

Децентрализованная “выживающая” схема

Тестовый пример: heat2d

Библиотека отказоустойчивости: UFLM

Параллельная программа решения двумерного уравнения теплопроводности

- ❑ Решение двумерного уравнения теплопроводности, реализовано методом *последовательных итераций Якоби*
- ❑ Метод распараллелен в стандарте MPI путем *двумерной декомпозиции расчетной области*
- ❑ Каждому процессу назначена *подматрица и теньевые ячейки* по всем четырем направлениям, для расчета значений на границах выделенной области
- ❑ Для обеспечения восстановления вычислительного процесса в программу добавлен функционал *формирования контрольных точек*

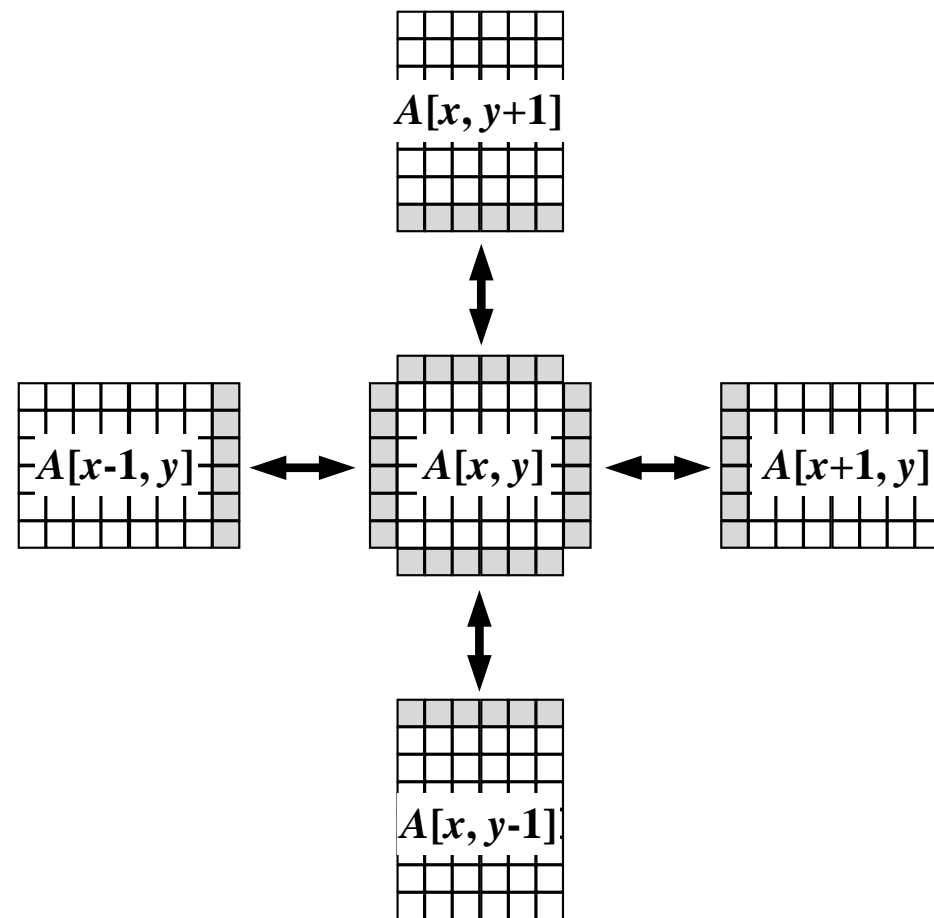


Схема обмена теньевыми ячейками подматрицы $A[x, y]$

Параллельная программа решения двумерного уравнения теплопроводности

```
while true do
  for 1 to ny
    for 1 to nx
      // Расчет значений внутренних точек
    end for
  end for

  for 1 to ny
    for 1 to nx
      maxdiff = fmax(maxdiff, ...)
    end for
  end for

  if maxdiff < EPS then
    // Проверка условия на выход
    break
  end if

  // Обмен теньвыми ячейками с соседями
  // Создание контрольной точки
end while
```

Главный цикл вычисления значений в подматрице $A[x, y]$
и обмена теньвыми ячейками

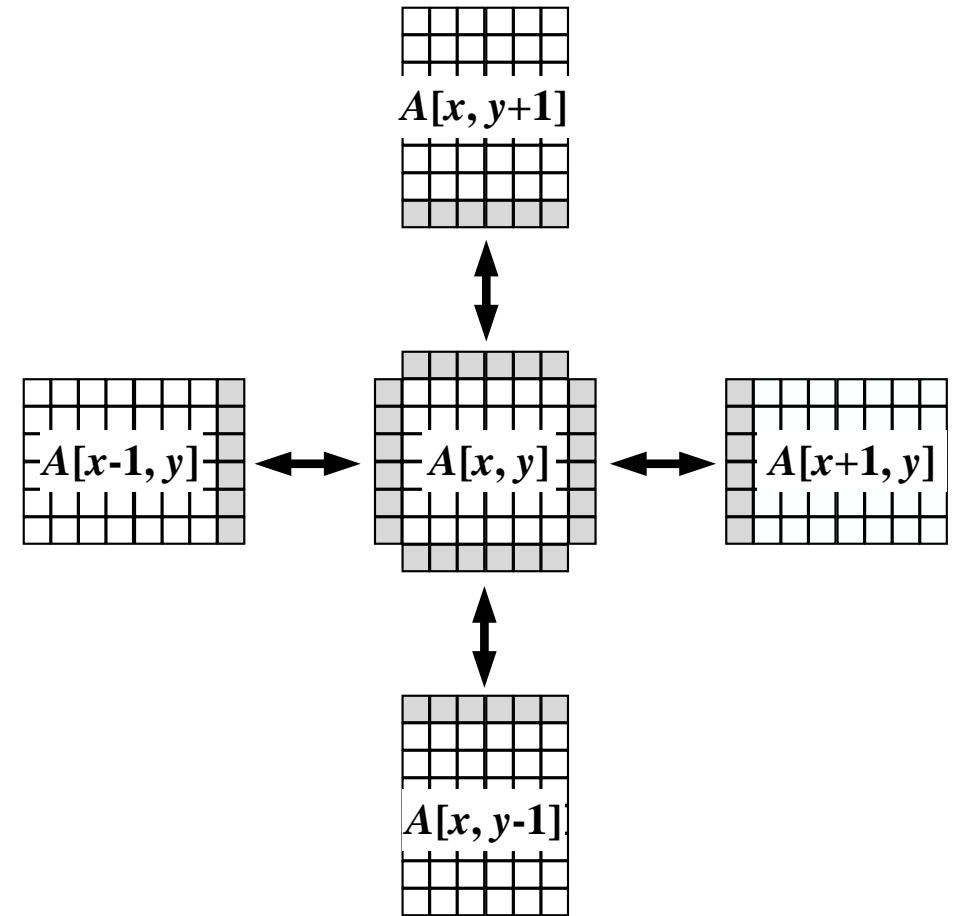


Схема обмена теньвыми ячейками подматрицы $A[x, y]$

Исходный код тестовой программы

```
int main(int argc, char *argv[])
{
    /* Block Initialize */
    /* Save checkpoint function */
    for (;;) {
        /* Update interior points */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; 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;
            }
        }
        /* Check termination condition */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; j++) {
                maxdiff = fmax(args);
            }
        }
        /* Exchange shadow points with neighbors */
        MPI_Irecv(args);
        MPI_Isend(arg);
        /* Save checkpoint function */
        /* In simulation case:
         * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
         */
    }
    /* Save checkpoint function */
    /* Block show result */
}
```

Паспорт задачи для каждого процесса

□ Хранить в памяти каждого процесса 2D паспорт задач:

```
typedef struct
{
    int    number; // block number
    int    rank;   // processor rank
    size_t x;      // size by 'x'
    size_t y;      // size by 'y'
} task_t;

task_t **gird_task;
```

1	2	3
4	5	6
7	8	9

Модификация исходного кода тестовой программы

1	1	2
3	4	5
6	7	8

task_t **gird_task;

```
int main(int argc, char *argv[])
{
    /* Block Initialize */
    /* Save checkpoint function */
    for (;;) {
        /* Update interior points */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; 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;
            }
        }
        /* Check termination condition */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; j++) {
                maxdiff = fmax(args);
            }
        }
        /* Exchange shadow points with neughbors */
        MPI_Irecv(args);
        MPI_Isend(arg);
        /* Save checkpoint function */
        /* In simulation case:
         * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
         */
    }
    /* Save checkpoint function */
    /* Block show result */
}
```

Модификация исходного кода тестовой программы

1	1	2
3	4	5
6	7	8

`task_t **gird_task;`

```
int main(int argc, char *argv[])
{
    /* Block Initialize */
    /* Save checkpoint function */
    for (;;) {
        /* Update interior points */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; 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;
            }
        }
        /* Check termination condition */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; j++) {
                maxdiff = fmax(args);
            }
        }
        /* Exchange shadow points with neighbors */
        MPI_Irecv(args);
        MPI_Isend(arg);
        /* Save checkpoint function */
        /* In simulation case:
         * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
         */
    }
    /* Save checkpoint function */
    /* Block show result */
}
```

Модификация исходного кода тестовой программы

```
int main(int argc, char *argv[])
{
    /* Block Initialize */
    /* Save checkpoint function */
    for (;;) {
        /* Update interior points */
```

1	1
3	4
6	7
	8

```
rc = MPI_Allreduce(MPI_IN_PLACE, &maxdiff, 1, MPI_DOUBLE, MPI_MAX,
MPI_COMM_WORLD);
if (MPI_ERR_PROC_FAILED == rc)
{
    do_repair();
};
```

task_t **gird_task;

```
    }
    /* Exchange shadow points with neughbors */
    MPI_Irecv(args);
    MPI_Isend(arg);
    /* Save checkpoint function */
    /* In simulation case:
    * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    */
}
/* Save checkpoint function */
/* Block show result */
}
```


Модификация исходного кода тестовой программы

1	1	2
3	4	5
6	7	8

`task_t **gird_task;`

```
int main(int argc, char *argv[])
{
    /* Block Initialize */
    /* Save checkpoint function */
    for (;;) {
        /* Update interior points */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; 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;
            }
        }
        /* Check termination condition */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; j++) {
                maxdiff = fmax(args);
            }
        }
        /* Exchange shadow points with neughbors */
        MPI_Irecv(args);
        MPI_Isend(arg);
        /* Save checkpoint function */
        /* In simulation case:
         * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
         */
    }
    /* Save checkpoint function */
    /* Block show result */
}
```

Модификация исходного кода тестовой программы

1
3
6

- ☐ Отказаться от обмена теневыми ячейками
- ☐ Обмениваться целым блоком (подматрицей процесса)
- ☐ Таким образом каждый процесс сможет «на лету» подхватывать вычисления в случае выхода из строя соседнего процесса

```
task_t **gird_task;
```

```
    MPI_Irecv(args);  
    MPI_Isend(arg);  
    /* Save checkpoint function */  
    /* In simulation case:  
     * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);  
     */  
}  
/* Save checkpoint function */  
/* Block show result */  
}
```

Модификация исходного кода тестовой программы

```
int main(int argc, char *argv[])
{
    /* Block Initialize */
    /* Save checkpoint function */
    for (;;) {
        /* Update interior points */
        for (int i = 1; i <= ny; i++) {
            for (int j = 1; j <= nx; j++) {
                local_newgrid[IND(i, j)] =
                    (local_grid[IND(i - 1, j)] +
```

1	1	
3	4	
6	7	

```
rc = MPI_Waitall(8, reqs, MPI_STATUS_IGNORE);
if (MPI_ERR_PROC_FAILED == rc)
{
    do_repair();
};
```

task_t **gird_task;

```
/* Sync point */
/* In simulation case:
 * call -> MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
 */
}
/* Save checkpoint function */
/* Block show result */
}
```