CSE 5441
Programming Assignment 4
Submitted by: Biplob Biswas (biswas.102)

In this assignment, a CUDA implementation of AMR is applied on “*testgrid_400_12206*” data file having 12206 boxes.

a) In CUDA implementation, one thread per box is set to compute and update the temperature. 6 different values for the number of threads per block are chosen and the number of blocks was calculated following this formula:

Number of Blocks = (Box Count + Threads per Block -1)/Threads per Block

Two different kernels are implemented, one for temperature calculation and the other to update temperature to the corresponding box. The first one contains the floating point operations.

The number of floating-point operations (flop) = IterationCount * flopPerIteration
= 794818 * 204810
= 162786674580

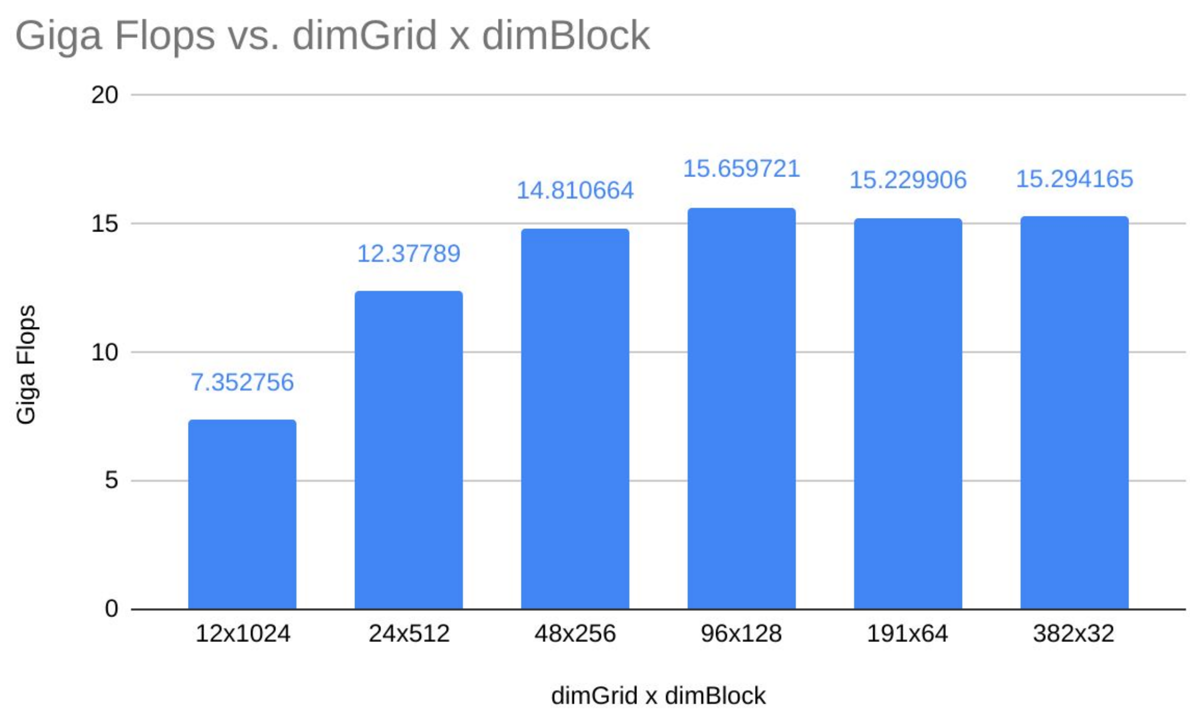
The time for this calculation was measured using the `clock()` system call on host, and using the functions `cudaEventCreate`, `cudaEventRecord` and `cudaEventElapsedTime` on the device. The time indicates the time spent only on new temperature calculations which involve flops. The measured results follow:

For Serial implementation,

Time taken on host (ms) = 133090.000000
Giga FLOPS/Sec on Host = 1.223132

For CUDA implementation on Device:

Number of Blocks (dimGrid)	**Threads per Block (dimBlock)**	**Time (ms)**	**Giga Flops/Sec**
12	1024	22139.543997	7.352756
24	512	13151.407155	12.377890
48	256	10991.180066	14.810664
96	128	10395.247646	15.659721
191	64	10688.619451	15.229906
382	32	10643.711377	15.294165



b) The serial version finishes the calculation in 133090 milliseconds, while the best performance from CUDA version with 96x128 grid configuration finishes in 10395.247646 milliseconds, which is 12.8 times faster.

c) **Description of Changes:**
The CUDA implementation required memory transfer from host to device. My serial implementation maintained a structure of a box having a pointer to the neighbor boxes.

```
typedef struct
{
    int id;
    int commonEdgeLength;
} Neighbor;
```

```
typedef struct
{
    int id;
    int upperLeftX, upperLeftY, height, width;
    int neighborCountInDir[4];
    int *neighborsInDir[4];
    double temp;
    int totalNeighborsCount;
    Neighbor *allNeighbors;  //Requires the extra mechanism to copy to device
    int uncommonEdgeLength;
    int perimeter;
} Box;
```

As a result, just copying the box structure using **cudaMemcpy** function is not enough as it can't transfer dynamically allocated memory to the neighbor box pointer. So, in addition to copying boxes, their neighbors were sent separately and then another **cudaMemcpy** hooked neighbors' to their corresponding box structure. Here is part of that code -

```
//Copying Box structure only
cudaMemcpy(deviceBoxes, boxes, boxes_memsize, cudaMemcpyHostToDevice);
for (int i=0; i<boxCount; i++)
{
    //Copying neighbors
    cudaMemcpy(dboxNeighbors[i], boxes[i].allNeighbors, boxes[i].totalNeighborsCount *
    sizeof(Neighbor), cudaMemcpyHostToDevice);


   //Hooking neighbors to boxes
   cudaMemcpy(&(deviceBoxes[i].allNeighbors), &dboxNeighbors[i], sizeof(Neighbor*),
   cudaMemcpyHostToDevice);
}
```

The following table shows how two kernels (one for new temperature calculation and the other to update the resulting temperature to the corresponding box) were used in CUDA implementation and the change from serial implementation. Inside the kernel, the particular thread (as boxId) corresponding to a box is identified by this formula-

```
int boxId = blockIdx.x*blockDim.x + threadIdx.x;
```

Serial on host

```
for (iteration = 0; !hasConverged(); iteration++)
{
  //Calculating new temperature
   for (int j = 0; j < boxCount; j++) {
      newTemp[j] = getNewTemp(j);
   }

   //Updating temperature
   for (int j = 0; j < boxCount; j++) {
      boxes[j].temp = newTemp[j];
   }
}
```

CUDA

```
for (iteration = 0; !hasConverged(); iteration++)
{
  //Calculating new temperature on device
   calcNewTemp<<<dimGrid, dimBlock>>>(deviceBoxes, newDeviceTemp,
AFFECT_RATE, boxCount);

   //Updating temperature on device
   updateTemp<<<dimGrid, dimBlock>>>(deviceBoxes, newDeviceTemp, boxCount);

   //Bringinging new temperature from device to host
   cudaMemcpy(newTemp, newDeviceTemp, temp_memsize,
cudaMemcpyDeviceToHost);

   //Updating temperature on host for convergence check
   for (j = 0; j < boxCount; j++)
   {
       boxes[j].temp = newTemp[j];
   }
}
```

Since each box temperature was calculated by a dedicated thread, it improved the performance dramatically.

Compilation:

```
[biplob@o0649 cse5441_lab4]$ make
icc -O3 -lrt biswas_biplob_serial.c -o serial
nvcc -O3 -lrt biswas_biplob_lab4p1.cu -o lab4p1
```

Here is the output for serial implementation:

```
[biplob@o0649 cse5441_lab4]$ time ./serial 0.02 0.02 < testgrid_400_12206
Number of flop per iteration: 204810
Total number of Giga flop in Device: 162.786675
***************************************************************************
dissipation converged in 794818 iterations,
   with max DSV = 0.0846372 and min DSV = 0.0829445
   affect rate  = 0.020000; epsilon = 0.020000

Time taken on host (ms) = 133090.000000
Giga FLOPS per sec on Host = 1.223132
***************************************************************************
real	3m8.381s
user	3m6.987s
sys	0m1.333s
```

Here is one of the outputs for CUDA implementation:

```
[biplob@o0649 cse5441_lab4]$ time ./lab4p1 0.02 0.02 < testgrid_400_12206
Number of flop per iteration: 204810
Total number of Giga flop in Device: 162.786675
***************************************************************************
dissipation converged in 794818 iterations,
   with max DSV = 0.0846372 and min DSV = 0.0829445
   affect rate  = 0.020000; epsilon = 0.020000

Time taken on device (ms) = 10395.247646
Giga FLOPS per sec on device = 15.659721
***************************************************************************
real	1m44.730s
user	1m37.369s
sys	0m7.298s
```