**N.Krishnateja-G01449836**

**Objective:**

Increase the performance of a regular sequential method of Smith-Waterman Dynamic programing using Multi-threading with the DNA files, *HIV-1\_db.fasta* being the known sequence and *HIV-1\_Polymerase.txt*  being the unknown sequence.

**Analysis:**

The analysis has been done in two approaches. First a sequential approach to find the base of the speedup ratio of the Multi-Threaded programming. Second, using the Multi threaded programing and calculating the performance variation at each threads.

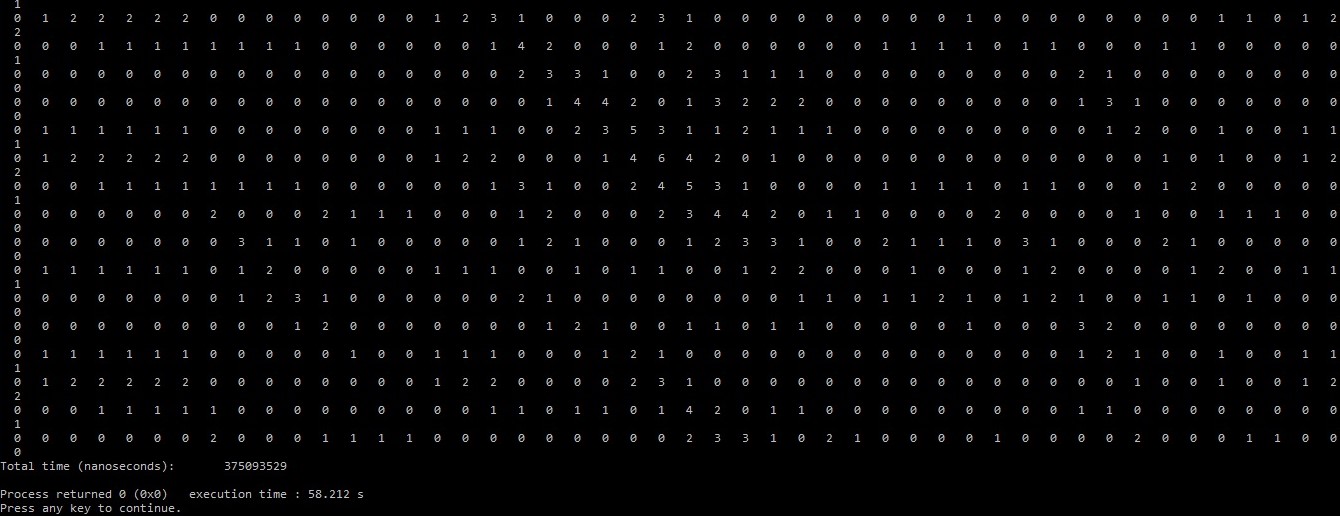
**Sequencial Programing:**

This approach has been the traditional way of matrix filling using Smith-waterman algorithm. This approach fills the matrix row by row sequentially.

Time calculated for the algorithm has been measured in Nano seconds to get sharp edge on minute performance parameters.

Time Calculated has been: 375093529 ns

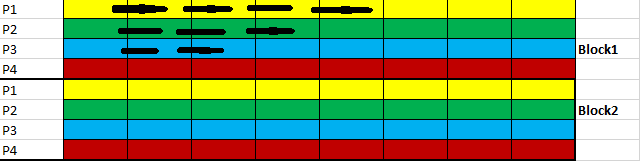
Sample output has been like:



**Multi\_threaded Programing:**

**Approach:**

For this algorithm to shape multi threading, there are various ways to perform Multi-Threading. Various approaches are like column by column, row by row and Diagonal by Anti-Diagonal approach. I have followed a simply modified row by row approach. A simple pictorial representation can be shown like this.



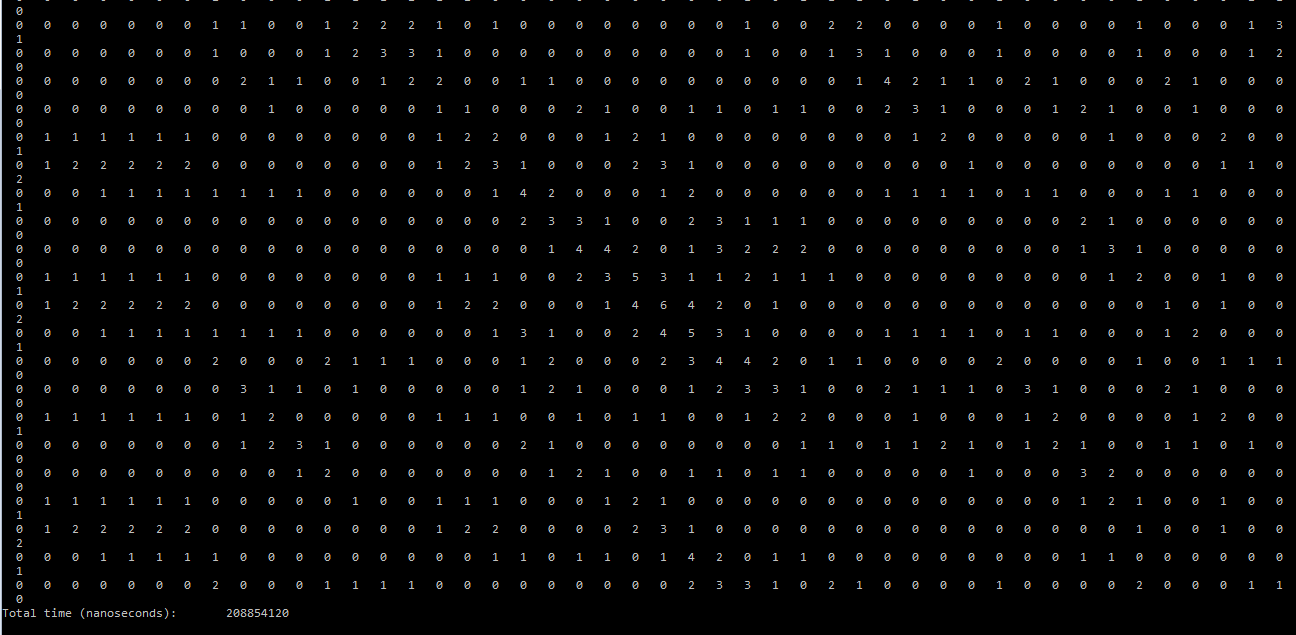
In our programming, the sequence (known) with bigger length has been chosen as rows and the sequence (unknown) with lesser length has been chosen for columns. Let us consider, we have chosen **p** threads to run the multi-threading program. The matrix is of length being  **m\*n .** Then the blocks that have been generated for the approach to proceed would be **m/p,** as the threads were chosen row wise. The threads are being synchronized at each block to proceed for the approach. Whenever, an element in **m[i-1][j+1]** has got a value, then the corresponding thread initiate. This approach is more pipeline oriented one.

**Programming:**

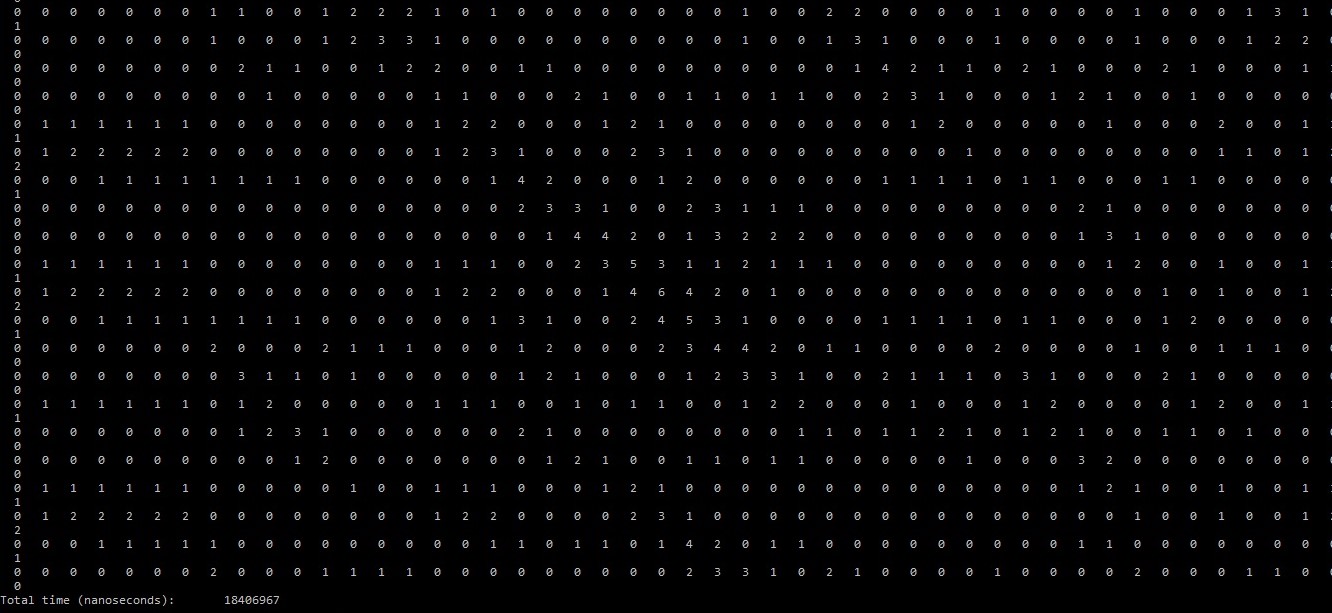
The sequences from the files were initially read through vectors defined as Sequence vectors. A 2D-Vector has been initialized globally to keep track of the values entered at each step and eliminate data loss. The Sequence Matrix has been initially set to -1 to differentiate the calculated cells and uncalculated cells of the matrix. A Functor has been used for an easy accessibility of threads. The functor contains the override method **operator()** that posses the actual approach of the thread movement and Matrix Calculation . Threads have been synchronized using mutex locks which were adopted from BOOST libraries.

**Results:**

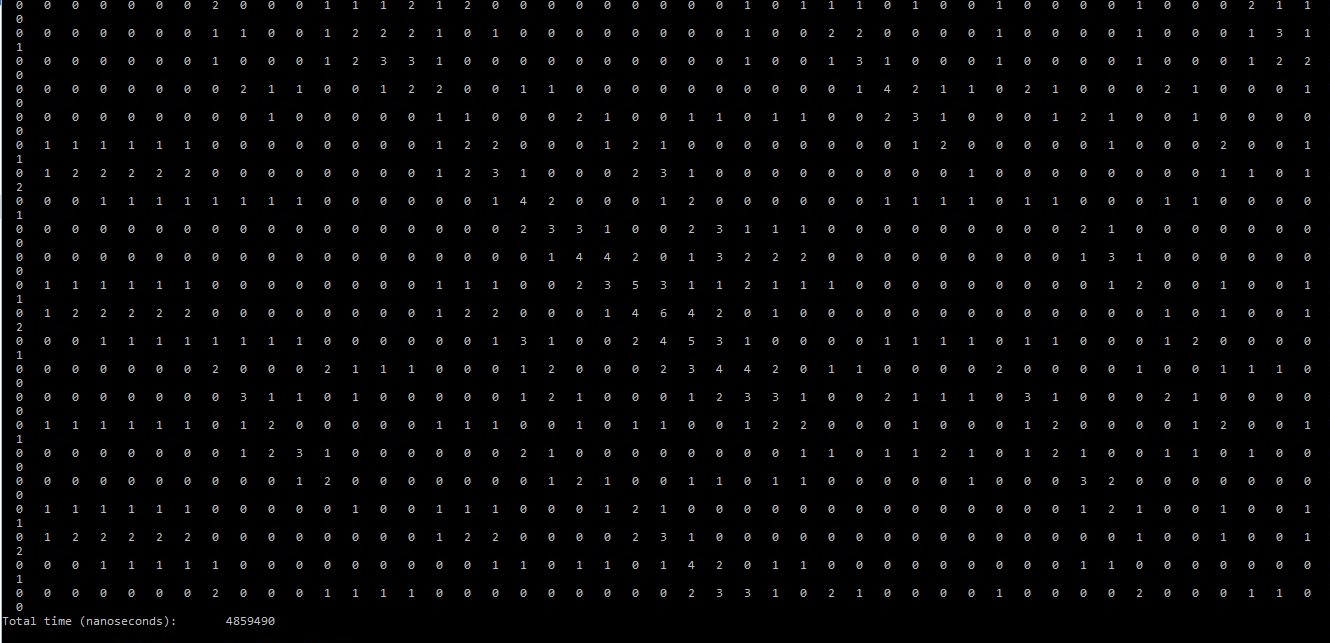
**Thread Run 4**: Time taken: 208854120 ns



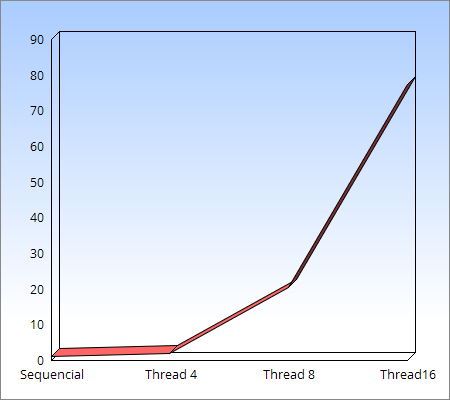
**Thread Run 8:** Time taken : 18406967



**Thread Run 16:** Time taken: 4859490



**Discussions:**

****

**Conclusion:**

The graph shows a gradual increase in performance from sequencial to Thread 16.

The highest Speed up has been measured at Thread 16 for 77.37 increase.