A NEW TOOL FOR CLASSIFICATION OF SATELLITE IMAGES AVAILABLE FROM GOOGLE MAPS: EFFICIENT IMPLEMENTATION IN GRAPHICS PROCESSING UNITS

Sergio Bernabé and Antonio Plaza

Hyperspectral Computing Laboratoy
Department of Technology of Computers and Communications
University of Extremadura, Avda. de la Universidad s/n, E-10071 Cáceres, Spain
E-mail: sbernabe@alumnos.unex.es, aplaza@unex.es

1. INTRODUCTION

The wealth of satellite imagery [1] available in web mapping service applications such as Google Maps[1], which now provides high-resolution satellite images from many locations around the Earth, has opened the appealing perspective of performing classification and retrieval tasks via programming libraries such as SwingX-WS[2]. In fact, the introduction of Google's mapping engine prompted a worldwide interest in satellite imagery exploitation. The combination of an easily pannable and searchable mapping and satellite imagery tool such as Google Maps with advanced image classification and retrieval features has the potential to significantly expand the functionalities of the tool and also to allow end-users to extract relevant information from a massive and widely available database of satellite images (the Google Maps service is free for non-commercial use).

In this paper, we describe a new tool [2] which allows an unexperienced user to perform unsupervised classification of satellite images obtained via Google Maps by means of the well-known k-means clustering algorithm [3], which can be followed by spatial post-processing based on majority voting. The classification stage has been implemented in parallel using commodity graphic processing units (GPUs) [4], which are specialized hardware cards that are nowadays widely available in standard PCs. Processing examples reported in this work include analyses of consensus or agreement in the classification achieved by our GPU implementation with regards to an alternative implementation of the k-means clustering algorithm available in commercial software (ITT Visual Information Solutions ENVI[3]). In addition, our parallel version of the k-means algorithm –implemented in NVidia^{TM} GPUs using the compute unified device architecture (CUDA)[4]– is shown to be more than 30 times faster than the serial version. This opens the patch for exciting new developments and potentials in efficient processing of large databases of satellite images, such as those available from Google Maps engine and used in this work for demonstration.

2. CLASSIFICATION SYSTEM FOR GOOGLE MAPS IMAGES

Our classification system for Google Maps images consists of the integration of the different software modules developed (unsupervised classifiers) with the functionalities provided by the SwingX-WS libraries, in the form of a general-purpose desktop application [2]. For this purpose, we have resorted to the Java programming language (see Fig. 1), which is a multi-platform environment that simplifies porting of our tool to different environments. Specifically, we have resorted to the NetBeans platform[5], which allows applications to be developed from a set of modular software components called modules. In this framework, applications can install modules dynamically and any application can include the update center module to allow users of the application to download digitally-signed upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again. The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application.

With the above ideas in mind, Fig. 2 shows a view of the developed tool. As shown by the figure, the tool allows selecting an area to be classified, obtaining classification results in unsupervised fashion, retaining the classified area at different zoom levels (although the classification is obtained at the maximum zoom level), and other functionalities such as spatial post-processing

[1] http://maps.google.com
[2] https://swingx-ws.dev.java.net
[3] http://www.ittvis.com/ProductServices/ENVI.aspx
[4] http://www.nvidia.com/object/cuda_home_new.html
[5] http://netbeans.org

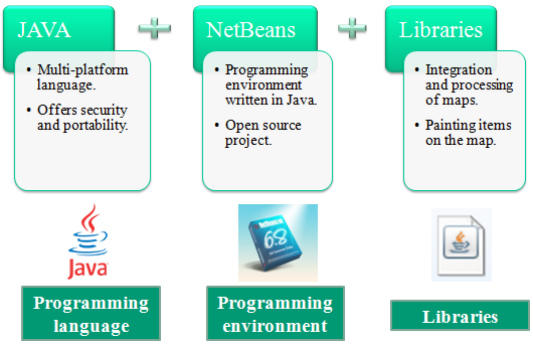


Fig. 1. Different software modules used for the development of our application.

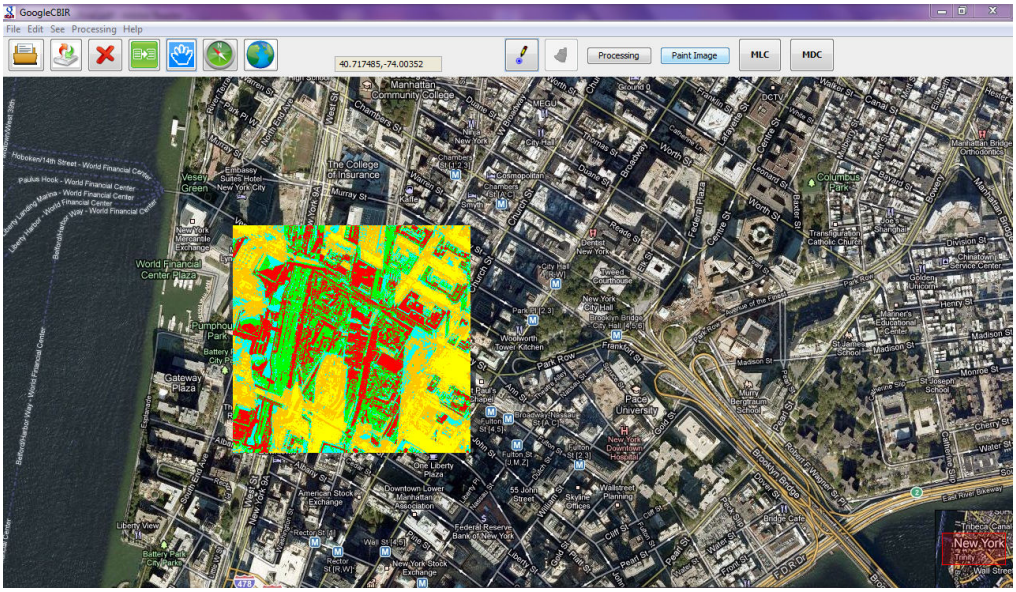


Fig. 2. A view of the developed classification tool for Google Map images: unsupervised classification result provided by our implementation of k-means for an area located in the World Trade Center of New York City.

of obtained results for increased spatial consistency, managing of the resulting classification and extracted satellite images, loading/storing of results via file logs which can be saved in a database, automatic positioning in any latitude and longitude coordinates in the entire Google Maps database, overlaying of classification results with different views (satellite, map, hybrid), etc. Overall, we feel that the developed tool incorporates interesting additional functionalities to the Google Maps engine (particularly in the possibility of better exploiting the satellite images available from this tool in different application domains).

3. GPU IMPLEMENTATION

GPUs can be abstracted in terms of a *stream model*, under which all data sets are represented as streams (i.e., ordered data sets) [4]. Algorithms are constructed by chaining so-called *kernels*, which operate on entire streams, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks, as displayed in Fig. 3(a), where each block is composed by a group of threads which share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. This figure also displays how each kernel is executed as a grid of blocks of threads. On the other hand, Fig. 3(b) shows the execution model in the GPU, which can be seen as a set of multiprocesors. In each clock cycle each processor of the multiprocessor executes the same instruction but operating on multiple data streams. Each processor has access to a local shared memory and also to local

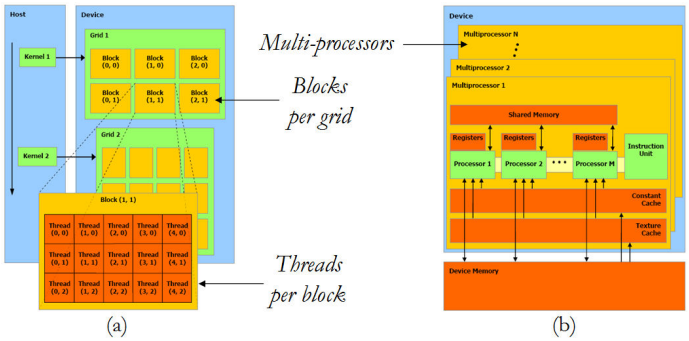


Fig. 3. Schematic overview of a GPU architecture. (a) Threads, blocks and grids. (b) Execution model in the GPU.

cache memories in the multiprocessor, while the multiprocessors have access to the global GPU (device) memory.

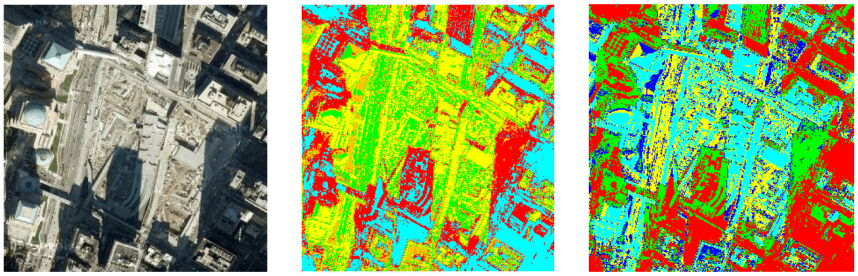
In the following we describe the different steps that we followed for the development of the GPU version of the k-means algorithm. In our implementation, pixels are distributed in the form of vectors which store the values of the red, green, blue and saturation components. In the following, we assume that images fit in the global GPU memory. The k-means algorithm calculates the euclidean distance from each pixel vector to the closest cluster center (centers are initialized randomly). In our CUDA implementation, we assigned the calculation of the distance of each pixel of the image to an independent processing thread. This way, each thread calculates of the distance between the values of each pixel and the nearest cluster center. Fig. 4 illustrates the *kernel* that performs this operation. Next, we recalculate the centers of each cluster and reassign new pixels to each of the clusters until convergence. This part has not been parallelized in the GPU, mainly because for a small number of clusters this computation can be performed in the CPU without representing a dominant factor in the overall time of the solution.

```
__constant__ centroid constData[SIZE_DATA];

__global__ void KMeans_kernel(pixel* g_idata, centroid *g_centroids, int numClusters,
                              unsigned long numElements) {
    unsigned long valindex = blockIdx.x*blockDim.x + threadIdx.x; //id. thread
    int k,myCentroid;
    unsigned long minDistance=0xFFFFFFFF;

    for(k=0;k<numClusters;k++){
        if(abs((long)(g_idata[valindex].value - constData[k].value/*g_centroids*/))<minDistance){
            minDistance=abs((long)(g_idata[valindex].value - constData[k].value));
            myCentroid=k;
        }
        g_idata[valindex].centroid= myCentroid;
        __syncthreads();
    }
}
```

Fig. 4. Main CUDA kernel developed for the implementation of k-means algorithm in GPUs.

4. EXPERIMENTAL RESULTS

In this section, we perform an experimental validation of our developed system using satellite images obtained from Google Maps. The experimental validation of k-means has been conducted by comparing the results provided by our GPU implementation with those available in a well-known commercial software package: the ENVI package distributed by ITT Visual Information Solutions. In our tests, we adopt exactly the same parameters when running our implementation and the one available in ENVI software, comparing the results in terms of the agreement between both solutions and their computational performance. Our experiments are conducted in a specific case study focused on classification of satellite images available from the World Trade Center (WTC) area in New York City. Fig 5(a) shows a satellite image extracted from Google Maps engine. The resolution of the image is quite high, with approximately 5 meters per pixel. Fig. 5(b) shows the unsupervised classification result provided by our GPU implementation as compared to the result provided by ENVI's k-means in Fig. 5(c). As shown by Fig. 5, the color labels for our implementation and the one available in ENVI are different, but the classification maps are very similar. In both cases, the parameters for both algorithms have been set to exactly the same values, with the number of clusters set empirically to five. Table 4 reports the classification agreement (in percentage) measured after comparing our k-means



(a) (b) (c)

Fig. 5. (a) Satellite image over the World Trade Center area. (b) Unsupervised classification result provided by our GPU implementation of k-means. (c) Unsupervised classification result provided by ENVI's implementation of k-means.

Shadows #1 [blue in Fig. 5(b)]	Shadows #2 [red in Fig. 5(b)]	Urban areas #1 [yellow in Fig. 5(b)]	Urban areas #2 [green in Fig. 5(b)]	Urban areas #3 [orange in Fig. 5(b)]	Overall agreement
78.26	95.49	85.56	90.65	97.39	89.47

Table 1. Classification agreement (in percentage) after comparing the classification map provided by our GPU implementation of k-means with the classification map obtained by ENVI's k-means for the image in Fig. 5(a).

classification map with the one obtained by ENVI (assuming the latter as the reference). As shown by Table 4, the agreement between both maps is quite high.

Finally, we analyze the computational performance of the developed GPU implementation with regards to its CPU (serial) version. In this work, we have used two different CPU-GPU configurations. In the first one, we use an Intel Core i7 920 CPU with the Ubuntu 9.04 Linux operating system and the NVidia Tesla C1060 GPU[6]. In the second one, we use an Intel Core 2 Duo P8700 2.53Ghz CPU with the Windows 7 operating system and an NVidia GeForce 9400M GPU[7]. Table 2 shows the execution times achieved for each of the CPU-GPU configurations used, as well as the speedups achieved for different image sizes and number of clusters. An important observation is that, as we increase the image size and number of clusters, the speedup achieved by the GPUs tends to be more significant. For instance, the implementation in the Tesla C1060 GPU achieves a speedup of about 37x with regards to the CPU version for 1024×1024 image size and 128 clusters. However, the implementation in the GeForce 9400M GPU saturates for certain image sizes, achieving a speedup of about 10x in the best case. In the final paper, we will analyze these scalability issues further and provide a more detailed experimentation with additional case studies.

Parameters considered		GeForce 9400M GPU		Tesla C1060 GPU	
Image size	Number of clusters	Time	Speedup	Time	Speedup
512×512	5	0.252	3.26x	0.145	5.67x
512×512	64	0.496	7.60x	0.210	17.95x
512×512	128	0.764	10.29x	0.268	29.33x
1024×1024	64	3.582	6.18x	0.715	30.97x
1024×1024	128	4.376	8.76x	1.044	36.69x

Table 2. Processing times (in seconds) and speedups achieved with regards to the corresponding CPU for different GPU implementations of the k-means algorithm (using different image sizes and number of clusters).

5. REFERENCES

[1] D. A. Landgrebe, *Signal theory methods in multispectral remote sensing*, John Wiley and Sons, Hoboken, NJ, 2003.

[2] S. Bernabe and A. Plaza, "A new system to perform unsupervised and supervised classification of satellite images from google maps," *Proc. SPIE Conference on Satellite Data Compression, Communications, and Processing*, vol. 7810, pp. 1–10, 2010.

[3] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, vol. 28, pp. 100–108, 1979.

[4] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, "Parallel morphological endmember extraction using commodity graphics hardware," *IEEE Geoscience and Remote Sensing Letters*, vol. 43, pp. 441–445, 2007.

[6] http://www.nvidia.com/object/product_tesla_c1060_us.html

[7] http://www.nvidia.com/object/product_geforce_9400m_g_us.html