Paralellisation of Arc-Consistency Algorithms in a Centralised Memory Machine

Marluce R. Pereira and Inês C. Dutra
Programa de Engenharia de Sistemas e Computação
COPPE/UFRJ, Brasil
(marluce,inês)@cos.ufrj.br

Maria Clicia Stelling de Castro
Departamento de Informática e Ciência da Computação
Instituto de Matemática e Estatística
Universidade do Estado do Rio de Janeiro, Brasil
clicia@ime.uerj.br

Abstract

This work presents the parallelisation of the AC-5 arc-consistency algorithm for a centralised memory machine (Enterprise). We conducted our experiments using an adapted version of the PCSOS parallel constraint solving system, over finite domains. In the implementation for a centralised memory machine (CMM) we use synchronisation based on atomic read-modify-write primitives supported in hardware. We ran four benchmarks used by the original PCSOS to debug and assess the performance of the system. Our results show that arc-consistency algorithms have very good speedups on CMM systems. We implemented different kinds of partitioning for the constraints, and different kinds of distributed labeling. We showed that performance of the benchmarks are greatly affected by these kinds of partitioning and distributed labeling. One of our applications achieves superlinear speedups due to distributed labeling. Speedups for the centralised memory machine are better than for the original version. Our results still show that we have a great potential to improve performance. Better organisation of shared data structures, analysis of the constraint graph could lead to a better distribution of labeling and indexicals, as well as algorithm restructuring could help to improve performance.

1. Introduction

Finite domain Constraint Satisfaction Problems (CSPs) usually describe NP-complete search problems. The arc-consistency algorithms can help to eliminate inconsistent values from the solution space. They can be used to reduce the size of the search space, allowing to find solutions for large CSPs.

Still, there are problems whose instance size make it impossible to find a solution with sequential algorithms. Concurrency and parallelisation can help to minimise this problem because a constraint network generated by a constraint program can be split among processes in order to speed up the arc-consistency procedure.

Parallelisation of constraint satisfaction algorithms brings two advantages: (1) programs can run faster, and (2) large instances of problems can be dealt with because of the amount of resources (memory and cpus).

Several works have been done on the parallelisation and/or distribution of constraint systems. In [13,14], authors describe logic gate level arc-consistency parallel algorithms, extending similar approaches of [24] and [7]. Gu and Sosic [12] implemented a parallel arc-consistency algorithm on a fine-grained, massively parallel hardware computer architecture. Fabiunke [9] presented a parallel framework to solve constraint satisfaction problems based on connectionist ideas of distributed information processing. Hermenegildo [16] introduced some of the problems faced by parallelising compilers for logic and constraint programs. Zhang and Mackworth developed two parallel algorithms to solve finite domain constraint satisfaction problems [25] and presented some results on parallel complexity generalising previous results achieved by Kasif and Delcher [17]. Nguyen and Deville presented a distributed arc-consistency algorithm based on the AC-4 algorithm for a distributed platform using message passing [20]. Baudot and Deville [3] proposed distributed versions of the AC-3 and AC-6 algorithms with static scheduling. Luo *et al.* [19] presented heuristics to guide the search for a solution in a CSP to improve the execution time of arc-consistency algorithms. Ferris and Mangasarian [10] presented a particular technique to parallelise execution of constraints in mathematical problems. Gregory and Yang [11] have shown that good speedups can be achieved in shared-memory platforms for solving finite domain CSPs. Andino *et al.* [1, 2] implemented a parallel version of the AC-5 algorithm [15] for a logically shared memory architecture, the Cray T3E, a high cost parallel platform.

Our work is based on Andino *et al.*'s implementation. We seek to obtain good performance on both a centralised memory machine and on-the-shelf low cost clusters of PCs. We adapted their algorithms and data structures to run first on a distributed-shared memory platform using TreadMarks, a software Distributed Shared-Memory (DSM) system [18]. We also performed some experiments on a centralised memory machine, the Enterprise 4500, in order to assess the performance of our applications.

Andino *et al.* implemented only sequential labeling and round-robin partitioning of indexicals. We added one more kind of labeling, distributed, and one more kind of partitioning, in blocks. We intended to show that different kinds of labeling and partitioning of indexicals contribute to improve performance.

Our results show that arc-consistency algorithms can achieve good speedups. One of our applications achieves superlinear speedups due to the distributed labeling. Our best results achieve linear speedups (7.97 for 8 processors) in the centralised memory version.

We expect to improve our results in four ways: (1) increasing the input data size, (2) restructuring the data structures and algorithms, (3) using adaptive software DSMs that reduce communication costs and hide latencies [6], and (4) doing a more sophisticated distribution of variables, indexicals and labeling

The paper is organised as follows. Section 2 describes the AC-5 arc-consistency algorithm and its parallelisation. Section 3 describes succinctly the porting of the PCSOS original system to our centralised memory machine version based on TreadMarks version previosly done. In Section 4 we present the methodology used in our experiments. In Section 5 we present our results, and finally, in Section 6 we draw our conclusions and future works.

2. The AC-5 Arc-Consistency Algorithm

A CSP can be modeled through constraints that can be expressed as relations involving variables. Each variable has an associated domain that can be finite or infinite. Arc-consistency algorithms solve CSPs over finite domains. There are several arc-consistency algorithms in the literature. Our work is based on the AC-5 algorithm that was firstly presented by Hentenryck [15]. AC-5 is a generalisation of other arc-consistency algorithms and it is parametrised on two procedures that are specified, but whose implementation are left open. It implements a constraint graph, where each edge and each node represents, respectively, a constraint and a variable.

Andino *et al.* [2] implemented a parallel version of the AC-5 algorithm, called PCSOS (Parallel Constraint Sthocastic Optimisation Solver), using the **indexical scheme** [5]. In this scheme, a constraint is translated into a set of **indexicals** that relate only two different variables. The execution of an indexical triggers changes in the domains of its set of related variables. These domains must be updated. The set of finite domains that keeps the current domain of each variable is called **store**.

The AC-5 algorithm presents two main steps. In the first step, all indexicals are considered once and the node-consistency is realised for each one. The store is updated and the variable related with this indexical is queued in a propagation queue. In the second step, while the propagation queue is not empty a variable is dequeued and arc-consistency is executed for all indexicals that depend on it.

The algorithm finishes when all variables are ground or when no solution is found or when we reach a fixed-point without being able to prune any more variable domain. At each arc-consistency step, if some variable is not yet ground, the algorithm enters the labeling phase to choose one non-ground variable and one of its values. During propagation, if an inconsistency is found, it is necessary to backtrack and choose another value to the non-ground variable.

3. PCSOS and PCSOS_SHM

PCSOS was implemented on a logically shared memory platform, the Cray T3E.

Our first step on porting the PCSOS to a DSM [21] platform (PCSOS_TMK version) was to study its data structures, understand them, and separate private data from shared data. We also needed to adapt the data structures to the software DSM we used, since the PCSOS system relied on the SHMEM library [23] to access logically shared data on remote nodes. We then established the right synchronisation points in the source code in order to obtain a parallel correct code.

Besides porting the original PCSOS code to a software DSM platform, we implemented two kinds of labeling: **sequential** and **distributed**, and two kinds of partitioning of indexicals: **round-robin**, and **block**.

The sequential labeling assumes that each processor can apply the labeling procedure over any variable. The distributed labeling partitions the set of variables in subsets of equal size, and each processor can execute the labeling procedure over its own subset. The round-robin partitioning of indexicals assumes that each indexical is allocated to each process at a time. The partition of indexicals in blocks assumes that a block of consecutive indexicals is allocated to each process. This partitioning is done in the beginning of the computation upon reading the input data file that contains the constraint network.

The version we used for the centralised memory machine (PCSOS_SHM) is very similar to the cluster version. Synchronisation in this platform is obtained through atomic read-modify-write primitives supported in hardware and taken from the *spinlock.h* include file used in the Linux kernel for Ultraparc processors. In the cluster implementation we needed synchronisation for reading and for multiple writes of shared data, because TreadMarks uses the Lazy Release Consistency memory model to keep memories coherent. In the centralised memory machine, we removed the synchronisation for reading shared data, because the consistency memory model is sequential in this machine.

4. Methodology and Applications

We used a centralised memory platform to make our experiments. This platform is a Sun Enterprise 4500 with 4 GBytes of memory, composed by 14 Ultrasparc processors with clock frequency of 400 Mhz, interconnected by a Gigaplane bus at 100 Mhz, that delivers up to 3.2 GBytes per second, and Solaris 2.8 operating system. Synchronisation in this platform is obtained through atomic read-modify-write primitives supported in hardware and taken from the *spinlock.h* include file used in the Linux kernel for Ultrasparc processors.

Our experiments were run on 8 processors in order to leave other processors to other users, and because our data samples are scalable up to 8 processors. We ran each experiment 10 times and presented the average execution time in seconds. The standard deviation was less than 5% in this platform.

We have used a set of four benchmarks: Arithmetic, Sudoku, N-Queens and Parametrizable Binary Constraint Satisfaction Problem (PBCSP). These applications are the same used by Andino *et al.* In their experiments.

Arithmetic. It is a synthetic benchmark. It is formed by sixteen blocks of arithmetic relations, $[B_1,...,B_{16}]$. Each block contains fifteen equations and inequations relating six variables. Blocks B_i and B_{i+1} are connected by an additional equation between a pair of variables, one from B_i and the other one from B_{i+1}. Coefficients were randomly generated. The goal is to find an integer solution vector. This kind of constraint programming is very much used for decomposition of large optimisation problems.

Sudoku. This application is a crypto-arithmetic Japanese problem. Given a grid of *25x25* squares, where 317 of them are filled with a number between 1 and 25, fill the rest of squares such that each row and column is a permutation of numbers 1 to 25. Furthermore, each of the twenty-five *5x5* squares starting in columns (rows) 1, 6, 11, 16, 21 must also be a permutation of numbers 1 to 25. The *25x25* grid is shown in Figure 1. Filled circles represent ground variables.

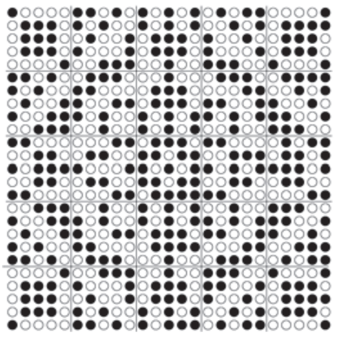

Figure 1. Sudoku: matrix representing the variables of the problem

***N*-Queens.** The *N*-Queens problem consists of placing *N* queens in an *NxN* chess board in such a way that no queen attacks each other in the same row, column and diagonal. The instance presented corresponds to *N =111*, size which leads to a significant execution time.

PBCSP. The Parametrizable Binary Constraint Satisfaction Problem is another synthetic benchmark. Instances of this problem are randomly generated given four parameters: number of variables, the size of the initial domains, density, and tightness. Density and tightness are defined as follows: $nc/nv-1$ and $1-np/ds^2$, respectively, where nv is the number of variables, nc is the number of constraints involving one variable (it is the same for all variables), np is the number of pairs that satisfies the constraint, and ds is the size of the initial domains.

In our experiments, we used two different sets of variables containing 100 and 200 variables, with tightness 75% and domain size equals to 20.

Table 1 presents a summary of the main characteristics of our applications. They spend from 100 to 10,000 times more time executing the arc-consistency algorithm than executing the labeling procedure. We consider this benchmark as a representative set of CSPs. Their constraint graphs range from weakly connected to totally connected.

Table 1. Applications Characteristics

Charact.	Applications				
	Arith	PBCSP_1	PBCSP_2	Queens	Sud
Variables	126	100	200	111	308
Constraints	254	3,713	7,463	6,105	13,942
Indexicals	1,468	7,426	14,925	12,210	27,884

5. Perfomance Analysis

In the results on the CMM, for each application we show execution times in seconds, speedups, and total of indexicals executed per each processor. The total number of acquires and barriers are similar to the implementation in the cluster, except that the implementation for the CMM does not need synchronisation for reading shared data. As explained before, the total number of failures corresponds to the number of times backtracking was called after the labeling phase.

In the next sections, we analyse the performance of each application, in decreasing order of performance.

Arithmetic

Table 2 shows the execution times for the application Arithmetic for 1, 2, 4 and 8 processors. Figure 2 shows the speedups achieved by this application. We obtained extraordinary speedups. We achieved superlinear speedups from a maximum of 83 for 8 processors in the CMM. Second, we achieved speedups from 2 to 4 (3.5) and from 4 to 8 (1.1) processors.

The speedups related to 1 processor improved from the cluster to the centralised memory version because the cluster has a very high synchronisation overhead compared with the centralised memory version [JCC]. This improvement is also due to the memory consistency model implemented by TreadMarks that requires extra synchronisation for reading shared data.

Table 2. Arithmetic. Execution times for 1,2,4, and 8 processors for the CMM

Number of Processors	Execution Times (sec.)
1	8.57
2	0.41
4	0.12
8	0.10

Observing the speedup graph, one may conclude that this application is not scalable up to 8 processors. However we still have room for improvements, because the input data size for this application can be increased in order to keep the processors busy, and the labeling can be better designed to obtain a better load balance. This issue is beyond the scope of this paper, since design of better labeling procedures is a hard problem. We have been investigating this problem in order to produce better speedups for our applications.

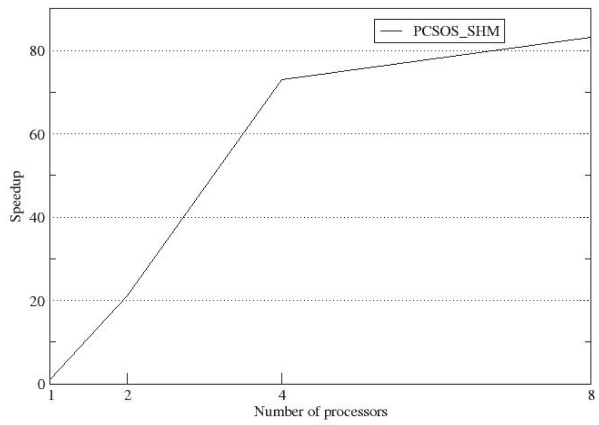

Figure 2. Arithmetc. Speedups for the CMM

Table 3 shows the number of indexicals executed per processor for 1, 2, 4 and 8 processors. Compared with the cluster version [21], we can observe that the load balance is very similar. However in the centralised memory version we had a decrease in total load from 4 to 8 processors. This decrease of load for 8 processors in the CMM is one of the factors that contributes for the improvement in performance. The other factor is related to using less synchronisation.

Table 3. Arithmetic. Total number of indexicals executed per processor for the CMM

Proc #	1	2	4	8
0	1,953,66	89,255	23,363	2,392
1	-	44,077	25,459	9,227
2	-	-	4,607	15,001
3	-	-	13,665	10,437
4	-	-	-	3,237
5	-	-	-	2,069
6	-	-	-	19,566
7	-	-	-	3,543
Total of indexicals	1,953,660	1333,332	67,094	65,472

PBCSP

Table 4 shows execution times for the application PBCSP for the two input data of 100 (PBCSP_1) and 200 (PBCSP_2) variables, for 1, 2, 4 and 8 processors.

Table 4. PBCSP. Execution times for 1, 2, 4, and 8 processors for the CMM

Number of Processors	Execution Times (sec.)	
	PBCSP_1	PBCSP_2
1	40.42	52.13
2	20.61	26.36
4	10.02	14.14
8	5.07	6.97

This application has linear speedups in this platform as shown in Figure 3.

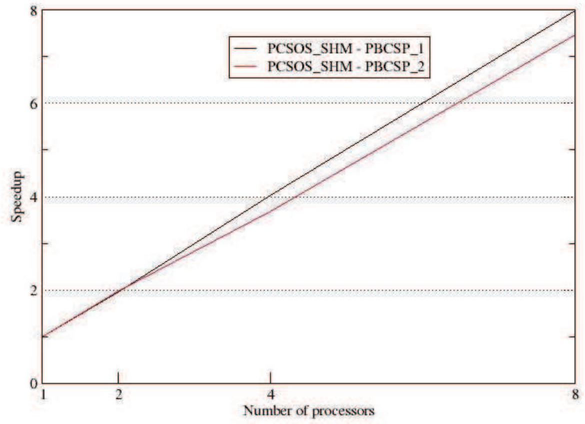

Figure 3. PBCSP. Speedups for the CMM

We can also observe from Table 5 that the load for 8 processors is much better balanced among the processors in this platform.

Table 5. PBCSP_1. Total number of indexicals

executed per processor for the CMM

Proc #	PBCSP_1	PBCSP_2	PBCSP_1	PBCSP_2
0	152,764	199,540	69,853	93,248
1	159,961	203,961	77,347	97,258
2	157,138	202,935	76,655	97,992
3	158,108	203,228	77,204	97,884
4	-	-	77,776	97,754
5	-	-	77,331	97,695
6	-	-	78,037	97,148
7	-	-	77,077	97,784
Total of indexicals	627,971	809,664	611,290	776,763

Queens

The Queens application produced better results on the CMM than on the cluster as can be seen in Table 6. As this application has a totally connected constraint graph, we did not manage to have a better performance using sequential labeling and round-robin partitioning of indexicals. As explained before, it is hard to modify the AC-5 algorithm in order to do distributed labeling or other kind of partitioning for this kind of constraint graph.

Table 6. Queens. Execution times for 1, 2, 4, and 8 processors for the CMM

Number of Processors	Execution Times (sec.)
1	10.63
2	10.45
4	8.22
8	8.62

The load balancing for this application running on the CMM is very similar to the one obtained on the cluster., as can be seen on Table 7.

Table 7. Queens. Total number of indexicals executed per processor for the CMM

Proc #	1	2	4	8
0	718,886	312,250	132,045	78,956
1	-	458,805	225,029	111,600
2	-	-	21,338	100,600
3	-	-	226,961	101,129
4	-	-	-	100,043
5	-	-	-	104,033
6	-	-	-	114,612
7	-	-	-	104,932
Total of indexicals	718,886	771,055	605,373	815,905

Sudoku

Our last application produced very bad results, mainly because of the constraint graph that has a very irregular connection pattern. The performance was very poor. Table 8 shows the execution times for 1, 2, and 4 processors.

For the same reasons presented for the Queens application, we need more sophisticated labeling and partitioning procedures in order to obtain better performance.

Table 8. Sudoku. Execution times for 1, 2, and 4 processors for the CMM

Number of Processors	Execution Times (sec.)
1	60.39
2	95.77
4	106.44

6. Discussion

Andino *et al.* managed to obtain speedups for all applications without using distributed labeling and different kinds of partitioning of indexicals. For the application Arithmetic, the maximum speedup was 3.0 for 8 processors. We managed to achieve superlinear speedups for this applications in both platforms, centralised shared memory and distrubuted shared memory because of our distributed labeling. We also managed to obtain better maximum speedup (3.5 from 2 to 4 processors) using the CMM version.

PBCSP_1 in the Andino *et al.* version presented maximum speedup of 18.0 for 34 processors. In our implementation we obtained almost linear speedups up to 8 processors for the CMM version, which lead us to conclude that performance will scale when we increase the number of processors.

Queens presented maximum speedup of 4.0 for 16 processors, and Sudoku presented speedup of 2.3 for 18 processors. We have not managed to achieve these speedups, but believe that by doing a more sophisticated analysis of the constraint graph for these applications, we could obtain better performance. As mentioned before, distributing the labeling and doing a better partitioning of indexicals is a hard problem that we have been investigating in order to obtain better load balancing and better performance for constraint applications.

Andino *et al.* experiments were done in a high performance architecture with a high bandwidth and low latency processor interconnection network, 3D-torus. The Cray T3E interprocessor network has 480 MBytes per second of nominal capacity (230 MBytes per second in practice). The CMM has a nominal capacity of 3.2 Gbytes per second, but it uses a high latency bus that suffers with contention on multiple memory accesses. Our interprocessor network achieves only 100 Mbits per second. The Cray T3E also provides efficient user library facilities for shared memory programming through the SHMEM library. This library provides fast atomic reads and writes to remote memories. This counts as an advantage to Andino *et al.*'s experiments, but we have shown that we can obtain better performance changing the labeling and partitioning algorithms. We believe that different kinds of labeling and partitioning would produce better results in their architecture. They only used round-robin partitioning of indexicals and sequential labeling. Given our hardware platforms restrictions and kind of synchronisation system we employed, we consider our results very positive.

We achieved superlinear speedups for the application Arithmetic, and achieved best speedup for the application PBCSP_1 of 7.97, with 8 processors, in the CMM, using sequential labeling and partitioning of indexicals round-robin, which practically corresponds to 100% of efficiency. As mentioned before, we still have several opportunities for improvement.

On the CMM, we achieved fairly good results for all applications, except for Sudoku. We can still improve on these results by implementing a more sophisticated labeling procedure and indexical distribution, and fine tuning data structures and the algorithms.

Regarding the cache coherence protocol, previous experiments on hardware DSM system have shown that hybrid protocols, where update-based protocols are switched to invalidate-based protocols dynamically, can yield better performance than pure invalidate-based protocols [4, 8]. Other experiments on simulation of adaptive software DSMs also suggests that other kind of platform can improve our results[6].

7. Conclusions and Future Works

This work presented the parallelisation of the arc-consistency algorithm AC-5. We conducted our experiments in a centralised memory architecture, an Enterprise 4500 with 14 processors. We ran four benchmarks already used in the literature to assess our experiments up to 8 processors. We implemented two kinds of partitioning of indexicals: round-robin and block, and two kinds of labeling: sequential and distributed. For one application we achieved superlinear speedups, because of our distributed labeling. Our best speedup (7.97) was achieved for the PBCSP_1 application, on the CMM, using sequential labeling and partitioning of indexicals round-robin, for 8 processors. In fact, our best results are achieved for the centralised memory architecture. The reasons for the low performance of some experiments are directly related to the size of the input data, and load imbalance caused by the kinds of labeling and distribution of indexicals performed in this work. These factors impedes better performance in the CMM version.

We still have many opportunities for improvements. Better organisation of shared data structures as well as algorithm restructuring could help to improve performance. Also, better analysis of the constraint graph could lead to a better distribution of labeling and indexicals.

Work is under way to accomplish these tasks and produce higher speedups for these and other constraint satisfaction applications, including experiments on software DSM systems with faster interprocessor networks such as Giganet, Gigabit and Myrinet.

Acknowledgments

The authors would like to thank Dr. Ruiz-Andino for his help with the CSOS system used in this paper. He kindly made available his C source code and benchmarks. We would also like to thank Dr. Amorim who granted us the utilisation of his cluster of PCs to realise our experiments. Dr. Pontelli granted us the utilisation of the centralised memory machine available in New Mexico State University to perform or experiments. We are very grateful for his kindness. This work benefitted from useful comments from Dr. Santos Costa. Inês Dutra would like to thank the CNPq CLoP^n project. This work is partially supported by the Brazilian Research Council CNPq, Capes and by FAPERJ.

References

[1] A.R. Andino, L. Araujo and J. Ruz, “Parallel Solver for Finite Domain Constraints”, *Technical Report SIP 71.98*, Department of Computer Science, Universidad Complutense de Madrid.

[2] A. R. Andino and L. Araujo and F. Sáenz and J. Ruz, “Parallel Execution Models for Constraint Programming Over Finite Domains”, In *Principles and Practice of Declarative Programming*, pp. 134-151, 1999.

[3] B. Baudot and Y. Deville, “Analysis of Distributed Arc-Consistency Algorithms”, *Technical Report RR 97-07*, University of Louvain, Belgium , 1997.

[4] V. M. Calegario and I. C. Dutra,”Performance Evaluation of Or-Parallel Logic Programming Systems on Distributed Shared Memory Architectures”, In *Proceedings of the EUROPAR'99*, pp. 1484-1491, August/September 1999.

[5] B. Carlson, “Compiling and Executing Finite Domain Constraints”, PhD thesis, University of Uppsala, Department of Computer Science, 1995.

[6] M.C.S.Castro and C.L. Amorim, “Efficient Categorization of Memory sharing Patterns in Software DSM Systems”, In *Proceedings of the 15th International Parrallel and Distributed Processing Symposium*, pp. 63, April 2001,CD-ROM.

[7] Z. Collin and R. Dechter and S. Kartz, “On the Feasibility of Distributed Constraint Satisfaction”, In *Proceedings of the 12th Proceedings of the International Joint Conference on Artificial Intelligence*, pp.318-324, 1991.

[8] I.C. Dutra, V. Santos Costa and R. Bianchini, “ The Impact of Cache Coherence Protocols on Parallel Logic

Programming Systems", In *International Conference on Computational Logic*, Lecture Notes in Artificial Intelligence, V. 1861, pp. 1285-1299, 2000.

[9] M. Fabiunke, "Parallel Distributed Constraint Satisfaction", In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 99)*, Las Vegas", pp. 1585-1591, June 1999.

[10] M.C. Ferris and O.L. Mangasarian."Parallel Constraint Dsitribution", *SIAM Journal on Optimization*, V. 4, pp. 487-500, 1991.

[11] S. Gregory and R. Yang, "Parallel Constraint Solving in Andorra-I", In *Proceeding of the Fifth Generation Computer Systems*, pp 843—850, June 1992.

[12] J. Gu and R. Sosic,"A Parallel Archicteture for Constraint Satisfaction", In *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 229-237, June 1991.

[13] H. W. Guesguen, "Connectionist Networks for Constraint Satisfaction", In Proceedings of the National Conference on Artificial Intelligence, Spring Symposium on Constraint-based Reasoning, pp. 182-190, 1991.

[1] H. W. Guesguen and J. Hertzberg, "A Perspective of Constraint-Based Reasoning", In *Lecture Notes on Artificial Intelligence*, Springer-Verlag, 1992.

[15] P. V. Hentenryck and Y. Deville and C. M. Teng, "A Generic Arc-Consistency Algorithm and its Specifications", *Artificial Intelligence*, V. 57, N..2-3, pp. 291-321, October 1992.

[16] M. Hermenegildo, "Parallelizing Irregular and Pointer-Based Computations Automatically: Perpectives from Logic and Constraint Programming" In *Parallel Computing*, V. 26, n. 13-14, 2000.

[17] S. Kasif, "On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks", In *Artificial Inteligence*, V. 45, pp. 275-328, 1990.

[18] P. Keleher and A. L. Cox and S. Dwarkadas and W. Zwaenepoel, "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems", In *Proceedings of the 1994 Winter Usenix*, pp.115-131, 1994.

[19] Q. Y. Luo and P. G. Hendry and J. T. Buchanan, "Heuristic Search for Distributed Constraint Satisfaction Problems", *Research Report KEG-6*-93, Department of Computer Science, University of Strathclyde, 1993.

[20] T. Nguyen and Y. Deville, "A Distributed Arc-Consistency Algorithm", In *Science of Computer Programming*, pp. 227-250, 1998.

[21] M. R. Pereira, I. C. Dutra, M. C. S. Castro, "Parallelisation of Arc-Consistency Algorithms", In *VI em Worshop Distribuidos y Paralelism*, Jornadas Chilenas de Computación 2002, Copiapó, Chile, CD-ROM.

[22] V. A. Saraswat, "Concurrent Constraint Programming Languages", In *MIT Press*, 1993.

[23] S. L. Scott, "Synchronization and Communication in the T3E Multiprocessor", In *ASPLOS7*, pp.26-36, October 1996.

[24] M. J. Swain and P. R. Cooper, "Parallel Hardware for Constraint Satisfaction", In Proceedings of the National Conference on Artificial Intelligence, pp. 682-686, 1988.

[25] Y. Zhang and A. K. Mackworth, "Parallel and Distributed Algorithms for Constraint Networks", *Technical Report 91.6*, Department of Computer Science, The University of British Columbia, Canadá, 1991.