UDC 004.4

I.V. Ponomarev

RESEARCH OF EFFICIENCY OF DEVELOPMENT METHODS PARALLEL APPLICATIONS ON .NET PLATFORM

Abstract. The development of parallel applications allows more efficient use of the computing power of modern computers that have many processors and cores. The Task Parallel Library for C# Multi-Thread Programming offers developers the methods for performing parallel tasks with varying productivity and speed. For parallel programs, a situation may arise when a test run on the same data yields different results due to changes in the order of calculations performed in parallel. The study of the efficiency of parallelization methods is useful in developing high-performance parallel applications.
Keywords: .NET, Task Parallel Library, Parallel, PLINQ, Task, Parallel.For, Parallel.ForEach, thread, data parallelism, task parallelism.

Formulation of the problem. Today, most contemporary programming languages in one form or another include tools that allow you to develop parallel programs.

To take advantage of several kernels is relatively simple for server applications, where each thread can independently handle a separate request from the client, but this is much more difficult to achieve for client applications, because in this case, it may usually be necessary to modify the code that intensively uses the calculation, as follows:

- divide it into small pieces,
- execute each fragment in parallel in different streams,
- at the end of the implementation, combine the results into an effective way.

The study of the efficiency of parallelization methods is useful in developing high-performance parallel applications.

Purpose of the research. Consider the Task Parallel Library (TPL) classes for multithreaded C# programming that enhance the efficiency of multi-core processors:

- Parallel LINQ (PLINQ);
- Parallel class;

 ISSN 1562-9945

- Constructs for Task Task Parallelism.

Conduct a comparison of the methods of parallelism on the possibilities of accelerating the program.

Main part. The TPL library includes two levels of functionality [1]:

- The top level is for structured data parallelism: PLINQ and the Parallel class.
- The lower level contains a class for task parallelism Task [2].

The library of task parallelism allows you to create thousands of tasks with minimal overhead. The Parallel and PLINQ classes automatically break down tasks per unit of work, since they are built on the basis of the tasks of parallel parallelism.

PLINQ automatically parallels local LINQ queries. PLINQ is easy to use, since splitting the task and combining results rests on it [3].

Parallel.For and Parallel.ForEach methods are similar to C # for loop operators for and foreach, except that iterations of the elements of the sequence occur in parallel, rather than sequentially.

The problem of developing effective programs with the help of the Task class is that parallel tasks must be quite voluminous, since the overhead of creating Task objects, planning individual tasks and waiting for their implementation can be much more than the cost of processing itself.

The PLINQ library provides the richest functionality: it automates all stages of parallelism, including the division of tasks into tasks, the execution of these tasks in different streams and the aggregation of results into one source sequence. Its use is called declarative, since it simply declares what needs to be done in parallel, and she cares about the details of implementation. In contrast, other approaches are imperative; in this case, it is necessary to clearly write the code for the division of the task and the combination of the results. In the case of the Parallel class, it is necessary to combine the results themselves, in the case of parallel structures of tasks, you also need to split the task independently.

In order to compare the efficiency of the methods of parallelism, the problem of finding the integral space was chosen, namely, the calculation of the number π is the simplest of the methods of numerical integration - the method of rectangles.

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi. \qquad (1)$$

In parallel execution of calculations, for each method of parallelism the sequential algorithm has been modified taking into account specific features of the

methods. Calculations are performed with different levels of parallelism - the number of tasks performed simultaneously to process the request.

The calculation with a parallel LINQ query uses the `ParallelEnumerable.Range (Int32, Int32)` method that generates a parallel sequence of integers in a given range, and the `ParallelEnumerable.WithDegreeOfParallelism <TSource> (ParallelQuery <TSource>, Int32)` method that specifies the degree of parallelism for use in request:

```
return (from i in ParallelEnumerable.Range(0,
(Int32)CountRect).WithDegreeOfParallelism((Int32)k)
let x = (i + 0.5) * step
select 4.0 / (1.0 + x * x)).Sum() * step;.
```

The calculation using Parallel.For uses the method

`Parallel.For(Int64, Int64, Action <Int64, ParallelLoopState>)`, which performs a for loop with 64-bit indices, providing the ability to run iterations in parallel, as well as control the state of the loop and control of this state, and the method

`ParallelEnumerable.WithDegreeOfParallelism <TSource> (ParallelQuery <TSource>, Int32),`

with the property `ParallelOptions.MaxDegreeOfParallelism`, which specifies the maximum number of parallel tasks included for this `ParallelOptions` instance.

The Parallel.ForEach algorithm uses the method

`Parallel.ForEach <TSource, TLocal> (Partitioner <TSource>, ParallelOptions, Func <TLocal>, Func <TSource, ParallelLoopState, TLocal, TLocal>, Action <TLocal>)`

which performs the foreach operation with the local flow data for the Partitioner object, providing the ability to run iterations in parallel, configure the loop parameters, as well as control the state of the loop and control that state, with the property `ParallelOptions.MaxDegreeOfParallelism`, which specifies the maximum number of parallel tasks that are included for this ParallelOptions instance:

```
Parallel.ForEach(Partitioner.Create(0, (Int64)CountRect),
new ParallelOptions
{ MaxDegreeOfParallelism = (Int32)k }, ( ) => 0.0,
(range, state, local) => ....
```

Parallel computation π by separating work between tasks Task:

1) creates an array of tasks

```
Task[ ] taskArray = new Task[k];
```

2) uses the `TaskFactory.StartNew(Action)` method, which creates and runs an integral amount calculation task:

```
taskArray[i] = Task.Factory.StartNew(obj => ...
```

3) Expects completion of all tasks

```
Task.WaitAll(taskArray);
```

4) from each task in the array of tasks, individual data is collected.

```
foreach(may id_task in taskArray) {
    can data = id_task.AsyncState as CustomData; ...
```

Testing of programs with different methods of parallelism and for different tasks in duration is carried out; the collected results are shown in Fig. 1 and Fig. 2.

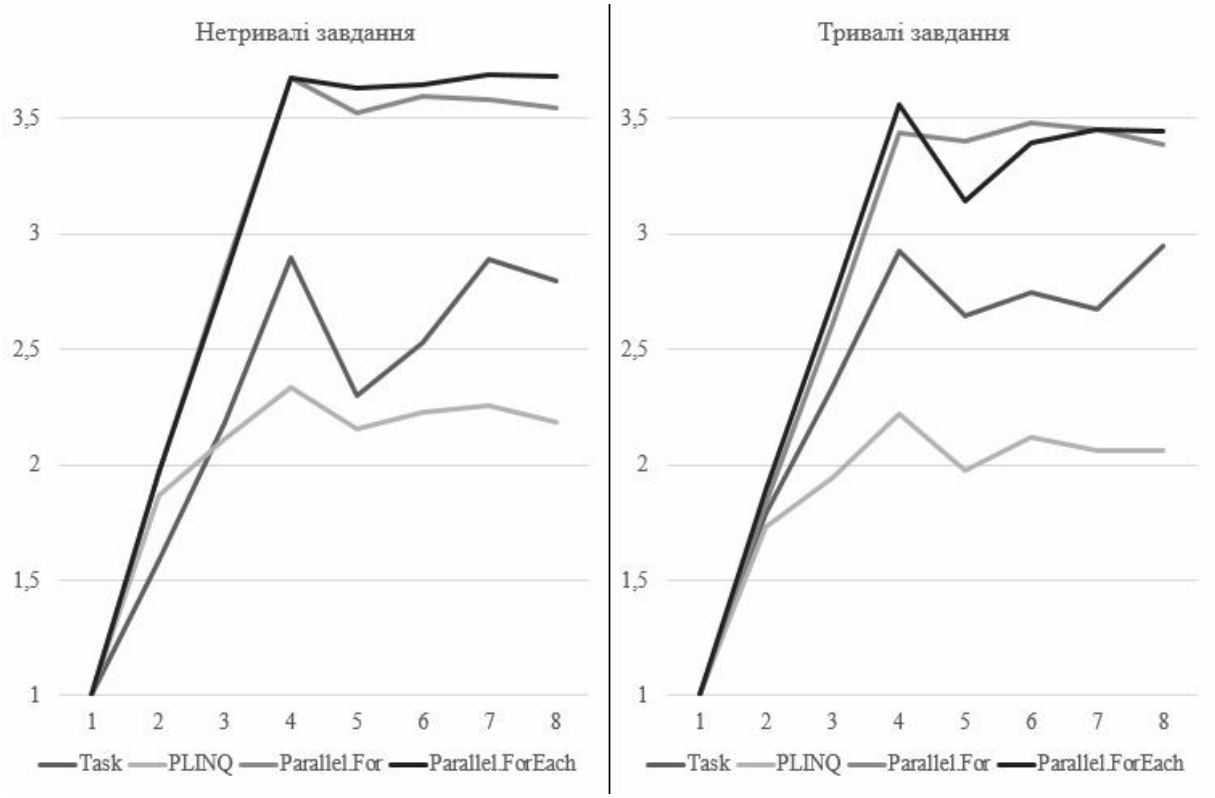

Figure 1 - Graphs of acceleration dependence on the number of threads for short and long tasks

Conclusions. For short tasks, the Parallel.ForEach, Parallel.For and PLINQ methods have a bit higher acceleration than long-term tasks.

All methods of parallelism show the greatest efficiency when matching the number of parallel threads to the number of CPU cores. This is due to the fact that the lack of overloading the processor with too many active threads prevents the performance drop that occurs when the operating system is forced to perform a large number of costly switching operations on the context.

The methods of parallelism to accelerate the work of the program are in the following sequence, from the fastest:

Parallel.ForEach,

Parallel.For,

Task,

PLINQ.

REFERENCES

1. Parallel Programming in .NET – 2018 [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/index.

2. Task Parallel Library (TPL) – 2017 [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl.

3. Parallel LINQ (PLINQ) – 2017 [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/parallel-linq-plinq.

REFERENCES

1. Parallel Programming in .NET – 2018 [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/index.

2. Task Parallel Library (TPL) – 2017 [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl.

3. Parallel LINQ (PLINQ) – 2017 [Electronic resource]. – Access mode: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/parallel-linq-plinq.