MINISTRY FOR EDUCATION AND SCIENCE OF RUSSIA

____

SAINT PETERSBURG ELECTROTECHNICAL UNIVERSITY

«LETI»

____

CONSTRUCTION AND OPTIMISATION THE ALGORITHME

REPORT ABOUT ONLY WAIT-FREE DATA STRUCTURES AND ALGORITME

4 pages

St. Petersburg / Санкт-Петербург
2021

CONTENT

INTRODUCTION

An algorithm is a finite and unambiguous sequence of operations or instructions for solving a problem. Having chosen the right types of data makes it possible to have a program, more readable self-documented car, easier to maintain and often faster, in any case easier to optimize.

We will insist in particular on two notions:

WHAT IS WAIT-FREE

- Let's we have some amount threads.
- At absolute any moment of time we stop random amount of working threads.
- Wait-free means that rest threads will not just complete their tasks but they guaranteed will do it in N steps.
- This N value not depends on amount of threads. The N value depend only on data structure size.

WHAT IS DATA STRUCTUTRE

- Method for storing and organizing data for ease of use access and modification
- A data structure groups together:
  - I a certain number of data to be managed
  - I a set of operations that can be applied to these data
- Data structure are not completely wait-free or lock-free
- Usually exits wait-free or lock-free operations on data structure which depends on purpose of the data structure.

- For example in lazy sync I can guarantee that searching always will be performed in linear time which depends only on data structure size and which not depends on amount of threads. This searching is wait-free.

- In non-blocking sync all operations are lock-free. Why are not wait-free? Because we cannot guarantee amount of steps that the thread will take to complete.
- If data structure size is more than machine word?
- We need to organize all data in structure
  struct S {
  …
  }
- And we will transfer in CAS(compare and set) pointer for struct item with guarantee that there is unambiguous correspondence between the structure and the pointer to it

- We need to use in CAS immutable structure

INTERPRETATION

- EXPLANATION

It was shown in the 1980s that all algorithms can be implemented wait-free, and many transformations from serial code, called *universal constructions*, have been demonstrated. However, the resulting performance does not in general match even naïve blocking designs. Several papers have since improved the performance of universal constructions, but still, their performance is far below blocking designs.

Several papers have investigated the difficulty of creating wait-free algorithms. For example, it has been shown that the widely available atomic *conditional* primitives, CAS(compare and set) and load-linked/store-conditional(LL/SC), cannot provide starvation-free implementations of many common data structures without memory costs growing linearly in the number of threads.

But in practice these lower bounds do not present a real barrier as spending a cache line or exclusive reservation granule (up to 2 KB on ARM) of store per thread in the shared memory is not considered too costly for practical systems (typically the amount of store logically required is a word, but physically CAS operations on the same cache line will collide, and LL/SC operations in the same exclusive reservation granule will collide, so the amount of store physically required

- PROBLEM

- First thread starts a change
- Another thread switches in
  - makes other changes but then switches something back
- First thread comeback
  - check the original condition which is true or incorrectly makes it changes

- PSEUDOCODE

```
slist() =default;
```

```
   ~slist() =default;

   class reference {
      shared_ptr<Node> p;
   public:
      reference(shared_ptr<Node> p_) : p{_p} {}
      T& operator*() { return p->t; }
      T* operator->() { return &p->t; }
   };
   auto find(T t) const {
      auto p = head.load();
      while (p && p->t != t)
         p = p->next;
      return reference{move(p)};

   void push_front(T t) {
      auto p = make_shared<Node>();
      p->t = t;
      p->next = head;
      while (head.compare_exchange_weak(p->next, p))
         {}
   }

   void pop_front() {
      auto p = head.load();
      while (p && !head.compare_exchange_weak(p, p->next))
         {}
   }
};
```

CONCLUSION

We find that lock-free and wait-free are related notions

A data structure is said to be lock-free if ***all*** its methods are lock-free (or wait-free)
A data structure is said to be wait-free if ***all*** its methods are wait-free