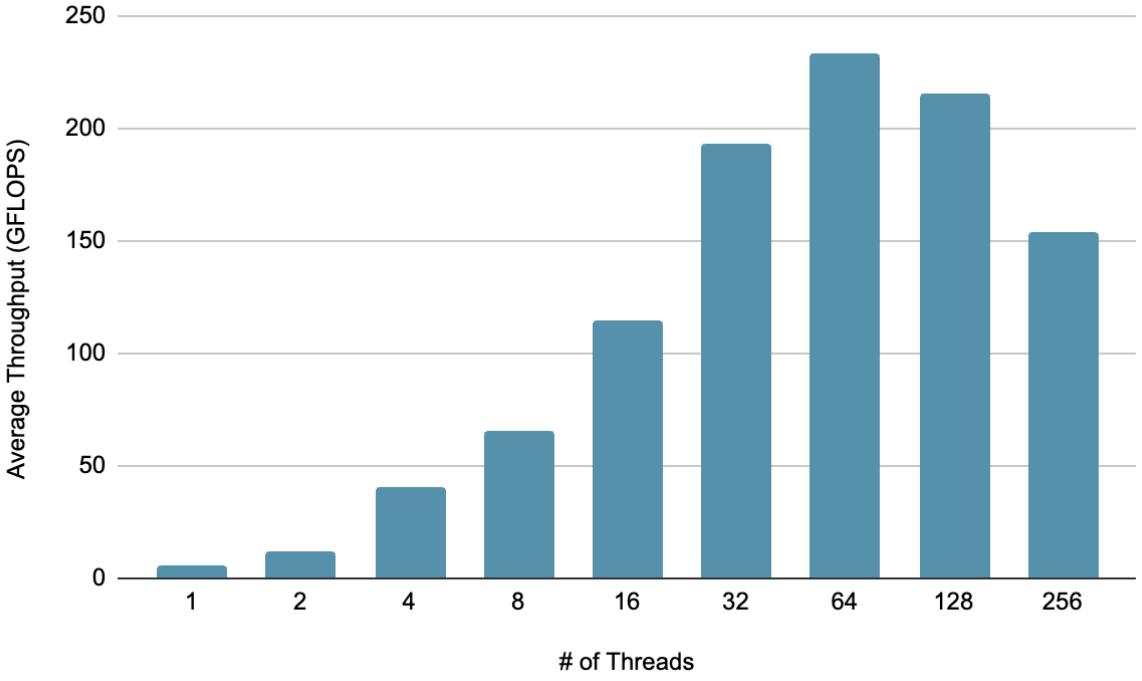
과제 #3

2022-29677 한주희

1 Matrix Multiplication using OpenMp

- 행렬곱에서는 tiling 방식을 사용하고, 중첩된 for문의 순서를 `ikj`로 변경하여 메모리 접근을 연속적으로 처리하도록 최적화하였다. 또한, OpenMP를 통해 loop iteration을 병렬 처리했다. `#pragma omp parallel for num_threads(num_threads) schedule(dynamic)`을 활용하여 `num_threads`개의 스레드를 생성하고, loop iteration을 병렬로 처리하였다.
- OpenMP는 `#pragma` directive를 통해 컴파일 시점에 스레드 생성 함수를 추가할 수 있다.
- 다음은 스레드 수를 1, 2, 4, 8, 16, 32, 64, 128, 256개로 늘리며 행렬곱의 성능을 측정한 결과이다. 반복 횟수는 3회로 설정하여 평균 성능을 계산하였다. 스레드가 64개일 때 성능이 가장 높았으며, 이후 점차 성능이 감소하여 256개에서는 성능이 급격히 떨어졌다. 이는 스레드가 많아질수록 각 스레드가 처리하는 작업량은 줄어들고, 스레드의 시작과 종료에 따른 오버헤드와 스레드 간 자원 공유에 필요한 오버헤드가 증가하기 때문이다.

Threads	1	2	4	8	16	32	64	128	256
Avg Throughput (GFLOPS)	6.10	12.39	40.25	65.31	115.0	193.3	233.5	215.7	154.0



- 다음은 스케줄링 방식을 static, dynamic, guided로 나누어 실험한 결과이다. 실험 결과, dynamic이 가장 높은 성능을 보이고 static 방식과 guided 방식이 유사한 성능을 보였다.
    - **static**: iteration은 `chunk_size` 크기의 chunk로 나뉘며, 스레드 번호 순서대로 round-robin 방식으로 스레드에 할당된다.
    - **dynamic**: 각 스레드는 할당된 chunk를 처리한 후, 남은 chunk를 요청하여 작업을 반복한다. 이를 chunk가 남아있지 않을 때까지 반복한다.
    - **guided**: dynamic과 유사하지만, 진행에 따라 chunk의 크기가 점차 감소한다. `chunk_size`를 기준으로 남은 작업량을 스레드 수로 나눈 값에 비례하여 chunk 크기가 감소한다.

Clause	static	dynamic	guided
Avg Throughput (GFLOPS)	236.6	242.5	230.1

2 Estimating Cache Size

<표 1>은 cache size를 추정하기 위해 행렬 A, B, C 크기를 변경해가며 실험한 결과이다.

스레드 수는 32로 설정하였다.

Matrix Size (M = N = K)	32	64	128	256	512	1024	2046	4096	8192	16384
Avg Time (sec)	0.000127	0.000135	0.000527	0.001693	0.0039	0.018	0.22	0.68	5.66	ABORT
Avg Throughput (GFLOPS)	0.51	3.87	7.95	19.82	67.75	118.76	76.63	199.82	193.98	ABORT

<표 1>: 행렬 크기별 실행 시간 및 처리량(GFLOPS) 측정 결과

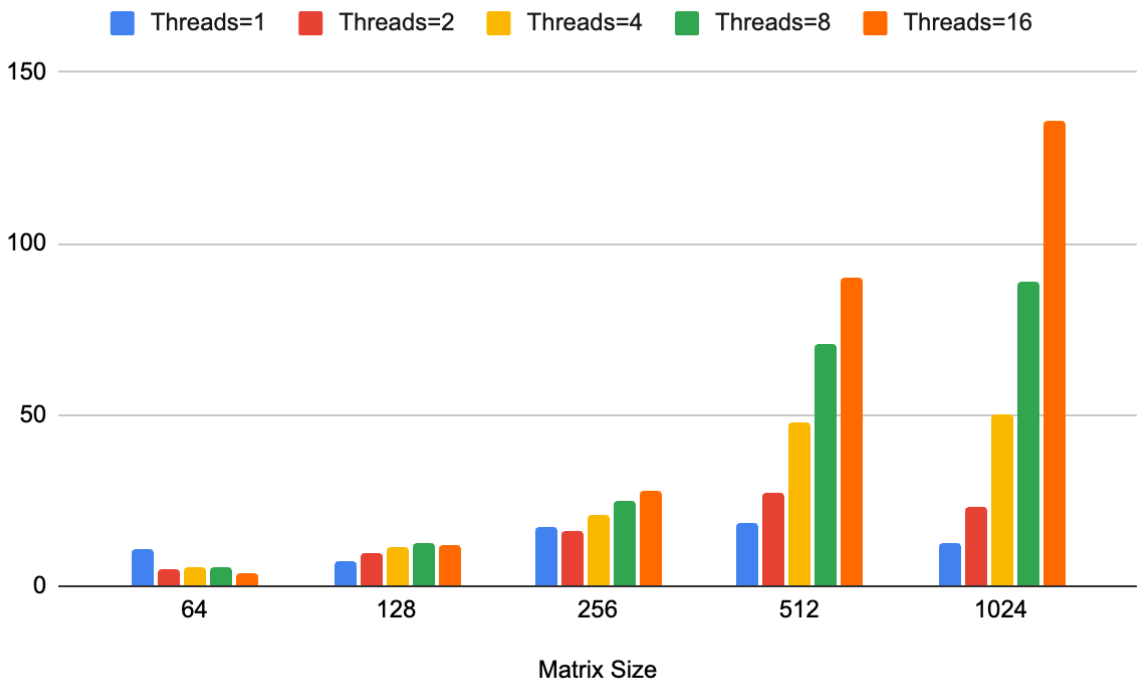
(캐시 용량 추정용)

<표 2> 는 private/shared cache 여부를 확인하기 위해, 행렬 크기와 스레드 수를 변경하며 GFLOPS를 측정한 실험한 결과이다.

Matrix Size (M = N = K)	Threads=1	Threads=2	Threads=4	Threads=8	Threads=16
64	10.9	5.2	5.82	5.5	3.73
128	7.43	9.66	11.2	12.68	11.91
256	17.06	16.07	21.03	24.89	27.76
512	18.2	27.06	47.72	70.65	90.11
1024	12.79	23.41	50.04	88.87	135.83

<표 2>: 행렬 크기 및 스레드 수에 따른 처리량(GFLOPS) 측정 결과

(private/shared cache 확인용)



- L1 cache size는 약 64KB이며 private cache이다.
    - <표 1> 실험 결과에 의하면, 32x32에서 64x64로 전환 시 실행 시간은 미세하게 증가하였으나, 성능은 크게 향상되었다. 이는 약 64x64(약 16KB) 크기의 행렬이 L1 캐시에 잘 맞아 효율적으로 접근되고 있음을 나타낸다. 또한, 128x128 크기에서의 큰 실행 시간 증가는 이 행렬이 L1 캐시에 모두 적재되지 못하면서, L1 캐시 용량의 한계를 보여준다. 따라서, L1 캐시 크기는 약 64KB로 추정된다.
    - <표 2> 실험 결과에 의하면, 64x64, 128x128 행렬 크기에서 스레드 수가 늘어나도 성능의 변화가 적고, 처리 시간이 비교적 짧다. 이는 L1 캐시가 각 코어에 private cache로 데이터가 잘 저장되기 때문으로 볼 수 있다.
- L2 cache size는 약 256KB이며 private cache이다.
    - <표 1> 실험 결과에 의하면, 128x128 및 256x256에서 실행 시간이 크게 증가하였다. 이는 L1 캐시를 초과하여 L2 캐시에 저장하게 되었음을 나타낸다. 128x128(약 64KB)와 256x256(약 256KB) 행렬 크기는 L2 캐시 용량의 한계를 보여준다. 따라서, L2 크기는 약 256 KB로 추정된다.

    - <표 2> 실험 결과에 따르면, 256x256 크기의 행렬에서 스레드 수가 증가하더라도 처리 속도가 비슷하기 때문에, L2 캐시가 각 코어에 private cache로 사용됨을 알 수 있다.
- L3 cache size는 약 8MB이며 shared cache이다.
    - <표 1> 실험 결과에 의하면, 512x512에서 1024x1024로 넘어가는 동안 실행 시간이 크게 증가하며 이는 L2 캐시를 초과하고 L3 캐시에 의존하게 됨을 나타낸다. 또한, 1024x1024(약 8MB)에서 2046x2046로 넘어가는 동안 다시 한번 실행 시간이 크게 증가하는데, 이는 L3 캐시에서 Main memory로 넘어가는 지점임을 알 수 있다. 따라서, L3 캐시의 크기는 약 8MB로 추정된다.
    - <표 2> 실험 결과에 의하면, 512x512 이상의 행렬 크기에서 스레드 수가 늘어남에 따라 성능이 상승한다. L3 캐시가 여러 코어 간에 공유되고 있는 것으로 보이며, 이는 각 스레드가 shared cache에서 데이터를 불러오면서 캐시 효율이 증가하기 때문으로 보인다.