0907731 Advanced Parallel Processing (Fall 2019)
<u>Midterm Exam</u>

الاسم: ....   رقم التسجيل: ....   رقم التسلسل: ....

==

<u>Instructions</u>: Time **70** min. Open book and notes exam. No electronics. Please answer all problems in the space provided and limit your answer to the space provided. No questions are allowed. There are three problems.

==

P1. The computation of a histogram is a frequent operation in image processing. The histogram simply counts the number of occurrences of each tonal value in the given image. Consider a 2-dimensional input gray-scale image *I* of size $n \times n$ and *h* tonal levels. Assume that *n* and *h* are powers of 2. Its histogram (initialized with all zeros) can be sequentially computed as follows:

```
for (i=0; i<n; i++)
     for (j=0; j<n; j++)
          histogram[I[i,j]]++
```

a) What is the computational complexity *O*(.) of this computation as a function of *n* and *h*?

[1 mark]

<u>The Solution is:</u>

$O(n^2)$

b) Give the pseudo-code for a CREW PRAM algorithm that uses *p* processors to compute the histogram assuming that the input image is partitioned between processors.

[5 marks]

<u>The Solution is:</u>

```
// The image rows are partitioned among the p processors
// Use array histograms[] of size h * p
// p_ID is the processor ID between 0 and p-1
for (i=0; i<n; i++)   do_in_parallel
     for (j=0; j<n; j++)
          histograms[I[i,j], p_ID]++

for (i=0; i<log(p); i++)
     for (j=0; j<p/power(2, i+1); j++)   do_in_parallel
       for (k=0; k<h; k++)
          histograms[k, j] += histograms[k, j + p/power(2, i+1)]
// The final result is in histograms[:, 0]
```

c) Repeat question (b) above assuming that the histogram slots are partitioned between processors and the input image is shared.

[5 marks]

The Solution is:

```
// p_ID is the processor ID between 0 and p-1
for (i=0; i<n; i++)   do_in_parallel
     for (j=0; j<n; j++)
          if (I[i,j] mod p == p_ID)
               histogram[I[i,j]]++
// Assuming the comparison takes negligible time wrt incrementing
// and the histogram is of uniform distribution
```

d) Analyze your algorithms in terms of time, speedup, efficiency, and cost.

[4 marks]

Algorithm	**Input image partitioned**	**Histogram slots partitioned**
Time	$O(n^2/p + h\log(p))$	$O(n^2/p)$
Speedup	$O(n^2) / O(n^2/p + h\log(p))$	$O(p)$
Efficiency	$O(n^2) / O(n^2 + hp\log(p))$	$O(1)$
Cost	$O(n^2 + hp\log(p))$	$O(n^2)$

P2. Assume you want to parallelize a given sequential program and you want achieve a speedup of at least 10 on sixteen processors. What is the maximum serial fraction of the program under consideration of

[5 marks]

a) Amdahl's law,

The solution:

$S \leq 1 / (f + (1-f)/p)$

$10 \leq 1 / (f + (1-f)/16)$

$10f + (10/16)*(1-f) \leq 1$

$f \geq (1-10/16) / (10-10/16) = 0.04$

b) Gustafson's law?

The solution:

$S \leq p + f(1-p)$

$10 \leq 16 + f(1-16)$

$f \geq (16-10) / (16-1) = 0.3125$

P3. The Euler–Riemann zeta function $\zeta(s)$ can be computed using the following formula:

$$\zeta(s) = 2^s \cdot \lim_{k\to\infty} \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{(-1)^{i+1}}{(i+j)^s} \quad .$$

We can approximate $\zeta(s)$ up to degree k using the following code snippet:

```
double Riemann_Zeta(double s, uint64_t k) {
     double result = 0.0;
     for (uint64_t i = 1; i < k; i++)
          for (uint64_t j = 1; j < k; j++)
               result += (2*(i&1)-1)/pow(i+j, s);
     return result*pow(2, s);
}
```

The asymptotic time complexity of a single call to `Riemann_Zeta` is obviously in $O(k^2)$. To investigate the approximation quality depending on the parameter k, we calculate the corresponding values of `Riemann_Zeta(s, k)` for all $k \in \{0, \ldots, n - 1\}$ and write the result to a vector `X` of length n:

```
for (uint64_t k = 0; k < n; k++)
     X[k] = Riemann_Zeta(2, k);
```

Compete the following code to efficiently parallelize this loop using the C++11 threads.

[10 marks]

```
#include <iostream> // std::cout
#include <cstdint>  // uint64_t
#include <vector>   // std::vector
#include <thread>   // std::thread

double Riemann_Zeta(double s, uint64_t k) {
   double result = 0.0;
   for (uint64_t i = 1; i < k; i++)
        for (uint64_t j = 1; j < k; j++)
             result += (2*(i&1)-1)/pow(i+j, s);
   return result*pow(2, s);
}

int main(int argc, char * argv[]) {
   const uint64_t num_threads = 4;
   std::vector<std::thread> threads;
   const uint64_t n = 1UL << 15;
   std::vector<double> X(n);

   auto cyclic = [&] (const index_t& id) -> void {

     for (uint64_t k = id; k < n; k += num_threads)
        X[k] = Riemann_Zeta(2, k);
   };

   for (uint64_t id = 0; id < num_threads; id++)
      threads.emplace_back(cyclic, id);

   for (auto& thread: threads)
      thread.join();









}
```

<Good Luck>