**Lab 03 - Parallel programming using only shell scripting**

**Goal**: Write a simple shell script that solves the problem in parallel by running N instances of the serial implementation (N parallel tasks, we assume N cores are available). Consider synchronization and communication issues. Use domain decomposition or functional decomposition as needed.

**Advantages:**

* No need to learn a new language [construct]
* No need to learn a new API or library

**Disadvantages:**

* We need to know some shell scripting
* Carefully consider the command line parameters
* Need to carefully consider issues such as scalability or load balancing

**Problem 1: Finding perfect numbers inside interval [A,B], say [2, 1.000.000]**

Hints:

1. Adjust the solution for “finding primes” to search for perfect numbers.  
   Perfect number = a number that is equal to the sum of its divisors (except itself, of course)  
   Ex: 6 = 1+2+3, 28 = 1+2+4+7+14, etc.
2. Split the interval in N equal parts (say N=4) then run the serial code for each subinterval in parallel; use “&” to run programs in background; use “wait” to force the script to wait until every task has ended.
3. How do we measure the execution time? Use the “time” command in Linux
4. What to do with the perfect numbers that were found? Display on screen? Write to file(s) + append? Problem: mixing the computation time with I/O time ! (I/O is inherently serial)  
   => serial fraction is higher -> Amdahl’s law.  
   SOLUTION: just display them on screen; fortunately there are very few of them so the impact is negligible. There are only 4 perfect numbers in the interval [2, 1.000.000].
5. Problem is embarrassingly parallel; no communication between tasks, no synchronization.
6. Compute the execution time T1 for N =1, and execution time T4 for N=4.   
   Compute the speedup S4 = T1 / T4. Compute efficiency E4 = S4/4.   
   [Disable Hyper-Threading if possible]
7. How does it compare to the expected [linear?] speedup? Explain the result.
8. Hardcoded script – see course / seminar
9. General script [hint]

#!/bin/bash

# Spawns $1 instances of the program $2

instances=$1

program=$2

for i in `seq 1 $instances`

do

$program &

done

wait

**Problem 2: Computing the entropy of a binary sequence**

Hints:

1. We are familiar with this problem by now; however, where do we get the binary sequence from? Let us assume the sequence is stored in an input file “input.bin”
2. Let us assume that the full sequence can be loaded into memory (length = 1GB)
3. Say we have N serial programs running in parallel; we divide the sequence in N subsequences of equal length, each program will read into memory only its corresponding subsequence (1GB / N)
4. How to read data efficiently? In large chunks! (128 KB or higher)
5. Once each program has loaded the data into memory, parallel processing begins. Each program computes the number of “1” bits from its corresponding subsequence, and writes it in a separate file (file1.txt, file2.txt…)
6. The shell waits for the program to finish execution, then reads the partial results from all the files, computes the overall number of ones, and the entropy; how? (what about running another serial C program that does that?)
7. Problem is nearly embarrassingly parallel; no communication between tasks, but there is synchronization in the end. How does the serial part in the end affect the performance ? (Amdahl’s law)
8. Compute the execution time T1 for N =1, and execution time T4 for N=4.   
   Compute the speedup S4 = T1 / T4. Compute efficiency E4 = S4/4.  
   [Disable Hyper-Threading if possible]
9. How does it compare to the expected [linear?] speedup? Explain the result.
10. General script [hint]

#!/bin/bash

binSize=$1

instances=$2

#./wef $binSize entropy.bin # write entropy file

offset=0

instanceSize=$(($binSize / $instances))

calcParam=$binSize

for i in `seq 1 $instances`

do

./count\_ones entropy.bin $offset $instanceSize $i.count & #count ones

calcParam="$calcParam $i.count"

offset=$(($offset + $instanceSize))

done

wait

./e $calcParam #calculate entropy

**Optional**: Consider the scalability issue; adapt your scripts; compute the execution time and the speedup programmatically.

**OBS**: programming in parallel in this manner was the paradigm used by the CERN physicists in their search for the Higgs Boson; the system designed for this huge computational task was called the “grid” and this paradigm is known as “grid computing”; both domain decomposition and functional decomposition were used.

**Hints:**- you should use Linux for this lab (although in theory you can do the same in Windows)   
- OR you can use a Linux virtual machine for Oracle Virtual Box (in Windows)

Oracle Virtual Box is available for free here:

<https://www.virtualbox.org/wiki/Downloads>