Общие процедуры MPI

В данном разделе мы остановимся на общих процедурах MPI, не связанных с пересылкой данных. Большинство процедур этого раздела необходимы практически в каждой содержательной параллельной программе.

```
MPI_INIT(IERR)
INTEGER IERR
```

Инициализация параллельной части программы. Все другие процедуры MPI могут быть вызваны только после вызова **MPI_INIT**. Инициализация параллельной части для каждого приложения должна выполняться только один раз. В языке Си функции **MPI_Init** передаются указатели на аргументы командной строки программы **argc** и **argv**, из которых системой могут извлекаться и передаваться в параллельные процессы некоторые параметры запуска программы.

```
MPI_FINALIZE(IERR)
INTEGER IERR
```

Завершение параллельной части приложения. Все последующие обращения к любым процедурам MPI, в том числе к **MPI_INIT**, запрещены. К моменту вызова **MPI_FINALIZE** каждым процессом программы все действия, требующие его участия в обмене сообщениями, должны быть завершены.

Пример простейшей MPI-программы на языке Фортран выглядит следующим образом:

```
program example1
include 'mpif.h'
integer ierr
print *, 'Before MPI_INIT'
call MPI_INIT(ierr)
print *, 'Parallel section'
call MPI_FINALIZE(ierr)
print *, 'After MPI_FINALIZE'
end
```

В зависимости от реализации MPI строчки **'Before MPI_INIT'** и **'After MPI_FINALIZE'** может печатать либо один выделенный процесс, либо все запущенные процессы приложения. Строчку **'Parallel section'** должны напечатать все процессы. Порядок вывода строк с разных процессов может быть произвольным.

Общая схема MPI-программы на языке Си выглядит примерно следующим образом:

```
#include "mpi.h"
main(int argc, char **argv)
{
 …
 MPI_Init(&argc, &argv);
 …
 MPI_Finalize();
 …
}
```

Другие параллельные программы на языке Си с использованием технологии MPI можно найти, например, в Вычислительном полигоне: http://polygon.parallel.ru.

```
MPI_INITIALIZED(FLAG, IERR)
LOGICAL FLAG
INTEGER IERR
```

Процедура возвращает в аргументе **FLAG** значение **.TRUE.**, если вызвана из параллельной части приложения, и значение **.FALSE.** - в противном случае. Это единственная процедура MPI, которую можно вызвать до вызова **MPI_INIT**.

```
MPI_COMM_SIZE(COMM, SIZE, IERR)
INTEGER COMM, SIZE, IERR
```

В аргументе **SIZE** процедура возвращает число параллельных процессов в коммуникаторе **COMM**.

```
MPI_COMM_RANK(COMM, RANK, IERR)
INTEGER COMM, RANK, IERR
```

В аргументе **RANK** процедура возвращает номер процесса в коммуникаторе **COMM**. Если процедура **MPI_COMM_SIZE** для того же коммуникатора **COMM** вернула значение **SIZE**, то значение, возвращаемое процедурой **MPI_COMM_RANK** через переменную **RANK**, лежит в диапазоне от **0** до **SIZE-1**.

В следующем примере каждый запущенный процесс печатает свой уникальный номер в коммуникаторе **MPI_COMM_WORLD** и число процессов в данном коммуникаторе.

```
      program example2
      include 'mpif.h'
      integer ierr, size, rank
      call MPI_INIT(ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
      print *, 'process ', rank, ', size ', size
      call MPI_FINALIZE(ierr)
      end
```

Строка, соответствующая вызову процедуры **print**, будет выведена столько раз, сколько процессов было порождено при запуске программы. Порядок появления строк заранее не определен и может быть, вообще говоря, любым. Гарантируется только то, что содержимое отдельных строк не будет перемешано друг с другом.

```
DOUBLE PRECISION MPI_WTIME(IERR)
INTEGER IERR
```

Эта функция возвращает на вызвавшем процессе астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом. Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка. Гарантируется, что момент времени, используемый в качестве точки отсчета, не будет изменен за время существования процесса. Заметим, что эта функция возвращает результат своей работы не через параметры, а явным образом. Таймеры разных процессоров могут быть не синхронизированы и выдавать различные значения, это можно определить по значению параметра **MPI_WTIME_IS_GLOBAL** (**1** – синхронизированы, **0** - нет).

```
DOUBLE PRECISION MPI_WTICK(IERR)
INTEGER IERR
```

Функция возвращает разрешение таймера на вызвавшем процессе в секундах. Эта функция также возвращает результат своей работы не через параметры, а явным образом.

```
MPI_GET_PROCESSOR_NAME(NAME, LEN, IERR)
CHARACTER*(*) NAME
INTEGER LEN, IERR
```

Процедура возвращает в строке **NAME** имя узла, на котором запущен вызвавший процесс. В переменной **LEN** возвращается количество символов в имени, не превышающее значения константы **MPI_MAX_PROCESSOR_NAME**. С помощью этой процедуры можно определить, на какие именно физические процессоры были спланированы процессы MPI-приложения.

В следующей программе на каждом процессе определяются две характеристики системного таймера: его разрешение и время, требуемое на замер времени (для усреднения получаемого значения выполняется **NTIMES** замеров). Также в данном примере показано использование процедуры **MPI_GET_PROCESSOR_NAME**.

```
 program example3
 include 'mpif.h'
 integer ierr, rank, len, i, NTIMES
 parameter (NTIMES = 100)
 character*(MPI_MAX_PROCESSOR_NAME) name
 double precision time_start, time_finish, tick
 call MPI_INIT(ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
 call MPI_GET_PROCESSOR_NAME(name, len, ierr)
 tick = MPI_WTICK(ierr)
 time_start = MPI_WTIME(ierr)
 do i = 1, NTIMES
    time_finish = MPI_WTIME(ierr)
 end do
 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
 print *, 'processor ', name(1:len),
&         ', process ', rank, ': tick = ', tick,
&         ', time = ', (time_finish-time_start)/NTIMES
 call MPI_FINALIZE(ierr)
 end
```

Задания

- Откомпилировать и проверить эффективность выполнения программы вычисления числа Пи на различном числе процессоров (программа обычно входит в качестве тестового примера в комплект поставки MPI и может находиться, например, в файлах **`/usr/local/examples/mpi/fpi.f`** или **`cpi.c`**).
- Можно ли в процессе работы MPI-программы порождать новые процессы, если в какой-то момент появились свободные процессоры?
- Может ли MPI-программа продолжать работу после аварийного завершения одного из процессов?
- Определить, сколько процессов выполняют текст программы до вызова процедуры **`MPI_INIT`** и после вызова процедуры **`MPI_FINALIZE`**.
- Определить, синхронизованы ли таймеры разных процессов конкретной системы.

Передача/прием сообщений между отдельными процессами

Практически все программы, написанные с использованием коммуникационной технологии MPI, должны содержать средства не только для порождения и завершения параллельных процессов, но и для взаимодействия запущенных