Homework 6

School of Data and Computer Science
17341015 Hongzheng Chen

1.Problem 1

Start from the provided skeleton code `error-test.cu` that provides some convenience macros for error checking. The macros are defined in the header file `error_checks_1.h`. Add the missing memory allocations and copies and the kernel launch and check that your code works.

1. What happens if you try to launch kernel with too large block size? When do you catch the error if you remove the `cudaDeviceSynchronize()` call?
2. What happens if you try to dereference a pointer to device memory in host code?
3. What if you try to access host memory from the kernel? Remember that you can use also cuda-memcheck!
4. If you have time, you can also check what happens if you remove all error checks and do the same tests again.

Answer. The code is in `error-test.cu`, and the result is shown below.

```
./error-test
  0.0
  2.0
  6.0
 12.0
 20.0
 30.0
 42.0
 56.0
 72.0
 90.0
110.0
132.0
156.0
182.0
210.0
240.0
272.0
306.0
342.0
380.0
```

1. The maximum block size of my GPU is 1024 (threads), and I request 2048 threads, which results in the following error.

```
Error: vector_add kernel at error-test.cu(50): invalid configuration argument
```

   Well, even I comment the `cudaDeviceSynchronize()`, the error is still caught after the `vectorAdd` function, and the reported error is the same.

2. It will cause a segment fault, since the memory is not allocated on the host.

3. It will cause an illegal memory access, since the memory is not allocated on the device. Error as shown below.

```
Error: vector_add kernel at error-test.cu(53): an illegal memory access was
    ↪ encountered
```

4. For 1 and 3, the program runs as usual and no errors will be reported. If the block size is set too large, then the calculation on GPU will be crashed. The result vector becomes all zeros. For 2, since the access happens on CPU, thus error still will be caught, which is a segment fault.

2.Problem 2

In this exercise we will implement a Jacobi iteration which is a very simple finite-difference scheme. Familiarize yourself with the provided skeleton (see `error_checks.h`, `jacobi.h`, and `jacobi.cu`). Then implement following things:

- Write the missing CUDA kernel `sweepGPU` that implements the same algorithm as the `sweepCPU` function.
- Check that the reported averate difference is in the order of the numerical accuracy.

Experiment with different grid and block sizes and compare the execution times.

Answer. Please see the code in `jacobi.cu`. We only need to take each iteration of Jacobi into a GPU kernel and make a GPU thread run for it. The hardest thing of this problem may be the index calculation. Also, be careful of the boundary of each index.

The result is shown below, where we can see the numerical accuracy is very high while GPU is much faster than CPU.

```
./jacobi
100 0.00972858
200 0.00479832
300 0.00316256
400 0.00234765
500 0.00186023
600 0.00153621
700 0.0013054
800 0.00113277
900 0.000998881
1000 0.000892078
1100 0.00080496
1200 0.000732594
1300 0.000671564
1400 0.000619434
1500 0.000574415
1600 0.000535167
1700 0.000500665
1800 0.000470113
CPU Jacobi: 4.54337 seconds, 1800 iterations
100 0.00972858
200 0.00479832
300 0.00316256
400 0.00234765
500 0.00186023
600 0.00153621
700 0.0013054
800 0.00113277
900 0.000998881
1000 0.000892078
1100 0.00080496
1200 0.000732594
1300 0.000671564
1400 0.000619434
1500 0.000574415
1600 0.000535167
1700 0.000500665
1800 0.000470113
GPU Jacobi: 0.026131 seconds, 1800 iterations
Average difference is 1.61266e-16
```