Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Раздел 4 1.

Параллельное программирование на основе MPI

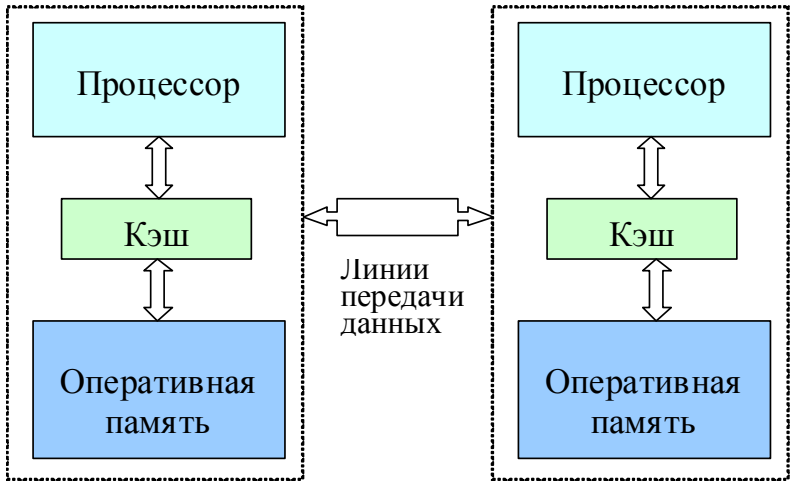
Гергель В.П., профессор, д.т.н.
Кафедра математического обеспечения ЭВМ

Содержание

- MPI: основные понятия и определения
- Введение в MPI
  - Инициализация и завершение MPI программ
  - Определение количества и ранга процессов
  - Прием и передача сообщений
  - Определение времени выполнение MPI программы
  - Коллективные операции передачи данных
- Пример: программа вычисления числа π
- Заключение

Введение…

В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга.



Для организации параллельных вычислений необходимо уметь:

– *распределять* вычислительную нагрузку,
– *организовать* информационное взаимодействие (*передачу данных*) между процессорами.

Решение всех перечисленных вопросов обеспечивает MPI - интерфейс передачи данных (message passing interface)

Введение…

- В рамках MPI для решения задачи разрабатывается одна программа, она запускается на выполнение одновременно на всех имеющихся процессорах
- Для организации различных вычислений на разных процессорах:
  - Есть возможность подставлять разные данные для программы на разных процессорах,
  - Имеются средства для идентификации процессора, на котором выполняется программа
- Такой способ организации параллельных вычислений обычно именуется как *модель "одна программа множество процессов"* (*single program multiple processes* or *SPMP*)

Введение…

- В MPI существует множество операций передачи данных:
  - Обеспечиваются разные способы пересылки данных,
  - Реализованы практически все основные коммуникационные операции.

Эти возможности являются наиболее сильной стороной MPI (об этом, в частности, свидетельствует и само название MPI)

Введение…

Что означает MPI?

- MPI - это стандарт, которому должны удовлетворять средства организации передачи сообщений.
- MPI – это программные средства, которые обеспечивают возможность передачи сообщений и при этом соответствуют всем требованиям стандарта MPI:
  - программные средства должны быть организованы в виде библиотек программных модулей (*библиотеки MPI*),
  - должны быть доступны для наиболее широко используемых алгоритмических языков C и Fortran.

Введение…

Достоинства MPI

- MPI позволяет существенно снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами.
- MPI содействует повышению эффективности параллельных вычислений - практически для каждого типа вычислительных систем существуют реализации библиотек MPI.
- MPI уменьшает сложность разработки параллельных программ:
  - большая часть основных операций передачи данных предусматривается стандартом MPI,
  - имеется большое количество библиотек параллельных методов, созданных с использованием MPI.

Введение

История разработки MPI

1992 г. Начало работ над стандартом библиотеки передачи сообщений (Oak Ridge National Laboratory, Rice University).

Ноябрь 1992 г. Объявление рабочего варианта стандарта MPI 1.

Ноябрь 1993 г. Обсуждение стандарта на конференции Supercomputing'93.

5 мая 1994 г. Окончательный вариант стандарта MPI 1.0.

12 Июня 1995 г. Новая версия стандарта - MPI 1.1.

18 Июля 1997 г. Опубликован стандарт MPI-2: Extensions to the Message-Passing Interface.

*Разработка стандарта MPI производится международным консорциумом **MPI Forum***

MPI: основные понятия и определения…

Понятие параллельной программы

- Под *параллельной программой* в рамках MPI понимается множество одновременно выполняемых *процессов*:
  - Процессы могут выполняться на разных процессорах; вместе с этим, на одном процессоре могут располагаться несколько процессов,
  - Каждый процесс параллельной программы порождается на основе копии одного и того же программного кода (*модель SPMP*).
- Исходный программный код разрабатывается на алгоритмических языках C или Fortran с использованием библиотеки MPI.
- Количество процессов и число используемых процессоров определяется в момент запуска параллельной программы средствами среды исполнения MPI программ. Все процессы программы последовательно перенумерованы. Номер процесса именуется *рангом* процесса.

MPI: основные понятия и определения…

В основу MPI положены четыре основные концепции:

- Тип операции передачи сообщения
- Тип данных, пересылаемых в сообщении
- Понятие коммуникатора (*группы процессов*)
- Понятие виртуальной топологии

MPI: основные понятия и определения…

Операции передачи данных

- Основу MPI составляют операции передачи сообщений.
- Среди предусмотренных в составе MPI функций различаются:
  - парные (*point-to-point*) операции между двумя процессами,
  - коллективные (*collective*) коммуникационные действия для одновременного взаимодействия нескольких процессов.

MPI: основные понятия и определения…

Понятие коммуникаторов…

- *Коммуникатор* в MPI - специально создаваемый служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (*контекст)*:
  - парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору,
  - Коллективные операции применяются одновременно для всех процессов коммуникатора.
- Указание используемого коммуникатора является обязательным для операций передачи данных в MPI.

MPI: основные понятия и определения…

Понятие коммуникаторов

- В ходе вычислений могут создаваться новые и удаляться существующие коммуникаторы.
- Один и тот же процесс может принадлежать разным коммуникаторам.
- Все имеющиеся в параллельной программе процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI_COMM_WORLD.
- При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (*intercommunicator*).

MPI: основные понятия и определения…

Типы данных

- При выполнении операций передачи сообщений для указания передаваемых или получаемых данных в функциях MPI необходимо указывать тип пересылаемых данных.
- MPI содержит большой набор *базовых типов данных*, во многом совпадающих с типами данных в алгоритмических языках C и Fortran.
- В MPI имеются возможности для создания новых *производных типов данных* для более точного и краткого описания содержимого пересылаемых сообщений.

MPI: основные понятия и определения

Виртуальные топологии

- Логическая топология линий связи между процессами имеет структуру полного графа (независимо от наличия реальных физических каналов связи между процессорами).
- В MPI имеется возможность представления множества процессов в виде *решетки* произвольной размерности При этом, граничные процессы решеток могут быть объявлены соседними и, тем самым, на основе решеток могут быть определены структуры типа *тор*.
- В MPI имеются средства и для формирования логических (виртуальных) топологий любого требуемого типа.

Основы MPI…

- Инициализация и завершение MPI программ
  - *Первой вызываемой функцией* MPI должна быть функция:

```
int MPI_Init ( int *agrc, char ***argv )
```

(служит для инициализации среды выполнения MPI программы; параметрами функции являются количество аргументов в командной строке и текст самой командной строки.)

  - *Последней вызываемой функцией* MPI обязательно должна являться функция:

```
int MPI_Finalize (void)
```

Основы MPI…

- Инициализация и завершение MPI программ
  - структура параллельной программы, разработанная с использованием MPI, должна иметь следующий вид:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  <программный код без использования MPI функций>
  MPI_Init ( &agrc, &argv );
    <программный код с использованием MPI функций >
  MPI_Finalize();
  <программный код без использования MPI функций >
  return 0;
}
```

Основы MPI…

- Определение количества и ранга процессов…
  - Определение *количества процессов* в выполняемой параллельной программе осуществляется при помощи функции:

```
int MPI_Comm_size ( MPI_Comm comm, int *size )
```

  - Для определения *ранга процесса* используется функция:

```
int MPI_Comm_rank ( MPI_Comm comm, int *rank )
```

Основы MPI…

- Определение количества и ранга процессов…
  - Как правило, вызов функций *MPI_Comm_size* и *MPI_Comm_rank* выполняется сразу после *MPI_Init*:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  int ProcNum, ProcRank;
  <программный код без использования MPI функций>
  MPI_Init ( &agrc, &argv );
  MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
    <программный код с использованием MPI функций >
  MPI_Finalize();
  <программный код без использования MPI функций >
  return 0;
}
```

Основы MPI…

- Определение количества и ранга процессов…
  - Коммуникатор *MPI_COMM_WORLD* создается по умолчанию и представляет все процессы выполняемой параллельной программы;
  - Ранг, получаемый при помощи функции *MPI_Comm_rank*, является рангом процесса, выполнившего вызов этой функции, и, тем самым, переменная *ProcRank* будет принимать различные значения в разных процессах.

Основы MPI…

- Передача сообщений…
  - Для *передачи сообщения* процесс-отправитель должен выполнить функцию:

```
int MPI_Send(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm),
где
  - buf   – адрес буфера памяти, в котором располагаются данные
            отправляемого сообщения,
  - count – количество элементов данных в сообщении,
  - type  - тип элементов данных пересылаемого сообщения,
  - dest  - ранг процесса, которому отправляется сообщение,
  - tag   - значение-тег, используемое для идентификации
            сообщений,
  - comm  - коммуникатор, в рамках которого выполняется передача
            данных.
```

Основы MPI…

- Передача сообщений…

Базовые типы данных MPI для алгоритмического языка C

MPI_Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Основы MPI…

- Передача сообщений
  - Отправляемое сообщение определяется через указание блока памяти (*буфера*), в котором это сообщение располагается. Используемая для указания буфера триада (*buf, count, type*) входит в состав параметров практически всех функций передачи данных,
  - Процессы, между которыми выполняется передача данных, обязательно должны принадлежать коммуникатору, указываемому в функции *MPI_Send*,
  - Параметр *tag* используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное целое число.

Основы MPI…

- Прием сообщений…
  - Для *приема сообщения* процесс-получатель должен выполнить функцию:

```
int MPI_Recv(void *buf, int count, MPI_Datatype type,
  int source,int tag, MPI_Comm comm, MPI_Status *status),
где
  - buf, count, type – буфер памяти для приема сообщения
  - source - ранг процесса, от которого должен быть выполнен прием
             сообщения,
  - tag    - тег сообщения, которое должно быть принято для
             процесса,
  - comm   - коммуникатор, в рамках которого выполняется передача
             данных,
  - status – указатель на структуру данных с информацией о
             результате выполнения операции приема данных.
```

Основы MPI…

- Прием сообщений…
  - Буфер памяти должен быть достаточным для приема сообщения, а тип элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения,
  - При необходимости приема сообщения от любого процесса-отправителя для параметра *source* может быть указано значение *MPI_ANY_SOURCE*,
  - При необходимости приема сообщения с любым тегом для параметра *tag* может быть указано значение *MPI_ANY_TAG*,

Основы MPI…

- Прием сообщений…
  - Параметр *status* позволяет определить ряд характеристик принятого сообщения:

```
-status.MPI_SOURCE – ранг процесса-отправителя принятого сообщения,
-status.MPI_TAG    - тег принятого сообщения.
```

Функция

```
MPI_Get_count(MPI_Status *status, MPI_Datatype type, int *count )
```

возвращает в переменной *count* количество элементов типа *type* в принятом сообщении.

Основы MPI…

- Прием сообщений

  Функция *MPI_Recv* является *блокирующей* для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если по каким-то причинам ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.

Основы MPI…

- Первая параллельная программа с использованием MPI…
  - Каждый процесс определяет свой ранг, после чего действия в программе разделяются (разные процессы выполняют различные действия),
  - Все процессы, кроме процесса с рангом 0, передают значение своего ранга нулевому процессу,
  - Процесс с рангом 0 сначала печатает значение своего ранга, а далее последовательно принимает сообщения с рангами процессов и также печатает их значения,
  - Возможный вариант результатов печати процесса 0:

```
Hello from process 0
Hello from process 2
Hello from process 1
Hello from process 3
```

Основы MPI…

- Первая параллельная программа с использованием MPI (замечания)…
  - Порядок приема сообщений заранее не определен и зависит от условий выполнения параллельной программы (более того, этот порядок может изменяться от запуска к запуску). Если это не приводит к потере эффективности, следует обеспечивать однозначность расчетов и при использовании параллельных вычислений:

```
MPI_Recv(&RecvRank, 1, MPI_INT, i, MPI_ANY_TAG,
         MPI_COMM_WORLD, &Status)
```

Указание ранга процесса-отправителя регламентирует порядок приема сообщений.

Основы MPI…

- Первая параллельная программа с использованием MPI (замечания)…
  - Можно рекомендовать при увеличении объема разрабатываемых программ выносить программный код разных процессов в отдельные программные модули (функции). Общая схема MPI программы в этом случае будет иметь вид:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoProcess0();
else if ( ProcRank == 1 ) DoProcess1();
else if ( ProcRank == 2 ) DoProcess2();
```

Основы MPI

- Первая параллельная программа с использованием MPI (замечания)
  - Для контроля правильности выполнения все функции MPI возвращают в качестве своего значения *код завершения*. При успешном выполнении функции возвращаемый код равен *MPI_SUCCESS*. Другие значения кода завершения свидетельствуют об обнаружении тех или иных ошибочных ситуаций в ходе выполнения функций:

```
- MPI_ERR_BUFFER – неправильный указатель на буфер,
- MPI_ERR_COMM – неправильный коммуникатор,
- MPI_ERR_RANK – неправильный ранг процесса
и др.
```

Определение времени выполнение MPI программы

- Необходимо определять время выполнения вычислений для оценки достигаемого ускорения за счет использования параллелизма,
- Получение времени текущего момента выполнения программы обеспечивается при помощи функции:

```
double MPI_Wtime(void)
```

- Точность измерения времени может зависеть от среды выполнения параллельной программы. Для определения текущего значения точности может быть использована функция:

```
double MPI_Wtick(void)
```

(время в секундах между двумя последовательными показателями времени аппаратного таймера используемой системы)

Начальное знакомство с коллективными операциями передачи данных…

- Будем использовать учебную *задачу суммирования* элементов вектора *x*:

$$S = \sum_{i=1}^{n} x_i$$

- Для решения необходимо разделить данные на равные блоки, передать эти блоки процессам, выполнить в процессах суммирование полученных данных, собрать значения вычисленных частных сумм на одном из процессов и сложить значения частичных сумм для получения общего результата решаемой задачи,
- Для более простого изложения примера процессам программы будут передаваться весь суммируемый вектор, а не отдельные блоки этого вектора.

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы…
  - Необходимо передать значения вектора *x* всем процессам параллельной программы,
  - Можно воспользоваться рассмотренными ранее функциями парных операций передачи данных:

```
MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
for (i=1; i<ProcNum; i++)
  MPI_Send(&x,n,MPI_DOUBLE,i,0,MPI_COMM_WORLD);
```

Повторение операций передачи приводит к суммированию затрат (латентностей) на подготовку передаваемых сообщений,

Данная операция может быть выполнена за меньшее число операций передачи данных.

Введение в разработку параллельных программ с использованием MPI…

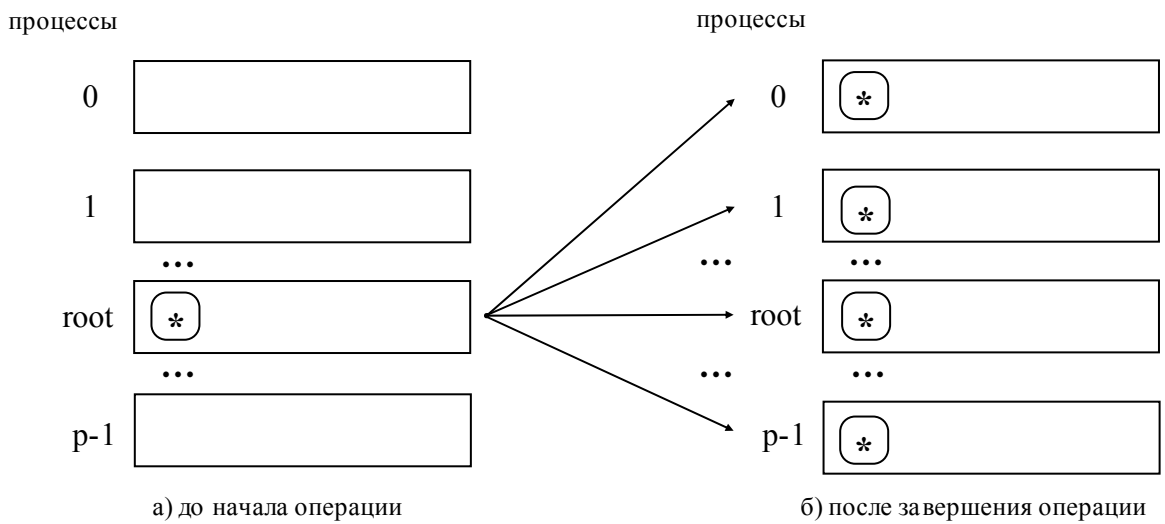
Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы…
  - *Широковещательная рассылка* данных может быть обеспечена при помощи функции MPI:

```
int MPI_Bcast(void *buf,int count,MPI_Datatype type,
       int root,MPI_Comm comm),
где
- buf, count, type – буфер памяти с отправляемым
          сообщением (для процесса с рангом 0), и для
          приема сообщений для всех остальных процессов,
- root - ранг процесса, выполняющего рассылку данных,
- comm - коммуникатор, в рамках которого выполняется
          передача данных.
```

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы…
  - Функция *MPI_Bcast* осуществляет рассылку данных из буфера *buf*, содержащего *count* элементов типа *type* с процесса, имеющего номер *root*, всем процессам, входящим в коммуникатор *comm*



Введение в разработку параллельных программ с использованием MPI…

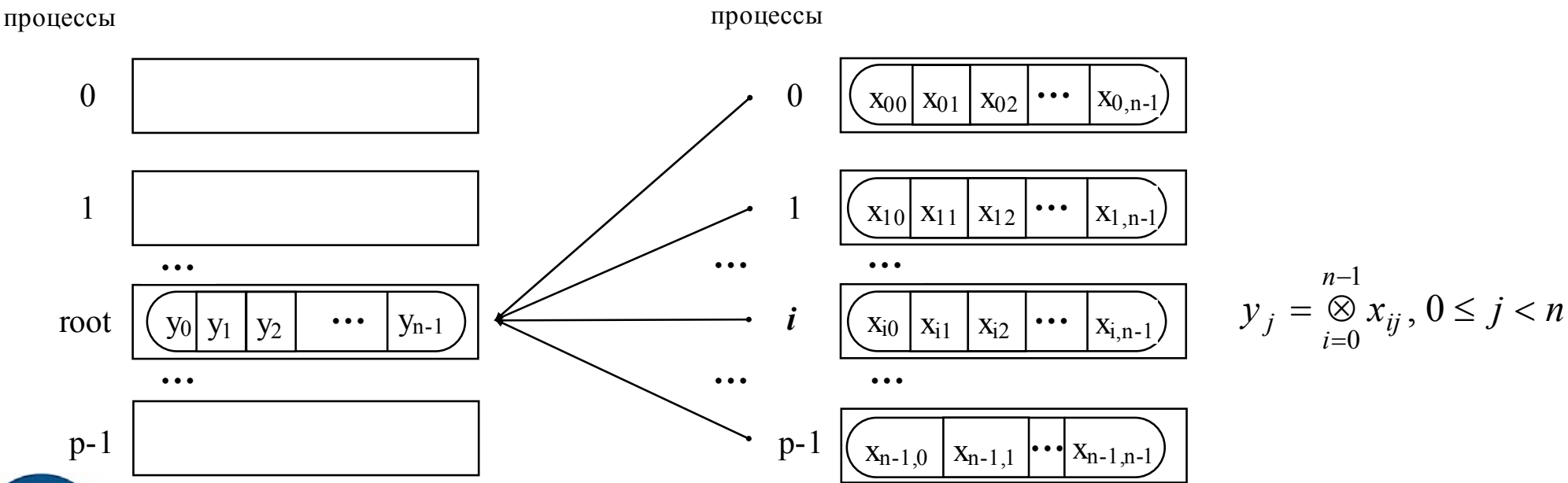
Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы
  - Функция *MPI_Bcast* определяет коллективную операцию, вызов функции *MPI_Bcast* должен быть осуществлен всеми процессами указываемого коммуникатора,
  - Указываемый в функции *MPI_Bcast* буфер памяти имеет различное назначение в разных процессах:
    - Для процесса с рангом *root*, с которого осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение.
    - Для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных.

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу…
  - Процедура сбора и последующего суммирования данных является примером часто выполняемой коллективной *операции передачи данных от всех процессов одному процессу*, в которой над собираемыми значениями осуществляется та или иная обработка данных.



$$y_j = \bigotimes_{i=0}^{n-1} x_{ij},\ 0 \le j < n$$

а) после завершения операции

б) до начала операции

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу…

```
int MPI_Reduce(void *sendbuf, void *recvbuf,int count,
      MPI_Datatype type, MPI_Op op,int root,MPI_Comm comm),
```

где

- **sendbuf** - буфер памяти с отправляемым сообщением,
- **recvbuf** – буфер памяти для результирующего сообщения (только для процесса с рангом root),
- **count** - количество элементов в сообщениях,
- **type** – тип элементов сообщений,
- **op** - операция, которая должна быть выполнена над данными,
- **root** - ранг процесса, на котором должен быть получен результат,
- **comm** - коммуникатор, в рамках которого выполняется операция.

Начальное знакомство с коллективными операциями передачи данных…

- Типы операций MPI для функций редукции данных…

Операция	Описание
MPI_MAX	Определение максимального значения
MPI_MIN	Определение минимального значения
MPI_SUM	Определение суммы значений
MPI_PROD	Определение произведения значений
MPI_LAND	Выполнение логической операции "И" над значениями сообщений
MPI_BAND	Выполнение битовой операции "И" над значениями сообщений
MPI_LOR	Выполнение логической операции "ИЛИ" над значениями сообщений
MPI_BOR	Выполнение битовой операции "ИЛИ" над значениями сообщений
MPI_LXOR	Выполнение логической операции исключающего "ИЛИ" над значениями сообщений
MPI_BXOR	Выполнение битовой операции исключающего "ИЛИ" над значениями сообщений
MPI_MAXLOC	Определение максимальных значений и их индексов
MPI_MINLOC	Определение минимальных значений и их индексов

Начальное знакомство с коллективными операциями передачи данных…

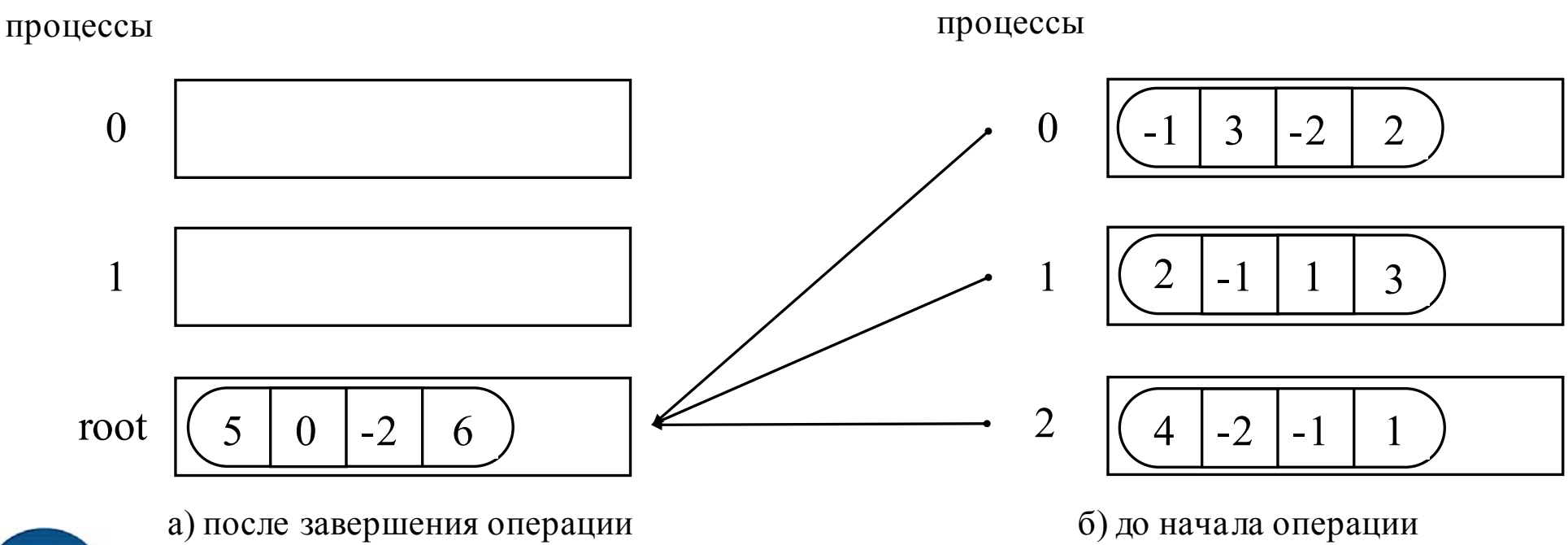
- Типы операций MPI для функций редукции данных…
  - **MPI_MAX** и **MPI_MIN** ищут поэлементные максимум и минимум;
  - **MPI_SUM** вычисляет поэлементную сумму векторов;
  - **MPI_PROD** вычисляет поэлементное произведение векторов;
  - **MPI_LAND**, **MPI_BAND**, **MPI_LOR**, **MPI_BOR**, **MPI_LXOR**, **MPI_BXOR** - логические и двоичные операции И, ИЛИ, исключающее ИЛИ;
  - **MPI_MAXLOC**, **MPI_MINLOC** - поиск индексированного минимума/максимума

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу…
  - Функция *MPI_Reduce* определяет коллективную операцию и, тем самым, вызов функции должен быть выполнен всеми процессами указываемого коммуникатора, все вызовы функции должны содержать одинаковые значения параметров *count*, *type*, *op*, *root*, *comm*,
  - Передача сообщений должна быть выполнена всеми процессами, результат операции будет получен только процессом с рангом *root*,
  - Выполнение операции редукции осуществляется над отдельными элементами передаваемых сообщений.

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу (пример для операции суммирования)



а) после завершения операции

б) до начала операции

Начальное знакомство с коллективными операциями передачи данных

- Синхронизация вычислений
  - *Синхронизация* процессов, т.е. одновременное достижение процессами тех или иных точек процесса вычислений, обеспечивается при помощи функции MPI:

```
int MPI_Barrier(MPI_Comm comm);
```

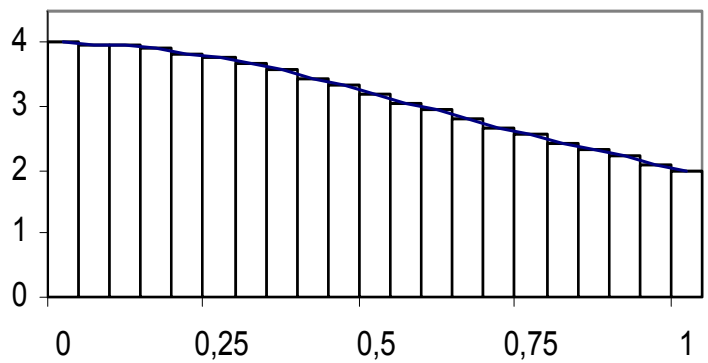
  - Функция *MPI_Barrier* определяет коллективную операцию, при использовании должна вызываться всеми процессами коммуникатора.
  - Продолжение вычислений любого процесса произойдет только после выполнения функции *MPI_Barrier* всеми процессами коммуникатора.

Пример: *Вычисление числа π...*

- Значение числа π может быть получено при помощи интеграла

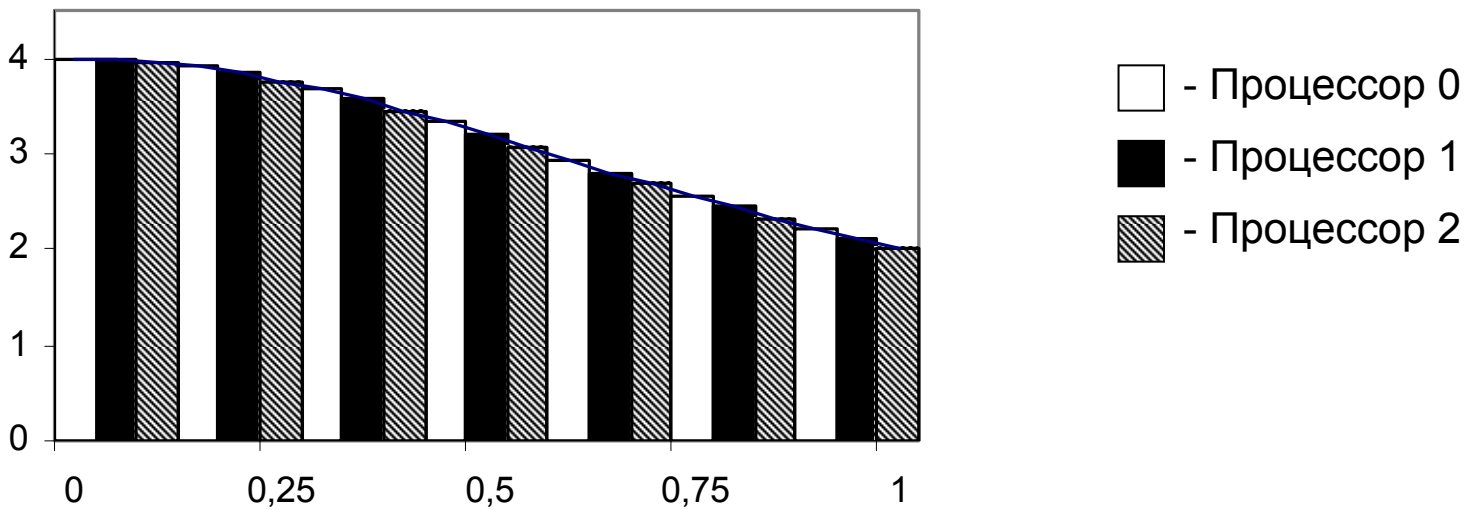
$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

- Для численного интегрирования применим метод прямоугольников



Пример: *Вычисление числа π...*

- Распределим вычисления между p процессорами (циклическая схема)
- Получаемые на отдельных процессорах частные суммы должны быть просуммированы



Пример: *Вычисление числа π...*

```
#include "mpi.h"
#include <math.h>
double f(double a) {
  return (4.0 / (1.0 + a*a));
}
int main(int argc, char *argv) {
  int ProcRank, ProcNum, done = 0, n = 0, i;
  double PI25DT = 3.14159265358979323846264 3;
  double mypi, pi, h, sum, x, t1, t2;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD,&ProcRank);
  while (!done ) { // основной цикл вычислений
    if ( ProcRank == 0) {
      printf("Enter the number of intervals: ");
      scanf("%d",&n);
      t1 = MPI_Wtime();
    }
```

Пример: *Вычисление числа* π

```
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  if (n > 0) { // вычисление локальных сумм
    h = 1.0 / (double) n;
    sum = 0.0;
    for (i = ProcRank + 1; i <= n; i += ProcNum) {
      x = h * ((double)i - 0.5);
      sum += f(x);
    }
    mypi = h * sum;
    // сложение локальных сумм (редукция)
   MPI_Reduce(&mypi,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
    if ( ProcRank == 0 ) { // вывод результатов
      t2 = MPI_Wtime();
      printf("pi is approximately %.16f, Error is
                  %.16f\n",pi, fabs(pi - PI25DT));
      printf("wall clock time = %f\n",t2-t1);
    }
  } else done = 1;
 }
 MPI_Finalize();
}
```

Заключение…

- В первой презентации раздела рассмотрены понятия и определения, являющиеся основополагающими для стандарта MPI (параллельная программа, операция передачи сообщения, тип данных, коммуникатор, виртуальная топология).
- Дано быстрое и простое введение в разработку параллельных программ с использованием MPI.
- Приведен пример параллельной программы с использованием MPI

Вопросы для обсуждения

- Сложность параллельных программ, разработанных с использованием MPI
- Проблема отладки параллельных программ

Темы заданий для самостоятельной работы

- Разработайте программу для нахождения минимального (максимального) значения среди элементов вектора.
- Разработайте программу для вычисления скалярного произведения двух векторов.
- Разработайте программу, в которой два процесса многократно обмениваются сообщениями длиной ***n*** байт. Выполните эксперименты и оцените зависимость времени выполнения операции данных от длины сообщения. Сравните с теоретическими оценками, построенными по модели Хокни.

Ссылки

- Информационный ресурс Интернет с описанием стандарта MPI: http://www.mpiforum.org
- Одна из наиболее распространенных реализаций MPI библиотека MPICH представлена на http://www-unix.mcs.anl.gov/mpi/mpich
- Библиотека MPICH2 с реализацией стандарта MPI-2 содержится на http://www-unix.mcs.anl.gov/mpi/mpich2
- Русскоязычные материалы о MPI имеются на сайте http://www.parallel.ru

Литература…

- **Воеводин** В.В., Воеводин Вл.В. (2002). Параллельные вычисления. – СПб.: БХВ-Петербург.
- **Корнеев** В.В. (2003) Параллельное программирование в MPI. Москва-Ижевск: Институт компьютерных исследований,2003
- **Немнюгин** С., Стесик О. (2002). Параллельное программирование для многопроцессорных вычислительных систем – СПб.: БХВ-Петербург.
- **Group**, W., Lusk, E., Skjellum, A. (1994). Using MPI. Portable Parallel Programming with the Message-Passing Interface. – MIT Press.
- **Group**, W., Lusk, E., Skjellum, A. (1999a). Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). - MIT Press.

Литература

- **Group**, W., Lusk, E., Thakur, R. (1999b). Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation). - MIT Press.
- **Pacheco,** P. (1996). Parallel Programming with MPI. - Morgan Kaufmann.
- **Quinn**, M. J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
- **Snir**, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. (1996). MPI: The Complete Reference. - MIT Press, Boston, 1996.

Авторский коллектив

Гергель В.П., профессор, д.т.н., руководитель

Гришагин В.А., доцент, к.ф.м.н.

Абросимова О.Н., ассистент (раздел 10)

Лабутин Д.Ю., ассистент (система ПараЛаб)

Курылев А.Л., ассистент (лабораторные работы 4, 5)

Сысоев А.В., ассистент (раздел 1)

Гергель А.В., аспирант (раздел 12, лабораторная работа 6)

Лабутина А.А., аспирант (разделы 7,8,9, лабораторные работы 1, 2, 3, система ПараЛаб)

Сенин А.В., аспирант (раздел 11, лабораторные работы по Microsoft Compute Cluster)

Ливерко С.В. (система ПараЛаб)

О проекте

Целью проекта является создание образовательного комплекса "Многопроцессорные вычислительные системы и параллельное программирование", обеспечивающий рассмотрение вопросов параллельных вычислений, предусматриваемых рекомендациями Computing Curricula 2001 Международных организаций IEEE-CS и ACM. Данный образовательный комплекс может быть использован для обучения на начальном этапе подготовки специалистов в области информатики, вычислительной техники и информационных технологий.

Образовательный комплекс включает **учебный курс "Введение в методы параллельного программирования" и лабораторный практикум "Методы и технологии разработки параллельных программ"**, что позволяет органично сочетать фундаментальное образование в области программирования и практическое обучение методам разработки масштабного программного обеспечения для решения сложных вычислительно-трудоемких задач на высокопроизводительных вычислительных системах.

Проект выполнялся в Нижегородском государственном университете им. Н.И. Лобачевского на кафедре математического обеспечения ЭВМ факультета вычислительной математики и кибернетики (http://www.software.unn.ac.ru). Выполнение проекта осуществлялось при поддержке компании Microsoft.