REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

Class Name : Embedded_System
Subject : 임베디드시스템설계
Professor : 오 윤 호
Department : 전자전기공학부
Student ID : 2019310649
Name : 임 용 성
Submission Date : 2021.10.01

1. CPU (C/C++)

CPU를 이용하여 single-threaded 방식으로 두 행렬을 이어보았다. 이때 배열은 1D array로 설정하고 진행하였다. 코드를 간략하게 중요한 부분만 설명하면 다음과 같다.

```
int* matrix1 = NULL;
int* matrix2 = NULL;
int* matrix3 = NULL;

int M = argc > 1 ? atoi(argv[1]) : 512;
int P = argc > 2 ? atoi(argv[2]) : 1024;
int Q = argc > 3 ? atoi(argv[3]) : 512;
int testNum = argc > 4 ? atoi(argv[4]) : 10;
int printkey = argc > 5 ? atoi(argv[5]) : 0;

struct timeval startTime, endTime;
double elapsedTime;
double elapsedTime_sum = 0;

matrix1 = (int *)malloc(sizeof(int) * M * P);
matrix2 = (int *)malloc(sizeof(int) * M * Q);
matrix3 = (int *)malloc(sizeof(int) * M * (P + Q));
```

과제를 위해 1D array를 할당하기 위해 malloc을 이용하여 동적으로 메모리를 할당하였다. 이때 할당을 위한 값은 cli환경에서 argv[]를 이용하여 받을 수 있게 설정하였다. 만약 이 값을 입력하지 않는다면 defalut로 512 x 1024 matrix와 512 x 512 matrix를 만들고, test는 10번 진행하고 결과를 출력하지 않게 설정하였다.
특히 여기서 argv[5]는 결과를 출력할지 말지에 대한 값으로 만약 0이 아니라면 matrix concat한 결과를 출력하게 설정하였다.

가장 중요한 execution time측정을 위한 코드와 concat 코드는 다음과 같다.

```
for (int i = 0 ; i < testNum ; i++)
{

    /*---------------------------------------- CPU TIME ESTIMATE START ----------------------------*/
    gettimeofday(&startTime, NULL);

    //concat
    matrix_concat(matrix3, matrix1, matrix2, M, P, Q);

    gettimeofday(&endTime, NULL);
    /*---------------------------------------- CPU TIME ESTIMATE END ------------------------------*/

    elapsedTime = (endTime.tv_sec - startTime.tv_sec) * 1000.00 + (endTime.tv_usec - startTime.tv_usec) / 1000.00 ;
    elapsedTime_sum += elapsedTime;
    printf(" Test : %d , Elapsed Time :  %lf ms\n", i + 1,  elapsedTime);
}
printf("-------------------------------------------------------------------------------------\n");
printf("\n Average Elapsed Time : %lf ms \n" , elapsedTime_sum / testNum);
printf("-------------------------------------------------------------------------------------\n\n");
```

gettimeofday function을 이용하여 시작시간과 끝시간을 각각의 주소에 할당하였고, elapsedTime 계산을 통해 ms 단위로 execution time을 측정하였다. 이때, for 문을 통해 testNum의 숫자만 큼 test를 하고 이를 결과에 나누어 평균 실행시간을 알 수 있게 프로그래밍하였다.

위에서 사용한 matrix_concat 함수는 다음과 같다.

```
void matrix_concat(int* matrix3, const int* matrix1, const int* matrix2, int m, int p, int q)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < p; j++)
        {
            matrix3[(p + q) * i + j] = matrix1[p * i + j];
        }
        for (int k =0; k < q; k++)
        {
            matrix3[(p + q) * i + (p + k)] = matrix2[q * i + k];
        }
    }
}
```

이제 작성한 코드를 실행하면 다음과 같은 결과를 얻을 수 있다. 코드를 실행하기 위해 다음의 명령어를 사용하였다.

./cpu_time 512 1024 512 10 0

```
user14@lab14:~/cuda/Lim_Yong/HW1/cpu$ ./cpu_time 512 1024 512 10 0
--------------------------------------------------------------------------------
CPU Execution Time Estimatation START
Input  Matrix : 512 x 1024
                512 x 512
Output Matrix : 512 x 1536
Number of Test : 10
 Test : 1 , Elapsed Time :  13.550000 ms
 Test : 2 , Elapsed Time :  12.408000 ms
 Test : 3 , Elapsed Time :  12.401000 ms
 Test : 4 , Elapsed Time :  12.403000 ms
 Test : 5 , Elapsed Time :  12.463000 ms
 Test : 6 , Elapsed Time :  12.410000 ms
 Test : 7 , Elapsed Time :  12.415000 ms
 Test : 8 , Elapsed Time :  12.416000 ms
 Test : 9 , Elapsed Time :  12.405000 ms
 Test : 10 , Elapsed Time :  12.404000 ms
--------------------------------------------------------------------------------

 Average Elapsed Time : 12.527500 ms
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
```

Fig. 1. Cpu_time 결과 출력

2. GPU (CUDA)

GPU를 사용하는 코드는 다음과 같다. 우선 cli 환경에서 값을 인가받도록 설계하였는데 그 부분을 보면 다음과 같다.

```
int block = atoi(argv[1]);
int thread = atoi(argv[2]);
int testNum = argc > 3 ? atoi(argv[3]) : 1;
int M = argc > 4 ? atoi(argv[4]) : 512;
int P = argc > 5 ? atoi(argv[5]) : 1024;
int Q = argc > 6 ? atoi(argv[6]) : 512;
int key = 0;                                    // matrix print?
key = argc > 7 ? atoi(argv[7]) : 0;
int findOptimal = 0;
findOptimal = argc > 8 ? atoi(argv[8]) : 0;
```

왼쪽의 코드를 보면 알 수 있듯이 argv를 이용하여 값을 인가받았다. Argv[3]부터는 값을 인가받지 않으면 default로 설정되게 하였다.

중요한 것은 argv[7]은 출력 결과를 print할지 말지를 정하는 부분이고, argv[8]은 최적의 thread와 block을 찾는 모드를 사용할지 말지를 정하는 부분이다.

이제 GPU의 측정시간을 측정하는 코드를 확인하면 다음과 같다. 아래의 코드는 findOptimal = 0 일때 GPU의 측정시간을 구하는 코드이다. CPU에서와 마찬가지로 testNum을 이용하여 여러 번 test후 결과의 평균을 출력하게 코딩하였고 그 단위는 ms로 cpu와 마찬가지로 통일하였다. GPU에서 execution time은 memcpy는 포함하지 않고 오로지 커널함수의 동작시간만을 측정하게 코딩하였다.

```
for(int i = 0; i < testNum ; i++)
{
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaMalloc((void **)&device_matrixA, sizeA);
    cudaMalloc((void **)&device_matrixB, sizeB);
    cudaMalloc((void **)&device_matrixC, sizeC);
    //Memory transfor Host To Device
    cudaMemcpy(device_matrixA, host_matrixA, sizeA, cudaMemcpyHostToDevice);
    cudaMemcpy(device_matrixB, host_matrixB, sizeB, cudaMemcpyHostToDevice);
    //ESTIMATE START
    cudaEventRecord(start, 0);
    //concat in GPU
    device_matrix_concat<<<block,thread>>>(device_matrixC, device_matrixA, device_matrixB, numOps, block, thread, M, P, Q);
    //ESTIMATE STOP
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    /*--------------------------- GPU Time Estimate End ------------------------------*/
    cudaMemcpy(host_matrixC, device_matrixC, sizeC, cudaMemcpyDeviceToHost);
    cudaEventElapsedTime( &time_ms, start, stop);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    cudaDeviceSynchronize();
    //Memory transfor Device To Host
    printf("TestNum : %2d , elapsedtime = %1f\n", i + 1, time_ms);
    elapsedtime_sum += time_ms;
    cudaFree(device_matrixA);
    cudaFree(device_matrixB);
    cudaFree(device_matrixC);
}
printf("\n-----------------------------------------------------------\n");
printf("\n Average ElapsedTime = %lf ms \n",elapsedtime_sum / testNum);
printf("\n-----------------------------------------------------------------------\n");
```

cudaEventCreate로 event를 생성하고, 이것은 마지막에 cudaEventDestory와 이어진다.

cudaEventRecord 함수를 사용하여 시간을 측정하였고 cudaEvnetElapsedTime 함수를 통해 측정시간을 계산하였다.

Test를 여러 번 돌리므로 메모리 할당과 제거를 for문안에 넣었다.

사용한 커널함수의 코드는 다음과 같다.

```
__global__
void device_matrix_concat(int * C, int * A, int * B, int numOps, int block, int thread, int m, int p, int q)
{
    int index = blockDim.x * blockIdx.x + threadIdx.x;
    int offset = gridDim.x * blockDim.x;


    for (int i = 0 ; i < numOps; i++)
    {
        if( index < m * (p + q))
        {
            if(index % (p + q) < p)
            {
                C[index] = A[p * (index / (p + q)) + index % (p + q)];
                //printf("HEllo  index : %d, A[%d] :)\n", index, p * (index / (p + q)) + index % (p + q) );
            }
            else
            {
                C[index] = B[q * (index / (p + q)) + index % (p + q) - p];
                //printf("HEllo  index : %d, B[%d] :)\n", index, q * (index / (p + q)) + index % (p + q) - p);
            }
            index += offset;
        }
    }
}
```

이제 실행시간을 얻기위해 실행시키면 결과는 다음과 같다.

```
user14@lab14:~/cuda/Lim_Yong/HW1/gpu$ ./gpu_time 16 32 10 512 1024 512 0 0

------------------------------------------------------------------------
GPU Execution Time Estimatation START
Input  Matrix :  512 x 1024
                 512 x 512
Output Matrix :  512 x 1536

Number of Blocks     :    16
Number of Threads    :    32
Number of Operation :  1536
Number of testnum    :    10
------------------------------------------------------------------------
TestNum :  1 , elapsedtime = 9.445052
TestNum :  2 , elapsedtime = 9.463021
TestNum :  3 , elapsedtime = 9.445313
TestNum :  4 , elapsedtime = 9.461718
TestNum :  5 , elapsedtime = 9.442500
TestNum :  6 , elapsedtime = 9.473750
TestNum :  7 , elapsedtime = 9.455834
TestNum :  8 , elapsedtime = 9.461979
TestNum :  9 , elapsedtime = 9.442552
TestNum : 10 , elapsedtime = 9.456302

--------------------------------------------------------

 Average ElapsedTime = 9.454802 ms

------------------------------------------------------------------------
```

3. Analyze & Optimal Configuration

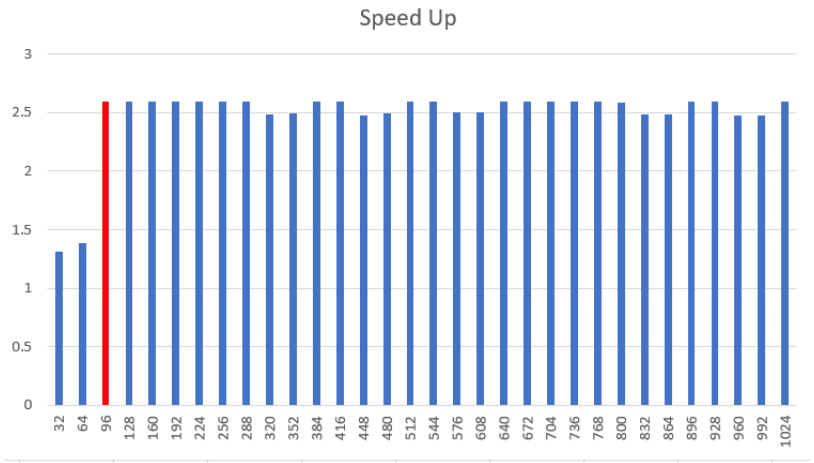
이제 분석을 해보면 다음과 같다. 우선 cpu의 execution time을 확인해보면 10번 시행하여 평균을 냈을 때, 12.5275ms가 나오는 것을 확인할 수 있다. 이제 gpu의 execution time이 가장 적을 때의 block과 thread를 찾아보면 다음과 같다.

앞서 GPU코드에서 만든 Optimal을 찾는 모드를 1로 설정(argv[8] = 1)하고 실행하면 다음과 같은 결과가 도출된다. Thread는 32의 배수로 설정해야하므로 32 ~ 1024 까지 32개의 case에 대해 모두 측정하였다. 각각의 Thread에 따른 block의 개수는 이번 과제에서 512 x (1024 + 512)의 matrix를 만들게 되므로 512 x 1536 = 786432 개의 Thread가 필요하게 되는데 이 값을 설정한 Thread수로 나누어 반올림한 값을 block의 개수로 설정하였다.

```
user14@lab14:~/cuda/Lim_Yong/HW1/gpu$ ./gpu_time 1891 416 20 512 1024 512 0 1

------------------------------------------------------------------
FIND OPTIMAL BLOCKS & THREADS START
Input  Matrix :  512 x 1024
                 512 x 512
Output Matrix :  512 x 1536


Wait a Minuite, This process needs a lot of time

 Block : 24576 , Thread :    32 ==> Average Elapsed Time = 9.569920 ms
 Block : 12288 , Thread :    64 ==> Average Elapsed Time = 9.033138 ms
 Block :  8192 , Thread :    96 ==> Average Elapsed Time = 4.825286 ms
 Block :  6144 , Thread :   128 ==> Average Elapsed Time = 4.827745 ms
 Block :  4916 , Thread :   160 ==> Average Elapsed Time = 4.830278 ms
 Block :  4096 , Thread :   192 ==> Average Elapsed Time = 4.832380 ms
 Block :  3511 , Thread :   224 ==> Average Elapsed Time = 4.828974 ms
 Block :  3072 , Thread :   256 ==> Average Elapsed Time = 4.834023 ms
 Block :  2731 , Thread :   288 ==> Average Elapsed Time = 4.827372 ms
 Block :  2458 , Thread :   320 ==> Average Elapsed Time = 5.033366 ms
 Block :  2235 , Thread :   352 ==> Average Elapsed Time = 5.030005 ms
 Block :  2048 , Thread :   384 ==> Average Elapsed Time = 4.832338 ms
 Block :  1891 , Thread :   416 ==> Average Elapsed Time = 4.832771 ms
 Block :  1756 , Thread :   448 ==> Average Elapsed Time = 5.068578 ms
 Block :  1639 , Thread :   480 ==> Average Elapsed Time = 5.031453 ms
 Block :  1536 , Thread :   512 ==> Average Elapsed Time = 4.831161 ms
 Block :  1446 , Thread :   544 ==> Average Elapsed Time = 4.831826 ms
 Block :  1366 , Thread :   576 ==> Average Elapsed Time = 5.001601 ms
 Block :  1294 , Thread :   608 ==> Average Elapsed Time = 4.998489 ms
 Block :  1229 , Thread :   640 ==> Average Elapsed Time = 4.831127 ms
 Block :  1171 , Thread :   672 ==> Average Elapsed Time = 4.829549 ms
 Block :  1118 , Thread :   704 ==> Average Elapsed Time = 4.831229 ms
 Block :  1069 , Thread :   736 ==> Average Elapsed Time = 4.832477 ms
 Block :  1024 , Thread :   768 ==> Average Elapsed Time = 4.832758 ms
 Block :   984 , Thread :   800 ==> Average Elapsed Time = 4.835792 ms
 Block :   946 , Thread :   832 ==> Average Elapsed Time = 5.031751 ms
 Block :   911 , Thread :   864 ==> Average Elapsed Time = 5.032469 ms
 Block :   878 , Thread :   896 ==> Average Elapsed Time = 4.832184 ms
 Block :   848 , Thread :   928 ==> Average Elapsed Time = 4.833568 ms
 Block :   820 , Thread :   960 ==> Average Elapsed Time = 5.062665 ms
 Block :   793 , Thread :   992 ==> Average Elapsed Time = 5.059316 ms
 Block :   768 , Thread :  1024 ==> Average Elapsed Time = 4.831630 ms
Number of testnum   :    20
Number of OptimalBlocks    :  8192
Number of OptimalThreads   :    96
Number of Operation :     1
------------------------------------------------------------------------

--------------------------------------------------

 Optimal Average ElapsedTime = 4.825286 ms

------------------------------------------------------------------------
```

Thread	Speed Up
32	1.30905
64	1.386838
96	2.596219
128	2.594897
160	2.593536
192	2.592408
224	2.594236
256	2.591527
288	2.595097
320	2.488891
352	2.490554
384	2.59243
416	2.592198
448	2.471601
480	2.489837
512	2.593062
544	2.592705
576	2.504693
608	2.506257
640	2.59308
672	2.593928
704	2.593026
736	2.592356
768	2.592205
800	2.590579
832	2.48969
864	2.489335
896	2.592513
928	2.591771
960	2.474487
992	2.476125
1024	2.59281

위의 결과는 각 시행에 대해 Test를 20번하였을 때의 평균 측정시간을 나타낸 것이다. Test를 20번 하였으므로 오차를 줄였다고 할 수 있다. 결국 위에서 알 수 있듯이 Block = 8192이고 Thread = 96일 때 Elapsed Time = 4.825286 ms 이고 Speed Up = 2.596219로 가장 빠르게 실행한다는 것을 알 수 있다. CPU의 실행시간을 벤치마크하여 구한 Speed up을 그래프로 나타내면 다음과 같다.



Thread가 32와 64를 제외한 대부분에 경우에 대해 Speed Up이 2.5 부근에 있는 것을 확인할 수 있다.

부가적으로 block과 thread를 다르게 하여 다양하게 하였을 때 어떤 결과가 나오는지를 확인해보면 다음과 같다.

blocks	threads	Speed Up	Excution time
1	32	0.30115	41.598392
2	32	0.64636	19.381657
4	32	0.47794	26.211493
8	32	1.02585	12.21177
16	32	1.32072	9.485326
32	32	1.86192	6.728265
64	32	1.74120	7.194741
128	32	1.90077	6.59075
256	32	1.98427	6.313406
512	32	2.02275	6.19331
CPU		1.00000	12.5275

blocks	threads	Speed Up	Excution time
1	64	0.63152	19.837204
2	64	1.14147	10.974882
4	64	0.95173	13.162828
8	64	1.31203	9.548163
16	64	1.95005	6.4242
32	64	1.96799	6.365624
64	64	2.00189	6.257826
128	64	2.06889	6.055174
256	64	2.08215	6.016628
512	64	2.06225	6.074683
CPU		1.00000	12.5275

blocks	threads	Speed Up	Excution time
1	96	0.85747	14.609785
2	96	0.71803	17.447147
4	96	1.01225	12.37595
8	96	1.79085	6.995282
16	96	1.96104	6.388193
32	96	1.95765	6.399268
64	96	2.00700	6.241893
128	96	2.08516	6.00792
256	96	2.10225	5.959078
512	96	2.07764	6.029672
CPU		1.00000	12.5275

blocks	threads	Speed Up	Excution time
1	128	0.66906	18.724068
2	128	0.69936	17.912926
4	128	1.32387	9.462766
8	128	2.04748	6.1185
16	128	1.98859	6.299693
32	128	2.00118	6.260057
64	128	2.06773	6.058578
128	128	1.86966	6.70042
256	128	2.10122	5.962026
512	128	2.08104	6.019838
CPU		1.00000	12.5275

blocks	threads	Speed Up	Excution time
1	256	0.69266	18.086067
2	256	1.33099	9.412172
4	256	2.06824	6.057073
8	256	2.10891	5.940271
16	256	2.02330	6.191604
32	256	2.08573	6.006292
64	256	2.10589	5.948797
128	256	2.10678	5.946271
256	256	2.08410	6.010979
512	256	1.73105	7.236948
CPU		1.00000	12.5275

blocks	threads	Speed Up	Excution time
1	512	1.34323	9.326369
2	512	2.02914	6.173787
4	512	2.07597	6.034531
8	512	2.08532	6.007469
16	512	2.10946	5.938729
32	512	2.12374	5.898797
64	512	2.11790	5.915068
128	512	2.09180	5.988865
256	512	1.84208	6.80075
512	512	1.62312	7.718177
CPU		1.00000	12.5275

blocks	threads	Speed Up	Excution time
1	1024	2.13372	5.871208
2	1024	2.06592	6.06388
4	1024	2.12430	5.89723
8	1024	2.13594	5.865109
16	1024	1.92800	6.497672
32	1024	2.08376	6.011968
64	1024	2.08369	6.012177
128	1024	1.89365	6.615537
256	1024	1.60330	7.813578
512	1024	1.35306	9.258635
CPU		1.00000	12.5275

위의 결과들은 처음에 최적의 block과 thread를 찾는 것에 비해 낮은 speedup을 갖는다. 이 이유에 대해 설명해보자면 다음과 같다.

이번 과제에서 주어진 matrix 는 512 x 1024 와 512 x 512 를 concat 하여 512 x 1536 의 결과를 도출하는 것인데 이때에 사용되는 thread 는 총 512 x 1536 = 786432 만 사용하면 될 것이다. 따라서 786432 / thread = block 으로 계산하여 그에 맞는 block 을 사용하는 것이 최적일 것이다. 그렇기 때문에 처음에 code 를 통해 도출한 것들이 Fig. 7 보다 더 빠른 것은 당연하다.

Find Best Configuration
Block : 8192
Thread : 96
ElpasedTime : 4.825286 ms

Find Best SpeedUp
SpeedUp = 2.596219