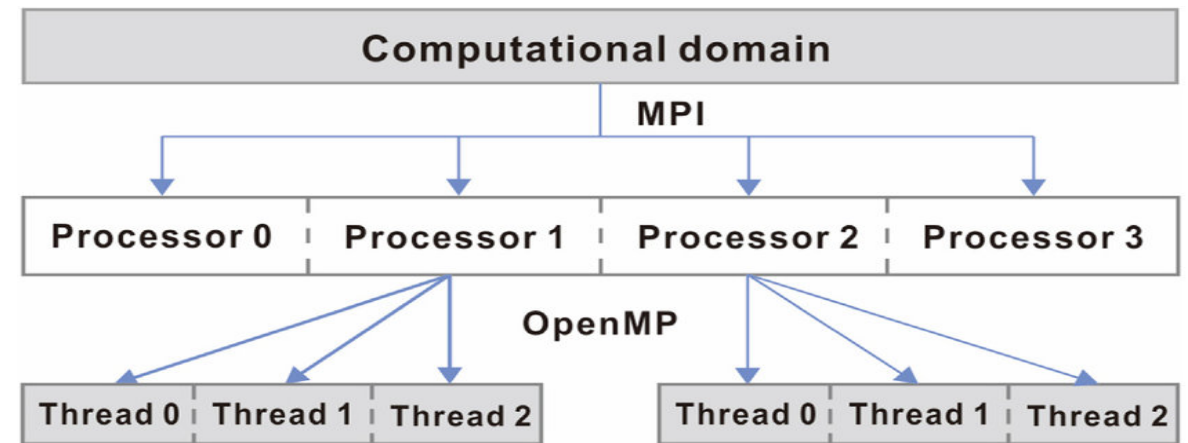
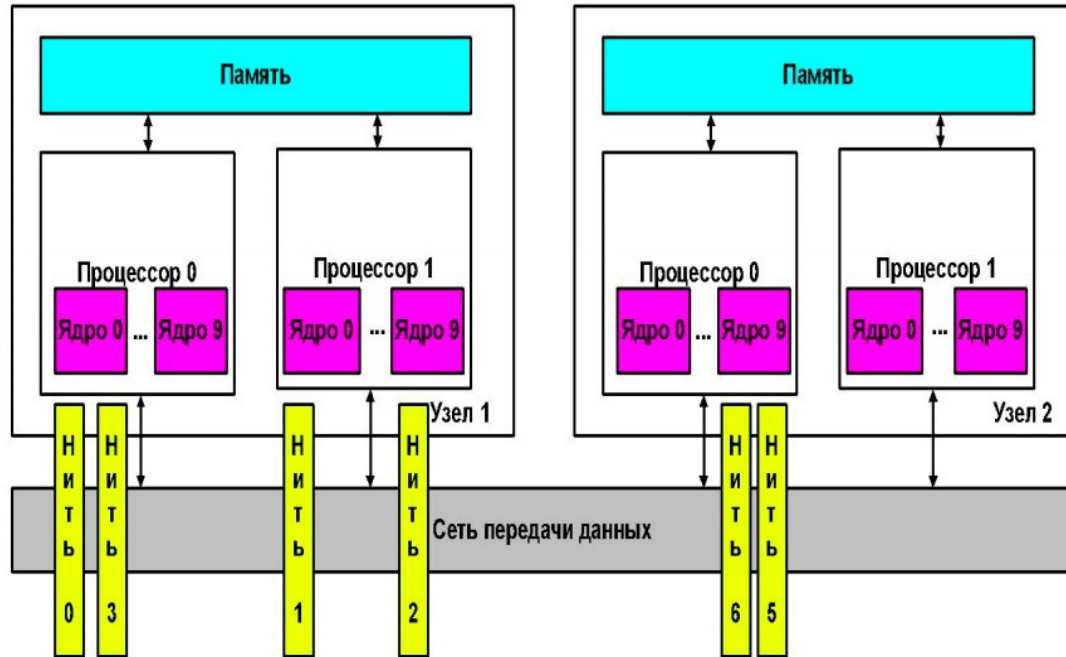


MPI

MPI

- **Message Passing Interface** (MPI, интерфейс передачи сообщений) — программный интерфейс для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу.
- Процессы обладают распределенной памятью.
- Модель SPMP - одна программа для множества процессов.

Структура работы MPI



Установка и запуск

- MPICH - одна из реализаций интерфейса MPI.
- Компилятор MPIC++
- Запуск: `mpirun -np 4 ./program`

Инициализация MPI

- `MPI_Init(&argc, &argv);`
- `MPI_Finalize();`
- `MPI_Comm_size(MPI_COMM_WORLD, &size);`
- `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`

MPI_Send

- `MPI_Send(`
- `void* data`, данные
- `int count`, размер данных
- `MPI_Datatype datatype`, тип посылаемых данных
- `int destination`, ранг процесса получателя
- `int tag`, тэг сообщения
- `MPI_Comm communicator`, Коммуникатор, в котором отправляется сообщение (`MPI_COMM_WORLD`)

MPI_Recv

- MPI_Recv(
• void* data, данные
• int count, размер данных
• MPI_Datatype datatype, тип посылаемых данных
• int source, ранг процессса отправителя
• int tag, тэг
• MPI_Comm communicator, коммуникатор
• MPI_Status* status, статус операции)

MPI_Status

- `typedef struct _MPI_Status {`
- `int count;`
- `int cancelled;`
- `int MPI_SOURCE;` Откуда пришло сообщение
- `int MPI_TAG;` Тэг
- `int MPI_ERROR;` Код ошибки `}`

`/* The following two fields (count, cancelled) are internal to the Open MPI implementation and should not be accessed by MPI applications. They are subject to change at any time. These are not the droids you're looking for.
*/`

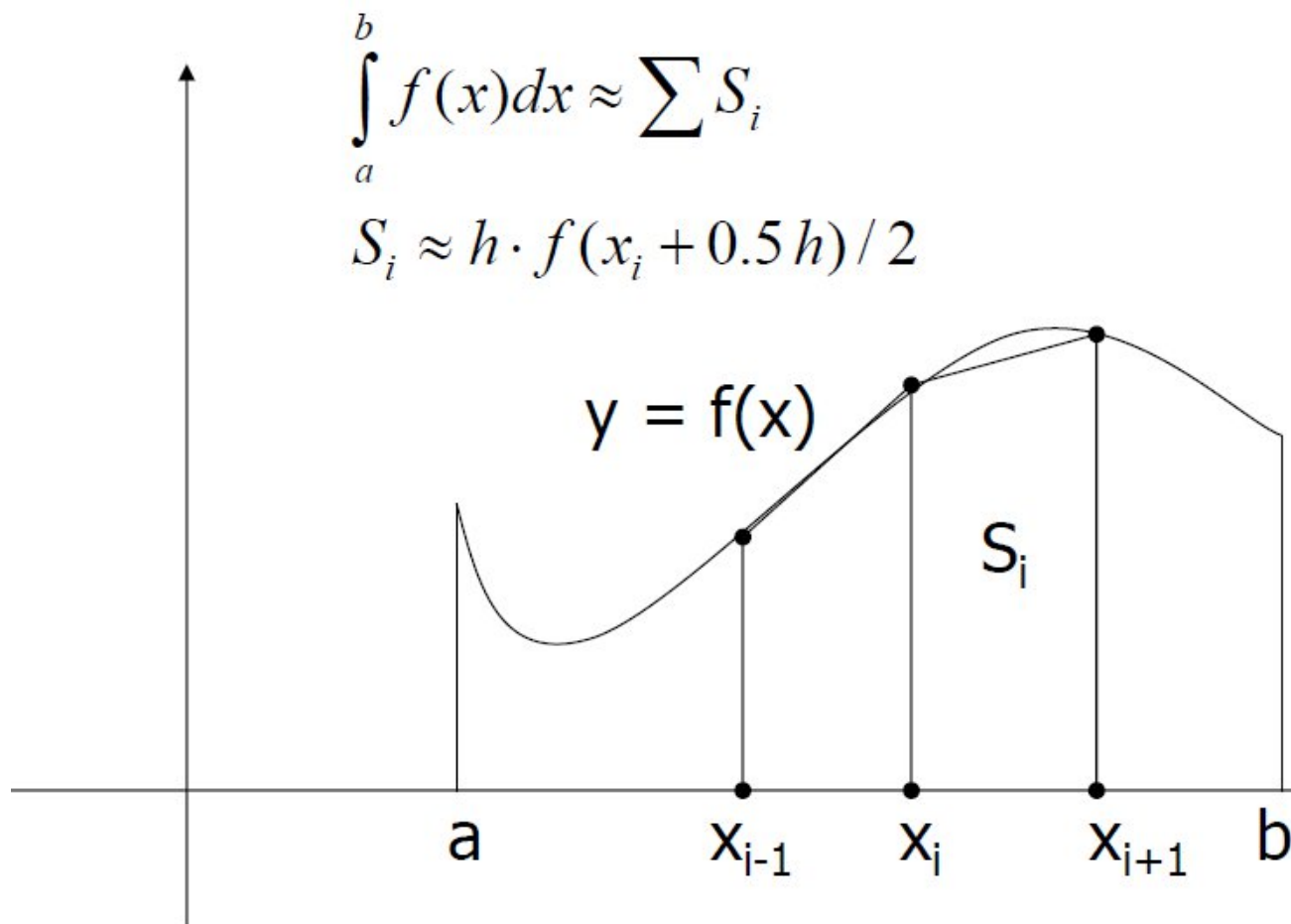
MPI_Probe

- MPI_Probe(
• int source,
• int tag,
• MPI_Comm communicator,
• MPI_Status* status)

MPI_Get_count

- MPI_Get_count(
• const MPI_Status *status, Возвращает статус операции
• MPI_Datatype datatype, Тип данных ожидаемых элементов
• int *count)

Задача на интеграл



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

Джокеры

- MPI_ANY_SOURCE
- MPI_ANY_TAG

MPI_Sendrecv

- Можно одновременно отправлять и получать сообщения. MPI гарантирует отсутствие deadlock.
- 12 аргументов, первые 5 как в MPI_Send, остальные 7 как в MPI_Recv
- MPI_Sendrecv_replace - использует общий буфер для отправки и получения. Принимаемые данные должны быть заведомо НЕ ДЛИННЕЕ отправляемых.

Упаковка/Распаковка структур

```
//Передача
//=====
int bufPos = 0;
char tempBuf[ sizeof(s) ];
MPI_Pack(&s.i, incout: 1, MPI_INT, tempBuf, sizeof(tempBuf), &bufPos, MPI_COMM_WORLD );
MPI_Pack( s.f, incout: 4, MPI_FLOAT, tempBuf, sizeof(tempBuf), &bufPos, MPI_COMM_WORLD );
MPI_Pack( s.c, incout: 8, MPI_CHAR, tempBuf, sizeof(tempBuf), &bufPos, MPI_COMM_WORLD );
MPI_Send( tempBuf, bufPos, MPI_BYTE, targetRank, msgTag, MPI_COMM_WORLD );

//Прием
//=====
int bufPos = 0;
char tempBuf[ sizeof(s) ];
MPI_Recv( tempBuf, sizeof(tempBuf), MPI_BYTE, sourceRank, msgTag,
          MPI_COMM_WORLD, &status );
MPI_Unpack( tempBuf, sizeof(tempBuf), &bufPos, &s.i, outcount: 1, MPI_INT, MPI_COMM_WORLD );
MPI_Unpack( tempBuf, sizeof(tempBuf), &bufPos, s.f, outcount: 4, MPI_FLOAT, MPI_COMM_WORLD );
MPI_Unpack( tempBuf, sizeof(tempBuf), &bufPos, s.c, outcount: 8, MPI_CHAR, MPI_COMM_WORLD );
```

```
#define msgTag 10
struct {
    int    i;
    float  f[4];
    char   c[8];
} s;
```

```
//Чит
Передача: MPI_Send( &s, sizeof(s), MPI_BYTE, ... );
Прием:    MPI_Recv( &s, sizeof(s), MPI_BYTE, ... );
```

Введение нового типа данных

- `MPI_Type_struct` - формирование нового типа-структуры
- `MPI_Type_commit` - сохранение типа в MPI
- `MPI_Type_free` - очистка данных о новом типе

```
#include <cstdlib> /* подключаем макрос 'offsetof()' */

typedef struct {
    int    i;
    double d[3];
    long   l[8];
    char   c;
} AnyStruct;

AnyStruct st;

MPI_Datatype anyStructType;

int          len[5] = { 1, 3, 8, 1, 1 };
MPI_Aint     pos[5] = { offsetof(AnyStruct,i), offsetof(AnyStruct,d),
                        offsetof(AnyStruct,l), offsetof(AnyStruct,c),
                        sizeof(AnyStruct) };
MPI_Datatype typ[5] = { MPI_INT, MPI_DOUBLE, MPI_LONG, MPI_CHAR, MPI_UB };

MPI_Type_struct( count: 5, len, pos, typ, &anyStructType );
MPI_Type_commit( &anyStructType );
/* подготовка закончена */

MPI_Send( st, 1, anyStructType, ... );
```

MPI_Isend

- int MPI_Isend(
 - const void *buf,
 - int count,
 - MPI_Datatype datatype,
 - int dest,
 - int tag,
 - MPI_Comm comm,
 - MPI_Request * request - Возвращаемый параметр для отслеживания завершения операции)

MPI_Irecv

- int MPI_Irecv(
• void *buf,
• int count,
• MPI_Datatype datatype,
• int source,
• int tag,
• MPI_Comm comm,
• MPI_Request * request)

MPI_Wait

- `int MPI_Wait(MPI_Request *request, MPI_Status *status)` - ждет завершения одного реквеста
- `int MPI_Waitall(int count, MPI_Request array_of_requests[], MPI_Status array_of_statuses[])` - ждет завершения всех реквестов в массиве
- `int MPI_Waitany(int count, MPI_Request array_of_requests[], int *indx, MPI_Status * status)` - ждет завершения одного из реквестов в массиве

MPI_Test

- `int MPI_Test(`
 - `MPI_Request *request,`
 - `int *flag`, `flag == true` если реквест завершен
 - `MPI_Status *status)`
-
- Есть такие же вариации как и для `MPI_Wait`

Режимы отправки сообщений

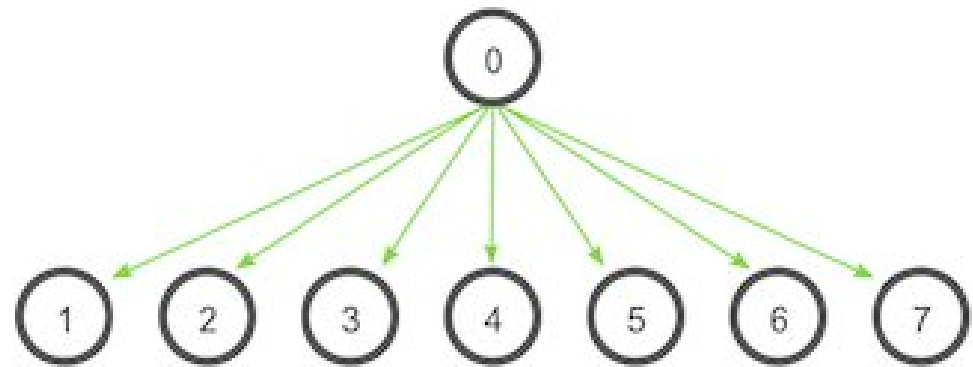
- **Bsend (buffered)** - Локальная операция, если операция приема не начата - отправляет сообщение в буффер. Контроль буффера отдается пользователю. Осуществляется при помощи MPI_Buffer_attach и MPI_Buffer_detach.
- **Ssend (synchronous)** - Отправка ждет пока другим процессом будет начат процесс получения сообщения.
- **Rsend (ready)** - Отправка требует начала операции приема другим процессом.

MPI_Barrier

- `int MPI_Barrier(MPI_Comm comm)`
- Все процессы в коммутаторе синхронизируются на этой команде

MPI_Bcast

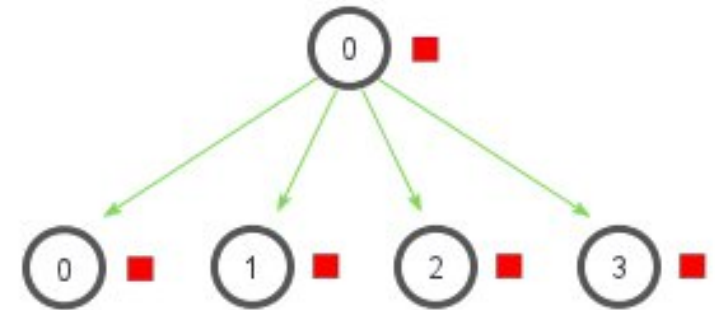
- MPI_Bcast(
• void* data,
• int count,
• MPI_Datatype datatype,
• int root,
• MPI_Comm communicator)



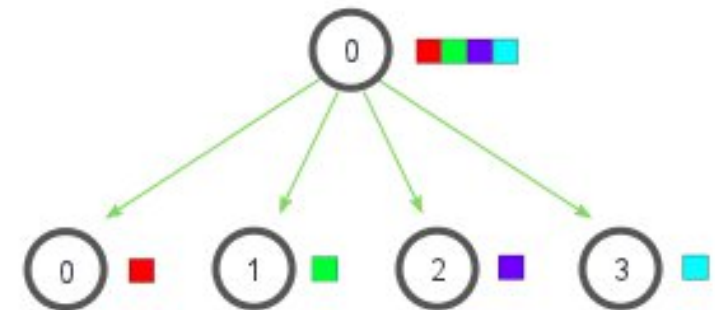
MPI_Scatter

- MPI_Scatter(
 - void* send_data,
 - int send_count,
 - MPI_Datatype send_datatype,
 - void* recv_data,
 - int recv_count,
 - MPI_Datatype recv_datatype,
 - int root,
 - MPI_Comm communicator)

MPI_Bcast

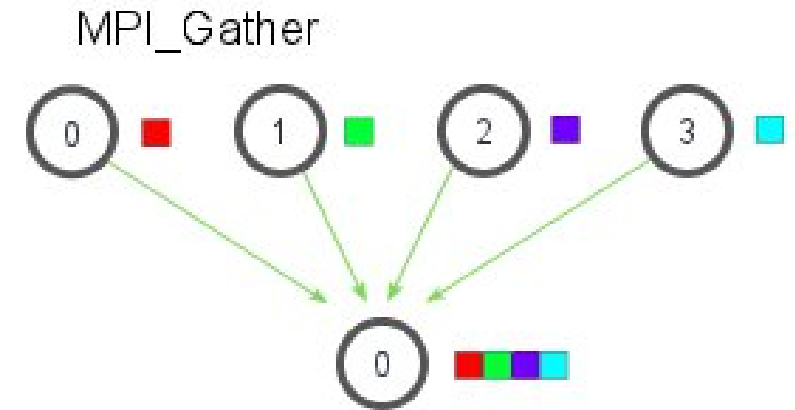


MPI_Scatter



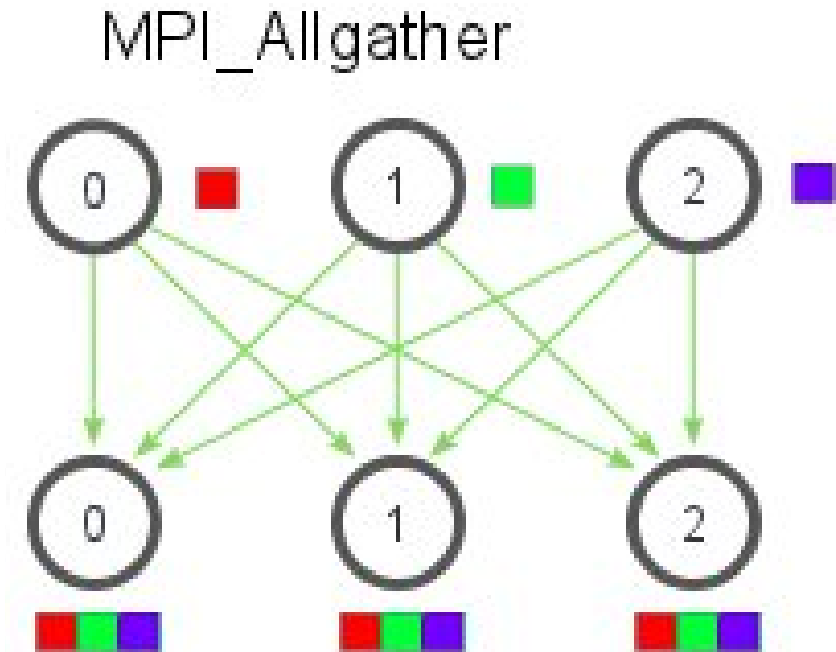
MPI_Gather

- MPI_Gather(
• void* send_data,
• int send_count,
• MPI_Datatype send_datatype,
• void* recv_data,
• int recv_count,
• MPI_Datatype recv_datatype,
• int root,
• MPI_Comm communicator)



MPI_Allgather

- MPI_Allgather(
• void* send_data,
• int send_count,
• MPI_Datatype send_datatype,
• void* recv_data,
• int recv_count,
• MPI_Datatype recv_datatype,
• MPI_Comm communicator)



Игра жизнь

- Поле N на N клеток
- Клетка может быть живой или мертвой
- В каждый следующий момент времени:
 - Если у клетки 3 соседа - в ней появляется жизнь
 - Если у клетки один сосед или меньше - она умирает
 - Если у клетки больше трех соседей - она умирает

