Технология программирования MPI (2)

Антонов Александр Сергеевич,
к.ф.-м.н., вед.н.с. лаборатории Параллельных информационных технологий НИВЦ МГУ

Летняя суперкомпьютерная академия
Москва, 2015

MPI

Коллективные взаимодействия процессов

MPI

В операциях коллективного взаимодействия процессов *участвуют все процессы коммуникатора*!

Как и для блокирующих процедур, возврат означает то, что разрешён свободный доступ к буферу приёма или посылки.

Сообщения, вызванные коллективными операциями, не пересекутся с другими сообщениями.

MPI

Нельзя рассчитывать на синхронизацию процессов с помощью коллективных операций (кроме процедуры **`MPI_Barrier`**).

Если какой-то процесс завершил свое участие в коллективной операции, то это не означает ни того, что данная операция завершена другими процессами коммуникатора, ни даже того, что она ими начата (если это возможно по смыслу операции).

MPI

`int MPI_Barrier(MPI_Comm comm)`

Работа процессов блокируется до тех пор, пока все оставшиеся процессы коммуникатора **`comm`** не выполнят эту процедуру. Все процессы должны вызвать **`MPI_Barrier`**, хотя реально исполненные различными процессами коммуникатора вызовы могут быть расположены в разных местах программы.

MPI

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

Рассылка **count** элементов данных типа **datatype** из массива **buf** от процесса **root** всем процессам данного коммуникатора **comm**, включая сам рассылающий процесс. Значения параметров **count**, **datatype**, **root** и **comm** должны быть одинаковыми у всех процессов.

MPI

данные

процессы

A_0

broadcast

A_0

A_0

A_0

A_0

A_0

A_0

MPI

```
int MPI_Gather(void *sbuf, int
scount, MPI_Datatype stype, void
*rbuf, int rcount, MPI_Datatype
rtype, int root, MPI_Comm comm)
```

Сборка **`scount`** элементов данных типа **`stype`** из массивов **`sbuf`** со всех процессов коммуникатора **`comm`** в буфер **`rbuf`** процесса **`root`**. Данные сохраняются в порядке возрастания номеров процессов.

MPI

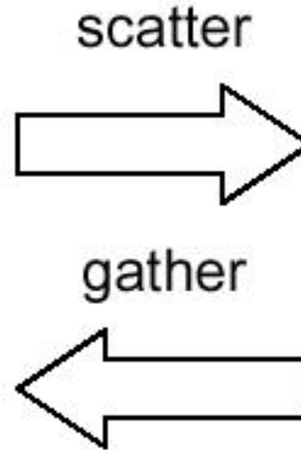
На процессе **root** существенными являются значения всех параметров, а на остальных процессах - только значения параметров **sbuf**, **scount**, **stype**, **root** и **comm**. Значения параметров **root** и **comm** должны быть одинаковыми у всех процессов. Параметр **rcount** у процесса **root** обозначает число элементов типа **rtype**, принимаемых от каждого процесса.

MPI

данные

процессы

A_0	A_1	A_2	A_3	A_4	A_5



A_0					
A_1					
A_2					
A_3					
A_4					
A_5					

MPI

```
int MPI_Gatherv(void *sbuf, int
scount, MPI_Datatype stype, void
*rbuf, int *rcounts, int
*displs, MPI_Datatype rtype, int
root, MPI_Comm comm)
```

Сборка различного количества данных из массивов **`sbuf`**. Порядок расположения задаёт массив **`displs`**.

MPI

rcounts – целочисленный массив, содержащий количество элементов, передаваемых от каждого процесса (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).

displs – целочисленный массив, содержащий смещения относительно начала массива **rbuf** (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).

MPI

```
int MPI_Scatter(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)
```

Рассылка по **scount** элементов данных типа **stype** из массива **sbuf** процесса **root** в массивы **rbuf** всех процессов коммуникатора **comm**, включая сам процесс **root**. Данные рассылаются в порядке возрастания номеров процессов.

MPI

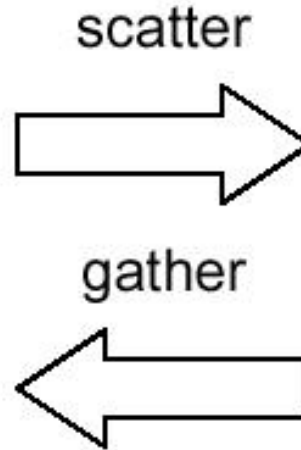
На процессе **`root`** существенными являются значения всех параметров, а на всех остальных процессах — только значения параметров **`rbuf`**, **`rcount`**, **`rtype`**, **`source`** и **`comm`**. Значения параметров **`source`** и **`comm`** должны быть одинаковыми у всех процессов.

MPI

данные

процессы

A_0	A_1	A_2	A_3	A_4	A_5



A_0					
A_1					
A_2					
A_3					
A_4					
A_5					

MPI

```
float sbuf[SIZE][SIZE], rbuf[SIZE];
if(rank == 0)
  for(i=0; i<SIZE; i++)
    for (j=0; j<SIZE; j++)
      sbuf[i][j]=…;
if (numtasks == SIZE)
  MPI_Scatter(sbuf, SIZE, MPI_FLOAT, rbuf,
SIZE, MPI_FLOAT, 0, MPI_COMM_WORLD);
```

MPI

```
int MPI_Scatterv(void *sbuf, int
*scounts, int *displs,
MPI_Datatype stype, void *rbuf,
int rcount, MPI_Datatype rtype,
int root, MPI_Comm comm)
```

Рассылка различного количества данных из массива **sbuf**. Начало рассылаемых порций задает массив **displs**.

MPI

scounts – целочисленный массив, содержащий количество элементов, передаваемых каждому процессу (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).

displs – целочисленный массив, содержащий смещения относительно начала массива **sbuf** (индекс равен рангу адресата, длина равна числу процессов в коммуникаторе).

MPI

```
int MPI_Allgather(void *sbuf,
int scount, MPI_Datatype stype,
void *rbuf, int rcount,
MPI_Datatype rtype, MPI_Comm
comm)
```

Сборка данных из массивов **sbuf** со всех процессов коммуникатора **comm** в буфере **rbuf** каждого процесса. Данные сохраняются в порядке возрастания номеров процессов.

MPI

данные

процессы

A_0					
B_0					
C_0					
D_0					
E_0					
F_0					

allgather

A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0

MPI

```
int MPI_Allgatherv(void *sbuf,
int scount, MPI_Datatype stype,
void *rbuf, int *rcounts, int
*displs, MPI_Datatype rtype,
MPI_Comm comm)
```

Сборка на всех процессах различного количества данных из **`sbuf`**. Порядок расположения задаёт массив **`displs`**.

MPI

```
int MPI_Alltoall(void *sbuf, int
scount, MPI_Datatype stype, void
*rbuf, int rcount, MPI_Datatype
rtype, MPI_Comm comm)
```

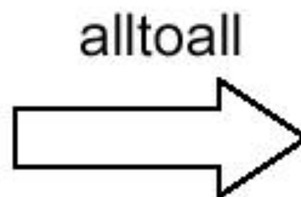
Рассылка каждым процессом коммуникатора **comm** различных порций данных всем другим процессам. **j**-й блок массива **sbuf** процесса **i** попадает в **i**-й блок массива **rbuf** процесса **j**.

MPI

данные

процессы

A_0	A_1	A_2	A_3	A_4	A_5
B_0	B_1	B_2	B_3	B_4	B_5
C_0	C_1	C_2	C_3	C_4	C_5
D_0	D_1	D_2	D_3	D_4	D_5
E_0	E_1	E_2	E_3	E_4	E_5
F_0	F_1	F_2	F_3	F_4	F_5



A_0	B_0	C_0	D_0	E_0	F_0
A_1	B_1	C_1	D_1	E_1	F_1
A_2	B_2	C_2	D_2	E_2	F_2
A_3	B_3	C_3	D_3	E_3	F_3
A_4	B_4	C_4	D_4	E_4	F_4
A_5	B_5	C_5	D_5	E_5	F_5

MPI

```
int MPI_Alltoallv(void* sbuf,
int *scounts, int *sdispls,
MPI_Datatype stype, void* rbuf,
int *rcounts, int *rdispls,
MPI_Datatype rtype, MPI_Comm
comm)
```

Рассылка со всех процессов коммуникатора **comm** различного количества данных всем другим процессам. Размещение данных в буфере **sbuf** отсылающего процесса определяется массивом **sdispls**, а в буфере **rbuf** принимающего процесса – массивом **rdispls**.

MPI

```
int MPI_Reduce(void *sbuf, void
*rbuf, int count, MPI_Datatype
datatype, MPI_Op op, int root,
MPI_Comm comm)
```

Выполнение **count** независимых глобальных операций **op** над соответствующими элементами массивов **sbuf**. Результат операции над **i**-ми элементами массивов **sbuf** получается в **i**-ом элементе массива **rbuf** процесса **root**.

MPI

Типы предопределённых глобальных операций:

MPI_MAX, MPI_MIN – максимальное и минимальное значения;

MPI_SUM, MPI_PROD – глобальная сумма и глобальное произведение;

MPI_LAND, MPI_LOR, MPI_LXOR – логические “И”, “ИЛИ”, искл. “ИЛИ”;

MPI_BAND, MPI_BOR, MPI_BXOR – побитовые “И”, “ИЛИ”, искл. “ИЛИ”.

MPI

```
int MPI_Allreduce(void *sbuf,
void *rbuf, int count,
MPI_Datatype datatype, MPI_Op
op, MPI_Comm comm)
```

Выполнение **count** независимых глобальных операций **op** над соответствующими элементами массивов **sbuf**. Результат получается в массиве **rbuf** каждого процесса.

MPI

```
for(i=0; i<n; i++) s[i]=0.0;
for(i=0; i<n; i++)
  for(j=0; j<m; j++)
    s[i]=s[i]+a[i][j];
MPI_Allreduce(s, r, n, MPI_FLOAT, MPI_SUM,
MPI_COMM_WORLD);
```

MPI

```
int MPI_Reduce_scatter(void
*sbuf, void *rbuf, int *rcounts,
MPI_Datatype datatype, MPI_Op
op, MPI_Comm comm)
```

Выполнение $\sum_i$ **rcounts(i)** независимых глобальных операций **op** над соответствующими элементами массивов **sbuf**.

MPI

Сначала выполняются глобальные операции, затем результат рассылается по процессам.

i-ый процесс получает **rcounts(i)** значений результата и помещает в массив **rbuf**.

MPI

```
int MPI_Scan(void *sbuf, void
*rbuf, int count, MPI_Datatype
datatype, MPI_Op op, MPI_Comm
comm)
```

Выполнение **count** независимых частичных глобальных операций **op** над соответствующими элементами массивов **sbuf**.

MPI

`i`-ый процесс выполняет глобальную операцию над соответствующими элементами массива **`sbuf`** процессов **`0`**...**`i`** и помещает результат в массив **`rbuf`**.

Окончательный результат глобальной операции получается в массиве **`rbuf`** последнего процесса.

MPI

```
int MPI_Op_create
(MPI_User_function *func, int
commute, MPI_Op *op)
```

Создание пользовательской глобальной операции. Если **commute=1**, то операция должна быть коммутативной и ассоциативной. Иначе порядок фиксируется по увеличению номеров процессов.

MPI

```
typedef void MPI_User_function
(void *invec, void *inoutvec,
int *len, MPI_Datatype type)
```

Интерфейс пользовательской функции.

```
int MPI_Op_free(MPI_Op *op)
```

Уничтожение пользовательской глобальной операции.

MPI

```
#include <stdio.h>
#include "mpi.h"
#define n 1000
void smod5(void *in, void *inout, int *l, MPI_Datatype
*type){
    int i;
    for(i=0; i<*l; i++) ((int*)inout)[i] = (((int*)in)[i]
+ ((int*)inout)[i])%5;
}
int main(int argc, char **argv)
{
    int rank, size, i;
    int a[n];
    int b[n];
```

MPI

```
   MPI_Op op;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   for(i=0; i<n; i++) a[i] = i + rank +1;
   printf("process %d a[0] = %d\n", rank, a[0]);
   MPI_Op_create(&smod5, 1, &op);
   MPI_Reduce(a, b, n, MPI_INT, op, 0, MPI_COMM_WORLD);
   MPI_Op_free(&op);
   if(rank==0) printf("b[0] = %d\n", b[0]);
   MPI_Finalize();
}
```

MPI

Задание 3: Напишите программу, в которой операция глобального суммирования моделируется схемой сдваивания (каскадная схема) с использованием пересылок данных типа точка-точка. Сравнить эффективность такого моделирования с использованием процедуры **`MPI_Reduce`**.

MPI

Литература

1. MPI: A Message-Passing Interface Standard Version 1.1. URL: http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html
2. MPI: A Message-Passing Interface Standard Version 2.2. URL: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf
3. MPI – the complete reference, second edition / Ed. by M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra. – The MIT Press, 1998.
4. Антонов А.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. - М.: Издательство Московского университета, 2012.-344 с.- (Серия "Суперкомпьютерное образование").
5. Антонов А.С. Введение в параллельные вычисления (методическое пособие). – М.: Изд-во Физического факультета МГУ, 2002.
6. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004.

MPI

Литература

7. Букатов А.А., Дацюк В.Н., Жегуло А.И. Программирование многопроцессорных вычислительных систем. – Ростов-на-Дону: Издательство ООО «ЦВВР», 2003.
8. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб: БХВ-Петербург, 2002.
9. Корнеев В.Д. Параллельное программирование в MPI. – Новосибирск: Изд-во СО РАН, 2000.
10. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем. – СПб: БХВ-Петербург, 2002.
11. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI: Пособие. – Минск: БГУ, 2002.