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:

```
static void errhandler(MPI_Comm* pcomm, int* perr, ...) {
    error_occured = 1;
    int err = *perr;
    char errstr[MPI_MAX_ERROR_STRING];
    int size, nf, len;
    MPI_Group group_f;

    MPI_Comm_size(main_comm, &size);
    MPIX_Comm_failure_ack(main_comm);
    MPIX_Comm_failure_get_acked(main_comm, &group_f);
    MPI_Group_size(group_f, &nf);
    MPI_Error_string(err, errstr, &len);

    MPIX_Comm_shrink(main_comm, &main_comm);
    MPI_Comm_rank(main_comm, &broken_rank);

    MPI_Comm_size(main_comm, &world_size);
}
```

В ней происходит определение отказавшего процесса и задание нового коммуникатора с доступными процессами. Обработчик ошибок определяется в функции main программы.

Затем, в код программы добавлена контрольная точка, для того чтобы узнать о

```
MPI_Comm_size(main_comm, &size);
MPIX_Comm_failure_ack(main_comm);
MPIX_Comm_failure_get_acked(main_comm, &group_f);
MPI_Group_size(group_f, &nf);
MPI_Error_string(err, errstr, &len);
```

прошедшем сбое в результате расчётов. В таком случае поолученный результат не является верным, и необходимо провести расчёты заново.

```
kernel_3mm(ni, nj, nk, nl, nm, E_local, A_local, B_to_recv, B_to_send,
                B_sendcounts, F_to_recv, F_to_send, F_sendcounts,
   C_local,
                D_to_recv, D_to_send, D_sendcounts, G_local);

if (error_occured == 1) {
  error_occured = 0;
  goto checkpoint;
}
```

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.

```
...
  int B_row_displs = 0;
  for (i = 0; i <= rank; i++)
    B_row_displs += nk / world_size;
  if (B_row_displs != 0)
    B_row_displs += nk % world_size;

  for (iter = 0; iter < world_size; iter++) {
    if (rank != iter)
      B_rows = nk / world_size;
    else
      B_rows = nk / world_size + (nk % world_size);

    B_row_displs = (B_row_displs - B_rows + nk) % nk;

    for (i = 0; i < A_rows; i++)
      for (j = 0; j < nj; j++) {
        for (k = 0; k < B_rows; ++k)
          E[i * nj + j] += A[i * nk + B_row_displs + k] * B_recv[k * nj
   + j];
      }
    float *tmp = B_recv;
    B_recv = B_send;
    B_send = tmp;

    MPI_Sendrecv(B_send, B_sendcounts[(world_size + rank - iter) %
   world_size],
                 MPI_FLOAT, (rank + 1) % world_size, 0, B_recv,
                 B_sendcounts[(world_size + rank - iter - 1) %
   world_size],
                 MPI_FLOAT, (world_size + rank - 1) % world_size, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
...
```

Рис. 1. Умножение матриц