Проект комиссии Президента
по модернизации и технологическому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров
в области суперкомпьютерных технологий и
специализированного программного обеспечения»

УТВЕРЖДАЮ
Председатель экспертного совета
системы НОЦ СКТ, член-корр. РАН
В.В. Воеводин
_____
"_____"_____201__г.

Конспект лекций дисциплины

«Параллельное программирование для многопроцессорных систем
с общей и распределенной памятью»

«**010400.62 – Прикладная математика и информатика**»

Разработчики: Лаева В.И. , Трунов А.А.
Рецензент: проф. Старченко А.В.

Москва

КОЛЛЕКТИВНЫЕ ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ

Некоторые функции библиотеки MPI позволяют программисту дать команду группе процессов виртуальной машины. В операциях коллективного взаимодействия *всегда* участвуют *все процессы*, связанные с некоторым коммуникатором. Несоблюдение этого правила приводит к «зависанию» или к аварийному завершению задачи.

В MPI определены следующие коллективные операции:

- синхронизация всех процессов с помощью барьеров;
- коллективные коммуникационные операции (рассылка и сбор данных);
- глобальные вычислительные операции (редукции и сканирования).

Отметим особенности коллективных операций обмена:

- коллективные обмены сообщениями *не взаимодействуют* с двухточечными обменами;
- все коллективные обмены являются *блокирующими* для инициировавшего их процесса, асинхронных коллективных операций нет;
- в операциях обмена не указываются идентификаторы сообщений, теги назначаются системой.

Синхронизация с помощью барьеров

Барьерная синхронизация может быть использована для завершения всеми процессами некоторого этапа решения задачи, результаты которого будут использоваться на следующем этапе. Использование барьера гарантирует, что ни один из процессов не приступит раньше времени к выполнению следующего этапа, пока результат работы предыдущего не будет окончательно сформирован. Эта операция не требует пересылки данных. Функция ***MPI_BARRIER*** блокирует вызвавший ее процесс до тех пор, пока все остальные процессы группы коммуникатора ***comm*** не вызовут ее.

Функция *синхронизации процессов*:

C: int MPI_BARRIER(MPI_Comm comm)

FORTRAN:

INTEGER comm, error

MPI_BARRIER(comm, error)

Завершение работы этой функции возможно только всеми процессами одновременно.

Широковещательная рассылка

Рассылка сообщения выполняется одним выделенным процессом ***root***, все остальные процессы, участвующие в обмене, получают по одной копии сообщения от главного процесса (рис. 4.1).

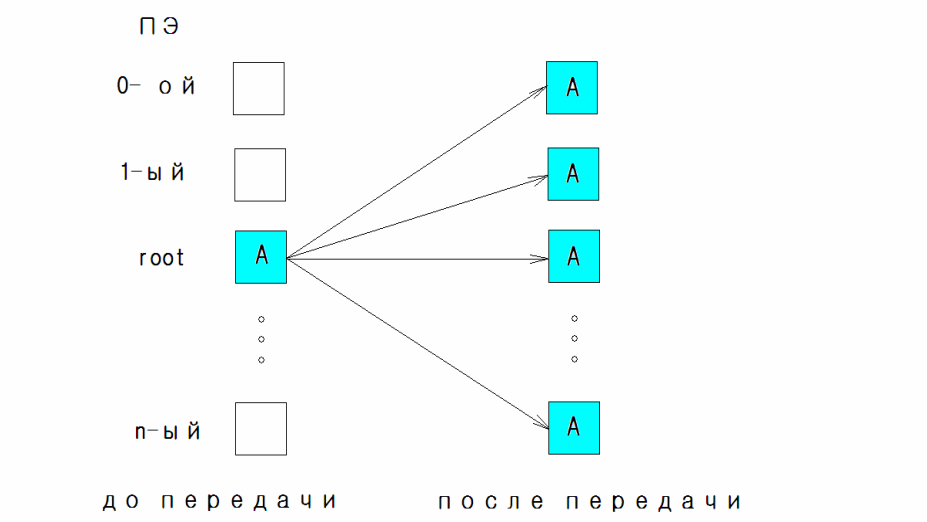

Рис. 4.1. Схема выполнения широковещательной рассылки

Функция *широковещательной рассылки сообщения*:

C: int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

FORTRAN:

INTEGER count, root,comm, error

MPI_Bcast(buf, count, datatype, root, comm, error)

Функция рассылает сообщение ***buf*** процесса с номером ***root*** остальным процессам группы с коммуникатором ***comm***. Каждый процесс получает копию ***count*** элементов типа ***datatype***. Значения параметров ***count, datatype, root, comm*** должны быть одинаковыми для всех процессов, На процессе с номером ***root*** параметр ***buf*** – адрес буфера передачи, на остальных процессах ***buf*** – адрес буфера приема.

Распределение данных

При широковещательной рассылке всем процессам передается один и тот же набор данных. Распределение блоков данных от одного процесса всем процессам группы выполняют функции ***MPI_Scatter*** и ***MPI_Scatterv.***

Функция *распределения блоков данных одинаковой длины*:

C: int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

FORTRAN:

INTEGER sendcount, recvcount, root, comm, error

MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, error)

Входные параметры:

sendbuf – адрес начала буфера посылки,

sendcount – количество элементов, посылаемых каждому процессу,

sendtype **–** тип данных буфера посылки,
recvcount **–** количество элементов в буфере приема,
recvtype **–** тип данных буфера приема,
root **–** номер (ранг) процесса – отправителя,
comm **–** идентификатор коммуникатора.

Выходные параметры:
recvbuf **–** начальный адрес буфера приема,
error **–** код ошибки.

Функция рассылает с процесса ***root*** блоки данных одинаковой длины ***sendcount*** из массива ***sendbuf*** всем процессам: *i* – блок посылается : *i* – ому процессу (рис 4.2). Значения ***root*** и ***comm*** должны быть одинаковы у всех процессов.

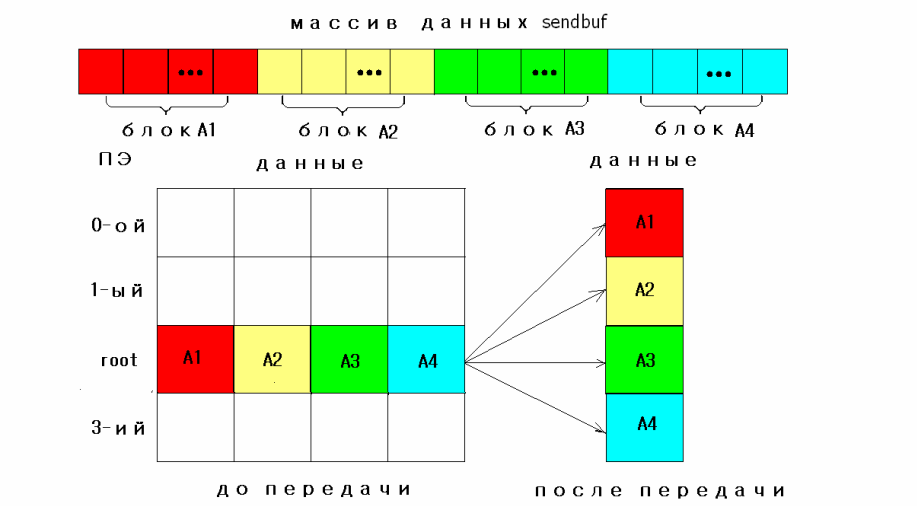

Рис. 4.2 Схема выполнения функции ***MPI_Scatter***

Функция *распределения блоков данных различной длины*:

C: int MPI_Scatterv(void *sendbuf, int sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

FORTRAN:

INTEGER sendcounts(size), recvcount, root, comm, error , displs(size)

MPI_Scatterv(sendbuf, sendcounts, displs,sendtype, recvbuf, recvcount, recvtype, root, comm, error)

Входные параметры:
sendbuf – адрес начала буфера передачи,
sendcounts – целочисленный массив, равный числу процессов (***size***) в коммуникаторе; ***i*** – компонента массива определяет число элементов, которое должно быть передано ***i*** – ому процессу,
displs – целочисленный массив, равный числу процессов в коммуникаторе; ***i*** – компонента массива определяет смещение ***i*** – го блока данных

относительно начала буфера передачи ***sendbuf***; ранг адресата равен значению индекса ***i,***
sendtype – тип данных буфера передачи,
recvcount – количество элементов в буфере приема,
recvtype – тип данных буфера приема,
root – номер (ранг) процесса – отправителя,
comm – идентификатор коммуникатора.

Выходные параметры:
recvbuf – начальный адрес буфера приема,
error – код ошибки.

Функция ***MPI_Scatterv*** является векторным вариантом функции ***MPI_Scatter***, позволяющим посылать каждому процессу различное количество элементов. Благодаря массиву ***displs***, который задает начало расположения элементов блока, пересылаемые данные могут не располагаться непрерывно в памяти главного процесса (***root***).

Пример программы с функцией ***MPI_Scatterv.***

На процессе ***root*** =1 сформирован массив размерности ***n*** =15
sbuf :10,20,30,40,50,60,70,80,90,100,110,120,130,140,150.
На 0 - й процесс в массив ***rbuf*** посылается ***rcount***=2 элемента: 10,20 со смещением 0 относительно начала буфера ***sbuf***;
на 1 - й процесс в массив ***rbuf*** посылается ***rcount***=3 элемента: 50,60,70 со смещением 4 относительно начала буфера ***sbuf***;
на 2 - й процесс в массив ***rbuf*** посылается ***rcount***=4 элемента: 90,100,110,120 со смещением 8 относительно начала буфера ***sbuf***.

```
Program Example_Scatterv
Implicit none Include 'mpif.h'
Integer Rank,Size,Comm,Ierr,n,k,root
Integer rcount,maxsize
Parameter(maxsize=3,n=15,root=1)
Integer scounts(maxsize),displs(maxsize)
Real sbuf(n),rbuf(n)
Call MPI_Init(Ierr)
comm=MPI_COMM_WORLD
Call MPI_Comm_size(Comm,size,Ierr)
Call MPI_Comm_rank(Comm,Rank,Ierr)
if(Rank==root) Then
Do k=1,n
sbuf(k)=10*k
Enddo
Do k=0,size-1
scounts(k+1)=k+2
displs(k+1)=4*k
Enddo
Endif
rcount=rank+2
Call MPI_Scatterv(sbuf(1),scounts,displs,MPI_REAL,
```

```
$       rbuf(1),rcount,MPI_REAL,root,comm,Ierr)
  write(*,*)'process=',rank,' massiv rbuf',(rbuf(k),k=1,rcount)
  Call MPI_Finalize(Ierr)
  end
```

Сбор данных

Сбор блоков данных от всех процессов группы выполняют функции ***MPI_Gather, MPI_Gatherv, MPI_Allgather, MPI_Allgatherv.*** Каждая из перечисленных функций выполняет различные варианты сбора данных.

Функция *сбора блоков данных одинаковой длины на одном процессе*:

C: int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

FORTRAN:

INTEGER sendcount, recvcount, root, comm, error

MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm, error)

Входные параметры:

sendbuf – адрес начала буфера посылки,
sendcount – количество элементов в буфере посылки,
sendtype – тип данных в буфере посылки,
recvcount – количество элементов, получаемых от каждого процесса,
recvtype – тип данных в буфере приема,
root – номер (ранг) процесса – получателя,
comm – идентификатор коммуникатора,

Выходные параметры:

recvbuf – начальный адрес буфера приема,
error – код ошибки.

Функция производит сборку блоков данных одинаковой длины ***sendcount*** от всех процессов группы в один массив ***recvbuf*** процесса с номером ***root.*** Данные, посланные процессом ***i*** из своего буфера ***sendbuf***, помещаются в ***i*** –ю порцию буфера приема (рис 4.3). Значения ***root*** и ***comm*** должны быть одинаковы у всех процессов. Функция ***MPI_Gather*** выполняет операцию обратную к ***MPI_Scatter***.

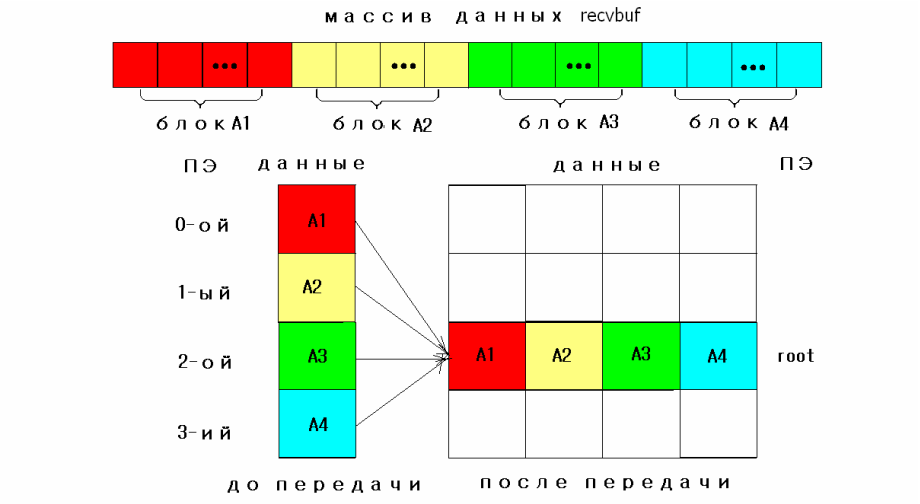

Рис. 4.3 Схема выполнения функции ***MPI_Gather***

Функция *сбора блоков данных одинаковой длины на всех процессах группы:*

C: int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

FORTRAN:

INTEGER sendcount, recvcount, comm, error

MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, error)

Функция ***MPI_Allgather*** выполняется так же, как ***MPI_Gather***, но результат ***recvbuf*** получат все процессы группы (рис 4.4). Смысл и значение параметров этой функции такие же, как у ***MPI_Gather***, только отсутствует номер процесса ***root***.

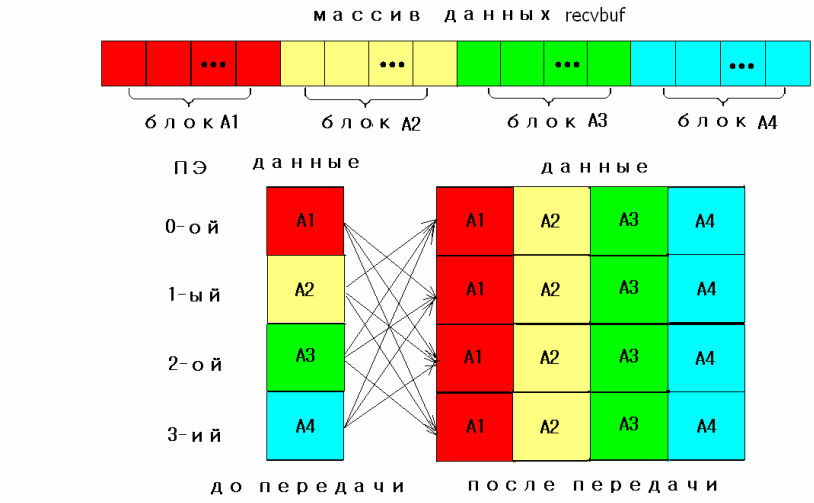

Рис 4.4 Схема выполнения функции ***MPI_Allgather***

Функция *сбора блоков данных различной длины на одном процессе*:

C: int MPI_Gatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

FORTRAN:

INTEGER sendcount, recvcounts(size), root, comm, error , displs(size)

MPI_Gatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm, error)

Входные параметры:

sendbuf – адрес начала буфера передачи,
sendcount – количество элементов в буфере передачи,
sendtype – тип элементов в буфере передачи,
recvcounts – целочисленный массив, равный числу процессов в коммуникаторе; ***i*** – компонента массива определяет число элементов, которое должно быть получено от ***i*** – процесса,
displs – целочисленный массив, равный числу процессов в коммуникаторе; ***i*** – компонента массива определяет смещение ***i*** – го блока данных относительно начала ***recvbuf***,
recvtype – тип элементов в буфере приема,
root – номер (ранг) процесса – получателя,
comm – идентификатор коммуникатора.

Выходные параметры:

recvbuf – начальный адрес буфера приема,
error – код ошибки.

Функция ***MPI_Gatherv*** является векторным вариантом функции ***MPI_Gather***, позволяющим собирать от каждого процесса различное количество элементов. Данные, посланные процессом ***i***, размещаются в адресном пространстве процесса ***root*** , начиная адреса ***recvbuf+displs[i].***

Пример программы с функцией ***MPI_Gatherv.***

На 0 - й процессе сформирован массив ***sbuf*** из ***scount***=3 элементов: 10,20,30 ;
на 1 - ом процессе сформирован массив ***sbuf*** из ***scount***=4 элементов: 11,21,31,41;
на 2 - ом процессе сформирован массив ***sbuf*** из ***scount***=5 элементов: 12,22,32,42,52.
Процесс ***root*** =1 собирает массив ***rbuf*** из ***ncount***=3+4+5 элементов:
10,20,30,11,21,31,41,12,22,32,42,52
со смещением ***displs*** относительно начала буфера ***rbuf***: 0,3,7.

```
Program Example_Gatherv
Implicit none
Include 'mpif.h'
Integer Rank,Size,Comm,Ierr,n,k,root,maxsize,scount
Parameter(maxsize=3,n=15,root=1)
```

```
      Integer rcounts(maxsize),displs(maxsize),ncount
      Real sbuf(n),rbuf(n)
      Call MPI_Init(Ierr)
      comm=MPI_COMM_WORLD
      Call MPI_Comm_size(Comm,size,Ierr)
      Call MPI_Comm_rank(Comm,Rank,Ierr)
      if(Rank==root) Then
      ncount=0
      Do k=1,sizer
      rcounts(k)=k+2
      ncount=ncount+rcounts(k)
      Enddo
      displs(1)=0
      Do k=2,size
      displs(k)=displs(k-1)+rcounts(k-1)
      Enddo
      Endif
      scount=rank+3
      Do k=1,scount
      sbuf(k)=rank+10*k
      Enddo
      Call MPI_Gatherv(sbuf(1),scount,MPI_REAL,
     $     rbuf(1),rcounts,displs,MPI_REAL,root,comm,Ierr)
      If(rank==root) Then
      write(1,*) 'ncount=',ncount, ' massiv rbuf',(rbuf(k),k=1,ncount)
      Endif
      Call MPI_Finalize(Ierr)
      end
```

Функция *сбора блоков данных различной длины на всех процессах группы:*

C: int MPI_Allgatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int *displs, MPI_Datatype recvtype, MPI_Comm comm)

FORTRAN:

INTEGER sendcount, recvcounts(size), comm, error , displs(size)

MPI_Allgatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, comm, error)

Функция ***MPI_Allgatherv*** выполняется так же, как ***MPI_Gatherv***, но результат ***recvbuf*** получат все процессы группы. Смысл и значение параметров этой функции такие же, как у ***MPI_Gatherv***, только отсутствует номер процесса ***root***.

Совмещенные операции сбора и распределения данных

Функции ***MPI_Alltoall*** и ***MPI_Alltoallv*** совмещают в себе операции распределения и сбора данных, когда каждый процесс посылает различные данные разным получателям.

Функция *сбора и распределения блоков данных одинаковой длины:*

C: int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

FORTRAN:

INTEGER sendcount, recvcount, comm, error

MPI_Alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, error)

Функция ***MPI_Alltoall*** совмещает в себе операции ***MPI_Scatter*** и ***MPI_Gather***, когда каждый процесс посылает различные данные разным получателям. Процесс ***i*** посылает ***j*** – ый блок своего буфера ***sendbuf*** процессу ***j***, который помещает его в ***i*** – ый блок своего буфера ***recvbuf*** (рис.4.5). Параметры этой функции такие же, как у функций ***MPI_Scatter*** и ***MPI_Gather***, только отсутствует номер процесса ***root***.

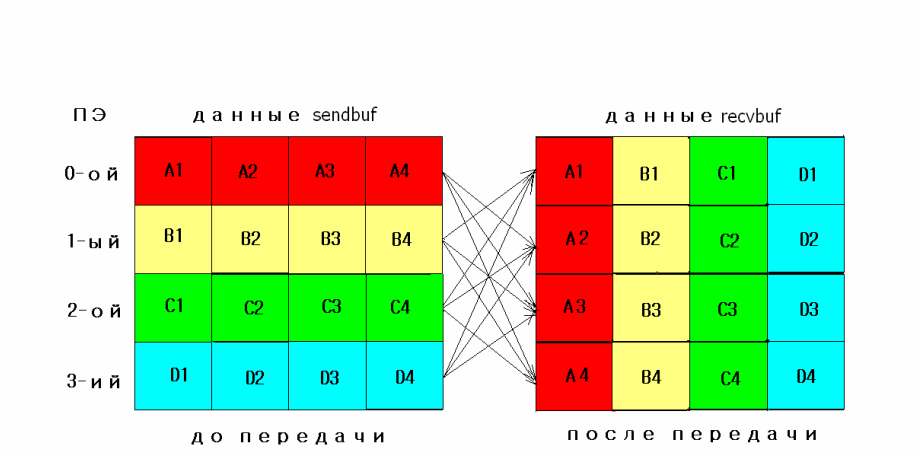

Рис 4.4 Схема выполнения функции ***MPI_Alltoall***

Функция *сбора и распределения блоков данных различной длины:*

C: int MPI_Alltoallv(void *sendbuf, int sendcounts, int *sdispls, MPI_Datatype sendtype, void *recvbuf, int recvcounts, int *rdispls, MPI_Datatype recvtype, MPI_Comm comm)

FORTRAN:

INTEGER sendcounts(size), sdispls(size), recvcounts(size), rdispls(size), comm, error

MPI_Alltoallv(sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcounts, rdispls, recvtype, comm, error)

Входные параметры:

sendbuf – адрес начала буфера передачи,

sendcounts – целочисленный массив, равный числу процессов в коммуникаторе; ***i*** – компонента массива определяет число элементов, которое должно быть передано ***i*** – ому процессу,

sdispls **–** целочисленный массив, равный числу процессов в коммуникаторе; ***i*** **–** компонента массива определяет смещение ***i*** **–** го блока данных относительно начала буфера передачи ***sendbuf;*** ранг адресата равен значению индекса ***i,***
sendtype **–** тип элементов в буфере передачи,
recvcounts – целочисленный массив, равный числу процессов в коммуникаторе; ***j*** – компонента массива определяет число элементов, которое должно быть получено от ***j*** – ого процесса,
rdispls – целочисленный массив, равный числу процессов в коммуникаторе; ***j*** – компонента массива определяет смещение относительно начала буфера приема ***rcvbuf***,
recvtype – тип элементов в буфере приема,
comm – идентификатор коммуникатора.

Выходные параметры:
recvbuf – начальный адрес буфера приема,
error – код ошибки.

Функция ***MPI_Alltoallv*** реализует векторный вариант ***MPI_Alltoall***, который допускает прием и передачу со смещением блоков различной длины от всех процессов всем процессам.

Глобальные вычислительные операции

В MPI предусмотрена возможность выполнения глобальных вычислительных операции над данными, распределенными по процессам (операции редукции). Операция может быть предопределенной операцией MPI или операцией, определенной пользователем. Глобальные операции редукции представлены функциями:

MPI_Reduce – сохраняет результат в адресном пространстве одного процесса,

MPI_Allreduce – сохраняет результат на всех процессах группы,

MPI_Scan – выполняет префиксную (частичную) операцию редукции,

MPI_Reduce_scatter – выполняет совмещенную операцию редукции и распределения результата.

Предопределенные операции MPI задаются именованными константами:

MPI_MAX, MPI_MIN – максимальное, минимальное значение;

MPI_SUM, MPI_PROD – сумма**,** произведение;

MPI_LAND, MPI_LOR – логическое «и», логическое «или»;

MPI_BAND, MPI_BOR – побитовое «и», побитовое «или»

и др.

Функция *редукции с сохранением результата на одном процессе*:

C: int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

FORTRAN:

INTEGER count, op, root,comm, error

MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm, error)

Входные параметры:

sendbuf – адрес начала входного буфера,
count – количество элементов во входном буфере,
datatype – тип данных,
op – идентификатор операции приведения,
root – номер (ранг) процесса – получателя,
comm – идентификатор коммуникатора.

Выходные параметры:

recvbuf – начальный адрес буфера приема результата операции,
error – код ошибки.

Функция выполняет операцию ***op*** к ***count*** элементам массива ***sendbuf***, имеющих тип ***datatype.*** Тип элементов ***datatype*** должен быть совместим с операцией ***op***. Результаты выполнения всех ***count*** операций записывается в массиве ***recvbuf*** на процессе с номером ***root*** (рис. 4.6). Значения ***count***, ***datatype, op, comm.***и ***root*** у всех процессов должны быть одинаковы.

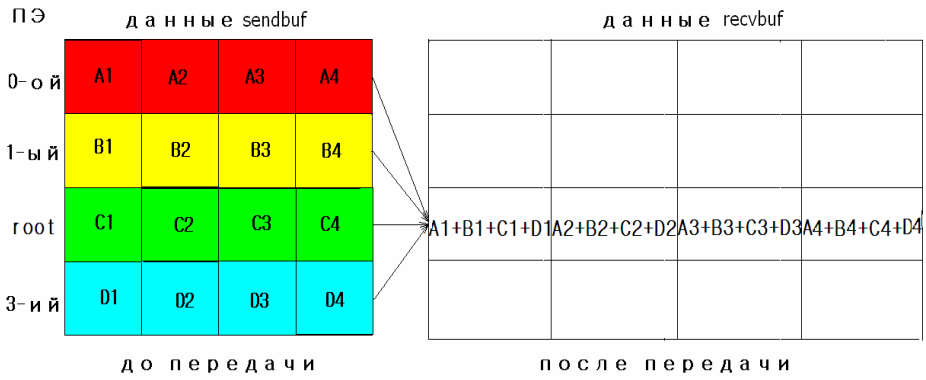

Рис 4.6 Схема выполнения функции ***MPI_Reduce***

Функция *редукции с сохранением результата на всех процессах группы*:

C: int MPI_ Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

FORTRAN:

INTEGER count, op, comm, error

MPI_Allreduce(sendbuf, recvbuf, count, datatype, op, comm, error)

Функция ***MPI_Allreduce*** выполняется так же, как ***MPI_Reduce***, но результат ***recvbuf*** получат все процессы группы. Смысл и значение

параметров этой функции такие же, как у ***MPI_Reduce***, только отсутствует номер процесса ***root***.

Функция *префиксной редукции* (*сканирования*):

C: ***int MPI_Scan(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)***

FORTRAN:

INTEGER count, op, comm, error

MPI_Scan(sendbuf, recvbuf, count, datatype, op, comm, error)

Функция ***MPI_Scan*** выполняет префиксную (частичную) редукцию. Параметры этой функции такие же, как в ***MPI_Allreduce***, но получаемые каждым процессом результаты отличаются друг от друга. Функция пересылает в буфер приема ***i*** – го процесса редукцию из значений входных буферов процессов с номерами ***0,1,…,i*** включительно (рис 4.7).

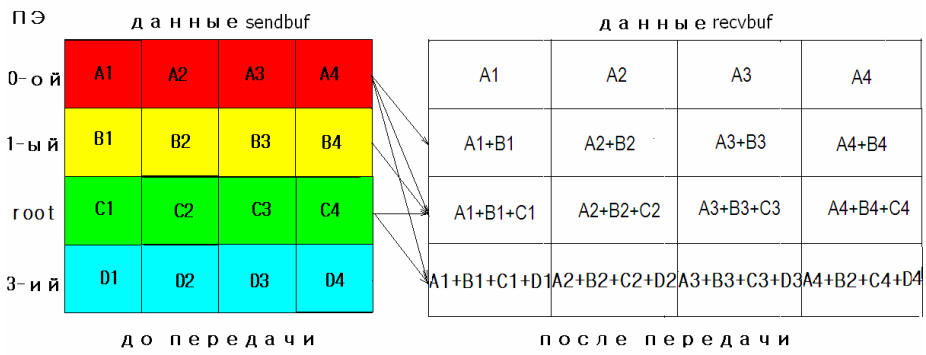

Рис 4.7 Схема выполнения функции ***MPI_Scan***

Совместная функция *редукции и распределения результата по процессам*:

C: ***int MPI_Reduce_scatter(void *sendbuf, void *recvbuf, int counts, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)***

FORTRAN:

INTEGER counts(size), op, comm, error

MPI_Reduce_scatter(sendbuf, recvbuf, counts, datatype, op, comm, error)

Входные параметры:

sendbuf – адрес начала входного буфера,
counts – массив размеров блоков данных, посылаемых процессам,
datatype – тип данных,
op – идентификатор операции приведения,
comm – идентификатор коммуникатора.

Выходные параметры:

recvbuf – начальный адрес буфера приема,
error – код ошибки.

Выполнение этой функции отличается от ***MPI_Allreduce*** тем, что результат этой операции разрезается на непересекающиеся части по числу процессов в группе, ***i*** – ая часть посылается , ***i*** – ому процессу в его буфер приема ***recvbuf*** (рис.4.8). Длины этих частей задаются в массиве ***counts***. Параметры ***counts, datatype, op*** и ***comm.*** должны быть одинаковыми для всех процессов.

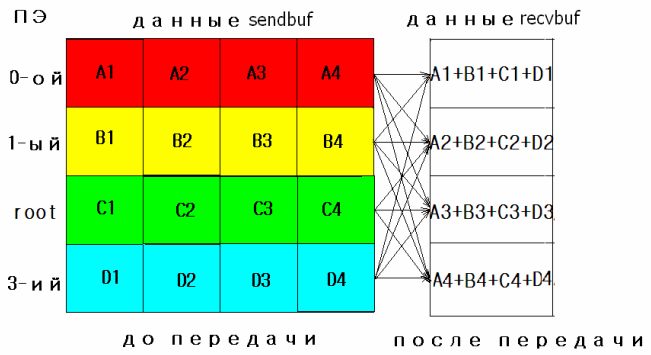

Рис 4.7 Схема выполнения функции ***MPI_Reduce_scatter***

Пример программы с функцией ***MPI_Reduce_scatter***

На всех процессах создаются массивы ***sbuf*** из ***count*** = 6 элементов:
на 0 – ом процессе массив: 10,20,…,60;
на 1 – ом процессе массив: 11,21,…,61;
на 2 – ом процессе массив: 12,22,…,62.

Массив $counts(0) = 2, counts(1) = 1, counts(2) = 3, \sum_{k=0}^{size-1} counts(k) = count, size = 3.$

После выполнения функции ***MPI_Reduce_scatter*** с операцией суммирования ***MPI_SUM*** будут вычислены массивы ***rbuf***

на 0 – ом процессе: $rbuf(1) = \sum_{rank=0}^{size-1} sbuf(1), rbuf(2) = \sum_{rank=0}^{size-1} sbuf(2),$

на 1 – ом процессе: $rbuf(1) = \sum_{rank=0}^{size-1} sbuf(3),$

на 2 – ом процессе:

$$rbuf(1) = \sum_{rank=0}^{size-1} sbuf(4), rbuf(2) = \sum_{rank=0}^{size-1} sbuf(5), rbuf(3) = \sum_{rank=0}^{size-1} sbuf(6).$$

```
Program Example
Implicit none
Include 'mpif.h'
```

```
 Integer Rank,Size,Comm,Ierr,k
 Integer count,maxsize
 Parameter(maxsize=3,count=6)
 Integer counts(0:maxsize-1)
 Real sbuf(count),rbuf(count)
 Call MPI_Init(Ierr)
 comm=MPI_COMM_WORLD
 Call MPI_Comm_size(Comm,size,Ierr)
 Call MPI_Comm_rank(Comm,Rank,Ierr)
 counts(0)=2
 counts(1)=1
 counts(2)=3
 Do k=1,count
 sbuf(k)=rank+10*k
 Enddo
 Call    MPI_Reduce_scatter(sbuf(1),rbuf(1),
$    counts,MPI_REAL,MPI_SUM,comm,Ierr)
 write(*,*)'process=',rank,' rbuf=',(rbuf(k),k=1,counts(rank))
 Call MPI_Finalize(Ierr)
 end
```