**COP5612 – Fall 2020**

**Project 1**

**Team Members:**

1. Arjun Tekkatte Raghavendra

UFID: 15012913

1. Harish Kugur Sreekanta Babu

UFID: 48297246

**Instructions**:

* Download the File
* Run command : **dotnet fsi --langversion:preview proj1.fsx 100000000** **24**

**System Specifications:**

* Operating System : Windows – Quad Core CPU
* dotnet sdk version : 3.1

1. **Size of the work unit Determined:**

Size of each work unit (w) we finalized on is **100.**

Range for each worker = **N/w**

For Example: When N = 1000, each worker gets the range of 10 numbers

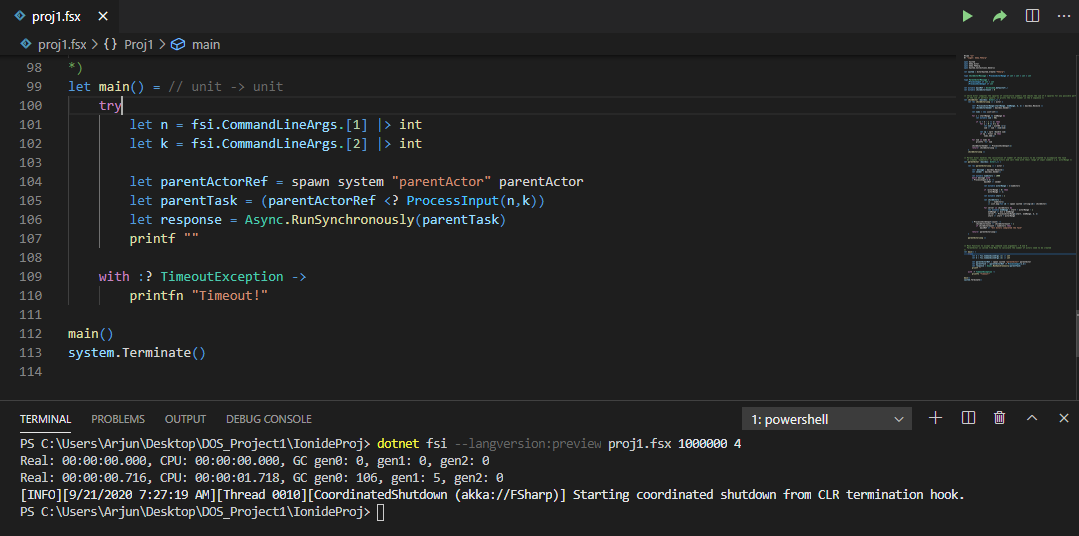
**Explanation**: We ran our tests on a machine with Quad Core CPU for very large inputs (N= 10^8, k=24). We initially set off our testing with say *w* =8000 work units (The machine we tested on has 4 cores and we assumed a workload of 2000 units per core). With this setting, the ratio of CPU to real time was 2.79. To achieve a ratio close to the value 4, we increased the number of work units to 10000 but the performance of the program degraded (*i.e.* the Real Time of execution considerably increased). Henceforth, we gradually decreased the number of work units and noticed that the ratio of CPU time to Real time (*i.e.* is core utilization) was gradually increasing with performance improvements. At *w* = 500, we achieved a ratio of 4. In that range of *w,* the program consistently achieved the ratio of 4 and upon slowly decreasing *w* further, performance was also improving. We achieved the best performance for the given input when *w* was set to 100. For values below 100 and close to 70, core utilization began to decrease and the ratio fell below 4 and consequentially, the performance degraded. Hence, we set the number of work units to 100, at which maximum performance was achieved (with efficient concurrency)

1. **Result for the given sample input:**

*Input*: N = 1000000, k = 4

Command: **dotnet fsi –langversion:preview proj1.fsx 1000000 4**

***Output***:



Note: For the above input, there are no sequences of consecutive numbers of length 4 that would result in a perfect square

1. **Run Time For Above Input and CPU time to Real Time Computations**

* Real Time: 00:00:00.716 (0.716 seconds)
* CPU Time: 00:00:01.718 (1.178 seconds)

**Number of Cores Effectively used in the computation = CPU Time / Real Time**

**= 2.3994**

1. **The Largest Problem We Managed to Solve:**

*Input*: N = 1000000000 (10^9), k = 24

*Run Times*:

* Real Time: 00:05:07.326 (~ 5 minutes 7 seconds)
* CPU Time: 00:21:13.368 (~ 21 minutes 13 seconds)