

Cooperating Multi-Core and Multi-GPU in the Computation of the Multidimensional Voronoi Adjacency in Machine Learning Datasets

Juan Mendez

Departamento de Informática y Sistemas
Universidad de las Palmas de Gran Canaria
Canary Islands, Spain
e-mail: jmendez@dis.ulpgc.es

Abstract—A cooperative framework is presented in this paper where multiple Cores in host and multiple GPUs cooperate to compute the Voronoi adjacency relationship in multidimensional Machine Learning datasets. Voronoi adjacency plays a very important role in neighbor based procedures in classification and data condensation. The proposal includes a system of Polytope Inclusion Agents, which computes the Delaunay polytope that contains a defined point, and a Coordination System among the agents that deals with the scheduling and load balance. The Polytope Inclusion Agent uses the Dual Simplex algorithm to solve a Linear Programming problem. The results show that for small datasets the use of GPU is a drawback, while for larger ones the GPUs take advantages of their massive parallelism.

Keywords: Parallel Computation, Multi-Core Systems, GPU, Linear Programming, Machine Learning, Voronoi Adjacency

1. Introduction

Voronoi adjacency and Delaunay tessellations are concepts of Computational Geometry widely used in many branches of the computation applied to science and technology. Voronoi and Delaunay tessellations provide partitions of representation spaces useful in applications concerning the spatial organization of data collections. Machine Learning (ML) is a discipline in which knowledge about such spatial organization of the data can improve the performance of the learning and classification procedures. The tessellation process makes a partition of the space in disjunct regions or cells called Delaunay polytopes or Voronoi polyhedra.

Many ML procedures, for example, neighborhood-based classification as well as dataset condensation, only need the adjacency relations between instances instead of full details of Voronoi or Delaunay tessellations. The Nearest Neighbor (NN) and k -NN are the most used algorithms in the family of neighborhood-based procedures. The Voronoi adjacency allows the data condensation or filtering of k -NN applications. The Voronoi adjacency deals with the problem of checking whether a pair of training instances share a common boundary, that is whether the two are neighbors in the Delaunay tessellation. Computing the Voronoi or

Delaunay tessellation in higher dimensional spaces can become impractical. However, computing only the Voronoi adjacency can be done very efficiently by using Linear Programming [1]. This approach has a sound theoretical background [2], [3], [4], [5] and can be continually improved with the advances in computer hardware and programming because Linear Programming (LP) takes continuous advantage of the increased performance of new computer architectures and programming paradigms. The efficient solving of LP problems has been a common topic in the areas of computer science and engineering dealing with highly efficient computing, parallelism and distributed systems. The emergence of new paradigms has generated new opportunities to improve a classical problem like LP. From the theoretical viewpoint these problems have sound and stable mathematical principles but, from the computational viewpoint, their matrix organization allows continuously evolving implementations.

Although there are some non-algebraic methods to solve LP problems, eg. the interior-point methods, the oldest and most used algorithm is the Simplex and its counterpart the Dual Simplex. Their algebraic origins allow parallel computing based on well founded algebraic libraries: eg., by using packages as the Basic Linear Algebra Subroutines (BLAS) [6]. Both algorithms, the Simplex and Dual Simplex, can be computed by using the Standard form or the Revised form. The first uses one large matrix, while the second deals with a smaller matrix containing the base of the problem. We can be dealing in LP problems with four different Simplex based algorithms: the Standard Simplex (SS), the Revised Simplex (RS), the Standard Dual Simplex (SDS) and the Revised Dual Simplex (RDS). Although the main task in these algorithms is algebraic, we will show in this paper that the non-algebraic parts of the algorithms play a more decisive role in modern implementations in a Graphical Processing Unit (GPU). Modern implementations of Simplex algorithms include the use of LAN distributed systems as shown by Yarmish and Slyke [7], using message passing in PVM, shown by Bixby and Martin [8] or more recently by using GPU as Greeff [9] and Spampinato and Elster [10]. Unfortunately, Greeff [9] could solve only small problems up to 200 variables, due to physical constraints in the GPU

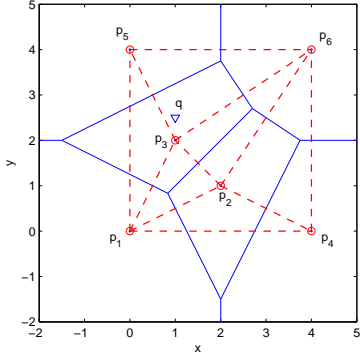


Fig. 1: A simple dataset[1] showing the Voronoi polyhedra as well as the Delaunay polytopes in \mathbf{R}^2 . The nearest neighbor of point \mathbf{q} is \mathbf{p}_3 and its natural neighbors are: $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_5, \mathbf{p}_6\}$ the computation of both concepts is a motivational problem in Machine Learning. Its efficiency is increased by the computation of Voronoi adjacency, e.g. the boolean V_{13} , which is the goal of this paper.

used, while Spampinato and Elster [10] have tested the RDS algorithm up to 2000 variables and constraints.

In modern computers, ranging from desktops to cluster nodes, it is usual that the available hardware includes several Cores and perhaps some GPU. This defines a hybrid or heterogeneous computational environment. Rather than focusing on the design of a Voronoi adjacency procedure to run in host or the GPU, the aim of this paper is to analyze the possibility of designing a *hybrid solution that allows the cooperative use all the available computational power of a system*, both multi-Cores and multi-GPUs. To achieve such a goal, this paper proposes the use of a system of coordinated agents. These, called Polytope Inclusion Agents, run in the heterogenous computational environment, some run in the host cores and others mainly run in several GPUs. They will cooperate for computing the boolean relationship of Voronoi adjacency in a multidimensional dataset. The main parts of the proposed systems are the Polytope Inclusion Agents and the Coordination System, which will be embedded into an OpenMP Critical Section.

The rest of the paper presents the mathematical theory of the adjacency test for a pair of points in a dataset, the implementation details of the Agents in CPU and GPU versions, the design of the Coordination System and, finally, the results and conclusions obtained by running the system in a computer with four cores and two GPUs.

2. Voronoi Adjacency

The Voronoi adjacency test for a pair of points in a dataset can be efficiently obtained from the computation of the polytope inclusion, which provides the polytope that contains a defined point. It is illustrated in Figure 1. Let

$\mathbf{P} = \{\mathbf{P}_1 \cdots \mathbf{P}_m\}$ be a dataset of m points in \mathbf{R}^n . Two points \mathbf{P}_i and \mathbf{P}_j are Voronoi adjacent if both are included in a Delaunay polytope obtained in the tessellation of \mathbf{P} . However, to obtain the Voronoi adjacency the computation of the whole tessellation is not necessary. The Voronoi adjacency of two points can be obtained from the polytope inclusion procedure of their middle point. We can obtain the Delaunay polytope in that a point $\mathbf{Q} \in \mathbf{R}^n$ is included based on computing the base of the following linear programming problem that provides $\mathbf{U} = \{u_1, \dots, u_m\} \in \mathbf{1}^m$:

$$\begin{aligned} \min \quad & \sum_{i=1}^m |\mathbf{P}_i|^2 u_i - |\mathbf{Q}|^2 \\ \text{st} \quad & \sum_{i=1}^m \mathbf{P}_i u_i = \mathbf{Q} \\ & \sum_{i=1}^m u_i = 1 \quad u_i \geq 0 \end{aligned} \quad (1)$$

where \mathbf{U} has a base defining the Delaunay polytope containing at most the $n+1$ points whose $u_i \in [0, 1]$ values are not null. The term $|\mathbf{Q}|^2$ is a constant value that can be avoided. The solution can be degenerated, in which case the cardinality of the base is $K \leq n+1$. Each pair of points in the base defines a Voronoi adjacency or a pair of Voronoi neighbors, which is represented by the boolean matrix V_{ij} .

The problem of polytope inclusion of a point may be unfeasible, having no solution if \mathbf{Q} is out of the convex hull enveloping the dataset \mathbf{P} . However, this case is avoided in the Voronoi adjacency procedure because the test point is always a middle point between two points of the dataset, which is obviously always within the convex hull. The multi-dimensional dataset containing m points in a n -dimensional space is represented by the matrix \mathbf{P} of dimension $n \times m$, and \mathbf{Q} is a n -dimensional vector:

$$\mathbf{P} = [\mathbf{P}_1 \cdots \mathbf{P}_m] = \begin{bmatrix} p_{11} & \cdots & p_{m1} \\ \vdots & \ddots & \vdots \\ p_{1n} & \cdots & p_{mn} \end{bmatrix} \quad (2)$$

$$\mathbf{Q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix} \quad (3)$$

We use the Standard Dual Simplex (SDS) algorithm in this paper. According to Yarmish and Slyke[7], Revised algorithms take better advantage of sparsity in problems, while Standard algorithms are more effective for dense problems such as addressed in this paper. The LP problem of the polytope inclusion can be represented by the matrix \mathbf{T} , the tableau of the LP problem. It is obtained by transforming the equality equations in (1) in pairs of inequality ones.

$$\mathbf{T} = \begin{array}{c|ccc|ccc|c} \hline |\mathbf{P}_1|^2 & \cdots & |\mathbf{P}_m|^2 & 0 & \cdots & 0 & |\mathbf{Q}|^2 \\ \hline p_{11} & \cdots & p_{m1} & & & & q_1 \\ \vdots & \ddots & \vdots & & & & \vdots \\ p_{1n} & \cdots & p_{mn} & & & & q_n \\ 1 & \cdots & 1 & & & & 1 \\ \hline -p_{11} & \cdots & -p_{m1} & & & & -q_1 \\ \vdots & \ddots & \vdots & & & & \vdots \\ -p_{1n} & \cdots & -p_{mn} & & & & -q_n \\ -1 & \cdots & -1 & & & & -1 \\ \hline \end{array} \quad (4)$$

The matrix \mathbf{I} is the $2(n+1)$ order identity matrix. The matrix \mathbf{T} has $2(n+1) + 1$ rows and $m + 2(n+1) + 1$ columns. The solving of the Simplex algorithms requires three subtasks related to find the pivot, which is related to the leaving and entering variables in the base of the problem. The Simplex and Dual Simplex forms differ in the order of search for these variables. Our problem is related to the Dual forms, in that the subtasks are:

- 1) Find the Leaving Variable (LV) in the base. In the SDS algorithm, this is accomplished by computing the minimum value and its position in the last column, excluding the first row, of the matrix \mathbf{T} .
- 2) Find the Entering Variable (EV) in the base. In the SDS this is accomplished by obtaining the column pivot based on the data of the first row and the pivot row. This task requires the computing of the maximum and its position in an intermediate result computed from these two rows[11]. In this step the problem can finish without a solution in an unfeasible state.
- 3) Normalize (N) the matrices of the problem according the pivot column and row. In the SDS it is based on the Gauss-Jordan pivoting to normalize the pivot column.

Steps 1 and 2 are non-algebraic procedures. In RDS these steps require the pre-computing of the last column, first row and pivot row. That tends to serialize the algorithm, while in SDS it is performed over the \mathbf{T} data directly. In SDS, step 3 is directly implemented by the $Dger()$ function in the BLAS Level 2 library. This function makes the following matrix transformation based on the outer product of vectors \mathbf{x} and \mathbf{y} :

$$\mathbf{A} = \mathbf{A} + \alpha \mathbf{x} \mathbf{y}^T \quad (5)$$

where α is a scalar, \mathbf{x} and \mathbf{y} are vectors and \mathbf{A} is a matrix. In our problem they correspond to $\alpha = -1$, the normalized pivot row and column and the tableau \mathbf{T} respectively. After the base has been computed, each pair of points, a and b , in the base are Voronoi adjacent and we can set the V_{ab} as true. If the original pair of points in the test, i and j , are in the base, they become adjacent, V_{ij} is set to true, otherwise they are not adjacent and V_{ij} is set to false. In practice we never set a false value because we use a sparse boolean matrix \mathbf{V} ,

which only includes the true cases. The cooperative system, for computing the whole Voronoi relationships included in a dataset, contains the following subsystems:

- 1) **Polytope Agent.** It computes the polytope in which the middle point of two dataset points, i and j , is included. This allows the boolean V_{ij} to be computed. Also, this polytope contains other points, which allows the collateral computation of other V_{ab} values. Each Polytope Agent runs in a different thread, its first task is to search for a point pair that is not yet computed. Many Agents are trying to get pair points, but they are coordinated by an exclusion procedure to prevent two agents trying to compute the same pair. However, due to the collateral computation, some values V_{ab} can be obtained many times. We name this a *collision*, although this is a drawback, collateral computation has many advantages such as the number of tried pairs is lower than the number of pairs in the dataset.
- 2) **Coordination.** The coordination of the Agents is implemented by means of two data structures that can only be modified inside a Critical Section. The coordination is organized by means of *non-tried yet* (NTY) pairs and *already computed* (AC) pairs. The two structures are not complementary in this problem, but obviously an already computed pair will not be tried in the future. This coordination also ensures a dynamic Load Balance because all the agents are computing one pair or are trying to access the Critical Section that implements the coordination.

Each Voronoi test requires different iteration number to solve the LP problem, but for large datasets the mean required load tends to be uniform. Therefore the load distribution or number of pairs that each agent carries out depends on its computational power. In homogeneous system, where all agents are host based or GPU based, the load distribution tends to be uniform. However, in heterogeneous systems the opposite happens and load distribution among agents depends on the computational power of each implementation.

A pair of points would be in NTY and at the same time in AC due to the collateral computation of Voronoi adjacency. In applications where the only way to compute the adjacency of a pair is by trying to compute it, the NTY+AC coordination is unnecessary. However, in the case when an agent tries to compute a pair, it is deleted from the NTY but collateral computation of other pairs would arise and include it in AC. This means that parallel computation of pairs increases the collisions. Figure 2 shows a case of potential parallel collisions if pair \mathbf{p}_1 and \mathbf{p}_2 is computed in parallel to pair \mathbf{p}_6 and \mathbf{p}_7 , because both middle points are within the polytope $\{\mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\}$.

3. Polytope Inclusion Agent

The algorithm of the Polytope Inclusion Agent is the same for the different environments and the implementations are

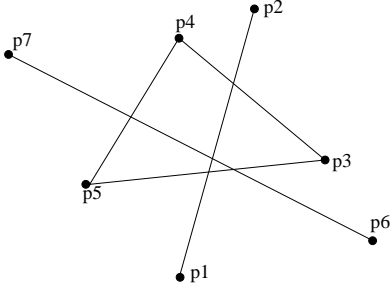


Fig. 2: Some collision cases. When the Voronoi Adjacency test for pair p_1 and p_2 is computed, it provides the polytope: $\{p_3, p_4, p_5\}$, which implies the true values for: $\{V_{34}, V_{35}, V_{45}\}$ and also the V_{12} becomes false. If previously the pair p_3 and p_4 , which are Voronoi neighbors, has previously been tried, the V_{34} is obtained in both cases: this is a collision.

similar in outline. The algebraic operations of the algorithm are computed by using BLAS functions. In the case of host agents, the *cblas* library included in the Intel’s Mathematical Kernel Library (MKL) was used in the sequential version. In the GPU agents, CUBLAS was used for the algebraic operations and CUDA runtime kernels for the non-algebraic ones.

Each agent runs in a host thread launched by OpenMP in a parallel directive. Prior to the threads forking the system acquires the number of available GPUs. At this time we can define whether none, some or all of the number of GPUs will be used. We can also define the total number of thread agents. Each thread tries to acquire a GPU by using its own thread number to get that device by using the set device function of CUDA. The rest of threads without device, whose thread number is greater than the defined number of devices, are host based running completely in the host cores. Each GPU agent runs partially in a host thread and partially in a GPU, but its tableau is resident in device memory.

The Normalize (N) subtask, which performs the update of the matrix T , of the LP problem was implemented by using BLAS in both agent types, while the Leaving Variable (LV) and Entering Variable (EV) are the non-algebraic subtasks that are implemented by kernel launches in the GPU dealing with the computing of the value and position of Maximum or Minimum in an array. Max/Min procedures based on the parallel reduction[12] are implemented.

The procedure was tested and its performance for different array sizes, from 1K to 4M elements, was obtained. Figure 3 shows the speedup of relative performance for the CPU and two GPU strategies, where CPU refers to serial computation in host by using plain C for an array in host memory, GPU(1) refers to a device procedure applied to an array in device memory, and GPU(2) to host computation of an array in device memory copied to host memory. The GPU(1)

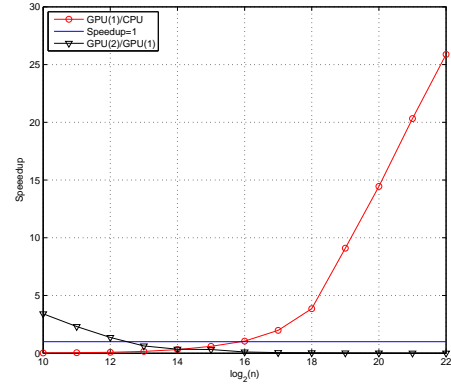


Fig. 3: Max/Min+Pos procedure, speedup ratio of GPU(1) over CPU and GPU(2) over GPU(1)

outperforms CPU for arrays greater than 64K elements. At 4M elements the speedup is 25.88 but at smaller array sizes it is too slow. The option to compute in host the Max/Min+Position of an array in device memory is better for array sizes up to 4K elements. In the practice we designed a procedure that decides the use of the GPU(1) or GPU(2) strategies at runtime. We used a threshold of 5000 elements. These results and criteria are strongly dependent on the system hardware and the programming tools used. Details of the hardware used are given in the Results Section.

4. Coordination and Load Balance

The easiest way to implement a data structure for non-tried yet pairs (NTY) is by means of a boolean matrix, in that a true value can be used to inform to Agents about suitable pairs. Coordinated access to the Critical Section assures that no two agents are trying the same pair. This approach requires of a large 2D boolean matrix containing $m \times m$ elements. In practice, we can reduce the needed data to one of the upper or lower triangular matrices, that is, we need only $m(m-1)/2$ elements although this is not useful for big datasets. The proposed solution includes only one index value, which is non concurrently accessed by the Agents in the critical section. The pairs are ordered according to their position in the upper triangular matrix, but this matrix is never constructed. The index is the in row major position in that formal ordering, therefore its value runs from 0 to $m(m-1)/2 - 1$. The use of 32, or 64, bits integers allows large datasets to be managed. More precisely, the index value is the position of the lower index pair not tried yet, which is the ideal candidate to be computed by the next agent that requests a pair of points. Every Agent will acquire this value to compute the associated pair.

Let n_1 and n_2 be two integer values corresponding to point indexes of a pair in upper triangular matrix such as $n_1 < n_2$, where $0 \leq n_1 \leq m-2$ and $1 \leq n_2 \leq m-1$. The

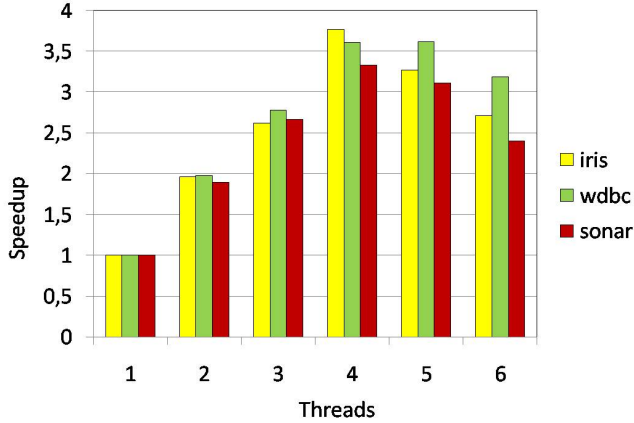


Fig. 4: Speedup ratio for the CPU based agents and the smallest datasets.

index value I for this pair is:

$$I = \frac{n_1(2m - 1 - n_1)}{2} + n_2 - n_1 - 1 \quad n_1 < n_2 \quad (6)$$

This value runs from 0 to $I_{max} = m(m - 1)/2 - 1$. The inverse problem is to obtain the values of n_1 and n_2 corresponding to a value I . It is achieved by computing n_1 as:

$$n_1 = \text{int} \left(\frac{(2m - 1) - \sqrt{(2m - 1)^2 - 8I}}{2} \right) \quad (7)$$

When a Polytope Inclusion Agent tries to get a suitable pair, it tries to enter in a named OpenMP Critical Section, gets a NTY pair and if it is not in AC computes the polytope inclusion code. When a polytope has been computed, the agent enters in the named Critical Section and sets all the pairs of Voronoi adjacency in the AC. The already computed pairs (AC) are implemented by means of a sparse boolean matrix. Note that the number of Voronoi neighbors of a point is smaller than the dataset cardinality m . The implementation of NTY by means of a sparse boolean matrix is not a good idea because, at the start of the procedure, all the pairs are NTY and therefore the matrix must be dense. The Load balance among the Polytope Agents is achieved dynamically because it depends on the strategy of concurrent access to the Critical Section NTY implemented by OpenMP. The fastest agents access more times and compute more pairs of Voronoi neighbors.

5. Results

Some of the most used datasets of the UCI Machine Learning Repository[13] have been used to test the system. The dataset *iris* is the smaller used with $m = 150$ samples or points. The largest used is the *shuttle* dataset, which has two versions, test and train: we have used the smaller test

version with $m = 14500$ points. The dimensionality ranges from the smaller *iris* with $n = 4$ to the larger *sonar* dataset with $n = 60$.

One of the fastest packages most used in scientific computation for fast prototyping as in MATLAB and Mathematica programs, is the qhull[14]. To solve several problems in computational geometry in \mathbf{R}^n , it uses the computation of the convex hull in \mathbf{R}^{n+1} as the kernel procedure. The family of programs based on qhull, which include quickvoronoi are very fast for problems with low dimensionality. However, they suffer the curse of dimensionality when applied in high dimension problems such as those used in ML. This algorithm is very efficient for low dimensional datasets, useful for 2D and 3D problems involved in finite elements for engineering problems and graphics. However for multi-dimensional problems, as involved in ML and Data Mining, it breaks down.

To test the system we used a desktop computer with one Intel Core 2 Quad Q8300 processor and 4Gb of main memory. The system included a graphic card NVIDIA GTX 295 with 2 GPUs (240 cores and 896Mb of global memory each). Times were measured by using the performance counters of the Windows XP 32 bits, which is claimed that on a multiprocessor computer it should not matter which processor is called. However, to avoid multi-threading problems, the timer code uses two calls, start and stop, in the same thread usually the main thread. The code was developed in Visual Studio using C++. The agent code includes an exit condition when the number of computed pairs exceeds a fraction of the total pair number. This criterion is used only for the big *shuttle* dataset, such as the results provided for this dataset are mean values related to a *fraction* of the 0.01% of all its pairs, which is advised by denoting *shuttle(f)*.

Table 1 contains the basic data of the used datasets, the tried pairs in the process, and also for one CPU agent the mean values for LV, EV and N subtasks and the total time used in the process. It also includes the total time reported by quickvoronoi, which notably outperforms others for small datasets, but it can not run for the bigger ones. The N subtask becomes critical for the big datasets. Table 2 contains the same data type for one GPU agent. It becomes clear that this agent performs better in the algebraic N subtasks than in the non-algebraic LV and EV. More specifically, it performs worse for the LV although this subtask is small if compared to the others. After analyzing the results, we concluded that the problem arises in the copy, uncoalescent in row major organization, of the data from the last column of matrix T to one buffer array where the minimum value was obtained. That is, the smaller subtask in data size and operation number becomes the worst in terms of time spent.

Figure 4 shows the speedup achieved by using 1, to 6 CPU agents, for the small datasets by using MKL package, while Figure 5 shows the results for the *shuttle(f)* dataset and Figure 6 shows the load distribution between agents for

Table 1: Time mean values and the number of Voronoi adjacency tests, qV means quickvoronoi and IM insufficient memory in host. Only a host based agent was used. The critical subtask is the Normalization for updating the matrix.

dataset	dim(n)	samples(m)	tests	LV(μs)	EV(μs)	N(μs)	Total(sec.)	qV(sec.)
iris	4	150	9543	0.23	1.73	1.41	0.49	0.016
bupa	6	345	47325	0.26	3.81	3.43	10.37	3.469
glass	9	214	12187	0.26	2.48	3.51	4.20	19120
wine	13	178	3902	0.29	2.06	4.96	1.89	IM
wdbc	30	569	42140	0.39	5.82	25.42	192.40	IM
sonar	60	208	5673	0.58	2.62	28.28	5.19	IM
shuttle(f)	9	14500	-	0.79	160.74	675.81	-	-

Table 2: Mean values for one GPU based Agent. The critical subtasks are those related to non-algebraic processes in special LV.

Dataset	LV(μs)	EV(μs)	N(μs)
iris	96.83	45.57	25.96
bupa	74.00	47.81	25.99
glass	61.11	46.85	25.92
wine	59.50	46.15	25.96
wdbc	56.55	50.52	26.13
sonar	85.62	47.49	26.03
shuttle(f)	167.62	124.10	25.95

this dataset. In small datasets the CPU agents perform as expected by increasing the speedup with the increasing of the threads. The CPU agents perform badly with the big dataset, because a poor speedup was achieved. However, the GPU agents perform better for the first thread, which used the GPU with device number 0, and the second thread, which used the GPU with device number 1. The third agent, which is based on CPU, increases the performance only a little, and the use of the next agents provides worse performance.

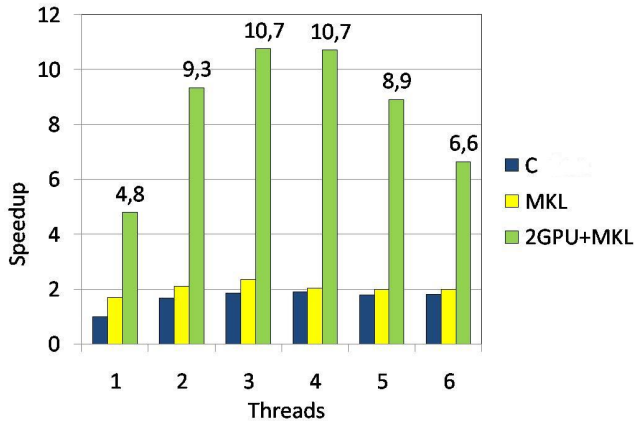


Fig. 5: Speedup of multi-core multi-GPUs related to one thread in CPU for the shuttle(f) dataset. First and second threads use one GPU each, next threads run in the multi-core system, but little additional improvement is achieved

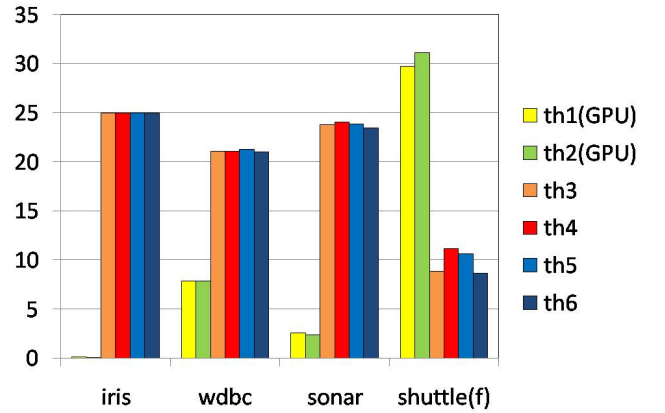


Fig. 6: Load Distribution between agents in percent (%) . Thread th1 and th2 manage a GPU each, while th4 to th6 run in the Host. For the iris dataset, the GPU agents are very slow while for the shuttle(f) dataset are the fastest.

6. Conclusion

For small datasets the use of GPU based agents is not a good option, it is rather a drawback than an advantage. In these cases multi-core systems perform better. However, for big datasets the GPU agents are the more efficient, this means that they are specially suitable for massive ML procedures as well as for Data Mining applications. In these cases, future studies are necessary to define the frontiers between the performance regions, that is when a dataset becomes small or big from the viewpoint of the use of GPU.

The systems of agents and coordination to compute the Voronoi adjacency allows the design of a cooperative strategy in which we can involve hardware with many cores and also heterogeneous computation. The design is extensible and is not constrained to a fixed number of agents. The inclusion of agents based on other paradigms of parallel and distributed computer as clusters and grid computing is a natural extension of what is proposed in this paper.

References

- [1] K. Fukuda, "Frequently asked questions in polyhedral computation," Swiss Federal Institute of Technology, Lausanne, Switzerland, Tech. Rep., June 2004.
- [2] G. Kalai, "Linear programming, the simplex algorithm and simple polytopes," *Math. Program.*, vol. 79, pp. 217–233, 1997.
- [3] K. Fukuda, T. M. Liebling, and F. Margot, "Analysis of backtrack algorithms for listing all vertices and all faces of convex polyhedron," *Computational Geometry*, vol. 8, pp. 1–12, 1997.
- [4] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete Comput. Geom.*, vol. 8, no. 3, pp. 295–313, 1992.
- [5] D. Bremner, K. Fukuda, and A. Marzetta, "Primal-dual methods for vertex and facet enumeration," in *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*. New York, NY, USA: ACM, 1997, pp. 49–56.
- [6] J. Dongarra, "Basic linear algebra subprograms technical forum standard," *International Journal of High Performance Applications and Supercomputing*, vol. 16, no. 1, pp. 1–111, 2002.
- [7] G. Yarmish and R. van Slyke, "retroLP, an implementation of the standard Simplex method," Dept. of Computer and Information Science, Brooklyn College, Tech. Rep., 2001.
- [8] R. E. Bixby and A. Martin, "Parallelizing the dual simplex method," *INFORMS Journal on Computing*, vol. 12(1), pp. 45–56, 2000.
- [9] G. Greeff, "The revised simplex algorithm on a GPU," Dept. of Computer Science, University of Stellenbosch, Tech. Rep., February 2005.
- [10] D. G. Spampinato and A. C. Elster, "Linear optimization on modern GPUs," in *IPDPS*. IEEE, 2009, pp. 1–8. [Online]. Available: <http://dx.doi.org/10.1109/IPDPS.2009.5161106>
- [11] W. L. Winston, *Operations Research Applications and Algorithms*. Wadsworth, 1994.
- [12] M. Harris, "Optimizing parallel reduction in CUDA," NVIDIA Corporation, Tech. Rep.
- [13] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: <http://www.ics.uci.edu/~mlern/MLRepository.html>
- [14] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, 1996.