Санкт-Петербургский государственный университет
кафедра вычислительной физики

С.А.Немнюгин

Лабораторная работа 0.2
Знакомство с процедурами буферизованного и неблокирующего двухточечного обмена MPI

Методические материалы к курсу «Средства программирования для многопроцессорных вычислительных систем»

Intel Multicore Curriculum Initiative

Санкт-Петербург
2007

Двухточечный буферизованный обмен

При передаче сообщения в буферизованном режиме источник копирует сообщение в буфер, а затем передает его в неблокирующем режиме. Выделение буфера и его размер контролируются программистом, который должен заранее создать буфер достаточного размера. Буферизованная передача завершается сразу, поскольку сообщение немедленно копируется в буфер для последующей передачи.

После завершения работы с буфером его необходимо отключить. После отключения буфера можно вновь использовать занимаемую им память, однако следует помнить, что в языке С данный вызов не освобождает автоматически память, отведенную для буфера.

Буферизованный обмен рекомендуется использовать в тех ситуациях, когда программисту требуется больший контроль над распределением памяти.

Двухточечные неблокирующие обмены

Вызов подпрограммы неблокирующей передачи инициирует, но не завершает ее. Передача данных из буфера или их считывание происходит одновременно с выполнением других операций. Завершается обмен вызовом дополнительной процедуры, которая проверяет, скопированы ли данные в буфер передачи. До завершения обмена запись в буфер или считывание из него производить нельзя, так как сообщение может быть еще не отправлено или не получено. Неблокирующая передача может быть принята подпрограммой блокирующего приема и наоборот.

Неблокирующий обмен выполняется в два этапа:

1. Инициализация обмена.
2. Проверка завершения обмена.

Для маркировки неблокирующих операций обмена используются *идентификаторы операций обмена*.

Проверка фактического выполнения передачи или приема в неблокирующем режиме осуществляется с помощью вызова подпрограмм ожидания, блокирующих работу процесса до завершения операции или неблокирующих подпрограмм проверки, возвращающих логическое значение "истина", если операция выполнена.

Подпрограмма `MPI_Wait` блокирует работу процесса до завершения приема или передачи сообщения. Функции `MPI_Wait` и `MPI_Test` можно использовать для завершения операций приема и передачи.

Подпрограммы-пробники

Получить информацию о сообщении до его помещения в буфер приема можно с помощью подпрограмм-пробников `MPI_Probe` и `MPI_IProbe`. На основании полученной информации принимается решение о дальнейших действиях. С помощью вызова подпрограммы `MPI_Probe` фиксируется поступление (но не прием!) сообщения. Затем определяется источник сообщения, его длина, выделяется буфер подходящего размера и выполняется прием сообщения.

Лабораторная работа

В заданиях лабораторной работы 0.2 предлагается дописать предлагаемые фрагменты программ на языках Fortran и C, написанные с использованием процедур MPICH 1.2.7. Пропущенные фрагменты обозначены многоточием.
Необходимый для выполнения данной лабораторной работы справочный материал можно найти на стр. 30 – 34 *методического пособия* «Средства программирования для многопроцессорных вычислительных систем».

Задание 1

В исходном тексте программы на языке C пропущены вызовы процедур буферизованного обмена. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int *buffer;
 int myrank;
 MPI_Status status;
 int buffsize = 1;
 int TAG = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
{
 buffer = (int *) malloc(buffsize + MPI_BSEND_OVERHEAD);
 …
 buffer = (int *) 10;
 …
}
else
{
 MPI_Recv(&buffer,  buffsize,  MPI_INT,  0,  TAG,  MPI_COMM_WORLD,
&status);
 printf("received: %i\n", buffer);
}
 MPI_Finalize();
 return 0;
}
```

Задание 2

В исходном тексте программы на языке Fortran пропущены вызовы процедур неблокирующих операций обмена. Добавить эти вызовы, откомпилировать и запустить программу.

```
program main_mpi
include 'mpif.h'
integer rank, tag, cnt, ierr, status(mpi_status_size)
integer request
real sndbuf(5) /1., 2., 3., 4., 5./
real rcvbuf(5)
cnt = 5
tag = 0
call mpi_init(ierr)
call mpi_comm_rank(mpi_comm_world, rank, ierr)
if(rank.eq.0) then
…
print *, "process ", rank, " send before wait", sndbuf
…
print *, "process ", rank, " send after wait", sndbuf
else
…
print *, "process ", rank, " received before wait", rcvbuf
…
print *, "process ", rank, " received after wait", rcvbuf
end if
call mpi_finalize(ierr)
stop
end
```

Задание 3

В исходном тексте программы на языке С пропущены вызовы подпрограмм-пробников. Добавить эти вызовы, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int myid, numprocs, **buf, source, i;
 int message[3] = {0, 1, 2};
 int myrank, data = 2002, count, TAG = 0;
 MPI_Status status;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if(myrank == 0)
{
 MPI_Send(&data, 1, MPI_INT, 2, TAG, MPI_COMM_WORLD);
}
else if (myrank == 1) {
 MPI_Send(&message, 3, MPI_INT, 2, TAG, MPI_COMM_WORLD);
}
else
{
…
source = status.MPI_SOURCE;
MPI_Get_count(…);
for (i = 0; i < count; i++){
buf[i] = (int *)malloc(count*sizeof(int));
}
MPI_Recv(&buf[0], count, MPI_INT, source, TAG, MPI_COMM_WORLD,
&status);
for (i = 0; i < count; i++){
printf("received: %d\n", buf[i]);
}
 }
 MPI_Finalize();
 return 0;
}
```

Задание 4

В исходном тексте программы на языке Fortran пропущены вызовы процедур блокирующего зондирования и некоторые другие важные фрагменты. Добавить эти вызовы, откомпилировать и запустить программу.

```
program main_mpi
include 'mpif.h'
integer rank, i, k, ierr, tag, dest, …
real x
tag = 0
dest = 2
call mpi_init(ierr)
call mpi_comm_rank(mpi_comm_world, rank, ierr)
if (rank.eq.0) then
 i = 2002
 call  mpi_send(i,  1,  mpi_integer,  dest,  tag,  mpi_comm_world,
ierr)
else if(rank.eq.1) then
x = 3.14159
call mpi_send(x, 1, mpi_real, dest, tag, mpi_comm_world, ierr)
else
do k = 1, 2
…
 if (status(mpi_source).eq.0) then
 call  mpi_recv(i,  1,  mpi_integer,  0,  tag,  mpi_comm_world,
status, ierr)
 print *, "received ", i, " from 0"
else
 call  mpi_recv(x,  1,  mpi_real,  1,  tag,  mpi_comm_world,  status,
ierr)
 print *, "received ", x, " from 1"
 end if
end do
end if
call mpi_finalize(ierr)
end
```

Задание 5

В исходном тексте программы на языке Fortran пропущены: вызов процедуры `MPI_Wait` и некоторые другие важные фрагменты. Добавить эти вызовы, откомпилировать и запустить программу.

```
program main_mpi
…
integer rank, tag1, tag2, cnt, ierr, …
integer request
real sndbuf1, sndbuf2, rcvbuf1, rcvbuf2
cnt = 1
tag = 0
sndbuf1 = 3.14159
sndbuf2 = 2.71828
…
call mpi_comm_rank(mpi_comm_world, rank, ierr)
if (rank.eq.0) then
call mpi_ssend(sndbuf1, cnt, mpi_real, 1, tag1, mpi_comm_world,
ierr)
print *, "process ", rank, " send ", sndbuf1
call mpi_send(sndbuf2, cnt, mpi_real, 1, tag2, mpi_comm_world,
ierr)
print *, "process ", rank, " send ", sndbuf2
else
call mpi_irecv(rcvbuf1, cnt, mpi_real, 0, tag1, mpi_comm_world,
request, ierr)
call mpi_recv(rcvbuf2, cnt, mpi_real, 0, tag2, mpi_comm_world,
status, ierr)
print *, "process ", rank, " received before wait", rcvbuf1
print *, "process ", rank, " received before wait", rcvbuf2
…
print *, "process ", rank, " received after wait", rcvbuf1
print *, "process ", rank, " received after wait", rcvbuf2
end if
…
end
```