**OS1 – CS3510**

**Report**: Programming Assignment 2 - Finding Perfect numbers using threads.

**Aditya Sridhar**

**ES21BTECH11004**

**Problem Statement**:

Our problem is to find all perfect numbers between 1 and the number ”n”. This is done by partitioning the work amongst “k” threads created by the main thread, such that no two threads work on the same number (by work, we mean finding whether the numbers assigned to that thread are perfect or not). “k” and “n” are input variables.

**Brief Summary:**

The k threads were allocated the numbers from 1 to n as follows:

**Thread 1 – {1, k + 1, 2k + 1, 3k + 1,….., k\*⌊n/k⌋ + 1}**

**Thread 2 – {2, k + 2, 2k + 2, 3k + 2,….., k\*⌊n/k⌋ + 2}**

**Thread 3 – {3, k + 3, 2k + 3, 3k + 3,….., k\*⌊n/k⌋ + 3}, etc**

This would be the allocation, provided none of these numbers exceed n. This makes sure that each thread, to some extent, while not doing the work of any other thread, does around the equal amount of work as the other threads due to near-equal distribution.

**Detailed Description of Algorithm:**

* We first open the input file that contains the values of n and k. We take this file as a command line argument.
* In the occurrence that k is greater than n, we require only n threads (each thread will take care of one number). In this case, all threads between n + 1 and k are not required. This is printed in the OutMain.txt
* We now create a pointer-“parameters”, which stores the values of n, k, and i (i is the counter variable for the threads). Since pthread\_create takes a (void \*) parameter, this (int \*) pointer is typecasted to (void \*) and then passed to the function threadAlloc\_perfCheck.
* Function **threadAlloc\_perfCheck**:
* This function does the following – using the passed parameters n, k and i (this is unique for each thread), the function computes all numbers it has to check for whether it is perfect or not. **(Math. 1)**
* The thread checks whether the number is perfect or not using the function ifPerf. (**Math. 2)**
* These perfect numbers are stored in an array isPerf. This array has a maximum length of ⌊n/k⌋. The number of elements in the array is also stored in an integer variable – numOfPerf.
* All numbers assigned to Threadi are stored in a file OutFilei – This file will store whether each number is a perfect number or not.
* To the main file OutMain.txt, we append all the newly found perfect numbers by that thread. After the completion of all threads, OutMain.txt will contain each thread, followed by all the perfect numbers they found among the numbers they were assigned.

**Math.1:**

The algorithm for allocating the numbers amongst the threads is pretty straight-forward.

**Threadi will have an “adder”** assigned to it. This will be equal to i (i ranging from 1 to k). To get the numbers for each thread described in the **Brief Summary**, we go in a loop, taking j as the looping variable.

We can see that **the last term in all the numbers assigned to Threadi is the adder**. Hence, the numbers will be **j\*k + adder**, j ranging from 0 to **⌊n/k⌋**, including 0 and **⌊n/k⌋.**

In every iteration, we make sure that the number it is checking is less than n. If not, it goes to the next number.

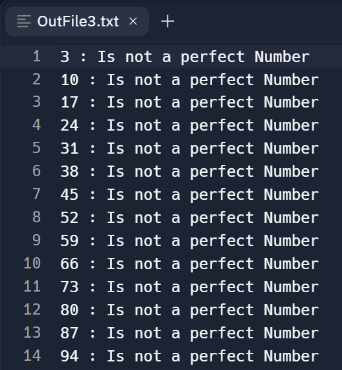
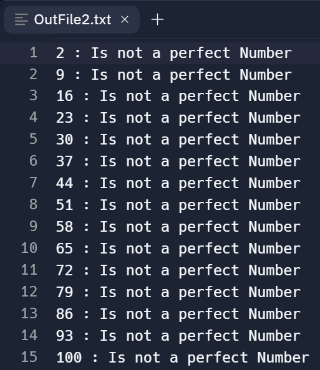
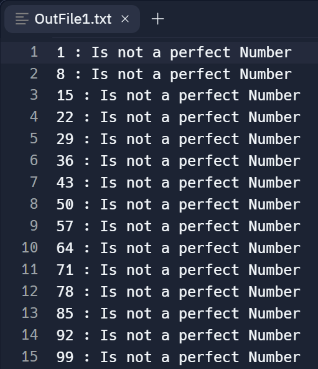
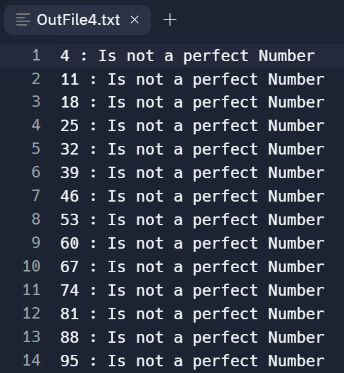
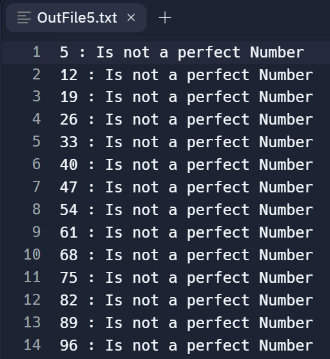
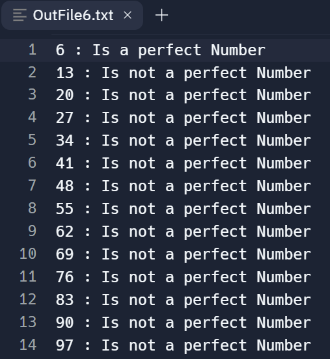
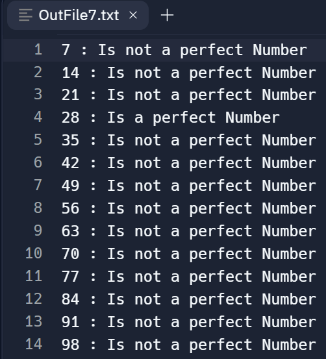
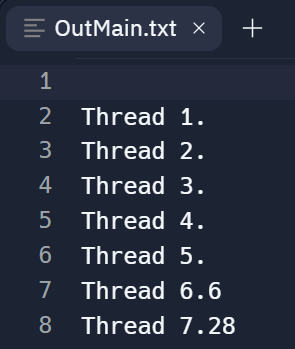
This way, we **near-equally distribute** all numbers 1 to n amongst k threads.

**Math. 2:**

The algorithm to check whether a given number is a perfect number is fairly simple. We find out all factors of the number and add them to the variable sumOfFact. In the end, if the sum of its factors (excluding itself) is equal to the number itself, the function returns 1; if not, it returns 0.

**Analysis of Output:**

1. **Input: n = 100, k = 7**

**Output**:

**Analysis:**

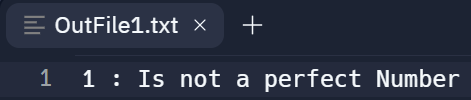
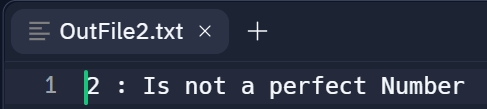
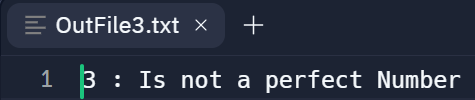
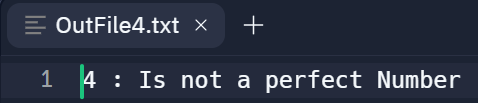
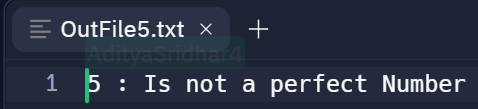
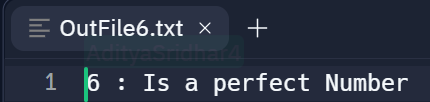
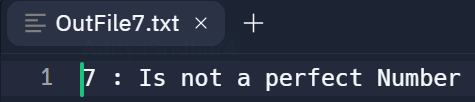
* We know that the only perfect numbers between 1 and 100 are 6 and 28. Hence, that computation has been done properly (Math. 2).
* We see that Threads 1 and 2 have computed 99 and 100. However, the subsequent threads have **NOT computed 101, 102, and so on**. Hence, that condition has been applied properly.
* The allocation has also been done properly. According to our algorithm,

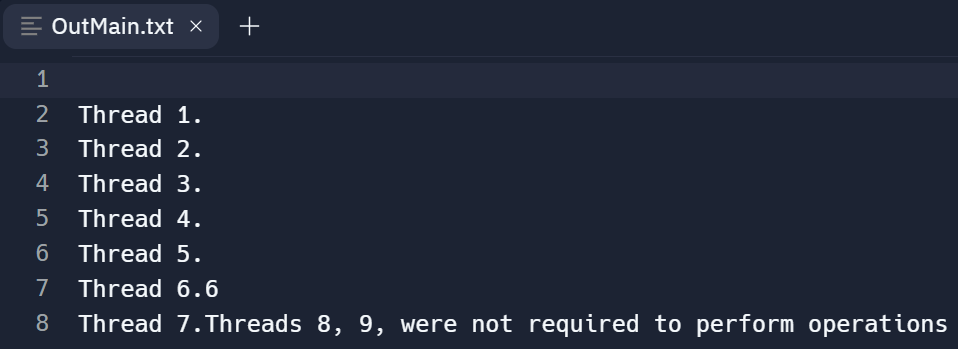
Thread6: {**6**, 13, 20, 27, …}

Thread7: {7, 14, 21, **28**, 35, …}

* As expected, Threads 6 and 7 have detected 6 and 28 to be perfect numbers, respectively.

1. **Input: n = 7, k = 9**

**Output:**

****

**Analysis:**

* The only perfect number between 1 and 7 is 6. This is detected by Thread6­.
* The allocation has been done properly. According to our algorithm,

Thread1: {1}

Thread2: {2}

Thread6: {**6**}

* Also, as per our code, threads 7, 8, and 9 are **not required and are hence not used**. This is also printed in the OutMain.txt file.