University of California, Riverside
Department of Computer Science & Engineering

Title: GPU & Parallel Programming – Lab1

Student Name:
Mahbod Afarin

Student ID:
862186340

Fall 2021

1- How many total thread blocks do we use?

Answer: the number of threads is 1000 and the block size is 256 which means that each block consists of 256 threads. Therefore, the number of blocks will be:

$$Number\ of\ blocks = \left[\frac{n-1}{Block\ size}\right] + 1 = \left[\frac{1000-1}{256}\right] + 1 = 3 + 1 = \mathbf{4}$$

2- Are all thread blocks full? That is, do all threads in the thread block have data to operate on?

Answer: we have 4 blocks and each block has 256 threads, so the total number of threads is 1024, but the number of operations is 1000. Therefore, there will be 24 additional threads which we do not need them for executing the program, so we have our 256 threads in each thread blocks, but we only need 1000 threads and the remaining 24 threads are not executed. We don't execute those 24 threads because they are additional threads and executing them will lead to wrong result. In the *kernel.cu* we have an *if (i < n)* part, and in that part, we specify that we do not want to execute the additional threads.

3- How can this basic vector add program be improved? (What changes do you think can be made to speed up the code?)

Answer: In order to speed up the code we changed the block size to find the best block size for the program. In the table 1, we can see the execution time of the program for different block size. For this program we only need to consider **Lunching Kernel** time, so the best Lunching Kernel time is 0.000033 and it belongs to the block size 512. Therefore, the best block size for the program is **512**.

Table 1: Execution time of the program for different block size

Block Size	16	32	64	128	256	512	1024	2048
Setting up the problem	0.000079	0.000073	0.000072	0.000073	0.000073	0.000075	0.000074	0.000087
Allocating device variables	2.927385	2.884131	2.957345	2.824187	2.887080	2.857776	2.845001	2.877224
Copying data from host to device	0.000067	0.000072	0.000066	0.000068	0.000065	0.000033	0.000085	0.000044
Launching kernel	0.000066	0.000039	0.000069	0.000068	0.000064	0.000033	0.000046	0.000047
Copying data from device to host	0.000032	0.000018	0.000033	0.00003	0.000032	0.000018	0.000023	0.000023

Total time	2.927629	2.884333	2.957585	2.824426	2.887314	2.857935	2.845229	2.877425

Appendix:

1- kernel.cu

```
#include <stdio.h>


__global__ void VecAdd(int n, const float *A, const float *B, float* C) {

    /***************************************************************************
     *
     * Compute C = A + B
     *   where A is a (1 * n) vector
     *   where B is a (1 * n) vector
     *   where C is a (1 * n) vector
     *
     ***************************************************************************/

    // INSERT KERNEL CODE HERE
    int i = threadIdx.x+blockDim.x*blockIdx.x;
            if(i<n) C[i] = A[i] + B[i];
}


void basicVecAdd(float *A,  float *B, float *C, int n)
{

    // Initialize thread block and kernel grid dimensions ---------------------

    const unsigned int BLOCK_SIZE = 128;

    //INSERT CODE HERE
    dim3 DimGrid((n-1)/BLOCK_SIZE + 1, 1, 1);
    dim3 DimBlock(BLOCK_SIZE, 1, 1); VecAdd<<<DimGrid,DimBlock>>>(n, A, B, C);
}
```

2- main.cu

```
/******************************************************************************
 *cr
 *cr            (C) Copyright 2010 The Board of Trustees of the
 *cr                        University of Illinois
 *cr                         All Rights Reserved
```

```
 *cr
 ******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "kernel.cu"
#include "support.cu"

int main (int argc, char *argv[])
{
    //set standard seed
    srand(217);

    Timer timer;
    cudaError_t cuda_ret;

    // Initialize host variables ----------------------------------------------

    printf("\nSetting up the problem..."); fflush(stdout);
    startTime(&timer);

    float *A_h, *B_h, *C_h;
    float *A_d, *B_d, *C_d;
    size_t A_sz, B_sz, C_sz;
    unsigned VecSize;

    dim3 dim_grid, dim_block;

    if (argc == 1) {
        VecSize = 1000;

      } else if (argc == 2) {
      VecSize = atoi(argv[1]);


      }

      else {
        printf("\nOh no!\nUsage: ./vecAdd <Size>");
        exit(0);
    }

    A_sz = VecSize;
    B_sz = VecSize;
    C_sz = VecSize;
    A_h = (float*) malloc( sizeof(float)*A_sz );
    for (unsigned int i=0; i < A_sz; i++) { A_h[i] = (rand()%100)/100.00; }
```

```
    B_h = (float*) malloc( sizeof(float)*B_sz );
    for (unsigned int i=0; i < B_sz; i++) { B_h[i] = (rand()%100)/100.00; }

    C_h = (float*) malloc( sizeof(float)*C_sz );

    stopTime(&timer); printf("%f s\n", elapsedTime(timer));
    printf("    size Of vector: %u x %u\n  ", VecSize);

    // Allocate device variables ----------------------------------------------------

    printf("Allocating device variables..."); fflush(stdout);
    startTime(&timer);

    //INSERT CODE HERE
    cudaMalloc((void **) &A_d, A_sz*sizeof(float));
    cudaMalloc((void **) &B_d, B_sz*sizeof(float));
    cudaMalloc((void **) &C_d, C_sz*sizeof(float));

    cudaDeviceSynchronize();
    stopTime(&timer); printf("%f s\n", elapsedTime(timer));

    // Copy host variables to device ------------------------------------------

    printf("Copying data from host to device..."); fflush(stdout);
    startTime(&timer);

    //INSERT CODE HERE
    cudaMemcpy(A_d, A_h, A_sz*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(B_d, B_h, B_sz*sizeof(float), cudaMemcpyHostToDevice);

    cudaDeviceSynchronize();
    stopTime(&timer); printf("%f s\n", elapsedTime(timer));

    // Launch kernel  ---------------------------
    printf("Launching kernel..."); fflush(stdout);
    startTime(&timer);
    basicVecAdd(A_d, B_d, C_d, VecSize); //In kernel.cu

    cuda_ret = cudaDeviceSynchronize();
    if(cuda_ret != cudaSuccess) FATAL("Unable to launch kernel");
    stopTime(&timer); printf("%f s\n", elapsedTime(timer));

    // Copy device variables from host ----------------------------------------

    printf("Copying data from device to host..."); fflush(stdout);
    startTime(&timer);

    //INSERT CODE HERE
```

```
    cudaMemcpy(C_h, C_d, C_sz*sizeof(float), cudaMemcpyDeviceToHost);

    cudaDeviceSynchronize();
    stopTime(&timer); printf("%f s\n", elapsedTime(timer));

    // Verify correctness -----------------------------------------------------

    printf("Verifying results..."); fflush(stdout);

    verify(A_h, B_h, C_h, VecSize);


    // Free memory ------------------------------------------------------------

    free(A_h);
    free(B_h);
    free(C_h);

    //INSERT CODE HERE
    cudaFree(A_d); cudaFree(B_d); cudaFree (C_d);
    return 0;

}
```