2022.1 Multicore Computing, Project #1

Problem 1

Document

소프트웨어학부

20176342 송민준

1. **Result**
2. **Execution environment**

CPU : AMD Ryzen 5 2600X Six-Core Processor (12 CPUs), ~3.6GHz

Memory : DDR4 16384MB RAM

OS : Windows 10

1. **Tables and graphs**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exec time(ms) | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| Static (block) | 5973 | 4410 | 2636 | 1904 | 1592 | 1444 | 1302 | 1227 | 1191 | 1112 |
| Static (cyclic) | 5935 | 5936 | 3036 | 3054 | 1678 | 1912 | 1715 | 1315 | 1192 | 1101 |
| dynamic | 5923 | 3061 | 1631 | 1264 | 1117 | 1062 | 1054 | 1035 | 1052 | 1077 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Performance  (1/exec time) | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| Static (block) | 0.000167 | 0.000226 | 0.000379 | 0.000525 | 0.000628 | 0.000692 | 0.000768 | 0.000814 | 0.000839 | 0.000899 |
| Static (cyclic) | 0.000168 | 0.000168 | 0.000329 | 0.000327 | 0.000595 | 0.000523 | 0.000583 | 0.000760 | 0.000838 | 0.000908 |
| dynamic | 0.000168 | 0.000168 | 0.000168 | 0.000168 | 0.000168 | 0.000168 | 0.000168 | 0.000168 | 0.000168 | 0.000168 |

**(c) Explanation of results**

**First, In static (block) load balancing method** decreases execution time exponentially by increasing number of threads until almost 10 threads..

Because each thread doing a job(get a prime number in number range) same range.

For example, when there are threads 1 and 2, a range of 1 ~ 100000 and 100001 ~ 200000 is allocated, respectively.

However, in the function of finding prime numbers, the repetition statements are used to determine if they are divided by i from 1~n, so passing over large numbers by parameters takes longer to calculate.

텍스트이(가) 표시된 사진

자동 생성된 설명

Above screen capture shows the result of static(block) method using 2 threads.

Thread-0 is assigned work number range 1~100000, and

Thread-1 is assigned work number range 100001~200000.

Consequently, Thread-0 execution time is about a third of the execution time of thread 1.

I can calculate number of iteration when work number range was given.

1~100000 is (1+100000)\*(100000/2) = 5000050000

100001~200000 is (100001+200000)\*(100000/2) = 15000050000

15000050000 / 5000050000 = almost 3

The execution time of Thread-1 is three times the execution time of Thread-0.However, from the time when there were more than 10 threads, the execution time was hardly reduced.

More threads cause a lot of overhead for "context switching." Therefore, it is important to find the appropriate number of threads, and in the static block method, the value seems to be about 10 to 12.

**Second, In static (cyclic) load balancing method**

In cyclic method, if there are 4 threads, each threads assigned their job(get a prime number from number)

1,5,9,13,,,

2,6,10,14,,,

3,7,11,15,,,

4,8,12,16,,, respectively

Comparing when there are one thread and when there are two threads, the execution time is almost the same.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

The screenshot above is the result of executing when there are two threads and four threads.

When there are two threads, there is little execution time of thread 0. When there are four threads, there is little execution time for thread 0 and thread 2.

  private static boolean isPrime(int x){

    int i;

    if(x<=1) return false;

    for(i=2;i<x;i++){

      if(x%i == 0) return false;

    }

    return true;

  }

The reason lies in the internal logic of the isPrime function.

An even thread is always assigned an even number.

When an even number enters isPrime function as a parameter, the result of modular operation is always 0. And returns result immediately.( just 1 iteration only needed )

Therefore, in the case of 'static (cyclic) method', there is little difference between one thread and two threads.

Likewise, if there are too many threads, the context switching overhead increases and the execution time is no longer faster.

**Finally, In dynamic load balancing method**

  public void work(){

    while(true){

      lock.lock();

      if(index >= end){

        lock.unlock();

        break;

      }

      int curr = p[index++];

      lock.unlock();

      if( isPrime(p[curr]) ){

        primeCount++;

      }

    }

  }

Dynamic load balancing, unlike static load balancing, induces a natural state of competition among threads, so it seems that the balancing was performed most appropriately.

In dynamic load balancing, threads compete to lock and unlock by using lock and unlock of reentrant lock.

Unlike the static block method and the static cyclic method, there is no case in which an operation is advantageous only for a specific thread, and thus the result is the best compared to other load balancing methods.

Similarly, dynamic load balancing also increased context switching overhead when the number of threads was too high, and the execution time was no longer faster when the number of threads exceeded a certain number (approximately 10).

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

As shown in the picture above, it can be seen that the execution time for each thread is not biased.

1. **entire JAVA source code and screen capture image of program execution and output**

pc\_static\_block.java

import java.util.ArrayList;

import java.util.Arrays;

public class pc\_static\_block {

  private static int NUM\_END = 200000;

  private static int NUM\_THREADS = 1;

  public static void main (String[] args){

    if(args.length==2){

      NUM\_THREADS = Integer.parseInt(args[0]);

      NUM\_END = Integer.parseInt(args[1]);

    }

    int[] problem = new int[NUM\_END];

    for(int i = 0; i<NUM\_END;i++){

      problem[i] = i;

    }

    int counter = 0;

    long startTime = System.currentTimeMillis();

    ArrayList<BlockThread> thread\_arr = new ArrayList<BlockThread>();

    for(int i = 0; i<NUM\_THREADS;i++){

      int start = i\*(NUM\_END/NUM\_THREADS)+1;

      int end = i == NUM\_THREADS-1 ? NUM\_END : (i+1)\*(NUM\_END/NUM\_THREADS);

      System.out.println("new thread range "+start+ " ~ "+end);

      BlockThread a = new BlockThread(Arrays.copyOfRange(problem, start , end));

      thread\_arr.add(a);

      a.start();

    }

    for(int i = 0;i<thread\_arr.size();i++){

      try {

        thread\_arr.get(i).join();

      } catch (InterruptedException e) {

        e.printStackTrace();

      }

    }

    for(int i = 0;i<thread\_arr.size();i++){

      counter += thread\_arr.get(i).getResult();

    }

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

    System.out.println("Program Execution Time: "+timeDiff+"ms");

    System.out.println("1..."+(NUM\_END-1)+" prime# counter=" + counter);

  }

}

class BlockThread extends Thread {

  int[] problem;

  int primeCount = 0;

  long startTime = System.currentTimeMillis();

  BlockThread( int[] problem ){

    this.problem = problem;

  }

  public void run(){

    System.out.println(this.getName()+" start!");

    for(var i = 0; i<this.problem.length;i++){

      if(isPrime(this.problem[i])){

        primeCount++;

      }

    }

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

    System.out.println(this.getName()+" Execution Time: "+timeDiff+"ms");

  }

  public int getResult(){

    return this.primeCount;

  }

  private static boolean isPrime(int x){

    int i;

    if(x<=1) return false;

    for(i=2;i<x;i++){

      if(x%i == 0) return false;

    }

    return true;

  }

}

pc\_static\_cyclic.java

import java.util.ArrayList;

import java.util.Arrays;

public class pc\_static\_cyclic {

  private static int NUM\_END = 200000;

  private static int NUM\_THREADS = 1;

  public static void main (String[] args){

    if(args.length==2){

      NUM\_THREADS = Integer.parseInt(args[0]);

      NUM\_END = Integer.parseInt(args[1]);

    }

    int[] problem = new int[NUM\_END];

    for(int i = 0; i<NUM\_END;i++){

      problem[i] = i;

    }

    int counter = 0;

    long startTime = System.currentTimeMillis();

    ArrayList<CyclicThread> thread\_arr = new ArrayList<CyclicThread>();

    for(int i = 0; i<NUM\_THREADS;i++){

      CyclicThread a = new CyclicThread();

      thread\_arr.add(a);

    }

    int k = 0;

    while(k<=NUM\_END){

      thread\_arr.get(k%NUM\_THREADS).addWork(k++);

    }

    for(int i = 0;i<thread\_arr.size();i++){

      thread\_arr.get(i).start();

    }

    for(int i = 0;i<thread\_arr.size();i++){

      try {

        thread\_arr.get(i).join();

      } catch (InterruptedException e) {

        e.printStackTrace();

      }

    }

    for(int i = 0;i<thread\_arr.size();i++){

      counter += thread\_arr.get(i).getResult();

    }

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

    System.out.println("Program Execution Time: "+timeDiff+"ms");

    System.out.println("1..."+(NUM\_END-1)+" prime# counter=" + counter);

  }

}

class CyclicThread extends Thread {

  ArrayList<Integer> problem;

  int primeCount = 0;

  long startTime = System.currentTimeMillis();

  CyclicThread( ){

    this.problem = new ArrayList<Integer>();

  }

  public void addWork(int num){

    this.problem.add(num);

  }

  public void run(){

    System.out.println(this.getName()+" start! work size = "+this.problem.size());

    for(var i = 0; i<this.problem.size();i++){

      if(isPrime(this.problem.get(i))){

        primeCount++;

      }

    }

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

    System.out.println(this.getName()+" Execution Time: "+timeDiff+"ms");

  }

  public int getResult(){

    return this.primeCount;

  }

  private static boolean isPrime(int x){

    int i;

    if(x<=1) return false;

    for(i=2;i<x;i++){

      if(x%i == 0) return false;

    }

    return true;

  }

}

pc\_dynamic.java

import java.util.ArrayList;

import java.util.Arrays;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class pc\_dynamic {

  private static int NUM\_END = 200000;

  private static int NUM\_THREADS = 1;

  public static void main (String[] args){

    if(args.length==2){

      NUM\_THREADS = Integer.parseInt(args[0]);

      NUM\_END = Integer.parseInt(args[1]);

    }

    int[] problem = new int[NUM\_END];

    for(int i = 0; i<NUM\_END;i++){

      problem[i] = i;

    }

    int counter = 0;

    long startTime = System.currentTimeMillis();

    ArrayList<DynamicThread> thread\_arr = new ArrayList<DynamicThread>();

    final Lock lock = new ReentrantLock();

    for(int i = 0; i<NUM\_THREADS;i++){

      DynamicThread a = new DynamicThread(lock);

      thread\_arr.add(a);

    }

    DynamicThread.p = problem;

    DynamicThread.end = NUM\_END;

    for(int i = 0;i<thread\_arr.size();i++){

      thread\_arr.get(i).start();

    }

    for(int i = 0;i<thread\_arr.size();i++){

      try {

        thread\_arr.get(i).join();

      } catch (InterruptedException e) {

        e.printStackTrace();

      }

    }

    for(int i = 0;i<thread\_arr.size();i++){

      counter += thread\_arr.get(i).getResult();

    }

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

    System.out.println("Program Execution Time: "+timeDiff+"ms");

    System.out.println("1..."+(NUM\_END)+" prime# counter=" + counter);

  }

}

class DynamicThread extends Thread {

  static ArrayList<Integer> problem = new ArrayList<Integer>();

  static int[] p;

  static int index = 0;

  static int end = 0;

  int primeCount = 0;

  long startTime = System.currentTimeMillis();

  private Lock lock;

  DynamicThread( Lock lock ){

    this.lock = lock;

  }

  public static void addWork(int num){

    problem.add(num);

  }

  public void work(){

    while(true){

      lock.lock();

      if(index >= end){

        lock.unlock();

        break;

      }

      int curr = p[index++];

      lock.unlock();

      if( isPrime(p[curr]) ){

        primeCount++;

      }

    }

  }

  public int getWork(){

    lock.lock();

    if(problem.size()>0){

        int val;

        val = problem.get(0);

        problem.remove(0);

        lock.unlock();

        return val;

    }

    else {

      System.out.println(this.getName()+" empty");

    }

    lock.unlock();

    return -1;

  }

  public void run(){

    work();

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

    System.out.println(this.getName()+" Execution Time: "+timeDiff+"ms");

  }

  public int getResult(){

    return primeCount;

  }

  private boolean isPrime(int x){

    int i;

    if(x<=1) return false;

    for(i=2;i<x;i++){

      if(x%i == 0) return false;

    }

    return true;

  }

}

**PC\_static\_block.java**

pc\_static\_block thread #1

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #2

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #4

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #6

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #8

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #10

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #12

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #14

텍스트이(가) 표시된 사진

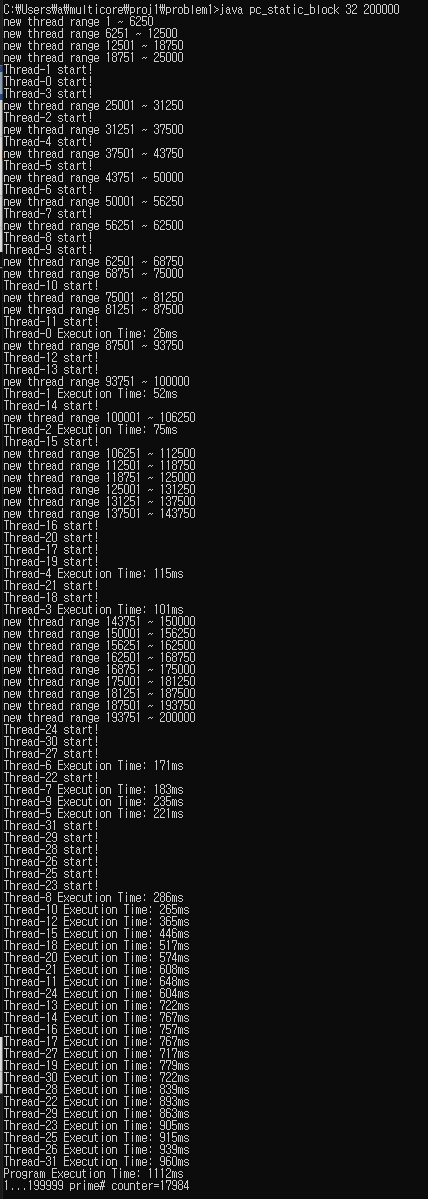
자동 생성된 설명

pc\_static\_block thread #16

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_block thread #32



**PC\_static\_cyclic.java**

pc\_static\_cyclic thread #1

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #2

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #4

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #6

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #8

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #10

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #12

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #14

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #16

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic thread #32

텍스트이(가) 표시된 사진

자동 생성된 설명

**pc\_dynamic.java**

pc\_dynamic thread #1

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #2

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #4

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #6

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #8

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #10

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #12

텍스트이(가) 표시된 사진

자동 생성된 설명  
pc\_dynamic thread #14

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #16

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic thread #32

텍스트이(가) 표시된 사진

자동 생성된 설명

**How to compile and execute**

텍스트이(가) 표시된 사진

자동 생성된 설명

Just use ‘javac’ to compile in my directory and run ‘java pc\_dynamic #num\_thread #num\_range’ like below

테이블이(가) 표시된 사진

자동 생성된 설명