Ленточные алгоритмы умножения матриц

В данных алгоритмах матрицы разбиваются на непрерывные последовательности строк или столбцов (*полосы*). В простейшем случае полосой может служить отдельная строка или столбец.

В рассматриваемых ниже алгоритмах каждый процесс используется для вычисления одной строки результирующего произведения матриц *AB*. В этом случае процесс должен иметь доступ к соответствующей строке матрицы *A* и всей матрице *B*. Поскольку одновременное хранение всей матрицы *B* во всех процессах параллельного приложения требует чрезмерных затрат памяти, вычисления организуются таким образом, чтобы в каждый момент времени процессы содержали лишь часть элементов матрицы *B* (один столбец или одну строку), а доступ к остальной части обеспечивался бы при помощи передачи сообщений.

При описании ленточных алгоритмов предполагается, что количество процессов *N* совпадает с порядком перемножаемых матриц *A* и *B*, а матрицы являются квадратными. В случае, когда порядок матриц *M* является кратным количеству процессов *N*, достаточно обрабатывать в каждом процессе *полосы*, содержащие *M* / *N* строк или столбцов.

Ленточный алгоритм 1 (горизонтальные полосы)

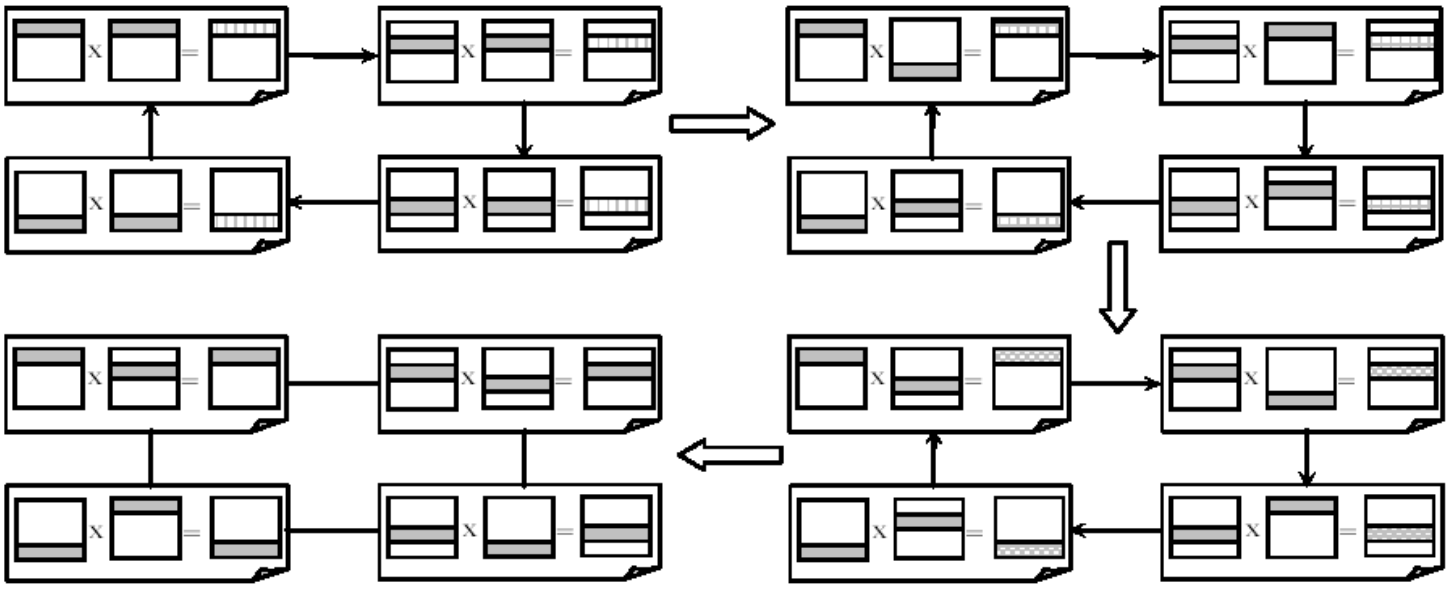
Вначале производится рассылка в процесс ранга *K* элементов *K*-й строки матрицы *A* и элементов *K*-й строки матрицы *B*. Элементы строки *c*, в которой будет содержаться соответствующая строка произведения *AB* результат, обнуляются.

Затем запускается цикл (число итераций равно *N*), в ходе которого выполняются два действия:

1) выполняется перемножение элементов строк матрицы *A* и матрицы *B* с одинаковыми номерами, и результаты добавляются к соответствующему элементу строки *c*;
2) выполняется циклическая пересылка строк матрицы *B* в соседние процессы (направление пересылки может быть произвольным: как по возрастанию рангов процессов, так и по их убыванию).

После завершения цикла в каждом процессе будет содержаться соответствующая строка произведения *AB*. Останется переслать эти строки главному процессу.

На рисунке приведена схема алгоритма 1 при условии, что циклическая пересылка строк матрицы *B* выполняется в направлении убывания рангов процессов.



Ленточный алгоритм 2 (горизонтальные и вертикальные полосы)

Вначале производится рассылка в процесс ранга *K* элементов *K*-й строки матрицы *A* и элементов *K*-го столбца матрицы *B*.

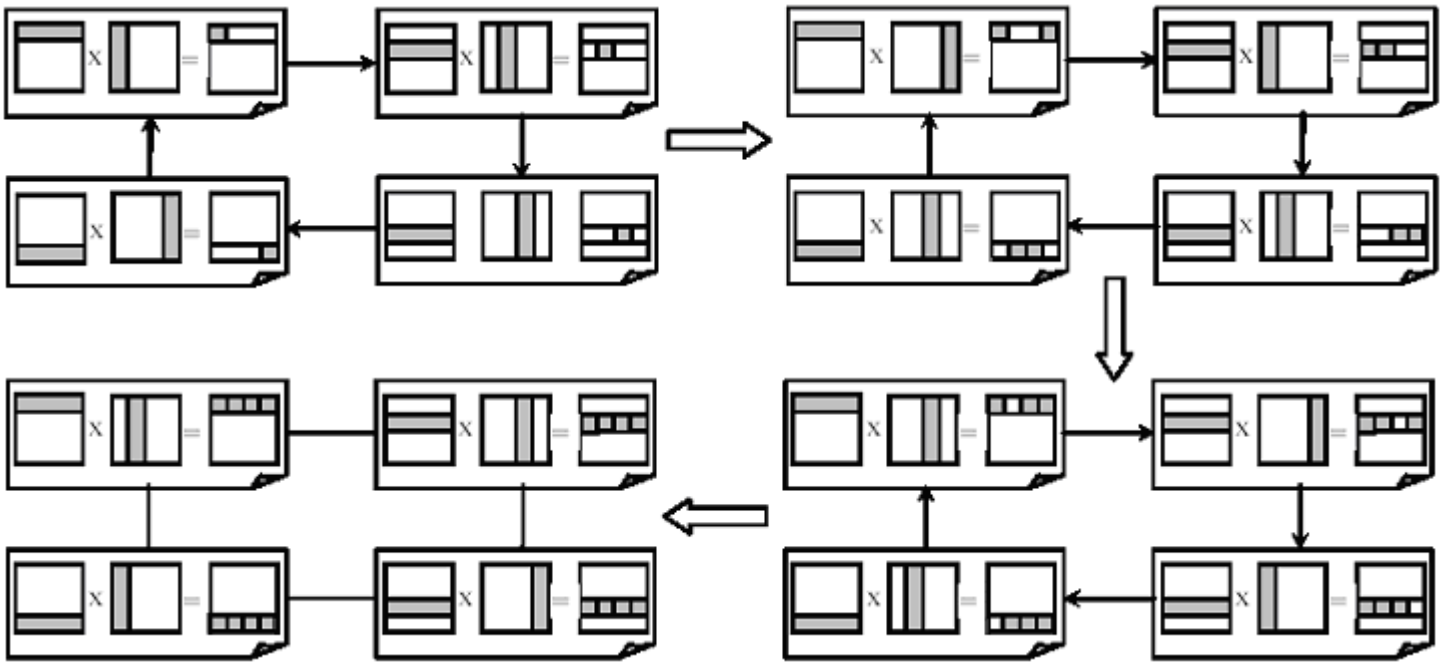
Затем запускается цикл (число итераций равно *N*), в ходе которого выполняются два действия:

1) выполняется перемножение строки матрицы *A* и столбца матрицы *B*, содержащихся в данном процессе, и результат записывается в соответствующий элемент строки *c*;

2) выполняется циклическая пересылка столбцов матрицы B в соседние процессы (направление пересылки может быть произвольным: как по возрастанию рангов процессов, так и по их убыванию).

После завершения цикла в каждом процессе будет содержаться строка c, равная одной из строк произведения AB. Останется переслать строки c главному процессу.

На рисунке приведена схема алгоритма 1 при условии, что циклическая пересылка столбцов матрицы B выполняется в направлении убывания рангов процессов.



Блочные алгоритмы умножения матриц

В данных алгоритмах матрицы разбиваются на блоки, представляющие собой подматрицы исходных матриц. Для простоты будем предполагать, что все матрицы являются квадратными размера $N \times N$, а количество блоков по горизонтали и по вертикали является одинаковым и равным q (при этом размер всех блоков равен $K \times K$, где $K = N / q$). В этом случае операция матричного умножения может быть представлена в блочном виде:

$$\begin{pmatrix} A_{00}A_{01}...A_{0q-1} \\ \cdots \\ A_{q-10}A_{q-11}...A_{q-1q-1} \end{pmatrix} \times \begin{pmatrix} B_{00}B_{01}...B_{0q-1} \\ \cdots \\ B_{q-10}B_{q-11}...B_{q-1q-1} \end{pmatrix} = \begin{pmatrix} C_{00}C_{01}...C_{0q-1} \\ \cdots \\ c_{q-10}C_{q-11}...C_{q-1q-1} \end{pmatrix}$$

При этом каждый блок C_{ij} матрицы C определяется как произведение соответствующих блоков матриц A и B:

$$C_{ij} = \sum_{s=0}^{q-1} A_{is} B_{sj}$$

При блочном разбиении данных естественно связать с каждым процессом задачу вычисления одного из блоков результирующей матрицы C. В этом случае процессу должны быть доступны все элементы соответствующих строк матрицы A и столбцов матрицы B. Поскольку размещение всех требуемых данных в каждом процессе приведет к их дублированию и существенному увеличению объема используемой памяти, необходимо организовать вычисления таким образом, чтобы в каждый момент времени процессы содержали лишь по одному блоку матриц A и B, требуемому для расчетов, а доступ к остальным блокам обеспечивался бы при помощи передачи сообщений.

В данных алгоритмах для процессов удобно ввести двумерную декартову топологию, сопоставив каждому процессу его координаты (i, j) в этой топологии ($i, j = 0, \ldots, q - 1$). При этом предполагается, что количество процессов равно q^2.

Блочный алгоритм Фокса

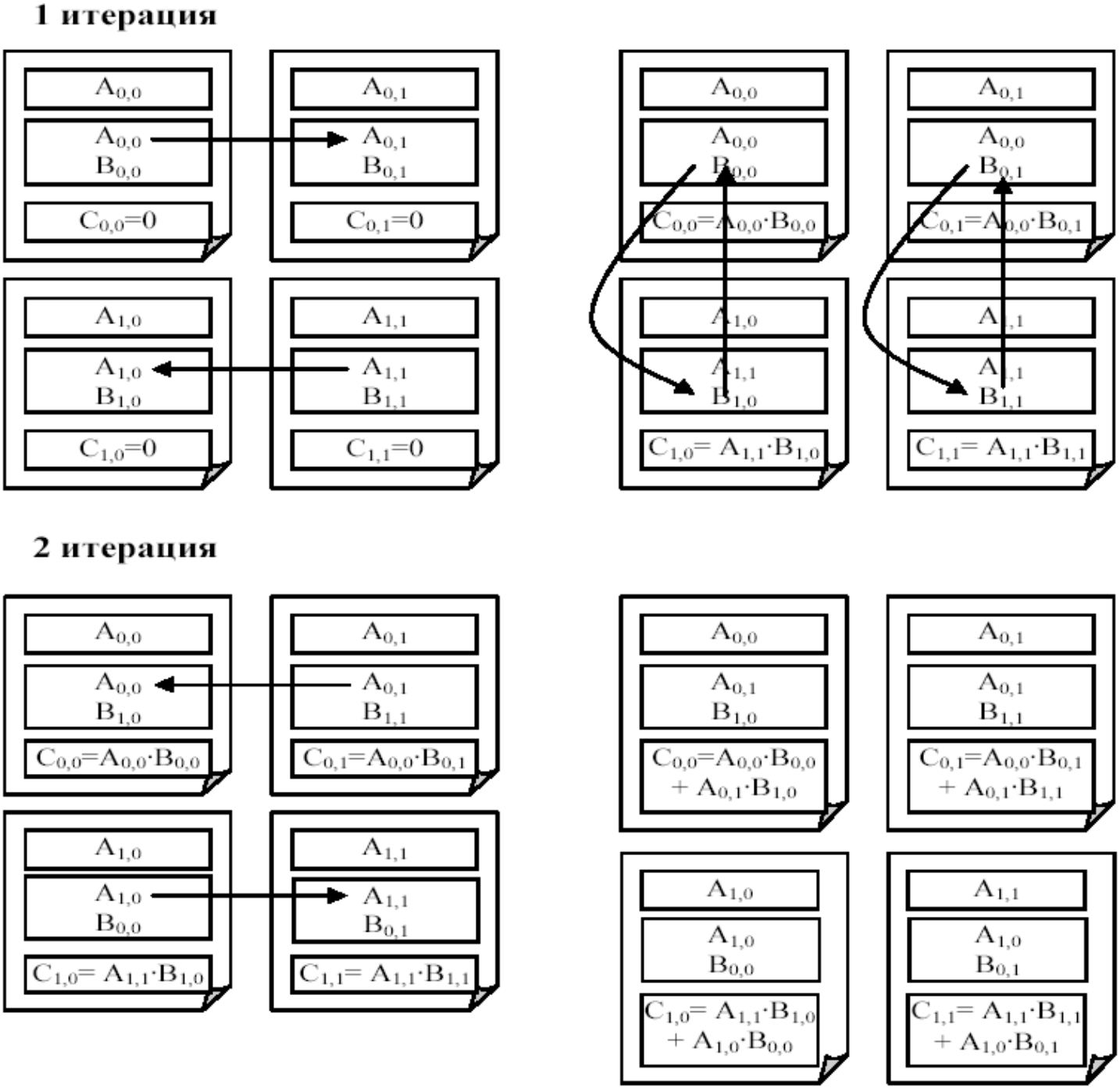
Вначале производится рассылка в процесс с координатами (i, j) блоков A_{ij}, B_{ij} исходных матриц. Кроме того, выполняется обнуление матрицы C_{ij}, предназначенной для хранения соответствующего блока результирующего произведения AB.

Затем запускается цикл по m ($m = 0, \ldots, q-1$), в ходе которого выполняются три действия:

1) для каждой строки i ($i = 0, \ldots, q-1$) блок A_{ij} одного из процессов пересылается во все процессы этой же строки; при этом индекс j пересылаемого блока определяется по формуле $j = (i + m) \bmod q$;
2) полученный в результате подобной пересылки блок матрицы A и содержащийся в процессе (i, j) блок матрицы B перемножаются, и результат прибавляется к матрице C_{ij};
3) для каждого столбца j ($j = 0, \ldots, q-1$) выполняется циклическая пересылка блоков матрицы B, содержащихся в каждом процессе (i, j) этого столбца, в направлении убывания номеров строк.

После завершения цикла в каждом процессе будет содержаться матрица C_{ij}, равная соответствующему блоку произведения AB. Останется переслать эти блоки главному процессу.

На рисунке приведена схема алгоритма Фокса в случае $q = 2$.

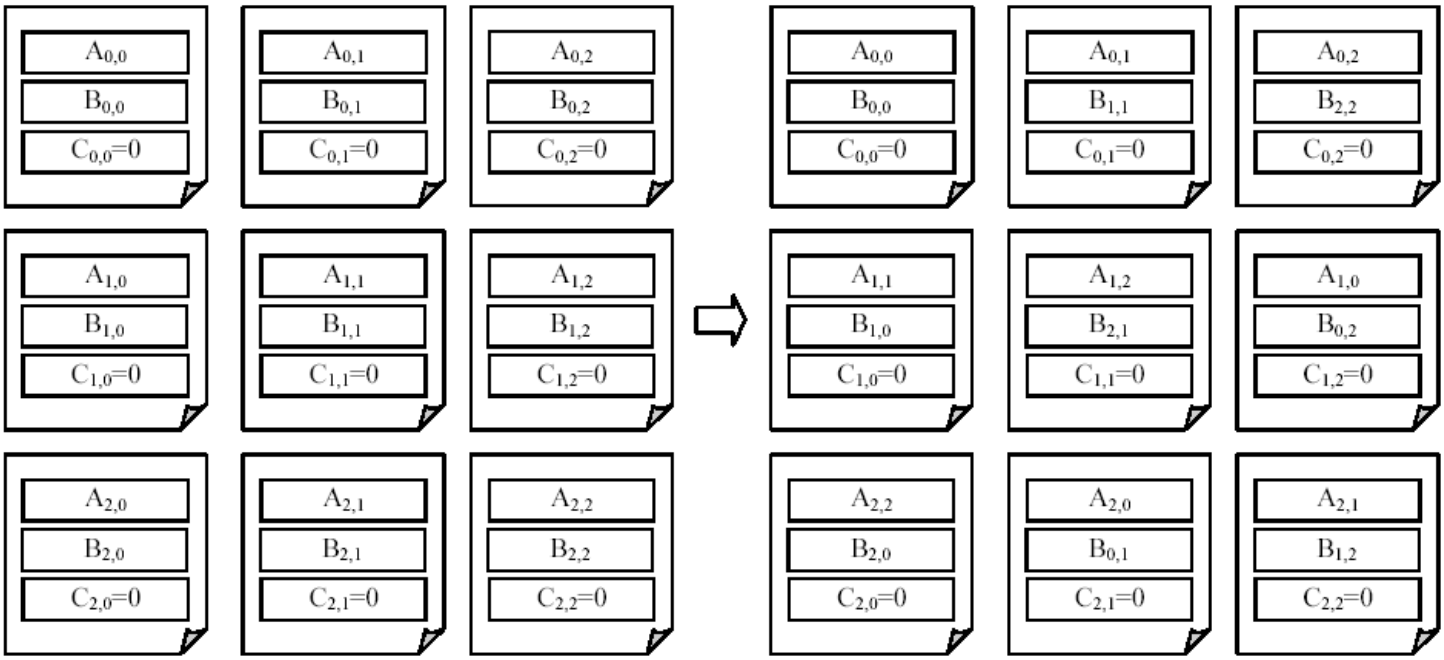


Блочный алгоритм Кэннона

Алгоритм Кэннона отличается от алгоритма Фокса двумя аспектами. Во-первых, начальная пересылка блоков матриц A и B в процессы выполняется таким образом, чтобы получаемые блоки сразу могли быть перемножены без каких-либо пересылок данных. Во-вторых, при организации цикла выполняется циклическая пересылка не только блоков матрицы B (по столбцам), но и блоков матрицы A (по строкам). Действия по начальной пересылке состоят из следующих шагов:

1) в каждый процесс (i, j) пересылаются блоки A_{ij} и B_{ij}, матрица C_{ij} обнуляется;
2) для каждой строки i ($i = 0, \ldots, q-1$) декартовой решетки процессов выполняется циклический сдвиг блоков матрицы A на i позиций влево (т. е. в направлении убывания номеров столбцов);
3) для каждого столбца j ($j = 0, \ldots, q-1$) декартовой решетки процессов выполняется циклический сдвиг блоков матрицы B на j позиций вверх (т. е. в направлении убывания номеров строк).

Результат подобного перераспределения блоков в случае $q = 2$ приведен на рисунке.



Затем запускается цикл из q итераций, в ходе которого выполняются три действия:

1) содержащиеся в процессе (i, j) блоки матриц A и B перемножаются, и результат прибавляется к матрице C_{ij};
2) для каждой строки i ($i = 0, \ldots, q-1$) выполняется циклическая пересылка блоков матрицы A, содержащихся в каждом процессе (i, j) этой строки, в направлении убывания номеров столбцов;
3) для каждого столбца j ($j = 0, \ldots, q-1$) выполняется циклическая пересылка блоков матрицы B, содержащихся в каждом процессе (i, j) этого столбца, в направлении убывания номеров строк.

После завершения цикла в каждом процессе будет содержаться матрица C_{ij}, равная соответствующему блоку произведения AB. Останется переслать эти блоки главному процессу.

Задания по теме «Параллельные матричные алгоритмы»

Все числа в заданиях являются целыми. Матрицы должны вводиться и выводиться по строкам. Файлы с элементами матриц также содержат их по строкам.

Количество процессов в заданиях, связанных с ленточными алгоритмами (MPI9Matr2–MPI9Matr20), не превосходит 5, а в заданиях, связанных с блочными алгоритмами (MPI9Matr21–MPI9Matr44), не превосходит 16.

Для хранения имени файла достаточно использовать массив char[12], а для его пересылки из главного процесса в подчиненные — функцию MPI_Bcast с параметром типа MPI_CHAR.

В заготовках для каждого задания уже содержатся описания целочисленных переменных для хранения числовых данных, упоминаемых в заданиях (в частности, размеров матриц), указателей на массивы для хранения самих матриц, а также переменных типа MPI_Datatype и MPI_Comm. Эти переменные следует использовать во всех функциях, которые требуется реализовать при выполнении заданий. Все имена переменных соответствуют обозначениям, применяемым в формулировках заданий. Для массивов, связанных с полосами или блоками матриц, используются имена a, b, c, t; для массивов, связанных с исходными матрицами *A*, *B* и их результирующим произведением *C*, используются имена с подчеркиванием: a_, b_, c_.

Задания, в которых используется файловый ввод-вывод (MPI9Matr8–MPI9Matr10, MPI9Matr18–MPI9Matr20, MPI9Matr29–MPI9Matr31, MPI9Matr42–MPI9Matr44), требуют подключения библиотеки MPI-2 (система MPICH2 1.3). Для выполнения остальных заданий данной группы можно использовать любую версию MPI (MPICH 1.3.5 или MPICH2 1.3).

Перед выполнением заданий из данного пункта следует ознакомиться с примерами решения задач MPI9Matr1, MPI9Matr2, MPI9Matr24, MPI9Matr19, приведенными в учебнике (п. 1.4.2–1.4.5).

Непараллельный алгоритм умножения матриц

MPI9Matr1. В главном процессе даны числа *M*, *P*, *Q* и матрицы *A* и *B* размера $M \times P$ и $P \times Q$ соответственно. Найти и вывести в главном процессе матрицу *C* размера $M \times Q$, являющуюся произведением матриц *A* и *B*.

Формула для вычисления элементов матрицы *C* в предположении, что строки и столбцы всех матриц нумеруются от 0, имеет вид:

$C_{I,J} = A_{I,0} \cdot B_{0,J} + A_{I,1} \cdot B_{1,J} + \ldots + A_{I,P-1} \cdot B_{P-1,J}$, где $I = 0, \ldots, M - 1$, $J = 0, \ldots, Q - 1$.

Для хранения матриц *A*, *B*, *C* использовать одномерные массивы размера $M \cdot P$, $P \cdot Q$ и $M \cdot Q$, размещая в них элементы матриц по строкам (при этом элемент матрицы с индексами *I* и *J* будет храниться в элементе соответствующего массива с индексом $I \cdot N + J$, где *N* —количество столбцов матрицы). При выполнении данного задания подчиненные процессы не используются.

Примечание. Решение данной задачи приведено в п. 1.4.2.

Ленточный алгоритм 1 (горизонтальные полосы)

MPI9Matr2*°. В главном процессе даны числа *M*, *P*, *Q* и матрицы *A* и *B* размера $M \times P$ и $P \times Q$ соответственно. В первом варианте ленточного алгоритма перемножения матриц каждая матрица-сомножитель разбивается на *K* горизонтальных полос, где *K* —

количество процессов (в дальнейшем полосы распределяются по процессам и используются для вычисления в каждом процессе части итогового матричного произведения).

Полоса для матрицы A содержит N_A строк, полоса для матрицы B содержит N_B строк; числа N_A и N_B вычисляются по формулам $N_A = \mathrm{ceil}(M/K)$, $N_B = \mathrm{ceil}(P/K)$, где операция «/» означает вещественное деление, а функция ceil выполняет округление с избытком. Если матрица содержит недостаточно строк для заполнения последней полосы, то полоса дополняется нулевыми строками.

Сохранить исходные матрицы, дополненные при необходимости нулевыми строками, в одномерных массивах в главном процессе, после чего организовать пересылку полос из этих массивов во все процессы: в процесс ранга R ($R = 0, 1, \ldots, K - 1$) пересылается полоса с индексом R, все полосы A_R имеют размер $N_A \times P$, все полосы B_R имеют размер $N_B \times Q$. Кроме того, создать в каждом процессе полосу C_R для хранения фрагмента матричного произведения $C = AB$, которое будет вычисляться в этом процессе; каждая полоса C_R имеет размер $N_A \times Q$ и заполняется нулевыми элементами.

Полосы, как и исходные матрицы, должны храниться по строкам в одномерных массивах соответствующего размера. Для пересылки размеров матриц использовать коллективную функцию MPI_Bcast, для пересылки полос матриц A и B использовать коллективную функцию MPI_Scatter.

Оформить все описанные действия в виде функции Matr1ScatterData (без параметров), в результате вызова которой каждый процесс получает значения N_A, P, N_B, Q, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C. После вызова функции Matr1ScatterData вывести в каждом процессе полученные данные (числа N_A, P, N_B, Q и полосы матриц A, B, C). Ввод исходных данных осуществлять в функции Matr1ScatterData, вывод результатов выполнять во внешней функции Solve.

Указание. Для уменьшения числа вызовов функции MPI_Bcast все пересылаемые размеры матриц можно поместить во вспомогательный массив.

Примечание. Решение данной задачи приведено в п. 1.4.3.

MPI9Matr3. В каждом процессе даны числа N_A, P, N_B, Q, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr2). Реализовать первый шаг ленточного алгоритма перемножения матриц, выполнив перемножение элементов, содержащихся в полосах A_R и B_R каждого процесса, после чего организовать циклическую пересылку каждой полосы B_R в процесс предыдущего ранга (из процесса 1 в процесс 0, из процесса 2 в процесс 1, …, из процесса 0 в процесс $K - 1$, где K — количество процессов).

Циклическую пересылку полос B_R выполнять с помощью функции MPI_Sendrecv_replace, используя для определения рангов процесса-отправителя и процесса-получателя операцию % взятия остатка от деления.

Оформить эти действия в виде функции Matr1Calc (без параметров). Вывести новое содержимое полос C_R и B_R в каждом процессе (ввод и вывод данных выполнять во внешней функции Solve).

Указание. В результате перемножения полос A_R и B_R каждый элемент полосы C_R будет содержать *часть* слагаемых, входящих в итоговое произведение AB; при этом будут использованы все элементы полосы B_R и часть элементов полосы A_R (в частности,

на первом шаге в процессе 0 будут использованы N_B первых столбцов полосы A_0, а в процессе $K-1$ — N_B последних столбцов полосы A_{K-1}).

MPI9Matr4. В каждом процессе даны числа N_A, P, N_B, Q, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr2). Модифицировать функцию Matr1Calc, реализованную в предыдущем задании, таким образом, чтобы она обеспечивала выполнение любого шага алгоритма ленточного умножения.

Для этого добавить к ней параметр step, определяющий номер шага (изменяется от 0 до $K-1$, где K — количество процессов), и использовать значение этого параметра в той части алгоритма, которая связана с пересчетом элементов полосы C_R (действия, связанные с циклической пересылкой полос B_R, от значения параметра step не зависят).

Используя два вызова модифицированной функции Matr1Calc с параметрами 0 и 1, выполнить два начальных шага ленточного алгоритма и вывести в каждом процессе новое содержимое полос C_R и B_R (ввод и вывод данных выполнять во внешней функции Solve).

Указание. Номер шага step определяет, какая часть элементов полосы A_R будет использована при очередном пересчете элементов полосы C_R (следует обратить внимание на то, что эти части перебираются циклически).

MPI9Matr5*. В каждом процессе даны числа N_A, P, N_B, Q, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr2). Кроме того, в каждом процессе дано одно и то же число L, лежащее в диапазоне от 3 до K (K — количество процессов) и определяющее требуемое число шагов ленточного алгоритма.

Вызывая в цикле по параметру I ($I = 0, \ldots, L-1$) функцию Matr1Calc(I), разработанную в предыдущем задании, выполнить L начальных шагов ленточного алгоритма и вывести в каждом процессе новое содержимое полос C_R и B_R (ввод и вывод данных выполнять во внешней функции Solve).

Примечание. Если значение L равно K, то полосы C_R будут содержать соответствующие фрагменты итогового матричного произведения AB.

MPI9Matr6*. В главном процессе дано число M — количество строк результирующего матричного произведения. Кроме того, в каждом процессе даны числа N_A, Q, а также одномерные массивы, заполненные полосами матрицы C (размера $N_A \times Q$), которые были получены в результате выполнения K шагов ленточного алгоритма перемножения матриц (см. MPI9Matr5). Переслать все полосы C_R в главный процесс и вывести в нем полученную матрицу C (размера $M \times Q$).

Для хранения результирующей матрицы C в главном процессе использовать одномерный массив, достаточный для хранения матрицы размера $(N_A \cdot K) \times Q$; для пересылки данных в этот массив использовать коллективную функцию MPI_Gather.

Оформить эти действия в виде функции Matr1GatherData (без параметров). Ввод данных выполнять во внешней функции Solve, вывод полученной матрицы включить в функцию Matr1GatherData.

MPI9Matr7**. В главном процессе даны числа M, P, Q и матрицы A и B размера $M \times P$ и $P \times Q$ соответственно (таким образом, исходные данные совпадают с исходными данными для задания MPI9Matr2).

Последовательно вызывая разработанные в заданиях MPI9Matr2–MPI9Matr6 функции Matr1ScatterData, Matr1Calc (в цикле) и Matr1GatherData, получить и вывести в главном процессе матрицу C, равную произведению исходных матриц A и B.

После каждого вызова функции Matr1Calc дополнительно выводить в каждом процессе текущее содержимое полосы C_R. Перед использованием в данном задании следует модифицировать разработанную в MPI9Matr4 функцию Matr1Calc таким образом, чтобы при значении параметра step, равном $K - 1$, не выполнялась пересылка полос B_R.

MPI9Matr8*. В главном процессе даны числа M, P, Q, а также имена двух файлов, содержащих элементы матриц A и B размера $M \times P$ и $P \times Q$ соответственно. Модифицировать начальный этап ленточного алгоритма перемножения матриц (см. MPI9Matr2) таким образом, чтобы каждый процесс считывал соответствующие полосы матриц A и B непосредственно из исходных файлов, используя коллективные функции MPI_File_seek и MPI_File_read_all (новый вид файловых данных создавать не требуется).

Для пересылки размеров матриц и имен файлов использовать коллективную функцию MPI_Bcast.

Оформить все действия в виде функции Matr1ScatterFile (без параметров), в результате вызова которой каждый процесс получает значения N_A, P, N_B, Q, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C. После вызова функции Matr1ScatterFile вывести в каждом процессе полученные данные (числа N_A, P, N_B, Q и полосы матриц A, B, C). Ввод исходных данных осуществлять в функции Matr1ScatterFile, вывод результатов выполнять во внешней функции Solve.

Примечание. Для некоторых полос часть элементов (последние строки) или даже вся полоса не должна считываться из исходных файлов и будет оставаться нулевой. Однако эта ситуация не требует специальной обработки, так как при достижении конца файла функция MPI_File_read_all автоматически прекращает считывание данных, не генерируя никакого сообщения об ошибке.

MPI9Matr9*. В каждом процессе даны числа N_A, Q, а также одномерные массивы, заполненные полосами C_R (размера $N_A \times Q$), полученными в результате выполнения K шагов ленточного алгоритма перемножения матриц (см. MPI9Matr5). Кроме того, в главном процессе дано число M — количество строк результирующего матричного произведения и имя файла для хранения этого произведения.

Переслать число M и имя файла во все процессы (используя функцию MPI_Bcast) и записать все фрагменты матричного произведения, содержащиеся в полосах C_R, в результирующий файл, который в итоге будет содержать матрицу C размера $M \times Q$.

Для записи полос в файл использовать коллективные функции MPI_File_seek и MPI_File_write_all.

Оформить считывание имени файла, пересылку значения M и имени файла, а также все действия по записи полос в файл в виде функции Matr1GatherFile (считывание всех исходных данных, кроме имени файла, должно осуществляться во внешней функции Solve).

Указание. При записи данных в результирующий файл необходимо учитывать, что некоторые полосы C_R могут содержать завершающие нулевые строки, не связанные с полученным произведением (именно для проверки этой ситуации требуется пересылать во все процессы значение M).

MPI9Matr10**. В главном процессе даны числа *M, P, Q*, а также имена трех файлов: вначале даются имена двух существующих файлов, содержащих элементы матриц *A* и *B* размера $M \times P$ и $P \times Q$ соответственно, а затем имя файла для хранения результирующего матричного произведения $C = AB$.

Последовательно вызывая разработанные в заданиях MPI9Matr8, MPI9Matr5 и MPI9Matr9 функции Matr1ScatterFile, Matr1Calc (в цикле) и Matr1GatherFile, получить в результирующем файле произведение исходных матриц *A* и *B*, найденное с помощью первого варианта ленточного алгоритма.

После каждого вызова функции Matr1Calc дополнительно выводить в каждом процессе текущее значение элемента c[step], где c — одномерный массив, содержащий полосу C_R, а step — номер шага алгоритма (0, 1, …, $K - 1$); таким образом, на первом шаге алгоритма следует вывести элемент c[0], на втором шаге — элемент c[1], и т. д.

Ленточный алгоритм 2 (горизонтальные и вертикальные полосы)

MPI9Matr11. В каждом процессе даны числа *P* и *Q*; кроме того, в главном процессе дана матрица *B* размера $P \times Q$. Известно, что число *Q* кратно количеству процессов *K*. Прочесть в главном процессе матрицу *B* в одномерный массив размера $P \cdot Q$ и определить новый тип MPI_BAND_B, содержащий вертикальную полосу матрицы *B* шириной $N_B = Q/K$ столбцов.

При определении типа MPI_BAND_B использовать функции MPI_Type_vector и MPI_Type_commit, оформив это определение в виде функции Matr2CreateTypeBand(p, n, q, t), где целочисленные параметры p, n, q являются входными, а параметр t типа MPI_Datatype является выходным; при этом параметры p и n определяют размер вертикальной полосы (число ее строк и столбцов), а параметр q — число столбцов матрицы, из которой извлекается эта полоса.

Используя тип MPI_BAND_B, переслать в каждый процесс (включая главный) соответствующую полосу матрицы *B*, перебирая полосы в порядке возрастания рангов процессов-получателей. Пересылку выполнять с помощью функций MPI_Send и MPI_Recv; полосы хранить в одномерных массивах размера $P \cdot N_B$. Вывести в каждом процессе полученную полосу.

Примечание. В реализации MPICH2 версии 1.3 с помощью функции MPI_Send нельзя выполнить пересылку данных в тот же процесс, из которого данные посылаются (происходит зависание программы). Для пересылки полосы в главный процесс можно использовать функцию MPI_Sendrecv; можно также заполнить соответствующую полосу в главном процессе, не прибегая к средствам библиотеки MPI.

MPI9Matr12*. В главном процессе даны числа *M, P, Q* и матрицы *A* и *B* размера $M \times P$ и $P \times Q$ соответственно. Во втором варианте ленточного алгоритма перемножения матриц первая матрица (*A*) разбивается на *K* горизонтальных полос, а вторая (*B*) — на *K* вертикальных полос, где *K* — количество процессов (в дальнейшем полосы распределяются по процессам и используются для вычисления в каждом процессе части итогового матричного произведения).

Полоса для матрицы *A* содержит N_A строк, полоса для матрицы *B* содержит N_B столбцов; числа N_A и N_B вычисляются по формулам $N_A = \text{ceil}(M/K)$, $N_B = \text{ceil}(Q/K)$, где операция «/» означает вещественное деление, а функция ceil выполняет округление с избытком. Если матрица содержит недостаточно строк (или столбцов) для заполнения последней полосы, то полоса дополняется нулевыми строками (столбцами).

Сохранить исходные матрицы, дополненные при необходимости нулевыми строками или столбцами, в одномерных массивах в главном процессе, после чего организовать пересылку полос из этих массивов во все процессы: в процесс ранга R ($R = 0, 1, \ldots, K - 1$) пересылается полоса с индексом R, все полосы A_R имеют размер $N_A \times P$, все полосы B_R имеют размер $P \times N_B$. Кроме того, создать в каждом процессе полосу C_R для хранения фрагмента матричного произведения $C = AB$, который будет вычисляться в этом процессе; каждая полоса C_R имеет размер $(N_A \cdot K) \times N_B$ и заполняется нулевыми элементами.

Полосы, как и исходные матрицы, должны храниться по строкам в одномерных массивах соответствующего размера. Для пересылки размеров матриц использовать коллективную функцию MPI_Bcast, для пересылки полос матрицы A использовать коллективную функцию MPI_Scatter, для пересылки полос матрицы B использовать функции MPI_Send и MPI_Recv, а также вспомогательный тип MPI_BAND_B, созданный с помощью функции Matr2CreateTypeBand (см. предыдущее задание и примечание к нему).

Оформить все описанные действия в виде функции Matr2ScatterData (без параметров), в результате вызова которой каждый процесс получает значения N_A, P, N_B, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C. После вызова функции Matr2ScatterData вывести в каждом процессе полученные данные (числа N_A, P, N_B и полосы матриц A, B, C). Ввод исходных данных осуществлять в функции Matr2ScatterData, вывод результатов выполнять во внешней функции Solve.

Указания. (1) При считывании матрицы B в главном процессе следует учитывать, что предназначенный для ее хранения массив может содержать элементы, соответствующие дополнительным нулевым столбцам.

(2) Для уменьшения числа вызовов функции MPI_Bcast все пересылаемые размеры матриц можно поместить во вспомогательный массив.

MPI9Matr13. В каждом процессе даны числа N_A, P, N_B, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr12). Реализовать первый шаг ленточного алгоритма перемножения матриц, выполнив перемножение элементов, содержащихся в полосах A_R и B_R каждого процесса, после чего организовать циклическую пересылку каждой полосы A_R в процесс предыдущего ранга (из процесса 1 в процесс 0, из процесса 2 в процесс 1, …, из процесса 0 в процесс $K - 1$, где K — количество процессов).

Циклическую пересылку полос A_R выполнять с помощью функции MPI_Sendrecv_replace, используя для определения рангов процесса-отправителя и процесса-получателя операцию % взятия остатка от деления.

Оформить эти действия в виде функции Matr2Calc (без параметров). Вывести новое содержимое полос C_R и A_R в каждом процессе (ввод и вывод данных выполнять во внешней функции Solve).

Указание. Поскольку в данном варианте ленточного алгоритма полосы A_R содержат полные строки матрицы A, а полосы B_R — полные столбцы матрицы B, в результате их перемножения уже на первом шаге алгоритма полоса C_R будет содержать часть элементов итогового матричного произведения (прочие элементы полосы останутся нулевыми). Расположение найденных элементов в полосе C_R зависит от ранга процесса (в частности, на первом шаге в процессе 0 будут заполнены N_A первых строк полосы C_0, а в процессе $K - 1$ — N_A последних строк полосы C_{K-1}).

MPI9Matr14. В каждом процессе даны числа N_A, P, N_B, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr12). Модифицировать функцию Matr2Calc, реализованную в предыдущем задании, таким образом, чтобы она обеспечивала выполнение любого шага алгоритма ленточного умножения.

Для этого добавить к ней параметр step, определяющий номер шага (изменяется от 0 до $K - 1$, где K — количество процессов) и использовать значение этого параметра в той части алгоритма, которая связана с пересчетом элементов полосы C_R (действия, связанные с циклической пересылкой полос A_R, от значения параметра step не зависят).

Используя два вызова модифицированной функции Matr2Calc с параметрами 0 и 1, выполнить два начальных шага ленточного алгоритма и вывести в каждом процессе новое содержимое полос C_R и A_R (ввод и вывод данных выполнять во внешней функции Solve).

Указание. Номер шага step определяет, какие наборы строк полосы C_R будут вычислены на данном этапе алгоритма (следует обратить внимание на то, что эти наборы строк перебираются циклически).

MPI9Matr15*. В каждом процессе даны числа N_A, P, N_B, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr12). Кроме того, в каждом процессе дано одно и то же число L, лежащее в диапазоне от 3 до K (K — количество процессов) и определяющее требуемое число шагов ленточного алгоритма.

Вызывая в цикле по параметру I ($I = 0, \ldots, L - 1$) функцию Matr2Calc(I), разработанную в предыдущем задании, выполнить L начальных шагов ленточного алгоритма и вывести в каждом процессе новое содержимое полос C_R и A_R (ввод и вывод данных выполнять во внешней функции Solve).

Примечание. Если значение L равно K, то полосы C_R будут содержать соответствующие фрагменты итогового матричного произведения AB.

MPI9Matr16*. В главном процессе даны числа M и Q — количество строк и столбцов результирующего матричного произведения. Кроме того, в каждом процессе даны числа N_A, N_B, а также одномерные массивы, заполненные полосами матрицы C (размера $(N_A \cdot K) \times N_B$), которые были получены в результате выполнения K шагов ленточного алгоритма перемножения матриц (см. MPI9Matr15). Переслать все полосы C_R в главный процесс и вывести в нем полученную матрицу C (размера $M \times Q$).

Для хранения результирующей матрицы C в главном процессе использовать одномерный массив, достаточный для хранения матрицы размера $(N_A \cdot K) \times (N_B \cdot K)$. Для пересылки полос C_R использовать функции MPI_Send и MPI_Recv, а также вспомогательный тип MPI_BAND_C, созданный с помощью функции Matr2CreateTypeBand (см. задание MPI9Matr11 и примечание к нему).

Оформить эти действия в виде функции Matr2GatherData (без параметров). Ввод данных выполнять во внешней функции Solve, вывод полученной матрицы включить в функцию Matr2GatherData.

Указание. При выводе матрицы C в главном процессе следует учитывать, что предназначенный для ее хранения массив может содержать элементы, соответствующие дополнительным нулевым столбцам.

MPI9Matr17**. В главном процессе даны числа M, P, Q и матрицы A и B размера $M \times P$ и $P \times Q$ соответственно (таким образом, исходные данные совпадают с исходными данными для задания MPI9Matr12).

Последовательно вызывая разработанные в заданиях MPI9Matr12− MPI9Matr16 функции Matr2ScatterData, Matr2Calc (в цикле) и Matr2GatherData, получить и вывести в главном процессе матрицу C, равную произведению исходных матриц A и B.

После каждого вызова функции Matr2Calc дополнительно выводить в каждом процессе текущее содержимое полосы C_R.

Перед использованием в данном задании следует модифицировать разработанную в MPI9Matr14 функцию Matr2Calc таким образом, чтобы при значении параметра step, равном $K - 1$, не выполнялась пересылка полос A_R.

MPI9Matr18*. В главном процессе даны числа M, P, Q, а также имена двух файлов, содержащих элементы матриц A и B размера $M \times P$ и $P \times Q$ соответственно. Известно, что число Q кратно количеству процессов K. Модифицировать начальный этап ленточного алгоритма перемножения матриц (см. MPI9Matr12) таким образом, чтобы каждый процесс считывал соответствующие полосы матриц A и B непосредственно из исходных файлов.

Для пересылки размеров матриц и имен файлов использовать коллективную функцию MPI_Bcast. Для считывания горизонтальных полос матрицы A использовать коллективные функции MPI_File_seek и MPI_File_read_all; для считывания вертикальных полос матрицы B задать соответствующий вид данных, используя функцию MPI_File_set_view и новый файловый тип MPI_BAND_B, определенный с помощью функции Matr2CreateTypeBand (см. MPI9Matr11), после чего также использовать функцию MPI_File_read_all.

Оформить все действия в виде функции Matr2ScatterFile (без параметров), в результате вызова которой каждый процесс получает значения N_A, P, N_B, а также одномерные массивы, заполненные соответствующими полосами матриц A, B, C. После вызова функции Matr2ScatterFile вывести в каждом процессе полученные данные (числа N_A, P, N_B и полосы матриц A, B, C). Ввод исходных данных осуществлять в функции Matr2ScatterFile, вывод результатов выполнять во внешней функции Solve.

Примечание. Дополнительное условие о кратности значения Q числу K позволяет организовать считывание полос B_R с использованием одинакового файлового типа во всех процессах.

При отсутствии этого условия потребовалось бы применять специальные типы, обеспечивающие корректное считывание из файла и запись в массив «укороченных» полос матрицы B в последних процессах (кроме того, в этом случае потребовалось бы переслать каждому процессу значение Q, необходимое для правильного определения типов для «укороченных» полос).

MPI9Matr19*°. В каждом процессе даны числа N_A, N_B, а также одномерные массивы, заполненные полосами C_R (размера $(N_A \cdot K) \times N_B$), полученными в результате выполнения K шагов ленточного алгоритма перемножения матриц (см. MPI9Matr15). Кроме того, в главном процессе дано число M — количество строк результирующего матричного произведения и имя файла для хранения этого произведения. Дополнительно известно, что количество столбцов Q результирующего матричного произведения кратно числу процессов (и, следовательно, равно $N_B \cdot K$).

Переслать число M и имя файла во все процессы (используя функцию MPI_Bcast) и записать все фрагменты матричного произведения, содержащиеся в полосах C_R, в результирующий файл, который в итоге будет содержать матрицу C размера $M \times Q$.

Для записи полос в файл задать соответствующий вид данных, используя функцию MPI_File_set_view и новый файловый тип MPI_BAND_C, определенный с помощью функции Matr2CreateTypeBand (см. MPI9Matr11), после чего использовать коллективную функцию MPI_File_write_all.

Оформить считывание имени файла, пересылку значения M и имени файла, а также все действия по записи полос в файл в виде функции Matr2GatherFile (считывание всех исходных данных, кроме имени файла, должно осуществляться во внешней функции Solve).

Указание. При записи данных в результирующий файл необходимо учитывать, что полосы C_R могут содержать завершающие нулевые строки, не связанные с полученным произведением (именно для проверки этой ситуации требуется пересылать во все процессы значение M).

Примечание. Решение данной задачи приведено в п. 1.4.5.

MPI9Matr20**. В главном процессе даны числа M, P, Q, а также имена трех файлов: вначале даются имена двух существующих файлов, содержащих элементы матриц A и B размера $M \times P$ и $P \times Q$ соответственно, а затем имя файла для хранения результирующего матричного произведения $C = AB$. Дополнительно известно, что число Q кратно количеству процессов K.

Последовательно вызывая разработанные в заданиях MPI9Matr18, MPI9Matr15 и MPI9Matr19 функции Matr2ScatterFile, Matr2Calc (в цикле) и Matr2GatherFile, получить в результирующем файле произведение исходных матриц A и B, найденное с помощью второго варианта ленточного алгоритма.

После каждого вызова функции Matr2Calc дополнительно выводить в каждом процессе текущее значение элемента c[step], где c — одномерный массив, содержащий полосу C_R, а step — номер шага алгоритма (0, 1, …, $K - 1$); таким образом, на первом шаге алгоритма следует вывести элемент c[0], на втором шаге — элемент c[1], и т. д.

Блочный алгоритм Кэннона

MPI9Matr21. В каждом процессе даны числа M и P; кроме того, в главном процессе дана матрица A размера $M \times P$. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$, а числа M и P кратны числу K_0. Прочесть в главном процессе матрицу A в одномерный массив размера $M \cdot P$ и определить новый тип MPI_BLOCK_A, содержащий блок матрицы A размера $M_0 \times P_0$, где $M_0 = M/K_0$, $P_0 = P/K_0$.

При определении типа MPI_BLOCK_A использовать функции MPI_Type_vector и MPI_Type_commit, оформив это определение в виде функции Matr3CreateTypeBlock(m0, p0, p, t), где целочисленные параметры m0, p0, p являются входными, а параметр t типа MPI_Datatype является выходным; при этом параметры m0 и p0 определяют размеры блока, а параметр p — число столбцов матрицы, из которой извлекается этот блок.

Используя тип MPI_BLOCK_A, переслать в каждый процесс (включая главный) соответствующий блок матрицы A, перебирая блоки по строкам и пересылая их в процессы в порядке возрастания их рангов (первый блок пересылается в процесс 0, следующий за ним блок в этой же строке пересылается в процесс 1, и т. д.). Пересылку выполнять с

помощью функций MPI_Send и MPI_Recv; блоки хранить в одномерных массивах размера $M_0 \cdot P_0$. Вывести в каждом процессе полученный блок.

Примечание. В реализации MPICH2 версии 1.3 с помощью функции MPI_Send нельзя выполнить пересылку данных в тот же процесс, из которого данные посылаются (происходит зависание программы). Для пересылки блока в главный процесс можно использовать функцию MPI_Sendrecv; можно также заполнить соответствующий блок в главном процессе, не прибегая к средствам библиотеки MPI.

MPI9Matr22. В каждом процессе даны числа M_0 и P_0, а также матрица A размера $M_0 \times P_0$. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$. Прочесть в каждом процессе матрицу A в одномерный массив размера $M_0 \cdot P_0$ и определить на множестве исходных процессов коммуникатор MPI_COMM_GRID, задающий декартову топологию двумерной квадратной циклической решетки порядка K_0 (исходный порядок нумерации процессов сохраняется).

При определении коммуникатора MPI_COMM_GRID использовать функцию MPI_Cart_create, оформив это определение в виде функции Matr3CreateCommGrid(comm) с выходным параметром comm типа MPI_Comm. Вывести в каждом процессе координаты (I_0, J_0) этого процесса в созданной топологии, используя функцию MPI_Cart_coords.

Для каждой строки I_0 полученной решетки осуществить циклический сдвиг матриц A на I_0 позиций влево (т. е. в направлении убывания рангов процессов), используя функции MPI_Cart_shift и MPI_Sendrecv_replace. Вывести в каждом процессе матрицу, полученную в результате сдвига.

MPI9Matr23*. В главном процессе даны числа M, P, Q и матрицы A и B размера $M \times P$ и $P \times Q$ соответственно. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$. В блочных алгоритмах перемножения матриц исходные матрицы разбиваются на K блоков, образуя квадратные блочные матрицы порядка K_0 (в дальнейшем блоки распределяются по процессам и используются для вычисления в каждом процессе части итогового матричного произведения).

Блок для матрицы A имеет размер $M_0 \times P_0$, блок для матрицы B имеет размер $P_0 \times Q_0$; числа M_0, P_0, Q_0 вычисляются по формулам $M_0 = \text{ceil}(M/K_0)$, $P_0 = \text{ceil}(P/K_0)$, $Q_0 = \text{ceil}(Q/K_0)$, где операция «/» означает вещественное деление, а функция ceil выполняет округление с избытком. Если матрица содержит недостаточно строк (или столбцов) для заполнения последних блоков, то блоки дополняются нулевыми строками (столбцами).

Сохранить исходные матрицы A и B, дополненные при необходимости нулевыми строками и столбцами до размеров $(M_0 \cdot K_0) \times (P_0 \cdot K_0)$ и $(P_0 \cdot K_0) \times (Q_0 \cdot K_0)$, в одномерных массивах в главном процессе, после чего организовать пересылку блоков из этих массивов во все процессы, перебирая блоки по строкам и пересылая их в процессы в порядке возрастания их рангов (процесс ранга R получит блоки A_R и B_R, $R = 0, \ldots, K - 1$). Кроме того, создать в каждом процессе блок C_R для хранения фрагмента матричного произведения $C = AB$, которое будет вычисляться в этом процессе; каждый блок C_R имеет размер $M_0 \times Q_0$ и заполняется нулевыми элементами.

Блоки, как и исходные матрицы, должны храниться по строкам в одномерных массивах соответствующего размера. Для пересылки размеров матриц использовать коллективную функцию MPI_Bcast, для пересылки блоков матриц A и B использовать функции MPI_Send и MPI_Recv, а также вспомогательные типы MPI_BLOCK_A и MPI_BLOCK_B, соз-

данные с помощью функции Matr3CreateTypeBlock (см. задание MPI9Matr21 и примечание к нему).

Оформить все описанные действия в виде функции Matr3ScatterData (без параметров), в результате вызова которой каждый процесс получает значения M_0, P_0, Q_0, а также одномерные массивы, заполненные соответствующими блоками матриц A, B, C. После вызова функции Matr3ScatterData вывести в каждом процессе полученные данные (числа M_0, P_0, Q_0 и блоки матриц A, B, C). Ввод исходных данных осуществлять в функции Matr3ScatterData, вывод результатов выполнять во внешней функции Solve.

Указания. (1) При считывании матриц A и B в главном процессе следует учитывать, что предназначенные для ее хранения массивы могут содержать элементы, соответствующие дополнительным нулевым столбцам.

(2) Для уменьшения числа вызовов функции MPI_Bcast все пересылаемые размеры матриц можно поместить во вспомогательный массив.

MPI9Matr24°. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, заполненные соответствующими блоками матриц A, B, C (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr23). Реализовать начальное перераспределение блоков, используемое в алгоритме Кэннона блочного перемножения матриц.

Для этого задать на множестве исходных процессов декартову топологию двумерной квадратной циклической решетки порядка K_0 (где $K_0 \cdot K_0$ равно количеству процессов), сохранив исходный порядок нумерации процессов, и выполнить для каждой строки I_0 полученной решетки ($I_0 = 0, \ldots, K_0 - 1$) циклический сдвиг блоков A_R на I_0 позиций влево (т. е. в направлении убывания рангов процессов), а для каждого столбца J_0 решетки ($J_0 = 0, \ldots, K_0 - 1$) циклический сдвиг блоков B_R на J_0 позиций вверх (т. е. также в направлении убывания рангов процессов).

Для определения коммуникатора MPI_COMM_GRID, связанного с декартовой топологией, использовать функцию Matr3CreateCommGrid, реализованную в задании MPI9Matr22. При выполнении циклического сдвига использовать функции MPI_Cart_coords, MPI_Cart_shift, MPI_Sendrecv_replace (ср. с MPI9Matr22).

Оформить описанные действия в виде функции Matr3Init (без параметров). Вывести в каждом процессе блоки A_R и B_R, полученные в результате сдвига (ввод и вывод данных выполнять во внешней функции Solve).

Примечание. Решение данной задачи приведено в п. 1.4.4.

MPI9Matr25. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, заполненные соответствующими блоками матриц A, B и C, причем известно, что блоки C_R являются нулевыми, а для блоков A_R и B_R уже выполнено их начальное перераспределение в соответствии с алгоритмом Кэннона (см. предыдущее задание). Реализовать один шаг алгоритма Кэннона перемножения матриц, выполнив перемножение элементов, содержащихся в блоках A_R и B_R каждого процесса, после чего организовать для каждой строки декартовой решетки циклический сдвиг блоков A_R на 1 позицию влево (т. е. в направлении убывания рангов процессов), а для каждого столбца решетки циклический сдвиг блоков B_R на 1 позицию вверх (т. е. также в направлении убывания рангов процессов).

Для циклической пересылки данных использовать коммуникатор MPI_COMM_GRID, создав его с помощью функции Matr3CreateCommGrid (см. задание MPI9Matr22), и функции MPI_Cart_shift и MPI_Sendrecv_replace (ср. с MPI9Matr22).

Оформить эти действия в виде функции Matr3Calc (без параметров). Вывести новое содержимое блоков C_R, A_R и B_R в каждом процессе (ввод и вывод данных выполнять во внешней функции Solve).

Примечание. Особенность алгоритма Кэннона состоит в том, что действия на каждом его шаге не зависят от номера шага.

MPI9Matr26*. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, заполненные соответствующими блоками матриц A, B и C, причем известно, что блоки C_R являются нулевыми, а для блоков A_R и B_R уже выполнено их начальное перераспределение в соответствии с алгоритмом Кэннона (см. задание MPI9Matr24). Кроме того, в каждом процессе дано одно и то же число L, лежащее в диапазоне от 2 до K_0 (где $K_0 \cdot K_0$ равно количеству процессов) и определяющее требуемое число шагов алгоритма Кэннона.

Вызывая в цикле функцию Matr3Calc, разработанную в предыдущем задании, выполнить L начальных шагов алгоритма Кэннона и вывести в каждом процессе новое содержимое блоков C_R, A_R и B_R (ввод и вывод данных выполнять во внешней функции Solve).

Примечание. Если значение L равно K_0, то блоки C_R будут содержать соответствующие фрагменты итогового матричного произведения AB.

MPI9Matr27*. В главном процессе даны числа M и Q — число строк и столбцов результирующего матричного произведения. Кроме того, в каждом процессе даны числа M_0, Q_0, а также одномерные массивы, заполненные блоками матрицы C (размера $M_0 \times Q_0$), которые были получены в результате выполнения K_0 шагов блочного алгоритма Кэннона перемножения матриц (см. MPI9Matr26). Переслать все блоки C_R в главный процесс и вывести в нем полученную матрицу C (размера $M \times Q$).

Для хранения результирующей матрицы C в главном процессе использовать одномерный массив, достаточный для хранения матрицы размера $(M_0 \cdot K_0) \times (Q_0 \cdot K_0)$. Для пересылки блоков C_R использовать функции MPI_Send и MPI_Recv, а также вспомогательный тип MPI_BLOCK_C, созданный с помощью функции Matr3CreateTypeBlock (см. задание MPI9Matr21 и примечание к нему).

Оформить эти действия в виде функции Matr3GatherData (без параметров). Ввод данных выполнять во внешней функции Solve, вывод полученной матрицы включить в функцию Matr3GatherData.

Указание. При выводе матрицы C в главном процессе следует учитывать, что предназначенный для ее хранения массив может содержать элементы, соответствующие дополнительным нулевым столбцам.

MPI9Matr28**. В главном процессе даны числа M, P, Q и матрицы A и B размера $M \times P$ и $P \times Q$ соответственно (таким образом, исходные данные совпадают с исходными данными для задания MPI9Matr23).

Последовательно вызывая разработанные в заданиях MPI9Matr23− MPI9Matr27 функции Matr3ScatterData, Matr3Init, Matr3Calc (в цикле) и Matr3GatherData, получить и вывести в главном процессе матрицу C, равную произведению исходных матриц A и B.

После каждого вызова функции Matr3Calc дополнительно выводить в каждом процессе текущее содержимое блока C_R.

Для того чтобы коммуникатор MPI_COMM_GRID, используемый в функциях Matr3Init и Matr3Calc, не создавался несколько раз, модифицировать функцию Matr3CreateCommGrid

таким образом, чтобы при ее вызове для уже определенного коммуникатора (не равного MPI_COMM_NULL) она не выполняла никаких действий.

Перед использованием в данном задании следует модифицировать разработанную в MPI9Matr25 функцию Matr3Calc, добавив в нее параметр step, равный номеру шага (step = 0, …, $K_0 - 1$), и изменив ее таким образом, чтобы при значении параметра step, равном $K_0 - 1$, не выполнялась пересылка блоков A_R и B_R.

MPI9Matr29*. В главном процессе даны числа *M, P, Q*, а также имена двух файлов, содержащих элементы матриц *A* и *B* размера $M \times P$ и $P \times Q$ соответственно. Модифицировать этап получения блоков для блочного алгоритма Кэннона перемножения матриц (см. MPI9Matr23) таким образом, чтобы каждый процесс считывал соответствующие блоки матриц *A* и *B* непосредственно из исходных файлов. В данном случае всем процессам требуется переслать не только размеры блоков M_0, P_0, Q_0, но и размеры исходных матриц *M, P, Q*, которые требуются для правильного определения позиций блоков в исходных файлах.

Для пересылки размеров матриц и имен файлов использовать коллективную функцию MPI_Bcast. Для считывания блоков использовать локальные функции MPI_File_read_at, вызывая отдельную функцию для считывания каждой строки блока (новый вид файловых данных создавать не требуется).

Оформить все действия в виде функции Matr3ScatterFile (без параметров), в результате вызова которой каждый процесс получает значения *M, P, Q*, M_0, P_0, Q_0, а также одномерные массивы, заполненные соответствующими блоками матриц *A, B, C*. После вызова функции Matr3ScatterFile вывести в каждом процессе полученные данные (числа *M, P, Q*, M_0, P_0, Q_0 и блоки матриц *A, B, C*). Ввод исходных данных осуществлять в функции Matr3ScatterFile, вывод результатов выполнять во внешней функции Solve.

Указание. При чтении файловых данных следует учитывать, что для некоторых блоков часть элементов (последние строки и/или столбцы) не должна считываться из исходных файлов и будет оставаться нулевой. Для определения фактического размера считываемого блока (числа строк и числа столбцов) потребуется использовать размеры исходных матриц и координаты блока (I_0, J_0) в квадратной декартовой решетке порядка K_0, которые легко определить по рангу процесса *R*: $I_0 = R/K_0$, $J_0 = R\%K_0$.

Примечание. В то время как значения *P* и *Q* необходимы для обеспечения правильного считывания файловых блоков, значение *M* можно не использовать, поскольку попытка чтения данных за концом файла просто игнорируется, не приводя к ошибке. Однако значение *M* потребуется на завершающем этапе алгоритма (см. следующее задание), поэтому его тоже необходимо переслать всем процессам.

MPI9Matr30*. В каждом процессе даны числа *M, Q*, M_0, Q_0, а также одномерные массивы, заполненные блоками C_R (размера $M_0 \times Q_0$), полученными в результате выполнения K_0 шагов блочного алгоритма Кэннона перемножения матриц (см. MPI9Matr25). Кроме того, в главном процессе дано имя файла для хранения результирующего матричного произведения.

Переслать имя файла во все процессы (используя функцию MPI_Bcast) и записать все фрагменты матричного произведения, содержащиеся в блоках C_R, в результирующий файл, который в итоге будет содержать матрицу *C* размера $M \times Q$.

Для записи данных в файл использовать локальные функции MPI_File_write_at, вызывая отдельную функцию для записи каждой строки блока (новый вид файловых данных создавать не требуется).

Оформить считывание имени файла, его пересылку, а также все действия по записи данных в файл в виде функции Matr3GatherFile (считывание всех исходных данных, кроме имени файла, должно осуществляться во внешней функции Solve).

Указание. При записи файловых данных следует учитывать, что для некоторых блоков C_R часть элементов (последние строки и/или столбцы, заполненные нулями) не должна записываться в результирующий файл. См. также указание и примечание к предыдущему заданию.

MPI9Matr31**. В главном процессе даны числа M, P, Q, а также имена трех файлов: вначале даются имена двух существующих файлов, содержащих элементы матриц A и B размера $M \times P$ и $P \times Q$ соответственно, а затем имя файла для хранения результирующего матричного произведения $C = AB$.

Последовательно вызывая разработанные в заданиях MPI9Matr29, MPI9Matr24, MPI9Matr25 и MPI9Matr30 функции Matr3ScatterFile, Matr3Init, Matr3Calc (в цикле) и Matr3GatherFile, получить в результирующем файле произведение исходных матриц A и B, найденное с помощью блочного алгоритма Кэннона.

После каждого вызова функции Matr3Calc дополнительно выводить в каждом процессе текущее значение элемента c[step], где c — одномерный массив, содержащий блок C_R, а step — номер шага алгоритма (0, 1, …, $K_0 - 1$); таким образом, на первом шаге алгоритма следует вывести элемент c[0], на втором шаге — элемент c[1], и т. д.

Блочный алгоритм Фокса

MPI9Matr32. В каждом процессе даны числа M и P; кроме того, в главном процессе дана матрица A размера $M \times P$. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$, а числа M и P кратны числу K_0. Прочесть в главном процессе матрицу A в одномерный массив размера $M \cdot P$ и определить новый тип MPI_BLOCK_A, содержащий блок матрицы A размера $M_0 \times P_0$, где $M_0 = M/K_0$, $P_0 = P/K_0$.

При определении типа MPI_BLOCK_A использовать функции MPI_Type_vector и MPI_Type_commit, оформив это определение в виде функции Matr4CreateTypeBlock(m0, p0, p, t), где целочисленные параметры m0, p0, p являются входными, а параметр t типа MPI_Datatype является выходным; при этом параметры m0 и p0 определяют размеры блока, а параметр p — число столбцов матрицы, из которой извлекается этот блок.

Используя тип MPI_BLOCK_A, переслать в каждый процесс (включая главный) соответствующий блок матрицы A, перебирая блоки по строкам и пересылая их в процессы в порядке возрастания их рангов (первый блок пересылается в процесс 0, следующий за ним блок в этой же строке пересылается в процесс 1, и т. д.). Пересылку выполнять с помощью коллективной функции MPI_Alltoallw; блоки хранить в одномерных массивах размера $M_0 \cdot P_0$. Вывести в каждом процессе полученный блок.

Указания. (1) При выполнении задания с применением библиотеки MPI-1 вместо функции MPI_Alltoallw следует использовать функции MPI_Send и MPI_Recv.

(2) Функция MPI_Alltoallw, введенная в MPI-2, является единственной коллективной функцией, которая позволяет указывать смещения для пересылаемых данных в *байтах*, а не в элементах. Это дает возможность использовать ее совместно со сложными типами данных для реализации любых вариантов коллективных обменов (в данном случае требуется реализовать вариант вида Scatter).

Следует учитывать, что при подобном варианте рассылки все параметры-массивы функции MPI_Alltoallw, связанные с посылаемыми данными, необходимо по-разному оп-

ределять в главном и подчиненных процессах. В частности, массив scounts (определяющий количество посылаемых данных) должен содержать значения 0 во всех подчиненных процессах и значения 1 в главном процессе (посылаемые элементы имеют тип MPI_BLOCK_A).

В то же время, массивы, связанные с принимаемыми данными, будут определяться одинаковым образом во всех процессах; в частности, в массиве rcounts (определяющем количество принимаемых данных) элемент с индексом 0 должен быть равен $M_0 \cdot P_0$, а все остальные элементы должны быть равны 0 (принимаемые элементы имеют тип MPI_INT).

Необходимо обратить особое внимание на правильное определение элементов в массиве sdispls смещений для посылаемых данных в главном процессе (в подчиненных процессах этот массив достаточно обнулить).

MPI9Matr33. В каждом процессе даны числа M_0 и P_0, а также матрица A размера $M_0 \times P_0$. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$. Прочесть в каждом процессе матрицу A в одномерный массив размера $M_0 \cdot P_0$ и определить на множестве исходных процессов коммуникатор MPI_COMM_GRID, задающий декартову топологию двумерной квадратной циклической решетки порядка K_0 (исходный порядок нумерации процессов сохраняется).

При определении коммуникатора MPI_COMM_GRID использовать функцию MPI_Cart_create, оформив это определение в виде функции Matr4CreateCommGrid(comm) с выходным параметром comm типа MPI_Comm. Вывести в каждом процессе координаты (I_0, J_0) этого процесса в созданной топологии, используя функцию MPI_Cart_coords.

На основе коммуникатора MPI_COMM_GRID создать набор коммуникаторов MPI_COMM_ROW, связанных со строками исходной двумерной решетки. Для определения коммуникаторов MPI_COMM_ROW использовать функцию MPI_Cart_sub, оформив это определение в виде вспомогательной функции Matr4CreateCommRow(grid, row) с входным параметром grid (коммуникатором, связанным с исходной двумерной решеткой) и выходным параметром row (оба параметра типа MPI_Comm). Вывести в каждом процессе его ранг R_0 для коммуникатора MPI_COMM_ROW (этот ранг должен совпадать со значением J_0).

Кроме того, для каждой строки I_0 полученной решетки осуществить пересылку матрицы A из столбца I_0 во все процессы этой же строки, используя коллективную функцию MPI_Bcast для коммуникатора MPI_COMM_ROW и сохранив результат во вспомогательной матрице T того же размера, что и матрица A (перед пересылкой необходимо скопировать в матрицу T рассылающего процесса элементы пересылаемой матрицы A). Вывести в каждом процессе полученную матрицу T.

MPI9Matr34. В каждом процессе даны числа P_0 и Q_0, а также матрица B размера $P_0 \times Q_0$. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$. Прочесть в каждом процессе матрицу B в одномерный массив размера $P_0 \cdot Q_0$ и определить на множестве исходных процессов коммуникатор MPI_COMM_GRID, задающий декартову топологию двумерной квадратной циклической решетки порядка K_0.

Для определения коммуникатора MPI_COMM_GRID использовать функцию Matr4CreateCommGrid (см. задание MPI9Matr33). Вывести в каждом процессе его координаты (I_0, J_0) в созданной топологии, используя функцию MPI_Cart_coords.

На основе коммуникатора MPI_COMM_GRID создать набор коммуникаторов MPI_COMM_COL, связанных со столбцами исходной двумерной решетки. Для определе-

ния коммуникаторов MPI_COMM_COL использовать функцию MPI_Cart_sub, оформив это определение в виде функции Matr4CreateCommCol(grid, col) с входным параметром grid (коммуникатором, связанным с исходной двумерной решеткой) и выходным параметром col (оба параметра типа MPI_Comm). Вывести в каждом процессе его ранг R_0 для коммуникатора MPI_COMM_COL (этот ранг должен совпадать со значением I_0).

Кроме того, для каждого столбца J_0 полученной решетки осуществить циклический сдвиг матриц B на 1 позицию вверх (т. е. в направлении убывания рангов процессов), используя функции MPI_Sendrecv_replace для коммуникатора MPI_COMM_COL (при определении рангов процесса-отправителя и процесса-получателя использовать операцию % взятия остатка от деления). Вывести в каждом процессе матрицу, полученную в результате сдвига.

MPI9Matr35*. В главном процессе даны числа M, P, Q и матрицы A и B размера $M \times P$ и $P \times Q$ соответственно. Известно, что количество процессов K является полным квадратом: $K = K_0 \cdot K_0$. В блочных алгоритмах перемножения матриц исходные матрицы разбиваются на K блоков, образуя квадратные блочные матрицы порядка K_0 (в дальнейшем блоки распределяются по процессам и используются для вычисления в каждом процессе части итогового матричного произведения).

Блок для матрицы A имеет размер $M_0 \times P_0$, блок для матрицы B имеет размер $P_0 \times Q_0$; числа M_0, P_0, Q_0 вычисляются по формулам $M_0 = \text{ceil}(M/K_0)$, $P_0 = \text{ceil}(P/K_0)$, $Q_0 = \text{ceil}(Q/K_0)$, где операция «/» означает вещественное деление, а функция ceil выполняет округление с избытком. Если матрица содержит недостаточно строк (или столбцов) для заполнения последних блоков, то блоки дополняются нулевыми строками (столбцами).

Сохранить исходные матрицы A и B, дополненные при необходимости нулевыми строками и столбцами до размеров $(M_0 \cdot K_0) \times (P_0 \cdot K_0)$ и $(P_0 \cdot K_0) \times (Q_0 \cdot K_0)$, в одномерных массивах в главном процессе, после чего организовать пересылку блоков из этих массивов во все процессы, перебирая блоки по строкам и пересылая их в процессы в порядке возрастания их рангов (процесс ранга R получит блоки A_R и B_R, $R = 0, \ldots, K - 1$). Кроме того, создать в каждом процессе два блока, заполненных нулевыми элементами: блок C_R размера $M_0 \times Q_0$ для хранения фрагмента матричного произведения $C = AB$, которое будет вычисляться в этом процессе, и вспомогательный блок T_R того же размера $M_0 \times P_0$, что и блок A_R.

Блоки, как и исходные матрицы, должны храниться по строкам в одномерных массивах соответствующего размера. Для пересылки размеров матриц использовать коллективную функцию MPI_Bcast, для пересылки блоков матриц A и B использовать коллективную функцию MPI_Alltoallw, а также вспомогательные типы MPI_BLOCK_A и MPI_BLOCK_B, созданные с помощью функции Matr4CreateTypeBlock (см. задание MPI9Matr32, а также указания к нему).

Оформить все описанные действия в виде функции Matr4ScatterData (без параметров), в результате вызова которой каждый процесс получает значения M_0, P_0, Q_0, а также одномерные массивы, содержащие элементы блоков A_R, B_R, C_R и T_R. После вызова функции Matr4ScatterData вывести в каждом процессе полученные данные (числа M_0, P_0, Q_0 и блоки A_R, B_R, C_R, T_R). Ввод исходных данных осуществлять в функции Matr4ScatterData, вывод данных выполнять во внешней функции Solve.

Указания. (1) При считывании матриц A и B в главном процессе следует учитывать, что предназначенные для ее хранения массивы могут содержать элементы, соответствующие дополнительным нулевым столбцам.

(2) Для уменьшения числа вызовов функции MPI_Bcast все пересылаемые размеры матриц можно поместить во вспомогательный массив.

MPI9Matr36. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, содержащие элементы блоков A_R, B_R, C_R и T_R (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr35). Каждый шаг блочного алгоритма Фокса перемножения матриц состоит из двух этапов.

На первом этапе первого шага для каждой строки I_0 квадратной решетки процессов порядка K_0 ($I_0 = 0, \ldots, K_0 - 1$, где $K_0 \cdot K_0$ равно количеству процессов) выполняется пересылка блока A_R из процесса, расположенного в строке I_0 и столбце с тем же номером I_0, во все процессы этой же строки (с сохранением пересланного блока в блоке T_R), после чего полученный в результате этой пересылки блок T_R умножается на блок B_R из этого процесса, и результат добавляется к блоку C_R.

Реализовать первый этап первого шага алгоритма Фокса. Для пересылки блоков A_R использовать функцию MPI_Bcast для коммуникатора MPI_COMM_ROW, создав этот коммуникатор с помощью функции Matr4CreateCommRow (см. задание MPI9Matr33, в котором описывается аналогичная пересылка данных).

Оформить все описанные действия в виде функции Matr4Calc1 (без параметров). Вывести новое содержимое блоков T_R и C_R в каждом процессе (ввод и вывод данных выполнять во внешней функции Solve).

MPI9Matr37. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, содержащие элементы блоков A_R, B_R, C_R и T_R (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr35).

Реализовать второй этап первого шага алгоритма Фокса, который состоит в циклическом сдвиге блоков B_R для каждого столбца декартовой решетки на 1 позицию вверх (т. е. в направлении убывания рангов процессов).

Для циклической пересылки блоков B_R использовать функцию MPI_Sendrecv_replace для коммуникатора MPI_COMM_COL, создав этот коммуникатор с помощью функции Matr4CreateCommCol (см. задание MPI9Matr34, в котором описывается аналогичная пересылка данных).

Оформить эти действия в виде функции Matr4Calc2 (без параметров). Вывести новое содержимое блока B_R в каждом процессе (ввод и вывод данных выполнять во внешней функции Solve).

MPI9Matr38. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, содержащие элементы блоков A_R, B_R, C_R и T_R (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr35).

Модифицировать функцию Matr4Calc1, реализованную в задании MPI9Matr36, таким образом, чтобы она обеспечивала выполнение первого этапа на любом шаге алгоритма Фокса. Для этого добавить к ней параметр step, определяющий номер шага (изменяется от 0 до $K_0 - 1$, где K_0 — порядок декартовой решетки процессов), и учесть значение этого шага при рассылке блоков A_R: на шаге step для каждой строки I_0 декартовой решетки должна выполняться рассылка блока A_R из процесса, расположенного в

столбце с номером $(I_0 + \text{step})\%K_0$ (действия, связанные с пересчетом блоков C_R, от номера шага не зависят).

Выполнить два начальных шага алгоритма Фокса, последовательно вызвав функции Matr4Calc1(0), Matr4Calc2() (обеспечивающую второй этап шага алгоритма — см. MPI9Matr37) и Matr4Calc1(1), и вывести в каждом процессе новое содержимое блоков T_R, B_R и C_R (ввод и вывод данных выполнять во внешней функции Solve).

MPI9Matr39*. В каждом процессе даны числа M_0, P_0, Q_0, а также одномерные массивы, содержащие элементы блоков A_R, B_R, C_R и T_R (таким образом, исходные данные совпадают с результатами, полученными в задании MPI9Matr35). Кроме того, в каждом процессе дано одно и то же число L, лежащее в диапазоне от 3 до K_0 и определяющее требуемое число шагов алгоритма Фокса.

Выполнить L начальных шагов алгоритма Фокса, вызвав функции, разработанные в заданиях MPI9Matr38 и MPI9Matr37, в следующей последовательности: Matr4Calc1(0), Matr4Calc2(), Matr4Calc1(1), Matr4Calc2(), …, Matr4Calc1($L - 1$). Вывести в каждом процессе новое содержимое блоков T_R, B_R и C_R (ввод и вывод данных выполнять во внешней функции Solve).

Примечание. Если значение L равно K_0, то блоки C_R будут содержать соответствующие фрагменты матричного произведения AB. Обратите внимание на то, что второй этап (связанный с вызовом функции Matr4Calc2) на последнем шаге алгоритма выполнять не требуется.

MPI9Matr40*. В главном процессе даны числа M и Q — число строк и столбцов результирующего матричного произведения. Кроме того, в каждом процессе даны числа M_0, Q_0, а также одномерные массивы, заполненные блоками матрицы C (размера $M_0 \times Q_0$), которые были получены в результате выполнения K_0 шагов блочного алгоритма Фокса перемножения матриц (см. MPI9Matr39).

Переслать все блоки C_R в главный процесс и вывести в нем полученную матрицу C (размера $M \times Q$). Для хранения результирующей матрицы C в главном процессе использовать одномерный массив, достаточный для хранения матрицы размера $(M_0 \cdot K_0) \times (Q_0 \cdot K_0)$. Для пересылки блоков C_R использовать коллективную функцию MPI_Alltoallw, а также вспомогательный тип MPI_BLOCK_C, созданный с помощью функции Matr4CreateTypeBlock (см. задание MPI9Matr32, а также указания к нему).

Оформить эти действия в виде функции Matr4GatherData (без параметров). Ввод данных выполнять во внешней функции Solve, вывод полученной матрицы включить в функцию Matr4GatherData.

Указание. При выводе матрицы C в главном процессе следует учитывать, что предназначенный для ее хранения массив может содержать элементы, соответствующие дополнительным нулевым столбцам.

MPI9Matr41**. В главном процессе даны числа M, P, Q и матрицы A и B размера $M \times P$ и $P \times Q$ соответственно (таким образом, исходные данные совпадают с исходными данными для задания MPI9Matr35).

Последовательно вызывая разработанные в заданиях MPI9Matr35− MPI9Matr40 функции Matr4ScatterData, Matr4Calc1, Matr4Calc2 и Matr4GatherData, получить и вывести в главном процессе матрицу C, равную произведению исходных матриц A и B. Функции Matr4Calc1 и Matr4Calc2 должны вызываться в цикле, причем количество вызовов функции Matr4Calc2 должно быть на 1 меньше количества вызовов функции Matr4Calc1.

После каждого вызова функции Matr4Calc1 дополнительно выводить в каждом процессе текущее содержимое блока C_R.

Для того чтобы вспомогательные коммуникаторы MPI_COMM_GRID, MPI_COMM_ROW и MPI_COMM_COL, используемые в функциях Matr4Calc1 и Matr4Calc2, не создавались несколько раз, модифицировать функции Matr4CreateCommGrid, Matr4CreateCommRow, Matr4CreateCommCol (см. MPI9Matr33 и MPI9Matr34) таким образом, чтобы при их вызове для уже определенного коммуникатора (не равного MPI_COMM_NULL) они не выполняли никаких действий.

MPI9Matr42*. В главном процессе даны числа M, P, Q, а также имена двух файлов, содержащих элементы матриц A и B размера $M \times P$ и $P \times Q$ соответственно. Дополнительно известно, что числа M, P и Q кратны порядку K_0 квадратной решетки процессов.

Модифицировать этап формирования блоков для блочного алгоритма Фокса перемножения матриц (см. MPI9Matr35) таким образом, чтобы каждый процесс считывал соответствующие блоки матриц A и B непосредственно из исходных файлов.

Для пересылки размеров матриц и имен файлов использовать коллективную функцию MPI_Bcast. Для считывания блоков задать соответствующий вид данных, используя функцию MPI_File_set_view и типы MPI_BLOCK_A и MPI_BLOCK_B, определенные с помощью функции Matr4CreateTypeBlock (см. MPI9Matr32) после чего использовать коллективную функцию MPI_File_read_all.

Оформить все действия в виде функции Matr4ScatterFile (без параметров), в результате вызова которой каждый процесс получает значения M_0, P_0, Q_0, одномерные массивы, содержащие элементы блоков A_R, B_R, C_R и T_R (блоки C_R и T_R должны быть нулевыми). После вызова функции Matr4ScatterFile вывести в каждом процессе полученные данные (числа M_0, P_0, Q_0 и блоки A_R, B_R, C_R, T_R). Ввод исходных данных осуществлять в функции Matr4ScatterFile, вывод результатов выполнять во внешней функции Solve.

Примечание. Дополнительное условие о кратности чисел M, P, Q числу K_0 означает, что блоки, полученные из матриц A и B, не требуется дополнять нулевыми строками и/или столбцами, и поэтому для чтения из файлов блоков матриц A и B в любые процессы можно использовать одинаковые файловые типы MPI_BLOCK_A и MPI_BLOCK_B.

При отсутствии этого условия потребовалось бы применять специальные типы, обеспечивающие корректное считывание из файла и запись в массив «укороченных» блоков матриц A и B (кроме того, в этом случае потребовалось бы переслать каждому процессу значения P и Q, необходимые для правильного определения типов для «укороченных» блоков).

MPI9Matr43*. В каждом процессе даны числа M_0, Q_0, а также одномерные массивы, заполненные блоками C_R (размера $M_0 \times Q_0$), полученными в результате выполнения K_0 шагов блочного алгоритма Фокса перемножения матриц (см. MPI9Matr39). Кроме того, в главном процессе дано имя файла для хранения результирующего матричного произведения. Дополнительно известно, что число строк M и число столбцов Q матричного произведения кратны порядку K_0 квадратной решетки процессов (таким образом, $M = M_0 \cdot K_0$, $Q = Q_0 \cdot K_0$).

Переслать имя файла во все процессы (используя функцию MPI_Bcast) и записать все фрагменты матричного произведения, содержащиеся в блоках C_R, в результирующий файл, который в итоге будет содержать матрицу C размера $M \times Q$.

Для записи блоков задать соответствующий вид данных, используя функцию MPI_File_set_view и файловый тип MPI_BLOCK_C, определенный с помощью функции

Matr4CreateTypeBlock (см. MPI9Matr32), после чего использовать коллективную функцию MPI_File_write_all.

Оформить считывание имени файла, его пересылку, а также все действия по записи данных в файл в виде функции Matr4GatherFile (считывание всех исходных данных, кроме имени файла, должно осуществляться во внешней функции Solve).

Примечание. Дополнительное условие о кратности чисел M и Q числу K_0 означает, что блоки C_R не содержат «лишних» нулевых строк и/или столбцов, и поэтому для их записи в файл из любых процессов можно использовать одинаковые файловые типы MPI_BLOCK_C.

MPI9Matr44**. В главном процессе даны числа M, P, Q, а также имена трех файлов: вначале даются имена двух существующих файлов, содержащих элементы матриц A и B размера $M \times P$ и $P \times Q$ соответственно, а затем имя файла для хранения результирующего матричного произведения $C = AB$. Дополнительно известно, что числа M, P и Q кратны порядку K_0 квадратной решетки процессов.

Последовательно вызывая разработанные в заданиях MPI9Matr42, MPI9Matr38, MPI9Matr37 и MPI9Matr43 функции Matr4ScatterFile, Matr4Calc1, Matr4Calc2 и Matr4GatherFile, получить в результирующем файле произведение исходных матриц A и B, найденное с помощью блочного алгоритма Фокса. Функции Matr4Calc1 и Matr4Calc2 должны вызываться в цикле, причем количество вызовов функции Matr4Calc2 должно быть на 1 меньше количества вызовов функции Matr4Calc1.

После каждого вызова функции Matr4Calc1 дополнительно выводить в каждом процессе текущее значение элемента c[step], где c — одномерный массив, содержащий блок C_R, а step — номер шага алгоритма (0, 1, …, $K_0 - 1$); таким образом, на первом шаге алгоритма следует вывести элемент c[0], на втором шаге — элемент c[1], и т. д.