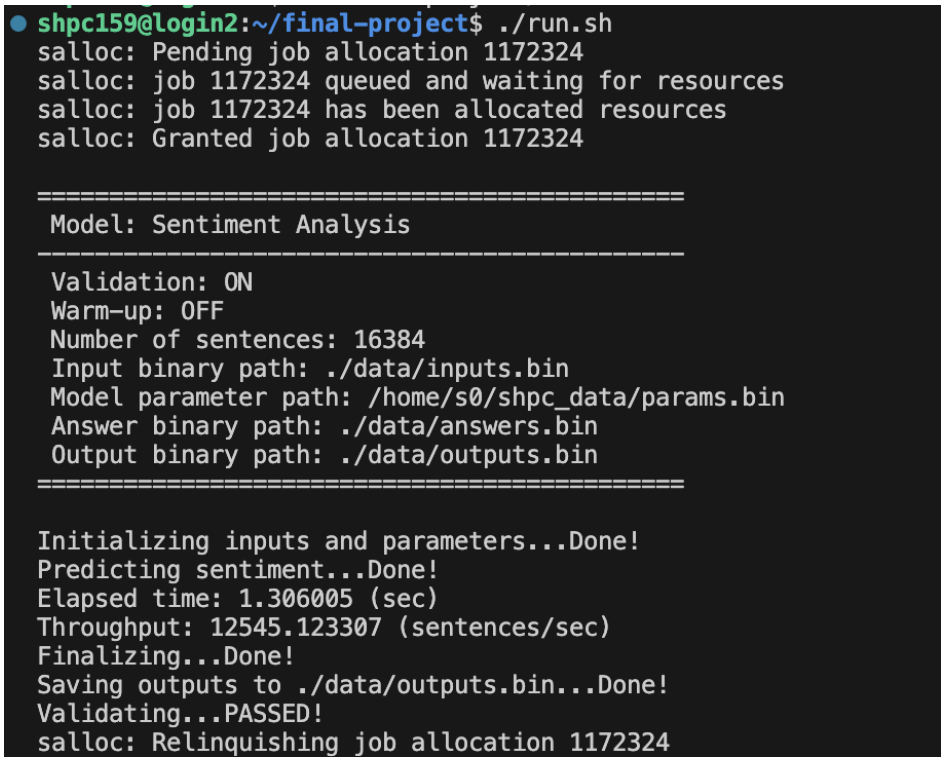
Final Project: Sentiment Analysis

홍경인
M1522.006700 확장형 고성능 컴퓨팅 (001)

December 20, 2024

1 최종 제출본 성능



최종 제출본 성능은 **12545.123307** (sentences/sec) 이다.

2 최적화 전략

2.1 Näive Conv1d Kernel

Skeleton code를 실행한 뒤 각 과정별로 소요시간을 확인한 결과, 1D convolution이 오래 걸리는 것을 확인하였다. 이를 메모리 계층이나 접근 방식 등을 생각하지 않고 우선 간단하게 구현하였다.

Number of sentences	Elapsed time	Throughput
2	0.243409 (sec)	8.216636 (sentences/sec)

2.2 Batch Input

Input을 batch 단위로 구분하여 넣어 줄 경우 커널의 parameter를 로드하는 과정을 줄일 수 있을 것으로 예상되어, batch input을 적용하였다.

Number of sentences	Elapsed time	Throughput
64	1.479244 (sec)	43.265345 (sentences/sec)

2.3 HW-aware Conv1d Kernel

Conv1d 커널을 엄밀하게 재설계하여 성능을 크게 향상시켰다. Input channel(`C`)의 값이 4096으로 크고 각 channel별 연산이 독립적이라는 점에 착안하여, 이를 기반으로 thread block을 분할하였다. 이후 kernel을 구현하는 과정에서 shared memory 크기가 너무 커 성능 저하를 야기하는 것을 확인하고, 그 크기를 조절하기 위해 input channel을 타일링하여 tile 하나에 해당하는 input과 weight을 그때그때 가져와 저장하는 것으로 수정하였다. 또, shared memory를 낭비하지 않으면서 이를 동적 할당하지 않고 compile time에 확정하기 위해 kernel을 4개로 split하고 각각에서 매크로를 활용하여 input과 weight 크기를 정적으로 결정하였다.
GPU data를 shared memory에 나누어 저장하고, 이를 register로 옮겨 와 계산하는 형태로 kernel을 구현하였다. 하나의 thread는 하나의 (`OC`, `os`)에 대하여 batch size만큼의 output을 채운다. 아울러 연산 구조 상 conv1d의 output이 늘 ReLU를 거치는 것을 확인하고, 이를 conv1d kernel 안으로 fusion하여 별도의 kernel launch overhead가 발생하지 않도록 하였다.
Batch size를 바꾸어 가며 테스트한 결과, `BATCH_SIZE = 64` 에서 성능이 제일 우수하면서 shared memory에 의한 kernel launching 실패가 발생하지 않는 것을 확인하여 해당 값을 활용하였다.

Number of sentences	Elapsed time	Throughput
256	1.206005 (sec)	212.271073 (sentences/sec)

2.4 HW-aware Linear Kernel

Hw6에서 활용하였던 GEMM kernel을 활용하였다. B에 해당하는 matrix가 transpose되어 기존 kernel을 적용하기 어려운 상황이었는데, 각 register가 shared memory를 읽는 패턴을 간단히 수정하는 것만으로도 해당 문제를 해결할 수 있어 기존 GEMM kernel을 적용할 수 있었다.

Number of sentences	Elapsed time	Throughput
256	0.801598 (sec)	319.362117 (sentences/sec)

2.5 End-to-End CUDA Optimization

코드를 리팩토링하여 코드 중간에 tensor를 쓰지 않고, tensor에 있는 데이터를 GPU로 모두 전송한 뒤 GPU에서 모든 연산을 수행하게 하였다. Batch size가 증가하면서, 다른 연산 대비 conv1d이 전체 연산에서 차지하는 비중이 점차 증가하였다. 프로파일링 결과 h2d에 비해 conv1d에 드는 시간이 더 컸었는데, 이를 고려하여 h2d와 커널 연산을 병렬화하는 것보단 h2d를 초기에 빠르게 수행한 뒤 커널 최적화를 적절히 수행하는 것이 더 유리할 것으로 판단하였다. 이에 pageable memory에서 pinned memory를 별도로 할당하지 않은 채 바로 GPU로 `cudaMemcpy`를 시행하였다.

또한 conv1d 구동 방식을 바꾸어, 네 종류의 1D convolution이 asynchronous하게 이루어지도록 kernel에 각각 stream을 할당하여 처리하였다. 동시에 batch를 네 개씩 별도의 stream에 할당하여 병렬 처리되도록 하였다. 또한 `BATCH_SIZE = 256`으로 두되, `MINI_BATCH_SIZE = 64`로 두어 conv1d kernel 외에는 `BATCH_SIZE = 256`씩, conv1d kernel은 `MINI_BATCH_SIZE = 64`씩 처리되도록 하였다. 이외의 kernel은 성능이 충분히 좋아 kernel launching overhead를 줄이는 것이 시급했는데, 이와 같은 운용은 kernel의 launching 빈도를 줄여 overhead를 효과적으로 줄였다.

Number of sentences	Elapsed time	Throughput
8192	8.073624 (sec)	1014.662078 (sentences/sec)

2.6 Multi-GPU Implementation

기존 단일 GPU 코드를 4대의 GPU에서 병렬적으로 실행되도록 구현했다. 동일한 모델을 input만 분리하여 구현하는 것만으로도 성능이 충분하면서 구조가 간결하게 병렬성을 구현할 수 있었다.

Number of sentences	Elapsed time	Throughput
16384	4.747485 (sec)	3451.090369 (sentences/sec)

2.7 Multi-Node Multi-GPU Implementation

기존 단일 노드 코드를 4개의 노드에서 데이터를 송수신하며 병렬적으로 수행할 수 있도록 구현하였다. 이 과정에서 input, weight, 그리고 bias 전송에 드는 비용이 증가하게 된다. 따라서 이전에 채택했던 방식처럼 데이터를 한꺼번에 미리 전송해 두는 전략이 성능 향상에 기여할 수 있다.

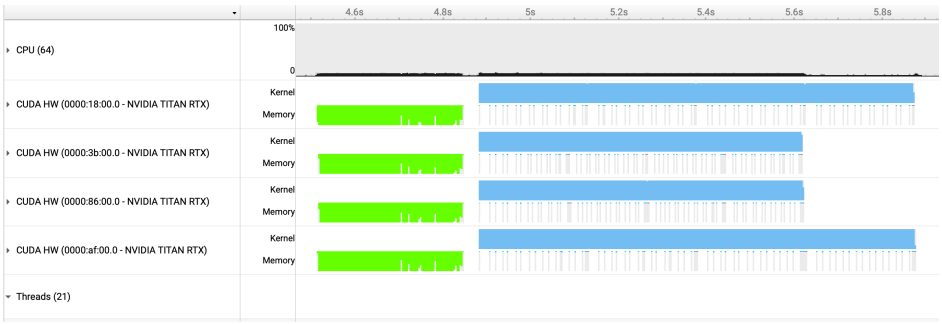
Nsight Systems를 활용한 프로파일링 결과, 단일 GPU에서 batch를 4개로 나누어 별도의 stream에 할당했음에도 불구하고 해당 stream들이 synchronous하게 실행되었다. 이는 Conv1D kernel이 과도한 shared memory를 사용하여 여러 thread block이 동시에 실행되기 어렵기 때문으로 분석된다. 실험 결과, 단일 GPU에서 multi-streaming을 제거하는 것이 성능 개선에 더 효과적이었으며, streaming은 Conv1D의 4개 kernel 실행에만 제한적으로 적용하는 것이 가장 효율적인 방법으로 나타났다.

Number of sentences	Elapsed time	Throughput
16384	1.306005 (sec)	12545.123307 (sentences/sec)

3 개선 여지가 있는 부분

3.1 Tensor Core 활용

Tensor Core를 활용하여 GEMM의 성능을 크게 개선시킬 수 있다. 이는 주요 병목이었던 conv1d 성능 향상에 도움을 줄 수 있는데, 특히 im2col 등을 활용하여 conv1d를 GEMM으로 환원시킬 경우 성능을 직접적으로 개선할 수 있다.



3.2 일부 GPU에서의 성능 불균형 해결

현재 MPI의 host node(NODE 0)에서 일부 GPU의 성능 저하가 관찰되고 있다. 이 문제는 single-node multi-GPU 환경에서는 나타나지 않았던 현상으로, 여러 node를 사용하는 분산 계산 환경에서 특정 node의 GPU 두 대의 성능이 저하되는 현상이다. 이는 MPI를 구동하는 과정에서 해당 GPU가 추가적인 작업에 일정 부분 참여하면서 GPU의 capacity가 줄어들고, 이에 따라 속도가 감소하는 것으로 예측된다. 대부분의 GPU는 유사한 성능을 보이지만, 이 특정 노드에서의 성능 저하가 전체 실행 시간을 지연시키고 있다. 이러한 문제를 해결하면 전체 성능이 15000-16000 sentences/sec 정도로 개선될 것으로 기대된다.