М. Э. Абрамян

ПРАКТИКУМ ПО ПАРАЛЛЕЛЬНОМУ ПРОГРАММИРОВАНИЮ

С ИСПОЛЬЗОВАНИЕМ ЭЛЕКТРОННОГО ЗАДАЧНИКА PROGRAMMING TASKBOOK FOR MPI-2

Часть I
Учебные задания

Если количество процессов в задании не определено, то можно считать, что это количество не превосходит 16. Под *главным процессом* всюду в формулировках заданий понимается процесс ранга 0 для коммуникатора MPI_COMM_WORLD. Для всех процессов ненулевого ранга в заданиях используется общее наименование *подчиненных процессов*.

Если в задании не указан тип обрабатываемых чисел, то числа считаются вещественными. Если в задании не определяется максимальный размер набора чисел, то его следует считать равным 20.

1. Процессы и их ранги

MPI1Proc1. В каждом из процессов, входящих в коммуникатор MPI_COMM_WORLD, прочесть одно вещественное число и вывести его противоположное значение. Для ввода и вывода данных использовать поток ввода-вывода pt, определенный в задачнике. Кроме того, отобразить найденное значение в разделе отладки, используя функцию Show, также определенную в задачнике.

MPI1Proc2. В каждом из процессов, входящих в коммуникатор MPI_COMM_WORLD, прочесть одно целое число *A* и вывести его удвоенное значение. Кроме того, для *главного процесса* (процесса ранга 0) вывести количество процессов, входящих в коммуникатор MPI_COMM_WORLD. Для ввода и вывода данных использовать поток ввода-вывода pt. В главном процессе продублировать вывод данных в разделе отладки, отобразив на отдельных строках удвоенное значение *A* и количество процессов (использовать два вызова функции ShowLine, определенной в задачнике наряду с функцией Show).

MPI1Proc3. В главном процессе прочесть вещественное число *X* и вывести его противоположное значение, в каждом из остальных процессов (*подчиненных процессов*, ранг которых больше 0) вывести его ранг. Кроме того, продублировать вывод данных в разделе отладки, отобразив значение $-X$ с комментарием «-X=», а значения рангов с комментариями «rank=» (использовать функцию Show с двумя параметрами).

MPI1Proc4. В процессах четного ранга (включая главный) ввести целое число и вывести его удвоенное значение. В процессах нечетного ранга не выполнять никаких действий.

MPI1Proc5. В процессах четного ранга (включая главный) ввести целое число, в процессах нечетного ранга ввести вещественное число. В каждом процессе вывести удвоенное значение введенного числа.

MPI1Proc6. В подчиненных процессах четного ранга ввести целое число, в процессах нечетного ранга ввести вещественное число. В каждом подчиненном процессе вывести удвоенное значение введенного числа. В главном процессе не выполнять никаких действий.

MPI1Proc7. В каждом процессе четного ранга (включая главный) дано целое число N (> 0) и набор из N вещественных чисел. Вывести в каждом из этих процессов сумму чисел из данного набора. В процессах нечетного ранга не выполнять никаких действий.

MPI1Proc8. В каждом процессе дано целое число N (> 0) и набор из N вещественных чисел. В процессах четного ранга (включая главный) вывести сумму чисел из данного набора, в процессах нечетного ранга вывести среднее арифметическое чисел из данного набора.

MPI1Proc9. В каждом процессе дано целое число N (> 0) и набор из N вещественных чисел. В подчиненных процессах четного ранга вывести сумму чисел из данного набора, в процессах нечетного ранга вывести среднее арифметическое чисел из данного набора, в главном процессе вывести произведение чисел из данного набора.

MPI1Proc10. В каждом процессе дано целое число N (> 0) и набор из N чисел, причем в подчиненных процессах нечетного ранга (1, 3, …) набор содержит вещественные числа, в подчиненных процессах четного ранга (2, 4, …) — целые числа, а тип элементов в главном наборе зависит от общего количества процессов: если количество процессов нечетное, то набор содержит целые числа, а если четное, то вещественные. В процессах четного ранга (включая главный) вывести минимальный элемент из данного набора, в процессах нечетного ранга вывести максимальный элемент.

2. Обмен сообщениями между отдельными процессами

2.1. Блокирующая пересылка данных

MPI2Send1. В каждом подчиненном процессе дано целое число. Переслать эти числа в главный процесс, используя функции MPI_Send и MPI_Recv (стандартные блокирующие функции для передачи и приема сообщения), и вывести их в главном процессе. Полученные числа выводить в порядке возрастания рангов переславших их процессов.

MPI2Send2. В каждом подчиненном процессе дано вещественное число. Переслать эти числа в главный процесс, используя функции MPI_Bsend (посылка сообщения с буферизацией) и MPI_Recv, и вывести их в главном процессе. Полученные числа выводить в порядке убывания рангов переславших их процессов. Для задания буфера использовать функцию MPI_Buffer_attach.

MPI2Send3. В каждом подчиненном процессе даны четыре целых числа. Переслать эти числа в главный процесс, используя по одному вызову функции MPI_Send для каждого передающего процесса, и вывести их в главном процессе. Полученные числа выводить в порядке возрастания рангов переславших их процессов.

MPI2Send4. В каждом подчиненном процессе дано целое число N ($0 < N < 5$) и набор из N целых чисел. Переслать данные наборы в главный процесс, используя по одному вызову функции MPI_Bsend для каждого передающего процесса, и вывести наборы в главном процессе в порядке возрастания рангов переславших их процессов. Для определения размера пересланного набора использовать функцию MPI_Get_count.

MPI2Send5. В главном процессе дан набор вещественных чисел; количество чисел равно количеству подчиненных процессов. С помощью функции MPI_Send переслать по одному числу в каждый из подчиненных процессов (первое число в процесс 1, второе — в процесс 2, и т. д.) и вывести в подчиненных процессах полученные числа.

MPI2Send6. В главном процессе дан набор вещественных чисел; количество чисел равно количеству подчиненных процессов. С помощью функции MPI_Bsend переслать по одному числу в каждый из подчиненных процессов, перебирая процессы в обратном порядке (первое число в последний процесс, второе — в предпоследний процесс, и т. д.), и вывести в подчиненных процессах полученные числа.

MPI2Send7. В главном процессе дано целое число N и набор из N чисел; $K - 1 \le N < 10$, где K — количество процессов. С помощью функции MPI_Send переслать по одному числу их данного набора в процессы 1, 2, …, $K - 2$, а оставшиеся числа — в процесс $K - 1$, и вывести полученные числа. В процессе $K - 1$ для определения количества полученных чисел использовать функцию MPI_Get_count.

MPI2Send8. В каждом подчиненном процессе дано целое число, причем только для одного процесса это число отлично от нуля. Переслать ненулевое число в главный процесс и вывести в главном процессе полученное число и ранг процесса, переславшего это число. Для приема сообщения в главном процессе использовать функцию MPI_Recv с параметром MPI_ANY_SOURCE.

MPI2Send9. В каждом подчиненном процессе дано целое число N, причем для одного процесса это число больше нуля, а для остальных равно нулю. В процессе с ненулевым N дан также набор из N чисел. Переслать данный набор чисел в главный процесс и вывести в главном процессе полученные числа и ранг процесса, переславшего этот набор. При приеме сообщения использовать параметр MPI_ANY_SOURCE.

MPI2Send10. В каждом подчиненном процессе дано целое число N, в главном процессе дано целое число K (> 0), равное количеству тех подчиненных процессов, в которых даны положительные числа N. Переслать все положительные числа N в главный процесс и вывести в нем сумму полученных чисел. Для приема сообщений в главном процессе использовать функцию MPI_Recv с параметром MPI_ANY_SOURCE.

MPI2Send11. В каждом процессе дано вещественное число. Переслать число из главного процесса во все подчиненные процессы, а все числа из подчиненных процессов — в главный, и вывести в каждом процессе полученные числа (в главном процессе числа выводить в порядке возрастания рангов переславших их процессов). Для отправки сообщений использовать функцию MPI_Ssend.

Указание. Функция MPI_Ssend обеспечивает *синхронный режим* пересылки данных, при котором операция отправки сообщения будет завершена только после начала приема этого сообщения процессом-получателем. В случае пересылки данных в синхронном режиме возникает опасность *взаимных блокировок* (deadlocks) из-за неправильного порядка вызова функций отправки и получения сообщений.

MPI2Send12. В каждом процессе дано целое число. С помощью функций MPI_Ssend и MPI_Recv осуществить для всех процессов циклический сдвиг данных с шагом 1, переслав число из процесса 0 в процесс 1, из процесса 1 в процесс 2, …, из последнего процесса в процесс 0. В каждом процессе вывести полученное число.

Указание. См. указание к задаче MPI2Send11.

MPI2Send13. В каждом процессе дано целое число. С помощью функций MPI_Ssend и MPI_Recv осуществить для всех процессов циклический сдвиг данных с шагом -1, переслав число из процесса 1 в процесс 0, из процесса 2 в процесс 1, …, из процесса 0 в последний процесс. В каждом процессе вывести полученное число.

Указание. См. указание к задаче MPI2Send11.

MPI2Send14. В каждом процессе даны два целых числа. С помощью функций MPI_Ssend и MPI_Recv переслать первое число в предыдущий процесс, а второе — в последующий процесс (для процесса 0

считать предыдущим последний процесс, а для последнего процесса считать последующим процесс 0). В каждом процессе вывести числа, полученные от предыдущего и последующего процесса (в указанном порядке).

Указание. См. указание к задаче MPI2Send11.

MPI2Send15. В каждом процессе даны два числа: вещественное A и целое N, причем набор чисел N содержит все значения от 0 до $K-1$, где K — количество процессов. Используя функции MPI_Send и MPI_Recv (с параметром MPI_ANY_SOURCE), выполнить в каждом процессе пересылку числа A в процесс N и вывести полученное число, а также ранг процесса, из которого число было получено.

MPI2Send16. В каждом процессе дано целое число N, причем для одного процесса значение N равно 1, а для остальных равно 0. В процессе с $N = 1$ дан также набор из $K-1$ числа, где K — количество процессов. Переслать из этого процесса по одному из чисел данного набора в остальные процессы, перебирая ранги получателей в возрастающем порядке, и вывести в каждом из них полученное число.

MPI2Send17. В каждом процессе дан набор из $K-1$ целого числа, где K — количество процессов. Для каждого процесса переслать по одному из данных в нем чисел в остальные процессы, перебирая ранги процессов-получателей в возрастающем порядке, и вывести полученные числа в порядке возрастания рангов переславших их процессов.

MPI2Send18. Количество процессов — четное число. В каждом процессе дано целое число N ($0 < N < 5$) и набор из N чисел. С помощью функции MPI_Sendrecv выполнить обмен исходными наборами между парами процессов 0 и 1, 2 и 3, и т. д. В каждом процессе вывести полученный набор чисел.

MPI2Send19. В каждом процессе дано вещественное число. С помощью функции MPI_Sendrecv_replace поменять порядок исходных чисел на обратный (число из процесса 0 должно быть передано в последний процесс, число из процесса 1 — в предпоследний процесс, …, число из последнего процесса — в процесс 0). В каждом процессе вывести полученное число.

MPI2Send20. В каждом подчиненном процессе дано вещественное число A и его порядковый номер N (целое число); набор всех номеров N содержит все целые числа от 1 до $K-1$, где K — количество процессов. Переслать числа A в главный процесс и вывести их в порядке, соответствующем возрастанию их номеров N. Массивы не использовать; для передачи номера N указывать его в качестве параметра msgtag функции MPI_Send.

MPI2Send21. В каждом подчиненном процессе дано целое число L (≥ 0) и набор из L пар чисел (A, N), где A — вещественное число, а N — его порядковый номер. Все числа L в сумме равны $2K$, где K — количество процессов; набор номеров N, данных во всех процессах, содержит все целые числа от 1 до $2K$. Переслать числа A в главный процесс и вывести их в порядке, соответствующем возрастанию их номеров N. Для передачи номера N указывать его в качестве параметра msgtag функции MPI_Send.

MPI2Send22. В главном процессе дан набор пар чисел (T, A); количество пар равно числу подчиненных процессов. Число T — целое, равное 0 или 1. Число A — целое, если $T = 0$, и вещественное, если $T = 1$. Переслать по одному числу A в каждый из подчиненных процессов (первое число в процесс 1, второе — в процесс 2, и т. д.) и вывести полученные числа. Для передачи информации о типе пересланного числа указывать число T в качестве параметра msgtag функции MPI_Send, для получения этой информации использовать функцию MPI_Probe с параметром MPI_ANY_TAG.

Указание. Чтобы избежать дублирования кода, используйте вспомогательные *шаблонные функции* `template<typename T> void send(int t, int dest, MPI_Datatype d)` (для отправки данных) и `template<typename T> void recv(MPI_Datatype d)` (для получения данных). В качестве параметра t указывайте число, равное 0 или 1, в качестве параметра dest — ранг процесса-получателя.

MPI2Send23. В каждом подчиненном процессе даны два целых числа T, N и набор из N чисел. Число T равно 0 или 1. Набор содержит целые числа, если $T = 0$, и вещественные числа, если $T = 1$. Переслать исходные наборы в главный процесс и вывести полученные числа в порядке возрастания рангов переславших их процессов. Для передачи информации о типе пересланного числа указывать число T в качестве параметра msgtag функции MPI_Send, для получения этой информации использовать функцию MPI_Probe с параметром MPI_ANY_TAG.

Указание. Чтобы избежать дублирования кода, используйте вспомогательные *шаблонные функции* `template<typename T> void send(int t, MPI_Datatype d)` (для отправки данных) и `template<typename T> void recv(MPI_Datatype d, MPI_Status s)` (для получения данных). В качестве параметра t указывайте число, равное 0 или 1, в качестве параметра s — результат, возвращенный функцией MPI_Probe.

MPI2Send24. Количество процессов K является четным: $K = 2N$. В процессах четного ранга (0, 2, …, $K - 2$) дан набор из N вещественных чи-

сел, в процессах нечетного ранга (1, 3, …, $K-1$) — набор из N целых чисел. Используя функцию MPI_Sendrecv_replace, выполнить циклический сдвиг всех наборов вещественных чисел в направлении увеличения рангов процессов и циклический сдвиг всех наборов целых чисел в направлении уменьшения рангов (таким образом, вещественные наборы надо переслать из процесса 0 в процесс 2, из процесса 2 — в процесс 4, …, из процесса $K-2$ — в процесс 0; целочисленные наборы надо переслать из процесса $K-1$ в процесс $K-3$, из процесса $K-3$ в процесс $K-5$, …, из процесса 1 в процесс $K-1$). Вывести в каждом процессе полученные числа. Для определения рангов процессов-получателей использовать выражение, содержащее операцию % нахождения остатка от деления, в качестве ранга процесса-отправителя достаточно указывать константу MPI_ANY_SOURCE.

Указание. Чтобы избежать дублирования кода, используйте вспомогательную *шаблонную функцию* `template<typename T> void sendrecv(int rank, int size, MPI_Datatype d, int step)`, где параметр step указывает величину сдвига, равную 2 для наборов вещественных чисел и −2 для целочисленных наборов.

MPI2Send25. Количество процессов K является четным: $K = 2N$. В первой половине процессов дан набор целых чисел размера $R+1$, где R — ранг процесса ($R = 0, 1, \ldots, N-1$), во второй половине процессов дан набор вещественных чисел размера $2N\text{-}R$, где R — ранг процесса ($R = N, N+1, \ldots, 2N-1$). Используя функцию MPI_Sendrecv, переслать исходные наборы из каждой половины процессов в соответствующий процесс другой половины (в частности, набор из процесса 0 надо переслать в процесс N, из процесса 1 — в процесс $N+1$, из процесса N — в процесс 0, из процесса $2N-1$ — в процесс $N-1$). Вывести в каждом процессе полученные числа.

Указание. Чтобы избежать дублирования кода, используйте вспомогательную *шаблонную функцию* `template<typename T1, typename T2> void sendrecv(MPI_Datatype d1, int cnt1, int rank2, MPI_Datatype d2, int cnt2)`, где параметры d1 и cnt1 определяют характеристики процесса, вызвавшего функцию (тип элементов в наборе и их количество), а параметры rank2, d2, cnt2 — ранг и аналогичные характеристики процесса, с которым выполняется обмен данными.

2.2. Неблокирующая пересылка данных

MPI2Send26. В каждом процессе дано целое число N; во всех процессах, кроме одного, значение N равно 0, в некотором выделенном процессе значение N равно 1. В выделенном процессе также дан набор целых

чисел A размера $K-1$, где K — количество процессов. Не сохраняя набор A в массиве, переслать по одному элементу этого набора из выделенного процесса во все остальные процессы, перебирая их в порядке возрастания рангов, и вывести в каждом процессе полученное число. Для пересылки данных использовать требуемое количество вызовов функций MPI_Issend (посылка сообщения в синхронном неблокирующем режиме) и MPI_Wait в выделенном процессе и функцию MPI_Recv в остальных процессах. Дополнительно отобразить в разделе отладки продолжительность каждого вызова функции MPI_Wait в миллисекундах; для этого вызвать функцию MPI_Wtime до и после MPI_Wait и с помощью функции Show вывести разность возвращенных функцией MPI_Wtime значений, умноженную на 1000. Проверить, как изменится отладочная информация, если вместо функции MPI_Issend использовать функцию MPI_Isend (посылка сообщения в стандартном неблокирующем режиме).

MPI2Send27. В каждом процессе дано целое число N; в некотором выделенном процессе значение N равно -1, а в остальных процессах N является одинаковым и равно рангу R выделенного процесса. Во всех процессах, кроме выделенного, также дано вещественное число A. Переслать данные числа A в выделенный процесс и вывести их в порядке возрастания рангов процессов-отправителей. Для пересылки данных использовать требуемое количество вызовов функции MPI_Recv в выделенном процессе и функции MPI_Issend и MPI_Test в остальных процессах. Вызов функции MPI_Test повторять в цикле до тех пор, пока она не вернет ненулевой флаг, и отобразить в разделе отладки потребовавшееся количество итераций этого цикла, используя функцию Show. Проверить, как изменится отладочная информация, если вместо функции MPI_Issend использовать функцию MPI_Isend.

MPI2Send28. В каждом процессе дано целое число N; в некотором выделенном процессе значение N равно -1, а в остальных процессах N является одинаковым и равно рангу R выделенного процесса. Во всех процессах, кроме выделенного, также дано вещественное число A. Переслать данные числа A в выделенный процесс и вывести их в порядке убывания рангов процессов-отправителей. Для пересылки данных использовать требуемое количество вызовов функций MPI_Irecv (прием сообщения в неблокирующем режиме) и MPI_Test в выделенном процессе и функцию MPI_Ssend в остальных процессах. После каждого вызова функции MPI_Irecv организовать цикл, в котором вызывать функцию MPI_Test, пока она не вернет ненулевой флаг, и отображать в разделе отладки потребовавшееся количество итераций этого цикла, используя функцию Show. Проверить, как изменится отладочная ин-

формация, если вместо функции MPI_Ssend использовать функцию MPI_Send.

MPI2Send29. В каждом процессе дано целое число *N*; в некотором выделенном процессе значение *N* равно −1, а в остальных процессах *N* является одинаковым и равно рангу *R* выделенного процесса. Во всех процессах, кроме выделенного, также дано вещественное число *A*. Переслать данные числа *A* в выделенный процесс и вывести их сумму *S*. Для пересылки данных использовать требуемое количество вызовов функций MPI_Irecv и MPI_Waitany в выделенном процессе и функцию MPI_Ssend в остальных процессах. В выделенном процессе описать массив *Q* типа MPI_Request и организовать вызов функций MPI_Irecv в отдельном цикле, указывая для каждого вызова свой элемент массива *Q*. После выхода из этого цикла организовать второй цикл, в котором вызывать функцию MPI_Waitany и накапливать сумму *S*. Дополнительно на каждой итерации второго цикла отображать в разделе отладки следующие данные (используя по одному вызову функций Show и ShowLine): значение *A*, добавленное к сумме на данной итерации цикла, и ранг процесса, переславшего это значение.

MPI2Send30. В каждом процессе дано целое число *N*; во всех процессах, за исключением двух, значение *N* равно 0; в одном из оставшихся процессов (*процессе-отправителе*) значение *N* равно 1, в другом (*процессе-получателе*) значение *N* равно 2. В процессе-отправителе также дано целое число *R* — ранг процесса получателя и набор *A* целых чисел размера *K*, где *K* — количество процессов. Не сохраняя набор *A* в массиве, переслать все его элементы процессу-получателю и вывести их в том же порядке. Для пересылки данных использовать единственный вызов функции MPI_Ssend_init и требуемое количество вызовов функций MPI_Start и MPI_Wait в процессе-отправителе и единственный вызов функции MPI_Recv_init и требуемое количество вызовов функций MPI_Start и MPI_Wait в процессе-получателе. Дополнительно отобразить в разделе отладки продолжительность каждого вызова функции MPI_Wait в миллисекундах (как для процесса-отправителя, так и для процесса-получателя); для этого вызвать функцию MPI_Wtime до и после MPI_Wait и с помощью функции Show вывести разность возвращенных функцией MPI_Wtime значений, умноженную на 1000. Проверить, как изменится отладочная информация, если вместо функции MPI_Ssend_init использовать функцию MPI_Send_init.

MPI2Send31. В каждом процессе дано целое число *N*; в одном из процессов (*процессе-получателе*) значение *N* равно 2, в некоторых процессах (*процессах-отправителях*) значение *N* равно 1, в остальных процессах

значение *N* равно 0. В каждом из процессов-отправителей также дано целое число *R* — ранг процесса-получателя и набор *A* целых чисел размера *K*, где *K* — количество процессов. В процессе-получателе дано целое число *C* — количество процессов-отправителей. Переслать все наборы *A* процессу-получателю и вывести набор *S* суммарных значений элементов всех наборов *A* с одинаковыми индексами (в порядке возрастания индексов). Для пересылки данных использовать единственный вызов функции MPI_Ssend в процессах-отправителях. В процессе-получателе описать массив *Q* типа MPI_Request и организовать вызов функций MPI_Recv_init в отдельном цикле, указывая для каждого вызова свой элемент массива *Q*. После выхода из этого цикла выполнить единственный вызов функции MPI_Startall и организовать второй цикл, в котором вызывать функцию MPI_Waitany и накапливать суммарные значения в массиве *S*. Дополнительно на каждой итерации второго цикла отображать в разделе отладки следующие данные (используя два вызова функции Show и один вызов ShowLine): продолжительность каждого вызова функции MPI_Waitany в миллисекундах, значение третьего параметра index, возвращенное функцией MPI_Waitany, и ранг процесса-отправителя, соответствующий параметру index. Для нахождения продолжительности вызвать функцию MPI_Wtime до и после MPI_Waitany и вычислить разность возвращенных функцией MPI_Wtime значений, умножив ее на 1000. Для нахождения ранга процесса-отправителя проанализировать значение последнего параметра (типа MPI_Status), возвращенного функцией MPI_Waitany.

MPI2Send32. В каждом процессе дано целое число *N*; в одном из процессов (*процессе-отправителе*) значение *N* равно 1, в некоторых процессах (*процессах-получателях*) значение *N* равно 2, в остальных процессах значение *N* равно 0. В процессе-отправителе также дано вещественное число *A*, целое число *C* — количество процессов-получателей и набор целых чисел *R* размера *C*, содержащий ранги процессов-получателей. Переслать число *A* всем процессам-получателям и вывести его в каждом из этих процессов. Для пересылки данных использовать функцию MPI_Recv в процессах-получателях. В процессе-отправителе описать массив *Q* типа MPI_Request и организовать вызов функций MPI_Ssend_init в отдельном цикле, указывая для каждого вызова свой элемент массива *Q*. После выхода из этого цикла выполнить единственный вызов функции MPI_Startall и организовать второй цикл, в котором вызывать функцию MPI_Testany во вложенном цикле, пока она не вернет ненулевой флаг. Дополнительно на каждой итерации второго цикла отображать в разделе отладки следующие данные (используя по одному вызову функций Show и ShowLine): значение

третьего параметра index, возвращенное функцией MPI_Testany (в ситуации, когда она вернула ненулевой флаг), и количество произведенных вызовов функции MPI_Testany (т. е. количество итераций вложенного цикла). Проверить, как изменится отладочная информация, если вместо функции MPI_Ssend_init использовать функцию MPI_Send_init.

3. Коллективные взаимодействия

3.1. Коллективная пересылка данных

MPI3Coll1. В главном процессе дано целое число. Используя функцию MPI_Bcast, переслать это число во все подчиненные процессы и вывести в них полученное число.

MPI3Coll2. В главном процессе дан набор из 5 чисел. Используя функцию MPI_Bcast, переслать этот набор во все подчиненные процессы и вывести в них полученные числа в том же порядке.

MPI3Coll3. В каждом процессе дано вещественное число. Используя функцию MPI_Gather, переслать эти числа в главный процесс и вывести их в порядке возрастания рангов переславших их процессов (первым вывести число, данное в главном процессе).

MPI3Coll4. В каждом процессе дан набор из 5 целых чисел. Используя функцию MPI_Gather, переслать эти наборы в главный процесс и вывести их в порядке возрастания рангов переславших их процессов (первым вывести набор чисел, данный в главном процессе).

MPI3Coll5. В каждом процессе дан набор из $R + 2$ целых чисел, где число R равно рангу процесса (в процессе 0 даны 2 числа, в процессе 1 даны 3 числа, и т. д.). Используя функцию MPI_Gatherv, переслать эти наборы в главный процесс и вывести полученные наборы в порядке возрастания рангов переславших их процессов (первым вывести набор, данный в главном процессе).

MPI3Coll6. В главном процессе дан набор из K чисел, где K — количество процессов. Используя функцию MPI_Scatter, переслать по одному числу в каждый процесс (включая главный) и вывести в каждом процессе полученное число.

MPI3Coll7. В главном процессе дан набор из $3K$ чисел, где K — количество процессов. Используя функцию MPI_Scatter, переслать по 3 числа в каждый процесс (включая главный) и вывести в каждом процессе полученные числа.

MPI3Coll8. В главном процессе дан набор из K чисел, где K — количество процессов. Не меняя порядок расположения чисел в исходном наборе

и используя функцию MPI_Scatterv, переслать по одному числу в каждый процесс; при этом первое число надо переслать в процесс $K - 1$, второе число — в процесс $K - 2$, …, последнее число — в процесс 0. Вывести в каждом процессе полученное число.

MPI3Coll9. В главном процессе дан набор из $K(K + 3)/2$ целых чисел, где K — количество процессов. Используя функцию MPI_Scatterv, переслать в каждый процесс часть чисел из данного набора; при этом в процесс ранга R надо переслать $R + 2$ очередных числа (в процесс 0 — первые два числа, в процесс 1 — следующие три числа, и т. д.). В каждом процессе вывести полученные числа.

MPI3Coll10. В главном процессе дан набор из $K + 2$ чисел, где K — количество процессов. Используя функцию MPI_Scatterv, переслать в каждый процесс три числа из данного набора; при этом в процесс ранга R должны быть пересланы числа с номерами от $R + 1$ до $R + 3$ (в процесс 0 — первые три числа, в процесс 1 — числа со второго по четвертое, и т. д.). В каждом процессе вывести полученные числа.

MPI3Coll11. В каждом процессе дано вещественное число. Используя функцию MPI_Allgather, переслать эти числа во все процессы и вывести их в каждом процессе в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

MPI3Coll12. В каждом процессе даны четыре целых числа. Используя функцию MPI_Allgather, переслать эти числа во все процессы и вывести их в каждом процессе в порядке возрастания рангов переславших их процессов (включая числа, полученные из этого же процесса).

MPI3Coll13. В каждом процессе дан набор из $R + 2$ целых чисел, где число R равно рангу процесса (в процессе 0 даны 2 числа, в процессе 1 даны 3 числа, и т. д.). Используя функцию MPI_Allgatherv, переслать эти наборы во все процессы и вывести их в порядке возрастания рангов переславших их процессов (включая числа, полученные из этого же процесса).

MPI3Coll14. В каждом процессе дан набор из K чисел, где K — количество процессов. Используя функцию MPI_Alltoall, переслать в каждый процесс по одному числу из всех наборов: в процесс 0 — первые числа из наборов, в процесс 1 — вторые числа, и т. д. В каждом процессе вывести числа в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

MPI3Coll15. В каждом процессе дан набор из $3K$ целых чисел, где K — количество процессов. Используя функцию MPI_Alltoall, переслать в каждый процесс три очередных числа из каждого набора (в процесс 0 — первые три числа, в процесс 1 — следующие три числа, и т. д.). В каждом процессе вывести числа в порядке возрастания рангов пере-

славших их процессов (включая числа, полученные из этого же процесса).

MPI3Coll16. В каждом процессе дан набор из $K(K + 1)/2$ целых чисел, где K — количество процессов. Используя функцию MPI_Alltoallv, переслать в каждый процесс часть чисел из каждого набора; при этом в процесс R должно быть переслано $R + 1$ очередное число (в процесс 0 — первое число каждого набора, в процесс 1 — следующие два числа, и т. д.). В каждом процессе вывести полученные числа.

MPI3Coll17. В каждом процессе дан набор из $K + 1$ числа, где K — количество процессов. Используя функцию MPI_Alltoallv, переслать в каждый процесс два числа из каждого набора; при этом в процесс 0 надо переслать первые два числа, в процесс 1 — второе и третье число, …, в последний процесс — последние два числа каждого набора. В каждом процессе вывести полученные числа.

MPI3Coll18. В каждом процессе дан набор из $K + 1$ числа, где K — количество процессов. Используя функцию MPI_Alltoallv, переслать в каждый процесс два числа из каждого набора; при этом в процесс 0 надо переслать последние два числа, в процесс 1 — два числа, предшествующих последнему, …, в последний процесс — первые два числа каждого набора. В каждом процессе вывести полученные числа.

3.2. Коллективные операции редукции

MPI3Coll19. В каждом процессе дан набор из $K + 5$ целых чисел, где K — количество процессов. Используя функцию MPI_Reduce для операции MPI_SUM, просуммировать элементы данных наборов с одним и тем же порядковым номером и вывести полученные суммы в главном процессе.

MPI3Coll20. В каждом процессе дан набор из $K + 5$ чисел, где K — количество процессов. Используя функцию MPI_Reduce для операции MPI_MIN, найти минимальное значение среди элементов данных наборов с одним и тем же порядковым номером и вывести полученные минимумы в главном процессе.

MPI3Coll21. В каждом процессе дан набор из $K + 5$ целых чисел, где K — количество процессов. Используя функцию MPI_Reduce для операции MPI_MAXLOC, найти максимальное значение среди элементов данных наборов с одним и тем же порядковым номером и ранг процесса, содержащего это максимальное значение. Вывести в главном процессе вначале все максимумы, а затем — ранги содержащих их процессов.

MPI3Coll22. В каждом процессе дан набор из $K + 5$ чисел, где K — количество процессов. Используя функцию MPI_Allreduce для операции

MPI_PROD, перемножить элементы данных наборов с одним и тем же порядковым номером и вывести полученные произведения во всех процессах.

MPI3Coll23. В каждом процессе дан набор из $K + 5$ чисел, где K — количество процессов. Используя функцию MPI_Allreduce для операции MPI_MINLOC, найти минимальное значение среди элементов данных наборов с одним и тем же порядковым номером и ранг процесса, содержащего минимальное значение. Вывести в главном процессе минимумы, а в остальных процессах — ранги процессов, содержащих эти минимумы.

MPI3Coll24. В каждом процессе дан набор из K целых чисел, где K — количество процессов. Используя функцию MPI_Reduce_scatter, просуммировать элементы данных наборов с одним и тем же порядковым номером, переслать по одной из полученных сумм в каждый процесс (первую сумму — в процесс 0, вторую — в процесс 1, и т. д.) и вывести в каждом процессе полученную сумму.

MPI3Coll25. В каждом процессе дан набор из $2K$ чисел, где K — количество процессов. Используя функцию MPI_Reduce_scatter, найти максимумы среди элементов этих наборов с одним и тем же порядковым номером, переслать по два найденных максимума в каждый процесс (первые два максимума — в процесс 0, следующие два — в процесс 1, и т. д.) и вывести в каждом процессе полученные данные.

MPI3Coll26. В каждом процессе дан набор из $K(K + 3)/2$ целых чисел, где K — количество процессов. Используя функцию MPI_Reduce_scatter, найти минимальные значения среди элементов этих наборов с одним и тем же порядковым номером и переслать первые два минимума в процесс 0, следующие три — в процесс 1, …, последние $K + 1$ минимумов — в процесс $K - 1$. Вывести в каждом процессе полученные данные.

MPI3Coll27. В каждом процессе дан набор из $K + 5$ чисел, где K — количество процессов. Используя функцию MPI_Scan, найти в процессе ранга R ($R = 0, 1, \ldots, K - 1$) произведения элементов с одним и тем же порядковым номером для наборов, данных в процессах с рангами от 0 до R, и вывести найденные произведения (при этом в процессе $K - 1$ будут выведены произведения элементов из всех наборов).

MPI3Coll28. В каждом процессе дан набор из $K + 5$ целых чисел, где K — количество процессов. Используя функцию MPI_Scan, найти в процессе ранга R ($R = 0, \ldots, K - 1$) максимальные значения среди элементов с одним и тем же порядковым номером для наборов, данных в процессах с рангами от 0 до R, и вывести в каждом процессе найденные максимумы.

4. Производные типы и упаковка данных

4.1. Использование простейших производных типов

MPI4Type1. В главном процессе дана $K - 1$ тройка целых чисел, где K — количество процессов. Используя производный тип, содержащий три целых числа, и одну коллективную операцию пересылки данных, переслать все данные из главного процесса в подчиненные и вывести их в подчиненных процессах в том же порядке.

MPI4Type2. В главном процессе дана $K - 1$ тройка целых чисел, где K — количество процессов. Используя производный тип, содержащий три целых числа, и одну коллективную операцию пересылки данных, переслать по одной тройке чисел в каждый из подчиненных процессов и вывести их в подчиненных процессах в том же порядке.

MPI4Type3. В каждом подчиненном процессе дана тройка целых чисел. Используя производный тип, содержащий три целых числа, и одну коллективную операцию пересылки данных, переслать числа из подчиненных процессов в главный и вывести полученные числа в порядке возрастания рангов переславших их процессов.

MPI4Type4. В главном процессе дана $K - 1$ тройка чисел, где K — количество процессов, причем первые два числа каждой тройки являются целыми, а третье число — вещественным. Используя производный тип, содержащий три числа (два целых и одно вещественное), переслать числа из главного процесса в подчиненные и вывести их в подчиненных процессах в том же порядке.

MPI4Type5. В главном процессе дана $K - 1$ тройка чисел, где K — количество процессов, причем первое и третье число каждой тройки являются целыми, а второе число — вещественным. Используя производный тип, содержащий три числа (целое, вещественное, целое), переслать по одной тройке чисел в каждый из подчиненных процессов и вывести их в подчиненных процессах в том же порядке.

MPI4Type6. В каждом подчиненном процессе даны три числа: одно вещественное и два целых. Используя производный тип, содержащий три числа (одно вещественное и два целых), переслать числа из подчиненных процессов в главный и вывести полученные числа в порядке возрастания рангов переславших их процессов.

MPI4Type7. В каждом процессе даны три числа: первое и третье являются целыми, а второе — вещественным. Используя производный тип, содержащий три числа (целое, вещественное, целое), переслать данные из каждого процесса во все процессы и вывести в каждом процессе

полученные числа в порядке возрастания рангов переславших их процессов (включая числа, полученные из этого же процесса).

MPI4Type8. В каждом подчиненном процессе даны R троек чисел, где R — ранг процесса. Два первых числа в каждой тройке являются целыми, а последнее — вещественным. Используя производный тип, содержащий три числа (два целых и одно вещественное), переслать числа из подчиненных процессов в главный и вывести полученные числа в порядке возрастания рангов переславших их процессов.

4.2. Пересылка упакованных данных

MPI4Type9. В главном процессе даны два набора: первый содержит K целых, а второй K вещественных чисел, где K — количество процессов. Используя функции упаковки MPI_Pack и MPI_Unpack и одну коллективную операцию пересылки данных, переслать все данные из главного процесса в подчиненные и вывести их в подчиненных процессах в том же порядке.

MPI4Type10. В главном процессе дана $K - 1$ тройка чисел, где K — количество процессов, причем первое и третье число каждой тройки является целым, а второе число — вещественным. Используя функции упаковки и одну коллективную операцию пересылки данных, переслать по одной тройке чисел в подчиненные процессы и вывести их в подчиненных процессах в том же порядке.

MPI4Type11. В главном процессе дана $K - 1$ тройка чисел, где K — количество процессов, причем первые два числа каждой тройки являются целыми, а третье число — вещественным. Используя функции упаковки и одну коллективную операцию пересылки данных, переслать все данные из главного процесса в подчиненные и вывести их в подчиненных процессах в том же порядке.

MPI4Type12. В каждом подчиненном процессе даны три числа: два целых и одно вещественное. Используя функции упаковки и одну коллективную операцию пересылки данных, переслать числа из подчиненных процессов в главный и вывести полученные числа в порядке возрастания рангов переславших их процессов.

MPI4Type13. В каждом подчиненном процессе дан набор из одного вещественного и R целых чисел, где значение R равно рангу процесса (в процессе 1 дано одно целое число, в процессе 2 — два целых числа, и т. д.). Используя функции упаковки и одну функцию передачи и приема, переслать все данные в главный процесс и вывести эти данные в порядке возрастания рангов переславших их процессов.

4.3. Более сложные виды производных типов

MPI4Type14. В главном процессе даны два набора целых чисел: A размера $3K$ и N размера K, где K — количество подчиненных процессов. Считая, что элементы наборов нумеруются от 1, переслать и вывести в каждом подчиненном процессе ранга R (R = 1, 2, …, K) N_R элементов из набора A, начиная с элемента A_R и перебирая их через один (например, если N_2 равно 3, то в процесс ранга 2 надо переслать элементы A_2, A_4, A_6). Использовать для пересылки каждого набора элементов по одному вызову функций MPI_Send, MPI_Probe и MPI_Recv, причем функция MPI_Recv должна возвращать массив, содержащий только те элементы, которые требуется вывести. Для этого в главном процессе определить новый тип, содержащий единственный целочисленный элемент и дополнительный конечный *пустой промежуток*, равный размеру элемента целого типа. Использовать в функции MPI_Send исходный массив A с требуемым смещением, указав в качестве второго параметра количество элементов, равное N_R, а в качестве третьего параметра — новый тип. В функции MPI_Recv использовать целочисленный массив размера N_R и тип MPI_INT. Для определения количества N_R переданных элементов использовать в подчиненных процессах функцию MPI_Get_count.

Указание. Для задания завершающего пустого промежутка при определении нового типа в MPI-2 следует использовать функцию MPI_Type_create_resized (в данном случае эту функцию надо применить к типу MPI_INT). В MPI-1 надо использовать *метку нулевого размера* типа MPI_UB совместно с функцией MPI_Type_struct (в стандарте MPI-2 тип MPI_UB объявлен устаревшим).

MPI4Type15. В главном процессе дана вещественная квадратная матрица порядка K, где K — количество подчиненных процессов (матрица должна храниться в одномерном массиве A, в котором элементы матрицы располагаются по строкам). Считая, что столбцы матрицы нумеруются от 1, переслать и вывести в каждом подчиненном процессе ранга R (R = 1, 2, …, K) элементы столбца матрицы с номером R. Использовать по одному вызову функций MPI_Send и MPI_Recv, причем функция MPI_Recv должна возвращать массив, содержащий только те элементы, которые требуется вывести. Для этого в главном процессе определить новый тип, содержащий единственный вещественный элемент и дополнительный конечный пустой промежуток требуемого размера. Использовать в функции MPI_Send массив A с требуемым смещением, указав в качестве второго параметра число K (т. е. количество элементов в каждом столбце), а в качестве третьего параметра

— новый тип. В функции MPI_Recv использовать вещественный массив размера *K* и тип MPI_DOUBLE.

Указание. См. указание к задаче MPI4Type14.

MPI4Type16. В каждом из подчиненных процессов дан столбец вещественной квадратной матрицы порядка *K*, где *K* — количество подчиненных процессов, причем в процессе ранга *R* (*R* = 1, …, *K*) содержится столбец с номером *R*, если считать, что столбцы нумеруются от 1. Переслать все столбцы матрицы в главный процесс, разместив их в одномерном массиве *A*, в котором элементы матрицы хранятся по строкам, и вывести полученный массив *A*. Для пересылки каждого столбца использовать по одному вызову функций MPI_Send и MPI_Recv, причем функция MPI_Recv должна в качестве первого параметра содержать массив *A* (с требуемым смещением), а в качестве второго параметра — число 1. Для этого в главном процессе определить новый тип, содержащий *K* вещественных элементов, после каждого из которых следует пустой промежуток требуемого размера. Определение нового типа провести в два этапа: на первом создать вспомогательный тип, состоящий из одного вещественного числа и требуемого конечного пустого промежутка (см. указание к MPI4Type14), на втором — создать на его основе итоговый тип с помощью функции MPI_Type_contiguous, который использовать в качестве третьего параметра функции MPI_Recv (функцию MPI_Type_commit достаточно вызвать только для итогового типа). В функции MPI_Send использовать вещественный массив размера *K* и тип MPI_DOUBLE.

MPI4Type17. Количество подчиненных процессов *K* кратно 3 и не превосходит 9. В каждом процессе дано целое число *N*; все числа *N* одинаковы и лежат в диапазоне от 3 до 5. Кроме того, в каждом подчиненном процессе дана целочисленная квадратная матрица порядка *N* (*блок*), которую следует прочесть в одномерный массив *B* по строкам. Переслать все массивы *B* в главный процесс и составить из них *блочную матрицу* размера (*K*/3) × 3 (размер указан в блоках), располагая блоки по строкам (первая строка блоков должна содержать блоки, полученные из процессов ранга 1, 2, 3, вторая — из процессов ранга 4, 5, 6, и т. д.). Блочная матрица должна храниться по строкам в одномерном массиве *A*. Использовать для пересылки каждого блока *B* по одному вызову функций MPI_Send и MPI_Recv, причем функция MPI_Recv должна в качестве первого параметра содержать массив *A* (с требуемым смещением), а в качестве второго параметра — число 1. Для этого в главном процессе определить новый тип, содержащий *N* наборов элементов, причем каждый набор содержит *N* целых чисел, а между наборами располагается пустой промежуток требуемого размера (при определении нового типа использовать функцию MPI_Type_vector).

Указать созданный тип в качестве третьего параметра функции MPI_Recv. В функции MPI_Send использовать целочисленный массив *B* (размера $N \cdot N$) и тип MPI_INT.

MPI4Type18. Количество подчиненных процессов *K* кратно 3 и не превосходит 9. В главном процессе дано целое число *N*, лежащее в диапазоне от 3 до 5, и целочисленная блочная матрица порядка $(K/3) \times 3$ (размер указан в блоках). Каждый блок представляет собой *нижнетреугольную* квадратную матрицу порядка *N*. Блочную матрицу следует прочесть в одномерный массив *A* по строкам. Переслать в каждый подчиненный процесс ненулевую часть соответствующего блока исходной матрицы, перебирая блоки по строкам (блоки из первой строки должны пересылаться в процессы ранга 1, 2, 3, блоки из второй строки — в процессы ранга 4, 5, 6, и т. д.), и вывести в каждом подчиненном процессе полученные данные (по строкам). Использовать для пересылки каждого блока по одному вызову функций MPI_Send, MPI_Probe и MPI_Recv, причем функция MPI_Send должна в качестве первого параметра содержать массив *A* (с требуемым смещением), а в качестве второго параметра — число 1. Для этого в главном процессе определить новый тип, содержащий *N* наборов элементов, причем каждый набор содержит ненулевую часть очередной строки блока (первый набор состоит из одного элемента, второй из двух, и т. д.), а между наборами располагается пустой промежуток требуемого размера (при определении нового типа использовать функцию MPI_Type_indexed). Указать созданный тип в качестве третьего параметра функции MPI_Send. В функции MPI_Recv использовать целочисленный массив *B* (содержащий ненулевую часть полученного блока) и тип MPI_INT. Для определения количества переданных элементов использовать в подчиненных процессах функцию MPI_Get_count.

MPI4Type19. Количество подчиненных процессов *K* кратно 3 и не превосходит 9. В каждом процессе дано целое число *N*; все числа *N* одинаковы и лежат в диапазоне от 3 до 5. Кроме того, в каждом подчиненном процессе дано целое число *P* и ненулевые элементы целочисленной квадратной матрицы порядка *N* (*Z-блока*), которую следует прочесть в одномерный массив *B* по строкам. Ненулевые элементы располагаются в *Z*-блоке в форме символа «Z», т. е. занимают первую и последнюю строку, а также побочную диагональ. Определить в главном процессе нулевую целочисленную прямоугольную матрицу размера $N \cdot (K/3) \times 3N$, создав для этого одномерный массив *A* и обнулив его элементы (которые должны храниться в массиве *A* по строкам). Перебирая подчиненные процессы в порядке возрастания их рангов, переслать из каждого подчиненного процесса данную в нем ненулевую часть *Z*-блока и записать полученный блок в массив *A*, начиная с эле-

мента массива *A* с индексом *P* (позиции *Z*-блоков могут перекрываться; в этом случае элементы блоков, полученных из процессов большего ранга, будут заменять некоторые из элементов ранее записанных блоков). Использовать для пересылки ненулевой части каждого *Z*-блока по одному вызову функций MPI_Send и MPI_Recv, причем функция MPI_Recv должна в качестве первого параметра содержать массив *A* (с требуемым смещением), а в качестве второго параметра — число 1. Для этого в главном процессе определить новый тип, содержащий *N* наборов элементов, причем первый и последний набор содержат по *N* целых чисел, остальные наборы — по одному числу, а между наборами располагается пустой промежуток требуемого размера (при определении нового типа использовать функцию MPI_Type_indexed). Указать созданный тип в качестве третьего параметра функции MPI_Recv. В функции MPI_Send использовать целочисленный массив *B*, содержащий ненулевую часть *Z*-блока (по строкам), и тип MPI_INT.

Указание. Для передачи в главный процесс позиции *P* вставки *Z*-блока используйте параметр msgtag; для этого в подчиненных процессах задавайте значение *P* в качестве параметра msgtag функции MPI_Send, а в главном процессе до вызова функции MPI_Recv вызывайте функцию MPI_Probe с параметром MPI_ANY_TAG и анализируйте возвращенный ею параметр типа MPI_Status.

MPI4Type20. Количество подчиненных процессов *K* кратно 3 и не превосходит 9. В каждом процессе дано целое число *N*; все числа *N* одинаковы и лежат в диапазоне от 3 до 5. Кроме того, в каждом подчиненном процессе дано целое число *P* и ненулевые элементы целочисленной квадратной матрицы порядка *N* (*U-блока*), которую следует прочесть в одномерный массив *B* по строкам. Ненулевые элементы располагаются в *U*-блоке в форме символа «U», т. е. занимают первый и последний столбец, а также последнюю строку. Определить в главном процессе нулевую целочисленную прямоугольную матрицу размера $N \cdot (K/3) \times 3N$, создав для этого одномерный массив *A* и обнулив его элементы (которые должны храниться в массиве *A* по строкам). Перебирая подчиненные процессы в порядке возрастания их рангов, переслать из каждого подчиненного процесса данную в нем ненулевую часть *U*-блока и записать полученный блок в массив *A*, начиная с элемента массива *A* с индексом *P* (позиции *U*-блоков могут перекрываться; в этом случае элементы блоков, полученных из процессов большего ранга, будут заменять некоторые из элементов ранее записанных блоков). Использовать для пересылки ненулевой части каждого *U*-блока по одному вызову функций MPI_Send и MPI_Recv, причем функция MPI_Recv должна в качестве первого параметра содержать

массив *A* (с требуемым смещением), а в качестве второго параметра — число 1. Для этого в главном процессе определить новый тип, содержащий требуемое количество наборов элементов с пустыми промежутками между ними (при определении нового типа использовать функцию MPI_Type_indexed). Указать созданный тип в качестве третьего параметра функции MPI_Recv. В функции MPI_Send использовать целочисленный массив *B*, содержащий ненулевую часть *U*-блока (по строкам), и тип MPI_INT.

Указание. См. указание к задаче MPI4Type19.

4.4. Коллективная функция MPI_Alltoallw (MPI-2)

MPI4Type21. Решить задачу MPI4Type15, используя для пересылки данных вместо функций MPI_Send и MPI_Recv одну коллективную операцию.

Указание. Использовать функции группы Scatter не удастся, так как значения смещений для рассылаемых элементов (столбцов матрицы) требуется указывать не в элементах, а в *байтах*. Поэтому подходящим вариантом будет введенная в MPI-2 функция MPI_Alltoallw, позволяющая наиболее гибким образом настраивать вариант коллективной пересылки. В данном случае с ее помощью следует реализовать вариант рассылки вида Scatter (при этом большинство параметров-массивов, используемых в этой функции, необходимо по-разному определять в главном и подчиненных процессах).

MPI4Type22. Решить задачу MPI4Type16, используя для пересылки данных вместо функций MPI_Send и MPI_Recv одну коллективную операцию.

Указание. См. указание к задаче MPI4Type21. В данном случае с помощью функции MPI_Alltoallw следует реализовать вариант рассылки вида Gather.

5. Группы процессов и коммуникаторы

5.1. Создание новых коммуникаторов

MPI5Comm1. В главном процессе дан набор из *K* целых чисел, где *K* — количество процессов четного ранга (0, 2, …). С помощью функций MPI_Comm_group, MPI_Group_incl и MPI_Comm_create создать новый коммуникатор, включающий процессы четного ранга. Используя одну коллективную операцию пересылки данных для созданного коммуникатора, переслать по одному исходному числу в каждый процесс четного ранга (включая главный) и вывести полученные числа.

MPI5Comm2. В каждом процессе нечетного ранга (1, 3, …) даны два вещественных числа. С помощью функций MPI_Comm_group, MPI_Group_excl и MPI_Comm_create создать новый коммуникатор, включающий процессы нечетного ранга. Используя одну коллективную операцию пересылки данных для созданного коммуникатора, переслать исходные числа во все процессы нечетного ранга и вывести эти числа в порядке возрастания рангов переславших их процессов (включая числа, полученные из этого же процесса).

MPI5Comm3. В каждом процессе, ранг которого делится на 3 (включая главный процесс), даны три целых числа. С помощью функции MPI_Comm_split создать новый коммуникатор, включающий процессы, ранг которых делится на 3. Используя одну коллективную операцию пересылки данных для созданного коммуникатора, переслать исходные числа в главный процесс и вывести эти числа в порядке возрастания рангов переславших их процессов (включая числа, полученные из главного процесса).

Указание. При вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

MPI5Comm4. В каждом процессе четного ранга (включая главный процесс) дан набор из трех элементов — вещественных чисел. Используя новый коммуникатор и одну коллективную операцию редукции, найти минимальные значения среди элементов исходных наборов с одним и тем же порядковым номером и вывести найденные минимумы в главном процессе. Новый коммуникатор создать с помощью функции MPI_Comm_split.

Указание. См. указание к задаче MPI5Comm3.

MPI5Comm5. В каждом процессе дано вещественное число. Используя функцию MPI_Comm_split и одну коллективную операцию редукции, найти максимальное из чисел, данных в процессах с четным рангом (включая главный процесс), и минимальное из чисел, данных в процессах с нечетным рангом. Найденный максимум вывести в процессе 0, а найденный минимум — в процессе 1.

Указание. Программа должна содержать единственный вызов функции MPI_Comm_split, создающий оба требуемых коммуникатора (каждый в соответствующей группе процессов).

MPI5Comm6. В главном процессе дано целое число K и набор из K вещественных чисел, в каждом подчиненном процессе дано целое число N, которое может принимать два значения: 0 и 1 (количество подчиненных процессов с $N = 1$ равно K). Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать по од-

ному вещественному числу из главного процесса в каждый подчиненный процесс с $N = 1$ и вывести в этих подчиненных процессах полученные числа.

Указание. См. указание к задаче MPI5Comm3.

MPI5Comm7. В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с $N = 1$). Кроме того, в каждом процессе с $N = 1$ дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать числа A в первый из процессов с $N = 1$ и вывести их в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

Указание. См. указание к задаче MPI5Comm3.

MPI5Comm8. В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с $N = 1$). Кроме того, в каждом процессе с $N = 1$ дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать числа A в последний из процессов с $N = 1$ и вывести их в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

Указание. См. указание к задаче MPI5Comm3.

MPI5Comm9. В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с $N = 1$). Кроме того, в каждом процессе с $N = 1$ дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать числа A во все процессы с $N = 1$ и вывести их в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

Указание. См. указание к задаче MPI5Comm3.

MPI5Comm10. В каждом процессе дано целое число N, которое может принимать два значения: 1 и 2 (имеется хотя бы один процесс с каждым из возможных значений). Кроме того, в каждом процессе дано целое число A. Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать числа A, данные в процессах с $N = 1$, во все процессы с $N = 1$, а числа A, данные в процессах с $N = 2$, во все процессы с $N = 2$. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

Указание. См. указание к задаче MPI5Comm5.

MPI5Comm11. В каждом процессе дано целое число N, которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с $N = 1$). Кроме того, в каждом процессе с $N = 1$ дано вещественное число A.

Используя функцию MPI_Comm_split и одну коллективную операцию редукции, найти сумму всех исходных чисел A и вывести ее во всех процессах с $N = 1$.

Указание. См. указание к задаче MPI5Comm3.

MPI5Comm12. В каждом процессе дано целое число N, которое может принимать два значения: 1 и 2 (имеется хотя бы один процесс с каждым из возможных значений). Кроме того, в каждом процессе дано вещественное число A. Используя функцию MPI_Comm_split и одну коллективную операцию редукции, найти минимальное значение среди чисел A, которые даны в процессах с $N = 1$, и максимальное значение среди чисел A, которые даны в процессах с $N = 2$. Найденный минимум вывести в процессах с $N = 1$, а найденный максимум — в процессах с $N = 2$.

Указание. См. указание к задаче MPI5Comm5.

5.2. Виртуальные топологии

MPI5Comm13. В главном процессе дано целое число N (> 1), причем известно, что количество процессов K делится на N. Переслать число N во все процессы, после чего, используя функцию MPI_Cart_create, определить для всех процессов декартову топологию в виде двумерной решетки — матрицы размера $N \times K/N$ (порядок нумерации процессов оставить прежним). Используя функцию MPI_Cart_coords, вывести для каждого процесса его координаты в созданной топологии.

MPI5Comm14. В главном процессе дано целое число N (> 1), не превосходящее количества процессов K. Переслать число N во все процессы, после чего определить декартову топологию для начальной части процессов в виде двумерной решетки — матрицы размера $N \times K/N$ (символ «/» обозначает операцию деления нацело, порядок нумерации процессов следует оставить прежним). Для каждого процесса, включенного в декартову топологию, вывести его координаты в этой топологии.

MPI5Comm15. Число процессов K является четным: $K = 2N$, $N > 1$. В процессах 0 и N дано по одному вещественному числу A. Определить для всех процессов декартову топологию в виде матрицы размера $2 \times N$, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на две одномерные строки (при этом процессы 0 и N будут главными процессами в полученных строках). Используя одну коллективную операцию пересылки данных, переслать число A из главного процесса каждой строки во все процессы этой же строки и вывести полученное число в каждом процессе (включая процессы 0 и N).

MPI5Comm16. Число процессов K является четным: $K = 2N$, $N > 1$. В процессах 0 и 1 дано по одному вещественному числу A. Определить для всех процессов декартову топологию в виде матрицы размера $N \times 2$, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на два одномерных столбца (при этом процессы 0 и 1 будут главными процессами в полученных столбцах). Используя одну коллективную операцию пересылки данных, переслать число A из главного процесса каждого столбца во все процессы этого же столбца и вывести полученное число в каждом процессе (включая процессы 0 и 1).

MPI5Comm17. Число процессов K кратно трем: $K = 3N$, $N > 1$. В процессах 0, N и $2N$ дано по N целых чисел. Определить для всех процессов декартову топологию в виде матрицы размера $3 \times N$, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на три одномерные строки (при этом процессы 0, N и $2N$ будут главными процессами в полученных строках). Используя одну коллективную операцию пересылки данных, переслать по одному исходному числу из главного процесса каждой строки во все процессы этой же строки и вывести полученное число в каждом процессе (включая процессы 0, N и $2N$).

MPI5Comm18. Число процессов K кратно трем: $K = 3N$, $N > 1$. В процессах 0, 1 и 2 дано по N целых чисел. Определить для всех процессов декартову топологию в виде матрицы размера $N \times 3$, после чего, используя функцию MPI_Cart_sub, расщепить матрицу процессов на три одномерных столбца (при этом процессы 0, 1 и 2 будут главными процессами в полученных столбцах). Используя одну коллективную операцию пересылки данных, переслать по одному исходному числу из главного процесса каждого столбца во все процессы этого же столбца и вывести полученное число в каждом процессе (включая процессы 0, 1 и 2).

MPI5Comm19. Количество процессов K равно 8 или 12, в каждом процессе дано целое число. Определить для всех процессов декартову топологию в виде трехмерной решетки размера $2 \times 2 \times K/4$ (порядок нумерации процессов оставить прежним). Интерпретируя эту решетку как две матрицы размера $2 \times K/4$ (в одну матрицу входят процессы с одинаковой первой координатой), расщепить каждую матрицу процессов на две одномерные строки. Используя одну коллективную операцию пересылки данных, переслать в главный процесс каждой строки исходные числа из процессов этой же строки и вывести полученные наборы чисел (включая числа, полученные из главных процессов строк).

MPI5Comm20. Количество процессов K равно 8 или 12, в каждом процессе дано целое число. Определить для всех процессов декартову топологию в виде трехмерной решетки размера $2 \times 2 \times K/4$ (порядок нумерации процессов оставить прежним). Интерпретируя полученную решетку как $K/4$ матриц размера 2×2 (в одну матрицу входят процессы с одинаковой третьей координатой), расщепить эту решетку на $K/4$ указанных матриц. Используя одну коллективную операцию пересылки данных, переслать в главный процесс каждой из полученных матриц исходные числа из процессов этой же матрицы и вывести полученные наборы чисел (включая числа, полученные из главных процессов матриц).

MPI5Comm21. Количество процессов K равно 8 или 12, в каждом процессе дано вещественное число. Определить для всех процессов декартову топологию в виде трехмерной решетки размера $2 \times 2 \times K/4$ (порядок нумерации процессов оставить прежним). Интерпретируя эту решетку как две матрицы размера $2 \times K/4$ (в одну матрицу входят процессы с одинаковой первой координатой), расщепить каждую матрицу процессов на $K/4$ одномерных столбцов. Используя одну коллективную операцию редукции, для каждого столбца процессов найти произведение исходных чисел и вывести найденные произведения в главных процессах каждого столбца.

MPI5Comm22. Количество процессов K равно 8 или 12, в каждом процессе дано вещественное число. Определить для всех процессов декартову топологию в виде трехмерной решетки размера $2 \times 2 \times K/4$ (порядок нумерации процессов оставить прежним). Интерпретируя полученную решетку как $K/4$ матриц размера 2×2 (в одну матрицу входят процессы с одинаковой третьей координатой), расщепить эту решетку на $K/4$ указанных матриц. Используя одну коллективную операцию редукции, для каждой из полученных матриц найти сумму исходных чисел и вывести найденные суммы в каждом процессе соответствующей матрицы.

MPI5Comm23. В главном процессе даны положительные целые числа M и N, произведение которых не превосходит количества процессов; кроме того, в процессах с рангами от 0 до $M \cdot N - 1$ даны целые числа X и Y. Переслать числа M и N во все процессы, после чего определить для начальных $M \cdot N$ процессов декартову топологию в виде двумерной решетки размера $M \times N$, являющейся периодической по первому измерению (порядок нумерации процессов оставить прежним). В каждом процессе, входящем в созданную топологию, вывести ранг процесса с координатами X, Y (с учетом периодичности), используя для

этого функцию MPI_Cart_rank. В случае недопустимых координат вывести −1.

Примечание. Если при вызове функции MPI_Cart_rank указаны недопустимые координаты (например, отрицательные координаты для измерений, по которым декартова решетка не является периодической), то сама функция возвращает ненулевое значение (являющееся признаком ошибки), а возвращаемое значение параметра rank является неопределенным. Таким образом, в данном задании число −1 следует выводить, если функция MPI_Cart_rank вернула ненулевое значение. Чтобы подавить возникающие при этом сообщения об ошибках, отображаемые в разделе отладки окна задачника, достаточно перед вызовом функции, который может привести к ошибке, установить с помощью функции MPI_Comm_set_errhandler (MPI_Errhandler_set в MPI-1) специальный вариант *обработчика ошибок* MPI_ERROR_RETURN, который при возникновении ошибки не выполняет никаких действий, кроме установки ненулевого возвращаемого значения для этой функции. Следует заметить, что в версии MPICH 1.2.5 в случае указания недопустимых координат функция MPI_Cart_rank возвращает значение параметра rank, равное −1, что позволяет упростить решение, избавившись от проверки возвращаемого значения функции. Однако и в этом случае желательно подавить вывод сообщений об ошибках описанным выше способом.

MPI5Comm24. В главном процессе даны положительные целые числа M и N, произведение которых не превосходит количества процессов; кроме того, в процессах с рангами от 0 до $M \cdot N - 1$ даны целые числа X и Y. Переслать числа M и N во все процессы, после чего определить для начальных $M \cdot N$ процессов декартову топологию в виде двумерной решетки размера $M \times N$, являющейся периодической по второму измерению (порядок нумерации процессов оставить прежним). В каждом процессе, входящем в созданную топологию, вывести ранг процесса с координатами X, Y (с учетом периодичности), используя для этого функцию MPI_Cart_rank. В случае недопустимых координат вывести −1.

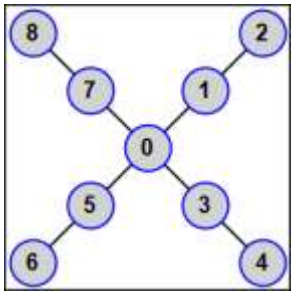
Примечание. См. примечание к заданию MPI5Comm23.

MPI5Comm25. В каждом подчиненном процессе дано вещественное число. Определить для всех процессов декартову топологию в виде одномерной решетки и осуществить простой сдвиг исходных данных с шагом −1 (число из каждого *подчиненного* процесса пересылается в предыдущий процесс). Для определения рангов посылающих и принимающих процессов использовать функцию MPI_Cart_shift, пересылку выполнять с помощью функций MPI_Send и MPI_Recv. Во всех процессах, получивших данные, вывести эти данные.

MPI5Comm26. Число процессов K является четным: $K = 2N$, $N > 1$; в каждом процессе дано вещественное число A. Определить для всех процессов декартову топологию в виде матрицы размера $2 \times N$ (порядок нумерации процессов оставить прежним) и для каждой строки матрицы осуществить циклический сдвиг исходных данных с шагом 1 (число A из каждого процесса, кроме последнего в строке, пересылается в следующий процесс этой же строки, а из последнего процесса — в главный процесс этой строки). Для определения рангов посылающих и принимающих процессов использовать функцию MPI_Cart_shift, пересылку выполнять с помощью функции MPI_Sendrecv. Во всех процессах вывести полученные данные.

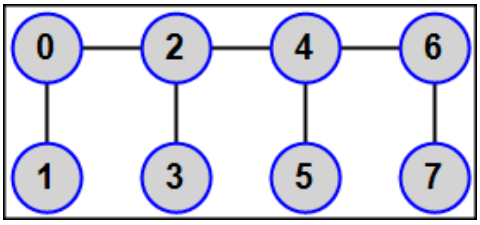
MPI5Comm27. Количество процессов K равно 8 или 12, в каждом процессе дано вещественное число. Определить для всех процессов декартову топологию в виде трехмерной решетки размера $2 \times 2 \times K/4$ (порядок нумерации процессов оставить прежним). Интерпретируя полученную решетку как $K/4$ матриц размера 2×2 (в одну матрицу входят процессы с одинаковой третьей координатой, матрицы упорядочены по возрастанию третьей координаты), осуществить циклический сдвиг исходных данных из процессов каждой матрицы в соответствующие процессы следующей матрицы (из процессов последней матрицы данные перемещаются в первую матрицу). Для определения рангов посылающих и принимающих процессов использовать функцию MPI_Cart_shift, пересылку выполнять с помощью функции MPI_Sendrecv_replace. Во всех процессах вывести полученные данные.

MPI5Comm28. Число процессов K является нечетным: $K = 2N + 1$ ($1 < N < 5$); в каждом процессе дано целое число A. Используя функцию MPI_Graph_create, определить для всех процессов топологию графа, в которой главный процесс связан ребрами со всеми процессами нечетного ранга (1, 3, …, $2N - 1$), а каждый процесс четного положительного ранга R (2, 4, …, $2N$) связан ребром с процессом ранга $R - 1$ (в результате получается N-лучевая звезда, центром которой является главный процесс, а каждый луч состоит из двух подчиненных процессов R и $R + 1$, причем ближайшим к центру является процесс нечетного ранга R).



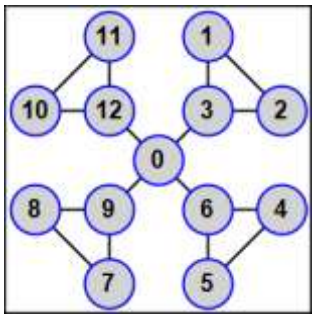
Переслать число A из каждого процесса всем процессам, связанным с ним ребрами (*процессам-соседям*). Для определения количества процессов-соседей и их рангов использовать функции MPI_Graph_neighbors_count и MPI_Graph_neighbors, пересылку выполнять с помощью функций MPI_Send и MPI_Recv. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов.

MPI5Comm29. Число процессов K является четным: $K = 2N$ ($1 < N < 6$); в каждом процессе дано целое число A. Используя функцию MPI_Graph_create, определить для всех процессов топологию графа, в которой все процессы четного ранга (включая главный процесс) связаны в цепочку: 0 — 2 — 4 — 6 — … — ($2N - 2$), а каждый процесс нечетного ранга R (1, 3, …, $2N - 1$) связан с процессом ранга $R - 1$ (в результате каждый процесс нечетного ранга будет иметь единственного соседа, первый и последний процессы четного ранга будут иметь по два соседа, а остальные — «внутренние» — процессы четного ранга будут иметь по три соседа).



Переслать число A из каждого процесса всем процессам-соседям. Для определения количества процессов-соседей и их рангов использовать функции MPI_Graph_neighbors_count и MPI_Graph_neighbors, пересылку выполнять с помощью функции MPI_Sendrecv. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов.

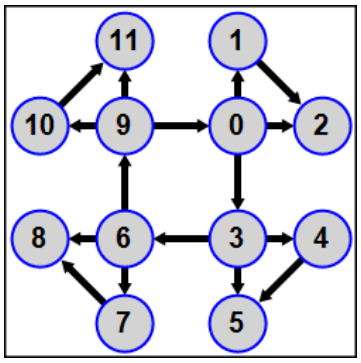
MPI5Comm30. Количество процессов K равно $3N + 1$ ($1 < N < 5$); в каждом процессе дано целое число A. Используя функцию MPI_Graph_create, определить для всех процессов топологию графа, в которой процессы R, $R + 1$, $R + 2$, где $R = 1, 4, 7, \ldots$, связаны между собой ребрами, и, кроме того, каждый процесс положительного ранга, кратного трем (3, 6, …), связан ребром с главным процессом (в результате получается N-лучевая звезда, центром которой является главный процесс, а каждый луч состоит из трех связанных между собой процессов, причем с центром связан процесс ранга, кратного трем).



Переслать число *A* из каждого процесса всем процессам-соседям. Для определения количества процессов-соседей и их рангов использовать функции MPI_Graph_neighbors_count и MPI_Graph_neighbors, пересылку выполнять с помощью функции MPI_Sendrecv. Во всех процессах вывести полученные числа в порядке возрастания рангов переславших их процессов.

5.3. Топология распределенного графа (MPI-2)

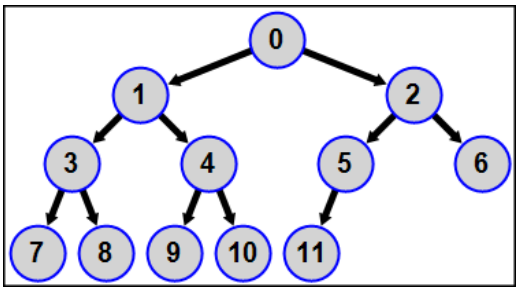
MPI5Comm31. Количество процессов *K* кратно 3; в каждом процессе дано целое число *A*. Используя функцию MPI_Dist_graph_create, определить для всех процессов топологию распределенного графа, в которой все процессы рангов, кратных 3 (0, 3, …, $K-3$) образуют кольцо, причем каждый из процессов, входящих в кольцо, является источником для последующего процесса (процесс 0 является источником для процесса 3, процесс 3 — для процесса 6, …, процесс $K-3$ — для процесса 0) и, кроме того, процесс ранга $3N$ ($N = 0, 1, \ldots, K/3-1$) является источником для процессов ранга $3N+1$ и $3N+2$, а процесс ранга $3N+1$ является источником для процесса ранга $3N+2$.



Полное определение топологии дать в главном процессе (в подчиненных процессах второй параметр функции MPI_Dist_graph_create должен быть равен 0); параметр weights положить равным MPI_UNWEIGHTED, параметр info — равным MPI_INFO_NULL, порядок нумерации процессов не изменять. Переслать числа *A* от процессов-источников процессам-приемникам и вывести в каждом процессе сумму исходного числа *A* и всех чисел, полученных от процессов-источников. Для определения количества источников и приемни-

ков, а также их рангов использовать функции MPI_Dist_graph_neighbors_count и MPI_Dist_graph_neighbors, пересылку выполнять с помощью функций MPI_Send и MPI_Recv.

MPI5Comm32. Количество процессов лежит в диапазоне от 4 до 15, в каждом процессе дано целое число *A*. Используя функцию MPI_Dist_graph_create, определить для всех процессов топологию распределенного графа, которая представляет собой бинарное дерево с корнем в главном процессе 0, вершинами первого уровня в процессах рангов 1 и 2, вершинами второго уровня в процессах рангов 3-6 (причем процессы ранга 3 и 4 являются дочерними вершинами процесса 1, а процессы ранга 5 и 6 — дочерними вершинами процесса 2), и т. д. Каждый процесс является источником для своих дочерних вершин; таким образом, каждый процесс имеет от 0 до 2 процессов-приемников.



Полное определение топологии дать в главном процессе (в подчиненных процессах второй параметр функции MPI_Dist_graph_create должен быть равен 0); параметр weights положить равным MPI_UNWEIGHTED, параметр info — равным MPI_INFO_NULL, порядок нумерации процессов не изменять. Найти и вывести в каждом процессе сумму исходного числа *A* и чисел, данных в процессах-предках *всех уровней* — от корня (главного процесса) до ближайшего предка (родительского процесса-источника). Для определения количества источников и приемников, а также их рангов использовать функции MPI_Dist_graph_neighbors_count и MPI_Dist_graph_neighbors, пересылку данных выполнять с помощью функций MPI_Send и MPI_Recv.

6. Параллельный ввод-вывод файловых данных (MPI-2)

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 первых двух подгруппах (задания MPI6File1–MPI6File16) не требуется настраивать вид файловых данных с помощью функции

MPI_File_set_view; достаточно использовать вид по умолчанию, при котором базовые и файловые элементы имеют тип MPI_BYTE, начальное смещение для всех процессов равно 0 и используется представление «native». Это же представление надо указывать и при явной настройке вида файловых данных в заданиях третьей подгруппы (MPI6File17–MPI6File30).

Для определения протяженности типов MPI_INT и MPI_DOUBLE следует применять функцию MPI_Type_get_extent.

Для указания завершающего пустого промежутка (в заданиях, связанных с настройкой вида файловых данных) следует использовать функцию MPI_Type_create_resized. Альтернативным вариантом является использование специального типа MPI_UB (*метки нулевого размера*), однако в стандарте MPI-2 этот тип объявлен устаревшим.

6.1. Локальные функции для файлового ввода-вывода

MPI6File1. В главном процессе дано имя существующего файла целых чисел. В каждом из подчиненных процессов дано количество файловых элементов, предназначенных для считывания в этом процессе, и порядковые номера этих элементов (элементы нумеруются от 1). Элементы с некоторыми номерами в исходном файле могут отсутствовать. Используя в каждом процессе требуемое количество вызовов локальной функции MPI_File_read_at, прочесть из файла существующие элементы с указанными номерами и вывести их в том же порядке. Для проверки наличия элемента с указанным порядковым номером можно либо использовать функцию MPI_File_get_size, либо анализировать информацию, возвращенную в параметре типа MPI_Status функции MPI_File_read_at.

MPI6File2. В главном процессе дано имя файла. В каждом из подчиненных процессов дано количество пар целых чисел и сами пары, в которых первый член равен порядковому номеру файлового элемента, а второй — значению этого файлового элемента (элементы нумеруются от 1, все номера различны и принимают все значения от 1 до некоторого целого числа). Создать новый файл целых чисел с указанным именем и записать в него данные элементы, используя в каждом подчиненном процессе требуемое количество вызовов локальной функции MPI_File_write_at.

MPI6File3. В главном процессе дано имя существующего файла вещественных чисел, содержащего элементы прямоугольной матрицы размера $K \times N$, где K — число подчиненных процессов. В каждом подчиненном процессе прочесть и вывести элементы R-й строки матрицы, где R — ранг подчиненного процесса (строки нумеруются от 1), ис-

пользуя один вызов локальной функции MPI_File_read_at. Для определения размера файла использовать функцию MPI_File_get_size.

MPI6File4. В главном процессе дано имя файла, в каждом из подчиненных процессов дано *R* вещественных чисел, где *R* — ранг процесса. Создать новый файл вещественных чисел с указанным именем и записать в него данные элементы в порядке возрастания рангов содержащих их процессов. Использовать в каждом подчиненном процессе один вызов локальной функции MPI_File_write_at.

MPI6File5. В главном процессе дано имя существующего файла целых чисел. Файл обязательно содержит все целые числа в диапазоне от 1 до *K*, где *K* — максимальный ранг процесса. В каждом подчиненном процессе прочесть и вывести два фрагмента файловых данных: первый содержит *начальную* часть элементов до первого элемента, равного рангу процесса (включая этот элемент), второй содержит *завершающую* часть файловых элементов и имеет тот же размер, что и первый фрагмент. Элементы в каждом фрагменте выводятся в порядке их следования в файле. Использовать в каждом подчиненном процессе необходимое количество вызовов локальной функции MPI_File_read (без применения массивов), а также вызовы функций MPI_File_get_position для определения размера первого фрагмента и MPI_File_seek с параметром MPI_SEEK_END для перемещения к началу второго фрагмента.

MPI6File6. В главном процессе дано имя файла, в каждом из подчиненных процессов дано целое число. Создать новый файл целых чисел с указанным именем и записать в него *K* подряд идущих копий каждого числа, где *K* равно количеству подчиненных процессов. Порядок следования чисел в файле должен быть обратным порядку подчиненных процессов (*K* копий числа из процесса 1 должны находиться в конце файла, перед ними должны располагаться *K* копий числа из процесса 2, и т. д.). Использовать в каждом подчиненном процессе один вызов локальной функции MPI_File_write, а также функцию MPI_File_seek с параметром MPI_SEEK_SET.

MPI6File7. В главном процессе дано имя существующего файла вещественных чисел. Известно, что сумма значений всех файловых элементов превосходит *K*, где *K* — количество подчиненных процессов. В каждом подчиненном процессе считывать начальные элементы файла, пока сумма их значений не превысит ранг процесса, после чего вывести найденную сумму *S* и количество *N* прочитанных чисел. После этого дополнительно прочесть и вывести значения *N последних* файловых элементов (в порядке их следования в файле). Использовать в каждом подчиненном процессе необходимое количество вызовов локальной

функции MPI_File_read (без применения массивов), а также вызовы функций MPI_File_get_position для определения *N* и MPI_File_seek с параметром MPI_SEEK_END для перемещения к началу фрагмента из *N* последних файловых элементов.

MPI6File8. В главном процессе дано имя файла, в каждом из подчиненных процессов дано вещественное число. Создать новый файл вещественных чисел с указанным именем и записать в него *R* подряд идущих копий каждого числа, где *R* равно рангу процесса, в котором дано это число. Порядок следования чисел в файле должен быть *обратным* порядку подчиненных процессов (в конце файла должно находиться единственное число из процесса 1, перед ним должны располагаться две копии числа из процесса 2, и т. д.). Использовать в каждом подчиненном процессе один вызов локальной функции MPI_File_write, а также функцию MPI_File_seek с параметром MPI_SEEK_SET.

6.2. Коллективные функции для файлового ввода-вывода

MPI6File9. В главном процессе дано имя существующего файла целых чисел. В каждом процессе прочесть и вывести *R* + 1 элемент файла, начиная с элемента с номером *R* + 1, где число *R* равно рангу процесса (0, 1, …). Файловые элементы нумеруются от 1; таким образом, в процессе ранга 0 требуется прочесть и вывести только начальный файловый элемент, в процессе ранга 1 — два следующих файловых элемента, в процессе ранга 2 — три элемента, начиная с третьего, и т. д. Если файл содержит недостаточное количество элементов, то в некоторых процессах число выведенных элементов может быть меньше требуемого. Использовать один вызов коллективной функции MPI_File_read_all, а также функцию MPI_File_seek с параметром MPI_SEEK_SET и функцию MPI_File_get_size для определения размера исходного файла.

Примечание. В реализации MPICH2 версии 1.3 функция MPI_File_read_all не позволяет определить количество фактически считанных файловых элементов на основе информации, содержащейся в параметре типа MPI_Status: в этом параметре количество прочитанных элементов всегда равно требуемому, а при недостаточном количестве файловых элементов в завершающую часть выходного массива записываются нулевые значения.

MPI6File10. В главном процессе дано имя файла, в каждом из подчиненных процессов дан набор из *R* целых чисел, где *R* — ранг процесса (1, 2, …). Создать новый файл целых чисел с указанным именем и записать в него все данные числа в порядке их следования (процессы перебираются в порядке возрастания их рангов). Использовать один вы-

зов коллективной функции MPI_File_write_all (для всех процессов, в том числе и для процесса ранга 0), а также функцию MPI_File_seek с параметром MPI_SEEK_SET.

MPI6File11. В главном процессе дано имя существующего файла вещественных чисел. Кроме того, в каждом процессе дано целое число, равное либо 0, либо порядковому номеру одного из существующих файловых элементов (элементы нумеруются от 1). Используя функцию MPI_Comm_split, создать новый коммуникатор, содержащий только те процессы, в которых дано число, отличное от 0, и с помощью коллективной функции MPI_File_read_at_all прочесть и вывести в каждом процессе из этого коммуникатора элемент, расположенный в позиции с указанным порядковым номером.

MPI6File12. В главном процессе дано имя существующего файла вещественных чисел. Кроме того, в каждом процессе дано целое число, равное либо 0, либо порядковому номеру одного из существующих файловых элементов (элементы нумеруются от 1). Используя функцию MPI_Comm_split, создать новый коммуникатор, содержащий только те процессы, в которых дано число, отличное от 0, и с помощью коллективной функции MPI_File_write_at_all заменить исходное значение файлового элемента в позиции с указанным порядковым номером на значение ранга процесса в новом коммуникаторе (преобразовав ранг в вещественное число).

MPI6File13. В главном процессе дано имя существующего файла целых чисел. Кроме того, в каждом процессе дано целое число, равное либо 0, либо 1. Используя функцию MPI_Comm_split, создать новый коммуникатор, содержащий только те процессы, в которых дано число 1, и с помощью коллективной функции MPI_File_read_ordered прочесть и вывести в каждом процессе из этого коммуникатора $R + 1$ элемент исходного файла, где R — ранг процесса в новом коммуникаторе (элементы считываются последовательно: один элемент в процессе 0, два следующих элемента — в процессе 1, три следующих — в процессе 2, и т. д.). Если файл содержит недостаточное количество элементов, то в некоторых процессах число выведенных элементов N может быть меньше требуемого или даже равно нулю. В каждом процессе, входящем в новый коммуникатор, дополнительно вывести количество N фактически прочитанных элементов и новое значение P коллективного файлового указателя. Для определения количества прочитанных элементов N использовать информацию, возвращенную в параметре типа MPI_Status функции MPI_File_read_ordered, для нахождения значения P использовать функцию MPI_File_get_position_shared (во всех процессах значение P должно быть одинаковым).

MPI6File14. В главном процессе дано имя файла. Кроме того, в каждом процессе дано целое число *N*. Создать новый файл целых чисел с указанным именем. Используя функцию MPI_Comm_split, создать новый коммуникатор, содержащий только те процессы, в которых число *N* не равно 0, и с помощью коллективной функции MPI_File_write_ordered записать в файл *K* подряд идущих копий каждого из чисел *N*, где *K* равно количеству процессов в *новом* коммуникаторе, а числа располагаются в порядке возрастания рангов содержащих их процессов.

MPI6File15. В главном процессе дано имя существующего файла целых чисел, содержащего не менее *K* элементов, где *K* — количество процессов. С помощью функции MPI_Comm_split создать новый коммуникатор, содержащий только процессы с нечетным рангом (1, 3, …). Используя по одному вызову коллективных функций MPI_File_seek_shared и MPI_File_read_ordered, прочесть и вывести в каждом процессе из нового коммуникатора по 2 элемента исходного файла, причем предпоследний и последний файловые элементы должны считываться и выводиться (в указанном порядке) в процессе с рангом 1 в исходном коммуникаторе MPI_COMM_WORLD, четвертый и третий с конца элементы — в процессе с рангом 3, и т. д.

Указание. Для того чтобы обеспечить требуемый порядок считывания данных в функции MPI_File_read_ordered, необходимо изменить в созданном коммуникаторе порядок следования процессов на противоположный (по сравнению с исходным порядком в коммуникаторе MPI_COMM_WORLD).

MPI6File16. В главном процессе дано имя файла. Кроме того, в каждом процессе дано целое число N (≥ 0) и N вещественных чисел. Создать новый файл целых чисел с указанным именем. Используя функцию MPI_Comm_split, создать новый коммуникатор, содержащий только те процессы, в которых число *N* не равно 0, и с помощью единственного вызова коллективной функции MPI_File_write_ordered записать в файл все вещественные числа в порядке, *обратном* их следованию в наборе исходных данных: вначале записываются (в обратном порядке) все числа из процесса с наибольшим рангом в коммуникаторе MPI_COMM_WORLD, затем все числа из процесса с предыдущим рангом, и т. д.

Указание. Для того чтобы обеспечить требуемый порядок записи данных в функции MPI_File_write_ordered, необходимо изменить в созданном коммуникаторе порядок следования процессов на противоположный (по сравнению с исходным порядком в коммуникаторе MPI_COMM_WORLD).

6.3. Настройка вида данных для файлового ввода-вывода

MPI6File17. В главном процессе дано имя существующего файла целых чисел, содержащего $2K$ элементов, где K — количество процессов. Используя единственный вызов коллективной функции MPI_File_read_all (и не применяя функцию MPI_File_seek), прочесть и вывести по 2 элемента в каждом процессе, перебирая элементы в порядке их следования в файле. Для этого предварительно с помощью функции MPI_File_set_view определить новый вид данных с базовым типом MPI_INT, таким же файловым типом и подходящим смещением (своим для каждого процесса).

MPI6File18. В главном процессе дано имя существующего файла целых чисел, содержащего элементы прямоугольной матрицы размера $K \times 5$, где K — количество процессов. Кроме того, в каждом процессе дано целое число N ($1 \le N \le 5$) — порядковый номер *выделенного* элемента в некоторой строке матрицы (элементы в строке нумеруются от 1), причем с процессом ранга 0 связывается первая строка матрицы, с процессом ранга 1 — вторая строка, и т. д. Используя единственный вызов коллективной функции MPI_File_write_at_all со вторым параметром, равным $N - 1$, изменить в каждой строке исходной матрицы выделенный элемент, заменив его значение на ранг связанного с ним процесса (выделенный элемент в первой строке следует заменить на 0, во второй строке — на 1, и т. д.). Для этого предварительно с помощью функции MPI_File_set_view определить новый вид данных с базовым типом MPI_INT, таким же файловым типом и подходящим смещением (своим для каждого процесса).

MPI6File19. В главном процессе дано имя существующего файла вещественных чисел, содержащего элементы прямоугольной матрицы размера $K \times 6$, где K — количество процессов. Кроме того, в каждом процессе дано целое число N ($1 \le N \le 6$) — порядковый номер *выделенного* элемента в некоторой строке матрицы (элементы в строке нумеруются от 1), причем с процессом ранга 0 связывается последняя строка матрицы, с процессом ранга 1 — предпоследняя строка, и т. д. Используя единственный вызов коллективной функции MPI_File_read_at_all со вторым параметром, равным $N - 1$, прочесть в каждой строке исходной матрицы выделенный элемент и вывести его значение в соответствующем процессе (выделенный элемент в первой строке следует вывести в последнем процессе, во второй строке — в предпоследнем процессе, и т. д.). Для этого предварительно с помощью функции MPI_File_set_view определить новый вид данных с базовым типом MPI_DOUBLE, таким же файловым типом и подходящим смещением (своим для каждого процесса).

MPI6File20. В главном процессе дано имя файла. Кроме того, в каждом процессе дан набор из $R + 1$ вещественного числа, где R — ранг процесса (0, 1, …). Создать новый файл целых чисел с указанным именем. Используя единственный вызов коллективной функции MPI_File_write_all (и не применяя функцию MPI_File_seek), записать в файл все данные числа в порядке, обратном их следованию в исходном наборе: вначале записываются (в обратном порядке) все числа из процесса с наибольшим рангом, затем все числа из процесса с предыдущим рангом, и т. д. Для этого предварительно с помощью функции MPI_File_set_view определить новый вид данных с базовым типом MPI_DOUBLE, таким же файловым типом и подходящим смещением (своим для каждого процесса).

MPI6File21. В главном процессе дано имя существующего файла целых чисел, содержащего $3K$ элементов, где K — количество процессов. В каждом процессе прочесть и вывести три элемента A, B, C, расположенных в исходном файле в следующем порядке (индекс указывает ранг процесса): A_0, A_1, …, A_{K-1}, B_0, B_1, …, B_{K-1}, C_0, C_1, …, C_{K-1}. Для этого использовать единственный вызов коллективной функции MPI_File_read_all, предварительно определив новый файловый вид данных с базовым типом MPI_INT, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из одного целочисленного элемента и завершающего пустого промежутка, размер которого равен протяженности набора из $K - 1$ целого числа.

MPI6File22. В главном процессе дано имя файла. Кроме того, в каждом процессе дано по 3 целых числа: A, B, C. Количество процессов равно K. Создать новый файл целых чисел с указанным именем и записать в него исходные числа в следующем порядке (индекс указывает ранг процесса): A_{K-1}, A_{K-2}, …, A_0, B_{K-1}, B_{K-2}, …, B_0, C_{K-1}, C_{K-2}, …, C_0. Для этого использовать единственный вызов коллективной функции MPI_File_write_all, предварительно определив новый файловый вид данных с базовым типом MPI_INT, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из одного целочисленного элемента и завершающего пустого промежутка, размер которого равен протяженности набора из $K - 1$ целого числа.

MPI6File23. В главном процессе дано имя существующего файла вещественных чисел, содержащего $6K$ элементов, где K — количество процессов. В каждом процессе прочесть и вывести шесть элементов A, B, C, D, E, F, расположенных в исходном файле в следующем порядке (индекс указывает ранг процесса): A_0, B_0, C_0, A_1, B_1, C_1, …, A_{K-1}, B_{K-1}, C_{K-1}, D_0, E_0, F_0, D_1, E_1, F_1, …, D_{K-1}, E_{K-1}, F_{K-1}. Для этого использовать единственный вызов коллективной функции MPI_File_read_all, пред-

варительно определив новый файловый вид данных с базовым типом MPI_DOUBLE, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из трех вещественных элементов и завершающего пустого промежутка подходящего размера.

MPI6File24. В главном процессе дано имя файла. Кроме того, в каждом процессе дано по 4 вещественных числа: *A*, *B*, *C*, *D*. Количество процессов равно *K*. Создать новый файл вещественных чисел с указанным именем и записать в него исходные числа в следующем порядке (индекс указывает ранг процесса): A_{K-1}, B_{K-1}, A_{K-2}, B_{K-2}, …, A_0, B_0, C_{K-1}, D_{K-1}, C_{K-2}, D_{K-2}, …, C_0, D_0. Для этого использовать единственный вызов коллективной функции MPI_File_write_all, предварительно определив новый файловый вид данных с базовым типом MPI_DOUBLE, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из двух вещественных элементов и завершающего пустого промежутка подходящего размера.

MPI6File25. В главном процессе дано имя файла. Кроме того, в процессе ранга *R* ($R = 0, 1, \ldots, K - 1$, где *K* — количество процессов) дано $3 \cdot (R + 1)$ целых чисел: в процессе 0 даны 3 числа *A*, *B*, *C*, в процессе 1 — 6 чисел *A*, *A′*, *B*, *B′*, *C*, *C′*, в процессе 2 — 9 чисел *A*, *A′*, *A″*, *B*, *B′*, *B″*, *C*, *C′*, *C″*, и т. д. Создать новый файл целых чисел с указанным именем и записать в него исходные числа в следующем порядке (индекс указывает ранг процесса): A_0, A_1, A'_1, A_2, A'_2, A''_2, …, B_0, B_1, B'_1, B_2, B'_2, B''_2, … . Для этого использовать единственный вызов коллективной функции MPI_File_write_all, предварительно определив новый файловый вид данных с базовым типом MPI_INT, подходящим смещением (своим для каждого процесса) и новым файловым типом (своим для каждого процесса), состоящим из $R + 1$ целого элемента и завершающего пустого промежутка подходящего размера.

MPI6File26. В главном процессе дано имя файла. Кроме того, в каждом процессе дано по 4 вещественных числа: *A*, *B*, *C*, *D*. Количество процессов равно *K*. Создать новый файл вещественных чисел с указанным именем и записать в него исходные числа в следующем порядке (индекс указывает ранг процесса): A_0, A_1, …, A_{K-1}, B_{K-1}, …, B_1, B_0, C_0, C_1, …, C_{K-1}, D_{K-1}, …, D_0. Для этого использовать единственный вызов коллективной функции MPI_File_write_all, предварительно определив новый файловый вид данных с базовым типом MPI_DOUBLE, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из двух вещественных элементов (с дополнительным пустым промежутком между этими элементами) и завершающего пустого промежутка подходящего размера.

MPI6File27. В главном процессе дано имя существующего файла вещественных чисел, содержащего элементы прямоугольной матрицы размера (K/2) × K, где K — количество процессов (четное число). В процессе ранга R, R = 0, …, K − 1, прочесть и вывести элементы столбца исходной матрицы с номером R + 1 (столбцы нумеруются от 1). Для этого использовать единственный вызов функции MPI_File_read_all, предварительно определив новый файловый вид данных с базовым типом MPI_DOUBLE, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из вещественного элемента и завершающего пустого промежутка подходящего размера.

MPI6File28. В главном процессе дано имя файла. Кроме того, в каждом процессе дано целое число N и K/2 вещественных чисел, где K — количество процессов (четное число). Числа N для всех процессов различны и лежат в диапазоне от 1 до K. Создать новый файл вещественных чисел с указанным именем и записать в него матрицу размера (K/2) × K, причем каждый процесс должен записать свой набор вещественных чисел в столбец матрицы с порядковым номером N (столбцы нумеруются от 1). Для этого использовать единственный вызов коллективной функции MPI_File_write_all, предварительно определив новый файловый вид данных с базовым типом MPI_DOUBLE, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из вещественного элемента и завершающего пустого промежутка подходящего размера.

MPI6File29. В главном процессе дано имя существующего файла целых чисел, содержащего элементы блочной прямоугольной матрицы. Размер матрицы в блоках равен (K/3) × 3, где K — количество процессов (число K кратно 3). Каждый блок представляет собой квадратную матрицу порядка N (значение N может лежать в диапазоне от 2 до 5). В каждом процессе требуется прочесть и вывести один блок исходной матрицы, перебирая блоки по строкам. Для этого использовать единственный вызов функции MPI_File_read_all, предварительно определив новый файловый вид данных с базовым типом MPI_INT, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из N целочисленных элементов и завершающего пустого промежутка подходящего размера. Для определения числа N использовать размер исходного файла, получив его с помощью функции MPI_File_get_size.

MPI6File30. В главном процессе дано имя файла и число N, которое может лежать в диапазоне от 2 до 5. Кроме того, в каждом процессе даны два целых числа I и J, определяющие позицию (номер строки и столбца) некоторого квадратного блока блочной прямоугольной матрицы раз-

мера (K/3) × 3 блоков, где K — количество процессов (число K кратно 3). Значения I лежат в диапазоне от 1 до K/3, значения J лежат в диапазоне от 1 до 3; все процессы содержат различные позиции блоков. Каждый блок представляет собой квадратную матрицу целых чисел порядка N. Создать новый файл целых чисел с указанным именем и записать в него блочную матрицу размера (K/3) × 3, причем каждый процесс должен записать в матрицу блок в позиции (I, J), а все элементы блока, записанного процессом ранга R (R = 0, 1, …, K − 1), должны быть равны числу R. Для этого использовать единственный вызов функции MPI_File_write_all, предварительно определив новый файловый вид данных с базовым типом MPI_INT, подходящим смещением (своим для каждого процесса) и новым файловым типом, состоящим из K целочисленных элементов и завершающего пустого промежутка подходящего размера. Для передачи значения N во все процессы использовать коллективную функцию MPI_Bcast.

7. Односторонние коммуникации (MPI-2)

При определении окна доступа с помощью функции MPI_Win_create рекомендуется всегда указывать смещение disp_unit (третий параметр), равным протяженности элемента данных соответствующего типа (в заданиях это всегда либо тип MPI_INT, либо тип MPI_DOUBLE; протяженность следует определять с помощью функции MPI_Type_get_extent). В этом случае при последующих вызовах функций одностороннего доступа MPI_Get, MPI_Put, MPI_Accumulate будет достаточно указывать смещение target_disp (пятый параметр любой из этих функций) равным начальному индексу используемого фрагмента массива (а не числу байтов от начала массива до требуемого фрагмента).

В качестве параметра info (четвертый параметр функции MPI_Win_create) достаточно указывать константу MPI_INFO_NULL.

Если в некоторых процессах окно доступа определять не требуется, то в качестве параметра size (второго параметра функции MPI_Win_create) в этих процессах следует указывать значение 0.

В качестве параметра assert во всех синхронизирующих функциях (MPI_Win_fence, MPI_Win_start, MPI_Win_post, MPI_Win_lock) достаточно указывать константу 0 (данный параметр является предпоследним параметром во всех перечисленных функциях).

В первой подгруппе (задания MPI7Win1–MPI7Win17) в качестве синхронизирующих функций следует использовать *коллективную* функцию MPI_Win_fence, которая должна вызываться как перед действиями, связанными с односторонней пересылкой данных, так и после этих действий (но перед действиями, связанными с доступом к пересланным данным).

Задания второй подгруппы (MPI7Win18–MPI7Win30) требуют применения *локальных* вариантов синхронизации: функций MPI_Win_start, MPI_Win_complete, MPI_Win_post, MPI_Win_wait или пары функций MPI_Win_lock, MPI_Win_unlock. В заданиях этой подгруппы всегда указывается, какой вариант синхронизации требуется использовать.

7.1. Односторонние коммуникации с простейшей синхронизацией

MPI7Win1. В каждом из подчиненных процессов дано одно целое число. В главном процессе определить окно доступа размера K целых чисел (K — количество подчиненных процессов) и, используя функцию MPI_Put в подчиненных процессах, записать в главный процесс все исходные числа, после чего вывести эти числа в порядке возрастания рангов переславших их процессов.

MPI7Win2. В каждом из подчиненных процессов дано R вещественных чисел, где R — ранг процесса (1, 2, …). В главном процессе определить окно доступа подходящего размера и, используя функцию MPI_Put в подчиненных процессах, записать в главный процесс все исходные числа, после чего вывести эти числа в порядке возрастания рангов переславших их процессов.

MPI7Win3. В главном процессе дан массив A из K целых чисел, где K — количество подчиненных процессов. Определить в главном процессе окно доступа, содержащее массив A, и, используя функцию MPI_Get в подчиненных процессах, получить и вывести в них по одному элементу массива A, перебирая элементы с конца (элемент A_0 получить в последнем процессе, A_1 в предпоследнем, и т. д.).

MPI7Win4. В главном процессе дан массив A из $K + 4$ вещественных чисел, где K — количество подчиненных процессов. Определить в главном процессе окно доступа, содержащее массив A, и, используя функцию MPI_Get в подчиненных процессах, получить и вывести в них по пять элементов массива A, начиная с элемента с индексом $R - 1$, где R — ранг подчиненного процесса ($R = 1, 2, \ldots, K - 1$).

MPI7Win5. В главном процессе дан массив A из K целых чисел, где K — количество подчиненных процессов. В каждом подчиненном процессе дан индекс N (число от 0 до $K - 1$) и целое число B. В главном процессе определить окно доступа, содержащее массив A, и, используя функцию MPI_Accumulate в каждом подчиненном процессе, умножить элемент A_N на число B, после чего вывести в главном процессе измененный массив A.

Примечание. Некоторые подчиненные процессы могут содержать совпадающие значения N; в этом случае элемент A_N должен умно-

жаться на несколько чисел. Данное обстоятельство не требует дополнительных действий по синхронизации в силу особенностей реализации функции MPI_Accumulate.

MPI7Win6. В главном процессе дан вещественный массив A размера $2K - 1$ (K — количество подчиненных процессов), в каждом подчиненном процессе дан вещественный массив B размера R, где R — ранг процесса (1, 2, …, $K - 1$). В главном процессе определить окно доступа, содержащее массив A, и, используя функцию MPI_Accumulate в каждом подчиненном процессе, прибавить к элементам массива A, начиная с индекса $R - 1$, значения всех элементов массива B из процесса ранга R, после чего вывести в главном процессе измененный массив A (единственный элемент B_0 из процесса 1 прибавляется к элементу A_0, элементы B_0 и B_1 из процесса 2 прибавляются соответственно к элементам A_1 и A_2, элементы B_0, B_1 и B_2 из процесса 3 прибавляются соответственно к элементам A_2, A_3 и A_4, и т. д.).

Примечание. К элементам массива A, начиная с индекса 2, потребуется прибавлять несколько значений из разных подчиненных процессов. Данное обстоятельство не требует дополнительных действий по синхронизации в силу особенностей реализации функции MPI_Accumulate.

MPI7Win7. В главном процессе дан массив A из $2K$ целых чисел, где K — количество подчиненных процессов. Во всех подчиненных процессах определить окно доступа из двух целых чисел и, используя несколько вызовов функции MPI_Put в главном процессе, записать и вывести в каждом подчиненном процессе по два элемента из массива A, перебирая их в исходном порядке (элементы A_0 и A_1 надо вывести в процессе 1, элементы A_2 и A_3 — в процессе 2, и т. д.).

MPI7Win8. В каждом процессе дано целое число R и вещественное число B. Все числа R различны и лежат в диапазоне от 0 до $K - 1$, где K — количество процессов. Во всех процессах определить окно доступа из одного вещественного числа и, используя функцию MPI_Put в каждом процессе, переслать число B из этого процесса в процесс R, после чего вывести полученные числа во всех процессах.

MPI7Win9. В каждом процессе дан массив A из K целых чисел, где K — количество процессов. Во всех процессах определить окно доступа, содержащее массив A, и, используя несколько вызовов функции MPI_Get в каждом процессе R ($R = 0, \ldots, K - 1$), получить и вывести элементы всех массивов A с индексом R, перебирая эти элементы в порядке убывания рангов содержащих их процессов (вначале выводится элемент, полученный из процесса ранга $K - 1$, затем элемент из процесса ранга $K - 2$, и т. д.).

Примечание. Функцию MPI_Get, как и другие функции, обеспечивающие односторонние коммуникации, можно использовать и для доступа к окну, определенному в вызывающем процессе.

MPI7Win10. В каждом процессе дан массив A из 5 вещественных чисел и целые числа N_1 и N_2, каждое из которых лежит в диапазоне от 0 до 4. Во всех процессах определить окно доступа, содержащее массив A, и, используя по два вызова функции MPI_Get в каждом процессе, получить из предыдущего процесса элемент его массива A с индексом N_1, а из последующего процесса — элемент с индексом N_2 (числа N_1 и N_2 берутся из вызывающего процесса, процессы перебираются циклически). Вывести в каждом процессе полученные числа в указанном порядке.

MPI7Win11. Количество процессов K — четное число. В каждом процессе дан массив A из $K/2$ целых чисел. Во всех процессах нечетного ранга (1, 3, …, $K-1$) определить окно доступа, содержащее массив A, и, используя требуемое число вызовов функции MPI_Accumulate в каждом процессе четного ранга, добавить элемент $A[I]$ из процесса ранга $2J$ к элементу $A[J]$ из процесса ранга $2I+1$ ($I, J = 0, \ldots, K/2-1$) и вывести измененные массивы A во всех процессах нечетного ранга.

Указание. Требуемое преобразование можно описать по-другому: если через B обозначить матрицу порядка $K/2$, строки которой совпадают с массивами A из процессов четного ранга, а через C — матрицу того же порядка, строки которой совпадают с массивами A из процессов нечетного ранга, то преобразование состоит в прибавлении к I-му столбцу матрицы C соответствующих элементов I-й строки матрицы B.

MPI7Win12. Решить задачу MPI7Win11, определив окна доступа в процессах четного ранга и используя вместо функций MPI_Accumulate функции MPI_Get в процессах нечетного ранга.

Указание. Поскольку числа, полученные из процессов четного ранга, необходимо добавлять к элементам массива A после вызова функции синхронизации MPI_Win_fence. удобно использовать вспомогательный массив для хранения полученных чисел.

MPI7Win13. В каждом процессе даны три целых числа N_1, N_2, N_3, каждое из которых лежит в диапазоне от 0 до $K-1$, где K — количество процессов (значения некоторых из этих чисел в каждом процессе могут совпадать). Кроме того, в каждом процессе дан массив A вещественных чисел размера $R+1$, где R — ранг процесса (0, …, $K-1$). Во всех процессах определить окно доступа, содержащее массив A, и, используя по три вызова функции MPI_Accumulate в каждом процессе, добавить ко всем элементам массива A в процессах рангов N_1, N_2 и N_3 ве-

щественное число, равное $R + 1$, где R — ранг процесса, вызвавшего функции MPI_Accumulate (например, если число N_1 в процессе ранга 3 равно 2, то ко всем элементам массива A из процесса 2 надо добавить вещественное число 4.0). Если некоторые из чисел N_1, N_2, N_3 в процессе R совпадают, то числа $R + 1$ надо добавлять к элементам соответствующих массивов несколько раз. Вывести измененные массивы A в каждом процессе.

MPI7Win14. В каждом процессе дан массив вещественных чисел размера K (K — количество процессов), содержащий строку верхнетреугольной матрицы A, включая ее начальную нулевую часть (процесс ранга R содержит R-ю строку матрицы в предположении, что строки нумеруются от 0). Во всех процессах определить окно доступа, содержащее исходный массив, и, используя требуемое количество вызовов функции MPI_Get в каждом процессе, записать в эти массивы (и затем вывести) строки матрицы, транспонированной к исходной матрице A, включая ее завершающую нулевую часть. Вспомогательные массивы не использовать.

Указания. (1) Строки транспонированной матрицы совпадают со столбцами исходной матрицы, таким образом, полученная матрица будет нижнетреугольной. (2) Обнулять требуемые элементы массивов перед их выводом необходимо после второго вызова функции MPI_Win_fence. (3) Окно доступа для последнего процесса можно не создавать.

MPI7Win15. Решить задачу MPI7Win14, используя вместо вызовов функций MPI_Get вызовы функций MPI_Put.

Указание. В данном случае можно не создавать окно доступа для главного процесса.

MPI7Win16. В каждом процессе дана одна строка вещественной квадратной матрицы A порядка K, где K — количество процессов (процесс ранга R содержит R-ю строку матрицы в предположении, что строки нумеруются от 0). Кроме того, в каждом процессе дано вещественное число B. Во всех процессах определить окно доступа, содержащее строку матрицы A, и, используя требуемое число вызовов функции MPI_Accumulate в каждом процессе ранга R ($R = 0, \ldots, K - 1$), заменить в строке матрицы из следующего процесса все элементы, меньшие числа B из процесса R, на это число (процессы перебираются циклически). Затем, используя K вызовов функции MPI_Get в каждом процессе, получить и вывести столбец преобразованной матрицы A с номером, совпадающим с рангом процесса (столбцы также нумеруются от 0).

Указание. При выполнении этого задания в каждом процессе необходимо *трижды* вызывать функцию синхронизации MPI_Win_fence.

MPI7Win17. В каждом процессе дана одна строка вещественной квадратной матрицы A порядка K, где K — количество процессов (процесс ранга R содержит R-ю строку матрицы в предположении, что строки нумеруются от 0). Кроме того, в каждом процессе дано вещественное число B. Во всех процессах определить окно доступа, содержащее строку матрицы A, и, используя требуемое число вызовов функции MPI_Accumulate в каждом процессе ранга R ($R = 0, \ldots, K - 1$), заменить в строке матрицы из предыдущего процесса все элементы, большие числа B из процесса R, на это число (процессы перебираются циклически). Затем, используя K вызовов функции MPI_Accumulate в каждом подчиненном процессе, добавить начальный элемент строки из каждого подчиненного процесса ранга R ($1, \ldots, K - 1$) ко всем элементам столбца преобразованной матрицы A с номером R (столбцы также нумеруются от 0). После всех описанных преобразований вывести в каждом процессе новое содержимое соответствующей строки матрицы A.

Указание. При выполнении этого задания в каждом процессе необходимо *трижды* вызывать функцию синхронизации MPI_Win_fence.

7.2. Дополнительные виды синхронизации

MPI7Win18. Количество процессов K — четное число. В каждом процессе четного ранга ($0, 2, \ldots, K - 2$) дано целое число A. Во всех процессах нечетного ранга ($1, 3, \ldots, K - 1$) определить окно доступа из одного целого числа и, используя функцию MPI_Put в каждом процессе четного ранга $2N$, переслать число A в процесс ранга $2N + 1$ и вывести его в этом процессе. Для синхронизации использовать функции MPI_Win_start и MPI_Win_complete в процессах четного ранга и функции MPI_Win_post и MPI_Win_wait в процессах нечетного ранга; для создания группы процессов, указываемой в качестве первого параметра функций MPI_Win_start и MPI_Win_post, использовать функцию MPI_Group_incl, применив ее к группе, полученной из коммуникатора MPI_COMM_WORLD (с помощью функции MPI_Comm_group).

Примечание. В отличие от *коллективной* функции MPI_Win_fence, использовавшейся в предыдущих заданиях данной группы, синхронизирующие функции, используемые в этом и последующих заданиях, являются *локальными* и, кроме того, позволяют явным образом определить группы инициирующих (origin) и целевых (target) процессов при односторонних коммуникациях.

MPI7Win19. В главном процессе дан массив A из K вещественных чисел, где K — количество подчиненных процессов. В главном процессе определить окно доступа, содержащее массив A, и, используя функцию MPI_Get в каждом подчиненном процессе, получить и вывести один из элементов массива A, перебирая элементы в обратном порядке (в процессе ранга 1 надо получить элемент массива с индексом $K - 1$, в процессе ранга 2 — элемент с индексом $K - 2$, и т. д.). Для синхронизации использовать функции MPI_Win_start и MPI_Win_complete в подчиненных процессах и функции MPI_Win_post и MPI_Win_wait в главном процессе; для создания группы процессов, указываемой в качестве первого параметра функции MPI_Win_start, использовать функцию MPI_Group_incl, для создания группы, указываемой в функции MPI_Win_post, использовать функцию MPI_Group_excl (применив обе эти функции к группе, полученной из коммуникатора MPI_COMM_WORLD).

MPI7Win20. Число процессов K кратно 3. В процессах ранга $3N$ ($N = 0, \ldots, K/3 - 1$) дан массив A из трех вещественных чисел. Во всех процессах, в которых дан массив, определить окно доступа, содержащее этот массив, и, используя по одному вызову функции MPI_Get в процессах ранга $3N + 1$ и $3N + 2$ ($N = 0, \ldots, K/3 - 1$), прочесть и вывести соответственно один элемент A_0 и два элемента A_1 и A_2 из процесса ранга $3N$ (процесс 1 должен вывести элемент A_0, полученный из процесса 0, процесс 2 — элементы A_1 и A_2, полученные из процесса 0, процесс 4 — элемент A_0, полученный из процесса 3, и т. д.). Для синхронизации использовать функции MPI_Win_post и MPI_Win_wait в процессах ранга $3N$ и функции MPI_Win_start и MPI_Win_complete в остальных процессах.

MPI7Win21. Количество процессов K — четное число. В главном процессе дан массив A из $K/2$ вещественных чисел и массив N целых чисел того же размера. Все элементы массива N различны и лежат в диапазоне от 1 до $K - 1$. В каждом подчиненном процессе определить окно доступа из одного вещественного числа и, используя требуемое количество вызовов функции MPI_Put в главном процессе, переслать в каждый из подчиненных процессов ранга N_I ($I = 0, \ldots, K/2 - 1$) число A_I и вывести полученное число (в остальных подчиненных процессах вывести вещественное число 0.0). Для синхронизации использовать функции MPI_Win_post и MPI_Win_wait в подчиненных процессах и функции MPI_Win_start и MPI_Win_complete в главном процессе.

MPI7Win22. В главном процессе дан массив A вещественных чисел размера K (K — количество подчиненных процессов) и массив N целых чисел размера 8. Все элементы массива N лежат в диапазоне от 1 до K;

некоторые элементы данного массива могут совпадать. В каждом подчиненном процессе дан вещественный массив B размера R, где R — ранг процесса ($R = 1, \ldots, K$). Определить в каждом подчиненном процессе окно доступа, содержащее массив B, и, используя требуемое количество вызовов функции MPI_Accumulate в главном процессе, добавить ко всем элементам массивов B из процессов ранга N_I ($I = 0, \ldots, 7$) элементы массива A с теми же индексами (к элементу B_0 добавляется элемент A_0, к элементу B_1 — элемент A_1, и т. д.); для некоторых массивов B элементы из массива A могут добавляться несколько раз. В каждом подчиненном процессе вывести массив B (который либо изменится, либо сохранит исходные значения элементов). Для синхронизации использовать функции MPI_Win_post и MPI_Win_wait в подчиненных процессах и функции MPI_Win_start и MPI_Win_complete в главном процессе.

MPI7Win23. Во всех процессах даны вещественные массивы A размера 5. Кроме того, в главном процессе даны целочисленные массивы N и M размера 5 каждый. Все элементы массива N лежат в диапазоне от 1 до K, где K — количество подчиненных процессов, все элементы массива M лежат в диапазоне от 0 до 4; некоторые элементы как в массиве N, так и в массиве M могут совпадать. В каждом подчиненном процессе определить окно доступа, содержащее массив A, и, используя требуемое количество вызовов функции MPI_Get в главном процессе, получить из процесса ранга N_I ($I = 0, \ldots, 4$) элемент массива A с индексом M_I и добавить его значение к элементу массива A главного процесса с индексом I. После изменения массива A в главном процессе выполнить следующую корректировку массивов A всех подчиненных процессов: заменить в них те элементы, которые больше элемента массива A с тем же индексом из главного процесса, на этот элемент, используя требуемое количество вызовов функции MPI_Accumulate в главном процессе. В каждом процессе вывести преобразованные массивы A. Для синхронизации использовать два вызова пары функции MPI_Win_post и MPI_Win_wait в подчиненных процессах и два вызова пары функции MPI_Win_start и MPI_Win_complete в главном процессе.

MPI7Win24. В каждом подчиненном процессе дано целое число N; все числа N различны и лежат в диапазоне от 0 до $K - 1$, где K — количество подчиненных процессов. В главном процессе определить окно доступа, содержащее целочисленный массив A размера K. Не выполняя никаких вызовов синхронизирующих функций в главном процессе (кроме вызова функции MPI_Barrier) и используя в подчиненных процессах последовательность вызовов синхронизирующих функции MPI_Win_lock, MPI_Win_unlock, MPI_Barrier, MPI_Win_lock,

MPI_Win_unlock, записать в элементы массива A с индексом N ранг подчиненного процесса, содержащего данное значение N (с помощью функции MPI_Put), после чего получить и вывести в каждом подчиненном процессе все элементы преобразованного массива A (с помощью функции MPI_Get). В качестве первого параметра функции MPI_Win_lock указывать MPI_LOCK_SHARED.

Примечание. Синхронизирующие функции MPI_Win_lock, MPI_Win_unlock используются преимущественно при односторонних коммуникациях с *пассивными целевыми процессами* (passive targets), при которых целевой процесс не обрабатывает переданные ему данные, а выступает в роли их хранилища, доступного для других процессов.

MPI7Win25. Количество процессов K кратно 3. В процессах ранга $3N$ ($N = 0, \ldots, K/3 - 1$) дан вещественный массив A размера 5, в процессах ранга $3N + 1$ дано целое число M, лежащее в диапазоне от 0 до 4, и вещественное число B. В каждом процессе, содержащем исходный массив A, определить окно доступа с этим массивом. Используя функцию MPI_Accumulate в каждом процессе ранга $3N + 1$ ($N = 0, \ldots, K/3 - 1$), преобразовать массив A из процесса ранга $3N$ следующим образом: если элемент массива с индексом M больше числа B, то он заменяется на число B (числа M и B берутся из процесса ранга $3N + 1$). После этого, используя функцию MPI_Get в каждом процессе ранга $3N + 2$, получить и вывести в нем все элементы преобразованного массива A из процесса ранга $3N$. Использовать синхронизирующие функции MPI_Win_Lock, MPI_Win_unlock, MPI_Barrier в процессах ранга $3N + 1$, функции MPI_Barrier, MPI_Win_Lock, MPI_Win_unlock в процессах ранга $3N + 2$ и функцию MPI_Barrier в процессах ранга $3N$. В качестве первого параметра функции MPI_Win_lock указывать MPI_LOCK_EXCLUSIVE.

MPI7Win26. В каждом подчиненном процессе дан вещественный массив A размера 5 с положительными элементами. В главном процессе определить окно доступа, содержащее вещественный массив B размера 5 с нулевыми элементами. Не выполняя никаких вызовов синхронизирующих функций в главном процессе (кроме вызова функции MPI_Barrier) и используя в подчиненных процессах последовательность вызовов синхронизирующих функции MPI_Win_lock, MPI_Win_unlock, MPI_Barrier, MPI_Win_lock, MPI_Win_unlock, записать в каждый из элементов массива B максимальный из элементов массивов A с тем же индексом (с помощью функции MPI_Accumulate), после чего получить и вывести в каждом подчиненном процессе все элементы преобразованного массива B (с помощью функции

MPI_Get). В качестве первого параметра функции MPI_Win_lock указывать MPI_LOCK_SHARED.

MPI7Win27. В каждом подчиненном процессе даны два вещественных числа X, Y — координаты точки на плоскости. Используя вызовы функции MPI_Get в главном процессе, получить в нем числа X_0, Y_0, равные координатам той точки (X, Y) среди точек, данных в подчиненных процессах, которая является наиболее удаленной от начала координат. После этого, используя вызовы функции MPI_Get в подчиненных процессах, получить и вывести в каждом из них числа X_0, Y_0, найденные в главном процессе. В каждом процессе определить окно доступа, содержащее два вещественных числа (X, Y для подчиненных процессов, X_0, Y_0 для главного процесса). Использовать синхронизирующие функции MPI_Win_lock, MPI_Win_unlock, MPI_Barrier в главном процессе и функции MPI_Barrier, MPI_Win_lock, MPI_Win_unlock в подчиненных процессах.

Примечание. Данную задачу невозможно решить, используя односторонние коммуникации только на стороне подчиненных процессов и при этом выполняя синхронизацию с помощью блокировок lock/unlock.

MPI7Win28. Решить задачу MPI7Win27, используя единственное окно доступа в главном процессе, содержащее числа X_0, Y_0. Для нахождения чисел X_0, Y_0 использовать функцию MPI_Get и функцию MPI_Put в подчиненных процессах (для некоторых процессов вызывать функцию MPI_Put не потребуется), для пересылки найденных чисел X_0, Y_0 во все подчиненные процессы использовать в них функцию MPI_Get (как и в задаче MPI7Win27). Для синхронизации обменов на этапе нахождения чисел X_0, Y_0 использовать два вызова пары функций MPI_Win_start и MPI_Win_complete в подчиненных процессах и вызовы *в цикле* двух пар функций MPI_Win_post и MPI_Win_wait в главном процессе (при этом на каждой итерации цикла необходимо определять *новую группу процессов*, используемую при вызове функций MPI_Win_post). Для синхронизации действий при пересылке чисел X_0, Y_0 в подчиненные процессы использовать, как и в задаче MPI7Win27, функцию MPI_Barrier в главном процессе и функции MPI_Barrier, MPI_Win_Lock, MPI_Win_unlock в подчиненных процессах.

Примечание. Описанный вариант решения (в отличие от варианта, приведенного в задаче MPI7Win27) позволяет использовать односторонние коммуникации только на стороне подчиненных процессов, однако при этом на первом этапе решения требуется применять вариант синхронизации, отличный от блокировок lock/unlock.

MPI7Win29. В каждом процессе дана одна строка целочисленной квадратной матрицы порядка *K*, где *K* — количество процессов (в процессе ранга *R* дана строка матрицы с номером *R*; строки нумеруются от 0). Используя вызовы функции MPI_Get в главном процессе, получить в нем строку матрицы с минимальной суммой элементов *S* и, кроме того, найти количество *N* строк с минимальной суммой (если $N > 1$, то в главном процессе надо сохранить последнюю из таких строк, т. е. строку с наибольшим номером). После этого, используя вызовы функции MPI_Get в подчиненных процессах, получить и вывести в каждом из них строку с минимальной суммой *S*, найденную в главном процессе, значение суммы *S* и количество *N* строк с минимальной суммой. В каждом процессе определить окно доступа, содержащее $K + 2$ целых числа; в первых *K* элементах окна содержатся элементы строки матрицы, в следующем элементе — сумма их значений *S*, а последний элемент предназначен для хранения числа *N*. Использовать синхронизирующие функции MPI_Win_lock, MPI_Win_unlock, MPI_Barrier в главном процессе и функции MPI_Barrier, MPI_Win_lock, MPI_Win_unlock в подчиненных процессах.

Примечание. Данную задачу невозможно решить, используя односторонние коммуникации только на стороне подчиненных процессов и выполняя синхронизацию с помощью блокировок lock/unlock.

MPI7Win30. Решить задачу MPI7Win29, используя единственное окно доступа в главном процессе, содержащее строку матрицы и числа *S* и *N*. Для нахождения строки с минимальной суммой элементов и связанных с ней характеристик использовать функцию MPI_Get и функцию MPI_Put в подчиненных процессах (для некоторых процессов вызывать функцию MPI_Put не потребуется), для пересылки найденных данных во все подчиненные процессы использовать в них функцию MPI_Get (как и в задаче MPI7Win29). На этапе нахождения строки с минимальной суммой функцию MPI_Get следует использовать только для получения значений *S* и *N*. Для синхронизации обменов на этапе нахождения строки с минимальной суммой использовать два вызова пары функций MPI_Win_start и MPI_Win_complete в подчиненных процессах и вызовы *в цикле* двух пар функций MPI_Win_post и MPI_Win_wait в главном процессе (при этом на каждой итерации цикла необходимо определять *новую группу процессов*, используемую при вызове функций MPI_Win_post). Для синхронизации действий при пересылке найденных данных в подчиненные процессы использовать, как и в задаче MPI7Win29, функцию MPI_Barrier в главном процессе и функции MPI_Barrier, MPI_Win_Lock, MPI_Win_unlock в подчиненных процессах.

Примечание. Описанный вариант решения (в отличие от варианта, приведенного в задаче MPI7Win29) позволяет использовать односторонние коммуникации только на стороне подчиненных процессов, однако при этом на первом этапе решения требуется применять вариант синхронизации, отличный от блокировок lock/unlock.

8. Интеркоммуникаторы и динамическое создание процессов (MPI-2)

Базовые возможности, связанные с созданием интеркоммуникаторов и их использованием для обмена сообщениями между отдельными процессами, определены в стандарте MPI-1. Поэтому 5 заданий данной группы можно выполнять с применением системы MPICH 1.2.5 (это задания MPI8Inter1–MPI8Inter4 и MPI8Inter9). Прочие задания посвящены новым возможностям по созданию интеркоммуникаторов (MPI8Inter5–MPI8Inter8), средствам коллективного обмена для интеркоммуникаторов (MPI8Inter10–MPI8Inter14) и использованию интеркоммуникаторов для динамического создания процессов (MPI8Inter15–MPI8Inter22). Все эти возможности появились в стандарте MPI-2, поэтому для выполнения заданий необходимо подключать к программе систему MPICH2 1.3.

В качестве коммуникатора-*посредника* (третьего параметра peer функции MPI_Intercomm_create) следует указывать *копию* коммуникатора MPI_COMM_WORLD, созданную с помощью функции MPI_Comm_dup.

При динамическом создании процессов с помощью функции MPI_Comm_spawn в заданиях MPI8Inter15–MPI8Inter22 в качестве первого параметра command следует указывать имя исполняемого файла ptprj.exe, в качестве второго параметра argv достаточно указать константу NULL, в качестве четвертого параметра info — константу MPI_NULL_INFO, в качестве последнего параметра array_of_errcodes — константу MPI_ERRCODES_IGNORE. Если в задании не указано, какой коммуникатор должен использоваться в качестве исходного при создании новых процессов, то предполагается, что этим коммуникатором является MPI_COMM_WORLD.

Вместо строки «ptprj.exe» можно использовать реализованную в задачнике функцию char* GetExename(), которая возвращает полное имя исполняемого файла.

8.1. Создание интеркоммуникаторов

MPI8Inter1. Количество процессов K — четное число. В каждом процессе дано целое число X. Используя функции MPI_Comm_group, MPI_Group_range_incl и MPI_Comm_create, создать коммуникаторы, первый из которых содержит процессы четного ранга в том же поряд-

ке (0, 2, …, $K/2 - 2$), а второй — процессы нечетного ранга в том же порядке (1, 3, …, $K/2 - 1$). Вывести ранги процессов R в созданных коммуникаторах. Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. Используя функции MPI_Send и MPI_Recv для созданного интеркоммуникатора, получить для каждого процесса число X из процесса того же ранга, входящего в другую группу этого же интеркоммуникатора, и вывести полученные числа.

MPI8Inter2. Количество процессов K — четное число. В каждом процессе дано целое число C и вещественное число X. Числа C равны либо 0, либо 1, количество значений 1 равно количеству значений 0, причем в процессе ранга 0 дано C, равное 0, а в процессе ранга $K - 1$ дано C, равное 1. Используя один вызов функции MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 0$ в том же порядке, а второй — процессы со значениями $C = 1$ в обратном порядке. Вывести ранги процессов R в созданных коммуникаторах (при этом значение $R = 0$ получат первый и последний процессы исходного коммуникатора MPI_COMM_WORLD). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. Используя функции MPI_Send и MPI_Recv для созданного интеркоммуникатора, получить для каждого процесса число X из процесса того же ранга, входящего в другую группу этого интеркоммуникатора, и вывести полученные числа.

MPI8Inter3. Количество процессов K кратно 3. В процессах ранга $3N$ ($N = 0$, …, $3K - 3$) дано вещественное число X, в процессах ранга $3N + 1$ даны вещественные числа X и Y, в процессах ранга $3N + 2$ дано вещественное число Y. Используя функции MPI_Comm_group, MPI_Group_range_incl и MPI_Comm_create, создать коммуникаторы, первый из которых содержит группу процессов ранга $3N$ в том же порядке (0, 3, …, $K - 3$), второй — группу процессов ранга $3N + 1$ в обратном порядке ($K - 2$, $K - 5$, …, 1), а третий — группу процессов ранга $3N + 2$ в том же порядке (2, 5, …, $K - 1$). Вывести ранги процессов R в созданных коммуникаторах. Затем, используя функцию MPI_Intercomm_create, объединить созданные коммуникаторы в два интеркоммуникатора: первый должен содержать первую и вторую группы процессов, а второй — вторую и третью группы. Используя функции MPI_Send и MPI_Recv для созданных интеркоммуникаторов, поменять числа X для процессов одинакового ранга, входящих в первую и вторую группу, и числа Y для процессов одинакового ранга, входящих во вторую и третью группу. Вывести в каждом процессе полученные числа.

Указание. Следует обратить внимание на то, что функция MPI_Intercomm_create должна вызываться один раз для процессов, входящих в первую и третью группы, и два раза для процессов, входящих во вторую группу, и такое же количество вызовов должно быть для функций MPI_Send и MPI_Recv.

MPI8Inter4. Количество процессов K кратно 3. В каждом процессе даны три целых числа. Первое число (обозначаемое буквой C) лежит в диапазоне от 0 до 2, причем каждое из значений 0, 1, 2 встречается одинаковое число раз (равное $K/3$) и при этом в процессах ранга 0, 1, 2 значения C совпадают с рангами этих процессов. Используя один вызов функции MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 0$ в том же порядке, второй — процессы со значениями $C = 1$ в том же порядке, а третий — процессы со значениями $C = 2$ в том же порядке. Вывести ранги процессов R в созданных коммуникаторах (при этом значение $R = 0$ получат первые три процесса исходного коммуникатора MPI_COMM_WORLD). Затем, используя по два вызова функции MPI_Intercomm_create в каждом процессе, объединить созданные коммуникаторы в три интеркоммуникатора: первый должен содержать группы процессов со значениями C, равными 0 и 1, второй — группы со значениями 1 и 2, третий — группы со значениями 0 и 2 (таким образом, созданные интеркоммуникаторы будут образовывать кольцо, связывающее все три ранее созданные группы). Считая, что в первой группе процессов два следующих исходных числа обозначаются буквами X и Y, во второй группе — Y и Z, а в третьей — Z и X (в указанном порядке) и используя в каждом процессе по два вызова функций MPI_Send и MPI_Recv для созданных интеркоммуникаторов, поменять числа X для процессов одинакового ранга, входящих в первую и вторую группу, числа Y — для процессов одинакового ранга, входящих во вторую и третью группу, и числа Z — для процессов одинакового ранга, входящих в первую и третью группу. Вывести в каждом процессе полученные числа.

MPI8Inter5. Количество процессов K кратно 4. В каждом процессе дано целое число X. Используя функции MPI_Comm_group, MPI_Group_range_incl и MPI_Comm_create, создать коммуникаторы, первый из которых содержит первую половину процессов (с рангами 0, 1, …, $K/2 - 1$ в указанном порядке), а второй — вторую половину (с рангами $K/2$, $K/2 + 1$, …, $K - 1$ в указанном порядке). Вывести ранги процессов R_1 в созданных коммуникаторах. Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. После этого, используя функцию MPI_Comm_create для созданного интеркоммуникатора, получить но-

вый интеркоммуникатор, первая группа которого содержит процессы из первой группы исходного интеркоммуникатора с четными рангами (в том же порядке), а вторая группа — процессы из второй группы исходного интеркоммуникатора с нечетными рангами в обратном порядке (таким образом, в первую группу нового интеркоммуникатора будут входить процессы исходного коммуникатора MPI_COMM_WORLD с рангами 0, 2, …, $K/2 - 2$, а во вторую — с рангами $K - 1$, $K - 3$, …, $K/2 + 1$). Вывести ранги процессов R_2, входящих в новый интеркоммуникатор. Используя функции MPI_Send и MPI_Recv для нового интеркоммуникатора, получить для каждого процесса число X из процесса того же ранга, входящего в другую группу этого интеркоммуникатора, и вывести полученные числа.

MPI8Inter6. Количество процессов K кратно 4. В каждом процессе дано вещественное число X. Используя функции MPI_Comm_group, MPI_Group_range_incl и MPI_Comm_create, создать коммуникаторы, первый из которых содержит первую половину процессов (с рангами 0, 1, …, $K/2 - 1$ в указанном порядке), а второй — вторую половину (с рангами $K/2$, $K/2 + 1$, …, $K - 1$ в указанном порядке). Вывести ранги процессов R_1 в созданных коммуникаторах. Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. После этого, используя один вызов функции MPI_Comm_split для созданного интеркоммуникатора, получить два новых интеркоммуникатора. Первый из новых интеркоммуникаторов содержит процессы исходного интеркоммуникатора с четными рангами, а второй — с нечетными, причем процессы во второй группе каждого из новых интеркоммуникаторов должны располагаться в обратном порядке (таким образом, если использовать ранги процессов для исходного коммуникатора MPI_COMM_WORLD, то первый интеркоммуникатор содержит группы процессов ранга 0, 2, …, $K/2 - 2$ и $K - 2$, $K - 4$, …, $K/2$, а второй — группы процессов ранга 1, 3, …, $K/2 - 1$ и $K - 1$, $K - 3$, …, $K/2 + 1$). Вывести ранги процессов R_2, входящих в новые интеркоммуникаторы. Используя функции MPI_Send и MPI_Recv для новых интеркоммуникаторов, получить для каждого процесса число X из процесса того же ранга, входящего в другую группу этого интеркоммуникатора, и вывести полученные числа.

MPI8Inter7. Количество процессов K — четное число. В каждом процессе дано целое число C, равное либо 0, либо 1, причем известно, что в первой половине процессов дано единственное значение $C = 1$, а во второй половине количество значений $C = 1$ больше одного и, кроме того, имеется хотя бы одно значение $C = 0$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит первую половину процессов (с рангами 0, 1, …, $K/2 - 1$ в указан-

ном порядке), а второй — вторую половину (с рангами $K/2$, $K/2 + 1$, …, $K - 1$ в указанном порядке). Вывести ранги процессов R_1 в созданных коммуникаторах. Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. После этого, используя функцию MPI_Comm_split для созданного интеркоммуникатора, получить новый интеркоммуникатор, группы которого содержат процессы из соответствующих групп исходного интеркоммуникатора со значениями $C = 1$, взятые в обратном порядке (таким образом, в первую группу нового интеркоммуникатора будет входить единственный процесс, а количество процессов во второй группе будет лежать в диапазоне от 2 до $K/2 - 1$). Вывести ранги процессов R_2, входящих в ту группу нового интеркоммуникатора, которая содержит более одного процесса. В единственном процессе первой группы нового интеркоммуникатора ввести массив Y из N целых чисел, где N — количество процессов во второй группе; в каждом процессе второй группы этого же интеркоммуникатора ввести по одному целому числу X. Используя требуемое количество вызовов функций MPI_Send и MPI_Recv для всех процессов нового интеркоммуникатора, получить в процессе первой группы все числа X из процессов второй группы (в порядке возрастания рангов этих процессов), а в каждом процессе второй группы ранга R_2 (0, 1, …, $N - 1$) получить элемент с индексом R_2 из массива Y, данного в процессе первой группы. Вывести полученные числа.

Примечание. В реализации MPICH 2 версии 1.3, используемой в задачнике, функция MPI_Comm_split приводит к ошибочной программе, если она применяется к интеркоммуникатору и при этом какие-либо из значений ее параметра color равны MPI_UNDEFINED. Таким образом, для корректной работы программ необходимо использовать только *неотрицательные* значения color. Кроме того, программа в дальнейшем может вести себя некорректно, если в результате применения функции MPI_Comm_split к интеркоммуникатору создаются *пустые группы* (это возможно, если для всех процессов одной из групп исходного интеркоммуникатора указываются одинаковые значения color, отличные от некоторых значений color для процессов другой группы).

MPI8Inter8. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что и у процессов четного ранга, и у процессов нечетного ранга имеется хотя бы по одному из значений 0, 1, 2. Используя один вызов функции MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы с четными рангами (в порядке возрастания рангов), а второй — процессы с нечетными рангами (также в порядке возрастания рангов). Вывести ранги процессов R_1 в созданных коммуникаторах. Затем объединить созданные комму-

никаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. После этого, используя один вызов функции MPI_Comm_split для созданного интеркоммуникатора, получить три новых интеркоммуникатора, группы которых содержат процессы из соответствующих групп исходного интеркоммуникатора с одинаковыми значениями C, взятые в том же порядке (таким образом, в первую группу первого интеркоммуникатора будут входить процессы четного ранга со значениями $C = 0$, а, например, во вторую группу третьего интеркоммуникатора будут входить процессы нечетного ранга со значениями $C = 2$). Вывести ранги процессов R_2 в новых интеркоммуникаторах. В процессах из первых групп созданных интеркоммуникаторов ввести по одному целому числу X, а в процессах из вторых групп — по одному целому числу Y. Используя требуемое количество вызовов функций MPI_Send и MPI_Recv для всех процессов новых интеркоммуникаторов, переслать все числа X каждому из процессов второй группы этого же коммуникатора, а все числа Y — каждому из процессов первой группы и вывести полученные числа в порядке возрастания рангов переславших их процессов.

MPI8Inter9. Количество процессов K — четное число. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что в процессе ранга 0 дано число $C = 1$, а первое из значений $C = 2$ имеется у процесса ранга $K/2$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 1$ в том же порядке, а второй — процессы со значениями $C = 2$ в том же порядке. Вывести ранги процессов R_1 в созданных коммуникаторах (если процесс не входит ни в один из созданных коммуникаторов, то вывести для него число −1). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create. После этого ввести в процессах первой группы созданного интеркоммуникатора (соответствующей значениям $C = 1$) по одному целому числу X, а в процессах второй группы — по одному целому числу Y. Используя требуемое количество вызовов функций MPI_Send и MPI_Recv для всех процессов созданного интеркоммуникатора, переслать все числа X каждому из процессов второй группы этого коммуникатора, а все числа Y — каждому из процессов первой группы и вывести полученные числа в порядке возрастания рангов переславших их процессов.

8.2. Коллективные операции для интеркоммуникаторов

MPI8Inter10. Количество процессов K — четное число. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что в процессе ранга 0 дано число $C = 1$, а первое из значений $C = 2$

имеется у процесса ранга $K/2$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 1$ в том же порядке, а второй — процессы со значениями $C = 2$ в том же порядке. Вывести ранги процессов R в созданных коммуникаторах (если процесс не входит ни в один из созданных коммуникаторов, то вывести для него число −1). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create (группа, содержащая процессы со значениями $C = 1$, считается первой группой созданного интеркоммуникатора, а группа процессов со значениями $C = 2$ — второй). После этого ввести в процессах обеих групп созданного интеркоммуникатора целые числа R_1 и R_2. Значения чисел R_1 во всех процессах совпадают и указывают ранг выделенного процесса первой группы; значения чисел R_2 во всех процессах также совпадают и указывают ранг выделенного процесса второй группы. В выделенном процессе первой группы дан набор из трех целых чисел X, в выделенном процессе второй группы — набор из трех целых чисел Y. Используя по два вызова коллективной функции MPI_Bcast в каждом процессе созданного интеркоммуникатора, переслать набор чисел X во все процессы второй группы, а набор чисел Y во все процессы первой группы, после чего вывести полученные числа.

MPI8Inter11. Количество процессов K — четное число. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что в процессе ранга 0 дано число $C = 1$, а первое из значений $C = 2$ имеется у процесса ранга $K/2$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 1$ в том же порядке, а второй — процессы со значениями $C = 2$ в том же порядке. Вывести ранги процессов R в созданных коммуникаторах (если процесс не входит ни в один из созданных коммуникаторов, то вывести для него число −1). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create (группа, содержащая процессы со значениями $C = 1$, считается первой группой созданного интеркоммуникатора, а группа процессов со значениями $C = 2$ — второй). После этого ввести в процессах обеих групп созданного интеркоммуникатора целое число R_1, которое совпадает во всех процессах и указывает ранг выделенного процесса первой группы. В выделенном процессе первой группы дан набор из K целых чисел X, где K — количество процессов во второй группе. Используя один вызов коллективной функции MPI_Scatter в каждом процессе созданного интеркоммуникатора, переслать по одному числу из набора X во все процессы второй группы (в порядке возрастания рангов процессов), после чего вывести полученные числа.

MPI8Inter12. Количество процессов K — четное число. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что в процессе ранга 0 дано число $C = 1$, а первое из значений $C = 2$ имеется у процесса ранга $K/2$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 1$ в том же порядке, а второй — процессы со значениями $C = 2$ в том же порядке. Вывести ранги процессов R в созданных коммуникаторах (если процесс не входит ни в один из созданных коммуникаторов, то вывести для него число −1). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create (группа, содержащая процессы со значениями $C = 1$, считается первой группой созданного интеркоммуникатора, а группа процессов со значениями $C = 2$ — второй). После этого ввести в процессах обеих групп созданного интеркоммуникатора целое число R_2, которое совпадает во всех процессах и указывает ранг выделенного процесса второй группы. В каждом процессе первой группы дано по одному целому числу X. Используя один вызов коллективной функции MPI_Gather в каждом процессе созданного интеркоммуникатора, переслать исходные числа X (в порядке возрастания рангов содержащих их процессов) в выделенный процесс второй группы, после чего вывести в этом процессе полученные числа.

MPI8Inter13. Количество процессов K — четное число. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что в процессе ранга 0 дано число $C = 1$, а первое из значений $C = 2$ имеется у процесса ранга $K/2$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 1$ в том же порядке, а второй — процессы со значениями $C = 2$ в том же порядке. Вывести ранги процессов R в созданных коммуникаторах (если процесс не входит ни в один из созданных коммуникаторов, то вывести для него число −1). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create (группа, содержащая процессы со значениями $C = 1$, считается первой группой созданного интеркоммуникатора, а группа процессов со значениями $C = 2$ — второй). В каждом процессе первой группы дано по одному целому числу X, в каждом процессе второй группы дано по одному целому числу Y. Используя один вызов коллективной функции MPI_Allreduce в каждом процессе созданного интеркоммуникатора, получить в каждом процессе первой группы число Y_{min} — минимальное из исходных чисел Y, а в каждом процессе второй группы число X_{max} — максимальное из исходных чисел X, после чего вывести полученные числа.

MPI8Inter14. Количество процессов K — четное число. В каждом процессе дано целое число C, лежащее в диапазоне от 0 до 2, причем известно, что в процессе ранга 0 дано число $C = 1$, а первое из значений $C = 2$ имеется у процесса ранга $K - 1$. Используя функцию MPI_Comm_split, создать коммуникаторы, первый из которых содержит процессы со значениями $C = 1$ в том же порядке, а второй — процессы со значениями $C = 2$ в обратном порядке. Вывести ранги процессов R в созданных коммуникаторах (если процесс не входит ни в один из созданных коммуникаторов, то вывести для него число −1). Затем объединить созданные коммуникаторы в интеркоммуникатор с помощью функции MPI_Intercomm_create (группа, содержащая процессы со значениями $C = 1$, считается первой группой созданного интеркоммуникатора, а группа процессов со значениями $C = 2$ — второй). В каждом процессе первой группы дан массив целых чисел X размера N_2, где N_2 — количество процессов во второй группе, в каждом процессе второй группы дан массив целых чисел Y размера N_1, где N_1 — количество процессов в первой группе. Используя один вызов коллективной функции MPI_Alltoall в каждом процессе созданного интеркоммуникатора, переслать в процесс ранга R_1 первой группы ($R_1 = 0, \ldots, N_1 - 1$) элементы с индексом R_1 из всех массивов Y, а в процесс ранга R_2 второй группы ($R_2 = 0, \ldots, N_2 - 1$) элементы с индексом R_2 из всех массивов X. Вывести полученные числа в порядке возрастания рангов переславших их процессов.

8.3. Динамическое создание процессов

MPI8Inter15. В каждом процессе дано вещественное число. Используя функцию MPI_Comm_spawn с первым параметром «ptprj.exe», создать один новый процесс. С помощью коллективной функции MPI_Reduce переслать в созданный процесс сумму исходных чисел и отобразить ее в разделе отладки, используя в этом процессе функцию Show. Затем с помощью коллективной функции MPI_Bcast переслать найденную сумму в исходные процессы и вывести эту сумму в каждом процессе.

MPI8Inter16. В каждом процессе дан массив из K вещественных чисел, где K — количество процессов. Используя один вызов функции MPI_Comm_spawn с первым параметром «ptprj.exe», создать K новых процессов. С помощью коллективной функции MPI_Reduce_scatter_block переслать в созданный процесс ранга R ($R = 0, \ldots, K - 1$) максимальный из элементов исходных массивов с индексом R и отобразить полученные максимальные элементы в разделе отладки, используя в каждом новом процессе функцию Show. Затем с помощью функций MPI_Send и MPI_Recv переслать найденный максимальный элемент из нового процесса ранга R ($R = 0, \ldots, K - 1$) в

исходный процесс того же ранга и вывести полученные элементы в исходных процессах.

MPI8Inter17. Количество процессов K — четное число. В процессах ранга 0 и 1 даны массивы вещественных чисел размера $K/2$. Используя один вызов функции MPI_Comm_spawn с первым параметром «ptprj.exe», создать два новых процесса. Используя один вызов функции MPI_Comm_split для интеркоммуникатора, связанного с созданными процессами, создать два новых интеркоммуникатора: первый содержит группу исходных процессов четного ранга (0, …, $K - 2$), а также, в качестве второй группы, первый из новых процессов (ранга 0), второй содержит группу исходных процессов нечетного ранга (1, …. $K - 1$), а также, в качестве второй группы, второй из новых процессов (ранга 1). Используя функции MPI_Send в исходных процессах и функции MPI_Recv в новых процессах, переслать все исходные числа из первого процесса первой группы каждого интеркоммуникатора в единственный процесс, входящий в его вторую группу. Отобразить полученные числа в разделе отладки, используя в новых процессах функцию Show. Затем с помощью коллективной функции MPI_Scatter для интеркоммуникаторов переслать по одному числу из новых процессов во все процессы первой группы соответствующего интеркоммуникатора (в порядке возрастания рангов процессов) и вывести полученные числа.

MPI8Inter18. Количество процессов K — четное число. В каждом процессе дан массив вещественных чисел размера $K/2$. Используя один вызов функции MPI_Comm_spawn с первым параметром «ptprj.exe», создать K новых процессов. Используя один вызов функции MPI_Comm_split для интеркоммуникатора, связанного с созданными процессами, создать два новых интеркоммуникатора: первый содержит группу исходных процессов четного ранга (0, …. $K - 2$), а также, в качестве второй группы, созданные процессы четного ранга, второй содержит группу исходных процессов нечетного ранга (1, …. $K - 1$), а также, в качестве второй группы, созданные процессы нечетного ранга. Для каждого из созданных интеркоммуникаторов выполнить следующие действия: среди элементов исходных массивов с индексом R ($R = 0, \ldots, K/2 - 1$), входящих в первую группу каждого интеркоммуникатора, найти минимальный (для первого интеркоммуникатора) или максимальный (для второго) и переслать этот экстремальный элемент в тот из новых процессов, который имеет ранг R во второй группе соответствующего интеркоммуникатора (например, минимальный из первых элементов массивов, которые даны в исходных процессах четного ранга, следует переслать в первый из созданных процессов, а максимальный из первых элементов массивов, которые даны в исходных процессах нечет-

ного ранга, следует переслать во второй из созданных процессов, поскольку этот процесс имеет ранг 0 в соответствующем интеркоммуникаторе). Для выполнения этих действий использовать коллективную функцию MPI_Reduce_scatter_block. Полученные экстремальные элементы отобразить в разделе отладки, используя в каждом новом процессе функцию Show. Затем с помощью коллективной функции MPI_Reduce найти минимальный элемент из тех минимумов, которые были получены во второй группе первого интеркоммуникатора, переслать и вывести его в первом процессе первой группы этого интеркоммуникатора (т. е. в процессе ранга 0 в исходном коммуникаторе MPI_COMM_WORLD), а также найти максимальный элемент из тех максимумов, которые были получены во второй группе второго интеркоммуникатора, переслать и вывести его в первом процессе первой группы этого интеркоммуникатора (т. е. в процессе ранга 1 в исходном коммуникаторе MPI_COMM_WORLD).

MPI8Inter19. В главном процессе дан массив A из $2K$ целых чисел, где K — количество процессов. Используя один вызов функции MPI_Comm_spawn с первым параметром «ptprj.exe», создать K новых процессов. С помощью коллективной функции MPI_Intercomm_merge для интеркоммуникатора, связанного с созданными процессами, создать новый интракоммуникатор, содержащий как исходные, так и новые процессы (параметр high функции MPI_Intercomm_merge необходимо задать таким образом, чтобы в созданном интракоммуникаторе вначале располагались исходные, а затем созданные процессы). Используя коллективную функцию MPI_Scatter для нового интракоммуникатора, переслать по одному элементу массива A из главного процесса во все процессы приложения (как исходные, так и созданные) в порядке возрастания их рангов в новом интракоммуникаторе. В исходных процессах вывести полученные числа, в новых процессах отобразить полученные числа в разделе отладки, используя в этих процессах функцию Show. Затем, используя коллективную функцию MPI_Reduce в этом же интракоммуникаторе, переслать в исходный процесс ранга 1 сумму всех чисел и вывести ее в этом процессе.

MPI8Inter20. Количество процессов K не делится на 4. В каждом процессе дано целое число A. Используя один вызов функции MPI_Comm_spawn с первым параметром «ptprj.exe», создать такое количество новых процессов (1, 2 или 3), чтобы общее число процессов K_0 в приложении стало кратным 4. В каждом из созданных процессов задать значение числа A, равное $-R-1$, где R — ранг созданного процесса. Используя функцию MPI_Intercomm_merge для интеркоммуникатора, связанного с созданными процессами, создать новый интракоммуникатор, содержащий как исходные, так и новые процессы (па-

раметр high функции MPI_Intercomm_merge необходимо задать таким образом, чтобы в созданном интракоммуникаторе вначале располагались исходные, а затем новые процессы). Используя функцию MPI_Cart_create для созданного интракоммуникатора, определить для всех процессов декартову топологию в виде двумерной решетки размера $(K_0/4) \times 4$, являющейся периодической по второму измерению (порядок нумерации процессов оставить прежним). Используя функцию MPI_Cart_coords, определить координаты *X, Y* каждого процесса в данной решетке; в исходных процессах вывести эти координаты, в новых процессах отобразить координаты в разделе отладки с помощью функций Show, снабдив их комментариями «X = » и «Y = ». Выполнить циклический сдвиг значений *A* в каждом столбце созданной решетки с шагом −1 (т. е. в направлении убывания рангов процессов), используя для этого функции MPI_Comm_shift и MPI_Sendrecv_replace. В исходных процессах вывести полученные значения *A*, в новых процессах отобразить их в разделе отладки с помощью функции Show, снабдив комментарием «A = ».

MPI8Inter21. В каждом процессе дано вещественное число: в главном процессе это число обозначается буквой *A*, в подчиненных процессах — буквой *B*. Используя два вызова функции MPI_Comm_spawn с первым параметром «ptprj.exe», создать две группы новых процессов: первая группа (группа-*сервер*) должна содержать один процесс, вторая группа (группа-*клиент*) должна содержать $K - 1$ процесс, где *K* — количество исходных процессов. Переслать в процесс группы-сервера число *A* из главного процесса, а в процессы группы-клиента — числа *B* из подчиненных процессов (в порядке возрастания рангов процессов) и отобразить полученные числа в разделе отладки, используя в каждом новом процессе функцию Show. С помощью функций MPI_Open_port, MPI_Publish_name и MPI_Comm_accept на стороне сервера и функций MPI_Lookup_name и MPI_Comm_connect на стороне клиента установить связь между двумя новыми группами процессов, создав для них новый интеркоммуникатор. Используя функции MPI_Send и MPI_Recv для этого интеркоммуникатора, получить в каждом процессе группы-клиента число *A* из процесса группы-сервера и найти сумму этого числа и числа *B*, полученного ранее из исходных подчиненных процессов. Отобразить найденную сумму $A + B$ в разделе отладки, используя в каждом процессе группы-клиента функцию Show, а также переслать эту сумму в соответствующий исходный подчиненный процесс и вывести полученную сумму в этом процессе (сумма, найденная в процессе ранга *R* группы-клиента, должна быть переслана в исходный процесс ранга $R + 1$).

Примечание. В программе необходимо обеспечить такую последовательность действий, чтобы вызов функции MPI_Lookup_name в процессах клиента гарантированно выполнялся *после* вызова функции MPI_Publish_name в процессе сервера. Можно, например, использовать функцию MPI_Barrier для исходных процессов и процесса сервера, вызвав ее в процессе сервера только после завершения вызова функции MPI_Publish_name, и вызвать функцию MPI_Comm_spawn для создания группы-клиента уже после выхода из функции MPI_Barrier в исходных процессах.

MPI8Inter22. В каждом процессе дано целое число N, которое может принимать три значения: 0, 1 и K ($K > 1$). Известно, что имеется ровно один процесс со значением $N = 1$ и ровно K процессов со значением $N = K$. В тех процессах, в которых число N не равно 0, также дано целое число A. Используя функцию MPI_Comm_split, разбить исходный коммуникатор MPI_COMM_WORLD на два, в каждый из которых входят процессы с одинаковым ненулевым значением N. Используя один вызов функции MPI_Comm_spawn для каждого из созданных коммуникаторов, создать два набора новых процессов; число новых процессов должно совпадать с количеством процессов в соответствующем коммуникаторе (таким образом, в одной новой группе будет содержаться 1 процесс, в другой — K процессов). Переслать в каждый из новых процессов число A из того исходного процесса, ранг которого в созданном коммуникаторе совпадает с рангом нового процесса. Отобразить полученные числа в разделе отладки, используя в каждом новом процессе функцию Show. Считая новую группу, состоящую из одного процесса, *группой-сервером*, а новую группу из K процессов — *группой-клиентом*, установить между ними связь с помощью функций MPI_Open_port, MPI_Publish_name и MPI_Comm_accept на стороне сервера и MPI_Lookup_name и MPI_Comm_connect на стороне клиента. Используя коллективную функцию MPI_Gather для интеркоммуникатора, связывающего группу-клиент и группу-сервер, переслать все числа A из процессов группы-клиента в процесс группы-сервера и отобразить полученные числа в разделе отладки, используя несколько вызовов функции Show в процессе группы-сервера. Кроме того, переслать эти числа с помощью функций MPI_Send и MPI_Recv из процесса группы-сервера в тот исходный процесс, который создал группу-сервер, и вывести полученные числа в этом исходном процессе.

Примечание. В программе необходимо обеспечить такую последовательность действий, чтобы вызов функции MPI_Lookup_name в процессах клиента гарантированно выполнялся *после* вызова функции MPI_Publish_name в процессе сервера. Можно, например, поступить следующим образом. Для пересылки числа A в процесс группы-

сервера применить функцию MPI_Ssend, вызвав после нее функцию MPI_Barrier для исходного коммуникатора MPI_COMM_WORLD и организовав прием этого числа в процессе группы-сервера только после выполнения функции MPI_Publish_name. В остальных процессах исходного коммуникатора MPI_COMM_WORLD вначале вызвать функцию MPI_Barrier, а затем организовать пересылку чисел *A* в созданные процессы группы-клиента. Таким образом, любой из процессов группы-клиента получит число *A* только тогда, когда процесс группы-сервера уже выполнит вызов функции MPI_Publish_name.

9. 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).

9.1. 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 предположе-

нии, 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.

9.2. 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 $N_A = \text{ceil}(M/K)$, $N_B = \text{ceil}(P/K)$, 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 $N_B \times Q$. 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. 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. 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.

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 $K-1$). Оформить эти действия в виде функции Matr1Calc (без параметров). 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. 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 перемножения. 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 $M \times Q$). Оформить эти действия в виде функции Matr1GatherData (без параметров). 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. 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 следу-

ет модифицировать разработанную в MPI9Matr5 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.

9.3. 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_COLS, содержащий вертикальную полосу матрицы B шириной $N_B = Q/K$ столбцов. При определении типа MPI_COLS использовать функции MPI_Type_vector и MPI_Type_commit, оформив это определение в виде функции Matr2CreateTypeCols(p, k, q, t), где целочисленные параметры p, k, q являются входными, а параметр t типа MPI_Datatype является выходным; при этом параметры p и k определяют размер вертикальной полосы, а параметр q — число столбцов матрицы, из которой извлекается эта полоса. Используя тип MPI_COLS, 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 $P \times N_B$. 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_COLS, созданный с помощью функции Matr2CreateTypeCols (см. предыдущее задание и примечание к нему). 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. 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 со-

держать 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 $K-1$). Оформить эти действия в виде функции Matr2Calc (без параметров). 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. 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 перемножения. 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 $M \times Q$). Оформить эти действия в виде функции Matr2GatherData (без параметров). 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_COLS, созданный с помощью функции Matr2CreateTypeCols (см. задание MPI9Matr11 и примечание к нему). 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). По-

следовательно 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 MPI9Matr16 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 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_COLS, определенный с помощью функции Matr2CreateTypeCols (см. MPI9Matr11), после чего также использовать функцию MPI_File_read_all. Для пересылки размеров матриц и имен файлов использовать коллективную функцию MPI_Bcast. 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 $(N_A \cdot K) \times N_B$), получен-

ными 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_COLS, определенный с помощью функции Matr2CreateTypeCols (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).

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. 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.

9.4. 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, $R = 0, \ldots, K - 1$). 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, созданные с помощью функции Matr3CreateTypeBlock (см. задание MPI9Matr21 и примечание к нему). 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. 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). Оформить описанные действия в виде функции Matr3Init (без параметров). 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). Вывести в каждом процессе блоки A_R и B_R, полученные в результате сдвига (ввод и вывод данных выполнять во внешней функции Solve).

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

Примечание. Особенностью алгоритма Кэннона является то, 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 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 $M \times Q$). Оформить эти действия в виде функции Matr3GatherData (без параметров). 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). 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 считывания блоков использовать локальные функции MPI_File_read_at, вызывая отдельную функцию для считывания каждой строки блока (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 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 Matr1ScatterFile, 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 прямоугольной 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_read_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.

9.5. 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.

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

Указание. 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 определять 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. 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. 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 определения 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, $R = 0, \ldots, K - 1$). 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 примечание и указание к нему). Кроме того, создать в каждом про-

цессе 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. 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 матрицы 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, оформив его в виде функции Matr4Calc1 (без параметров). Для пересылки блоков A_R использовать метод MPI_Bcast для коммуникатора MPI_COMM_ROW, создав этот коммуникатор с помощью функции Matr4CreateCommRow (см. задание MPI9Matr33, в котором описывается аналогичная пересылка данных). 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 направ-

лении убывания рангов процессов). Оформить эти действия в виде функции Matr4Calc2 (без параметров). Для циклической пересылки блоков B_R использовать метод MPI_Sendrecv_replace для коммуникатора MPI_COMM_COL, создав этот коммуникатор с помощью функции Matr4CreateCommCol (см. задание MPI9Matr34, в котором описывается аналогичная пересылка данных). 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 $M \times Q$). Оформить эти действия в виде функции Matr4GatherData (без параметров). 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 примечание и указание к нему). 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 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. Для пересылки размеров матриц и имен файлов использовать коллективную функцию MPI_Bcast. 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, заполненные соответствующими блоками A_R, B_R, а также нулевые блоки C_R и T_R. 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_read_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.