Межфакультетские учебные курсы МГУ, факультет ВМК, кафедра Суперкомпьютеров и квантовой информатики

Введение в суперкомпьютерные вычисления, квантовую информатику, нейросетевые и генетические алгоритмы

Лекция

Введение в параллельное программирование для суперкомпьютеров

Лектор: доцент **Попова Нина Николаевна**

ПЛАН

- Введение
- Архитектуры параллельных вычислительных систем
- Основы разработки параллельных программ
- Обзор технологий параллельного программирования OpenMP и MPI
- Пример параллельной программы: численное интегрирование

Что надо знать, приступая к параллельному программированию

- Базовые основы построения параллельных вычислительных систем.
- Понятия: программа -> процесс -> поток (легковесный поток)
- Один из языков программирования, для которых поддерживаются возможности организации параллельных вычисления:

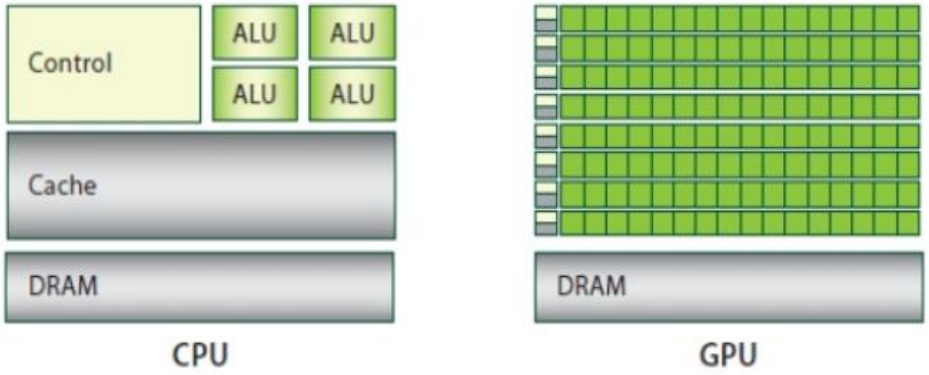
  - параллельный язык программирования (X10, …) – **В БУДУЩЕМ**

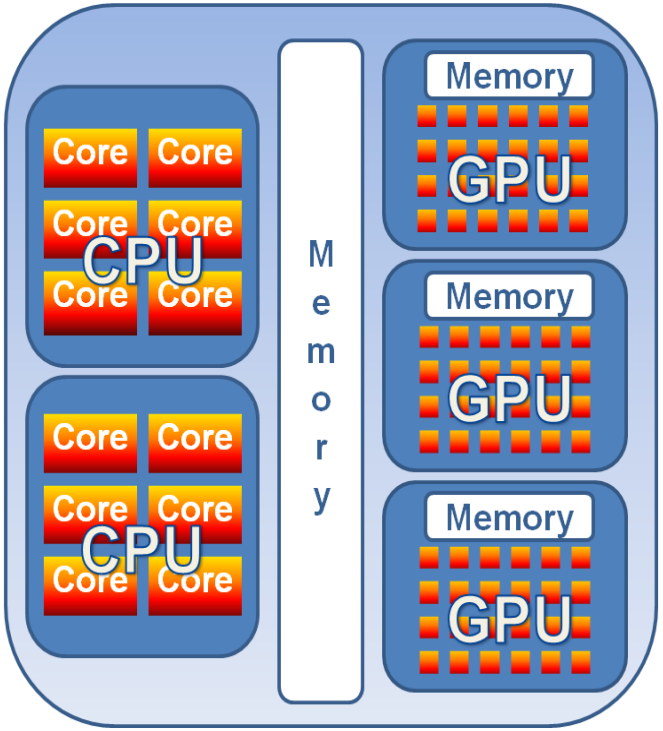
  - **C,** Fortran (базовые), JAVA, Python
- Основы разработки программ в рамках операционных системах Linux (лучше), Windows

Тенденции развития современных процессоров

• Реализация нескольких вычислительных ядер на одном чипе – многоядерные процессоры

• Добавление сопроцессоров – GPGPU, manycores





Базовые основы построения параллельных вычислительных систем

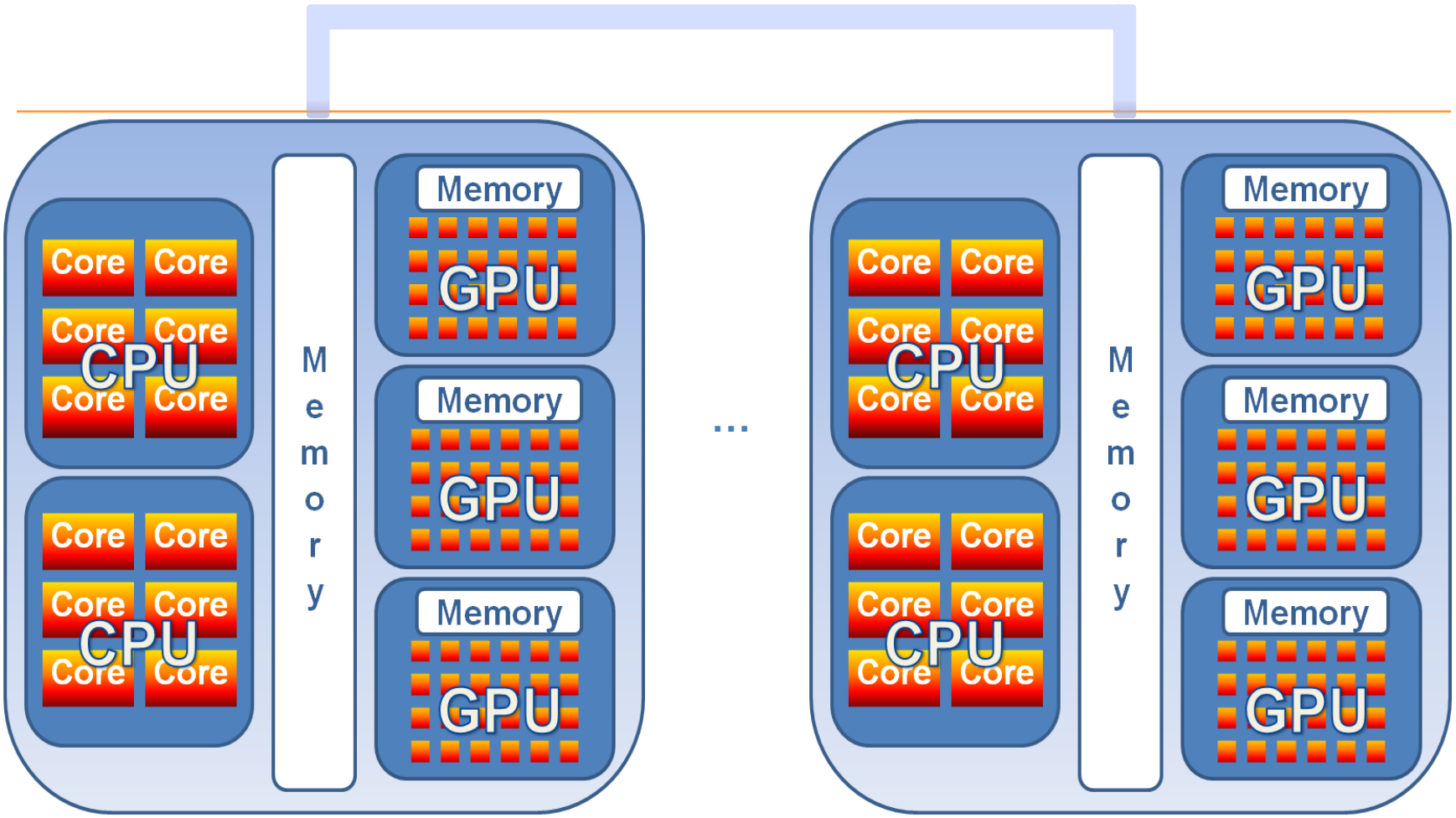
- 2 основных класса параллельных ВС:
  - **системы с общей памятью** (многоядерные процессоры, специальные SMP-процессоры)
  - **системы с распределенной памятью**

    вычислительные узлы =

    многоядерные процессоры,

    графические ускорители

Пример гибридной вычислительной системы



Список Top500.

Суперкомпьютер МГУ Ломоносов-2

Пиковая производительность - 2,96 Пфлопс
Производительность на тесте Linpack – 2,2 Пфлопс
Выч.узлы – 14-ядерные Xeon E5-26973v3
Ускорители - NVIDIA® Tesla™ K40
Количество узлов – 5x256= 1280
Число ядер - 45 688
Количество стоек – 5
Производительность 1 стойки – 535 Тфлопс
Оперативная память - 80 Тбайт
Коммуник.сети – 2 Ethernet , 2 InfiniBand

Суперкомпьютеры МГУ

Суперкомпьютер «Ломоносов»

(http://parallel.ru/cluster/lomonosov.html)

Основные технические характеристики:

- Пиковая производительность - **1,7 Пфлопс**
- Производительность на тесте Linpack - 901.9 Тфлопс
- Число процессоров/ядер - 12 346 / 52 168
- Число графических ядер - 954 240
- Оперативная память - 92 Тбайт

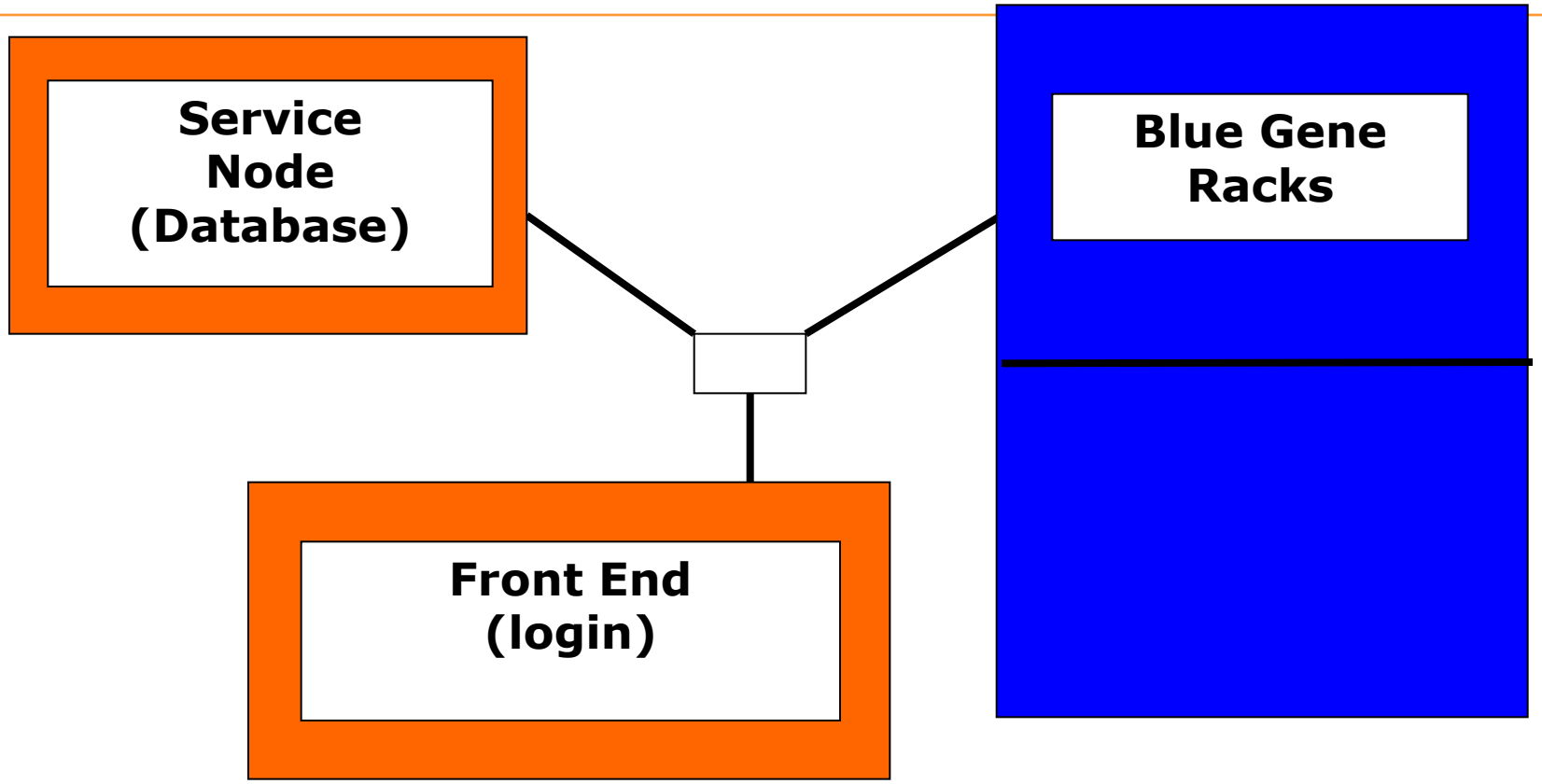
Суперкомпьютеры МГУ

IBM Blue Gene/P

hpc.cs.msu.ru

2048 4-ядерных процессоров
Пиковая производительность:
27.2 **TFlops**
Реальная производительность
по тесту **Linpack:**
23.2TFlops (85% от пиковой)
Общий объем ОЗУ: 4Тб

BlueGene/P



Компоненты Blue Gene/P

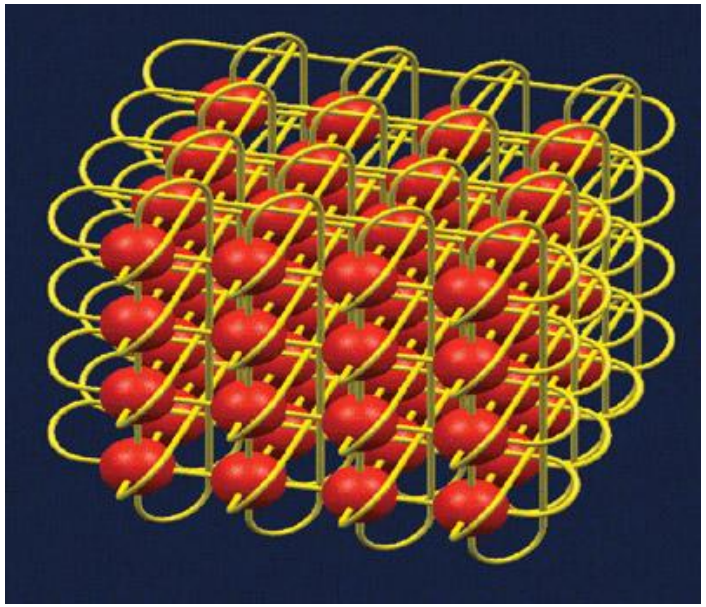


Рис. 1. Топология Blue Gene/P: 3х-мерный тор

Основы разработки параллельных программ

Модель параллельных вычислений

- ***Параллельный алгоритм*** – алгоритм, который может быть реализован по частям на множестве различных вычислительных устройств с последующим объединением полученных результатов и получением корректного результата (Википедия)

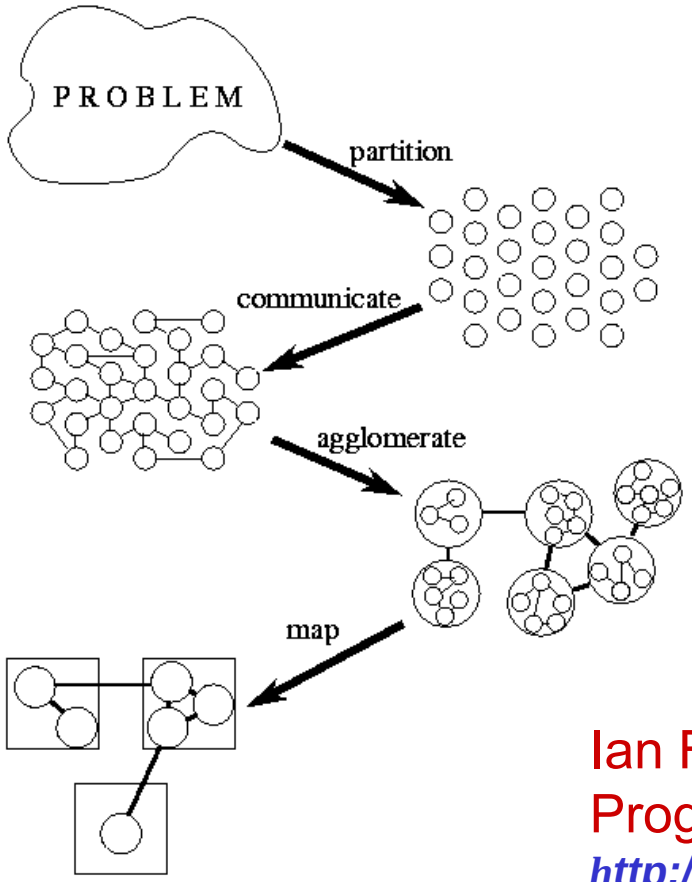
- ***Модель параллельных вычислений:***

- решение задачи разбивается на множество подзадач, которые выполняются одновременно
- обеспечивается координация между подзадачами

Параллельная программа

- ***Параллельная программа*** – программа, в которой явно определено параллельное выполнение всей программы либо ее фрагментов (блоков, операторов, инструкций). Программу, в которой параллелизм поддерживается ***неявно***, не будем относить к параллельным.
- ***Процесс*** – программа во время выполнения (интуитивно). Существует несколько более формальных определений Параллельная программа, как правило, выполняется в рамках ***нескольких процессов, ВЗАИМОДЕЙСТВУЩИХ!***
- ***Поток*** – легковесный процесс. В рамках одного процесса может существовать ***НЕСКОЛЬКО ПОТОКОВ*** – модель OpenMP- программ.

Этапы разработки параллельных программ



1. Декомпозиция
2. Проектирование коммуникаций
3. Укрупнение
4. Планирование вычислений

Ian Foster “Designing and Building Parallel Program”

http://www.mcs.anl.gov/~itf/dbpp/text/node4.html

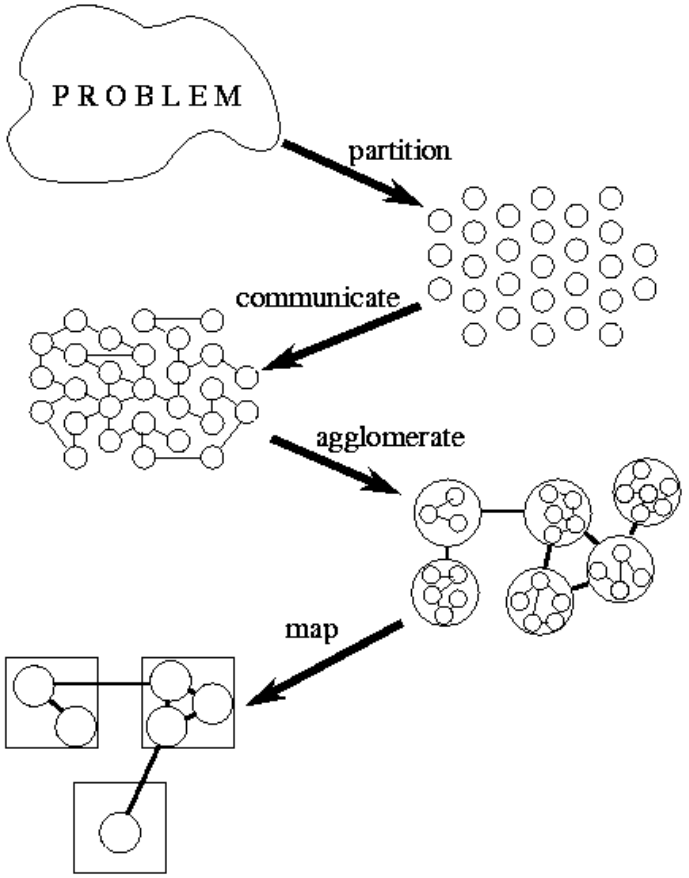
Этапы разработки параллельной программы. Пример.

- ***Проблема:****Параллельный Генетический Алгоритм.*
- ***Декомпозиция:*** *Островная модель.*

  Разделяем популяцию на подпопуляции.
- ***Коммуникации:***

  вводим оператор миграции -

  обмен индивидуумами между соседними островами
- ***Агрегация*** *(укрупнение):*
- ***Планирование вычислений****:*



Подходы к созданию параллельных программ

- Разработка новых параллельных языков.

  - Например, Erlang, X10

- Расширение существующих языков программирования специальными директивами.

  Например, HPF (data parallelism),

  OpenMP (shared memory + data parallelism), OpenACC

- Реализация параллелизма посредством вызова внешних библиотечных функций в существующих языках (например, передача сообщений в MPI)

- потоки (shared memory – Pthreads, Java threads)

- автоматическое распараллеливание компиляторами

- другие подходы. Например, объектно-ориентированный параллелизм

Модели параллельных программ

- **Системы с общей памятью**
  - Программирование, основанное на потоках
  - Программа строится на базе последовательной программы
  - Возможно автоматическое распараллеливание компилятором с использованием соответствующего ключа компилятора
  - Директивы компиляторов (OpenMP, OpenACC,…)

- **Системы с распределенной памятью**
  - Программа состоит из параллельных процессов
  - Явное задание коммуникаций между процессами – обмен сообщениями **“Message Passing”**

  Реализация - Message Passing библиотек:
    - MPI (“Message Passing Interface”)
    - PVM (“Parallel Virtual Machine”)
    - Shmem,
    - MPT (Cray)

MPI

- MPI реализует модель передачи сообщений:

  Вычисления – множество процессов, взаимодействующих друг с другом посредством передачи сообщений

- MPI – спецификация, не язык

- MPI - библиотека функций , вызываемых из языков программирования высокого уровня: C, Fortran

- Все производители параллельных компьютеров предлагают свою реализацию библиотеки MPI. Широко доступны свободно распространяемые реализации библиотек MPI (например, openmpi, lam-mpi,mpich)

OpenMP

- OpenMP : **Open** specifications for **M**ulti **P**rocessing

  – многопоточный интерфейс, специально разработанный для поддержки параллельных вычислений

  - не предназначен для систем с распределенной памятью, не поддерживает передачу сообщений

  - рекомендуется для достижения максимального параллелизма на системах с общей памятью

Пример параллельной программы программы (C, OpenMP)

Сумма элементов массива

```
#include <stdio.h>
#define N 1024
int main()
{ double sum;
  double a[N];
  int   i, n =N;
  for (i=0; i<n; i++){
    a[i] = i*0.5;  }
 sum =0;
#pragma omp for reduction (+:sum)
 for (i=0; i<n; i++)
  sum = sum+a[i];
printf (“Sum=%f\n”, sum);
}
```

Пример параллельной программы программы (C, MPI)

```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, all_sum;
  double a[N];
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
```

```
n= n/ size;
  for (i=rank*n;  i<n; i++){
    a[i] = i*0.5;  }

 sum =0;
for (i=0; i<n; i++)
  sum = sum+a[i];
MPI_Reduce(& sum,& all_sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf ("Sum =%f\n", all_sum);
MPI_Finalize();
return 0;
}
```

Гибрид: MPI+OpenMP

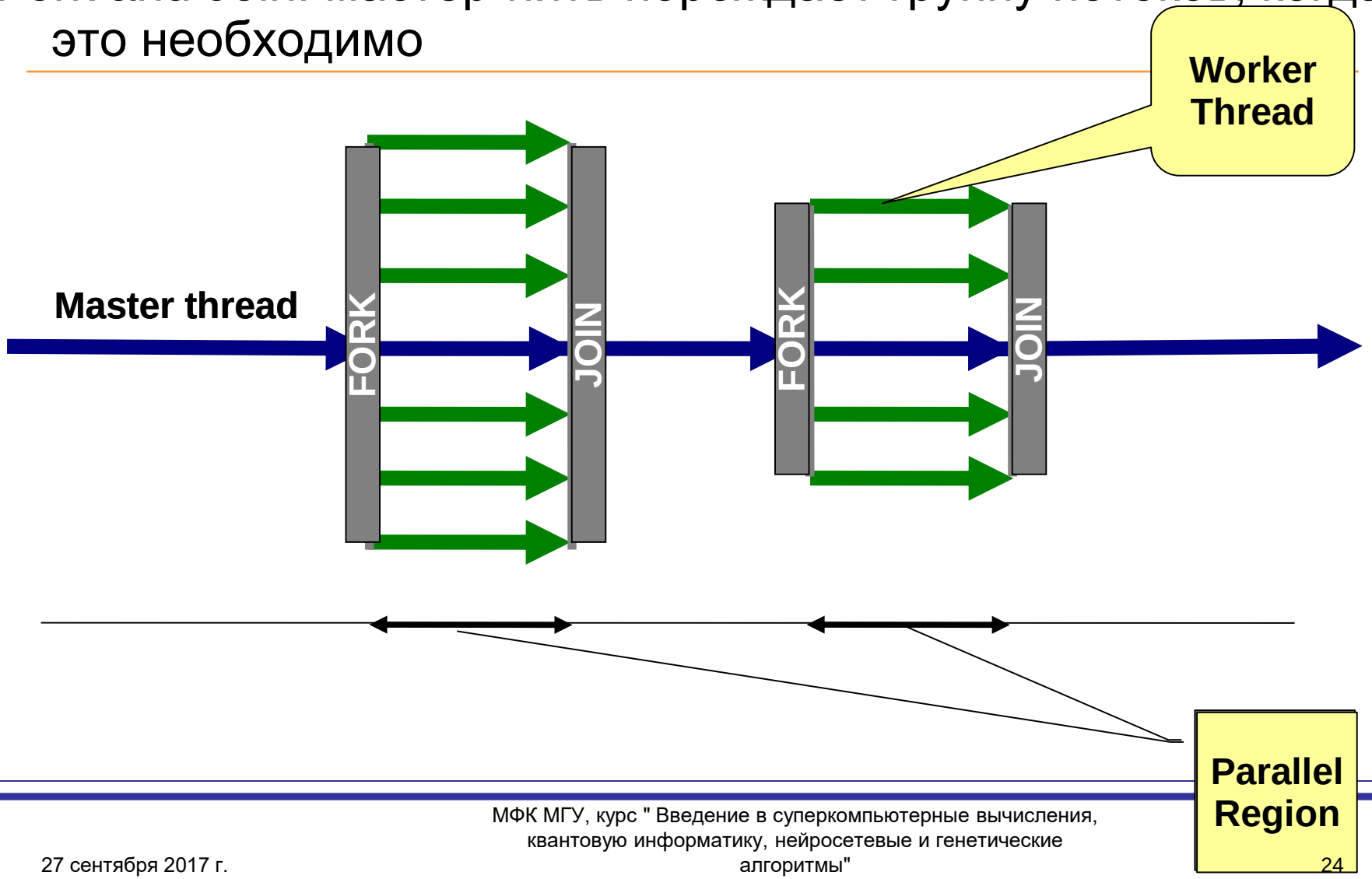
```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, all_sum;
  double a[N];
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
```

```
n= n/ size;
  for (i=rank*n;  i<n; i++){
    a[i] = i*0.5;  }

 sum =0;
#pragma omp for reduction (+:sum)
for (i=0; i<n; i++)
  sum = sum+a[i];
MPI_Reduce(& sum,& all_sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf ("Sum =%f\n", all_sum);
MPI_Finalize();
return 0;
}
```

OpenMP модель выполнения

Fork and Join: мастер-нить порождает группу потоков, когда это необходимо



OpenMP модель памяти

- Модель разделяемой памяти
  - Потоки взаимодействуют через общие (разделяемые) переменные
- Разделение определяется синтаксически
  - Любая переменная, видимая двумя и более потоками, является разделяемой (shared)
  - Любая переменная, видимая только одной нитью является приватной (private)
- Возможно возникновение условий гонок (Race conditions)
  - Используется синхронизация для предотвращения конфликтов

OpenMP синтаксис

- Большинство OpenMP конструкций - прагмы
  - #pragma omp construct [clause [clause] …]
  - OpenMP конструкции применяются к ***структурному блоку***
- Категории OpenMP конструкций
  - Parallel regions
  - Work sharing
  - Data Environment
  - Synchronization
  - Runtime functions/environment variables
- Кроме этого:
  - несколько omp_<something> вызовов функций
  - несколько OMP_<something> переменных окружения

Структурный блок

Действие директив распространяется на структурный блок:

#pragma omp название-директивы*[клауза[[,]клауза]...]*

{

структурный блок

}

Структурный блок: блок кода с одной точкой входа и одной точкой выхода.

```
#pragma omp parallel
{
  …
  mainloop: res[id] = f (id);
  if (res[id] != 0) goto mainloop;
  …
  exit (0);
}
```

Структурный блок

```
#pragma omp parallel
{
  …
  mainloop: res[id] = f (id);
  …
}
if (res[id] != 0) goto mainloop;
```

Не структурный блок

OpenMP: параллельные области

Выполняет один и тот же код несколько раз (равное количеству нитей)

• Сколько нитей?
omp_set_num_threads_num (n)
• D – shared переменная
i, sum - private

• Что означает выполнение одной и той же работы параллельно несколько раз?
Можно использовать **omp_thread_num()** для распределения работ между нитями

```
double D[1000];
#pragma omp parallel
{
  int i; double sum = 0;
  for (i=0; i<1000; i++) sum +=
     D[I];
  printf("Thread %d computes
     %f\n",
        omp_thread_num(),
     sum);
}
```

OpenMP: конструкции разделения работ (1)

```
answer1 = long_computation_1();
answer2 = long_computation_2();
if (answer1 != answer2) { … }
```

- Как распараллелить?
  - Это два независимых блока вычислений!

```
#pragma omp sections
{
  #pragma omp section
  answer1 = long_computation_1();
  #pragma omp section
  answer2 = long_computation_2();
}
if (answer1 != answer2) { … }
```

OpenMP: конструкции разделения работ (2)

Последовательный блок

```
for (int i=0; i<N; i++) { a[i]=b[i]+c[i]; }
```

(Полу) ручное распараллеливание

```
#pragma omp parallel
{
  int id = omp_get_thread_num();
  int Nthr = omp_get_num_threads();
  int istart = id*N/Nthr, iend = (id+1)*N/Nthr;
  for (int i=istart; i<iend; i++) { a[i]=b[i]+c[i]; }
}
```

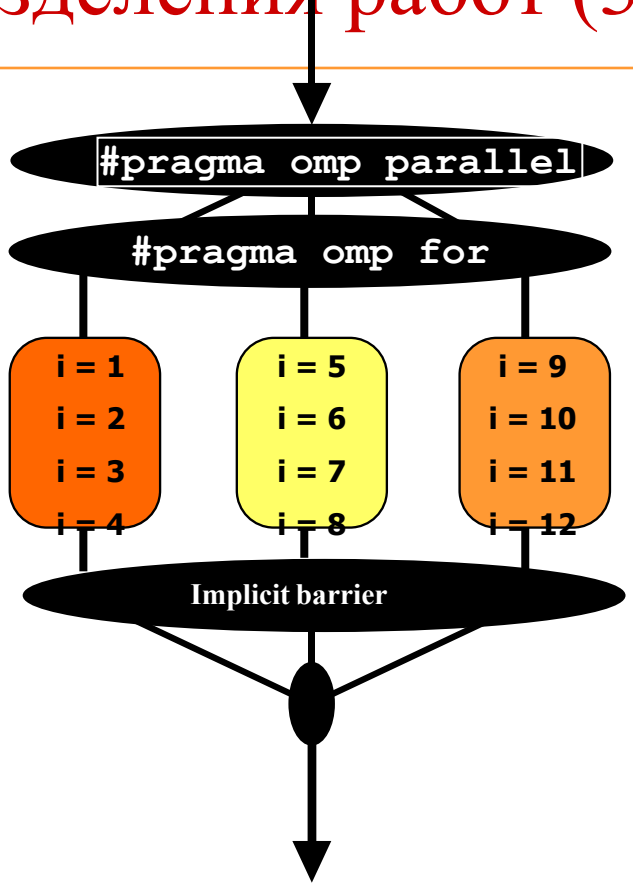
Автоматическое распараллеливание цикла

```
#pragma omp parallel
#pragma omp for schedule(static)
{
  for (int i=0; i<N; i++) { a[i]=b[i]+c[i]; }
}
```

OpenMP: конструкции разделения работ (3)

```
#pragma omp parallel
#pragma omp for
   for(i = 1; i < 13; i++)
      c[i] = a[i] + b[i];
```

- Каждой нити назначается определенное число итераций
- Нити должны ждать завершения всех итераций



#pragma omp reduction

- Синтаксис: **#pragma omp reduction (op:list)**
- Переменные в “*list*” должны быть shared
- Внутри parallel или конструкций разделения работ :
  - Создается PRIVATE копия каждой переменной из списка, которая инициализируется в зависимости от “op”
  - Эти копии локально обновляются нитью
  - При завершении над локальными копиями выполняется операция “op”

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for reduction(+:sum)
   for(int i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

Пример параллельной программы программы (C, OpenMP)

Сумма элементов массива

```
#include <stdio.h>
#define N 1024
int main()
{ double sum;
  double a[N];
  int   i, n =N;
  for (i=0; i<n; i++){
    a[i] = i*0.5;  }
 sum =0;
#pragma omp for reduction (+:sum)
 for (i=0; i<n; i++)
  sum = sum+a[i];
printf (“Sum=%f\n”, sum);
}
```

Компиляция OpenMP-программ

- **Важно**: компилятор должен понимать OpenMP-директивы
- Необходимо указать компилятору соответствующую опцию

Compiler/Platform	Compiler	Flag
Intel	icc icpc ifort	-openmp
GNU	gcc g++ g77 gfortran	-fopenmp

MPI стандарт для построения параллельных программ для вычислительных систем с распределенной памятью

Реализации MPI - библиотеки

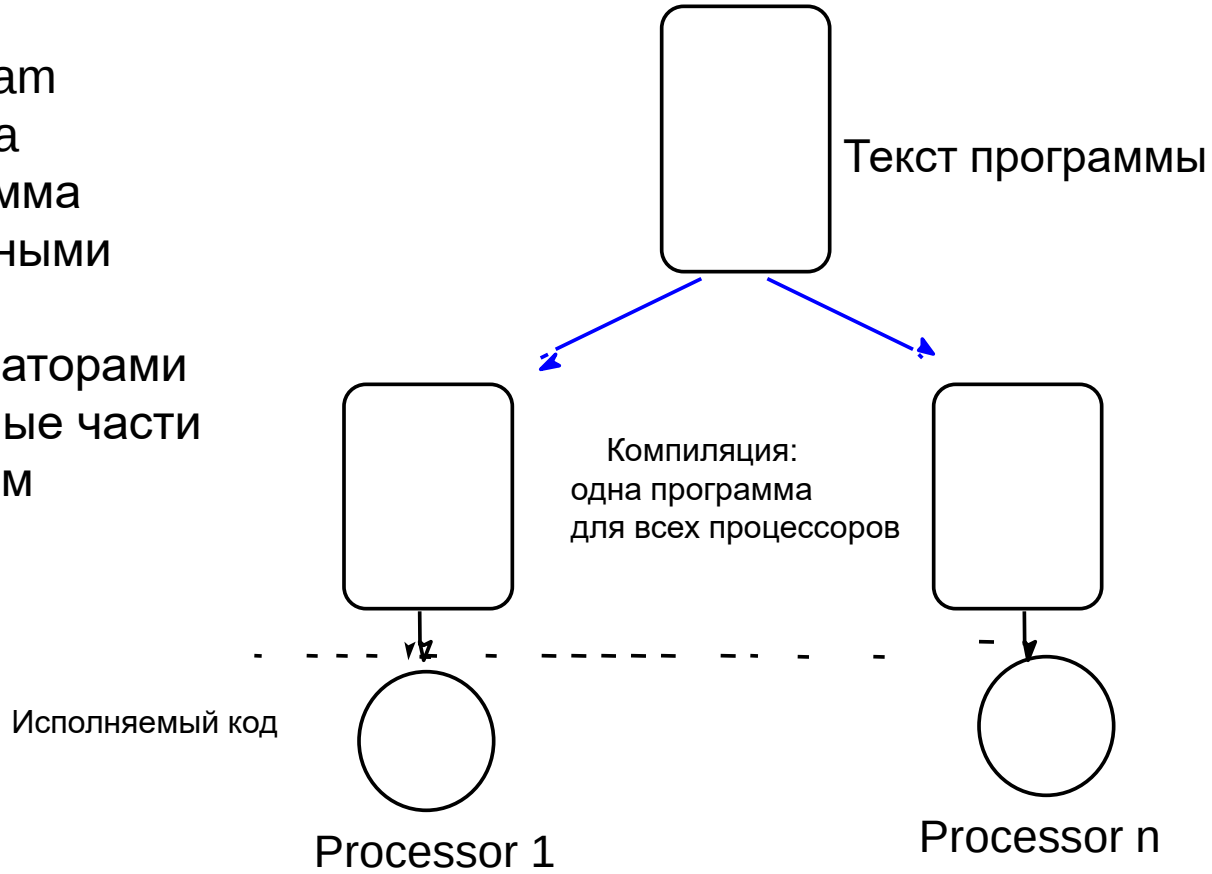
- MPICH
- LAM/MPI
- Mvapich
- OpenMPI
- Коммерческие реализации Intel,IBM и др.

Модель MPI программ

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

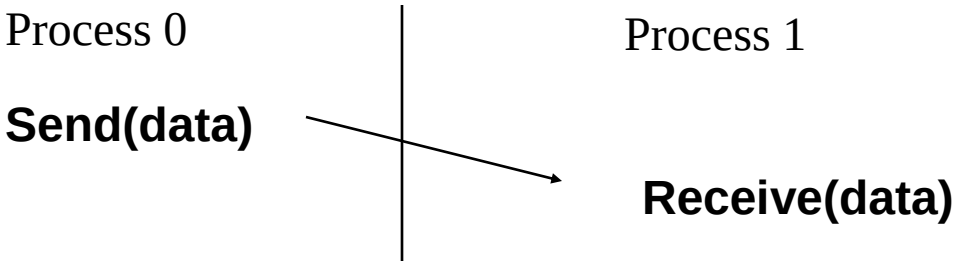
Модель MPI-программ

- SPMD – Single Program
  Multiple Data
- Одна и та же программа выполняется различными процессорами
- Управляющими операторами выбираются различные части программы на каждом процессоре.



Основы передачи данных в MPI

- Данные посылаются одним процессом и принимаются другим.
- Передача и синхронизация совмещены.



6 основных функций MPI

- **Как стартовать/завершить параллельное выполнение**
  - MPI_Init
  - MPI_Finalize
- **Кто я (и другие процессы), сколько нас**
  - MPI_Comm_rank
  - MPI_Comm_size
- **Как передать сообщение коллеге (другому процессу)**
  - MPI_Send
  - MPI_Recv

Основные понятия MPI

- Процессы объединяются в ***группы***.
- Группе приписывается ряд свойств (как связаны друг с другом и некоторые другие). Получаем ***коммуникаторы***
- Процесс идентифицируется своим номером в группе, привязанной к конкретному коммуникатору.
- При запуске параллельной программы создается специальный коммуникатор с именем ***MPI_COMM_WORLD***
- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.

Понятие коммуникатора MPI

- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.
- Наиболее часто используемый коммуникатор `MPI_COMM_WORLD`
  - определяется при вызове `MPI_Init`
  - содержит ВСЕ процессы программы

Типы данных MPI

- Данные в сообщении описываются тройкой: (***address, count, datatype***)
- ***datatype*** (типы да5ных MPI)

Signed
- MPI_CHAR
- MPI_SHORT
- MPI_INT
- MPI_LONG

Unsigned
- MPI_UNSIGNED_CHAR
- MPI_UNSIGNED_SHORT
- MPI_UNSIGNED
- MPI_UNSIGNED_LONG

- MPI_FLOAT
- MPI_DOUBLE
- MPI_LONG_DOUBLE

Тэг собщения

- Сообщение сопровождается определяемым пользователем признаком – целым числом – ***тэгом*** для идентификации принимаемого сообщения
- Теги сообщений у отправителя и получателя должны быть согласованы. Можно указать в качестве значения тэга константу **`MPI_ANY_TAG`**.
- Некоторые не-MPI системы передачи сообщений называют тэг типом сообщения. MPI вводит понятие тэга, чтобы не путать это понятие с типом данных MPI.

Формат MPI-функций

C :

```
error = MPI_Xxxxx(parameter,...);
MPI_Xxxxx(parameter,...);
```

C++ :

```
error = MPI::Xxxxx(parameter,...);
MPI::Xxxxx(parameter,...);
```

Fortran:

```
call MPI_XXXXX(parameter,...,IERR);
```

Основные группы функций MPI

- Определение среды
- Передачи «точка-точка»
- Коллективные операции
- Производные типы данных
- Группы процессов
- Виртуальные топологии
- Односторонние передачи
- Параллельный ввод-вывод
- Динамическое создание процессов
- Средства профилирования

Hello, MPI world! (2)

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
   int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, MPI world! I am %d of %d\n“,rank,size);
    MPI_Finalize();
    return 0; }
```

Трансляция MPI-программ

- Трансляция

  mpicc –o <имя_программы> <имя>.c <опции>

  Например:

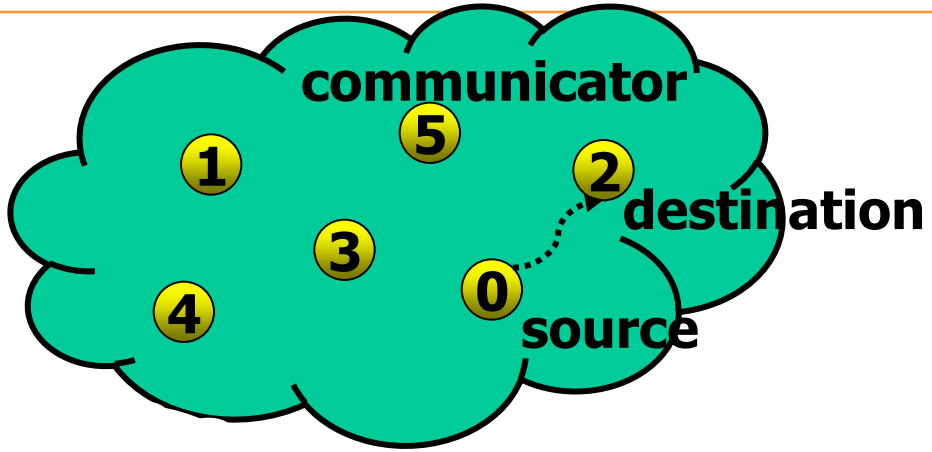
  mpicc –o hw helloworld.c
- Запуск в интерактивном режиме

  mpirun –np 128 hw

Взаимодействие «точка-точка»

- Самая простая форма обмена сообщением
- Один процесс посылает сообщения другому
- Несколько вариантов реализации того, как пересылка и выполнение программы совмещаются

Передача сообщений типа «точка-точка»



- Взаимодействие между двумя процессами
- Процесс-отправитель(Source process) ***посылает*** сообщение процессу-получателю (Destination process)
- Процесс-получатель ***принимает*** сообщение
- Передача сообщения происходит в рамках заданного коммуникатора
- Процесс-получатель определяется рангом в коммуникаторе

MPI_Send

*int MPI_Send(void *buf,int count, MPI_Datatype datatype,int dest,*
int tag, MPI_Comm comm)

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
dest	-	rank процесса-получателя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор

Пример:

```
MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)
```

MPI_Recv

*int MPI_Recv(void *buf,int count, MPI_Datatype datatype,int source, int tag, MPI_Comm comm,* ***MPI_Status *status****)*

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
source	-	rank процесса-отправителя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор,
status	-	статус

Пример:

```
MPI_Recv(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD,&stat)
```

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Send.html

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются ВСЕМИ процессами в коммуникаторе
- Примеры:
  - Broadcast, scatter, gather (рассылка данных)
  - Global sum, global maximum, и т.д. (Коллективные операции)
  - Барьерная синхронизация

Барьерная синхронизация

- Приостановка процессов до выхода ВСЕХ процессов коммуникатора в заданную точку синхронизации

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

Пример – упорядоченный вывод:

```
int size,rank;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
for (i=0;i<size;i++)
{ MPI_Barrier(MPI_COMM_WORLD);
if (rank==i) printf("%d\n", rank);
}
```

Широковещательная рассылка

- One-to-all передача: один и тот же буфер отсылается от процесса root всем остальным процессам в коммуникаторе

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

Глобальные операции редукции

- Операции выполняются над данными, распределенными по процессам коммуникатора
- Примеры:
  - Глобальная сумма или произведение
  - Глобальный максимум (минимум)
  - Глобальная операция, определенная пользователем

Общая форма

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

- **count** число операций “*op*” выполняемых над последовательными элементами буфера **sendbuf**
- (также размер **recvbuf**)

- **op** является ассоциативной операцией, которая выполняется над парой операндов типа **datatype** и возвращает результат того же типа

Предопределенные операции редукции

MPI Name	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

Пример параллельной программы программы (C, MPI)

```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, all_sum;
  double a[N];
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
```

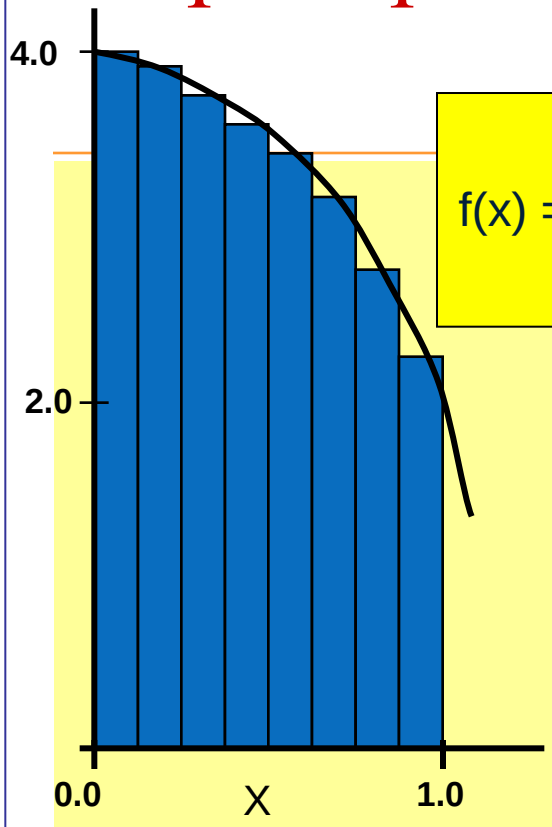
```
n= n/ size;
  for (i=rank*n;  i<n; i++){
    a[i] = i*0.5;  }

 sum =0;
for (i=0; i<n; i++)
  sum = sum+a[i];
MPI_Reduce(& sum,& all_sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf ("Sum =%f\n", all_sum);
MPI_Finalize();
return 0;
}
```

Литература

- *Антонов А. С.* Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. — Издательство Московского университета М.:, 2012. — С. 344.
- Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем систем.- СПб, БХВ-Петербург
- Интернет ресурсы:
- www.parallel.ru, intuit.ru, www.mpi-forum.org, http://www.mcs.anl.gov/research/projects/mpi/www/www3

Пример: численное интегрирование



$$f(x) = \frac{4.0}{(1+x^2)}$$

```
static long num_steps=100000;
 double step, pi;

void main()
{   int i;
    double x, sum = 0.0;

    step = 1.0/(double) num_steps;
    for (i=0; i< num_steps; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0 + x*x);
    }
    pi = step * sum;
    printf("Pi = %f\n",pi);
}
```

Вычисление числа π с использованием MPI

```
#include "mpi.h"
#include <stdio.h>
int main (int argc, char *argv[])
{
    int n =100000, myid, numprocs, i;
    double mypi, pi, h, sum, x;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    h   = 1.0 / (double) n;
    sum = 0.0;
```

Вычисление числа π с использованием MPI

```
  for (i = myid + 1; i <= n; i += numprocs)
    {
                x = h * ((double)i - 0.5);
                sum += (4.0 / (1.0 + x*x));
     }
     mypi = h * sum;
     MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0, MPI_COMM_WORLD);
     if (myid == 0) printf("pi is approximately
%.16f”, pi);
     MPI_Finalize();
     return 0;
}
```

Вычисление числа π с использованием OpenMP

```
#include <stdio.h>
int main (){
    int n =100000, i;
    double pi, h, sum, x;
    h   = 1.0 / (double) n;
    sum = 0.0;
#pragma omp parallel for reduction(+:sum) private(x)
    for (i = 1; i <= n; i ++) {
                x = h * ((double)i - 0.5);
                 sum += (4.0 / (1.0 + x*x));
     }
     pi = h * sum;
     printf("pi is approximately %.16f", pi);
     return 0;
}
```