*Демьянович Юрий Казимирович*

# ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ УРОК 4. ПРОГРАММИРУЕМ НА MPI

Предыдущие уроки (см. [1–3]) были посвящены основным принципам распараллеливания. Там мы получили ответы на ряд естественно возникающих вопросов: что означает термин «распараллеливание алгоритма», и «распараллеливание программы», зачем вообще нужно распараллеливание, какие задачи решаются на параллельных вычислительных системах. Кроме того, стало в определенной степени ясно, как пишут параллельную программу, какова роль программиста при написании программы, что следует потребовать от программных средств распараллеливания. Подытоживая полученные там сведения, кратко ответим на перечисленные вопросы:

– суперсложные задачи требуют большого быстродействия вычислительной системы (ВС),

– технологические сложности не позволяют существенно ускорить однопроцессорные системы, так что ускорение возможно лишь за счет распараллеливания вычислений,

– количество параллельных процессоров или ядер (называем их вычислительными модулями, или просто ВМ) может быть весьма велико – сотни тысяч – так что автоматизация процесса распределения подзадач по ВМ актуальна,

– определение зон распараллеливания – трудная проблема (см. [3]); ее решение пока не под силу вычислительной системе, и потому задача программиста состоит в том, чтобы правильно указать зоны распараллеливания.

Ввиду быстрого развития технологической базы, появляются все более совершенные ВС, так что резко расширяются возможности решения суперсложных задач, однако программисты не успевают обеспечивать программами новые ВС; ввиду этого возникает проблема автоматизации параллельного программирования: для ее решения создаются институты «экзафлопных» вычислений.

Программы для суперсложных задач разрабатываются большими коллективами, и их разработка подчас занимает значительное время, поэтому весьма важно поддерживать переносимость созданных программ на новые ВС. Этому помогают стандарты параллельного программирования, среди которых особое место занимают стандарты MPI и Open MP.

На прошлом уроке мы ознакомились с некоторыми средствами стандарта MPI (а именно с подпрограммами[1] **`MPI_Bcast`** и **`MPI_Reduce`**).

Цель этого урока в том, чтобы рассмотреть другие подпрограммы этого стандарта (**`MPI_Send`**, **`MPI_Recv`**, **`MPI_Scatter`** и **`MPI_Gather`**) и написать параллельную программу приближенного вычисления определенного интеграла.

## 1. ОПЕРАЦИИ ИНДИВИДУАЛЬНОГО ОБМЕНА

В алгоритме решения задачи выделяют параллельные области, то есть такие части алгоритма, которые можно реализовывать на параллельной системе; в них рассматривают несколько не зависящих друг от друга потоков инструкций (команд), каждый из ко-

[1] Подпрограммы иногда называют процедурами.

торых предназначается определенному параллельному ВМ. Параллельные области чередуются с последовательными областями (то есть с частями алгоритма, где распараллеливание не производится). Переход от параллельной области к последовательной и наоборот требует взаимодействия параллельных процессов: один процесс посылает информацию, а другой принимает ее. Для этой цели в MPI предусмотрены операции индивидуального обмена между процессами.

Посылающий процесс использует подпрограмму **MPI_Send**, а принимающий – подпрограмму **MPI_Recv**. Обращение к первой из этих подпрограмм (на Фортране) имеет вид

```
MPI_SEND(BUF, COUNT, DATATYPE, DEST,
                        TAG, COMM,IERR)
```

Здесь

**buf** – имя буфера[1] передачи,

**count** – количество элементов в буфере передачи,

**datatype** – тип MPI каждого пересылаемого элемента,

**dest** – ранг процесса-получателя сообщения,

**tag** – тег (признак) сообщения,

**comm** – коммуникатор,

**ierr** – код завершения (если не было ошибки, то в **ierr** находится нуль).

Принимающий процесс должен иметь обращение вида

```
MPI_RECV(buf,count,datatype,source,
                  tag,comm,status,ierr)
```

В этом случае

**buf** – имя буфера приема (буфер приема должен иметь достаточный объем для приема сообщения),

**count** – максимальное количество элементов в буфере приема,

**datatype** – тип MPI принимаемых данных,

**source** – ранг процесса-источника сообщения,

**tag** – тег сообщения,

**comm** – коммуникатор,

**ierr** – код завершения.

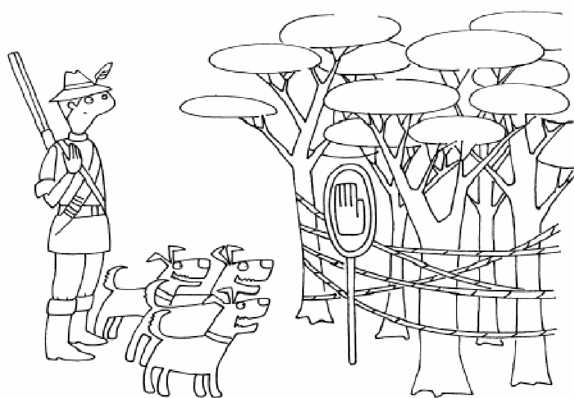
Обе рассмотренные операции обмена обладают тем свойством, что передача начинается независимо от того, был ли инициирован соответствующий прием; каждая из них заканчивается только после того, как сообщение принято. Операции, до завершения которых вычисления приостанавливаются, называются *блокирующими*. Таким образом, операции посылки/приема, реализуемые подпрограммами **MPI_Send** и **MPI_Recv**, являются блокирующими.

Перед тем как привести пример программы, использующей эти операции, напомним основные правила написания программ на Фортране.

Строки программы во многих версиях Фортрана нужно обязательно начинать с седьмой позиции и на одной строке помещать один оператор (процедуру), а если необходим перенос оператора на другую строку, в шестой позиции можно использовать символ продолжения **&**. Позиции с первой по пятую служат для размещения меток. Нумерация строк в приведенных здесь программах служит лишь для удобства пояснений и должна быть опущена при наборе программ. Обязателен символ **END** в конце программы (без него компилятор не приступит к компиляции). Отметим еще, что в Фортране (в отличие от языка С) не различаются заглавные и маленькие буквы (так что в тексте программы они используются програм-

*...один процесс посылает информацию, а другой принимает ее.*

[1] Буфером называется область памяти, идентифицируемая своим именем.



*Операции, до завершения которых вычисления приостанавливаются, называются блокирующими.*

мистом исключительно как изобразительное средство). Кроме того, если не предполагается иное, идентификаторы, начинающиеся с букв **i**, **j**, **k**, **l**, **m**, **n**, считаются имеющими тип **integer**, а идентификаторы, начинающиеся с остальных букв, – имеющими тип **real**; в указанных случаях идентификаторы можно не описывать. Отметим также, что оператор **PRINT *** означает вывод на стандартное устройство (обычно таким устройством является видеотерминал). Строки, начинающиеся с одного из символов **C**, *****, **!**, являются комментариями и не влияют на алгоритм выполнения программы.

Теперь приведем программу (на Фортране), которая иллюстрирует использование операций **MPI_Send** и **MPI_Recv** (см. листинг 1).

В первой строке объявляется имя программы, во второй строке подключается библиотека MPI, третья строка служит для описания массива типа **integer**, который содержит системную информацию (а именно поля **MPI_SOURCE**, **MPI_TAG** и **MPI_ERROR**, подробнее на этом останавливаться не будем). В четвертой строке происходит инициализация интерфейса MPI. Пятая и шестая строки служат для определения числа процессов в группе, описываемой исходным коммуникатором **MPI_COMM_WORLD**. Тэг сообщения устанавливается в единицу (см. строку 7), передача информации происходит от процесса с нулевым рангом (номером) к процессу с рангом 1 (см. строки 8–13); длина сообщения равна 1, а само сообщение представляет собой цифру 9 (см. строки 8–9). Буфером приема является **irbuf**, его содержимое печатается в строке 15. Работа интерфейса MPI завершается вызовом **MPI_FINALIZE** (см. 17-ю строку); заметим, что отсутствие подобного вызова может привести к неправильной работе программы.

**Листинг 1.**

```
 1.   PROGRAM send
 2.   INCLUDE 'mpif.h'
 3.   INTEGER istatus(MPI_STATUS_SIZE)
 4.   CALL MPI_INIT(ierr)
 5.   CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
 6.   CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
 7.   itag = 1
 8.   IF (myrank==0) THEN
 9.   isbuf = 9
10.   CALL MPI_SEND(isbuf, 1, MPI_INTEGER, 1, itag,
11.  & MPI_COMM_WORLD, ierr)
12.   ELSEIF (myrank==1) THEN
13.   CALL MPI_RECV(irbuf, 1, MPI_INTEGER, 0, itag,
14.  & MPI_COMM_WORLD, istatus, ierr)
15.   PRINT *,'irbuf =',irbuf
16.   ENDIF
17.   CALL MPI_FINALIZE(ierr)
18.   END
```

Работа рассматриваемой программы завершится распечаткой

```
1: irbuf = 9
```

## 2. КОЛЛЕКТИВНЫЙ ОБМЕН ДАННЫМИ

К числу операций, осуществляющих коллективный обмен данными, относятся **MPI_Bcast**, **MPI_Scatter**, **MPI_Gather** и некоторые другие.

Напомним, что операция **MPI_Bcast** была рассмотрена в третьей лекции. Здесь остановимся на операциях **MPI_Scatter** и **MPI_Gather**.

На Фортране обращение к операции **MPI_Scatter** имеет вид

```
MPI_SCATTER(sendbuf,sendcount,sendtype,
rcvbuf,rcvcount,rcvtype,root,comm,ierr)
```

Здесь **sendbuf** – имя буфера передачи,

**sendcount** – количество элементов, пересылаемых каждому процессу,

**sendtype** – тип MPI передаваемых данных,

**rcvbuf** – имя буфера приема,

**revcount** – количество элементов в буфере приема,

**revtype** – тип принимаемых данных,

**root** – ранг принимающего процесса,

**comm** – коммуникатор,

**ierr** – код завершения.

| A0 | A1 | A2 | A3 | A4 | A5 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

↓ **MPI_SCATTER**

| A0 | | | | | |
|---|---|---|---|---|---|
| A1 | | | | | |
| A2 | | | | | |
| A3 | | | | | |
| A4 | | | | | |
| A5 | | | | | |

**Рис. 1.** Работа подпрограммы **MPI_SCATTER**: раздача данных из одного процесса во все процессы группы

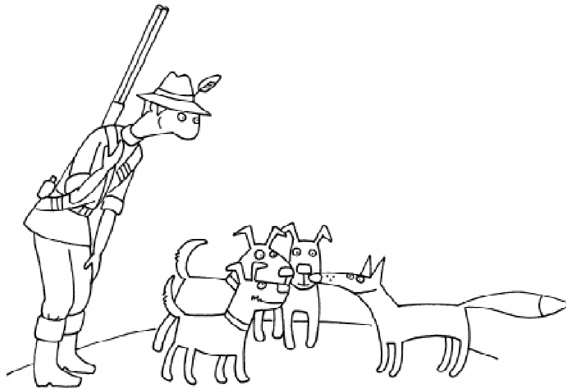
Процесс с рангом **root** распределяет содержимое буфера передачи **sendbuf** среди всех процессов: содержимое буфера передачи разбивается на несколько частей, каждая из которых содержит **sendcount** элементов.

Первая часть передается процессу с номером 0, вторая – процессу с номером 1 и т. д. (см. рис. 1). Аргументы с **send** учитываются только в процессе **root**.

Приведем пример программы, в которой фигурирует операция **MPI_Scatter** (см. листинг 2).

**Листинг 2**

```
 1.  PROGRAM scatter
 2.  INCLUDE 'mpif.h'
 3.  INTEGER isend(3)
 4.  CALL MPI_INIT(ierr)
 5.  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
 6.  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
 7.  IF (myrank==0) THEN
 8.   DO i=1,nprocs
 9.    isend(i)=i
10.   ENDDO
11.  ENDIF
12.  CALL MPI_SCATTER(isend, 1, MPI_INTEGER,
13. & irecv, 1, MPI_INTEGER, 0,
14. & MPI_COMM_WORLD, ierr)
15.  PRINT *,'irecv =',irecv
16.  CALL MPI_FINALIZE(ierr)
17.  END
```



*Коллективный обмен данными*

| A0 | | | | | |
|---|---|---|---|---|---|
| A1 | | | | | |
| A2 | | | | | |
| A3 | | | | | |
| A4 | | | | | |
| A5 | | | | | |

↓ **MPI_GATHER**

| A0 | A1 | A2 | A3 | A4 | A5 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

**Рис. 2.** Работа подпрограммы **MPI_GATHER**: сбор данных из всех процессов группы в один процесс

В результате работы этой программы на стандартном устройстве появится распечатка

```
0: irecv = 1
1: irecv = 2
2: irecv = 3
```

Операция **MPI_Gather** производит в определенном смысле обратное действие.

Обращение к ней имеет вид

```
MPI_GATHER(sendbuf, sendcount,
sendtype,rcvbuf,rcvcount,
rcvtype,root,comm,ierr)
```

где

**sendbuf** – имя буфера передачи,

**sendcount**– количество элементов, пересылаемых каждому процессу,

**sendtype** – тип MPI передаваемых данных,

**rcvbuf** – имя буфера приема,

**revcount** – количество элементов в буфере приема,

**revtype** – тип принимаемых данных,

**root** – ранг принимающего процесса,

**comm** – коммуникатор,

**ierr** – код завершения.

Здесь процесс **root** соединяет получаемые данные в буфере приема. Порядок соединения соответствует возрастанию рангов соединяемых процессов (см. рис. 2). Аргументы с **rcv** учитываются лишь процессом **root**.

Приведем пример, иллюстрирующий применение подпрограммы **MPI_GATHER** (см. листинг 3).

**Листинг 3**

```
  PROGRAM gather
  INCLUDE 'mpif.h'
  INTEGER irecv(3)
  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  isend = myrank + 1
  CALL MPI_GATHER(isend, 1, MPI_INTEGER,
 &irecv, 1, MPI_INTEGER, 0, & MPI_COMM_WORLD, ierr)
  IF (myrank==0) THEN
  PRINT *,'irecv =',irecv
  ENDIF
  CALL
  MPI_FINALIZE(ierr)
  END
```

В результате работы этой программы на стандартном устройстве появится распечатка

```
0: irecv = 1 2 3
```

## 3. ПРИМЕНЕНИЕ КВАДРАТУРНОЙ ФОРМУЛЫ ДЛЯ ВЫЧИСЛЕНИЯ ЧИСЛА π НА ПАРАЛЛЕЛЬНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Число $\pi$ представляет собой отношение длины окружности к диаметру, однако, вычисление его с использованием этого определения с надлежащей точностью затруднительно.

Это число встречается во многих естественных науках, ибо оно отражает существенные закономерности природы. В некоторых формулах неевклидовой геометрии также участвует число $\pi$, но уже не как отношение длины окружности к диаметру (там это отношение не является постоянным). Число $\pi$ иррационально и выражается бесконечной непериодической десятичной дробью

$$\pi = 3.141592653589793...$$

К этому числу, в частности, приводит отыскание пределов некоторых арифметических последовательностей, составляемых по простым законам. Например, ряд Лейбница

$$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-...$$

сходится к числу $\pi/4$ (но весьма медленно). Один из эффективных способов отыскания этого числа – использование определенного интеграла для представления арктангенса с аргументом, равным единице:

$$\int_0^1 \frac{dx}{1+x^2} = arctg\,1 = \pi/4 . \qquad (1)$$

Интеграл (1) будем вычислять приближенно, используя квадратурную формулу *средних прямоугольников*

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n} f(a+(i-1/2)h), \qquad (2)$$

где

$$h \overset{def}{=} (b-a)/n . \qquad (3)$$

В рассматриваемом случае получаем

$$\int_0^1 \frac{dx}{1+x^2} \approx h\sum_{i=1}^{n}\frac{1}{1+((i-1/2)h)^2},$$
$$h \overset{def}{=} 1/n . \qquad (4)$$

Кроме приведенной формулы средних прямоугольников можно было бы использовать другие квадратурные формулы: формулу трапеций, формулу Симпсона и т. п. Все квадратурные формулы представляют собой линейную комбинацию значений функции (а иногда – и некоторых ее производных), вычисленных в точках, называемых узлами квадратурной формулы. Коэффициентами таких линейных комбинаций являются некоторые априори заданные фиксированные числа, называемые коэффициентами квадратурной формулы.

Напомним квадратурные формулы трапеций и Симпсона (в следующих ниже формулах $h$ вычисляется по формуле (3)).

*Формула трапеций:*

$$\int_a^b f(x)dx \approx h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+ih)\right]. \qquad (5)$$

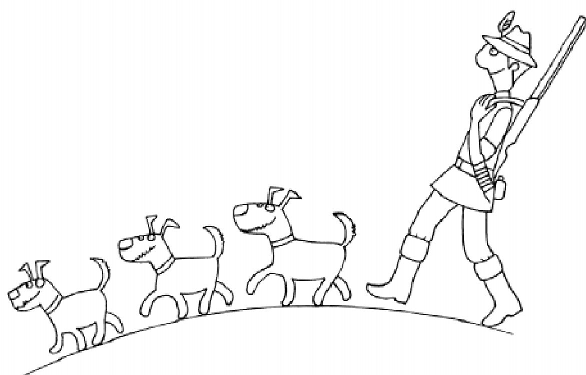
*Формула Симпсона:*

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[\frac{f(a)+f(b)}{2} + 2\sum_{i=1}^{n} f(a+(i-1/2)h) + \sum_{i=1}^{n-1} f(a+ih)\right]. \qquad (6)$$

Здесь ограничимся распараллеливаем вычислений с использованием формулы средних прямоугольников, хотя аналогичным образом можно распараллелить и другие квадратурные формулы.

Следующая программа (на Фортране) дает иллюстрацию такого использования для вычисления числа $\pi$ по формуле (4) (см. листинг 4).

При использовании стандарта MPI программа копируется во все рассматриваемые параллельные процессы; при этом, конечно, каждый процесс – в зависимости от своего номера – выполняет специфическую для него работу.



*Порядок соединения соответствует возрастанию рангов соединяемых процессов.*

Первая строка программы объявляет имя программы, во второй строке подключается стандартная библиотека **mpif.h**, реализующая стандарт MPI для Фортрана, третья и четвертая строки дают описания переменных. Пятая строка представляет собой комментарий. В шестой строке описана функция, которую предполагается интегрировать, строки 7, 8, 9 используются для вызова подпрограмм MPI. Рассмотрим вызываемые здесь подпрограммы подробнее (иногда повторяя сказанное ранее).

Процедура **MPI_Init(ierr)** инициализирует библиотеку MPI (ее применение обя-

**Листинг 4.**

```
 1     program compute_pi
 2      include 'mpif.h'
 3      double precision my, pi, w, sum, x, f, a
 4      integer data, oper
 5  c   function to integrate
 6      f(a)=1.e0/(1.e0+a*a)
 7      call MPI_INIT(ierr)
 8     call MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)
 9     call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
10      if (myrank.eq.0) then
11       print*,'Enter number of iterations:'
12       read*,n
13      endif
14      call MPI_BCAST(n,1,MPI_INTEGER,0,
       & MPI_COMM_WORLD,ierr)
15  c   calculate the integral size
16      h=1.0e0/n
17      sum=0.0e0
18      do i=myrank+1,n,numprocs
19       x=h*(i-0.5e0)
20       sum=sum+f(x)
21      enddo
22      my=4.0*h*sum
23  c   collect all the partial sums
24      data=MPI_DOUBLE_PRECISION
25      oper=MPI_SUM
26     call MPI_Reduce(my,pi,1,data,oper,0,
      & MPI_COMM_WORLD,ierr)
27  c   node 0 prints the answer
28      if(myrank.eq.0) then
29       print*,'computed pi=',pi
30      endif
31      call MPI_FINELIZE(ierr)
32       stop
33    end
```

зательно перед тем, как начать использование упомянутой библиотеки).

Процедура **MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)** позволяет определить число процессов в группе с коммуникатором **MPI_COMM_WORLD**; это число присваивается выходному параметру **nprocs**.

Процедура **MPI_Comm_rank(MPI_COMM_WORLD, myrank,ierr)** определяет номер процесса (он присваивается переменной **myrank**) в группе с коммуникатором **MPI_COMM_WORLD**.

Этот коммуникатор определяет группу всех процессов, запущенных для решения данной задачи. Процессы в любой группе нумеруются целыми числами от **0** до **nprocs-1**.

В строках 10–13 инициируется ввод числа **n** слагаемых в сумме (4) из процесса с номером **0** (в MPI привилегированного процесса не существует: роль такого процесса может выполнять любой процесс по желанию пользователя).

В строке 14 работает подпрограмма **MPI_Bcast(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)**, которая производит рассылку информации (в данном случае числа *n*) из процесса с номером **0** по всем процессам группы с коммуникатором **MPI_COMM_WORLD**. Здесь

первый параметр – указатель на рассылаемые значения (в данном случае рассылается число **n**),

второй параметр – количество рассылаемых значений (в данном случае их количество равно 1, ибо рассылается одно число **n**),

третий параметр означает тип рассылаемого значения (здесь должен использоваться тип, определенный в стандарте MPI: в нашем случае это **MPI_INTEGER**),

четвертый параметр – номер рассылающего процесса (в нашем случае его номер равен **0**),

пятый параметр – имя коммуникатора группы; в нашем случае это имя **MPI_COMM_WORLD**,

шестой параметр **ierr** хранит код ошибки (при ее отсутствии этот код равен нулю: **ierr=0**).

В 16-й строке отыскивается число **h** (см. формулу (4)). Напомним, что программа копируется во все параллельные процессы, каждая переменная присутствует в каждом процессе под тем же именем, и каждый процесс рассматривает эту переменную как свою собственность, недоступную другим процессам. Таким образом, значение **h** вычисляется независимо в каждом процессе.

Дальше (см. строки 18–21) каждый процесс вычисляет часть квадратурной суммы, выбирая из суммы, фигурирующей в квадратурной формуле, слагаемые, номера которых сравнимы по модулю **nprocs** с его номером (напомним, что **nprocs** содержит общее число процессов). Благодаря такому распределению, количества слагаемых, поручаемых тому или иному процессу, хорошо сбалансированы (они отличаются друг от друга не более чем на одно слагаемое). Таким образом, каждый процесс заполняет свою копию переменной **sum** определенной частью всей суммы, которую необходимо было вычислить.

В 22-й строке полученная процессом сумма умножается на **4h**; такое умножение позволяет своевременно «нормализовать» результат вычислений в данном процессе, с тем чтобы по-возможности он находился в середине диапазона представимости чисел с плавающей точкой (после сложения всех полученных сумм нарушение этого правила, возможно, привело бы к ухудшению точности вычислений за счет сдвига к краю упомянутого диапазона). В каждом процессе результат получается в принадлежащей ему переменной **term**.

Строки 24–26 предназначены для сложения (полученных процессами значений переменных с именем **my**) частей интересующей нас суммы с помощью подпрограммы

**MPI_Reduce(my, pi, 1,data,oper, 0, MPI_COMM_WORLD,ierr)**,

где

первый параметр представляет собой адрес обрабатываемых переменных (в нашем случае адрес **my**),

второй параметр является адресом переменной, которой присваивается результат обработки (в данном случае, это – адрес переменной **pi**, а «обработка» представляет собой сложение),

третий параметр – число передаваемых данных из каждого процесса (в рассматриваемом случае передается по одному данному – слагаемому **sum**),

четвертый параметр определяет тип данных, подвергающихся обработке (в данном случае **data**),

пятый параметр определяет операцию обработки данных (в нашем случае это операция сложения, идентифицируемая константой **MPI_SUM**),

шестой параметр указывает номер процесса, в котором сохраняется результат обработки (в данном случае указан процесс с номером 0),

седьмой параметр указывает имя коммуникатора (у нас это **MPI_COMM_WORLD**),

наконец, восьмой параметр **ierr** предназначен для кода ошибки (**ierr=0**, если обработка прошла успешно).

Строки 28–30 содержат вывод полученного значения $\pi$ на стандартное устройство вывода.

Строка 31 содержит вызов функции **MPI_Finalize(ierr)**, чем всегда завершается работа с библиотекой MPI, и строки 32–33 содержат стандартное завершение программы на Фортране.

### 4. ЗАПУСК ПРОГРАММЫ[3]

Для запуска программы следует

1) в любом редакторе набрать программу, представленную в пункте 3, и сохранить ее в файле с именем **comp_pi.f**,

2) выйти в операционную среду и определить доступ к транслятору **mpif77** с помощью команды

```
>> echo $PATH
```

(если доступа нет, то установить путь командой **PATH**),

3) оттранслировать набранную программу командой

```
>> comp_pi -o msnrv
```

4) запустить результат трансляции на счет командой

```
>> mpirun -np <number of procs> -
machinefile machines msnrv
```

где вместо **<number of procs>** должно стоять число процессов (натуральное число, не превосходящее количества процессов, предусмотренных в файле **machines**); ввести параметр **n** и прочитать результат.

В заключение предлагается ответить на некоторые вопросы, а при возможности использования интерфейса MPI – решить некоторые задачи на компьютере.

1. Зачем нужно распараллеливание программ?
2. Каковы пути ускорения процесса вычислений при использовании современных вычислительных систем?
3. Почему необходимо автоматизировать процесс распараллеливания?
4. Что такое параллельные и последовательные зоны алгоритма?
5. Какие операции в MPI относятся к операциям индивидуального обмена?
6. Какие коллективные операции обмена в MPI вам известны?
7. Какой смысл вкладывается в понятие «блокирующая операция»?
8. Что означает термин «буфер» в программировании?
9. Что такое «ранг процесса» в MPI?
10. Какую роль играет коммуникатор в MPI?
11. Какое значение имеет «код завершения», определяемый переменной **ierr**?
12. Почему параллельные программы пишут лишь на алгоритмических языках C и Fortran?
13. В чем состоят основные правила написания программ на Фортране?
14. Модифицируйте программу вычисления числа $\pi$ из третьего раздела, заменив формулу средних прямоугольников на формулу трапеций (см. формулу (5)).
15. Проведите еще одну модификацию только что указанной программы: используйте для вычисления интеграла формулу Симпсона, даваемую соотношением (6).

---

[1] Способ запуска программы зависит от варианта установки матобеспечения на вычислительной системе. Здесь приводится один из возможных вариантов (если этот вариант не приводит к результату, проконсультируйтесь с системным программистом, который осуществлял установку матобеспечения).

16. Следует ли ожидать ускорения вычисления интеграла

а) при использовании формулы трапеций?

б) при использовании формулы Симпсона?

17. Попробуйте запустить программы, приведенные во втором пункте на параллельной системе (см. пункт 4 и имеющуюся там сноску). Удалось ли получить указанные в этих программах результаты?

18. Запустите на параллельной системе программу, данную в пункте 3, а также ее модификации, сделанные согласно задачам 14 и 15. Сравните полученные результаты.

## Литература


1. *Демьянович Ю.К.* Параллельные вычисления. Урок 1. Параллельная форма // Компьютерные инструменты в школе, 2011. № 1. С. 36–39.
2. *Демьянович Ю.К.* Параллельные вычисления. Урок 2. О средствах распараллеливания... Элементарная параллельная программа // Компьютерные инструменты в школе, 2011. № 2. С. 17–21.
3. *Демьянович Ю.К.* Параллельные вычисления. Урок 3. О проблемах распараллеливания // Компьютерные инструменты в школе. 2011. № 3. С. 35–41.
4. *Бартеньев О.В.* Фортран для студентов. М. 1999.
5. *Немнюгин С.А., Стесик О.Л.* Параллельное программирование для многопроцессорных вычислительных систем. СПб., 2002.
6. *Бурова И.Г., Демьянович Ю.К.* Алгоритмы параллельных вычислений и программирование. Курс лекций. СПб: Изд-во СПб. ун-та, 2007.
7. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб: БХВ-Петербург, 2002.


***Демьянович Юрий Казимирович,
доктор физико-математических наук, профессор, заведующий кафедрой параллельных алгоритмов математико-механического факультета СПбГУ.***

Наши авторы, 2011.
Our authors, 2011.