Московский Государственный Университет имени М. В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

Отчет по практическому заданию по курсу Распределённые системы

Воробьев Евгений
428 группа

Москва
2023

Содержание

1 Задача 1

- Разработать программу которая реализует заданный алгоритм.
- Получить временную оценку работы алгоритма.

1.1 Описание

Все 25 процессов, находящихся на разных ЭВМ сети, одновременно выдали запрос на вход в критическую секцию. Реализовать программу, использующую древовидный маркерный алгоритм для прохождения всеми процессами критических секций. Критическая секция:

```
<проверка наличия файла ‘‘critical.txt’’>;
if (<файл ‘‘critical.txt’’ существует>) {
<сообщение об ошибке>;
<завершение работы программы>;
} else {
<создание файла ‘‘critical.txt’’>;
sleep (<случайное время>);
<уничтожение файла ‘‘critical.txt’’>;
}
```

Для передачи маркера использовать средства MPI. Получить временную оценку работы алгоритма. Оценить сколько времени потребуется, если маркером владеет нулевой процесс. Время старта (время «разгона» после получения доступа к шине для передачи сообщения) равно 100, время передачи байта равно 1 (Ts=100,Tb=1). Процессорные операции, включая чтение из памяти и запись в память, считаются бесконечно быстрыми.

1.2 Реализация

Все процессы представлены в виде сбалансированного двоичного дерева. Каждый процесс имеет очередь запросов от себя и соседних процессов (1-го, 2-х или 3-х) и указатель в направлении владельца маркера. Вход в критическую секцию

- Если есть маркер, то процесс выполняет КС.
- Если нет маркера, то процесс:
    1. помещает свой запрос в очередь запросов;
    2. посылает сообщение «ЗАПРОС» в направлении владельца маркера и ждет сообщений.

Процесс, не находящийся внутри КС должен реагировать на сообщения двух видов -«МАРКЕР» и «ЗАПРОС».

1. Пришло сообщение «МАРКЕР»:

- M1. Взять 1-ый запрос из очереди и послать маркер его автору (концептуально, возможно себе).
- M2. Поменять значение указателя в сторону маркера.
- M3. Исключить запрос из очереди.
- M4. Если в очереди остались запросы, то послать сообщение «ЗАПРОС» в сторону маркера.

2. Пришло сообщение «ЗАПРОС»:

- Поместить запрос в очередь.
- Если нет маркера, то послать сообщение «ЗАПРОС» в сторону маркера, иначе (если есть маркер) - перейти на пункт M1.

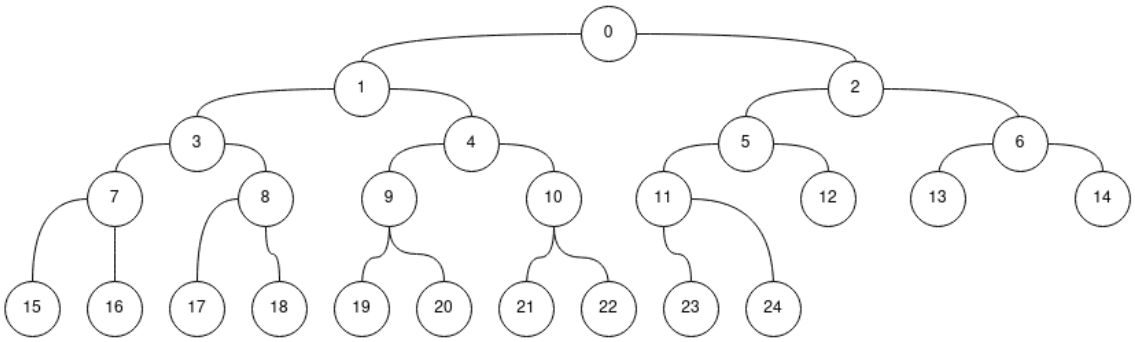

Рис. 1: Схема системы передачи маркера

1.2.1 Запуск

```
mpicc raymond.c -o raymond
mpiexec -np 25 --oversubscribe ./raymond
```

или (безопаснее)

```
mpicc raymond.c -o raymond
mpiexec -np 25 --host=hostname:25 ./raymond
```

1.3 Временная оценка

Сообщений передачи маркера с запросом – 21, маркера без запроса – 24, всего передач маркера – 45.

(Lz - запрос, Lm - маркер; считаем эти сообщения равными 1 байту)

Таким образом, (Ts+Tb*Lz)*21+(Ts+Tb*Lm)*45 = 6666.

2 Задача 2

2.1 Описание

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

2.2 Реализация

Была выбран стратегия продолжения работы на исправных процессах, которые бы продолжили работу.

Был добавлен *error_ handler*, перераспределяющий процессы и переводящий их в точку начала прерванной итерации.

```
static void error_handler(MPI_Comm *comm, int *err, ...) {
    int len;
    char errstr[MPI_MAX_ERROR_STRING];

    rank_to_kill = -200;

    MPIX_Comm_shrink(*comm, &global_comm);

    MPI_Comm_rank(global_comm, &rank);
    MPI_Comm_size(global_comm, &size);

    MPI_Error_string(*err, errstr, &len);
    printf("Rank %d / %d: Notified of error %s\n", rank, size, errstr);

    MPI_Barrier(global_comm);
    eps = 0;
    //adaptive choice of rows(depends on process count)
    fst_r = ((rank * (N-2)) / size) + 1;
    lst_r = ((rank + 1) * (N-2)) / size + 1;
    cnt_r = lst_r - fst_r;
    printf("%d %d %d %d\n", rank, fst_r, lst_r, cnt_r);

    free(A);
    A = calloc((cnt_r + 2) * N2, sizeof(*A));

    longjmp(jbuf, 0);
}
```

Были добавлены *save_ checkpoint* и *load_ checkpoint*, сохраняющие и загружающие текущее состояние данных. Данные хранятся в специальном файле состояния.

```
void save_checkpoint() {
    MPI_File file;
    MPI_File_open(global_comm, file_path, MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_
    for (int i = 1; i <= cnt_r; i++) {
        MPI_File_write_at(file, sizeof(MPI_DOUBLE) * N2 * (fst_r + i), &A[i], N2, MPI_
    }
    MPI_Barrier(global_comm);
    MPI_File_close(&file);
}

void load_checkpoint() {
    free(A);
    A = calloc((cnt_r + 2) * N2, sizeof(*A));
    MPI_File file;
    MPI_File_open(global_comm, file_path, MPI_MODE_RDONLY, MPI_INFO_NULL, &file);
    for (int i = 1; i <= cnt_r; i++) {
        MPI_File_read_at(file, sizeof(MPI_DOUBLE) * N2 * (fst_r + i), &A[i], N2, MPI_D
    }
    MPI_Barrier(global_comm);
    MPI_File_close(&file);
}
```

2.2.1 Запуск

```
mpicc redb_3d_mpi_flthandle.c -o redb_3d_mpi_flthandle
mpiexec -np 4 --with-ft ulfm ./redb_3d_mpi_flthandle
```

3 Ссылки

- https://github.com/user-vo2/Skipod-tasks/tree/main/Skipod2