УДК 004.25

Коваль Ю.В., асистент,
Крак Ю.В., д.ф.-м.н., проф.

Iu. V. Koval, Assistant,
Iu. V. Krak, Prof.

Алгоритм сортування решетами

Sieve sorting algorithm

Київський національний університет імені
Тараса Шевченка, 83000, м. Київ, пр-т.
Глушкова 4д,
e-mail: kafedraTK@unicyb.kiev.ua

Taras Shevchenko National University of
Kyiv, 83000, Kyiv, Glushkov ave., 4d,

e-mail: kafedraTK@unicyb.kiev.ua

Метою даної роботи є представити алгоритм сортування, який дозволяє паралельно виконувати операції порівняння, проаналізувати складність та час виконання алгоритму, обговорювати віртуальний процес для цього алгоритму та реалізацію програми мовою C++, яка імітує такий віртуальний процес. У цій роботі були вирішені такі проблеми: запропоновано алгоритм сортування решетами, доведено правильність його виконання, досліджено складність та терміни роботи цього алгоритму, запропонована структура віртуального процесу, що реалізує цей алгоритм, обговорювалася програма мовою C++, що імітує такий віртуальний процес. Код програми може бути взятий у авторів. Також було вдосконалено алгоритм та доведено правильність такого вдосконалення. У процесі вирішення проблем було виявлено незручність класичної UNIX-подібної реалізації для роботи з механізмами пайпів та створення процесами.

Ключові слова: сортування решетами, віртуальний процес.

The purpose of this paper is to present a sorting algorithm that allows parallel execution of comparison operations, analysing complexity and runtime of the algorithm, discussing about a virtual process for this algorithm, and the implementation of the C++ program that simulates such a virtual process. The following problems were solved in this paper: the algorithm of sieve sorting was proposed, the correctness of its execution were proved, the complexity and timing of this algorithm were investigated, the structure of the virtual process implementing this algorithm was proposed, the C++ program that simulates such a virtual process was discussed. Program code may be taken from the authors. The improvement for the algorithm also provided and correctness of such improvement proved. In the process of problem solving the inconvenience of classical UNIX-like implementation for working with the mechanisms of piping and the processes creating was found out.

Key Words: sieve sorting, virtual process.

Статтю представив д.ф.-м.н., проф. Анісімов А.В.

1. Introduction and formulation of the problem

A traditional computer has one processor. The consequence of this was the technology of sequential programming. Increasing the number of processors from one to two can not solve the problem of running a parallel program. The number of simultaneously evaluated commands limited to number of processors. The maximum achievement is the parallel execution of two programs. Such an approach to parallel computing was in operating systems similar to UNIX [1]. The command line operator pipe [2] allows you to build chain of processes, each of which is desirable to be performed in parallel. In a classic UNIX-like system with one

processor, parallel execution was impossible and was imitated using time-sharing technology. The parallelism was in that that it was impossible to predict which process would be executed at a given time point. This approach has allowed to extend such parallelism to multiprocessor systems where the number of processors is not significantly more than one. Simultaneous use of the pipe with two processes executed by two different processors practically does not differ from the one-processor case.

Let's consider the problem of sorting the values of some type with the missing property of the neighborhood [3]. For a single-processor system, it is proved that the complexity of the algorithm with comparison for solving such a problem can not be

less than $n \cdot \log(n)$, where n is the number of values for processing [4].

The purpose of this paper is to present a sorting algorithm that allows parallel execution of comparison operations, analysing complexity of the algorithm, discussing about a virtual process for this algorithm, and the implementation of the C++ program that simulates such a virtual process.

2. Algorithm for Sieve Sorting

Sieve sorting of the sequence of values is performed by the system of sequential adaptive sieves each of which selects the value from the sequence according to the specified criterion. The number of sieves is determined by the number of elements to be sorted. The algorithm of the first element of the system of sieves is the algorithm for the generation the sequence of values to be sorted. Also this element gather sorted sequence. The algorithm of each next sieve is the same and looks as follows:

- Step 1. Get the first value. Make it as a value that has been found.
- Step 2. Get the first next value.
- Step 3. If the is no such a value, then process of sieves constructing is finished. Return the value that has been found to the previous element (sieve). Stop.
- Step 4. Compare the first next value and the value that has been founded. In the case of the criteria violation for sorting, swap them.
- Step 5. Create the next sieve and pass the first next value to it.
- Step 6. Get the next value.
- Step 7. If the is no next value, then process of sieves constructing is finished. Return the value that has been found to the previous element (sieve). Go to Step B.
- Step 8. Compare the next value and the value that has been founded. In the case of the criteria violation of sorting, swap them.
- Step 9. Transfer the next value to the next sieve.
- Step A. Go to step 6.
- Step B. Get one by one all ordered values from next sieve and send them to previous element (sieve). Stop.

The specified sorting system works correctly in accordance to the following theorem.

Theorem 1. The sieve that performed specified algorithm correctly sorts a given sequence of values.

Let prove this theorem by induction on the length of sequence to be sorted.

Induction base. Let the length of sequence is 1. This sequence is always sorted by any criterion. The first value that is received by sieve is returned as sorted by Step 3. So for such sequences system perform sorting correctly.

Induction step. Assume that sieve correctly sort the sequence of length n . Show now that the sequence of length $n+1$ will be sorted correctly. Each next sieve sorts the sequence which length is one shorter. The value that founded by the sieve is satisfying the sort criterion in comparison to values passed to next sieve according to Steps 4 and 8. The resulting (sorted) sequence of the sieve is composed from the founded value and the resulting sequence from the next sieve (Steps 7 and B). This sequence is correctly sorted according to the sorting criterion. So the induction step is proved.

A schema of evaluation of such an algorithm is presented in Figure 1.

Fig. 1. Sieve sorting schema

3. Complexity and runtime of the Algorithm

The complexity of sorting algorithms is determined by the number of comparison performed

on sorted values. Let's have n elements to be sorted. Then the first sieve will perform $n-1$ comparisons. Each next sieve performs one comparison less. In general, the total number of comparisons is estimated as n^2 . This coincides with the complexity of bubble sorting [5]. Thus, the following theorem is valid.

Theorem 2. The complexity of the sieve sorting algorithm is n^2 where n is the number of values to be sorted.

The execution time of the algorithm as well as its complexity is estimated in the number of executed single actions. For sequential algorithms, these two values coincide, since only one action is performed at a time. For parallel algorithms, the execution time and the complexity of the algorithms do not coincide because of the ability to perform some actions simultaneously.

The next figure shows dependence of the number of simultaneous actions from the algorithm execution step number.

Fig. 2. The amount of actions from the step number dependence

All sieve and generator (the first element) of the proposed algorithm work in parallel. In the first step of the execution of the algorithm, the following actions will be performed: the generator creates the first sieve and transmits the value to it. In the second step, the generator transmits the next value to the first sieve. In the third step, the generator again sends the next value (new one) to the first sieve. At the same time, the first sieve compares two values, performs sorting, creates the next sieve and passes the corresponding value to it. In the fourth step, the generator again passes the next value to the first sieve. At the same time, the first sieve compares two values, performs sorting actions, and passes the corresponding value to the second one. At the same time, the second sieve compares two values, performs sorting, creates the next sieve and passes the corresponding value to it. Thus, at each subsequent step, the number of comparative

operations performed simultaneously increases by one. After the n steps are completed, the generator stops generating new values. Starting from this moment, the number of comparisons performed simultaneously, with each step, is decreased by one. As the returning of the result values requires n additional actions, the time of performance increased by n action equivalent to the comparison of values. Thus, the total execution time of the algorithm provided the simultaneous execution of the action, is $3n$.

This allows us to formulate the following theorem and consider it as proven.

Theorem 3. The execution time of the sieve sorting is linear to the number of specified values.

4. Improvement of sieve sorting

In the Step 3 and 7 resulting values returned to previous element of the sorting system. As a result this process takes n time unit. It is possible to change this by changing the previous element to the first element. The correctness of improved algorithm based on the fact, that each sieve returns the founded (resulted) value after the last value passed this sieve. So the next theorems considered as proven.

Theorem 4. The improved sieve algorithm perform correct sorting of given sequence of values.

Theorem 5. The execution time of the improved sieve sorting is linear to the number of specified values.

5. Virtual Process for the Sieve Sorting discussion

The complexity in the practical implementation of the above algorithm and practical achievement of the declared time-of-execution evaluation is the lack of computer systems with a sufficient number of processors. The virtual process [6] that implements the algorithm can be performed on many computers at the same time. The generator and each of the sieves are implemented by a separate atomic process. When the ratio of the number of processors to the number of values to be sorted reach the values one to one, the execution time of the virtual process will correspond to the theoretically estimated.

6. Realization of the Virtual Sorting Process in Linux discussion

Linux Mint 18.3 32-bit was chosen as Linux version. This choice is motivated by the greatest popularity of this distribution among all others today. The distribution has been installed on a virtual computer with 2GB of RAM and four processors. The host computer has 8GB of RAM and an Intel i5 processor.

In the program, one atomic virtual process is implemented by a separate Linux process. Creating

new processes is done by calling the `fork()` function. Pipes are used to transfer the values between the processes. The value `-1` is used as a sign of completion of the value sequence. The processing of false situations is carried out in the final actions of the `main()` function after executing the cutting `goto`. The program does not try to be a `good_style_program`, but a program that solves the problem. The use of cutting `goto` is due to the lack of "good" implementations of work with `pipe(s)` and `fork(s)`.

7. Conclusion

In this paper the following problems were solved: the algorithm of sieve sorting was proposed, the

correctness of its execution were proved, the complexity and timing of this algorithm were investigated, the structure of the virtual process implementing this algorithm was proposed, the C++ program that simulates such a virtual process was proposed. In the process of problem solving the inconvenience of classical UNIX-like implementation for working with the mechanisms of piping and the processes creating was found out. This will be the direction of further research for the development of virtual processes.

Список використаних джерел

1. Thompson K., Ritchie D. M., Unix programmer's manual // Bell Labs, USA. November 1971. - p. 194.
2. Kernighan B. W., Pike R., The unix programming environment // Bell Labs, USA. 1984.
3. Knuth D., The art of computer programming Volume 3: Sorting and Searching, Second Edition. // Addison-Wesley. 1998.
4. Cormen T.H., Cormen T.H., Leiserson C.E., Rivest R.L., Stein C., Introduction to algorithm // MIT Press. 2001.- p. 1180.
5. Demuth H., Electronic data sorting // PhD thesis. Stanford University. 1956.
6. Крак Ю.В., Коваль Ю.В., Ставровський А.Б. Віртуальний процес: означення та застосування в створенні системи жестового інтерфейсу // Вісник Київського національного університету імені Тараса Шевченка Серія фізико-математичні науки. – 2015. – №1. - 141-144 с.

References

1. THOMPSON, K. & RITCHIE, D. M. (November 1971) Unix programmer's manual. *Bell Labs, USA*. p. 194.
2. KERNIGHAN, B. W. & PIKE, R. (1984) The unix programming environment. *Bell Labs, USA*.
3. KNUTH, D. (1998) The art of computer programming. Volume 3: Sorting and Searching, Second Edition. *Addison-Wesley*.
4. CORMEN, T.H. & CORMEN, T.H. & LEISERSON, C.E. & RIVEST, R.L. & STEIN, C. (2001) Introduction to algorithms. *MIT Press*. p. 1180.
5. DEMUTH, H. (1956) Electronic data sorting. *PhD thesis. Stanford University*.
6. KRAK, I.U.V. & KOVAL, I.U.V. & STAVROVSKYI, A.B. (2015) Virtual process: definition and application for gestures interface system creation. In *Bulletin of Taras Shevchenko National University of Kyiv Series Physics & Mathematics*. Vol. 1. p.141-144.

Надійшла до редколегії 15.10.2018