Библиотека MPI

Message Passing Interface

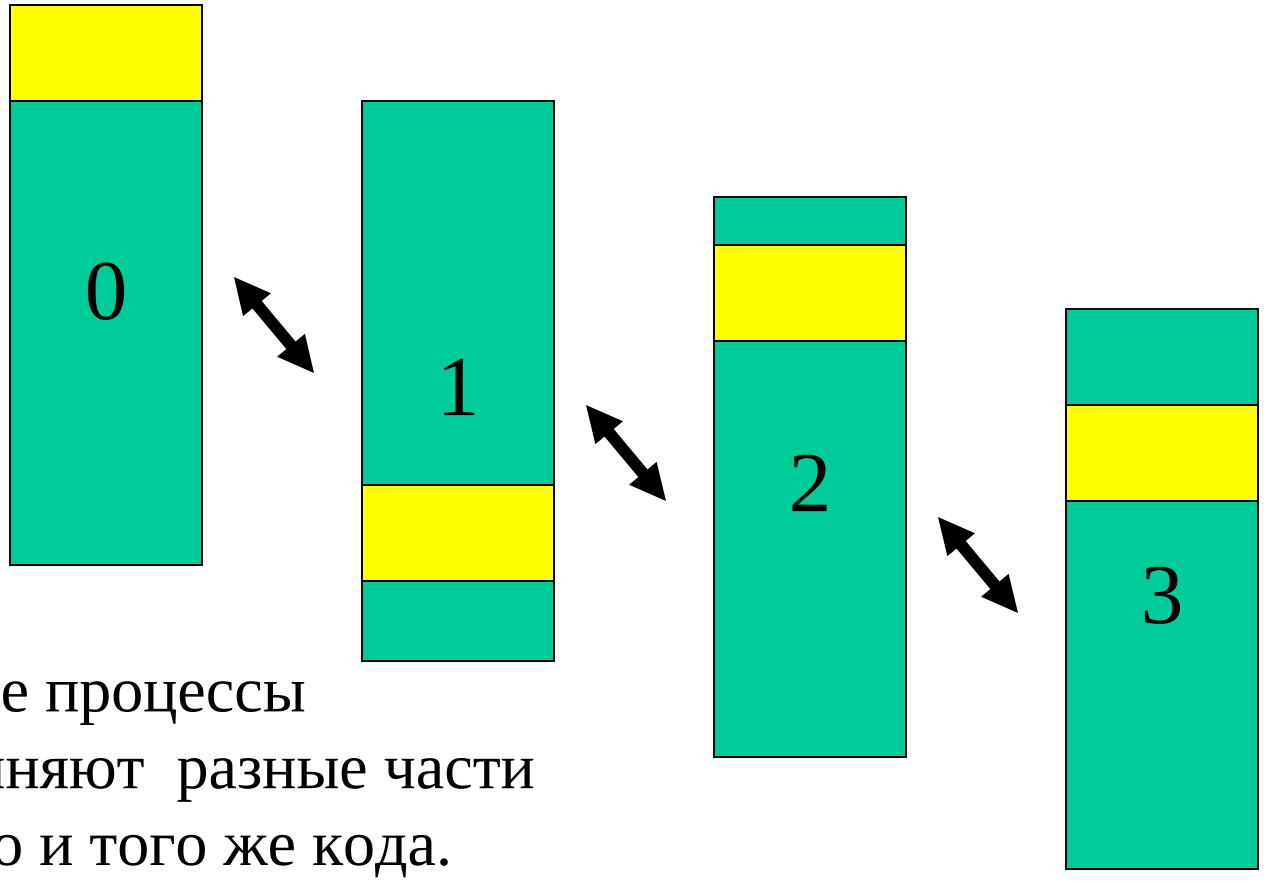
История MPI

Стандарт MPI 1.0 1995 год, MPI 2.0 1998 год. Определяет API (варианты для Си, C++, Fortran, Java).

«Комплект поставки» MPI

- Библиотека.
- Средства компиляции и запуска приложения.

SPMD-модель.



Разные процессы выполняют разные части одного и того же кода.

Сборка MPI-приложения.

Сборка MPI-приложения осуществляется с помощью специальной утилиты. В случае Си – **mpicc**. Пример:

```
mpicc –o mpihello mpihello.c
```

Запуск MPI-приложения осуществляется с помощью команды **mpirun.**

```
mpirun –np 4 mpihello
```

Функции инициализации и завершения работы.

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

argc – указатель на счетчик аргументов командной строки

argv – указатель на список аргументов

```
int MPI_Finalize()
```

Тоже простая MPI-программа

```
#include <mpi.h>
#include <stdio.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( "I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Функции определения ранга и числа процессов.

int MPI_Comm_size (MPI_Comm comm, int* size)

comm - коммуникатор
size – число процессов

int MPI_Comm_rank(MPI_Comm comm, int* rank)

comm – коммуникатор
rank – ранг процесса

Пример простейшей пересылки

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
  int rank;
  MPI_Status st;
  char buf[64];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if(rank == 0) {
    sprintf(buf, "Hello from processor 0");
    MPI_Send(buf, 64, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
  } else {
    MPI_Recv(buf, 64, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &st);
    printf("Process %d received %s \n", rank, buf);
  }
  MPI_Finalize();
}
```

Функции обменов точка-точка

```
int MPI_Send( buf, count, datatype, dest, tag, comm )

void *buf;                                  /* in */
int  count, dest, tag;                 /* in */
MPI_Datatype datatype;    /* in */
MPI_Comm comm;                /* in */
```

buf - адрес начала буфера посылаемых данных
count - число пересылаемых объектов типа, соответствующего datatype
dest - номер процесса-приемника
tag - уникальный тэг, идентифицирующий сообщение
datatype - MPI-тип принимаемых данных
comm - коммуникатор

```
int MPI_Recv( buf, count, datatype, source, tag, comm, status )

void *buf;                  /* in */
int count, source, tag;     /* in */
MPI_Datatype datatype;      /* in */
MPI_Comm comm;              /* in */
MPI_Status *status;         /* out */
```

buf - адрес буфера для приема сообщения
count - максимальное число объектов типа datatype, которое может быть записано в буфер
source - номер процесса, от которого ожидается сообщение
tag - уникальный тэг, идентифицирующий сообщение
datatype - MPI-тип принимаемых данных
comm - коммуникатор
status - статус завершения

MPI-программа численного интегрирования

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &r);
MPI_Comm_size(MPI_COMM_WORLD, &p);

if(r == 0)
  t = MPI_Wtime();

MPI_Barrier(MPI_COMM_WORLD);

sum = 0.0;
h = (b - a) / n;
for(i = r; i < n; i += p)
  sum += f(a + (i + 0.5) * h);

sum *= h;
MPI_Send(&sum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
sum = 0;
```

MPI-программа численного интегрирования

```
  if(r == 0) {
     double s;

     for(i = 0; i < p; i ++) {
       MPI_Recv(&s, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, &st);
       sum += s;
     }

     t = MPI_Wtime() - t;
     printf("Integral value = %lf.  Time = %lf sec.\n", sum, t);
   }

   MPI_Finalize();
}
```

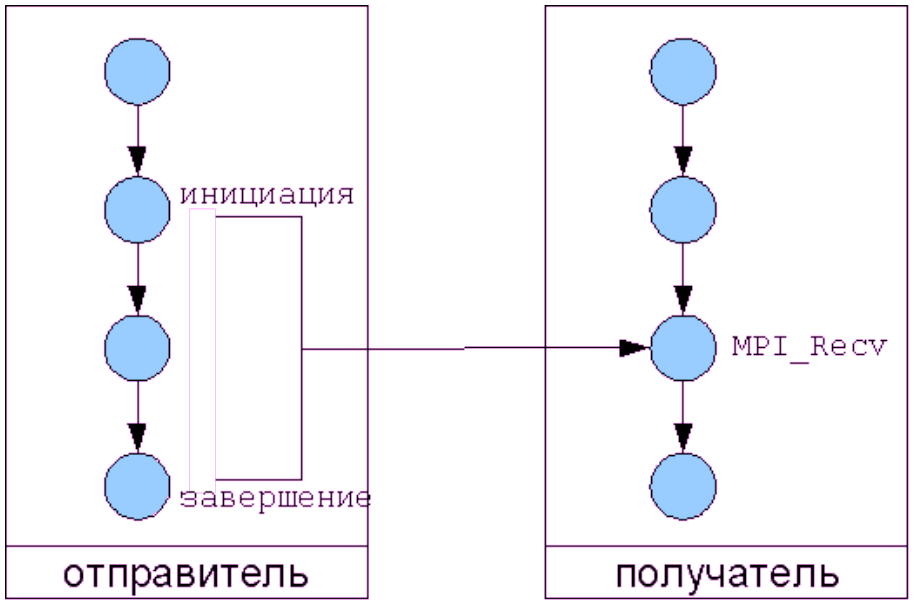
Виды точечных взаимодействий

MPI_Send	**блокирующая пересылка** функция возвращает управление тогда, когда исходный буфер можно освобождать (т.е. данные или скопированы в промежуточный или отправлены)
MPI_Bsend	**буферизованная пересылка** функция возвращает управление тогда, когда данные скопированы в буфер, выделяемый пользователем

MPI_Ssend	**синхронная пересылка** функция возвращает управление тогда, когда процесс-приемник преступил к выполнению соответствующей операции приема
MPI_Rsend	**интерактивная пересылка** поведение функции не определено, если соответствующая операция приема не начала выполнения (для увеличения производительности)

Неблокирующие пересылки

- Предназначены для перекрытия обменов и вычислений.
- Операция расщепляется на две: инициация и завершение.



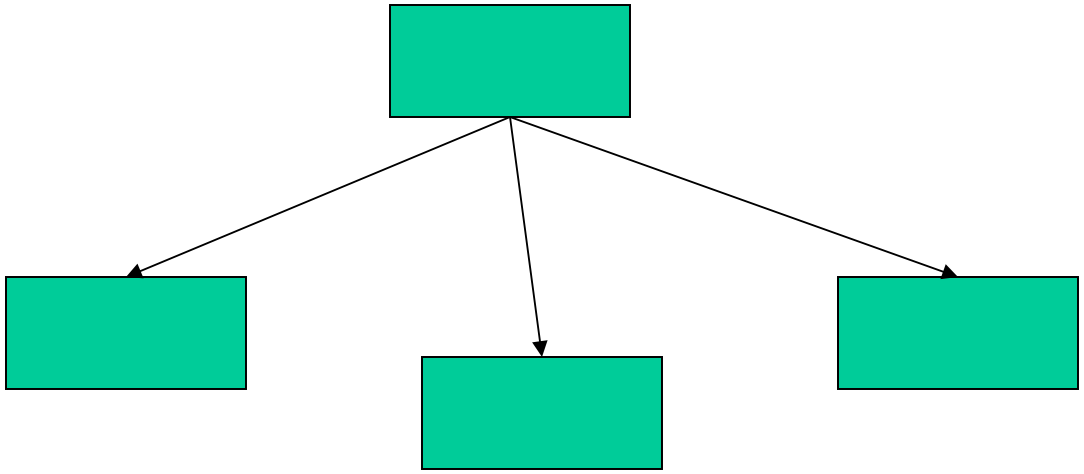
Завершение:

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

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

int MPI_Waitall(int count, MPI_Request array_of_requests[],
      MPI_Status array_of_statuses[] )

int MPI_Waitany(int count, MPI_Request array_of_requests[],
      int* index, MPI_Status *status )
```

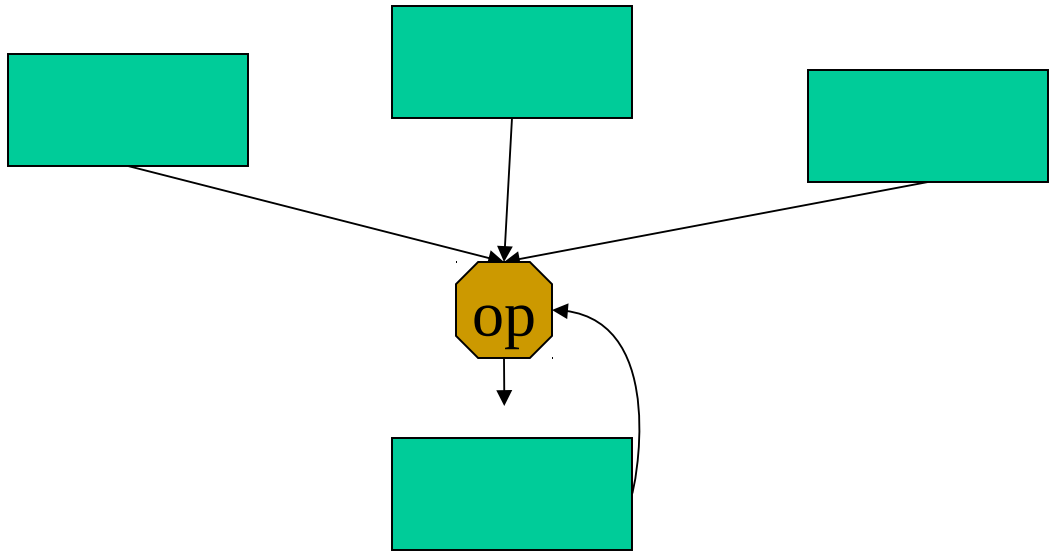
Коллективные взаимодействия процессов

int **MPI_Bcast** (buffer, count, datatype, root, comm)

void* buffer - начальный адрес буфера для передачи собщений
int count - число передаваемых элементов данных
MPI_Datatype datatype - тип передаваемых данных
int root - ранг процесса, посылающего данные
MPI_Comm comm - коммуникатор



```
int MPI_Reduce ( sendbuf, recvbuf, count,
      datatype, op, root, comm )

void  *sendbuf;    буфер операндов
void *recvbuf;     буфер приема
int  count;        число данных
MPI_Datatype datatype; тип данных
MPI_Op op;         операция
int root;          ранг процесса, содержащего результат
MPI_Comm comm;     коммуникатор
```



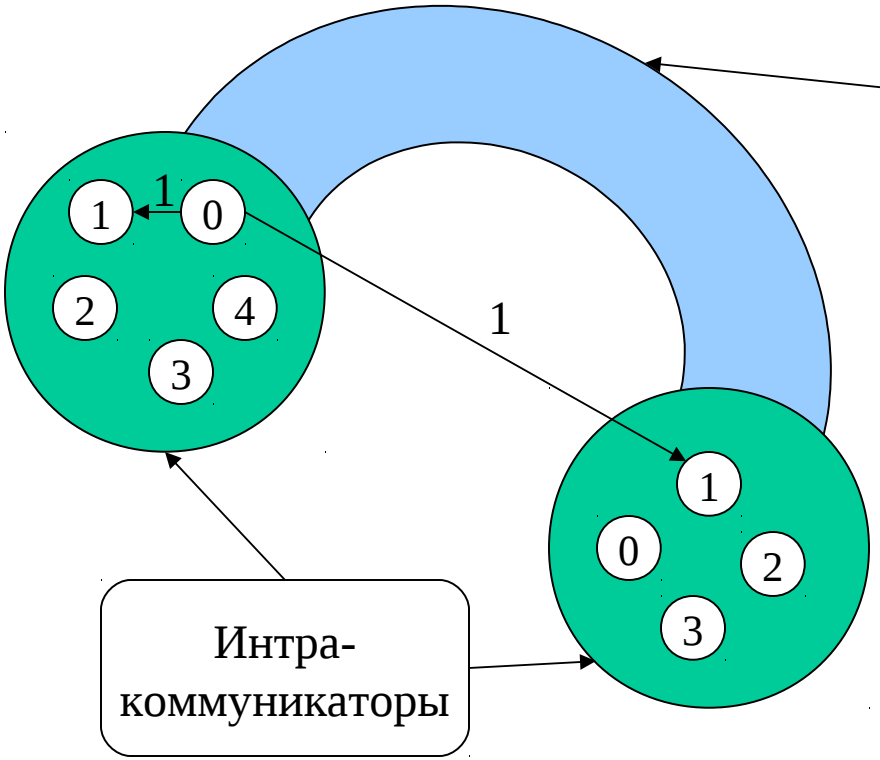
MPI_MAX	максимум
MPI_MIN	минимум
MPI_SUM	сумма
MPI_PROD	произведение
MPI_LAND	логическое "и"
MPI_BAND	побитовое "и"
MPI_LOR	логическое "или"
MPI_BOR	побитовое "или"
MPI_LXOR	логическое исключающее "или"
MPI_BXOR	побитовое исключающее "или"

Вычисление числа Pi

```
#include "mpi.h"
#include <math.h>

int main(argc,argv)
int argc;
char *argv[];
{
  int n, myid, numprocs, i;
  double PI25DT = 3.141592653589793238462643;
  double mypi, pi, h, sum, x, a;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
```

```
  while (1)
  {
    if (myid == 0) {
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d",&n);
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n == 0) break;

    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);
    }
    mypi = h * sum;
      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
                 MPI_COMM_WORLD);

    if (myid == 0)
        printf("pi is approximately %.16f, Error is %.16f\n",
               pi, fabs(pi - PI25DT));
  }
  MPI_Finalize();
}
```

Интер- и интра-коммуникаторы



Интра-коммуникаторы объединяют процессы из одной группы.

Интер-коммуникатор позволяет передавать данные между процессами из разных **интра**-коммуникаторов.

Интер-коммуникаторы не могут использоваться в коллективных взаимодействиях.

Назначение коммуникаторов

- Поддержка параллельных библиотек.
- Поддержка коллективных операций на части вычислительного пространства.
- Повышение уровня абстракции параллельных приложений.

Информационные функции для работы с группами

Определение размера группы:

int MPI_Group_size(MPI_Group group, int *size)

group – группа;

size – указатель на область памяти для записи информации о количестве процессов в группе;

Определение номера процесса, выполняющего вызов функции, в группе:

int MPI_Group_rank(MPI_Group group, int *rank)

group – группа;

rank – указатель на область памяти для сохранения номера процесса;

Разность двух групп:

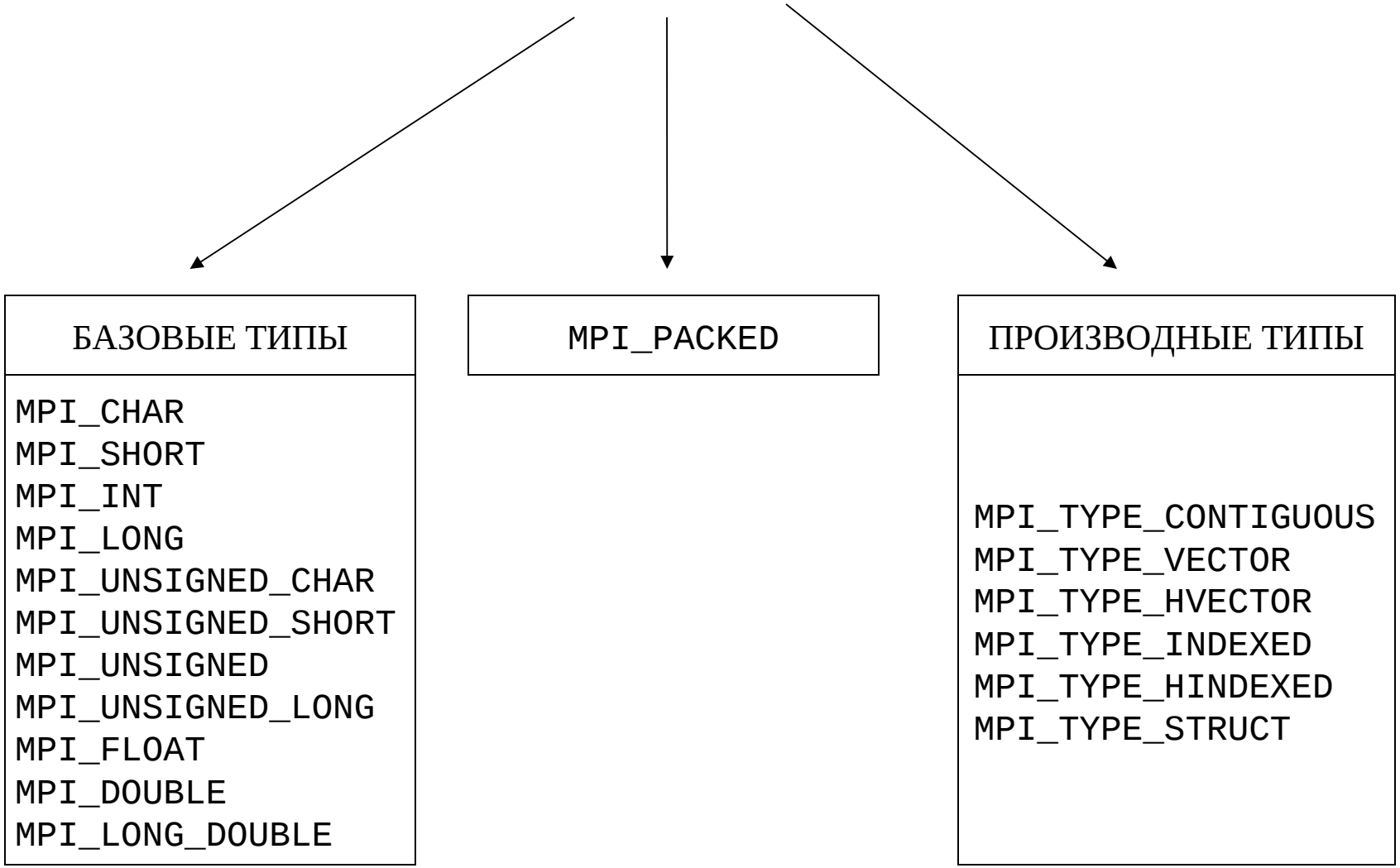
```
int MPI_Group_difference(MPI_Group gr1, MPI_Group
g2, MPI_Group* gr3)
      gr1 – первая группа;
      gr2 – вторая группа;
      gr3 – указатель на область для сохранения
результата операции;
Группа gr3 составлена из процессов, входящих в gr1,
но не входящих в gr2,  расположенных в том же
порядке, что и в gr1.
```

Переупорядочивание (с возможным удалением) процессов в существующей группе:

```
int MPI_Group_incl(MPI_Group* group, int n, int*
ranks, MPI_Group* newgroup)
      group – исходная группа;
      n – число элементов в массиве ranks;
      ranks – массив номеров процессов, из которых
будет создана новая группа;
      newgroup – указатель на область для сохранения
результата операции;
Созданная группа newgroup содержит элементы группы
group, перечисленные в массиве ranks: i-й процесс
создаваемой группы newgroup совпадает с процессом,
имеющим номер ranks[i] в группе group.
```

Система типов сообщений MPI

Типы в MPI


БАЗОВЫЕ ТИПЫ
MPI_CHAR
MPI_SHORT
MPI_INT
MPI_LONG
MPI_UNSIGNED_CHAR
MPI_UNSIGNED_SHORT
MPI_UNSIGNED
MPI_UNSIGNED_LONG
MPI_FLOAT
MPI_DOUBLE
MPI_LONG_DOUBLE
MPI_PACKED
ПРОИЗВОДНЫЕ ТИПЫ
MPI_TYPE_CONTIGUOUS
MPI_TYPE_VECTOR
MPI_TYPE_HVECTOR
MPI_TYPE_INDEXED
MPI_TYPE_HINDEXED
MPI_TYPE_STRUCT

БАЗОВЫЕ ТИПЫ

тип MPI	тип Си
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
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Назначение производных типов

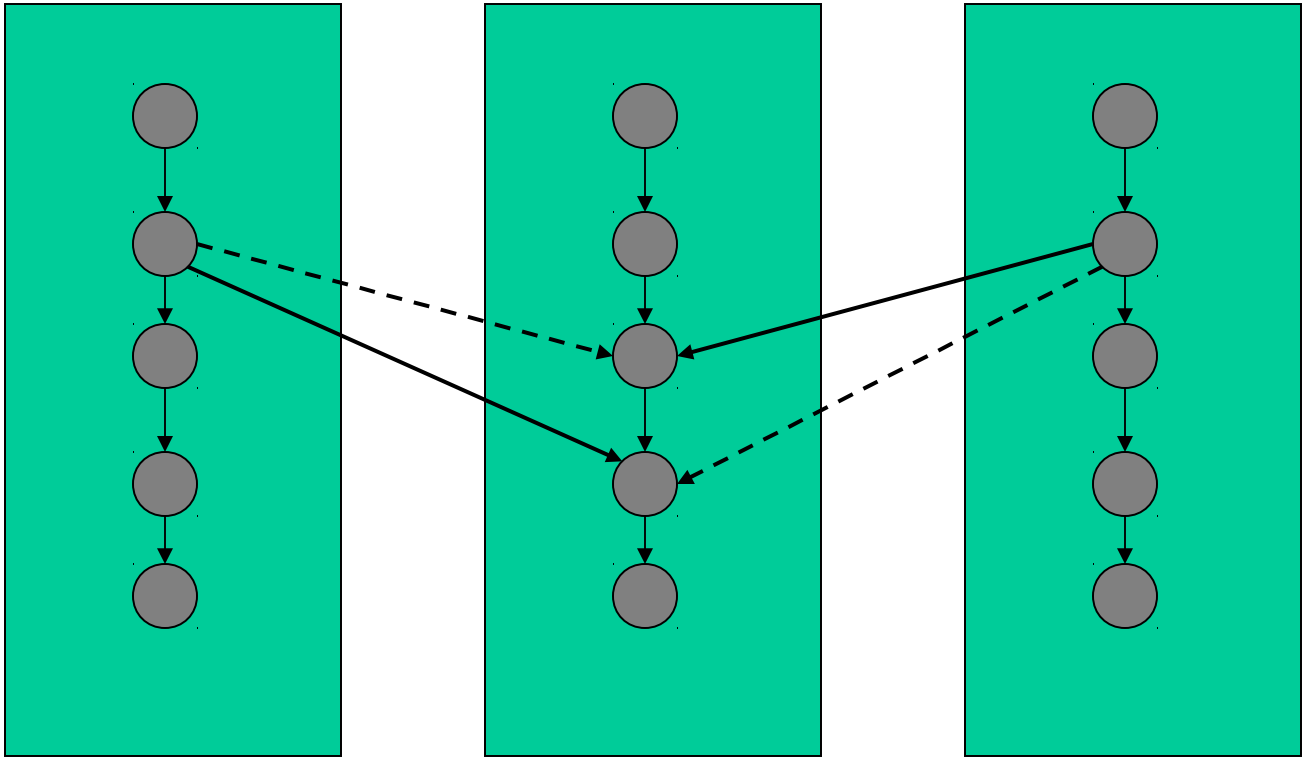
- пересылка данных, расположенных в несмежных областях памяти в одном сообщении;
- пересылка разнотипных данных в одном сообщении;
- облегчение понимания программы;

ХАРАКТЕРНЫЕ ОШИБКИ В MPI-ПРОГРАММАХ

ВИДЫ ОШИБОК

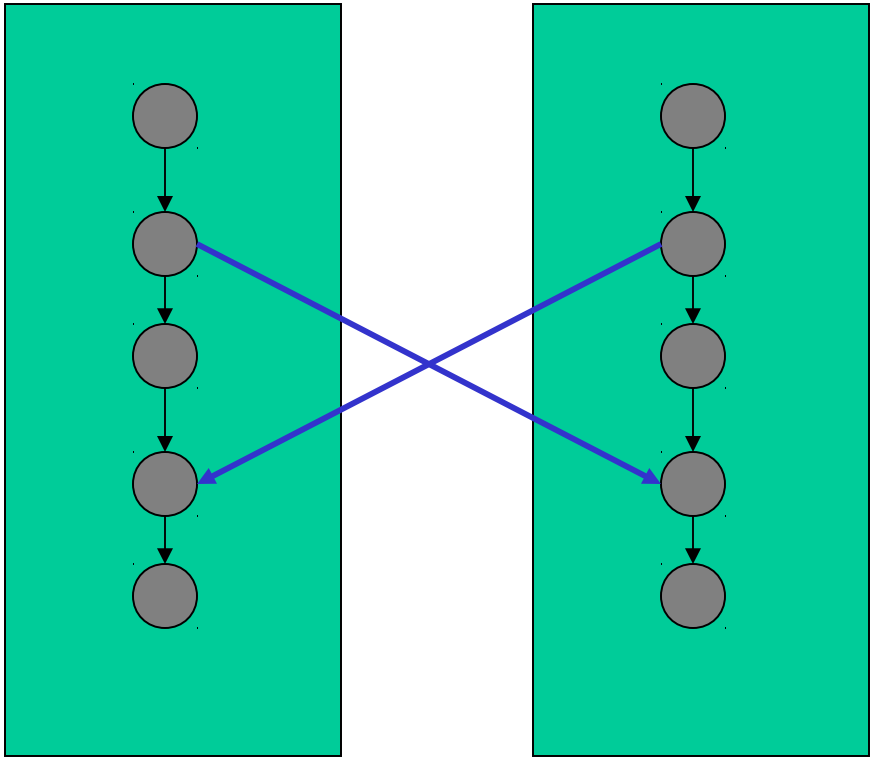
- Ошибки последовательных программ.
- Ошибки несоответствия типов.
- Ошибки работы с MPI-объектами.
- Взаимные блокировки.
- Недетерминизм.

Недетерминизм за счет разницы в относительных скоростях процессов (race condition)



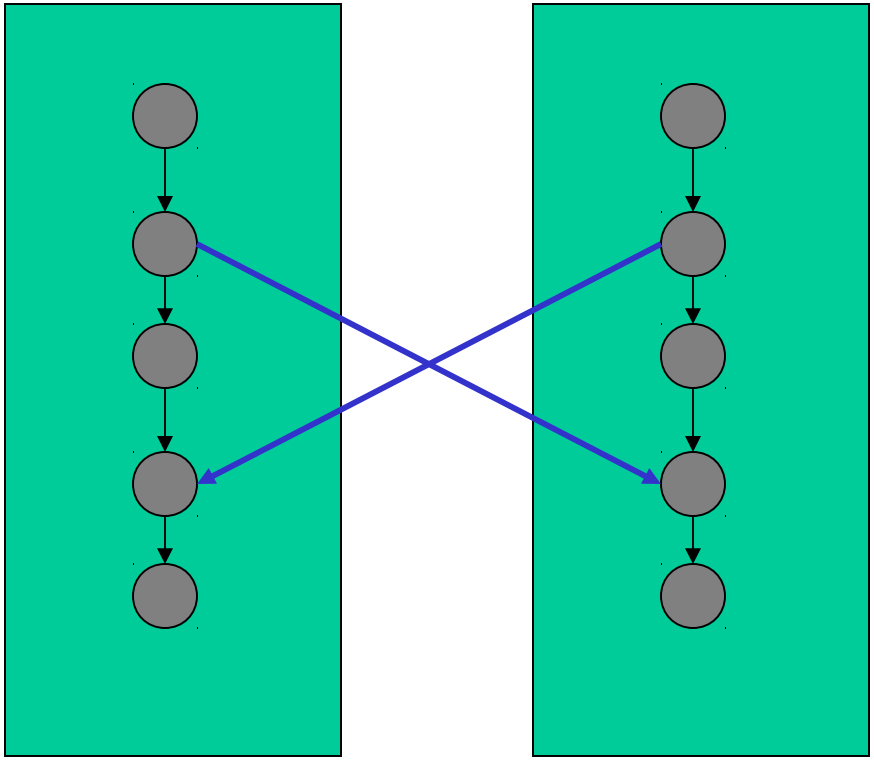
Deadlock

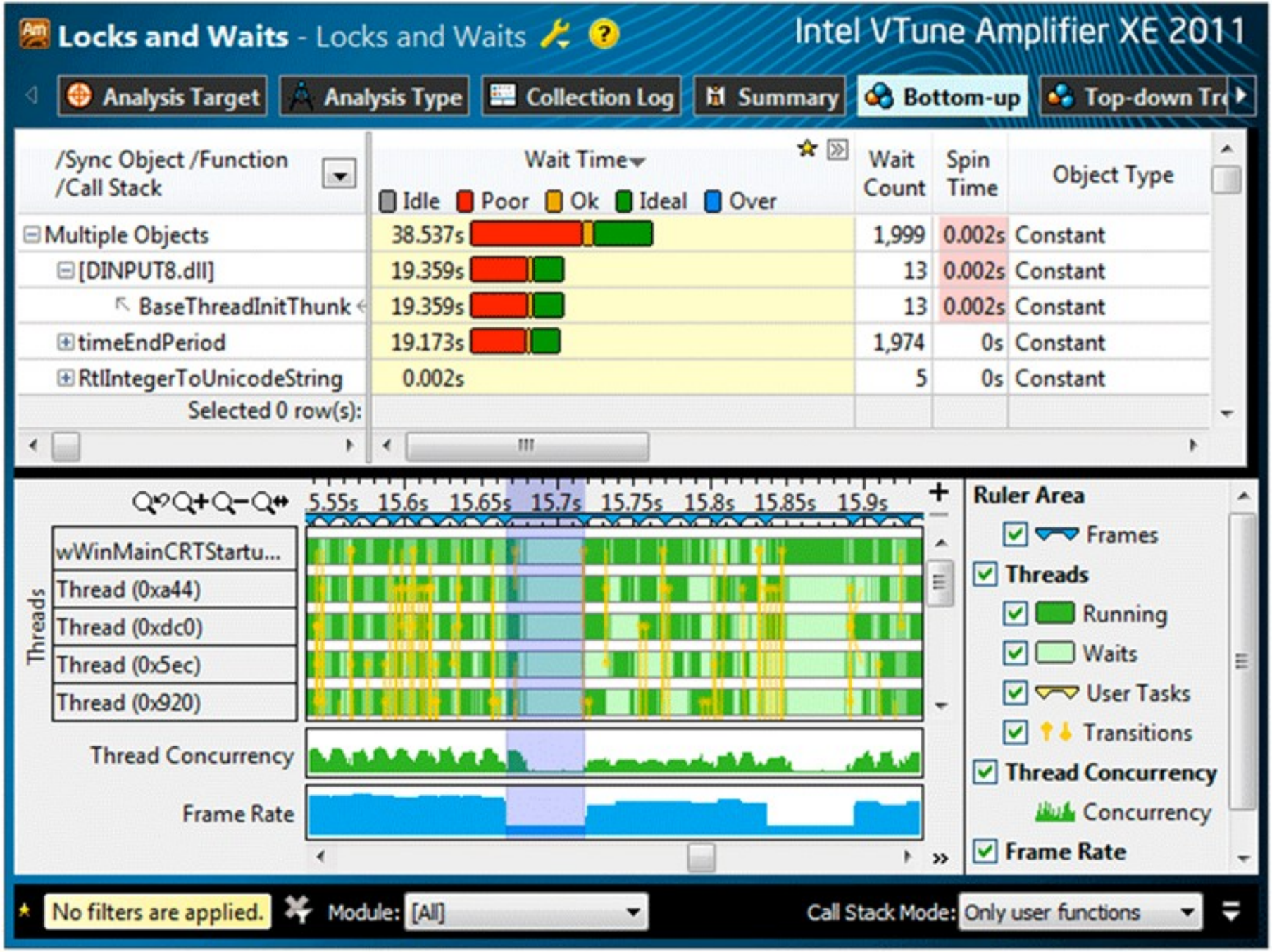
```
if(rank == 0) {
        MPI_Ssend(… 1 …)
        MPI_Recv(…1…)
} else {
        MPI_Ssend(… 0 …)
        MPI_Recv(…0…)
}
```

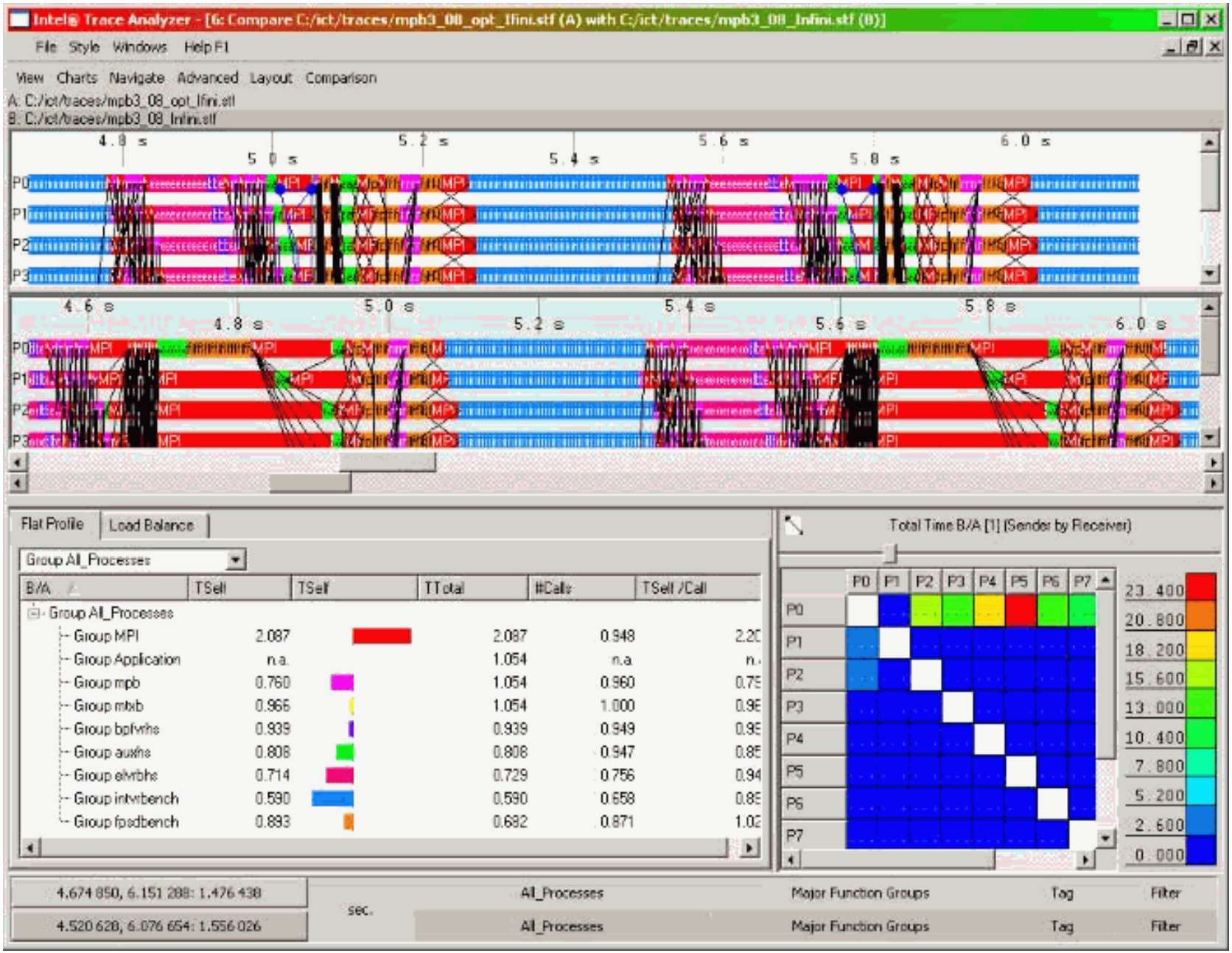


«Недетерминированный» deadlock

```
if(rank == 0) {
        MPI_Send(… 1 …)
        MPI_Recv(…1…)
} else {
        MPI_Send(… 0 …)
        MPI_Recv(…0…)
}
```




Locks and Waits - Locks and Waits
Intel VTune Amplifier XE 2011
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Top-down Tre
/Sync Object /Function /Call Stack
Wait Time
Idle Poor Ok Ideal Over
Wait Count
Spin Time
Object Type
Multiple Objects 38.537s 1,999 0.002s Constant
[DINPUT8.dll] 19.359s 13 0.002s Constant
BaseThreadInitThunk 19.359s 13 0.002s Constant
timeEndPeriod 19.173s 1,974 0s Constant
RtlIntegerToUnicodeString 0.002s 5 0s Constant
Selected 0 row(s):
5.55s 15.6s 15.65s 15.7s 15.75s 15.8s 15.85s 15.9s
Threads
wWinMainCRTStartu...
Thread (0xa44)
Thread (0xdc0)
Thread (0x5ec)
Thread (0x920)
Thread Concurrency
Frame Rate
Ruler Area
Frames
Threads
Running
Waits
User Tasks
Transitions
Thread Concurrency
Concurrency
Frame Rate
No filters are applied.
Module: [All]
Call Stack Mode: Only user functions


Intel® Trace Analyzer - [6: Compare C:/ict/traces/mpb3_08_opt_Ifini.stf (A) with C:/ict/traces/mpb3_08_Inlini.stf (B)]
File Style Windows Help F1
View Charts Navigate Advanced Layout Comparison
A: C:/ict/traces/mpb3_08_opt_Ifini.stf
B: C:/ict/traces/mpb3_08_Inlini.stf
4.8 s
5.0 s
5.2 s
5.4 s
5.6 s
5.8 s
6.0 s
P0
P1
P2
P3
MPI
4.6 s
4.8 s
5.0 s
5.2 s
5.4 s
5.6 s
5.8 s
6.0 s
Flat Profile
Load Balance
Group All_Processes
B/A
TSelf
TSelf
TTotal
#Calls
TSelf /Call
Group All_Processes
Group MPI 2.087 2.087 0.948 2.20
Group Application n.a. 1.054 n.a n.
Group mpb 0.760 1.054 0.960 0.75
Group mtxb 0.965 1.054 1.000 0.96
Group bpfvrhs 0.939 0.939 0.949 0.95
Group auxfns 0.808 0.808 0.947 0.85
Group elvrbhs 0.714 0.729 0.756 0.94
Group intvrbench 0.590 0.590 0.658 0.85
Group fpsdbench 0.893 0.682 0.871 1.02
Total Time B/A [1] (Sender by Receiver)
P0 P1 P2 P3 P4 P5 P6 P7
23.400
20.800
18.200
15.600
13.000
10.400
7.800
5.200
2.600
0.000
4.674 850, 6.151 288: 1.476 438
4.520 628, 6.076 654: 1.556 026
sec.
All_Processes
Major Function Groups
Tag
Filter
All_Processes
Major Function Groups
Tag
Filter