[CSED490C] Assignment Report: Lab4_cuda

- Student Id : 20220848
- Name : 선민수

1. Answering follwing questions

Q: For the histogram kernel, how many atomic operations are being performed by your kernel? Explain.

A: Total `inputLength + 2 * NUM_BINS` times atomic operations. `inputLength` times atomic operations for the privated histogram operation, and `NUM_BINS` for collecting the privated histograms and another `NUM_BINS` for the cleaning up the bins.

Q: For the histogram kerenl, what contentions would you expect if every element in the array has the same value?

A: Since the atomic operation concentrates on the value that every element has, the each time for computing the atomic operation will increase.

Q: For the histogram kernel, what contentions would you expect if every element in the input array has a random value?

A: Since the values in the input array are random, bank conflicts in the shared memory by using the privatization would occur.

2. Template.cu

```
#include <gputk.h>

#define NUM_BINS 4096

#define CUDA_CHECK(ans)                                                   \
  { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char *file, int line,
                      bool abort = true) {
  if (code != cudaSuccess) {
    fprintf(stderr, "GPUassert: %s %s %d\n", cudaGetErrorString(code),
            file, line);
    if (abort)
      exit(code);
  }
}

//@@ CUDA Kernel
__global__ void histo_kernel(unsigned int *Input, unsigned int *Bins, int inputLength)
{
  int tid = threadIdx.x;
  int idx = blockDim.x * blockIdx.x + tid;
  __shared__ unsigned int _private[NUM_BINS];

  while (tid < NUM_BINS)
  {
    _private[tid] = 0;
    tid += blockDim.x;
  }
  __syncthreads();

  while (idx < inputLength)
  {
    atomicAdd (&(_private[Input[idx]]), 1);
    idx += blockDim.x;
  }
  __syncthreads();

  tid = threadIdx.x;
  while (tid < NUM_BINS)
  {
    atomicAdd (&(Bins[tid]), _private[tid]);
```

```
    atomicMin (&(Bins[tid]), 127);
    tid += blockDim.x;
  }
  __syncthreads();
}

int main(int argc, char *argv[]) {
  gpuTKArg_t args;
  int inputLength;
  unsigned int *hostInput;
  unsigned int *hostBins;
  unsigned int *deviceInput;
  unsigned int *deviceBins;

  args = gpuTKArg_read(argc, argv);

  gpuTKTime_start(Generic, "Importing data and creating memory on host");
  hostInput = (unsigned int *)gpuTKImport(gpuTKArg_getInputFile(args, 0),
                                          &inputLength, "Integer");
  hostBins = (unsigned int *)malloc(NUM_BINS * sizeof(unsigned int));
  gpuTKTime_stop(Generic, "Importing data and creating memory on host");

  gpuTKLog(TRACE, "The input length is ", inputLength);
  gpuTKLog(TRACE, "The number of bins is ", NUM_BINS);

  gpuTKTime_start(GPU, "Allocating GPU memory.");
  //@@ Allocate GPU memory here
  cudaMalloc((void **)&deviceInput, inputLength * sizeof(unsigned int));
  cudaMalloc((void **)&deviceBins, NUM_BINS * sizeof(unsigned int));
  CUDA_CHECK(cudaDeviceSynchronize());
  gpuTKTime_stop(GPU, "Allocating GPU memory.");

  gpuTKTime_start(GPU, "Copying input memory to the GPU.");
  //@@ Copy memory to the GPU here
  cudaMemcpy(deviceInput, hostInput, inputLength * sizeof(unsigned int), cudaMemcpyHost
  CUDA_CHECK(cudaDeviceSynchronize());
  gpuTKTime_stop(GPU, "Copying input memory to the GPU.");

  // Launch kernel
  // ----------------------------------------------------------
  gpuTKLog(TRACE, "Launching kernel");
  gpuTKTime_start(Compute, "Performing CUDA computation");
  //@@ Perform kernel computation here
```

```
  dim3 dimGrid(1, 1, 1);
  dim3 dimBlock(1024, 1, 1);
  histo_kernel<<<dimGrid, dimBlock, NUM_BINS * sizeof(unsigned int)>>> (deviceInput, de
  gpuTKTime_stop(Compute, "Performing CUDA computation");

  gpuTKTime_start(Copy, "Copying output memory to the CPU");
  //@@ Copy the GPU memory back to the CPU here
  cudaMemcpy(hostBins, deviceBins, NUM_BINS * sizeof(unsigned int),cudaMemcpyDeviceToHo
  CUDA_CHECK(cudaDeviceSynchronize());
  gpuTKTime_stop(Copy, "Copying output memory to the CPU");

  gpuTKTime_start(GPU, "Freeing GPU Memory");
  //@@ Free the GPU memory here
  cudaFree(deviceInput);
  cudaFree(deviceBins);
  gpuTKTime_stop(GPU, "Freeing GPU Memory");

  // Verify correctness
  // -----------------------------------------------------
  gpuTKSolution(args, hostBins, NUM_BINS);

  free(hostBins);
  free(hostInput);
  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`

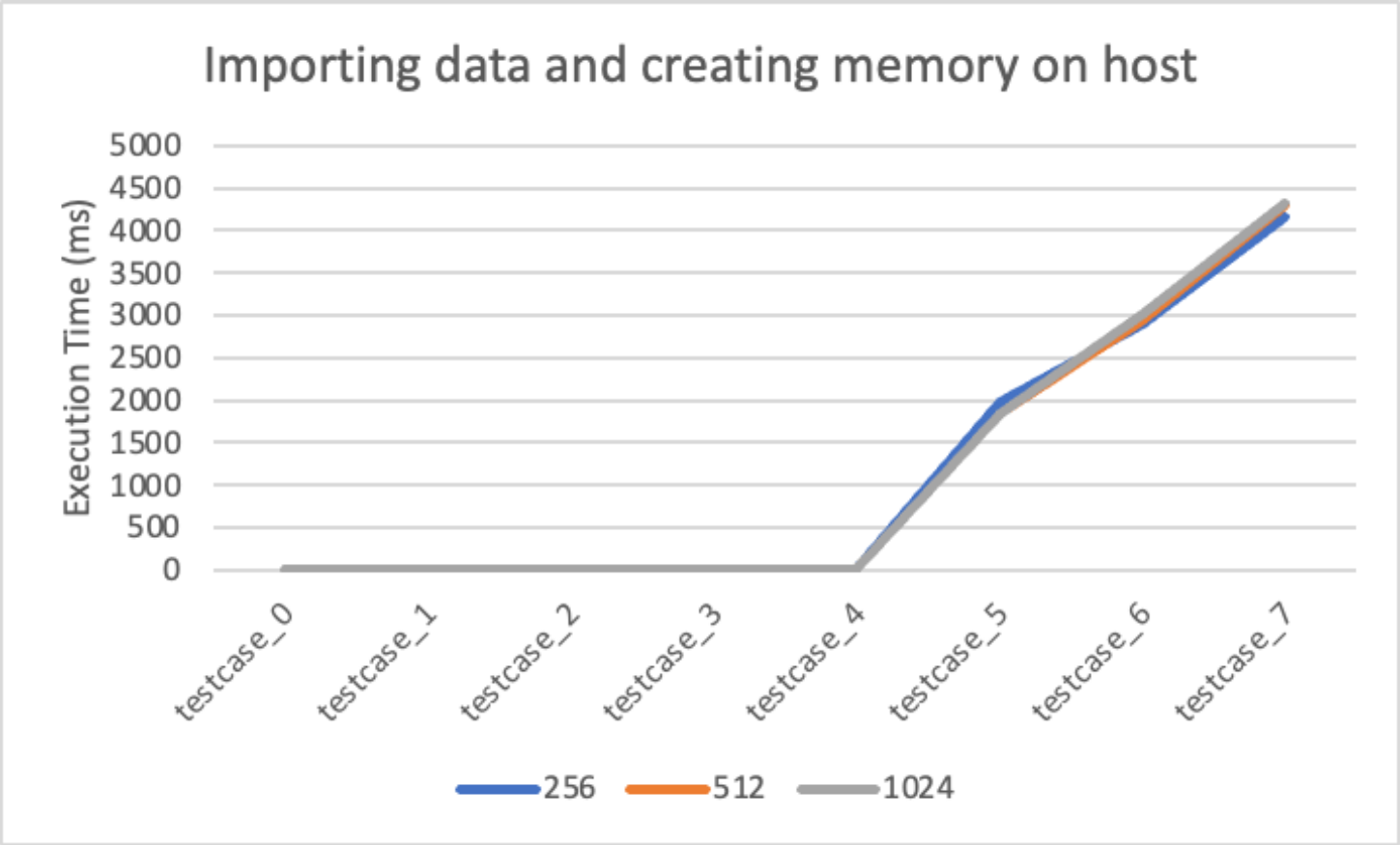
- Driver Version : `525.85.12`
- Cuda Version : `12.0`

Execution Script

```
base="/workspace"
cd $base/sources
make template
echo > $base/result
for idx in {0..7}
do
    echo "Testcase $idx"
    cd $base/sources/Histogram/Dataset/$idx
    ./../../../Histogram_template -e output.raw -i input.raw -o o.raw -t integral_vecto
    echo >> $base/result
done
```

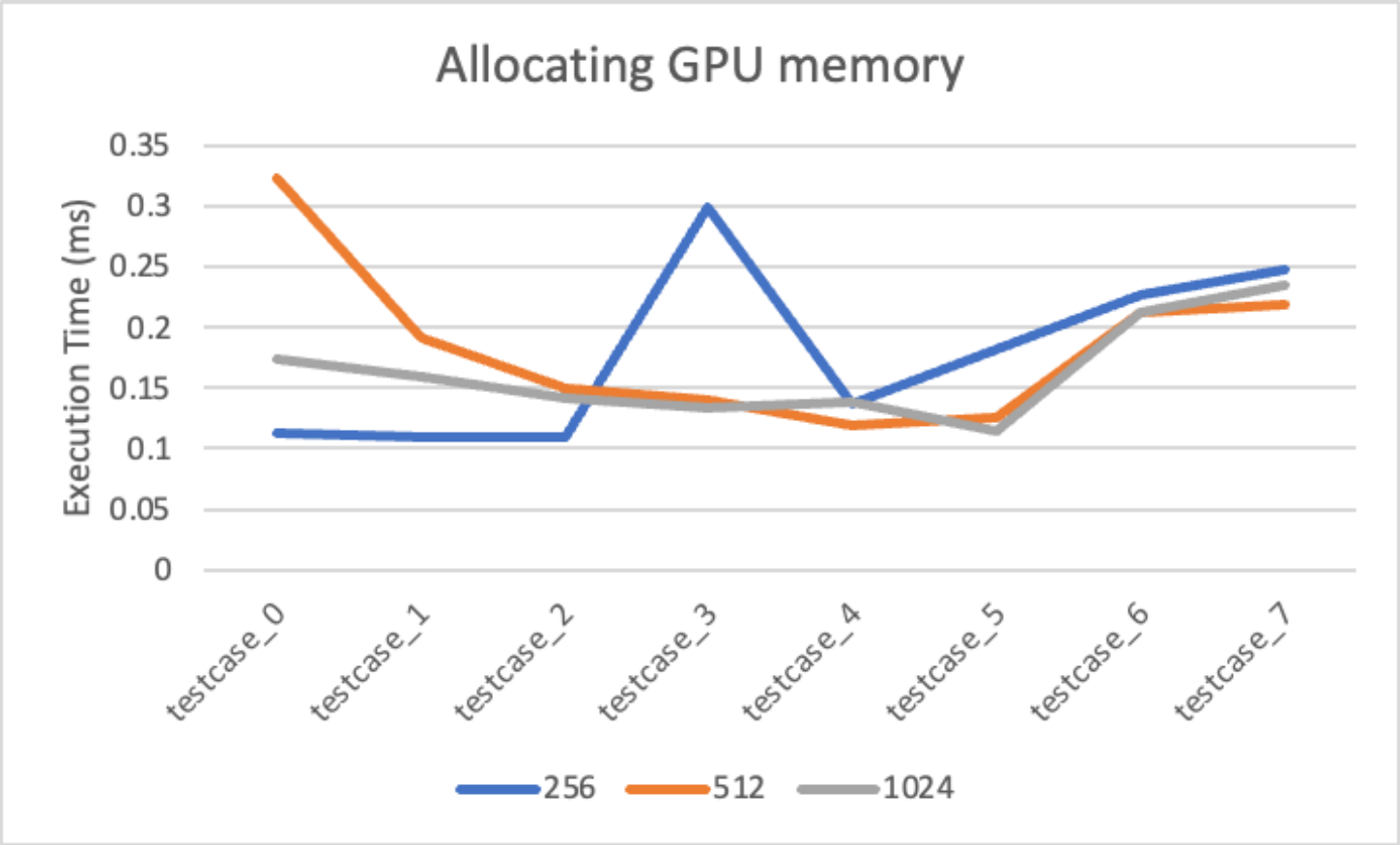
1 [Doing GPU memory allocation]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7
256	0.681389	5.09705	2.86542	7.06933	0.659477	1983.91	2902.25	4150.29
512	0.963682	9.42581	3.77294	3.30562	0.511611	1843.79	2946.13	4292.36
1024	0.663061	7.41069	3.55907	3.2868	0.509823	1830.75	3012.8	4330.04



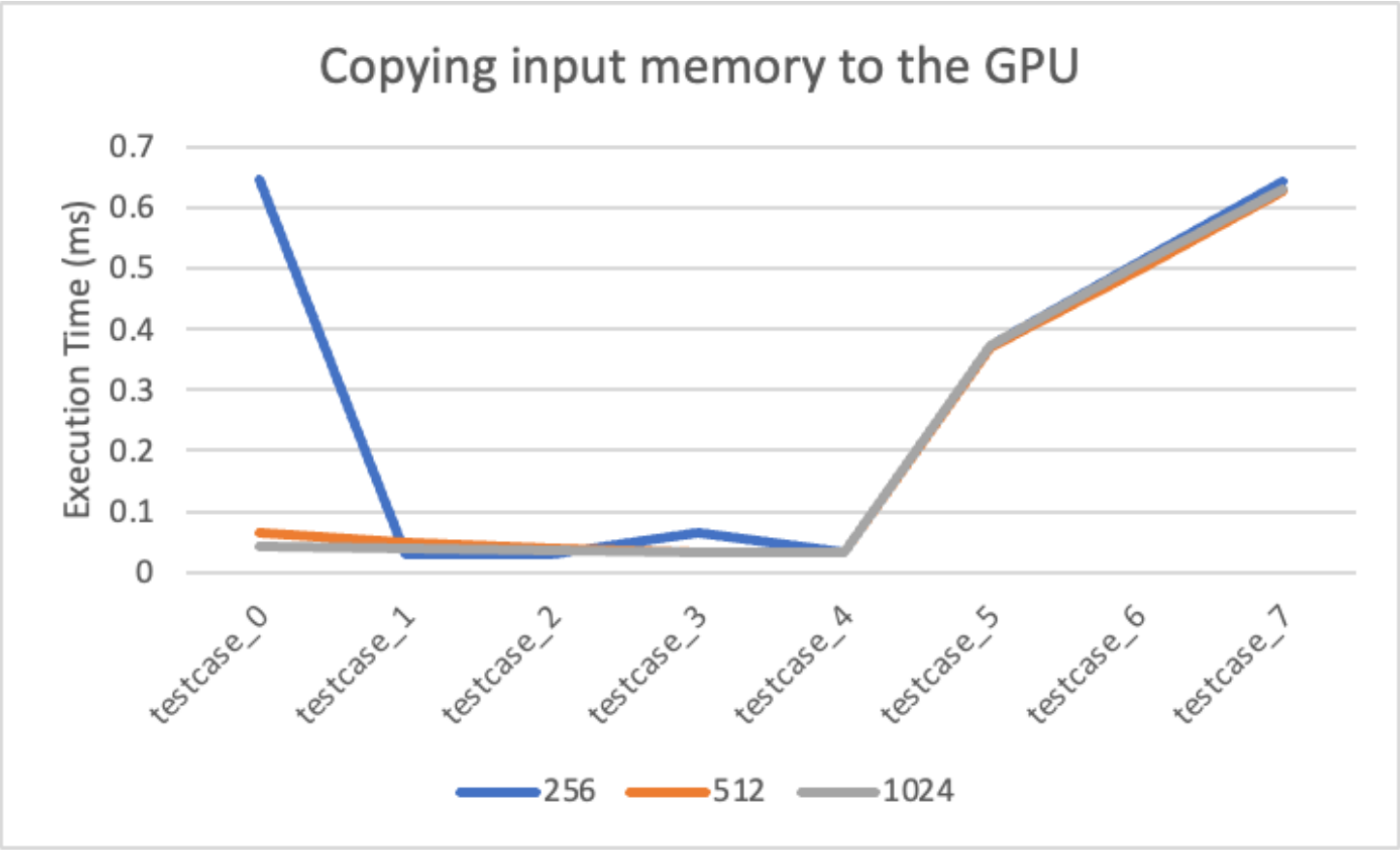
2 [Copying data to the GPU]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7
256	0.112083	0.109958	0.109936	0.299795	0.136711	0.182385	0.227204	0.247762
512	0.322962	0.192058	0.150385	0.140709	0.119741	0.125422	0.211594	0.21879
1024	0.174378	0.159254	0.14222	0.13334	0.137834	0.115081	0.212611	0.234154



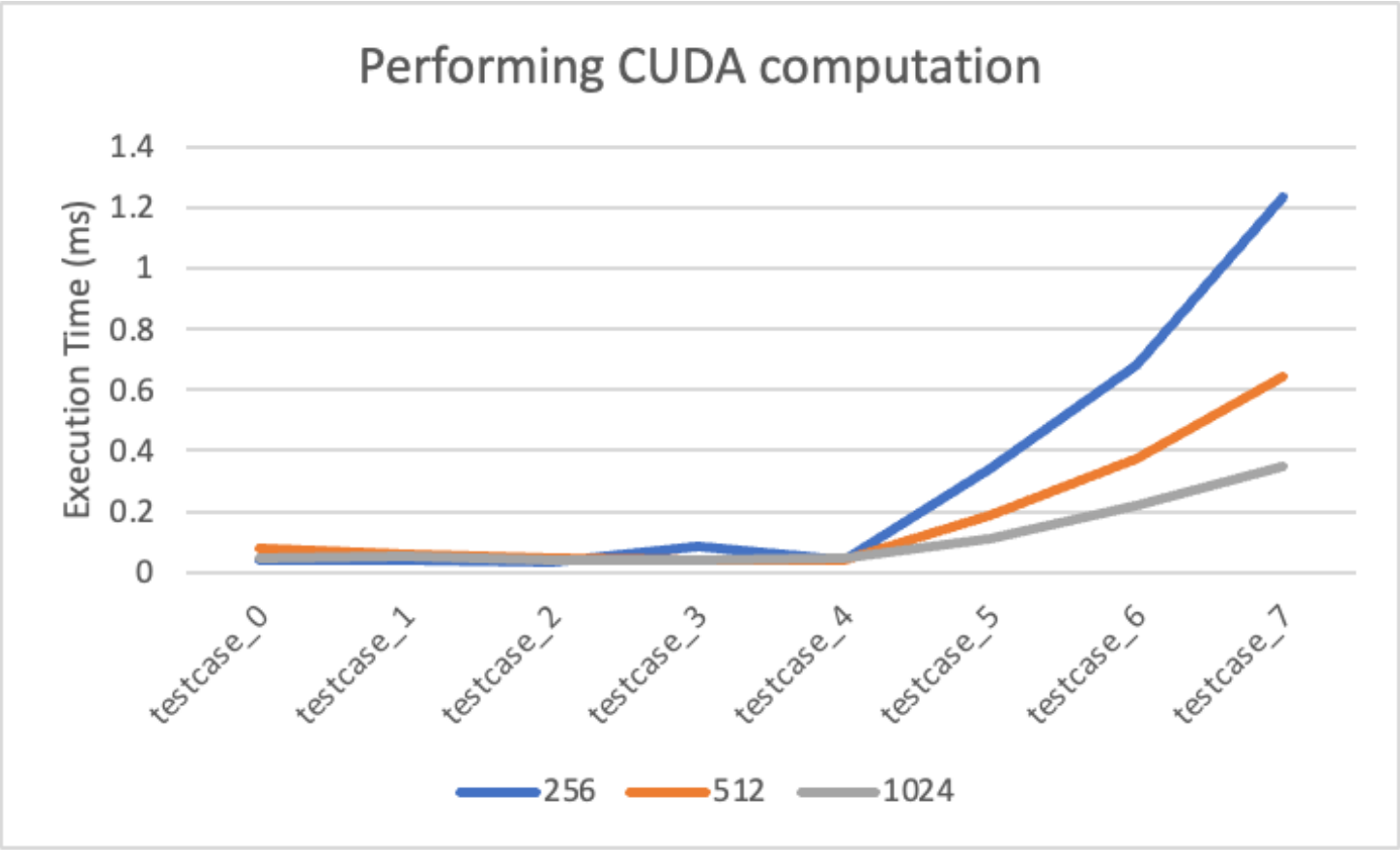
3 [Doing the computation on the GPU]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7
256	0.646429	0.028891	0.028808	0.064373	0.033474	0.373463	0.508715	0.642942
512	0.0666	0.049216	0.038761	0.032653	0.031809	0.368715	0.495535	0.628214
1024	0.041585	0.040987	0.036898	0.032454	0.031896	0.372253	0.505721	0.630526



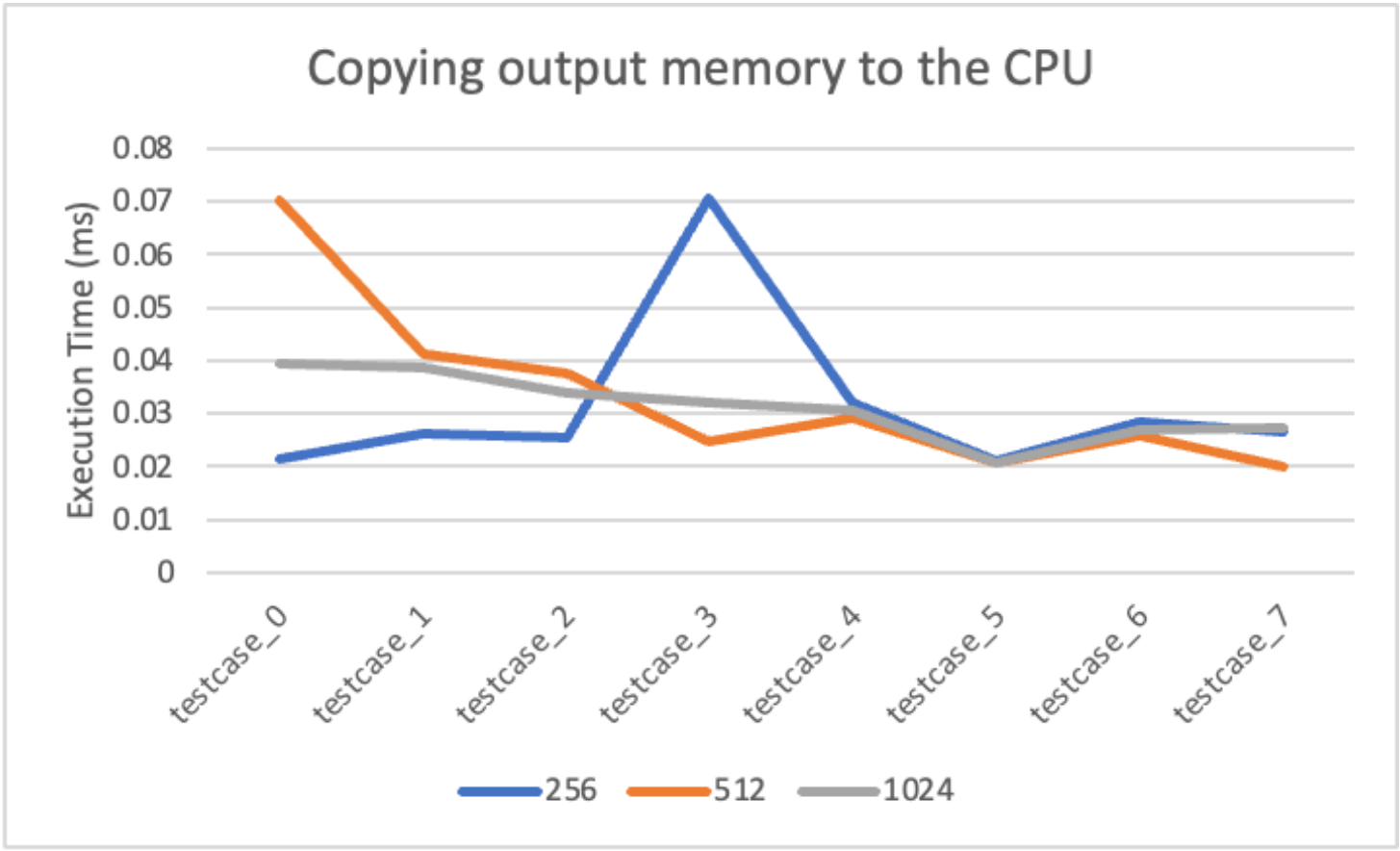
4 [Copying data from the GPU]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7
256	0.037896	0.038054	0.036994	0.085338	0.039197	0.339798	0.680096	1.23182
512	0.080759	0.056437	0.046833	0.042552	0.040064	0.189743	0.371661	0.642952
1024	0.048028	0.049978	0.043399	0.04111	0.043854	0.114001	0.217546	0.350943



5 [Doing GPU Computation (memory + compute)]

	testcase_0	testcase_1	testcase_2	testcase_3	testcase_4	testcase_5	testcase_6	testcase_7
256	0.021397	0.02625	0.025451	0.070736	0.032079	0.020839	0.028426	0.026585
512	0.070109	0.041298	0.037582	0.02481	0.0292	0.02052	0.025946	0.019965
1024	0.039416	0.038536	0.033923	0.03186	0.03042	0.020602	0.026864	0.027134


Copying output memory to the CPU
Execution Time (ms)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
testcase_0
testcase_1
testcase_2
testcase_3
testcase_4
testcase_5
testcase_6
testcase_7
256
512
1024