

Санкт-Петербургский государственный политехнический университет
Институт Информационных Технологий и Управления
Кафедра компьютерных систем и программных технологий

Отчет по лабораторным работам
по предмету "Параллельные вычисления"

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА Б-ДЕРЕВЬЕВ

Работу выполнил студент гр. 53501/3 _____ Мартынов С. А.

Работу принял преподаватель _____ Моисеев М.Ю.

Санкт-Петербург
2014

Оглавление

Введение	3
Реализация однопоточного алгоритма	4
Многопоточные программирование с использованием Pthreads	11
Многопоточные программирование с использованием OpenMP	14
Многопоточные программирование с использованием C++11	16
Заключение	18
Список литературы	19

Введение

В последнее время тактовая частота центральных процессоров прекратила бурный рост, который наблюдался в последние десятилетия и производители сосредоточились на увеличении количества ядер. Это открыло новые возможности для разработчиков, т.к. позволило увеличить производительность программ не прибегая к механическому вертикальному масштабированию.

Воспользоваться преимуществами нескольких ядер относительно просто для серверных приложений, где каждый поток может независимо обрабатывать отдельный запрос от клиента, но этого значительно сложнее добиться для клиентских приложений, поскольку в этом случае обычно потребуется преобразовать код, интенсивно использующий вычисления и средства межпроцессной и межпоселковой синхронизации.

В данной работе рассмотрена работа со следующими инструментами:

1. Pthreads;
2. OpenMP;
3. C++11.

В качестве задачи было принято решение реализовать поточную обработку В-дерева, часто применяемого в базах данных.

Исходный код всех представленных листингов доступен по адресу
https://github.com/SemenMartynov/SPbPU_ParallelProgramming.

Реализация однопоточного алгоритма

Для начала была реализована однопоточная версия. Она состоит из двух классов BTree и BTreeNode. Заголовочные файлы этих классов представлены в листинге 1 и листинге 2. Реализацию можно изучить на гитхабе, ссылка была приведена во введении.

Листинг 1: заголовочный файл класса BTree

```
1 #ifndef TEST_BTREE_H_
2 #define TEST_BTREE_H_
3
4 #include "BTreeNode.h"
5
6 /**
7  * BTree
8  */
9 class BTree {
10 public:
11     /**
12      * @brief Constructor (Initializes tree as empty)
13      *
14      * @param t minimum degree
15      */
16     BTree(int t);
17
18     /**
19      * @brief Destructor
20      */
21     virtual ~BTree();
22     BTree(BTree&&) = delete;
23     BTree(const BTree&) = delete;
24     BTree& operator=(BTree&&) = delete;
25     BTree& operator=(const BTree&) = delete;
26
27     /**
28      * A function to traverse all nodes in a subtree
29      *
30      * @return
```

```

31     */
32     std::string to_str();
33
34     /**
35      * function to search a key in this tree
36      *
37      * @param key
38      * @return
39      */
40     bool exist(int key);
41
42     /**
43      * The main function that inserts a new key in this B-Tree
44      *
45      * @param key
46      * @return
47      */
48     bool insert(int key);
49
50     /**
51      * The main function that removes a new key in thie B-Tree
52      *
53      * @param key
54      * @return
55      */
56     bool remove(int key);
57
58 private:
59     const int t; /**< Minimum degree*/
60     BTreeNode *root; /**< Pointer to root node*/
61 };
62
63 #endif /* TEST_BTREE_H_ */

```

Листинг 2: заголовочный файл класса BTreeNode

```

1 #ifndef TEST_BTREENODE_H_
2 #define TEST_BTREENODE_H_
3
4 /**
5  * BTreeNode
6  */
7 class BTreeNode {
8 public:
9     /**

```

```

10  * @brief Constructor
11  *
12  * @param t minimum degree
13  * @param leaf
14  */
15  BTreeNode(int t, bool leaf);
16
17  /**
18   * @brief Destructor
19   */
20  virtual ~BTreeNode();
21  //virtual ~BTreeNode() = default;
22  BTreeNode(BTreeNode&&) = delete;
23  BTreeNode(const BTreeNode&) = delete;
24  BTreeNode& operator=(BTreeNode&&) = delete;
25  BTreeNode& operator=(const BTreeNode&) = delete;
26
27  /**
28   * A function to traverse all nodes in a subtree rooted with this node
29   *
30   * @return
31   */
32  std::string to_str();
33
34  /**
35   * A function to search a key in subtree rooted with this node.
36   *
37   * @param key
38   * @return NULL if key is not present
39   */
40  bool exist(int key);
41
42  /**
43   * A function that returns the index of the first key that is greater
44   * or equal to key
45   *
46   * @param key
47   * @return
48   */
49  int findKey(int key);
50
51  /**
52   * A utility function to insert a new key in the subtree rooted with
53   * this node. The assumption is, the node must be non-full when this
54   * function is called

```

```

55  *
56  * @param key
57  */
58  void insertNonFull(int key);
59
60  /**
61   * A utility function to split the child y of this node. i is index
62   * of y in child array C[]. The Child y must be full when this
63   * function is called
64   *
65   * @param i
66   * @param y
67   */
68  void splitChild(int i, BTreeNode *y);
69
70  /**
71   * A wrapper function to remove the key key in subtree rooted with
72   * this node.
73   *
74   * @param key
75   */
76  void remove(int key);
77
78  /**
79   * A function to remove the key present in idx-th position in
80   * this node which is a leaf
81   *
82   * @param idx
83   */
84  void removeFromLeaf(int idx);
85
86  /**
87   * A function to remove the key present in idx-th position in
88   * this node which is a non-leaf node
89   *
90   * @param idx
91   */
92  void removeFromNonLeaf(int idx);
93
94  /**
95   * A function to get the predecessor of the key- where the key
96   * is present in the idx-th position in the node
97   *
98   * @param idx
99   * @return

```

```

100     */
101     int getPred(int idx);
102
103     /**
104      * A function to get the successor of the key- where the key
105      * is present in the idx-th position in the node
106      *
107      * @param idx
108      * @return
109      */
110     int getSucc(int idx);
111
112     /**
113      * A function to fill up the child node present in the idx-th
114      * position in the C[] array if that child has less than t-1 keys
115      * @param idx
116      */
117     void fill(int idx);
118
119     /**
120      * A function to borrow a key from the C[idx-1]-th node and place
121      * it in C[idx]th node
122      *
123      * @param idx
124      */
125     void borrowFromPrev(int idx);
126
127     /**
128      * A function to borrow a key from the C[idx+1]-th node and place it
129      * in C[idx]th node
130      *
131      * @param idx
132      */
133     void borrowFromNext(int idx);
134
135     /**
136      * A function to merge idx-th child of the node with (idx+1)th child of
137      * the node
138      *
139      * @param idx
140      */
141     void merge(int idx);
142
143     /**
144      * Make BTree friend of this so that we can access private members of

```



```

145     * this class in BTree functions
146     */
147     friend class BTree;
148
149 private:
150     const int t; /**< Minimum degree (defines the range for number of keys)*/
151     BTreeNode** children; /**< An array of child pointers*/
152     int* keys; /**< An array of keys*/
153     int keysnum; /**< Current number of keys*/
154     bool leaf; /**< Is true when node is leaf. Otherwise false*/
155 };
156
157 #endif /* TEST_BTREENODE_H_ */

```

Корректность кода проверялась при помощи Google test framework (смотри листинг 3) и valgrind (смотри изображение 1).

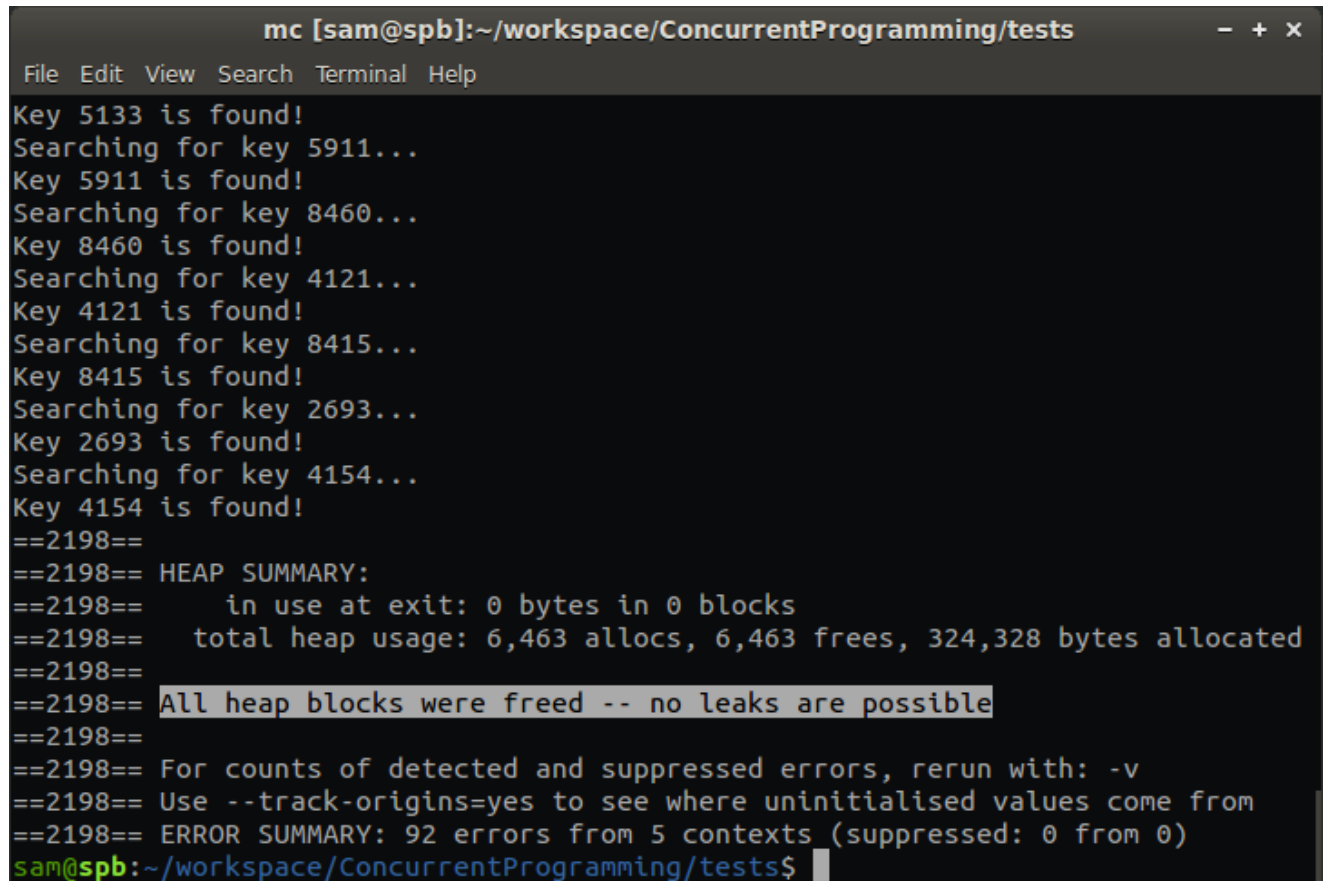
Листинг 3: Лог тестирование программы при помощи Google test framework

```

1 Running tests...
2 [=====] Running 8 tests from 3 test cases.
3 [-----] Global test environment set-up.
4 [-----] 4 tests from BTree3Test
5 [ RUN      ] BTree3Test.Add
6 [          OK ] BTree3Test.Add (0 ms)
7 [ RUN      ] BTree3Test.Remove
8 [          OK ] BTree3Test.Remove (1 ms)
9 [ RUN      ] BTree3Test.EmptyRemove
10 [          OK ] BTree3Test.EmptyRemove (0 ms)
11 [ RUN      ] BTree3Test.BigTest
12 [          OK ] BTree3Test.BigTest (0 ms)
13 [-----] 4 tests from BTree3Test (1 ms total)
14
15 [-----] 3 tests from BTree4Test
16 [ RUN      ] BTree4Test.BigTest
17 [          OK ] BTree4Test.BigTest (0 ms)
18 [ RUN      ] BTree4Test.EmptyRemove
19 [          OK ] BTree4Test.EmptyRemove (0 ms)
20 [ RUN      ] BTree4Test.RandomTest
21 [          OK ] BTree4Test.RandomTest (1 ms)
22 [-----] 3 tests from BTree4Test (1 ms total)
23
24 [-----] 1 test from BTree2Test
25 [ RUN      ] BTree2Test.BigTest
26 [          OK ] BTree2Test.BigTest (0 ms)
27 [-----] 1 test from BTree2Test (0 ms total)

```

```
28 |
29 | [-----] Global test environment tear-down
30 | [=====] 8 tests from 3 test cases ran. (2 ms total)
31 | [ PASSED ] 8 tests.
```



```
mc [sam@spb]:~/workspace/ConcurrentProgramming/tests
File Edit View Search Terminal Help
Key 5133 is found!
Searching for key 5911...
Key 5911 is found!
Searching for key 8460...
Key 8460 is found!
Searching for key 4121...
Key 4121 is found!
Searching for key 8415...
Key 8415 is found!
Searching for key 2693...
Key 2693 is found!
Searching for key 4154...
Key 4154 is found!
==2198==
==2198== HEAP SUMMARY:
==2198==    in use at exit: 0 bytes in 0 blocks
==2198==   total heap usage: 6,463 allocs, 6,463 frees, 324,328 bytes allocated
==2198==
==2198== All heap blocks were freed -- no leaks are possible
==2198==
==2198== For counts of detected and suppressed errors, rerun with: -v
==2198== Use --track-origins=yes to see where uninitialised values come from
==2198== ERROR SUMMARY: 92 errors from 5 contexts (suppressed: 0 from 0)
sam@spb:~/workspace/ConcurrentProgramming/tests$
```

Рис. 1: Поиск утечек памяти при помощи valgrind

Многопоточные программирование с использованием Pthreads

В той или иной форме, библиотека Pthread представлена на всех современных операционных системах. Она предоставляет низкоуровневый механизм управления процессами что даёт большую производительность но и требует больших трудозатрат. В листинге 4 один поток пополняет дерево, один удаляет элементы, и ещё несколько осуществляют поиск по дереву.

Листинг 4: Работа с B-деревом при помощи Pthreads

```
1 #include <iostream>
2 #include <numeric>
3 #include <algorithm>
4 #include <chrono>
5 #include <thread>
6 #include <pthread.h>
7 #include "BTree.h"
8
9 pthread_rwlock_t rwlock;
10 BTree tree4(4); // A B-Tree with minimum degree 4
11
12 std::vector<int> keys(1000); // vector with 10000 ints.
13
14 // random generator
15 std::default_random_engine generator;
16 std::uniform_int_distribution<int> distribution(0, 1000);
17
18 void* check(void* param) {
19     int tasks = *(int*) param;
20
21     for (int i = 0; i != tasks; ++i) {
22         pthread_rwlock_rdlock(&rwlock);
23         int key = distribution(generator);
24         std::cout << "Searching for key " << key << "..." << std::endl;
25         if (tree4.exist(key)) {
26             std::cout << "Key " << key << " is found!" << std::endl;
```

```

27     std::cout.flush();
28 }
29 pthread_rwlock_unlock(&rwlock);
30 std::this_thread::sleep_for(std::chrono::milliseconds(distribution(
    generator)/2));
31 }
32 return nullptr;
33 }
34
35 void* insert(void* param) {
36     // feel vector
37     std::for_each(keys.begin(), keys.end(), [&](int key) {
38         pthread_rwlock_wrlock(&rwlock);
39         tree4.insert(key);
40         pthread_rwlock_unlock(&rwlock);
41         std::this_thread::sleep_for(std::chrono::milliseconds(10));
42     });
43
44     return nullptr;
45 }
46
47 int main(int argc, char* argv[]) {
48     pthread_rwlock_init(&rwlock, nullptr);
49
50     std::iota(keys.begin(), keys.end(), 0); // Fill with 0, 1, ..., 9999.
51     std::random_shuffle(std::begin(keys), std::end(keys)); // the first shuffle
52
53     // threads
54     std::vector<pthread_t> threads(3);
55     pthread_t inserter;
56     int tasks = 10;
57
58     // starts threads
59     pthread_create(&inserter, NULL, insert, NULL);
60     std::this_thread::sleep_for(std::chrono::seconds(1));
61     for (auto thread : threads)
62         pthread_create(&thread, NULL, check, &tasks);
63
64     // clean vector
65     std::this_thread::sleep_for(std::chrono::seconds(2));
66     std::for_each(keys.begin(), keys.end(), [&](int key) {
67         pthread_rwlock_wrlock(&rwlock);
68         tree4.remove(distribution(generator));
69         pthread_rwlock_unlock(&rwlock);
70         std::this_thread::sleep_for(std::chrono::milliseconds(100));

```

```
71     });  
72  
73  
74     // wait for all threads to complete.  
75     for (auto thread : threads)  
76         pthread_join(thread, NULL);  
77     pthread_join(insertter, NULL);  
78  
79     pthread_rwlock_destroy(&rwlock);  
80     return 0;  
81 }
```

Многопоточные программирование с использованием OpenMP

Наиболее удобным средством для быстрого распараллеливания уже написанного кода является OpenMP. С другой стороны, его сложнее отлаживать и контролировать. К тому же, не являясь частью языка или библиотеки, OpenMP не поддерживается некоторыми статическими анализаторами на этапе разработки, что тоже создаёт неудобства. На листинге 5 представлена реализация задачи построения В-дерева с использованием OpenMP.

Листинг 5: Работа с В-деревом из OpenMP

```
1 #include <iostream>
2 #include <numeric>
3 #include <algorithm>
4 #include <chrono>
5 #include <thread>
6 #include <omp.h>
7 #include "BTree.h"
8
9 int main(int argc, char* argv[]) {
10
11     std::default_random_engine generator;
12     std::uniform_int_distribution<int> distribution(0, 10000);
13     BTree tree4(4); // A B-Tree with minium degree 4
14
15     std::vector<int> keys(10000); // vector with 10000 ints.
16     std::iota(keys.begin(), keys.end(), 0); // Fill with 0, 1, ..., 9999.
17
18     std::random_shuffle(std::begin(keys), std::end(keys)); // the first shuffle
19     int nthreads, tid;
20     /* Fork a team of threads with each thread having a private tid variable */
21     #pragma omp parallel private(tid)
22     {
23         /* Obtain and print thread id */
24         tid = omp_get_thread_num();
```

```

25     std::cout << "Thread " << tid << " active" << std::endl;
26
27     /* Only master thread does this */
28     if (tid == 0) {
29         nthreads = omp_get_num_threads();
30         std::cout << "Number of threads = " << nthreads << std::endl;
31         std::for_each(keys.begin(), keys.end(), [&tree4](int key) { // add
32             tree4.insert(key);
33         });
34     } else { // In other parallel threads
35 #pragma omp parallel for
36     for (int i = 0; i < 100; ++i) {
37         int key = distribution(generator);
38         std::cout << "#" << tid << " Searching for key " << key << "..."
39             << std::endl;
40         if (tree4.exist(key))
41             std::cout << "Key " << key << " is found!" << std::endl;
42         std::this_thread::sleep_for(
43             std::chrono::milliseconds(distribution(generator) / 2));
44     }
45 }
46
47 /* All threads join master thread and terminate */
48
49 std::random_shuffle(std::begin(keys), std::end(keys)); // the second
50 shufle
51 std::for_each(keys.begin(), keys.end(), [&tree4](int key) { // remove
52     tree4.remove(key);
53 });
54
55 return 0;
56 }

```

Многопоточные программирование с использованием C++11

Представленный в 2011-м году стандарт языка C++ получил поддержку потоков. По сути, это более высокоуровневая абстракция над библиотекой Pthreads, но работать становится на много приятнее за счёт использования таких механизмов как лямбды. Код в итоге получается удобно читаемым и компактным. На листинге 6 представлена работа с C++11.

Листинг 6: Потоки C++11

```
1 #include <iostream>
2 #include <vector>
3 #include <future>
4 #include <algorithm>
5 #include <chrono>
6 #include <thread>
7 #include <thread>
8 #include "BTree.h"
9
10 int main() {
11     std::default_random_engine generator;
12     std::uniform_int_distribution<int> distribution(0, 10000);
13     BTree tree4(4); // A B-Tree with minium degree 4
14
15     std::vector<int> keys(10000); // vector with 10000 ints.
16     std::iota(keys.begin(), keys.end(), 0); // Fill with 0, 1, ..., 9999.
17
18     std::random_shuffle(keys.begin(), keys.end()); // the first shuffle
19     std::for_each(keys.begin(), keys.end(), [&tree4](int key) { // add
20         tree4.insert(key);
21     });
22
23     std::cout << "Main thread id: " << std::this_thread::get_id() << std::endl
24         ;
25     std::vector<std::future<void>> futures;
26     for (int i = 0; i < 20; ++i) {
```



```

26     auto fut = std::async([&]
27     {
28         int key = distribution(generator);
29         std::cout << "Searching for key " << key << "..." << std::endl;
30         if (tree4.exist(key))
31             std::cout << "Key " << key << " is found!" << std::endl;
32     });
33     futures.push_back(std::move(fut));
34 }
35 std::for_each(futures.begin(), futures.end(), [](std::future<void> &fut)
36 {
37     fut.wait();
38 });
39
40 std::random_shuffle(std::begin(keys), std::end(keys)); // the second
    shuffle
41 std::for_each(keys.begin(), keys.end(), [&tree4](int key) { // remove
42     tree4.remove(key);
43 });
44
45 return 0;
46 }

```

Заключение

Многопоточность является естественным продолжением многозадачности, точно также как виртуальные машины, позволяющие запускать несколько ОС на одном компьютере, представляют собой логическое развитие концепции разделения ресурсов.

Уровень контроля над потоками в многопоточном приложении выше, чем уровень контроля приложения над дочерними процессами. Кроме того, многопоточные программы не склонны оставлять за собой вереницы зомби или «осиротевших» независимых процессов. Для отслеживания подобных ситуаций можно использовать специальные отладчики и средства, предоставляемые операционной системой.

Из инструментов, рассмотренных в данной работе наилучшее впечатление оставил о себе C++11. В C++11, работа с потоками осуществляется по средствам класса `std::thread` (доступного из заголовочного файла `<thread>`), который может работать с регулярными функциями, лямбдами и функторами. В данной работе не покрыты все возможности языка, вопрос требует дополнительного изучения.

Список литературы

1. Х.М. Дейтел, П.Дж. Дейтел, Д.Р. Чофнес. Операционные системы: Основы и принципы. Третье издание. Пер. с англ. - М.:ООО"Бинм-Пресс 2009 г. - 1024 с.
2. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления — СПб: БХВ-Петербург, 2002. — 608 с.
3. Немнюгин С., Стесик О. - Параллельное программирование для многопроцессорных вычислительных систем. - СПб. БХВ-Петербург, 2002. - 400с.
4. Робачевский А., Немнюгин С., Стесик О. - Операционная система UNIX, 2 изд., СПб: БХВ 2010.- 656с.
5. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М:ДМК-Пресс. 2010, -232с.
6. B. Nichols, D. Buttler, J.P. Farrell: Pthreads Programming - A POSIX Standard for Better Multiprocessing, O'Reilly, 1996.-288p.
7. B. Eckel. Thinking in Java (4th Edition). Prentice Hall, 2006.-1150p.
8. B. Goetz, T. Peierls. Java concurrency in practice. Addison-Wesley Professional, 2006, - 384p.
9. IEEE standard SystemC language reference manual, IEEE Std 1666, 2005
– <http://standards.ieee.org/getieee/1666/download/1666-2005.pdf>
10. Официальный сайт OpenMP – <http://openmp.org/wp/>
11. Message Passing Interface Forum – <http://www.mpi-forum.org/>