SHPC FINAL PROJECT REPORT

2024 – 21081 방원준

1. How to run my program?

./run.sh -v -n 8192 -t 8 (4 개의 노드 모두 사용)

예상성능 : 64000 ~ 65000 tokens/sec

2. Batch size used

BATCH_SIZE = 256 (코드 상에 하드코딩 되어있음)

3. Which techniques did you use?

단계별로 설명해보도록 하겠다.

(STEP 1) : Port all the kernels into CUDA

먼저, skeleton 으로 주어진 모든 CPU kernel 들을 CUDA 로 포팅해 보았다. 원래 스켈레톤 코드를 -n 4 -t 8 option 을 주고 돌렸을 때 약 1.0 ~ 1.1 tokens/sec 의 값을 얻을 수 있었는데, CUDA porting 후에는 약 1.6 ~ 1.8 정도의 성능을 얻을 수 있었다. CUDA porting 시, 성능에 큰 영향을 미칠 것이라고 예상되는 linear 와 matmul kernel 은 hw5 의 것을 거의 그대로 이용하였다.

(STEP 2) : Pre-allocation of model weights and activations

아무래도 CUDA porting 이 성능이 좋지 않아서, 대부분의 병목이 CPU-GPU data transfer 로부터 나온다고 예상하였다. 그래서, model weight 와 activation 을 initialize phase 에 미리 모두 할당해 주었고, CPU-GPU data transfer 가 top-1-sampling 에서만 일어나도록 코드를 수정하였다.

성능 : ~12 tokens/sec

(STEP 3) : batching & multi-gpu & multi-node

스켈레톤 코드를 살펴보면, 한번에 prompt 를 하나씩만 다루는 구조로 되어 있다. 그래서, GPU 의 parallelization power 를 이용하기 위해서, 한번에 for loop 에서 batching 이 완료된 input 을 처리할 수 있도록 해 주었다. 뿐만 아니라, matmul, linear 등 몇 개의 kernel 들도 batched input 을 효율적으로 처리할 수 있도록 수정하였다.

또한, data parallelism 을 최대로 활용하기 위해서, 주어진 모든 GPU resource 인 16(4GPUs, 4 nodes)개의 GPU 를 사용할 수 있도록 코드를 수정하였다.

작업은 상당한 시간이 걸렸으나, 성능 향상이 극적으로 이루어졌다.

성능 : ~25000 tokens/sec

(STEP 4) : KV – Caching

이전 iteration 에서 생성되었던 attention layer 의 key, value 값은 저장해 두었다가, 다음 iteration(next token generation)에서 사용하는 것이 가능하다. 이는 KV-Caching 이라고 불리는 유명한 optimization 방법이다. 이를 프로젝트 모델에 적용해 보았다.

Attention layer 연산이 끝난 후에, 각 token 이 생성한 kv value 를 저장하는 tensor 를 새로 정의하였다.

KV-Caching 을 적용해 보니, 성능이 약 2.5 배 정도 향상된 것을 확인할 수 있었다. 이후, 실험적인 hyper parameter search(tile size, batch size 등)를 통해 최적의 성능이 나오도록 코드를 조정하였다.

성능 : ~ 65000 tokens/sec

4. Conclusion

결론적으로, 초기에 주어진 skeleton code 에 비해 약 65000 배 가량의 성능향상을 시키는데 성공하였다.

이번 프로젝트를 통해, nsight 를 이용한 병목 찾기, kernel 최적화, 효율적인 MPI 및 OpenMP 사용 등 병렬 컴퓨팅에 필요한 핵심적인 기술들을 다시 한 번 복습하고 익힐 수 있었다. 아주 유익한 경험이었다.