Lab 1

George Onwubuya

September 5, 2018

1 Main File

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

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

int main(int argc, char**argv) {

   Timer timer;
    cudaError_t cuda_ret;

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

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

    unsigned int n;
    if(argc == 1) {
        n = 10000;
    } else if(argc == 2) {
        n = atoi(argv[1]);
    } else {
        printf("\n    Invalid input parameters!"
           "\n    Usage: ./vecadd               # Vector of size 10,000 is used"
           "\n    Usage: ./vecadd <m>           # Vector of size m is used"
           "\n");
        exit(0);
}

float* A_h = (float*) malloc( sizeof(float)*n );
for (unsigned int i=0; i < n; i++) { A_h[i] = (rand()%100)/100.00 + 1; }

float* B_h = (float*) malloc( sizeof(float)*n );
for (unsigned int i=0; i < n; i++) { B_h[i] = (rand()%100)/100.00 + 1; }

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

stopTime(&timer); printf("%f s\n", elapsedTime(timer));
printf("    Vector size = %u\n", n);


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

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

//INSERT CODE HERE
float* A_d;
cuda_ret = cudaMalloc((void**)&A_d, sizeof(float)*n );
    if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");

float* B_d;
cuda_ret = cudaMalloc((void**)&B_d, sizeof(float)*n );
    if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");

float* C_d;
cuda_ret = cudaMalloc((void**)&C_d, sizeof(float)*n );
    if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");

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
cuda_ret = cudaMemcpy(A_d, A_h, sizeof(float)*n, cudaMemcpyHostToDevice);
    if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to device");

cuda_ret = cudaMemcpy(B_d, B_h, sizeof(float)*n, cudaMemcpyHostToDevice);
    if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to device");
```

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

// Launch kernel ----------------------------------------------------------

printf("Launching kernel..."); fflush(stdout);
startTime(&timer);

//INSERT CODE HERE
const unsigned int THREADS_PER_BLOCK = 256;
const unsigned int numBlocks = (n-1)/THREADS_PER_BLOCK + 1;
dim3 gridDim(numBlocks, 1, 1), blockDim(THREADS_PER_BLOCK, 1, 1);

vecAddKernel<<<numBlocks, THREADS_PER_BLOCK>>>(A_d, B_d, C_d, n);

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
cuda_ret = cudaMemcpy(C_h, C_d, sizeof(float)*n, cudaMemcpyDeviceToHost);
    if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory from device");

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

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

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

verify(A_h, B_h, C_h, n);

// 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;

}
```

2 Kernel File

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

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

    // Calculate global thread index based on the block and thread indices ----

    //INSERT KERNEL CODE HERE
    int i = blockDim.x*blockIdx.x+threadIdx.x;

    // Use global index to determine which elements to read, add, and write ---

    //INSERT KERNEL CODE HERE

    if (i<n) C[i] = A[i] + B[i];

}
```

3 Output File

```
Setting up the problem...0.000044 s
    Vector size = 1000
Allocating device variables...0.398394 s
Copying data from host to device...0.012909 s
Launching kernel...0.006373 s
Copying data from device to host...0.000021 s
Verifying results...TEST PASSED
```

```
Setting up the problem...0.000246 s
    Vector size = 10000
Allocating device variables...0.375915 s
Copying data from host to device...0.006982 s
Launching kernel...0.002321 s
Copying data from device to host...0.000048 s
Verifying results...TEST PASSED
```

```
Setting up the problem...0.002300 s
    Vector size = 100000
Allocating device variables...0.377880 s
Copying data from host to device...0.000241 s
Launching kernel...0.002300 s
Copying data from device to host...0.000308 s
Verifying results...TEST PASSED
```

```
Setting up the problem...0.019750 s
    Vector size = 1000000
Allocating device variables...0.372675 s
Copying data from host to device...0.001425 s
Launching kernel...0.000158 s
Copying data from device to host...0.002088 s
Verifying results...TEST PASSED
```

4 Output Analysis

Time Taken for Each Sub Process			
Elements(n)	Setting Up(s)	Device Variables(s)	Kernel Launch(s)
1000	0.000044	0.398394	0.006373
10000	0.000246	0.375915	0.002321
100000	0.0023003	0.377880	0.002300
1000000	0.019750	0.372675	0.000158

Time Taken for Each Sub Process		
Elements(n)	Host to Device(s)	Device to Host(s)
1000	0.012909	0.000021
10000	0.006982	0.000048
100000	0.000241	0.000308
1000000	0.001425	0.002088