[CSED490C] Assignment Report: Lab6_cuda

- Student Id : 20220848
- Name : 선민수

1. Answering follwing questions

Q: How many global memory reads are being performed by your kernel?

A: `numElements + numAux1 + numAux2 + 2 * numAux1 + 2 * numElements`

Q: How many global memory writes are being performed by your kernel?

A: `numElements + numAux1 + numAux1 + numAux2 + numAux2 + numAux1 + numElements`

Q: How many times does a single thread block synchronize to reduce its portion of the array to a single value?

A: `O(logN)` **, where N is the block size**

Q: Is it possible to get different results from running the serial version and parallel version of scan? Explain.

A: It is possible to get different results. It is because the input array is changed in the scanning time, the parallel version of can has different result. When the input chagne occurs after the local scan(scan in block), the change does not affect the result since the partial sum of that block where the change occured is already computed and never computed again.

2. Template.cu

```
// Given a list (lst) of length n
// Output its prefix sum = {lst[0], lst[0] + lst[1], lst[0] + lst[1] + ...
// +
// lst[n-1]}

#include <gputk.h>

#define BLOCK_SIZE 512 //@@ You can change this

#define gpuTKCheck(stmt)                                                     \
  do {                                                                       \
    cudaError_t err = stmt;                                                  \
    if (err != cudaSuccess) {                                                \
      gpuTKLog(ERROR, "Failed to run stmt ", #stmt);                         \
      gpuTKLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));      \
      return -1;                                                             \
    }                                                                        \
  } while (0)

__global__ void add_block_sum(float *input, float *output, int len){
  int bid = blockIdx.x;
  int idx = bid * blockDim.x + threadIdx.x;

  if (bid > 0 && idx < len)
    output[idx] += input[bid - 1];
}

__global__ void scan(float *input, float *output, float *aux, int len) {
  //@@ Modify the body of this function to complete the functionality of
  //@@ the scan on the device
  //@@ You may need multiple kernel calls; write your kernels before this
  //@@ function and call them from here
  __shared__ float T[BLOCK_SIZE];

  int tid = threadIdx.x;
  int base_idx = blockIdx.x * blockDim.x;

  // import data
  if (base_idx + tid < len)
    T[tid] = input[base_idx + tid];
  else
```

```
    T[tid] = 0.0f;

  // pre-scan step
  for (unsigned stride = 1; stride <= blockDim.x; stride *= 2) {
    __syncthreads();

    int index = (tid + 1) * 2 * stride - 1;

    if (index < BLOCK_SIZE)
      T[index] += T[index - stride];
  }

  // post-scan step
  for (int stride = BLOCK_SIZE / 4; stride > 0; stride /= 2) {
    __syncthreads();

    int index = (tid + 1) * stride * 2 - 1;

    if (index + stride < BLOCK_SIZE)
      T[index + stride] += T[index];
  }

  // export data
  __syncthreads();
  if (base_idx + tid < len)
    output[base_idx + tid] = T[tid];
  if (aux != NULL && tid == blockDim.x - 1)
    aux[blockIdx.x] = T[tid];
}

int main(int argc, char **argv) {
  gpuTKArg_t args;
  float *hostInput;  // The input 1D list
  float *hostOutput; // The output list
  float *deviceInput;
  float *deviceOutput;
  int numElements; // number of elements in the list

  float *aux_1_in;
  float *aux_1_out;
  float *aux_2_in;
  float *aux_2_out;
  int numAux1;
```

```
int numAux2;

args = gpuTKArg_read(argc, argv);

gpuTKTime_start(Generic, "Importing data and creating memory on host");
hostInput = (float *)gpuTKImport(gpuTKArg_getInputFile(args, 0), &numElements);
hostOutput = (float *)malloc(numElements * sizeof(float));
gpuTKTime_stop(Generic, "Importing data and creating memory on host");

gpuTKLog(TRACE, "The number of input elements in the input is ",
      numElements);

gpuTKTime_start(GPU, "Allocating GPU memory.");
gpuTKCheck(cudaMalloc((void **)&deviceInput, numElements * sizeof(float)));
gpuTKCheck(cudaMalloc((void **)&deviceOutput, numElements * sizeof(float)));

numAux1 = (numElements - 1) / BLOCK_SIZE + 1;
numAux2 = (numAux1 - 1) / BLOCK_SIZE + 1;
gpuTKCheck(cudaMalloc((void **)&aux_1_in, numAux1 * sizeof(float)));
gpuTKCheck(cudaMalloc((void **)&aux_1_out, numAux1 * sizeof(float)));
gpuTKCheck(cudaMalloc((void **)&aux_2_in, numAux2 * sizeof(float)));
gpuTKCheck(cudaMalloc((void **)&aux_2_out, numAux2 * sizeof(float)));
gpuTKTime_stop(GPU, "Allocating GPU memory.");

gpuTKTime_start(GPU, "Clearing output memory.");
gpuTKCheck(cudaMemset(deviceOutput, 0, numElements * sizeof(float)));
gpuTKTime_stop(GPU, "Clearing output memory.");

gpuTKTime_start(GPU, "Copying input memory to the GPU.");
gpuTKCheck(cudaMemcpy(deviceInput, hostInput, numElements * sizeof(float),
                  cudaMemcpyHostToDevice));
gpuTKTime_stop(GPU, "Copying input memory to the GPU.");

//@@ Initialize the grid and block dimensions here
dim3 dimBlock(BLOCK_SIZE, 1, 1);

dim3 dimGrid_1(numAux1, 1, 1);
dim3 dimGrid_2(numAux2, 1, 1);
dim3 dimGrid_3((numAux2 - 1) / BLOCK_SIZE, 1, 1);
dim3 dimGrid_4(numAux2, 1, 1);
dim3 dimGrid_5(numAux1, 1, 1);

gpuTKTime_start(Compute, "Performing CUDA computation");
```

```
  //@@ Modify this to complete the functionality of the scan
  //@@ on the deivce
  scan<<<dimGrid_1, dimBlock, BLOCK_SIZE * sizeof(float)>>>(deviceInput, deviceOutput,
  cudaDeviceSynchronize();
  scan<<<dimGrid_2, dimBlock, BLOCK_SIZE * sizeof(float)>>>(aux_1_in, aux_1_out, aux_2_
  cudaDeviceSynchronize();
  scan<<<dimGrid_3, dimBlock, BLOCK_SIZE * sizeof(float)>>>(aux_2_in, aux_2_out, NULL,
  cudaDeviceSynchronize();
  add_block_sum<<<dimGrid_4, dimBlock>>>(aux_2_out, aux_1_out, numAux1);
  cudaDeviceSynchronize();
  add_block_sum<<<dimGrid_5, dimBlock>>>(aux_1_out, deviceOutput, numElements);
  cudaDeviceSynchronize();

  gpuTKTime_stop(Compute, "Performing CUDA computation");

  gpuTKTime_start(Copy, "Copying output memory to the CPU");
  gpuTKCheck(cudaMemcpy(hostOutput, deviceOutput, numElements * sizeof(float),
                     cudaMemcpyDeviceToHost));
  gpuTKTime_stop(Copy, "Copying output memory to the CPU");

  gpuTKTime_start(GPU, "Freeing GPU Memory");
  cudaFree(deviceInput);
  cudaFree(deviceOutput);

  cudaFree(aux_1_in);
  cudaFree(aux_1_out);
  cudaFree(aux_2_in);
  cudaFree(aux_2_out);

  gpuTKTime_stop(GPU, "Freeing GPU Memory");

  gpuTKSolution(args, hostOutput, numElements);

  free(hostInput);
  free(hostOutput);

  return 0;
}
```

3. Execution times

Execution Systems

All compilation and the executions are made on docker container.
The number in the indices in the table and the legend in the chart means the number of threads per block.

TITANXP

```
srun -p titanxp -N 1 -n 6 -t 02:00:00 --gres=gpu:1 --pty /bin/bash -l
```

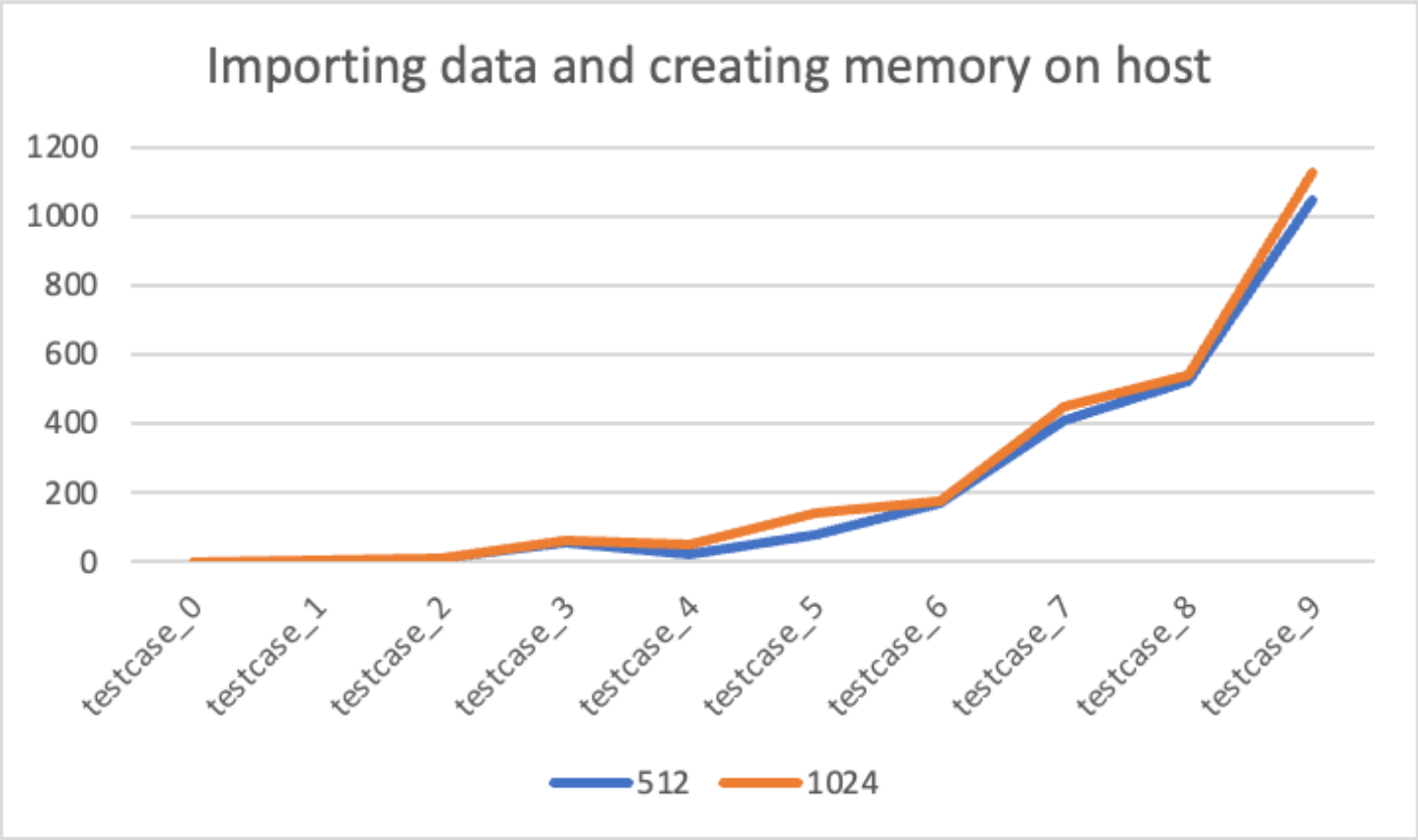
- Cluster : `cse-cluster1.postech.ac.kr`
- Docker Image : `nvidia:cuda/12.0.1-devel-ubuntu22.04`
- Driver Version : `525.85.12`
- Cuda Version : `12.0`

Execution Script

```
base="/workspace"
cd $base/sources
make template
echo > $base/result
for idx in {0..9}
do
    echo "Testcase $idx"
    cd $base/sources/ListScan/Dataset/$idx
    ./../../../ListScan_template -e output.raw -i input.raw -o o.raw -t vector >> $base
    echo >> $base/result
done
```

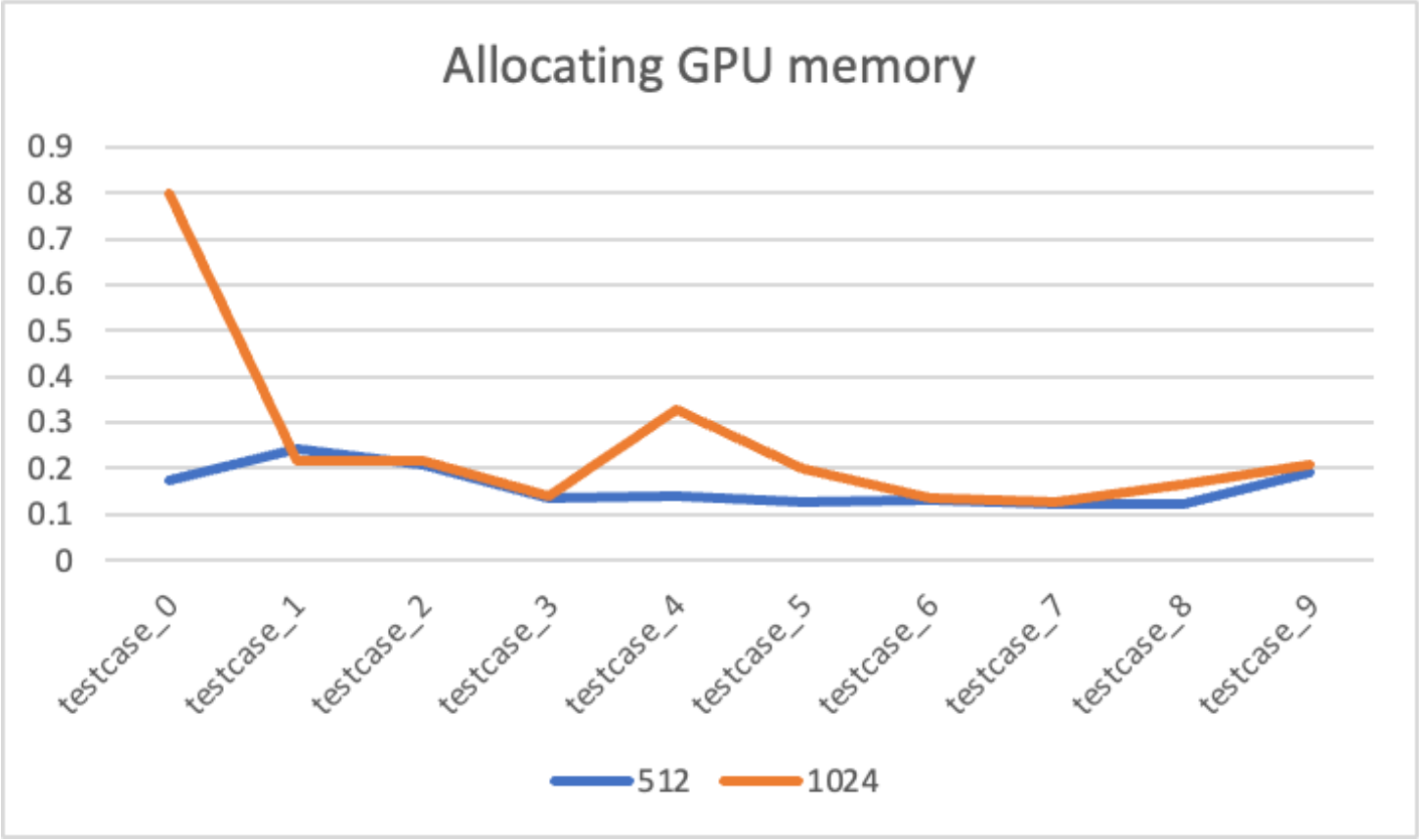
1 [Importing data and creating memory on host]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	1.07818	1.69885	10.0834	54.438	22.6888	81.1099	168.278	410.867	520.782	1045.42
1024	1.048	1.77805	10.5393	59.6358	50.104	140.55	174.371	449.914	537.913	1125.29



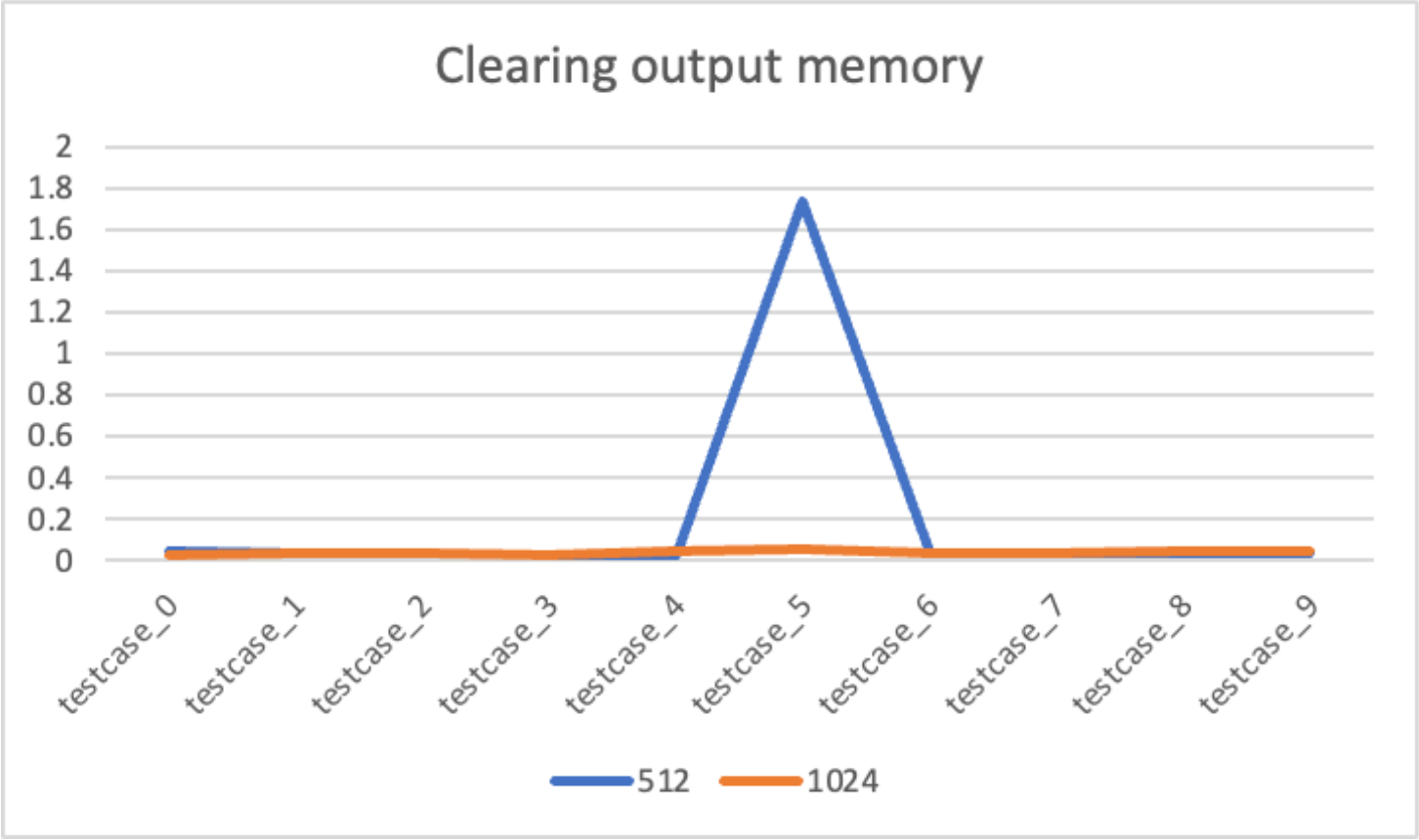
2 [Allocating GPU memory]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	0.172555	0.240581	0.207134	0.135254	0.139347	0.125575	0.131448	0.121845	0.123383	0.191593
1024	0.797845	0.218963	0.21611	0.140921	0.32711	0.200643	0.134361	0.128536	0.163523	0.20681



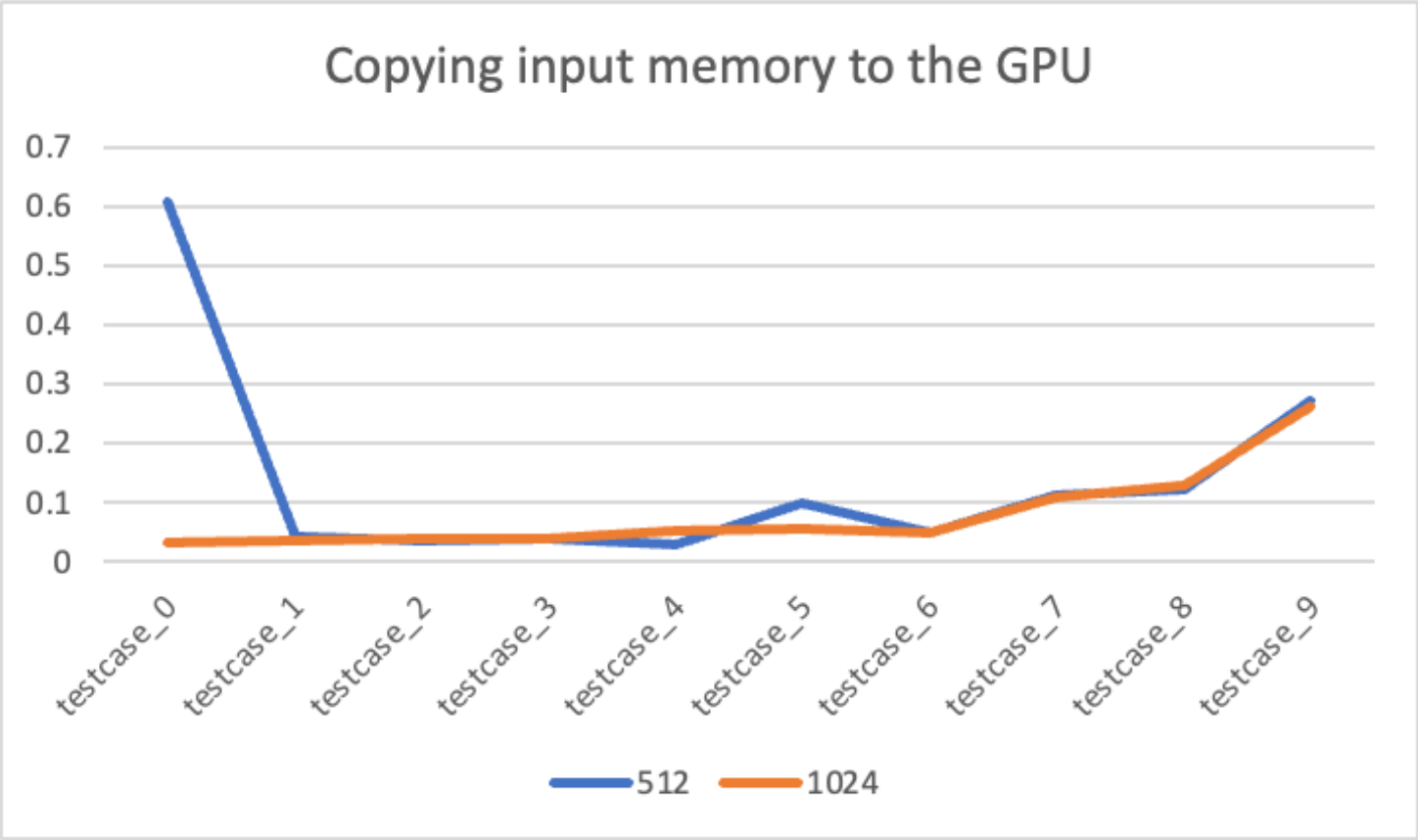
3 [Clearing output memory]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	0.0408	0.035809	0.033809	0.022315	0.022525	1.74027	0.040179	0.036348	0.03812	0.035932
1024	0.028089	0.035725	0.036184	0.021868	0.046429	0.058279	0.038765	0.03972	0.04212	0.046183



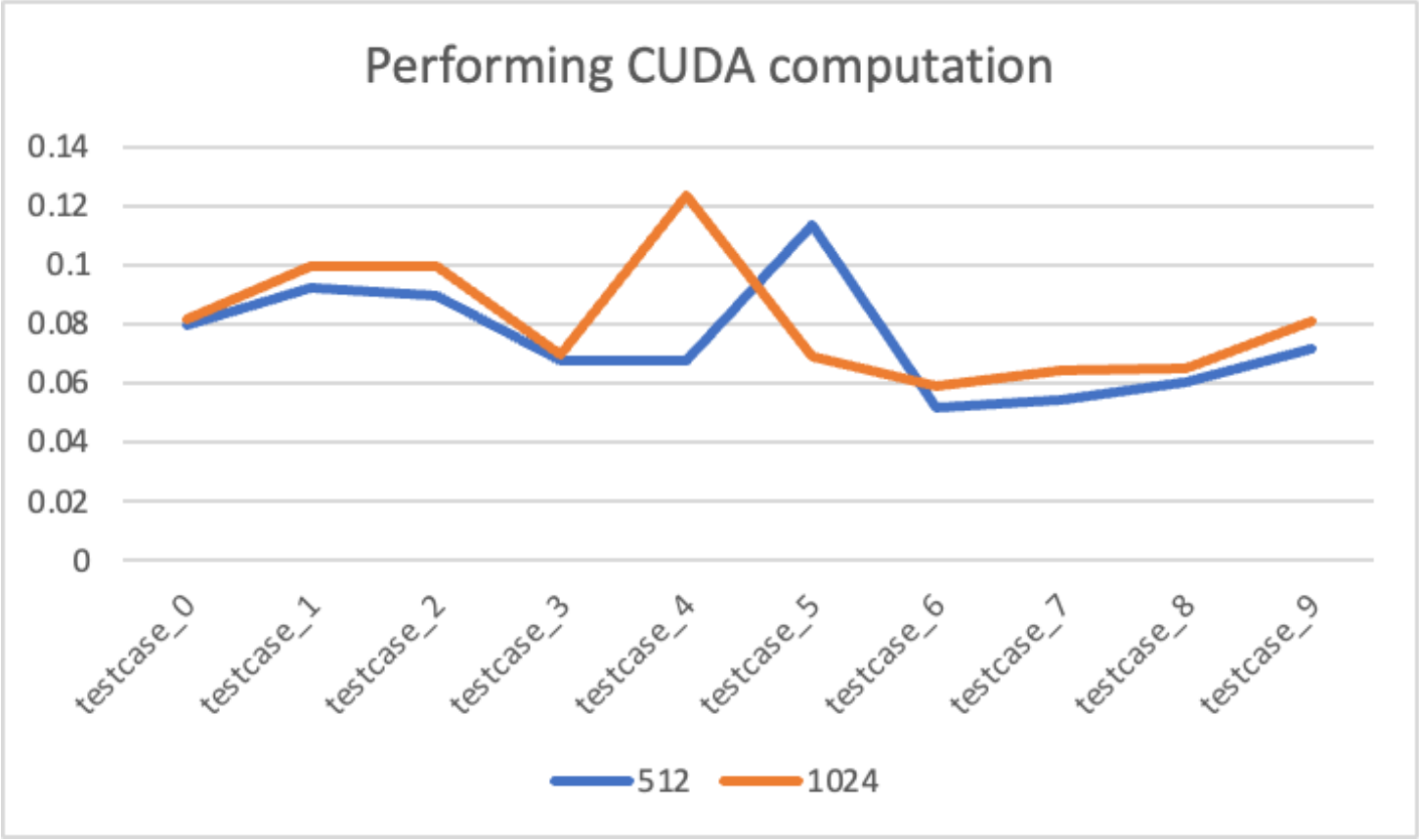
4 [Copying input memory to the GPU]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	0.606441	0.04196	0.037215	0.039353	0.02912	0.099491	0.048744	0.112839	0.121293	0.272813
1024	0.033644	0.035914	0.040287	0.037493	0.05258	0.055095	0.049664	0.107587	0.128493	0.262387



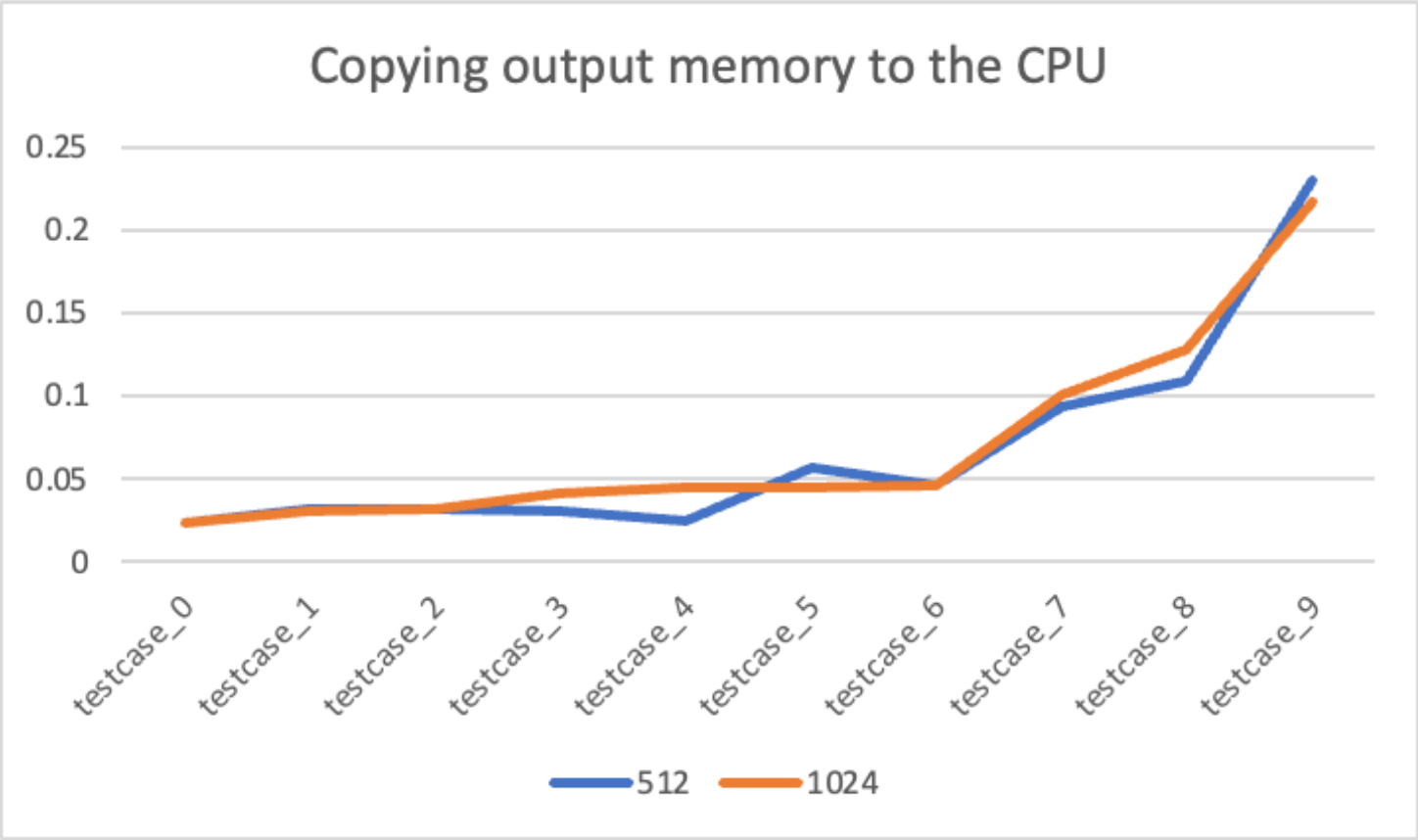
5 [Performing CUDA computation]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	0.079329	0.092104	0.089778	0.067832	0.067727	0.113374	0.051875	0.054633	0.060093	0.071447
1024	0.081846	0.099399	0.099452	0.069342	0.123539	0.068949	0.059065	0.064596	0.065193	0.080933



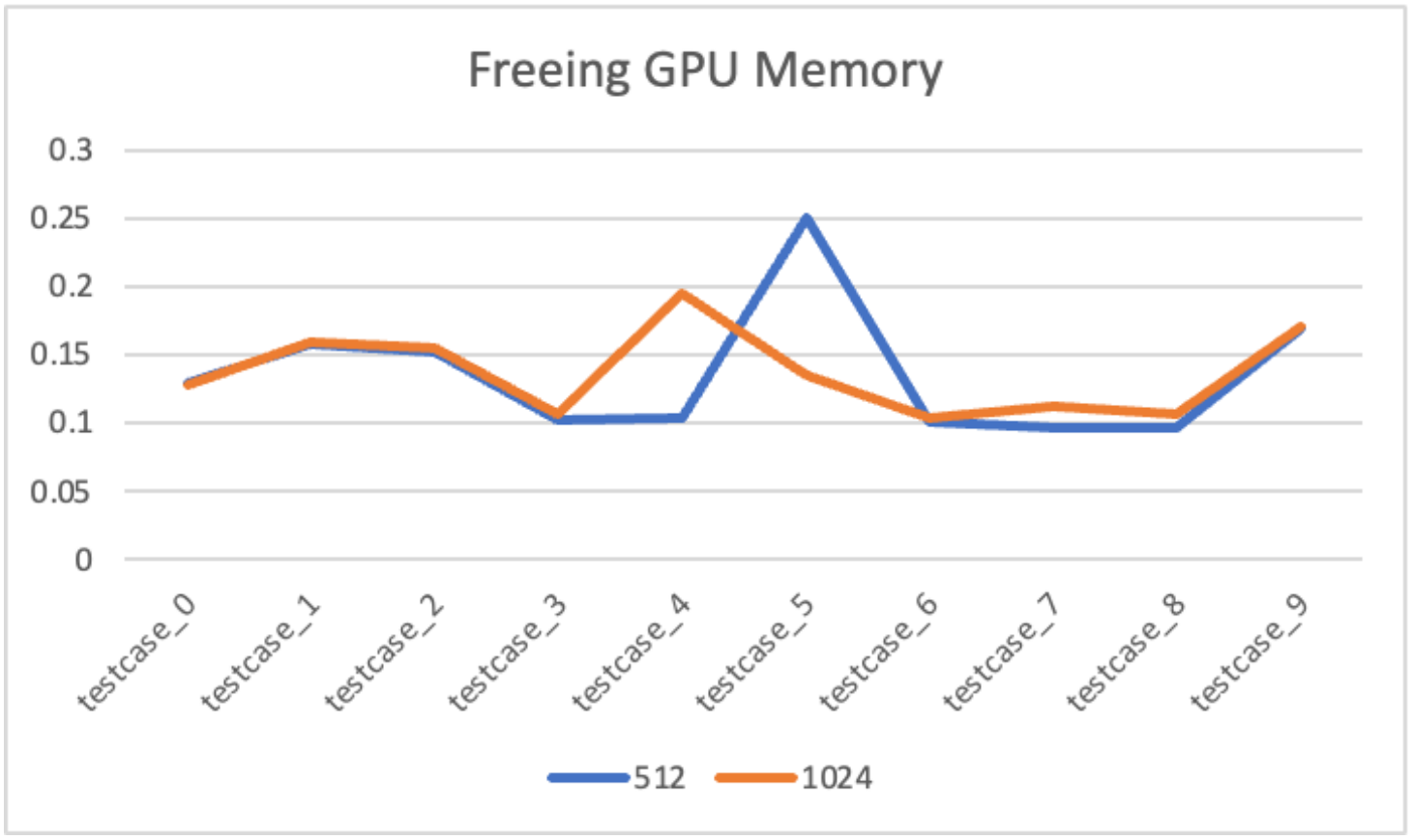
6 [Copying output memory to the CPU]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	0.023702	0.031721	0.031263	0.030531	0.024294	0.056429	0.046303	0.092996	0.108644	0.229908
1024	0.023228	0.030641	0.031173	0.041045	0.044311	0.044435	0.045696	0.100118	0.128312	0.216865



5 [Freeing GPU Memory]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7	testcase_8	testcase_9
512	0.1293	0.158528	0.152645	0.10193	0.103022	0.249847	0.100306	0.096524	0.095945	0.169321
1024	0.127458	0.158796	0.155047	0.106879	0.195584	0.135108	0.103473	0.111634	0.105905	0.171005


Freeing GPU Memory
0.3
0.25
0.2
0.15
0.1
0.05
0
testcase_0
testcase_1
testcase_2
testcase_3
testcase_4
testcase_5
testcase_6
testcase_7
testcase_8
testcase_9
512
1024