CS 3510

Assignment 1: Finding Tetrahedral Numbers from 1 to N

Name: Swapnil Bag

Roll No: ES22BTECH11034

Report

The main aim of the program is to check for tetrahedral numbers in the range of 1 to N. It is achieved by a multi - process approach. Here the parent process creates a shared memory buffer that can be accessed by the child processes. K separate child processes are created with their own local child buffers that determine if a number is tetrahedral or not from specific section of the parent buffer and stores them. These local child buffers are shared with the main Parent process that consolidates the search results.

Low level design of program

- **is_tetrahedral ():** This function returns true if the number is tetrahedral. It takes the cube root of the integer n and rounds it to nearest integer and compares if it is tetrahedral.
- **main function:**
    a. The values of N and K are obtained from the input file using command line arguments

    b. The parent process creates a shared memory buffer that populates it with numbers from 1 to N. Size of parent buffer **(PARENT_BUFFER)** is **N**. It is named **"/SWAPPY".** It is created using **shm_open**() , truncated to appropriate size by **ftruncate()** , mapped to memory space by **mmap()** and accessed by parent pointer **P_ptr**

    c. The start time for the creation of process is recorded using **gettimeofday()**

    d. K Child processes are created using a for loop. A new process is created by using the **fork()** function call stored in **pid**. If pid is 0 it means it is a Child process. Within each Child process buffers of size **(N/K + 1)** are created. Each process is named uniquely by concatenating the child number with it

"/Process1" , …..."/ProcessK" and a pointer to each memory section **C_ptr.** These then identify the tetrahedral numbers and stores it in their buffer

e. A loop is run to wait for each child process to complete using **wait(NULL)**

f. Finally the parent process reads the share memory buffers of the K child processes and logs the results in the OutMain.log file (using **print_output()** function). The end time of the program is recorded

- print_output() : This function reopens the shared memory buffers created by the child processes. For each buffer index; if at all it is a non-zero number then it prints that number to the log file. It then unmaps the memory using **munmap(),** closes the file descriptors using **fclose()** and unlinks each shared memory segment using **shm_unlink** to relinquish memory.

Scheme to distribute workload among various processes

The shared memory segment is initialized . An index counter **idx** is set to 0, as and when a tetrahedral number is found it stores it in the buffer and increments the index. The workload is distributed as follows with each process working on the following numbers by accessing parent buffer

i. **Process 1** : 1, k+1 , 2k+1 ……
ii. **Process 2** : 2, k+2 , 2k+2 ....
iii. **Process 3** : 3, k+3, 2k +3 ……
iv. **Process k** : k , 2k ,3k ……..

Complications

- Setting up the shared memory buffers for the children process, and ensuring that that the values are being written in the correct indexes was one challenge to overcome. Reopening the shared memory buffers in the parent process with the correct names and file descriptors was one of the other complications that arose.
- Dividing the numbers across the different process so that each process gets equal work load, required an efficient scheme. Implementing it to ensure all numbers are covered and buffers are not overwritten or leading to segmentation faults.

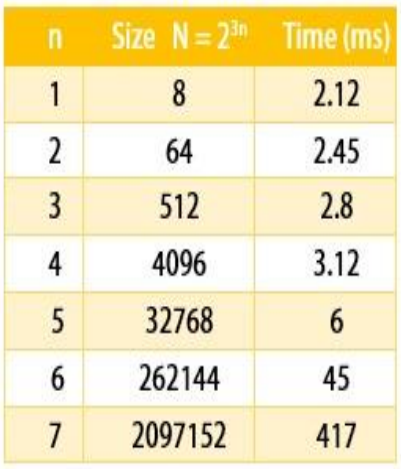
Output

Each process creates a **OutFilei.log** that contains data about whether a given number is tetrahedral or not.

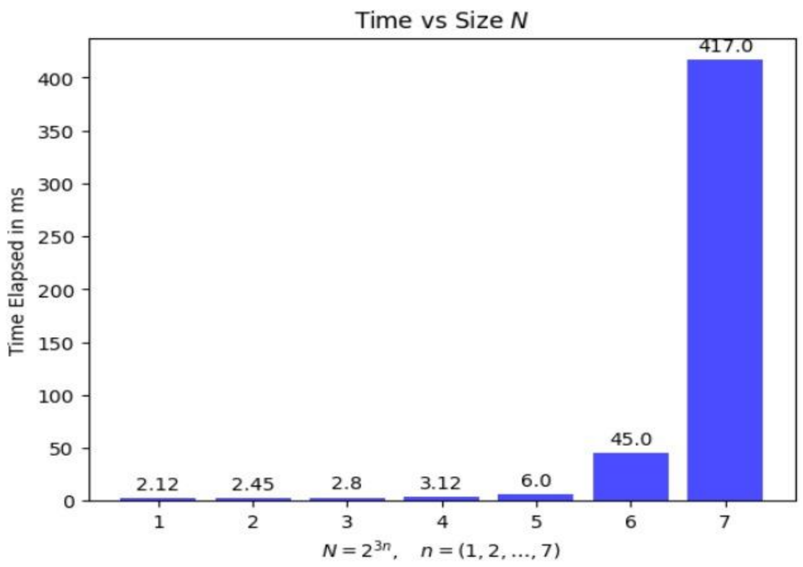
The main parent process creates a **OutMain.log** that logs which process has found which number. A common observation is that the process **i** where **i** is a multiple of 4 tends to identify a greater number of tetrahedral numbers, because of the properties of divisibility of the tetrahedral number. Furthermore these process do not create a bottleneck since to check whether a number is tetrahedral or not, it takes same time for computation either case.

Graphs

Time vs Size, N



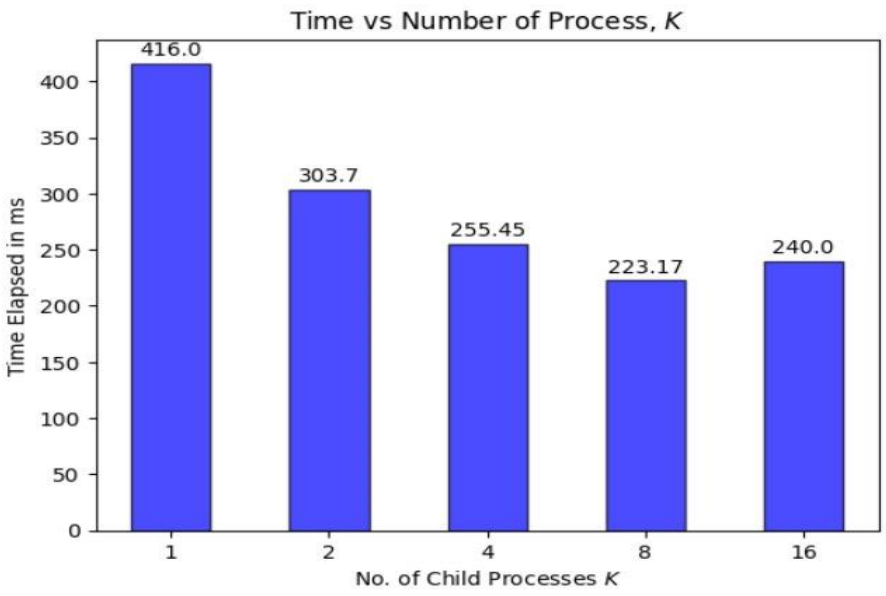
n	Size $N = 2^{3n}$	Time (ms)
1	8	2.12
2	64	2.45
3	512	2.8
4	4096	3.12
5	32768	6
6	262144	45
7	2097152	417



Here the Size of N increases and the number of processes is held constant at K = 8. This graph indicates that as N increases more time is required for each child process to check through its assigned section of numbers and the time elapsed increases

Time vs Number of Processes, K

K	Time (ms)
1	416
2	303.7
4	255.45
8	223.17
16	240



Here N is 1000000 and the number of process varies. The time elpased highly depends on the number of cores of the machine on which it runs. Here as the value of k increases from 1 to 8, the the time taken reduces as the numbers are checked in parallel and work is divided . Since the program was run on a 8 core machine, for the K =16 case,the time increases because overhead is incurred to schedule the processes to the different cores. The time taken also largely depends on how the OS schedules the process and what other processes is running in the background.