Cabaret: A Fortran Case Base Reasoning System for Porting Code to High Performance Systems

F. Bodin* ,Y. Mével*†, R. Quiniou*
†SIMULOG - 1 rue James Joule - 78286 Guyancourt - France
*IRISA-INRIA - Campus de Beaulieu - 35042 Rennes Cedex - France

1 Introduction

High performance architectures are now widely available. They are characterized by the use of fast microprocessors and they are moderately parallel with a shared memory (for instance the SGI Power Challenge). Reaching good performance on these machines requires to make good usage of the parallelism and of cache memories. Porting Fortran codes to these machines requires generally three phases : code cleaning [4], performance measurements and finally code transformations. The last two phases are iteratively repeated to achieve the desired performance. The code cleaning phase must not be seen as a simple phase. In many cases, this may consist in converting a program written in old style Fortran 77 to Fortran 90.

Part of porting the code can be done automatically, but there still remains a large amount to be done by an expert "hand". The Cabaret project aims at providing a set of integrated tools for speeding up the process of porting applications to high performance architecture using a novel approach based on case base reasoning techniques [2]. The main idea is to be able to recover easily an existing solution to a current problem (or close to). In other words we want to capitalize experience. One advantage of the case base reasoning techniques is that it does not depend on a formalization of the problem. This is essential since porting code to high performance architectures can hardly be reduced to a set of rules.

Some works [5, 1, 3] have been proposed to detect patterns in code that can be replaced by calls to library functions. Our approach radically differs because our system allows the user to update, automatically, the database with its own solutions and patterns.

Because program transformations and performance measurements are very time consuming we complement the case base reasoning techniques with a set of predefined program transformations and a set of automatic instrumentation capabilities. Because we want to validate the idea on large programs the implementation is based on, a professional Fortran parser, the Foresys system [6]. The organization of the system is shown in Figure 1. In the following we present an overview of the system.

2 Case-Base Reasoning for Code Porting

The main assumption the system is built on it that software engineers tend to repeat themselves. When one has a solution to port a program to a given machine, it can be reused and shared by many others. This solution can be considered as a "case" and it is important that the expertise can be shared or retrieved as fast as possible. To give an example, when implementing in parallel a

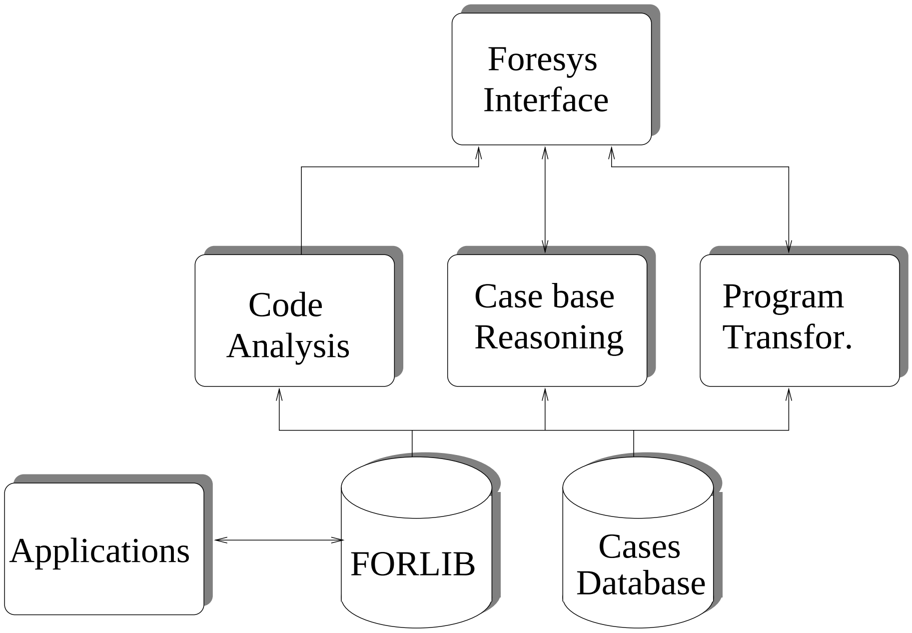

Figure 1: Organization of the system

Jacobi loop using message passing, the structure of the loop is usually always the same and a block parallelization is usually the one giving the best result. If such a loop is to be parallelized, the Cabaret system would be able to propose the user the "jacobi case". Using cut-and-past or the comments from the solution the user can save a lot of time, for instance by reusing (and eventually adapting) the same call to the message passing library.

There is four basic operations proposed by the case base reasoning: retrieve, reuse, revise, retain. The **retrieve** operation consists in finding a case similar to the current one. This operation is based on a similarity computation between the current case and the one in the database. The similarity computation is not always based on a syntax comparison, but rather on a set of abstractions. For instance, if we are looking to data locality optimisation, then the program is abstracted by it array access patterns, if we are looking for parallelism, then data dependence may be the best abstraction to work on. A retrieved case typically contains, the code original code fragment, the code fragment corresponding to the solution, a pattern corresponding to an abstraction of the original code fragment, a context (which machine, etc.), the performance obtained and, very important, the "manual" of the case. The **reuse** operation can be done automatically if solution consists in a set of predefined loop transformations, or it can be done manually via cut-and-past operation. In most case it is necessary to **revise** the proposed solution to answer to the the current case. If the user whish to keep the new solution it can be stored in the database, as a new case, using the **retain** operation. We are currently developping basic "cases" to experiment with this new concept for porting code.

References

[1] J.Ning A. Engberts, W. Kozaczynski. Program concept recognition and transformation. *IEEE Transactions on Software Engineering*, 18(12):1065 – 1075, 1992.

[2] A. Aamodt and E. Plaza. Case-based reasoning : Foundational issues, methodological variations and system approaches. *AICOM*, 7(1), March 1994.

[3] Santanu Paul Atul Prakash. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463 – 475,, 1994.

[4] Bernard Dion. Ré-ingénierie et parallèlisation d'application scientifiques avec foresys/partita. *Calculateurs Parallèles*, 6(24), 1994.

[5] J.R. Hagemeister S. Bhansali. A pattern-matching approach for reusing software libraries in parallel systems. In *First International Workshop on Knowledge-Based ystems for the (re) Use of Program Libraries,INRIA,*, november 1995.

[6] SIMULOG. *FORESYS, Fortran Engineering system, Manuel de référence*, Avril 1994.