텍스트이(가) 표시된 사진

자동 생성된 설명



**20185999 임태규**

1. **Execution environment**

* **CPU Type: Intel ® CORE™ i5 – 1135G7**
* **Number of cores: 4 cores**
* **Memory size: 8GB**
* **OS Type: Windows**
* **IDE: Visual Studio Code**

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **Execution Result**
2. **Execution Time**

* **Graph/Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| exec time | 1 | 2 | 4 | 6 | 8 |
| static(block) | 2517 | 1935 | 1238 | 1015 | 868 |
| static (cyclic) [task size : 10 numbers] | 2519 | 1394 | 889 | 938 | 694 |
| dynamic [task size : 10 numbers] | 2516 | 1382 | 877 | 730 | 698 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| exec time | 10 | 12 | 14 | 16 | 32 |
| static(block) | 806 | 768 | 739 | 761 | 727 |
| static (cyclic) [task size : 10 numbers] | 700 | 738 | 691 | 692 | 721 |
| dynamic [task size : 10 numbers] | 688 | 691 | 717 | 685 | 688 |

1. **Performance**

* **Graph/Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Performance | 1 | 2 | 4 | 6 | 8 |
| static(block) | 0.000397 | 0.000517 | 0.000808 | 0.000985 | 0.001152 |
| static (cyclic) [task size : 10 numbers] | 0.000397 | 0.000717 | 0.001125 | 0.001066 | 0.001441 |
| dynamic [task size : 10 numbers] | 0.000397 | 0.000724 | 0.00114 | 0.00137 | 0.001433 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Performance | 10 | 12 | 14 | 16 | 32 |
| static(block) | 0.001241 | 0.001302 | 0.001353 | 0.001314 | 0.001376 |
| static (cyclic) [task size : 10 numbers] | 0.001429 | 0.001355 | 0.001447 | 0.001445 | 0.001387 |
| dynamic [task size : 10 numbers] | 0.001453 | 0.001447 | 0.001395 | 0.00146 | 0.001453 |

1. **Explanation / Analysis**

Please look at the table and graph above. Although the execution times are not significantly different and have nearly approximate values in three methods, generally dynamic method slightly better in terms of performance. I think this is because in the case of static block and cyclic methods, they are not divided into significantly efficient methods in terms of load balancing. So, the advantage of gaining by performing the next task rather than waiting for other threads to finish their work, is greater than the overhead generated by dynamically allocating tasks.

Furthermore, looking at the correlation between the number of threads and execution time/performance, initially, as the number of threads increases from one to two, and from two to four…six…eight, the performance also shows a rapid increase. However, after a certain number, the rate of increase decreases or is almost nonexistent, and sometimes even decreases. This shows that if more than a certain number of threads are used, the operations that need to be performed, i.e., overhead, become greater than the benefits obtained through the number of threads.

1. **Source Code & Output**
2. **Static Block Thread**
   1. **Code**

public class pc\_static\_block {

    private static int NUM\_END = 200000;

    private static int[] THREAD\_NUMS = { 1, 2, 4, 6, 8, 10, 12, 14, 16, 32 };

    public static void main(String[] args) throws InterruptedException {

        if (args.length == 2) {        // 인자를 정하면 해당 쓰레드 개수로 실행

            THREAD\_NUMS = new int[1];

            THREAD\_NUMS[0] = Integer.parseInt(args[0]);

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

        }             // args가 없으면 사전에 정의된 THREAD\_NUMS로 for문을 돌며

        for (int i = 0; i < THREAD\_NUMS.length; i++) {// 1~32개의 쓰레드 개수로   
한 번씩 실행

            int threadNum = THREAD\_NUMS[i];

            static\_block\_Thread.counter = 0;

            static\_block\_Thread[] thread = new static\_block\_Thread[threadNum];

            for (int t = 0; t < threadNum; t++) {

                thread[t] = new static\_block\_Thread(t, threadNum);

            }

            long startTime = System.currentTimeMillis();

            for (int t = 0; t < threadNum; t++) {

                thread[t].start();

            }

            for (int t = 0; t < threadNum; t++) {

                thread[t].join();

            }

            long endTime = System.currentTimeMillis();

            long timeDiff = endTime - startTime;

            System.out.println(threadNum + " Thread\n");

            for (int t = 0; t < threadNum; t++) {

                System.out.println("Thread " + t + " Execution Time: " + thread[t].timeDiff + "ms");

            }

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

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

            System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        }

    }

    private static boolean isPrime(int x) {

        if (x <= 1) {

            return false;

        }

        for (int i = 2; i < x; i++) {

            if (x % i == 0)

                return false;

        }

        return true;

    }

    static class static\_block\_Thread extends Thread {

        int start;

        int end;

        static int counter;

        int temp = 0;

        long startTime;

        long endTime;

        long timeDiff;

        public static\_block\_Thread(int threadNum, int threadAmount) {

            this.start = (NUM\_END / threadAmount) \* threadNum;

            if (threadNum != threadAmount - 1) {

                this.end = (NUM\_END / threadAmount) \* (threadNum + 1);

            } else {

                this.end = NUM\_END;

            }

        }

        @Override

        public void run() {

            startTime = System.currentTimeMillis();

            for (int i = start; i < end; i++) {

                if (isPrime(i))

                    temp++;

            }

            counter += temp;

            endTime = System.currentTimeMillis();

            timeDiff = endTime - startTime;

        }

    }

}

* 1. **Output**

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

1. **Static Cyclic Thread**
   1. **Code**

public class pc\_static\_cyclic {

    private static int NUM\_END = 200000;

    private static int NUM\_THREADS = 32;

    private static int JOB\_SIZE = 10;

    private static int[] THREAD\_NUMS = { 1, 2, 4, 6, 8, 10, 12, 14, 16, 32 };

    public static void main(String[] args) throws InterruptedException {

        if (args.length == 2) { // 인자를 정하면 해당 쓰레드 개수로 실행

            THREAD\_NUMS = new int[1];

            THREAD\_NUMS[0] = Integer.parseInt(args[0]);

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

        } // args가 없으면 사전에 정의된 THREAD\_NUMS로 for문을 돌며

        for (int i = 0; i < THREAD\_NUMS.length; i++) { // 1~32개의 쓰레드   
개수로 한 번씩 실행

            NUM\_THREADS = THREAD\_NUMS[i];

            static\_cyclic\_Thread.counter = 0;

            static\_cyclic\_Thread[] thread =   
new static\_cyclic\_Thread[NUM\_THREADS];

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

                thread[t] = new static\_cyclic\_Thread(t);

            }

            long startTime = System.currentTimeMillis();

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

                thread[t].start();

            }

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

                thread[t].join();

            }

            long endTime = System.currentTimeMillis();

            long timeDiff = endTime - startTime;

            System.out.println(NUM\_THREADS + " Thread\n");

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

                System.out.println("Thread " + t + " Execution Time: "  
 + thread[t].timeDiff + "ms");

            }

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

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

            System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        }

    }

    private static boolean isPrime(int x) {

        if (x <= 1) {

            return false;

        }

        for (int i = 2; i < x; i++) {

            if (x % i == 0)

                return false;

        }

        return true;

    }

    static class static\_cyclic\_Thread extends Thread {

        int startPoint;

        static int counter;

        int temp = 0;

        long startTime;

        long endTime;

        long timeDiff;

        public static\_cyclic\_Thread(int threadNum) {

            this.startPoint = threadNum \* JOB\_SIZE;

        }

        @Override

        public void run() {

            startTime = System.currentTimeMillis();

            for (int i = startPoint; i < NUM\_END; i = i + NUM\_THREADS \* JOB\_SIZE) {

                for (int j = 0; j < JOB\_SIZE; j++) {

                    if (isPrime(i + j))

                        temp++;

                }

            }

            counter += temp;

            endTime = System.currentTimeMillis();

            timeDiff = endTime - startTime;

        }

    }

}

* 1. **Output**

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 달력이(가) 표시된 사진

자동 생성된 설명

1. **Dynamic Thread**
   1. **Code**

public class pc\_dynamic {

    private static int NUM\_END = 200000;

    private static int JOB\_SIZE = 10;

    private static int[] THREAD\_NUMS = { 1, 2, 4, 6, 8, 10, 12, 14, 16, 32 };

    public static void main(String[] args) throws InterruptedException {

        if (args.length == 2) { //인자를 정하면 해당 쓰레드 개수로 실행

            THREAD\_NUMS = new int[1];

            THREAD\_NUMS[0] = Integer.parseInt(args[0]);

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

        } // args가 없으면 사전에 정의된 THREAD\_NUMS로 for문을 돌며

        for (int i = 0; i < THREAD\_NUMS.length; i++) { // 1~32개의 쓰레드   
개수로 한 번씩 실행

            int threadNum = THREAD\_NUMS[i];

            dynamic\_Thread.counter = 0;

            dynamic\_Thread.current\_value = -10;

            dynamic\_Thread[] thread = new dynamic\_Thread[threadNum];

            for (int t = 0; t < threadNum; t++) {

                thread[t] = new dynamic\_Thread(t);

            }

            long startTime = System.currentTimeMillis();

            for (int t = 0; t < threadNum; t++) {

                thread[t].start();

            }

            int thread\_cnt = 0;

            for (; thread\_cnt < threadNum; thread\_cnt++) {

                thread[thread\_cnt].join();

            }

            long endTime = System.currentTimeMillis();

            long timeDiff = endTime - startTime;

            System.out.println(threadNum + " Thread\n");

            for (int t = 0; t < threadNum; t++) {

                System.out.println("Thread " + t + " Execution Time: "   
+ thread[t].timeDiff + "ms");

            }

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

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

            System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

        }

    }

    private static boolean isPrime(int x) {

        if (x <= 1) {

            return false;

        }

        for (int i = 2; i < x; i++) {

            if (x % i == 0)

                return false;

        }

        return true;

    }

    static class dynamic\_Thread extends Thread {

        int x;

        static int counter;

        static int current\_value;

        int temp = 0;

        long startTime;

        long endTime;

        long timeDiff;

        int threadNum;

        public dynamic\_Thread(int threadNum) {

            this.threadNum = threadNum;

            this.x = update();

        }

        @Override

        public void run() {

            temp = 0;

            startTime = System.currentTimeMillis();

            while (this.x < NUM\_END) {

                for (int i = 0; i < JOB\_SIZE; i++) {

                    if (isPrime(this.x + i)) {

                        temp++;

                    }

                }

                this.x = update();

            }

            addFindNumber(temp);

            endTime = System.currentTimeMillis();

            timeDiff = endTime - startTime;

        }

        static synchronized int update() {

            current\_value += JOB\_SIZE;

            return current\_value;

        }

        static synchronized void addFindNumber(int temp) {

            counter += temp;

        }

    }

}

* 1. **Output**

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 명판이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트, 달력이(가) 표시된 사진

자동 생성된 설명

1. **How to start**
2. **Please go to proj1 directory in your terminal.**
3. **enter the command** 
   1. **pc\_static\_block : “ java problem1/pc\_static\_block.java “**
   2. **pc\_static\_cyclic : “ java problem1/pc\_static\_block.java “**
   3. **pc\_dynamic : “ java problem1/pc\_dynamic.java “**
4. **Program’s default setting is NUM\_END 200000, for loop 1 ~ 32 thread at once.  
   if you want to try just one kind of thread number, enter the command  
   “ java problem1/${filename}[ex)pc\_dynamic.java] ${num of thread} ${NUM\_END} “**