Technical sciences

UDC 336.72

Kolesnyk Valeriia

Student of the

Kharkiv National University of Radioelectronics

MPI AS A WAY OF MESSAGING

Summary. *The theoretical aspects and characteristics of the message transfer interface MPI were investigated.*

Key words: *MPI, parallel programming, message passing.*

The parallel programming is widely used in today's development process. To produce parallel computing the multiple resources are used to solve the problem. The computations are executed simultaneously using such multiple resources as, for example, processors. To make understandable the messages of different resources some specifications for passing messages need to be used. One of such specifications is Message Passing Interface (MPI).

MPI is a message transfer interface between processes performing a single task. It is primarily intended for Massive Parallel Processor (MPP) as opposed to, for example, OpenMP. A distributed (cluster) system is typically a set of computational nodes connected by high-performance communication channels (such as InfiniBand).

MPI is the most common standard of the data transfer interface in parallel programming. MPI is standardized by the MPI Forum. MPI implementations exist for most modern platforms, operating systems, and languages. MPI is widely used in computational physics, pharmacy, materials science, genetics and other fields of knowledge.

A parallel program from the MPI point of view is a set of processes running on different computing nodes. Each process is generated on the basis of the same code. The main operation in MPI is message transmission. The MPI implements practically all the basic communication patterns: point-to-point (point-to-point), collective (collective) and one-sided (one-sided) [1]. Scalability is the main goal of parallel processing. MPI allows to support scalability.

The main advantages of establishing a message-passing standard are portability and ease of use. In a distributed memory communication environment in which the higher level routines and/or abstractions are built upon lower level message-passing routines the benefits of standardization are particularly apparent. Furthermore, the definition of a messagepassing standard, such as that proposed here, provides vendors with a clearly defined base set of routines that they can implement efficiently, or in some cases for which they can provide hardware support, thereby enhancing scalability. The goal of the Message-Passing Interface simply stated is to develop a widely used standard for writing message-passing programs. As such the interface should establish a practical, portable, efficient, and flexible standard for message passing. MPI does not define a runtime model for each process. The process may be sequential or consist of sub-processes that can be performed simultaneously. In this case, MPI blockers only block the preprocessor to which the call is directed. MPI does not provide mechanisms for determining the initial distribution of processes on physical processors. This is done by the system on which the MPI is implemented. Also the MPI-1 standard does not provide the dynamic generation and removal of processes [2].

The effectiveness of MPI is also achieved by not doing the unnecessary task of forwarding large amounts of excess information with each message or encoding-decoding message headers. MPI was designed to support simultaneous execution of computation and communication to use coprocessors. This is achieved by using unblockable communication challenges that share the initiation and completion of communication. To increase speed, MPI uses techniques that

application programmers often do not think about. For example, built-in buffering avoids delays in sending data - the control to the transmitting branch is returned immediately, even if the receiving branch is not ready to receive. MPI uses multithreading (multithreading), taking most of its work to low-priority threads(threads). Buffering and multithreading minimize the negative impact of unavoidable delays in sending on the application's performance [3]. The "one-all" data transmission takes time proportional to the number of branches involved, but to the logarithm of that number.

To describe the MPI program it is necessary to consider the basic terms used in MPI programming. MPI is a set of processes that can send messages to each other via different MPI functions. Each process has a special identifier - rank (rank). Process rank can be used in different MPI message sending operations, for example, rank can be specified as message recipient ID. The value of the process identifier is based on orders. Each process is assigned an order from each medium to which it relates. The value of an order is some integer assigned to it, starting from scratch and indicating all individual processes in the context of a certain communication medium (communicator). Common practice is to define the process whose global order is 0 as the host process. Through the value of order (rank), the developer can determine who the sender is and who the recipients are instead. In addition, there are special objects called communicators (communicators) in MPI that describe groups of processes. Each process within a single communicator has a unique rank [4]. The same process may apply to different communicators and may therefore have different ranks within different communicators. Each data transfer to MPI must be performed within a communicator. A more efficient implementation is achieved by taking advantage of associativity and using a logarithmic tree reduction.

It is well-known fact, that the necessity to forward data may not be adjacent and may consist of different types of values. Of course, and such fragmented data can be transmitted using multiple messages, but the solution method will not be

effective because of the latency of the set of implementations data transfer operations. Another possible approach may be pre-packaging data transmitted to a continuous vector format, but there are also redundancies of data copying operations, and the comprehensibility of such transmission operations, will be far from desirable. To improve the ability to determine the composition of messages in MPI a mechanism of derived data types.

MPI provides to programmers a single mechanism for the interaction of branches within the parallel application independently of the machine architecture (single processor / multiprocessor with shared/separate memory), mutual branch arrangement (on one processor / on different) and the API of the operating system. (API = "applications programmers interface" = "application developer interface"). The program using MPI is more easily debugged (it narrows down the amount of space available to make stereotypical errors in parallel programming) and is transferred more quickly to other platforms (ideally, simple recompilation).

Among the advantages of MPI could be highlighted the following thing below:

1. MPI is originally a FAST tool. It uses techniques that application programmers often do not think about. For example: built-in buffering avoids delays in sending data - the control to the transmitting branch is returned immediately, even if the receiving branch is not ready to receive.

2. Using multithreading (multithreading), taking most of its work to low-priority threads (threads). Buffering and multithreading minimize the negative impact of unavoidable delays in sending on the performance of the application.

3. "One-all" data transmission time proportional to, and logarithm of, the number of branches involved.

The attractiveness of the message-passing paradigm at least partially stems from its wide portability. Programs expressed this way may run on distributed-memory multiprocessors, networks of workstations, and combinations of all of

these. In addition, shared-memory implementations, including those for multi-core processors and hybrid architectures, are possible. The paradigm will not be made obsolete by architectures combining the sharedand distributed-memory views, or by increases in network speeds. It thus should be both possible and useful to implement this standard on a great variety of machines, including those "machines" consisting of collections of other machines, parallel or not, connected by a communication network. The interface is suitable for use by fully general MIMD programs, as well as those written in the more restricted style of SPMD. MPI provides many features intended to improve performance on scalable parallel computers with specialized interprocessor communication hardware. Thus, we expect that native, high-performance implementations of MPI will be provided on such machines. At the same time, implementations of MPI on top of standard Unix interprocessor communication protocols will provide portability to workstation clusters and heterogenous networks of workstations [5].

However, MPI developers have been severely criticized for being too cumbersome and complex for an application programmer. The interface was also difficult to implement. As a result, there are currently almost no MPI implementations that fully integrate exchange and computation.

References

1. William Gropp, Ewing Lusk, and Anthony Skjellum. Using MPI: Portable Parallel Programming with the Message-Passing Interface. 328 с.
2. MPI: A Message-Passing Interface Standard Version 3.0 URL: https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf (дата звернення: 29.09.2020).
3. Міллер, Р. Послідовні і паралельні алгоритми: Загальний підхід / Р. Міллер, Л. Боксер; пер. з англ. М.: БИНОМ. Лабораторія знань, 2006. 406 с.

4. Воеводін В. В. Технологии параллельного программирования. Message Passing Interface (MPI). URL: https://scinse.donntu.edu.ua/sp/sukhomlinov/library/7.pdf (дата звернення: 29.09.2020).
5. MPMD Launch Mode URL: https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-guide-linux/top/running-applications/mpmd-launch-mode.html (дата звернення: 29.09.2020).