과제 #2 평가

4190.414A 멀티코어 컴퓨팅(001)

September 27, 2022

1 문제 1: pthread를 이용한 병렬화 (80점)

pthread를 이용한 가장 쉬운 병렬화 방법은 outermost loop를 나누는 것이다. 행렬 A와 행렬 C를 세로 방향으로 thread 개수 T 만큼 나누고, 각 thread가 $A_{\frac{M}{T}\times K} \times B_{K\times N} = C_{\frac{M}{T}\times N}$를 계산하게 된다. 코드는 다음과 같다.

```
(...)
static void* mat_mul_thread(void *data) {
  int tid = (long)data;
  int is = M / num_threads * tid + min(tid, M % num_threads);
  int ie = M / num_threads * (tid + 1) + min(tid + 1, M % num_threads);
  for (int i = is; i < ie; ++i) {
    for (int k = 0; k < K; ++k) {
      for (int j = 0; j < N; ++j) {
        C[i * N + j] += A[i * K + k] * B[k * N + j];
      }
    }
  }
}

void mat_mul(float *_A, float *_B, float *_C, int _M, int _N, int _K, int _num_threads) {
  A = _A, B = _B, C = _C; M = _M, N = _N, K = _K; num_threads = _num_threads;
  pthread_t threads[num_threads];
  for (long i = 0; i < num_threads; ++i) {
    pthread_create(&threads[i], NULL, mat_mul_thread, (void*)i);
  }
  for (long i = 0; i < num_threads; ++i) {
    pthread_join(threads[i], NULL);
  }
}
```

위 코드가 `mat_mul_ref`의 코드이며 성능은 약 120 GFLOPS이다. 과제 1에서 배운 vector instruction도 활용해 볼 수 있다.

```
// align 및 예외 처리 코드(예, N이 8의 배수가 아닌 경우)는 생략
for (int i = is; i < ie; ++i) {
  for (int k = 0; k < K; ++k) {
        __m256 a = _mm256_broadcast_ss(&A[i * K + k]);
```

```
        for (int j = 0; j < N; j += 8) {
          __m256 b = _mm256_load_ps(&B[k * N + j]);
          __m256 c = _mm256_load_ps(&C[i * N + j]);
          _mm256_store_ps(&C[i * N + j], _mm256_fmadd_ps(a, b, c));
    }
  }
}
```

위와 같이 vector instruction 을 사용할 경우 150GFLOPS 내외의 성능을 얻을 수 있다.

이론 성능과 큰 차이가 나는 이유 중 하나는 메모리 load/store가 연산 속도를 따라가지 못하기 때문이다. 아래와 같이 코드를 수정해보자.

```
const int ISTEP = 2;
for (int i = is; i + ISTEP <= ie; i += ISTEP) {
  for (int k = 0; k < K; ++k) {
    __m256 a0 = _mm256_broadcast_ss(&A[(i + 0) * K + k]);
    __m256 a1 = _mm256_broadcast_ss(&A[(i + 1) * K + k]);
    for (int j = 0; j < N; j += 8) {
      __m256 b = _mm256_load_ps(&B[k * N + j]);
      __m256 c0 = _mm256_load_ps(&C[(i + 0) * N + j]);
      __m256 c1 = _mm256_load_ps(&C[(i + 1) * N + j]);
      _mm256_store_ps(&C[(i + 0) * N + j], _mm256_fmadd_ps(a0, b, c0));
      _mm256_store_ps(&C[(i + 1) * N + j], _mm256_fmadd_ps(a1, b, c1));
    }
  }
}
(... 남은 i 처리)
```

Outermost loop의 한 iteration에 두 개의 i 값에 대한 연산을 하도록 하였다. (이러한 기법은 blocking, tiling 등의 이름으로 불린다.) A, C 행렬의 값이 레지스터를 차지하는 공간이 2배가 된 대신 (a0, a1, c0, c1) B 행렬을 메인 메모리에서 읽어오는 양은 절반이 된다. 위 코드는 register 수준에서 blocking을 한 것으로, 성능 점수에서 만점을 받을 수 있다. (약 220 GFLOPS) L1, L2, L3 cache 크기까지 고려하여 여러 수준에서 이러한 최적화를 적용하면 훨씬 더 좋은 성능을 달성할 수 있다. 이론 성능과 차이가 나는 또 다른 이유로는 dependence나 loop overhead로 인해 functional unit이 100% 활용되지 못하는 것인데, 이는 unrolling을 적용하여 어느 정도 해결할 수 있다.

Thread 개수에 따른 성능을 측정해보면, 보통 20개까지는 선형적으로 증가하다가 21개에서 조금 감소하는 경향을 보인다. 이는 실습 서버에 20코어의 CPU가 2개 장착되어 있어서 21번째 thread가 다른 코어에 할당되는 경우 다른 NUMA node로의 메모리 접근이 일어나서 그렇다. 40개와 80개 비교에서는, 한 개의 physical core에 할당된 두 thread는 functional unit을 공유하기 때문에 functional unit이 100%에 가깝게 활용되고 있다면 성능 증가가 없거나 미미할 것이다. 그러나 우리가 작성한 프로그램은 그 수준으로 최적화가 되어있지 않기 때문에 2배까지는 아니더라도 어느 정도의 성능 향상을 얻을 수 있다.

보고서 (40%), 32점 과제에서 언급한 내용들을 적당히 포함하면 기본 점수 20점. 여기서 조교의 판단에 따라 가점 또는 감점. 실험 결과에 대한 원인 분석 및 개선방향을 잘 설명하면 좋은 점수를 받을 수 있음.

정확성 (40%), 32점 40개 이하의 스레드, 4096 이하의 M, N, K에 대해서 32개의 랜덤한 테스트 케이스를 실행, -v 옵션을 통한 validation을 통과하면 개당 1점.

성능 (20%), 16점 실습 서버에서 40 스레드, $M = N = K = 4096$, $n = 50$ 옵션을 주고 실행했을 때, 150 GFLOPS를 넘으면 만점. 그 이하는 비율에 따라 점수를 부여한다. (e.g., 135 GFLOPS인 경우 성능 점수의 90%를 부여) 답이 틀린 경우 0점.

사용한 테스트 케이스는 다음과 같다.

```
./main -v -t 26 831 538 2304
./main -v -t 9 3305 1864 3494
./main -v -t 38 618 3102 1695
./main -v -t 30 1876 3453 3590
./main -v -t 16 1228 2266 1552
./main -v -t 2 3347 171 688
./main -v -t 39 3583 962 765
./main -v -t 30 2962 373 1957
./main -v -t 9 3646 2740 3053
./main -v -t 26 1949 3317 3868
./main -v -t 33 2882 2363 2223
./main -v -t 3 3052 778 1237
./main -v -t 39 2471 2819 3654
./main -v -t 7 3863 1335 2284
./main -v -t 38 2145 2831 2785
./main -v -t 18 2148 367 241
./main -v -t 8 129 2519 636
./main -v -t 18 1599 4068 2544
./main -v -t 15 3359 336 2346
./main -v -t 38 1029 428 108
./main -v -t 5 1434 1488 3338
./main -v -t 4 2530 3439 4002
./main -v -t 38 3699 2293 1806
./main -v -t 38 3353 1746 1840
./main -v -t 13 1287 2934 3213
./main -v -t 10 3644 3637 2650
./main -v -t 31 3891 378 2919
./main -v -t 37 633 3794 3565
./main -v -t 25 1731 3839 3236
./main -v -t 20 3893 294 1250
./main -v -t 35 1392 312 3249
./main -v -t 8 754 1638 3495
```

2 문제 2: 캐시의 크기 추정(20점)

M, N, K를 변화시키면서 성능 변화가 큰 폭으로 일어나는 지점을 찾으면 된다. Capacity miss 외에 cold miss, conflict miss, associativity 등에도 영향을 받기 때문에 M, N, K의 간격을 **좁게**하여 실험해야 경향성을 파악하기 쉽다. (예, 2의 승수 단위로 실험하면 찾기 힘듬) 아니면 위에서 언급한 blocking을 적용한 후 block 크기를 변화시켜보면서 추정해 보아도 좋다.

보고서 (100%), 20점 최소한의 시도를 했다면 기본 점수 10점 부여. 조교의 판단에 따라 가점 또는 감점. 실험을 적절히 설계하고 실제 캐시 크기와 추정값이 다른 원인을 잘 분석할수록 좋은

점수를 받을 수 있음.