2022.1 Multicore Computing, Project #1

Problem1

Document

소프트웨어학부

20176342 송민준

1. **JAVA source code**

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);

    }

    int k = 0;

    while(k<=NUM\_END){

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

      DynamicThread.addWork(k++);

    }

    System.out.println(DynamicThread.problem.size());

    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 DynamicThread extends Thread {

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

  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 synchronized void work(){

    int currWork = 0;

    while(true){

      currWork=getWork();

      if(currWork == -1){

        break;

      }

      if( isPrime(currWork) ){

        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(){

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

    work();

    long endTime = System.currentTimeMillis();

    long timeDiff = endTime - startTime;

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

  }

  public int getResult(){

    // System.out.println(this.getName()+ " calculated prime count= "+primeCount);

    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;

  }

}

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 | 7944 | 4501 | 3089 | 2746 | 2778 | 2753 | 2733 | 2817 | 2717 | 2746 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 0.000125 | 0.000222 | 0.000323 | 0.000364 | 0.000359 | 0.000363 | 0.000365 | 0.000354 | 0.000368 | 0.000364 |

1. Explanation of results

First, static block load balancing method

pc\_static\_block thread #1

텍스트이(가) 표시된 사진

자동 생성된 설명

Pc\_static\_block thread 2

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

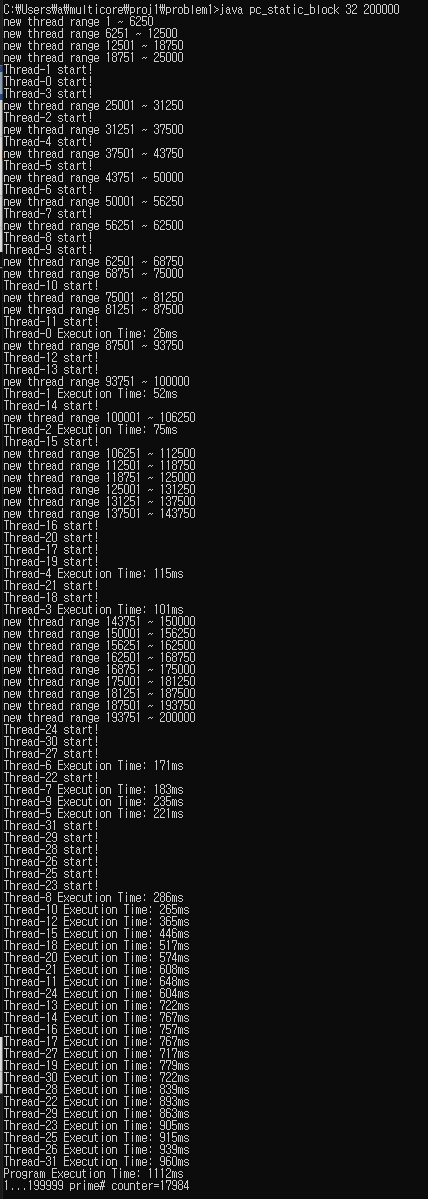
자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명



PC\_static\_cyclic.java

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Pc\_dynamic.java

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

