멀티코어 기말 프로젝트 보고서

2014-19768 허성연

A.

A 는 CPU 만을 이용해서 병렬화를 하는 과제였다. A 를 병렬화하기 위해 다음과 같은 방법을 이용했다.

1. **Pthread 를 이용한 병렬화**
   뼈대로 받은 코드에서는 1개의 쓰레드만을 이용하여 작업을 수행했다. 천둥의 CPU 는 멀티코어이므로 이를 32개의 쓰레드로 나누어 작업을 병렬화하여 수행했다. N 개의 이미지가 들어왔다면, 각 쓰레드에는 N / 32 개의 이미지를 처리하도록 병렬화했다.

2. **Tiling**
   뼈대로 받은 코드에서는 단순히 계산만 하는 코드였다. 메모리를 최대한 활용하기 위해 타일링을 수행했다.타일링은 C 와 K 에 대해서 수행했고, 사이즈는 실험을 통해 적절한 크기를 정했다.

3. **Filter 차원 바꾸기**
   Conv2d 의 경우 filter 의 차원이 r, s, c, k 로 되어 있었는데, 이를 r, s, k, c 를 이용하도록 바꾸어, 연속된 메모리를 최대한 이용할 수 있도록 바꿨다.

4. **Vector instruction / loop unrolling**
   타일링이 된 코드에 avx 옵션을 통해 CPU 벡터 연산을 사용하도록 explicit 하게 코딩하였다. 또한 루프 언롤링을 통해 벡터 연산이 효율적으로 처리되도록 했다.

5. **루프 도는 횟수 줄이기**
   conv2d_transpose 의 경우 루프를 oh, ow 를 기준으로 돌게 되는데, 2 로 나누어지지 않는 경우가 많으므로 헛도는 횟수가 많다. 따라서 루프를 ih, iw 기준으로 돌리게 되면 루프 내부의 내용이 수행되는 횟수 자체가 적어지므로 최적화할 수 있었다.

위와 같은 코드를 작성해서 테스트를 수행했다. 128개 이미지를 이용해 테스트를 수행했으며, 그 결과는 다음과 같다.

```
Reading inputs... done! (0.146164 s)
Initializing pix2pix... done! (0.000639 s)
Calculating pix2pix... done! (25.707377 s)
Writing result... done! (0.035381 s)
Elapsed time : 25.707377 sec
Throughput : 4.979116 img / sec
```

B.

B 는 CPU 1개와 GPU 1개를 이용해서 병렬화를 하는 과제였다. B 를 병렬화하기 위해 다음과 같은 방법을 이용했다. 크게는 GPU 커널 코드 최적화와 CPU 에서의 연산 스케줄링으로 나눌 수 있다.

먼저, GPU 커널 코드 최적화는 다음과 같은 방식을 이용했다.

1. **Shared memory 이용**
   Input 의 경우 여러 K 에 대해 재활용되기 때문에 이를 GPU 의 shared memory 에 저장해놓고 사용하는 것이 더 유리하다. 따라서 shared memory 에 input 의 일부를 저장해두고 재활용하여 최적화를 했다.

2. **워크 그룹 설정**
   1번에서 input 을 최대한 활용하기 위해 적절한 work-group 을 설정하여 메모리를 최대한 이용할 수 있도록 했다. Work-group 의 크기는 실험을 통해 적절한 값을 도출해냈다.

3. **루프 도는 횟수 줄이기**
   A 와 마찬가지로, conv2d_transpose 의 경우 필요없는 루프 횟수 자체를 줄여서 성능을 향상시킬 수 있었다.

그리고 스케줄링 관련 해서는 다음과 같은 방식을 이용했다.

1. **더블 버퍼링**
   Data queue 와 Compute queue 를 이용해서 GPU 에서 연산이 이루어지는 중에도 GPU 와의 데이터 전송이 이루어지게 했다. 이를 통해 GPU 의 연산능력을 최대한 활용할 수 있도록 했다.

2. **Filter, Bias 전송 최적화**
   입력으로 주어지는 weight 값 중, filter 와 bias 의 경우 이미지와 상관없이 같은 값이다. 따라서 최초 한 번만 전송을 한 뒤, 이를 재활용한다면 PCI-E 전송에 소모되는 시간을 줄일 수 있다. 따라서 처음 한 번만 filter 와 bias 를 전송한 뒤, 다음에는 이미 전송된 데이터를 활용하도록 했다.

3. **동시에 여러 이미지 진행**
   한번에 하나의 이미지만 연산을 하게 되면, conv 계산을 한 뒤 추가적인 다른 계산 딜레이 때문에 아무 작업도 하지 않고 기다리는 시간이 많다. 따라서 Pthread 를 이용해 동시에 여러 이미지를 수행하게 되면, 한 이미지가 메모리를 전송하고 있는 시간에 다른 연산을 수행할 수 있도록 했다.

위와 같은 코드를 작성해서 테스트를 수행했다. 256개 이미지를 이용해 테스트를 수행했으며, 그 결과는 다음과 같다.

Initializing pix2pix...Detected OpenCL device: GeForce GTX 1080
done! (1.954655 s)
Calculating pix2pix... done! (18.898881 s)
Writing result... done! (0.066425 s)
Elapsed time : 18.898881 sec
Throughput : **13.545776 img / sec**

C.

C 는 CPU 1개와 GPU 4개를 이용해서 병렬화를 하는 과제였다. 먼저, 커널 코드와 pix2pix.cpp 의 경우 B 에서 최적화한 코드를 그대로 가지고 왔다.
C 에서는 GPU 를 4개까지 사용할 수 있으므로 1개의 GPU 에 나눠지는 연산량을 4개의 GPU 에 나눴다. 4개의 GPU 를 쓰므로, filter 와 bias 를 4번 전달하는 오버헤드가 생겼지만, 연산성능 자체가 늘었기 때문에 충분히 커버가 가능했다.

위와 같은 코드를 작성해서 테스트를 수행했다. 512개 이미지를 이용해 테스트를 수행했으며, 그 결과는 다음과 같다.

Calculating pix2pix... done! (25.300094 s)
Writing result... done! (0.163040 s)
Elapsed time : 25.300094 sec
Throughput : **20.237079 img / sec**

D.

D 는 4개의 노드와 GPU 16 개를 이용하는 과제였다. 먼저, 커널 코드는 B 에서 최적화한 코드를 그대로 가져왔다. 그 다음, 여러 노드를 사용하기 위해 다음과 같은 방식을 사용했다.

1. **MPI 를 이용해서 노드 간 전송**
   여러 노드에 전송을 하기 위해, MPI 를 이용해 통신을 했다. Rank 가 0 인 노드에서 1, 2, 3 노드로 나눠진 input, filter, bias 를 전송 했다. 또한, 연산이 끝난 다음의 output 또한 MPI 를 이용하여 rank 0 인 노드에 모았다.

2. **병렬적으로 이미지 처리**
   각 노드를 최대한으로 활용하기 위해, 각 노드에 균일하게 이미지를 배분한 다음 C 에서 처리했던 것과 같이 연산을 수행했다. 따라서 C 에 비해 동시에 처리할 수 있는 이미지가

더 많아졌다.

위와 같은 코드를 작성해서 테스트를 수행했다. 1024개 이미지를 이용해 테스트를 수행했으며, 그 결과는 다음과 같다.

Calculating pix2pix... done! (15.571371 s)
Writing result... done! (0.260356 s)
Elapsed time : 15.571371 sec
Throughput : **65.761712 img / sec**