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

ДВУХТОЧЕЧНЫЙ НЕБЛОКИРУЮЩИЙ ОБМЕН СООБЩЕНИЯМИ

Библиотека MPI содержит набор функций для выполнения асинхронного (без блокировки) обмена сообщениями. В отличие от блокирующих, которые приостанавливают вызвавший их процесс до тех пор, пока операция передачи не будет завешена, неблокирующие подразумевают совмещение операций обмена с другими операциями. Использование неблокирующих функций более безопасно с точки зрения возникновения тупиковых ситуаций, а также может увеличить скорость работы программы за счет совмещения выполнения вычислительных и коммуникационных операций.

Вызов функции неблокирующей передачи инициируют, но не завешают ее. Завершиться выполнение неблокирующей передачи может еще до того, как сообщение будет скопировано в буфер передачи. Для завершения обмена требуется вызов дополнительной функции, которая проверяет, скопированы ли данные буфер передачи. Таким образом, асинхронный обмен выполнятся в два этапа:

- *инициирование* операции,
- *проверка завершения* операции.

Для связи между функциями обмена и функциями опроса их завершения неблокирующие операции используют специальный скрытый объект – *идентификатор асинхронной операции обмена* (request). Для прикладных программ доступ к идентификатору обмена возможен только через вызовы MPI функций. Если операция обмена завершена, функция проверки снимает «запрос обмена», устанавливая его значение ***MPI_REQUEST_NULL***.

Инициализация неблокирующего обмена

Как было отмечено ранее, асинхронная передача сообщения может выполняться в четырех режимах:

неблокирующий стандартный (***MPI_Isend***),
неблокирующий синхронный (***MPI_Issend***),
неблокирующий буферизованный (***MPI_Ibsend***),
неблокирующий «по готовности» (***MPI_Irsend***)

(буква ***I*** в имени от *immediately – немедленно*).

Функция *инициализации стандартной неблокирующей* передачи сообщения:

C: int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm,MPI_Request *request)

FORTRAN:

INTEGER count, dest, tag, comm, error,request

MPI_Isend(buf, count, datatype, dest, tag, comm, request, error)

Функция инициирует посылку сообщения ***buf*** с идентификатором (тегом) ***tag***, состоящего из ***count*** элементов типа ***datatype***, процессу с номером ***dest*** в области связи с коммуникатором ***comm***. Тип передаваемых элементов ***datatype*** должен указываться с помощью с помощью соответствующих типов MPI. Перечисленные параметры являются входными. Выходной параметр ***request –*** идентификатор запроса на выполнение операции имеет тип *MPI_Request* в ***C*** и тип INTEGER в языке FORTRAN.

Возврат из функции ***MPI_Isend*** происходит сразу без ожидания обработки всего сообщения, находящегося в буфере ***buf.*** Это означает, что нельзя что – то записывать в данный буфер без получения дополнительной информации, подтверждающей завершения пересылки. Определить, когда можно использовать повторно буфер передачи без опасения испортить передаваемое сообщение, можно с помощью параметра ***request*** и функций проверки выполнения обмена.

Функции ***MPI_Isend MPI_Issend***,***MPI_Ibsend***,***MPI_Irsend*** имеют одинаковые параметры.

Функция *инициализации неблокирующего* приема сообщения:

C: int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,MPI_Request *request)

FORTRAN:

INTEGER count, source, tag,comm, error, request

MPI_Irecv(buf, count, datatype, source, tag, comm, request, error)

Функция инициирует прием сообщения ***buf*** (выходной параметр) с идентификатором ***tag*** от процесса ***source***. Число элементов в принимаемом сообщении не должно превосходить значения ***count***. Тип получаемых данных ***datatype*** должен указываться с помощью соответствующих типов MPI. Выходной параметр ***request –*** идентификатор запроса на выполнение операции.

Возврат из функции ***MPI_Irecv*** происходит сразу без ожидания получения и записи всего сообщения в буфере ***buf.*** Окончание процесса приема можно определить с помощью параметра ***request*** и функций проверки выполнения обмена.

Сообщение, отправленное функцией ***MPI_Send***, ***MPI_Isend*** и любой из трех их модификаций, может быть принято любой функцией ***MPI_Recv***, ***MPI_Irecv***.

Функции проверки выполнения обмена

Проверка фактического выполнения передачи/приема сообщения в неблокирующем режиме осуществляется с помощью вызова функций проверки. Блокирующие функции проверки приостанавливают работу

процесса до завершения операции обмена (***MPI_Wait, MPI_Waitall, MPI_Waitany, MPI_Waitsome***). Неблокирующие функции проверки возвращают логическое значение «истина», если операция выполнена (***MPI_Test, MPI_Testall, MPI_Testany***), а функция ***MPI_Testsome*** возвращает количество завершившихся обменов.

Функция *проверки с блокировкой*:

C: int MPI_Wait (MPI_Request *request, MPI_Status *status)

FORTRAN:

INTEGER status(MPI_STATUS_SIZE), request , error

MPI_Wait (request, status, error)

Функция блокирует работу процесса до тех пор, пока не будет завершен обмен с запросом ***request***. Параметр ***request*** – входной и выходной, после завершения этой функции, он становится равным ***MPI_REQUEST_NULL***. Выходной параметр ***status*** возвращает информацию о выполненной операции.

Функция *проверки без блокировки*:

C: int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

FORTRAN:

LOGICAL flag

INTEGER request, error, status(MPI_STATUS_SIZE)

MPI_Test(request, flag, status, error)

Входной параметр :

request – идентификатор запроса на выполнения операции асинхронного обмена.

Выходные параметры:

flag – признак завершенности операции обмена (если ***flag=true***, то операция обмена завершилась успешно),

status– статус (информация о полученном сообщении),

error – код ошибки.

Когда сразу несколько процессов обмениваются сообщениями, можно использовать проверки, которые применяются одновременно к нескольким обменам:

MPI_Waital, ***MPI_Testall*** – завершились все операции обмена,

MPI_Waitany, ***MPI_Testany*** – завершилась по крайней мере одна операция обмена,

MPI_Waitsome, ***MPI_Testsome*** – завершилась одна из списка проверяемых операций обмена.

Функция *блокирующей проверки завершения всех обменов*:

C: int MPI_Waitall (int count, MPI_Request *requests[], MPI_Status *statuses[])

FORTRAN:

INTEGER count, requests(count) ,error, statuses(count)

MPI_Waitall(count, requests, statuses, error)

Входные параметры:

count – количество запросов на обмен (размерность массивов ***requests*** и ***statuses***);

requests – массив идентификаторов операции асинхронного обмена.

Выходные параметры:

statuses – массив статусов сообщений,

error – код ошибки.

В результате выполнения функции ***MPI_Waitall*** аннулируются запросы, сформированные неблокирующими операциями обмена, и соответствующим элементам массива ***requests*** присваивается значение ***MPI_REQUEST_NULL.*** Если операция выполнена успешно, полю (элементу массива) ошибки статуса присваивается ***MPI_SUCCESS.*** В случае неуспешного выполнения одной или более операций возвращается код ошибки ***MPI_ERR_IN_STATUS*** и полю присваивается значение кода ошибки соответствующей операции. Если операция не выполнена, но и не было ошибки, полю ошибки присваивается ***MPI_ERR_PENDING*** (это соответствует наличию запросов на выполнение операции обмена, ожидающих обработки).

Функция *неблокирующей проверки завершения всех обменов*:

C: int MPI_Testall (int count, MPI_Request *requests[], int *flag, MPI_Status *statuses[])

FORTRAN:

INTEGER count, requests(count) ,error, statuses(count)

LOGICAL flag

MPI_Testall(count, requests, flag, statuses, error)

Смысл и значение параметров ***count, requests*** и ***statuses*** тот же, что и для функции ***MPI_Waitall.*** Выходной параметр ***flag=true***, если все обмены, связанные с активными запросами в массиве ***requests*** выполнены. Если не все обмены завершены, то ***flag=false***, а массив ***statuses*** не определен.

Функция *блокирующей проверки завершения любого числа обменов*:

C: int MPI_Waitany(int count, MPI_Request *requests[], int *index, MPI_Status *statuses[])

FORTRAN:

INTEGER count, requests(count) ,error, statuses(count), index

MPI_Waitany(count, requests, index, statuses, error)

Смысл и значение параметров ***count, requests*** и ***statuses*** тот же, что и для функции ***MPI_Waitall.*** Выходной параметр ***index*** – номер элемента в массиве ***requests***, содержащего идентификатор завершенной операции.

Если в списке вообще нет активных запросов или он пуст, вызовы завершаются сразу со значением индекса ***MPI_UNDEFINED.***

Функция блокирует работу процесса до тех пор, пока, по крайней мере, один обмен из массива запросов ***requests*** , не будет завершен.

Функция *неблокирующей проверки завершения любого числа обменов*:

C: int MPI_Testany(int count, MPI_Request *requests[], int *index, int *flag, MPI_Status *statuses[])

FORTRAN:

INTEGER count, requests(count) ,error, statuses(count) , index

LOGICAL flag

MPI_Testany(count, requests, index, flag, statuses, error)

Смысл и значение параметров ***count, index, requests*** и ***statuses*** тот же, что и для функции ***MPI_Waitany***, но выходной параметр ***flag=true***, если одна из операций завершена.

Функция *блокирующей проверки завершения обменов из указанного списка*:

C: int MPI_Waitsome(int count, MPI_Request *requests[], int *outcount, int *indexes[], MPI_Status *statuses[])

FORTRAN:

INTEGER count, requests(count), statuses(count), indexes(count), error, outcount

MPI_Waitsome(count, requests, outcount, indexes, statuses, error)

Смысл и значение параметров ***count, requests*** и ***statuses*** тот же, что и для функции ***MPI_Waitany.*** Выходные параметры: ***outcount*** – количество выполненных запросов из массива ***requests***, в первых ***outcount*** элементах массива ***indexes*** – возвращаются индексы этих операций, в первых ***outcount*** элементах массива ***statuses*** – возвращается статус завершенных операций. Если в списке нет активных запросов, выполнение функции завершается сразу, а параметру ***outcount*** присваивается значение ***MPI_UNDEFINED.***

Функция *неблокирующей проверки завершения обменов из указанного списка*:

C: int MPI_Testsome(int count, MPI_Request *requests[], int *outcount, int *indexes[], MPI_Status *statuses[])

FORTRAN:

INTEGER count, requests(count), statuses(count), indexes(count), outcount, error

MPI_Testsome(count, requests, outcount, indexes, statuses, error)

Функция работает так же, как ***MPI_Waitsome***, за исключением того, что возврат из функции происходит немедленно. Если ни одна из указанных операций не завершилась, то ***outcount=0.***

В функциях ***MPI_Waitany*** и ***MPI_Testany*** обмен из числа завершенных выбирается произвольно и для него возвращается статус. В ***MPI_Waitsome*** и ***MPI_Testsome*** статус возвращается для всех завершенных обменов, поэтому эти функции можно использовать для того, чтобы определить количество завершенных обменов.

Пример программы обмена сообщениями по «кольцу» с использованием неблокирующих функций обмена.

```
Program Example
Implicit none
Include 'mpif.h'
Integer Rank,Size,Comm,Ierr,n,tag,k
Parameter (n=2)
Integer next,secc,requests(n),statuses(n)
Real sbuf,rbuf
tag=0
Call MPI_Init(Ierr)
comm=MPI_COMM_WORLD
Call MPI_Comm_size(Comm,size,Ierr)
Call MPI_Comm_rank(Comm,Rank,Ierr)
next=rank+1
secc=rank-1
if(rank==0) secc=size-1
if(rank==size-1) next=0
Call Random_seed()
do k=1,rank+1
Call Random_number(sbuf)
enddo
Call MPI_Irecv(rbuf,1,MPI_REAL,secc,tag,comm,requests(1),Ierr)
Call MPI_Isend(sbuf,1,MPI_REAL,next,tag,comm,requests(2),Ierr)
write(*,*) 'process=',rank,' send sbuf=',sbuf
write(*,*) 'process=',rank,' before Wait recv rbuf=',rbuf
Call MPI_Waitall(n,requests,statuses,Ierr)
write(*,*) 'process=',rank,'  after Wait recv rbuf=',rbuf
Call MPI_Finalize(Ierr)
end
```

Объединение запросов на взаимодействие

MPI позволяет для неблокирующих операций формировать пакеты запросов на коммуникационные операции. Несколько запросов на обмен можно объединить вместе для того, чтобы далее их можно было запустить одной командой. Это позволяет минимизировать накладные расходы на организацию связи между процессором и сетевым контроллером.

Для формирования отложенного запроса на передачу сообщения можно использовать функции: ***MPI_Send_init, MPI_Ssend_init,***

MPI_Bsend_ init, MPI_Rsend_init. Сама операция пересылки при вызове этих функций не начинается.

Функция *формирование запроса на выполнение стандартной передачи сообщения*:

C: int MPI_Send_init(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request request)

FORTRAN:

INTEGER count, datatype, dest, tag, comm, error,request

MPI_Send_init(buf, count, datatype, dest, tag, comm, request,error)

Все параметры этой функции такие же, как у функции ***MPI_Isend.***

Функция *формирование запроса на выполнение приема сообщения*:

C: int MPI_Recv_init(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request request)

FORTRAN:

INTEGER count, datatype, source, tag, comm, error,request

MPI_Recv_init(buf, count, datatype, source, tag, comm, request,error)

Все параметры этой функции такие же, как у функции ***MPI_Irecv.*** Функция формирует отложенный запрос на прием сообщения, но операцию приема не выполняет.

Отложенный обмен сообщениями инициализируется вызовом специальных функций.

Функция *инициализации отложенного обмена*:

C: int MPI_Start(MPI_Request *request)

FORTRAN:

INTEGER error,request

MPI_Start (request,error)

Входным параметром ***request*** является идентификатор запроса на выполнение операции обмена. Вызов ***MPI_Start*** с запросом на обмен, созданным ***MPI_Send_init,*** инициирует обмен с теми же свойствами, что и вызов ***MPI_Isend, MPI_Bsend_init*** –вызову ***MPI_Ibsend*** и т.д.

Функция *инициализации всех отложенных обменов*:

C: int MPI_Startall(int count,MPI_Request *requests)

FORTRAN:

INTEGER count,error,requests(count)

MPI_Startall (count,requests,error)

Функция инициализирует ***count*** отложенных запросов на обмен, соответствующих значениям элементов массива ***requests***.

Сообщение, отправленное с помощью объединения запросов или обычным способом, может быть принято обычным способом или с помощью объединения запросов.

Значение параметра ***request*** после завершения коммуникационной операций, запущенной при помощи отложенного запроса, *сохраняется* и *может быть использовано* в дальнейшем.

Функции отмены «ждущих» асинхронных обменов

Аннулировать неблокирующие ожидающие обработки («ждущие») коммуникационные операции можно с помощью функций ***MPI_Request_free*** и ***MPI_Cancell.***

Функция *снятия запроса без ожидания завершения неблокирующей операции:*

C: int MPI_Request_free(MPI_Request *request)
FORTRAN:
INTEGER error,request
MPI_Request_free(request,error)

При вызове эта функция помечает запрос на обмен ***request*** для удаления и присваивает ему значение ***MPI_REQUEST_NULL.*** Операция обмена, связанная с этим запросом завершается, а сам запрос удаляется только после завершения обмена.

Функция *аннулирования неблокирующего обмена*:

C: int MPI_Cancel(MPI_Request *request)
FORTRAN:
INTEGER error,request
MPI_Cancel(request,error)

Функция отменяет неблокирующий обмен с запросом ***request.*** Вызов этой функции завершается сразу, возможно еще до реального аннулирования обмена, поэтому этот обмен следует завершить с помощью функций проверки выполнения асинхронного обмена. После вызова ***MPI_Cancel*** и следующего за ним ***MPI_Wait*** или ***MPI_Test,*** запрос на выполнение операции обмена становится неактивным и может быть активирован для нового обмена. ***MPI_Cancel*** может быть использована для отмены отложенных и обычных обменов.

Функция *проверки аннулирования неблокирующего обмена*:

C: int MPI_Test_cancelled(MPI_Status *status, int *flag)
FORTRAN:
INTEGER error, status(MPI_STATUS_SIZE)
LOGICAL flag
MPI_Test_cancelled(status, flag, error)

Функция возвращает значение ***flag=true,*** если обмен с указанным статусом ***status*** успешно аннулирован.

Пример программы с функциями ***MPI_Send_init*** и ***MPI_Recv_init***.

```
Program Example
Implicit none
Include 'mpif.h'
Integer Rank,Size,Comm,Ierr,n,tag,m,k,i
Parameter (n=2,m=3)
Integer next,secc,rqs(2),sts(2)
Real sbuf,rbuf
tag=0
Call MPI_Init(Ierr)
comm=MPI_COMM_WORLD
Call MPI_Comm_size(Comm,size,Ierr)
Call MPI_Comm_rank(Comm,Rank,Ierr)
next=rank+1
secc=rank-1
if(rank==0) secc=size-1
if(rank==size-1) next=0
Call MPI_Recv_init(rbuf,1,MPI_REAL,secc,tag,comm,rqs(1),Ierr)
Call MPI_Send_init(sbuf,1,MPI_REAL,next,tag,comm,rqs(2),Ierr)
Call Random_seed()
do k=1,m
do i=1,rank+k
Call Random_number(sbuf)
enddo
Call MPI_Startall(n,rqs,Ierr)
write(*,*) 'process=',rank,' k= ',k,' send sbuf=',sbuf
Call MPI_Waitall(n,rqs,sts,Ierr)
write(*,*) 'process=',rank,' k=',k,
&            '  after Wait recv rbuf=',rbuf
enddo
call MPI_Request_free(rqs(1),Ierr)
call MPI_Request_free(rqs(2),Ierr)
Call MPI_Finalize(Ierr)
end
```