

COP5615-Distributed-Systems: Project 1

Group Members -

1. Aditi Page (UFID: 0427-9968)
2. Kartik Rode (UFID: 8971-1791)

README

1. Size of the work done and distribution strategy -

We have used 10 actors to distribute the work that has been provided in the input. We distribute the tasks by dividing the given numbers by 10 so that each actor gets to work on (input/10) numbers. If the input number is less than 10, we have designed the program in a way that one actor can handle all of the tasks. In this way, we have separated the mechanism for larger numbers and smaller numbers. This task is handled by the boss actor. In this way the program utilizes all the 4 machine cores and 8 logical cores.

if input < 10:

1 actor handles everything

else:

10 actors are spawned and work is distributed

Determination of the subproblems -

When the boss actor is called by the main program, it takes the input and window size as a parameter. On getting these values, it creates a separate message for each of the worker actors which determines the values obtained by doing input/10. These worker actors are spawned immediately and start performing their work concurrently. So, if n = 10,000 the actors will get the chunk of work in the following way-

- Actor1 - 1 .. 1000
- Actor2 - 1001 .. 2000
- | | |
- Actor10 - 9001 .. 10,000

In this way all the actors are working on equal amounts of numbers.

2. The result for running the program for 1000000 and 4 -

Running the above test case did not return any value for a lucas square number

3. The running time for 1000000 and 4 is -

Real - 00:00:02.320 CPU - 00:00:10.359

For this input we are getting CPU time/Real time as approximately 5 which tells us we have effectively used 5 cores for our computation. When we increase the number, the core utilization increases.

4. The largest problem that we managed to solve was -
 $n = 100000000$ and $k = 24$

The screenshot shows a Visual Studio Code window with a script file named 'Script10.fsx' and a 'Developer PowerShell' terminal. The script is a F# program that calculates the sum of a large range of numbers (1 to 100,000,000) using a parallel approach with 10 actors. The script includes a loop that iterates over the range, calculates the sum for each part, and then aggregates the results. The terminal output shows the execution progress, including the number of actors used and the final result. The status bar at the bottom indicates the CPU time and real time for the execution.

```

let input = com.[0] >> int
let window_size = com.[1] >> int
let number_of_actors = 10
let part = input / number_of_actors
let mutable resultlist = []
for i = 1 to 9 do
    resultlist <- [i*part] |> List.append resultlist

let message1 = "1" + (resultlist.[0] |> string)
spawn system "0" worker1 <| Multisum message1
for i = 1 to 8 do
    let message = ((resultlist.[i-1] + 1) |> string) + " "
    spawn system (sprintf "%i" i) worker1 <| Multisum message

let message10 = ((resultlist.[8] + 1) |> string) + " " + (input |> string)
spawn system (sprintf "%s" "10") worker1 <| Multisum message10

Done meaningful ->
actors used <- actors used + 1
if(actors used = 10) then
    flag <- true
return! loop()
}
loop()

let message_string = (input |> string) + " " + (window_size |> string)
if(window_size > 0) >> int then
    let boss_caller = spawn system "boss" boss
    boss_caller <| Divide message_string
else
    flag <- true

let mutable Break = false
while not Break do
    if(flag) then
        Break <- true
    done

let message_string = (input |> string) + " " + (window_size |> string)
if(window_size > 0) >> int then
    let boss_caller = spawn system "boss" boss
    boss_caller <| Divide message_string
else
    flag <- true

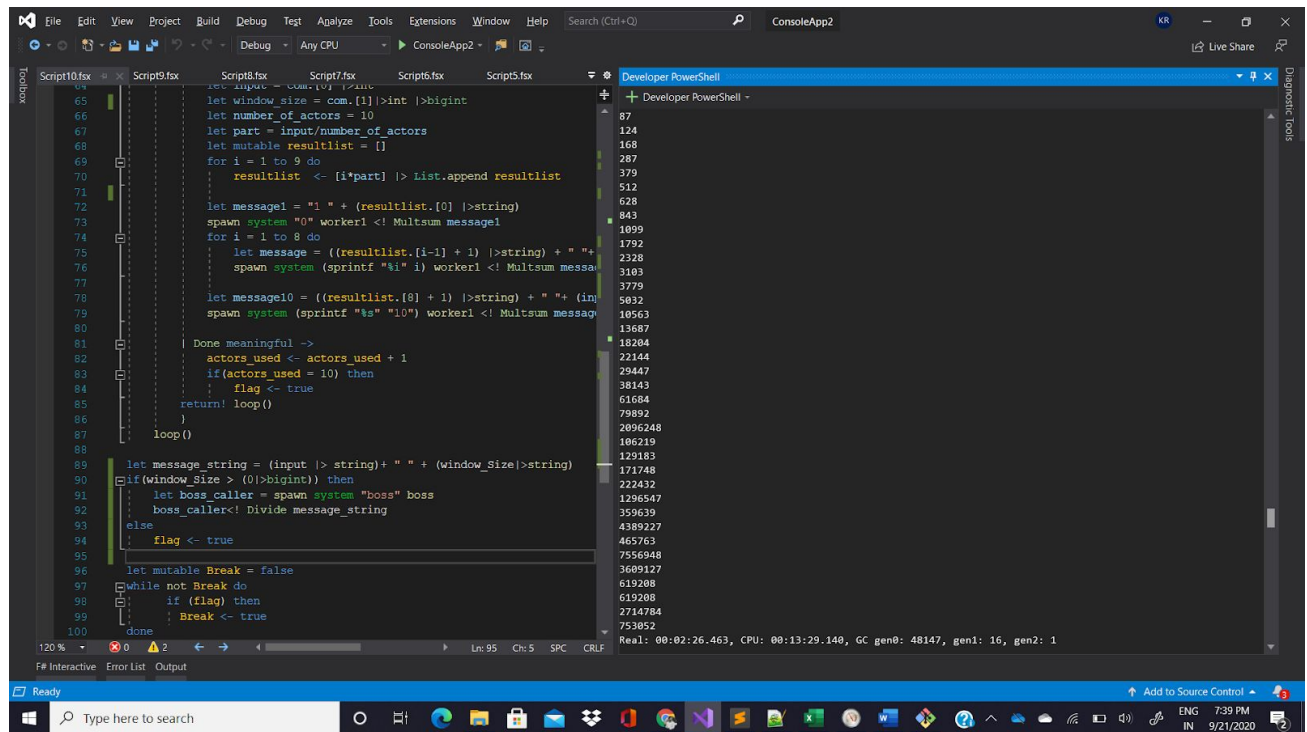
let mutable Break = false
while not Break do
    if(flag) then
        Break <- true
    done

```

Real: 00:10:37.572, CPU: 00:57:24.359, GC gen0: 242656, gen1: 74, gen2: 6
 PS C:\Users\kartz\source\repos\ConsoleApp2\ConsoleApp2>

Real - 00:10:37.572 CPU - 00:57:24.359 where the CPU time/Real time gives us approximately 6 cores for our computation.

5. Example:
 $n = 100000000$ and $k = 50$



Real - 00:02:26.462 CPU - 00:13:29.143 where the CPU time/Real time gives us approximately 6.5 cores for our computation.

Instructions to run:

1. Unzip the file
 2. Run the project using:
 - `dotnet fsi --langversion:preview proj1.fsx input1 input2`
- if your system is running a language version lower than 5.0.