Progressive Generation of Parallel Solutions for Formally Specified Problems *

M. Buffo, E. Urland, J. Rolim

Centre Universitaire d'Informatique
Université de Genève
24, rue du Général Dufour
1211 Genève 4, Suisse
email: {buffo,urland,rolim}@cui.unige.ch

D. Buchs

LGL - DI
Swiss Federal Institute of Technology
1015 Lausanne
Switzerland
email: buchs@di.epfl.ch

Abstract

In this paper we show a progressive way for managing abstract algebraic specifications in order to obtain efficient parallel or distributed algorithms. Our approach is based on TSPP technique (Transformations de Spécifications de problèmes orientées vers le Parallélisme et le Probabilisme). We claim that the process of transformation of specifications can be done, in many cases, automatically in direction from a high level of abstraction to concrete one using a sequence of derivation steps. The resulting specification, obtained by such a transformation process, is then well suited for execution on target parallel machine. Further we shortly describe the SANDS system (Structured Algebraic Nets Development System [3]) and the TTool - transformation tool package which was developed for performing transformations on specifications. To illustrate the TSPP technique we present here an example developed in SANDS system environment by application of TTool. Finally we discuss the obtained speedup of execution such a derived specification on parallel machine with various number of processors.

1 Introduction

From the past few years the effort in parallel software engineering is concentrating on development of correct and efficient software packages for different parallel architectures. However, up today there is no common methodology, which can guarantee the correctness end efficiency of designed algorithms. The classical approach to this problem is usually based either on the experience of programmer, either on some ad-hoc method used only for this particular case, especially concerning the implementation on target parallel architecture. The problem becomes much more crucial when a large software product is required. From the theoretical point of view, some formal methods, based on predicate logic are now under development [1] [9]. These methods can give an excellent results for some special family of objects, usually graphs which are in some sense recursively decomposable, but they are not very useful for general purposes.

As an alternative way for the design and the development of parallel software, based on exact theoretical background, we propose a technique using transformation on specifications. This technique - used in the past with some difficulties [6] [8] - is arising from the area of rewriting systems but in this case suited into the context of parallelism; by transformation we understand an equivalence relation between two specifications. The process of handling (deriving) specifications in direction from abstract to more concrete one is called transformation process and the emphasis is given to such a transformation steps, called transformation methods, which introduces parallelism and leads closer to implementation level. We claim that this process can be done automatically and to support our conjecture the TTool - transformation tool package - was developed. The TTool performs transformations on specifications under the bisimulation condition of these specifications. This guarantee the correctness of the resulting specification on implementation level. In order to obtain an efficient implementation of the initial specification, the application of certain transformation step is evaluated partly to the target parallel architecture and partly to other possi-

*This research was supported by FNRS, program SPP IF 5003-034295.

ble transformation methods; the most suitable transformation step is then chosen and performed. The last step in the process of deriving parallel program is to apply a suitable compiler for generating executable code for the target machine.

TSPP is an open technique in the sense of further improvements and developments. New transformation methods can be simply added into the transformation corpus. Moreover, this technique is independent of the chosen specification language and can be considered as a general framework for deriving parallel or distributed solutions from formally specified problems. Nevertheless, in order to illustrate the practical capabilities of the TSPP technique, we have included the specification language CO-OPN (Concurrent Object - Oriented Petri Nets [2]) into the TTool. This language takes advantage of easily managing algebraic specifications as well as good possibilities of expressing parallelism. The object - oriented features naturally supports distributed description of specifications.

The paper starts with a description of our transformation notion in section 2. Then the section 3 introduces briefly the compilation phase. Section 4 contains a description of the CASE tool we develop for the management of software development from the first abstract specification to the implementation, while in section 5 we illustrate our technique with a simple and concrete example.

2 Transformation on algebraic specifications

By *specification* we understand an exact formal description of a problem. If the formal description exploits algebraic data types, then the specification is called *algebraic*. The CO-OPN language is an example of a specification language, which merge algebraic data types and Petri nets into the more powerful algebraic nets model. Let S be a set of specifications and '$\sim$' be an equivalence relation defined on SxS. Let $T = S\text{x}S$ and $T_c \subseteq T$, such that

$$(x, y) \in T_c \iff x \sim y, \text{ where } x, y \in S.$$

The set T is the *set of transformations* and an element $t \in T$ is called *transformation*. The transformation $t \in T$ is said to be *correct* iff $t \in T_c$. Performing a transformation step passing from specification $x \in S$ to specification $y \in S$ is implicated by transformation $(x, y) \in T$.

Transformation rules are a generic way to transform a specification. The semantic is based on rewriting systems. A transformation rule R induce a family of transformations $T_R \subseteq T$. We say that a transformation rule R is *correct* iff

$$\forall t \in T_R, t \in T_R \wedge t \in T_c.$$

In other words, if the rule was once prove to be correct, by any application of it we obtain a specification which is in the relation '$\sim$' with the previous one.

Using Transformation methods we can efficiently describe a sequence of transformation steps. In this case the semantic of Petri nets is used. From this point of view applying a transformation method can be considered as a firing of certain transitions in a Petri net model. The advantage of transformation method implies from grouping simple transformation steps into the more complex one, following a common methodology [4].

Assume that we have an initial abstract description of a problem. Usually the formal description on this level does not contain any algorithmic construction. In order to obtain a parallel algorithm we have to perform a sequence of transformation steps, refining the specification from an abstract level to a more concrete one, as it is shown on Fig. 1. Each transformation step consist of simple rewriting of a part of specification. As we have mentioned before, the transformation steps are performed under the condition of some equivalence relation (in our case it is bisimulation) between specifications.

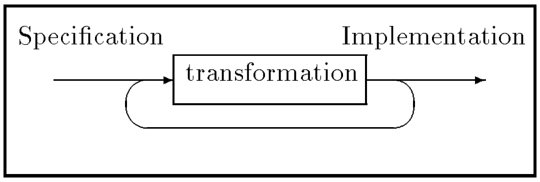

Fig. 1: Transformation step scheme

According to the initial specification and the target parallel machine architecture, the most suitable skeleton of a family of solution is derived. The refinement process is done by further application of transformation steps, more and more restricting the set of possible solutions. After a finite number of transformation steps we will end with required specification, describing the problem very concretely, by some algorithmical scheme including parallel operations. Actually, such a specification is a program written in specification language. This level of transformation process is called implementation level. At this point we aplly a suitable compiler to generate executable code for target parallel machine.

3 Compilation

After the transformation phase, managed by the TSPP technique, we must compile the resulting specification in order to obtain real executable code. Using our approach, this compilation phase can be handled using a simple and traditional compiler. This is possible because the output of the transformation phase is supposed to be a concrete parallel specification.

By concrete specification, we mean a specification in which algorithmic aspects and operations are defined. In other words, a concrete specification is a program, described using a specification language. Compiling such programs does not invoke any special compilation techniques.

By parallel specification, we understand specification in which parallel aspects are already explicitly described. This is the case, as the TSPP was actually developed to achieve this goal. The use of a parallelizing compiler is therefore unnecessary.

Indeed we will use a simple parallel compiler, without any parallelizing phase, and without any concretization phase. Notwithstanding the translation among formalisms, the main task of such a compiler will consist in managing the distribution of the created processes and the communications between them.

4 The SANDS system

The TTool - Transformation Tool package - was developed for performing transformations on specifications, following the TSPP technique. Likewise, a compiler for MIMD machines - in this case, a 32-transputers hypercube - was written. As both of them are a part of the SANDS system, we introduce briefly the whole system.

The SANDS system is a complete CASE toolset for developing new applications using the CO-OPN language [3]. This toolset, based on graphical and convenient interactions with the user, allows the specification of problems and the development of the solutions according to basic software engineering principles. Besides the TTool and the compiler, it is composed of a convenient graphical editor for algebraic Petri nets, of a monitor/simulator and of a general purpose theorem prover.

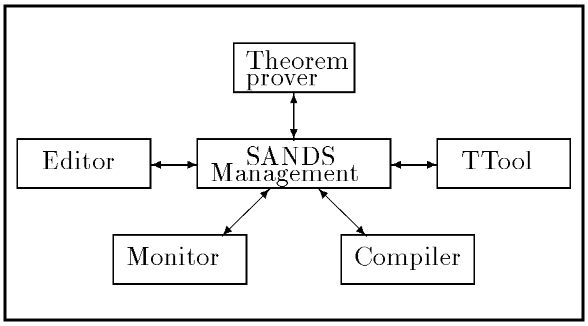

Fig. 2: SANDS system

The graphical editor is in fact the center of the SANDS system. It is a user-friendly tool, which permits to create or modify part of the current specification. This complex editor is splited into three sub-editors, the algebraic abstract data type editor (for creating and manipulating such data types), the objects (Petri nets) editor (for the edition of the inner part of the objects) and the system editor (for the definition of the synchronizations of the hierarchical nets system).

The monitor/simulator tool permits to prototype and to graphically simulate part of the current specification. Is is an essential tool for the user, as it permit to understand what one really specifies, analyzing the modelization and the simulation of the behavior of the edited specification. Using the simulator, the user can examine the abstract data types comportment, examine the behavior of some objects in the system or simulate part of the current system.

The theorem prover is a tool which permit to check or prove some properties on the current specification. It is based on the "Temporal Logic" formalism, adapted to the CO-OPN specifications. It is used by the user to interactively prove some properties during the edition of the specification, or by other tools as an internal properties prover.

The parallel compiler is used to obtain real code from specifications. More precisely, it traduces CO-OPN specification into C code, which can then be executed on parallel machines. For the moment, this compiler is suited for MIMD machine; more precisely the output code is targeted for transputer networks. Moreover, a graphical performance analyzer is offered.

The transformation tool is used to transform the current specification, according to the TSPP technique. The transformation tool package consists in the tool itself, which performs the transformation, and in the transformations library, which is used by the tool and which can be dynamically modified. This tool is a user-friendly xview-based application. The refinement is performed step by step, according to the transformation technique. At each step, the user can decide to stop the transformation process, to backtrack (if the developed specification is not satisfactory), to evaluate the adaptation of the current specification to the execution platform, or decide to perform a new transformation on the current specification. The transformation itself is very simple to manage; the user select the correct transformation rule or a transformation method to apply, TTool displays then the possible transformation steps derived from the selected rule or method, the user choose one of these steps, and TTool apply it. The library of transformations is composed of several textual modules, written in a CO-OPN-like syntax. These modules are splited into two classes, according to the TSPP techniques. The first class is composed of the rules modules, which contains transformation rules and are similar to the algebraic modules in CO-OPN, because of the similarity between the rewriting rules and the algebraic operations.The second class is composed of the method modules, each of them containing a transformation method. They are similar to the object modules in CO-OPN, because of the fact that all are essentially Petri net descriptions.

5 Example

As an illustration of our methodology, we describe briefly how to reach a parallel efficient solution for the "byte parity" problem, using a virtual session of the SANDS system. By "byte parity", we mean the function defined as "parity(n) = parity of the number of bits set in the binary representation of n". Another example, illustrating the TSPP technique with more details, can be found in [5].

We start using the graphical editor, and create the first formal description of the problem, composed of two modules. The first one describes the data structure, in our case the abstract description of the function "parity". The second module describes the root process, which invoke the above function.

We use then the monitor to confirm the correctness of the just created modules. We start with the evaluation of several algebraic terms, to verify the exactitude of the "parity" function, after which we begin a simulation session of the whole system. This monitor session leads us to the certitude that our system is correct with respect to the first informal description of the problem to solve.

Now, we must refine our specification and reach a concrete algorithmic description of the solution. This goal is achieved using the TTool package, which introduces semi-automatically parallel, distributed and possibly probabilistic aspects into the solution, and which guarantee the correctness of the result with respect to the first formal description. This TTool session itself can be splited into three phase. During the first phase, we analyse the abstract problem and find convenient and performant algorithms. In our case, we find that a "divide to conquer" technique can be applied (even for such a simple problem like our "parity" function; in fact, we do not split arrays or list as in the usual case, but we split the integer argument itself into two suitable parts), and we derive the proper algorithm. In the second phase, we concretize this algorithm, creating the processes we need according to the hardware of the execution platform. In our example, we create a computation tree with 15 processes (among them 8 processes are terminals, the leaves of the tree). As a result, we obtain a system with 16 processes, the root one and the members of the computation tree. During the last phase, we optimize separately each process according to the real processors in which they will take place. In our case, assuming we want to execute the result in a transputer hypercube, we perform generic low-level optimizations for MIMD machines. Thus we have now a concrete parallel specification, ready to be compiled and executed.

We use then the compiler to reach concrete executable code. For our example, we ask it to compile for 8 processors (the leaves of the computation tree) of a transputer hypercube, with monitoring of the execution and suitable input/output.

After the execution of the program, we can finally monitor and analyse the transputer session, using the Paragraph software [7]. This software provides several displays to analyse the behavior of our program, such as a clock, a Gantt chart, a space-time communication display, a network display, a processor utilization summary, and so on.

The following figure shows the speed-up (in function of the number of processors, with respect to the original unstructured problem) that we achieve concretely on our 32-transputers hypercube for our parity example.

#processors	2	4	8	16
speed-up	1.3	2.6	6.1	11

Fig. 3: Achieved speed-up

This result seems to be promising, because of the fact that for such particular kind of simple example, where the computation time is much less than the communication time, the multiplication of the processors leads in fact to the multiplication of useless communications. Nevertheless, we achieve a nearly-linear speed-up.

6 Conclusion

We have shortly described the TSPP technique used for progressive generation of parallel algorithms from formally specified problems. The technique is based on the principles of rewriting systems oriented to the parallel context. We showed the advantage of such a approach, in order to obtain correct and efficient parallel implementation from the initial abstract specification. Moreover, this process can be done in many cases automatically.

The TTool package was developed for the refinement of specifications from abstract to more concrete one. Together with the suitable compiler they are inherent parts of the SANDS system. The TSPP technique we proposed is a general framework in the sense that a variety of specification languages, formal methods and algorithmic constructions can be included. From a software engineering point of view it is an alternative way to design parallel software packages. As TSPP is an open technique, the research is concentrated on development of new transformation rules and methods, covering a vide spectrum of algorithmic principles, including probabilistic and randomized approaches. Easy ways of description of real parallel architectures as well as theoretical models are under study. Examples from industrial applications are used to test the capabilities and performances of TSPP technique and the whole SANDS system.

References

[1] Borie, R.B. - Parker, R.G. and Towey, C.A.: Automatic Generation of Linear Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families, *Technical Report, Department of Computer Science, University of Alabama* , 1993.

[2] Buchs, D. and Guelfi N.: A Concurrent Object - Oriented Petri Nets approach for system specification, in *12th International Conference on Application and Theory of Petri Nets* , Aahrus Denmark, (M. Silva ed.), p. 432-454, June 1991.

[3] Buchs, D. - Flumet, J. and Racloz, P.: SANDS: Structured Algebraic Nets Development System, in *14th International Conference on Application and Theory of Petri Nets, Tool presentation abstracts*, Chicago, (Buy Ugo ed.), june 1993.

[4] Buffo, M.: Collection de Transformations, *Technical Report no.79, CUI, Université de Genève* , September 1993.

[5] Buffo, M. - Buchs, D. and Rolim, J.: Developing parallel programs based on rewriting techniques, in *proceedings of the IEEE/USP Workshop on High Performance Computing*, São Paulo Brazil, (J. Zuffo ed.), march 1994.

[6] CIP language group: The Munich Project CIP, volume 2 The program Transformation System CIP-S, *Lecture Note in Computer Science 292, Springler-Verlag*, 1987.

[7] Heath, M. T. and Etheridge, J. A.: Visualizing Performance of Parallel Programs, *Technical Report ORNL/Tm-11813, Oak Ridge National Laboratory*, 1991.

[8] Krieg-Brückner: PROgram development by SPEcification TRAnsformation, in *Technique et Science Informatiques vol. 9 no 2*, 1990.

[9] Smith, D.: KIDS: A Semantic Program Development System, in *IEEE Transactions on Software Engineering, vol. 16 no 9*, 1990.