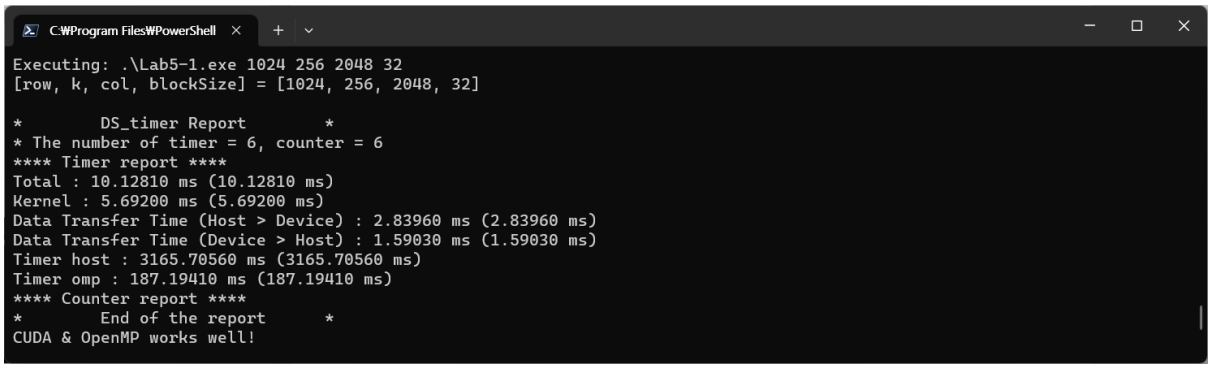
목차

1. 실행 결과

Lab 5-1. Matrix Multiplication



cudaMalloc, cudaMemcpy, cudaFree등의 함수를 이용하여 디바이스 측에 CPU 메모리에 있는 데이터를 적절히 복사하고, __global__ 예약어와 함께 선언한 matrixMul 함수를 이용하여 GPU 측에서 행렬곱 연산을 구현하였고, 2차원 그리드, 2차원 블록으로 구성된 스레드 레이아웃을 이용하여, 블록 사이즈보다 큰 행렬을 GPU를 이용하여 계산할 수 있게 구성하였다. (각 스레드별로 행렬의 한 요소를 계산함, "blockDim.[x,y] * blockIdx.[x,y] + threadIdx.[x,y]" 형태로 각 행과 열의 색인을 계산)

row	k	col	matrixSi	blockSi	Total	Kerne	Data Transfer Time (Host > Device)	Data Transfer Time (Device > Host)	Host	Host omp
512	2048	4096	2097152	32	54.4668	43.7469	9.2453	1.4679	204475.5433	7377.5558
256	4096	4096	1048576	32	56.6926	40.4719	15.4185	0.794	190956.3665	7585.3674
256	2048	4096	1048576	32	29.4758	20.4558	8.0043	1.0074	103356.1192	3605.431
512	1024	4096	2097152	32	26.7728	19.8365	5.471	1.4592	87184.5645	799.41
256	1024	4096	1048576	32	15.8977	10.3527	4.6788	0.8626	43698.6362	433.5801
512	512	4096	2097152	32	15.6963	10.1158	3.7492	1.8251	37274.1672	313.3803
256	512	4096	1048576	32	8.7562	5.3939	2.5984	0.7588	17974.8001	187.8576
512	256	4096	2097152	32	9.5286	5.2013	2.8293	1.4929	17355.8842	182.48
512	2048	2048	1048576	32	26.486	20.3828	5.2848	0.8138	14771.93	837.6239
512	4096	1024	524288	32	26.7503	20.4686	5.7055	0.5702	14303.7496	937.4654
256	4096	2048	524288	32	28.8775	20.4639	7.9167	0.4881	13588.2409	3443.7496
256	256	4096	1048576	32	5.5639	2.8118	1.9301	0.8188	8732.731	111.7868
256	2048	2048	524288	32	15.2991	10.3746	4.4097	0.5076	7093.2367	442.1235
256	4096	1024	262144	32	15.249	10.2707	4.7114	0.2625	7067.4904	498.5021
1024	1024	1024	1048576	32	14.4479	11.0118	2.6394	0.7925	6419.663	312.1556
512	2048	1024	524288	32	13.7151	10.3806	2.8976	0.4317	6404.4726	337.8771
512	1024	2048	1048576	32	14.042	10.3787	2.9116	0.7462	6291.5222	303.2946
256	2048	1024	262144	32	8.45	5.3678	2.6529	0.4243	3247.3013	188.7902
512	512	2048	1048576	32	8.1066	5.4113	1.8966	0.7949	3195.8878	170.7342
512	1024	1024	524288	32	7.7596	5.3754	1.9141	0.4662	3166.7713	169.2211
256	1024	2048	524288	32	8.579	5.4777	2.6243	0.4736	3158.4721	154.6305
512	4096	512	262144	32	14.4783	10.324	3.8398	0.3107	3056.7434	140.5992
256	512	2048	524288	32	4.9311	2.8616	1.5602	0.5063	1612.2967	91.3724
256	1024	1024	262144	32	4.4457	2.7593	1.4242	0.2588	1584.8739	91.5315
512	256	2048	1048576	32	5.0123	2.8668	1.3833	0.7589	1577.5381	97.2755
512	512	1024	524288	32	4.5016	2.8509	1.1764	0.4706	1570.5172	88.9437
256	4096	512	131072	32	8.5458	5.247	3.1069	0.1871	1517.4343	86.2321
512	2048	512	262144	32	7.784	5.2695	2.1448	0.3663	1512.3889	82.3057

[행렬 크기를 변경해가며 테스트한 프로그램 실행 결과 (RTX 3080, Ryzen 9 3900X)]

row, k, col: 행렬의 크기

matrixSize: row * col

Total: Kernel + Data Transfer Time + Host 시간

Kernel: 커널 부분 (행렬 곱) 부분만의 실행 시간

Data Transfer Time: 데이터 이동간 걸리는 시간

Host: 호스트의 행렬 곱 소요 시간

Host omp: OpenMP를 이용하여 계산한 경우 소요 시간

*. 시간 단위는 밀리 초를 사용

거의 모든 경우에서 데이터 전송 시간을 고려해도, GPU를 이용한 계산이 압도적으로 빠른 것을 확인할 수 있었다. 데이터가 많아질수록 이 차이는 더 극명하게 나타나서, CPU 직렬로 구성한 코드보다 최대 3754배, CPU 병렬로 구성한 코드보다는 최대 135배 더 빠른 속도를 보였다.

2. 소감

이번 프로젝트를 통해 GPU의 병렬 처리 능력이 얼마나 강력한지를 직접 체험해볼 수 있었다. 특히 이번 과제에서는 지난 과제와 달리 CPU와의 성능 차이가 극명하게 나타나서 놀라웠다. 또한 조금 더 복잡한 데이터 형태에 따른 스레드 레이아웃과 로직을 구상하고 적용해보면서, GPU를 이용한 병렬 프로그래밍이 어떤 것인지에 대해 더 자세히 이해할 수 있었다.