

Towards Petri nets application in parallel program debugging

D. I. Kharitonov¹, G. V. Tarasov²

*1. Institute for Automation and Control Processes, 5 Radio st., Vladivostok, Russia, 690041
(demiurg@iacp.dvo.ru)*

*2. Institute for Automation and Control Processes, 5 Radio st., Vladivostok, Russia, 690041
(george@dvo.ru)*

Abstract

In spite of variety of existing tools and approaches, debugging remains the most difficult and laborious stage of parallel software development process. This article investigates application of Petri net theory in parallel programs debugging process. It is shown, that in terms of Petri nets can be represented such valuable aspects of debugging process as visual model of parallel program, debugging language and actions. Moreover in terms of Petri nets we can reason about correctness of parallel program modification with debug information.

Keyword: *parallel programming, parallel program debugging, debugging techniques, Petri nets*

1. Introduction

Parallel programming gives us an opportunity for solving those problems, that couldn't be solved in sequential programming due to resource restrictions on memory volume or on solving time. However, when instead of sequential programming we use parallel one, we come across on a set of problems, which was more or less successfully solved in sequential programming. In this set we can distinguish such problems as scalability of parallel programs, reusing of source text, and problems of correctness and debugging of parallel programs.

The problem of parallel programs debugging have an especial actuality, because time consumption of parallel programs debugging at the present time often exceed such expenses for initial parallel program source writing. The base of this problem is non-deterministic behaviour of parallel program execution, which makes cyclic repetition of erroneous situation very difficult and even more difficult investigation of error reason.

In the basis of modern debugging tools, most powerful of which is TotalView, lays down an idea of source program modification with addition of debugging information and code. This relatively small additional part of program deliver to debugging system necessary information for monitoring program state and realizing usual set of service functions, by means of which programmer can control execution of program and analyze its current state. Those service functions are stop and resume sequential process execution, setup breakpoints and examination of process memory and stack. Historically, all above-mentioned functions control execution of one single process, and could be expanded on user defined process group. In this sense parallel program debug tool differs from sequential programs debuggers by extended functionality of process group definition for debugging actions. This functionality directly linked with description language of parallel program state. At the present time most of debuggers for this purpose use the same language that was used for initial coding. Therefore using OpenMP standard we can use "teams" like groups of processes, and when using MPI standard - communicators define their own groups of processes. But using OpenMP standard programmers are free from routine and sophisticated work on interprocess communications, and this result in quite easy debugging stage of programming comparable with than one in sequential programming. On the contrary, when using MPI standard, parallel program correctness is directly depend on correctness of programmer written interprocess communication procedures, and increased complexity of parallel program debugging does not compensate by some simplification of process grouping with communicator help.

In this paper we propose new approach to debugging of parallel MPI-programs with help of Petri nets - formal language of parallel and distributed system specification. Using Petri nets we will

gain natural language for parallel program state specification and new powerful way for implicit definition of sequential process subsets by means of Petri nets markings and steps for the sake of parallel program debugging (Work is carried out at financial support of Presidium of the Russian Academy of Sciences – program №17 and the Russian Foundation of Basic Research – grant №04-07-00287).

2. Petri net model of communicating sequential processes

Petri nets and the most of extensions of this language [2, 3] are usually grounded on the algebra-plural approach to creation of descriptions. The descriptions received at this approach, strongly differ from usual programming languages text representation. In addition standard for the most of programming languages possibilities of usage of data type definition with a lot of values (for example, real, float, integer and other) complicate application of Petri nets for the exact programs specification.

For simplification of the program description authors have developed the extension of the Petri nets, called Petri nets for programming (PNP). Its purpose is to model structure of parallel program with a lot of values set, described in terms of imperative programming language. There are two kinds of model in PNP: plain Petri net and hierarchical Petri net.

Plain Petri net are specified by its own structure and a set of special inscriptions, attached to each element of the net. Structure of plain Petri net, as well as in common Petri nets, consists of a set of places, transitions and arcs. Places are used to specify a state of model and are represented by circles on pictures. Inscriptions described tokens can be attached to any place. Transitions are used to specify events possible in model and represented by rectangles on pictures. Excitation predicate can be attached to transition to specify whether transition can be fired or not. Transition can be fired only if value of predicate equals true. Inscriptions called substitutions are attached to arcs ingoing to transition. This type of inscription gives ingoing tokens names that are used in predicates and expressions. Expression is a type of inscription that attached to arc outgoing from transition. Based on ingoing tokens expression calculate new values of tokens. In addition to predicate inscriptions transition can contain inscriptions, described rules of access point participation. To ensure independency of inscription language from notation rules interpretator is used. It provides functioning of constructions of imperative programming languages in models described in PNP.

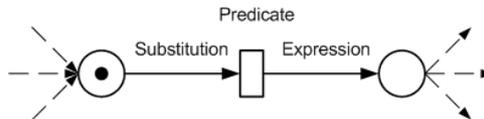


Fig. 1. Plain Petri net

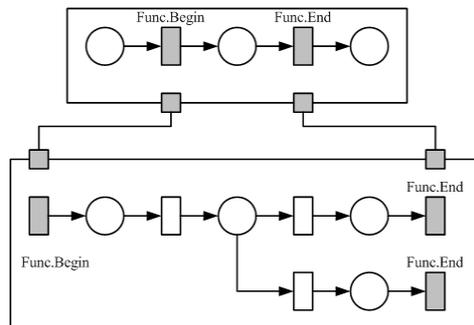


Fig. 2. Hierarchical Petri net (function call and body)

Hierarchical PNP is defined as composition of a number of PNP; each of them is represented as rectangle on a picture. Composition operation is described by two nets and two access points defined in each net. Access points are represented by little squares on net rectangle. Access points are linked with lines. Rules of nets fusion guarantee that transitions with same marks can be fired only simultaneously. Example of model of function call and function body is displayed on Fig. 2. Results of composition of function call with its body are displayed on Fig. 3.

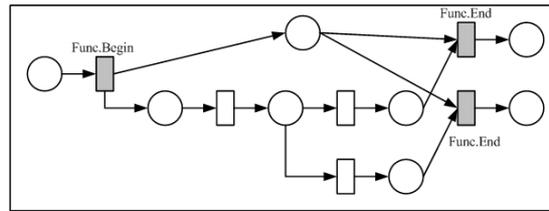


Fig. 3. Result of composition

Using PNP we can define method of parallel program model construction as following. Parallel program control flow transforms to the Petri net structure, parallel program processes with its own data transform into tokens. The control flow of the received model is handled by data, the description of operations above which remains in terms of the source programming language [4].

The models of the parallel program received by the given method possess graphics representation which can be used at debugging.

3. Debugging language in terms of Petri nets

Debugging of programs is a laborious process which for long time of the development has got own terminology which we name language of debugging. During debugging the developer uses such terms, as obtaining of a debug code, start of the program on debugging, a stop and resume of execution, execution of the program on steps, setting a breakpoints and research state of the program. In Petri nets researcher performs actually similar operations.

The evident tool of investigation of properties of the modeled program is simulation which allows displaying program functioning in dynamics. Simulation possesses many concepts similar to debugging. So, start of the program on execution, corresponds to start of simulation from initial marking. Initial marking in this case models a point of start of the program.

Each following step of simulation is an execution of some set of transitions and moving of tokens from ingoing places to outgoing ones. The given action is similar to step-by-step execution of the debugged program. In the theory of Petri nets distinguish interleaving and non-interleaving semantics of transition fired. Interleaving semantics defines firing of each separate transition for once. Non-interleaving operation allows to define firing step of several transitions. In conformity with debugging of parallel programs it can mean, that step firing is similar to detailed execution of each separate command of process. Non-interleaving operation can mean group execution of one or more operations in several processes of the program. Alternation and classification of the given possibilities gives the flexible tool for definition of a set of operations in the parallel program which the developer plans to perform for one step of execution. Also it is necessary to note, that if transition models function call, that, as well as real debuggers, it is possible to do step-by-step execution of each operation of function, and possible to treat transition firing as execution of one complex action. It is possible to interrupt execution of simulation if some marking is reached. In this case marking of a Petri net corresponds to concept of a breakpoint of traditional debugging tools.

Reachability of some marking corresponds to reachability of some state in the program. Representing the marking on the Petri net graph allows to present evidently a current state of each process separately and the program entirely. The data values, described in tokens, allows to receive the additional information on the possible reasons of an error.

4. Debugging technique in terms of Petri nets

The main purpose of debugging is detection program errors, search and correction of the reasons of their occurrence, testing of the corrected code. For this purpose the developer performs multiple execution of the program with the same data and conditions of execution. Using the mechanism of breakpoints and step-by-step execution, the developer reaches a place of occurrence of an error.

Usage PNP in this case allows visualizing process of execution of the program, producing more information for user on a current state of the parallel program.

Besides advantages of visual representation, simulation of model of the program allows to save sequence of transitions firing and sequence of accessible states. This information represents history of program execution till the given moment and allows to receive the additional information for the analysis of the reasons of error. For highlighting of the certain elements in Petri nets there is a concept of a marks which can be used at saving history of firing transitions. Allocation of marks in real programs is possible in two variants. Either it is performed by the user manually, or is automatically performed by various libraries for monitoring parallel program state. In Petri nets both cases are possible. It is possible to setup a mark according to own reasons in the given conditions of debugging, and it is possible to realize automated setup of marks according to some criteria which the program possesses. Representation by transitions of some special function calls can be one of criteria. For example, in MPI-program this criterion can be a function call of interaction. In general, it is necessary to note, that presence of representation of the program in the form of model allows to build as much as complex algorithms of automatic arrangement of a marks according to the user criteria.

Based on event tracing, the debugger can handle execution of the program, achieving identical program behavior. Presence of the given possibility simplifies debugging in conditions of non-determined execution environment, when behavior of the program is varied in parallel program restarts.

5. Correctness of debugging process

For debug process to be correct it is necessary to show, that the program modified by adding debug information does not change its visible behavior. Or in other words that debug version of program is equivalent by behavior to release version of program.

At first we should note that parallel program, written with MPI standard is a set of communicating sequential processes [1]. And sequential process can be described in terms of plain PNP. Therefore parallel program can be represented as hierarchical PNP net, composed from sequential processes nets and nets, which describe functions used in program. As for the executing program we can model it by adding PNP net that model executing environment, to hierarchical PNP net.

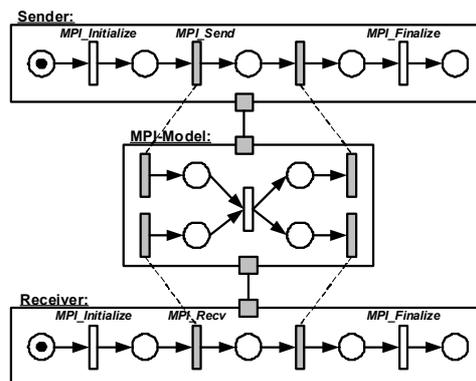


Fig. 4. Example of parallel MPI-program

The part of events that occur in parallel program is in one or another way visible, that is appears in interaction with physical devices or with other processes and other part of events is invisible. Only visible events are important for the user of the program. That is if two programs will display identical visible behaviour, for the user they will be undistinctable or just identical. In terms of Petri nets it means, that for matching parallel programs presented by Petri nets bisimulation equivalence criteria can be used.

