

Domain Decomposition for Parallel Resolution of Constraint Satisfaction Problems with OpenMP

Zineb HABBAS

Université de Metz
Ile du Saulcy

F-57045 Metz Cedex

zineb@iut.univ-metz.fr

Michaël KRAJECKI

Université de Reims Champagne-Ardenne
BP 1039

F-51687 Reims Cedex2

michael.krajecki@univ-reims.fr

Daniel SINGER

Université de Metz
Ile du Saulcy

F-57045 Metz Cedex

singer@lita.univ-metz.fr

ABSTRACT

Many problems in computer science, especially in Artificial Intelligence, can be represented as constraint satisfaction problems (CSP). For example, scene labeling in computer vision involves testing possible interpretation of objects against relation rules. Other constraint satisfaction problems include theorem proving, scheduling, expert systems. These problems are typically NP-Complete because they require extensive searches to find a solution and the basic search algorithm is the naive Backtracking strategy. In order to improve its performances different approaches have been explored: filtering strategies, heuristics for search algorithms, decomposition methods. Although parallelization seems to be a good candidate to obtain further practical improvements the research in this direction is fewly developed. In this paper we explore the benefit of a domain decomposition strategy for parallel CSP resolution. Mainly we solve in parallel the different subproblems resulting from the decomposition step on a shared memory architecture with an OpenMP library. All the experiments were realized with the Silicon Graphics Origin2000 parallel machine.

KEYWORDS: Constraint Satisfaction Problems (CSP), Parallel Processing, OpenMP, Domain Decomposition, Irregular Applications.

1 Introduction

Many problems in computer science, especially in Artificial Intelligence, can be represented as constraint satisfaction problems or CSP. For example, scene labeling in computer vision involves testing possible interpretation of objects against relation rules. Other constraint satisfaction problems include theorem proving, scheduling, expert systems. Typically these problems are NP-Complete and they require extensive search to find a solution: the basic search algorithm is the naive Backtracking strategy. Formally, a CSP is defined by a set of variables and by a set of constraints. A set of allowed values is associated at each variable. Each constraint is given by a compatibility relation over the domains of variables. Solving a CSP means finding an assignment for each variable that satisfies all the constraints. Most of the proposed algorithms are improvements of the simple enumerative search: *Backtracking* (BT) algorithm. Its main drawback is the computational cost and many different approaches have been proposed in order to improve its performance. We can roughly classify them in the three following categories:

- performing constraint propagation as a preprocessing step before or during search with different filtering techniques based on an Arc-Consistency property [15].
- improving the search algorithm itself by choosing a good variable and value orderings for the next variable to be instantiated and value to be assigned [1, 8].
- performing subproblems decomposition as a preprocessing step [2, 4, 7, 9].

Since CSP is NP-Complete, an efficient general search algorithm is unlikely to exist. Although parallelization seems to be a good candidate to obtain further practical improvements, the research in this direction is unfortunately fewly developed. Surprisingly, unlike the Computational Science and Engineering area [14], the decomposition methods which generate "inherent parallel" algorithms have not been explored for this purpose. In [16], we have already studied the parallelization of the search algorithm itself. Mainly we proposed a generic parallel scheme combining both sequential search algorithms and MIMD dynamic load balancing

policies. The Dynamic Load balancing strategies are essential in order to maintain all processors busy. This approach presents a serious drawback due to the overhead induced by the number of communications. Some decomposition methods of CSP developed in the literature permit to split a given CSP into a set of smaller problems which can be independently solved in parallel.

In this work, we explore this last idea by studying more precisely the Domain Decomposition method of CSP [2]. First, this strategy is used as a preprocessing step for generating a collection of subproblems, each one potentially containing the solution of the initial problem if it exists. Second, we solve in parallel all these subproblems using a classical search strategy. We investigate the programming feasibility of this approach with the OpenMP directive based parallel computation environment citeOpen97.

2 Sequential resolution of CSP

First, we recall some basic definitions of the Constraint Satisfaction Problems framework and the general method to solve a CSP.

2.1 Preliminaries

Definition 2.1 ([17]) A **Constraint Satisfaction Problem** is a tuple $Pb = (X, D, C, R)$, where :

- $X = \{X_1, \dots, X_n\}$ is a set of n variables.
- $D = \{D_1, \dots, D_n\}$ is a set of n domains. Each D_j is associated with X_j .
- $C = \{C_1, \dots, C_m\}$ is a set of m constraints. Each constraint C_i is defined by a set of variables $\{X_{i_1}, \dots, X_{i_{n_i}}\} \subseteq X$.
- $R = \{R_1, \dots, R_m\}$ is a set of m relations. Each relation R_i defines a set of n_i -tuples on $D_{i_1} \times \dots \times D_{i_{n_i}}$ compatible with respect to C_i .

A *binary CSP* is a CSP where all the constraints are sets of two variables at most. An *instanciation* of a set of variables A is a k -tuple (a_1, \dots, a_k) , representing an assignment of $a_j \in D_j$ to X_j , for all $X_j \in A$. A *consistent instanciation* of A is a set of assignments that satisfies all the constraints C_k such that $C_k \subseteq A$. A consistent instanciation on X is a *solution* of the CSP.

In this paper, we consider that solving a CSP is *finding one solution*. We also restrict the study to Binary CSP on Finite Domains where the relations are given in extension as couples sets.

The main method to find a solution for a particular CSP is the **Backtrack** algorithm. Unfortunately, it presents an important drawback: it is exponential in the number of variables n .

2.2 Improvements of Backtrack

To improve the performances of the search algorithms, some filtering techniques have been proposed to restrict the size of search space by early detecting inconsistency. Based on some local partial consistency definition, the filtering algorithms either remove values of domains that cannot be extended to a global solution or modify the set of constraints (see Debruyne and Bessi ere for a recent overview [3]). This filtering effort may be done as a preprocessing procedure or throughout search. Practically the overhead caused by the filtering has to be outweighed by its gain and this is the key for finding the best level of consistency to achieve. Arc Consistency filtering is widely used on binary CSP because it does not change the set of constraints and it is obtained by a Constraint propagation process. AC is a basic level of local partial consistency and it is also the most used one since the initial work of Waltz in 1972 until now.

Different works have been done to improve the Backtrack algorithm itself. Among them P. Prosser proposed an interesting approach which permits the combination of two characteristic behaviours of search algorithms namely the backward and the forward moves [19]. The main idea is to make explicite the notion of backward and forward moves in the tree search algorithms instead of keeping them hidden by the classical recursivity. Explicit formulation of the basic operations simplifies the combining of algorithms and allows the study of a family of new hybrid ones.

This last decade many algorithms and heuristics have been developed along different axes so that it was necessary to compare their performances. The general comparative study is still in progress.

The Forward-Checking algorithm (**FC**) is called a "*looking-forward scheme*" since, when the search process makes a trial instantiation of a variable, it looks ahead toward the future variables (not yet instantiated) and removes from their domain all values that are incompatible with the trial instantiation. Therefore, when FC goes forward and considers a new variable X_j , each value in D_j is consistent with the past variables (yet instantiated). When the domain of a forward variable becomes empty while checking consistency a dead-end occurs and FC backtracks chronologically. The goal of FC is to *fail early* by detecting inconsistencies within the tree search as soon as possible cutting the exploration of fruitless branches.

Dynamic Variables Ordering Heuristics have been proposed to enhance the search and the Minimum Remaining Value (**MRV**) is designate as one of the best. Moreover, a number of works on randomly generated problems have shown how the CSP hardness varies with simple parameters describing their structure. Some identified parameters values distinguish regions with high probability of hard problem instances from ones

with high probability of easier ones. This is called the phase transition phenomenon [20, 21].

We present in the sequel the generic resolution algorithm of a CSP according to Prosser [19].

Algorithm 1. *Generic resolution of a CSP: main function.*

```

function CSP_Resolve(Pb)
  Consistent := True
  State := Unknown
  i := 1
  while State = Unknown do
    if Consistent then
      i := Label(i, Consistent)
    else
      i := Unlabel(i, Consistent)
    endif
    if (i > n) then
      State := Solution
    else if (i = 0) then
      State := impossible
    endif
  endif
  return State
endwhile.

```

2.3 Decomposition methods

Most of the proposed techniques for decomposing a CSP exploit the fact that CSP tractability is intimately connected to the topological structure of their underlying constraints graph [5, 9]. Mainly two strategies can be cited: the "**cycle-cutset**" due to Freuder [6] and the "**tree-clustering**" due to Dechter and Pearl [5].

The "**cycle-cutset**" method cuts each cycle of the given constraints graph by instantiating one variable per cycle. The resulting CSP is tree-structured and this leads to a sequential polynomial-time resolution [6] or parallel logarithmic-time [13]. The main drawback of this method is to remain exponential in the size of the cycle-cutset and finding a minimal cycle-cutset is also NP-hard. Moreover, as far as we know, an efficient method to find a small cycle-cutset does not exist [11].

The "**tree-clustering**" method is an application to CSP of the acyclic databases properties given by the Freuder's condition for backtrack free search [6]. As for the previous method, the desirable structure is obtained by transforming any constraints graph into a tree. More precisely, this strategy considers the primal constraints graph first and forms clusters of variables by using an efficient triangulation algorithm of Tarjan [22]. It results a dual constraints graph whose clusters form the set of nodes. The dual constraints graph is then transformed on a so called join-graph by pruning some redundant arcs and the subproblems defined by the clusters are separately solved. Finally the tree-structured problem is solved by considering the clusters as singleton variables.

These two approaches are structurally oriented and we refer to section 3.2 for the "**Domains Decomposition**" method we use for Parallel Resolution.

3 Parallel Resolution of CSP

3.1 Related works

There are mainly two different approaches to use parallel computation for constraints solving (see figure 1).

1. Distributing the search tree among the processors.
2. Splitting the initial CSP into a collection of easier subproblems to be solved in parallel.

Most of the significant developed works are related to the first one and more precisely in the MIMD parallelism model.

The crucial point leading to efficiency of the parallel tree search algorithm is the load balancing strategy. The aim of these techniques is to keep the set of processors busy without an undue overhead. The first question is to choose between static or dynamic load balancing. In the static case an initial partitioning of the root nodes in the search tree is performed before the search is initiated. The problem is then divided into a number of subproblems pools : one pool for each processor. Each processor now performs a sequential search on the pool assigned to it. Communication is limited to initial distribution of the subproblems pools and termination tests. The main drawback of this approach is that the time needed to process subproblems pools can totally differ. Then some processors may become idle long before termination, resulting in poor processors utilization: this phenomenon is known as starvation. In the second approach the subproblems are dynamically distributed. The main features of dynamic load balancing strategies in the context of parallel backtrack are classified as follows:

what is the initial work distribution?

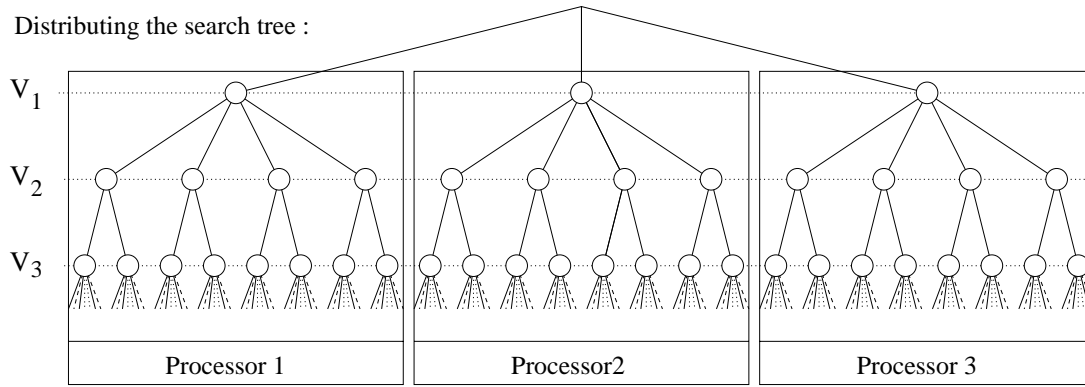
who initiates the load balancing request?

what is the partner selection policy? The partner selection policy determines from or to which remote node, tasks should be received or sent.

what is the transfer policy? The transfer policy determines which tasks should be sent.

What is named overloaded and underloaded processor?

How determining current state and the termination of the system?



Domain Decomposition :

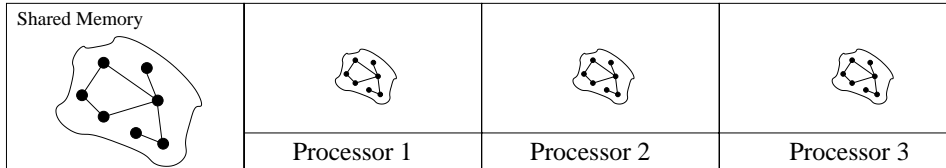


Figure 1: Two parallel CSP resolution approaches.

Based on the message passing parallel model and the Prosser's approach we have proposed in [10] a generic parallel search algorithm. A general paradigm is introduced to parallelize different search algorithms including both the generic procedures relative to the sequential search and the load balancing strategies relative to the parallelization. One particular parallel search algorithm called FC-CBJ that performs forward domain filterings has been studied since it represents a class of efficient sequential search algorithms. In parallel, reasonable efficiencies were observed and even superlinear efficiency were obtained for satisfiable problems. Our first objective was to study the benefits of parallelism in the forward search algorithms.

Remark: Along literature we can notice that works on parallel approach to solve CSP are not significant. Most amazingly, no work was done on the use of decomposition methods for parallel resolution. \diamond

3.2 Domains Decomposition for parallel resolution

The classical CSP decomposition methods explore structural properties of the constraints graphs. The "**Domains Decomposition**" method at the opposite explores the relations between the values inside domains. Here we use the strategy based on the "**Micro-Structure**" of the CSP introduced by P. Jegou and studied in [2] as a Domains Decomposition method. The Micro-Structure of a CSP is the graph defined by the compatible rela-

tions between the variable-value pairs. Vertices are exactly these pairs and the edges relate two compatible pairs. Given a CSP Pb this decomposition produces a collection of δ subproblems $[Pb_1, \dots, Pb_\delta]$ equivalent to the initial problem Pb . The time complexity for solving each subproblem Pb_i is necessarily less than the time complexity of the initial problem Pb since the domains of each subproblem are reduced by the decomposition. Both the tree-clustering and the Domains Decomposition methods are based on the efficient Tarjan-Yannakakis triangulation algorithm [22]. It transforms any graph into a chordal graph by adding edges and by computing a simplicial ordering of the nodes. The most fundamental difference between tree clustering and domain decomposition methods is that for the last one the triangulation algorithm works on the Micro-Structure instead of the initial constraints graph. The resulting maximal cliques derived from the chordal Micro-Structure correspond to independent subproblems when they covered all the variables set of the initial problem. The Domains Decomposition method can be summed up in the following steps:

1. Given a CSP $Pb = (X, D, C, R)$ such that (X, C) is a complete graph, use the function $Build_Micro_Structure(\mu(Pb))$ to build the associated micro-structure.
2. Process the triangulation algorithm to generate a simplicial order σ and a triangulated micro-structure $\Gamma(\mu(Pb))$ (using the function $Triangulate(\mu(Pb), \Gamma(\mu(Pb)), \sigma)$).

Remark: The Domains Decomposition method is based on a basic property of the micro-structure: given a CSP Pb and its associated micro-structure $\mu(Pb)$ then (a_1, \dots, a_n) is a solution of Pb iff $((x_1, a_1), \dots, (x_n, a_n))$ is a n-clique of $\mu(Pb)$. \diamond

3. From the triangulated micro-structure $\Gamma(\mu(Pb))$ and the simplicial order σ build the derived maximal cliques.
4. From the maximal cliques set generate the set of subproblems: each one corresponding to clique which covers the variables set.

Remark: The triangulation step builds both the order σ and the triangulated micro-structure $\Gamma(\mu(Pb))$. Splitting this into separate steps gives a trivial parallelization of the domains decomposition strategy. \diamond

We give in the sequel the basic domain decomposition procedure and the parallel resolution of CSP based on this decomposition strategy.

Algorithm 2. *Domain decomposition strategy: main function.*

```
function Domain_Decomposition(Pb, [ Pb1, ... Pbδ])
  Build_Micro_Structure( $\mu(Pb)$ )
```

```
Triangulate( $\mu(Pb)$ ,  $\Gamma(\mu(Pb))$ ,  $\sigma$ )
Generate_MaxCliques( $\Gamma(\mu(Pb))$ ,  $\sigma$ , {C1, ..., Cγ})
Generate_Sub_CSP({C1, ..., Cγ}, Pb, [ Pb1, ... Pbδ]).
```

Algorithm 3. *Parallel Generic resolution of a CSP: main function.*

```
function Parallel_Resolve(Pb)
  Domain_Decomposition(Pb, [ Pb1, ... Pbδ])
  for || each Pbi do
    CSP_Resolve(Pbi)
  endfor.
```

4 Implementation and first results

The reader may notice that this work constitutes the first element of a more ambitious project we outline in figure 2. The sequential search algorithms are developed in an Object Oriented style with C++ (the MAC algorithm for Maintaining Arc Consistency is in progress). This work is only concerned with the significance of the Domains Decomposition method for Parallel Resolution with a shared memory. We only present in this section our first results on both sequential and parallel experiments. All the experiments are fulfilled by using FC with MRV (FC-MRV) as a basic search algorithm.

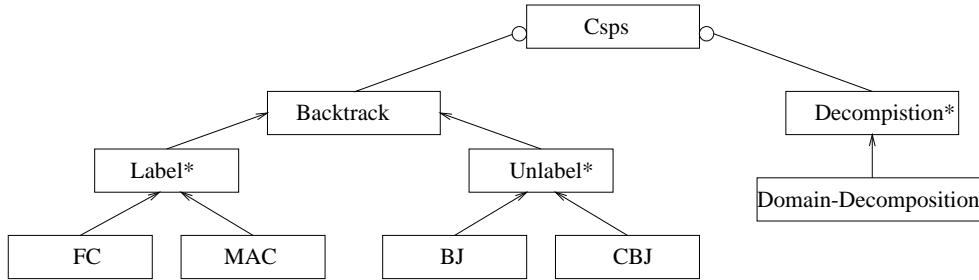


Figure 2: Project overview.

4.1 Dynamic data structures and openMP

To evaluate the performance of the parallel domain decomposition approach we use OpenMP for shared memory parallelism [18]. OpenMP derives from the ANSI X3115 efforts. It is a set of compiler directives and runtime library routines that extend a sequential programming language to express parallelism with a shared memory. OpenMP conforms to SPMD programming language style. Passing from sequential resolution to a parallel one after the domain decomposition step is straightforward within the OpenMP paradigm. We only have to parallelize the for loop solving all the subproblems using the OpenMP compiler directives. The domains decomposition method can be seen as a static load

sharing strategy.

4.1.1 The binary CSP memory representation

We need two data structures to represent a particular CSP: the first one is an adjacency matrix ($Adj[][]$) which enables to check whether a relation between two variables of the problem exists or not.

The second data structure we need is a list of couples when a constraint between v_i and v_j exists. $Adj[v_i, v_j]$ is a reference to the list of allowed couples for the relation R_{ij} .

We can now evaluate the space complexity of these data structures. For the adjacency matrix, the space

complexity is $O(n^2)$ where n is the variables number. In the worst case, the CSP contains $\frac{n \times (n-1)}{2}$ relations and the length of each list is $O(d^2)$ where d is the uniform domain size.

4.1.2 The memory cost of the tree search management

Each search algorithm needs some data structures to efficiently investigate the search tree. For example, FC maintains n tables of size d called *currentdomain* to compute the value allowed for each variable while the search is performed. This algorithm also uses n stacks used to perform the filtering and to restore the tree when inconsistency is discovered.

In C and C++ it is possible to dynamically allocate the memory for each table. Moreover the memory used by the application in this case is exactly equal to the required space to solve the particular CSP. This is the reason why we have decided to use dynamic memory allocation in our first experiments. But doing this we obtain a very poor efficiency of our parallel algorithm and it was especially true for 8 or 16 processors.

In fact the following problem appears. When an OpenMP program allocates dynamically the memory (using `new` or `malloc`), the new object is shared and there is no way to make it private. In our application, we need private structures for the FC algorithm in order to be as efficient as possible. At the end we choose to allocate statically at the compilation time the table *currentdomain* and the needed stacks.

4.1.3 Detection of the termination in parallel

When a processor finds a solution for a subproblem, we shall stop all the other processors even if all the subproblems have not been solved. To achieve this goal, we have defined a shared integer variable, called *pend* initially equals to 0. When a processor proves the satisfiability of the problem it writes a positive value for this variable. Each processor solving a problem takes periodically a look at *pend* and if the value is positive then it gives up search in progress.

The reader may notice that thanks to OpenMP we do not need to take care of this variable update. In the same way, the subproblems distribution is carried out dynamically by OpenMP. In our study these are the two key advantages of OpenMP.

4.2 Randomly generated problems

The uniform model of binary randomly generated CSP [20] is characterized by the 4-tuple $\langle n, d, m, p \rangle$

where n is the number of variables, d is the unique domain size, m is the number of constraints and p is the number of incompatible pairs of values in a relation. Our analysis is based on the problems class proposed by Frost and Dechter in [8] which provide a large spectrum permitting us to make an opinion on "hard random problems". The problems we consider are taken in the neighbouring of the "Transition Phase" region within the family of sparse graphs (the density of these graphs is 0.1) of the form $\langle 50, d, m, p \rangle$ where d varies from 25 to 40. Because we only consider sparse graphs the parameter value m is fixed to 123.

First, we studied the Domains Decomposition strategy in the sequential case. That is why we compared the results obtained by FC-MRV on the initial problem with the same search algorithm on all the subproblems obtained by decomposition. Our experiments confirm the first results obtained by P. Jegou in [12] :

- the behaviour of the decomposition method is not the same for consistent and non consistent problems.
- the number of subproblems obtained from the decomposition method is proportional to the size of the domains.

In the following, we present the performance results obtained for inconsistent problems and consistent ones both in the sequential and in the parallel cases.

4.3 Analysis on hard CSP: Decomposition and inconsistent problems

In order to study the performance of the Domains Decomposition method on inconsistent problems, we propose to compare the CPU time in seconds (t_{Seq}) consuming by FC-MRV on the initial problem with the cumulated CPU time of the FC-MRV execution on each subproblem obtained after decomposition (t_{Tr}). If the problem is inconsistent, all the subproblems have to be completely solved before to decide the final negative result. Table 1 shows that for inconsistent problems the decomposition method alone offers good efficiencies in the sequential case. In fact, the efficiency considered corresponds to the benefit of the domains decomposition for sequential resolution: $Eff = \frac{t_{Seq}}{t_{Tr}}$. Even if the CPU time decreases the obtained parallel efficiencies are not significant when the processors number grows. The parallel efficiencies noted Eff_i where i is the number of processors are computed as follows: $Eff_i = \frac{t_{Seq}}{(t_{par_i} \times i)}$. In this case, a set of very small and easy subproblems is obtained from the initial problem by the Decomposition method. Consequently, it is not quite advantageous to solve them on a parallel machine.

Problems	Results									
			1 proc		4 proc		8 proc		16 proc	
	NB_Pb	T_seq	t_Tr	Eff_Tr	tpar_4	Eff_4	tpar_8	Eff_8	tpar_16	Eff_16
50 25	11	297.13	178.064	1.66	68.98	0.65	41.69	0.53	33.79	0.33
50 30	20	207.12	160.1	1.29	54.7	0.76	38.52	0.51	32.6	0.31
50 35	19	230.4	310.04	0.75	98.2	0.79	61.7	0.62	42.33	0.45
50 40	20	270.98	200.088	1.35	66.5	0.75	43.5	0.57	32.9	0.37

Table 1: Results of tests: inconsistent problems

4.4 Analysis on hard CSP: Decomposition and consistent problems

On the contrary, for consistent problems, the decomposition method alone is not always efficient for sequential resolution. But in this case, as expected, the benefit

offered by the parallel resolution is more significant and we mostly observe superlinear efficiencies. This last observation may be justified by the fact that as soon as a processor gets a solution the whole parallel execution is stopped.

Problems	Results									
			1 proc		4 proc		8 proc		16 proc	
	NB_Pb	T_seq	t_Tr	Eff_Tr	tpar_4	Eff_4	tpar_8	Eff_8	tpar_16	Eff_16
50 25	18	235.16	233.93	1.08	54.59	1.06	21.4	1.36	7.22	2.08
50 30	6	632	976.3	0.70	228	1.05	98.4	1.22	24.3	2.48
50 35	12	1125.28	169.5	6.63	34	1.26	18.16	1.17	12.25	0.88
50 40	6	1138.16	1012.83	1.12	250.17	1.02	50	2.5	44.2	1.43

Table 2: Results of tests: consistent problems

5 Conclusion and some perspectives

In this paper we have mainly studied the Domains Decomposition method of CSP.

From the sequential point of view we have confirmed the results of P. Jegou: the domains decomposition of CSP is efficient before parallelization for inconsistent problems. In fact, we remark that the decomposition method creates in the most cases a collection of easy subproblems. Then in this case the cumulated time for solving each of these subproblems is not penalizing. Concerning the consistent problems case, the domains decomposition turns out to be less efficient since the cumulated time for solving all the subproblems deriving from decomposition is comparable (and sometimes is greater than) the sequential time in the worst case: the problem containing the solution is often solved in the last. Moreover, it seems difficult to improve these results since it is impossible to order them in advance.

From the parallel point of view we have mainly investigated the performance of this approach based on a shared memory model. The parallelization is very naive. It consists in the distribution of the independent subproblems obtained by the decomposition method on a

collection of processors. We have measured the parallelism benefits by considering the efficiencies for 1, 4, 8 and 16 processors. We observe in the case of consistent problems superlinear efficiencies and appreciable efficiencies for inconsistent problems. We conclude that the parallel solving of CSP after decomposition is a promising approach.

These general remarks give some indications for possible perspectives. First we have to analyze our results more deeply, that is to evaluate the impact of our strategy on a larger spectrum of hard problems. We plan to estimate the benefit of the parallelization for non-sparse graphs and on real applications.

In this paper we have considered without loss of generality FC-MRV as a basic search algorithm. Another interesting perspective consists in leading this approach with the MAC search algorithm and the development of an effective MAC is actually in progress.

As shown in this work, the Domains Decomposition method can be used as an initial load sharing step. This method can also be combined with a parallel distribution of the search tree. This perspective will give us the opportunity to compare this solution to our previous work on the parallel evaluation of the CSP search tree [10].

Acknowledgements

This work was partly supported by the Centre Lorrain de Calcul Hautes Performances: Centre Charles Hermite (CCH).

The authors want to thank P. Jégou from the University of Aix-Marseille for discussions on the Domain Decomposition method.

References

- [1] BACCHUS, F., AND VAN RUN, P. Dynamic variable ordering in CSPs. In *Proceedings of the first International Conference on Principles and Practice of Constraint Programming* (Cassis, 1995), pp. 258–274.
- [2] CHMEISS, A. *Réseaux de contraintes : algorithmes de propagation et de décomposition*. PhD thesis, Université des Sciences et Techniques du Languedoc, Aix-Marseille I, 1996.
- [3] DEBRUYNE, R., AND BESSIÈRE, C. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97* (Nagoya, Japan, 1997), pp. 412–417.
- [4] DECHTER, R. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence* 41 (1990), 273–312.
- [5] DECHTER, R., AND PEARL, J. Tree-clustering schemes for constraint-processing. In *Proceedings of the sixth National Conference on Artificial Intelligence (AAAI-88)* (Saint Paul, MN, 1988), pp. 150–154.
- [6] FREUDER, E. C. A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery* 29 (1982), 24–32.
- [7] FREUDER, E. C., AND HUBBE, P. D. Extracting constraint satisfaction subproblems. In *Proceedings of the fourteenth International Joint Conference on Artificial Intelligence* (Montreal, 1995), pp. 548–555.
- [8] FROST, D., AND DECHTER, R. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the fourteenth International Joint Conference on Artificial Intelligence* (Montreal, 1995), pp. 572–578.
- [9] GOTTLÖB, G., LEONE, N., AND SCARCELLO, F. A comparison of structural csp decomposition methods. In *Proceedings of IJCAI'99* (1999), pp. 394–399.
- [10] HABBAS, Z., HERRMANN, F., MÉREL, P.-P., AND SINGER, D. Load balancing strategies for parallel forward search algorithm with conflict based backjumping. In *Proceedings of the International Conference on Parallel and Distributed Systems* (Séoul, 1997).
- [11] JÉGOU, P. *Contribution à l'étude des problèmes de satisfaction de contraintes : algorithmes de propagation et de résolution – propagation de contraintes dans les réseaux dynamiques*. PhD thesis, Université des Sciences et Techniques du Languedoc, Montpellier, 1991.
- [12] JÉGOU, P. Csp decomposition methods, parallel implementations, a future prospect? In *Proceedings of JIM'99* (Metz, France, 2000).
- [13] KASIF, S., AND DELCHER, A. L. Local consistency in parallel constraint satisfaction networks. *Artificial Intelligence* 69 (1994), 307–327.
- [14] KEYES, D., SAAD, Y., AND TRUHLAR, D. *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*. SIAM, Philadelphia, 1994.
- [15] MACKWORTH, A. K. Consistency in networks of relations. *Artificial Intelligence* 8 (1977), 99–118.
- [16] MÉREL, P.-P. *Les problèmes de satisfaction de contraintes : recherche n-aire et parallélisme – Application au placement en CAO*. PhD thesis, Université de Metz, 1998.
- [17] MONTANARI, U. Networks of constraints: Fundamental properties and applications to pictures processing. *Information Sciences* 7 (1974), 95–132.
- [18] OPENMP ARCHITECTURE REVIEW BOARD. *OpenMP C and C++ Application Program Interface*, Oct. 1997. <http://www.openmp.org>.
- [19] PROSSER, P. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9 (1993), 268–299.
- [20] PROSSER, P. Binary constraint satisfaction problems: Some are harder than others. In *Proceedings of the eleventh European Conference on Artificial Intelligence, ECAI'94* (Amsterdam, 1994), pp. 95–99.
- [21] SMITH, B. M. Phase transition and the mushy region in constraint satisfaction problems. In *Proceedings of the eleventh European Conference on Artificial Intelligence* (1994), pp. 100–104.
- [22] TARJAN, R. E., AND YANNAKAKIS, M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *Siam journal of computing* 13 (1984), 566–579.