Journal of Physics: Conference Series

Using relaxed concurrent data structures for contention minimization in multithreaded MPI programs

To cite this article: Andrey V Tabakov and Alexey A Paznikov 2019 *J. Phys.: Conf. Ser.* **1399** 033037

Using relaxed concurrent data structures for contention minimization in multithreaded MPI programs

Andrey V Tabakov and Alexey A Paznikov

Department of Computer Science and Engineering Saint-Petersburg Electrotechnical University “LETI”, St. Petersburg, Russia

E-mail: komdosh@yandex.ru, apaznikov@gmail.com

Abstract. Parallel computing is one of the top priorities in computer science. The main means of parallel processing information is a distributed computing system (CS) - a composition of elementary machines that interact through a communication medium. Modern distributed VSs implement thread-level parallelism (TLP) within a single computing node (multi-core CS with shared memory), as well as process-level parallelism (PLP) process-level parallelism for the entire distributed CS. The main tool for developing parallel programs for such systems is the MPI standard. The need to create scalable parallel programs that effectively use compute nodes with shared memory has determined the development of the MPI standard, which today supports the creation of hybrid multi-threaded MPI programs. A hybrid multi-threaded MPI program is the combination of the computational capabilities of processes and threads. The standard defines four types of multithreading: Single - one thread of execution; Funneled - a multi-threaded program, but only main thread can perform MPI operations; Serialized - only one thread at the exact same time can make a call to MPI functions; Multiple - each program flow can perform MPI functions at any time. The main task of the multiple mode is the need to synchronize the communication flows within each process. This paper presents an overview of the work that addresses the problem of synchronizing processes running on remote machines and synchronizing internal program threads. Method for synchronization of threads based on queues with weakened semantics of operations is proposed.

1. Introduction

Multi-core and multiprocessor computing systems (CS) include a wide class of systems – from embedded systems and mobile devices to cluster computing systems, massively parallel systems, GRID systems, and cloud-based CS. Algorithmic and software tools for parallel programming is the basis for building modern systems for processing big data, machine learning and artificial intelligence. The main class of systems used for high-performance information processing are distributed CS - collectives of elementary machines interacting through a communication environment. In design of parallel programs for distributed CS, the de facto standard is the messaging model, which is primarily represented by the MPI (Message Passing Interface) standard. The scalability of MPI programs depends significantly on the efficiency of the implementation of collective information exchange operations (collectives). Such operations are used in most of the MPI programs, they account for a significant proportion of the total program execution time. An adaptive approach for development of collectives is promising. Nowadays, using only the message passing model (MPI-everywhere) may not be sufficient to develop effective MPI programs. In this regard, a promising approach is to use MPI for interaction between computer

nodes and multithreading support systems (PThreads, OpenMP, Intel TBB) inside the nodes. The main task of implementation of hybrid mode is the organization of scalable access of parallel threads to shared data structures (context identifiers, virtual channels, message queues, request pools, etc.).
Types of MPI standard hybrid mode:

- MPI_THREAD_SINGLE - one thread of execution
- MPI_THREAD_FUNNELED - is a multi-threaded program, but only one thread can perform MPI operations
- MPI_THREAD_SERIALIZED - only one thread at the exact same time can make a call to MPI functions
- MPI_THREAD_MULTIPLE - each program flow can perform MPI functions at any time.

One of the implementations of the hybrid multi-threaded MPI program in the MPI_THREAD_MULTIPLE mode is the MPICH version CH4 library, which defines standards for using lock-free data structures. In this mode, two types of synchronization are available: trylock - in which the program cyclically tries to capture the mutex and access the queue; handoff - a thread-safe queue when accessed which causes an active wait for an item by a thread.

2. Hybrid mpi+threads synchronization models

While developing multi-threaded MPI programs, the main problem is the synchronization of thread accesses to distributed data structures stored in shared memory. Representatives of these structures are: arrays, lists, queues, stacks, trees, and graphs.

In [1], the problem of ensuring thread-safe execution of basic MPI functions is considered. An algorithm is proposed for calculating an integer context identifier (context id), which is necessary for creating a new MPI communicator. In hybrid MPI programs, this procedure can be performed in the MPI_THREAD_MULTIPLE mode, in which each thread can perform information exchanges. In single-threaded mode, an identifier is allocated for each process based on a global structure containing valid process identifiers. Each time a new process connects to the communicator, it is issued a new identifier using the operation MPI_Allreduce. This operation works with an integer identifier as an array of bits, realizing the bitwise multiplication of all identifiers existing in the communicator, then the last position of the bit set in the unit is calculated, which serves as the identifier. The described approach requires modification for a multi-threaded MPI program. Since the threads are executed simultaneously, the mutex capture order protecting the change of identifier in the global memory can be different from MPI, including simultaneous, which can lead to a deadlock in the MPI_Allreduce method. The algorithm presented in the article solves this problem with the help of additional local copies of global variables, which are distributed among the processes.

In [2], the authors propose four approaches to thread synchronization in MPI + threads programs: Global, Brief Global, Per Object, Lock-free. The Global method assumes the organization of a single lock for all processes, which is used for all MPI functions (except functions that can block the execution of communication operations). Brief Global also uses a single lock, however, unlike Global, the critical section is organized only in those functions that access shared data structures, while other functions can be performed in parallel in different threads. The implementation of this approach requires much more effort than Global, since it requires a more thorough analysis of the code. Per Object - separate critical sections for different objects and classes of objects, for example, several critical sections can be used to refer to the same process; Lock-free - in this approach, threads are synchronized due to atomic operations implemented by the processor's capabilities.

The article [3] shows that in the existing implementations of the MPI_THREAD_MULTIPLE mode, unfair thread synchronization algorithms are used, which leads to a decrease in the efficiency of information exchanges. The blocking algorithms used in MPI libraries do not guarantee fair capture of critical sections by threads, which can cause access monopolization by one of the threads. In addition, in computing nodes with non-uniform memory access (NUMA architecture), high latency of access to remote memory segments increases the transfer time of the right to perform the critical section (the interval between releasing the lock and capturing it by another thread) between threads. The paper proposes two thread synchronization algorithms that ensure fair performance of critical sections and reduce overhead in nodes based on NUMA architecture. The first algorithm implements the Ticket Lock method of locking threads, which allows you to organize a queue of threads by the time the operation is called. The second algorithm is a modified version of the first, with the difference that some tasks have an increased priority and are executed earlier with less priority tasks.

In [4], the analysis of the validity of the capture of Pthread mutexes was carried out when implementing the MPI + threads hybrid model. Based on the result presented in [3], an improved version of the thread-locking algorithm based on Ticket Lock was created and an algorithm based on the CLH-algorithm and its modification with the addition of priority (CLH-LPW) for locking was proposed. This approach allows efficient use of critical sections, which reducing threads starvation.

The paper [5] proposed several methods for optimizing the execution of hybrid MPI + threads programs. The first method created a thread-safe hash table based on locks, which is proposed to be used to find correspondences between the received messages and the corresponding queries. The second method aims to optimize thread planning. The authors have developed a light-weight threads scheduler (light-weight threads) that uses a bitmap to indicate running threads. In the third method, it is proposed to change the message packet pool by dividing the centralized pool into private pools for each thread. Initially, each thread has a fixed number of packets for processing, however, during the execution of the program, a work-stealing approach is used, allowing unused threads to process packets from foreign pools.

In [6], the authors proposed two optimization algorithms for thread synchronization. The essence of the first algorithm is to reduce the number of threads to select from a pool of waiting threads within one critical section. Other threads should wait for the execution of their request outside the critical section. This will reduce the search time for free thread to perform the operation, as it reduces the delay time and network load. The second algorithm implements the thread selection for the next operation, preference is given to the threads that just completed the work, and the waiting threads are queued with a lower priority.

The authors of [7] presented an approach that allows to adaptively select the most suitable barrier synchronization algorithm at which runtime will be minimized.

3. Use of relaxed concurrent queues with priority for mpi + threads model

A popular implementation of the MPI standard is the open-source MPICH library. The current version of CH4, as a work queue uses the izem library, which provides thread-safe data structures, such as a thread-safe queue, etc., as well as various synchronization mechanisms.

Relaxed data structures are based on the idea of an indefinite order of operations; thanks to this feature, it is not necessary to perform expensive synchronization of the data structure when working with it in several threads at the same time. Relaxed thread-safe data structures provide sufficient permissible accuracy when performing operations [8].

It is proposed to replace the thread-safe work queue from the izem library with a relaxed thread-safe queue - Multiqueues, in which the mechanism for accessing queue elements has been improved [9].

At the heart of the approach of relaxed semantics of performing operations lies the trade-off between scalability (performance) and the correctness of the semantics of performing operations. It is proposed to relaxed semantics of operations to increase the possibility of scaling. For example, when searching for the maximum element in an array, the flow may skip, blocked by other threads, sections of the array to improve the performance of the search operation, while losing the accuracy of this operation. The implementation of the optimized relaxed Multiqueue uses threads affinity to core, for reducing cache misses while reading the local kernel cache [10].

The use of a Multiqueues with relaxed semantics for performing operations will avoid the occurrence of bottlenecks when synchronizing threads (figure 1). Unlike most existing lock-free thread-safe data structures and locking algorithms, where there is a single point of execution of operations on the structure, a set of simple sequential structures is used in relaxed data structures, the composition of which is considered as a logical single structure. As a result, the number of possible points of access to this structure increases. This approach will allow to achieve much greater throughput compared to existing data structures.

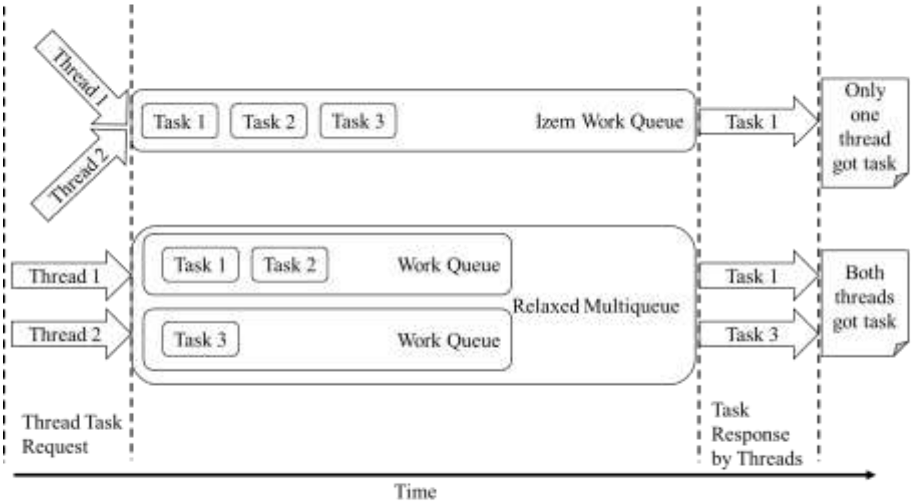

Figure 1. Izem and Multiqueue work queue implemetation comparison.

Conclusion

The presented paper consider various approaches are described and proposed for increasing the capacity for synchronization of threads in hybrid multi-threaded MPI programs (MPI + threads model). It is proposed to use a scalable thread-safe queue with relaxed semantics, which will reduce the overhead of synchronizing threads when performing operations with a working task queue.

References

[1] Gropp W and Thakur R 2007 Thread-safety in an MPI implementation: Requirements and analysis *Parallel Computing* **33(9)** 595-604

[2] Balaji P *et al.* 2010 Fine-grained multithreading support for hybrid threaded MPI programming *The International Journal of High Performance Computing Applications* **24(1)** 49-57

[3] Amer A *et al.* 2015 MPI+ threads: Runtime contention and remedies *ACM SIGPLAN Notices* 50(8) 239-48

[4] Amer A *et al.* 2016 Locking aspects in multithreaded MPI implementations *Argonne National Lab., Tech. Rep. P6005-0516*

[5] Dang H V, Snir M and Gropp W 2016 Towards millions of communicating threads *Proc. of the 23rd European MPI Users' Group Meeting. ACM* pp 1-14

[6] Dang H V *et al.* 2017 Advanced thread synchronization for multithreaded MPI implementations *Cluster, Cloud and Grid Computing (CCGRID)* pp 314-24

[7] Zharikov V, Paznikov A, Pavsky K and Pavsky V Adaptive Barrier Algorithm in MPI Based on Analytical Evaluations for Communication Time in the LogP Model of Parallel Computation *Proc. Of the International Multi-conference on Industrial Engineering and Modern Technologies (Far East Con-2018), 2018* pp 1-5

[8] Paznikov A and Anenkov A 2018 Implementation and Analysis of Distributed Relaxed Concurrent Queues in Remote Memory Access Model *Proc. of the 13th International Symposium "Intelligent Systems – 2018" (INTELS'18). Procedia Computer Science, 2019* vol 150 pp 654-62

[9] Tabakov A and Paznikov A 2019 Algorithms for Optimization of Relaxed Concurrent Priority Queues in Multicore Systems *IEEE Conference of Russian Young Researchers in Electrical*

and Electronic Engineering (EIConRus) pp 360-5

[10] Paznikov A and Shichkina Y 2018 Algorithms for Optimization of Processor and Memory Affinity for Remote Core Locking Synchronization in Multithreaded Applications *Information* **1** 1-12