Chapter 24: Multithreaded Algorithms

Examples for some multithreaded algorithms.

Section 24.1: Square matrix multiplication multithread

```
multiply-square-matrix-parallel(A, B)
    n = A.lines
    C = Matrix(n,n) //create a new matrix n*n
    parallel for i = 1 to n
        parallel for j = 1 to n
            C[i][j] = 0
            pour k = 1 to n
                C[i][j] = C[i][j] + A[i][k]*B[k][j]
    return C
```

Section 24.2: Multiplication matrix vector multithread

```
matrix-vector(A,x)
    n = A.lines
    y = Vector(n) //create a new vector of length n
    parallel for i = 1 to n
        y[i] = 0
    parallel for i = 1 to n
        for j = 1 to n
            y[i] = y[i] + A[i][j]*x[j]
    return y
```

Section 24.3: merge-sort multithread

A is an array and *p* and *q* indexes of the array such as you gonna sort the sub-array *A[p..r]*. *B* is a sub-array which will be populated by the sort.

A call to *p-merge-sort(A,p,r,B,s)* sorts elements from *A[p..r]* and put them in *B[s..s+r-p]*.

```
p-merge-sort(A,p,r,B,s)
    n = r-p+1
    if n==1
        B[s] = A[p]
    else
        T = new Array(n) //create a new array T of size n
        q = floor((p+r)/2))
        q_prime = q-p+1
        spawn p-merge-sort(A,p,q,T,1)
        p-merge-sort(A,q+1,r,T,q_prime+1)
        sync
        p-merge(T,1,q_prime,q_prime+1,n,B,s)
```

Here is the auxiliary function that performs the merge in parallel.
p-merge assumes that the two sub-arrays to merge are in the same array but doesn't assume they are adjacent in the array. That's why we need *p1,r1,p2,r2*.

```
p-merge(T,p1,r1,p2,r2,A,p3)
    n1 = r1-p1+1
    n2 = r2-p2+1
    if n1<n2     //check if n1>=n2
```

```
        permute p1 and p2
        permute r1 and r2
        permute n1 and n2
    if n1==0     //both empty?
        return
    else
        q1 = floor((p1+r1)/2)
        q2 = dichotomic-search(T[q1],T,p2,r2)
        q3 = p3 + (q1-p1) + (q2-p2)
        A[q3] = T[q1]
        spawn p-merge(T,p1,q1-1,p2,q2-1,A,p3)
        p-merge(T,q1+1,r1,q2,r2,A,q3+1)
        sync
```

And here is the auxiliary function dichotomic-search.

x is the key to look for in the sub-array T[p..r].

```
dichotomic-search(x,T,p,r)
    inf = p
    sup = max(p,r+1)
    while inf<sup
        half = floor((inf+sup)/2)
        if x<=T[half]
            sup = half
        else
            inf = half+1
    return sup
```