Speed comparison of non-blocking list and global synchronized list

Abstract. *We tested the implementation of non-blocking linked lists described by Timothy L. Harris [1] on the MSIM simulator [2]. As the opponent in this comparison was choosen the global synchronized list – synchronization was in this case provided by mutex synchronization primitive.*

Introduction

The study of Timothy L. Harris shows, that non-blocking linked lists can in many cases replace clasical synchronized lists and to do so with a significant optimalization of executing speed. This optimalization is achieved by appropriate algoritm by using synchronization instruction CAS (compare-and-swap) [3]. We tried to find out, if it is possible to use this measure of optimalization also for the simulator MSIM [2]. In the environment of simulator is impossible to guarantee the constant time of execution of the instruction, because this time depends on the architecture of PC, on which the simulator is running. Unfortunately to this fact is this study limited and does not try to compare non-blocking linked lists objectively. It is only about its informative character.

First part is dedicated to the description of the architecture of PC, then comes the evaluation of both tested structures and special attention is devoted to the description of benchmark.

The other part consists of results of the test, speed comparison of individual operations and structures. Last part contains final evaluation of the study and objective appraisal of whole work.

The purpose of this study is also the opportunity of implementation of non-blocking linked lists in semestral project of the subject Operating systems at MFF UK, which takes place in the environment of simulator MSIM.

Parametres & Architecture

a) Architecture

Following section consists of information about architecture of PC, on which the MSIM was installed. Then the parametres of testing and other important information for evaluating the test are published.

The simulator MSIM was installed on PC Siemens-Fujitsu Lifebook S6410, on the procesor Intel Core Duo T7250 with operating memory 2 GB RAM and operating system Ubuntu 11.10 Oneiric Ocelot.

b) Structures

Non-blocking linked lists were implemented with the same interface and algorithm as presented the study of Timothy L. Harris [1]. Elements of the list were modified so that they can include the key and data variable. The list is internally sorted by the key variables.

For the structure of non-blocking linked lists was used the interface:

```
bool      nblock_list_insert (nblock_list_t* list, unsigned long key, void* data);
bool      nblock_list_remove (nblock_list_t* list, unsigned long search_key, void** data_ptr);
bool      nblock_list_find   (nblock_list_t* list, unsigned long search_key, void** data_ptr);
```

Global synchronized list was implemented by using mutex synchronization primitive. The interface was used similar as in the case of non-blocking linked lists, so that speeds of individual operations could be compared. In every operation that operates with the elements of the list, the global mutex is locked. That allows synchronization of paralel operations.

For the structure of global synchronized list was used the interface:

```
bool      sync_list_insert   (sync_list_t* list, unsigned long key, void* data);
bool      sync_list_remove   (sync_list_t* list, unsigned long search_key, void** data_ptr);
bool      sync_list_find     (sync_list_t* list, unsigned long search_key, void** data_ptr);
```

Benchmark

Benchmark was designed so that it can test all the operations included in the interface of both structures and to their speed comparison. Because possibilities of measuring the exact time are limited in MSIM only to unit of seconds (MSIM mediates only the time of hosting device), it is appropriate to design the test quite long, so that the results can be comparable.

The benchmark itself consists of six parts. First three of them test the speed of operations for elements with random generated keys. The other three parts test operations only for fixed keys. In the first and fourth part are the elements inserted to the list, the second and fifth is used for their research and finally the third and sixth for their removing.

In every test is first of all made the set number of threads and each of them makes the constant number of operations on the set shared list. The important condition is the fact, that all the threads start the test code at the same time. The time that runs from the start of the test to the final end of all threads is measured and recorded.

By using the modification of macros of the source files of benchmark it is possible to modify the number of operations for every thread, scale of generating of random numbers atc.

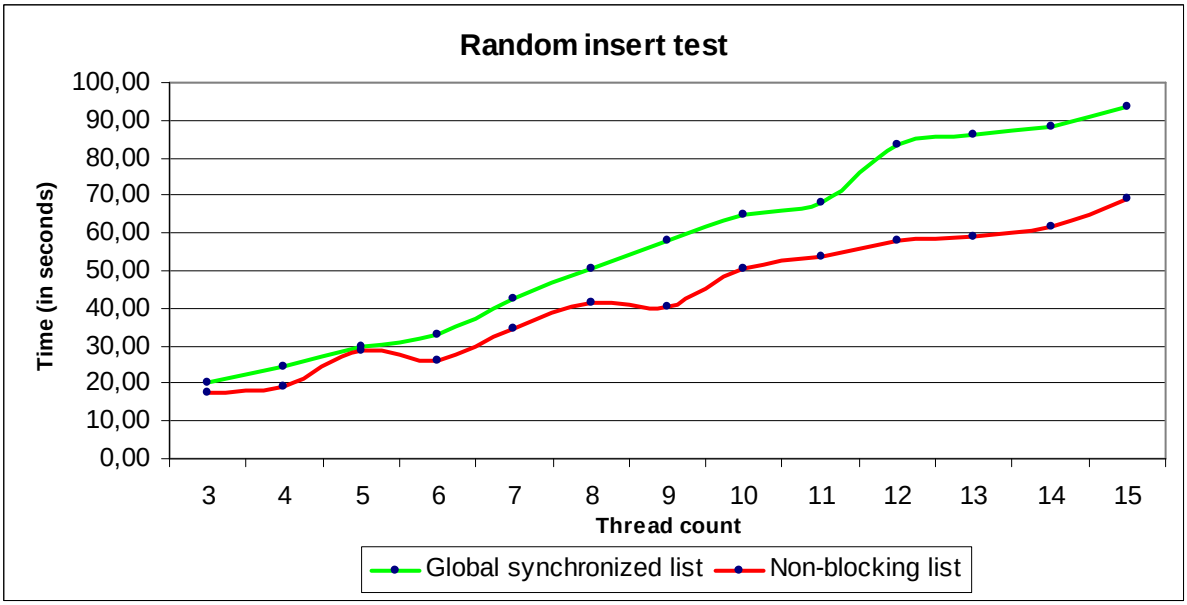
Results

In this section are presented the results for 3-15 threads with constant number of operations 10000 and the scale of generated keys 0-255. For the set number of threads was always made 30 tests and for the following graphs are used aritmetic averages and corresponding standard deviations.

For every test of benchmark is constructed a graph, which compares the time of the test for non-blocking and synchronized structure.

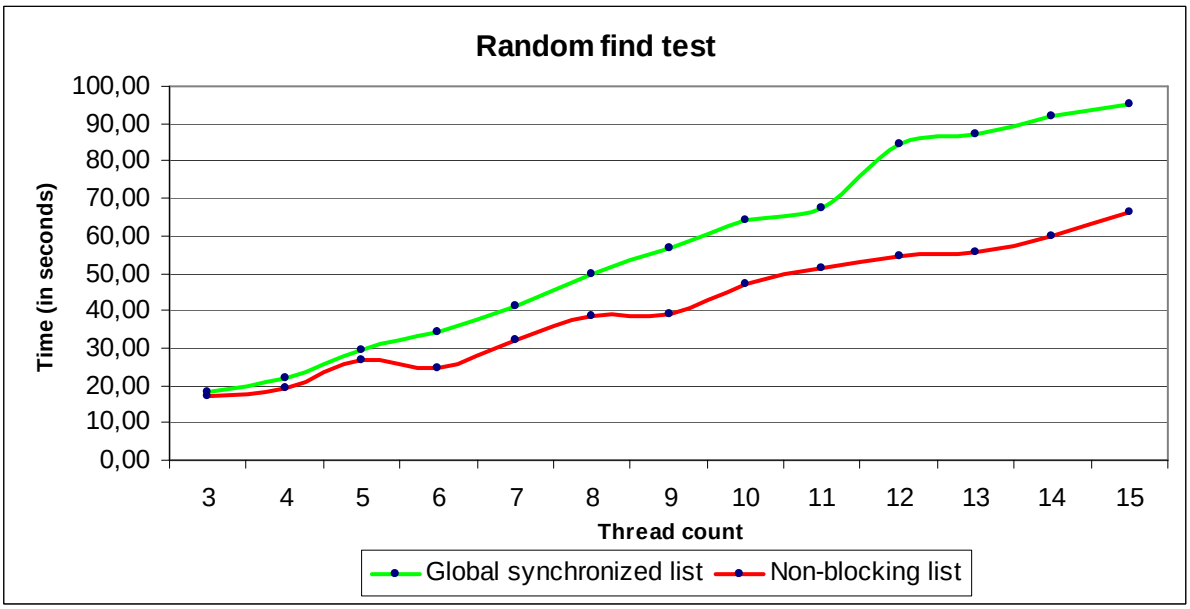
a) Random insert test

In this section is tested operation *insert* with random generated key elements in the range of 0-255. It is 10000 operation for every thread.



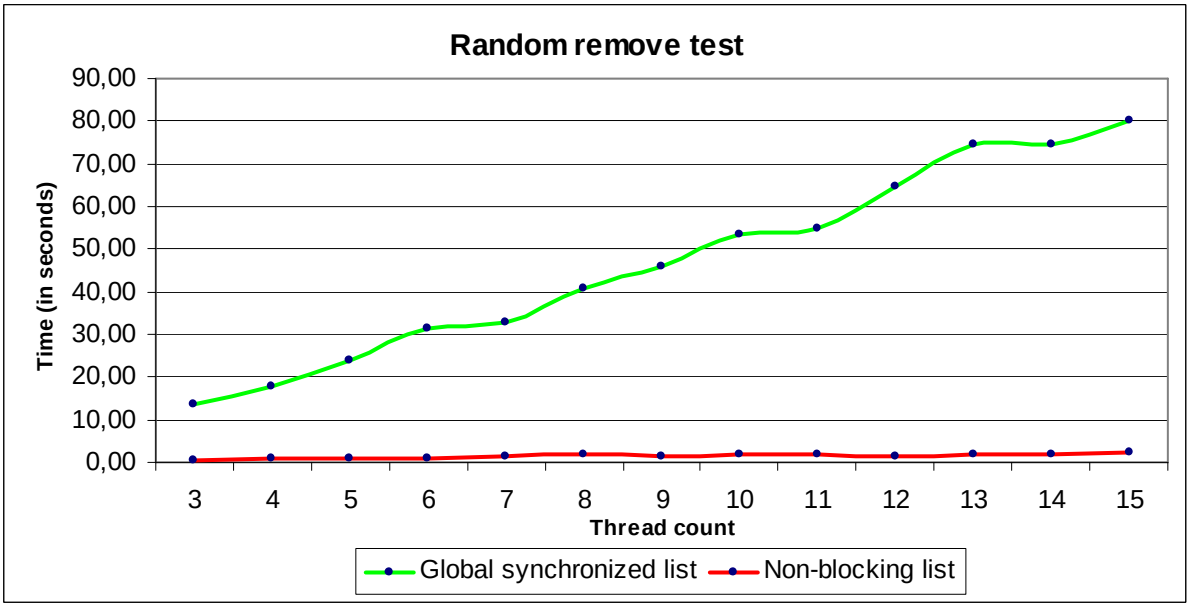
b) Random find test

In this section is tested operation *find* with keys generated in section a). The keys are used in the same order. It is 10000 operation for every thread.



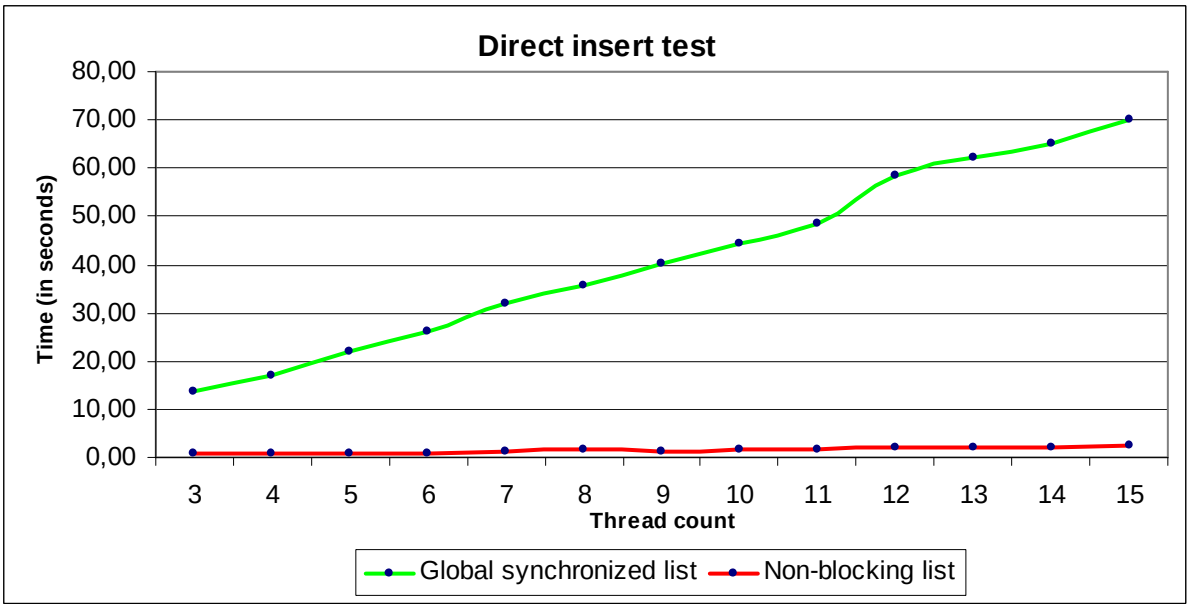
c) Random remove test

In this section is tested operation *remove* with keys generated in section a). The keys are used in the same order. It is 10000 operation for every thread.



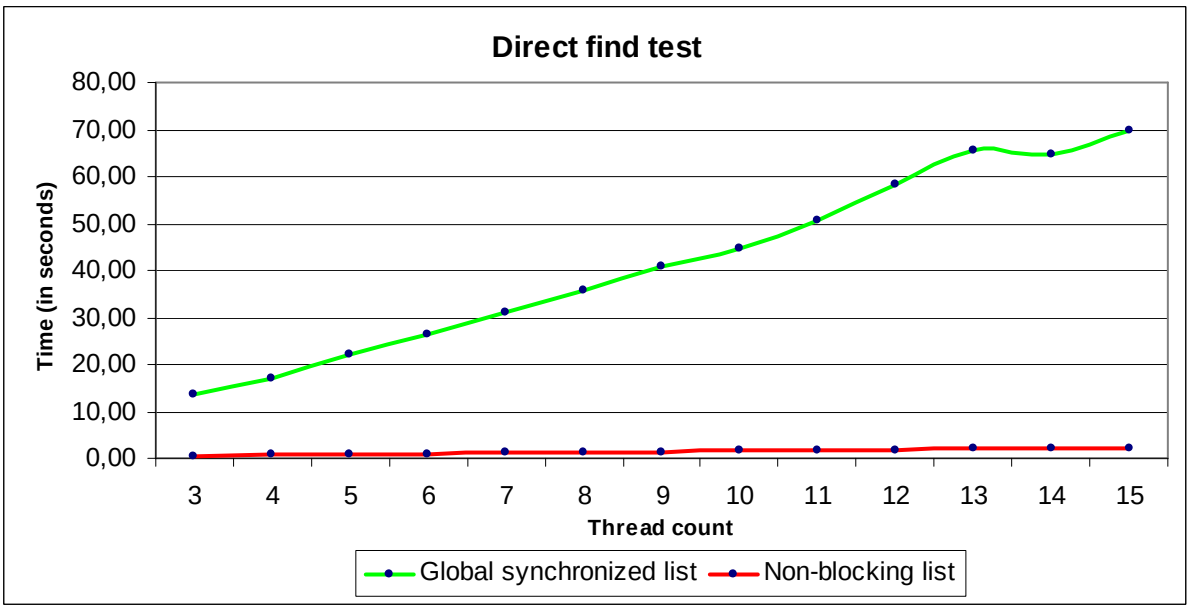
d) Direct insert test

In this section is tested operation *insert* with constant key. It is 10000 operation for every fibre.



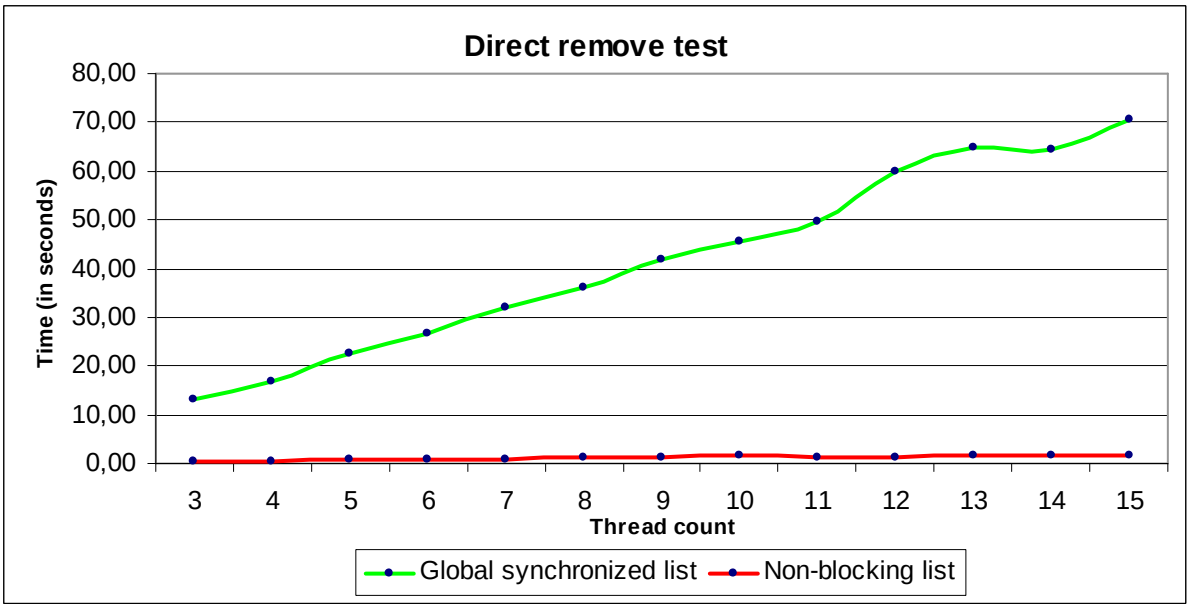
e) Direct find test

In this section is tested operation *find* with constant key. It is 10000 operation for every thread.



f) Direct remove test

In this section is tested operation *remove* with constant key. It is 10000 operation for every thread.



Conclusion

From stated results we can see, that with set parametres and number of threads are non-blocking linked lists more efective in executing speed.

For operations *random insert* and *random find* is speed difference noticeable, but not in the order of magnitude, while for operation random remove the difference in the order of magnitude is. This effect is made mainly by the fact, that non-blocking algoritms according to Timothy L. Harris use logical and physical deleting. Logical deleting is extremely fast, it is only a write operation to shared variable. Physical deleting is not required immediately during the operation remove of set element and because of that it can be postponed to later passage through the list.

Another interesting phenomena is testing of direct operations. The study shows, how is the locking by using synchronization primitives ineffective in the case of often busy memory locations.

Finally it should be noted that non-blocking algoritms use as a strong condition of their functioning single linked list, which limits the number of synchronized pointers to just one, while synchronization primitives are universal instrument and may be succesfully apply to whole scale of different tasks. It is also important to underline that the speed of these tests depends also on the architecture of host PC, which means that these results cannot be considered objective.

All results of the tests is appended in the CVS file app_res.csv.

References

[1] Harris T. L.: A Pragmatic Implementation of Non-blocking Linked-Lists, Proceedings of the 15th International Conference on Distributed Computing, LNCS 2180, Springer-Verlag, 2001
http://research.microsoft.com/en-us/um/people/tharris/papers/2001-disc.pdf

[2] MSIM - light-weight computer simulator based on MIPS R4000
http://d3s.mff.cuni.cz/~holub/sw/msim/

[3] CAS (compare-and-swap) instruction
http://en.wikipedia.org/wiki/Compare-and-swap