 output matrix C의 크기인 {M,N} 으로 설정하고, local work size는 {1,1}로 설정하였다. 각각의 work element를 C행렬의 원소 하나를 계산하게 함으로써 가장 기본적인 병렬화를 사용하였다. (매우 쉬운 방법이고, 모든 크기의 행렬에 대해 항상 VALID한 결과를 준다)

- BLOCK_SIZE 정의 후, Tiling 및 Local memory 사용. **(670 GFLOPS)**

  BLOCK_SIZE라는 변수를 선언한 후, BLOCK_SIZE x BLOCK_SIZE를 크기로 갖는 타일을 지정해서 하나의 work group이 하나의 타일 영역을 연산하도록 구현하였다. 몇번의 실험을 통해서 BLOCK_SIZE = 8 일때 성능이 가장 좋게 나오는것을 확인하여 그 값을 사용하였다. 또한, kernel 안에서 matrices들을 __local을 이용해서 local memory caching 방법을 적용해주었다. 더 효율적인 메모리 접근과 cache miss를 줄여줌으로써 성능 향상을 목격할 수 있었다고 생각한다.

  Global work size는 임의의 크기 matrix를 오류없이 연산할 수 있도록 (BS=블록사이즈) {((M + BS - 1) /BS) * BS, ((N + BS - 1) /BS) * BS} 으로 설정하고, local work size는 타일 크기인 {BS, BS}로 설정하였다.

- 하나의 work-item(thread) 이 Block안 여러 element들을 연산. **(1618 GFLOPS)**

  이 방법을 사용하면 work-item의 개수를 줄일 수 있어서 thread들 간의 synchronization overhead 또한 줄일 수 있었다. 하나의 Block안에서 work-item들은 하나가 아닌 여러개의 element들을 담당해서 연산한다.

  NUM_ELEM_PER_THREAD라는 변수를 선언해서 하나의 work-item(thread)들은 이제 NUM_ELEM_PER_THREAD x NUM_ELEM_PER_THREAD 크기의 sub-block을 연산하게 된다. 몇번의 실험을 통해서 BLOCK_SIZE는 32, NUM_ELEM_PER_THREAD는 8일때 성능이 가장 높게 나오는 것을 확인하고 사용하였다.

  Global work size는 임의의 크기 matrix를 오류없이 연산할 수 있도록
  {(M + BS* NEPT - 1) / (BS * NEPT) * BS, (N + BS* NEPT - 1) / (BS * NEPT) * BS}
  으로 설정해주었다. (BS=블록사이즈, NEPT=NUM_ELEM_PER_THREAD) 이렇게 설정된 global work size는 오류없이 전체 matrix C를 커버할 수 있다.

Local work size는 { (BS / NEPT), (BS / NEPT) }로 설정하였다. 하나의 work-group은 BS x BS 크기에 대해 작업을 하지만, 그 속 하나의 work-item은 NEPT x NEPT 크기의 sub-block에 대한 연산을 담당하기 때문이다.

- Bank Conflict 감소를 위한 padding. **(1952 GFLOPS)**

  Local memory에 저장되는 matrix A와 B의 row에 대해 2개의 padding을 추가해줌으로서 bank conflict를 효과적으로 감소시키고, 성능 향상을 확인할 수 있었다. (BS x BS 크기로 선언해주었던 local memory를 BS x (BS+2) 로 선언하였다)

  Local memory는 여러개의 bank으로 나누어진 구조로 이루어져 있고, 많은 thread들이 하나의 bank를 동시에 접속하는 일이 벌어지면 bank conflict가 발생하게 되서 병렬성능을 감소시킨다. 2개의 padding은 각 row들을 서로 다른 bank에 배정되게 해주고, conflict를 줄여준다.

-정확성 : ./run_validation.sh 10개 모두 VALID

-성능 : ./run_performance.sh 실행결과 최대값 1959GFLOPS

```
shpc106@login3:~/skeleton/hw4/matmul$ ./run_performance.sh
srun: job 993387 queued and waiting for resources
srun: job 993387 has been allocated resources
Options:
  Problem size: M = 4096, N = 4096, K = 4096
  Number of iterations: 10
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Detected OpenCL platform: NVIDIA CUDA
Detected OpenCL device: NVIDIA TITAN RTX
Calculating...(iter=0) 0.067568 sec
Calculating...(iter=1) 0.067567 sec
Calculating...(iter=2) 0.080141 sec
Calculating...(iter=3) 0.067587 sec
Calculating...(iter=4) 0.067581 sec
Calculating...(iter=5) 0.067547 sec
Calculating...(iter=6) 0.067829 sec
Calculating...(iter=7) 0.067569 sec
Calculating...(iter=8) 0.067554 sec
Calculating...(iter=9) 0.080465 sec
Validating...
Result: VALID
Avg. time: 0.070141 sec
Avg. throughput: 1959.468681 GFLOPS
```