

Московский Государственный Университет имени М. В. Ломоносова  
Факультет вычислительной математики и кибернетики

**Практикум по курсу  
"Распределенные системы"**

**Улучшение задания jacobi-1d**

**Отчет**

студента 428 учебной группы

Ковтуна Данилы Петрович

Москва, 2021

## **Содержание**

- 1) Постановка задачи
- 2) Описание алгоритма
- 3) Реализация алгоритма
- 4) Запуск и проверка программы
- 5) Временная оценка

## **Постановка задачи**

Доработать MPI-программу, реализованную в рамках курса “Суперкомпьютеры и параллельная обработка данных”. Добавить контрольные точки для продолжения работы программы в случае сбоя. Реализовать один из 3-х сценариев работы после сбоя:

- a) продолжить работу программы только на “исправных” процессах;
- b) вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов;
- c) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

## Описание алгоритма

Из доступных на старте программы процессов будем оставлять некоторое количество в качестве запасных. Когда какой-нибудь из работающих процессов упадет, будем выделять один из запасных процессов на его место, а итерацию перезапускать на новом коммуникаторе, полученном из старого путем удаления упавших процессов. Пересоздание коммуникатора в таком случае можно запускать после каждой итерации.

Чтобы падения процессов не приводили к падению программы, зарегистрируем обработчик на возврат кодов ошибок из MPI\_ функций взаимодействия. На возвращаемые значения функций можно не смотреть, т.к. на каждой итерации у нас происходит MPI\_Comm\_shrink(), после которого все процессы в коммуникаторе будут живыми.

Операция MPI\_Comm\_shrink() может перенумеровать процессы, поэтому после её вызова нужно обновлять ранки.

Чтобы сохранять промежуточные вычисления, после каждой итерации для всех работающих процессов будем сохранять матрицы A и B в файлы. В начале итерации будем читать файлы, соответствующие процессу с указанным ранком.

## Реализация алгоритма

```
/* Include benchmark-specific header. */
#include "jacobi-1d.h"
#include <mpi.h>
#include <mpi-ext.h>
#include <signal.h>
double bench_t_start, bench_t_end;
int size, rank;
MPI_Comm comm;

double *A;
double *B;

static
double rtclock() {
    struct timeval Tp;
    int stat;
    stat = gettimeofday (&Tp, NULL);
    if (stat != 0)
        printf ("Error return from gettimeofday: %d", stat);
    return (Tp.tv_sec + Tp.tv_usec * 1.0e-6);
}

void bench_timer_start() {
    bench_t_start = rtclock ();
}

void bench_timer_stop() {
    bench_t_end = rtclock ();
}

void bench_timer_print() {
    printf ("Time in seconds = %0.6lf\n", bench_t_end - bench_t_start);
}

static
void init_array (int n, double *A, double *B) {
    int i;

    for (i = 0; i < n; i++) {
        A[i] = ((double) i + 2) / n;
        B[i] = ((double) i + 3) / n;
    }
}
```

```

static
void print_array(int n, double *A) {
    int i;

    fprintf(stderr, "==BEGIN DUMP_ARRAYS==\n");
    fprintf(stderr, "begin dump: %s", "A");
    for (i = 0; i < n; i++) {
        if (i % 20 == 0)
            fprintf(stderr, "\n");
        fprintf(stderr, "%0.21f ", A[i]);
    }
    fprintf(stderr, "==END DUMP_ARRAYS==\n");
}

static
void kernel_jacobi_1d(int tsteps, int n) {
    int t, i, count, ibeg, iend, right_rank, left_rank;
    MPI_Status status;
    MPI_Request req;
    right_rank = 0;
    left_rank = 0;

    // половина процессов запасные
    size = size / 2 + 1;

    count = (n - 3) / size + 1;
    ibeg = rank * count + 1;
    iend = (rank + 1) * count;
    // Если нитей больше чем данных, которые необходимо обработать
    if (ibeg >= n - 2) {
        ibeg = -1;
        iend = -1;
    }
    // У последней нити, которая попадает в нужный нам диапазон меняем iend на
    // нужный нам и
    // сообщаем какой у нити ранг процессу size - 1
    if (iend >= n - 3) {
        iend = n - 3;
        right_rank = size - 1;
        MPI_Isend(&rank, 1, MPI_INT, size - 1, 0, comm, &req);
    }
    // У нити size - 1 корректируем ibeg & iend и получаем номер последней
    // нормальной нити
    if (rank == size - 1) {
        MPI_Recv(&left_rank, 1, MPI_INT, MPI_ANY_SOURCE, 0, comm, &status);
        if (ibeg == -1)
            ibeg = n - 2;
        iend = n - 2;
        //Случай, когда нормально разделилось без постороннего вмешательства
        if (left_rank == right_rank)

```

```

        left_rank--;
    }

    if (ibeg != -1) {
        char fname[50];
        sprintf(fname, "matrixes/A%d.txt\0", rank);
        FILE* fdA = fopen(fname, "w");
        for (int i = 0; i < n; ++i) {
            fprintf(fdA, "%lf ", A[i]);
        }
        fclose(fdA);

        sprintf(fname, "matrixes/B%d.txt\0", rank);
        FILE* fdB = fopen(fname, "w");
        for (int i = 0; i < n; ++i) {
            fprintf(fdB, "%lf ", B[i]);
        }
        fclose(fdB);
    }

    for (t = 0; t < tsteps; t++) {
        static int cnt = 0;
        ++cnt;

        right_rank = 0;
        left_rank = 0;
        count = (n - 3) / size + 1;
        ibeg = rank * count + 1;
        iend = (rank + 1) * count;
        // Если нитей больше чем данных, которые необходимо обработать
        if (ibeg >= n - 2) {
            ibeg = -1;
            iend = -1;
        }
        // У последней нити, которая попадает в нужный нам диапазон меняем iend
на нужный нам и
        // сообщаем какой у нити ранг процессу size - 1
        if (iend >= n - 3) {
            iend = n - 3;
            right_rank = size - 1;
            MPI_Send(&rank, 1, MPI_INT, size - 1, 0, comm);
        }
        // У нити size - 1 корректируем ibeg & iend и получаем номер последней
нормальной нити
        if (rank == size - 1) {
            MPI_Recv(&left_rank, 1, MPI_INT, MPI_ANY_SOURCE, 0, comm, &status);
            if (ibeg == -1)
                ibeg = n - 2;
            iend = n - 2;
            //Случай, когда нормально разделилось без постороннего вмешательства
            if (left_rank == right_rank)

```

```

        left_rank--;
    }

    int err = MPI_Barrier(comm);
    if (err != MPI_SUCCESS) {
        t = t - 2;
        MPIX_Comm_shrink(comm, &comm);
        MPI_Comm_rank(comm, &rank);

        char fname[50];
        sprintf(fname, "matrixes/A%d.txt\0", rank);
        FILE* fdA = fopen(fname, "r");
        for (int i = 0; i < n; ++i) {
            fscanf(fdA, "%lf ", &A[i]);
        }
        fclose(fdA);

        sprintf(fname, "matrixes/B%d.txt\0", rank);
        FILE* fdB = fopen(fname, "r");
        for (int i = 0; i < n; ++i) {
            fscanf(fdB, "%lf ", &B[i]);
        }
        fclose(fdB);

        continue;
    }

    char fname[50];
    sprintf(fname, "matrixes/A%d.txt\0", rank);
    FILE* fdA = fopen(fname, "w");
    for (int i = 0; i < n; ++i) {
        fprintf(fdA, "%lf ", A[i]);
    }
    fclose(fdA);

    sprintf(fname, "matrixes/B%d.txt\0", rank);
    FILE* fdB = fopen(fname, "w");
    for (int i = 0; i < n; ++i) {
        fprintf(fdB, "%lf ", B[i]);
    }
    fclose(fdB);

    if (ibeg == -1) {
        continue;
    }

    if (cnt == 1 && rank == 1) {
        raise(SIGKILL);
    }

    for (i = ibeg; i <= iend; i++)

```



```

        B[i] = 0.33333 * (A[i - 1] + A[i] + A[i + 1]);

    if (rank == 0) {
        if (size != 1) {
            MPI_Send(&B[iend], 1, MPI_DOUBLE, rank + 1, 0, comm);
            MPI_Recv(&B[iend + 1], 1, MPI_DOUBLE, rank + 1, 0, comm,
&status);
        }
    }
    // Кидаем данные левой нити
    else if (rank == size - 1) {
        MPI_Send(&B[ibeg], 1, MPI_DOUBLE, left_rank, 0, comm);
        MPI_Recv(&B[ibeg - 1], 1, MPI_DOUBLE, left_rank, 0, comm, &status);
    }
    // кидаем данные нити size - 1
    else if (right_rank) {
        MPI_Send(&B[iend], 1, MPI_DOUBLE, right_rank, 0, comm);
        MPI_Send(&B[ibeg], 1, MPI_DOUBLE, rank - 1, 0, comm);
        MPI_Recv(&B[iend + 1], 1, MPI_DOUBLE, right_rank, 0, comm, &status);
        MPI_Recv(&B[ibeg - 1], 1, MPI_DOUBLE, rank - 1, 0, comm, &status);
    }
    // Кидаем и получаем данные у соседних нитей
    else {
        MPI_Send(&B[iend], 1, MPI_DOUBLE, rank + 1, 0, comm);
        MPI_Send(&B[ibeg], 1, MPI_DOUBLE, rank - 1, 0, comm);
        MPI_Recv(&B[iend + 1], 1, MPI_DOUBLE, rank + 1, 0, comm, &status);
        MPI_Recv(&B[ibeg - 1], 1, MPI_DOUBLE, rank - 1, 0, comm, &status);
    }

    // Аналогично для матрицы A
    for (i = ibeg; i <= iend; i++)
        A[i] = 0.33333 * (B[i-1] + B[i] + B[i + 1]);

    if (rank == 0) {
        if (size != 1) {
            MPI_Send(&A[iend], 1, MPI_DOUBLE, rank + 1, 0, comm);
            MPI_Recv(&A[iend + 1], 1, MPI_DOUBLE, rank + 1, 0, comm,
&status);
        }
    }
    else if (rank == size - 1) {
        MPI_Send(&A[ibeg], 1, MPI_DOUBLE, left_rank, 0, comm);
        MPI_Recv(&A[ibeg - 1], 1, MPI_DOUBLE, left_rank, 0, comm, &status);
    }
    else if (right_rank) { // != 0
        MPI_Send(&A[iend], 1, MPI_DOUBLE, right_rank, 0, comm);
        MPI_Send(&A[ibeg], 1, MPI_DOUBLE, rank - 1, 0, comm);
        MPI_Recv(&A[iend + 1], 1, MPI_DOUBLE, right_rank, 0, comm, &status);
        MPI_Recv(&A[ibeg - 1], 1, MPI_DOUBLE, rank - 1, 0, comm, &status);
    }
    else {

```

```

        MPI_Send(&A[iend], 1, MPI_DOUBLE, rank + 1, 0, comm);
        MPI_Send(&A[ibeg], 1, MPI_DOUBLE, rank - 1, 0, comm);
        MPI_Recv(&A[iend + 1], 1, MPI_DOUBLE, rank + 1, 0, comm, &status);
        MPI_Recv(&A[ibeg - 1], 1, MPI_DOUBLE, rank - 1, 0, comm, &status);
    }
}

// Собираем новые данные на нити size - 1
if (ibeg != -1 && size != 1) {
    if (rank != size - 1) {
        MPI_Send(&B[ibeg], iend - ibeg + 1, MPI_DOUBLE, size - 1, 0, comm);
    }
    else {
        int i;
        for (i = 0; i <= left_rank; i++) {
            MPI_Recv(&B[i * count + 1], (i == left_rank) ? (n - 3 - i *
count) : count, MPI_DOUBLE, i, 0, comm, &status);
        }
    }
}

if (ibeg != -1 && size != 1) {
    if (rank != size - 1) {
        MPI_Send(&A[ibeg], iend - ibeg + 1, MPI_DOUBLE, size - 1, 0, comm);
        //printf("%d %d %d\n", rank, ibeg, ibeg + count - 1);
    }
    else {
        int i;
        for (i = 0; i <= left_rank; i++) {
            MPI_Recv(&A[i * count + 1], (i == left_rank) ? (n - 3 - i *
count) : count, MPI_DOUBLE, i, 0, comm, &status);
        }
    }
}
}

int main(int argc, char *argv[]) {
    int n_s[] = {30, 120, 400, 2000, 4000, 8000, 16000, 32000, 64000};
    int step_s[] = {20, 40, 100, 500, 1000, 2000, 4000, 8000, 16000};
    int i;

    MPI_Init(&argc, &argv);

    comm = MPI_COMM_WORLD;
    MPI_Comm_set_errhandler(comm, MPI_ERRORS_RETURN);
    MPI_Comm_size(comm, &size);
    MPI_Comm_rank(comm, &rank);

```

```

for (i = 0; i < 1; i++) {
    int n = n_s[i];
    int tsteps = step_s[i];

    if (rank == size - 1) {
        printf("n=%d tsteps=%d threads=%d\n", n, tsteps, size);
    }

    A = (double *) malloc (n * sizeof(double));
    B = (double *) malloc (n * sizeof(double));

    init_array (n, A, B);

    MPI_Barrier(comm);
    if (rank == size - 1) {
        printf("Начато замеряться время\n");
        bench_timer_start();
    }

    kernel_jacobi_1d(tsteps, n);
    MPI_Barrier(comm);

    if (rank == size - 1) {
        bench_timer_stop();
        bench_timer_print();

        //printf("A\n");
        //print_array(n, A);
        printf("B\n");
        print_array(n, B);
    }

    free(A);
    free(B);
}
MPI_Finalize();
return 0;
}

```

## Запуск и проверка программы

```
danila@danila-Lenovo-ideapad-330-15IKB:~/4-course/distributed-systems$ source tut104-sc21/dockervars.sh
Using default tag: latest
latest: Pulling from abouteiller/mpi-ft-ulfm
Digest: sha256:8762045a9a5f35c618f78e4e814c317354a533a7bcf680c9881099eb3a78ce56
Status: Image is up to date for abouteiller/mpi-ft-ulfm:latest
docker.io/abouteiller/mpi-ft-ulfm:latest
# Function alias set for 'make', 'mpirun', 'mpiexec', 'mpicc', 'mpif90'.
source dockervars.sh unload # remove these aliases.
# These commands now run from the ULFM Docker image.
# Use 'mpiexec --with-ft ulfm' to turn ON fault tolerance.
prterun (Open MPI) 5.1.0a1

Report bugs to https://www.open-mpi.org/community/help/
danila@danila-Lenovo-ideapad-330-15IKB:~/4-course/distributed-systems$ mpicc error-handler-jacobi-1d.c -o jacobi-1d
danila@danila-Lenovo-ideapad-330-15IKB:~/4-course/distributed-systems$ mpirun -np 32 --with-ft ulfm --oversubscribe ./
jacobi-1d
-----
WARNING: A deprecated command line option was used.

  Deprecated option:  --oversubscribe
  Corrected option:   --map-by :oversubscribe:OVERSUBSCRIBE

We have updated this for you and will proceed. However, this will be treated
as an error in a future release. Please update your command line.
-----

***** Corrected cmd line: prterun --map-by :oversubscribe:OVERSUBSCRIBE --mca btl tcp,self --np 32 --with-ft ulfm ./
jacobi-1d

n=30 tsteps=20 threads=32
Начато замеряться время
Time in seconds = 1671179085.911191
B
==BEGIN DUMP_ARRAYS==
begin dump: A
0.10 0.11 0.15 0.18 0.21 0.24 0.27 0.30 0.34 0.37 0.40 0.43 0.47 0.50 0.53 0.57 0.60 0.63 0.67 0.70
0.73 0.77 0.80 0.84 0.87 0.91 0.94 0.98 1.01 1.07 ==END DUMP_ARRAYS==
danila@danila-Lenovo-ideapad-330-15IKB:~/4-course/distributed-systems$
```

Как мы видим, в ходе работы программы, все было посчитано корректно и был получен правильный вывод несмотря на то, что некоторые процессы были выведены из строя.

## Временная оценка

Указанные улучшения надежности приводят к следующим негативным с точки зрения производительности эффектам:

- 1) Теперь процессы после каждой итерации будут ждать завершения самого медленного из них
- 2) Тратятся память и время на запись и чтение матриц A и B из файлов
- 3) На перезапуски итераций в случае падения процессов также уходит дополнительное время
- 4) Запасные процессы, по сути, простаивают, в то время как они могли бы выполнять некоторую работу. В частности, мы делаем половину процессов запасными, что, без учета предыдущих пунктов приводит к снижению производительности в 2 раза
- 5) В худшем случае программа будет перезапускаться на каждой итерации, пока не исчерпает свой лимит запасных процессов, после чего все равно упадет.