Solution for the Exercise 3 of Lab 4

We provide below a `Julia` code for a function `dacmm` which

- computes the product of two square matrices `A` and `B` of order `s` and,
- writes the result in a matrix `C` (which is also square of order `s`).

This `Julia`'s function follows the principle given in the statement of Exercise 3 of Lab 4. To implement this principle, one needs parameters recording which block is currently being considered in either `A`, `B` or `C`:

- `(i0,i1)` are the coordinates of the top-left corner in the current block of `A`,
- `(j0,j1)` are the coordinates of the top-left corner in the current block of `B`,
- `(k0,k1)` are the coordinates of the top-left corner in the current block of `C`.

In addition, to make the code efficient, we have added an extra parameter `X` for the *base-case*. In the description of Exercise 3 of Lab 4, this value is 2. The base-case `X` is defined as the maximum order for which matrices are multiplied using the naive matrix multiplication method. In practice, the base-case is often a number like 8, 16, 32 or 64. In fact, the theory that we developed in class suggests that the base-case should be the largest (power of 2) `X` such that three square matrices of order `X` fit in L1 cache.

In the experimental results reported below, you can see that with `X` = 8 and `s` = 1024, the divide-and-conquer matrix multiplication method (as implemented in `dacmm`) is clearly faster than the naive matrix multiplication method.

This observation is coherent with what we discussed in the chapter about *cache memories*. In fact, the divide-and-conquer matrix multiplication method implemented in `dacmm` is similar to the matrix multiplication method based on a blocking strategy: they both partition `A`, `B`, `C` into blocks and compute the product matrix `C` block-wise.

```
============================================================
divide and conquer version:
C = A*B
(i0,i1): coordinates of the top-left corner of the current
         block from Matrix A

(j0,j1): coordinates of the top-left corner of the current
         block from Matrix B

(k0,k1): coordinates of the top-left corner of the current
         block from Matrix C
```

```
s: order of the matrices A, B, C (note that this
   parameter is divided by 2, 4, 8, in the subsequent
   recursive calls)

X: the size of basecase (can be taken equal to 2 in order
   to make the story simple, but in practice X should
   be a bit larger for various optimization reasons
   that we shall discuss in class).


==========================================================
function dacmm(i0, i1, j0, j1, k0, k1, A, B, C, s, X)
        if s > X
                s = s/2
                dacmm(i0, i1, j0, j1, k0, k1, A, B, C, s,X)
                dacmm(i0, i1, j0, j1+s, k0, k1+s, A, B, C, s,X)
                dacmm(i0+s, i1, j0, j1, k0+s, k1, A, B, C, s,X)
                dacmm(i0+s, i1, j0, j1+s, k0+s, k1+s, A, B, C, s,X)
                dacmm(i0, i1+s, j0+s, j1, k0, k1, A, B, C, s,X)
                dacmm(i0, i1+s, j0+s, j1+s, k0, k1+s, A, B, C, s,X)
                dacmm(i0+s, i1+s, j0+s, j1, k0+s, k1, A, B, C, s,X)
                dacmm(i0+s, i1+s, j0+s, j1+s, k0+s, k1+s, A, B, C, s,X)
         else
                for  i= 1:s, j=1:s, k=1:s
                         C[i+k0,k1+j] += A[i+i0,i1+k] * B[k+j0,j1+j]
                end
          end
end
=============================================================

n=1024
base=8
A = [rem(rand(Int32),5) for i =1:n, j = 1:n];
B = [rem(rand(Int32),5) for i =1:n, j = 1:n];
C = zeros(n,n);
@time dacmm(0, 0, 0, 0, 0, 0, A, B, C, n, base)


============================================================
naive version:

function mmult(A,B)
(M,N) = size(A)
C = zeros(M,M)
for i=1:M
for j=1:M
```

```
for k=1:M
C[i,j] += A[i,k]*B[k,j]
end
end
end
C
end
===============================================================

@time mmult(A,B)

===============================================================
 @time dacmm(0, 0, 0, 0, 0, 0, A, B, C, n, base, n)
elapsed time: 8.579287052154541 seconds

@time mmult(A,B)
elapsed time: 12.142310857772827 seconds
```