UDC 004.272.2

Oleksii Krutko, Oleksandr Korochkin

ANALYSIS OF THREADS CONTROL TOOLS IN MODERN LANGUAGES AND LIBRARIES OF PARALLEL PROGRAMMING

The paper deals with the analysis tools for thread control used in modern language and libraries of parallel programming. Languages Java, Ada, C#, Python, libraries WinAPI, OpenMP, MPI are considered. Means optimal for solving the problems of mutual exclusion and synchronization for scalable parallel programs are defined.

Key words: threads, organization of treads communication.

Tabl.: 1. Bibl.: 6.

Target setting. The problem of developing software for parallel computer systems is becoming more urgent in connection with the growing market of multi-core processors. This work is devoted to the analysis of various means of programming and threads control in modern parallel programming libraries and languages.

Actual scientific researches and issues analysis. The organization of the interaction of threads is an important part of a parallel program, the execution of which is critical depending on both the choice and the application of means of interaction. The increase in the number of processors in modern computer systems and, accordingly, the number of interacting threads poses the task of choosing and using reliable thread synchronization tools. This is especially true for scalable systems.

Uninvestigated parts of general matters defining. This article is devoted to the selection and application of thread interaction tools for scalable computer systems where the number of processors and, accordingly, the number of threads, can change dynamically. The development of applications for scalable systems has its own characteristics and requires the use of optimal tools that will ensure the correct execution of the program and the absence of deadlocks. Therefore, this work focuses on the analysis thread control tools for scalable systems.

The research objective. The task is to analyze the existing means of interaction of threads, which will provide the possibility of choosing the means optimal for scalable systems when solving the task of reliable interaction of a large number of threads, the number of which may change.

The purpose of this work is to increase the efficiency of development and execution of parallel programs for scalable computer systems.

The statement of basic materials. Software development for parallel computer systems is based on the use of modern parallel programming languages and libraries.

The emergence of new and improvement of existing means of interaction of threads requires their constant tracking and analysis for the purpose of optimal use when building parallel programs, including for scalable computer systems.

Well-known parallel programming languages and libraries were selected for the analysis of the means of creating and organizing the interaction of threads of different levels: Java, Ada, C#, Python, WinAPI, OpenMP, MPI [1-6].

Creation (declaration) of a thread is related to the description of the thread (group of threads), the formation of the ID of the thread, setting the priority, choosing the processor for execution, the size of the stack, actions of the thread, starting and terminating.

This can be done in different ways:

- through the use of special modules (classes) (Java: class Thread, Ada: module task);
- through thread functions that define the actions and parameters of threads (C#, WinAPI) - with the help of so-called thread functions. (C#, WinAPI) Here, the thread's actions are specified through a pre-designed function that defines the thread's behavior;
- through the definition in the sequential program of the sections that will be run in parallel (OpenMP)
- through creating copies of the entire program and parallel execution of these copies (MPI, PVM).

Additional possibilities are provided by combining threads into groups (pools) and using queues of various types, which allow optimizing the execution of threads (Java, Python, Ada, C#, MPI). This is a development of the queuing mechanisms (previously proposed in the Ada language) and communicators (MPI).

Each approach has its advantages, the use of which allows you to simplify the development of a parallel program, its debugging, modification, and scaling.

Important for scalable parallel systems are solution:

- problem of dynamically creating thread
- problem of access to shared resources (mutual exclusion problem)
- thread synchronization problem.

Solutions to the first problem are provided by modern languages (libraries) of parallel programming, where the possibility of dynamic creation of threads, characteristic of scalable systems, is realized. Dynamic generation of threads requires correspondingly dynamic identification of threads. In the OpenMP and MPI libraries, identification is carried out automatically; each new thread receives an integer identifier in a corresponding parallel block or communicator. Another situation occurs if the thread name is formed directly by the developer. In Ada, dynamic thread creation provides a task type that allows you to create arrays of threads

```
task type RS is . . . end RS;
T is an array of RS (1.. N);
```

and together with the use of a discriminate, form an internal identifier:

```
task type RS(Id: integer ) is . . . end RS;
T1: RS(1); . . . T10: RS(10);
```

The second problem. As a rule, when the number of threads changes, the number of shared resources does not change, so the problem of mutual resolution can be solved by low-level means (semaphores, mutexes) and lock type means. But if it is necessary to take into account quantitative characteristics for complex accesses to shared resources, then more powerful tools are needed - monitors, for example, protected inputs in the protected unit of the Ada language, which provides additional logic for accessing shared data.

Recently, non-blocking means of thread interaction (atomic variables, non-blocking queues) have been developed, which allow minimizing the time of using shared resources.

In scalable systems, the solution to the problem of thread synchronization is the most complicated. If the use of binary semaphores and events is enough to synchronize two threads, then multiple synchronization requires more powerful means.

To implement multiple synchronization, you can use the barrier mechanism, which was developed in the form of a counting barrier and is used in almost all languages and libraries.

The most powerful tool remains the monitor mechanism. In Ada language, special constructions of the protected module are proposed for solving the synchronization task - protected entries, which have barriers that additionally define various conditions for blocking and unlocking flows. The use of these entries allows to program complex forms of thread interaction, which is important for scalable programs:

```
entry Wait_Task when Cond
```

Here, in `Wait_Task` entry barrier (`when Cond` construct), a logical variable is formed that defines the condition of blocking and unblocking the thread.

Table 1 provides data on the presence in the considered languages and libraries of parallel programming means of organizing the interaction of threads.

Table 1

Tools	Java	C#	Ada	Python	WinAPI	OpenMP	MPI
Semaphores	+	+	+	+	+		
Mutexes		+		+	+		
Events		+		+	+		
Critical Section	+	+		+	+	+	
Barriers	+	+	+	+	+	+	+
Atomic/Volatile	+	+	+	+	+	+	
Monitors			+				
Queue	+	+	+	+			

Pool	+	+		+	+	+	
Messages			+				+

Conclusions. Means of modern languages and parallel programming libraries for creating and control threads are analyzed. The given results will make it possible to choose the optimal tools when creating software for scalable parallel computer systems.

References

1. Oaks S. *Java Performance: In-depth Advice for Tuning and Programming Java 8, 11, and Beyond.* O'Reilly Media, Inc.; 2end Edition, (Februare 4, 2020), p.452.
2. Barnes J. *Programming in Ada 2012*. Cambridge University Press; 2nd edition (May 19, 2022), p. 992.
3. Nagel Chrisian, *Professional C# and .NET.* Wrox; 2021st Edition, (September, 2021), pp. 1008.
4. Gorelick M., Ozsvald I. *High Performance Python,* O'Reilly Media, Inc.; 2end Edition, (April 30, 2020), p. 470.
5. Klemm M., Cownie J. *High Performance Parallel Runtimes Design and Implementation.* Berlin, Boston: De Gruyter OldenDurg 2021, p.328.
6. Muhammad Nufail Farooqi, Miquel Pericàs *Vectorized Barrier and Reduction in LLVM OpenMP Runtime* In Proceeding of 17th International Workshop on OpenMP, IWOMP 2021, Bristol, UK, September 14–16, 2021, pp. 18-32.

AUTHORS

Krutko Oleksii – student of National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.
E-mail: aleksey.krutko @gmail.com

Korochkin Oleksandr – associate professor, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.
E-mail: avcora@gmail.com