CIS 786: Programming Component : Homework 5/Problem 1 (100 points)

1 LU decomposition with pivoting in parallel (50 points)

The purpose of this assignment is to do parallel LU decomposition using BSPlib or MPI (the choice is yours).

The following header file `alexmult.h` will contain some common definitions that will be used in this assignment. You are not allowed in any way to add or remove information from that file or edit it.

```
/* This is  alexmult.h */
typedef float FTYPE;
/* End  of alexmult.h */
```

```
void  lucmpar (FTYPE *A, int n , int *indx, FTYPE *d, int *stripe);
```

The arguments of `lucmpar` are those of `lucmp` of Homework 4 except the last one. Input A is valid only on processor 0 (HW 4, matrix multiplication) and holds the original matrix A in column major form. You need to distribute it to the p processors that will participate in the computation (available through `bsp_nprocs` in BSPlib or similarly in MPI). You must use a column-based cyclic distribution. The thickness of a stripe however is your choice. This is the value available in `stripe`. The result of the decomposition at the end of the computation will be available at A as well (overwriting). The value of `stripe` that you used in your code, must be returned at the completion of `lucmpar`. When `lucmpar` is called the location pointed by `stripe` may contain nonsense. It is up to you to determine the best choice for a given n and tell us about your choice at the end.

What determines your grade

- Effort (20 points).
- Correctness (20 points). List bugs if any, or assumptions made during the course of the design. Otherwise, do not expect any credit. Assumptions can be in the form of comments within your code.
- Time efficiency (10 points). For a program that computes C correctly the parallel time will be compared to the performance of the sequential algorithm written by you (the winner of HW4 competition). How many processor will be used for the testing will not be revealed. You may assume that it will be between 4 (inclusive) and 16 (inclusive). (Testing may occur on a homogeneous 16-machine, 32-cpu cluster). If the efficiency of your implementation is between 25 and 50 percent, you get 5 points, between 50 and 75 percent you get 10 points.

Note. 50 bonus points is the prize for the author of the fastest correct implementation.

2 Parallel Matrix multiplication with MPI (50 points)

Do Homework 4, section 2 with MPI. Use RMA primitives for communication (20 points) and Message passing primitives (20 points). The two functions called `mmcmRMA` and `mmcmMPI` have the same interface as `mmcm` of Homework 4.

Note. 50 bonus points is the prize for the author of the fastest correct implementation!

3 Parallel Sorting (Homework 1, Problem 3) (50 points)

Implement the algorithm outlined in the solutions of Homework 1, Problem 3.

```
#include "alexmult.h"
 pselsort (FTYPE *inseq, FTYPE *outseq, int  n );
```

The input/output will be simplified and the same conventions as those used in matrix multiplication will be followed. The input sequence consisting of `n` keys of type `FTYPE` are stored in array `inseq` in processor 0. No assumption can be made whether n is or is not a multiple of p (number of processors). You need to distribute `inseq` to the remaining processors as evenly as possible. How? One suggestion (easy) is to send n/p keys to processors $0, \ldots, p-2$ and the remaining $n-(n/p)*(p-1)$ keys to processor $p-1$. For sequential sorting you have one choice: the intstructor supplied `bubble` bubble-sort function that has the interface of the Standard C library `qsort` function. Familiarize yourselves with that interface. You CAN NOT USE ANYTHING ELSE. The result of the sorting should be copied to processor 0 in output array `outseq`. You may assume that `outseq` has been succesfully allocated (in processor 0) and also initialized to 0 using `memset`. If you don't know what the latter does do a `man` and inspect the code in Homework 4.

Note. 50 bonus points is the prize for the author of the fastest correct implementation!