Exercises for lab 3 of CS2101a

Instructor: Marc Moreno Maza, TA: Li Zhang

Thursday 25 September 2014

1 Exercise 1

The following is a `Julia` function for computing the sequence of the Fibonacci numbers in a serial fashion:

```
@everywhere function fib(n)
                 if (n < 2) then
                     return n
                 else return fib(n-1) + fib(n-2)
                 end
              end
```

This other `Julia` function, seen in class, computes the sequence of the Fibonacci numbers in a parallel fashion:

```
@everywhere function fib_parallel(n)
          if (n < 40) then
              return fib(n)
          else
              x = @spawn fib_parallel(n-1)
              y = fib_parallel(n-2)
              return fetch(x) + y
          end
       end
```

1. Study experimentally the influence of the threshold (for switching between parallel and serial execution, which is 40 in the above code) on the speedup factor w.r.t. purely serial code. You could try other thresholds, say 10, 15, 20, 25, 30, 35, 45 for n between `35` and `45.`
2. Use the `Winston` package for plotting your experimental data.
3. If you are a Matlab user, here's `fib(n)` in Matlab for you to perform the same measurement.

```
function f=fib(n)
  if n <= 2
    f=1.0;
  else
    f=fib(n-1)+fib(n-2);
  end
end
```

Hint 1: For experimenting with thresholds, it is convenient to make the threshold an input parameter of the `fib_parallel(n)` function. For instance:

```
@everywhere function fib_parallel(n, t)
          if (n < t) then
              return fib(n)
          else
              x = @spawn fib_parallel(n-1, t)
              y = fib_parallel(n-2, t)
              return fetch(x) + y
          end
       end
```

Hint 2: Here's an example of a `Julia` session for collecting timings with the above `fib` function. Note that this implies using the auxiliary `runningtime` below. Once the timings are collected in the array `T`, one passes the arrays `T` and `N` to the `plot` commands of the `Winston` package.

```
@everywhere function fib(n)
          if (n < 2) then
              return n
          else return fib(n-1) + fib(n-2)
          end
       end

function runningtime(f,n)
       tic()
       y = f(n)
       t = toc()
       t
       end

T = [runningtime(fib,30+i) for i=1:10]

N = [30+i for i=1:10]

using Winston
```

```
plot(N, T)
```

Hint 3: For plotting in `Julia`, the `Winston` package is convenient. See the documentation at `http://homerreid.dyndns.org/teaching/18.330/JuliaProgramming.shtml#Plotting` Please note that this documentation has specific installation instructions for `Ubuntu`, `Mac OSX` and `Windoows`. The example in the section "More complex plots" shows how to plot several graphs in the same frame. It also shows how to label those graphs and how to save the generated plots into a file!

2 Exercise 2

The following is a `C` function for computing the product of two square matrices (in a naive and inefficient way). Note that in this `C` code, the output result `AB` is passed as an input argument. If you are not familiar with the `C` programming language, just consider this function as a pseudo-code or formula. If you need to review matrix multiplication, visit `http://en.wikipedia.org/wiki/Matrix_multiplication`

```
void mmult(double A[M][M],double B[M][M], double AB[M][M]){
  int i,j,k;
  for(i=0; i<M; i++)
    for(j=0; j<M; j++){
      AB[i][j] = 0;
      for(k=0; k<M; k++)
          AB[i][j] += A[i][k]*B[k][j];
    }
}
```

1. Write a Julia program that computes `mmult(A,B)` where `A` and `B` are two square matrices of the same order `M` (using the same naive and inefficient algorithm as in C).

2. Using the `@time` macro, measure the running times of your Julia function `mmult(A,B)` for `M` equal to `500`, `1000`, `1500`, `2000`. Your input matrices will be randomly generated using `rand(M,M)`.

3. If you are a Matlab user, here's `mmult(A,B,C)` in Matlab for you to perform the same measurement.

```
function C=mmult(A,B,C)
  [M,N] = size(A);
  for i=1:M
    for j=1:M
      for k=1:M
        C(i,j) = C(i,j) + A(i,k)*B(k,j);
```

```
            end
          end
        end
      end
```

3 Exercise 3

The following Julia function implements a famous algorithm for sorting called *quicksort*. Look at its wikipedia page to learn how this algorithm works! `http://en.wikipedia.org/wiki/Quicksort`

```
function qsort!(a,lo,hi)
    i, j = lo, hi
    while i < hi
        pivot = a[(lo+hi)>>>1]
        while i <= j
            while a[i] < pivot; i = i+1; end
            while a[j] > pivot; j = j-1; end
            if i <= j
                a[i], a[j] = a[j], a[i]
                i, j = i+1, j-1
            end
        end
        if lo < j; qsort!(a,lo,j); end
        lo, j = i, hi
    end
    return a
end

function sortperf(n)
    qsort!(rand(n), 1, n)
end
@time sortperf(5000)
```

1. Go through the code and make sure you agree that it is an implementation of the algorithm presented in the wikipedia page.
2. Record the running time of `sortperf`(2^e∗1000000) for $e = 0, 1, 2, 3, 4, 5, 6, 7, 8$.
3. Are your results coherent with the theoretical prediction (see the section *Formal analysis* in the wikipedia page) that sorting of an array of size n with *quicksort* runs in a time asymptotically proportional to $O(n\log(n))$?