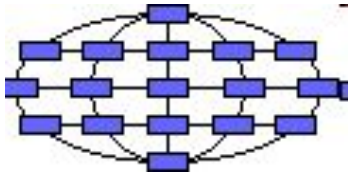


Введение в параллельное программирование

Максим Городничев

Летняя школа отдела МО ВВС ИВМиМГ СО РАН

http://ssd.sscc.ru

Содержание лекции

1) Формальный подход к определению параллельной программы
2) Меры качества параллельных программ
3) Предел ускорения вычислений при распараллеливании
4) Реализация параллельных программ в модели общей памяти и в модели распределенной памяти
5) Программирование в распределенной памяти: MPI
6) Пример задачи: численное решение уравнения Пуассона
7) Работа на вычислительных системах Сибирского суперкомпьютерного центра

По ходу лекции: комментарии об ошибках, характерных для параллельного программирования (дедлоки, несогласованный доступ к данным)

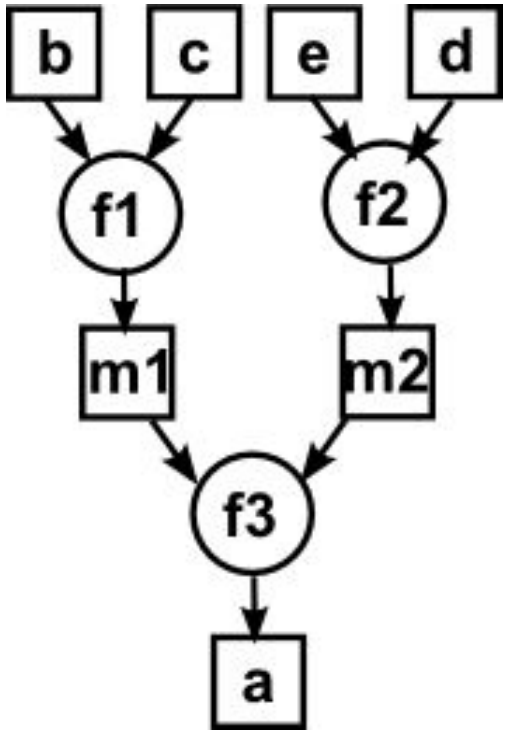
Последовательное и параллельное

Задача: a = b*c+d*e;

Последовательная программа:

Load r1,b
Load r2, c
Mul r1, r1, r2
Load r2, d
Load r3, d
Mul r2, r2, r3
Sum r1, r1, r2
Store a, r1

Алгоритм задает необходимый *частичный порядок (*потоковое управление*)*:



Представление алгоритма A, вычисляющего функцию F

Представление алгоритма A – это набор
S = (X, F, C, M), где

X – конечное множество переменных

F – конечное множество операций

C – управление (множество ограничений на порядок операций) = потоковое управление + прямое управление

M – функция, задающая отображение множеств X и F в физические устройства параллельной вычислительной машины

Реализация алгоритма A, представленного в форме S

– это выполнение операций в некотором порядке, не противоречащим управлению C

Представление корректно, если любая реализация вычисляет функцию F, т.е. C содержит потоковое управление.

Последовательное и параллельное: резюме

Если множество реализаций алгоритма A, представленного в форме S содержит более одной реализации, то представление S называется **параллельным**.

Если это множество одноэлементно, то представление S назвается **последовательным**.

Меры качества параллельной реализации

Ускорение параллельной реализации относительно последовательной:

$S = T_s/T_p$, где T_s – время выполнения последовательной программы, T_p – параллельной

Эффективность использования N процессоров относительно одного

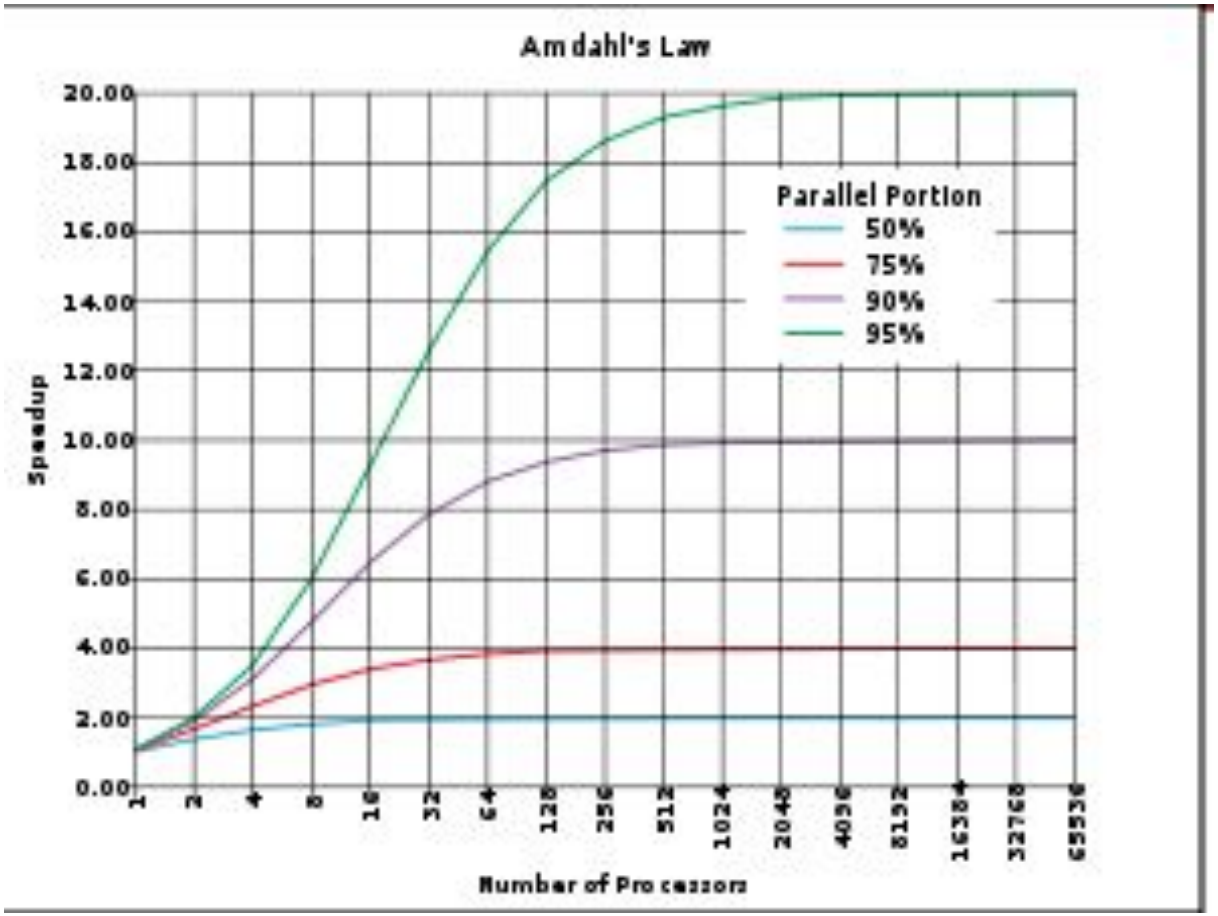
$E = T_s/(N*T_N)$, где T_N – время выполнения параллельной программы на N процессорах

Предел ускорения: Закон Амдала

Если P – это доля вычислений, которые могут быть выполнены параллельно, а 1-P – доля последовательной части, то максимальное ускорение, которое можно получить на N процессорах равно

$$\frac{1}{((1-P)+(P/N))}$$

Закон Амдала: диаграмма

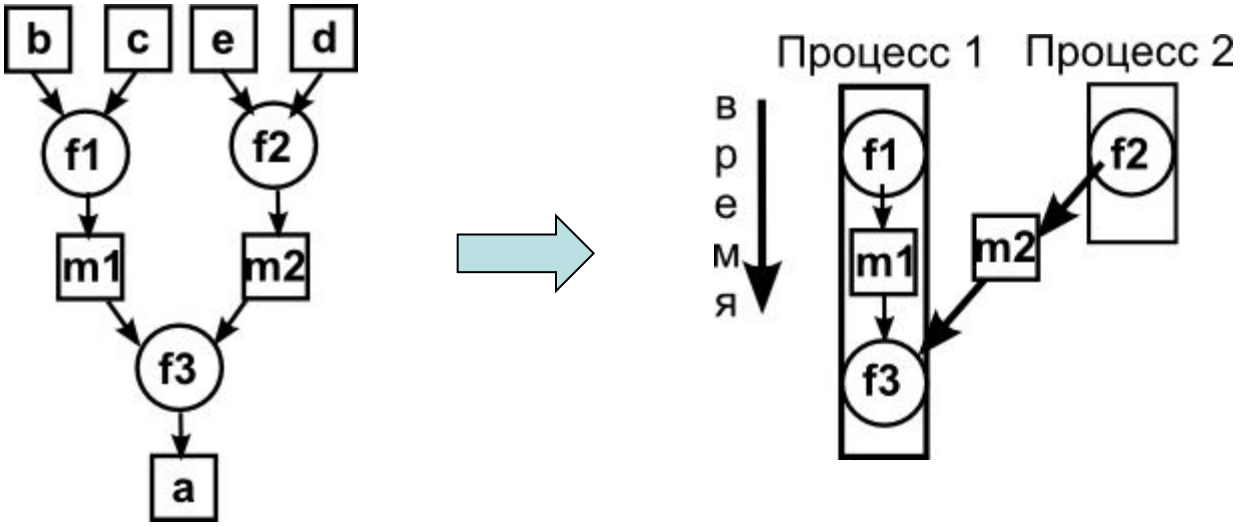


Изображение из Википедии

Иерархия параллельных вычислительных систем

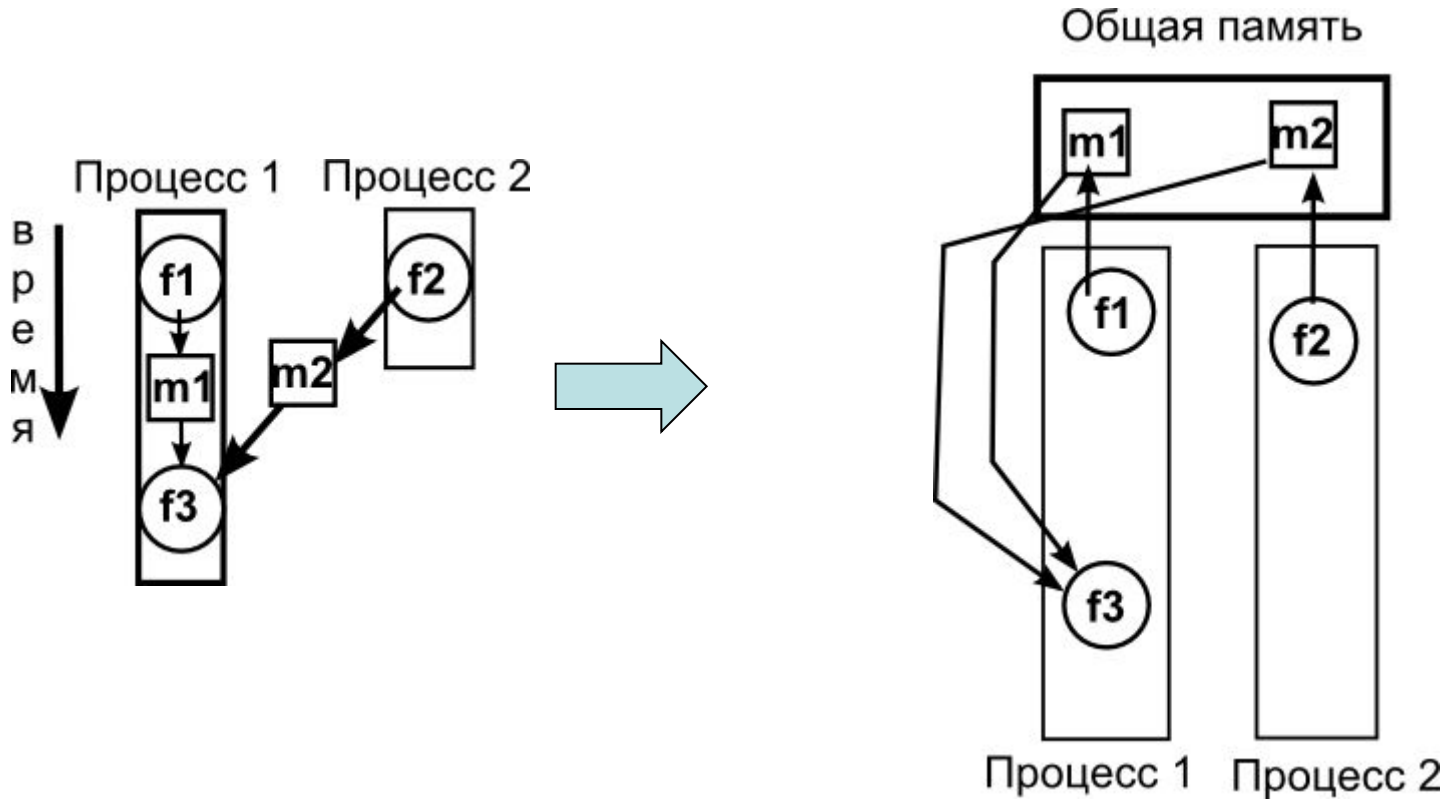
- Микропроцессоры
- SMP-узлы: объединения микропроцессоров над общим полем памяти
- Мультикомпьютеры: SMP-узлы, связанные выделенной сетью передачи данных
- Grid: объединение произвольных ресурсов

От алгоритма к программе: взаимодействующие процессы

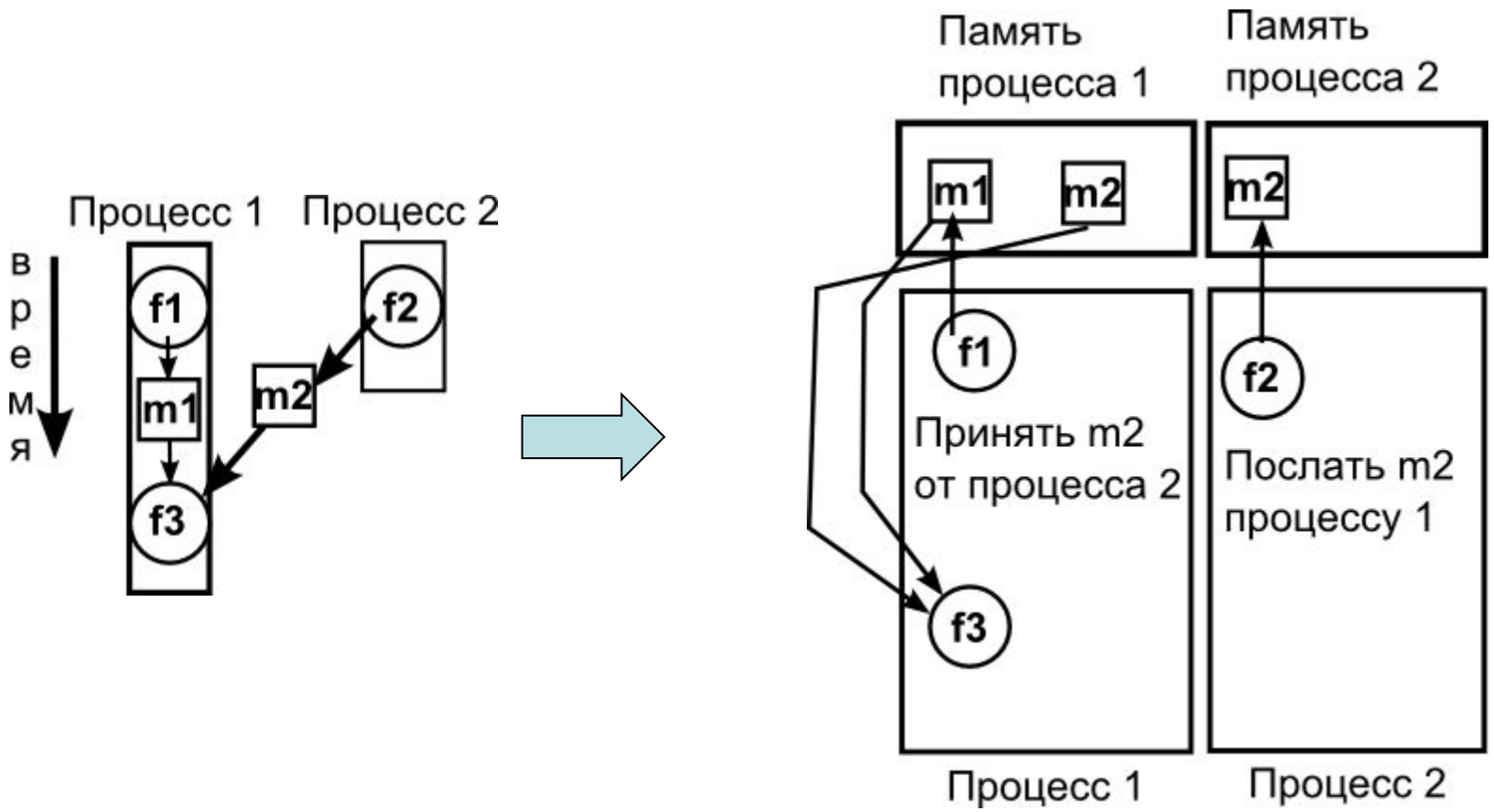


Программа часто задается как множество взаимодействующих процессов. Системы программирования отличаются средствами задания взаимодействия.

Модель программирования в общей памяти



Модель программирования в разделенной памяти: обмен сообщениями



Пример параллельной программы в модели общей памяти: OpenMP

```
#include <omp.h>
<…>
int main()
{
<…>
#pragma omp parallel shared(a, b, c) private(i)
{
#pragma omp for
    for (i = 0; i < N; i++)
      a[i] = b[i] + c[i];
 }
<…>
}
```

Параллельное программирование в модели разделенной памяти:

Message Passing Interface

Первый пример: идентификация процессов

```
#include <mpi.h>
#include <iostream>
#include <unistd.h>

int main(int argc, char** argv)
{
	int rank, size;
	char hostname[50];
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	gethostname(hostname, 50);
	std::cerr << "Rank = " << rank << ", hostname = "
		<< hostname << ", size = " << size << "\n";
	MPI_Finalize();
	return 0;
}
```

Пример команды запуска программы: mpirun –np 4 first_example.exe

Запускается 4 копии программы – 4 процесса, каждый получает свой идентификатор rank: 0, 1, 2, 3

Взаимодействия точка-точка: MPI_Send, MPI_Recv

```
if(size!=2){
   std::cerr << "2 processes required\n";
   MPI_Finalize();
   return 1;
}
int intBuff = 1234;
MPI_Status st;
if(rank==0){
   MPI_Send(&rank, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
   std::cerr << "Rank " << rank << ": initial value of intBuff = " << intBuff << "\n";
   MPI_Recv(&intBuff, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &st);
   std::cerr << "Rank " << rank << ": received from Rank 1 intBuff = " << intBuff << "\n";
}
else
   if(rank==1) {
      std::cerr << "Rank " << rank << ": initial value of intBuff = " << intBuff << "\n";
      MPI_Recv(&intBuff, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &st);
      std::cerr << "Rank " << rank << ": received from Rank 0 intBuff = " << intBuff << "\n";
      MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
   }
```

MPI_Sendrecv

```
intBuff = 1234;

std::cerr << "Rank " << rank << ": initial value of intBuff = " << intBuff << "\n";

MPI_Sendrecv(&rank, 1, MPI_INT, !rank, 0,
                    &intBuff, 1, MPI_INT, !rank, 0, MPI_COMM_WORLD, &st);
std::cerr << "Rank " << rank << ": received from Rank "
          << (!rank) << " intBuff = " << intBuff << "\n";
```

Групповые взаимодействия: MPI_Bcast, MPI_Barrier

```
int intBuff = rank;
std::cerr << "Rank = " << rank << ", intBuff = " << intBuff << "\n";

MPI_Bcast(&intBuff, 1, MPI_INT, 0, MPI_COMM_WORLD);

std::cerr << "Rank = " << rank << ", after Bcast intBuff = " << intBuff << "\n";

//example of another scenario

MPI_Barrier(MPI_COMM_WORLD);
if(rank==0)
   std::cerr << "---- Another scenario ----\n";
MPI_Barrier(MPI_COMM_WORLD);

intBuff = rank;
std::cerr << "Rank = " << rank << ", again intBuff = rank = " << intBuff << "\n";
if(rank==0)
   MPI_Bcast(&rank, 1, MPI_INT, 0, MPI_COMM_WORLD);
else
   MPI_Bcast(&intBuff, 1, MPI_INT, 0, MPI_COMM_WORLD);
std::cerr << "Rank = " << rank << ", after second Bcast intBuff = " << intBuff << "\n";
```

MPI_Reduce, MPI_Allreduce

```
int intBuff = 0;
std::cerr << "Rank = " << rank << ", intBuff = "
          << intBuff << "\n";
MPI_Reduce(&rank, &intBuff, 1, MPI_INT,
                  MPI_SUM, 0, MPI_COMM_WORLD);
std::cerr << "Rank = " << rank
          << ", after Reduce intBuff = " << intBuff << "\n";

intBuff = 0;

MPI_Allreduce(&rank, &intBuff, 1,
                    MPI_INT, MPI_SUM, MPI_COMM_WORLD);
std::cerr << "Rank = " << rank
          << ", after Allreduce intBuff = " << intBuff << "\n";
```

Асинхронные взаимодействия точка-точка: MPI_Isend, MPI_Iresv

```
int intBuff = rank;
MPI_Request rq;
MPI_Isend(&intBuff, 1, MPI_INT, !rank, 0, MPI_COMM_WORLD, &rq);

for(int i = 0; i<100; i++)
    std::cout << "working\n";

int flag = 0;
MPI_Status st;
MPI_Wait(&rq, &st);
//we can write to intBuff again: receiving into intBuff now.
//we can use rq again.

MPI_Irecv(&intBuff, 1, MPI_INT, !rank, 0, MPI_COMM_WORLD, &rq);

for(int i = 0; i<100; i++)
    std::cout << "working again\n";

MPI_Wait(&rq, &st);

//now intBuff contains a message received from a peer
std::cerr << "Rank " << rank << ": intBuff = " << intBuff << "\n";
```

Другие функции

Сбор распределенных векторов: MPI_Gather, MPI_Allgather

Распределение вектора: MPI_Scatter

и т.д.

“The standard includes:”

(Section 1.7 of MPI 2.1 Standard)

- Point-to-point communication
- Datatypes
- Collective operations
- Process groups
- Communication contexts
- Process topologies
- Environmental Management and inquiry
- The info object
- Process creation and management
- One-sided communication
- External interfaces
- Parallel file I/O
- Language Bindings for Fortran, C and C++
- Profiling interface

Пример решения уравнения Пуассона явным методом

Двумерное уравнение:

$d^2u/dx^2+d^2u/dy^2-a*u=f$

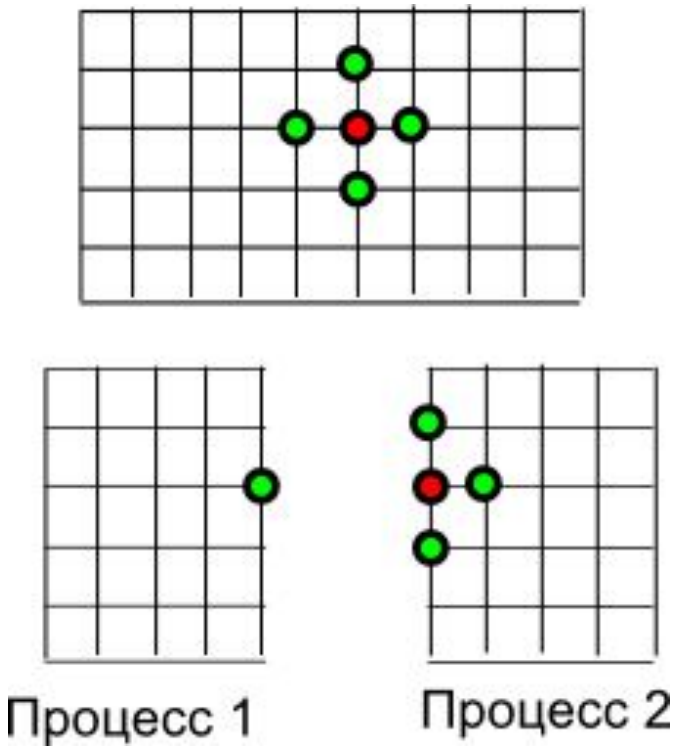
Дискретизация:

$(u_{(i+1)j}-2u_{ij}+u_{(i-1)j})/h^2+ (u_{i(j+1)}-2uij+u_{i(j-1)})/h^2-a*u_{ij} = f_{ij}$

Итеративный метод:

$u^{n+1}_{ij} = 0.25(u^n_{(i+1)j}+u^n_{(i-1)j}+u^n_{i(j+1)}+u^n_{i(j-1)}-h^2f_{ij})$

Декомпозиция задачи



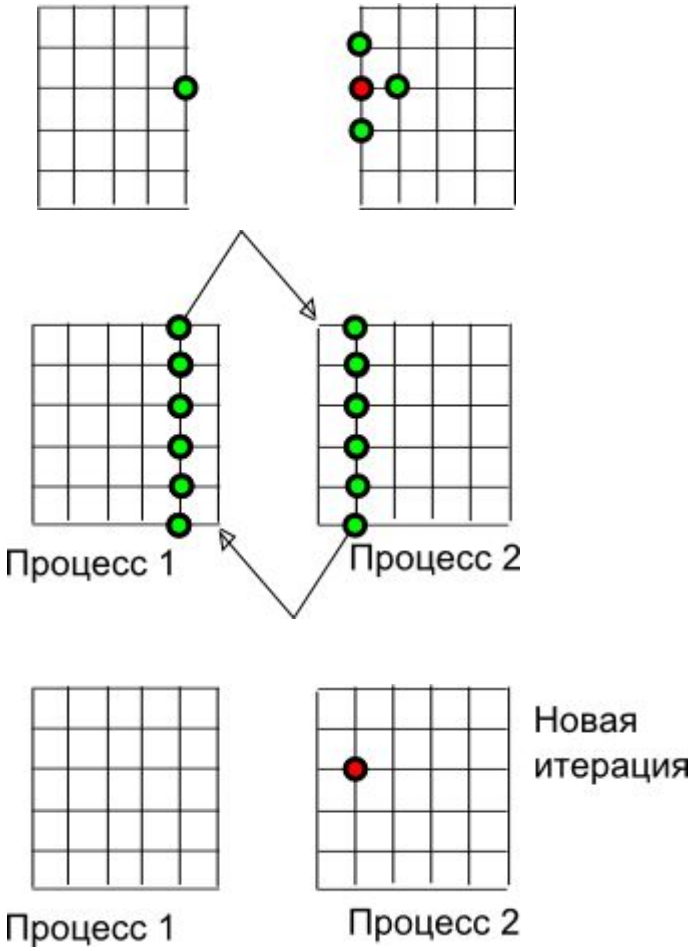
Как организовать коммуникации на разрезе?

Организация коммуникаций с помощью дополнительных слоев

Создается 2 массива в каждом процессе: для значений предыдущей итерации и новой.

Когда значения на границах посчитаны на предыдущей итерации, производится обмен, при этом значения от соседа помещаются в дополнительный слой того же массива.

Вычисляются значения на новой итерации с использованием массива предыдущей итерации.



О кластере НКС-30Т nks-30t.sscc.ru

32 двойных блейд-серверов HP BL2x220c, в каждом узле два 4-х ядерных процессора Intel Xeon E5450 (Intel(R) Xeon(R) CPU E5450 @ 3.00GHz)

= 512 ядер

Пиковая производительнось: ~6 ТФлопс.

Операционная система: RedHat Linux (RHEL 5u2)

Очередь заданий / система пакетной обработки: PBS Pro 10.0

Компиляторы: Intel C/C++ и Fortran 11.0.081 Professional Edition,
включают в себя Intel MKL, Intel IPP ...
Intel MPI 3.2.0.011 и Intel TraceAnalyzer&Collector 7.2.0.011

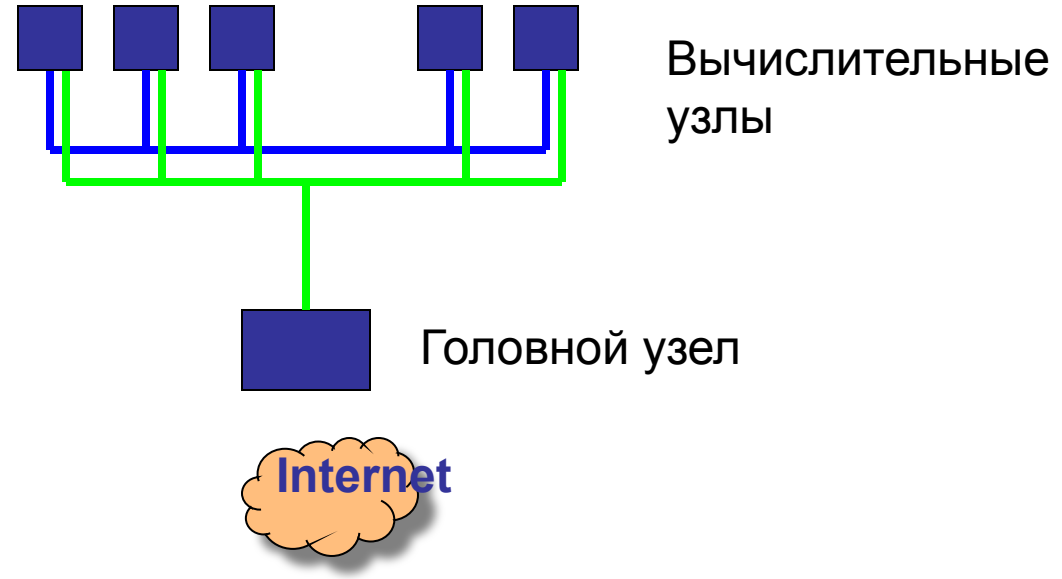
Работа на кластере НКС-30Т

1 Вход с помощью программы **putty.exe**:
1.1Имя сервера: **nks-30t.sscc.ru**
1.2 Логин:
1.3 Пароль:
2. Панельный файловый менеджер: **mc**
3. Создать в домашней директории папку для своей работы
4. Скопировать себе примеры (содержимое папки ~/**examples**)
5. Запустить пример: **qsub run.sh**
6. Смотреть очередь: **qstat**
7. Удалить задачу из очереди: **qdel <jobID>**
8. Компиляция программы mpiexample.cpp:
   mpiicpc –o mpiexample.exe mpiexample.cpp
9. Компиляция программы mpiexample.c:
   mpiicc –o mpiexample.exe mpiexample.c

Схема устройства кластера

Высокоскоростная сеть

«Медленная» сеть для управления и устройства общей файловой системы



Устройство скрипта запуска задачи **run.sh**

```
#!/bin/bash
 #PBS -V
 #PBS -r n
#PBS -l nodes=<количество узлов>:ppn=<количество процессов на
    узле>,cput=00:10:00,walltime=15:00
#PBS -k oe
#PBS -N name_of_the_job
 #PBS -j oe
 date
 cd $PBS_O_WORKDIR
 pwd
 mpirun -r ssh -genv I_MPI_DEVICE rdma -genv
    I_MPI_RDMA_TRANSLATION_CACHE disable -n <количество
    процессов MPI>  <путь к исполняемому файлу>
 date
```

Где найти информацию?

MPI: http://www.mpi-forum.org/