2022-01 Multicore Computing

Project 1 Problem 1

**Getting Prime Number Using Multi Thread,**

**Analysis on Performance By Distribution Methods**

20172848 정석우 Jeong Seok Woo

**1. source code**

pc\_static\_block.java

package proj1.problem1;

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 counter = 0;

        int i;

        BlockThread.blockSize = NUM\_END / NUM\_THREADS;

        if (BlockThread.blockSize \* NUM\_THREADS != NUM\_END) counter = 1;

        long startTime = System.currentTimeMillis();

        BlockThread[] threadArr = new BlockThread[NUM\_THREADS];

        for (i=1; i<(1+NUM\_THREADS); i++) {

            threadArr[i-1] = new BlockThread(i);

            threadArr[i-1].start();

        }

        try {

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

                threadArr[i].join();

                counter += threadArr[i].getCnt();

            }

        }

        catch (InterruptedException e) {

            e.printStackTrace();

        }

        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);

    }

    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;

    }

    static class BlockThread extends Thread {

        static int blockSize;

        private int endNum;

        private int cnt = 0;

        BlockThread(int i) {

            this.endNum = i \* blockSize;

        }

        @Override

        public void run() {

            long startTime = System.currentTimeMillis();

            for (int i = endNum-blockSize; i<endNum; i++) {

                if(pc\_static\_block.isPrime(i)) cnt++;

            }

            long endTime = System.currentTimeMillis();

            long timeDiff = endTime - startTime;

            System.out.println("[Thread " + endNum/blockSize + "] execution time : " + timeDiff + " ms");

        }

        public int getCnt() {

            return this.cnt;

        }

    }

}

How to compile this code : enter `javac proj1/problem1/pc\_static\_block.java`

How to execute this code : `java proj1/problem1/pc\_static\_block (thread number) (end number)`

pc\_static\_cyclic.java

package proj1.problem1;

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 counter = 0;

        int i;

        CyclicThread.threadSize = NUM\_THREADS;

        CyclicThread.limit = NUM\_END;

        CyclicThread[] threadList = new CyclicThread[NUM\_THREADS];

        long startTime = System.currentTimeMillis();

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

            threadList[i] = new CyclicThread(i);

            threadList[i].start();

        }

        try {

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

                threadList[i].join();

                counter += threadList[i].getCnt();

            }

        }

        catch (InterruptedException e) {

            e.printStackTrace();

        }

        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);

    }

    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;

    }

    static class CyclicThread extends Thread {

        private int primeCnt = 0;

        private int threadNum = 0;

        static int limit = 0;

        static int threadSize = 0;

        CyclicThread(int residual) {

            this.threadNum = residual;

        }

        @Override

        public void run() {

            long startTime = System.currentTimeMillis();

            for (int i = threadNum; i<limit; i+=threadSize) {

                if (pc\_static\_cyclic.isPrime(i)) primeCnt++;

            }

            long timeDiff = System.currentTimeMillis() - startTime;

            System.out.println("[Thread " + threadNum + "] execution time : " + timeDiff + " ms");

        }

        public int getCnt () {

            return this.primeCnt;

        }

    }

}

How to compile this code : enter `javac proj1/problem1/pc\_static\_cyclic.java`

How to execute this code : `java proj1/problem1/pc\_static\_cyclic (thread number) (end number)`

pc\_dynamic.java

package proj1.problem1;

import java.util.List;

import java.util.ArrayList;

import java.util.Collections;

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 counter = 0;

        int i;

        DynamicThread.t = NUM\_THREADS \* 2 - 1;

        DynamicThread.endNum = NUM\_END;

        DynamicThread.primeNumList.add(2);

        DynamicThread[] threadList = new DynamicThread[NUM\_THREADS];

        long startTime = System.currentTimeMillis();

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

            threadList[i] = new DynamicThread(i);

            threadList[i].start();

        }

        try {

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

                threadList[i].join();

            }

            counter = DynamicThread.primeNumList.size();

        }

        catch (InterruptedException e) {

            e.printStackTrace();

        }

        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);

    }

    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;

    }

    static class DynamicThread extends Thread {

        private int startNum, threadNum;

        static int endNum, t;

        long startTime, timeDiff;

        static List<Integer> primeNumList = Collections.synchronizedList(new ArrayList<Integer>());

        DynamicThread(int threadNum) {

            this.startNum = threadNum \* 2 + 1;

            this.threadNum = threadNum;

        }

        @Override

        public void run() {

            startTime = System.currentTimeMillis();

            while(t < endNum) {

                if (isPrime(startNum)) {

                    primeNumList.add(startNum);

                };

                startNum = increment();

            }

            timeDiff = System.currentTimeMillis() - startTime;

            System.out.println("[Thread " + threadNum + "] execution time : " + timeDiff + " ms");

        }

        static synchronized int increment() {

            t++;

            return t;

        }

    }

}

How to compile this code : enter `javac proj1/problem1/pc\_dynamic.java`

How to execute this code : `java proj1/problem1/pc\_dynamic (thread number) (end number)`

**2. Execution Picture**

pc\_static\_block

number of threads = 1

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 2

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 4

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 6

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 8

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 10

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 12

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 14

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 16

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 32

테이블이(가) 표시된 사진

자동 생성된 설명

pc\_static\_cyclic

number of threads = 1

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 2

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 4

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 6

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 8

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 10

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 12

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 14

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 16

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 32

텍스트이(가) 표시된 사진

자동 생성된 설명

pc\_dynamic

number of threads = 1

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 2

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 4

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 6

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 8

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 10

텍스트, 화면이(가) 표시된 사진

자동 생성된 설명

number of threads = 12

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 14

텍스트이(가) 표시된 사진

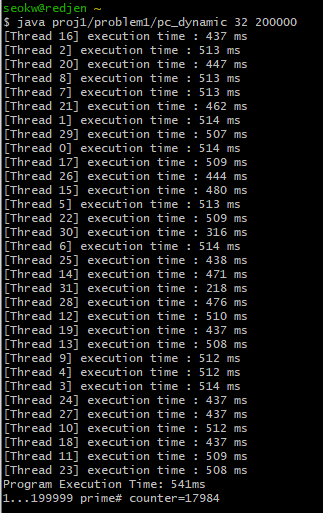
자동 생성된 설명

number of threads = 16

텍스트이(가) 표시된 사진

자동 생성된 설명

number of threads = 32



**3. performance**

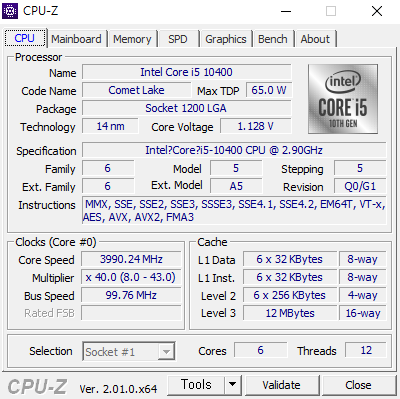
3-1. Execution Time By Number of Threads

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exec time | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| static (block) | 3521 | 2537 | 1526 | 1096 | 914 | 806 | 716 | 704 | 650 | 572 |
| static (cyclic) | 3449 | 3525 | 1934 | 2046 | 1099 | 1131 | 1149 | 808 | 671 | 603 |
| dynamic | 3439 | 1767 | 961 | 723 | 640 | 685 | 529 | 529 | 528 | 541 |

3-2. Performance By Number of Threads

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| performance | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
| static (block) | 0.000284 | 0.000394 | 0.000655 | 0.000912 | 0.001094 | 0.001241 | 0.001397 | 0.00142 | 0.001538 | 0.001748 |
| static (cyclic) | 0.00029 | 0.000284 | 0.000517 | 0.000489 | 0.00091 | 0.000884 | 0.00087 | 0.001238 | 0.00149 | 0.001658 |
| dynamic | 0.000291 | 0.000566 | 0.001041 | 0.001383 | 0.001563 | 0.00146 | 0.00189 | 0.00189 | 0.001894 | 0.001848 |

**4. Execution Environment**



This project was executed on 6 CPU, 3990MHz, 12 Threads, DDR4 16GB RAM, Windows 10 environment.

**5. Result Analysis**

First, in the case of static (block), the execution time decreased logistically as the number of threads increased. When counting a few using 32 threads, the efficiency increased by 3521/572 = about 6.15 times compared to when using 1 thread.

In the case of static (cyclic), the count of prime number was measured by determining whether (i \* number of threads + order of these threads) becomes prime when i increases from zero.

However, contrary to expectations, workload distribution using multi-threads was not properly carried out. This is because even numbers other than 2 cannot be prime numbers. When determining whether it is a prime number or divided into two, the even number-threads ended much faster when only even number of threads were executed in this project. Therefore, the execution performance graph when the workload was distributed by cyclic method showed a pattern of increasing stepwise as the number of threads used increased by 2. In cyclic, when 32 threads were used, 3449 / 603 = about 5.72 times faster than when 1 thread was used.

In the case of dynamic, the prime number was collected by adding multiple threads to the primeNumList, which is a shared resource, when the current number is a prime number. Even in the dynamic method, as the number of threads used for calculation increased, the execution time generally tended to be shorter, but this was not always the case. Because of the overhead required to execute each thread and control shared resources, the execution time was slower when using 32 threads than when using 16 threads.

The most excellent method of performance was the dynamic method. The next was the block method, and the slowest was the cyclic method. However, when the number of threads used in all three methods became 32, similar execution times were shown. In the dynamic method, it is not possible to know what number to determine in each thread in advance. Even though the number of numbers allocated to each thread was unbalanced, the distribution of execution time between each thread was much more uniform, which consequently made the dynamic method the fastest. In addition, since all numbers from 1 to 200,000 are subject to the isPrime() method, which is a prime number decision function once, it can be assumed that other operations are minimized the fastest.

The reason why the Cyclic method is slower than the block method is that, as described above, the Cyclic method evenly distributes the same number of numbers to each thread, but due to the specificity of prime number judgment, the computational number distributed to a specific thread varies. This can be seen from the maximum difference in execution time between the cyclic method and the block method when using 6 threads. The use of six threads in the cyclic method results in decimals for 6k, 6k+1, 6k+2, 6k+3, 6k+4, and 6k+5, but since 6k, 6k+2, and 6k+4 are even numbers, these numbers end at 2, and 6k+3 ends at 3. Therefore, it is concluded that cyclic's 6-thread operation is not different from the result of using two threads.