

# THE GPU HYPERBOLIC SMOOTHING CLUSTERING METHOD

**Marcelo Signorelli Mendes**

PESC – COPPE – UFRJ ([www.cos.ufrj.br](http://www.cos.ufrj.br))  
marcelosm@cos.ufrj.br

**Sergio Barbosa Villas-Boas**

PESC – COPPE – UFRJ ([www.cos.ufrj.br](http://www.cos.ufrj.br))  
sbvb@cos.ufrj.br

**Ricardo Farias**

PESC – COPPE – UFRJ ([www.cos.ufrj.br](http://www.cos.ufrj.br))  
rfarias@cos.ufrj.br

**Adilson Elias Xavier**

PESC – COPPE – UFRJ ([www.cos.ufrj.br](http://www.cos.ufrj.br))  
adilson@cos.ufrj.br

## ABSTRACT

The traditional approach to solve minimum sum-of-squares clustering problems takes it as a non-differentiable problem that has the dimension similar to the input data (which can be very large). The Hyperbolic Smoothing Clustering Method – HSCM (Xavier, 2010) introduces a rupture improvement that turns the clustering problem into a much simpler sequence of low dimension differentiable unconstrained optimization problems. The dimension magnitude of HSCM is similar to the number of clusters, not the input data.

We propose the GPU Hyperbolic Smoothing Clustering Method (GPU-HSCM) that explores the parallelizable parts of the HSCM. These parallelizable parts are implemented on GPU, based on the NVIDIA's CUDA Architecture. Experimental results confirm the performance gain, as expected.

**KEYWORDS:** mathematical programming, cluster analysis, min-sum-min problems, non-differentiable problems, GPU.

# 1 Introduction

Cluster analysis deals with the problems of classification of a set of patterns or observations, in general represented as points in a multidimensional space, into clusters, following two basic and simultaneous objectives: patterns in the same clusters must be similar to another (homogeneity objective) and different from patterns of other clusters (separation objective), see (Hartigan 1973) and (Späth 1980).

A particular clustering problem formulation is considered. Among many criteria used in cluster analysis, the most natural, intuitive and frequently adopted criterion is the minimum sum-of-squares clustering (MSSC). It is a criterion for both the homogeneity and the separation objectives. The minimum sum-of-squares clustering (MSSC) formulation produces a mathematical problem of global optimization. It is both a non-differentiable and a non-convex mathematical problem, with a very large number of local minima. It is a very important numeric problem. Its solution demands strong computing power. This is the problem that takes observation input data (can be a large set) and defines the clusters of it, that is, the groups where the data inside it share some common property (e.g. points near each other). That is, to clusterize is to determine the “cluster information” for a set of observation data. Clustering is a common technique for the analysis of statistical data and is used in many fields, such as machine learning, data mining, pattern recognition, image analysis and bioinformatics.

The “raw” clustering problem is non-differentiable. That happens because the concept of belonging of a given point to a cluster (a group) is to belong to it or not to belong to it (there’s no partial belonging). For example: let a given point get a small perturbation in its position; that eventually produces an effect of making that point to cease belonging to cluster A and to belong to cluster B. That is: small perturbation in input data may produce large differences in output (the cluster information). The non-differentiable aspects of the clustering problem conduct the traditional clustering methods to be NP-hard.

HSCM (Xavier, 2010) introduced a rupture improvement that reduced substantially the complexity of the solution of the clustering problem. The rupture improvement was to make hyperbolic smoothing of the original problem, turning it into a sequence of low dimension differentiable unconstrained optimization problems. The dimension is similar to the number of clusters, not the input data. Thus, the reduction of the complexity of the clustering problem can be enormous. Another quality of the HSCM approach is that by requiring far less computations, the accumulation of rounding errors is lower than traditional methods; that is, the result is of better quality.

Our research showed that the HSCM contains inside itself parts that allow parallel computation. This means that a variation of this method can be proposed that takes advantage of the parallelizable parts of the method. The new method must be implemented and experimented in a computer that has parallel capabilities. We chose to explore the potential of the GPU as the parallel processing unit for the implementation. The parallelizable parts of the HSCM were identified and re-implemented using GPU parallel code. That produced a new method that we call “GPU-HSCM”, presented in this paper.

GPU – Graphics Processing Unit – is the main hardware inside a computer’s graphic card. This kind of hardware was originally intended only to improve the performance of graphics-intensive applications, such as games. (Nvidia 1999) defined GPU as “a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second.”

In 2007 NVidia released cards supporting an API extension to the C/C++ programming language named “Compute Unified Device Architecture” – CUDA, which allows functions from a normal C/C++ program to run on the GPU’s stream processors. This makes C/C++ programs capable of taking advantage of a GPU’s many cores and threads, while still making use of the CPU when appropriate. CUDA contains an API that allows CPU-based applications access directly the resources of a GPU for more general purpose computing without the limitations of

using the graphics API. This kind of program is called GPGPU (General-Purpose GPU); this is used for the GPU-HSCM proposed in this paper.

In this paper we will review briefly the cluster problem (section 2) and the Hyperbolic Smoothing Clustering Method (section 3); we discuss briefly about GPU programming in section 4. Our proposition, the GPU Hyperbolic Smoothing Clustering Method, is presented in section 5. The results are presented in section 6. The conclusions are presented in section 7.

## 2 Cluster Analysis

Let  $S = \{s_1, \dots, s_m\}$  denote a set of  $m$  observations from an Euclidean  $n$ -space, to be clustered into a given number  $q$  of disjoint clusters. To formulate the original clustering problem as a min-sum-min problem, we proceed as follows. Let  $x_i, i = 1, \dots, q$  denote the centroids of the clusters, where each  $x_i \in \mathfrak{R}^n$ . The set of these centroid coordinates will be represented by  $X \in \mathfrak{R}^{nq}$ . Given a point  $s_j$  of  $S$ , we initially calculate the distance from  $s_j$  to the center in  $X$  that is nearest. This is given by  $z_j = \min_{x_i \in X} \|s_j - x_i\|_2$ .

For a set of clusters  $x$ , it can be defined a “measurement of its quality with respect to the observation set  $S$ ”, also known as “cost function”. The most frequently used one is the sum of the squares of these distances, that is,  $\sum_{j=1}^m z_j^2$ .

The min-sum-min clustering problem is to find  $x$  that minimizes the cost function, as shown in equation (1).

$$X^* = \arg \min_{X \in \mathfrak{R}^{nq}} \sum_{j=1}^m \min_{x_i \in X} \|s_j - x_i\|_2^2 \dots\dots\dots (1)$$

Where  $X^*$  denotes the optimum location of centroids (that are expected to be calculated). It is a global optimization mathematical problem. It is both non-smooth (non-differentiable) and non-convex, with a large number of local minima. It is in the class of NP-hard problems (Brucker, 1978).

## 3 The Hyperbolic Smoothing Clustering Method

By performing a set of transformations the Hyperbolic Smoothing Clustering Method (HSCM) obtains a solution of the original problem (1) by solving a sequence of completely differentiable unconstrained problems (2).

$$\text{minimize } f(x) = \sum_{j=1}^m z_j^2 \dots\dots\dots (2)$$

where each  $z_j(x)$  results from the calculation of the unique zero of each equation (3).

$$h_j(z_j, x) = \sum_{j=1}^m \phi(z_j - \theta(s_j, x_i, \gamma), \tau) - \varepsilon = 0, j = 1, \dots, m \dots\dots\dots (3)$$

$\phi$  and  $\theta$  are the smoothing functions given by equations (4) and (5).

$$\phi(y, \tau) = \frac{(y + \sqrt{y^2 + \tau^2})}{2} \dots\dots\dots (4)$$

$$\theta(s_j, x_i, \gamma) = \sqrt{\sum_{l=1}^n (s_j^l - x_i^l)^2 + \gamma^2} \dots\dots\dots (5)$$

This transformation smoothes the problem, that is, it is possible to determine the derivative of equation (2), shown in equation (6).

$$\nabla f(x) = \sum_{j=1}^m \frac{df(z_j(x))}{dz_j} \nabla z_j(x) \dots\dots\dots (6)$$

where

$$\nabla z_j(x) = -\nabla h(x, z_j) / \frac{\partial h(x, z_j)}{\partial z_j} \dots\dots\dots (7)$$

Equation (3) can be solved using Broyden–Fletcher–Goldfarb–Shanno (BFGS) method or L-BFGS method; these methods are also known as “quasi Newton”. There exist some versions for BFGS and L-BFGS. To implement HSCM or GPU-HSCM requires choosing a version of BFGS.

## 4 GPU Programming

The GPU is a piece of hardware located in the graphics card. It contains processors (cores). The GPU is also known as “device”, while the CPU is also known as “host”. Traditional C-like functions, known as “kernels”, are executed in threads on the device. A large number of threads can be dispatched for the device to handle.

The CPU (host) has RAM memory. The GPU (device) has a RAM of its own. There exist more than one category of memory for the device. The main existing types are: global, block local and thread local.

The global memory is the largest one, for general purpose. The device cores are used in blocks of threads, and each block of threads has a memory for it, that is smaller and much faster than the global memory. Each thread has a memory for it, that is smaller than the block memory and much faster than the block memory.

For the host to call processing from the device, there’s a flow of operations, as described below, and illustrated on figure 1.

- 1) Copy data from main memory to GPU memory
- 2) CPU instructs the process to GPU
- 3) GPU execute parallel in each core
- 4) Copy the result from GPU memory to main memory

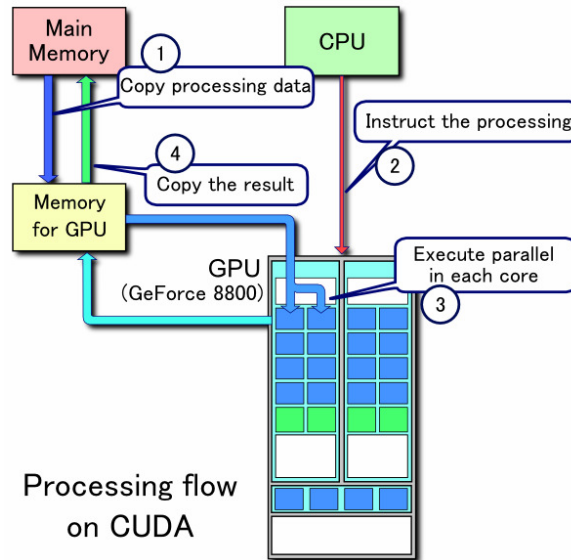


Figure 1: NVidia CUDA architecture for GeForce 8800

## 5 The GPU Hyperbolic Smoothing Clustering Method

The basic idea of the GPU-HSCM is to identify the parallelizable parts of the HSCM and implement them in the GPU. In our detailed analysis of the HSCM, we identified the parallelizable parts below.

- 1) Calculus of the objective function. This value is the summation of several portions; each portion depends on the distance of a given cluster point to each observation point; each portion can be calculated independently, that is, in parallel. This corresponds to equation (3).
- 2) Calculus of the gradient of the objective function. This is a set of values (one for each dimension). Each value is the summation of several portions; each portion depends on the distance of a given cluster point to each observation point; each portion can be calculated independently, that is, in parallel. This corresponds to equation (6).
- 3) Calculus of the list of associations. After the clusters are determined (the cluster points are known), it must be defined a “list of associations” (what observation is associated to what cluster). This requires the calculation of the distance of each observation to each cluster, which is independent and thus parallelizable. This corresponds to equation (2).

## 6 Results

Experimental implementation of both HSCM and GPU-HSCM were developed using 32-bit version of the C/C++ programming language with compiler “Microsoft Visual Studio 2008” and operating system “Windows Vista”. HSCM was developed as reference in performance and also to assert the correctness of result, when analyzing the output of GPU-HSCM, the number of cuda-threads was 128.

All tests were done using the same reference input data called “TSPLIB3038 data set”, with  $m=3038$  observations. The points are in  $\mathfrak{R}^2$ , that is,  $n=2$ . The figure 2 shows these points before and after the clusterization with  $q=6$  clusters. The location of the 6 centroids is emphasized.

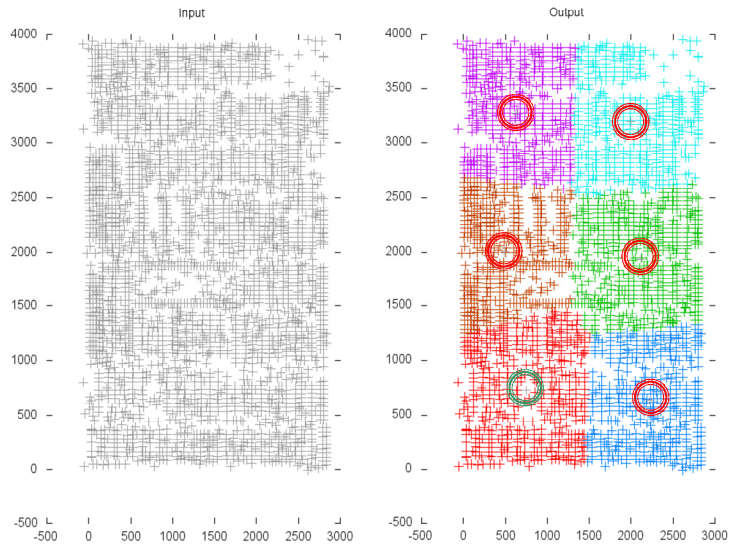


Figure 2: The TSPLIB3038 data set (left),  
and the clusterized solution with  $q=6$  clusters (right).  
The cluster information is shown as colors over the observation data.  
The centroid locations are emphasized.

HSCM was developed in some different versions, using different implementations of BFGS to solve equation (3). 2 versions of HSCM were implemented BFGS using GSL (GNU Scientific Library). The difference of version 1 and version 2 is the internal representation of the data arrays. In version 1 non-canonical C++ arrays were used; the results were less-than-expected, and a new version 2 was developed using canonical C arrays, with better results. A version 3 of HSCM was developed using (libLBFGS) for BFGS.

For each version of HSCM, a correspondent GPU-HSCM was developed. For each version, the same data was given and the clusterized data at output was the same in all cases. The comparison is done on processing time (the less time, the better). For the relative variation ( $\Delta\%$ ), negative means decrease of time, that is, improved performance. The table 1 shows the results.

Version	HSCM time (s)	GPU-HSCM time(s)	$\Delta\%$
1	28.392	17.596	-38,02%
2	22.292	12.433	-44,23%
3	16.063	18.914	17,75%

Table 1: comparison of results

## 7 Conclusion

The minimum sum-of-squares clustering problem is a very important numeric problem. Its solution demands strong computing power.

The Hyperbolic Smoothing Clustering Method – HSCM (Xavier 2010) introduces a rupture improvement to solve theustering problem, with great results over traditional methods.

In our detailed analysis of the HSCM, we identified its parallelizable parts. We chose to explore the GPU as the parallel hardware architecture for computation, and proposed the GPU-HSCM.

The experimental results (see table 1) conduct to the conclusions below.

- 1) The performance is sensitive to the coding standards (difference of version 1 and 2). The usage of canonical C arrays gave better results than C++ arrays.
- 2) The performance is sensitive to the choice of BFGS. To change from one implementation of BFGS to some other implementation requires considerable changes in the code. So we did not test many options for BFGS implementations, because that would require too much effort. But future work can undergo such investigation.
- 3) The GPU-HSCM did not always produce improvement of performance, as expected. GPGPU software development is very sophisticated (related to physical implementations). The usage of one type or other of device memory produces great differences in performance. The choice of number cuda-threads also influences the results. We are currently working on fine tuning of the GPGPU code. We are optimistic that great results will be achieved after this fine tuning.

## 8 References

**Brucker, P., 1978:** "On the complexity of clustering problems", Lecture Notes in Economics and Mathematical Systems 157 (1978) 45–54.

**GNU Scientific Library:** <http://www.gnu.org/software/gsl/>.

**Hartigan, J. A., 1975:** "Clustering Algorithms", John Wiley and Sons, Inc., New York, NY.

**libLBFGS:** <http://www.chokkan.org/software/liblbfgs/>.

**Nvidia, 1999:** <http://www.nvidia.com/page/geforce256.html>

**Späth, H., 1980:** "Cluster Analysis Algorithms for Data Reduction and Classification", Ellis Horwood, Upper Saddle River, NJ.

**Xavier, A. E., 2010:** "The hyperbolic smoothing clustering method", Pattern Recognition, Volume 43, Issue 3, March 2010, Pages 731-737, ISSN 0031-3203, DOI: 10.1016/j.patcog.2009.06.018. (<http://www.sciencedirect.com/science/article/B6V14-4WRD3G0-2/2/cbb627f11e8d7b71ec2e0c317c9bbf77>)