Лекция 10
Параллельные сеточные вычисления

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Параллельные вычислительные технологии»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Весенний семестр, 2017

Расчет стационарного распределения тепла

- С течением времени в теле устанавливается некоторое не зависящее от времени распределение температуры (тепловое состояние выходит на стационарный режим)
- Распределение температуры в таком случае описывается уравнением теплопроводности
- Стационарное двумерное уравнение Лапласа (Laplace equation)

$$\Delta \boldsymbol{U} = \frac{\boldsymbol{d^2 U}}{\boldsymbol{dx^2}} + \frac{\boldsymbol{d^2 U}}{\boldsymbol{dy^2}} = \mathbf{0} \qquad (1)$$

- Функция $U(x, y)$ – неизвестный потенциал (теплота)
- **Задано** уравнение (1), значения функции $U(x, y)$ на границах расчетной 2D-области
- **Требуется** найти значение функции $U(x, y)$ во внутренних точках расчетной 2D-области

Расчет стационарного распределения тепла

- **Найти** решение стационарного двумерного уравнения Лапласа (Laplace equation)

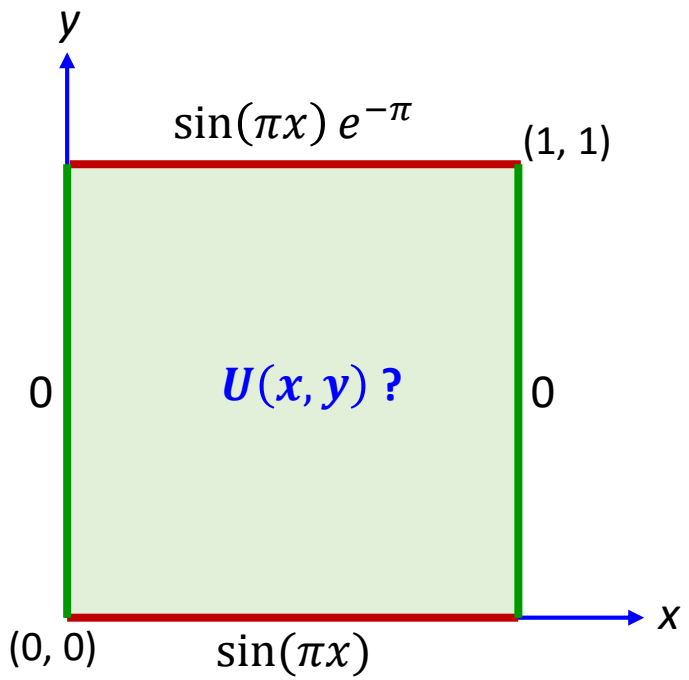
$$\frac{d^2U}{dx^2} + \frac{d^2U}{dy^2} = 0$$

- Расчётная область (domain) – квадрат [0, 1] x [0, 1]

- Граничные условия (boundary conditions):
  - $U(x, 0) = \sin(\pi x)$
  - $U(x, 1) = \sin(\pi x)\, e^{-\pi}$
  - $U(0, y) = U(1, y) = 0$

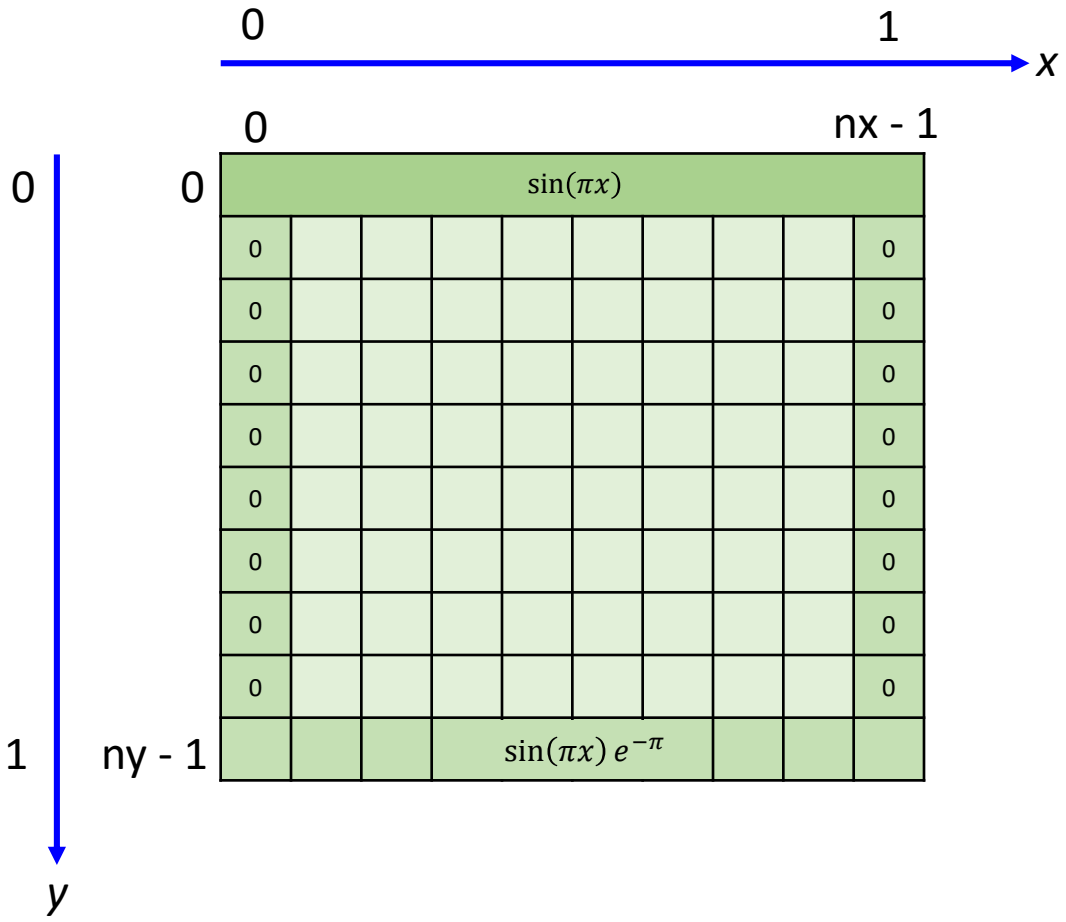
- Для данной задачи известно аналитическое решение

$$U(x, y) = \sin(\pi x)\, e^{-\pi y}$$



Дискретизация расчетной области

- Расчетная область [0, 1] x [0, 1] покрывается прямоугольной сеткой с постоянным шагом: nx точек по оси OX и ny точек по оси OY (дискретизация)



- Расчетная сетка – массив [ny, nx] чисел (температура)
- Переход от индекса ячейки [i, j] к координатам в области [0, 1] x [0, 1]:

```
x = j * 1.0 / (nx – 1.0)
y = i * 1.0 / (ny – 1.0)
```

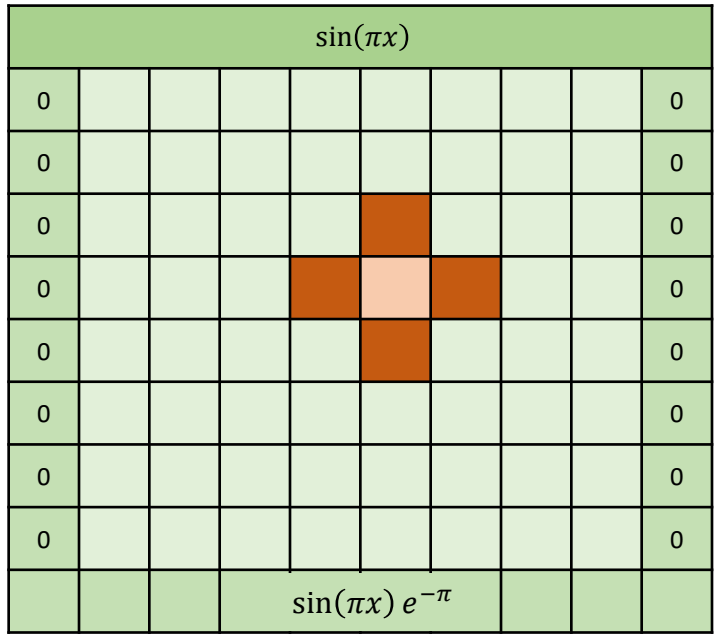
Разностная аппроксимация оператора Лапласа

- Вторые производные аппроксимируются на расчетной сетке разностным уравнением с применением четырехточечного шаблона

$$\Delta U = \frac{d^2U}{dx^2} + \frac{d^2U}{dy^2} = 0$$

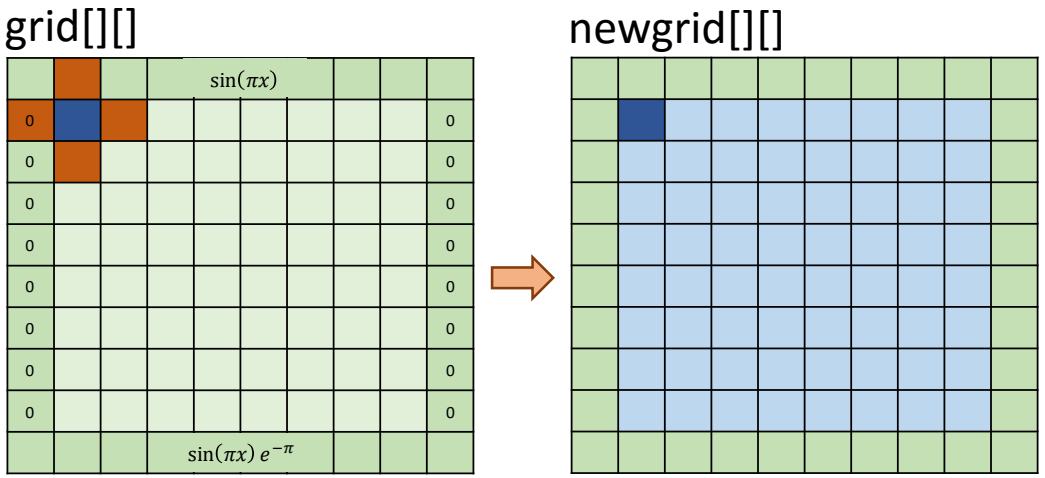
- Новое значение в каждой точке сетки равно среднему из предыдущих значений четырех ее соседних точек (схема «крест»)

```
grid_new[i ,j] = (grid[i - 1, j] + grid[i, j + 1] +
                  grid[i + 1, j] + grid[i, j - 1]) / 4
```



Метод последовательных итераций Якоби (Jacobi)

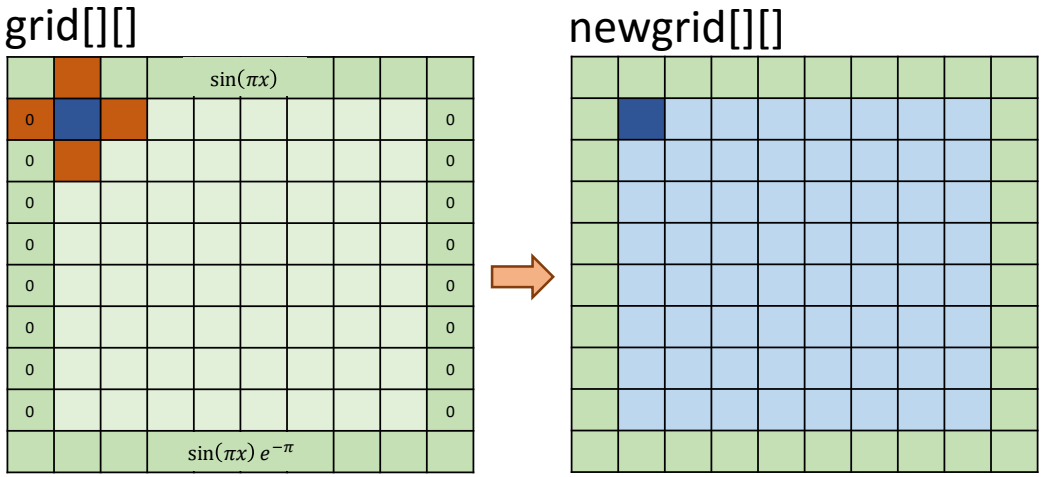
1. Инициализируем сетку grid[][]
2. Вычисляем значение в каждой внутренней ячейке сетки newgrid[i][j] как среднее из предыдущих значений четырех ее соседних точек в сетке grid (схема «крест»)




```
for (int i = 1; i < ny - 1; i++) {
    for (int j = 1; j < nx - 1; j++) {
        newgrid[IND(i, j)] = (
            grid[IND(i - 1, j)] +
            grid[IND(i + 1, j)] +
            grid[IND(i, j - 1)] +
            grid[IND(i, j + 1)]) * 0.25;
    }
}
```

Метод последовательных итераций Якоби (Jacobi)

1. Инициализируем сетку grid[][]
2. Вычисляем значение в каждой внутренней ячейке сетки newgrid[i][j] как среднее из предыдущих значений четырех ее соседних точек в сетке grid (схема «крест»)
3. Заканчиваем итерационный процесс, если максимум из попарных разностей соответствующих ячеек двух сеток меньше заданного значения EPSILON

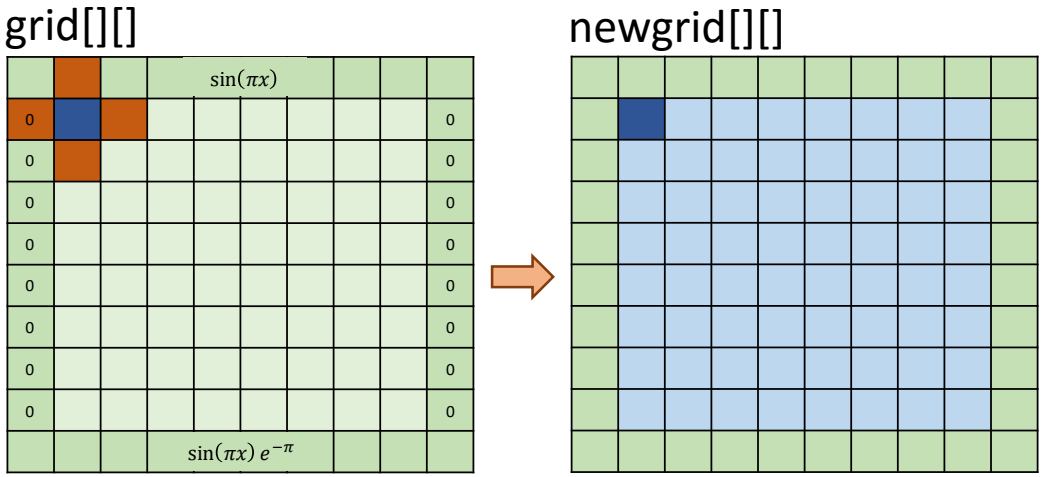



```
double maxdiff = -DBL_MAX;
for (int i = 1; i < ny - 1; i++) {
    for (int j = 1; j < nx - 1; j++) {
        int ind = IND(i, j);
        maxdiff = fmax(
            maxdiff,
            fabs(grid[ind] - newgrid[ind])
        );
    }
}

if (maxdiff < EPS) break;
```

Метод последовательных итераций Якоби (Jacobi)

1. Инициализируем сетку grid[][]
2. Вычисляем значение в каждой внутренней ячейке сетки newgrid[i][j] как среднее из предыдущих значений четырех ее соседних точек в сетке grid (схема «крест»)
3. Заканчиваем итерационный процесс, если максимум из попарных разностей соответствующих ячеек двух сеток меньше заданного значения EPSILON
4. На следующей итерации текущей делаем обновленную сетку newgrid




```
// Вместо копирования ячеек newgrid в grid,
// обмениваем значения указателей

double *p = grid;
grid = newgrid;
newgrid = p;
```

Метод итераций Якоби (последовательная версия)

```
#include <inttypes.h>
#include <math.h>
#include <float.h>
#include <sys/time.h>

#define EPS 0.001
#define PI 3.14159265358979323846

#define IND(i, j) ((i) * nx + (j))

double wtime()
{
    struct timeval t;
    gettimeofday(&t, NULL);
    return (double)t.tv_sec + (double)t.tv_usec * 1E-6;
}

int main(int argc, char *argv[])
{
    double ttotal = -wtime();
    int rows = (argc > 1) ? atoi(argv[1]) : 100;
    int cols = (argc > 2) ? atoi(argv[2]) : 100;
    const char *filename = (argc > 3) ? argv[3] : NULL;
    if (cols < 1 || rows < 1) {
        fprintf(stderr, "Invalid size of grid: rows %d, cols %d\n", rows, cols);
        exit(EXIT_FAILURE);
    }
```

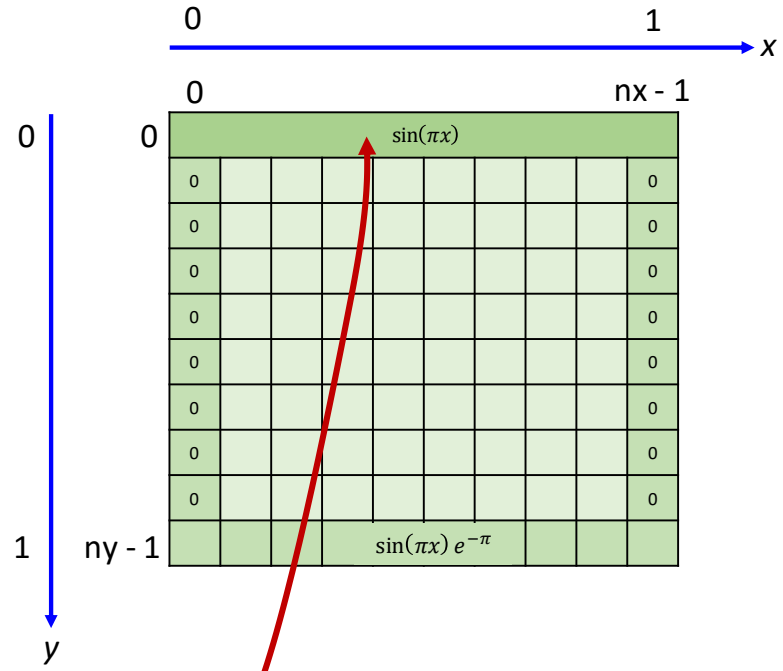
Метод итераций Якоби (последовательная версия)

```
// Allocate memory for grids [0..ny - 1][0..nx - 1]
int ny = rows;
int nx = cols;
double talloc = -wtime();

//double *local_grid = cmalloc(ny * nx, sizeof(*local_grid));
//double *local_newgrid = cmalloc(ny * nx, sizeof(*local_newgrid));
double *local_grid = malloc(ny * nx * sizeof(*local_grid));
double *local_newgrid = malloc(ny * nx * sizeof(*local_newgrid));
talloc += wtime();

double tinit = -wtime();
// Fill boundary points:
//   - left and right borders are zero filled
//   - top border: u(x, 0) = sin(pi * x)
//   - bottom border: u(x, 1) = sin(pi * x) * exp(-pi)
double dx = 1.0 / (nx - 1.0);

// Initialize top border: u(x, 0) = sin(pi * x)
for (int j = 0; j < nx; j++) {
    int ind = IND(0, j);
    local_newgrid[ind] = local_grid[ind] = sin(PI * dx * j);
}
```



Метод итераций Якоби (последовательная версия)

```
// Initialize bottom border: u(x, 1) = sin(pi * x) * exp(-pi)
for (int j = 0; j < nx; j++) {
    int ind = IND(ny - 1, j);
    local_newgrid[ind] = local_grid[ind] = sin(PI * dx * j) * exp(-PI);
}

// Initialize inner cells (we can use calloc for zeroing)
for (int i = 1; i < ny - 1; i++) {
    for (int j = 1; j < nx - 1; j++) {
        local_newgrid[IND(i, j)] = 0.0;
        local_grid[IND(i, j)] = 0.0;
    }
}
tinit += wtime();
```

Метод итераций Якоби (последовательная версия)

```
int niters = 0;
for (;;) {
    niters++;

    // Update interior points
    for (int i = 1; i < ny - 1; i++) {
        for (int j = 1; j < nx - 1; j++) {
            local_newgrid[IND(i, j)] =
                (local_grid[IND(i - 1, j)] + local_grid[IND(i + 1, j)] +
                 local_grid[IND(i, j - 1)] + local_grid[IND(i, j + 1)]) * 0.25;
        }
    }

    double maxdiff = -DBL_MAX;
    for (int i = 1; i < ny - 1; i++) {          // Check termination condition
        for (int j = 1; j < nx - 1; j++) {
            int ind = IND(i, j);
            maxdiff = fmax(maxdiff, fabs(local_grid[ind] - local_newgrid[ind]));
        }
    }

    double *p = local_grid;       // Swap grids (after termination local_grid will contain result)
    local_grid = local_newgrid;
    local_newgrid = p;

    if (maxdiff < EPS) break;
}
ttotal += wtime();
```

Метод итераций Якоби (последовательная версия)

```
    printf("# Heat 2D (serial): grid: rows %d, cols %d\n", rows, cols);
    printf("# niters %d, total time (sec.): %.6f\n", niters, ttotal);
    printf("# talloc: %.6f, tinit: %.6f, titers: %.6f\n", talloc, tinit, ttotal - talloc - tinit);

    // Save grid
    if (filename) {
        FILE *fout = fopen(filename, "w");
        if (!fout) {
            perror("Can't open file");
            exit(EXIT_FAILURE);
        }
        for (int i = 0; i < ny; i++) {
            for (int j = 0; j < nx; j++)
                fprintf(fout, "%.4f ", local_grid[IND(i, j)]);
            fprintf(fout, "\n");
        }
        fclose(fout);
    }

    return 0;
}
```

Анализ последовательной версии метода Якоби

- Инициализация сеток – $O(2N + N^2)$
- Одна итерация основного цикла метода Якоби
  - Обновление сетки – $O(N^2)$
  - Вычисление условия завершения цикла (maxdiff) – $O(N^2)$
  - Обмен указателей – $O(1)$

Профилирование последовательной версии

```
# Кластер Jet (Intel Xeon E5420)

$ ./heat 5000 5000
# Heat 2D (serial): grid: rows 5000, cols 5000
# niters 243, total time (sec.): 92.918615
# talloc: 0.000062, tinit: 0.226622, titers: 92.691931
```

- 99% времени выполнения приходится на основной цикл по итерациям метода Якоби

Профилирование последовательной версии

```
$ perf record ./heat 2000 2000
# Heat 2D (serial): grid: rows 2000, cols 2000
# niters 243, total time (sec.): 4.471221
# talloc: 0.000042, tinit: 0.020840, titers: 4.450339
[ perf record: Woken up 3 times to write data ]
[ perf record: Captured and wrote 0.700 MB perf.data (17852 samples) ]

$ perf report
```

```
Samples: 17K of event 'cycles:u', Event count (approx.): 12008426498
Overhead  Command  Shared Object      Symbol
  75.07%  heat     heat               [.] main
  24.81%  heat     libm-2.24.so       [.] __fmax
   0.06%  heat     [kernel.kallsyms]  [k] page_fault
   0.03%  heat     heat               [.] fmax@plt
   0.02%  heat     [kernel.kallsyms]  [k] __irqentry_text_start
   0.00%  heat     ld-2.24.so         [.] _dl_lookup_symbol_x
   0.00%  heat     ld-2.24.so         [.] _dl_start
   0.00%  heat     ld-2.24.so         [.] _start
```

- 75% циклов процессора пришлось на функцию main
- 24.8% циклов процессора потрачено на выполнение функции fmax

Профилирование последовательной версии

```
main  /home/data/teaching/pct/pct-spring2017/lecture10/src/heat.serial/heat
                        local_newgrid[IND(i, j)] =
                            (local_grid[IND(i - 1, j)] + local_grid[IND(i + 1, j)] +
      350:    movsd  (%rdi,%rax,8),%xmm0
 0.13         addsd  (%rsi,%rax,8),%xmm0
14.74         addsd  -0x8(%rdx,%rax,8),%xmm0
                             local_grid[IND(i, j - 1)] + local_grid[IND(i, j + 1)]) * 0.25;
 2.27         addsd  0x8(%rdx,%rax,8),%xmm0
 2.78         mulsd  0x512(%rip),%xmm0        # 4010a0 <__dso_handle+0xf8>
                  niters++;

                  // Update interior points
                  for (int i = 1; i < ny - 1; i++) {
                      for (int j = 1; j < nx - 1; j++) {
                          local_newgrid[IND(i, j)] =
12.76         movsd  %xmm0,(%rcx,%rax,8)
11.20         add    $0x1,%rax
                for (;;) {
                  niters++;

                  // Update interior points
                  for (int i = 1; i < ny - 1; i++) {
                      for (int j = 1; j < nx - 1; j++) {
 0.07           cmp    %eax,%ebp
              ↑ jg     350
      37b:    add    $0x1,%r9d
                int niters = 0;
                for (;;) {
                  niters++;

                  // Update interior points
                  for (int i = 1; i < ny - 1; i++) {
                cmp    %r10d,%ebx
 0.01           movslq %r8d,%rax
              ↑ jg     318
                mov    0x18(%rsp),%eax
                movsd  0x4cd(%rip),%xmm0        # 401080 <__dso_handle+0xd8>
Press 'h' for help on key bindings
```

Аннотированный исходный код

- **14.74% + 12.76% + 11.20%** процентов циклов процессора потрачены на обновление ячеек в главном цикле

Профилирование последовательной версии

```
main  /home/data/teaching/pct/pct-spring2017/lecture10/src/heat.serial/heat
                        int ind = IND(i, j);
                        maxdiff = fmax(maxdiff, fabs(local_grid[ind] - local_newgrid[ind]));
 10.62  3d0:    movsd  0x8(%r13,%rbp,8),%xmm1
  1.53          subsd  0x8(%rbx,%rbp,8),%xmm1
  8.72          add    $0x1,%rbp
                andpd  0x4a7(%rip),%xmm1        # 4010b0 <__dso_handle+0x108>
 23.73       → callq  fmax@plt
                }

                // Check termination condition
                double maxdiff = -DBL_MAX;
                for (int i = 1; i < ny - 1; i++) {
                    for (int j = 1; j < nx - 1; j++) {
 10.90          cmp    %rbp,%r12
  0.01        ↑ jne    3d0
                    }
                }

                // Check termination condition
                double maxdiff = -DBL_MAX;
                for (int i = 1; i < ny - 1; i++) {
  0.01  3f3:    add    $0x1,%r14d
  0.01          add    0x18(%rsp),%r15d
  0.06          cmp    0x1c(%rsp),%r14d
              ↑ jne    3a8
                // Swap grids (after termination local_grid will contain result)
                double *p = local_grid;
                local_grid = local_newgrid;
                local_newgrid = p;

                if (maxdiff < EPS)
              movsd  0x47d(%rip),%xmm1        # 4010a8 <__dso_handle+0x100>
              mov    0x20(%rsp),%r13
              mov    0x10(%rsp),%ebp
              ucomis %xmm0,%xmm1
              ja     439
Press 'h' for help on key bindings
```

Аннотированный исходный код

- **10.62% + 8.72% + 23.73% + 10.90%** процентов тактов процессора потрачены на вычисление условия завершения цикла (maxdiff)

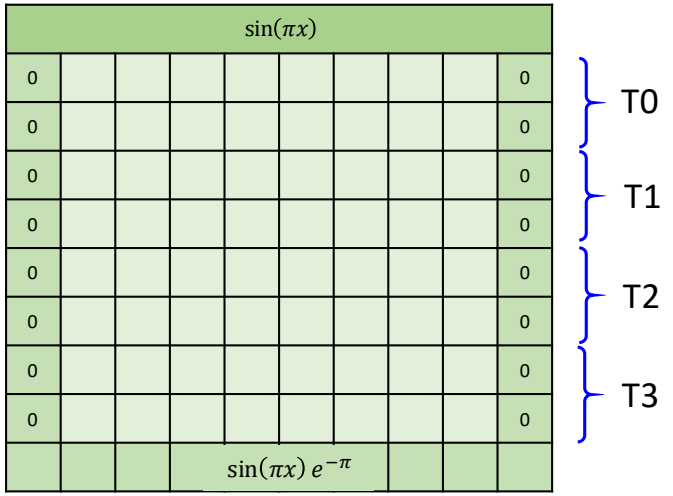
План создания параллельной версии метода Якоби

1. Распараллеливание вычисления условия завершения цикла (maxdiff) – $O(N^2)$
2. Распараллеливание обновления сетки – $O(N^2)$
3. Инициализация сеток – $O(2N + N^2)$

Параллельная версия метода Якоби (v1)

```
double tinit = -omp_get_wtime();
// Fill boundary points:
//   - left and right borders are zero filled
//   - top border: u(x, 0) = sin(pi * x)
//   - bottom border: u(x, 1) = sin(pi * x) * exp(-pi)
double dx = 1.0 / (nx - 1.0);
// Initialize top border: u(x, 0) = sin(pi * x)
#pragma omp parallel for
for (int j = 0; j < nx; j++) {
    int ind = IND(0, j);
    local_newgrid[ind] = local_grid[ind] = sin(PI * dx * j);
}
// Initialize bottom border: u(x, 1) = sin(pi * x) * exp(-pi)
#pragma omp parallel for
for (int j = 0; j < nx; j++) {
    int ind = IND(ny - 1, j);
    local_newgrid[ind] = local_grid[ind] = sin(PI * dx * j) * exp(-PI);
}
#pragma omp parallel for
for (int i = 1; i < ny - 1; i++) {
    for (int j = 1; j < nx - 1; j++) {
        local_newgrid[IND(i, j)] = 0.0;
        local_grid[IND(i, j)] = 0.0;
    }
}
tinit += omp_get_wtime();
```

Параллельная инициализация



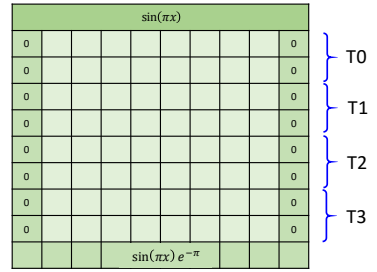
Параллельная версия метода Якоби (v1)

```
for (;;) {
    niters++;

    #pragma omp parallel for
    for (int i = 1; i < ny - 1; i++) {
        for (int j = 1; j < nx - 1; j++) {
            local_newgrid[IND(i, j)] =
                (local_grid[IND(i - 1, j)] + local_grid[IND(i + 1, j)] +
                 local_grid[IND(i, j - 1)] + local_grid[IND(i, j + 1)]) * 0.25;
        }
    }

    double maxdiff = -DBL_MAX;
    #pragma omp parallel for reduction(max:maxdiff)
    for (int i = 1; i < ny - 1; i++) {
        for (int j = 1; j < nx - 1; j++) {
            int ind = IND(i, j);
            maxdiff = fmax(maxdiff, fabs(local_grid[ind] - local_newgrid[ind]));
        }
    }

    double *p = local_grid; local_grid = local_newgrid; local_newgrid = p;
    if (maxdiff < EPS) break;
}
```



Анализ параллельного метода Якоби (v1)

1. **Верификация** – проверяем корректность работы параллельной программы

```
$ ./heat 1000 1000 result.serial
$ ./heatomp 1000 1000 result.omp
$ diff ./result.serial ./result.omp
```

2. **Анализ эффективности (масштабируемости)** – оцениваем значение коэффициента ускорения

```
$ ./heat 5000 5000
# niters 243, total time (sec.): 92.839720
# talloc: 0.000060, tinit: 0.226680, titers: 92.612980
```

```
$ OMP_NUM_THREADS=2 ./heatomp 5000 5000
# niters 243, total time (sec.): 57.545329
# talloc: 0.000018, tinit: 0.183661, titers: 57.361649
```

$S_2 = 1.6$

Низкая масштабируемость

```
$ OMP_NUM_THREADS=8 ./heatomp 5000 5000
# niters 243, total time (sec.): 52.546717
# talloc: 0.000018, tinit: 0.288810, titers: 52.257888
```

$S_8 = 1.8$

План создания параллельной версии метода Якоби (v2)

- **Сократим накладные расходы на активацию параллельных регионов #pragma omp parallel**
- Активируем параллельный регион один раз – перед запуском основного цикла по итерациям
- Совместим в одном цикле обновление сетки и вычисление условия завершения цикла
- Для сокращения числа разделяемых потоками объектов, каждый из них содержит локальные копии:
  - Указателей на сетки: p, local_grid, local_newgrid
  - Счетчика числа итераций niters

Параллельная версия метода Якоби (v2)

```
double maxdiff;
#pragma omp parallel
{
    // Thread-private copy of shared objects
    double *grid = local_grid;
    double *newgrid = local_newgrid;
    int niters = 0;

    for (;;) {
        #pragma omp barrier    // All threads finished to check break condition
        maxdiff = -DBL_MAX;
        #pragma omp barrier    // All threads updated maxdiff and ready to start reduction

        #pragma omp for reduction(max:maxdiff)
        for (int i = 1; i < ny - 1; i++) {
            for (int j = 1; j < nx - 1; j++) {
                int ind = IND(i, j);
                newgrid[ind] =
                    (grid[IND(i - 1, j)] + grid[IND(i + 1, j)] +
                     grid[IND(i, j - 1)] + grid[IND(i, j + 1)]) * 0.25;
                maxdiff = fmax(maxdiff, fabs(grid[ind] - newgrid[ind]));
            }
        }
```

Параллельная версия метода Якоби (v2)

```
            double *p = grid;     // Swap grids (after termination grid will contain result)
            grid = newgrid;
            newgrid = p;
            niters++;

            if (maxdiff < EPS)
                break;

        } // for iters

        #pragma omp barrier
        #pragma omp master
        {
            ttotal += omp_get_wtime();
            printf("# Heat 2D (OMP %d): grid: rows %d, cols %d\n", omp_get_num_threads(), rows, cols);
            printf("# niters %d, total time (sec.): %.6f\n", niters, ttotal);
            printf("# talloc: %.6f, tinit: %.6f, titers: %.6f\n", talloc, tinit, ttotal - talloc - tinit);
            // Restore shared objects
            local_grid = grid;
        }

    } // pragma omp parallel
```

Анализ параллельного метода Якоби (v2)

1. **Верификация** – проверяем корректность работы параллельной программы

2. **Анализ эффективности (масштабируемости)** – оцениваем значение коэффициента ускорения

Кластер Jet

```
$ ./heat 5000 5000
# niters 243, total time (sec.): 92.839720
# talloc: 0.000060, tinit: 0.226680, titers: 92.612980
```

```
$ OMP_NUM_THREADS=8 ./heatomp 5000 5000
# niters 243, total time (sec.): 33.912818
# talloc: 0.000018, tinit: 0.286725, titers: 33.626075
```

$S_8 = 2.8$

Анализ параллельного метода Якоби (v2)

1. **Верификация** – проверяем корректность работы параллельной программы

2. **Анализ эффективности (масштабируемости)**

Двухпроцессорный сервер – 2 x Intel Xeon E5-2620 v4 (16 ядер на сервер)

```
$ ./heat 5000 5000
# niters 243, total time (sec.): 28.630018
# talloc: 0.000089, tinit: 0.163370, titers: 28.466559
```

```
$ OMP_NUM_THREADS=8 ./heatomp 5000 5000
# niters 243, total time (sec.): 3.198653
# talloc: 0.000018, tinit: 0.029181, titers: 3.16945
```

$S_8 = 8.9$

```
$ OMP_NUM_THREADS=16 ./heatomp 5000 5000
# niters 243, total time (sec.): 2.265093
# talloc: 0.000019, tinit: 0.021303, titers: 2.243771
```

$S_{16} = 12.7$

План создания параллельной версии метода Якоби (v3)

- Изменим условие завершения итерационного процесса – завершение при достижении предельного числа итераций (не всегда допустимо!)

Параллельная версия метода Якоби (v3)

```
double maxdiff;
#pragma omp parallel
{
    // Thread-private copy of shared objects
    double *grid = local_grid;
    double *newgrid = local_newgrid;
    int niters = 0;

    for (int it = 0; it < 243; it++) {
        #pragma omp for
        for (int i = 1; i < ny - 1; i++) {
            for (int j = 1; j < nx - 1; j++) {
                newgrid[IND(i, j)] =
                    (grid[IND(i - 1, j)] + grid[IND(i + 1, j)] +
                     grid[IND(i, j - 1)] + grid[IND(i, j + 1)]) * 0.25;
            }
        }
```

Параллельная версия метода Якоби (v3)

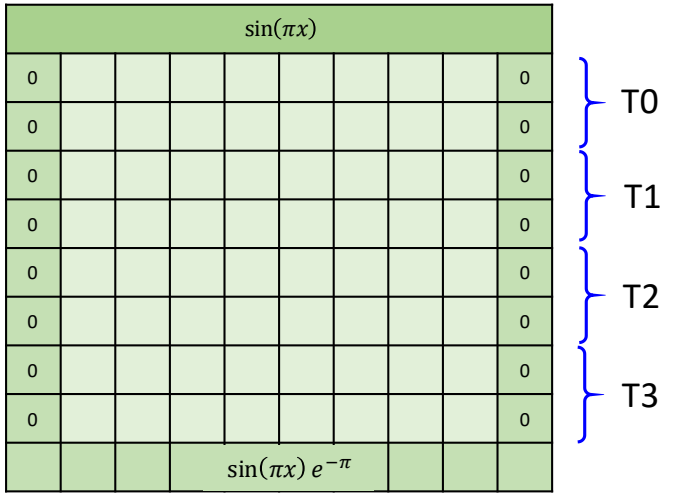
```
            // Swap grids (after termination grid will contain result)
            double *p = grid;
            grid = newgrid;
            newgrid = p;
            niters++;
        } // for iters

        #pragma omp for reduction(max:maxdiff)
        for (int i = 1; i < ny - 1; i++) {
            for (int j = 1; j < nx - 1; j++) {
                int ind = IND(i, j);
                maxdiff = fmax(maxdiff, fabs(grid[ind] - newgrid[ind]));
            }
        }
        #pragma omp master
        {
            printf("# maxdiff %.8f, total time (sec.): %.6f\n", maxdiff, ttotal);
            local_grid = grid;
        }
}
```

Активация параллельных регионов по условию (v1)

```
double tinit = -omp_get_wtime();
// Fill boundary points:
//   - left and right borders are zero filled
//   - top border: u(x, 0) = sin(pi * x)
//   - bottom border: u(x, 1) = sin(pi * x) * exp(-pi)
double dx = 1.0 / (nx - 1.0);
// Initialize top border: u(x, 0) = sin(pi * x)
#pragma omp parallel for if (nx > 1000)
for (int j = 0; j < nx; j++) {
    int ind = IND(0, j);
    local_newgrid[ind] = local_grid[ind] = sin(PI * dx * j);
}
// Initialize bottom border: u(x, 1) = sin(pi * x) * exp(-pi)
#pragma omp parallel for if (nx > 1000)
for (int j = 0; j < nx; j++) {
    int ind = IND(ny - 1, j);
    local_newgrid[ind] = local_grid[ind] = sin(PI * dx * j) * exp(-PI);
}
#pragma omp parallel for if (ny > omp_get_num_threads() * 100)
for (int i = 1; i < ny - 1; i++) {
    for (int j = 1; j < nx - 1; j++) {
        local_newgrid[IND(i, j)] = 0.0;
        local_grid[IND(i, j)] = 0.0;
    }
}
tinit += omp_get_wtime();
```

Параллельная инициализация



Объединение вложенных циклов

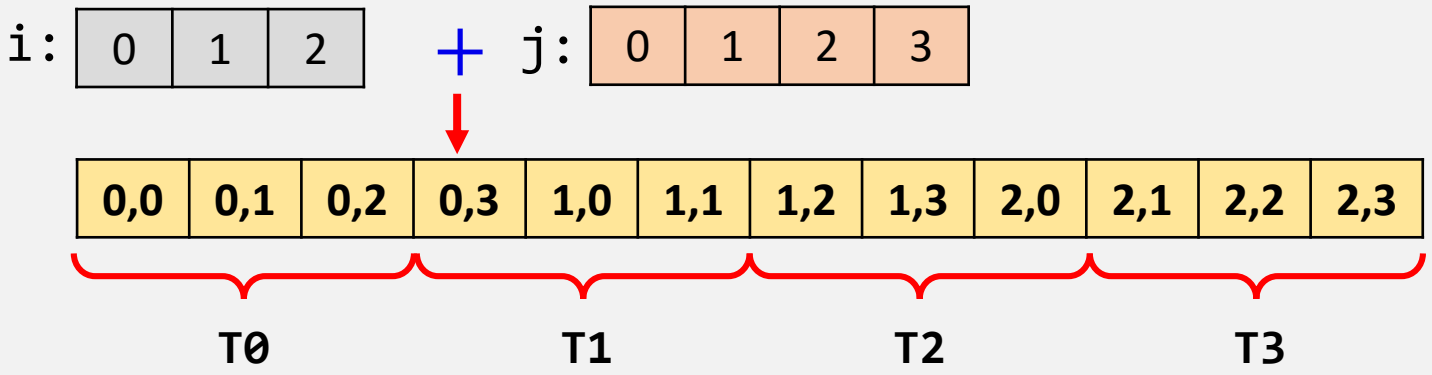
```
// N < количество потоков; как эффективно загрузить потоки?
for (j = 0; j < N; j++) {
    for (i = 0; i < M; i++) {
        A[i][j] = work(i, j);
    }
}
```

```
// Объединяем циклы
#pragma omp parallel for private(j, i)
for (ij = 0; ij < N * M; ij++) {
    j = ij / M;
    i = ij % M;
    A[i][j] = work(i, j);
}
```

Объединение пространств итераций циклов

```
#define N 3
#define M 4

#pragma omp parallel
{
    #pragma omp for collapse(2)
    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++)
            printf("Thread %d i = %d\n", omp_get_thread_num(), i);
    }
}
```



Директива **collapse(n)** объединяет пространства итераций *n* циклов