```c
/* ======================================================================
 *
 * CS 566 - Assignment 03
 * Camillo Lugaresi, Cosmin Stroe
 *
 * This code implements LU decomposition using a 2D mesh 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 "common.h"
#include "cannon.h"

void alloc_fmatrix(struct fmatrix *m, int n)
{
        m->n = n;
        m->data = malloc(sizeof(*m->data) * m->n * m->n);
}

void best_pivot( void *invec, void *inoutvec, int *len, MPI_Datatype *datatype)
{
        struct pivot *a = invec, *b = inoutvec;
        int i;
        /* it's important to break ties, or different rows may believe they have
           the pivot */
        for (i = 0; i < *len; i++, a++, b++) {
                if ((fabs(a->value) > fabs(b->value)) ||
                        (fabs(a->value) == fabs(b->value) && a->row < b->row)) {
                        *b = *a;
                }
        }
}

int setup_pivot_struct(MPI_Datatype *pivot_type, MPI_Op *best_pivot_op)
{
        int blocklen[2] = {1,1};
        MPI_Aint offsets[2] = { 0, offsetof(struct pivot, 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);

        MPI_Op_create(best_pivot, 1, best_pivot_op);
        return 0;
}

double pivot_time = 0;
double pivot_allr_time = 0;
double pivot_bcast_time = 0;
double m_bcast_time = 0;
double a_bcast_time = 0;

void pipeline_right(struct problem *info, int pivot_h, void *buf, int count, MPI_Datatype datatype, int
tag, MPI_Request *req)
{
        MPI_Status status;

        if (info->coords[HDIM] > pivot_h && info->coords[HDIM] < info->sqp) {
                MPI_Recv(buf, count, datatype, info->coords[HDIM]-1, tag, info->hcomm, &status);
        }
        if (info->coords[HDIM] >= pivot_h && info->coords[HDIM] < info->sqp-1) {
```

```
                MPI_Isend(buf, count, datatype, info->coords[HDIM]+1, tag, info->hcomm, req);
        }
}

void LU_decomp(struct problem *info, struct fmatrix *X, int *reorder, MPI_Datatype pivot_type, MPI_Op
best_pivot_op)
{
        MPI_Request req_spiv, req_sa, req_sm;
        MPI_Status status;

        number_type *m = malloc(info->blksz * sizeof(*m));
        int diag;
        for (diag = 0; diag < info->n; diag++) {
                /* we do partial pivoting, so the proc with the pivot is on this column: */
                int pivot_h = diag / info->blksz;
                int r, c, i;

                double start_time = MPI_Wtime();
                double start_time2;

                struct pivot pivot = { -1, 0. };
                /* choose pivot across the column */
                if (info->coords[HDIM] == pivot_h) {
                        /* column with pivot in block */
                        int pivot_c = diag % info->blksz;
                        /* Argo doesn't want aliasing in allreduce */
                        struct pivot pivot_cand = { -1, 0. };
                        for (i = 0; i < info->blksz; i++) {
                                if (reorder[i] > diag && fabs(CELL(X, i, pivot_c)) > fabs
(pivot_cand.value)) {
                                        pivot_cand.row = info->blksz*info->coords[VDIM] + i;
                                        pivot_cand.value = CELL(X, i, pivot_c);
                                }
                        }
                        start_time2 = MPI_Wtime();
                        MPI_Allreduce(&pivot_cand, &pivot, 1, pivot_type, best_pivot_op, info->vcomm);
                        pivot_allr_time += MPI_Wtime() - start_time2;
                }
                /* broadcast pivot choice across row towards the right */
                start_time2 = MPI_Wtime();
                pipeline_right(info, pivot_h, &pivot, 1, pivot_type, 45, &req_spiv);
                pivot_bcast_time += MPI_Wtime() - start_time2;

                pivot_time += MPI_Wtime() - start_time;

                /* find rank of proc with pivot on the vertical communicator */
                int pivot_v = pivot.row / info->blksz;

                /* fill in reorder */
                if (info->coords[VDIM] == pivot_v) {
                        reorder[pivot.row % info->blksz] = diag;
                }

                /* calculate and distribute the ms */
                for (r = 0; r < info->blksz; r++) {
                        if (reorder[r] > diag) {
                                if (info->coords[HDIM] == pivot_h) {
                                        int pivot_c = diag % info->blksz;
                                        m[r] = CELL(X, r, pivot_c) / pivot.value;
                                        CELL(X, r, pivot_c) = m[r];
                                }
                                /* broadcast m towards right */
                                start_time = MPI_Wtime();
                                pipeline_right(info, pivot_h, &m[r], 1, MPI_number_type, 64, &req_sm);
                                m_bcast_time += MPI_Wtime() - start_time;
                        }
                }
```

```
                /* distribute the pivot row and eliminate */
                int startc = 0;
                if (info->coords[HDIM] == pivot_h) startc = (diag+1) % info->blksz;
                if (info->coords[HDIM] < pivot_h) startc = info->blksz;
                /* elimination */
                for (c = startc; c < info->blksz; c++) {
                        number_type a;
                        if (info->coords[VDIM] == pivot_v) {
                                a = CELL(X, pivot.row % info->blksz, c);
                        }

                        start_time = MPI_Wtime();

                        int up = (info->coords[VDIM]+info->sqp-1)%info->sqp;
                        int down = (info->coords[VDIM]+1)%info->sqp;
                        if (info->coords[VDIM] != pivot_v) {
                                MPI_Recv(&a, 1, MPI_number_type, up, 78, info->vcomm, &status);
                        }
                        if (down != pivot_v) {
                                MPI_Isend(&a, 1, MPI_number_type, down, 78, info->vcomm, &req_sa);
                        }

                        a_bcast_time += MPI_Wtime() - start_time;

                        for (r = 0; r < info->blksz; r++) {
                                if (reorder[r] > diag) {
                                        CELL(X, r,c) -= m[r]*a;
                                }
                        }
                        if (down != pivot_v) MPI_Wait(&req_sa, &status);
                }
        }
}

double convert_time;
double lu_time;

/* note that this only returns the correct value in processor with rank 0 */
number_type lu2d_determinant(struct problem *info)
{
        int i;

        /* compute the determinant using LU decomposition */
        /* setup data type for pivoting */
        MPI_Datatype pivot_type;
        MPI_Op best_pivot_op;
        setup_pivot_struct(&pivot_type, &best_pivot_op);

        int *reorder = malloc((info->rank == 0 ? info->n : info->blksz) * sizeof(*reorder));
        for (i = 0; i < info->blksz; i++) reorder[i] = INT_MAX;

        /* convert int matrix to float */
        struct fmatrix fC;
        alloc_fmatrix(&fC, info->C.n);
        for (i = 0; i < info->blkcells; i++) {
                fC.data[i] = info->C.data[i];
        }
        MPI_Barrier(info->mesh);
        convert_time = MPI_Wtime();

        /* do the LU */
        LU_decomp(info, &fC, reorder, pivot_type, best_pivot_op);

        MPI_Barrier(info->mesh);
        lu_time = MPI_Wtime();
        /* calculate the determinant */
```

```
	number_type prod = 1.0;
	for (i = 0; i < info->blksz; i++) {
		if (reorder[i] == INT_MAX) continue;
		int c = reorder[i] - info->coords[HDIM]*info->blksz;
		if (c >= 0 && c < info->blksz) {
			prod *= CELL(&fC, i, c);
		}
	}
	number_type determinant;
	MPI_Reduce(&prod, &determinant, 1, MPI_number_type, MPI_PROD, 0, info->mesh);

	/* we must adjust the determinant's sign based on the permutations */
	/* but only the rightmost column has the full reorder! */
	MPI_Request req;
	MPI_Status status;
	if (info->coords[HDIM] == info->sqp-1) {
		MPI_Isend(reorder, info->blksz, MPI_INT, 0, 75, info->hcomm, &req);
	}
	if (info->coords[HDIM] == 0) {
		MPI_Recv(reorder, info->blksz, MPI_INT, info->sqp-1, 75, info->hcomm, &status);

		MPI_Gather(reorder, info->blksz, MPI_INT, reorder, info->blksz, MPI_INT, 0, info->vcomm);
		if (info->rank == 0) {
			if (count_swaps(reorder, info->n) % 2) determinant *= -1;
		}
	}
	return determinant;
}
```