Уральский федеральный университет

Параллельные вычисления

Технология MPI

Созыкин Андрей Владимирович

к.т.н.
Заведующий кафедрой высокопроизводительных компьютерных технологий
Институт математики и компьютерных наук

MPI

Message Passing Interface (MPI) – наиболее популярная сейчас технология параллельного программирования для систем с распределенной памятью

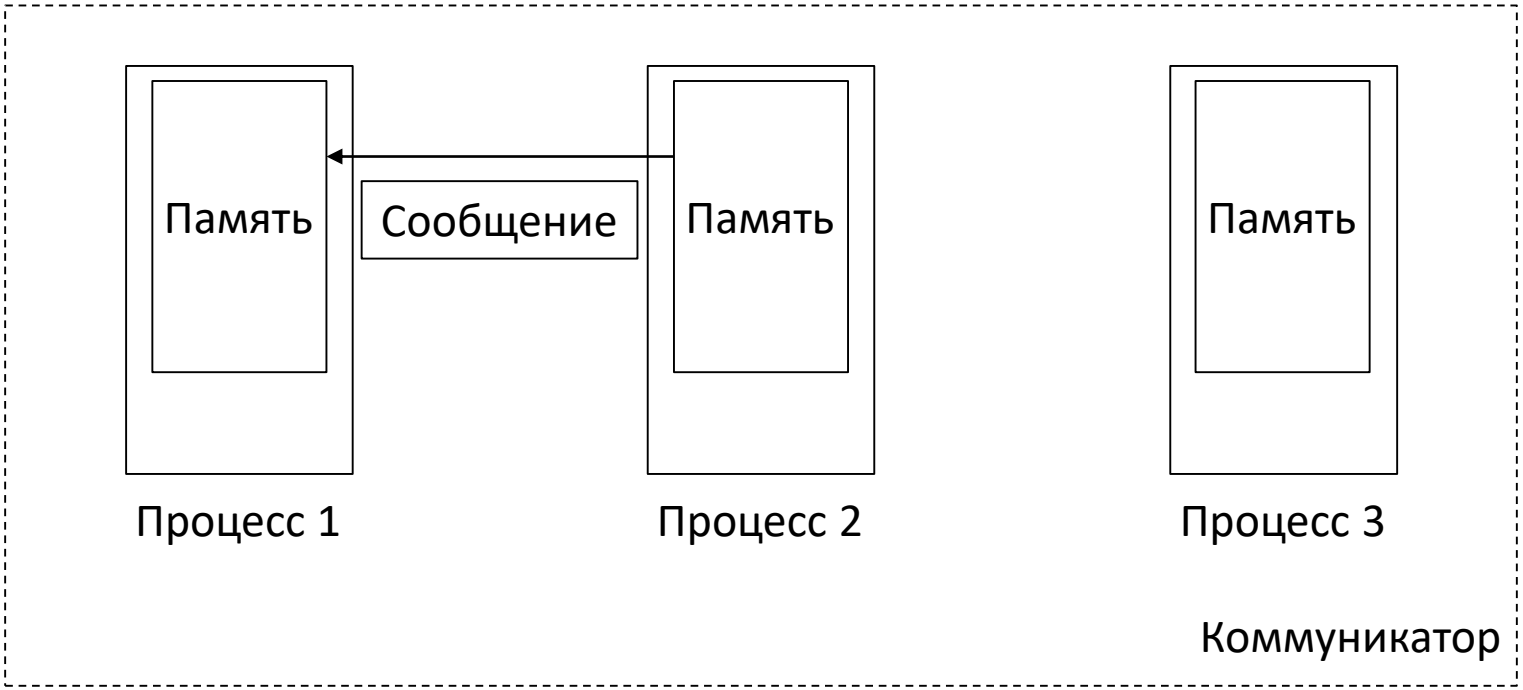
Использует модель передачи сообщение для взаимодействия между процессами

Альтернативы:

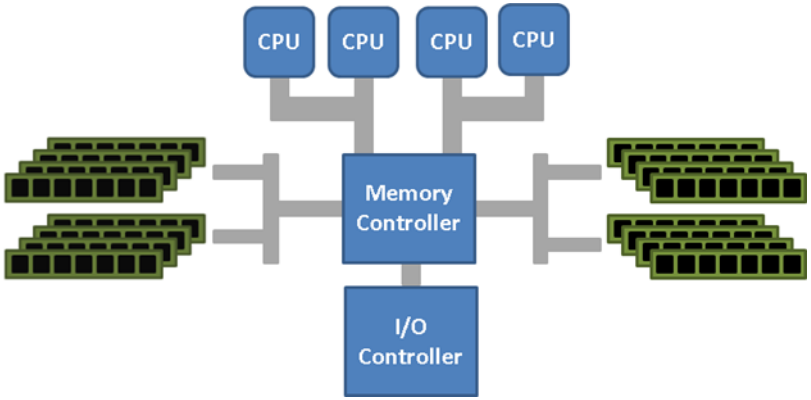
- Symmetric Hierarchical Memory access (SHMEM)
- Partitioned Global Address Space (PGAS): Unified Parallel C (UPC), Co-Array Fortran (CAF)

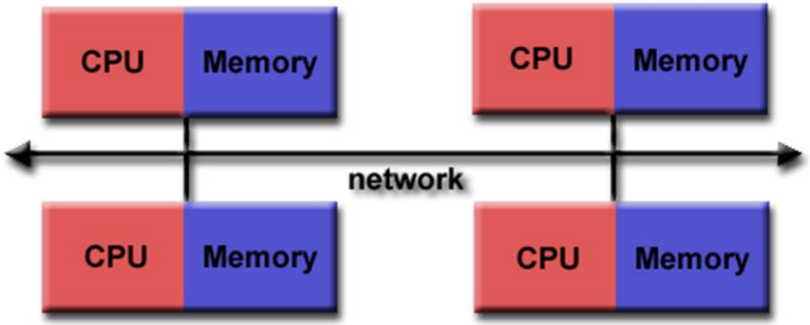
Модель программирования MPI

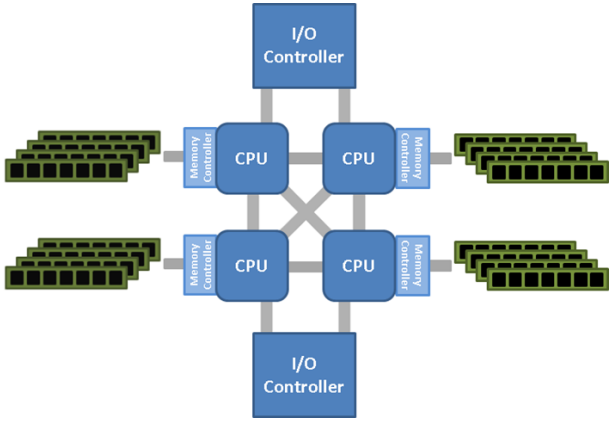
Модель программирования MPI

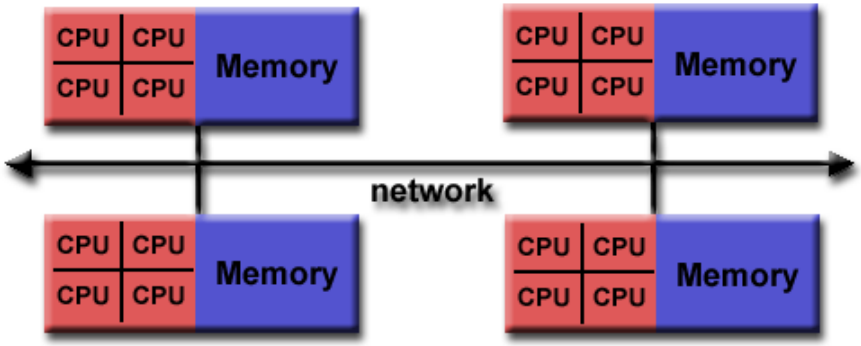


Аппаратные архитектуры









MPI

Message Passing Interface (MPI):

- Стандарт группы MPI Forum (http://mpi-forum.org/)
- Интерфейс прикладного программирования на основе передачи сообщений
- Привязка (Binding) к языкам C и Fortran
- Реализация в виде библиотеки

Реализации MPI

- MPICH (MPI over CHameleon), www.mpich.org
- OpenMPI, http://www.open-mpi.org/
- Реализации производителей: Intel MPI, Microsoft MPI (MS-MPI) и др.

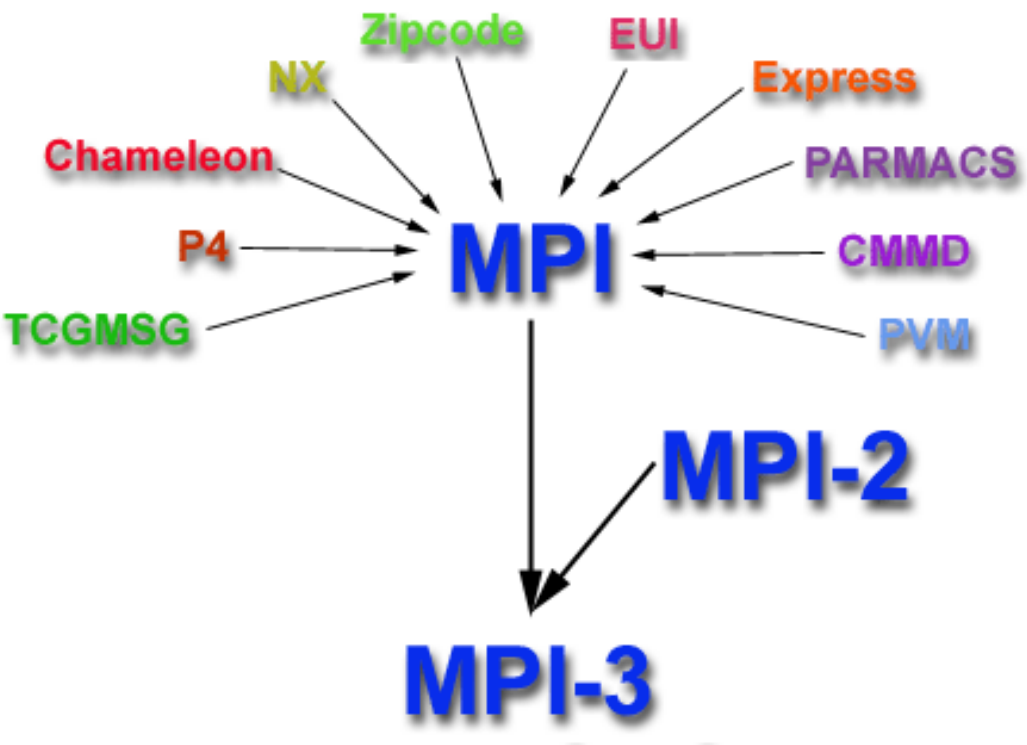
Цели MPI

Переносимость

Высокая производительность

Масштабируемость

Версии MPI



MPI-1 – 1994

MPI-2 – 1996

MPI-3 – 2012

Общая структура программы

```
#include "mpi.h"

int main(int argc, char **argv){

    MPI_Init(&argc, &argv);

    // Параллельная часть

    MPI_Finalize();
}
```

Общая структура программы

Программа начинается с последовательной части

Параллельные процессы создаются одновременно вызовом функции MPI_Init()

- Количество процессов пользователь указывает при запуске программы
- В стандарте MPI-2 можно создавать дополнительные процессы во время работы

Параллельные процессы останавливаются вызовом функции MPI_Finalize();

MPI_Init() и MPI_Finalize() имеют высокие накладные расходы:

- Рекомендуется вызывать один раз

Привязка MPI к C

Функции MPI:

- Определены в заголовочном файле mpi.h
- Начинаются с префикса MPI_

Возвращают код выполнения операции

- MPI_SUCCESS
- Код ошибки (действие по умолчанию – аварийный выход)

Привязка MPI к C++:

- Предложена в стандарте MPI-1.2
- Объявлена устаревшей (deprecated) в MPI-2.2
- Удалена из стандарта в MPI-3
- Boost.MPI

Hello, MPI world!

```
int main(int argc, char **argv){
    int rank, size, len;
    char hostname[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(hostname, &len);

    printf("Hello, MPI world! I am %d of %d on %s\n",
        rank, size, hostname);
    MPI_Finalize();

    return 0;
}
```

Коммуникатор

Группа процессов, которые могут взаимодействовать друг с другом с помощью MPI

Могут создаваться и удаляться в процессе работы программы

Предопределенные коммуникаторы:

- MPI_COMM_WORLD – все процессы в MPI программе
- MPI_COMM_SELF – только один процесс
- MPI_COMM_NULL – пустой коммуникатор

Компиляция

Стандарт MPI не содержит требований к компиляции и запуску программ

MPICH/OpenMPI

- `mpicc hw.c -o hw`

mpicc:

- bash-скрипт
- Подключает необходимые библиотеки MPI
- Устанавливает необходимые опции компилятора
- Вызывает компилятор (gcc, Intel, PGI и т.п.)
- Передает опции компилятору (-std=c99 и т.п.)

Запуск

Запуск программы с помощью MPI:

- `mpirun –np число_процессоров программа`

Пример:

- `mpirun –np 24 hw`

Запуск

```
Hello, MPI world! I am 1 of 24 on tesla45
Hello, MPI world! I am 17 of 24 on tesla46
Hello, MPI world! I am 2 of 24 on tesla45
Hello, MPI world! I am 4 of 24 on tesla45
Hello, MPI world! I am 5 of 24 on tesla45
Hello, MPI world! I am 3 of 24 on tesla45
Hello, MPI world! I am 6 of 24 on tesla45
Hello, MPI world! I am 8 of 24 on tesla45
Hello, MPI world! I am 7 of 24 on tesla45
...
Hello, MPI world! I am 20 of 24 on tesla46
Hello, MPI world! I am 21 of 24 on tesla46
Hello, MPI world! I am 23 of 24 on tesla46
Hello, MPI world! I am 22 of 24 on tesla46
```

Системы запуска задач

Крупные кластеры общего пользования работают в пакетном режиме:

- Задача не запускается сразу, а устанавливается в очередь
- Когда освобождаются необходимые ресурсы, задача запускается

Система запуска задач SLURM:

- `sbatch -n 24 example1.sh`
- Результаты записываются в файл slurm-<номер-задачи>.out

Подробнее о системе запуска задач в конце лекции

Модель передачи сообщений

Передача данных между процессами осуществляется путем отправки и приема сообщения

Требует совместной работы отправляющего и принимающего процессов

Изменение памяти принимающего процесса происходит при его явном участии

Объединение коммуникации и синхронизации

Функции передачи сообщений

Точка-точка

- Участвуют два процесса

Коллективные

- Участвуют все процессы коммуникатора

Передача сообщений

Какая адресация?

- Кому отправлять/от кого получать данные

Где находятся данные?

- Буфер

Что за данные передаются/принимаются?

- Какой тип данных?
- Какой объем данных?

Как организовать платформо-независимую доставку данных?

- Разный размер одного типа данных
- Разное представление данных (little/big endian)

Передача сообщений

Какая адресация?

- Номер процесса в коммуникаторе (MPI_Comm_rank)

Буфер

- Адрес начала буфера
- Тип элементов данных
- Количество элементов

Как организовать платформо-независимую доставку данных?

- Используются специализированные типы MPI
- Преобразование в типы данных конкретной платформы выполняется библиотекой MPI

Типы данных MPI для C

MPI_CHAR	signed char
MPI_WCHAR	wchar_t - wide character
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT MPI_LONG_LONG	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

MPI_C_COMPLEX MPI_C_FLOAT_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_C_BOOL	_Bool
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_INT8_T MPI_INT16_T MPI_INT32_T MPI_INT64_T	int8_t int16_t int32_t int64_t
MPI_UINT8_T MPI_UINT16_T MPI_UINT32_T MPI_UINT64_T	uint8_t uint16_t uint32_t uint64_t
MPI_BYTE	8 binary digits
MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack

Передача сообщения точка-точка

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

buf – адрес начала буфера, где находятся данные

count – количество элементов в буфере

type – MPI тип данных в буфере

dest – номер процесса, которому нужно отправить сообщение

tag – тег сообщения, может использоваться программистом произвольно

comm – коммуникатор, в котором передается сообщение

Прием сообщения точка-точка

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

buf – адрес начала буфера, куда записать данные

count – количество принимаемых элементов данных

type – MPI тип принимаемых данных

source – номер процесса, от которого нужно принять данные

tag – тег сообщения, может использоваться программистом произвольно

comm – коммуникатор, в котором передается сообщение

status – статус сообщения (источник и тег)

Hello, MPI world 2

```
int main(int argc, char **argv) {
    int rank, size, len, tag=1;
    char host[MPI_MAX_PROCESSOR_NAME];
    char msg[50];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(host, &len);
    if (rank == 0) {
        int i;
        MPI_Status status;
        printf("Hello, MPI world. I am %d of %d on %s\n", rank, size, host);
        for (i = 1; i < size; i++) {
            MPI_Recv(msg, 50, MPI_CHARACTER, MPI_ANY_SOURCE, tag,
                MPI_COMM_WORLD, &status);
            printf("Msg from %d: '%s'\n", status.MPI_SOURCE, msg);
        }
    } else {
        snprintf(msg, 50, "Hello, master. I am %d of %d on %s", rank, size, host);
        MPI_Send(msg, 50, MPI_CHARACTER, 0, tag, MPI_COMM_WORLD);
    }
    MPI_Finalize();
}
```

Запуск

```
Hello, MPI world! I am 0 of 24 on tesla45
Msg from 17: Hello, master! I am 17 of 24 on tesla46
Msg from 2: Hello, master! I am 2 of 24 on tesla45
Msg from 4: Hello, master! I am 4 of 24 on tesla45
Msg from 5: Hello, master! I am 5 of 24 on tesla45
Msg from 3: Hello, master! I am 3 of 24 on tesla45
Msg from 6: Hello, master! I am 6 of 24 on tesla45
Msg from 8: Hello, master! I am 8 of 24 on tesla45
Msg from 7: Hello, master! I am 7 of 24 on tesla45
...
Msg from 20: Hello, master! I am 20 of 24 on tesla46
Msg from 21: Hello, master! I am 21 of 24 on tesla46
Msg from 23: Hello, master! I am 23 of 24 on tesla46
Msg from 22: Hello, master! I am 22 of 24 on tesla46
```

Какое сообщение пришло?

```
int MPI_Probe( int source, int tag, MPI_Comm comm,
    MPI_Status *status)
```

Получение информации о структуре ожидаемого сообщения с блокировкой

Параметры сообщения записываются в status

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

Записывает в count число принятых (после MPI_Recv) или принимаемых (после MPI_Probe) элементов сообщения

Блокирующая передача сообщений

Функции MPI_Send и MPI_Recv блокирующие

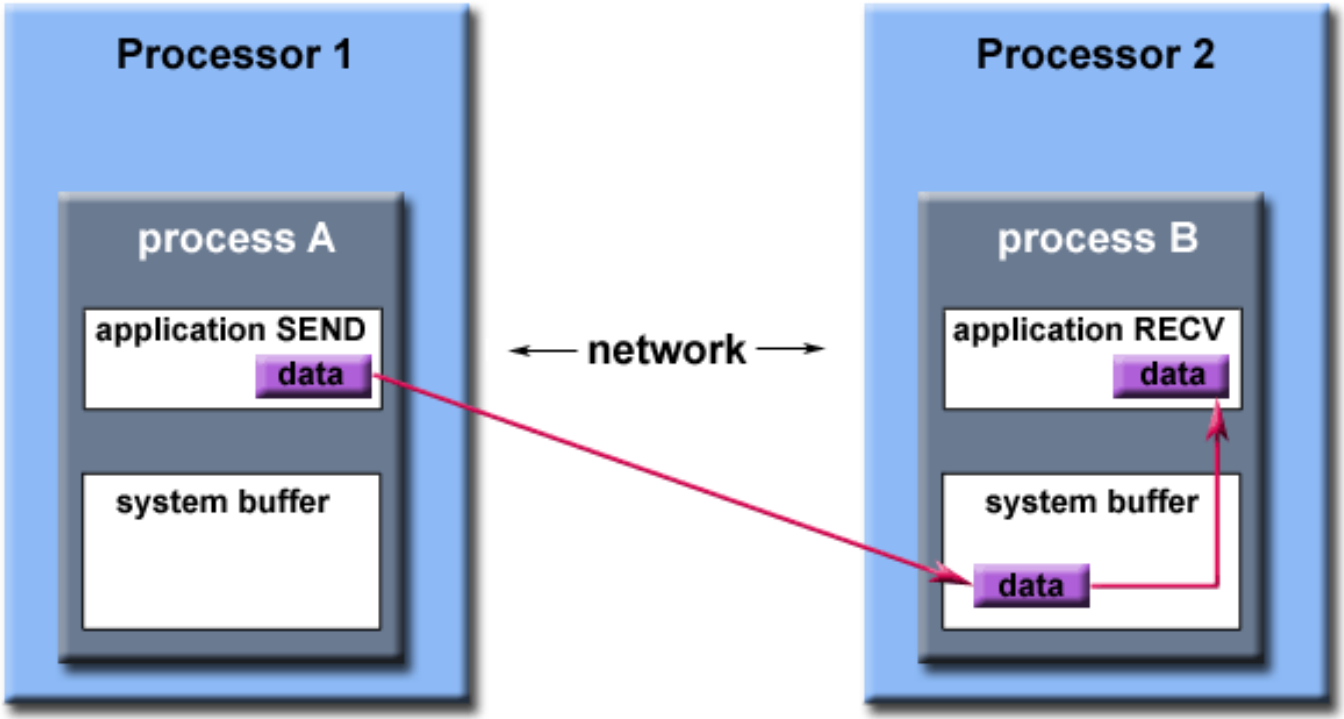
Возвращение из функции MPI_Send:

- Буфер сообщения можно безопасно использовать
- Сообщение может быть не доставлено получателю

Возвращение из функции MPI_Recv:

- Сообщение записано в буфер и может быть считано оттуда

Передача сообщений



https://computing.llnl.gov/tutorials/mpi/

Передача сообщений

Системный буфер:

- Область памяти, куда записываются сообщения, если получатель не готов их принять
- Не стандартизован в MPI
- Размер и правила использования определяются реализацией MPI
- Программист не имеет контроля над системным буфером

Буфер, управляемый пользователем:

- MPI позволяет создать дополнительный буфер, управляемый пользователем (application buffer)
- `int MPI_Buffer_attach(void *buffer, int size)`
- `int MPI_Buffer_detach(void *bufferptr, int *size)`

Виды блокирующих вызовов

MPI_Send – стандартная функция
- Сообщение переписано в системный буфер
- Исходный буфер можно безопасно использовать

MPI_Bsend – функция, использующая application buffer
- Сообщение переписано в application buffer
- Необходимо выделить буфер

MPI_Ssend – синхронная передача
- Получатель начал принимать сообщение

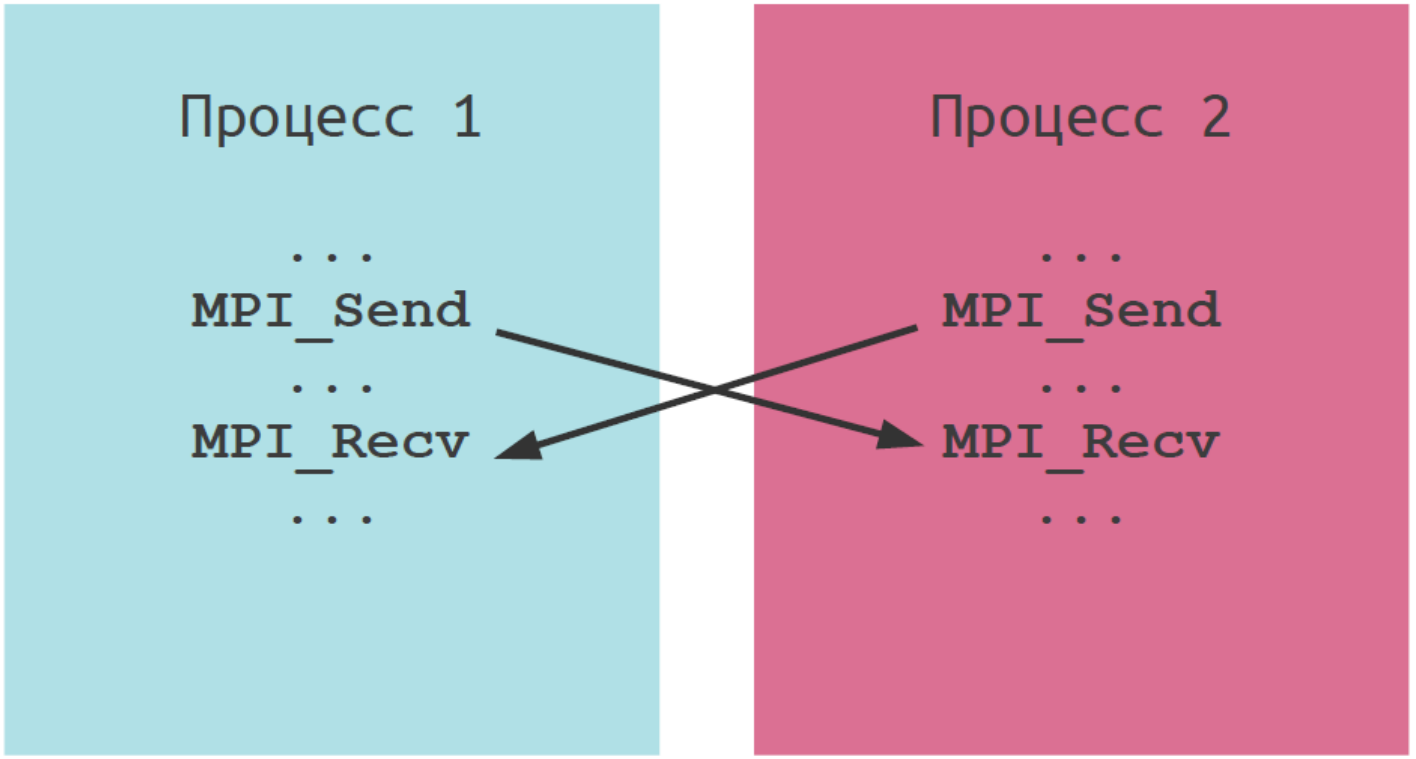
MPI_Rsend – передача по готовности
- Передача без буферизации
- MPI_Rrecv должен быть обязательно вызван до MPI_Rsend

Виды блокирующих вызовов

MPI_Recv может принимать сообщения, отправленные любой блокирующей функцией

- MPI_Send
- MPI_Bsend
- MPI_Ssend
- MPI_Rsend

Чем опасна такая ситуация?



Предотвращение взаимной блокировки в MPI

Поменять порядок операций в одном процессе

Использовать функцию для совмещенного приема и передачи

Использовать неблокирующие функции передачи сообщений

Совмещенные прием и передача

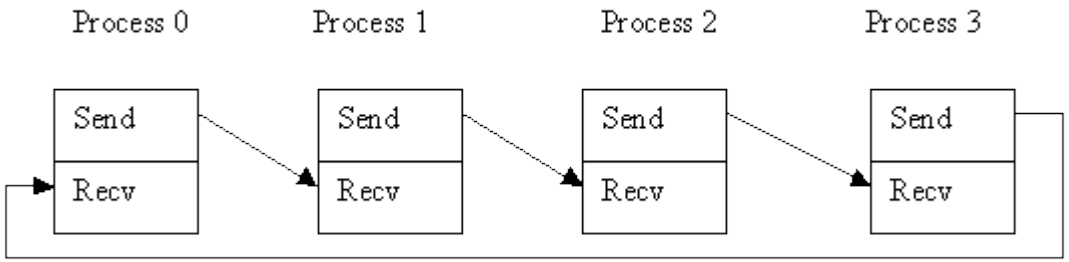
```
int MPI_Sendrecv( void *sendbuf, int sendcount,
    MPI_Datatype sendtype, int dest, int sendtag,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    int source, int recvtag, MPI_Comm comm,
    MPI_Status *status )
```

Предотвращает взаимоблокировку для двух процессов

Совмещенные прием и передача

```
int MPI_Sendrecv( void *sendbuf, int sendcount,
    MPI_Datatype sendtype, int dest, int sendtag,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    int source, int recvtag, MPI_Comm comm,
    MPI_Status *status )
```

Предотвращает взаимоблокировку для двух процессов



Неблокирующие функции

Возврат управления сразу же после вызова

- Передача не завершена
- Буфер использовать нельзя

```
int MPI_Isend( void *buf, int count, MPI_Datatype
    datatype, int dest, int tag, MPI_Comm comm,
    MPI_Request *request )
```

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

Проверка окончания передачи

Блокирующие функции

```
int MPI_Wait ( MPI_Request  *request,
    MPI_Status   *status)
```

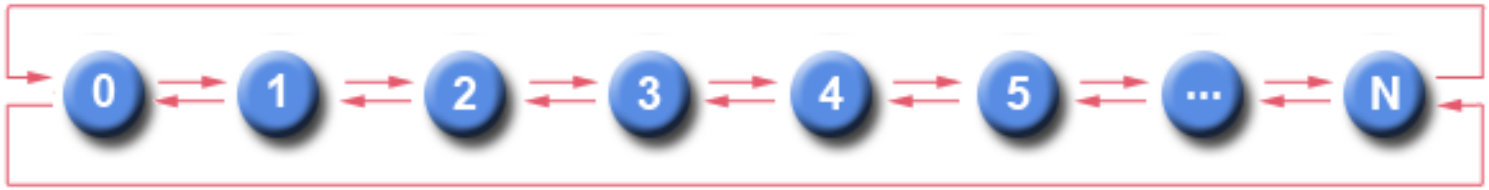
```
MPI_Waitall(), MPI_Waitany(), MPI_Waitsome()
```

Неблокирующие функции

```
int MPI_Test (MPI_Request  *request, int *flag,
    MPI_Status   *status)
```

```
MPI_Testall(), MPI_Testany(), MPI_Testsome()
```

Неблокирующая передача по кольцу



https://computing.llnl.gov/tutorials/mpi/

Неблокирующая передача по кольцу

```
MPI_Request reqs[4]; MPI_Status stats[4];
...
prev = rank-1; next = rank+1;
if (rank == 0)  prev = numtasks - 1;
if (rank == (numtasks - 1))  next = 0;

MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD,
    &reqs[0]);
MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD,
    &reqs[1]);

MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD,
    &reqs[2]);
MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD,
    &reqs[3]);

MPI_Waitall(4, reqs, stats);
```

Вычисления и обмен данными

```
void slave() {
    double *buf1, *buf2, *tmp;
    MPI_Status status;
    buf1 = (double *) malloc(sizeof(double) * size);
    buf2 = (double *) malloc(sizeof(double) * size);
    MPI_Recv(buf1, size, MPI_DOUBLE, MPI_ANY_SOURCE,
        MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    while (!finished()) {
        MPI_Request request;
        MPI_Irecv(buf2, size, MPI_DOUBLE, MPI_ANY_SOURCE,
            MPI_ANY_TAG, MPI_COMM_WORLD, &request);
        /* Обработка данных в buf1 */
        MPI_Wait(&request, &status);
        tmp = buf1; buf1 = buf2; buf2 = tmp;
    }
}
```

Коллективные взаимодействия

Участвуют все процессы коммуникатора

Соответствующая функция должна быть вызвана каждым процессом

Все коллективные функции являются блокирующими и не используют теги

- В стандарте MPI-3 появились неблокирующие коллективные функции

Коллективные и взаимодействия «точка-точка» в рамках одного коммуникатора используют различные контексты

- Нельзя отправить сообщение вызовом функции «точка-точка» и получить вызовом коллективной функции

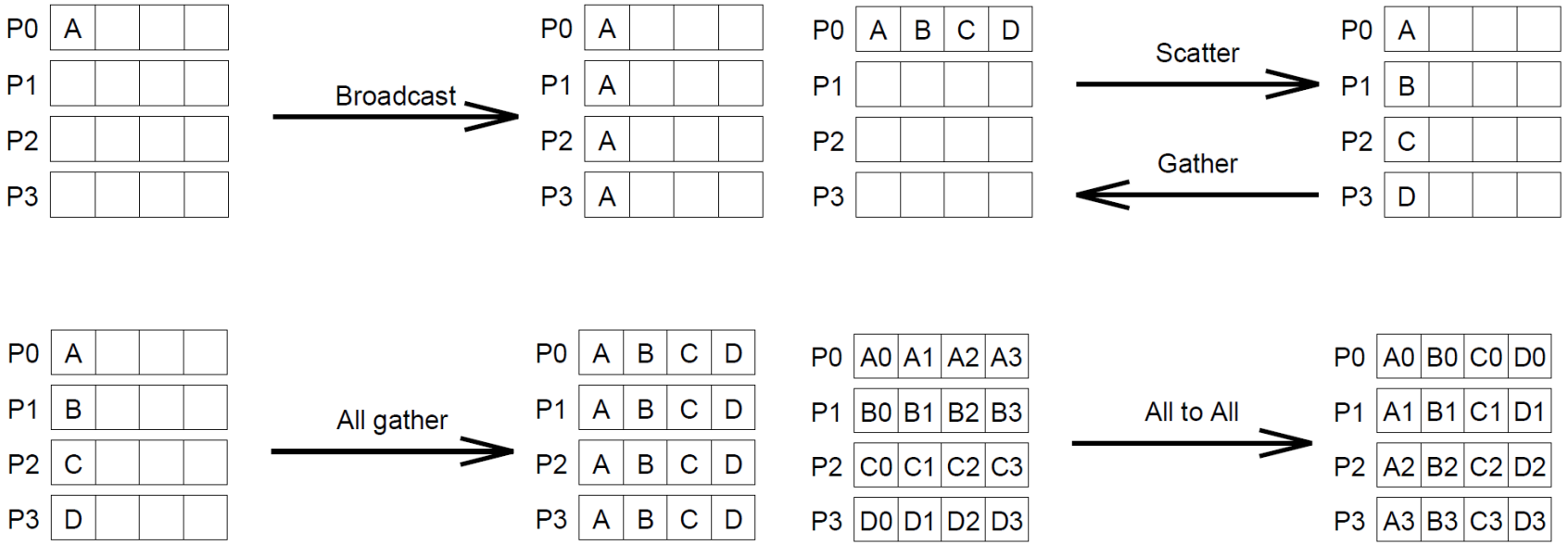
Типы коллективных взаимодействий

Обмен данными

Коллективные вычисления

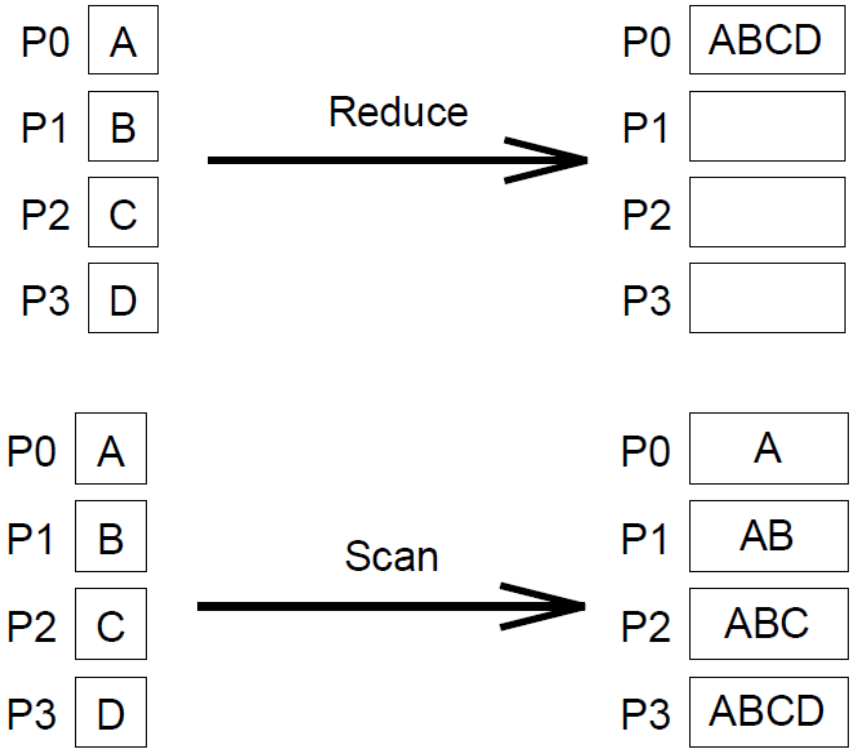
Синхронизация

Обмен данными



http://www.idi.ntnu.no/~elster/tdt4200-f09/gropp-mpi-tutorial.pdf

Коллективные вычисления



http://www.idi.ntnu.no/~elster/tdt4200-f09/gropp-mpi-tutorial.pdf

Операции в коллективных вычислениях

MPI Name	Operation
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_PROD	Product
MPI_SUM	Sum
MPI_LAND	Logical and
MPI_LOR	Logical or
MPI_LXOR	Logical exclusive or (xor)
MPI_BAND	Bitwise and
MPI_BOR	Bitwise or
MPI_BXOR	Bitwise xor
MPI_MAXLOC	Maximum value and location
MPI_MINLOC	Minimum value and location

http://www.idi.ntnu.no/~elster/tdt4200-f09/gropp-mpi-tutorial.pdf

Операции синхронизации

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

Блокирует работу процессов, вызвавших данную функцию, до тех пор, пока все оставшиеся процессы группы comm также не выполнят эту процедуру

Операции синхронизации

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

Блокирует работу процессов, вызвавших данную функцию, до тех пор, пока все оставшиеся процессы группы comm также не выполнят эту процедуру

Насколько сложен MPI?

Количество функций

- MPI-1 ≈ 125
- MPI-2 ≈ 500
- MPI-3 ≈ 400

Насколько сложен MPI?

Количество функций

- MPI-1 ≈ 125
- MPI-2 ≈ 500
- MPI-3 ≈ 400

Не обязательно знать все функции MPI, чтобы успешно писать программы

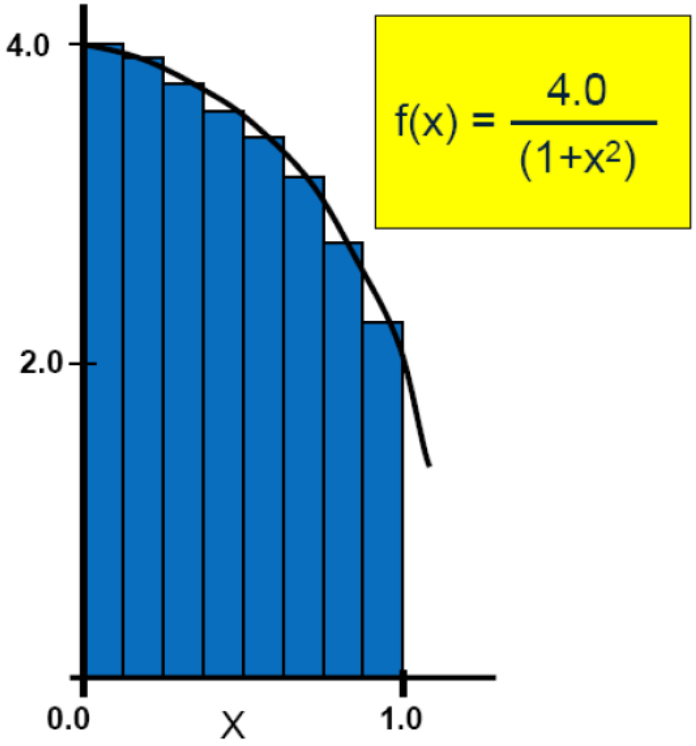
- Минимум 8 функций

8 функций MPI

```
MPI_Init()
MPI_Finalize()
MPI_Comm_size()
MPI_Comm_rank()
MPI_Send()
MPI_Recv()
MPI_Bcast()
MPI_Reduce()
```

http://www.idi.ntnu.no/~elster/tdt4200-f09/gropp-mpi-tutorial.pdf

Вычисление π




```
int main(int argc, char *argv[]) {
    const int num_steps = 1E9;
    double x, sum = 0.0;
    const double step = 1.0/num_steps;
    int i;
    for (i = 0; i < num_steps; i++) {
        x = (i+0.5)*step;
        sum += 4.0/(1.0+x*x);
    }
    printf("Pi = %.10f\n", step*sum);
}
```

Вычисление π

```
int n, myid, numprocs, i;
double mypi, pi, h, sum, x, time;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if (myid == 0) {
    printf("Enter the number of intervals: "); fflush(stdout);
scanf("%d",&n);
    time = MPI_Wtime();
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
h = 1.0 / (double) n;
sum = 0.0;
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += 4.0 / (1.0 + x*x);
}
```

Вычисление π (продолжение)

```
mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
if (myid == 0) {
    time = MPI_Wtime() - time;
    printf("pi is approximately %.16f, Run time is %fs\n", pi,
time);
}
MPI_Finalize();
```

Подходы к разработке программ на MPI

Multiple Programs Multiple Data

- Наиболее общий подход
- Сложен в реализации и практическом использовании
- Почти не применяется

Single Program Multiple Data

- Все процессы выполняют одну программу
- Обрабатываются разные порции данных в зависимости от номера и количества процессов

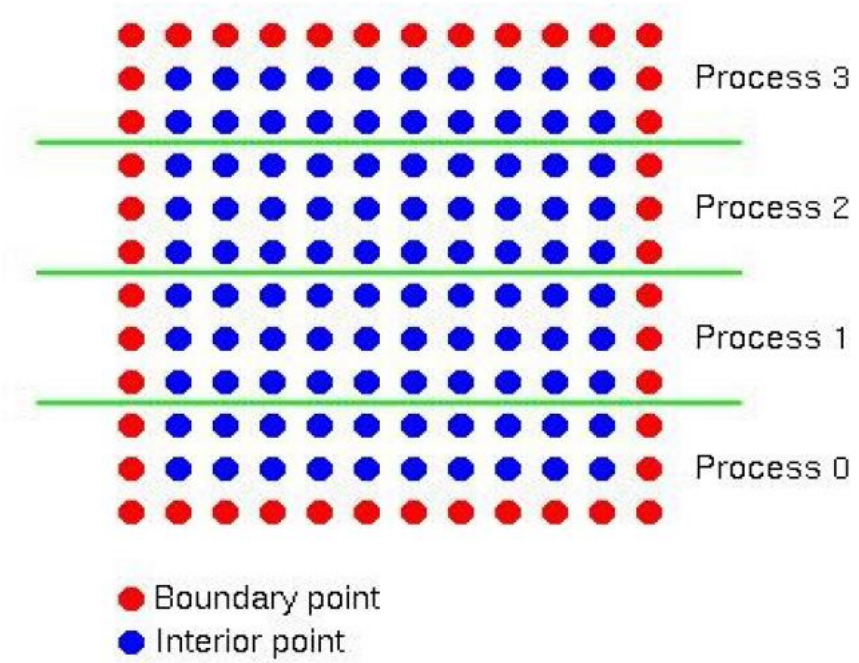
Master-Slave:

- Один главный процесс (Мастер), раздает задания остальным процессам (Slave)

Hello, MPI world (Master-Slave)

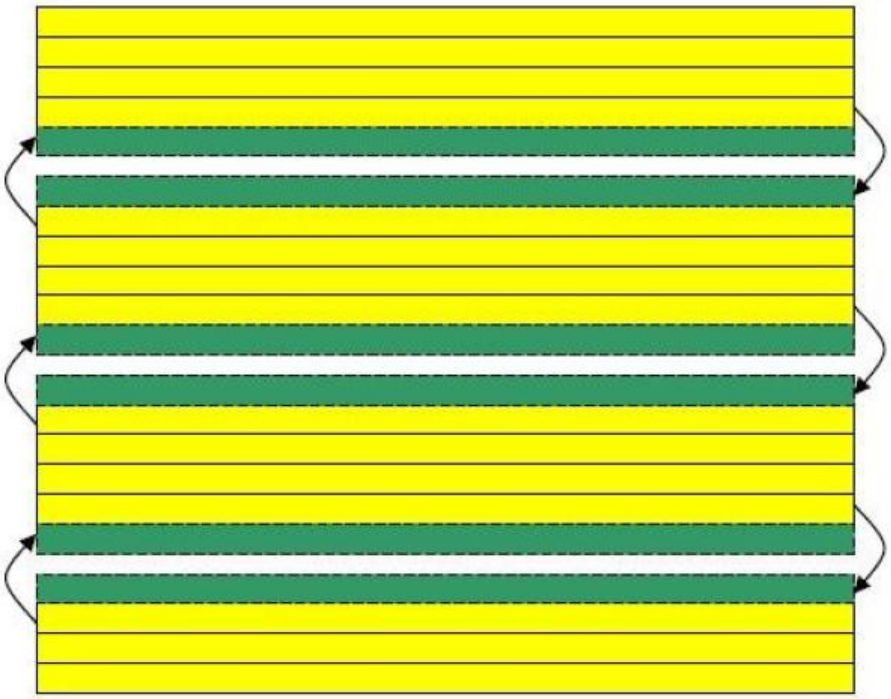
```
int main(int argc, char **argv) {
    int rank, size, len, tag=1;
    char host[MPI_MAX_PROCESSOR_NAME];
    char msg[50];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(host, &len);
    if (rank == 0) {
        int i;
        MPI_Status status;
        printf("Hello, MPI world. I am %d of %d on %s\n", rank, size, host);
        for (i = 1; i < size; i++) {
            MPI_Recv(msg, 50, MPI_CHARACTER, MPI_ANY_SOURCE, tag,
                MPI_COMM_WORLD, &status);
            printf("Msg from %d: '%s'\n", status.MPI_SOURCE, msg);
        }
    } else {
        snprintf(msg, 50, "Hello, master. I am %d of %d on %s", rank, size, host);
        MPI_Send(msg, 50, MPI_CHARACTER, 0, tag, MPI_COMM_WORLD);
    }
    MPI_Finalize();
}
```

Решение уравнения Лапласа методом Якоби




```
while (not converged) {
    for (i,j)
        xnew[i][j] = (x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4;
    for (i,j)
        x[i][j] = xnew[i][j];
    }
}
```

Схема пульсации



Отправить свои границы соседям

Получить границы от соседей

Вычислить значения внутри своей полосы

Решение уравнения Лапласа методом Якоби

```
/* 12 x 12 mesh, 4 processors */
#define maxn 12
...
/* xlocal[][0] is lower ghostpoints */
/* xlocal[][maxn+2] is upper */
double xlocal[(12/4)+2][12];
double xnew[(12/3)+2][12];
double x[maxn][maxn];
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
/* Read the data from the named file */
if (rank == 0) {
    ...
    MPI_Scatter( x[0], maxn * (maxn/size), MPI_DOUBLE,
    xlocal[1], maxn * (maxn/size), MPI_DOUBLE,
    0, MPI_COMM_WORLD );
```

Решение уравнения Лапласа методом Якоби

```
itcnt = 0;
do {
    /* Send up unless I'm at the top, then receive from below */
    if (rank > 0)
        MPI_Send( xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1,
            MPI_COMM_WORLD );
    if (rank < size - 1)
        MPI_Recv( xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank + 1, 1,
            MPI_COMM_WORLD, &status );
    /* Send down unless I'm at the bottom */
    if (rank < size - 1)
        MPI_Send( xlocal[maxn/size], maxn, MPI_DOUBLE, rank + 1, 0,
            MPI_COMM_WORLD );
    if (rank > 0)
        MPI_Recv( xlocal[0], maxn, MPI_DOUBLE, rank - 1, 0,
            MPI_COMM_WORLD, &status );
```

Решение уравнения Лапласа методом Якоби

```
/* Compute new values (but not on boundary) */
itcnt ++;
diffnorm = 0.0;
for (i=i_first; i<=i_last; i++)
    for (j=1; j<maxn-1; j++) {
        xnew[i][j] = (xlocal[i][j+1] + xlocal[i][j-1] +
            xlocal[i+1][j] + xlocal[i-1][j]) / 4.0;
            diffnorm += (xnew[i][j] - xlocal[i][j]) *
            (xnew[i][j] - xlocal[i][j]);
    }
/* Only transfer the interior points */
for (i=i_first; i<=i_last; i++)
    for (j=1; j<maxn-1; j++)
        xlocal[i][j] = xnew[i][j];
MPI_Allreduce( &diffnorm, &gdiffnorm, 1, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD );
gdiffnorm = sqrt( gdiffnorm );
if (rank == 0) printf( "At iteration %d, diff is %e\n", itcnt,
    gdiffnorm );
} while (gdiffnorm > 1.0e-2 && itcnt < 100);
```

Решение уравнения Лапласа методом Якоби

```
/* Collect the data into x and print it */
MPI_Gather( xlocal[1], maxn * (maxn/size), MPI_DOUBLE,
    x, maxn * (maxn/size), MPI_DOUBLE,
    0, MPI_COMM_WORLD );
if (rank == 0) {
    printf( "Final solution is\n" );
    ...
}
MPI_Finalize( );
```

Оптимизации

Сокращение обменов

- Обмениваться границами через итерацию

2D-декомпозиция

- Почему?

Совмещение вычислений и обмена данными

- Как?

Влияние способа декомпозиции

Время передачи n данных между процессами

$$T = L + \frac{n}{B}$$

1D

$$T = 2(L + \frac{n}{B})$$

2D

$$T = 4(L + \frac{n}{\sqrt{p}B})$$

Совмещение вычислений и обмена данными

Обновить края своей полосы

Приготовиться к приему краев соседей (MPI_Irecv)

Начать отправку своих краев соседям (MPI_Isend)

Обновить внутренние клетки своей полосы

Принять края от соседей (MPI_Waitall)

Новые возможности MPI

MPI-2

- Динамическое создание/удаление процессов
- Односторонние взаимодействия (Put/Get)
- Параллельный ввод/вывод
- Привязка к C++ объявлена устаревшей
- Коллективные операции между процессами в разных коммуникаторах

MPI-3

- Неблокирующие коллективные операции
- Новые односторонние операции взаимодействия
- Привязка к Fortran 2008
- Привязка к C++ удалена из стандарта

Дополнительные материалы

MPI Tutorials

- https://computing.llnl.gov/tutorials/mpi/
- http://www.idi.ntnu.no/~elster/tdt4200-f09/gropp-mpi-tutorial.pdf

Упражнения по MPI

- http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/

Учебники по MPI на русском от МГУ

- Антонов А.С. Параллельное программирование с использованием технологии MPI. http://parallel.ru/tech/tech_dev/MPI/
- Антонов А.С. Вычислительный практикум по технологии MPI. http://parallel.ru/tech/tech_dev/MPIcourse/

Домашнее задание

Реализуйте параллельную версию игры «Жизнь» с использованием MPI

- Загрузка начальной конфигурации клеток из файла, выполнение заданного количества шагов эволюции, сохранение полученной конфигурации в файл
- Визуализация не требуется

Постройте графики зависимости ускорения и эффективности от количества процессов и размера задачи

Прокомментируйте полученные результаты

Суперкомпьютер «УРАН» ИММ УрО РАН

Производительность:

- Пиковая – 225 TFlops
- «Реальная» (тест Linpack) – 109 TFlops

Технические характеристики:

- 204 вычислительных узла
- 408 процессоров Intel Xeon
- 352 графических ускорителя NVIDIA Tesla GPU
- Коммуникационная сеть Infiniband DDR
- Дисковая емкость 250 ТБ

Центр коллективного пользования с доступом через Интернет

Как работать?

Инструкции:

- parallel.uran.ru
- Базовые инструкции в Wiki курса

ОС Linux

Доступ по SSH

Компиляторы:

- gcc
- Intel
- PGI

Запуск задач

Система запуска задач

- SLURM (Simple Linux Utility for Resource Management)

Интерактивный запуск задачи:

- `srun -n 24 hello_world`

Пакетный запуск задачи (предпочтительно):

- `sbatch -n 24 hello_world.sh`

```
$ cat hello_world.sh
#!/bin/bash
srun /home/u1213/parallel_course/hello_world
```

Очередь

```
[u1213@umt parallel_course]$ squeue
  JOBID     NAME  USER ST START_TIME           TIME    TIMELIMIT  NODES PRI
 308058      vasp u2573  R 2015-04-06T12:30  1:52:33     20:00:00      2 999
 307954   sbatch u2567  R 2015-04-05T23:28 14:54:31   1-04:20:00      1 986
 307953   sbatch u2567  R 2015-04-05T23:19 15:04:08   1-04:20:00      1 986
 307941   sbatch u2567  R 2015-04-05T21:55 16:27:57   1-06:00:00      1 986
 308129      vasp u2585  R 2015-04-06T14:06    16:39      4:00:00      2 981
 308123      vasp u2585  R 2015-04-06T14:01    22:15      5:00:00      2 981
 307956   sbatch u2626  R 2015-04-06T08:05  6:17:55     16:40:00      8 963
 307961   sbatch u2626  R 2015-04-06T08:25  5:57:29     16:40:00      8 962
 307903   sbatch u2655  R 2015-04-06T04:18 10:05:06     12:00:00     20 959
 307960   sbatch u2733  R 2015-04-06T08:18  6:04:48     16:40:00      8 955
 307959   sbatch u2733  R 2015-04-06T08:15  6:07:39     16:40:00      8 955
 307958   sbatch u2733  R 2015-04-06T08:11  6:12:03     16:40:00      8 955
```

Раздел debug

Кластер как правило очень высоко загружен

- Не оставляйте на последний день!

Ночью (с 00:00 до 05:00) работает бэкап

- Зайти на управляющую машину почти невозможно (100% загрузка ввода/вывода)
- Расчетные задачи работают на узлах

Раздел коротких отладочных задач debug

- Продолжительность до 20 минут

Использование

- `sbatch -p debug -n 24 hello_world.sh`

Вопросы?