CS2101 Due: Friday 10-th of October 2014

Problem Set 1

CS2101 *Submission instructions on last page*

PROBBLEM 1. [40 points]

The value of π can be calculated in a number of ways. Consider the following method of approximating π:

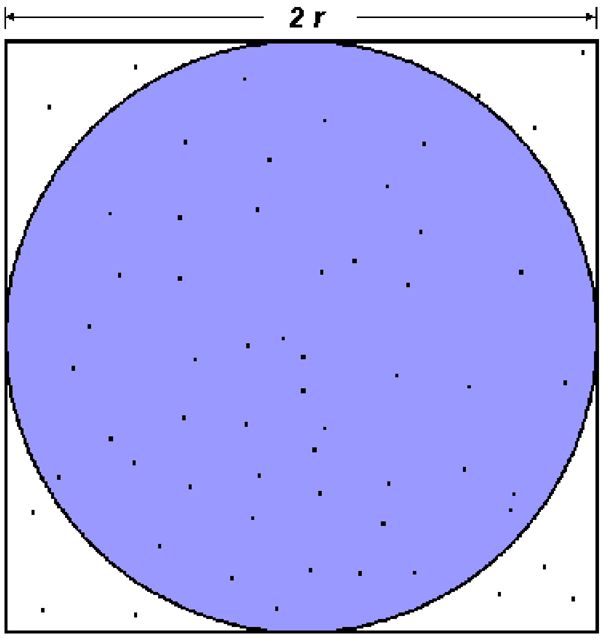
(1) Inscribe a circle (of radius r) into a square of size $2r$.

(2) Randomly generate N points in the square (for a large N, say $N = 1000000$).

(3) Determine the number of points in the square that are also in the circle.

(4) Let f be the number of points in the circle divided by the number of points in the square.

(5) π can be estimated as $4f$.



$$A_S = (2r)^2 = 4r^2$$
$$A_C = \pi r^2$$
$$\pi = 4 \times \frac{A_C}{A_S}$$

Question 1. [10 points] Explain why π can be estimated by $4f$.

Question 2. [10 points] Write a serial `Julia` program implementing the above algorithm.

Question 3. [10 points] Write a parallel `Julia` program implementing the above algorithm.

Question 4. [10 points] Compare the running times of the two programs for various values of N (say $N = i \times 20000$, for $i = 1 : 10$) and various values of the number p of workers ($p = 2, 3, 4$).

PROBBLEM 2. [20 points] The goal of the exercise is to experiment with *tasks* in `Julia` and use them for implementing a simple program based on the producer-consumer scheme.

In this scheme, the producer generates the prime numbers $2, 3, 5, 7, 11, 13, 17, 23, 29, 31 \ldots$ The consumer generates all pairs of consecutive primes (p_1, p_2) with a gap of 2. that is such that $p_2 = p_1{+}2$. Hence the consumer generates the pairs (3, 5), (5, 7), (11, 13), (29, 31), ...

You can learn about the topic of *prime gap* from `http://en.wikipedia.org/wiki/Prime_gap`.

Question 1. [10 points] Write a `Julia` task called `primeStream` such that the i-th call `consume(primeStream)` returns the i-th prime number, starting at 2. Recall that `Julia`'s function call `isprime(n)` returns *true* if `n` is a prime, `false` otherwise.

Question 1. [10 points] Write a `Julia` task called `TwoGapPrimesStream` such that the i-th call `consume(TwoGapPrimesStream)` returns the i-th pair of consecutive primes (p_1, p_2) with a gap of 2. This second task should make use of the task `primeStream`.

PROBBLEM 3. [40 points] The goal of the exercise is to experiment with the performance degradation caused by high rate of data transfer within the memory hierarchy (so called *cache misses*). We will experiment with three implementations of the same operation and compare their running times within `Julia`. To this end, we will consider a fundamental and simple operation: *matrix transposition.* Reviewing this operation can be done at the wikipedia page `http://en.wikipedia.org/wiki/Transpose`

Question 1. [10 points] A simple algorithm performing matrix transposition is stated below in the form of pseudo-code. Please note that the stated algorithm takes as input a rectangular matrix `A`, with `m` rows and `n` columns, and produces as output a matrix `B` (thus rectangular with `n` rows and `m` columns):

```
for i=1:m
    for j=1:n
        B[j,i] = A[i,j]
```

Hence this algorithm is necessarily working in an *out-of-place* fashion.

You are required to write a `Julia` function that takes an `m` $\times$ `n` matrix A and returns its transpose ${}^tA$ following the transpose principle stated above. Please note that this `Julia` function is not required to use any of the parallelism constructs of the `Julia` language. However, please feel free to experiment with those parallelism constructs, if you like.

Question 2. [10 points] We investigate another approach for computing the transpose ${}^tA$. For simplicity, we focus on the case where A is a square matrix. The proposed approach is based on a divide and conquer scheme. In the formula below, we assume that n is a power of 2 and that $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}$ denote square blocks of order $n/2$.

$$ {}^tA = \begin{cases} \begin{pmatrix} {}^tA_{1,1} & {}^tA_{2,1} \\ {}^tA_{1,2} & {}^tA_{2,2} \end{pmatrix} & \text{if} \quad A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \\ \hfill A & \text{if} \quad n = 1 \end{cases} \tag{1} $$

You are required to write a `Julia` function that

- takes as input a positive integer value n, which is assumed to be a power of 2,
- generate an $n \times n$ matrix `A` with random entries of type integer with values in the range $0 \cdots n-1$.
- transpose this matrix in-place using the above divide-and-conquer approach.

Once again, using parallelism constructs is not required, but you are welcome to experiment with it.

Question 3. [10 points] We consider a third approach for computing the transpose ${}^tA$. This is again a divide and conquer, called Rec-Transpose. Now, no hypothesis is made on the shape or size of A. Thus, the input matrix A is rectangular with `m` rows and `n` columns, as in Question 1. We present the principle below. Please note that the algorithm takes as input parameters the matrix A and the output transpose $B =^t A$. This implies that the algorithm modifies B and works out-of-place as the algorithm in Question 1.

- If $n \geq m$, the Rec-Transpose algorithm partitions

  $$A = (A_1 \ A_2) \ , \quad B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$$

  and recursively executes Rec-Transpose(A_1, B_1) and Rec-Transpose(A_2, B_2).

- If $m > n$, the Rec-Transpose algorithm partitions

  $$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \ , \quad B = (B_1 \ B_2)$$

  and recursively executes Rec-Transpose(A_1, B_1) and Rec-Transpose(A_2, B_2).

You are required to write a `Julia` function that

- takes as input two positive integer value m and n,
- generates an $m \times n$ matrix `a` with random entries of type `int` with values in the range $0 \cdots n-1$.
- computes the transposed matrix ${}^tA$ using the Rec-Transpose divide and conquer scheme.

Question 4. [10 points] You are required to compare experimentally the running times of the above three implementations of matrix transposition. This implies to choose a series of matrix sizes and apply these three implementations to each selected size.

Submission instructions.

Format: Problems 1, 2 and 3 involve programming with Julia: the corresponding programs must be submitted as three input **text** files to be called `Pb1.jl`, `Pb2.jl`, `Pb3.jl` respectively. Each of these three files must be a valid input file for Julia. In addition, each user defined function must be documented, using comments. Moreover, the answers to Questions 1 and 4 of Problem 1, and Question 4 of Problem 3 should be typed and submitted as a PDF file called `OtherAnswers.pdf`. No format other than PDF will be accepted. To summarize: each assignment submission consists of four files: `Pb1.jl`, `Pb2.jl`, `Pb3.jl` and `OtherAnswers.pdf`.

Submission: The assignment should be returned to the instructor **and** the TA by email.

Collaboration. You are expected to do this assignment *on your own* without assistance from anyone else in the class. However, you can use literature and if you do so, briefly list your references in the assignment. Be careful! You might find on the web solutions to our problems that are not appropriate. So please, avoid those traps and work out the solutions by yourself. You should not hesitate to contact the instructor or the TA if you have any question regarding this assignment. We will be more than happy to help.

Marking. This assignment will be marked out of 100. A 10 % bonus will be given if your answers are clearly organized, precise and concise. Messy assignments (unclear statements, lack of correctness in the reasoning, many typographical or language mistakes) may give rise to a 10 % malus.