Scalability and Fuzzy Systems: What Parallelization Can Do

Perfecto Malaquias Quintero Flores, Anne Laurent, Federico del Razo Lopez, Nicolas Sicard, Pascal Poncelet

Scalability and Fuzzy Systems: What Parallelization Can Do

P. Malaquias Quintero F., Anne Laurent, Federico Del Razo, Nicolas Sicard and Pascal Poncelet

Abstract (Fuzzy) Database management systems aim at providing tools for data storage and querying. Based on the information stored, systems can offer analytical functionalities in order to deliver decisional database environments. In this framework, fuzzy systems have been proven to be efficient for modeling, reasoning, and predicting. So far, success has been achieved in several application areas. However, expanding the frontiers of such areas or exploring new domains is often limited when facing real world data: more possibilities must be explored, more computation time and memory space are required. In this paper, we discuss how the parallelization of fuzzy algorithms is crucial to tackle the problem of scalability and optimal performance in the context of database mining. More precisely, we present the parallelization of fuzzy database mining algorithms on multi-core architectures of four knowledge discovery paradigms, namely fuzzy association rules, fuzzy clustering, fuzzy gradual dependencies, and fuzzy tree mining (for example in the case of XML databases).

P. Malaquias Quintero F.
LIRMM - University Montpellier 2 CNRS, 161 rue Ada 34095, Montpellier, France, and Instituto Tecnológico de Apizaco, Tlaxcala, Mexico. e-mail: quinterofl@lirmm.fr

Anne Laurent
LIRMM - University Montpellier 2 CNRS, 161 rue Ada 34095, Montpellier, France, e-mail: laurent@lirmm.fr

Federico Del Razo
Instituto Tecnológico de Toluca, DGEST-SEP Av. S/N Metepec Edo. de Mexico, Mexico, e-mail: delrazo@ittoluca.edu.mx

Nicolas Sicard
AllianSTIC-EFREI, Paris, 30-32 av. de la République, 94800 Villejuif Cedex, France, e-mail: nicolas.sicard@efrei.fr

Pascal Poncelet
LIRMM - University Montpellier 2 CNRS, 161 rue Ada 34095, Montpellier, France, e-mail: Pascal.Poncelet@lirmm.fr

1 Introduction

In recent years fuzzy set and fuzzy logic theory have found application in mathematical theory, artificial intelligence, non-linear control, real-time systems, database mining, machine learning, database management systems, decision making, consumer electronics, expert systems, economics, finance, software engineering, among other interesting areas of application [19] [23] [45] [51]. The expression "*fuzzy systems*" is the name commonly used to refer in general to the resulting systems from the different applications of fuzzy logic [23]. Whereas for referring to specific systems, we are using expressions such as *fuzzy control*, *fuzzy database management systems*, *fuzzy database mining techniques*, ***etc.***.

Fuzzy systems are computer systems inspired in the linguistic processing of information, where representation and processing of imprecise and uncertain data is done through fuzzy set theory and fuzzy logic (fuzzy inference) respectively [23] [45]. Such systems aim at implementing on the machines, models and algorithms related to information processing by *approximate reasoning* and methods to extract *automatically knowledge* from large and complex databases efficiently [19] [51].

Approximate reasoning is defined as the process of obtaining precise information from imprecise data [23]. The *automatic extraction of knowledge from databases*, also known as *Knowledge Discovery in Databases* and defined as a multi-step process of discovering potentially useful information from large and complex databases [40]. In this framework, *fuzzy database management systems* and *fuzzy database mining techniques* have an important role [3] [19] [21] [34].

With the emergence of innovative and accessible models of parallel computation, fuzzy systems can improve their performance by using parallel computing architectures. In this paper, we discuss how important the parallelization of fuzzy algorithms can be to tackle the problem of scalability and optimal performance in the framework of fuzzy database mining. More precisely, we discuss the parallelization of fuzzy database mining algorithms on multi-core architectures of four knowledge discovery paradigms, namely fuzzy association rules, fuzzy clustering, fuzzy gradual dependencies, and fuzzy tree mining (for example in the case for XML databases). In all cases highlighting the important role of fuzzy databases in the process of extraction of fuzzy patterns.

The outline of this paper is as follows: In Section 2, we present an overview about fuzzy database management systems and about fuzzy database mining techniques. We present a brief review about multi-core programming and *GPU* computing in Section 3. In Section 4, we present our review of the parallelization of four fuzzy database mining algorithms on multi-core architectures, namely fuzzy association rules, fuzzy clustering, fuzzy gradual dependencies, and fuzzy tree mining. Finally, we conclude and give some suggestions for future research directions in Section 5.

2 Fuzzy Databases and Fuzzy Database Mining Techniques

2.1 Fuzzy Databases

In the framework of traditional database management systems, it is common to assume that the data are precise and certain. Unfortunately, real-world data are often uncertain, imprecise, inconsistent, ambiguous and vague, due to different causes such as: human errors, instrument errors, recording errors, noisy data, ***etc.*** [50]. *Fuzzy database management systems* aim at providing tools for storage and querying data with the previously mentioned imperfections.

During the last thirty years has been carried out extensive scientific research work aimed at developing different approaches of how to incorporate different kinds of *fuzziness* into of the relational database model (FRDBM) and into of the object-oriented database model (FOODBM) [33] [38].

In FRDBM there are two kinds of fuzziness called *fuzziness in attribute values* and *fuzziness in tuple*, which can be incorporated according to three principal approaches called: 1) Fuzzy relation-based model, 2) Similarity relation- based model and 3) Possibility distribution. In FOODBM there are four kinds of fuzziness called: *Fuzziness in object*, *fuzziness in class*, *fuzziness in object-class* and *Fuzziness in class-subclass*. An extended review of these kinds of fuzziness is presented in [33] [38] [46].

An interesting analysis of the application and commercialization of *fuzzy database management systems* is presented in [38] [46]. Currently, *fuzzy database management systems* and *fuzzy database mining techniques* are closely related.

2.2 Fuzzy Database Mining Techniques

The aim of database mining can be defined as finding patterns or rules that describe the meaning of the relationships or dependencies between the data contained in big and complex databases. Database mining is an interdisciplinary field, which combines research from areas such as machine learning, statistics, theory of fuzzy sets and logic fuzzy, neural networks, evolutionary computing, high performance computing and parallel programming, and databases [12] [16] [19].

In the pattern mining field, an important problem is the extraction of patterns that are intrinsically vague, imprecise, uncertain and that can involve data disturbed by noise [20]. This problem comes from the fact that real-world data tends to be uncertain due to human errors, instrument errors, recording errors, noisy data, so on [50]. Fuzzy databases allow an natural and flexible representation of patterns and data

with the characteristics mentioned above. In this framework, in recent years, several extensions of database mining have been developed on the basis of fuzzy sets and fuzzy logic theory [19], such extensions are known as fuzzy database mining techniques (FDMT) [3].

The following list outlines several representative examples of FDMT:

- Fuzzy association rules, gradual itemsets and gradual dependencies;
- Fuzzy decision-tree: Classification based on models of patterns in the database, all of the existing data are mapped to predefined set categories;
- Fuzzy frequent subtree: Patterns that are embedded in the semi-structured database, even if the patterns are only partially present.
- Fuzzy regression: Fuzzy relations describing data in the form of some linear functional dependencies between variables that are encountered in the database;
- Fuzzy clustering: Classes (groups) of data are developed based on their similarities and differences;
- Fuzzy summarization: A compact data description, such as fuzzy contingency table, linguistic data summarization, etc.

Scaling algorithms of FDMT is a challenge [15], because their search spaces, requirements of computation time and memory are larger than of the algorithms used in crisp database mining methods.

3 Multicore Programming and GPU Computing: An overview

Parallel computing is a viable means to improve performance of algorithms of fuzzy computing [49]. With the emergence of new generations of multicore processors and the new generations of graphics processing units $(GPUs)$ as key components of high performance hardware of a computer system, optimization of fuzzy systems through its parallelization is possible on general purpose computing platforms [5] [11] [25].

Parallelization-based optimizations of algorithms [2] [49] aim to: i) Reduce the execution time, ii) Allow Real-time processing, iii) Solve large problems, and iv) Exploit the computing power of the more and more present high-performance systems (e.g., multicore processors that now even equip mobile phones and tablets).

3.1 Process & Thread

In multi-core architectures, a parallel program is executed by the processors through one or multiple control flows referred to as processes or threads [18] [49]. A pro-

cess can consist of several threads which share a common address space whereas each process works on a different address space [18]. In order to achieve efficiency, the multi-core *CPUs* can use only a few threads, while *GPUs* may use thousands [4].

In the massively multi-threaded SIMD (Single Instruction Multiple-Data) architecture provided by *GPUs*, threads are extremely lightweight and grouped into *threads blocks* [4]. Threads within the same *thread block* are divided into SIMD groups, called *warps*, each of which contain 32 threads [4] [11].

The parallel portions of an application are executed on the device GPU as *kernels*, one *kernel* is executed at a time by an array of threads, where all treads run the same code and each thread has an ID that it uses to compute memory addresses and make control decisions [4] [11].

3.2 Parallel Programming Models

The means to program a parallel computer is named *parallel programming model* [42] [49] [42]. There are numerous parallel programming models, most notably *Distributed memory*, *Shared memory*, and the *Hybrid parallel programming model*.

The shared memory model for multi-core *CPUs* may be implemented through standardized programming interfaces, like *Posix threads* (C/C++), *OpenMP* (C/C++, Fortran), Python, Java, *Melinda*, and *Automatic parallelization* [42] [49].

The CUDA parallel programming model for many-core *GPUs* provides support for parallel programming with C/C++, OpenCL, DirectX Compute, Fortran, Java, and Python [5].

4 Parallel Fuzzy Database Mining

Parallelizing fuzzy database mining algorithms is a viable means to improve their performance and for making feasible fuzzy database mining to large-scale [16]. Within the framework of multiprocessor/multi-core architectures of share/distributed memory, parallel fuzzy database mining as well as parallel database mining follows two approaches of parallelization: *task parallelization* and *data parallelization* [11] [12] [43].

In *task parallelization* the processors/cores execute a different task on the (fuzzy) database. In *data parallelization* the (fuzzy) database is partitioned among the processors/cores and all execute the same task.

Moreover, researchers from academia, software industry such as Microsoft, Oracle, SAP, and other corporations are seeing to *CUDA architecture for GPUs* as a

scalable solution of their toolboxes of database mining techniques-based data analysis [7] [25]. For example, Tanay Data Analytics from Fuzzy Logix in-database analytics engine accelerates financial simulations, database mining, and statistical methods using *GPUs* [13] [14].

4.1 Parallel Mining of Fuzzy Association Rules

Fuzzy association rules mining is a process to find out the fuzzy patterns or fuzzy attributes which frequently occur together from a fuzzy database [8] [32] [34].

Given a subset of fuzzy data records $D_R = \{r_1, r_2, ..., r_n\}$ obtained from a fuzzy database (fD), a fuzzy association rule is defined as an fuzzy implication of the form: $P_I \Rightarrow P_J$, where P_I, P_J are fuzzy itemsets belonging to a set of fuzzy items defined as **P**=$\{\mu_1 \in P_1,\ \mu_2 \in P_2, ...,\ \mu_m \in P_m\}$, such that $\mu_i \in P_I \neq \mu_j \in P_J$, for $i = 1,2,...,m$, $j = 1,2,...,m$, and $m =$ *number of fuzzy items* $\in$ **P**.

A $r_k \in D_R$ denotes the $k-$th record, represented as a vector with m values, $[\mu_1(P_1[r_k]),\ \mu_2(P_2[r_k]), ...,\ \mu_m(P_m[r_k])]$, where $\mu_j(P_j[r_k])$ is membership degree that item value $P_j[r_k]$ belongs to fuzzy set μ_j, and $\mu_j(P_j[r_k]) \Rightarrow$[0, 1], for $k = 1,2,...,n$ and $n =$*number of records* $\in D_R$.

In order to mine rules of the form $P_I \Rightarrow P_J$, fuzzy support $fsupp(\{P_I, P_J\})$ and fuzzy confidence $fconf(P_I \Rightarrow P_J)$ can be computed as in equations (1) and (2).

$$fsupp(\{P_I, P_J\}) = \frac{\sum_{k=1}^{n} min(\mu_i(P_I[r_k]), \mu_j(P_J[r_k]) \mid I,J \in \{1,2,...,m\}, I < J)}{n} \quad (1)$$

$$fconf(P_I \Rightarrow P_J) = \frac{fsupp(\{P_I, P_J\})}{fsupp(P_I)} \quad (2)$$

The fuzzy association rules with at least a minimum support and a minimum confidence respectively are extracted and considered as interesting [8] [32] [34].

Bao-wen et al., and Jian-jian et al., presented the adaptation of the Count Distribution Parallel Algorithm to design the parallel algorithm for mining fuzzy association rules [1] [27]. Where, quantitative attributes are partitioned into several fuzzy sets by the parallel fuzzy c-means algorithm (PFCM). The parallel algorithm for mining Boolean association rules is improved to extract frequent fuzzy patterns. Finally, the fuzzy association rules with at least fuzzy confidence are generated on all processors. The parallel algorithm based on the Message Passing Interface (MPI) was implemented on the distributed linked PC/workstation of six computers with 128 MB RAM, interconnected via a 10 M/100 M hub. The results of experimental

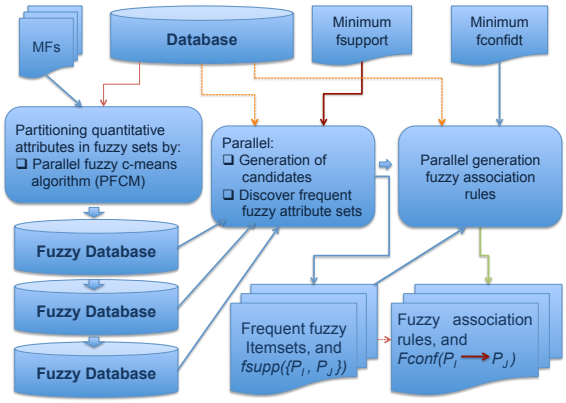

Fig. 1 Scheme of parallel mining fuzzy association rules.

work showed that the parallel mining algorithm had an excellent *scaleup*, *sizeup* and *speedup*.

In another approach [17], in order to extract both association rules and membership functions from quantitative attributes, Hong et al., propose a parallel genetic-fuzzy mining algorithm based on the master-slave architecture. Where the master processor uses a single population as a simple genetic algorithm, and distributes the tasks of fitness evaluation to slave processors. The crossover, mutation and production are performed by the master processor. The results showed that the speed-up can increase nearly linear along with the number of individuals to be evaluated.

Fig. 1 shows the general structure of our interpretation of the parallel process of extracting fuzzy association rules.

4.2 Parallel Fuzzy Clustering: c-means

Clustering is defined as the process of grouping a data set, where the similarity between data within a cluster is maximised while the similarity between data of different clusters is minimised [26] [39]. There are two main approaches to clustering: i) *Crisp clustering method*, and ii) *Fuzzy clustering method*.

Crisp clustering method is used in clustering problems, where the patterns belong only to one cluster. The *fuzzy clustering method* is used in the classification of patterns that may belong to more than one cluster [26].

Examples of areas of application of fuzzy clustering methods are: *Pattern recognition*, *classification*, *database mining*, *image segmentation*, *medical image data*

analysis and *modeling*, *etc*. The most widely used fuzzy clustering algorithm is the *Fuzzy c-Means* (FCM) algorithm proposed by Dunn and generalised by Bezdek.

In fuzzy database mining, fuzzy clustering is used to partition the quantitative attributes of crisp database into several fuzzy sets. As the database size becomes larger and larger, this usually requires a high volume of computations, and considerable amount of memory which may lead to frequent disk access, making the process inefficient. With the development of high performance parallel systems, parallel fuzzy clustering may be used to improve performance and efficiency of fuzzy clustering algorithm [26] [37].

In [26], Kwok et al. present a parallel version of FCM algorithm implemented on an AlphaServer computing cluster with a total of 128 processors and 64 Gigabytes of main memory. 32 Compaq ES40 workstations form the cluster, each with 4 Alpha EV68 processors (nodes) running at 833 MHz with 2 Gigabytes of local main memory. A Quadrics interconnect provides a bandwidth (approx. 200 Mb/sec per ES40) and low-latency (6 msec) interconnect for the processors. The proposed parallel FCM algorithm is written in C, compiled and linked with the MPI library installed on top of the UNIX operating system. The resulted object code is distributed to each processor for parallel execution. In their experimental work, their approach of parallel FCM algorithm demonstrated to reach almost ideal *speedups* and excellent *scaleup* for larger data sets, and it performs equally well when more clusters are requested.

A high speed parallel FCM algorithm for brain tumor image segmentation has been proposed by Murugavalli and Rajamani in [37]. This algorithm converts image into pixel values ($X = \{x_0, x_1, ..., x_{n-1}\}$), then pixel values are equally divided and distributed to all processors so that each processor has *n/p* number of pixels (where *n* is total number of pixel, *p* is a total number of processor) for execution. The membership function is provided for each processor to calculate the degree of membership function, $\mu_{ci}(x_i)$. This algorithm avoids the use of external storage device because each processor uses its own main memory. The initiating processor divides *X* into *n/p*, each processor computes in parallel its center of clusters, its degree of membership matrix (*FCM)*, and each processor defuzzifies its local data and sends it back to the initiating processor to form the segmented image.

4.3 Parallel Mining of Gradual Patterns

In *gradual pattern mining*, the aim is to find dependencies between the *variation* and *direction of change* of attribute values of patterns in the database instead of between the degree of presence or absence of attributes in a transaction [24][22].

Given a database (D_S), constituted of $\boldsymbol{n}$ objects (transactions or data record), described by $\boldsymbol{m}$ numerical attributes. Where $\boldsymbol{T}$=$\{t_1, t_2, ..., t_{\boldsymbol{n}}\}$ are the $\boldsymbol{n}$ transactions, $\boldsymbol{A}$=$\{A_1, A_2, .., A_{\boldsymbol{m}}\}$ are the $\boldsymbol{m}$ numerical attributes, each record t_i is represented as a vector with $\boldsymbol{m}$ values, $[A_1(u_i), A_2(u_i), ..., A_{\boldsymbol{m}}(u_i)]$, such that each $A_h(u_i)$ indicates the value of attribute A_h in the transaction t_i for $h = 1, 2, ..., \boldsymbol{m}$, and $i = 1, 2, ..., \boldsymbol{n}$. Each attribute A_h is a vector with $\boldsymbol{n}$ values, $[u_1, u_2, ..., u_{\boldsymbol{n}}]$.

In this framework, a *gradual pattern* is defined as a relation of simultaneous variation between values of the attributes of two or more gradual items. Denoted as an ordered combination of two or more gradual items of the form: GP=$\{gI_1 gI_2 ... gI_s\}$, such that $gI_1 \neq gI_2 \neq ... \neq gI_s$, for s= $2, 3, ..., \boldsymbol{m}$, where each gI is a *gradual item*.

A *gradual item* is defined as the variation associated to the values of a attribute $A \in D_s$ denoted as $A\mathbf{v}$. Where such variation ($\boldsymbol{v}$) can be ascending ($\geq$) if the attribute values increase, descending ($\leq$) if the attribute values decrease.

In order to measure the strength of the dependency or correlation between the variation and direction of change of attribute values of a gradual pattern/dependency, there are various approaches and each has its own method to compute the support (see [24] [30] and [35] for more details). We comment them briefly as follows:

- Numerical approach: Such as analysis of contingency diagrams by means of techniques from statistical regression analysis, suggested in [22], the validity of the gradual tendency is evaluated from the quality of the regression, measured by the normalised mean squared error R^2, together with the slope of the regression line;
- Qualitative alternative: Count the number of pairs of points $(A_X(u_i), A_Y(u_i))$ and $(A_X(u_j), A_Y(u_j))$ for which $A_X(u_i) < A_X(u_j)$ and $A_Y(u_i) < A_Y(u_j)$, association rules in [36], and fuzzy association rules in [35] are used in order to mine gradual dependencies type $\{$ *the more* A_X, *the more* A_Y $\}$. Other methods and algorithms of this category are: approach based on conflict sets [30], approach based on the precedence graph [28] [29], and approach based on rank correlation measures (GRAANK) [30];
- Numerical-qualitative approaches: This kind of techniques combines properties of both approaches, the numerical and the qualitative one, in order to measure not only the existence of a tendency, but its strength in terms of a *fuzzy rank correlation* measure [24] [41], or terms of fuzzy association rules and fuzzy gradual dependence [35] [36].

Recently, in [28] and [29], Laurent et al. have presented PGP-mc a multicore parallel approach for mining gradual patterns where the evaluation of the correlation and support is based on conflict sets and precedence graph approaches [30]. In this approach, new tasks are dynamically assigned to a pool of threads on a "*first come, first server*" basis.

PGP-mc was implemented using the g++ 3.4.6 and 4.3.2 with POSIX threads, on two different workstations: i) COYOTE machine, with 8 AMD Opteron 852 processors (each with 4 cores), 64 GB of RAM with Linux Centos 5.1. and ii) IDKONN machine, with 4 Intel Xeon 7460 processors (each with 6 cores), 64 GB of RAM with Linux Debian 5.0.2. Experiments were led on synthetic databases au-

tomatically generated by a tool based on adapted version of IBM Synthetic Data Generation Code for Associations and Sequential Patterns. For example, the sequential processing of the 350 attributes database took more than five hours while it spend approximatively 13 minutes using 24 threads on IDKONN. Furthermore, speed-up results are particularly stable from one architecture to another (IDKONN and COYOTE). Detailed results are available on-line at http://www.lirmm.fr/∼ laurent/DASFAA10.

An efficient parallel mining of *closed frequent gradual patterns*, named PGLCM has been proposed by Do et al. en [44]. This approach is based on the principle of the LCM algorithm for mining *closed frequent patterns*, an adaptation of LCM named GLCM in order to mine *closed frequent gradual patterns*, and parallelization of the GLCM algorithm named PGLCM based on the Melinda parallelism environment. It consists of shared memory space, called *TupleSpace*, where all the threads can either deposit or retrieve a data unit called *Tuple*, via two primitives *get(Tuple)* and *put(Tuple)*. All the synchronizations for accessing the *TupleSpace* are handled by the Melinda framework.

The comparative experiment is based on synthetic databases produced with the same modified version of IBM Synthetic Data Generator for Association and Sequential Patterns. All the experiment is conduced on a 4-socket server with 4 Intel Xeon 7460 with 6 cores each, for a total of 24 cores and 64 GB of RAM.
The experimental work has been conducted in two stages, the first one to evaluate the performance of the sequential version of GLCM and PGP-mc (known as Grite), the second one to evaluate the scaling capacities of PGLCM and PGP-mc (known as Grite-MT). Evaluation criteria were the execution time and memory consumption. Where GLCM/PGLCM compute only the *closed frequent gradual patterns*, whereas Grite/PGP-mc compute all the *frequent gradual patterns* (see [28] and [29] for more details).

4.3.1 Parallel Fuzzy Orderings for Fuzzy Gradual Pattern Mining

We addressed the problem of automatically finding correlations between positive and/or negative small variations in the values of attributes affected by noise and non-linear nature. Consequently, we implemented the *Fuzzy Ordering-Based Rank Correlation Coefficient* according to the formal description presented by Bodenhofer and Klawond in [6] and used by Koh and Hüllermeier in [24] and Quintero in [41]. We adopted the idea of storing the fuzzy concordance degrees $C(i, j)$ in sparse matrices. In order to reduce memory consumption, each matrix of concordance degrees $C(i, j)$ is represented and stored according to the *Yale Sparse Matrix Format*, such as only non-zero coefficients are retained.

The evaluation of the correlation, support and generation of gradual pattern candidates are tasks that require huge amounts of processing time, memory consump-

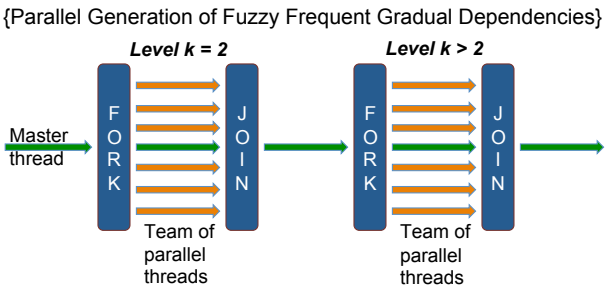

Fig. 2 Parallel Generation of Fuzzy Frequent Gradual Dependencies: OpenMP model-based approach.

tion, and load balance. In this context we addressed these problems using the shared memory architecture API of OpenMP, which is ideally suited for multi-core architectures [42].

The pseudo-code of the Algorithms 1, 2 and 3 shows a simplified view of our parallel process of gradual dependencies mining. The set of fuzzy frequent gradual patterns of output is represented by $\mathscr{F}_{FGP}$, the set of patterns of level 1 is denoted by $\mathscr{L}ist\_gIs$, the fuzzy frequent gradual patterns of level $k = 2$ are represented by $\mathscr{F}_{k=2}$, the fuzzy frequent gradual patterns of level $k > 2$ are represented by $\mathscr{F}_k$ generated from $(k-1)-$frequent gradual patterns in $\mathscr{F}_{k-1}$.

Algorithm 1: Fuzzy Gradual Pattern Mining

Data: Transactions Database D_s, $Set\_of\_Attributes\{A_M\}$, $minSupp$
Result: Fuzzy Frequent Gradual Dependencies $\mathscr{F}_{FGP}$
$\mathscr{F}_{FGP} \leftarrow \emptyset$;
$\mathscr{L}ist\_gIs \leftarrow Gen\_gItems(\{A_M\} \times v\{\leq, \geq\})$;
$k \leftarrow 2$;
$\mathscr{F}_{k=2} \leftarrow GenFuzzyFrequent_{k=2}(\mathscr{L}ist\_gIs, aValues, minSupp)$;
$\mathscr{F}_{FGP} \leftarrow \mathscr{F}_{FGP} \cup \{\mathscr{F}_k\}$;
$k++$;
repeat
  $\mathscr{F}_k \leftarrow GenFuzzyFrequent_k(\mathscr{F}_{k-1}.Matrices, level(k), minSupp)$;
  $\mathscr{F}_{FGP} \leftarrow \mathscr{F}_{FGP} \cup \{\mathscr{F}_k\}$;
  $Delet(\mathscr{F}_{k-1}.Matrices)$;
  k++;
until $\mathscr{F}_{FGP}$ *does not grow any more*;

In our approach, we exploit the advantages of the memory and execution models of OpenMP. Fig. 2 gives an overall view of our approach, where we propose to parallelize the two mains regions of our sequential algorithm, these being the following:

Algorithm 2: Parallel Fuzzy Frequent Gradual Dependency Mining: Level $k = 2$

Data: $List\_of\_gradual\_Items(\mathscr{L}ist\_gIs)$, $Attributevalues(aValues)$, $minSupp$, $\#Threads(Nt)$
Result: Fuzzy Frequent Gradual Dependencies $(\mathscr{F}_{k=2})$
$\mathscr{F}_{k=2} \leftarrow \emptyset$; $q \leftarrow 1$;
/*Each thread computes its fuzzy frequent gradual dependency of level $k = 2$*/;
for ***all***$(thread\ in(0,1,2,...,N-1))$ **do**
  $\mathscr{G}_{DCk=2} \leftarrow GenCand(\{\mathscr{L}ist\_gIs\}, size(k=2), level(k=2))$;
  $Gdc_{k,q} \leftarrow FirstCandidate \in \mathscr{G}_{DCk=2}$;
  foreach $Gdc_{k,q} \in \mathscr{G}_{DCk=2}$ **do**
    $Gdc_{k,q}.Matrix \leftarrow EvaluationFuzzyCorrelation(Gdc_{k,q}, Gdc_{k,q}.aValues)$;
    $Support(Gdc_{k,q}) \leftarrow EvalSupport(Gdc_{k,q}.Matrix)$;
    /* *minSupp stands for a user-specified minimum support value* */
    if $Support(Gdc_{k,q}) \geq minSupp$ **then**
      $Critical\ section: \mathscr{F}_k \leftarrow \mathscr{F}_k \cup \{Gdc_{k,q}\}$;
    $q++$;
    $Gdc_{k,q} \leftarrow NextCandidate \in \mathscr{G}_{DCk}$;

Algorithm 3: Parallel Fuzzy Frequent Gradual Dependency Mining: Level $k > 2$

Data: Fuzzy Frequent Gradual Dependency$(\mathscr{F}_{k-1})$, $level(k > 2)$, $minSupp$, #Threads(Nt)
Result: Fuzzy Frequent Gradual Dependencies $\mathscr{F}_k$
$\mathscr{F}_k \leftarrow \emptyset$; $q \leftarrow 1$;
/*Each thread computes its fuzzy frequent gradual dependency of level $k > 2$ */;
for ***all***$(thread\ in(0,1,2,...,N-1))$ **do**
  $\mathscr{G}_{DCk} \leftarrow GenCand(\{\mathscr{F}_{k-1}\}, level(k))$;
  $Gdc_{k,q} \leftarrow FirstCandidate \in \mathscr{G}_{DCk}$;
  foreach $Gdc_{k,q} \in \mathscr{G}_{DCk}$ **do**
    $Gdc_{k,q}.Matrix \leftarrow$*T-norm*$(Gdc_{k-1,a}.Matrix, Gdc_{k-1,b}.Matrix)$;
    $Support(Gdc_{k,q}) \leftarrow EvalSupport(Gdc_{k,q}.Matrix)$;
    /* *minSupp stands for a user-specified minimum support value* */
    if $Support(Gdc_{k,q}) \geq minSupp$ **then**
      $Critical\ section: \mathscr{F}_k \leftarrow \mathscr{F}_k \cup \{Gdc_{k,q}\}$;
    $q++$;
    $Gdc_{k,q} \leftarrow NextCandidate \in \mathscr{G}_{DCk}$;

- Automatic extraction of frequent gradual patterns of level k=2. The pseudo-code of the Algorithms 2 shows the corresponding parallel region.
- Automatic extraction of frequent gradual patterns of level k>2. The pseudo-code of the Algorithms 3 shows the corresponding parallel region.

Open Multi-Processing (OpenMP) is a shared memory architecture API, that supports multi-platform for writing shared memory parallel applications in C, C++, and Fortran on many architectures, including UNIX and Microsoft Windows platforms. It consists of a set compiler directives, runtime routines, and environment variables that influence runtime behaviour [47] [48].

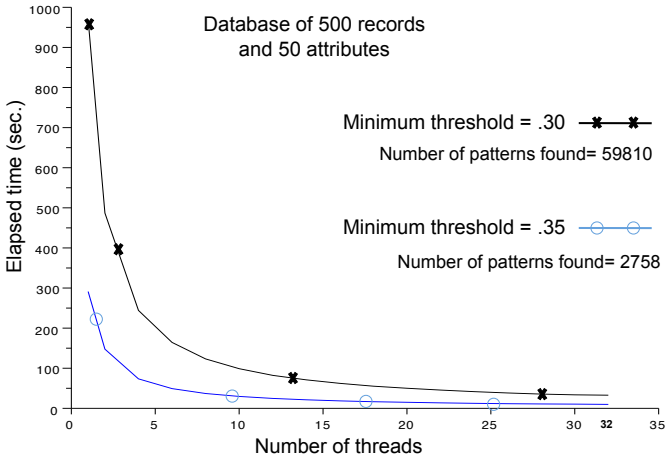

Fig. 3 Threads vs. elapsed time with a database of 500x50 and minSupp=.30 and .35, using uncompressed binary matrices of concordance degrees.

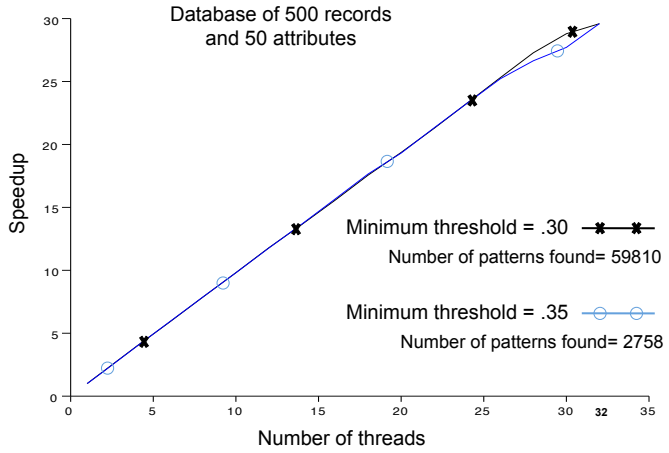

Fig. 4 Speedup with a database of 500x50 and minSupp=.30 and .35, using uncompressed binary matrices of concordance degrees .

OpenMP provides a multi-threaded capacity, where a loop can be parallelized easily by invoking subroutine calls from OpenMP thread libraries and inserting the OpenMP compiler directives. In this way, the threads can obtain new tasks, the unprocessed loop iterations, directly from local shared memory and data can be shared or private [42] [47] [48].

Experiments

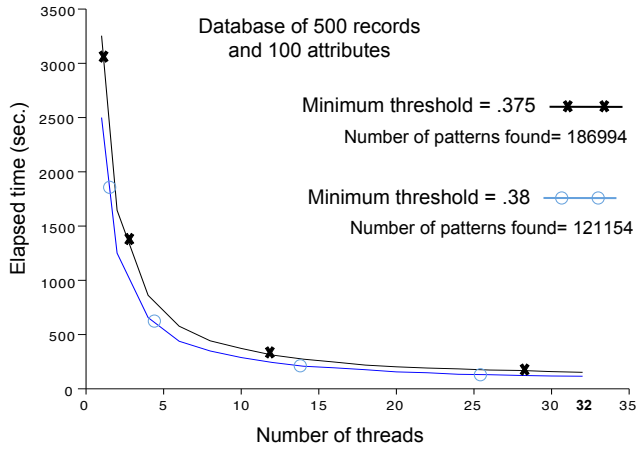

Fig. 5 Threads vs. elapsed time with a database of 500x100 and minSupp=.375 and .38, using uncompressed binary matrices of concordance degrees.

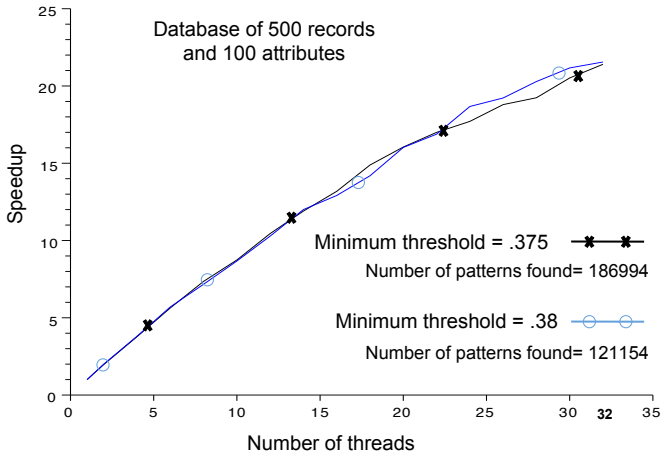

Fig. 6 Speedup with a database of 500x100 and minSupp=.375 and .38, using uncompressed binary matrices of concordance degrees.

We present an experimental study of the scaling capacities of our approach on several cores, for the database C500A50 with 500 records and 50 attributes, and database C500A100 with 500 records and 100 attributes, which were used in [28] [44] and produced with the IBM Synthetic Data Generator for Association and Sequential Patterns.

Our experiments were performed on a workstation with up to 32 processing cores, named COYOTE, with 8 AMD Opteron 852 processors (each with 4 cores),

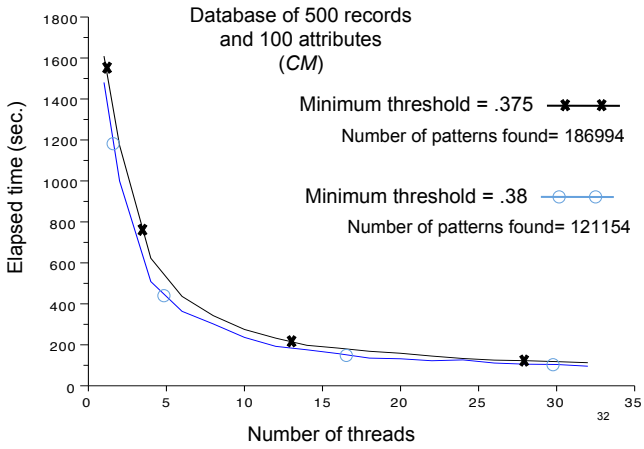

Fig. 7 Threads vs. elapsed time with a database of 500x100 and minSupp=.375 and .38, using compressed matrices of concordance degrees.

64 GB of RAM with Linux Centos 5.1, GCC OpenMP 3.1.

Results

The first experiment involves a database with 500 lines and 50 attributes, Figures 3 and 4 depict the execution time and speed-up related to: 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, and 32 threads, on the test database, from which were found 59810 frequent gradual patterns for a minimum threshold of 0.30, and

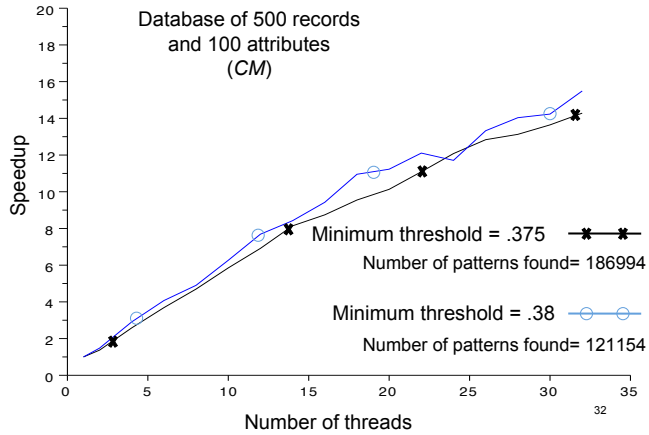

Fig. 8 Speedup with a database of 500x100 and minSupp=.375 and .38, using compressed matrices of concordance degrees.

2758 frequent gradual patterns for a minimum threshold of 0.35. In the first case we reach a memory consumption of 12.6% with uncompressed binary matrices of concordance degrees and 3.5% with compressed matrices of concordance degrees (Yale Sparse Matrix Format). While for the second case we obtained a memory consumption of 0.6% with uncompressed binary matrices of concordance degrees and of 0.3% with compressed matrices of concordance degrees.

The second experiment involves a database with 500 lines and 100 attributes, Figures 5 and 6 depict the execution time and speed-up related to: 1, 2, 4, 6, . . . , to 32 threads, on the test database, from which were found 186994 frequent gradual patterns for a minimum threshold of 0.375, and 121154 frequent gradual patterns for a minimum threshold of 0.38. In the first case we reach a memory consumption of 36.2% using uncompressed binary matrices of concordance degrees and of 14.4% using compressed matrices of concordance degrees (Yale Sparse Matrix Format). While for the second case we obtained a memory consumption of 24.7% with uncompressed binary matrices of concordance degrees and of 10.3% with compressed matrices of concordance degrees. Within this experimental framework, Figures 7 and 8 illustrate the execution time and speed-up of our approach using compressed matrices of concordance degrees.

4.4 Parallel Mining of Fuzzy Trees

With the development of Internet and Web, frequent pattern mining has been extended to more complex patterns like tree mining, graph mining, and fuzzy tree mining. Such applications arise in complex domains like bioinformatics, Web mining, banking, marketing, biology, health, *etc*. especially to handle complex databases such as semi-structured data or tree databases (for example in the case of XML databases) [9] [10].

Tree mining consists in discovering all the frequent subtrees F_S from a database D of trees [10], as shown on Fig. 9. The frequency is computed using the notion of support: Given a database D, the *support* of a tree S is the proportion of trees T from D where S is included:

$$Support(S) = \frac{\#\ of\ trees\ T\ where\ S\ is\ embedded}{\#\ of\ trees\ in\ D}$$

S is said to be frequent if *Support(S)* $\geq \sigma$ where σ is a user-defined minimal support threshold. There are two types of inclusion: induced inclusion and embedded inclusion, see Fig. 10, where a tree S is included in another tree T of a database D of trees, if all nodes in S are included in T.

Fuzzy approaches have been proposed in order to soften the constraint on the patterns (*frequent subtrees*) found by the algorithms. In fuzzy tree mining a tree S is included in another tree T of a database D of trees, with a degree of inclusion $\tau(S,T)$.

Four types of fuzzy inclusion has been proposed: *Ancestor-descendant degree*, *Sibling ordering degree*, *Partial inclusion*, and *Node similarity*. A detailed treatment of these approaches is given in [10] [31]. The frequency is computed using the notion of *fuzzy support*: Given a database D and a tree S, the *support* of S in D is given by:

$$Support(S) = Agg_{T \in D}(\tau(S,T))$$

where Agg is a function of aggregation. For instance, the average is:

$$Support(S) = \frac{sum_{T \in D}(\tau(S,T))}{\#\, of\ trees\ in\ D}$$

The core of the process for fuzzy tree mining is briefly described in algorithm 4. Several methods have been proposed for generating candidates from frequent subtrees. Most of the methods rely on the construction of candidates by using an extension on the right most branch. The trees are numbered in a depth-first enumeration from the root to the right most leaf. Then, for every frequent tree of size k (containing k nodes), candidates are generated by adding a node on the right after considering all the possible 2-trees whose first node corresponds to the anchoring node.

Recently, [43] have developed *PaFUTM: Parallel Fuzzy Tree Mining* a parallel version of algorithm 4. Fig. 11 illustrates the general structure of *PaFUTM* where the computation of the fuzzy support is parallelized using a pool of 1 to 32 threads and a dynamic queue of tasks type "*first come, first served*". For each level k, potential $k+1$ candidates are queued. Then each idle thread extract a non-processed candidate, calculates its frequency and fuzzy support according to a fuzzy inclusion $\tau(S,T)$ type *Ancestor-descendant degree*. This fuzzy inclusion is defined by a discrete fuzzy set interpreted as a fuzzy scope for the ancestor-descendant relationship "*scope no more than 5 nodes"*.

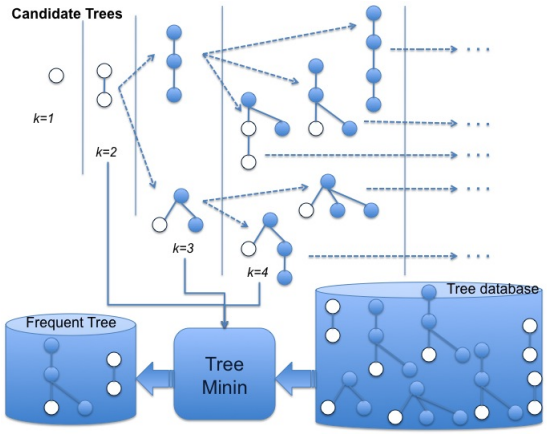

Fig. 9 Scheme of process of tree mining.

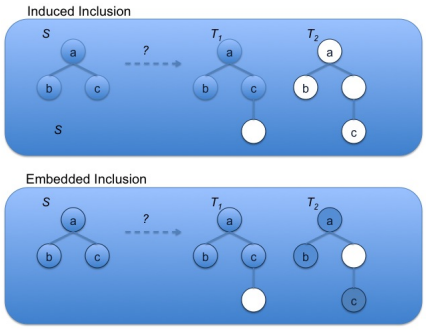

Fig. 10 Types of inclusion of trees.

Algorithm 4: Fuzzy Frequent SubTrees Mining

Data: Tree Database D
Result: Fuzzy Frequent Subtrees $\mathscr{F}$
$\mathscr{F} \leftarrow \emptyset$;
$k \leftarrow 1$;
repeat
  $\mathscr{S}_k \leftarrow Gen\_Cand(k)$;
  foreach $s \in \mathscr{S}_k$ **do**
    $Support(s) \leftarrow 0$;
    foreach $T \in D$ **do**
      / If degree of fuzzy inclusion is relevant, computing Support(s) */*
      if *Fuzzy_ inclusion_ degree* τ(s,T) **then**
        Support(s) = $Agg_{T \in D}(\tau(s,T))$;
    / minSupp stands for a user-specified minimum support value */*
    if $Support(s) \geq minSupp$ **then**
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{s\}$;
  k++;
until $\mathscr{F}$ *does not grow any more*;

PatFUTM was implemented using the g++ 3.4.6 and evaluated with POSIX threads, on a 32-core machine, with 8 AMD Opteron 852 processors (each with 4 cores), 64 GB of RAM with Linux Centos 5.1, g++ 3.4.6. And evaluated with two types of datasets: *B* datasets (*BA*, *BB* and *BC*) that contain lots of relatively small trees and *C* datasets (*CG*, *CH*, *CJ*) that contain a smaller amount of larger trees.

5 CONCLUSION

In this paper, we discuss the importance of the scalability of fuzzy systems in general and the scalability of fuzzy database mining algorithms in particular. We analyze the

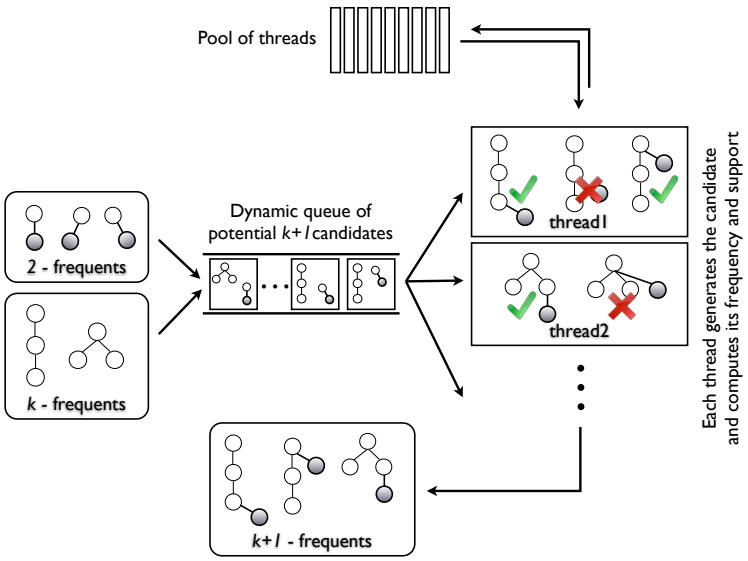

Fig. 11 Parallel fuzzy tree mining.

possibilities of scalability offered by architectures of multi-core processors and its potential for parallel processing. We present a study of parallel programming of fuzzy database mining algorithms based on Multithreading.

We present our review of the parallelization of four fuzzy database mining algorithms on multi-core architectures, namely fuzzy association rules, fuzzy clustering, fuzzy gradual dependencies, and fuzzy tree mining. This in order to substantiate our position that parallelization can help address the problem of search space and requirements of computation time and memory for fuzzy models that are larger than of crisp models.

We focus mainly on the parallelization of fuzzy database mining algorithms on multi-core architectures of shared memory and on multi-core architectures of distributed memory. The algorithms that we are interested in researching their parallelization are: Fuzzy orderings-based extraction of gradual patterns, fuzzy orderings-based extraction of sequential gradual patterns. Parallel programming models we are interested in exploring are: Task Parallelism, Data parallelism, and Task-Data parallelism.

References

1. Bao-wen, X., Jian-jiang, L., Yingz-hou, Z., Lei, X., Huowang, C., Hong-ji, Y.: Parallel algorithm for mining fuzzy association rules. In: Proc. of International Conference on Cyber-

worlds, IEEE, 2003.
2. Barney, B.: Introduction to Parallel Computing, Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/parallel_comp/#ModelsData
3. Basterretxea, K., Del Campo, I.: Electronic Hardware for Fuzzy Computing. In: A. Laurent and M.-J. Lesot (eds.) Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, pp. 1-30. Information Science Reference, 2010.
4. CUDA Training: Cuda Parallel Programming Model Overview. In Downloadable CUDA Training Podcast. http://www.developer.nvidia.com/cuda-training, 2012.
5. CUDA-NVIDIA: What is GPU computing?. In GPU Computing Solutions. http://www.nvidia.com/object/GPU_Computing.html, 2012.
6. Bodenhofer, U., and Klawonn, F.: Roboust Rank Correlation Coefficients on the Basis of Fuzzy Orderings: Initial Steps. In Mathware & Soft Computing 15, 5-20, 2008.
7. Data Mining, Analytics, and Databases. In GPU Computing Solutions. http://www.nvidia.com/object/data_mining_analytics_database.html. 2011.
8. Delgado, M., Marin, N., Martín-Bautista, M., J., Sánchez, D., Vila, M.-A.: Mining Fuzzy Association Rules: An Overview. In Soft Computing for Information Processing and Analysis: Studies in Fuzziness and Soft Computing, Volume 115 / 2005 - Volume 276, Springer 2005.
9. Del Razo, F., Laurent, A., Poncelet, P., Teisseire, M.: Fuzzy Tree Mining: Go Soft on Your Nodes. In: Foundations of Fuzzy Logic and Soft Computing, 12th International Fuzzy Systems Association World Congress IFSA, pp. 145-154. Springer, Heidelberg Lecture Notes in Computer Science, 2007.
10. Del Razo, F., Laurent, A., Poncelet, P., Teisseire, M.: FTMnodes:Fuzzy tree mining based on partial inclusion. Elsevier, ScienceDirect Fuzzy sets and systems. 2009.
11. Fang, W.-F., LU, M., Xiao X., He, B., Luo, Q.: Frequent Itemset Mining on Graphics Processors. In Proceedings of the Fifth International Workshop on Data Management on New Hardware (DaMoN 209), ACM, 2009.
12. Freitas, A. A.: A Survey of Parallel Data Mining. In 2nd International Conference on the Practical Applications of Knowledge Discovery and Data Mining, pp. 287-300, 1998.
13. Golkar, C.: Predictive In-Database Analytics Bringing Analytics to the Data. In Fuzzy Logix. www.fuzzyl.com/products/in-database-analytics/, 2011.
14. Golkar, C.: Fuzzy Logix Unveils NVIDIA GPU-Based Analytics Appliance- The Tanay ZXnW Series. www.fuzzyl.com/ press-releases/ fuzzy-logix-uneils-nvidia-gpu-based-analytics-appliance/, 2011.
15. Hall, L., O., Goldgof, D., B., Canul-Reich, J., Hore, P., Cheng W., Shoemaker, L.: Scaling Fuzzy Models. In: A. Laurent and M.-J. Lesot (eds.) Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, pp. 31-53. Information Science Reference 2010.
16. Hirota, K., Pedrycz, W.: Fuzzy Computing for Data Mining. Proceedings of the IEEE, Vol. 87, No. 9 Septembre 1999.
17. Hong, T., P., Lee, Y., C., Wu, M., T.: Using the master-slave parallel architecture for genetic-fuzzy data mining. In: Proc. IEEE International Conference on Systems, Man and Cybernetics, 2005.
18. Hughes, C., Hughes, T.: Professional Multicore Programming: Design and Implementation for C++ Developers. Wrox Wiley Publishing, Inc. 2008.
19. Hüllermeier, E.: Fuzzy Sets in Machine Learning and Data Mining. Applied Soft Computing Journal, 11:1493-1505, 2011.
20. Hüllermeier, E.: Why Fuzzy Set Theory is Useful in Data Mining. In Successes and New Directions in Data Mining, IGI Global, 2008.
21. Hüllermeier, E.: Fuzzy Methods in Machine Learning and Data Mining: Status and Prospects. Fuzzy Sets and Systems 156(3), 387-407, 2005.
22. Hüllermeier, E.: Association Rules for Expressing Gradual Dependencies. In PKDD, LNAI 2431, Springer-Verlag Berlin Heidelberg 2002.
23. Jang, J.-S. R., Sun, C.-T., Mizutani, E.: Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. Prentice Hall Engineering Science Mathematics, 1997.

24. Koh, H., W., Hüllermeier, E.: Mining Gradual Dependencies based on Fuzzy Rank Correlation. In: Proc. SMPS 2010, 5th Int. Conf. on Soft Methods in Probability and Statistics. Oviedo/Mieres (Asturias), Spain, October 2010.
25. Kim, S.: A GPU Based Parallel Hierarchical Fuzzy ART Clustering. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE-Computational Intelligence Society, 2011.
26. Kwok, T., Smith, K., Lozano, S., Taniar, D.: Parallel Fuzzy c-Means Clustering for Large Data Sets. In Euro-Par, LNCS 2400, Springer-Verlag Berlin Heidelberg, 2002.
27. Jian-jiang, L., Bao-wen, X., Xiao-feng, Z., Da-zhou, K., Yan-hui, L., Jin Z.: Parallel mining and application of fuzzy association rules. In Higher Education Press and Springer-Verlag, 2006.
28. Laurent, A., Negrevergne, B., Sicard, N., Termier, A.: PGP-mc: Towards a Multicore Parallel Approach for Mining Gradual Patterns. In: Proc. DASFAA, 2010.
29. Laurent, A., Negrevergne, B., Sicard, N., Termier, A.: Efficient parallel mining of gradual patterns on multicore processors. In AKDM-2, Advances in Knowledge Discovery and Management. Vol. 2. Springer. 2010.
30. Laurent, A., Lesot, M., J., Fifqi, M., GRAANK: Exploiting Rank Correlations for Extracting Gradual Itemsets. In FQAS 2009, LNAI 5822, Springer-Verlag Berlin Heidelberg 2009.
31. Laurent A., Poncelet, P., Teisseire, M.: Fuzzy data mining for the semantic web: building XML mediator schemas. In: Fuzzy Logic and the Semantic Web, pp. 249-265. Elsevier, Amsterdam, 2006.
32. Lin, N., P., Chueh H., E.: Fuzzy Correlation Rules Mining. In: Proceedings of the 6th WSEAS International conference on Applied Computer Science, Hangzhou, China, 2007.
33. Ma, Z., M., Yan, L.: A Literature Overview of Fuzzy Database Models. In Journal of Information Science and Engineering 24, 2008.
34. Martin, T., Shen, Y.: Fuzzy Association Rules to Summarise Multiple Taxonomies in Large Databases. In: A. Laurent and M.-J. Lesot (eds.) Scalable Fuzzy Algorithms for Data Management and Analysis: Methods and Design, pp. 273-301. Information Science Reference (2010)
35. Molina, C., Serrano, J., M., Sánchez, D., Vila, M., A.: Mining Gradual Dependencies with Variation Strength. In Mathware Soft Computing 15, 2008.
36. Molina, C., Serrano, J., M., Sánchez, D., Vila, M., A.: Measuring Variation Strength in Gradual Dependencies. In: Proceedings of the 5th EUSFLAT Conference Contents of Volume I, New Dimensions in Fuzzy Logic and Related Technologies, 2007.
37. Murugavalli, S., Rajamani, V.: A High Speed Parallel Fuzzy C-Mean Algorithm for Brain Tumor Segmentation. BIME Journal, Volume (06), Issue (1), Dec. 2006.
38. Petry, F. and Bosc, P.: Fuzzy databases: principles and applications. Kluwer Academic Publishers, 1996.
39. Polimi, D.: A Tutorial on Clustering Algoritms: Fuzzy C-Means Clustering. In http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html.
40. Piatetsky-Shapiro, G. and Frawley, W.J.: Knowledge Discovery in Databases, AAAI Press / The MIT Press, 1991.
41. Quintero, M., Laurent, A., Poncelet, P.: Fuzzy Ordering for Fuzzy Gradual Patterns. In FQAS 2011, LNAI 7022, Springer-Verlag Berlin Heidelberg, 2011.
42. Rauber, T., Rnger, G.: Parallel Programming: for Multicore and Cluster Systems. Springer-Verlag Berlin Heidelberg, 2010.
43. Sicard, N., Laurent, A., Del Razo, F., Quintero Flores, P., M.: Towards Multi-Core Parallel Fuzzy Tree Mining. In: FUZZ-IEEE'2010, IEEE World Congress on Computational Intelligence, IEEE Computational Intelligence Society, 2010.
44. Thac, Do, T., D., Laurent, A., Termier, A.: PGLCM: Efficient Parallel Mining of Closed Frequent Gradual Itemsets. In: Proc. International Conference on Data Mining (ICDM), 2010.
45. Timothy, J., R.: Fuzzy Logic with Engineering Applications. John Wiley Sons Ltd. West Sussex, PO19 8SQ, United Kindom 3nd, 2010.
46. Touzi, A., G., Ben Hassine, A., B.: New Architecture of Fuzzy Database Management Systems. In The International Arab Journal of Information Technology, vol. 6, No. 3, 2009.

47. Yang, C.-T., Huang, C.-L., Lin C.-F.: Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. Computer Physics Communications, ELSEVIER, Volume (182), pp. 266-269, Dec. 2011.
48. Van der Pas, R.: An Overview of OpenMP. In OpenMP the OpenMP API specification for parallel programming. http://openmp.org/wp/resources/Tutorials. 2011.
49. Van der Pas, R.: Basic Concepts in Parallelization. In OpenMP the OpenMP API specification for parallel programming. http://openmp.org/wp/resources/Tutorials. 2011.
50. Wen, C., H., Chen Y., L.: Mining fuzzy association rules from uncertain data. Knowledge and Information Systems, Volume 23, Number 2, Springer, 2010.
51. Zadeh, L., A., Hirota, K., Klir, G., J., Sanchez, E., Wang, P.-Z., Yager, R., R.: Advances in Fuzzy Systems: Applications and Theory. World Scientific, 2011.