Methodology of Parallel Programming using Compositions of Parallel Objects or CPANS for the solution of NP-Complete problems

Mario Rossainz López
Faculty of Computer Science
University Autonomous of Puebla
72000, Puebla, México
E-mail: mariorl@siu.buap.mx

Manuel I. Capel Tuñón
E.T.S. Ingeniería Informática
University of Granada
18071, Granada, Spain
E-mail: mcapel@ugr.es

KEYWORDS
Parallel Objects, Structured Parallel Programming, High Performance Computing, Object Oriented Programming, Concurrent Programming, CPANs.

ABSTRACT

Within an environment of Parallel Objects, an approach of Structured Parallel Programming and the paradigm of the Orientation to Objects, shows a programming method based on High Level Parallel Compositions or HLPCs (CPANs in Spanish) by means of classes. The synchronous, asynchronous communication ways and asynchronous future of the pattern of Parallel Objects (Rossainz and Capel 2005-2), the predefined patterns of communication/interaction of the structured approach, the encapsulation and the abstraction of the Orientation to Objects, to provide reusability to this patterns, together with a set of predefined restrictions of synchronization among processes (maxpar, mutex, sync) are used. The implementation of the commonly used communication patterns is explained, by means of the application of the method, which conform a library of susceptible classes of being used in applications within the environment of programming of the C++ and of the standard POSIX of programming with threads such as the solution to the Travelling Salesman Problem (TSP) by means of the parallel Branch & Bound technique like a CPAN.

INTRODUCTION

As it is known, exist infinity of applications that using machines with a single processor tries to obtain the maximum performance from a system when solving a problem; however, when such a system can not provide the performance that is required (Capel and Troya 1994), a possible solution it consists on opting for applications, architectures and structures of parallel or concurrent processing. The parallel processing is therefore, an alternative to the sequential processing when the limit of performance of a system is reached. In the sequential computation a processor only carries out at the same time an operation, on the contrary of what happens in the calculation parallel, where several processors they can cooperate to solve a given problem, which reduces the time of calculation since several operations can be carried out simultaneously. From the practical point of view, today in day is enough justified carrying out compatible investigations within the area of the parallel processing and areas related (Concurrence, Distributed Systems, Systems of Real Time, etc.), since the recent advance in massively parallel systems, communications of great band width, quick processors for the treatment of signs, etc., they allow this way it. Important part of those investigations are the parallel algorithms, methodologies and models of parallel programming that at the moment are developing. The parallel processing includes many topics that include to the architectures, algorithms, languages of programming parallel and different methods of performance analysis, to mention some of the most excellent.
The present investigation centers its attention in the Methods of Structured Parallel Programming, proposing a new implementation with C++ and the library of threads POSIX of the programming method based on the pattern of the High Level Parallel Compositions or CPANs (Corradi 1995; Danelutto), the which it is based on the paradigm of Orientation to Objects to solve problems parallelizable using a class of concurrent active objects. In this work supply a library of classes that provides the programmer the communication/interaction patterns more commonly used in the parallel programming, in particular, the pattern of the pipeline, the pattern denominated farm and the pattern tree of the technique Divide and Conquer of design of algorithms, well-known as it.

MOTIVATION

At the moment the construction of concurrent and parallel systems has less conditioners every time, since the existence of systems parallel computation of high performance, or HPC (High Performance Computing), more and more affordable, has made possible obtaining a great efficiency in the processing of data without the cost is shot; even this way, open problems that motivate the investigation in this area still exist; in particular, interest us those that have to do with the parallel applications that use communication patterns predetermined among their component software. At the moment are identified as important open problems, at least, the following ones:
The lack of acceptance of environments of parallel programming structured to develop applications: the

Proceedings 23rd European Conference on Modelling and Simulation ©ECMS Javier Otamendi, Andrzej Bargiela, José Luis Montes, Luis Miguel Doncel Pedrera (Editors)
ISBN: 978-0-9553018-8-9 / ISBN: 978-0-9553018-9-6 (CD)

structured parallelism is a type of parallel programming based on the use of communication/interaction patterns (pipelines, farms, trees, etc.) predefined among the processes of user's application. The patterns also encapsulate the parallel parts of this application, of such form that the user only programs the sequential code of this one. Many proposals of environments exist for the development of applications and structured parallel programs, but until the moment, they are only used by a very limited circle of expert programmers. At the moment, in HPC, a great interest exists in the investigation of environments as those previously mentioned ones.

The necessity to have patterns or high level parallel compositions: a high level parallel composition or CPAN, as well as is denominated in (Corradi 1995), it must be been able to define and to use within an infrastructure (language or environment of programming) oriented to objects. The components of a parallel application not interaction in an arbitrary way, but regular basic patterns follow (Hartley 1998). An environment of parallel programming must offer its users a set of components that implement the patterns or CPANs more used in algorithms and parallel and distributed applications, such as trees, farms, pipes, etc. The user, in turn, must can to compose and to nest CPANs to develop programs and applications. The user must be limited to a set of predefined CPANs, but rather, by means of the use of the inheritance mechanism, he must can to adapt them to his necessities. The development environment must contemplate, therefore, the concept of class of parallel objects. Interest exists in exploring the investigation line related with the definition of complete sets of patterns, as well as in its semantic definition, for concrete classes of parallel applications.

Determination of a complete set of patterns as well as of their semantics: in this point, the scientific community doesn't seem to accept in a completely satisfactory way and with the enough generality none of the solutions that have been obtained to solve this problem today. It doesn't seem, therefore, easy the one that can be found a set the sufficiently useful and general thing, for example a library of patterns or set of constructos of a programming language, to be used in the development, in a structured way, of a parallel application not specific.

Adoption of a approach oriented to objects: Integrating a set of classes within an infrastructure oriented to objects is a possible solution to the problem described in the previous point, since would allow adding new patterns to an incomplete initial set by means of the subclasses definition. Therefore, one of the lines of followed investigation has been finding representations of parallel patterns as classes, starting from which instance parallel objects is been able to (CPANs) that are, in turn, executed as consequence from an external petition of service to this objects and coming from user's application. For example, the derived pattern of the execution for stages of the processes would come defined by pattern's denominated pipeline class; the number of stages and the sequential code of each specific stage would not be established until the creation of a parallel object of this class; the data to process and the results would be obtained of user's application; the internal storage in the stages could adapt in a subclass that inherits of pipeline. Several advantages are obtained when following a approach oriented to objects (Corradi and Leonardi 1991), regarding a approach only based on skeletons algorithmic and programs model (Hartley 1998), it is necessary to point out, for example, the following improvements:

Uniformity: All the entities within the programming environment are objects.

Genericity: The capacity to generate references dynamically, within an environment of software development oriented to objects, makes possible the creation of generic patterns, by means of the definition of its components as generic references to objects. Reusability: The inheritance mechanism simplifies the definition of specialized parallel patterns. The inheritance applied to the behavior of a concurrent object helps in the specification of the parallel behavior of a pattern.

EXPOSITION OF THE PROBLEM

From the work carried out to obtain the investigating sufficiency, redefining and modernizing the investigation, the problem to solve is defining a Parallel Programming Method based on High Level Parallel Compositions (CPANS) (Corradi 1995). For it the following properties have considered as indispensable requirements that should be kept in mind for the good development of this investigation. It is required, in principle, a environment of Programming Oriented to Objects that it provides: Capacity of invocation of methods of the objects that contemplates the asynchronous communication ways and asynchronous future. The asynchronous way doesn't force to wait the client's result that invokes a method of an object. The asynchronous future communication way makes the client to wait only when needs the result in a future instant of her execution. Both communication ways allow a client to continue being executed concurrently with the execution of the method (parallelism inter-objects).

The objects must can to have internal parallelism. A mechanism of threads it must allow to an object to serve several invocations of their methods concurrently (parallelism intra-objects).

Availability of synchronization mechanisms when parallel petitions of service take place. It is necessary so that the objects can negotiate several execution flows concurrently and, at the same time, to guarantee the consistency of their data.

Availability of flexible mechanisms of control of types. The capacity must be had of associating types dynamically to the parameters of the methods of the objects. It is needed that the system can negotiate types of generic data, since the CPANs only defines the parallel part of an interaction pattern, therefore, they must can to adapt to the different classes of possible components of the pattern.

Transparency of distribution of parallel applications. It must provide the transport of the applications from a system centralized to a distributed system without the user's code is affected. The classes must maintain their properties, independently of the environment of execution of the objects of the applications.

Performance. This is always the most important parameter to consider when one makes a new proposal of development environment for parallel applications. A approach based on patterns as classes and parallel objects must solve the denominated problem PPP (Programmability, Portability, Performance) so that it is considered an excellent approach to the search of solutions to the outlined problems.

The environment of programming oriented to Objects that it has been considered as suitable to cover the 6 previously mentioned properties is the programming language C++, together with the use of the standard POSIX Thread.

SCIENTIFIC OBJETIVES OF INTEREST

The development of a programming method is based on High Level Parallel Compositions or CPANs that implement a library of classes of utility in the Programming Concurrent/Parallel Oriented to Objects (Rossainz 2005). The method must provide the programmer the commonly used parallel patterns of communication, in such a way that this can exploit the generalization mechanisms for inheritance and parametrization to define new patterns according to the pattern of the CPAN.

HIGH LEVEL PARALLEL COMPOSITIONS OR CPANS

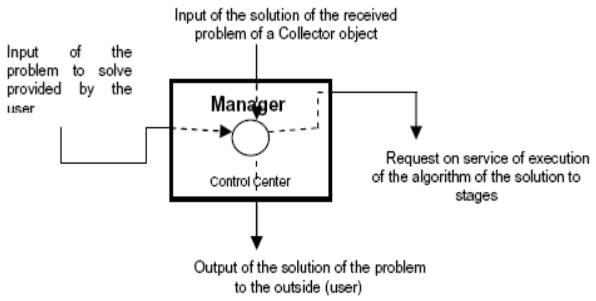
Some of the problems of the environments of parallel programming it is that of their acceptance for the users, which depends that they can offer complete expressions of the behavior of the parallel programs that are built with this environments (Corradi 1995). At the moment in the systems oriented to objects, the programming environments based on parallel objects are only known by the scientific community dedicated to the study of the Concurrence.

A first approach that tries to attack this problem it is to try to make the user to develop his programs according to a style of sequential programming and, helped of a system or specific environment, this can produce his parallel tally. However, intrinsic implementation difficulties exist to the definition of the formal semantics of the programming languages that impede the automatic parallelization without the user's participation, for what the problem of generating parallelism in an automatic way for a general application continues being open. A promising approach alternative that is the one that is adopted in the present investigation to reach the outlined objectives, is the denominated structured parallelism. In general the parallel applications follow predetermined patterns of execution. These communication patterns are rarely arbitrary and not structured in their logic (Brinch Hansen 1993). The High Level Parallel Compositions or CPANs are patterns parallel defined and logically structured that, once identified in terms of their components and of their communication, they can be taken to the practice and to be available as abstractions of high level in the user's applications within an environment or programming environment, in this case the one of the orientation to objects. The structures of interconnection of more common processors as the pipelines, the farms and the trees can be built using CPANs, within the environment of work of the Parallel Objects that is the one used to detail the structure of the implementation of a CPAN.

Definition of the pattern CPAN

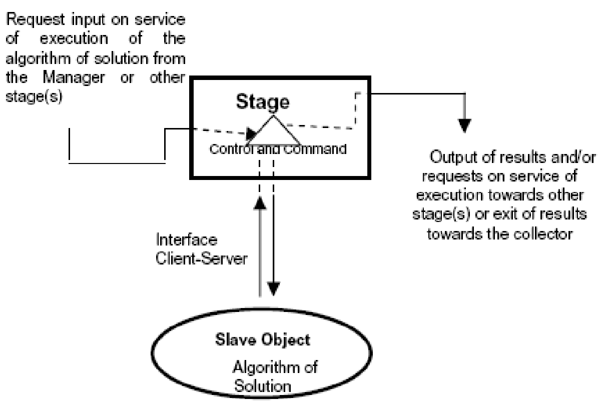
The basic idea is the one of implementing any type of parallel patterns of communication between the processes of an application or distributed/parallel algorithm as classes, following the paradigm from the Orientation to Objects. Starting from this classes, an object can be instanced and the execution of a method of the object in question you can carry out through a petition of service. A CPAN comes from the composition of a set of objects of three types (Rossainz and Capel 2005-2):

An object manager (Figure.1) that it represents the CPAN in itself. The manager controls the references of a set of objects (an object Collector and several objects Stage) that represent the components of the CPAN and whose execution is carried out in parallel and it should be coordinated by the own manager.


Input of the solution of the received problem of a Collector object
Input of the problem to solve provided by the user
Manager
Control Center
Request on service of execution of the algorithm of the solution to stages
Output of the solution of the problem to the outside (user)

Figures 1: Component MANAGER of model CPAN

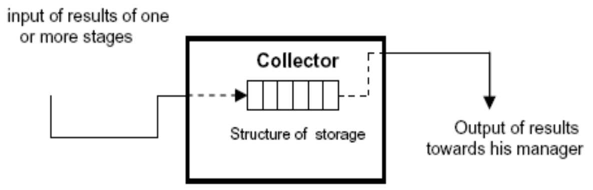
The objects Stage (Figure.2) that are objects of specific purpose, in an interface type client-server that settles down between the manager and the objects slaves (external entities that contain the sequential algorithm that constitutes the solution of a given problem).



Request input on service of execution of the algorithm of solution from the Manager or other stage(s)
Stage
Control and command
Output of results and/or requests on service of execution towards other stage(s) or exit of results towards the collector
Interface Client-Server
Slave Object
Algorithm of Solution

Figures 2: Component Stage of model CPAN and its associated slave object.

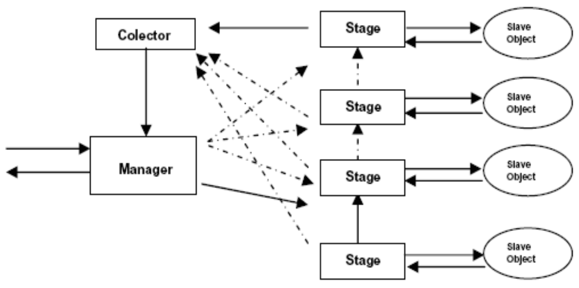
And an object Collector (Figure.3) that it is an object that storing in parallel the results that he receives of the objects stage that has connected.



input of results of one or more stages
Collector
Structure of storage
Output of results towards his manager

Figures 3: Component Collector of model CPAN

Composition of the CPAN.
In summary, a CPAN is composed of an object manager that it represents the CPAN in itself, some objects stage and an object of the class Collector, for each petition that should be treated within the CPAN. Also, for each stage, an object slave will be taken charge of the implementation of the necessary functionalities to solve the sequential version of the problem that you pretend to solve (Figure.4).



Figures 4: Internal structure of a CPAN. Composition of its components

The CPAN seen as composition of parallel objects.
The objects manager, collector and stages are included within the definition of Parallel Object (PO) (Corradi 1995). The Parallel Objects are active objects that have execution capacity in themselves. An object PO has a similar structure to that of an object in Smalltalk, but it also includes a politics of scheduling that specifies the form of synchronizing an or more operations of a class in parallel. The synchronization policies are expressed in terms of restrictions; for example, the mutual exclusion in processes readers/writers or the maximum parallelism in processes writers.

Types of communication in the parallel objects.
The parallel objects define 3 communication ways

1. **The synchronous way** stops the client's activity until the object active server gives him the answer.
2. **The asynchronous way** doesn't force the wait in the client's activity; the client simply sends the petition to the object active server and her execution continues.
3. **The asynchronous future way** makes only wait the client's activity when, within its code, the result of the method is needed to evaluate an expression.

The Synchronization restrictions MaxPar, Mutex y Sync.
It is necessary having synchronization mechanisms, when parallel petitions of service take place in a CPAN, so that the objects that conform it can negotiate several execution flows concurrently and, at the same time, guarantee the consistency in the data that are processing. Within any CPAN the restrictions MAXPAR (The maximum), MUTEX (mutual exclusion among processes that want to consent to a shared object) and SYNC (synchronization of the type producer/consumer) can be used for the correct programming of their methods.

DESIGN AND CONSTRUCTION OF THE CPANS FARM, PIPE Y TREEDV

The parallel patterns worked in the present investigation have been the pipeline, the farm and the treeDV to be a significant set of reusable patterns in multiple applications and algorithms.

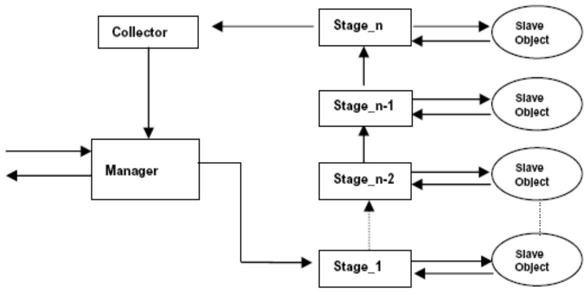
1. **The pipeline**, this compound for a set of interconnected states one after another. The information follows a flow from a state to another.
2. **The farm**, is composed of a set of worker processes and a controller. The workers are executed in parallel until reaching a common objective. The controller is the one in charge of distributing the work and of controlling the progress of the global calculation.
3. **In the treeDV**, the information flows from the root toward the leaves or vice versa. The nodes that are in the same level in the tree are executed in parallel making use of the denominated technique of design of algorithms it Divide and Conquer for the solution of the problem.

These parallel patterns conform the library of classes proposed within the pattern of the CPAN.

The Cpan PipeLine

It is presented the technique of the parallel processing of the pipeline as a High Level Parallel Composition or CPAN, applicable to a wide range of problems that you/they are partially sequential in their nature. The

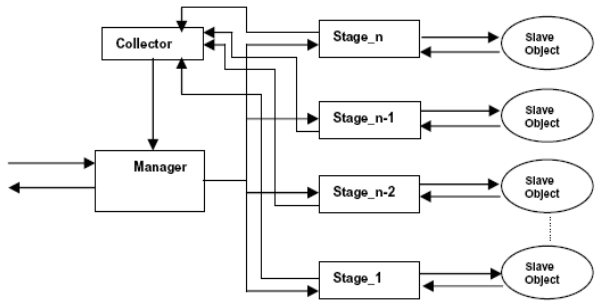
CPAN Pipe guarantees the parallelization of sequential code using the patron PipeLine (see figure 5).



Figures 5: The CPAN of a Pipeline

The CPAN Farm

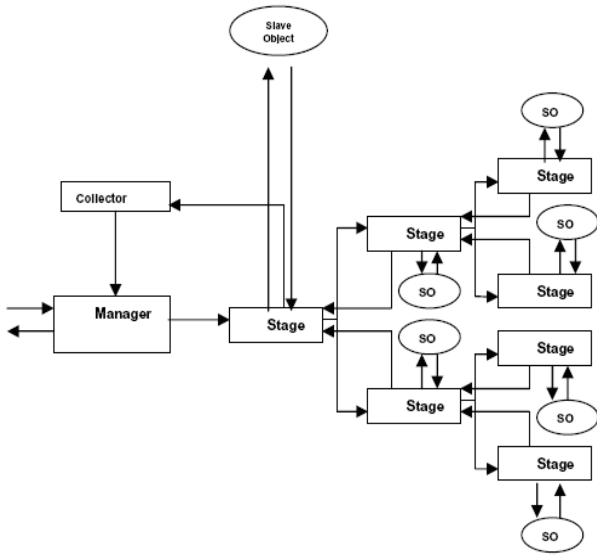
It is shown the technique of the parallel processing of the FARM as a High Level Parallel Composition or CPAN. The representation of parallel pattern FARM as a CPAN is show in Figure. 6.



Figures 6: The Cpan of a Farm

The Cpan TreeDV

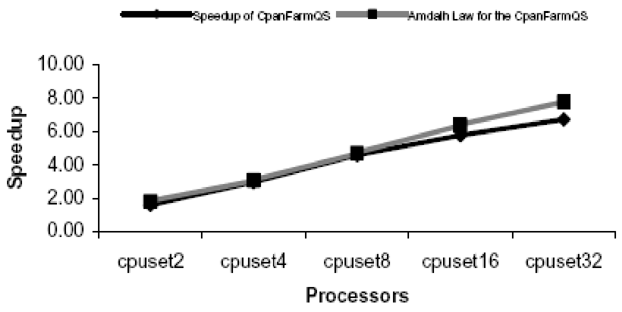
Finally, the programming technique is presented it Divide and Conquer as a CPAN, applicable to a wide range of problems that can be parallelizable within this scheme (see Figure. 7).



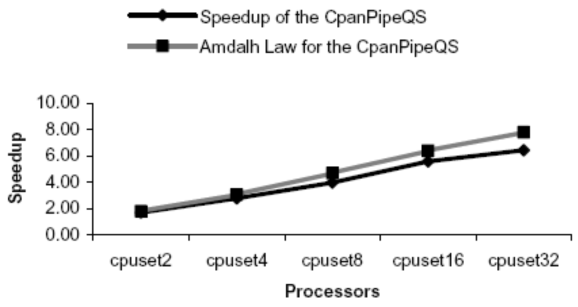
Figures 7: The Cpan of a TreeDV

RESULTS OBTAINED

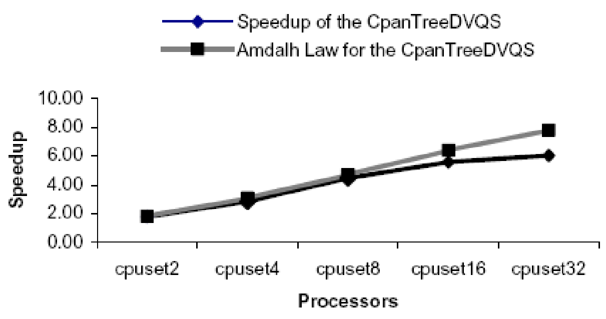
Some CPANs adapt better to the communication structure of a given algorithm than others, therefore yielding different speedups of the whole parallel application.
We carried out a Speedup analysis of the Farm, Pipe and TreeDV CPANs for several algorithms in an Origin 2000 Silicon Graphics Parallel System (with 64 processors) located at the European Center for Parallelism in Barcelona (Spain) this analysis is discussed below.
Assuming that we want to sort an array of data, some CPANs will adapt better to communication structure of a Quicksort algorithm than others. These different parallel implementations of the same sequential algorithm will therefore yield different speedups.
This analysis of speedup of the CPANs appears in Figures 8, 9 and 10. In all cases the implementation and test of the CPANs Farm, Pipe and TreeDV 50000 integer numbers were randomly generated to load each CPAN.



Figures 8: Scalability of the Speedup found for the CpanFarm in 2, 4, 8, 16 and 32 processors



Figures 9: Scalability of the Speedup found for the CpanPipe in 2, 4, 8, 16 and 32 processors



Figures 10: Scalability of the Speedup found for the CpanTreeDV in 2, 4, 8, 16 and 32 processors

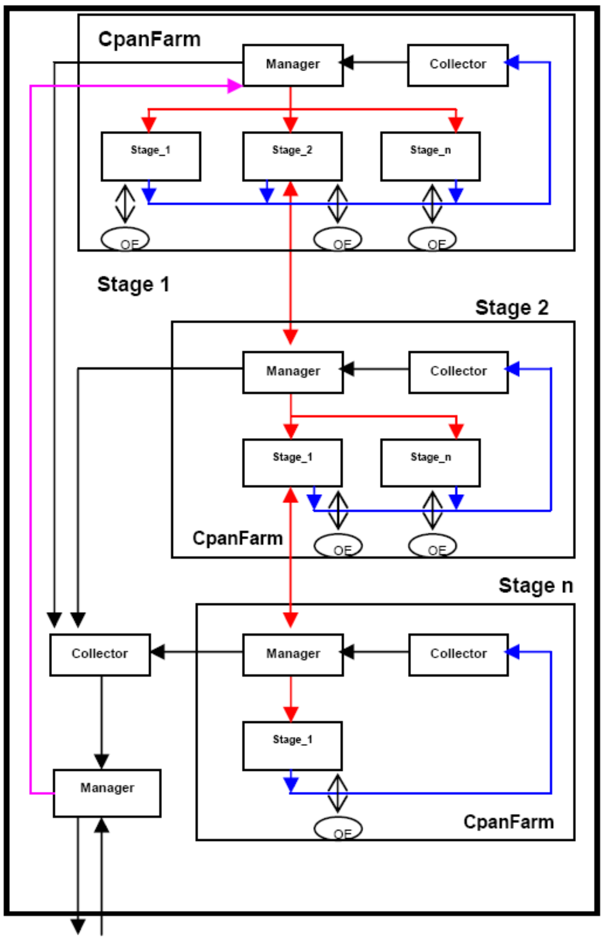
PARALLELIZATION OF BRANCH & BOUND TECHNIQUE

Branch-and-bound (BB) makes a partition of the solution space of a given optimization problem. The entire space is represented by the corresponding BB *expansion tree*, whose root is associated to the initially unsolved problem. The children nodes at each node represent the subspaces obtained by *branching,* i.e. subdividing, the solution space represented by the parent node. The leaves of the BB tree represent nodes that cannot be subdivided any further, thus providing a final value of the cost function associated to a possible solution of the problem. Three stages are performed during the execution of a program based on a BB algorithm,

1. ***Selection:*** A node belonging to the set of live nodes, i.e. those not pruned yet, is extracted. Node selection depends on the strategy search on the live node list, which was previously decided for its use in the algorithm.
2. ***Branch:*** the node selected in the previous step is subdivided in its children nodes by following a ramification scheme to form the expansion tree. Each child receives from its father node enough information to enable it to search a suboptimal solution.
3. ***Bound:*** Some of the nodes created in the previous stage are deleted, i.e. those whose partial cost, which is given by the cost function associated to this BB algorithm instance, is greater than the best minimum bound calculated up to that point.

The ramification is generally separated from the bounding of nodes on the expansion tree in parallel BB implementations. Each object node of the expansion tree must contain all the necessary information to be an active object.

The ramification and bounding are implemented using the CPAN Farm of the proposed library, so that the ramification and the distribution of work to the application processes are carried out by using *the Farm communication scheme*. The expansion tree, for a given instance of the BB algorithm, is obtained by iteratively subdividing the stage objects according to the farm pattern until a stage representing a leaf-node of the expansion tree is found, see Figure 11. On the other hand, the pruning is implicitly carried out within another *farm* construction by using a *totally connected scheme* between all the processes. The manager can therefore communicate a sub-optimal bound found by a process to the rest of branching processes to avoid unnecessary ramifications of sub-problems, i.e., those that do not lead to improving the best upper bound obtained up to that moment.



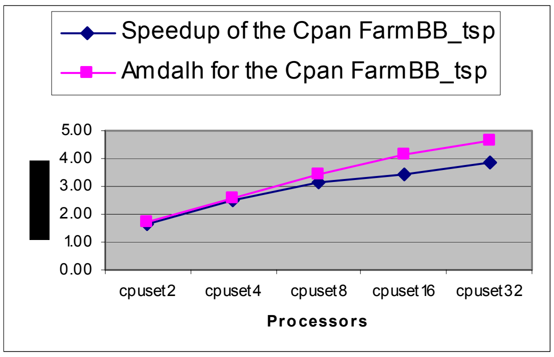
Figures 11: The Cpan Branch & Bound

Speedup of CpanBB in the solution of Travelling Salesman Problem (TSP)

With the model of the Cpan Branch & Bound we have been able to offer an optimal solution of a Travelling Salesman Problem (TSP) NP-Complete problem (GoodMan and Hedetniemi, 1977), with 50 cities and by using the first best search strategy driven by a least cost function associated to each live node (Capel and Palma, 1992). The results obtained yielded a deviation ranging from 2% (2 processors) to 16% (32 processors) with respect to the optimal ones, as predicted by the Amdalh law for this parallelized algorithm. See Table 1 and Figure 12.

Table 1: Statistics of Speedup the CpanBB

	Seq	CPU2	CPU4	CPU8	CPU16	CPU32
Run time	**35.42 Seg.**	**21.88 Seg.**	**14.21 Seg.**	**11.34 Seg.**	**10.27 Seg.**	**9.10 Seg.**
Time Usuary	25.80 Seg.	22.14 Seg.	20.16 Seg.	18.16 Seg.	21.67 Seg.	21.14 Seg.
Time System	1.30 Seg.	1.11 Seg.	1.01 Seg.	1.03 Seg.	1.02 Seg.	1.04 Seg.
Time CPU	**27.10 Seg.**	**23.25 Seg.**	**21.17 Seg.**	**19.19 Seg.**	**22.69 Seg.**	**22.18 Seg.**
Cycles	266948719	69877161	69073500	67663433	66942422	65992570
Instructions	202106112	73380571	73219450	72932731	72454022	72235919
CPI	**1.321**	**0.952**	**0.943**	**0.928**	**0.924**	**0.914**
Speed Up	**1.00**	**1.62**	**2.49**	**3.12**	**3.45**	**3.89**
Amdalh	**1.00**	**1.68**	**2.55**	**3.43**	**4.16**	**4.64**



Figures 12: Speedup of parallel CpanBB_tsp with N=50 cities in 2, 4, 8, 16 and 32 processors

This analysis of Speedup of the CPANS B&B was carried out also in the Origin 2000 Silicon Graphics Parallel System; see (Rossainz and Capel, 2005) for details.

CONCLUSIONS

An programming method has been presented, which is based on the High Level Parallel Compositions of Corradi , but updated and adapted to be used with C++ programming language and POSIX standard for thread programming. Several patterns of communication/interaction between processes have been implemented as generic, reusable library of CPANs, which can even be used by inexperienced parallel application programmers to obtain efficient code by only programming the sequential parts of their applications.

The CPANs Pipe, Farm and TreeDV conform a first version of a library of classes intended to be applied to the solution of several complex problems, as the parallelization of the B&B technique to offer an optimal solution of the TSP NP-Complete problem.

REFERENCES

Bacci, Danelutto, Pelagatti, Vaneschi, 1999, SklE: A Heterogeneous Environment for HPC Applications. Parallel Computing, Volume 25, pp. 1827-52.

Brassard, G., Bratley, P., 1997, Fundamentos de Algoritmia, Spain, Prentice-Hall.

Brinch Hansen, 1993, Model Programs for Computational Science: A programming methodology for multicomputers. Concurrency: Practice and Experience. Volume 5, Number 5, pp. 407-423.

Brinch Hansen, 1994, SuperPascal- a publication language for parallel scientific computing. Concurrency: Practice and Experience, Volume 6, Number 5, pp. 461-483.

Capel, M., Troya, J., M., 1994, An Object-Based Tool and Methodological Approach for Distributed Programming. Software Concepts and Tools, Volume 15, pp. 177-195.

Capel M.I., Palma A., “A Programming tool for Distributed Implementation of Branch-and-Bound Algorithms”. Parallel Computing and Transputer Applications. IOS Press/CIMNE. Barcelona 1992.

Capel, M., Rossainz, M., 2004. A parallel programming methodology based on high level parallel compositions. 14th International Conference on Electronics, Communications and Computers IEEE CS press. México, pp. 242-247.

Corradi, A., Leonardi, L., 1991, PO Constraints as tools to synchronize active objects. Journal Object Oriented Programming, Volume 10, pp. 42-53.

Corradi, A., Leonardo, L., Zambonelli, F., 1995, Experiences toward an Object-Oriented Approach to Structured Parallel Programming. Technical report no. DEIS-LIA-95-007. Italy.

Danelutto, M., Orlando, S., et al. Parallel Programming Models Based on Restricted Computation Structure Approach. Technical Report. Universitá de Pisa.

Darlington et al, 1993, Parallel Programming Using Skeleton Functions. PARLE’93, Munich.

GoodMan S.E., Hedetniemi S.T. “Introduction to the Design and Analysis of Algorithms. Mc Graw Hill Book Company. United States of America 1977. ISBN:0-07-023753-0.

Hartley, Stephen J., 1998, Concurrent Programming. The JAVA Programming Language. New York, Oxford University Press.

Roosta, Séller, 1999, Parallel Processing and Parallel Algorithms. Theory and Computation. Springer.

Rossainz, M., 1999, Una Metodología de Programación Paralela en Java. Technical Report. Universidad de Granada.

Rossainz, M., 2005, Una Metodología de Programación Basada en Composiciones Paralelas de Alto Nivel (CPANs), PhD dissertation, Universidad de Granada.

Rossainz M, Capel M. “Design and use of the CPAN Branch & Bound for the solution of the Travelling Salesman Problem (TSP)”. Proceedings of the 15th International Conference on Electronics, Communications and Computers, 2005, IEEE CS press. 0-7695-2283-1/05. pp. 262-267

Rossainz, M., Capel, M., 2005-2, An Approach to Structured Parallel Programming Based on a Composition of Parallel Objects. XVI Jornadas de Paralelismo, Granada, Spain, Thomson, pp. 495-502.