PARALLEL GENERATION FOR HIERARCHICAL DECLARATIVE SCENE MODELING

Pierre-François Bonnefoi, Dimitri Plemenos
MSI Laboratory, 83 rue d'Isle, 87000 Limoges FRANCE
E-mail : bonnefoi, plemenos@unilim.fr

ABSTRACT

Declarative modeling transforms deeply the task of scene designing by adding to it flexibility, and more creativity and by being more user-friendly. However it's difficult to generalize its use, because it requires too much processing time and memory. A possible way for improving the efficiency of declarative modeling is to parallelize the scene generation process. In this paper, a method for parallelizing the generation of the first solution is presented and applied to hierarchical declarative modeling. It is very important to get the first solution as soon as possible because this solution gives the user the possibility to interactively guide the remaining of the resolution process.

KEYWORDS: Declarative Modeling, Constraint Logic Programming, Object Oriented Programming, Parallel scene generation.

1 INTRODUCTION

The intention of this paper is to report the status of ideas and concepts presented within the last two years through two previously published papers, one at SCCG'98 ([BP98]) and the other at WSCG'99 ([BP99]). The presented results are obtained through the finishing and the release of a Ph.D. thesis ([Bonn99]), and the implementation of a new version of the declarative modeler MultiFormes ([Pleme91]).
The purpose pursued by this work is to demonstrate the prevalence of declarative modeling over the geometric modeling. Imperative modeling, or classical geometric modeling, is practiced by giving all the characteristics of the scene in order to construct it. So, the user must have a complete knowledge about the scene and its different parts, from the coordinates of each point to the complete spatial structure of each object. No room and no support are provided in the modeler to maintain some useful information like non geometric information and very useful information like logical structure of the scene, that is: relationships between the different parts of the scene. A contrario, the declarative modeler offers a way to handle geometrical informations and to use the logical informations in order to construct the scene.
So, the user provides a description of the scene, made from properties applied on the different elements of the scene he wants to obtain and this description is used by the modeler to construct the scene satisfying the description.
Declarative modeler offers many good features: freeing the user from giving all the details of the scene and letting him reach a more abstract level of description, combining logical information to ensure that a produced scene fulfil them, generating creative scenes from the description that could be imprecise. The declarative modeler is also more close to the progressive work of modeling a scene.
The major obstacle that prevents declarative modeling to be massively adopted is due to its most impressive feature: let the user give an imprecise description So, the declarative modeler can provide a large amount of scenes that are fulfilling the description and the amount of time required to construct these scenes can be prohibitive.
Significant improvements have been achieved by using new techniques ([BP99]), but a new step has been reached by parallelizing the work of the declarative modeler.
Parallelization has been introduced in MultiFormes in order to extend its capabilities without sacrificing its object oriented design and its most impressive enhancements. The purpose to achieve is not to define the best parallelization scheme, but to obtain

significant improvements without lost of the declarative behavior of the modeler, *i.e.* by using automatic parallelization. MultiFormes is the implementation and the studying framework for a special and very efficient form of declarative modeling based on the so-called "hierarchical decomposition".
In this paper, a description of MultiFormes is provided, followed by an explanation of the parallelization process. Finally, some results are given and discussed with a comparison made with the "sequential" version of MultiFormes.

2 DECLARATIVE SCENE MODELING BY HIERARCHICAL DECOMPOSITION

2.1 Declarative modeling and modeled objects

Declarative modelers are characterized by the kind of the smallest element that can be characterized in the description given at the input of the modeler and the relation between these elements and those provided at the output of the modeler. In the case of MultiFormes the elements handled at the input and output are bounding boxes. So, a scene produced by MultiFormes is constituted of "objects" given basically by their bouding boxes. Each of these bounding box, is described by a position (x, y and z coordinates) and a displacement vector defining the width, the height and the depth of the associated bounding box (eventually some other relevant information are associated like the colors or the texture of the object). Each bounding box is described by at least six numeric variables.
So, properties in MultiFormes are essentially related to position and size of one bounding boxes in relation with others. The work of the declarative modeler consists in generating all the scenes verifying the given high level properties.
In order to allow computation of all possibilities, the different variables characterizing the scene must take a finite number of values (a bounded set of integers for example). The exhaustive exploration of all possible values according to the achievement of a solution, combined to a large amount of bounding boxes (therefore a large amount of variables) and an imprecise description, can yield to a combinatorial kind of problem (classified as NP-Hard problem).
For example, the scene "building with 3 floors" presented in the fifth section of this article, is described with 2500 variables and the average number of values that can be taken by each of these variables is 21. So, the number of combinations that the modeler can potentially explore is 21 to the power of 2500.
This is explained by the fact that the space of elements that can be handled in input of the modeler is the space of solutions provided in output of the modeler.

2.2 The hierarchical decomposition

The declarative modeling by hierarchical decomposition is specific to MultiFormes ([Pleme95]). This technique, WSCG'99, employs top-down description and criteria of complexity to partition the scene in various sub-scenes: *When a scene is too difficult to describe with the language recognized by the modeler (it requires a too large amount of properties), it is partitioned recursively into two or more sub-scenes easier to describe, in accordance with the logical and spatial structure of the whole scene.* An example is given below (see Fig.1), where a "hierarchical decomposition tree" is associated to the description :
to each scene of the description, a node of the hierarchical decomposition tree is related (the node "Building" for the scene). For each node, the set of all sub-scenes produced by its decomposition corresponds to the set of children nodes.

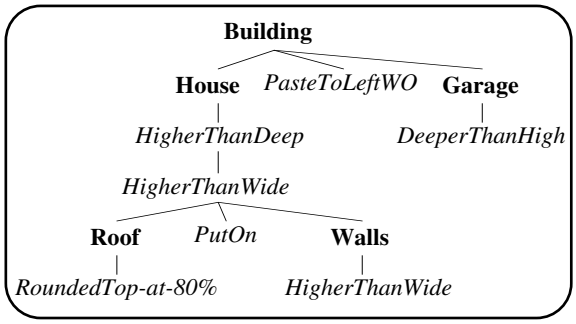

Fig.1 — An example of hierarchical description giving a tree: The hierarchical decomposition tree.

The hierarchical decomposition offers some major advantages to the declarative modeling:
— offers the ability to specify a scene at different levels of detail,
— enables the designer to express scenes in a gradual manner,
— enforces the locality of the description (the designer's scope is focused on explaining a part of the whole scene without thinking about other parts)
— allows factorizing of the properties,
— induces strong relationships between a scene and its sub-scenes (inheritance of constraints for a node from its parent: the bounding box of each sub-scene is included in the bounding box of the scene from which it arises).
— allows interaction with the users for guiding the generation process: the user can choose a partial solution according to a particular depth level in the tree (similar to fold some nodes into leafs), and from this solution as mould, he can obtain complete solution. This is a special kind of interactivity based on the levels of detail.

2.3 Some concerns about MultiFormes

— All possibilities must be explored to design a scene from its description. When the designer enters a description in the modeler, he expects to obtain all occurrences of scenes filling his description.
— There is a strong independence between elements of sub-trees issued from the same node of the hierarchical decomposition tree. Properties can only be expressed on features of only one node or one node and its direct children nodes. So there is no constraints linking variables characterizing a node with variables characterizing an other node if these nodes don't share a parent link (all properties in MultiFormes can be expressed as numeric constraints applied on the variables describing the whole scene).
This feature yield to the definition of a graph that exhibits all possible links that can be established between the full set of variables characterizing one node and a variable characterizing another node (see Fig.2).

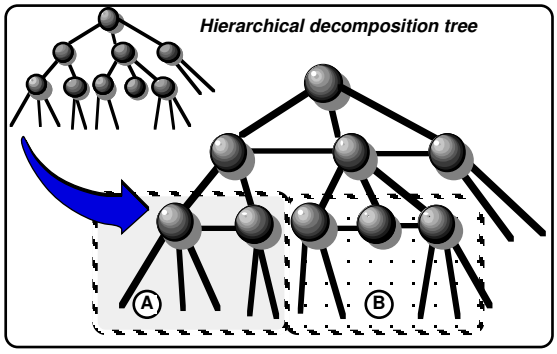

Fig.2 — A constraint graph obtained from a hierarchical decomposition tree, and two independent areas A and B.

In Fig.2 , the area "A" is independent from the area "B". The concept of "*sub-level*" can be defined: *a sub-level is defined by all the nodes from the same level of the hierarchical decomposition tree that are issued from the same parent node.* So the independence between different parts of the hierarchical decomposition tree is expressed by examining the different sub-levels of the hierarchical decomposition tree: *all the sub-levels from the same level of the hierarchical decomposition tree are independent from each other.* All the sub-trees issued from one of these sub-levels is independent from all sub-trees issued from another of these sub-levels (from the same level of the hierarchical decomposition tree).
— Interactivity is used in order to allow guidance by the user of the generation process of the declarative modeler. Different kinds of interactivity are provided into the modeler: selection of resolution method according to the level of detail, adding of properties during the generation of the solution scenes by using comments on a given scene (this scene is too large, the width is correct etc.), freezing of some chosen features (the width of this scene must be maintained fix).

3 PARALLELIZATION

3.1 The parallelization method

A possible way to reduce the time complexity of declarative modeling is to achieve the parallelization of the most expensive work, scene generation.
No common scheme exists to achieve the parallelization, only some and trusted "good recipes" are provided.
In the case of MultiFormes, the choice of producing all possible solution scenes does not allow the use of numerical resolution for the set of constraints with algebraic methods, and the well-known associated parallelization processes.
So, parallelizing of the generation process can be done in five phases
— Partitioning of the program into a set of tasks allowing a good exploitation of parallelism. This partitioning depends of the amount of needed parallelization: the more possible parallelization to obtain, the more fine division to make.
— Construction of a "Data Dependency Graph", defining the needs for communication between tasks, (what data should be exchanged between the different tasks).
— Construction of a "Precedence Graph", defining the processing order of the tasks. If this order is too strict, the possibilities of exploitation of parallelism will be very limited.
From the precedence graph, the constraints time are determined and by this way the tasks that are independent from each other and so, that could be processed in parallel.
— Establishment of a "Communication Graph", from the precedence graph and the data dependency graph. This graph organizes the exchange between the different tasks of the data processed by these tasks (bound to the input or output variables).
— Putting together some tasks, taking into account communication and synchronization costs needed to compute these tasks. This operation defines the "grain" of parallelization.
— Additional grouping of tasks, depending on available resources (amount of available processors for example).
It is important to note that the precedence graph could be not known before the run of the application, so it could be with the different tasks.

3.2 Parallelization framework for MultiFormes

The parallelization must be applied to the generation of a scene.
The amount of time needed to obtain the first solution scene is frequently very large, at the contrary of the amount of time required to get the next scene solution. The use of MultiFormes is based on the powerful usage of interactivity. Interactivity reduce greatly the number of solution scenes to browse before obtaining the desired scene(s). The sooner the use of interactivity could take place, the more improvement could be achieved.
So, the generation of the first scene must be accelerated and must get benefit from the parallelization.
The declarative modeler must remain declarative: the modeler must not substitute to the hassle of geometric scene design, the hassle to guide the parallelization of the scene generation.
So, the parallelization must be made automatically by the modeler without user intervention.
The parallelization relies on the description given by the user in input of the declarative modeler.
The process of generating the solution scenes depends of the description given by the user. This description organizes the resolution process due to its hierarchical structure and the elements involved (properties and constraints induced, scene and numeric variables associated).
So, the parallelization process depends on the analyzing of the description and must be dynamically made.
The parallelization process must be integrated in the existing object oriented version.
The current state of MultiFormes has been designed to provide a object oriented framework to study different techniques, methods and improvements. Parallelization must be smoothly integrated in the object oriented design.
So, the model of parallelization according to "Message passing" and "Communicating Sequential Processes" has been chosen and mapped to object oriented behavior.
An available parallel machine must be chosen.
The choice of parallel machine as target for the parallelization process induces some strong design constraints. The most available computer resources are the personal workstations.
So, a network of personal workstations has been used as a virtual parallel machine (use of Parallel Virtual Machine library [BDG95]). The grain of parallelism is less fine that with a real parallel machine and the number of processor is greatly reduced.

4 RESOLUTION PROCESS AND PARALLELIZATION

4.1 The use of a special form of binary constraint and of CSP techniques

The current state of MultiFormes uses "Constraint Solving Problem" associated techniques in order to generate the set of solution scenes.
All properties used in MultiFormes consists of a set of numeric constraints. Each of these constraints is rewritten automatically by the resolution engine of the declarative modeler in order to obtain only binary constraints.
So, the whole generation process can be seen as the resolution of a CSP, and associated resolution techniques can be used successfully, in particular the "Arc consistency" algorithm ([Mackw77], [Kumar92], [Tsang93], AC-5 algorithm).
To take the most benefit of arc-consistency, the binary constraints must be chosen to fit the best the needs of arc-consistency algorithm (use of "min-max" representation for example).
This powerful behavior is obtained by the use of a special form of binary constraint: the so-called "primitive constraint" ([Hente89], [CD96]). So, each constraint is automatically rewritten in terms of set of primitive constraints.
This approach offers a declarative manner to express new constraints (without having to know how the modeler works and without the hassle to manually having to optimize the expression of the constraints).

4.2 The resolution process: a two-phases process

Solving a CSP can be defined as: Each variable must take a single value according to the satisfaction of all constraints.

4.2.1 The propagation phase

The arc-consistency algorithm defines one of the phases of resolution: the "propagation".
This algorithm is based on a special view of the constraint problem:
the variables are viewed as nodes and a constraint applied on two variables as an arc between the two connected nodes. A graph is obtained illustrating the relationships between variables, the link established representing a way to check satisfiability of these relationships, by carefully examining all values that could be taken by the two nodes; this introduces the arc-consistency concept. Each variable is defined according to a finite domain of values.
Arc Consistency: *Arc (Vi, Vj) is arc consistent if for every value x in the current domain of Vi, there is some value y in the domain of Vj such that Vi =x*

and Vj =y is permitted by the binary constraint between Vi and Vj. The concept of arc-consistency is directional; i.e., if an arc (Vi,Vj) is consistent, then it does not automatically mean that (Vj,Vi) is also consistent.
The method for achieving arc-consistency is to select each arc of the graph, and then filter any inconsistent values from each interval involved by the constraint (with one node and the other to accomplish bidirectional checking) and, in case of interval modification, propagate this modification to all other arcs affected (i.e. all the arcs sharing the altered node).

4.2.2 The selection phase

Each variable of the problem must take a value, so each of these variables must be selected and for one variable each of its possible values must be tried.
This is due to the fact that all solution scenes must be provided by the modeler.
These different selections can be made according to different orders, each of these orders yielding to different results for the amount of time required to produce solution.
The definition of orders that lead to the most impressive results take into account the hierachical structure of the description given in input of the modeler ([BP99]).

4.2.3 The resolution process

The resolution process consists of different steps that eventually can be repeated:
— Selection of a variable
— Selection of a value for this variable
— Propagation of the effect of this selection to the domains of other variables through the affected constraints.
— In case of failure (the domain related to a variable becomes empty), a "backtrack" is made:
choose another value for the variable or chronologically another variable with not all of its values already chosen.
— In case of success: restart the resolution process until all variables have taken a value.

4.3 Independence during the resolution process

The generation of a scene solution relies on the resolution process viewed above. In order to proceed to the parallalelization of this process, some independencies must be determined between some tasks involved in this process.
The selection phase is intrinsically sequential and affects all variables of the whole scene.
The propagation phase starts from the current selected variable and concerns only variables that share constraints with it, or variables for which a path of constraints exists in the graph constraints.
So, the propagation process can run from a node to another if a constraint links these two nodes.
If a variable has already taken a value no more modification can be propagated from it. So, an instanciated node blocks the propagation.

During the resolution process, the different phases alternate with each other.
In 2.3, the complete possible graph of constraint deduced from the hierachical decomposition tree has been defined.
From the study of this graph some independence for the resolution process can be obtained (see Fig.3).
The enumeration process order is made according to an "in-depth" and "in-width" walk of the hierarchical decomposition tree. When the first node is instanciated, the propagation can affect all parts of the tree. When the second node is instanciated the propagation can touch all parts of the tree, but not the already instantiated node 1. When the third node is instanciated, the propagation is blocked by node 2 and could not affect the sub-tree of this node.

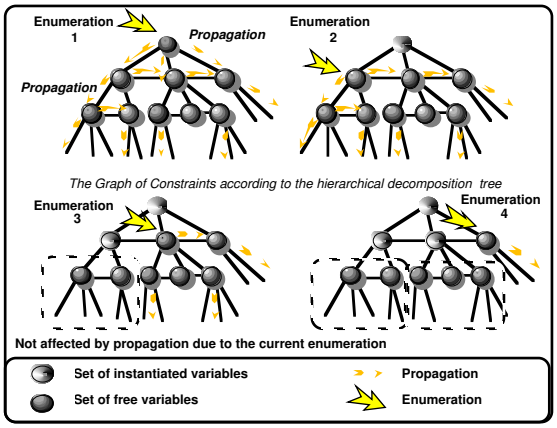

Fig.3 — Independence during the resolution process.

So, after enumerating of node 1, 2 and 3, the sub-trees associated to node 2 and node 3 are independent and could be resolved independently and so concurrently.
A new definition can be given:
The sub-levels from the same level of the hierarchical decomposition tree and the set of all of their sub-trees are independent from each other, if all the variables from the previous nodes (from a superior level) are already instanciated.

4.4 Parallelization of the resolution process

Given a hierarchical decomposition tree, this tree could be partitioned into different parts in order to exploit parallelism.

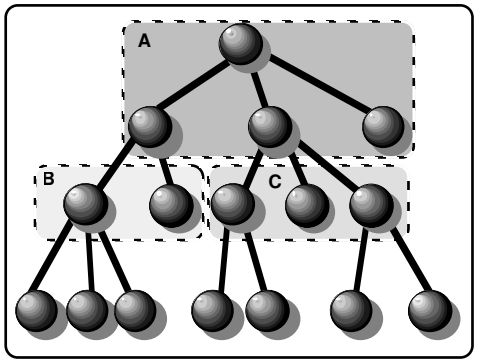

Fig.4 — Different parts of a heriarchical decomposition tree defined in order to allow parallel resolution.

In this example (see Fig.4), the tree has been divided into three different parts: the first level from the top of the tree that contains more than one sub-level is selected and each of these sub-levels defines a different part (part B and part C). The part A corresponds to the nodes that must be instantiated in order to get part B and part C independant during the resolution process.
All required information has been determined in order to organize the parallel resolution process:

— the part A is the "must be shared" part of the parallel resolution: the values taken by the variables that belong to this part, and the resulting propagations (towards parts B and C) must be transmitted to parts B and C.
— the parts B and C that represent the amount of work that could be made concurrently.

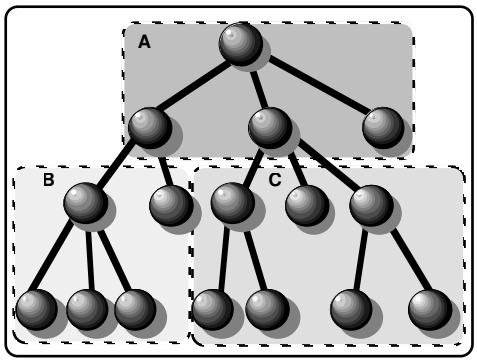

Fig.5 — The different parts that will be distribute between the processors of the parallel machine.

So, we can define the algorithm that allows a parallel resolution according to a fast analysis of the hierarchical decomposition tree:
First, distribute the work among the processors of the parallel machine.

— determine the level of the tree that contains some sub-levels. The number of sub-levels represents the maximal number of processors that could be used for the parallel resolution. More this level is near the top of the tree, less number of processor could be used but more the sub-trees associated to the nodes of each sub-level are large. At the contrary, more this level is far from the top of the tree, more number of processors could be involved in the resolution process and more the sub-trees associated to the nodes of each sub-level are small. So, a shared part is obtained (related to the top of the tree) and a set of potentially independent parts.
— distribute between each processor of the parallel machine a sub-level with all its associated sub-trees.
Each processor receives the responsibility to explore all solutions for its part of the initial tree.

Each processor computes a different sub-solution, so, in order to obtain a complete solution, all the sub-solutions must be combined with each other.
A "compatibility criterion" must be defined in order to allow this combination:
• All these different solutions rely on a particular solution for the shared part.
• All sub-solutions can be combined if they are obtained for the same solution of the shared part.
Therefore, each solution of the shared part is numbered and this number is used to tag each sub-solution, and all sub-solutions obtained with the same tag can be combined.

Second, organize the resolution process.

— synchronize the work of the different processors:
• before letting each processor work concurrently to elaborate a solution for its given part, a solution for the shared part must be obtained (part A in Fig.5). Obtaining a solution for the shared part can affect by propagation the domain of variables associated to the distributed parts.
• during the resolution, the different processors must synchronize with each other in the case of failure. If a complete failure occurs for a processor, a new solution for the shared part must be computed and the tag of this solution will be incremented. So, all processors that are being computing a solution for the previous value of the tag must stop and go to the next value.
• after getting a sub-solution, each processor must wait in case of failure of an other processor that forces it to compute a new sub-solution.

Third, present a complete solution to the user.

— centralize all the obtained sub-solutions into the processor in charge of the user interface,
— present the whole solution to the user.

5 RESULTS

Various scenes have been used in order to evaluate the parallel resolution process:

— "building with one floor": the floor contains a school room with some chairs grouped into ranges and regurlaly interleaved and with a desk centered on a platform (for information: 868 variables, 6233 constraints and 143 nodes for the hierarchical decomposition tree).

— "building with two floors": the description is the same that above for each floor, but the description of a floor is more constrained and so, quicker to generate (1726 variables, 12372 constraints and 285 nodes),

— "building with three floors": same description for each floor as above, just one floor added (2588 variables, 18558 constraints and 427 nodes),

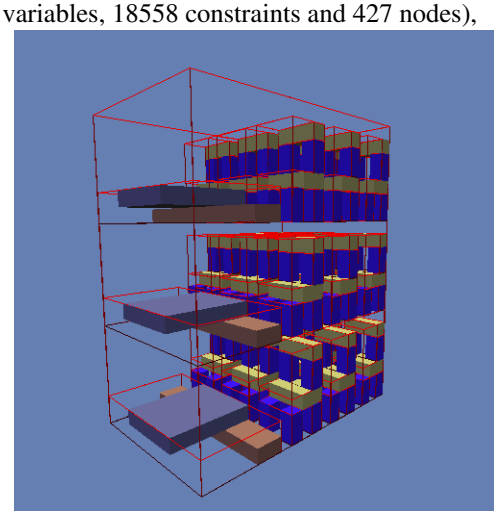

Fig.6 — The scene "building three floors".

— "little castle": the description uses dungeon, walls crenels, the hierachical description decomposes the scene according to three areas: "front" (walls, towers and portal), very large "center" (the yard with two walls to each side of it) and "back" (walls, towers and dungeon), (445 variables, 2881 constraints and 67 nodes),

— "castle (x)": the same as above but bigger, with more walls elements. Two methods have been used to define variants A and B:

• A: relaxation of dimensions of the walls according to the width, but the depth of the scene is more constrained and the "center" will get its proper size fast during the resolution process.

• B: relaxation of dimensions of tha walls according to the depth, but the width of the scene is more constrained. In this case the "center" will need a huge amount of time to get its proper size.

Scene	Sequential resolution	Parallel resolution	Processors used	Percentage of gains
Building 1 floor	21 minutes 41 secondes	12 minutes 17 secondes	4	44%
Building 2 floors	2 minutes 29 secondes	1 minute 33 secondes	2	38%
Building 3 floors	7 minutes 3 secondes	2 minutes 41 secondes	3	62%
Little Castle	3 minutes 4 secondes	2 minutes 20 secondes	3	24%
Big Castle (A)	1 hour 24 minutes	54 minutes 53 secondes	3	35%
Big Castle (B)	>24 heures	1 hour 15 minutes	3	:-)

Table 1 — Results of sequential and parallel resolution.

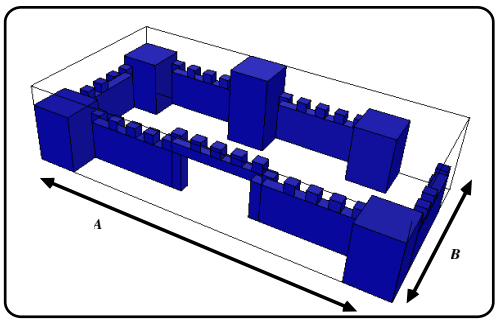

Fig.7 — The scene "castle".

From Table 1, some comments can be deduced:

— No additional cost has been noted due to the parallelization, all scenes are produced faster or with the same amount of time with the parallel version of the modeler.

— The parallel resolution offers not only improvement for reducing the time needed to obtain a solution scene, but also a new method of resolution (as in the case of the scene "castle (B)"). The sequential version explores the hierarchical decomposition tree with a fixed order and can not detect some kinds of failure (not properly sized "center" area that leads repeatedly to a complete failure). The parallel version explores concurrently

different parts of the hierarchical decomposition tree and so, can detect these kinds of failure.
— The gain of acceleration obtained is always superior to 20%, with use of a limited number of workstations (model “DEC Personal WorkStation 500au” with system “OSF v4.0”). The percentage of gain is the ratio of the difference between the sequential time and the parallel time needed for the resolution of the first solution scene, by the sequential time.

6 CONCLUSION AND FUTURE WORK

The integration of parallelization into MultiFormes has been made by extension of the object oriented design of the modeler (objects are transmitted through the network of personal workstation between the different processors). No benefit from the object oriented design has been lost.

The parallel version takes benefit of all the improvements of the sequential version (“intelligent backtracking” for example [BP99]). Each processor of the parallel machine runs an enhanced version of the sequential version of the resolution engine.

The parallel process relies on the structure of the decomposition tree: the more the tree is deep and well-balanced, the more parallelism can be exploited. The design is also easier for the user.

The generation of all solution scenes is very powerful due to the fact that all sub-solutions can be processed concurrently and combined by cartesian product in order to get all complete solutions.

The future of this work would consist of:
— Use of more parallelism: dividing the hierarchical decomposition tree in more sub-trees and dividing these sub-trees according to the parallelization scheme (more complex synchronizing will be needed).
— Generation of all solutions by cartesian product: it will require to define and use an object oriented database to maintain the knowledge of all the generated sub-solutions.
— Use of more processors with a larger network of less powerful workstations (typically a set of PCs under Linux).

REFERENCES

[BDG95] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam, Recent Enhancements to PVM, International Journal of Supercomputing Applications and High Performance Computing, Vol. 9, No. 2, Summer 1995.

[Bonn99] Bonnefoi Pierre-François, Constraint Satisfaction techniques for Declarative Modeling, PhD. Thesis, June 99, Limoges, France.

[BP98] Bonnefoi Pierre-François, Plemenos Dimitri, Proposals for Declarative Modeling Parallelization, in SCCG’98, April 98, Bratislava, Slovakia.

[BP99] Bonnefoi Pierre-François, Plemenos Dimitri, Object Oriented Constraint Satisfaction for Hierarchical Declarative Scene Modeling, in WSCG’99, February 99, Plzen, Czech Republic.

[CD96] P. Codognet, D. Diaz, Compiling Constraints in CLP(FD), *The Journal of Logic Programming* 1996:27:1-199.

[Hente89] Van Hentenryck P., *Constraint Satisfaction in Logic Programming.* Logic Programming Series, MIT Press, 1989.

[Kum92] Kumar Vipin, Algorithms for Constraint Satisfaction Problems: A survey, Appeared in *AI Magazine* 13(1):32-44, 1992.

[Mackw77] Mackworth A. K., Consistency in networks of relations. *Artificial Intelligence*, vol. 8, no 1, 1977.

[Pleme91] Plemenos Dimitri, *Contribution à l’étude et au développement des techniques de modélisation, génération et visualisation de scènes - Le projet MultiFormes.* Professorial dissertation, Nantes, november 1991.

[Pleme95] Plemenos Dimitri, Declarative modeling by hierarchical decomposition. The actual state of the MultiFormes project. In GraphiCon’95, St Petersbourg (Russie), 3-7 july 1995.

[Tsang93] Tsang Edward, *Foundations of Constraint Satisfaction*, Computation in Cognitive Science, Academic Press, ISBN 0-12-701610-4.