

```

/* =====
 *
 * CS 566 - Assignment 03
 * Camillo Lugaresi, Cosmin Stroe
 *
 * This code implements LU decomposition using a ring topology,
 * with pipelined communication on a cluster using MPI.
 *
 * ===== */

#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>
#include <stddef.h>
#include <unistd.h>
#include <string.h>
#include "common.h"
#include "cannon.h"

    rmatrix {
        int n;
        int m;
        int *data;
};

    rfmatrix {
        int n;
        int m;
        number_type *data;
};

void print_rfmatrix(        rfmatrix *m)
{
    int i;
    int count = m->n*m->m;
    number_type *p = m->data;

    (i = 1; i <= count; i++) {
        printf("% f", *p);
        (i % m->n == 0) printf("\n");
        printf(" ");
        p++;
    }
}

    cpivot {
        int column;
        number_type value;
};

int setup_cpivot_struct(MPI_Datatype *pivot_type)
{
    int blocklen[2] = {1,1};
    MPI_Aint offsets[2] = { 0, offsetof(        cpivot, value) };
    MPI_Datatype types[2] = { MPI_INT, MPI_number_type };
    MPI_Type_create_struct(2, blocklen, offsets, types, pivot_type);
    MPI_Type_commit(pivot_type);
    0;
}

void pipeline_down(        problem *info, int start, void *buf, int count, MPI_Datatype datatype, int tag,
MPI_Request *req)
{
    MPI_Status status;
    int up = (info->rank+(info->p)-1) % (info->p);

```

```

    int down = (info->rank+1)%(info->p);
    (info->rank != start) {
        MPI_Recv(buf, count, datatype, up, tag, info->rowring, &status);
    }
    (down != start) {
        MPI_Isend(buf, count, datatype, down, tag, info->rowring, req);
    }
}

void LU_decomp_1d(
    problem *info,
    rfmatrix *X, int *reorder, MPI_Datatype pivot_type)
{
    int gr, r, c;
    number_type *pivot_row = malloc(
        (*pivot_row)*info->n);
    MPI_Request pivot_req = MPI_REQUEST_NULL, pivot_row_req = MPI_REQUEST_NULL;
    MPI_Status status;

    (gr = 0; gr < info->n; gr++) {
        /* global row number */
        (pivot_req != MPI_REQUEST_NULL) MPI_Wait(&pivot_req, &status);

        int rproc = gr % (info->p);
        r = gr / (info->p);
        /* proc with this row */
        /* row within this proc */

        /* we do partial pivoting, so the pivot is on this row */
        cpivot pivot = { -1, 0. };
        (info->rank == rproc) {
            (c = 0; c < info->n; c++) {
                (reorder[c] > gr && (pivot.value == 0 || fabs(CELL(X, r, c)) > fabs
(pivot.value))) {
                    pivot.column = c;
                    pivot.value = CELL(X, r, c);
                }
            }
        }
        pipeline_down(info, rproc, &pivot, 1, pivot_type, 89, &pivot_req);

        /* fill in reorder */
        reorder[pivot.column] = gr;

        /* distribute a vector */
        (pivot_row_req != MPI_REQUEST_NULL) MPI_Wait(&pivot_row_req, &status);
        (info->rank == rproc) memcpy(pivot_row, &CELL(X, r, 0),
            (*pivot_row)*info->n);
        pipeline_down(info, rproc, pivot_row, info->n, MPI_number_type, 94, &pivot_row_req);

        /* elimination */
        int startr = gr / (info->p);
        (info->rank <= rproc) startr++;

        (r = startr; r < info->rowblksz; r++) {
            number_type m = CELL(X, r, pivot.column) / pivot.value;
            CELL(X, r, pivot.column) = m;

            (c = 0; c < info->n; c++) {
                (reorder[c] > gr) { /* this also excludes the pivot column */
                    CELL(X, r, c) -= m*pivot_row[c];
                }
            }
        }
    }
}

double convert_time;
double lu_time;

/* note that this only returns the correct value in processor with rank 0 */
number_type lu1d_determinant(
    problem *info)
{

```

```

int i;
info->rowblkksz = info->n / info->p;
int *allblockdata, *blockdata;
int rowblkcells = info->n*info->rowblkksz;

    (info->rank == 0) {
        allblockdata = malloc(      (*allblockdata)*info->n*info->n);
        matrix_to_rowblocks_cyclic(&info->Xpow, allblockdata, info->rowblkksz);
    }

/* setup rowblock ring */
int dims[1] = {info->p};
int periods[1] = {1}; /* wraparound */
MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &info->rowring);

/* distribute row blocks */
blockdata = malloc(      (*blockdata)*rowblkcells);
MPI_Scatter(allblockdata, rowblkcells, MPI_INT, blockdata, rowblkcells, MPI_INT, 0, info->rowring);

/* convert to float */
rfmatrix fblk;
fblk.n = info->n;
fblk.m = info->rowblkksz;
fblk.data = malloc(      (*fblk.data)*rowblkcells);
    (i = 0; i < rowblkcells; i++) fblk.data[i] = blockdata[i];
convert_time = MPI_Wtime();

/* setup data type for pivoting */
MPI_Datatype pivot_type;
setup_cpivot_struct(&pivot_type);

/* prepare reorder array */
int *reorder = malloc(info->n *      (*reorder));
    (i = 0; i < info->n; i++) reorder[i] = INT_MAX;

LU_decomp_ld(info, &fblk, reorder, pivot_type);
lu_time = MPI_Wtime();

/* calculate the determinant */
number_type prod = 1.0;
    (i = 0; i < info->n; i++) {
        int gr = reorder[i];
        int rproc = gr % (info->p); /* proc with this row */
        (rproc == info->rank) {
            int r = gr / (info->p); /* row within this proc */
            prod *= CELL(&fblk, r, i);
        }
    }
number_type determinant;
MPI_Reduce(&prod, &determinant, 1, MPI_number_type, MPI_PROD, 0, info->rowring);

/* we must adjust the determinant's sign based on the permutations */
    (info->rank == 0) {
        (count_swaps(reorder, info->n) % 2) determinant *= -1;
    }

    determinant;
}

```