```
This software is dervied from Professor Wei-keng Liao's parallel k-means
clustering code obtained on November 21, 2010 from
 http://users.eecs.northwestern.edu/~wkliao/Kmeans/index.html
(http://users.eecs.northwestern.edu/~wkliao/Kmeans/simple_kmeans.tar.gz).

With his permission, I am publishing my CUDA implementation based on his code
under the open-source MIT license. See the LICENSE file for more details.

For starters, run the benchmark.sh script to see how fast this code runs.

It's pretty fast! Depending on your hardware, data set, and k, you should see
dramatic improvements in performance over CPU implementations.

On an 8-core 2.4 GHz Intel Xeon E5620 machine with an NVIDIA Tesla C1060 card,
the CUDA implementation runs almost 50 times faster than the sequential version
on the color17695.bin data set (for k = 128)!

Here's some sample output for the same data set on an 8-core 2.67 GHz Intel Core
i7 920 machine with an NVIDIA GeForce GTX 480 card. For k = 128, the speedup is
over 75x!

--------------------------------------------------------------------------------
k =   2  seqTime = 0.2870s  ompTime = 0.1497s  cudaTime = 0.1483s  speedup = 1.9x
k =   4  seqTime = 0.0945s  ompTime = 0.0351s  cudaTime = 0.0652s  speedup = 1.4x
k =   8  seqTime = 0.4379s  ompTime = 0.1439s  cudaTime = 0.0919s  speedup = 4.7x
```

```
k =  16  seqTime = 2.0539s  ompTime = 0.6628s  cudaTime = 0.1611s  speedup = 12.7x
k =  32  seqTime = 3.0699s  ompTime = 1.0319s  cudaTime = 0.1461s  speedup = 21.0x
k =  64  seqTime = 8.4675s  ompTime = 3.1946s  cudaTime = 0.2487s  speedup = 34.0x
k = 128  seqTime = 22.9694s ompTime = 10.0000s cudaTime = 0.3031s  speedup = 75.7x
--------------------------------------------------------------------------------

The CUDA implementation may need some tweaking to work with larger data sets,
but the basic functionality is there. I've optimized the code with the general
assumption that k and the number of clusters will be relatively small compared
to the number of data points.

In fact, you may run into problems if k is too big or if your data has a large
number of dimensions because there won't be enough space in block shared memory
to hold all the clusters (the C1060 and GTX 480 have 16 and 48 KiB of block
shared memory, respecitvely). If you hit this limitation, you should be able to
get around it easily. Do the following:

1) Run 'make clean'
2) Edit the Makefile. Find the line at the top of the file that looks like this:

    CFLAGS = $(OPTFLAGS) $(DFLAGS) $(INCFLAGS) -DBLOCK_SHARED_MEM_OPTIMIZATION=1

3) Set -DBLOCK_SHARED_MEM_OPTIMIZATION=0
4) Run 'make cuda' and try again

Please don't hesitate to contact me with any questions you may have. I'd love to
help you out if you run into a problem. Of course, the more information you give
me about your CUDA hardware and your data set (number of data points,
dimensionality, number of clusters), the more helpful I can be.

The original README, with some additions, is reproduced below.

Cheers!

Serban Giuroiu
http://serban.org

# ------------------------------------------------------------------------------

Parallel K-Means Data Clustering

The software package of parallel K-means data clustering contains the
followings:

```

```
  * A parallel implementation using OpenMP and C
  * A parallel implementation using MPI and C
  * A parallel implementation using CUDA and C
  * A sequential version in C

To compile:
Although I used Intel C compiler, icc, version 7.1 during the code
development, there is no particular features required except for OpenMP.
Thus, the implementation should be fairly portable. Please modify
Makefile to change the compiler if needed.

You will need the NVIDIA CUDA toolkit, which contains nvcc, to build the CUDA
version. It works fine in concert with gcc.

To run:
  * The Makefile will produce executables
     o "omp_main" for OpenMP version
     o "mpi_main" for MPI version
     o "cuda_main" for CUDA version
     o "seq_main" for sequential version

  * The list of available command-line arguments can be obtained by
    running -h option
     o For example, running command "omp_main -h" will produce:
       Usage: main [switches] -i filename -n num_clusters
             -i filename    : file containing data to be clustered
             -b             : input file is in binary format (default no)
             -n num_clusters: number of clusters (K must > 1)
             -t threshold   : threshold value (default 0.0010)
             -p nproc       : number of threads (default system allocated)
             -a             : perform atomic OpenMP pragma (default no)
             -o             : output timing results (default no)
             -d             : enable debug mode

Input file format:
The executables read an input file that stores the data points to be
clustered. A few example files are provided in the sub-directory
./Image_data. The input files can be in two formats: ASCII text and raw
binary.

  * ASCII text format:
    o Each line contains the coordinates of a single data point
    o The number of coordinates must be equal for all data points
  * Raw binary format:
```

```
    o There is a header of 2 integers.
    o The first 4-byte integer must be the number of data points.
    o The second integer must be the number of coordinates.
    o The rest of the file contains the coordinates of all data
      points and each coordinate is of type 4-byte float.

Output files: There are two output files:
  * Coordinates of cluster centers
    o The file name is the input file name appended with ".cluster_centres".
    o It is in ASCII text format.
    o Each line contains an integer indicating the cluster id and the
      coordinates of the cluster center.
  * Membership of all data points to the clusters
    o The file name is the input file name appended with ".membership".
    o It is in ASCII text format.
    o Each line contains two integers: data point index (from 0 to
      the number of points) and the cluster id indicating the membership of
      the point.

Limitations:
    * Data type -- This implementation uses C float data type for all
      coordinates and other real numbers.
    * Large number of data points -- The number of data points cannot
      exceed 2G due to the 4-byte integers used in the programs. (But do
      let me know if it is desired.)


Wei-keng Liao (wkliao@ece.northwestern.edu)
EECS Department
Northwestern University

Sep. 17, 2005

```