

Отчет о выполнении второго задания практикума по предмету "Распределенные системы"

Р.М. Куприй, 423 группа

Факультет ВМК МГУ имени М.В. Ломоносова

1. Описание задания

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.

В данной задаче был выбран вариант "а"

2. Выполнение задания

Доработана MPI версия программы 3mm. Реализованы контрольные точки для продолжения работы программы в случае сбоя.

Для усовершенствования программы реализована функция errhandler:

```
1 static void errhandler(MPI_Comm* pcomm, int* perr, ...) {
2     error_occured = 1;
3     int err = *perr;
4     char errstr[MPI_MAX_ERROR_STRING];
5     int size, nf, len;
6     MPI_Group group_f;
7
8     MPI_Comm_size(main_comm, &size);
9     MPIX_Comm_failure_ack(main_comm);
10    MPIX_Comm_failure_get_acked(main_comm, &group_f);
11    MPI_Group_size(group_f, &nf);
12    MPI_Error_string(err, errstr, &len);
13
14    MPIX_Comm_shrink(main_comm, &main_comm);
15    MPI_Comm_rank(main_comm, &broken_rank);
16
17    MPI_Comm_size(main_comm, &world_size);
18 }
```

Изначально получаем код ошибки `err`, создаем переменную `errstr` для хранения полного текстового кода ошибки. Затем, получаем число процессов, определяем область ошибки и область где возникла ошибка. Определяем размер группы `nf` и передаем текст ошибки идентификатору. В конце концов, обновляем общий комму-

никатор, исключая из него умерший процесс и переприсваиваем ранги живым процессам.

Функция обработчик задаётся в main-е.

```
1 MPI_Comm_size(main_comm, &size);
2 MPIX_Comm_failure_ack(main_comm);
3 MPIX_Comm_failure_get_acked(main_comm, &group_f);
4 MPI_Group_size(group_f, &nf);
5 MPI_Error_string(err, errstr, &len);
```

Затем, в код программы добавлена контрольная точка, для того чтобы узнать о прошедшем сбое в результате расчёта. В таком случае полученный результат не является верным, и необходимо провести расчёты заново.

```
1 kernel_3mm(ni, nj, nk, nl, nm, E_local, A_local, B_to_recv, B_to_send,
2           B_sendcounts, F_to_recv, F_to_send, F_sendcounts,
3           C_local,
4           D_to_recv, D_to_send, D_sendcounts, G_local);
5 if (error_occured == 1) {
6     error_occured = 0;
7     goto checkpoint;
8 }
```

Для эмуляции отказа системы уничтожаем один из процессов. Тогда при работе следующей глобальной коммуникации программа обработает эту ошибку.

```
1 if (rank == (world_size - 1)) {
2     printf("process %d has been killed", rank);
3     raise(SIGKILL);
4 }
```

В остальном умножение матриц проходит также. Вычисление выполняется по алгоритму ленточного (построчного) умножения матриц: матрицы разделяются между процессами построчно, затем процессы обмениваются между собой строками матрицы сомножителя. В 3mm алгоритме - этот шаг выполняется трижды. Один шаг этого алгоритма приведен ниже.

```

1  ...
2  int B_row_displs = 0;
3  for (i = 0; i <= rank; i++)
4      B_row_displs += nk / world_size;
5  if (B_row_displs != 0)
6      B_row_displs += nk % world_size;
7
8  for (iter = 0; iter < world_size; iter++) {
9      if (rank != iter)
10         B_rows = nk / world_size;
11     else
12         B_rows = nk / world_size + (nk % world_size);
13
14     B_row_displs = (B_row_displs - B_rows + nk) % nk;
15
16     for (i = 0; i < A_rows; i++)
17         for (j = 0; j < nj; j++) {
18             for (k = 0; k < B_rows; ++k)
19                 E[i * nj + j] += A[i * nk + B_row_displs + k] * B_recv[k * nj
+ j];
20         }
21     float *tmp = B_recv;
22     B_recv = B_send;
23     B_send = tmp;
24
25     MPI_Sendrecv(B_send, B_sendcounts[(world_size + rank - iter) %
world_size],
26                 MPI_FLOAT, (rank + 1) % world_size, 0, B_recv,
27                 B_sendcounts[(world_size + rank - iter - 1) %
world_size],
28                 MPI_FLOAT, (world_size + rank - 1) % world_size, 0,
29                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
30 }
31 ...

```

Рис. 1. Умножение матриц