Лычева Екатерина, 421 группа

Задача 2

Постановка задачи по курсу «Суперкомпьютеры и параллельная обработка данных»: Даны матрицы A[ni, nk], B[nk, nj], C[nj, nm], D[nm, nl]. Найти матрицу G = ABCD.

ОПИСАНИЕ АЛГОРИТМА РАБОТЫ ПРОГРАММЫ

(реализация изменена в сравнении с прошлым годом, т. к. массивы не были распределены между процессами)

Для реализации вычисления произведения матриц был использован блочный алгоритм. Последовательность умножений выглядит так: A*B = E; E*C = F; F*D = G. Матрицы A, E, F представляются в процессе в виде блоков, содержащих несколько последовательных строк матрицы в пересечении со всеми столбцами этой матрицы. Матрицы B, C, D наоборот — в виде блоков, содержащих несколько последовательных столбцов матрицы в пересечении со всеми её строками. Рассмотрим алгоритм перемножения двух матриц:

|A1| |A1B1 A1B2 A1B3| |E1|

|A2| x |B1 B2 B3| = |A2B1 A2B2 A2B3| = |E2|

|A3| |A3B1 A3B2 A3B3| |E3|

На каждой итерации вычисляется произведение двух блоков, хранящихся в одном процессе (например, A1B1). После того, как все такие произведения вычислены, процессы обмениваются столбцами правой матрицы и вычисляют новый блок той же строки результирующей матрицы. То есть, например, первый процесс вычисляет блоки A1B1, A1B2, A1B3. Обмен происходит пока каждый процесс не поработает с каждым блоком. После этого в каждом процессе оказывается строка результирующей матрицы.

Таким образом произведены все три перемножения матриц.

Функции:

void initialize_matrix(float* matrix, int rows, int cols, int rank, int real);

- задаёт случайные значения для матрицы (в демонстрационной программе в каждом процессе инициализируется сразу блок, хранящийся в нём)

void print_matrix(float* local_matrix, int local_rows, int cols, const char* title, int rank, int size);

- выполняет печать матрицы (в демонстрационной программе печатаются блоки каждого процесса)

int exchange_blocks(float* my_block, int my_block_size, int my_rank, int num_procs);

- производит обмен блоками между процессами по кругу (в процессе умножения матриц пересылаем блоки из столбцов правой матрицы)

void matrix_multiply_block(float* A, float* B, float* C, int rows, int cols, int commonDim, int ni, int y, int realSize) ;

- реализует перемножение двух блоков, результат записывается в соответствующее пространство результирующего блока

int multiply_matr(float* A, float* B, float* C, int rows, int commonDim, int realSize, int ni);

- координирует перемножение двух матриц: умножили блоки; обменялись блоками; умножили; обменялись...

Для обеспечения надёжности был выбран подход «а) продолжить работу программы только на “исправных” процессах».

Решение было реализовано следующим образом:

Написана функция – обработчик ошибок:

```
static void ehandle(MPI_Comm *comm, int *err, ...) {
   int len;
   char errstr[MPI_MAX_ERROR_STRING];
   MPI_Comm_rank(main_comm, &rank);
   MPI_Comm_size(main_comm, &size);
   MPI_Error_string(*err, errstr, &len);
   printf("Rank %d: notified of error %s\n", rank, errstr);
   MPIX_Comm_shrink(main_comm, &main_comm);
   MPI_Comm_rank(main_comm, &rank);
   MPI_Comm_size(main_comm, &size);
   errcode = 1;
}
…
MPI_Errhandler eh;
MPI_Comm_create_errhandler(ehandle, &eh);
MPI_Comm_set_errhandler(main_comm, eh);
```

Для контроля выполнения на экран выводится сообщение от каждого процесса о том, что он получил сигнал об ошибке. После этого коммуникатор заменяется на новый коммуникатор с исключением убитого процесса. Получаем новые данные о задаче и возвращаемся в программу.

Также было добавлено сохранение матриц в файл и чтение из него в случае сбоя:

void write_matrix_to_file(MPI_File file, float* matrix, int rows, int cols, int offset);

- реализует запись блока матрицы в соответствующую часть файла

void read_matrix_from_file(MPI_File file, float* matrix, int rows, int cols, int offset);

- реализует чтение блока матрицы из соответствующей части файла.

В результате, общий алгоритм программы работает так:

1) Вычисляем A*B = E.

2) Если произошла ошибка, возвращаемся и вычисляем заново.

3) Если вычисление успешно, сохраняем матрицу E в файл.

4) Вычисляем E*C = F.

5) Если произошла ошибка, восстанавливаем из файла, записанного в пункте 3, матрицу E, после чего возвращаемся в пункт 4.

6) Если E*C = F вычислено успешно, сохраняем матрицу F в файл.

7) Вычисляем F*D = G.

8) Если произошла ошибка, восстанавливаем из файла, записанного в пункте 6, матрицу F, после чего возвращаемся в пункт 7.

9) Если F*D = G вычислено успешно, то работа программы закончена.

Таким образом, реализована обработка ошибок и восстановление состояния программы из файла. В случае сбоя вычисления, успешно выполненные ранее, не будут производиться повторно.

Конечно, у этого подхода есть свои недостатки – в случае, если отключатся все процессы, работа будет завершена, и задача не будет выполнена. С

другой стороны, не создавая новые процессы, мы исключаем возможность возникновения проблем при создании нового процесса и нам не требуется выделять программе больше ресурсов, чем могло требоваться изначально (например, если бы мы заранее создавали дополнительные процессы, то в отсутствие сбоя мы бы заняли лишние ресурсы на процессы, которые не понадобились)