CS3101 Due: 13-th of February 2015

Problem Set 1

CS3101 *Submission instructions on last page*

PROBBLEM 1. [70 points] `Merge sort` is a particularly good example of the divide and conquer algorithmic paradigm. `Merge_sort` is a recursive procedure that uses at most $O(n\log(n))$ comparisons for sorting an an input array of n entries (typically numbers). To sort an array of n elements, the algorithm performs the following steps in sequence:

- if $n < 2$ then the array is already sorted.
- otherwise, we perform the following three steps in sequence:
  - sort the left half of the array using `Merge_sort` recursively,
  - sort the right half of the array using `Merge_sort` recursively,
  - `Merge` the sorted left and right halves.

Usually, programmers simply parallelize the above algorithm by spawning one of the recursive calls. Unfortunately, this only provides a (theoretical) speedup factor like $\Theta(\log(n))$, which is not satisfactory on fat multi-core nodes, say with 16 or 32 cores and input array with a few millions of entries. However, if the `Merge` subroutine is parallelized as well, then the (theoretical) speedup factor jumps to $\Theta(n/\log^2(n))$, as we shall explain in class.

In this assignment, we propose to write a `Julia` program parallelizing the `Merge_sort` algorithm using this enhanced scheme, thus by parallelizing the `Merge` subroutine as well.

The key ideas for parallelizing the `Merge` subroutine can be found on Page 17 of `http://www.csd.uwo.ca/~moreno/cs3101_Winter_2013/Analysis_of_Multithreaded_Algorithms-CS3101.pdf`.

Question 1. [10 points] Write a serial `Julia` function `Merge_Sort_Serial` which

- takes as input an array and
- returns this array sorted.

Therefore, your serial algorithm should work in place. However, the `Merge` subroutine can allocate extra storage.

Question 2. [20 points] Write a parallel `Julia` function `Merge_Parallel` (using the language constructs `spawn` and `fetch`) according to the algorithm presented in `http://www.csd.uwo.ca/~moreno/cs3101_Winter_2013/Analysis_of_DnC_Algorithms.pdf` To do so, you will need a `Julia` function implementing `BinarySearch`. You can take advantage of one of the `search` function at `http://web.info.uvt.ro/~dzaharie/algeng/alg2011_sem6.pdf`. Note that the *binary_search* algorithm (as described at `http://en.wikipedia.org/wiki/Binary_search_algorithm`) does not do exactly what we want, since it may return `KEY_NOT_FOUND`.

Question 3. [20 points] Write a parallel `Julia` function `Merge_Sort_Parallel` (using the language constructs `spawn` and `fetch`) with the same specification as `Merge_Sort_Serial`.

Question 4. [20 points] Compare experimentally the running times of `Merge_Sort_Serial` and `Merge_Sort_Parallel`

- for input sizes 2^i for $i = 10$ to $i = 24$,
- for a number of cores varying from 1 to 8.

Using the `Winston` package for displaying speedup curves is required. To have examples of those, please look at the elements of solutions for `Lab 3` for the CS2101 course from `http://www.csd.uwo.ca/~moreno/cs2101a_moreno/index.html`.

PROBBLEM 2. [30 points]

In this problem, we propose to use `Julia` for plotting *Mandelbrot sets* and *Julia sets*. One can find related resources here:

- `https://www.bowdoin.edu/~dfrancis/askanerd/mandelbrot/`,
- `http://rosettacode.org/wiki/Mandelbrot_set#Julia`,
- `http://en.wikipedia.org/wiki/Mandelbrot_set`.

The basic formula for constructing a Mandelbrot set or a *Julia set* is:

$$Z = Z^2 + C$$

which is used for defining a recurrence relation. The Mandelbrot set (or *Julia set*) is determined by iterating with this equation. With initial values for Z and C, one can compute a new value for Z, then we use that value of Z in the formula and get a new for Z, and so on.

In this assignment, we propose to write `Julia` programs for computing and plotting Mandelbrot sets. Our starting point is a `Julia` program for computing and plotting a Julia set which can be found at

- `http://mathemartician.blogspot.ca/2012/07/julia-set-in-julia.html`

Other sources of interest for developing implementation are:

- `http://rosettacode.org/wiki/Mandelbrot_set#Julia` (serial implementation)
- `https://github.com/dcjones/Gadfly.jl` (plotting method).

In order to generate beautiful pictures, we propose to use `gnuplot`, see `http://www.gnuplot.info/`. To help you using `gnuplot`, you can adapt the `gnuplot` script `testplt.sh` posted on the assignment page on the CS3101 course web site. The input data file for `gnuplot` is (essentially) a list of points to plot, where each point is expected a triple $(x, y, value)$:

- x is the value of x-axis,
- y is the value of y-axis,

- *value* is used to generate color.

So the output of a `Julia` program generating input data for `gnuplot` is a files containing a list of such points. We can organize them into three columns:

- the first column is the set of x-coordinates,
- the second column is the set of y-coordinates,
- the third column is the value used to generate color corresponding to that point.

The file `testplt.sh` is a minimal `gnuplot` code to complete our job:

```
 1 #!/bin/bash
 2 gnuplot << EOF
 3 set term png
 4 set output "test.png"
 5 set xrange [0:10]
 6 set yrange [0:10]
 7
 8 set xtics 0,2,10
 9 set ytics 0,2,10
10 set palette
11 set view map
12 splot 'xyz.tsv' with points palette pt 5 ps 0.5
13
14 EOF
```

Some explanations:

- lines 3 and 4 set the output picture,
- lines 5 and 6 set the length along the x- and y-axes,
- lines 8 and 9 set the scales along the x- and y-axes,
- at line 10 the `palette` command will create an enhanced color scale,
- line 11 sets the map view
- line 12 reads the data from input file `xyz.tsv`,
  - "pt 5" specifies pointtype 5, meaning a triangle;
  - "ps 0.5" specifies pointsize 0.5, meaning so small that it almost looks like a filled dot.

Question 1. [15 points] Adapt the code provided at

- `http://mathemartician.blogspot.ca/2012/07/julia-set-in-julia.html`

so as to use shared arrays instead of distributed arrays. Collect and plot experimental data using the same input sizes as in the web site.

Question 2. [15 points] Adapt the code provided at

- `http://mathemartician.blogspot.ca/2012/07/julia-set-in-julia.html`

so as to generate an input data file for `gnuplot`, following the above guidelines.

Submission instructions.

Format: Problems 1 and 2 involve programming with Julia: they must be submitted as two input files to be called `Pb1.jl` and `Pb2.jl`, respectively. Each of these two files must be a valid input file for Julia. In addition, each user defined function must be **documented.**

Submission: The assignment should be returned to the instructor 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. For instance, because the cache memory model is different. So please, avoid those traps and work out the solutions by yourself. You should not hesitate to contact the instructor if you have any question regarding this assignment. I 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.