Отчет

Задание №2
Решение двумерного уравнения теплопроводности

(неявная схема, MPI+OpenMP)
приложение

выполнил
студент 510 группы ВМК МГУ
Кулагин Алексей

7. Таблицы и графики, содержащие сведения о размерах сеток, времени решения и эффективности распараллеливания

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k\left(x,y\right)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\left(x,y\right)\frac{\partial u}{\partial y}\right) + f\left(x,y,t\right), \quad (x,y) \in (0,1)\times(0,1), \quad t_{max} = 1$$

$$k(x,y) = 1, \qquad f(x,y,t) = -exp(-t)\cdot(1-\pi^2)\cdot(sin(\pi x)+cos(\pi y))$$

Точное решение: $u(x,y,t) = exp(-t)\cdot(sin(\pi x)+cos(\pi y)) + 10$

1) nx = 1000, ny = 1000, nt = 10

Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	21.030	1	100
64	256	0.215	24.430	38.173
128	512	0.460	11.411	8.915

```
edu-cmc-ski16-052@fen1:~/IMPLICIT_MPI_OPENMP> make accel
perl accel.pl
Nodes       Cores           Time                    Speed Up                Efficiency, %           Abs. error          Rel. error

1           1               21.03000000             1                       100                     0.01021743          0.00105858

64          256             0.21519931              24.4308404148694        38.1731881482334        0.01021743          0.00105858

128         512             0.46070012              11.4119787943619        8.91560843309527        0.01021743          0.00105858
```

2) nx = 2000, ny = 2000, nt = 10

Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	84.980	1	100
64	256	0.640	33.161	51.814
128	512	0.881	24.095	18.824

```
edu-cmc-ski16-052@fen1:~/IMPLICIT_MPI_OPENMP> make accel
perl accel.pl
Nodes       Cores           Time                    Speed Up                Efficiency, %           Abs. error          Rel. error

1           1               84.98000000             1                       100                     0.01026958          0.00106405

64          256             0.64065958              33.1611368396302        51.8142763119222        0.01026958          0.00106405

128         512             0.88168831              24.09581680855          18.8248568816797        0.01026958          0.00106405
```

3) nx = 3000, ny = 3000, nt = 10

Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	192.790	1	100
64	256	1.313	36.687	57.324
128	512	1.565	30.784	24.050

```
edu-cmc-ski16-052@fen1:~/IMPLICIT_MPI_OPENMP> make accel
perl accel.pl
Nodes       Cores           Time                    Speed Up                Efficiency, %           Abs. error          Rel. error

1           1               192.79000000            1                       100                     0.01028702          0.00106588

64          256             1.31372073              36.687782189446         57.3246596710094        0.01028702          0.00106588

128         512             1.56563626              30.7846089359223        24.0504757311893        0.01028702          0.00106588
```

4) nx = 4000, ny = 4000, nt = 10

Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	344.460	1	100
64	256	2.486	34.630	54.110
128	512	2.400	35.874	28.026

```
edu-cmc-ski16-052@fen1:~/IMPLICIT_MPI_OPENMP> make accel
perl accel.pl
Nodes       Cores           Time                    Speed Up                Efficiency, %           Abs. error          Rel. error

1           1               344.46000000            1                       100                     0.01029575          0.00106680

64          256             2.48665432              34.6308689983093        54.1107328098583        0.01029575          0.00106680

128         512             2.40047447              35.874157828473         28.0266858034945        0.01029575          0.00106680
```

5) nx = 5000, ny = 5000, nt = 10

Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	540.010	1	100
64	256	3.838	35.174	54.959
128	512	3.425	39.413	30.791

```
edu-cmc-ski16-052@fen1:~/IMPLICIT_MPI_OPENMP> make accel
perl accel.pl
Nodes       Cores           Time                    Speed Up                Efficiency, %           Abs. error          Rel. error

1           1               540.01000000            1                       100                     0.01030099          0.00106735

64          256             3.83812213              35.1741021852267        54.9595346644167        0.01030099          0.00106735

128         512             3.42531047              39.413215585097         30.791574675857         0.01030099          0.00106735
```

6) nx = 6000, ny = 6000, nt = 10

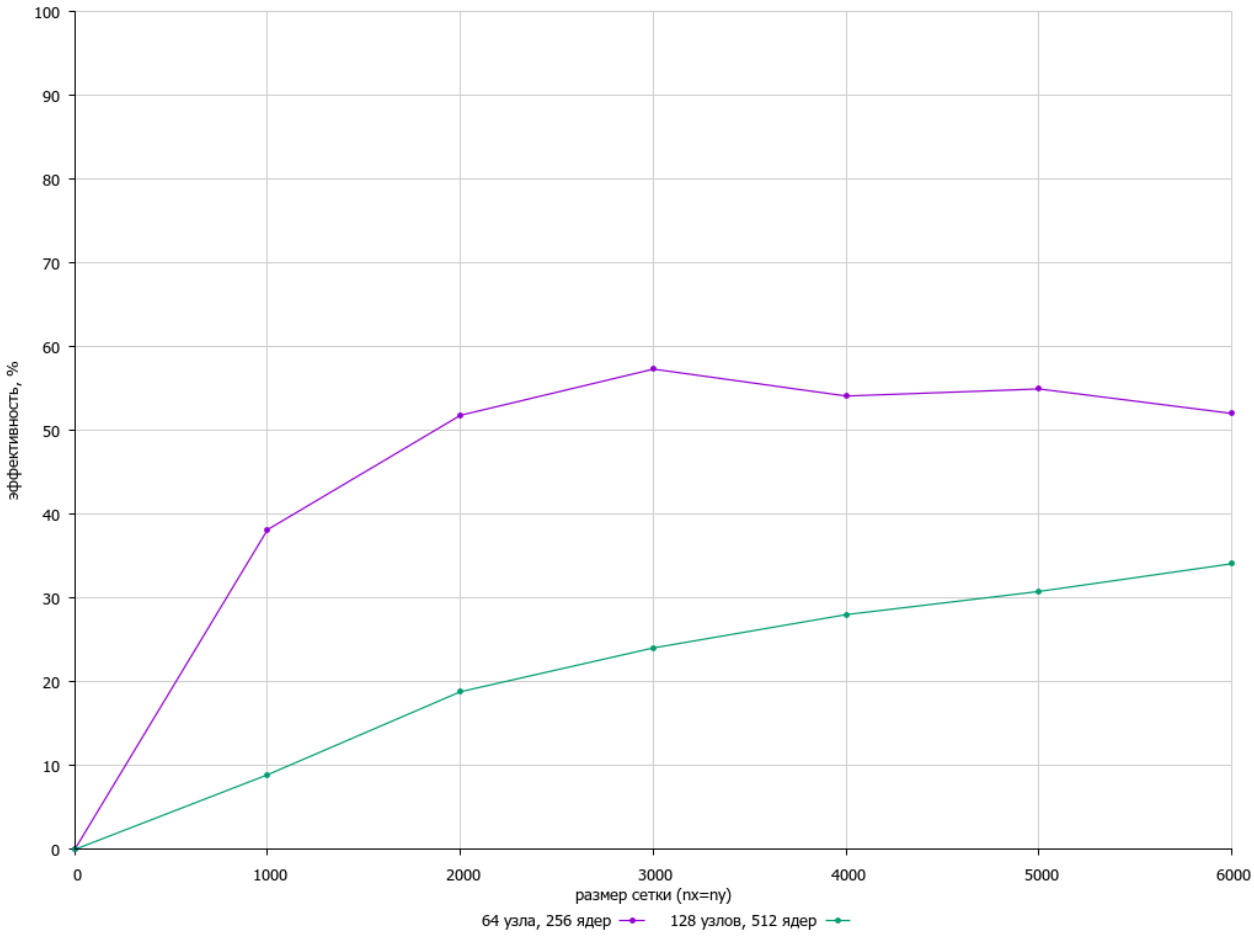
Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	776.620	1	100
64	256	5.830	33.301	52.034
128	512	4.448	43.649	34.100

```
edu-cmc-ski16-052@fen1:~/IMPLICIT_MPI_OPENMP> make accel
perl accel.pl
Nodes       Cores           Time                    Speed Up                Efficiency, %           Abs. error          Rel. error

1           1               776.62000000            1                       100                     0.01030449          0.00106771

64          256             5.83015086              33.30188268919          52.0341917018593        0.01030449          0.00106771

128         512             4.44807063              43.6492619272999        34.1009858807031        0.01030449          0.00106771
```



9. Дополнительные материалы

Компиляция и запуск

Makefile

```
compile_XL_O3:
mpixlcxx_r implicit_single.cpp -O3 -qhot -qstrict           -o single.x;
mpixlcxx_r implicit_mpi.cpp    -O3 -qhot -qstrict -qsmp=omp -o mpi.x;

compile_XL_O4:
mpixlcxx_r implicit_single.cpp -O4 -qhot -qstrict           -o single.x;
mpixlcxx_r implicit_mpi.cpp    -O4 -qhot -qstrict -qsmp=omp -o mpi.x;

compile_XL_O5:
mpixlcxx_r implicit_single.cpp -O5 -qhot -qstrict           -o single.x;
mpixlcxx_r implicit_mpi.cpp    -O5 -qhot -qstrict -qsmp=omp -o mpi.x;

run_omp_1:
mpisubmit.bg -n 1 ./single.x data --stdout=1.out --stderr=1.err

run_omp_2:
mpisubmit.bg -n 2 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=2.out --stderr=2.err

run_omp_4:
mpisubmit.bg -n 4 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=4.out --stderr=4.err

run_omp_8:
mpisubmit.bg -n 8 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=8.out --stderr=8.err

run_omp_16:
mpisubmit.bg -n 16 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=16.out --stderr=16.err

run_omp_32:
mpisubmit.bg -n 32 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=32.out --stderr=32.err

run_omp_64:
mpisubmit.bg -n 64 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=64.out --stderr=64.err

run_omp_128:
mpisubmit.bg -n 128 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=128.out --stderr=128.err

run_omp_256:
mpisubmit.bg -n 256 -w 00:10:00 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=256.out --stderr=256.err

run_omp_512:
mpisubmit.bg -n 512 -w 00:05:00 ./mpi.x data -mode SMP -env OMP_NUM_THREADS=4 --stdout=512.out --stderr=512.err

cancel:
llcancel -u edu-cmc-ski16-052

clean:
rm *.out *.err core* 2>/dev/null || echo > /dev/null

llq:
llq -u edu-cmc-ski16-052

accel:
perl accel.pl

run1_16_O3:
make cancel
make clean
make compile_XL_O3
make run_omp_1
for i in 2 4 8 16; do make run_omp_$$i; done;

run1_16_O4:
make cancel
```

```
make clean
make compile_XL_O4
make run_omp_1
for i in 2 4 8 16; do make run_omp_$$i; done;

run1_16_O5:
make cancel
make clean
make compile_XL_O5
make run_omp_1
for i in 2 4 8 16; do make run_omp_$$i; done;

run1_32_O3:
make cancel
make clean
make compile_XL_O3
make run_omp_1
for i in 2 4 8 16 32; do make run_omp_$$i; done;

run1_32_O4:
make cancel
make clean
make compile_XL_O4
make run_omp_1
for i in 2 4 8 16 32; do make run_omp_$$i; done;

run1_32_O5:
make cancel
make clean
make compile_XL_O5
make run_omp_1
for i in 2 4 8 16 32; do make run_omp_$$i; done;

run1_64_O3:
make cancel
make clean
make compile_XL_O3
make run_omp_1
for i in 2 4 8 16 32 64; do make run_omp_$$i; done;

run1_64_O4:
make cancel
make clean
make compile_XL_O4
make run_omp_1
for i in 2 4 8 16 32 64; do make run_omp_$$i; done;

run1_64_O5:
make cancel
make clean
make compile_XL_O5
make run_omp_1
for i in 2 4 8 16 32 64; do make run_omp_$$i; done;

run1_128_O3:
make cancel
make clean
make compile_XL_O3
make run_omp_1
for i in 2 4 8 16 32 64 128; do make run_omp_$$i; done;

run1_128_O4:
make cancel
make clean
make compile_XL_O4
make run_omp_1
for i in 2 4 8 16 32 64 128; do make run_omp_$$i; done;

run1_128_O5:
make cancel
make clean
```

```
make compile_XL_O5
make run_omp_1
for i in 2 4 8 16 32 64 128; do make run_omp_$$i; done;

run1_256_O3:
make cancel
make clean
make compile_XL_O3
make run_omp_1
for i in 2 4 8 16 32 64 128 256; do make run_omp_$$i; done;

run1_256_O4:
make cancel
make clean
make compile_XL_O4
make run_omp_1
for i in 2 4 8 16 32 64 128 256; do make run_omp_$$i; done;

run1_256_O5:
make cancel
make clean
make compile_XL_O5
make run_omp_1
for i in 2 4 8 16 32 64 128 256; do make run_omp_$$i; done;

run1_512_O3:
make cancel
make clean
make compile_XL_O3
make run_omp_1
for i in 2 4 8 16 32 64 128 256 512; do make run_omp_$$i; done;

run1_512_O4:
make cancel
make clean
make compile_XL_O4
make run_omp_1
for i in 2 4 8 16 32 64 128 256 512; do make run_omp_$$i; done;

run1_512_O5:
make cancel
make clean
make compile_XL_O5
make run_omp_1
for i in 2 4 8 16 32 64 128 256 512; do make run_omp_$$i; done;
```

Скрипт вывода времени работы, ускорения и эффективности вычислений для серии запусков (парсер выходных файлов):

accel.pl

```
use strict;
use warnings;

my $coeff = 80;

$coeff = $ARGV[0] if defined $ARGV[0];

my @files = ();

$files[$_] = (2 ** $_) for 0..10;

my %time = ();

my %accel = ();

my %param = ();

my %abs_err = ();
my %rel_err = ();

for my $name(@files){
    my $fh;

    if($name == 1){
        open($fh, "<", $name.".out") or last;
    }else{
        open($fh, "<", $name.".out") or next;
    }

    while(<$fh>){
        if(/^Time:\s+(.+)$/){
            $time{$name} = $1;
        }elsif(/^Abs. error:\s+(.+)$/){
            $abs_err{$name} = $1;
        }elsif(/^Rel. error:\s+(.+)$/){
            $rel_err{$name} = $1;
        }

        if(/^0:\s+(.+):\s+(.+)$/){
            $param{$1} = {} unless exists($param{$1});

            $param{$1}->{$name} = $2;
        }
    }

    close($fh);
}

if(keys %time){
    return unless defined $time{1};

    for(keys %time){
        $accel{$_} = $time{1} / $time{$_};
        $accel{$_} /= 4 if $_ != 1;
    }

    printf("%-10s %-15s %-25s %-25s %-25s %-20s %-20s\n", "Nodes", "Cores", "Time", "Speed Up", "Efficiency, %", "Abs. error", "

    for(sort {$a <=> $b} keys %time){
        if($_ == 1){
            printf("%-10s %-15s %-25s %-25s %-25s %-20s %-20s\n", $_, $_, $time{$_}, $accel{$_}, ($accel{$_} / $_ * 100), $abs_e
        }else{
            printf("%-10s %-15s %-25s %-25s %-25s %-20s %-20s\n", $_, $_ * 4, $time{$_}, $accel{$_}, ($accel{$_} / $_ * 100), $a
        }
    }
```

```
}else{
    print "No tasks completed\n";
}

print "\n";

if(keys %param){
    for my $i(sort {$a cmp $b} keys %param){
        return unless defined $param{$i}{1};

        my $h = $param{$i};

        my $show = 0;

        for(keys %{$h}){
            next if $h->{$_} == 0;

            if($h->{1} / $h->{$_} < $_ * $coeff / 100){
                $show = 1;
            }
        }

        if($show){
            printf("%-10s %-25s %-25s", "np", $i, "%");
            print "\n";

            for(sort {$a <=> $b} keys %{$h}){
                printf("%-10s %-25s %-25s\n", $_, $h->{$_}, $h->{1} / $h->{$_});
            }

            print "\n";
        }
    }
}
```

Пример использования:

`make run1_64_O3 (очистка очереди, компиляция, запуск на 1, 2, 4, 8, 16, 32, 64 ядра)`
`make accel (вывод результатов)`

10. Приложение

Листинг программы

1) Последовательная версия

implicit_single.cpp

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <mpi.h>

using namespace std;

// ------------------------
char* filename;
// ------------------------
double a1, b1;
double a2, b2;

double t_max;

int nx, ny, nt;

double dx, dy, dt;
double dx2, dy2;
double dt_1_2;

double* x;
double* y;
double* t;

double** u;
double** u_half;
// ------------------------
double *A_i, *B_i, *C_i;
double *A_j, *B_j, *C_j;
// ------------------------
double* F_i;
double* F_j;

double* X_i;
double* X_j;
// ------------------------
void print_grid();

void print_x();
void print_y();
void print_t();

void print_u();
void print_u_half();
void print_u_exact(int);
void print_error();

void free_memory();
// ------------------------

// BEGIN---------------------------------------- EXACT SOLUTION ---------------------------------------------
double u_exact(double x, double y, double t){
    // return exp(-t) * sin(M_PI * x) * cos(M_PI * y);
    return exp(-t) * (sin(M_PI * x) + cos(M_PI * y)) + 10;
}
// END------------------------------------------ EXACT SOLUTION ---------------------------------------------
```

```
// BEGIN---------------------------------------- F(x, y, t) ----------------------------------------------
double F(double x, double y, double t){
    // return exp(-t) * sin(M_PI * x) * cos(M_PI * y) * (2 * M_PI * M_PI - 1);
    return - exp(-t) * (sin(M_PI * x) + cos(M_PI * y)) * (1 - M_PI * M_PI);
}
// END------------------------------------------ F(x, y, t) ----------------------------------------------

// BEGIN---------------------------------------- READ DATA ----------------------------------------------
void read_data(){
    FILE* f;

    f = fopen(filename, "r");

    if(f == NULL){
        cout << "Cannot open file \"" << filename << "\"" << endl;

        exit(1);
    }

    int count = fscanf(f, "%lf %lf %lf %lf %lf %d %d %d", &a1, &b1, &a2, &b2, &t_max, &nx, &ny, &nt);

    if(count < 8){
        cout << "Wrong data format in file \"" << filename << "\"" << endl << endl;
        cout << "Usage: a1 b1 a2 b2" << endl;
        cout << "       t_max" << endl;
        cout << "       nx ny nt" << endl;
        cout << endl;

        cout << "Example: 0 1 0 1" << endl;
        cout << "         1" << endl;
        cout << "         10 10 5" << endl;
        cout << endl;

        exit(1);
    }

    fclose(f);
}
// END------------------------------------------ READ DATA ----------------------------------------------

// BEGIN---------------------------------------- THOMAS ALGORITHM ----------------------------------------
void solveMatrix(const int n, const double *_A, const double *_B, const double *_C, double *f, double *x){
    // ---------------------------
    double A[n];
    double B[n];
    double C[n];

    for(int i = 0; i < n; i++){
        A[i] = _A[i];
        B[i] = _B[i];
        C[i] = _C[i];
    }
    // ---------------------------
    double coeff;

    for(int i = 1; i < n; i++){
        coeff = A[i] / B[i-1];

        B[i] -= coeff * C[i-1];
        f[i] -= coeff * f[i-1];
    }

    x[n-1] = f[n-1] / B[n-1];

    for(int i = n-2; i >= 0; i--){
        x[i] = (f[i] - C[i] * x[i+1]) / B[i];
    }
}
// END------------------------------------------ THOMAS ALGORITHM ----------------------------------------
```

```
// BEGIN---------------------------------------- SET TRIAG COEFF ----------------------------------------
void set_triag_coeff(){
    A_i = new double[ny];
    B_i = new double[ny];
    C_i = new double[ny];

    for(int i = 0; i < ny; i++){
        A_i[i] = -1.0 / dy2;
        B_i[i] = 2.0 / dt + 2.0 / dy2;
        C_i[i] = -1.0 / dy2;
    }

    A_i[0] = 0;
    C_i[ny-1] = 0;
    // -------------------------------
    A_j = new double[nx];
    B_j = new double[nx];
    C_j = new double[nx];

    for(int i = 0; i < nx; i++){
        A_j[i] = -1.0 / dx2;
        B_j[i] = 2.0 / dt + 2.0 / dx2;
        C_j[i] = -1.0 / dx2;
    }

    A_j[0] = 0;
    C_j[nx-1] = 0;
}
// END------------------------------------------ SET TRIAG COEFF ----------------------------------------

// BEGIN---------------------------------------- SET GRID ----------------------------------------------
void set_grid(){
    read_data();

    assert(a1 < b1);
    assert(a2 < b2);

    assert(t_max > 0);

    assert(nx > 0);
    assert(ny > 0);
    assert(nt > 0);

    dx = (b1 - a1) / nx;
    dy = (b2 - a2) / ny;
    dt = t_max / nt;

    dx2 = dx * dx;
    dy2 = dy * dy;

    dt_1_2 = dt / 2.0;

    try{
        x = new double[nx+1];
        y = new double[ny+1];
        t = new double[nt+1];

        u = new double*[ny+1];
        u_half = new double*[ny+1];

        for(int i = 0; i <= ny; i++){
            u[i] = new double[nx+1];
            u_half[i] = new double[nx+1];

            for(int j = 0; j <= nx; j++){
                u[i][j] = 0;
                u_half[i][j] = 0;
            }
        }
    }catch(std::bad_alloc){
        cout << "Cannot allocate memory" << endl;
```

```
        exit(1);
    }

    for(int i = 0; i <= ny; i++){
        y[i] = a2 + i * dy;
    }

    for(int j = 0; j <= nx; j++){
        x[j] = a1 + j * dx;
    }

    for(int k = 0; k <= nt; k++){
        t[k] = k * dt;
    }
}
// END---------------------------------------- SET GRID ----------------------------------------------------

// BEGIN-------------------------------------- INITIAL CONDITIONS -------------------------------------------
double g_0(int j, int i){
    return u_exact(x[j], y[i], t[0]);
}

void set_initials(){
    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            u[i][j] = g_0(j, i);
        }
    }
}
// END---------------------------------------- INITIAL CONDITIONS -------------------------------------------

// BEGIN-------------------------------------- BOUNDARY CONDITIONS ------------------------------------------
void set_boundaries(double** _u, double _t){
    for(int i = 0; i <= ny; i++){
        _u[i][0] = u_exact(x[0], y[i], _t);
        _u[i][nx] = u_exact(x[nx], y[i], _t);
    }

    for(int j = 0; j <= nx; j++){
        _u[0][j] = u_exact(x[j], y[0], _t);
        _u[ny][j] = u_exact(x[j], y[ny], _t);
    }
}
// END---------------------------------------- BOUNDARY CONDITIONS -------------------------------------------

// BEGIN-------------------------------------- SOLVE -------------------------------------------------------
void solve(){
    set_triag_coeff();

    F_i = new double[ny-1];
    F_j = new double[nx-1];

    X_i = new double[ny-1];
    X_j = new double[nx-1];

    set_initials();

    clock_t t1 = clock();

    for(int k = 0; k < nt; k++){
        double t_half = t[k] + dt_1_2;
        double t_k = t_half + dt_1_2;

        // BEGIN------------------------------- x-direction ------------------------------------------------
        set_boundaries(u_half, t_half);

        for(int i = 1; i < ny; i++){
            for(int j = 1; j < nx; j++){
                F_j[j-1] = (u[i+1][j] - 2 * u[i][j] + u[i-1][j]) / dy2 + F(x[j], y[i], t_half) + 2 * u[i][j] / dt;
```

```
                if(j == 1){
                    F_j[j-1] += u_exact(x[j-1], y[i], t_half) / dx2;
                }else if(j == nx-1){
                    F_j[j-1] += u_exact(x[j+1], y[i], t_half) / dx2;
                }
            }

            solveMatrix(nx-1, A_j, B_j, C_j, F_j, X_j);

            for(int j = 0; j < nx-1; j++){
                u_half[i][j+1] = X_j[j];
            }
        }
        // END---------------------------------- x-direction ------------------------------------------------

        // BEGIN-------------------------------- y-direction ------------------------------------------------
        set_boundaries(u, t_k);

        for(int j = 1; j < nx; j++){
            for(int i = 1; i < ny; i++){
                F_i[i-1] = (u_half[i][j-1] - 2 * u_half[i][j] + u_half[i][j+1]) / dx2 + F(x[j], y[i], t_half) + 2 * u_half[i][j]
            }

            F_i[0] += u_exact(x[j], y[0], t_k) / dy2;
            F_i[ny-2] += u_exact(x[j], y[ny], t_k) / dy2;

            solveMatrix(ny-1, A_i, B_i, C_i, F_i, X_i);

            for(int i = 0; i < ny-1; i++){
                u[i+1][j] = X_i[i];
            }
        }
        // END---------------------------------- y-direction ------------------------------------------------
    }

    clock_t t2 = clock();

    cout << "Time: " << (double)(t2 - t1) / (double)CLOCKS_PER_SEC << endl << endl;
}
// END---------------------------------------- SOLVE ------------------------------------------------------



int main(int argc, char** argv){
    cout << fixed << setprecision(8);

    if(argc < 2){
        cout << "Usage: ./a.out data" << endl;

        exit(1);
    }

    filename = argv[1];

    set_grid();

    solve();

    print_error();

    free_memory();

    return 0;
}



// BEGIN---------------------------------------- PRINT GRID ------------------------------------------------
void print_x(){
    cout << "x: ";
```

```
    for(int j = 0; j <= nx; j++){
        cout << x[j] << " ";
    }

    cout << endl << endl;
}

void print_y(){
    cout << "y: ";

    for(int i = 0; i <= ny; i++){
        cout << y[i] << " ";
    }

    cout << endl << endl;
}

void print_t(){
    cout << "t: ";

    for(int k = 0; k <= nt; k++){
        cout << t[k] << " ";
    }

    cout << endl << endl;
}

void print_grid(){
    print_x();
    print_y();
    print_t();
}
// END---------------------------------------- PRINT GRID ------------------------------------------------

// BEGIN---------------------------------------- PRINT SOLUTION ------------------------------------------
void print_u(){
    cout << "u:" << endl;

    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            cout << setw(12) << setprecision(8) << u[i][j] << " ";
        }

        cout << endl;
    }

    cout << endl;
}

void print_u_half(){
    cout << "u_half:" << endl;

    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            cout << setw(12) << setprecision(9) << u_half[i][j] << " ";
        }

        cout << endl;
    }

    cout << endl;
}

void print_u_exact(int k){
    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            cout << u_exact(x[j], y[i], t[k]) << " ";
        }

        cout << endl;
    }
```

```
    cout << endl;
}
// END---------------------------------------- PRINT SOLUTION ----------------------------------------

// BEGIN---------------------------------------- PRINT ERROR ----------------------------------------
void print_error(){
    double max_abs_error = 0;
    double max_rel_error = 0;

    for(int i = 1; i < ny; i++){
        for(int j = 1; j < nx; j++){
            double sln = u[i][j];

            double sln_e = u_exact(x[j], y[i], t[nt]);

            if(abs(sln_e) < 1e-10 || abs(sln) < 1e-10){
                continue;
            }

            double abs_error = abs(sln - sln_e);
            double rel_error = abs_error / abs(sln_e);

            max_abs_error = max(max_abs_error, abs_error);
            max_rel_error = max(max_rel_error, rel_error);
        }
    }

    cout << "Abs. error: " << setprecision(8) << max_abs_error << endl;
    cout << "Rel. error: " << setprecision(8) << max_rel_error << endl;
}
// END---------------------------------------- PRINT ERROR ----------------------------------------

// BEGIN---------------------------------------- FREE MEMORY ----------------------------------------
void free_memory(){
    // -------------------------
    delete[] x;
    delete[] y;
    delete[] t;
    // -------------------------
    delete[] A_i;
    delete[] B_i;
    delete[] C_i;
    // -------------------------
    delete[] A_j;
    delete[] B_j;
    delete[] C_j;
    // -------------------------
    for(int i = 0; i <= ny; i++){
        delete[] u[i];
        delete[] u_half[i];
    }

    delete[] u;
    delete[] u_half;
    // -------------------------
    delete[] F_i;
    delete[] F_j;

    delete[] X_i;
    delete[] X_j;
    // -------------------------
}
// END---------------------------------------- FREE MEMORY ----------------------------------------
```

2) Параллельная версия

implicit_mpi.cpp

```
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <mpi.h>

#include "thomas_x.cpp"
#include "thomas_y.cpp"
#include "thomas_y_helper.h"

using namespace std;

// ------------------------
char* filename;
// ------------------------
double a1, b1;
double a2, b2;

double t_max;

int nx, ny, nt;

double dx, dy, dt;
double dx2, dy2;
double dt_1_2;

double* x;
double* y;
double* t;

double** u;
double** u_half;
// ------------------------
double *A_i, *B_i, *C_i;
double *A_j, *B_j, *C_j;
// ------------------------
int mpirank;
int mpisize;

bool rank_0;
bool rank_np;

int src_up, src_dn;
int dst_up, dst_dn;

int snd_up_tag, snd_dn_tag;
int rcv_up_tag, rcv_dn_tag;

MPI_Datatype row, row_1, column;
// ------------------------
int ny_per_node;
int ny_tail;

int my_ny;

int row_0_ny;
int row_n_ny;

double** u_up;
double** u_down;

double** u_left;
double** u_right;
// ------------------------
void print_grid();
```

```
void print_x();
void print_y();
void print_t();

void print_u();
void print_u_half();
void print_u_exact(int);
void print_error();

void free_memory();
// ------------------------

// BEGIN---------------------------------------- EXACT SOLUTION ---------------------------------------------
double u_exact(double x, double y, double t){
    // return exp(-t) * sin(M_PI * x) * cos(M_PI * y);
    return exp(-t) * (sin(M_PI * x) + cos(M_PI * y)) + 10;
}
// END------------------------------------------ EXACT SOLUTION ---------------------------------------------

// BEGIN---------------------------------------- F(x, y, t) ------------------------------------------------
double F(double x, double y, double t){
    // return exp(-t) * sin(M_PI * x) * cos(M_PI * y) * (2 * M_PI * M_PI - 1);
    return - exp(-t) * (sin(M_PI * x) + cos(M_PI * y)) * (1 - M_PI * M_PI);
}
// END------------------------------------------ F(x, y, t) ------------------------------------------------

// BEGIN---------------------------------------- READ DATA -------------------------------------------------
void read_data(){
    FILE* f;

    f = fopen(filename, "r");

    if(f == NULL){
        cout << "Cannot open file \"" << filename << "\"" << endl;

        exit(1);
    }

    int count = fscanf(f, "%lf %lf %lf %lf %lf %d %d %d", &a1, &b1, &a2, &b2, &t_max, &nx, &ny, &nt);

    if(count < 8){
        cout << "Wrong data format in file \"" << filename << "\"" << endl << endl;
        cout << "Usage: a1 b1 a2 b2" << endl;
        cout << "       t_max" << endl;
        cout << "       nx ny nt" << endl;
        cout << endl;

        cout << "Example: 0 1 0 1" << endl;
        cout << "         1" << endl;
        cout << "         10 10 5" << endl;
        cout << endl;

        exit(1);
    }

    fclose(f);
}
// END------------------------------------------ READ DATA -------------------------------------------------

// BEGIN---------------------------------------- SET TRIAG COEFF -------------------------------------------
void set_triag_coeff(){
    A_i = new double[ny];
    B_i = new double[ny];
    C_i = new double[ny];

    for(int i = 0; i < ny; i++){
        A_i[i] = -1.0 / dy2;
        B_i[i] = 2.0 / dt + 2.0 / dy2;
        C_i[i] = -1.0 / dy2;
    }
```

```
    C_i[ny-1] = 0;
    // ------------------------------
    A_j = new double[nx];
    B_j = new double[nx];
    C_j = new double[nx];

    for(int i = 0; i < nx; i++){
        A_j[i] = -1.0 / dx2;
        B_j[i] = 2.0 / dt + 2.0 / dx2;
        C_j[i] = -1.0 / dx2;
    }

    C_j[nx-1] = 0;
}
// END---------------------------------------- SET TRIAG COEFF ----------------------------------------

// BEGIN---------------------------------------- SET GRID ----------------------------------------------------
void set_grid(){
    read_data();

    assert(a1 < b1);
    assert(a2 < b2);

    assert(t_max > 0);

    assert(nx > 0);
    assert(ny > 0);
    assert(nt > 0);

    dx = (b1 - a1) / nx;
    dy = (b2 - a2) / ny;
    dt = t_max / nt;

    dx2 = dx * dx;
    dy2 = dy * dy;

    dt_1_2 = dt / 2.0;
    // ------------------------------------------------
    ny_per_node = ((ny + 1) + 2 * (mpisize-1)) / mpisize;

    ny_tail = ((ny + 1) + 2 * (mpisize-1)) % mpisize;

    my_ny = ny_per_node - 1;

    if(mpirank < ny_tail)
        my_ny++;
    // ------------------------------------------------
    try{
        x = new double[nx+1];
        y = new double[my_ny+1];
        t = new double[nt+1];

        u = new double*[my_ny+1];
        u_half = new double*[my_ny+1];

        for(int i = 0; i <= my_ny; i++){
            u[i] = new double[nx+1];
            u_half[i] = new double[nx+1];

            for(int j = 0; j <= nx; j++){
                u[i][j] = 0;
                u_half[i][j] = 0;
            }
        }
    }catch(std::bad_alloc){
        cout << "Cannot allocate memory" << endl;

        exit(1);
    }
    // ------------------------------------------------
    double y0 = 0;
```

```
    if(mpisize == 1){
        y0 = a2;
    }else if(!rank_0){
        int _y0 = 0;

        if(ny_tail){
            _y0 = (mpirank <= ny_tail) ? mpirank : ny_tail;
        }

        y0 = a2 + ((ny_per_node-2) * (mpirank) + _y0) * dy;
    }

    for(int i = 0; i <= my_ny; i++){
        y[i] = y0 + i * dy;
    }
    // -------------------------------------------------
    for(int j = 0; j <= nx; j++){
        x[j] = a1 + j * dx;
    }

    for(int k = 0; k <= nt; k++){
        t[k] = k * dt;
    }
}
// END---------------------------------------- SET GRID ------------------------------------------------

// BEGIN-------------------------------------- INITIAL CONDITIONS ----------------------------------------
double g_0(int j, int i){
    return u_exact(x[j], y[i], t[0]);
}

void set_initials(){
    int i_1 = (rank_0) ? 0 : 1;
    int i_2 = (rank_np) ? (my_ny) : (my_ny-1);

    for(int i = i_1; i <= i_2; i++){
        for(int j = 0; j <= nx; j++){
            u[i][j] = g_0(j, i);
        }
    }
}
// END---------------------------------------- INITIAL CONDITIONS ----------------------------------------

// BEGIN-------------------------------------- BOUNDARY CONDITIONS ----------------------------------------
void set_boundaries(double** _u, int k){
    if(!rank_0){
        #pragma omp parallel for
        for(int j = 0; j <= nx; j++){
            _u[my_ny][j] = u_down[k][j];
        }
    }

    if(!rank_np){
        #pragma omp parallel for
        for(int j = 0; j <= nx; j++){
            _u[0][j] = u_up[k][j];
        }
    }

    #pragma omp parallel for
    for(int i = 0; i <= my_ny; i++){
        _u[i][0] = u_left[k][i];
        _u[i][nx] = u_right[k][i];
    }
}
// END---------------------------------------- BOUNDARY CONDITIONS ----------------------------------------

// BEGIN-------------------------------------- MPI_PREPARE -----------------------------------------------
void mpi_prepare(){
    MPI_Type_contiguous(nx, MPI_DOUBLE, &row);
```

```
    MPI_Type_commit(&row);

    MPI_Type_contiguous(nx-1, MPI_DOUBLE, &column);
    MPI_Type_commit(&column);

    MPI_Type_vector(nx-1, 1, mpisize, MPI_DOUBLE, &row_1);
    MPI_Type_commit(&row_1);

    src_dn = dst_dn = mpirank+1;
    src_up = dst_up = mpirank-1;

    snd_up_tag = 1000 * dst_up  + 2;
    snd_dn_tag = 1000 * dst_dn  + 0;

    rcv_up_tag = 1000 * mpirank  + 0;
    rcv_dn_tag = 1000 * mpirank  + 2;
}
// END---------------------------------------- MPI_PREPARE ------------------------------------------------

// BEGIN-------------------------------------- PREPARE_BOUNDARIES ----------------------------------------
void prepare_boundaries(){
    u_up = new double*[(nt+1)*2];
    u_down = new double*[(nt+1)*2];
    u_left = new double*[(nt+1)*2];
    u_right = new double*[(nt+1)*2];

    for(int k = 0; k <= nt*2; k++){
        u_up[k] = new double[nx+1];
        u_down[k] = new double[nx+1];

        u_left[k] = new double[my_ny+1];
        u_right[k] = new double[my_ny+1];
    }

    for(int k = 0; k <= nt; k++){
        for(int j = 0; j < nx; j++){
            u_up[k*2][j] = u_exact(x[j], y[0], t[k]);
            u_down[k*2][j] = u_exact(x[j], y[my_ny], t[k]);

            if(k < nt){
                u_up[k*2+1][j] = u_exact(x[j], y[0], t[k]+dt_1_2);
                u_down[k*2+1][j] = u_exact(x[j], y[my_ny], t[k]+dt_1_2);
            }
        }

        for(int i = 0; i <= my_ny; i++){
            u_left[k*2][i] = u_exact(x[0], y[i], t[k]);
            u_right[k*2][i] = u_exact(x[nx], y[i], t[k]);

            if(k < nt){
                u_left[k*2+1][i] = u_exact(x[0], y[i], t[k]+dt_1_2);
                u_right[k*2+1][i] = u_exact(x[nx], y[i], t[k]+dt_1_2);
            }
        }
    }
}
// END---------------------------------------- PREPARE_BOUNDARIES ----------------------------------------

// BEGIN-------------------------------------- SOLVE ----------------------------------------------------
void solve(){
    mpi_prepare();
    // --------------------------
    set_triag_coeff();
    // --------------------------
    double* F_j = new double[nx-1];

    double* X_j = new double[nx-1];
    // --------------------------
    prepare_boundaries();
    // ----------------------------------------------------
    set_initials();
```

```
// -------------------------------------------------------
double** A_nx_i = new double*[nx];
double** B_nx_i = new double*[nx];
double** C_nx_i = new double*[nx];

double** F_nx_i = new double*[nx];

double** X_nx_i = new double*[nx];

for(int i = 0; i < nx; i++){
    A_nx_i[i] = new double[my_ny];
    B_nx_i[i] = new double[my_ny];
    C_nx_i[i] = new double[my_ny];

    F_nx_i[i] = new double[my_ny];

    X_nx_i[i] = new double[my_ny+1];
}
// -------------------------------------------------------
Helper* helper = new Helper(mpirank, mpisize, nx, my_ny, int(y[0] / dy));
// -------------------------------------------------------
MPI_Barrier(MPI_COMM_WORLD);
double t1 = MPI_Wtime();

for(int k = 0; k < nt; k++){
    double t_half = t[k] + dt_1_2;
    double t_k = t_half + dt_1_2;

    // BEGIN------------------------------- x-direction ------------------------------------------------
    MPI_Status st1, st2;

    MPI_Request req_snd_up, req_snd_dn;
    MPI_Request req_rcv_up, req_rcv_dn;

    if(mpisize != 1){
        if(!rank_0){
            MPI_Irecv(&u[0][1], 1, row, src_up, rcv_up_tag, MPI_COMM_WORLD, &req_rcv_up);
        }

        if(!rank_np){
            MPI_Irecv(&u[my_ny][1], 1, row, src_dn, rcv_dn_tag, MPI_COMM_WORLD, &req_rcv_dn);
        }
    }

    set_boundaries(u_half, 2*k+1);

    if(mpisize != 1){
        if(!rank_0){
            MPI_Isend(&u[1][1], 1, row,   dst_up, snd_up_tag, MPI_COMM_WORLD, &req_snd_dn);
        }

        if(!rank_np){
            MPI_Isend(&u[my_ny-1][1], 1, row,   dst_dn, snd_dn_tag, MPI_COMM_WORLD, &req_snd_up);
        }
    }

    for(int i = 2; i < my_ny-1; i++){
        u_half = thomas_x(u, A_j, B_j, C_j, F_j, X_j, k, i, t_half);
    }

    if(!rank_0){
        MPI_Wait(&req_rcv_up, &st1);
    }

    u_half = thomas_x(u, A_j, B_j, C_j, F_j, X_j, k, 1, t_half);

    if(!rank_np){
        MPI_Wait(&req_rcv_dn, &st2);
    }

    u_half = thomas_x(u, A_j, B_j, C_j, F_j, X_j, k, my_ny-1, t_half);
```

```
        // END---------------------------------- x-direction ------------------------------------------------

        // BEGIN------------------------------- y-direction ------------------------------------------------
        set_boundaries(u, k*2+2);

        #pragma omp parallel for
        for(int _i = 0; _i < my_ny; _i++){
            A_nx_i[1][_i] = A_i[_i];
            B_nx_i[1][_i] = B_i[_i];
            C_nx_i[1][_i] = C_i[_i];
        }

        #pragma omp parallel for
        for(int j = 1; j < nx; j++){
            for(int i = 1; i < my_ny; i++){
                F_nx_i[j][i-1] = (u_half[i][j-1] - 2 * u_half[i][j] + u_half[i][j+1]) / dx2 + F(x[j], y[i], t_half) + 2 * u_half
            }

            if(rank_0){
                F_nx_i[j][0] += u_up[2*k+2][j] / dy2;
            }else if(rank_np){
                F_nx_i[j][my_ny-2] += u_down[2*k+2][j] / dy2;
            }
        }

        X_nx_i = thomas_y(A_nx_i, B_nx_i, C_nx_i, F_nx_i, X_nx_i, helper);

        #pragma omp parallel for
        for(int j = 1; j < nx; j++){
            for(int i = 0; i < my_ny-1; i++){
                u[i+1][j] = X_nx_i[j][i];
            }
        }
        // END---------------------------------- y-direction ------------------------------------------------
    }

    MPI_Barrier(MPI_COMM_WORLD);
    double t2 = MPI_Wtime();

    if(rank_0){
        cout << "nprocs: " << mpisize << endl;
        cout << "Time: " << (t2 - t1) << endl << endl;
    }

    MPI_Barrier(MPI_COMM_WORLD);
}
// END---------------------------------------- SOLVE ------------------------------------------------------



int main(int argc, char** argv){
    cout << fixed << setprecision(8);

    if(argc < 2){
        cout << "Usage: ./a.out data" << endl;

        exit(1);
    }

    filename = argv[1];

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);

    rank_0 = (mpirank == 0);
    rank_np = (mpirank == mpisize-1);

    set_grid();
```

```
    solve();

    print_error();

    free_memory();

    MPI_Finalize();

    return 0;
}



// BEGIN---------------------------------------- PRINT GRID ----------------------------------------------
void print_x(){
    MPI_Barrier(MPI_COMM_WORLD);

    if(rank_0){
        cout << "x: ";

        for(int j = 0; j <= nx; j++){
            cout << x[j] << " ";
        }

        cout << endl << endl;
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

void print_y(){
    MPI_Barrier(MPI_COMM_WORLD);

    for(int r = 0; r < mpisize; r++){
        if(r == mpirank){
            cout << "rank: " << r << endl;
            cout << "my_ny: " << my_ny << endl;
            cout << "y: ";

            for(int i = 0; i <= my_ny; i++){
                cout << y[i] << " ";
            }

            cout << endl << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

void print_t(){
    MPI_Barrier(MPI_COMM_WORLD);

    if(rank_0){
        cout << "t: ";

        for(int k = 0; k <= nt; k++){
            cout << t[k] << " ";
        }

        cout << endl << endl;
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

void print_grid(){
    print_x();
    print_y();
```

```
    print_t();
}
// END---------------------------------------- PRINT GRID ------------------------------------------------

// BEGIN---------------------------------------- PRINT SOLUTION ------------------------------------------
void print_u(){
    MPI_Barrier(MPI_COMM_WORLD);

    if(mpirank == 0){
        cout << "u:" << endl;
    }

    MPI_Barrier(MPI_COMM_WORLD);

    for(int r = 0; r < mpisize; r++){
        if(r == mpirank){
            for(int i = 0; i <= my_ny; i++){
                for(int j = 0; j <= nx; j++){
                    cout << setw(12) << setprecision(8) << u[i][j] << " ";
                }

                cout << endl;
            }

            cout << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

void print_u_half(){
    MPI_Barrier(MPI_COMM_WORLD);

    cout << setprecision(8);

    if(mpirank == 0){
        cout << "u_half:" << endl;
    }

    MPI_Barrier(MPI_COMM_WORLD);

    for(int r = 0; r < mpisize; r++){
        if(r == mpirank){
            for(int i = 0; i <= my_ny; i++){
                for(int j = 0; j <= nx; j++){
                    cout << setw(12) << setprecision(9) << u_half[i][j] << " ";
                }

                cout << endl;
            }

            cout << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

void print_u_exact(int k){
    MPI_Barrier(MPI_COMM_WORLD);

    cout << setprecision(8);

    if(mpirank == 0){
        cout << "u_half:" << endl;
    }
```

```
    MPI_Barrier(MPI_COMM_WORLD);

    for(int r = 0; r < mpisize; r++){
        if(r == mpirank){
            for(int i = 0; i <= my_ny; i++){
                for(int j = 0; j <= nx; j++){
                    cout << setw(12) << setprecision(9) << u_exact(x[j], y[i], t[k]) << " ";
                }

                cout << endl;
            }

            cout << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}
// END---------------------------------------- PRINT SOLUTION ----------------------------------------

// BEGIN---------------------------------------- PRINT ERROR ----------------------------------------
void print_error(){
    double max_abs_error = 0;
    double max_rel_error = 0;

    int i1, i2;
    int j1 = 0, j2 = nx;

    if(mpirank == 0){
        i1 = 0;
        i2 = my_ny-1;
    }else if(mpirank == mpisize-1){
        i1 = 1;
        i2 = my_ny;
    }else{
        i1 = 1;
        i2 = my_ny-1;
    }

    for(int i = i1; i <= i2; i++){
        for(int j = j1; j <= j2; j++){
            double sln = u[i][j];

            double sln_e = u_exact(x[j], y[i], t[nt]);

            if(abs(sln_e) < 1e-10 || abs(sln) < 1e-9){
                continue;
            }

            double abs_error = abs(sln - sln_e);
            double rel_error = abs_error / abs(sln_e);

            max_abs_error = max(max_abs_error, abs_error);
            max_rel_error = max(max_rel_error, rel_error);
        }
    }

    double* abs_err = new double[mpisize];
    double* rel_err = new double[mpisize];

    MPI_Gather(&max_abs_error, 1, MPI_DOUBLE, abs_err, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Gather(&max_rel_error, 1, MPI_DOUBLE, rel_err, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if(mpirank == 0){
        double max_abs_error = 0;
        double max_rel_error = 0;

        for(int i = 0; i < mpisize; i++){
```

```
            max_abs_error = max(abs_err[i], max_abs_error);
            max_rel_error = max(rel_err[i], max_rel_error);
        }

        cout << "Abs. error: " << setprecision(8) << max_abs_error << endl;
        cout << "Rel. error: " << setprecision(8) << max_rel_error << endl;
    }
}
// END---------------------------------------- PRINT ERROR ----------------------------------------------

// BEGIN---------------------------------------- FREE MEMORY ----------------------------------------------
void free_memory(){
    delete[] x;
    delete[] y;
    delete[] t;

    for(int i = 0; i <= my_ny; i++){
        delete[] u[i];
        delete[] u_half[i];
    }

    delete[] u;
    delete[] u_half;
}
// END---------------------------------------- FREE MEMORY ----------------------------------------------
```

thomas_x.cpp

```
#ifndef _THOMAS_X_
#define _THOMAS_X_

extern int nx;

extern double** u_left;
extern double** u_right;

extern double dx2;
extern double dy2;

extern double* x;
extern double* y;

extern double dt;

extern double F(double x, double y, double t);

// BEGIN---------------------------------------- THOMAS ALGORITHM ----------------------------------------
void solveMMatrix(const int n, const double *_A, const double *_B, const double *_C, double *f, double *x){
    // ---------------------------
    double A[n];
    double B[n];
    double C[n];

    for(int i = 0; i < n; i++){
        A[i] = _A[i];
        B[i] = _B[i];
        C[i] = _C[i];
    }
    // ---------------------------
    double coeff;

    for(int i = 1; i < n; i++){
        coeff = A[i] / B[i-1];

        B[i] -= coeff * C[i-1];
        f[i] -= coeff * f[i-1];
    }

    x[n-1] = f[n-1] / B[n-1];

    for(int i = n-2; i >= 0; i--){
        x[i] = (f[i] - C[i] * x[i+1]) / B[i];
    }
}
// END---------------------------------------- THOMAS ALGORITHM ----------------------------------------

extern double** u_half;

double** thomas_x(double** u, double* A_j, double* B_j, double* C_j, double* F_j, double* X_j, int k, int i, double t_half){
    #pragma omp parallel for
    for(int j = 1; j < nx; j++){
        F_j[j-1] = (u[i+1][j] - 2 * u[i][j] + u[i-1][j]) / dy2 + F(x[j], y[i], t_half) + 2 * u[i][j] / dt;
    }

    F_j[0] += u_left[2*k+1][i] / dx2;
    F_j[nx-2] += u_right[2*k+1][i] / dx2;

    solveMMatrix(nx-1, A_j, B_j, C_j, F_j, X_j);

    #pragma omp parallel for
    for(int j = 0; j < nx-1; j++){
        u_half[i][j+1] = X_j[j];
    }

    return u_half;
}

#endif
```

thomas_y.cpp

```
#ifndef THOMAS_CPP_
#define THOMAS_CPP_

#include <iostream>
#include <cstring>
#include <mpi.h>

#include "thomas_y_helper.h"

using namespace std;

extern int mpirank;
extern int mpisize;

extern bool rank_0;
extern bool rank_np;

extern int my_ny;
extern int nx;

extern int src_up, src_dn;
extern int dst_up, dst_dn;

extern int snd_up_tag, snd_dn_tag;
extern int rcv_up_tag, rcv_dn_tag;

extern MPI_Datatype row_1, column;

void solveMatrix(int n, double *a, double *b, double *c, double *f, double *x){
    double m;

    int i_prev;

    for(int i = 1; i < n; i++){
        i_prev = i-1;

        m = a[i] / b[i_prev];

        b[i] -= m * c[i_prev];
        f[i] -= m * f[i_prev];
    }

    x[n-1] = f[n-1] / b[n-1];

    for(int i = n-2; i >= 0; i--){
        x[i] = (f[i] - c[i] * x[i+1]) / b[i];
    }
}

void down(double** A, double** B, double** C, double** F_i, double** d_l, int i_down){
    double coeff;

    int i_next;

    for(int i = 0; i <= i_down; i++){
        i_next = i+1;
        // ---------------------------------------
        coeff = B[1][i];

        B[1][i] /= coeff;
        C[1][i] /= coeff;

        if(!rank_0)
            d_l[1][i] /= coeff;

        int J;

        #pragma omp parallel for private(J)
        for(J = 1; J < nx; J++){
            F_i[J][i] /= coeff;
```

```
        }
        // ---------------------------------------
        coeff = A[1][i];

        B[1][i_next] -= C[1][i] * coeff;
        A[1][i] -= B[1][i] * coeff;

        if(!rank_0)
            d_l[1][i_next] -= d_l[1][i] * coeff;

        #pragma omp parallel for private(J)
        for(J = 1; J < nx; J++){
            F_i[J][i_next] -= F_i[J][i] * coeff;
        }
        // ---------------------------------------
    }

    coeff = B[1][i_down];

    B[1][i_down] /= coeff;
    A[1][i_down] /= coeff;
    C[1][i_down] /= coeff;

    if(!rank_0)
        d_l[1][i_down] /= coeff;

    #pragma omp parallel for
    for(int J = 1; J < nx; J++){
        F_i[J][i_down] /= coeff;
    }
}

void up(double** A, double** B, double** C, double** F_i, double** d_l, double** d_r, int i_down){
    double coeff;

    d_r[1][i_down-1] = C[1][i_down-1];

    C[1][i_down-1] = 0;

    int i_next;

    for(int i = i_down-2; i >= 0; i--){
        i_next = i+1;

        coeff = C[1][i];

        C[1][i] -= B[1][i_next] * coeff;

        if(!rank_0)
            d_l[1][i] -= d_l[1][i_next] * coeff;

        d_r[1][i] -= d_r[1][i_next] * coeff;

        #pragma omp parallel for
        for(int J = 1; J < nx; J++){
            F_i[J][i] -= F_i[J][i_next] * coeff;
        }
    }
}

double** thomas_y(double** A_i, double** B_i, double** C_i, double** F_i, double** X_i, Helper* helper){
    // ---------------------------------------
    int M = helper->M;

    int H = helper->H;
    int W = helper->W;

    int i_0 = helper->i_0;
    int i_down = helper->i_down;

    double** d_l = helper->d_l;
```

```
double** d_r = helper->d_r;

helper->flush();
// ----------------------------------------
int* block_size = helper->block_size;

int* block_start = helper->block_start;
int* block_fin = helper->block_fin;

int* block_shift = helper->block_shift;

int block_size_max = helper->block_size_max;
// ----------------------------------------
double A[nx][H];
double C[nx][H];
double F[nx][H];

double triple_dn[nx][3];
double triple_up[nx][3];

double sln[mpisize][block_size_max][H];
// ----------------------------------------------
MPI_Request* req_sln_rcv = helper->req_sln_rcv;
MPI_Request* req_sln_snd = helper->req_sln_snd;

MPI_Status* stat_sln = helper->stat_sln;
// ----------------------------------------------
if(!rank_np){
    MPI_Irecv(helper->triple_dn, nx+2, MPI_DOUBLE, src_dn, rcv_dn_tag * 1000, MPI_COMM_WORLD, &(helper->req_triple_down_rcv)
}

for(int r = 0; r < mpisize; r++){
    MPI_Irecv(sln[r], block_size[r] * H, MPI_DOUBLE, r, (r+1), MPI_COMM_WORLD, &req_sln_rcv[r]);

    MPI_Irecv(&A[1][r], 1, row_1, r, r * 3, MPI_COMM_WORLD, &helper->req_diags_rcv[r]);
    MPI_Irecv(&C[1][r], 1, row_1, r, r * 4, MPI_COMM_WORLD, &helper->req_diags_rcv[r + mpisize]);
    MPI_Irecv(&F[1][r], 1, row_1, r, r * 5, MPI_COMM_WORLD, &helper->req_diags_rcv[r + mpisize * 2]);
}
//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
double* a_down_all = helper->a_down_all;
double* c_down_all = helper->c_down_all;
double* f_down_all = helper->f_down_all;
//-------------------------------------------------------------------------------------------------------------------
d_l[1][0] = A_i[1][0];
d_r[1][i_down] = B_i[1][i_down];
//-------------------------------------------------------------------------------------------------------------------
down(A_i, B_i, C_i, F_i, d_l, i_down);
up(A_i, B_i, C_i, F_i, d_l, d_r, i_down);

#pragma omp parallel for
for(int J = 1; J < nx; J++){
    helper->triag_snd_buf[J] = F_i[J][0];
}

helper->triag_snd_buf[nx] = d_l[1][0];
helper->triag_snd_buf[nx+1] = d_r[1][0];

if(!rank_0){
    MPI_Isend(helper->triag_snd_buf, nx+2, MPI_DOUBLE, dst_up, snd_up_tag * 1000, MPI_COMM_WORLD, &(helper->req_triags_snd))
}

int J;

#pragma omp parallel for private(J)
for(J = 1; J < nx; J++){
    for(int i = 0; i < M; i++){
        C_i[J][i] = C_i[1][i];

        d_l[J][i] = d_l[1][i];
```

```
        d_r[J][i] = d_r[1][i];
    }
}
//-----------------------------------------------------------------------------------------------------------------
if(!rank_np){
    MPI_Wait(&(helper->req_triple_down_rcv), &(helper->stat_triple_down));
}

double A_i_1_down = A_i[1][i_down];
double B_i_1_down = B_i[1][i_down];

#pragma omp parallel for private(J)
for(J = 1; J < nx; J++){
    double coef;

    double* h = helper->triple_dn;
    // ---------------------------------------
    triple_dn[J][0] = h[nx];
    triple_dn[J][1] = h[nx+1];
    triple_dn[J][2] = h[J];
    // ---------------------------------------
    A_i[J][i_down] = A_i_1_down;
    B_i[J][i_down] = B_i_1_down;

    coef = C_i[J][i_down];

    B_i[J][i_down] -= triple_dn[J][0] * coef;
    F_i[J][i_down] -= triple_dn[J][2] * coef;

    triple_dn[J][1] *= -coef;
    // ---------------------------------------
    coef = B_i[J][i_down];

    A_i[J][i_down] /= coef;
    B_i[J][i_down] /= coef;
    C_i[J][i_down] /= coef;
    F_i[J][i_down] /= coef;

    f_down_all[J-1] = F_i[J][i_down];

    if(!rank_0){
        d_l[J][i_down] /= coef;

        a_down_all[J-1] = d_l[J][i_down];
    }

    triple_dn[J][1] /= coef;

    c_down_all[J-1] = triple_dn[J][1];

    d_r[J][i_down] /= coef;
    // ---------------------------------------
}

for(int i = 0; i < mpisize; i++){
    MPI_Isend(a_down_all, 1, column, i, mpirank * 3, MPI_COMM_WORLD, &helper->req_diags_snd[i]);
    MPI_Isend(c_down_all, 1, column, i, mpirank * 4, MPI_COMM_WORLD, &helper->req_diags_snd[i + mpisize]);
    MPI_Isend(f_down_all, 1, column, i, mpirank * 5, MPI_COMM_WORLD, &helper->req_diags_snd[i + mpisize * 2]);
}

MPI_Waitall(mpisize * 3, helper->req_diags_rcv, helper->stat_recv);
//-----------------------------------------------------------------------------------------------------------------
#pragma omp parallel for private(J)
for(J = block_start[mpirank]; J < block_fin[mpirank]; J++){
    double B[H];
    double X[H];

    for(int i = 0; i < H; i++){
        B[i] = 1;
    }
```

```
        solveMatrix(H, A[J], B, C[J], F[J], X);

        for(int i = 0; i < H; i++){
            sln[mpirank][J - block_shift[mpirank]][i] = X[i];
        }
    }

    for(int r = 0; r < mpisize; r++){
        MPI_Isend(sln[mpirank], block_size[mpirank] * H, MPI_DOUBLE, r, (mpirank+1), MPI_COMM_WORLD, &req_sln_snd[r]);
    }

    int numdone;

    int _mpisize = mpisize;

    while(_mpisize--){
        MPI_Waitany(mpisize, req_sln_rcv, &numdone, stat_sln);

        int r = numdone;

        #pragma omp parallel for
        for(int J = block_start[r]; J < block_fin[r]; J++){
            if(rank_0){
                X_i[J][i_down] = sln[r][J-block_shift[r]][0];

                for(int i = M-2; i >= 0; i--){
                    X_i[J][i] = F_i[J][i] - X_i[J][i_down] * d_r[J][i];
                }
            }else{
                double _l = sln[r][J-block_shift[r]][mpirank-1];
                double _r = sln[r][J-block_shift[r]][mpirank];

                X_i[J][i_down] = _r;

                for(int i = M-2; i >= 0; i--){
                    X_i[J][i] = F_i[J][i] - (d_l[J][i] * _l + d_r[J][i] * _r);
                }
            }
        }
    }
    // ------------------------------------------------------------------------------------------------------------------
    return X_i;
}

#endif
```

thomas_y_helper.h

```
#ifndef MPI_Helper_H
#define MPI_Helper_H

#include <mpi.h>

class Helper{

public:
    Helper(int _mpirank, int _mpisize, int _nx, int _my_ny, int _i_0){
        // -------------------------------------
        mpirank = _mpirank;
        mpisize = _mpisize;
        // -------------------------------------
        nx = _nx;
        my_ny = _my_ny;
        // -------------------------------------
        M = my_ny-1;

        H = mpisize;
        W = H + 1;
        // -------------------------------------
        i_0 = _i_0;
        i_down = M-1;
        // -------------------------------------
        req_diags_rcv = new MPI_Request[mpisize*3];
        req_diags_snd = new MPI_Request[mpisize*3];

        stat_recv = new MPI_Status[mpisize*3];
        // -------------------------------------
        req_sln_rcv = new MPI_Request[mpisize];
        req_sln_snd = new MPI_Request[mpisize];

        stat_sln = new MPI_Status[mpisize];
        // -------------------------------------
        B = new double[H];

        X = new double[H];
        // -------------------------------------
        triag_snd_buf = new double[nx+2];
        triple_dn = new double[nx+2];
        // -------------------------------------
        d_l = new double*[nx];
        d_r = new double*[nx];

        for(int i = 0; i < nx; i++){
            d_l[i] = new double[my_ny-1];
            d_r[i] = new double[my_ny-1];
        }
        // -------------------------------------
        block_size = new int[mpisize];

        block_start = new int[mpisize];
        block_fin = new int[mpisize];

        block_shift = new int[mpisize];

        for(int r = 0; r < mpisize; r++){
            block_size[r] = nx / mpisize;
        }

        block_size[mpisize-1] += nx % mpisize;

        block_start[0] = 1;

        for(int r = 1; r < mpisize; r++){
            block_start[r] = 0;

            for(int i = 0; i < r; i++){
                block_start[r] += block_size[i];
            }
```

```
    }

    block_fin[0] = block_size[0];

    for(int r = 1; r < mpisize; r++){
        block_fin[r] = block_start[r] + block_size[r];
    }

    block_shift[0] = 0;

    for(int r = 1; r < mpisize; r++){
        block_shift[r] = block_start[r];
    }

    block_size_max = 0;

    for(int r = 0; r < mpisize; r++){
        block_size_max = std::max(block_size_max, block_size[r]);
    }
    // ---------------------------
    a_down_all = new double[nx-1];
    c_down_all = new double[nx-1];
    f_down_all = new double[nx-1];
    // ---------------------------
}

~Helper(){
    // --------------------
    delete[] req_diags_rcv;
    delete[] req_diags_snd;

    delete[] stat_recv;
    // --------------------
    delete[] req_sln_rcv;
    delete[] req_sln_snd;

    delete[] stat_sln;
    // --------------------
    delete[] triag_snd_buf;
    delete[] triple_dn;
    // --------------------
    for(int i = 0; i < nx; i++){
        delete[] d_l[i];
        delete[] d_r[i];
    }

    delete[] d_l;
    delete[] d_r;
    // --------------------
    delete[] block_size;

    delete[] block_start;
    delete[] block_fin;

    delete[] block_shift;
    // --------------------
    delete[] a_down_all;
    delete[] c_down_all;
    delete[] f_down_all;
    // --------------------
}

void flush(){
    for(int J = 1; J < nx; J++){
        for(int i = 0; i < M; i++){
            d_l[J][i] = d_r[J][i] = 0;
        }
    }
}

// ----------------------
```

```
    int nx;
    int my_ny;
    // ----------------------
    int M;

    int H;
    int W;
    // ----------------------
    int i_0;
    int i_down;
    // ----------------------
    double** A;
    double* B;
    double** C;

    double** F;

    double* X;
    // ----------------------
    MPI_Request* req_diags_snd;
    MPI_Request* req_diags_rcv;

    MPI_Status* stat_recv;
    // ----------------------
    MPI_Request* req_sln_rcv;
    MPI_Request* req_sln_snd;

    MPI_Status* stat_sln;
    // ----------------------
    double* triag_snd_buf;
    double* triple_dn;
    // ----------------------
    double** d_l;
    double** d_r;
    // ----------------------
    MPI_Request req_triple_down_rcv;
    MPI_Request req_triags_snd;

    MPI_Status stat_triple_down;
    // ----------------------
    int* block_size;

    int* block_start;
    int* block_fin;

    int* block_shift;

    int block_size_max;
    // ----------------------
    double* a_down_all;
    double* c_down_all;
    double* f_down_all;
    // ----------------------

private:
    int mpirank;
    int mpisize;
};

#endif
```