Отчет

Задание №1
Решение двумерного уравнения теплопроводности
(сплошная пластина)

(явная схема, MPI+OpenMP)

выполнил
студент 510 группы ВМК МГУ
Кулагин Алексей

1. Постановка задачи

Рассмотрим задачу описания динамики распределения температуры на плоской поверхности прямоугольной пластины.

Данный процесс описывается уравнением теплопроводности с тремя независимыми переменными x, y, t:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k\left(x,y\right)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\left(x,y\right)\frac{\partial u}{\partial y}\right) + f\left(x,y,t\right), \quad (x,y) \in (a_1,b_1)\times(a_2,b_2), \tag{1}$$

где

$u(x,y,t)$ - искомая функция температуры,

$k(x,y)$ - функция распределения коэффициента температуропроводности,

$(x,y) \in (a_1,b_1)\times(a_2,b_2)$ - область пространства, занимаемая пластиной.

Пусть в начальный момент времени $t = 0$ задана температура в каждой точке пластины. Тогда граничные условия (первого рода) задаются уравнением:

$$u(x,y,0) = g_0(x,y). \tag{2}$$

Пусть, кроме того, температура на границах пластины задается функциями, зависящими от времени:

$$\begin{aligned} u(a_1,y,t) = g_{11}(t), \quad u(b_1,y,t) = g_{12}(t) \\ u(x,a_2,0) = g_{21}(t), \quad u(x,b_2,0) = g_{22}(t) \end{aligned} \tag{3}$$

Требуется определить значение функции $u(x,y,t)$ в момент времени $t = t_{max}$ $\forall (x,y) \in (a_1,b_1)\times(a_2,b_2)$, где $[0,\ t_{max}]$ - рассматриваемый промежуток времени.

2. Описание точного аналитического решения

Решение данной задачи будем производить с использованием метода конечных разностей.

Для аппроксимации дифференциального уравнения (1) разностным введем пространственно-временную сетку

$$\Omega = \omega_x \times \omega_y \times \omega_t, \tag{4}$$

$$\omega_y = \left\{ y = i \cdot h_y, \quad i = 0, \ldots, n_y, \quad h_y = \frac{b_2 - a_2}{n_y} \right\}, \tag{5}$$

$$\omega_x = \left\{ x = j \cdot h_x, \quad j = 0, \ldots, n_x, \quad h_x = \frac{b_1 - a_1}{n_x} \right\}, \tag{6}$$

$$\omega_t = \left\{ t = k \cdot dt, \quad k = 0, \ldots, n_t, \quad dt = \frac{t_{max}}{n_t} \right\}, \tag{7}$$

$n_x,\ n_y,\ n_t$ - число узлов сетки по соответствующему направлению,

$[0,\ t_{max}]$ - рассматриваемый промежуток времени.

Явная разностная схема позволяет найти значения $u(x, y, t + dt)$ при помощи значений $u(x, y, t)$, полученных для предыдущего момента времени t:

$$\frac{u_{ij}^{k+1} - u_{ij}^{k}}{dt} = \frac{1}{dy}\left\{ k_{i+1/2,j}\frac{u_{i+1,j} - u_{i,j}}{dy} \ - \ k_{i-1/2,j}\frac{u_{i,j} - u_{i-1,j}}{dy} \right\} + \frac{1}{dx}\left\{ k_{i,j+1/2}\frac{u_{i,j+1} - u_{i,j}}{dx} \ - \ k_{i,j-1/2}\frac{u_{i,j} - u_{i,j-1}}{dx} \right\} + f_{ij}^{k}$$

$u_{i,j}^{0} = g_0\left(x_j, y_i\right),$

$u_{0,j}^{k} = g_{11}\left(t_k\right), \quad u_{n_y,j}^{k} = g_{12}\left(t_k\right), \quad u_{i,0}^{k} = g_{21}\left(t_k\right), \quad u_{i,n_x}^{k} = g_{22}\left(t_k\right),$

$k_{i\pm1/2,j} = \dfrac{2k_{i,j}k_{i\pm1,j}}{k_{i,j} + k_{i\pm1,j}}, \quad k_{i,j\pm1/2} = \dfrac{2k_{i,j}k_{i,j\pm1}}{k_{i,j} + k_{i,j\pm1}}, \quad k_{i,j} = k\left(x_j, y_i\right),$

$f_{ij}^{k} = f\left(x_j, y_i, t_k\right).$

Схема устойчива при условии $dt < \dfrac{h^2}{4 \cdot k_0}, \quad k_0 = \max\limits_{x,y}\ k(x, y), \quad h = \min\{dx, dy\}.$

3. Описание метода решения и способа декомпозиции области

Процессы образуют декартову топологию "прямоугольник"(наилучшее приведение к квадрату). Каждый процесс вычисляет интервалы x, y (свои локальные фрагменты), в которых им будут производиться вычисления. Граничные и начальные условия также задаются независимо в рамках каждого отдельного процесса.

Соседние процессы обмениваются между собой смежными (прилегающими/граничными) строками и столбцами, получение которых (от других процессов) необходимо для совершения вычислительного цикла (каждым отдельным процессом).

4. Описание используемой вычислительной системы

Расчеты проводились на вычислительной системе IBM Blue Gene/P факультета ВМК МГУ.

Интерконнект для операций «точка-точка»: коммуникационная сеть с топологией «трехмерный тор».

Основные характеристики:

- сеть общего назначения, объединяющие все вычислительные узлы
- вычислительный узел имеет двунаправленные связи с шестью соседями
- пропускная способность каждого соединения — 425 MB/s (5,1 GB/s для всех 12 каналов)
- латентность (ближайший сосед): 32-байтный пакет - 0,1 μs, 256-байтный пакет - 0,8 μs

Вычислительный узел:

- четыре микропроцессорных ядра PowerPC 450 (4-way SMP)
- пропускная способность памяти: 13,6 GB/sec
- 2 ГБ общей памяти
- 2 x 4 МБ кэш-памяти 2-го уровня
- асинхронные операции межпроцессорных обменов (выполняются параллельно с вычислениями)

Всего:

- 2048 вычислительных узлов
- 2048 процессоров (1 процессор на узел)
- 8192 ядра

Для расчетов последовательно использовались 1, 2, 4, 8, 16, 32, 64, 128 и 256 ядер.

Глобальные коллективные операции и глобальные прерывания в ходе вычислений не применялись.

5. Аналитическая зависимость ожидаемого времени решения от параметров задачи и от параметров вычислительной системы

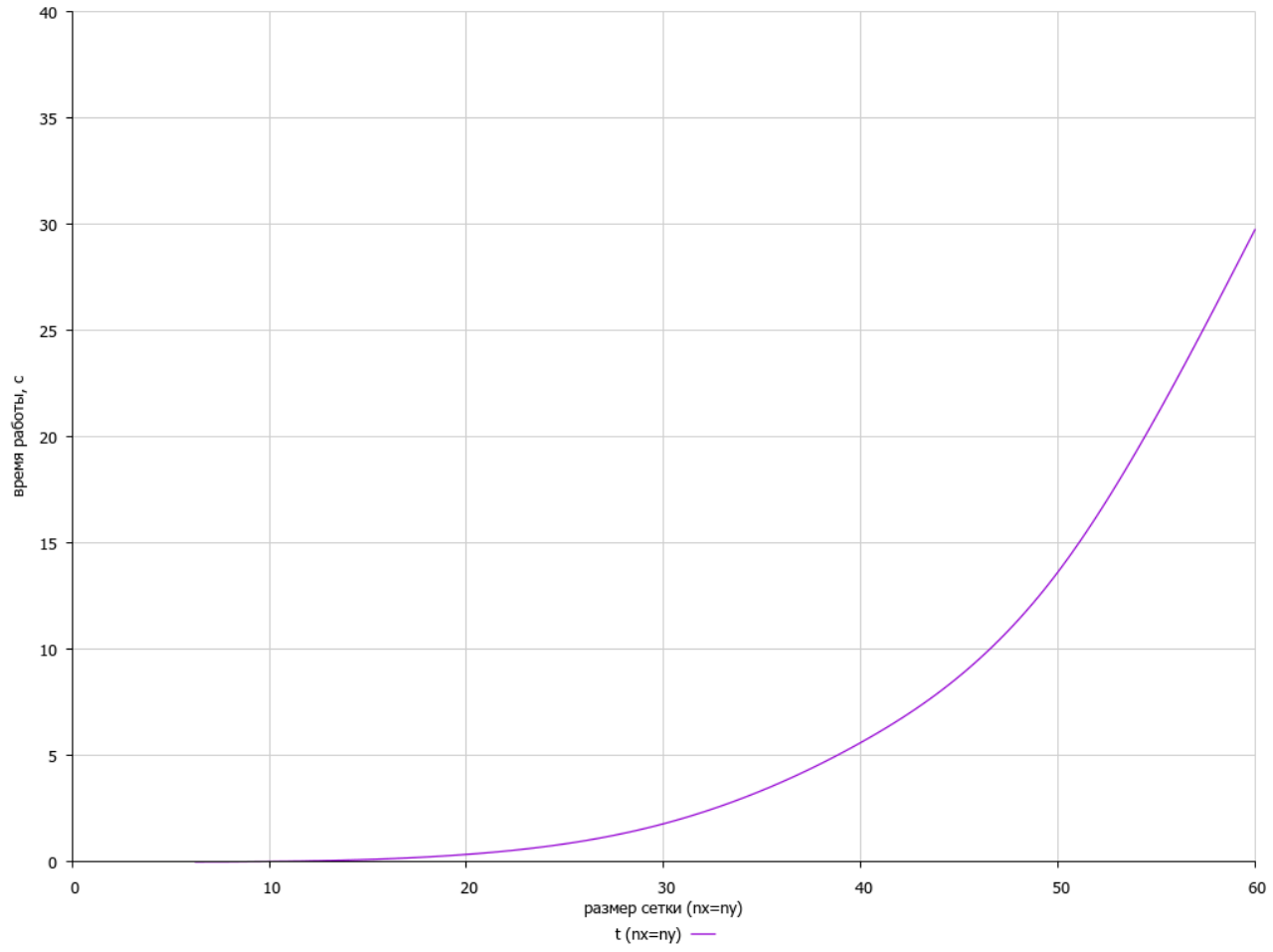
6. Сведения о значении ошибки (отклонения от точного решения) при выполнении расчетов на последовательности сгущающихся сеток

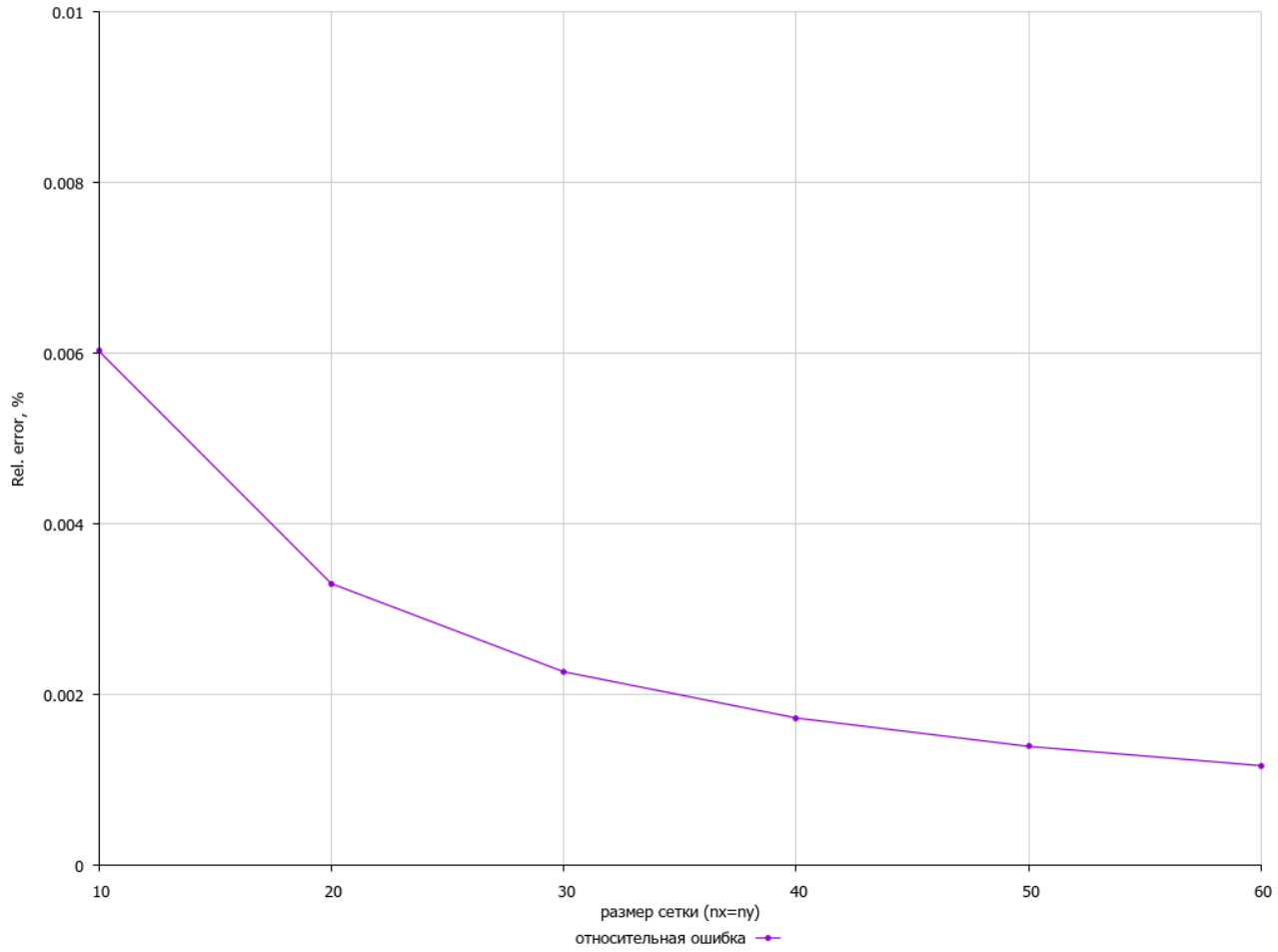
$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k\left(x,y\right)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\left(x,y\right)\frac{\partial u}{\partial y}\right) + f\left(x,y,t\right), \quad (x,y) \in (0,1)\times(0,1), \quad t_{max} = 1$$

$$f(x,y,t) = (1-2t)\cdot(x\ cos(xt) - y\ sin(yt)) + t^2\ (x^2+y^2)\cdot(sin(xt)+cos(yt))$$

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

nx	ny	nt	Время работы, с	Abs. error	Rel. error
10	10	1000	0.0230000000	0.0065972200	0.0060257522
20	20	4000	0.3550000000	0.0034625389	0.0033016513
30	30	9000	1.7990000000	0.0023432365	0.0022688815
40	40	16000	5.6200000000	0.0017703663	0.0017277178
50	50	25000	13.6510000000	0.0014224790	0.0013948626
60	60	36000	29.7740000000	0.0011888298	0.0011695014





Абсолютная ошибка: $Abs.\ error = \max\limits_{(a_1,b_2)\times(a_2,b_2)} \left\| u(x,y,t_{max}) - u_{exact}(x,y,t_{max}) \right\|$

Относительная ошибка: $Rel.\ error = \max\limits_{(a_1,b_2)\times(a_2,b_2)} \left\| \dfrac{Abs.\ error}{u_{exact}(x,y,t_{max})} \right\|$

Явная разностная схема обладает первым порядком аппроксимации по dh (здесь: $dh = dx = dy$).

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

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k\left(x,y\right)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\left(x,y\right)\frac{\partial u}{\partial y}\right) + f\left(x,y,t\right), \quad (x,y) \in (0,100)\times(0,100), \quad t_{max} = 0.001$$

$$f(x,y,t) = (1-2t)\cdot(x\ cos(xt) - y\ sin(yt)) + t^2\ (x^2+y^2)\cdot(sin(xt)+cos(yt))$$

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

1) nx = 200, ny = 200, nt = 2000

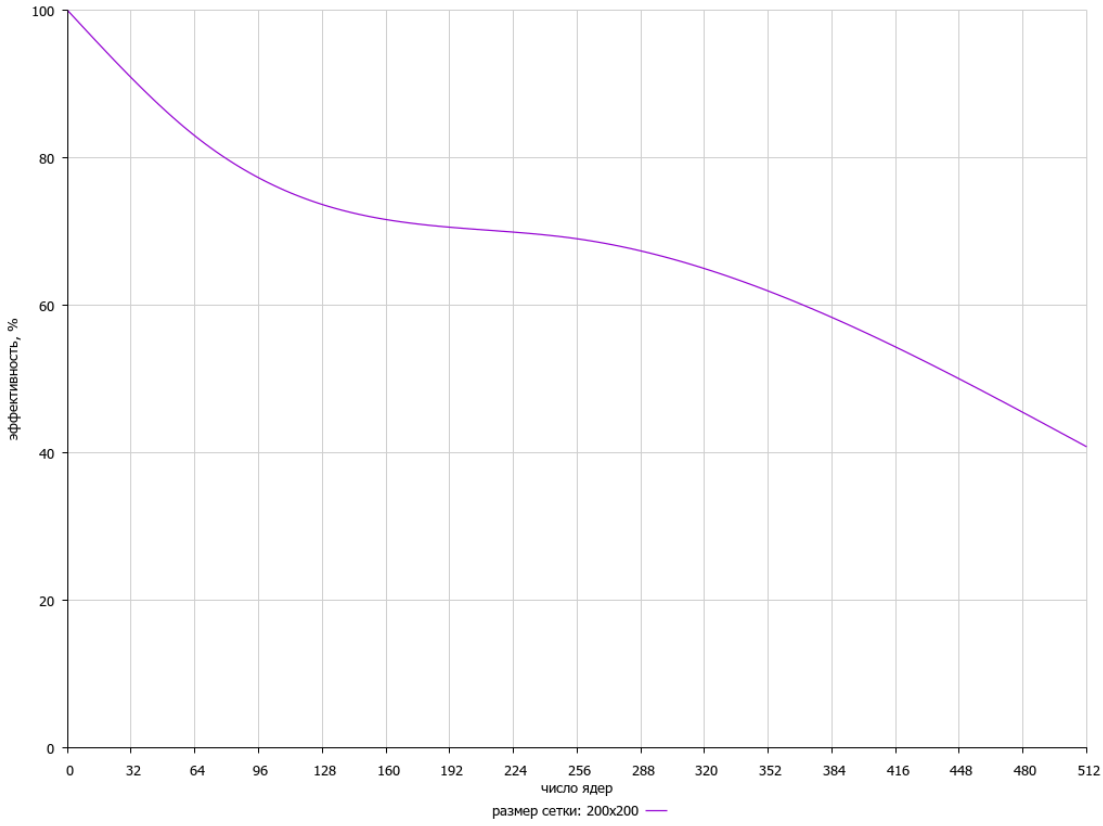
Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	60.738	1	100
16	64	1.143	53.106	82.978
32	128	0.644	94.307	73.677
64	256	0.343	176.715	69.029
128	512	0.290	209.099	40.839

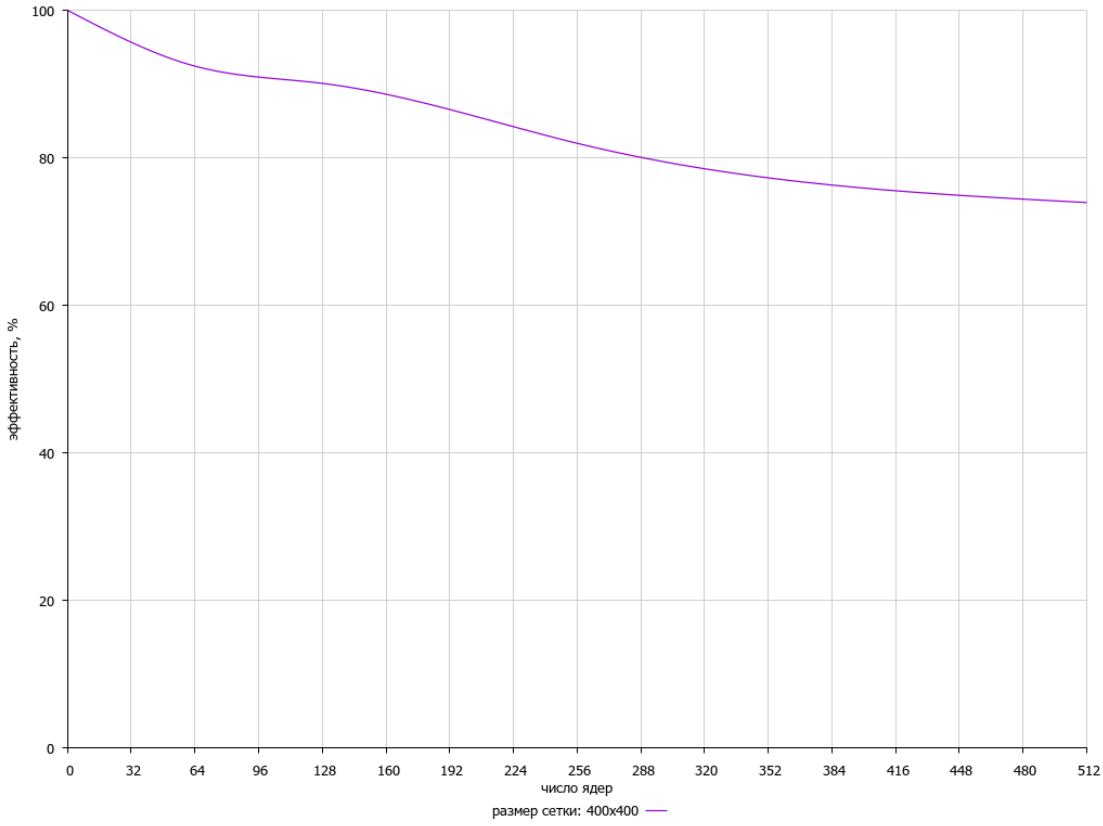
```
edu-cmc-ski16-052@fen1:~/EXPLICIT_MPI> make accel
perl accel.pl
Useless use of hash element in void context at accel.pl line 54.
Nodes      Cores          Time                   Speed Up               Efficiency, %          Abs. error          Rel. error

1          1              60.7383551518          1                      100                    0.0000020761        0.0000020605

16         64             1.1437185388           53.1060336011754       82.9781775018365       0.0000020761        0.0000020605

32         128            0.6440438047           94.3078012839396       73.6779697530778       0.0000020761        0.0000020605

64         256            0.3437062565           176.715884576282       69.0296424126102       0.0000020761        0.0000020605

128        512            0.2904757624           209.099563591678       40.8397585139996       0.0000020761        0.0000020605
```

2) nx = 400, ny = 400, nt = 4000

Число узлов	Число ядер	Время выполнения, с	Ускорение	Эффективность, %
1	1	492.934	1	100
16	64	8.332	59.157	92.433
32	128	4.274	115.317	90.091
64	256	2.348	209.851	81.973
128	512	1.302	378.589	73.943

```
edu-cmc-ski16-052@fen1:~/EXPLICIT_MPI> make accel
perl accel.pl
Useless use of hash element in void context at accel.pl line 54.
Nodes      Cores          Time                   Speed Up               Efficiency, %          Abs. error          Rel. error

1          1              492.9346045941         1                      100                    0.0000010385        0.0000010308

16         64             8.3325499435           59.1577137774764       92.4339277773068       0.0000010385        0.0000010308

32         128            4.2746001318           115.317126607239       90.0915051619053       0.0000010385        0.0000010308

64         256            2.3489715388           209.85124615257        81.9731430283477       0.0000010385        0.0000010308

128        512            1.3020277976           378.589923734897       73.9433444794721       0.0000010385        0.0000010308
```


100
80
60
40
20
0
эффективность, %
0
32
64
96
128
160
192
224
256
288
320
352
384
416
448
480
512
число ядер
размер сетки: 200x200


100
80
60
40
20
0
эффективность, %
0
32
64
96
128
160
192
224
256
288
320
352
384
416
448
480
512
число ядер
размер сетки: 400x400

8. Анализ полученных результатов

На сетке размерностью 400x400 точек с использованием 512 ядер достигнута эффективность вычислений 74%.

9. Дополнительные материалы

Компиляция и запуск

Makefile

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

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

compile_XL_O5:
mpixlcxx_r explicit_single.cpp -O5 -qhot -qstrict           -o single.x;
mpixlcxx_r explicit_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} / $time{$_};
        $accel{$_} 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) / 4
        }
    }
```

```
}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

explicit_single.cpp

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

#define sqr(x) (x*x)

using namespace std;

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

double t_max;

int nx, ny, nt;

double dx, dy, dt;

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

double** u;
double** u_k;

double _dx, _dy;
double _dx2, _dy2;
// ---------------
int mpirank;
int mpisize;

bool mpiroot;

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

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

// BEGIN---------------------------------------- K(x, y) ---------------------------------------------------
double K(double x, double y){
    return sqr(x) + sqr(y);
}

double K_ij(int i, int j){
    return K(x[j], y[i]);
}

double** K_i_plus;
double** K_i_minus;
```

```
double** K_j_plus;
double** K_j_minus;

void init_K(){
    try{
        K_j_plus = new double*[ny+1];
        K_j_minus = new double*[ny+1];

        K_i_plus = new double*[ny+1];
        K_i_minus = new double*[ny+1];

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

            K_i_plus[i] = new double[nx+1];
            K_i_minus[i] = new double[nx+1];
        }
    }catch(std::bad_alloc){
        cout << "Cannot allocate memory" << endl;

        exit(1);
    }

    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            K_j_plus[i][j] = 2 * K_ij(i, j) * K_ij(i, j+1) / (K_ij(i, j) + K_ij(i, j+1));
            K_j_minus[i][j] = 2 * K_ij(i, j) * K_ij(i, j-1) / (K_ij(i, j) + K_ij(i, j-1));

            K_i_plus[i][j] = 2 * K_ij(i, j) * K_ij(i+1, j) / (K_ij(i, j) + K_ij(i+1, j));
            K_i_minus[i][j] = 2 * K_ij(i, j) * K_ij(i-1, j) / (K_ij(i, j) + K_ij(i-1, j));
        }
    }
}
// END---------------------------------------- K(x, y) ------------------------------------------------

// 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 1000" << endl;
        cout << endl;

        exit(1);
    }

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

// 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;

    _dy = 1. / dy;
    _dx = 1. / dx;

    _dy2 = sqr(_dy);
    _dx2 = sqr(_dx);

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

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

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

    init_K();
}
// END---------------------------------------- SET GRID ------------------------------------------------

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

void set_initials(){
    for(int i = 1; i < ny; i++){
        for(int j = 1; j < nx; j++){
            u[i][j] = g_0(i, j);
        }
    }
}
// END---------------------------------------- INITIAL CONDITIONS ---------------------------------------

// BEGIN-------------------------------------- BOUNDARY CONDITIONS ---------------------------------------
```

```
double g_11(int i, int k){
    return exact(x[0], y[i], t[k]);
}

double g_12(int i, int k){
    return exact(x[nx], y[i], t[k]);
}

double g_21(int j, int k){
    return exact(x[j], y[0], t[k]);
}

double g_22(int j, int k){
    return exact(x[j], y[ny], t[k]);
}

void set_boundaries(int k){
    for(int i = 0; i <= ny; i++){
        u[i][0] = g_11(i, k);
        u[i][nx] = g_12(i, k);
    }

    for(int j = 0; j <= nx; j++){
        u[0][j] = g_21(j, k);
        u[ny][j] = g_22(j, k);
    }
}
// END---------------------------------------- BOUNDARY CONDITIONS ---------------------------------------

// BEGIN---------------------------------------- CHECK COURANT CONDITION ----------------------------------
void check_courant(){
    double max_k = 0;

    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            max_k = max(max_k, K_ij(i, j));
        }
    }

    double h = min(dx, dy);

    if(dt >= (1./4) * sqr(h) / max_k){
        cout << endl << "Current condition is not implemented" << endl << endl;

        exit(1);
    }
}
// END---------------------------------------- CHECK COURANT CONDITION ----------------------------------

// BEGIN---------------------------------------- SOLVE ---------------------------------------------------
void solve(){
    set_initials();
    set_boundaries(0);

    MPI_Barrier(MPI_COMM_WORLD);
    double t1 = MPI_Wtime();

    for(int k = 1; k <= nt; k++){
        for(int i = 1; i < ny; i++){
            for(int j = 1; j < nx; j++){
                double d_i_plus = u[i+1][j] - u[i][j];
                double d_i_minus = u[i][j] - u[i-1][j];

                double d_j_plus = u[i][j+1] - u[i][j];
                double d_j_minus = u[i][j] - u[i][j-1];

                u_k[i][j] = u[i][j] + dt * (
                    _dy2 * (K_i_plus[i][j] * d_i_plus - K_i_minus[i][j] * d_i_minus)
                    +
                    _dx2 * (K_j_plus[i][j] * d_j_plus - K_j_minus[i][j] * d_j_minus)
                    +
```

```
                    F(x[j], y[i], t[k-1])
                );
            }
        }

        swap(u_k, u);

        set_boundaries(k);
    }

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

    if(mpiroot){
        cout << "Time: " << (t2 - t1) << endl << endl;
    }
}
// END---------------------------------------- SOLVE ------------------------------------------------------

// -----------------------
void print_grid();

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

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

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

int main(int argc, char** argv){
    cout << fixed << setprecision(10);

    MPI_Init(&argc, &argv);

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

    mpiroot = (mpirank == 0);

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

            exit(1);
        }
    }

    filename = argv[1];

    set_grid();

    check_courant();

    solve();

    print_error();

    free_memory();

    MPI_Finalize();

    return 0;
}

// BEGIN---------------------------------------- PRINT GRID -------------------------------------------------
void print_y(){
    cout << "y: ";
```

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

    cout << endl << endl;
}

void print_x(){
    cout << "x: ";

    for(int i = 0; i <= nx; i++){
        cout << x[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(){
    for(int i = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            cout << u[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 << 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 = 0; i <= ny; i++){
        for(int j = 0; j <= nx; j++){
            double sln = u[i][j];

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

```
            if(sln_e == 0){
                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: " << max_abs_error << endl;
    cout << "Rel. error: " << max_rel_error << endl;
}
// END---------------------------------------- PRINT ERROR ----------------------------------------------

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

    for(int i = 0; i <= ny; i++){
        delete[] K_j_plus[i];
        delete[] K_j_minus[i];
        delete[] K_i_plus[i];
        delete[] K_i_minus[i];

        delete[] u[i];
        delete[] u_k[i];
    }

    delete[] K_j_plus;
    delete[] K_j_minus;
    delete[] K_i_plus;
    delete[] K_i_minus;

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

2) Параллельная версия

explicit_mpi.cpp

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

#define sqr(x) (x*x)

using namespace std;

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

double t_max;

int nx, ny, nt;

double dx, dy, dt;

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

double** u;
double** u_k;

double _dx, _dy;
double _dx2, _dy2;
// ---------------
bool mpiroot;

int mpirank;
int mpisize;

int nprows;
int npcols;

int my_row;
int my_col;

int my_nx;
int my_ny;

int ny_per_node;
int nx_per_node;
// ---------------

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

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

// BEGIN---------------------------------------- K(x, y) ---------------------------------------------------
double K(double x, double y){
    return sqr(x) + sqr(y);
```

```
}

double K_ij(int i, int j){
    return K(x[j], y[i]);
}

double** K_i_plus;
double** K_i_minus;

double** K_j_plus;
double** K_j_minus;

void init_K(){
    try{
        K_j_plus = new double*[my_ny+1];
        K_j_minus = new double*[my_ny+1];

        K_i_plus = new double*[my_ny+1];
        K_i_minus = new double*[my_ny+1];

        for(int i = 0; i <= my_ny; i++){
            K_j_plus[i] = new double[my_nx+1];
            K_j_minus[i] = new double[my_nx+1];

            K_i_plus[i] = new double[my_nx+1];
            K_i_minus[i] = new double[my_nx+1];
        }
    }catch(std::bad_alloc){
        cout << "Cannot allocate memory" << endl;

        exit(1);
    }

    for(int i = 0; i <= my_ny; i++){
        for(int j = 0; j <= my_nx; j++){
            K_j_plus[i][j] = 2 * K_ij(i, j) * K_ij(i, j+1) / (K_ij(i, j) + K_ij(i, j+1));
            K_j_minus[i][j] = 2 * K_ij(i, j) * K_ij(i, j-1) / (K_ij(i, j) + K_ij(i, j-1));

            K_i_plus[i][j] = 2 * K_ij(i, j) * K_ij(i+1, j) / (K_ij(i, j) + K_ij(i+1, j));
            K_i_minus[i][j] = 2 * K_ij(i, j) * K_ij(i-1, j) / (K_ij(i, j) + K_ij(i-1, j));
        }
    }
}
// END---------------------------------------- K(x, y) ------------------------------------------------

// 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 1000" << endl;
        cout << endl;
```

```
        exit(1);
    }

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

// BEGIN---------------------------------------- SET PROC GRID ------------------------------------------
void set_proc_grid(){
    MPI_Barrier(MPI_COMM_WORLD);

    nprows = sqrt(mpisize);

    while(mpisize % nprows){
        nprows--;
    }

    npcols = mpisize / nprows;

    my_row = mpirank / npcols;
    my_col = mpirank - my_row * npcols;

    MPI_Barrier(MPI_COMM_WORLD);
}
// END---------------------------------------- SET PROC GRID ------------------------------------------

// BEGIN---------------------------------------- SET GRID ----------------------------------------------
void set_grid(){
    if(mpiroot){
        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;
    }

    MPI_Bcast(&a1, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&a2, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    MPI_Bcast(&nx, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&ny, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&nt, 1, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Bcast(&dx, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dy, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&dt, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    _dy = 1. / dy;
    _dx = 1. / dx;

    _dy2 = sqr(_dy);
    _dx2 = sqr(_dx);

    set_proc_grid();

    ny_per_node = (ny + 1) / nprows;
    nx_per_node = (nx + 1) / npcols;

    int ny_tail = (ny + 1) % nprows;
    int nx_tail = (nx + 1) % npcols;

    my_ny = ny_per_node;
```

```
my_nx = nx_per_node;

if(nprows == 1){
    my_ny = ny;
}else if(my_row == 0){
    my_ny += ny_tail;
}else if(my_row != nprows-1){
    my_ny++;
}

if(npcols == 1){
    my_nx = nx;
}else if(my_col == 0){
    my_nx += nx_tail;
}else if(my_col != npcols-1){
    my_nx++;
}

MPI_Barrier(MPI_COMM_WORLD);

for(int r = 0; r < mpisize; r++){
    if(r == mpirank){
        if(my_col != 0 && my_col != npcols-1 && my_nx < 2
           ||
           my_row != 0 && my_row != nprows-1 && my_ny < 2
        ){
            cout << "Set another process grid" << endl;

            exit(1);
        }
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

MPI_Barrier(MPI_COMM_WORLD);

double x0, y0;

if(my_row == 0){
    y0 = a2;
}else{
    y0 = a2 + (ny_tail + ny_per_node * my_row - 1) * dy;
}

if(my_col == 0){
    x0 = a1;
}else{
    x0 = a1 + (nx_tail + nx_per_node * my_col - 1) * dx;
}

try{
    y = new double[my_ny+1];
    x = new double[my_nx+1];
    t = new double[nt+1];

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

    for(int i = 0; i <= my_ny; i++){
        u[i] = new double[my_nx+1];
        u_k[i] = new double[my_nx+1];
    }
}catch(std::bad_alloc){
    cout << "Cannot allocate memory" << endl;

    exit(1);
}

for(int i = 0; i <= my_ny; i++){
    y[i] = y0 + i * dy;
```

```
    }

    for(int j = 0; j <= my_nx; j++){
        x[j] = x0 + j * dx;
    }

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

    init_K();
}
// END---------------------------------------- SET GRID ------------------------------------------------

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

double g_11(int i, int k){
    return exact(x[0], y[i], t[k]);
}

double g_12(int i, int k){
    return exact(x[my_nx], y[i], t[k]);
}

double g_21(int j, int k){
    return exact(x[j], y[0], t[k]);
}

double g_22(int j, int k){
    return exact(x[j], y[my_ny], t[k]);
}

void set_initials(){
    for(int i = 1; i < my_ny; i++){
        for(int j = 1; j < my_nx; j++){
            u[i][j] = g_0(i, j);
        }
    }
}

void set_boundaries(int k){
    if(my_row == 0){
        for(int j = 0; j <= my_nx; j++){
            u[0][j] = g_21(j, k);
        }
    }

    if(my_row == nprows - 1){
        for(int j = 0; j <= my_nx; j++){
            u[my_ny][j] = g_22(j, k);
        }
    }

    if(my_col == 0){
        for(int i = 0; i <= my_ny; i++){
            u[i][0] = g_11(i, k);
        }
    }

    if(my_col == npcols - 1){
        for(int i = 0; i <= my_ny; i++){
            u[i][my_nx] = g_12(i, k);
        }
    }
}
// END---------------------------------------- BOUNDARY CONDITIONS ----------------------------------------

// BEGIN-------------------------------------- CHECK COURANT CONDITION ------------------------------------
```

```
void check_courant(){
    double my_max_k = 0;

    for(int i = 0; i <= my_ny; i++){
        for(int j = 0; j <= my_nx; j++){
            my_max_k = max(my_max_k, K_ij(i, j));
        }
    }

    double max_k[mpisize];

    MPI_Gather(&my_max_k, 1, MPI_DOUBLE, max_k, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    if(mpiroot){
        double MAX_K = 0;

        for(int i = 0; i < mpisize; i++){
            MAX_K = max(MAX_K, max_k[i]);
        }

        double h = min(dx, dy);

        if(dt >= (1./4) * sqr(h) / MAX_K){
            cout << endl << "Current condition is not implemented" << endl << endl;

            exit(1);
        }
    }
}
// END---------------------------------------- CHECK COURANT CONDITION ------------------------------------

// BEGIN---------------------------------------- SOLVE ----------------------------------------------------
void solve(){
    set_boundaries(0);
    set_initials();

    MPI_Datatype row;
    MPI_Type_contiguous(my_nx-1, MPI_DOUBLE, &row);
    MPI_Type_commit(&row);

    MPI_Datatype column;
    MPI_Type_contiguous(my_ny-1, MPI_DOUBLE, &column);
    MPI_Type_commit(&column);

    int dest_up = mpirank - npcols;
    int dest_down = mpirank + npcols;
    int dest_left = mpirank - 1;
    int dest_right = mpirank + 1;

    int source_up = dest_up;
    int source_down = dest_down;
    int source_left = dest_left;
    int source_right = dest_right;

    int snd_up_tag = 1000 * dest_up  + 2;
    int snd_dn_tag = 1000 * dest_down  + 0;

    int snd_left_tag = 1000 * dest_left  + 1;
    int snd_right_tag = 1000 * dest_right + 3;

    int rcv_up_tag = 1000 * mpirank  + 0;
    int rcv_dn_tag = 1000 * mpirank  + 2;

    int rcv_left_tag = 1000 * mpirank  + 3;
    int rcv_right_tag = 1000 * mpirank  + 1;

    // map<string, MPI_Status*> st = {
    //     {"up", new MPI_Status},
    //     {"down", new MPI_Status},
    //     {"left", new MPI_Status},
    //     {"right", new MPI_Status}
```

```
// };

// map<string, int> send_tag = {
//     {    "up", 1000 * dest_up    + 2 },
//     {  "down", 1000 * dest_down  + 0 },

//     {  "left", 1000 * dest_left  + 1 },
//     { "right", 1000 * dest_right + 3 },
// };

// map<string, int> recv_tag = {
//     {    "up", 1000 * mpirank + 0 },
//     {  "down", 1000 * mpirank + 2 },

//     {  "left", 1000 * mpirank + 3 },
//     { "right", 1000 * mpirank + 1 },
// };

vector<double> left_sd;
vector<double> right_sd;

double left[ny_per_node];
double right[ny_per_node];

MPI_Status st_rcv_up;
MPI_Status st_rcv_down;

MPI_Status st_rcv_left;
MPI_Status st_rcv_right;

MPI_Barrier(MPI_COMM_WORLD);
double t1 = MPI_Wtime();

for(int k = 1; k <= nt; k++){
    left_sd.clear();
    right_sd.clear();

    for(int i = 1; i < my_ny; i++){
        left_sd.push_back(u[i][1]);
        right_sd.push_back(u[i][my_nx-1]);
    }

    if(npcols > 1){
        if(my_col == 0){
            MPI_Send(right_sd.data(), 1, column,   dest_right, snd_right_tag, MPI_COMM_WORLD);
            MPI_Recv(          right, 1, column, source_right, rcv_right_tag, MPI_COMM_WORLD, &st_rcv_right);

            for(int i = 0; i < my_ny; i++){
                u[i+1][my_nx] = right[i];
            }
        }else if(my_col == npcols-1){
            MPI_Send(left_sd.data(), 1, column,   dest_left, snd_left_tag, MPI_COMM_WORLD);
            MPI_Recv(          left, 1, column, source_left, rcv_left_tag, MPI_COMM_WORLD, &st_rcv_left);

            for(int i = 0; i < my_ny; i++){
                u[i+1][0] = left[i];
            }
        }else{
            MPI_Send( left_sd.data(), 1, column,  dest_left,  snd_left_tag, MPI_COMM_WORLD);
            MPI_Send(right_sd.data(), 1, column, dest_right, snd_right_tag, MPI_COMM_WORLD);

            MPI_Recv( left, 1, column,  source_left,  rcv_left_tag, MPI_COMM_WORLD, &st_rcv_right);
            MPI_Recv(right, 1, column, source_right, rcv_right_tag, MPI_COMM_WORLD,  &st_rcv_left);

            for(int i = 0; i < my_ny; i++){
                u[i+1][0] = left[i];
                u[i+1][my_nx] = right[i];
            }
        }
    }
```

```
        if(nprows > 1){
            if(my_row == 0){
                MPI_Send(&u[my_ny-1][1], 1, row,   dest_down, snd_dn_tag, MPI_COMM_WORLD);
                MPI_Recv(  &u[my_ny][1], 1, row, source_down, rcv_dn_tag, MPI_COMM_WORLD, &st_rcv_down);
            }else if(my_row == nprows - 1){
                MPI_Send(&u[1][1], 1, row,   dest_up, snd_up_tag, MPI_COMM_WORLD);
                MPI_Recv(&u[0][1], 1, row, source_up, rcv_up_tag, MPI_COMM_WORLD, &st_rcv_up);
            }else{
                MPI_Send(      &u[1][1], 1, row,     dest_up,   snd_up_tag, MPI_COMM_WORLD);
                MPI_Send(&u[my_ny-1][1], 1, row, source_down, snd_dn_tag, MPI_COMM_WORLD);

                MPI_Recv(    &u[0][1], 1, row,     dest_up,   rcv_up_tag, MPI_COMM_WORLD,   &st_rcv_up);
                MPI_Recv(&u[my_ny][1], 1, row, source_down, rcv_dn_tag, MPI_COMM_WORLD, &st_rcv_down);
            }
        }

        #pragma omp parallel for
        for(int i = 1; i < my_ny; i++){
            for(int j = 1; j < my_nx; j++){
                double d_i_plus = u[i+1][j] - u[i][j];
                double d_i_minus = u[i][j] - u[i-1][j];

                double d_j_plus = u[i][j+1] - u[i][j];
                double d_j_minus = u[i][j] - u[i][j-1];

                u_k[i][j] = u[i][j] + dt * (
                    _dy2 * (K_i_plus[i][j] * d_i_plus - K_i_minus[i][j] * d_i_minus)
                    +
                    _dx2 * (K_j_plus[i][j] * d_j_plus - K_j_minus[i][j] * d_j_minus)
                    +
                    F(x[j], y[i], t[k-1])
                );
            }
        }

        swap(u_k, u);

        set_boundaries(k);
    }

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

    if(mpiroot){
        cout << "Time: " << (t2 - t1) << endl << endl;
    }
}
// END---------------------------------------- SOLVE ------------------------------------------------------

// -----------------------
void print_grid();

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

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

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

int main(int argc, char** argv){
    cout << fixed << setprecision(10);

    MPI_Init(&argc, &argv);

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

```
    mpiroot = (mpirank == 0);

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

            exit(1);
        }
    }

    filename = argv[1];

    set_grid();

    check_courant();

    solve();

    print_error();

    free_memory();

    MPI_Finalize();

    return 0;
}

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

    for(int r = 0; r < mpisize; r++){
        if(r == mpirank){
            cout << "rank: " << r << 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_x(){
    MPI_Barrier(MPI_COMM_WORLD);

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

            cout << "x: ";
            cout << nx_per_node << endl;
            for(int i = 0; i <= my_nx; i++){
                cout << x[i] << " ";
            }

            cout << endl << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}
```

```
void print_t(){
    MPI_Barrier(MPI_COMM_WORLD);

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

            cout << "t: ";

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

            cout << endl << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Barrier(MPI_COMM_WORLD);
}

void print_grid(){
    MPI_Barrier(MPI_COMM_WORLD);

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

            cout << "x: ";

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

            cout << endl << endl;

            cout << "y: ";

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

            cout << endl << endl;

            cout << "t: ";

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

            cout << endl << endl;
        }

        MPI_Barrier(MPI_COMM_WORLD);
    }

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

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

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

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

```
                for(int j = 0; j <= my_nx; j++){
                    cout << u[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);

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

            for(int i = 0; i <= my_ny; i++){
                for(int j = 0; j <= my_nx; j++){
                    cout << 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 = 1, i2 = my_ny;
    int j1 = 1, j2 = my_nx;

    if(my_row == 0){
        i1--;
    }else if(my_row == nprows-1){
        i2++;
    }

    if(my_col == 0){
        j1--;
    }else if(my_col == npcols-1){
        j2++;
    }

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

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

            if(sln_e == 0){
                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(mpiroot){
        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: " << max_abs_error << endl;
        cout << "Rel. error: " << max_rel_error << endl;
    }
}
// END---------------------------------------- PRINT ERROR ------------------------------------------------

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

    for(int i = 0; i <= my_ny; i++){
        delete[] K_j_plus[i];
        delete[] K_j_minus[i];
        delete[] K_i_plus[i];
        delete[] K_i_minus[i];

        delete[] u[i];
        delete[] u_k[i];
    }

    delete[] K_j_plus;
    delete[] K_j_minus;
    delete[] K_i_plus;
    delete[] K_i_minus;

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