Parallel and Distributed Programming with ParMod–C

Andreas Weininger
Thomas Schnekenburger
Michael Friedrich
Institut für Informatik
Technische Universität München*

Abstract

ParMod is a set of language independent constructs for parallel and distributed programming. These constructs can be added to different sequential procedural programming languages. This paper describes an extension of the ParMod constructs. They are embedded in conventional C leading to the parallel and distributed programming language ParMod–C.

In addition, a new storage class concept is introduced, which is needed to improve the efficiency of ParMod–C implementations.

Several examples will demonstrate the usage of the ParMod–C language. We will also show how ParMod–C can be implemented on different hardware architectures.

1 Introduction

There exist a lot of approaches for the programming of parallel and distributed systems [BST89], [GM88]. Most parallel computers are programmed either in a sequential language like C or Fortran with the help of a special system specific synchronization and communication library [Sun90], or in a parallel language which is only efficient on a small subset of parallel and distributed architectures.

ParMod [Eic86], [Eic87] is a language independent concept for parallel and distributed programming which can be added to different sequential procedural programming languages. This has been done for instance for Pascal [Eic86] and will be done for Modula–2 [Tuf90], [Got90].

In this paper we will present an extension of the original ParMod incorporated in C. The resulting language is called ParMod–C. The goals of of ParMod–C are

- to offer high level constructs for parallel and distributed programming which are efficiently implementable.
- to allow the user to easily specify a high degree of coarse–grain parallelism.
- to allow the reuse of sequential program parts and to minimize the effort for parallelizing existing sequential programs by preserving the spirit of C.

[Gre89] describes the first version of ParMod–C. The experiences made with this version yielded to the development of the ParMod–C versions described in [SW90] and this paper. The reasons for the extensions are

- to allow a more efficient implementa-
tion which forced us to introduce storage classes.

- improvements in the amount of communication which are one reason for new parameter passing mechanisms. Some examples are the introduction of out parameters in global procedure calls, which save one transfer of the parameter compared to inout parameters, and the introduction of dynamic length arrays as parameters (cf. section 2.3).

- improvements for the programmer's ease of use through a more obvious syntax

- to support data parallelization which resulted in replicated modules.

In the following section we will give an overview of the parallel language constructs in ParMod-C and how they are integrated in sequential C. The next section will present some examples to clarify the syntax and to show the usefulness and expressiveness of the constructs introduced in section two. Finally, we will show some current implementations and give a survey of future ParMod developments.

2 ParMod-C

A ParMod-C program is a finite non-empty set of several ParMod-C modules which are executed in parallel. A new feature in this version of ParMod-C is the possibility that each module may be replicated, i.e. there exist \( n \geq 1 \) instances from each module. The number of instances of each module is determined at program start from a special configuration file.

In the following text we will briefly write module instead of module instance if there is no danger of confusion. We define the parallelism among modules as global parallelism.

![Figure 1: Structure of a ParMod-C program](image)

A module contains local and global parallel procedures. The local parallel procedures, in the following named parallel procedures, can only be invoked within the module, in which they are defined.

The global parallel procedures, in the following named global procedures can be invoked from any module of the program. The mechanism of a global procedure call is not a synchronous remote procedure call, but an asynchronous remote service invocation\(^1\): The call of a global procedure always generates an activity in the module where the procedure is defined. This activity, called task, is a new incarnation of the procedure. The caller of a global procedure immediately proceeds. Calls of parallel procedures are asynchronous too. The parallelism within a module is called local parallelism.

Of course, in ParMod-C also exist usual C functions. A task which calls a function simply executes its code as in C.

Figure 1 shows an example for the static structure of a ParMod-C program. The \( p_i \) and \( q_i \) are the procedures defined in the modules \( M \) and \( N \).

\(^1\) a similar mechanism is used in [AOC+86]