TA Session 6

조교 김진표, 박대영, 신준식, 정재훈

pp-ta@aces.snu.ac.kr

THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

실습 및 시험 일정

- 시험은 실습 과제와 비슷한 형식으로 진행할 예정
  - 배운 병렬화 방식 모두 사용 가능
  - 5/9 (월) 공개, 5/18 (수) 23:59 마감
  - 별도의 필기시험은 없음
  - 실습 시간에 matmul 과제와 함께 Q&A 진행

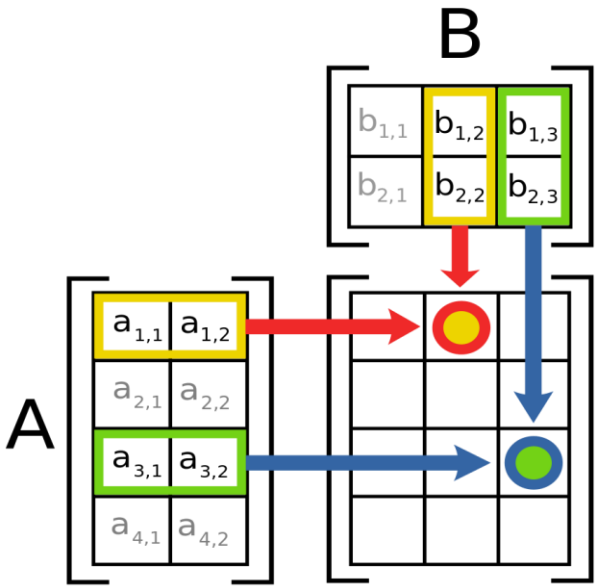
- 앞으로의 일정
  - OpenMP + MPI matmul (5/11 14:29 마감)
  - 5/4 (수) - OpenCL matmul
  - 5/9 (월) - OpenCL matmul (5/13 14:29 마감)
  - 5/11 (수) - CUDA matmul
  - ~~5/13 (금) - CUDA matmul (5/18 23:59 마감)~~

구성

- 과제 4 OpenMP + MPI matmul 해설
- 과제 6 CUDA matmul 소개
  - 채점을 하는 과제임
- 과제 5/6 및 기말과제 진행

과제6: CUDA matmul

- FP32 행렬 둘을 곱하는 프로그램을 CUDA 를 이용해 최적화
  - 1개 노드의 4개 GPU 사용
  - NVIDIA Geforce RTX 3090
- 과제6 은 채점을 함



https://en.wikipedia.org/wiki/Matrix_multiplication#/media/File:Matrix_multiplication_diagram_2.svg

CUDA matmul

- /skeleton/HW6 에 주어진 뼈대 코드를 각자의 홈 디렉토리로 복사해 작업
  - cp -r /skeleton/HW6 ~/
  - Makefile, run.sh, …
  - make 로 컴파일이 가능하도록 Makefile 을 제공
  - 실행을 위한 run.sh 스크립트를 제공

- 뼈대 코드를 수정해 행렬 곱셈을 최적화하는 것이 목표
  - mat_mul.cu 만 수정 가능

CUDA matmul - 실행

- make 로 컴파일
- `run.sh, run_performance.sh, run_validate.sh` 으로 실행

```
kjp4155@login:~/HW6$ ./run.sh -n 5 -v 1024 1024 1024
Options:
  Problem size: M = 1024, N = 1024, K = 1024
  Number of iterations: 5
  Print matrix: off
  Validation: on

Initializing... done!
Initializing OpenCL...
Using 4 devices
[GPU 0] NVIDIA GeForce RTX 3090
[GPU 1] NVIDIA GeForce RTX 3090
[GPU 2] NVIDIA GeForce RTX 3090
[GPU 3] NVIDIA GeForce RTX 3090
Calculating...(iter=0) 0.034087 sec
Calculating...(iter=1) 0.031368 sec
Calculating...(iter=2) 0.031325 sec
Calculating...(iter=3) 0.030135 sec
Calculating...(iter=4) 0.028099 sec
Validating...
Result: VALID
Avg. time: 0.031003 sec
Avg. throughput: 69.267393 GFLOPS
kjp4155@login:~/HW6$
```

CUDA matmul - skeleton code

```
#define CUDA_CALL(f)                                                    \
  {                                                                     \
    cudaError_t err = (f);                                              \
    if (err != cudaSuccess) {                                           \
      fprintf(stderr, "CUDA error at [%s:%d] %d %s\n", __FILE__, __LINE__,  \
              err, cudaGetErrorString(err));                            \
      exit(1);                                                          \
    }                                                                   \
  }

#define MAX_NUM_GPU 4
int num_devices = 0;
```

CUDA matmul - skeleton code

```
55 void mat_mul_init(float *A, float *B, float *C, int _M, int _N, int _K) {
56     M = _M, N = _N, K = _K;
57
58     CUDA_CALL( cudaGetDeviceCount(&num_devices) );
59
60     printf("Using %d devices\n", num_devices);
61     for (int i = 0; i < num_devices; i++) {
62         cudaDeviceProp prop;
63         CUDA_CALL( cudaGetDeviceProperties(&prop, i) );
64
65         // Try printing more detailed information here
66         printf("[GPU %d] %s\n", i, prop.name);
67     }
68
69     if (num_devices <= 0) {
70         printf("No CUDA device found. Aborting\n");
71         exit(1);
72     }
73
74     // Setup problem size for each GPU
75     for (int i = 0; i < num_devices; i++) {
76         Mbegin[i] = (M / num_devices) * i;
77         Mend[i] = (M / num_devices) * (i + 1);
78     }
79     Mend[num_devices - 1] = M;
80
```

CUDA matmul - skeleton code

```
81      // Allocate device memory for each GPU
82      for (int i = 0; i < num_devices; i++) {
83          CUDA_CALL( cudaSetDevice(i) );
84          CUDA_CALL( cudaMalloc(&a_d[i], (Mend[i] - Mbegin[i]) * K * sizeof(float)) );
85          CUDA_CALL( cudaMalloc(&b_d[i], K * N * sizeof(float)) );
86          CUDA_CALL( cudaMalloc(&c_d[i], (Mend[i] - Mbegin[i]) * N * sizeof(float)) );
87      }
88
89      // Upload A and B matrix to every GPU
90      for (int i = 0; i < num_devices; i++) {
91          CUDA_CALL( cudaMemcpy(a_d[i], A + Mbegin[i] * K,
92                                (Mend[i] - Mbegin[i]) * K * sizeof(float),
93                                cudaMemcpyHostToDevice) );
94          CUDA_CALL( cudaMemcpy(b_d[i], B, K * N * sizeof(float), cudaMemcpyHostToDevice) );
95      }
96
97      // DO NOT REMOVE; NEEDED FOR TIME MEASURE
98      for (int i = 0; i < num_devices; i++) {
99          CUDA_CALL( cudaDeviceSynchronize() );
00      }
01  }
```

CUDA matmul - skeleton code

```
38 void mat_mul(float *_A, float *_B, float *_C, int _M, int _N, int _K) {
39
40   // Launch kernel on every GPU
41   for (int i = 0; i < num_devices; i++) {
42     dim3 blockDim(1, 1, 1);
43     dim3 gridDim(Mend[i] - Mbegin[i], N, 1);
44
45     CUDA_CALL( cudaSetDevice(i) );
46     sgemm<<<gridDim, blockDim>>>(a_d[i], b_d[i], c_d[i], M, N, K);
47   }
48
49   // DO NOT REMOVE; NEEDED FOR TIME MEASURE
50   for (int i = 0; i < num_devices; i++) {
51     CUDA_CALL( cudaDeviceSynchronize() );
52   }
53 }
```

CUDA matmul - skeleton code

- OpenCL과 indexing 방식이 조금 차이 나는 것에 유의

```
19  __global__ void sgemm(float *A, float *B, float *C, int M, int N, int K) {
20      int i = blockDim.x * blockIdx.x + threadIdx.x;
21      int j = blockDim.y * blockIdx.y + threadIdx.y;
22      if (i >= M || j >= N)
23          return;
24
25      C[i * N + j] = 0;
26      for (int k = 0; k < K; ++k) {
27          C[i * N + j] += A[i * K + k] * B[k * N + j];
28      }
29  }
30
```

CUDA matmul - skeleton code

```
103  void mat_mul_final(float *A, float *B, float *C, int M, int N, int K) {
104      // Do any post-matmul cleanup work here.
105
106      // Download C matrix from GPUs
107      for (int i = 0; i < num_devices; i++) {
108          CUDA_CALL( cudaMemcpy(C + Mbegin[i] * N, c_d[i],
109                                (Mend[i] - Mbegin[i]) * N * sizeof(float),
110                                cudaMemcpyDeviceToHost) );
111      }
112
113      // DO NOT REMOVE; NEEDED FOR TIME MEASURE
114      for (int i = 0; i < num_devices; i++) {
115          CUDA_CALL( cudaDeviceSynchronize() );
116      }
117  }
```

Hint

- `mat_mul.cu` 에 이미 상당 부분이 구현되어 있음
  - multi-GPU 코드도 구현이 되어 있음
- 주어진 뼈대 코드를 잘 이해하고 시작할 것
- Single-GPU 에서 충분한 성능을 달성한 뒤 Multi-GPU 로 넘어가는 것을 추천
  - `num_devices` 변수 조절

CUDA matmul - 평가 기준

- 정확성 점수 50%
  - 무작위 N M K 에 대해 -v 옵션을 주어 값 검증을 통과해야 함
  - 총 10개의 test case 로 채점할 예정
  - N, M, K 각각은 2048 이하

- 성능 점수 50%
  - **4GPU,** 10 iteration, 8192 x 8192 x 8192 기준 **6000 GFLOPS** 달성 시 만점
  - 그 이하는 성능에 비례해 점수 부여
  - 답이 틀리면 0점 (-v 옵션)
  - ex. 3000 GFLOPS 달성 시 성능 점수의 절반 부여

CUDA matmul - 생각해 볼 것

- 뼈대 코드를 읽고 이해해보기

- OpenCL 최적화와 CUDA 최적화의 차이점
  - 어떤 것이 더 편한지?
  - OpenCL 의 장점?
  - CUDA 의 장점?

CUDA matmul - 주의사항

- `mat_mul.cu` 만 수정 가능
- 다양한 행렬 크기에 대해 답이 맞는지 체크를 할 것 (-v 옵션)
- 로그인 노드에서 성능 측정을 하지 말 것
  - 반드시 주어진 run.sh 혹은 srun 으로 계산 노드에서 실행

CUDA matmul - 주의사항

- GPU 에서만 연산을 수행할 것
  - CPU-GPU 로드 밸런싱은 본 과제의 범위가 아님

- CUDA 이외의 병렬화 방식 (Pthread, OpenMP, MPI, OpenCL 등) 사용 금지
  - CUDA 커널 내부에서 vector instruction 은 사용 가능

제출 방법

- 각자의 홈 디렉토리에 `~/submit/HW6` 디렉토리 생성
- 구현한 mat_mul.cu 파일을 디렉토리 내에 복사

- **디렉토리명, 파일명이 정확히 일치하도록**
- 즉, `ls` 명령의 출력이 아래 스크린샷과 같이 나와야 함

```
kjp4155@login:~$ ls ~/submit/HW6
mat_mul.cu
kjp4155@login:~$
```

- check-submit 유틸리티 활용

```
kjp4155@login:~$ check-submit
Username: kjp4155
-------------------------------------
HW1 prob1.txt   : X file not found
HW1 prob2.txt   : OK
HW1 main.cpp    : X empty file
-------------------------------------
HW2 mat_mul.cpp : OK
-------------------------------------
HW4 mat_mul.cpp : OK
-------------------------------------
HW5 mat_mul.cpp : OK
HW5 kernel.cl   : OK
-------------------------------------
HW6 mat_mul.cu  : OK
-------------------------------------
```

주의사항

- **제출 기한: 5월 18일 (수) 23:29**
  - 기한이 되면 프로그램이 자동으로 제출 파일을 복사해 감
  - 파일명, 디렉토리, 형식을 정확히 지켰는지 체크
  - 평가를 하는 과제

- **조교가 실수를 수정해줄 것이라 기대하지 않을 것**
  - 제출 디렉토리 확인
  - 제출 파일명 확인
  - 제출한 파일 내용 확인
  - 프로그램이 불필요한 출력을 하는 지 확인
  - 디버깅 코드 등 불필요한 코드를 모두 제거했는지 확인