НИУ ВШЭ
Факультет компьютерных наук
Программная инженерия
Построение многопоточного приложения на примере задачи определения множества взаимнопростых чисел
Куприхин Дмитрий Алексеевич, БПИ199

Оглавление

1. Текст задания

Вариант 14. Определить множество индексов i, для которых A[i] и B[i] не имеют общих делителей (единицу в роли делителя не рассматривать). Входные данные: массивы целых положительных чисел A и B, произвольной длины ≥ 1000. Количество потоков является входным параметром.

2. Применяемый метод

Для решения этой задачи была выбрана модель итеративного параллелизма. Входные массивы разбиваются на равные блоки, количество которых равно количесту потоков. Так как вычисление НОДа двух чисел никак не зависит от других данных, в программе не возникнет ситуации неуправляемого изменения данных несколькими потоками.

3. Тестирование

Для тестирования были сгенерированы 4 файла, содержащие одинаковые элементы массивов и различающиеся количеством используемых потоков. Количество элементов во всех файлах равно 1e6. Количество потоков равно 1, 2, 4 и 8. Для тестирования и запуска программ использовался скрипт в командной оболочке Bash.

Результаты тестирования:

```
dima@dimaPC:~/comp_arch_hw4$ ./test
1 thread:
real    0m0,273s
user    0m0,261s
sys     0m0,012s
2 threads:
real    0m0,232s
user    0m0,273s
sys     0m0,008s
4 threads:
real    0m0,207s
user    0m0,274s
sys     0m0,008s
8 threads:
real    0m0,212s
user    0m0,271s
sys     0m0,008s
dima@dimaPC:~/comp_arch_hw4$
```

Приложение 1. Текст скрипта test

```
#!/bin/bash
g++ -o hw4 hw4.cpp -lpthread
printf "1 thread:"
time ./hw4 < inputs/input1.txt > outputs/output1.txt
printf "2 threads:"
time ./hw4 < inputs/input2.txt > outputs/output2.txt
printf "4 threads:"
time ./hw4 < inputs/input4.txt > outputs/output4.txt
printf "8 threads:"
time ./hw4 < inputs/input8.txt > outputs/output8.txt
```

Приложение 2. Текст файла hw4.cpp

```
#include <vector>
#include <iostream>
#include <pthread.h>

// Куприхин Дима, БПИ199, Вариант 14
// Задание: определить такие индексы, для которых a[i] и b[i] не
имееют общих
// делителей больших единицы, т.е. НОД(a[i], b[i]) = 1.
// Входные параметры: размер массивов (>=1000), количество
потоков,
// элементы массивов.

// Для вычисления НОДа будем использовать алгоритм Евклида.
Результаты будем
// сохранять в вектор, в котором значение элемента 1 значит
// взаимнопростость чисел с этим индексом, а 0 - НОД > 1. Каждый
поток
// записывает и читает из своего отдельной области, поэтому нам не
надо
// использовать мьютексы.

// Класс, хранящий ссылки на два массива, значения начала и конца
// обрабатываемого отрезка [start; end), ссылку на вектор с
результатом
// программы. Для каждого потока сконструирован специальный
экземпляр блока с
// соответствующими значениями концов обрабатываемого интервала.
class ArraysBlock {
 public:
    ArraysBlock(std::vector<int>& first, std::vector<int>& second,
int start,
                int end, std::vector<int>& numbers) :
            firstArray(first),
            secondArray(second),
            start(start),
            end(end),
            coprimeNumbers(numbers) {}

    std::vector<int>& firstArray;
    std::vector<int>& secondArray;
    int start;
    int end;
    std::vector<int>& coprimeNumbers;
};
```

```
void inputIntVector(std::vector<int>& vector) {
    std::cout << "Input " << vector.size() << " elements of
vector" << std::endl;
    for(int i = 0; i < vector.size(); ++i){
        std::cin >> vector[i];
    }
}

// Алгоритм Евклида для поиска НОД'а двух чисел.
int gcd(int a, int b){
    if(a == 0)
        return b;
    return gcd(b % a, a);
}

// Функция, которую выполняют потоки. Каждый поток обрабатывает
свой отдельный
// участок массивов и записывает результат в общий вектор.
void* countCoprime(void* args) {
    ArraysBlock* block = (ArraysBlock*)args;
    // Проходимся по всем индексам этого блока.
    for(int i = block->start; i < block->end; ++i){
        int g = gcd(block->firstArray[i], block->secondArray[i]);
        if(g == 1){
            block->coprimeNumbers[i] = 1;
        }
    }
    return 0;
}

int main() {
    std::ios_base::sync_with_stdio(0);
    std::cin.tie(0);
    std::cout.tie(0);

    // Вводим размеры массивов и количество потоков.
    int arraySize, threadsNumber;
    std::cout << "Input array size and number of threads" <<
std::endl;
    std::cin >> arraySize >> threadsNumber;

    // Инициализируем и заполняем массивы.
    // потоков и результата.
    std::vector<int> firstArray(arraySize);
    std::vector<int> secondArray(arraySize);
```

```
    inputIntVector(firstArray);
    inputIntVector(secondArray);
    std::vector<int> result(arraySize, 0);
    std::vector<pthread_t> threads(threadsNumber);
    // Инициализирум блоки. Каждому блоку выделяем равные(с
точностью
    // целочисленного деления) куски, которые будет обрабатывать
    // соответствующий поток.
    std::vector<ArraysBlock*> blocks(threadsNumber);
    for(int i = 0; i < threadsNumber; ++i){
        blocks[i] = new ArraysBlock(firstArray, secondArray,
                        arraySize / threadsNumber * i,
                        arraySize / threadsNumber * (i + 1),
result);
    }
    // Запускаем потоки и ждем их завершения.
    for(int i = 0; i < threadsNumber; ++i){
        pthread_create(&threads[i], NULL, countCoprime,
blocks[i]);
    }
    for(int i = 0; i < threadsNumber; ++i){
        pthread_join(threads[i], NULL);
    }
    // Считаем количество взаимнопростых чисел.
    int resultSize = 0;
    for(int i = 0; i < result.size(); ++i) {
        if(result[i]) {
            ++resultSize;
        }
    }
    // Выводим результат.
    std::cout << "There are " << resultSize <<
            " coprime numbers in the arrays:\n";
    for(int i = 0; i < result.size(); ++i) {
        if(result[i]) {
            std::cout << i << " ";
        }
    }
    // Освобождаем выделенную в куче память.
    for(int i = 0; i < blocks.size(); ++i)
        delete blocks[i];
    return 0;
}
```