# Параллелизм в вычислительной математике

(Занятие 4)

Игорь Николаевич Коньшин

МГУ - 04.07.2012

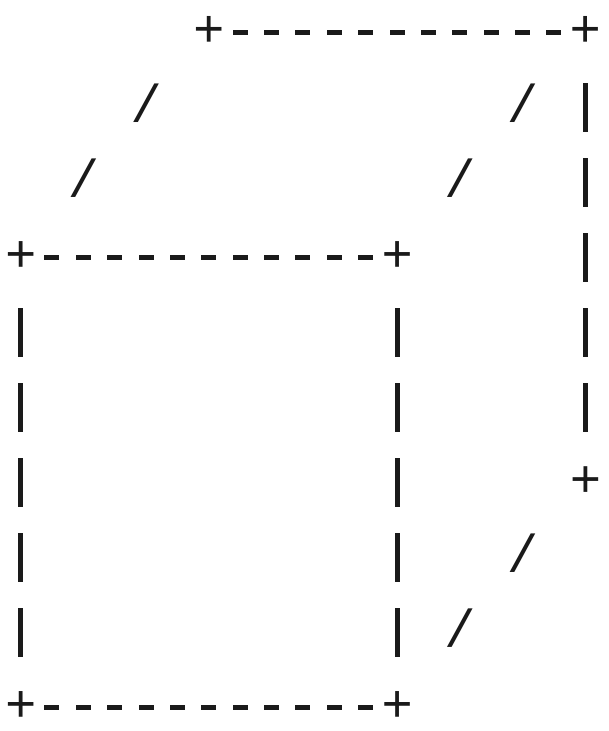
# // методы вычислительной математики

- Разбиение области по процессорам
- Метод разбиения области (DD, Domain Decomposition)

- СТАТИЧЕСКОЕ распараллеливание

- Вычисления в ячейках на новом временном слое независимы (явная схема решения)

# 3D расчетная область



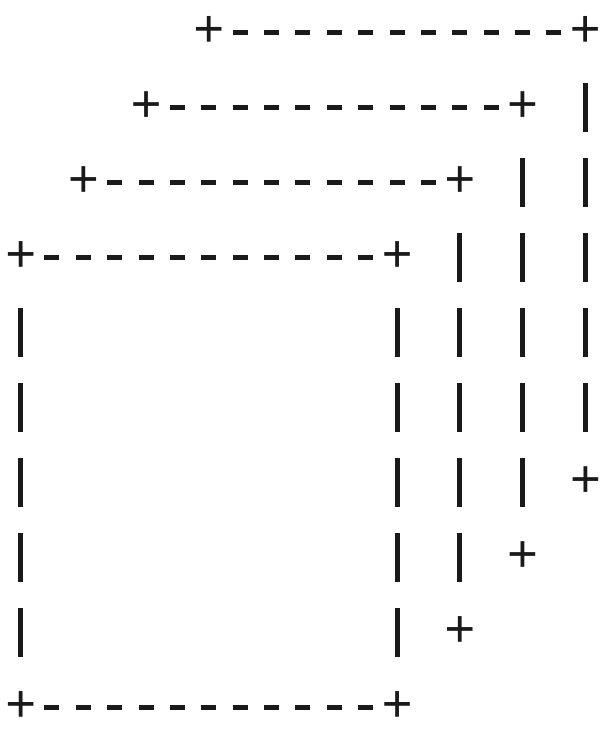
N = n*n*n,

V - количество неизвестных функций на узел (u,v,w,p,t)

C - среднее количество арифметических операций на узел

E = 1 / (1 + tau / L),  L = La / Lc

# 1D распределение данных



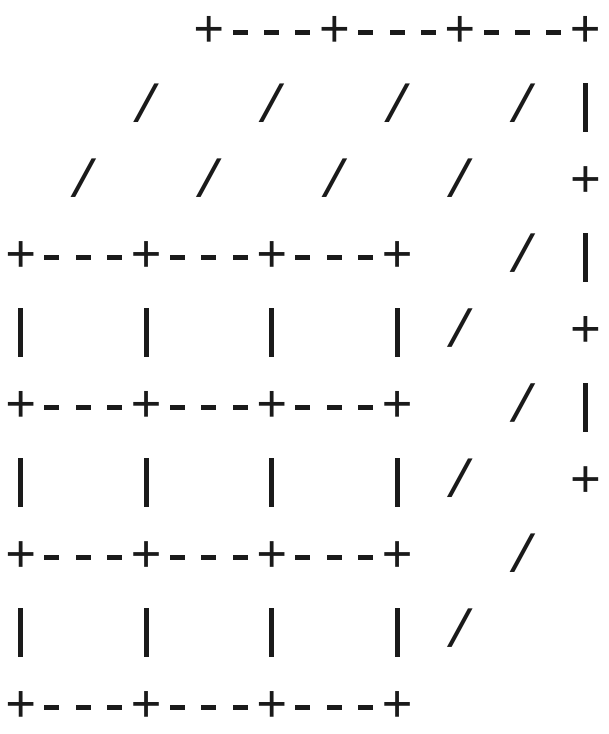
1DD: p=q

La = C N / p

Lc = 2 V n n

L = La / Lc = C n / (2 V q) ~ n/q

# 2D распределение данных



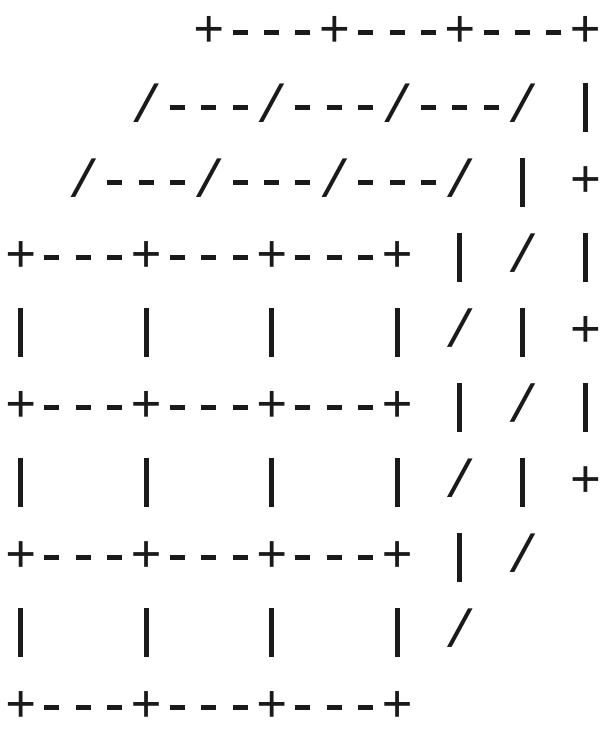
2DD: p=q*q

La = C N / p

Lc = 4 V n (n/q) = 4 V n n / q

L = La / Lc = C n / (4 V q) ~ n/q

# 3D распределение данных



3DD: p=q*q*q

La = C N / p

Lc = 6 V (n/q) (n/q) = 6 V n n / (q*q)

L = La / Lc = C n / (6 V q) ~ n/q

# // эффективность в %

N=n*n*n,  n=1000,  V=5,  C=30,  D=1,2,3,  tau=10

E = 1 / (1 + tau / L) =

= 1 / (1 + tau (2 D V q) / (C n)) ~ 1 / (1 + 0.0033 * D * p^(1/D))

| | 1DD | 2DD | 3DD |
|---|---|---|---|
| p=1 | 100 | 100 | 100 |
| p=10 | 97 | 98 | 98 |
| p=64 | 82 | 95 | 96 |
| p=729 | 29 | 85 | 92 |

# MPI: практика

- Как сделать чтобы выдача с процессов была последовательной (по порядку номеров процессоров)?

# MPI: замер времени

```
double MPI_Wtime( void )

int MPI_Barrier(
  MPI_Comm comm //- идентификатор группы
)

double t;
MPI_Barrier(MPI_COMM_WORLD);
t = MPI_Wtime();
::::::::: //- вычисления и/или обмены
MPI_Barrier(MPI_COMM_WORLD);
t = MPI_Wtime() - t;
```

# MPI: основные функции

```
int MPI_Init(...);        //- начало работы с MPI
int MPI_Finalize();   //- завершение работы с MPI
int MPI_Comm_size(...);      //- количество проц.
int MPI_Comm_rank(...);     //- номер проц.
int MPI_Send(...);    //- послать сообщение
int MPI_Recv(...);    //- получить сообщение
int MPI_Allreduce(...);     //- глобальная операция
int MPI_Barrier(...);      //- барьер
double MPI_Wtime();  //- замер времени
```

# MPI_Send / MPI_Recv

```
int MPI_Send(
  void* buf, //- адрес начала буфера посылки сообщения
  int count, //- число передаваемых элементов в сообщении
  MPI_Datatype datatype, //- тип передаваемых элементов
  int dest, //- номер процесса-получателя
  int msgtag, //- идентификатор сообщения
  MPI_Comm comm //- идентификатор группы
)
int MPI_Recv(
  void* buf, //- (OUT) адрес начала буфера приема сообщения
  int count, //- максимальное число элементов в принимаемом сообщении
  MPI_Datatype datatype, //- тип элементов принимаемого сообщения
  int source, //- номер процесса-отправителя
  int msgtag, //- идентификатор принимаемого сообщения
  MPI_Comm comm, //- идентификатор группы
  MPI_Status *status //- (OUT) параметры принятого сообщения
)
```

# MPI_Get_count

int MPI_Get_count(

MPI_Status *status, //- параметры принятого сообщения

MPI_Datatype datatype, //- тип элементов принятого сообщения

int *count //- (OUT) число элементов сообщения

)

- По значению параметра status данная подпрограмма определяет число уже принятых (после обращения к MPI_Recv) или принимаемых (после обращения к MPI_Probe или MPI_Iprobe) элементов сообщения типа datatype.

# MPI_Probe

```
int MPI_Probe(
  int source, //- номер процесса-отправителя или MPI_ANY_SOURCE
  int msgtag, //- идентификатор ожидаемого сообщения или MPI_ANY_TAG
  MPI_Comm comm, //- идентификатор группы
  MPI_Status *status //- (OUT) параметры обнаруженного сообщения
)
```

- Получение информации о структуре ожидаемого сообщения с блокировкой. Возврата из подпрограммы не произойдет до тех пор, пока сообщение с подходящим идентификатором и номером процесса-отправителя не будет доступно для получения. Атрибуты доступного сообщения можно определить обычным образом с помощью параметра status. Следует обратить внимание, что подпрограмма определяет только факт прихода сообщения, но реально его не принимает.

# MPI_Isend

```
int MPI_Isend(
  void *buf, //- адрес начала буфера посылки сообщения
  int count, //- число передаваемых элементов в сообщении
  MPI_Datatype datatype, //- тип передаваемых элементов
  int dest, //- номер процесса-получателя
  int msgtag, //- идентификатор сообщения
  MPI_Comm comm, //- идентификатор группы
  MPI_Request *request //- (OUT) идентификатор асинхронной передачи
)
```

- Передача сообщения, аналогичная MPI_Send, однако возврат из подпрограммы происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в буфере buf. Это означает, что нельзя повторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. того момента, когда можно переиспользовать буфер buf без опасения испортить передаваемое сообщение) можно определить с помощью параметра request и процедур MPI_Wait и MPI_Test.
- Сообщение, отправленное любой из процедур MPI_Send и MPI_Isend, может быть принято любой из процедур MPI_Recv и MPI_Irecv.

# MPI_Irecv

int MPI_Irecv(

void *buf, //- (OUT) адрес начала буфера приема сообщения

int count, //- максимальное число элементов в принимаемом сообщении

MPI_Datatype datatype, //- тип элементов принимаемого сообщения

int source, //- номер процесса-отправителя

int msgtag, //- идентификатор принимаемого сообщения

MPI_Comm comm, //- идентификатор группы

MPI_Request *request //- (OUT) идентификатор асинхронного приема сообщения

)

- Прием сообщения, аналогичный MPI_Recv, однако возврат из подпрограммы происходит сразу после инициализации процесса приема без ожидания получения сообщения в буфере buf. Окончание процесса приема можно определить с помощью параметра request и процедур MPI_Wait и MPI_Test.

# MPI_Wait

int MPI_Wait(

MPI_Request *request, //- идентификатор асинхронного приема или передачи

MPI_Status *status //- (OUT) параметры сообщения

)

- Ожидание завершения асинхронных процедур MPI_Isend или MPI_Irecv, ассоциированных с идентификатором request. В случае приема, атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра status.

# MPI_Waitall

int MPI_Waitall(

int count, //- число идентификаторов

MPI_Request *requests, //- массив идентификаторов асинхронного приема или передачи

MPI_Status *statuses //- (OUT) параметры сообщений

)

- Выполнение процесса блокируется до тех пор, пока **все** операции обмена, ассоциированные с указанными идентификаторами, не будут завершены. Если во время одной или нескольких операций обмена возникли ошибки, то поле ошибки в элементах массива statuses будет установлено в соответствующее значение.

# MPI_Waitany

int MPI_Waitany(

int count, //- число идентификаторов

MPI_Request *requests, //- массив идентификаторов асинхронного приема или передачи

int *index, //- (OUT) номер завершенной операции обмена

MPI_Status *status //- (OUT) параметры сообщений

)

- Выполнение процесса блокируется до тех пор, пока **какая-либо** операция обмена, ассоциированная с указанными идентификаторами, не будет завершена. Если несколько операций могут быть завершены, то случайным образом выбирается одна из них. Параметр index содержит номер элемента в массиве requests, содержащего идентификатор завершенной операции.

# MPI_Waitsome

```
int MPI_Waitsome(
  int incount, //- число идентификаторов
  MPI_Request *requests, //- массив идентификаторов асинхронного приема или передачи
  int *outcount, //- (OUT) число идентификаторов завершившихся операций обмена
  int *indexes, //- (OUT) массив номеров завершившихся операции обмена
  MPI_Status *statuses //- (OUT) параметры завершившихся сообщений
)
```

- Выполнение процесса блокируется до тех пор, пока **по крайней мере одна** из операций обмена, ассоциированных с указанными идентификаторами, не будет завершена. Параметр outcount содержит число завершенных операций, а первые outcount элементов массива indexes содержат номера элементов массива requests с их идентификаторами. Первые outcount элементов массива statuses содержат параметры завершенных операций.

# MPI_Test

int MPI_Test(

MPI_Request *request, //- идентификатор асинхронного приема или передачи

int *flag, //- (OUT) признак завершенности операции обмена

MPI_Status *status //- (OUT) параметры сообщения

)

- Проверка завершенности асинхронных процедур MPI_Isend или MPI_Irecv, ассоциированных с идентификатором request. В параметре flag возвращает значение 1, если соответствующая операция завершена, и значение 0 в противном случае. Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра status.

# MPI_Test...

- int MPI_Testall(...)
- int MPI_Testany(...)
- int MPI_Testsome(...)
- int MPI_Iprobe(...)

# Объединение запросов: MPI_Send_init

```
int MPI_Send_init(
  void *buf, //- адрес начала буфера посылки сообщения
  int count, //- число передаваемых элементов в сообщении
  MPI_Datatype datatype, //- тип передаваемых элементов
  int dest, //- номер процесса-получателя
  int msgtag, //- идентификатор сообщения
  MPI_Comm comm, //- идентификатор группы
  MPI_Request *request //- (OUT) идентификатор асинхронной передачи
)
```

- Формирование запроса на выполнение пересылки данных. Все параметры точно такие же, как и у подпрограммы MPI_Isend, однако в отличие от нее пересылка не начинается до вызова подпрограммы MPI_Startall.

# Объединение запросов: MPI_Recv_init

```
int MPI_Recv_init(
  void *buf, //- (OUT) адрес начала буфера приема сообщения
  int count, //- число принимаемых элементов в сообщении
  MPI_Datatype datatype, //- тип принимаемых элементов
  int source, //- номер процесса-отправителя
  int msgtag, //- идентификатор сообщения
  MPI_Comm comm, //- идентификатор группы
  MPI_Request *request //- (OUT) идентификатор асинхронного приема
)
```

- Формирование запроса на выполнение приема данных. Все параметры точно такие же, как и у подпрограммы MPI_Irecv, однако в отличие от нее реальный прием не начинается до вызова подпрограммы MPI_Startall.

# MPI_Startall

MPI_Startall(

int count, //- число запросов на взаимодействие

MPI_Request *requests //- (OUT) массив идентификаторов приема/передачи

)

- Запуск всех отложенных взаимодействий, ассоциированных вызовами подпрограмм MPI_Send_init и MPI_Recv_init с элементами массива запросов requests. Все взаимодействия запускаются в режиме без блокировки, а их завершение можно определить обычным образом с помощью процедур MPI_Wait и MPI_Test.

# Список практических заданий (1)

Параметры // компьютера (nodes x ppn):

- максимальное число nodes
- максимальное число ppn
- максимальная память на 1 node
- максимальная память на 1 ppn

Обмены:

- внутри узла (ppn>1)
- между узлами (ppn=1)
- синхронные обмены "пинг-понг"
- синхронные обмены "чет./нечет."
- асинхронные обмены "по кольцу"
- асинхронные обмены "все всем"

# Список практических заданий (2)

Замер времени:

- время одного вызова MPI_Wtime()
- время одного вызова MPI_Barrier()
- время ta одной арифметической операции: y=a*x+y
- время tc передачи одного числа
- коэффициент // эф-ти компьютера: tau=tc/ta
- измерение латентности t0 передачи данных: Tc=t0+tc*Lc

Зависимость скорости обменов от:

- длины сообщения
- количества процессоров
- числа ppn
- типа передачи (синхр./асинхр.)