High-Performance Computing

OpenACC - Kmeans

Anand Kamble
Department of Scientific Computing
Florida State University

1 Introduction

In this assignment, we are using OpenACC for parallelization. We are using the `pgc++` compiler for compiling this project.

2 Implementation

2.1 Data Transfer

The first step in the implementation is to transfer the input image data from the host (CPU) memory to the device (GPU) memory. This is achieved using the `#pragma acc data` directive, which specifies that the `hostDataBuf` array should be copied to the device before the parallel region, and copied back to the host after the parallel region.

```
#pragma acc data copyin(hostDataBuf[0:N*3])
{
    // Parallel region
}
```

2.2 Assigning Pixels to Clusters

A key step in the K-means algorithm is assigning each pixel to the nearest centroid based on the Euclidean distance. This operation is parallelized using the following OpenACC directive:

```
#pragma acc parallel loop gang vector reduction(+ : groupColorSum[ : k * 3])
    reduction(+ : groupCount[ : k]) present(colors, generators, groupColorSum,
    groupCount)
for (int i = 0; i < N; i++) {
    double minDist = INFINITY;
    int minIndex = 0;

    for (int j = 0; j < k; j++) {
        double dist = euclideanDistance(&colors[i], &generators[j]);
        if (dist < minDist) {
            minDist = dist;
            minIndex = j;
        }
    }

    groupColorSum[minIndex * 3] += colors[i].r;
    groupColorSum[minIndex * 3 + 1] += colors[i].g;
    groupColorSum[minIndex * 3 + 2] += colors[i].b;
    groupCount[minIndex]++;
}
```

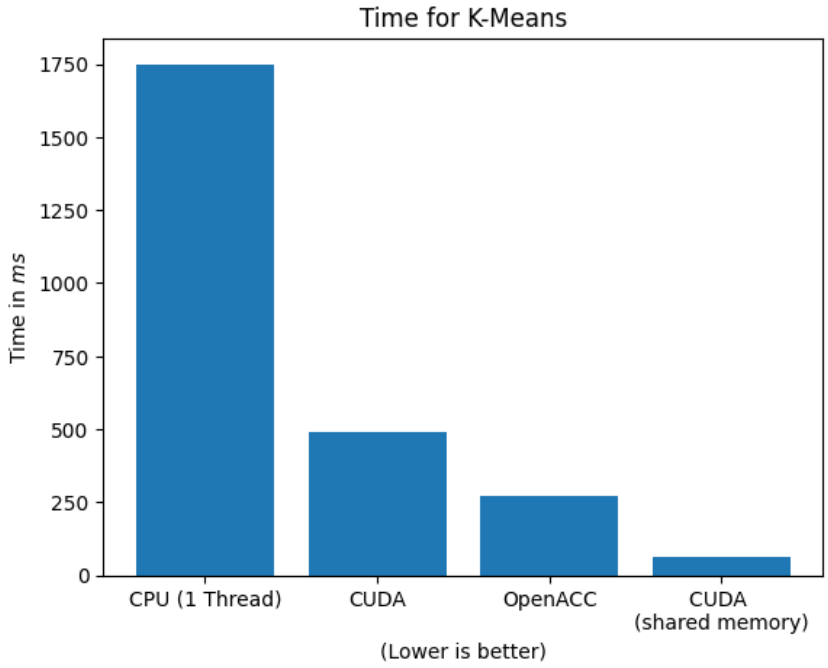
2.2.1 Updating Pixel Values

After the iterative clustering process, the final step is to update the pixel values in the image data based on the nearest centroid. This operation is parallelized using the following OpenACC directive:

```
#pragma acc parallel loop gang vector present(colors, generators)
for (int i = 0; i < N; i++) {
    double minDist = INFINITY;
    int minIndex = 0;

    for (int j = 0; j < k; j++) {
        double dist = euclideanDistance(&colors[i], &generators[j]);
        if (dist < minDist) {
            minDist = dist;
            minIndex = j;
        }
    }

    colors[i] = generators[minIndex];
}
```

3 Results

After the successful execution of the program, we get the following results:



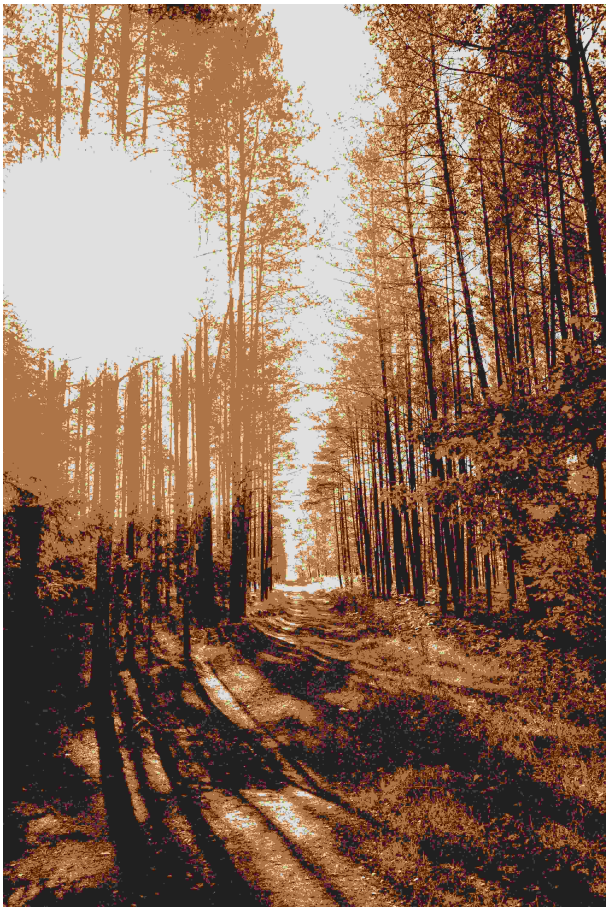
We can see that OpenACC does very good optimization for our program, but it is not as fast as the highly optimized version of CUDA which uses shared memory.

4 Profiling

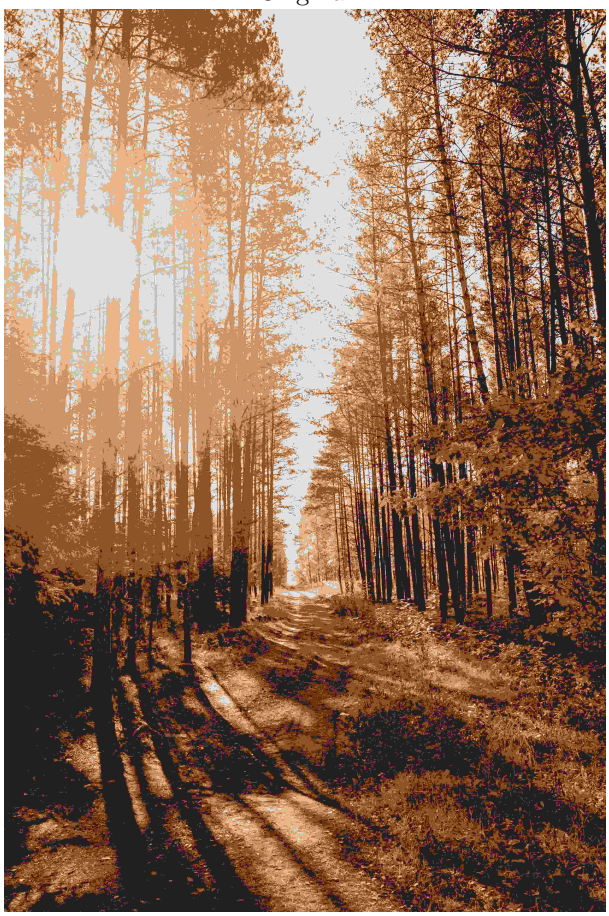
Here are the results of the profiling done using `nsys`. We can see that most of the time spent is for `cuStreamSynchronize`.

```
** OS Runtime Summary (osrt_sum):

 Time (%)  Total Time (ns)  Num Calls      Avg (ns)         Med (ns)        Min (ns)       Max (ns)      StdDev (ns)           Name
 --------  ---------------  ---------  ---------------  ---------------  -------------  -------------  -------------  ----------------------
     69.9    2,557,904,210          1  2,557,904,210.0  2,557,904,210.0  2,557,904,210  2,557,904,210            0.0  getc
     13.8      506,750,195         15     33,783,346.3      8,942,798.0         26,941    100,484,956   43,668,021.6  poll
     13.7      500,460,872          1    500,460,872.0    500,460,872.0    500,460,872    500,460,872            0.0  pthread_cond_timedwait
      1.1       41,744,304        475         87,882.7          6,480.0          1,009      9,998,443      739,666.5  ioctl
      1.0       36,824,974          8      4,603,121.8          2,932.0          1,070     36,448,016   12,867,890.5  fclose
      0.4       13,103,404         33        397,072.8          2,513.0          1,040     11,377,940    1,985,006.2  fopen
      0.0          809,663         27         29,987.5          5,168.0          3,411        490,458       92,789.8  mmap64
      0.0          543,110          5        108,622.0         81,358.0         36,740        200,946       73,605.4  pthread_create
      0.0          491,792        193          2,548.1          2,358.0          1,487         18,419        1,676.4  fread
      0.0          421,445          9         46,827.2         23,452.0         13,831        102,149       39,092.3  sem_timedwait
      0.0          321,869         67          4,804.0          4,563.0          1,868         11,352        1,247.7  fwrite
      0.0          220,570         19         11,608.9          3,789.0          1,307        140,930       31,456.9  mmap
      0.0          161,291         45          3,584.2          2,878.0          1,209         10,720        2,005.0  open64
      0.0          158,669          1        158,669.0        158,669.0        158,669        158,669            0.0  pthread_cond_wait
      0.0           67,379          2         33,689.5         33,689.5         25,507         41,872       11,571.8  socket
      0.0           57,352          1         57,352.0         57,352.0         57,352         57,352            0.0  connect
      0.0           33,131          9          3,681.2          3,123.0          1,403          9,924        2,567.5  munmap
      0.0           32,146          1         32,146.0         32,146.0         32,146         32,146            0.0  fgets
      0.0           31,876          8          3,984.5          3,577.0          1,041          7,359        2,335.8  open
      0.0           12,830          7          1,832.9          1,355.0          1,148          3,469          830.3  write
      0.0           11,286          2          5,643.0          5,643.0          5,059          6,227          825.9  fflush
      0.0            8,482          3          2,827.3          3,291.0          1,849          3,342          847.6  pipe2
      0.0            6,955          4          1,738.8          1,828.0          1,018          2,281          532.6  read
      0.0            6,203          3          2,067.7          1,957.0          1,270          2,976          858.4  pthread_cond_broadcast
      0.0            2,638          1          2,638.0          2,638.0          2,638          2,638            0.0  bind
      0.0            1,705          1          1,705.0          1,705.0          1,705          1,705            0.0  listen

Processing [./report1.sqlite] with [/opt/nvidia/hpc_sdk/Linux_x86_64/24.3/profilers/Nsight_Systems/host-linux-x64/reports/cuda_api_sum.py]...

** CUDA API Summary (cuda_api_sum):

 Time (%)  Total Time (ns)  Num Calls    Avg (ns)      Med (ns)     Min (ns)    Max (ns)    StdDev (ns)           Name
 --------  ---------------  ---------  ------------  ------------  ----------  ----------  -------------  ----------------------
     87.8      146,236,754         71  2,059,672.6       2,658.0         333  13,317,737  4,827,862.2  cuStreamSynchronize
      6.3       10,475,363          2  5,237,681.5   5,237,681.5       7,989  10,467,374  7,395,902.1  cuMemcpyHtoDAsync_v2
      5.2        8,701,079          1  8,701,079.0   8,701,079.0   8,701,079   8,701,079          0.0  cuMemcpyDtoHAsync_v2
      0.3          534,464          1    534,464.0     534,464.0     534,464     534,464          0.0  cuMemAllocHost_v2
      0.2          409,521          9     45,502.3      49,942.0       1,668      96,340     41,356.9  cuMemAlloc_v2
      0.1          146,914         43      3,416.6       2,560.0       2,139      20,989      3,189.2  cuLaunchKernel
      0.1          137,550          1    137,550.0     137,550.0     137,550     137,550          0.0  cuModuleLoadDataEx
      0.0            1,779          3        593.0         344.0         132       1,303        623.9  cuCtxSetCurrent
```

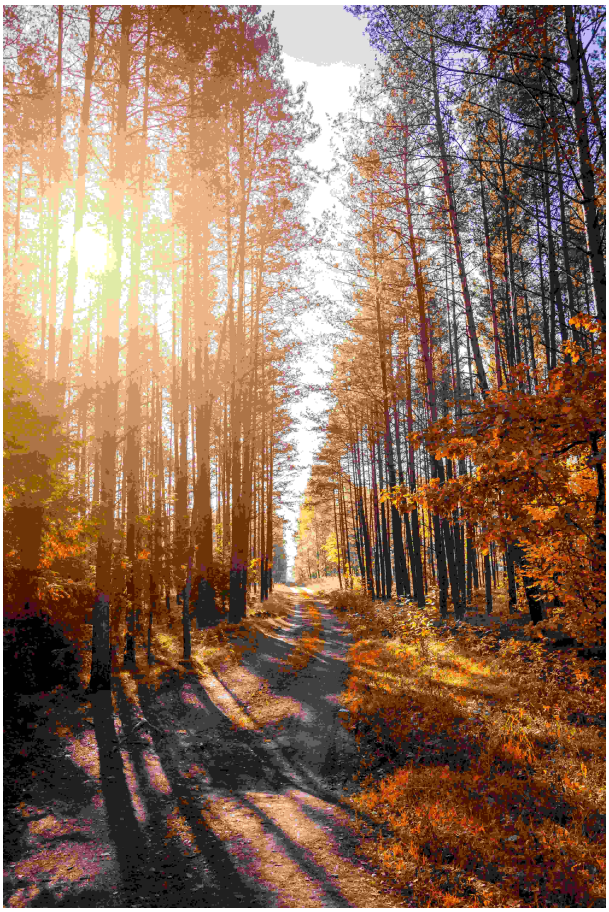
Original



k = 3



k = 5



k = 8