Skeletal Parallel Programming

F. Saez M. Printista F. Piccoli

LIDIC- UNSL

Ejército de los Andes 950, (5700) San Luis, Argentina. TE: 2652-420823

e-mail: {**bfsaez@unsl.edu.ar, mprinti@unsl.edu.ar, mpiccoli@unsl.edu.ar**}

Abstract

In the last time the high-performance programming community has worked to look for new templates or skeletons for several parallel programming paradigms. This new form of programming allows to programmer to reduce the time of development, since it saves time in the phase of design, testing and codification. We are concerned in some issues of skeletons that are fundamental to the definition of any skeletal parallel programming system. This paper present commentaries about these issues in the context of three types of $D\&C$ skeletons.

1 Programming with Skeletons

A skeleton is a generic tool that allows to define a concrete problem creating instances of the general method that implements it. From these skeletons programs can be derived models that illustrate as the paradigm solves specific problems [2].

The essence of this methodology of programming is that all program model has a parallel component that implements a pattern or paradigm (provided by the skeletons) and a specific sequential component of an application (in charge of the user). The clear separation of the issues of parallelism and the details of applications are essential for writing programs that are easy to understand.

The objective of these templates is to reduce difficulties of software development by means of parallel program's reusability. For example, we are familiar with concepts such as "divide and conquer", "pipeline" and "dynamic programming". When investigating a new problem we may try to formulate a solution in one of these well known styles. Since we already know how to implement the essential computational structure of each technique, it will only be necessary to introduce problem specific details to produce a new program. The user will only choose one of these methods of resolution without worried in aspects related to them, neither to the dependent code of target architecture.

We have defined several sketelons as a set of C functions on top of MPI library [3, 4, 6]. In this paper we will provide a simple example, which involves three different implementations from skeletons corresponding to the Divide and Conquer $(D\&C)$ paradigm. These skeletons or high level constructors extend MPI model, allowing the programmer codifies the sequential components of a specific $D\&C$ problem, whereas the communication and synchronization aspects between processors are solved by the skeleton and its support. The section 2 describes the $D\&C$ Skeletons.

This simplification allows us to disscus some issues of skeletons that are fundamental to the definition of any skeletal parallel programming system. We are concerned in examinig the ways

in which particular skeletons divide responsability betweeen implementation and the programmer. In order to that, it is useful to distinguish the external and internal view of one skeleton. The external view is caracterized by the degree of "programmability"; it is to say the posibility to write and modify parallel programs easily, and to prove their correctness against specifications. The internal view is represented by the way in which the constituent processes of skeleton ($activities$) may interact and the communications constrains between these activities ($interactions$) [1]. The sections 3 and 4 present commentaries about these issues in the context of $D\&C$ skeletons. Our conclusions are in section 5.

2 Divide and Conquer Skeletons

The divide and conquer approach, presented in Figure 1, finds the solution of a problem x by dividing x in subproblems x_0 and x_1 (divide) and applying recursively the same resolution scheme. This recursive procedure ends when the subproblems are small enough, in which case, another procedure (conquer) is used to solve the problem. The two subproblems (lines 7 and 8) in this typical structure can be done in parallel. This techniques is the basis of all parallel programming, in one form or another. Therefore, we can find several parallel formulations for this kind of computation.

```
1  procedure DC(x: Problem; r: Result);
2  begin
3    if trivial(x) then conquer(x, r)
4    else
5    begin
6      divide(x, x0, x1);
7      DC(x0,r0);
8      DC(x1,r1);
9      combine(r, r0, r1);
10   end;
11 end;
```

Figure 1: Scheme of algorithm D&C

We are developed three types of skeletons, called $ParD\&C$ [4], $D\&C-H$ [6] and $forall$ [3].

The $ParD\&C$ skeleton is the straight implementation of algorithm "divide and conquer". It maintains a recursive structure and it generates, while there are available processors, a binary tree of groups of processes whose leaves contains a single processor. In each recursion, a group of processors is divided in two sub-groups that solve the subproblems to be combined [5].

Other skeleton to solve algorithms $D\&C$, is a parallel clause derived of model OTMP [3]. Design and implementation of this clause were made in MPI and proven for several problems in different targets. To explain the work of $forall$ is necessary to indicate that in OTMP model the processors are organized in groups. In any moment, the state of the memory (input data and program) is the same for each processor in the group. When computation starts, all processors belong to same group, then when they reach $forall$ clause, the group is divided in sub-groups and each processor decides in terms of his identification what sub-group will belong. When clause ends, the sub-groups are joined in original group. The constructor definition takes the form:

```
forall (i,first, last, FunctionStetament[i],
        PosMemoryArea[i], SizeMemoryArea[i])
```

The programmer states that the different *FunctionStetament[i]* of the loop can be performed independently in parallel. The results of the execution of the i^{th} iteration are stored in the memory area pointed by *PosMemoryArea[i]*, and *SizeMemoryArea[i]* is the size in bytes. *first* and *last* indicate how is the wide of process's tree.

The third skeleton, Hypercube Divide and Conquer ($D\&C-H$) provides a structure with hyper-cubical communication between processes. Taking care of hiding the mapping and routing problems, this structure will be transparent to the programmers. The constructor definition takes the form:

```
void MPI_Hypercube (MPI_Comm Comm, int Tag,
          TMessage TMInput,
          TMessage TMOutput,
          TOperacionInicial PtrFtDistribution,
          TOperacionFinal PtrFtProcessing,
          unsigned int SetupSecuence)
```

The first two parameters are used to coordinate communications in MPI. Two phases must be performed in order to ensure a proper use of skeletons: initialization and sweep phases. During initialization phase, the user must configure the parameters in order to allow the sweeping takes place. After that, by each level of hypercube, the following task are executed: "Call to a Distribution Function", "Send and Receive of Messages" and "Call to a Processing Function". The implementation of $D\&C - H$ skeleton introduces two new data types. *THyperInfo* provides to programmer with information of the current state of execution, and *TMessage* implements a generic buffer used by programmer to send (*TMInput*) or receive (*TMOutput*) data from skeleton. *SetupSecuence* is a stream of bits that indicates the direction of the sweeping or sequence of levels to follow in the execution of hypercube.

The key of this skeleton are their distributing and processing functions. The call to distribution function ($PtrFtDistribution$) is made previous to the communication among processes, with the purpose to "divide" or prepare the data. Then, in case of receiving a message from the partner in the hypercube, the skeleton combines this message and the data of the local state, calling to processing function ($PtrFtProcessing$). Finally the output corresponds with the data stored in the local state, when finalizing the sweep. The processing function has the responsibility to combine the received message from the partner with its local data. If there is no message to receive, then the call to the processing function is ignored. Both functions depend on a specific problem to solve and they must be provided by the user. The programmers will have to consider the responsibilities that concern to each one. Nevertheless they are under the control of *MPI_Hypercube* primitive.

3 Programmability

We were introduced three skeletons or high level constructors to obtain solutions to Divide and Conquer problems. Each one $ParD\&C$, $forall$ and $D\&C - H$ provides different way to help programmer to develop parallel algorithms. Some commentaries are exposed in next sentences.

In the $ParD\&C$ skeleton are necessary to prepare the structures to send, besides explicitly to indicate the partner of a process in the hypercube, situation that is totally transparent for the user in the other two skeletons.

As an important abstraction, $D\&C - H$ and $forall$ skeleton allow to hide the stop point of the hypercube. However in instances of $ParD\&C$ is necessary to make a control on the availability of processors.

With respect to encapsulation, $D\&C - H$ is the only implementation that allows access to internal state of the execution through *Hyperinfo* structure; on the other hand, $ParD\&C$ use global variables, and the case of implementation with $forall$ does not allow any type of interactivity once reached the clause *forall*.

The separation of sequential and parallel aspects in $D\&C - H$ establishes an forced interface for distribution and processing function. In these functions the programmer is forced to know and to respect several guidelines. e.g. distribution function must to create the message to be sent, in another case the send operation is ignored. Viewed the different instances with this skeleton we noticed that no there clarity in the location of components to divide and combine an algorithm $D\&C$. Perhaps the most significant and complicated point at the time to think a solution with this constructor is when we thought about recursive solution of problem (natural in the paradigm $D\&C$), and we must solve it with a recursion in distribution function without take

advantage of hypercube's sweep that implements $D\&C - H$.

The $parD\&C$ constructor although with less problems cannot completely isolate the parallel component of paradigm $D\&C$, since the use of global variables is necessary for implement certain control in the constructor. On the other hand, the $forall$ constructor reaches a well defined interface that separate all the parallel aspects of problem. However these skeletons clarify the zone to implement the sequential solutions.

The hiding of parallel aspects in $forall$ is total. With $D\&C-H$ constructor was necessary to interact in distribution and processing functions with details like e.g. which level of the sweep hypercube is the execution?, What processor is executing now?, etc. In the case of $parD\&C$ although hides most of the parallel aspects, leaves the synchronization of the nodes in the hypercube to programmer.

$ParD\&C$ skeleton models in an abstract way the concept divide and conquer, it is clearest to understand and it provides the best performance in two instantiated applications. A clear point that must be improve is hiding of information respect to partnership between processors.

Is easily understood that $forall$ skeleton arises with the necessity to satisfy a greater set of requirements for $D\&C$ algorithms (e.g. load balance). This amplitude of cases entails to gain a little of complexity, by to add entrance parameters to cover the different cases. However this complexity does not influence in the clear separation between the parallel and sequential aspects, neither in the run times reached, very similar to reached for $ParD\&C$. The third skeleton analyzed in this work, $D\&C - H$, it was very criticized in most of the analyzed properties. However we found in it desirable characteristic like the possibility of to provide structures (e.g. *Hyperinfo* and *TMessage*) that allows store information about execution of algorithm. This is a very important characteristic at the time of debug a parallel application.

4 Interaction Mode

The purpose of any skeleton is to abstract a pattern of *activities* (process) and their *interactions* (messages for distributed memory MIMD). It is useful to further distinguish internal and external interactions. *Internal interactions* occur between two or more activities, whereas *external interactions* occur between activities and the enclosing context. For example, in a pipeline, an interaction between two stages is internal, but an interaction between the initial stage and the source of pipeline inputs is external. Abstraction of skeletons constrains the way in which its constituent activities may interact. There are two types of constraints in the interaction between activities, *spatial constraints* determine the activities with which it may interact and the directions these interactions may take (partner activities), and *temporal constraints* determine the allowable orderings of interactions between partners. For example, in a pipeline, a spatial constraints determine the partner stages (activities) of any stage in the pipeline are the preceding and succeding stages. A temporal restriction may require that a stage interacts first with its predecessor then its successor.

Recursive implementation of the $ParD\&C$ generates in run time a binary tree of groups or sets of activities. The root is a group formed by all the activities (with binary addresses from 0 to $2^n - 1$ that are adjusted to hypercube topology), and the leaves nodes with groups of a single activity. In each recursive call, the group of activities is divided in two, the first subgroup has activities with first bit of address in 0, the second subgroup has activities with first bit of address in 1. The skeleton internal spatial constraints are imposed by hypercube topology, therefore each activity in the first subgroup makes a partnership with an activity of the second group in a way butterfly. For temporal constraints, partners activities communicates in synchronous way, sending half results to partner and receiving of it other half results, then skeleton combines results in a

single solution. It is necessary to highlight that all the activities have external interactions with main program, they receiving all input data from main program, and returning a complete result to main program (Class of Replied-Replied problems (RR) [6]).

The structure of $D\&C - H$ skeleton consists of an initialization stage and a level block stage. The Initialization stage is in charge of external interactions with main program. Each one of activities receive a portion of input data, and generate a portion of the solution (Class of Distributed-Distributed problems (DD) [6]). The level block stage is in charge of to call to distribution function, the interaction between activities and to call to processing function. The interaction between activities is organized in the same way that previous skeleton, activities have a temporal constraints imposed by hypercube. The synchronization of processes is sincrnomous between partners too.

The *forall* parallel clause is derived from OTMP model. This model extends the sequential paradigm with a constructor of parallel iteration and global communication. This clause, like $ParD\&C$, behaves with an Replied-Replied external interaction. In this skeleton the number of independent activities is determined by the number of iterations (M iterations), forming this way a hypercube of orden M of groups of processes. The number of processes assigned to each group can be balanced if the problem demands its. The spatial constraints are imposed by partnership relation among processors give place to comunication patterns among processors that are topologically similar to a hypercube.

5 Conclusion

We have analysed the issues of "Programmability" and "Interaction mode" as aspects of design of skeletal parallel programming frameworks.

We were examined, by means of several $D\&C$ examples, that these characteritics are not always easy to abord. However, on explicit specification of them will clarify the use of a particular skeleton.

Acknowledgments

We wish to thank the Universidad Nacional de San Luis, the ANPCYT and the CONICET from which we receive continuous support.

References

[1] A. Benoit and M. Cole. Two fundamental concepts in skeletal parallel programming. *V.S. Sunderam et al. (Editors), ICCS 2005, LNCS 3515*, pages 764–771, 2005.

[2] Murray Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. 1989.

[3] F. Piccoli, M. Printista, and C. Rodriguez Len. *Dynamic Hypercubic Parallel Computations*. Proceeding (466) Parallel and Distributed Computing and Systems, 2005.

[4] F.D. Saez. *Paradigmas de Programacin Paralela*. Degree Thesis submited to UNSL, 2004.

[5] F.D. Saez, R. Gallard, and M. Printista. *Paradigms of Parallel Programming*. Workshop de Investigadores en Ciencias de la Computacin (WICC 2003), Tandil, Argentina, May 2003.

[6] J.G. Zanabria. *Hypercubic Comunications in MPI*. Degree Thesis submited to UNSL, 2005.