Суперкомпьютерные вычислительные технологии.

Лекционно-практический курс
для студентов 5 курса факультета ВМиК МГУ
сентябрь – декабрь 2013 г.

Лектор доцент Н.Н.Попова

Лекция 4
27 сентября 2013 г.

Задание 1. Исследование эффективности решения задачи Дирихле для уравнения Лапласа. OpenMP-реализация

- Задача Дирихле для уравнения Лапласа (1).

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2} = 0, (x, y, z) \in D, \\ u(x, y, z) = g(x, y, z), (x, y, z) \in D^0, \end{cases} \quad (1)$$

где $u(x, y, z)$ - функция, удовлетворяющая в области D уравнению Лапласа и принимающая на границе D^0 области D значения $g(x, y, z)$.

М.В.Абакумов, А.В.Гулин Лекции по численным методам математической физики. Учебное пособие. – М.-:ИНФРА-М, 2013.-158

Постановка задачи. Разностная схема.

- Численный подход к решению задачи (1) основан на замене производных соответствующими конечными разностями (2).

$$\frac{u_{i+1,j,k}-2u_{i,j,k}+u_{i-1,j,k}}{h^2}+\frac{u_{i,j+1,k}-2u_{i,j,k}+u_{i,j-1,k}}{h^2}+\frac{u_{i,j,k+1}-2u_{i,j,k}+u_{i,j,k-1}}{h^2}=0, \quad (2)$$

где $h \geq 0$ *- шаг сетки,* $u_{i,j,k}$ *- значение функции u(x, y, z) в точке* $x = x_i = ih, i = \overline{0, M+1}, y = y_j = jh, j = \overline{0, N+1}, z = z_k = kh, h = \overline{0, L+1},$ *где* M, N, L *- количество внутренних узлов по каждой координате в области* D.

Метод Якоби.

- *итерационный метод Якоби (3).*

$$u^{n}_{i,j,k} = (u^{n-1}_{i-1,j,k} + u^{n-1}_{i+1,j,k} + u^{n-1}_{i,j+1,k} + u^{n-1}_{i,j-1,k} + u^{n-1}_{i,j,k+1} + u^{n-1}_{i,j,k-1})/6$$

$$u^{n}_{i,j,k} = g_{i,j,k}, (x,y,z) \in D^{0}, n = 1,2,.. \qquad (3)$$

где n - номер итерации.

Уравнение Лапласа (2D)

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad x, y \in [0,1] \tag{1}$$

Краевые условия:

$$\begin{aligned} u(x,0) &= \sin(\pi x) && 0 \le x \le 1 \\ u(x,1) &= \sin(\pi x) e^{-x} && 0 \le x \le 1 \\ u(0,y) &= u(1,y) = 0 && 0 \le y \le 1 \end{aligned} \tag{2}$$

Аналитическое решение:

$$u(x,y) = \sin(\pi x) e^{-xy} \qquad x, y \in [0,1] \tag{3}$$

Дискретизация уравнения Лапласа

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \;\; j = 1,2,\ldots,m \qquad (4)$$

где *n* и *n+1* текущий и следующий шаг,

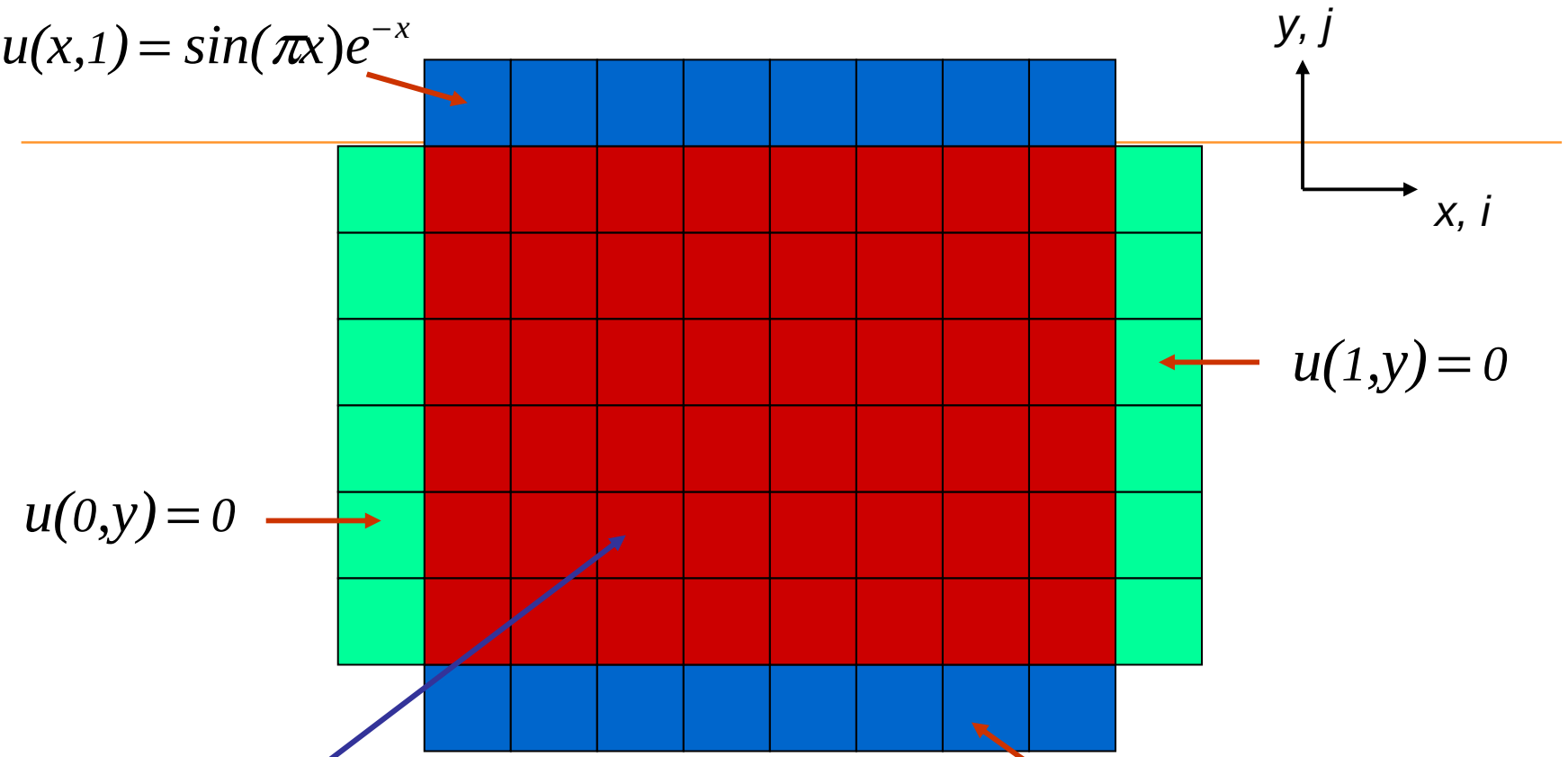
$$u_{i,j}^{n} = u^{n}(x_i, y_j) \qquad i = 0,1,2,\ldots,m+1; \;\; j = 0,1,2,\ldots,m+1 \qquad (5)$$

$$= u^{n}(i\Delta x,\, j\Delta y)$$

Для простоты

$$\Delta x = \Delta y = \frac{1}{m+1}$$

Вычислительная область



$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \quad j = 1,2,\ldots,m$$

5-точечный шаблон

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4}$$

внутренняя область, на которой ищется решение уравнения

граничная область.

Голубые клетки – неоднородные граничные условия,

Зеленые - однородные

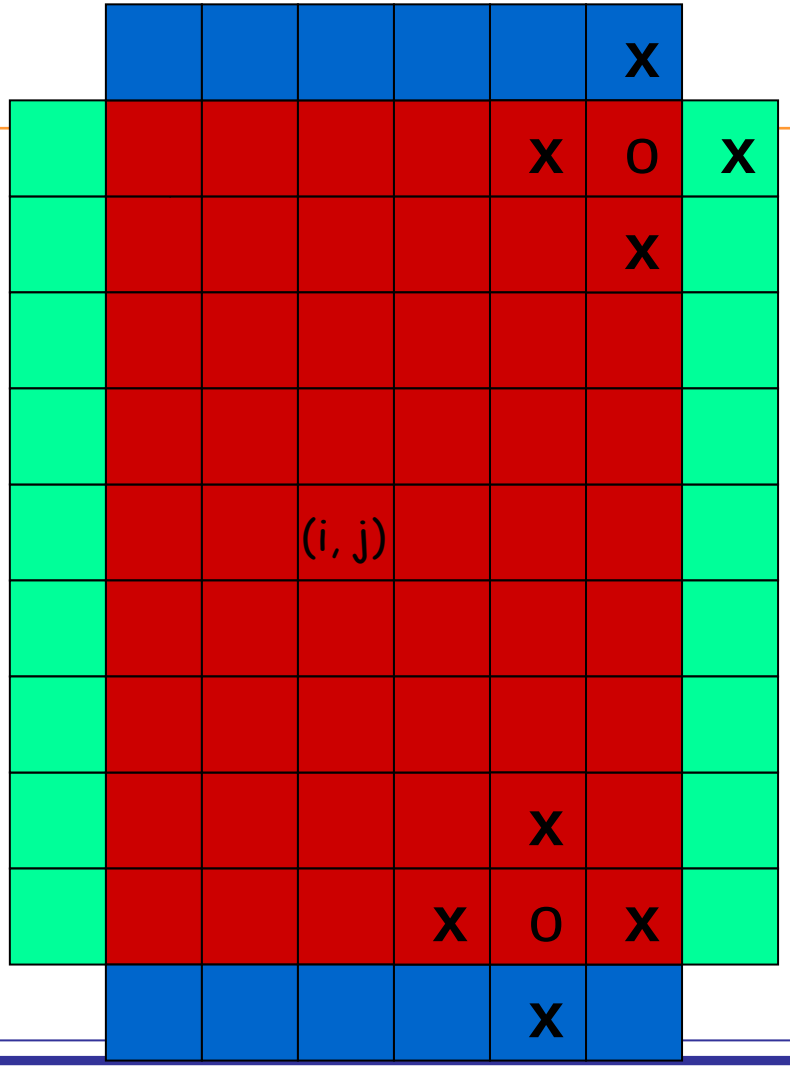
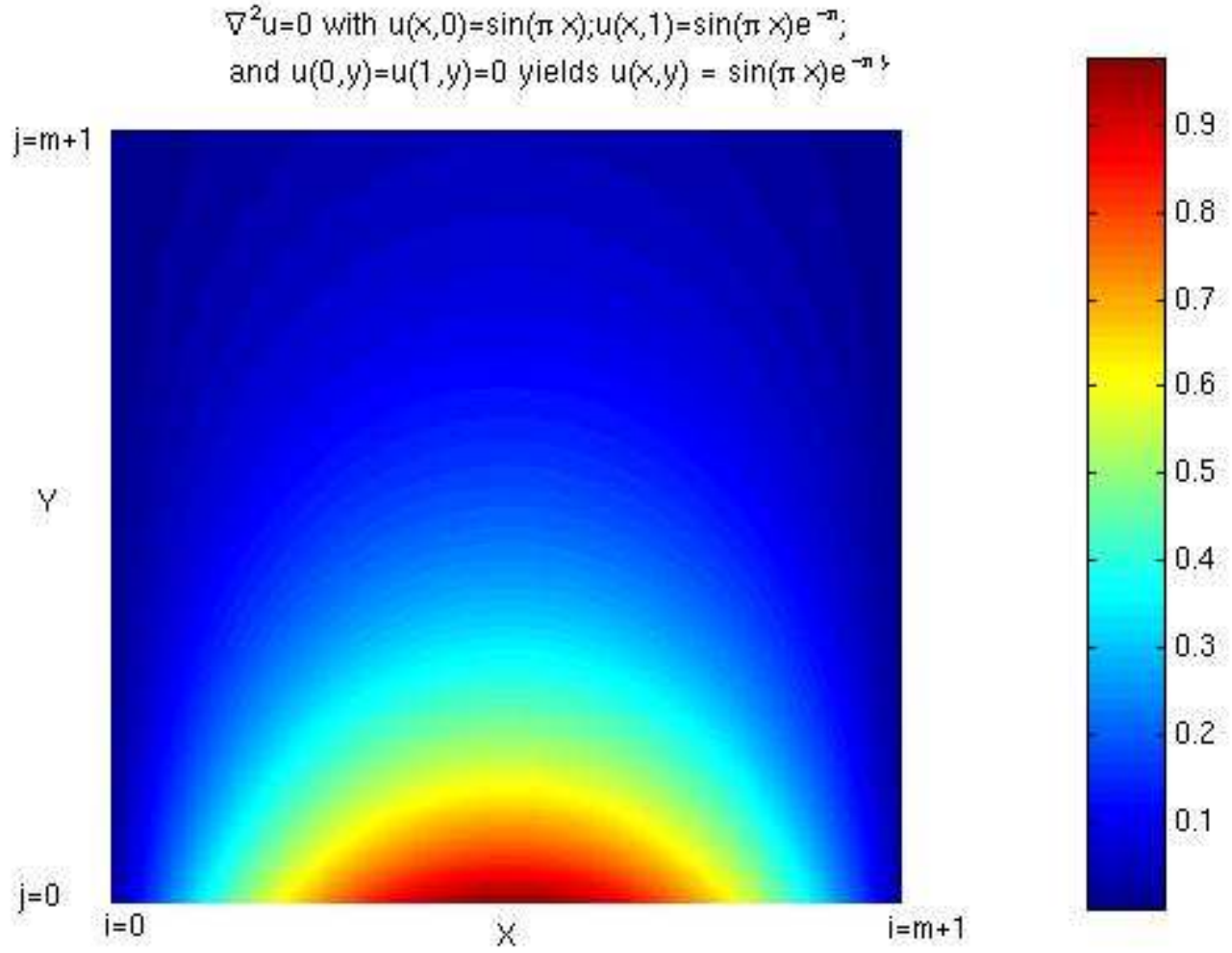


Схема метода Якоби

1. Задание начальных значений *и во всех внутренних точках (i,j) в момент времени n=0*.
2. Используя 5-точечный шаблон вычислить значения во внутренних точках $u_{i,j}^{n+1}$ *(i,j)*.
3. Завершить процесс, если заданная точность достигнута.
4. Иначе: $u_{i,j}^{n} = u_{i,j}^{n+1}$ для всех внутренних точек.
5. Перейти на шаг 2.

Это очень простая схема. Медленно сходится, поэтому не используется для решения реальных задач.

Решение (линии уровня)



Варианты первого задания «**Численное решение задачи Дирихле**»

- Вариант 1. Параллельный алгоритм Якоби.
- Вариант 2. Метод попеременных направлений.
- Вариант 3. Метод красно-черного упорядочивания переменных (Red-Black).

Метод последовательной верхней релаксации (Successive Over Relaxation)

1. Определяем начальное значение *u во всех внутренних* точках *(i,j)*.
2. Определяем $\omega_n \quad (0 < \omega_n < 2)$
3. Используя 5-точечный шаблон вычисляем $u'_{i,j}$ во всех внутренних точках *(i,j)*.
4. Вычисляем $u_{i,j}^{n+1} = \omega_n u'_{i,j} + (1-\omega_n) u_{i,j}^{n}$
5. Завершаем процесс, если точность достигнута, иначе
6. Обновляем: $u_{i,j}^{n} = u_{i,j}^{n+1} \quad \forall\ i, j$
7. Переход на пункт 2.

$$\omega_0 = 0 \ ; \ \omega_1 = \frac{1}{1-\rho^2/2} \ ; \ \omega_2 = \frac{1}{1-\rho^2\omega_1/4}$$

$$\omega_n = \frac{1}{1-\rho^2\omega_{n-1}/4}; \quad n > 2$$

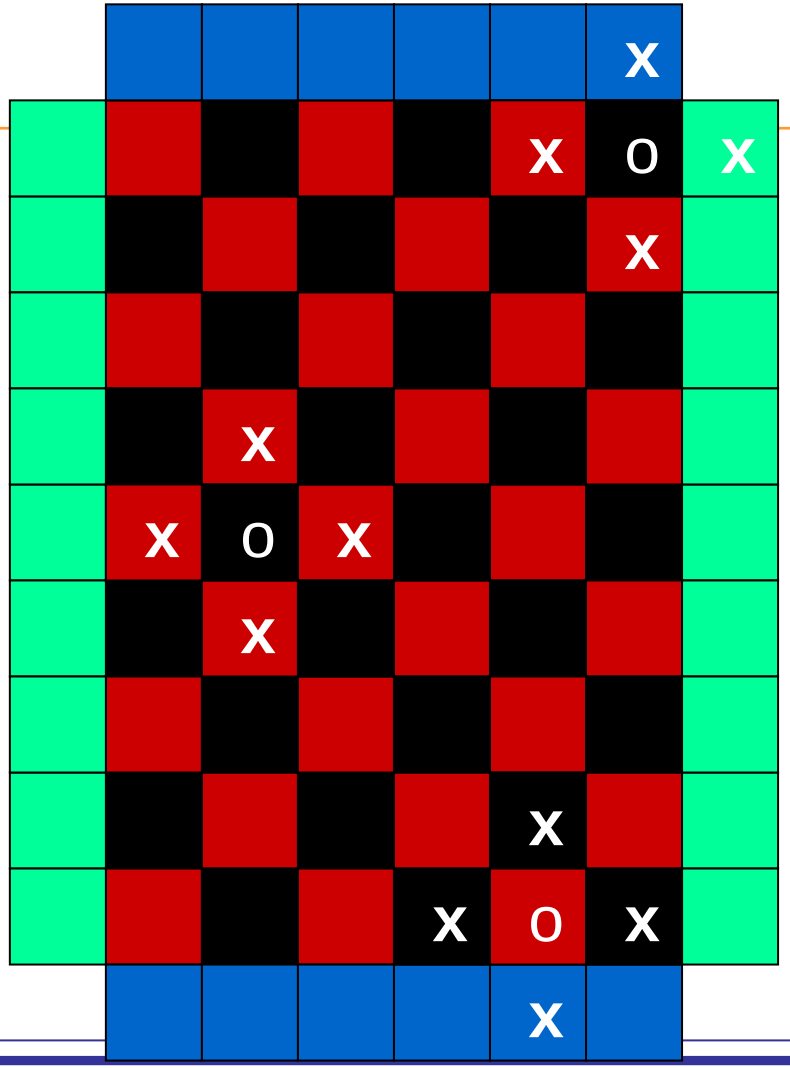
$$\rho = 1 - \left(\frac{\pi}{2(m+1)}\right)^2$$

На шаге 3 вычисляем u‘, используя u в момент n+1.

Red-Black SOR

Учитывая 5-точечный шаблон, решение в черных клетках зависят от 4-ех красных клеток. Соответственно, красные клетки зависят только от 4-ех черных клеток. Параллельный алгоритм:

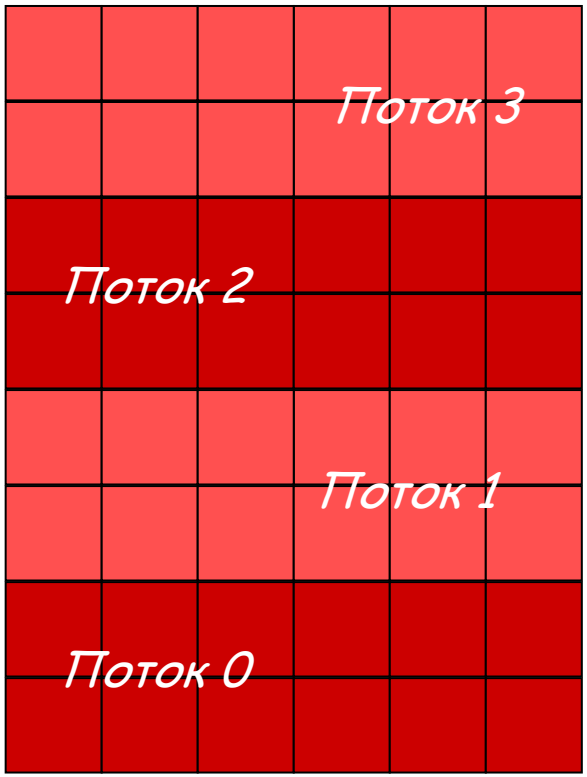
1. Вычисляем *u* в черных клетках в момент времени *n+1* используя u, вычисленные в красных клетках в момент времени *n*.
2. Вычисляем *u* в красных клетках в момент времени *n+1, используя u* в черных клетках, вычисленные для *n+1*.
3. Повторяем шаги 1 и 2 пока не будет достигнута точность.



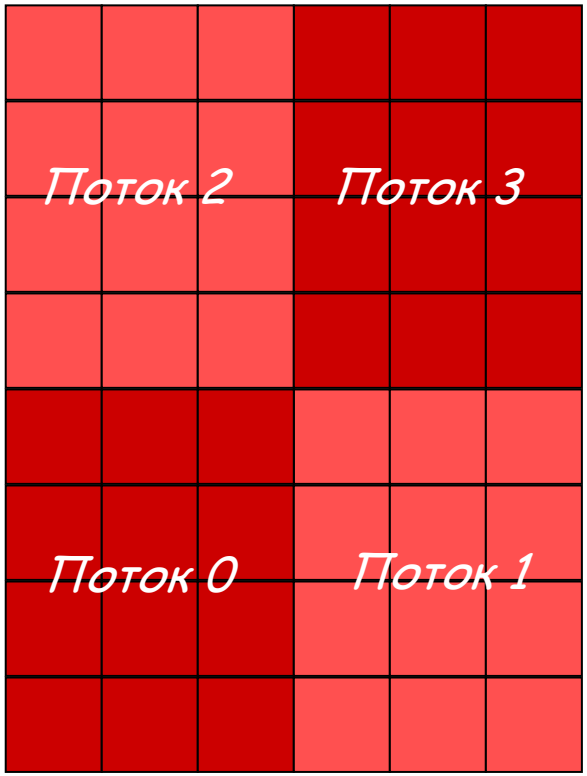
Порядок выполнения шагов 1 и 2 может быть произвольным

Параллельная реализация метода Якоби

1D Domain Decomposition



2D Domain Decomposition



Распределение данных – 1D

5-точечная схема требует значений от соседних потоков

Необходим обмен данными на границах области

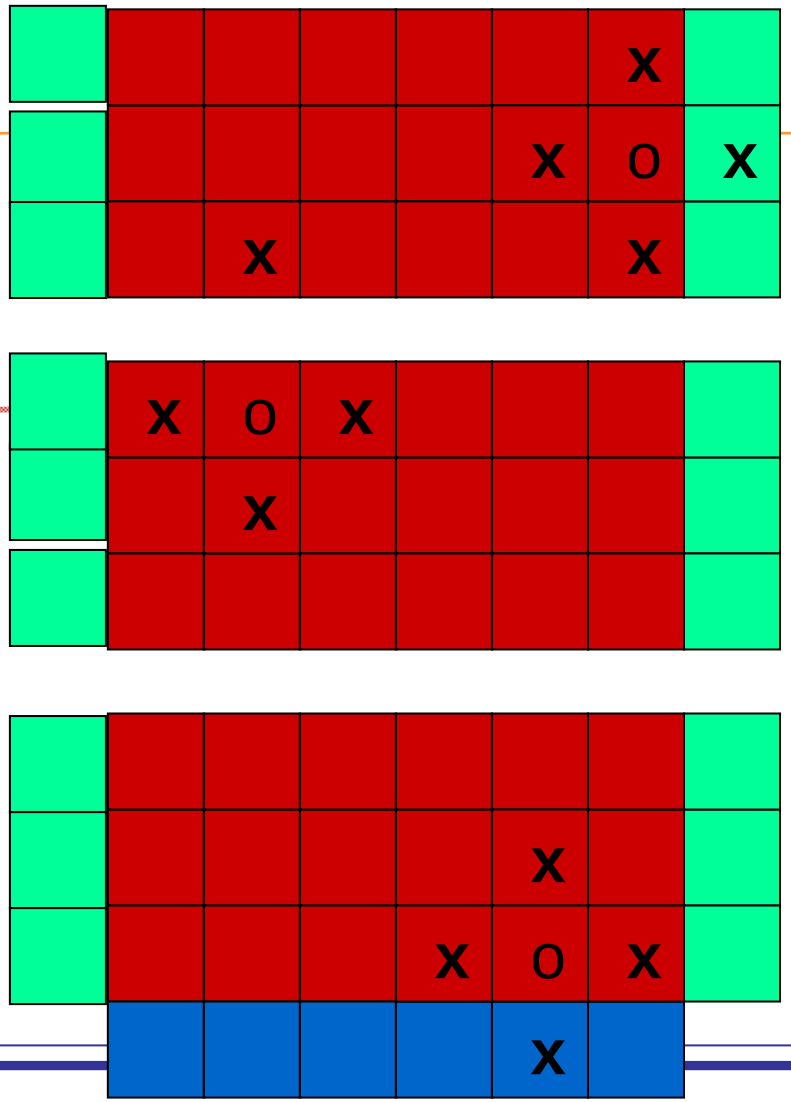
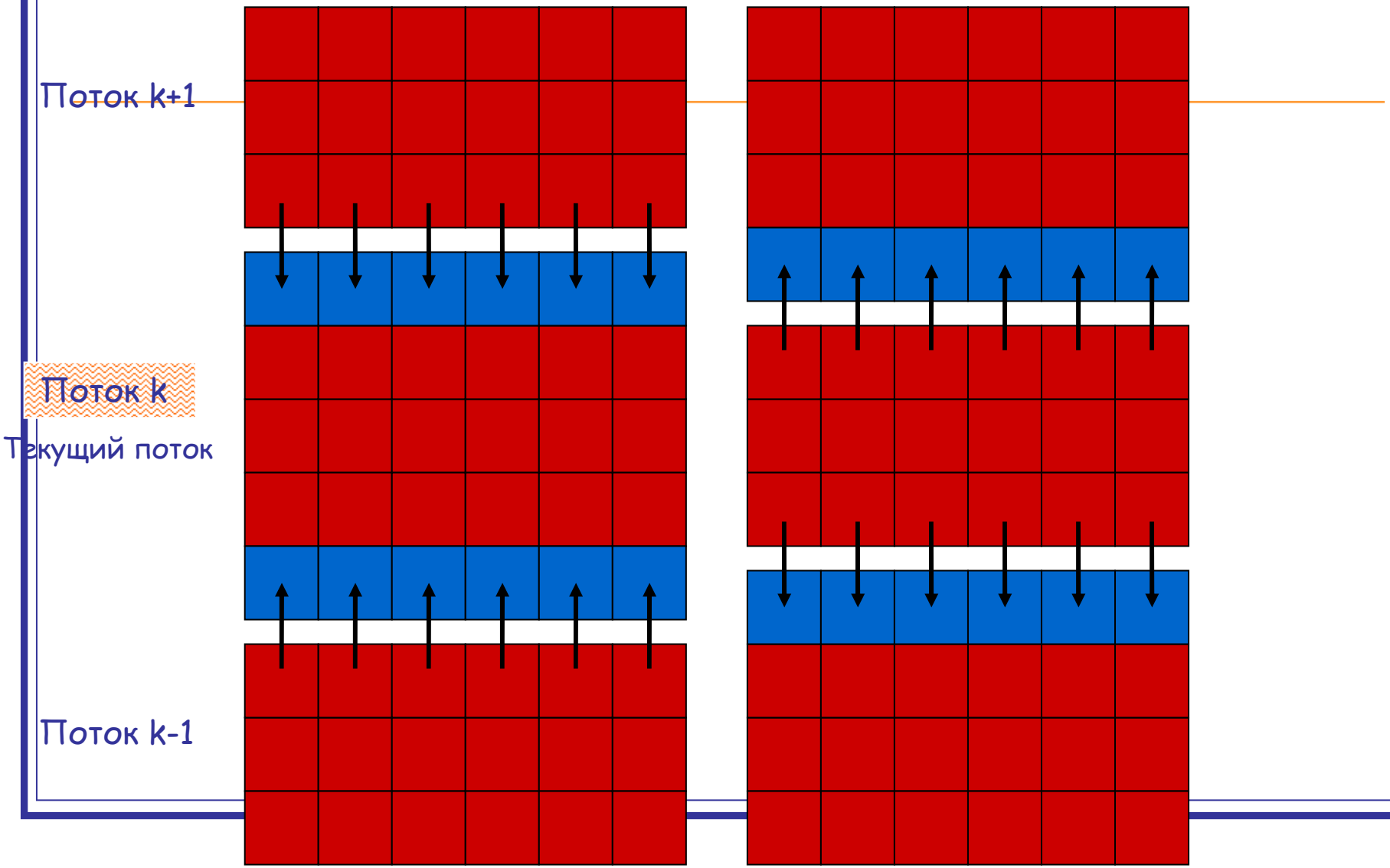


Схема передачи данных



Параллельное программирование для многопроцессорных систем. MPI.

Основные возможности.

Литература

- **MPI: A Message-Passing Interface Standard**
  http://www.mpi-forum.org/docs/
  (http://parallel.ru/docs/Parallel/mpi1.1/mpi-report.html)
- **Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. -М.: Изд-во МГУ, 2004.-71 с.**
- **Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем систем.- СПб, БХВ-Петербург**
- **Интернет ресурсы:**
- https://computing.llnl.gov/tutorials/mpi, **www.parallel.ru, intuit.ru, www.mpi-forum.org, www.open-mpi.org, www.openmp.org**
- **http://www.mcs.anl.gov/research/projects/mpi/www/www3**

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, …)

- **Системы с распределенной памятью**
  - Программа состоит из параллельных процессов
  - Явное задание коммуникаций между процессами - “**Message Passing”**
  - Реализация - Message Passing библиотеки:
    - MPI (“Message Passing Interface”)
    - PVM (“Parallel Virtual Machine”) Shmem, MPT (Cray)

MPI

- MPI 1.1 Standard разрабатывался 92-94
- MPI 2.0 - 95-97
- MPI 2.1 - 2008
- MPI 3.0 – 2012
- Стандарты
  - **http://www.mcs.anl.gov/mpi**
  - **http://www.mpi-forum.org/docs/docs.html**

  Описание функций
  - **http://www-unix.mcs.anl.gov/mpi/www/**

Цель MPI

- Основная цель:
  - Обеспечение переносимости исходных кодов
  - Эффективная реализация
- Кроме того:
  - Большая функциональность
  - Поддержка неоднородных параллельных архитектур

Реализации MPI

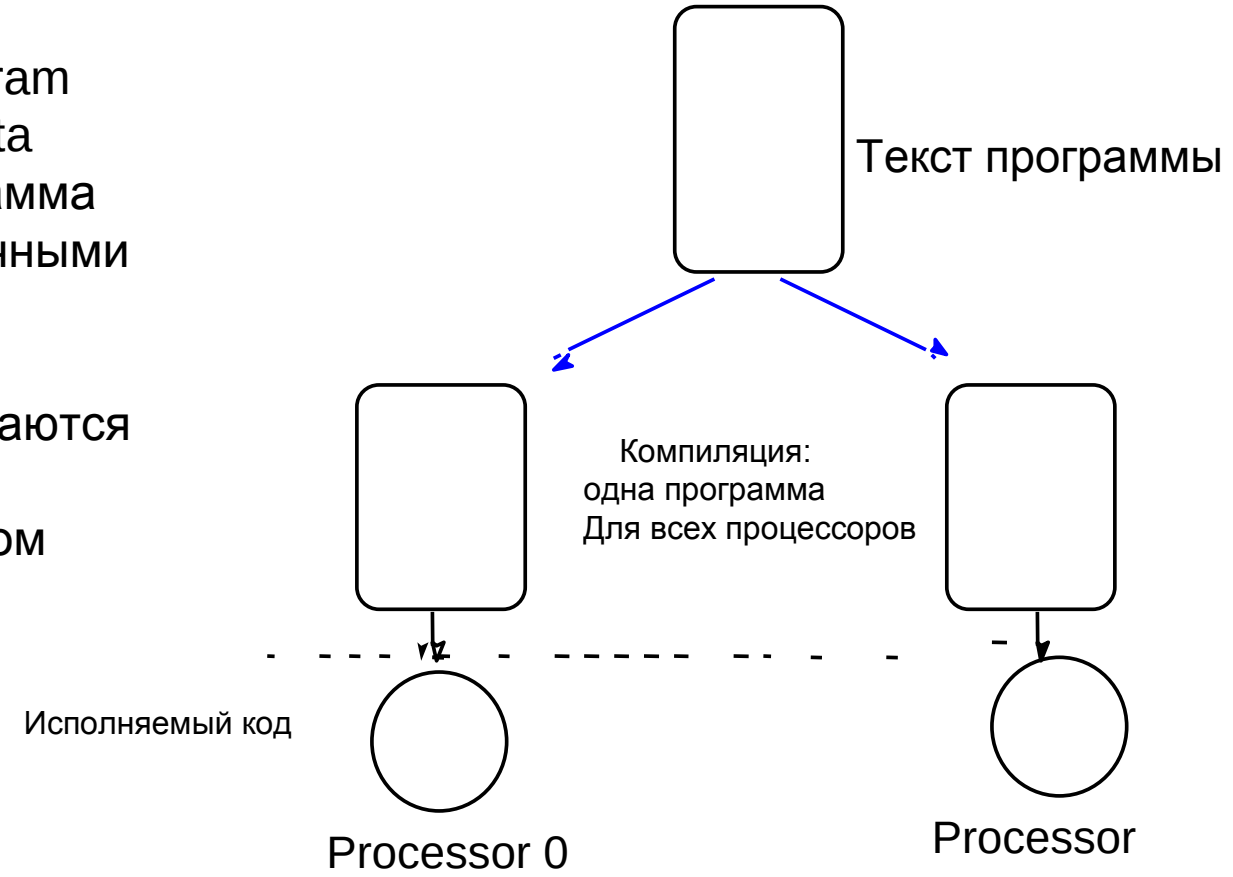
- MPICH
- LAM/MPI
- Mvapich
- OpenMPI
- Коммерческие реализации Intel,IBM и др.

Модель MPI

- 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.

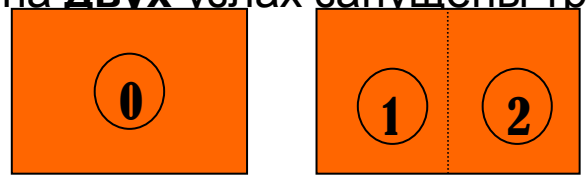
Модель MPI-программ

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



Модель выполнения MPI- программы

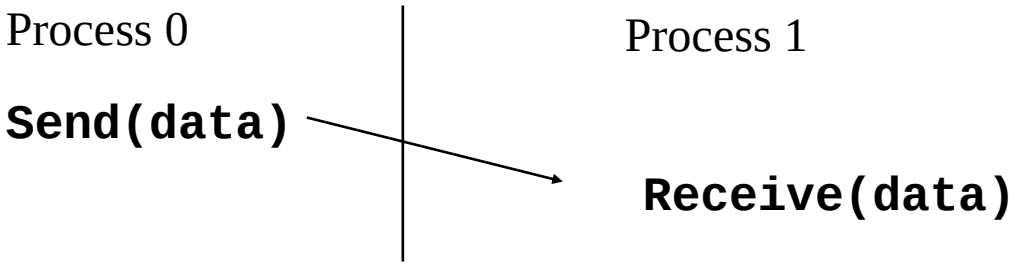
- Запуск: ***mpirun***
- При запуске указываем число требуемых процессоров **np** и название программы: пример: ***mpirun –np 3 prog***
- На выделенных узлах запускается ***np*** копий (процессов) указанной программы
  - Например, на **двух** узлах запущены три копии программы.



- Каждый процесс MPI-программы получает два значения:
  - ***np –*** *число процессов*
  - ***rank*** из диапазона [0 … ***np-1***] – номер процесса
- Любые два процесса могут непосредственно обмениваться данными с помощью функций передачи сообщений

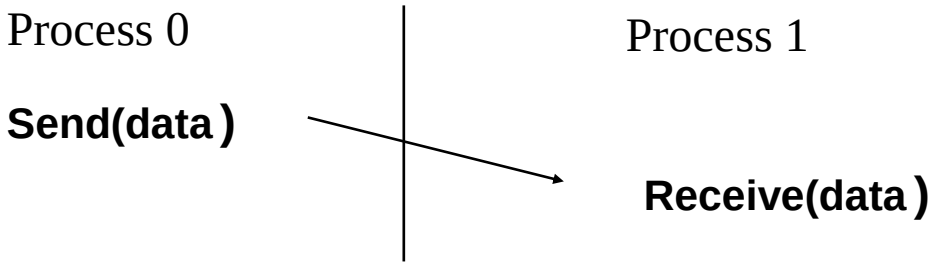
Основы передачи данных в MPI

- Технология передачи данных MPI предполагает кооперативный обмен.
- Данные посылаются одним процессом и принимаются другим.
- Передача и синхронизация совмещены.



Основы передачи данных в MPI

- Необходимы уточнения процесса передачи



- Требуется уточнить:
  - Как должны быть описаны данные ?
  - Как должны идентифицироваться процессы?
  - Как получатель получит информацию о сообщении?
  - Что значить завершение передачи?

Основные понятия

- Процессы объединяются в *группы*.
- Каждое сообщение посылается в рамках некоторого контекста и должно быть получено в том же контексте.
- Группа и контекст вместе определяют коммуникатор.
- Процесс идентифицируется своим номером в группе, ассоциированной с коммуникатором.

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), где
- *datatype* определяется рекурсивно как :
  - предопределенный базовый тип, соответствующий типу данных в базовом языке (например, MPI_INT, MPI_DOUBLE_PRECISION)
  - Непрерывный массив MPI типов
  - Векторный тип
  - Индексированыый тип
  - Произвольные структуры
- MPI включает функции для построения пользовательских типов данных, например, типа данных, описывающих пары (int, float).

Базовые MPI-типы данных

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Специальные типы MPI

- MPI_Comm
- MPI_Status
- MPI_datatype

Понятие тэга

- 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.

C: MPI helloworld.c

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    printf("Hello, MPI world\n");
    MPI_Finalize();
    return 0; }
```

Формат MPI-функций

C (большие и маленькие буквы различаются):

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

C++ (case sensitive):

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

Имена констант – большими буквами. Например:

MPI_COMM_WORLD

Функции определения среды

```
int MPI_Init(int *argc, char ***argv)
```
должна первым вызовом, вызывается только один раз

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```
число процессов в коммуникаторе

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```
номер процесса в коммуникаторе (нумерация с 0)

```
int MPI_Finalize()
```
завершает работу процесса

```
int MPI_Abort (MPI_Comm_size(MPI_Comm comm,
int*errorcode)
```
завершает работу программы

Инициализация MPI

- ***MPI_Init*** должна быть первым вызовом, вызывается только один раз

C:

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

http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Init.html

Обработка ошибок MPI-функций

Определяется константой ***MPI_SUCCESS***

```
int error;
……
error = MPI_Init(&argc, &argv));
If (error != MPI_SUCCESS)
{
 fprintf (stderr, “ MPI_Init error \n”);
return 1;

}
```

MPI_Comm_size

Количество процессов в коммуникаторе

- **Размер коммуникатора**

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

Результат – число процессов

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

MPI_Comm_rank
номер процесса (process rank)

- Process ID в коммуникаторе
  - Начинается с 0 до (*n*-1), где *n* – число процессов
- Используется для определения номера процесса-отправителя и получателя

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

Результат – номер процесса

Завершение MPI-процессов

- Никаких вызовов MPI функций после

C:

int MPI_Finalize()

int MPI_Abort (MPI_Comm_size(MPI_Comm comm, int*errorcode)

Если какой-либо из процессов не выполняет MPI_Finalize, программа зависает.

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

Трансляция MPI-программ на ВС ***Регатта***

- Трансляция

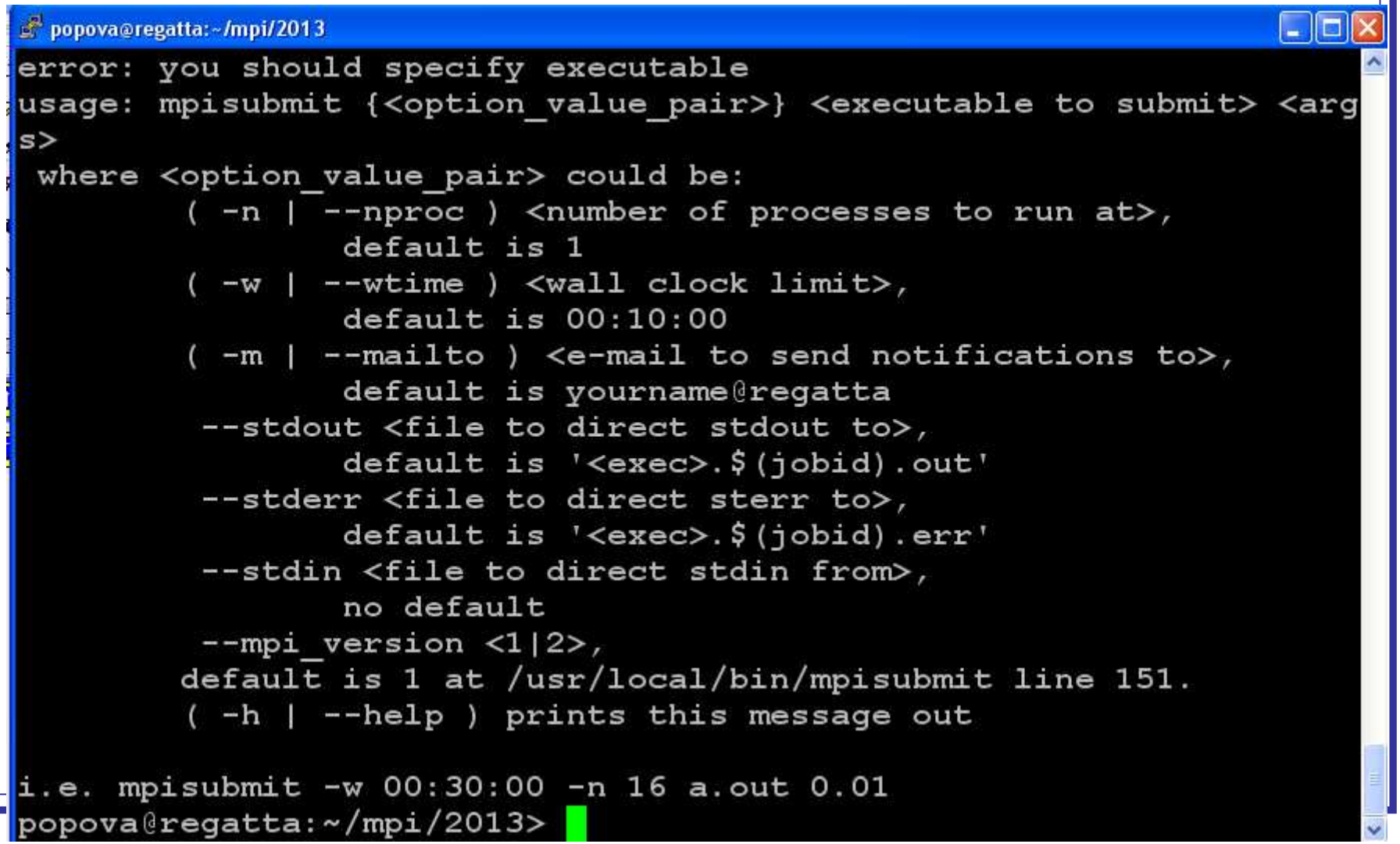
  mpicc –o hw helloworld.c

- Постановка в очередь на выполнение

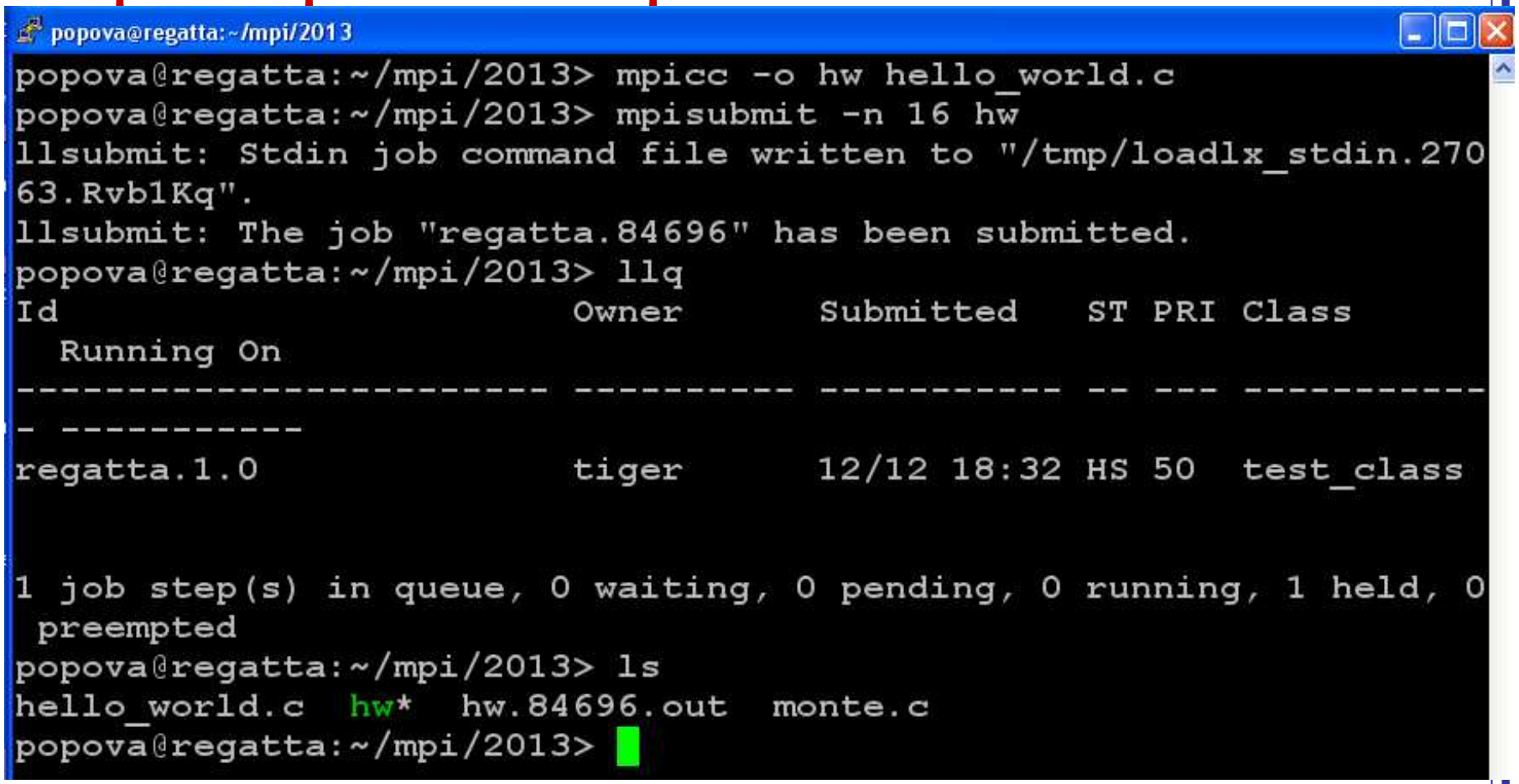
  mpisubmit –n 128 hw

  mpisubmit -help

mpisubmit -help



Пример сессии работы на Регатте




popova@regatta:~/mpi/2013
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
Hello,world!
"hw.84696.out" 16L, 208C
1,1
All

График сдачи заданий

- Задание 1 (OpenMP, ***Регатта***)
- Срок: **15 октября 2013**
- Задание 2. 2D Дирихле (MPI, ***Регатта, BGP, Ломоносов***)
- Срок: **15 ноября 2013**
- Задание 3*. 3D (MPI,MPI/OpenMP, ***BGP, Ломоносов***)
- Возможно индивидуальные задания.
- Срок: **15 декабря 2013**

Взаимодействие «точка-точка»

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

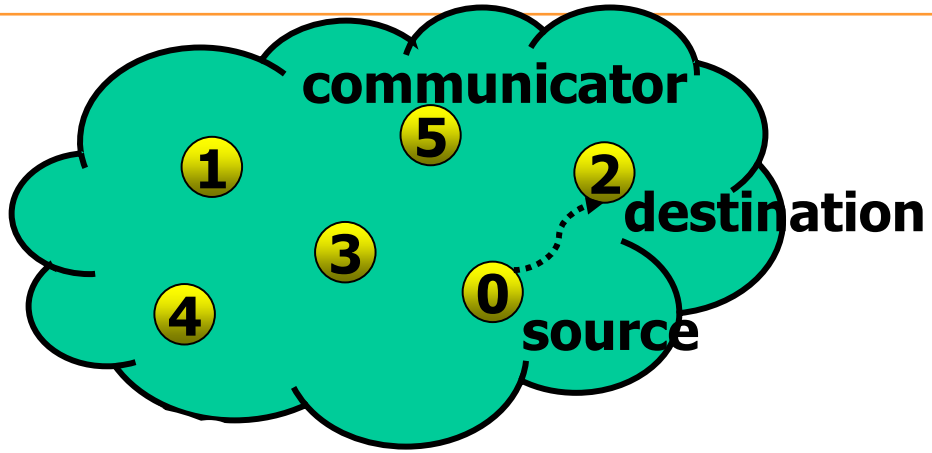
Варианты передачи «точка-точка»

- Синхронные пересылки
- Асинронные передачи
- Блокирующие передачи
- Неблокирующие передачи

Функции MPI передачи «точка-точка»

Point-to-Point Communication Routines		
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Start	MPI_Startall
MPI_Test	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome

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

Завершение

- “Завершение” передачи означает, что буфер в памяти, занятый для передачи, может быть безопасно использован для доступа, т.е.
  - Send: переменная, задействованная в передаче сообщения, может быть доступна для дальнейшей работы
  - Receive: переменная, получающая значение в результате передачи, может быть использована

2 типа операций передачи сообщений: блокирующие и неблокирующие

- Определяют, при каких условиях операции передачи завершаются
  - **Блокирующие**: возврат из функций передачи сообщений только по завершению коммуникаций
  - **Неблокирующие** (асинхронные): немедленный возврат из функций, пользователь должен контролировать завершение передач

Фунции передачи сообщений «точка-точка» (блокирующие)

Режим (MODE)	MPI функции
Standard send	MPI_Send
Synchronous send	MPI_Ssend
Buffered send	MPI_Bsend
Ready send	MPI_Rsend
Receive	MPI_Recv