МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)
Кафедра ВТ

ОТЧЁТ
ЛАБОРАТОРНАЯ РАБОТА №1
по дисциплине «Параллельные алгоритмы и системы»
Тема: Блокировки

Студент гр. 9892 _____ Лескин К.А.

Преподаватель _____ Пазников А.А.

Санкт-Петербург
2022

1 Цель работы

Изучение алгоритмов блокировки в многопоточных системах.

2 Задание

1. Реализовать алгоритм блокировки потоков (языки: ‘C++/C‘ желательно, ‘Java‘ допускается, но хуже)
2. Разработать простой тест
3. Оценить эффективность, построить графы, проанализировать, сформулировать выводы.
4. Написать отчет с результатами экспериментов и выводами.

3 Выполнение задания

Работа выполнена на языке `C++14`.

Для тестирования блокировки был реализован класс `TestLock`. `TestLock` запускает несколько потоков, вызывающих определённую функцию.

Входные параметры:

- `name` – Имя блокировки (отображается в логах);
- `spins` – Количесво выполнений для нагрузочной функции (по умолчанию = 32768);
- `start` – Стартовое количество потоков (по умолчанию = 1);
- `end` – Конечное количество потоков (по умолчанию = максимальному количеству потоков на процессоре);
- `job` – Нагрузочная функция (по умолчанию = TestLock::spinner).

В цикле `TestLock` запускает $(start, start + 1, \ldots, end)$ потоков на нагрузочной функции и замеряет время выполнения. Результаты записываются в ассоциативный массив пар (кол-во потоков : время выполнения) и возвращаются на выходе.

Тестирование проводилось на `CPU` модели
`AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx` с 4 `CPUs`.

Тест (spins=4096)

Результаты тестирования представленны в таблице 1

Кол-во потоков	Время выполнения (мс)
1	0
2	1
3	1
4	5

Таблица 1 – Результаты тестирования для spins=4096

График результатов тестирования представлен на рис.1

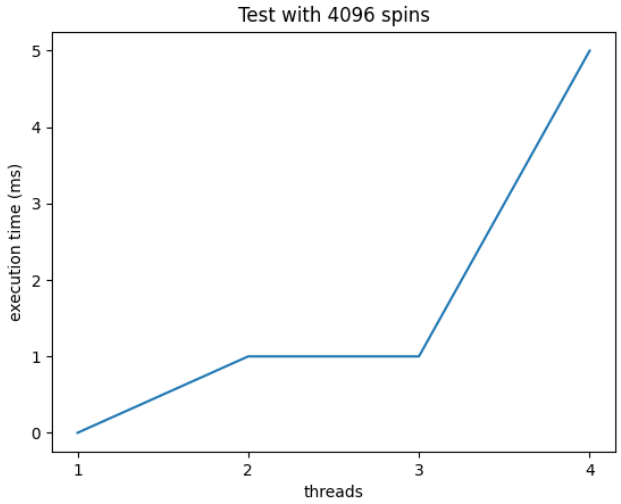


Рис. 1 – График результатов тестирования для spins=4096

Тест (spins=32768)

Результаты тестирования представленны в таблице 2

Кол-во потоков	Время выполнения (мс)
1	2
2	18
3	4
4	10

Таблица 2 – Результаты тестирования для spins=32768

График результатов тестирования представлен на рис.2

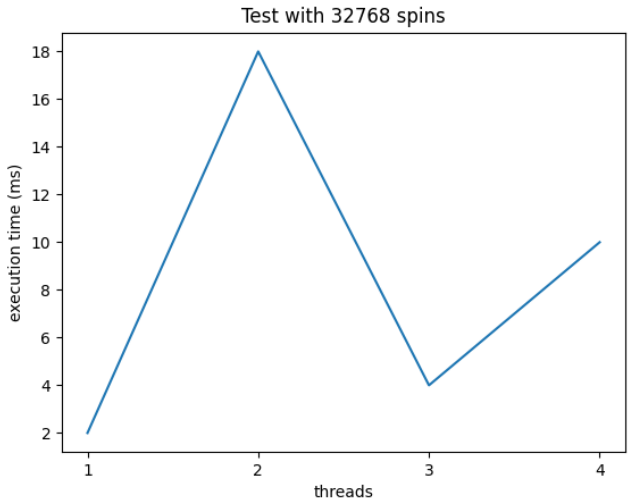


Рис. 2 – График результатов тестирования для spins=32768

Тест (spins=131072)

Результаты тестирования представленны в таблице 3

Кол-во потоков	Время выполнения (мс)
1	6
2	17
3	94
4	224

Таблица 3 – Результаты тестирования для spins=131072

График результатов тестирования представлен на рис.3

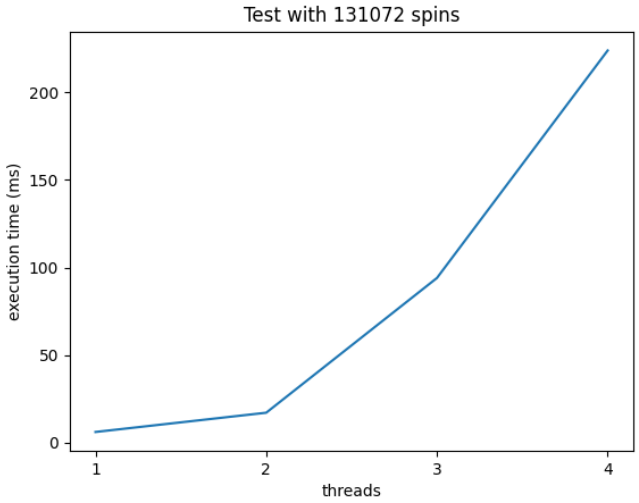


Рис. 3 – График результатов тестирования для spins=131072

Тест (spins=524288)

Результаты тестирования представленны в таблице 4

Кол-во потоков	Время выполнения (мс)
1	26
2	101
3	274
4	380

Таблица 4 – Результаты тестирования для spins=524288

График результатов тестирования представлен на рис.4

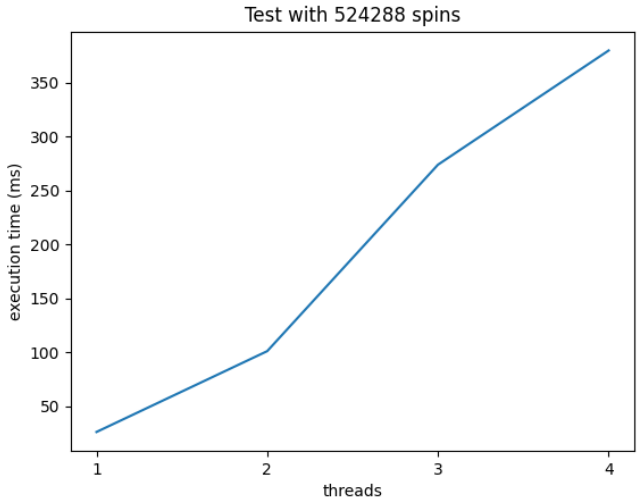


Рис. 4 – График результатов тестирования для spins=524288

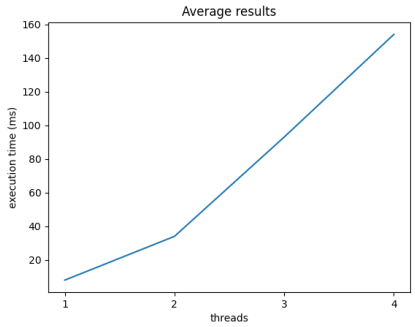
Усреднённые результаты тестирования

Усреднённые результаты тестирования представленны в таблице 5

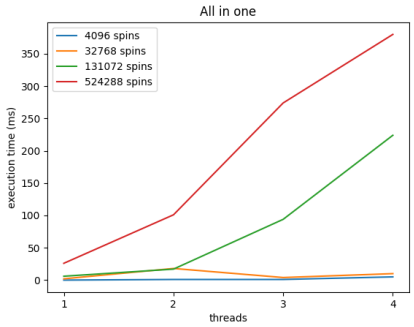
Кол-во потоков	Время выполнения (мс)
1	8
2	34
3	93
4	154

Таблица 5 – Усреднённые результаты тестирования

График усреднённых результатов тестирования представлен на рис.5а



(a) График усреднённых результатов тестирования



(b) Графики результатов тестирования для всех spins

Рис. 5 – Графики результатов тестирования

4 Выводы

В результате выполнения лабораторной работы был изучен и реализован `MCS` алгоритм блокировки потоков, проведены тесты на различном количестве потоков при варьирующейся нагрузке.

По результатам тестов можно сделать вывод, что распараллеливание некоторых простых задач не гарантирует рост производидельности, так как большая часть времени выполнения уходит на распределение ресурсов между потоками, а не на само выполнение задачи.

5 Приложение А – Листинг

```
#include <iostream>
#include <thread>
#include <chrono>
#include <map>
#include <vector>
#include <atomic>
#include <algorithm>
#include <functional>

#define LOG(msg) {std::cout << msg;} while(0)

#define DBG

#ifdef DBG
#define DEBUG(msg) {LOG(msg);} while(0)
#else
#define DEBUG(msg) {} while(0)
#endif


int getThreadsNumber()
{
    return static_cast<int>(std::thread::hardware_concurrency
   ());
}


template<typename Clock>
class Timer {
public:
    Timer() : _start_time(Clock::now()) {}
    Timer(const Timer&) = delete;
    Timer& operator=(const Timer&) = delete;
    int elapsed_ms()
    {
        return std::chrono::duration_cast<std::chrono::
   milliseconds>(Clock::now() - _start_time).count();
    }
private:
    std::chrono::time_point<Clock> _start_time;
};


class McsLock {
public:
    McsLock(const McsLock&) = delete;
    McsLock& operator=(const McsLock&) = delete;
```

```
46      McsLock(): _tail(nullptr) {}
47
48      class ScopedLock
49      {
50      public:
51          ScopedLock(const ScopedLock&) = delete;
52          ScopedLock& operator=(const ScopedLock&) = delete;
53          ScopedLock(McsLock &lock) : _lock(lock), _next(
    nullptr), _owned(false)
54          {
55              ScopedLock *tail = _lock._tail.exchange(this);
56              if (tail != nullptr)
57              {
58                  tail->_next = this;
59                  while (!_owned) {} /* spin */
60              }
61          }
62          ~ScopedLock()
63          {
64              ScopedLock *tail = this;
65              if (!_lock._tail.compare_exchange_strong(tail,
    nullptr))
66              {
67                  while (_next == nullptr) {} /* spin */
68                  _next->_owned = true;
69              }
70          }
71      private:
72          McsLock &_lock;
73          ScopedLock * volatile _next;
74          volatile bool _owned;
75      };
76  private:
77      std::atomic<ScopedLock *> _tail;
78  };
79
80
81  template<class mutex, class lock_object>
82  class TestLock
83  {
84  public:
85      explicit TestLock(
86          const std::string &name,
87          int spins = 1 << 15,
88          int start = 0,
89          int end = 0,
90          std::function<void(mutex *m, int *val, std::size_t n)
    > job = TestLock::spinner
91      )
```

```
92      {
93          _job = job;
94          _name = name;
95          _spins = spins > 0 ? spins : -spins;
96          if(1 <= start && start <= getThreadsNumber()) _start
    = start; else _start = 1;
97          if(_start <= end && end <= getThreadsNumber()) _end =
     end; else _end = getThreadsNumber();
98      }
99      std::map<int, int> run()
100     {
101         std::map<int, int> results;
102         for (int i = _start; i <= _end; ++i)
103         {
104             DEBUG(_name << ": Running test with " << i << "
    number of threads (");
105             int t = _testLock(i);
106             DEBUG("took: " << t << "ms)\n");
107             results[i] = t;
108         }
109         return results;
110     }
111     static void spinner(mutex *m, int *val, std::size_t n)
112     {
113         for (std::size_t i = 0; i < n; ++i)
114         {
115             lock_object lk(*m);
116             (*val)++;
117         }
118     }
119
120 private:
121     std::string _name;
122     std::function<void(mutex *m, int *val, std::size_t n)>
    _job = TestLock::spinner;
123     int _spins, _start, _end;
124
125     int _testLock(std::size_t threads_number) {
126         std::vector<std::thread> threads;
127         mutex m;
128         int i = 0;
129
130         Timer<std::chrono::high_resolution_clock> t;
131         DEBUG("spins: " << _spins << "; ");
132         for(int thread_num = 0; thread_num < threads_number;
    ++thread_num)
133         {
134             threads.emplace_back(_job, &m, &i, _spins);
135         }
```

```
136
137         std::for_each(threads.begin(), threads.end(), std::
    mem_fn(&std::thread::join));
138         return t.elapsed_ms();
139     }
140 };
141
142
143 int main()
144 {
145     DEBUG("This computer has " << getThreadsNumber() << "
    threads\n");
146     std::vector<int> times{1<<12, 1<<15, 1<<17, 1<<19};
147     for(auto t : times)
148     {
149         std::cout << t << std::endl;
150         TestLock<McsLock, McsLock::ScopedLock> tl("McsLock",
    t);
151         auto results = tl.run();
152     }
153     return 0;
154 }
```