About Parallel Programming: Paradigms, Parallel Execution and Collaborative Systems

Loredana MOCEAN, Monica CIACA
Business Information Systems Department,
Babeş-Bolyai University of Cluj-Napoca, Romania
{loredana.mocean|monica.ciaca}@econ.ubbcluj.ro

In the last years, there were made efforts for delineation of a stabile and unitary frame, where the problems of logical parallel processing must find solutions at least at the level of imperative languages. The results obtained by now are not at the level of the made efforts. This paper wants to be a little contribution at these efforts. We propose a overview in parallel programming, parallel execution and collaborative systems.
***Keywords**: Parallel Programming, Parallel Execution, Collaborative systems, Collaborative parallel execution*

1 Introduction

The need of influence on a large scale from sequential programming to simultaneous processing takes at least the following basic motivations:

- Natural barrier dictated by the speed of electric current, that, irrespective of future performance of one processor, makes impossible the unlimited increase of uni-processor system capacity and efficiency performing.
- Inherent kind and strong parallel of many algorithms; this kind of algorithm suggest the designing of a program by initiation of multiple processes, which co-operate to fulfill a common purpose.

Investigations about methods through which informatics community can make this forced passing started during the seventh decade ([1], [20], [18]). Unlike of the other problems which found a much quicker solution, parallel programming didn't succeed yet to be imposed as general way of designing, implementation and performing of algorithms. Problems with a new approach proved extremely difficult to be solved, so at the level of theoretic substantiation and especially at the level of conceptions of the programmers community. Learning and profound study by the mass of informatics community of an inherent sequential algorithmic approach used at the level of imperative programming languages(actually welled from the inherent sequential kind of our every day actions) it makes that this forced transition to become extremely complicated.

1.1 Purpose of the study

Parallel algorithms and collaborative systems are two concepts which exist also in theory and practice of global marketplace but currently, there is no effective approach to find the link between collaborative systems and data parallelism and algorithms.

The purpose of this study is to develop concepts to describe, conceptualize and analyze the synergetic aspects between parallelism and collaborative systems, from the point of view of the researcher, the programmer and the user. In this study, existing concepts and theoretical models describing parallelism are explored and brought into the specific context of collaborative systems and its software components. The research includes preliminary theory about parallelism. The theory building in this study is about using the existing conceptualizations and theoretical models, and on the basis of these, compiling a conceptualization that is applicable in the specific research branch. The purpose of this study is also helpful for the businesses and software companies that desire to implement or upgrade their business information systems ERP basis systems, enhance competitive capacity and actively dispose global strategy, especially in the current trend of the postal communication, collaboration, coordination and communication.

1.2 Materials and Methods

In our research we followed some important lines about:

- Appreciation and understanding of the reader;
- Scope of our research;
- Lack of methodology;
- The existing literature and critical thinking about the written articles and books in this domain;
- Useful aspects after the paper is ready;

• Advantages and the possibility of continuing the research in this field of activity
• First of all we studied the theoretical concepts about parallel programming and collaborative systems. We studied these concepts from many books and we took part at many presentations, PhD Thesis and Workshops on this research branch;
• In our research we also used published studies on the Internet and extensive websites;
• The synergetic aspects between data parallelism, data programming and collaborative systems were hard to study;
• The documentation base is being provided by multiple sources, one of them being the free articles and academic magazines offered by libraries and on-line databases.

1.3 Results

In a logical way there are two practical possibilities of passing from sequential programming to the parallel one:

1. *projecting algorithms* that are create from the beginning in parallel approach and then to implement it at the level of some programming languages especially projected for this kind of performing;
2. *elaborating specialized software* for automatic transform of sequential programs in efficient parallel versions of them (see figure 1).

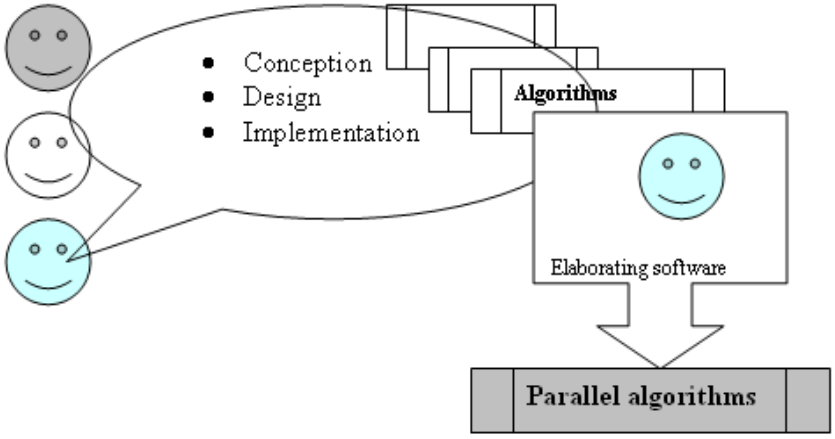

Fig. 1. Possibilities of passing from sequential programming to the parallel one

Even if we think that the future will impose finally, because of necessity, the first variant it is noticed that for the actual period only the level of technological development of hard equipments could contribute of quick finalize of this type of transition (even if there are no architectural standards for parallel computers). Unfortunately ingenious and especially correct administration of the resources involved at the level of an algorithm prove to be quite difficult when it is desirable parallel approach of algorithm elaboration and that's way it remains more a research domain than a practical methodology well edit and universal accepted. For this reasons the largest part of the actual research is oriented on the parallel processing and it is directed on the development and analyze of some theoretic methods that allow substantiation of some general constructive principles as well as efficient implementation of some reorganized compilers (translators that reorganize a sequential program for parallel execution). We consider that for the actual stage this approach is the most suitable. It comes to support the ones that already have impressive soft resources which is implementation it becomes inefficient with the increasing of the problems complexity that must be solved. On the other hand is very important the psychological aspect of the contact of the programmers with already "parallel programs" that perform efficient. The study and the successfully use of this programs will bring in an objective way acquiring the necessary experience and interest growing of the programmers for directly parallel approach in projecting their application. Even if in the first phase this practice will be adopted (and already is!) only for the advanced and special educated part of the programmers in this purpose, *developing algorithms in parallel approach that will remain for sure in time the unique methodology of program that could justify and realize the advance of software informatics.* For the actual step, in the favor of developing of some **parallelizing translators**, it comes the argument that usually this realizes the extraction of maximum possible parallelism having as simultaneous objects also the generating and optimizing in parallel code. That's why that can be used inclusive

for optimizing of some parallel source programs, developed from it's very beginning like this for the programmers, but whom it might miss the optimum performing characteristic.

2 Parallel programming paradigms

The analyze of parallel processing possibilities it leads on identifying some programming paradigms well outlined expressing the essence of classifying criterions of the languages that integrates this kind of facilities.

For the execution of a program at the level that it is desirable the parallel processing, the programming language it must offer ways for (see also figure 2):

1. identifying the parallelism that is materialized by decisions related to the components of the program that will be (potential) performed by different processors;
2. initiating and finalizing the parallel processes;
3. coordinating parallel execution by specifying and implementing the ways of interaction between the components of the program planned to execute in parallel.

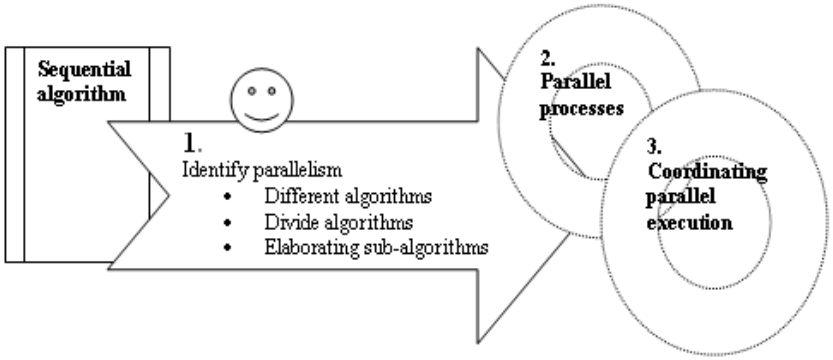

Fig. 2. Steps in parallel processing

It is obvious that these facilities might be thought and implemented at the level of some programming language or in an explicit way or in an implicit one creating two types of parallelism: explicit parallelism and implicit parallelism.

Explicit parallelism is characterized by the presence of some explicit syntactic constructions in the programming languages we are talking about, having the mission to describe until a certain level of detailing, the modality in which will take place the parallel perform.

For this meaning we can identify in time different approach mechanisms of explicit parallel perform and we can remember here the primitives like fork-join, traffic lights, parabegin-parend mechanism and so on.

Even if this promote a high level of flexibility(in the meaning that exists the possibility of describing of any parallelism form based on this primitives) they have the disadvantage of let in programmer's task the majority of the decisions connected with the identifying and concrete expression of parallelism at the level of a program.

That's why, in time had been proposed diverse sophisticate approaches, trying to promote some facilities with a higher level of abstraction and though to prove that are useful in a practical way at the programmer's level. This proposals began from developing some specialized libraries in primitives of communication with the role of diminish the complexity of parallelism administration (PVM [21] and Linda[11]) until to the development of some special languages in this meaning, like PCN language[10].

Implicit parallelism allow the programmers writing some source code without their implication in the decisions connected with parallelism exploitation at the level of the program. This is automatic perform by the compiler or by the execution environment leading at obtaining some transparency of parallelism manifestation confronted by the programmer and succeeding to maintain like this the complexity of developing parallel programs at an identical level with the one of developing purely sequential programs.

Implicit extraction of parallelism, even is the most comfortable choice for the programmer, isn't at all an easy task to accomplish. According to what we show in previous section, in the case of imperative languages of programming, the complexity of the problem is nearly prohibitive, allowing to obtain acceptable results only for a limited class of applications (the ones that use intensely repetitive operations over the keyboard/control panel – so using cycles with known number of steps)[24].

Fortunately, the situation is different in the case of declarative programming languages. Function-

al and logical programming languages are characterized with a higher level of abstraction, allowing to the programmer to focus on the description of problem's requirements and not on relative details about the way its solution must be obtained. In this meaning the declarative languages (even if they appeared later in the programming languages spectrum and maybe that's why insufficient exploited until now from the point of view of their algorithmic potential) had opened new perspectives concerning the possibility of automatic exploit of parallelism.
The higher level of abstraction of these languages, as well as their strongly mathematic nature becomes strong advantages in offering the possibility of an automatic exploitation of parallelism. In private, declarative programming languages are characterized by:

- referential transparency, this meaning that the variables are looked as mathematics entities whom value won't modify during the performing(that's why declarative languages are characterized as programming languages that adopt the single assignment rule-single assignment languages);
- their operational semantic is based on a certain form of indeterminism (the selection of clauses in logical programming languages and respectively operators like *apply* in the case of functional languages);
- the possibility of using some immediate evaluation schemes(eager evaluation schemes)that allow to obtain dataflow-like computations extremely proper to a parallel performing.

These three reasons are the main causes of the conclusion that declarative programming languages show a implicit parallelism a lot more higher than imperative programming languages.
Despite of these theoretic advantages, the so-called performing can be inefficient because of the fallowing potential disadvantages:

- in a declarative system misses the information about the dimension of participants components in a computational process. So, like this you can decide the parallel execution of some components for that (because of too fine granulation of the tasks) the control of parallel execution is more expensive than the potential benefits of speed that parallel execution could bring;
- if in those particular cases we do not apply complete analysis techniques of the existing dependences between the components (as we have seen in the previous section we cannot obtain in the general case, the problem being the complete NP) than the system can try to make a code sequence parallel even though that sequence is seemingly already parallel, decision that brings inefficiency because of the need to synchronization and communication between processes.

An ideal parallel execution system should allow the automatic extraction of a very high degree of parallelism (request that is compatible with the basic operational principles of declarative programming) without restricting the possibility for the programmer to intervene if a certain situation demands it (this last request does not comply with the semantic requests of most declarative programming).
Only a few declarative paradigms [23] seem to offer the availability and flexibility needed for an immanent parallel execution: the pure objectual programming ([22],[23]) some forms of functional programming [16] and *logical programming* [13]. Among these, logical programming establishes itself as the most adequate for obtaining an ideal parallel behavior.

3 Parallel execution in logical programming languages

Logical programming offers clear opportunities for the implicit exploitation of parallelism. This becomes obvious if we analyze its operational semantic. The resolution algorithm displays different degrees of indetermination, identifying many moments of the execution in which different alternatives for the continuation of the deductive computational process are possible. In a sequential implementation these decisions are serielized, being explored through an order that depends on the definition of the selection operations. Parallel execution uses the possibility to explore these alternatives concomitantly, without affecting the programs semantic.
Moreover, logical programming allows different natural ways for the programmer to explicitly intervene by introducing different annotations in the source code, annotations that keep the structure and the semantic of the program.
The parallelism forms that can be identified in the logical programs depend on the selection operations that are decided to be possible to be executed simultaneously. Thus, in logical programming there are to parallelism forms that can be identified [8]:

1. ***OR parallelism*** **–** this parallelism appears when we simultaneously execute the stipulations that can be used as alternatives for solving the selected purpose. Each (sub) purpose can be solved using more stipulations (precisely using any clause whose heading can be united with the re-

spective purpose).

2. ***AND parallelism*** **–** another vision of the previous mentioned parallelism can be obtained through the consideration of the indetermination that is present in the sub purpose selection of a stipulation. In certain conditions, it is possible to try satisfying a stipulation through the concurring resolution of its constituent sub purposes.

Different possible parallelism forms have been identified, an example being the *unifying parallelism*, but because of its sensible granulation, the efficiency of this parallelism is conditioned by the implementation in a specialized architecture. A more thorough discussion of the aspects involved in this type of parallelism can be found in [3].

Regardless of the positive aspect that have been mentioned until now relating to the considerable potential of the logical programming languages to take on the parallel execution we have to recognize that there are many aspects still to be considered in this matter. Next, we will enumerate these aspects followed-up by a short cause of the persistent problems.

4 Efficiency

Even though the specialized literature has many propositions for exploiting parallelism in a logical program, very few of these propositions have proven themselves to be of practical use. The main factor for this is that in reality the main concern is *efficiency* that is understood in the logical programming languages as:

I. The ability to minimize the costs for the parallelization overhead up to a level where the cost becomes insignificant compared to the profits that are brought through the parallel execution.

II. If we are talking about a forced sequential execution of a parallel version that is caused by lack of resources then we should obtain efficiency comparable with the best implementation known for the sequential version.

The main request for a parallel implementation is that the parallel execution shouldn't be any slower than the corresponding sequential version. In real life this request is hard to obtain for the general case because the exploitation overhead for parallelism can make certain operations to be more expensive (slower) than the sequential case (backtracking is a common example). Planning efficient logical parallel programming systems is a complex problem that continues to remain an unsolved issue.

5 Multi-paradigm systems, Lack of optimization and Integration with other models

There are efficient implementations that only exploit one form of parallelism [13], but systems that want to exploit more than one form have certain disadvantages, such as:

- They count on implementation schemes that lead to sequential efficiency loss ([15], [4], [3]) leading to inefficient implementations;
- They limit the amount of potential parallelism to be exploited;
- They change the language semantics to obtain an easier exploitation of parallelism ([9], [22]).

Even the models that are based on standard sequential implementation (like WAM- Warren Abstract Machine)[Warr83] can display a higher degree of inefficiency due to the promotion of parallel execution. On the other hand, the advantages of parallel execution are often planned for the most disadvantageous cases, cases that rarely appear in real life programming. The weak performance of many implementations comes from not including some optimization functions that adjust the parallelism exploitation cost to the complexity of every particular case.

Logical programming has evolved a lot in the last years bringing up new models, capable to give solutions more efficiently to different type of problems that appear in parallel implementation. Thus, considering *constraint handling* and *data parallelism* represent only two examples of models that can be integrated in between other parallel implementations of logical programming languages.

6 Data parallelism and collaborative systems

Data parallelism (also known as loop-level parallelism) is a form of parallelization of computing across multiple processors in parallel computing environments. Data parallelism focuses on distributing the data across different parallel computing nodes. It contrasts to task parallelism as another form of parallelism [28].

During the last decade, all the database systems included in their components parallel processing functions. The evolution was generated by the growing in a considerable rhythm of the database dimensions. The rhythm of growing of the database dimensions surpass the growing in speed of the processors, in capacity of the memories, so that are necessary supplementary actions in order to outface to the happened effects behind the fast growing of the data volume.

The parallelism help to the constant multiplication of the data even if is considerable enlarged

the dimension of database.
This effect is important in loading data, index creating, administration operations and applications which use grand lots of data.

In the perspective of applications of data parallelism, the principal's uses of data parallelism are shown in the next figure.

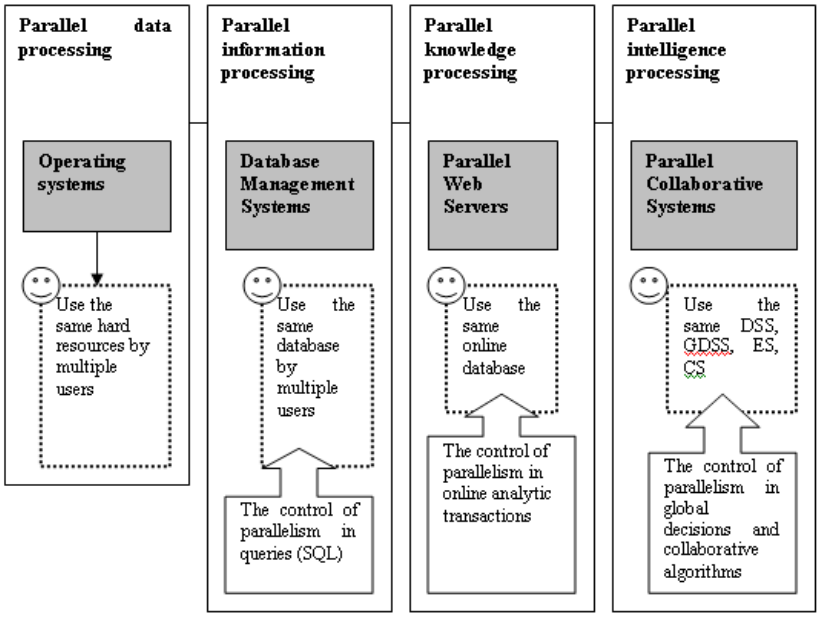

Fig. 3. The uses of data parallelism

The combining of computer networks and parallel algorithms had inspired a new class of applications which support the collaborative activities. In a collaborative application participants are hired in a series of transactions for achieve a common scope. The collaboration brings new requests in computational parallel services.
The most important is the communication between the computers and then the distribution of information which must permit identifying the person and the location, hard resources, communication protocol, validation of communication and collaboration between user's applications.
When we think to a collaborative system based on parallel data and applications, we must think to the following:

- The organizing of parallel services
- The management of parallel information
- The group parallel communication
- The parallel management of the team
- The parallel management of the process
- The parallel management of information.

The parallel services may be dived into multiple tiers, as we see in figure 4. We can work with the help of multiple layers in which parallel work is the central point of the system. The parallel applications which works with such a system allow members of the group to work in a collaborative manner and the final information will be a data combination between all the groups of the system. Collaborative technologies allow members to communicate and collaborate as they cope with the opportunities and challenges of cross-boundary work. Collaborative technologies can serve to enhance the efficiency and effectiveness of organizational work processes and decision making [29]. When we think to a parallel collaborative system there are important two dimensions:

- The place where are located team members.
- The time when the team members work.

Data parallelism can be implemented into following technologies.
a. *Messaging applications*. The parallelism can be implemented in all the level of messaging: Intranet, Internet and E-mail.
b. *Conferencing Applications*. The parallelism can be implemented in all the level of these technologies including video-conferencing, audio-conferencing, web conferencing.
c. *Team Collaborative Applications* is also a branch which can parallelize applications in electronic group calendars, project management systems, etc.

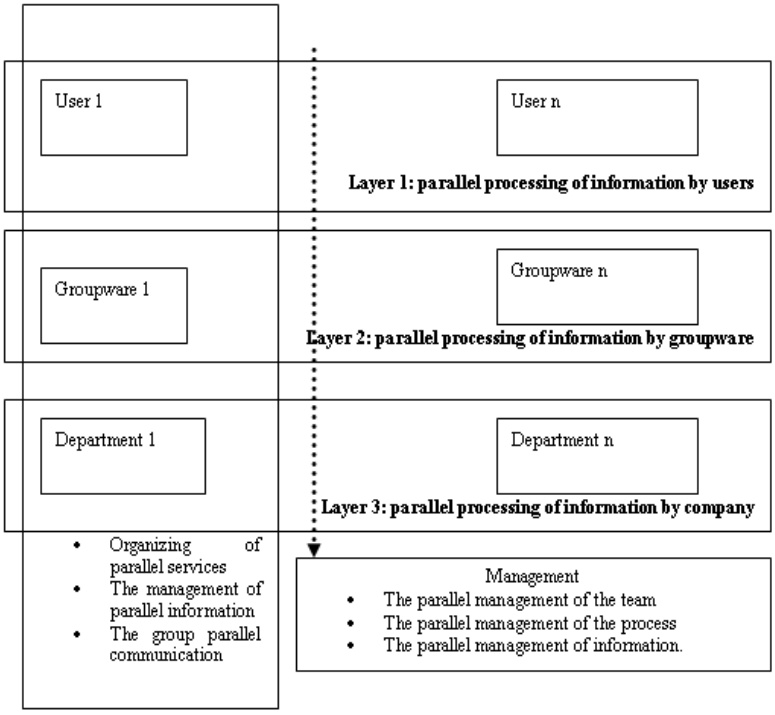

Fig. 4. Parallel services in a collaborative parallel system

In the collaborative systems, we can successful use *multidimensional data systems*. They are the most intuitive, feeble but most expensive tools for analytical processing [33].

7 Conclusions

This paper wants to be an overview in parallel programming and the link with collaborative systems. The parallelism is divided into two large categories: implicit parallelism and explicit one. Logical programming offers clear opportunities for the implicit exploitation of parallelism.

Process parallelism may be important in many applications of computer science, but because of the ubiquitous ness of arrays in such applications, data parallelism is likely to be useful in most and critical in many.

It is great for programmers but much harder to implement. To achieve the parallel collaboration between the systems and their users, in order to obtain the needed information to solve the problems of the users, it is useful to use some parallel algorithms based technologies, referring not only to individual intelligent systems but to many intelligent systems that are working together, collaborating. The parallel collaboration supports large scale design, improves the speed and at the same time improves the computing resource utilization of collaborative design. A collaborative parallel design prototype system can be developed to realize parallel assembly, parallel collaboration and parallel integration for virtual local and global organizations.

References

[1] F. E. Allen, "Program optimization", in *Annual Review in Automatic Programming 5, International Tracts in Computer Science and Technology and their Applications*, vol.13, Pergamon Press, Oxford, England, pp.239-307, 1969.

[2] R. Bahgat, *Pandora: Non-Deterministic Parallel Logic Programming*, PhD Thesis, Department of Computing, Imperial College of Science and Technology, Feb. 1991, World Scientific Publishing Co. 1993.

[3] J. Barklund, *Parallel Unification*, PhD Thesis, Uppsala University, 1990.

[4] U. Baron, J.C. de Kergommeaux et al., "The Parallel ECRC Prolog System PEPSys: An Overview and Evaluation of Results", in *Proceedings of the International Conference on Fifth GenerationComputer Systems*, Tokyo, 1988, pp. 841-850.

[5] P. Borgwardt, "Parallel Prolog using Stack Segments on Shared Memory Multiprocessors", *Proceedings of the 1984 International Symposium on Logic Programming*, Atlantic City, NJ, 1984, pp 2-11.

[6] A. Ciepielewski and S. Haridi, *A Formal Model for Or-Parallel Execution of Logic Programs, IFIP 83,* North Holland, P.C. Mason (ed.), pp.299-305, 1983.

[7] J.S. Conery, *The AND/OR Process model for parallel Interpretation of logic Programs*, PhD. Dissertation, Univ. California, Irvine, 1983.

[8] J.S. Conery , *Parallel Execution of Logic Programs,* Kluwer, Dordrecht, 1987.

[9] V. Santos Costa, D. Warren and R. Yang, "Andor-

ra-I: A parallel Prolog system that transparently exploits both and- and or-parallelism", in *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice od Parallel Programming*, ACM Press, April 1991, pp. 83-93.
[10] I. Foster and S. Tuecke, *Parallel Programming with PCN*. Argonne National Laboratory, January 1993.
[11] D. Gelernter, *Generative communication in Linda*. *ACM Transactions on Programming Language Systems*, 7(1), 1985, pp.80-112.
[12] G. Gupta, M. Hermenegildo and V. Santos Costa, "And-Or Parallel Prolog: A RecomputationBased Approach", *New Generation Computing*, 11 (3-4) pp.297-323, 1993.
[13] G. Gupta, *Multiprocessor Execution of Logic Porgrams*, Kluwer Academic Publishers, Norwell, 1994.
[14] Z. Halim, "A data-driven machine for OR-parallel evaluation of logic programs", in *New Generation Comput.* , pp.5-33, 1986.
[15] L. V. Kale, "Parallel Execution of Logic Programs: the REDUCE-OR Process Model", in *Fourth International Conference on Logic Programming*, pages 616-632. Melbourne, Australia, May 1987.
[16] P.H.J. Kelly - *Functional Programming for Loosely-coupled Multiprocessors* MIT Press, 1989.
[17] R.M. Karp, R.E.Miller and S. Winograd, "The organization of computations for uniform recurrence equations", in *Journal of the ACM*, 14(3), pp.563-590, July 1967.
[18] L. Lamport, "The parallel execution of DO loops," in *Communications of the ACM*, 17(2), 1974, pp. 83-93.
[19] G. Lindstrom, "OR-parallelism on applicative architecture," in *Proceedings of 2nd International Logic Programming Conf*, pp.159-170, July 1984.
[20] Y. Muraoka, *Parallelism exposure and exploitation in programs*, Ph.D. thesis, Tech.Rep. 71-424, University of Illinois at Urbana-Champaign, 1971.
[21] V. S. Sunderam, "PVM: a framework for parallel distributed computing", in *Concurrency: Practice & Experience*, 2(4), 1990, pp.315-339.
[22] S. Janson and S. Haridi - *Programming Paradigms of the Andorra Kernel Language*, Technical Report PEPMA Project, Sweden, November 1990.
[23] P. C. Treleaven, *Parallel Computers: Object-oriented, Functional, Logic*. J. Wiley & Sons, 1990.
[24] A. Vancea, *Paralelizarea automată a programelor*, Teză de doctorat, Universitatea "Babeş-Bolyai" Cluj-Napoca, 1999.
[25] M. (Ciaca) Vancea, A. Vancea, *An Analysis of Models for Parallel Logic Programming*, in *Research Seminars, Preprint No. 1*, 2001, pp. 21-32.
[26] M. Vancea, *Tehnici de implementare în limbaje de programare logică paralelă*, Presa Universitară Clujeană, 2004
[27] D. H. D. Warren, *An Abstract Prolog Instruction Set.* Technical Report 309, Artificial Intelligence Center, SRI International, 1983.
[28] http://en.wikipedia.org/wiki/Data_parallelism
[29] A. P. Massey, *Collaborative Technologies*
[30] P. Brezillon, F. ADAM and J.C. Pomerol, "Supporting complex decision making processes in organizations with collaborative applications - A case study", In Favela J. and Decouchant D. (Eds.) *Groupware: Design, Implementation, and Use*, LNCS 2806, Springer Verlag, 2003, pp. 261-276.
[31] P. Brezillon and P. Zarate, "Group Decision Making: A Context oriented view", in *Proceedings of tenth International Conference IFIP8/WG8.3*, Meredith R., Shanks G., Arnott D., Carlsson S. (Eds), Prato, Italie, ISBN 0 7326 2269, 1-3 Juillet, 2004, pp. 123-133.
[32] S. Smollar and R. Sprague, "Communication and Understanding for Decision Support", *Proceedings of the International Conference IFIP TC8/WG8.3*, Cork, Ireland, 2002, pp. 107-119.
[33] D.A. Sitar – Tăut, *Baze de date distribuite*, Ed. Risoprint, Cluj – Napoca, 2005
[34] T. Connoly, C. Begg and A. Strachan, *Baze de date. Proiectare. Implementare. Gestionare*. Bucuresti: Editura Teora, 2001.

Loredana MOCEAN has graduated Babes-Bolyai University of Cluj-Napoca, the Faculty of Computer Science in 1993, she holds a PhD diploma in Economics from 2003 and she had gone through didactic position of assistant and lecturer, since 2000 when she joined the staff of the Babes- Bolyai University of Cluj-Napoca, Faculty of Economics and Business Administration. She is the author of more than 10 books and over 35 journal articles in the field of Databases, Data mining, Web Services, Web Ontology, ERP Systems and much more.

Monica CIACA has graduated Babes-Bolyai University of Cluj-Napoca, the Faculty of Computer Science in 1993, she holds a PhD diploma in Mathematics from 2002 and she had gone through didactic position of assistant, lecturer and associate professor, since 1994 when she joined the staff of the Babes-Bolyai University of Cluj-Napoca, Faculty of Economics and Business Administration. She is the author of more than 10 books and over 45 journal articles in the field of Databases, Software Engineering, Artificial Intelligence and Distributed Databases.