The Software Simulator of a Parallel Computing System with Message Passing

Eugene Gavrin
SUAI
Saint-Petersburg, Russia
eugene.gavrin@guap.ru, eugene.a.gavrin@gmail.com

Abstract

This article describes the development of simulator for parallel computing systems with message passing. It explains in details the work mechanisms of static and dynamic components of parallel programs supported by the simulator such as conditionals and looping operators, with the explanation of their work logic.

INDEX TERMS: SIMULATION, PARALLEL PROGRAMS, DYNAMIC GRAPHS.

I. INTRODUCTION

The need to solve complex applications with large amounts of computations and data processing, as well as the limitations of classical single-processor computing systems, have led to the creation of multiprocessor systems. Usage of such systems can significantly increase computational performance at any level of computer technologies. Also it should be noted that the existing methods for solving computational problems need to be modified for parallel computing to allow the execution of different processes on independent processor elements. The syntax of programming language should allow developing of parallel programs using synchronizations, asynchronous processes, threads, etc.

With the progress of technology developers got the ability to use more processing elements for computing. Such complexes that have multiple processing elements have great potential productivity and may be used for solving much more challenging tasks than classical single-processor systems can do. But parallel programs can be written without access to multi-processor system – for this purposes developers can use software simulators that allows to simulate work of program on different platforms.

II. MAIN PART

Nowadays, multiprocessor systems can be used for a wide range of tasks: scientific computations, personal computers, game consoles, mobile systems etc. Such systems are expensive to manufacture, so in order to justify financial and technological costs of production it is required to find the computational algorithms that can effectively use them.

There was a need to be able to simulate parallel programs for the purpose of debugging. To solve this problem the software complex was developed: a software system that simulates the execution of a parallel algorithm on a distributed computing system. Due to the nature of our work, investigation and development of algorithms, it can be simulated without low-level modeling.

In this work it was decided to use directed graphs to describe parallel program algorithms. Program graph is a structural model of the program that shows the relationship between its

elements. Graph nodes represent operators and data-objects. Links represent directed communications between operators.

During development it was decided to work with a system based on message passing, because these systems have no disadvantages of systems with shared memory. At the same time message passing approach allows to organize multi-processor computer systems that will be more scalable then systems with shared memory. In addition, minor improvements of simulator will allow to implement a popular class of multiprocessor systems: System-on-Chip.

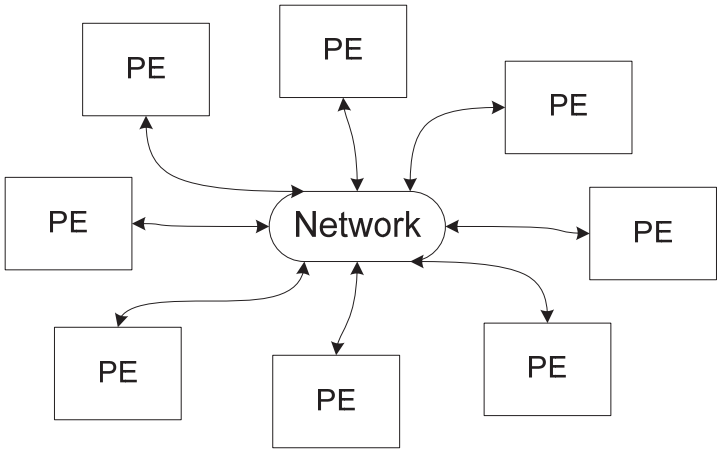

Fig. 1. Structure of computing system.

A. Development

In the developed simulator is implemented the possibility to connect a user source code, to edit the settings of the simulation and to write custom add-ons. In addition, the simulator is only intended to implement simulation of software part. Hardware part is simulated only for adequate evaluation of algorithms. During the development was provided an availability to extend the functionality for more detailed simulation of hardware platforms.

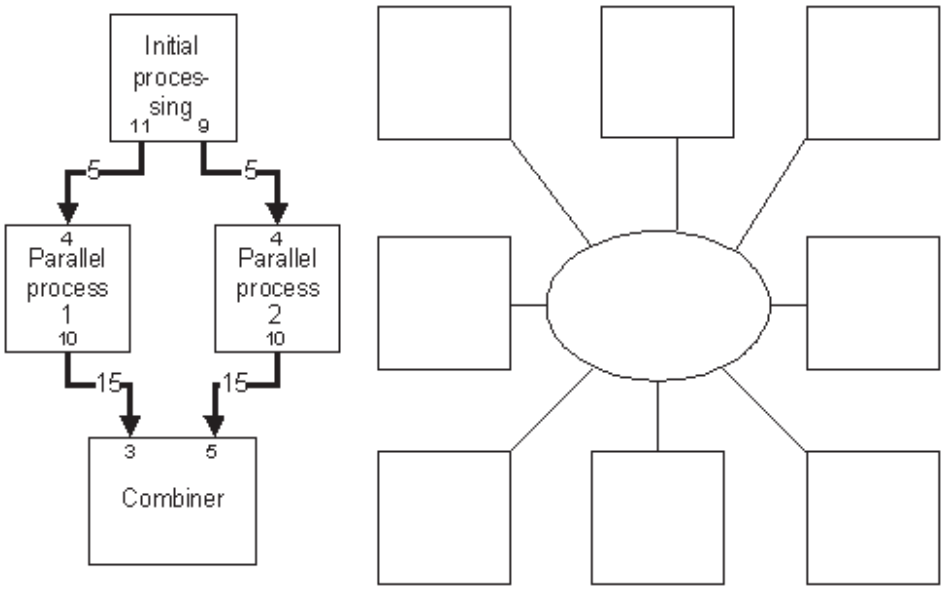

Fig. 2. Program graph and platform example.

The software implementation of the simulator consists of different types of computing tasks: terminal operators, data objects, control and conditional operators. Each has its unique properties and characteristics, which are sufficient to describe the most types of algorithms. All types of processes have general properties such as: unique process number, process name and execution cost of tasks on a given type of processor. Also processes have set of input and output ports, through which process exchanges data with other processes.

Simulator structure was divided into several components. This division of the program structure potentially allows working on the project by several people, separating the zone of responsibility. For example that will allow to test parts independently. Furthermore, such modular structure enables the programmer to modify some modules, to add or reduce the functionality, without affecting neighboring modules of the rest of the system.

In developing of simulator it was decided to use platform model with a fully connected topology network, where each processing element can directly send and receive data from any processing element. As the input files it uses a standardized description of the platform and a description of the program graph containing all the necessary data.

As the results of processing the simulator generates a set of output files, containing all the information necessary for debugging of parallel programs. In the process of simulation it creates set of files: a specialized data dump, intended for debugging with a specialized program, a simplified log file that duplicates the screen output and some other files.

B. Implemented dynamics

When we write programs for platform with limited resources, it may happen that the platform will not be able to place the entire program. In such cases, we have to use dynamic placement in order to save platform resources - only a part of the program is placed to the platform, and the rest of the program will be placed only when it will be needed. It may also happen that part of the code will never be placed on platform. In any real-world program there can be code that will never be placed because of the initial conditions of the program, for example, the branch of conditional statement. Accordingly, placement of this part of the code on the platform with scarce resources is unnecessary.

Simulator provides the ability to place and remove operators dynamically. Each operator has the property that tells simulator either the operator should be removed after completion of its execution or not. In other words, the work of this mechanism is the following: the operator is placed on the platform, simulator waits for the moment of its execution, and then operator is removed from the platform.

The system also contains data storages named “Data Objects”. For such objects the mechanism of removal from the platform is slightly different from others. This was done because such objects can run only on requests from the associated operators. Data Object is removed from the platform in the case it is no longer linked to any operator, as it is no reason to keep in the system object that will never be able to execute. The mechanism of the addition is the same as for other operators.

Also simulator supports working with conditional statements such as “If” and “Switch”. In general, the mechanism of these operators is very simple: the control port receives data from a terminal operator, and then the branch which condition corresponds to these data is placed on the platform. Thus, we save platform resources by placing only necessary operators.

Conditional operators have two mechanisms for branch unrolling: static and dynamic. In the first case, a correct branch will be placed on the platform and never be removed. In the second case, the branch will be placed on the platform. All operators of placed branches are dynamic. It means that they should be removed immediately after execution.

One of the major operators is the "Parallel For", which provides the mechanism of parallel data processing in the system. Control port of the operator receives a condition that indicates the amount of branches (iterations) to which it is necessary to divide the processing of data. Some data that are sent to the input port of the operator have to be distributed between placed branches. After processing, all branches have to be removed and will no longer occupy the resources of the platform.

III. Conclusion

While developing of the simulation software I considered the most common types of organization of parallel computing systems. I investigated their advantages and disadvantages and chose the kind of parallel systems that best meets the requirements set out in the beginning.

In the course of work were developed and described in detail methods for the simulation of parallel programs. A detailed review of all components of the simulator was done.

The developed simulator can be used by the end user without any additional or special skills. User needs only to launch applications, which will be simulated, and submits to the simulator input some command line arguments. At the end of the simulation of the application special data dumps necessary for debugging will be generated.

Acknowledgment

The first author would like to acknowledge the help and guidance of Alexey Syshikov of the SUAI University, as well as the supervision of Professor Yuri Sheynin.

References

[1] Ahmed Jerraya, Hannu Tenhunen, Wayne Wolf, “Multiprocessor Systems-on-Chips”, Computer, volume 38, number 7, pp. 36-38, 2005

[2] Abdelhamid Helali, Adel Soudani, Jamila Bhar, Salem Nasri, “Study of Network on Chip resources allocation for QoS Management”, Journal of Computer Science, volume 2, pp. 770-774, 2006

[3] Caglan M. Aras, James F. Kurose, Douglas S. Reeves, Henning Schulzrinne, “Real-Time Communication in Packet-Switched Networks”, Proceedings of the IEEE, Volume 82, Issue 1, pp. 122-139, 1994

[4] Almasi, G.S. and A. Gottlieb (1989). Highly Parallel Computing. Benjamin-Cummings publishers, Redwood City, CA.

[5] Maslennikov, Oleg (2002). "Systematic Generation of Executing Programs for Processor Elements in Parallel ASIC or FPGA-Based Systems and Their Transformation into VHDL-Descriptions of Processor Element Control Units". Lecture Notes in Computer Science, 2328/2002: p. 272.