Основы параллельного программирования с использованием MPI

Лекция 8

Немнюгин Сергей Андреевич

Санкт-Петербургский государственный университет

кафедра вычислительной физики

snemnyugin@mail.ru

Интернет-Университет
Суперкомпьютерных Технологий
High-Performance Computing University

Лекция 8

Аннотация

В заключительной лекции даётся краткий обзор тех возможностей MPI, которые не рассматривались в предыдущих лекциях. Среди них – динамический запуск процессов, односторонние обмены, дополнительные вопросы организации коллективных обменов, а также средства управления процессами из MPI-программы в процессе её выполнения.

План лекции

- Динамический запуск процессов.
- Односторонние обмены.
- Коллективные обмены.
- Параллельные операции ввода-вывода.
- Программные инструменты оптимизации параллельных MPI-программ.

Динамический запуск процессов

В MPI-1 параллельная программа запускается в определённом и фиксированном количестве процессов. Это не позволяет приложению, например, «подстраиваться» под изменяющуюся трудоёмкость расчёта. Такая возможность появилась в MPI-2. Новый процесс (несколько процессов) может быть запущен во время выполнения программы и из неё. Процесс может быть также остановлен.

Запуск одного процесса выполняется с помощью обращения к подпрограмме:

```
int MPI_Comm_spawn(char *command, char *argv[], int
maxprocs, MPI_Info info, int root, MPI_Comm comm, MPI_Comm
*intercomm, int array_of_errcodes[])

MPI_Comm_spawn(command, argv, maxprocs, info, root, comm,
intercomm, array_of_errcodes, ierror)
```

Входные параметры:

- `command` – командная строка запуска процесса;
- `argv` – аргументы командной строки запуска процесса;
- `maxprocs` – максимальное количество запускаемых процессов;
- `info` – указывает системе как и где запускается процесс;
- `root` – ранг главного процесса.

Выходные параметры:

- `intercomm` – интеркоммуникатор между исходной группой процессов и вновь запущенными процессами;
- `array_of_errcodes` – коды завершения для запущенных процессов.

При запуске группы процессов для них создаётся собственный коммуникатор `MPI_COMM_WORLD`, отличный от такого же для родительских процессов. Это коллективная операция. Она завершается после того, как во всех дочерних процессах состоится вызов `MPI_Init`. Завершение данного вызова в родительском процессе не означает, что в дочерних процессах завершены все вызовы `MPI_Init`.

Интеркоммуникатор `intercomm` содержит родительский процесс в локальной группе и дочерние процессы.

Командная строка запуска дочернего процесса представляет собой строку, содержащую имя исполняемого файла.

Дочерний процесс обязательно должен вызывать `MPI_Init`.

Аргумент `argv` является массивом строковых значений, представляющих аргументы запускаемых программ.

При вызове `MPI_Comm_spawn` предпринимается попытка запустить `maxprocs` процессов. Если какой-то процесс не может быть запущен, возвращается значение `MPI_ERR_SPAWN`.

Аргумент `info` создаётся вызовом специальной подпрограммы `MPI_Info_create`.

В том случае, когда необходимо запустить несколько разных исполняемых файлов или один файл, но с разными параметрами, можно использовать подпрограмму `MPI_Comm_spawn_multiple`. Опуская подробное описание её параметров, приведём интерфейс:

```
int MPI_Comm_spawn_multiple(int count, char
*array_of_commands[], char **array_of_argv[], int
array_of_maxprocs[], MPI_Info array_of_info[], int root,
MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[])

MPI Comm_spawn_multiple(count, array_of_commands,
array_of_argv, array_of_ maxprocs, array_of_info, root, comm,
intercomm, array_of_errcodes, ierror)
```

Подпрограмма `MPI_Get_parent` возвращает родительский интеркоммуникатор вызывающего процесса:

```
int MPI_Comm_get_parent(MPI_Comm *parent)

MPI_Comm_get_parent(parent, ierr)
```

Если данный процесс не является дочерним по отношению к какому-либо другому процессу, возвращается значение «пустого» коммуникатора `MPI_COMM_NULL`.

Пример
Master

```
#include "mpi.h"
int main(int argc, char *argv[])
{
int world_size, universe_size, *universe_sizep, flag;
MPI_Comm everyone;
char worker_program[100];
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
if (world_size != 1) error("Top heavy with management");
MPI_Attr_get(MPI_COMM_WORLD, MPI_UNIVERSE_SIZE, &universe_sizep, &flag);
if (!flag) {
printf("This MPI does not support UNIVERSE_SIZE. How many\n\ processes
total?");
scanf("%d", &universe_size);
} else universe_size = *universe_sizep;
if (universe_size == 1) error("No room to start workers");
choose_worker_program(worker_program);
MPI_Comm_spawn(worker_program, MPI_ARGV_NULL,
universe_size-1, MPI_INFO_NULL, 0, MPI_COMM_SELF, &everyone,
MPI_ERRCODES_IGNORE);
MPI_Finalize();
return 0;
 }
}
```

Slave

```
#include "mpi.h"
int main(int argc, char *argv[])
{
int size;
MPI_Comm parent;
MPI_Init(&argc, &argv);
MPI_Comm_get_parent(&parent);
if (parent == MPI_COMM_NULL) error("No parent!");
MPI_Comm_remote_size(parent, &size);
if (size != 1) error("Something's wrong with the parent");
MPI_Finalize();
return 0;
}
```

Взаимодействие между группами процессов

MPI-2 допускает организацию обмена сообщениями между группами процессов, которые запущены независимо друг от друга.
Это позволяет «подключиться» к параллельной программе приложению, выполняющему обработку данных.

Такая возможность полезна при создании клиент-серверных приложений и в других ситуациях.

Основные механизмы взаимодействия:
- *связь по имени*;
- *связь через порт*.

Программная реализация взаимодействия:

```
MPI_Open_port
MPI_Close_port
MPI_Comm_accept
MPI_Publish_name
MPI_Unpublish_name
MPI_Lookup_name
```

и др.

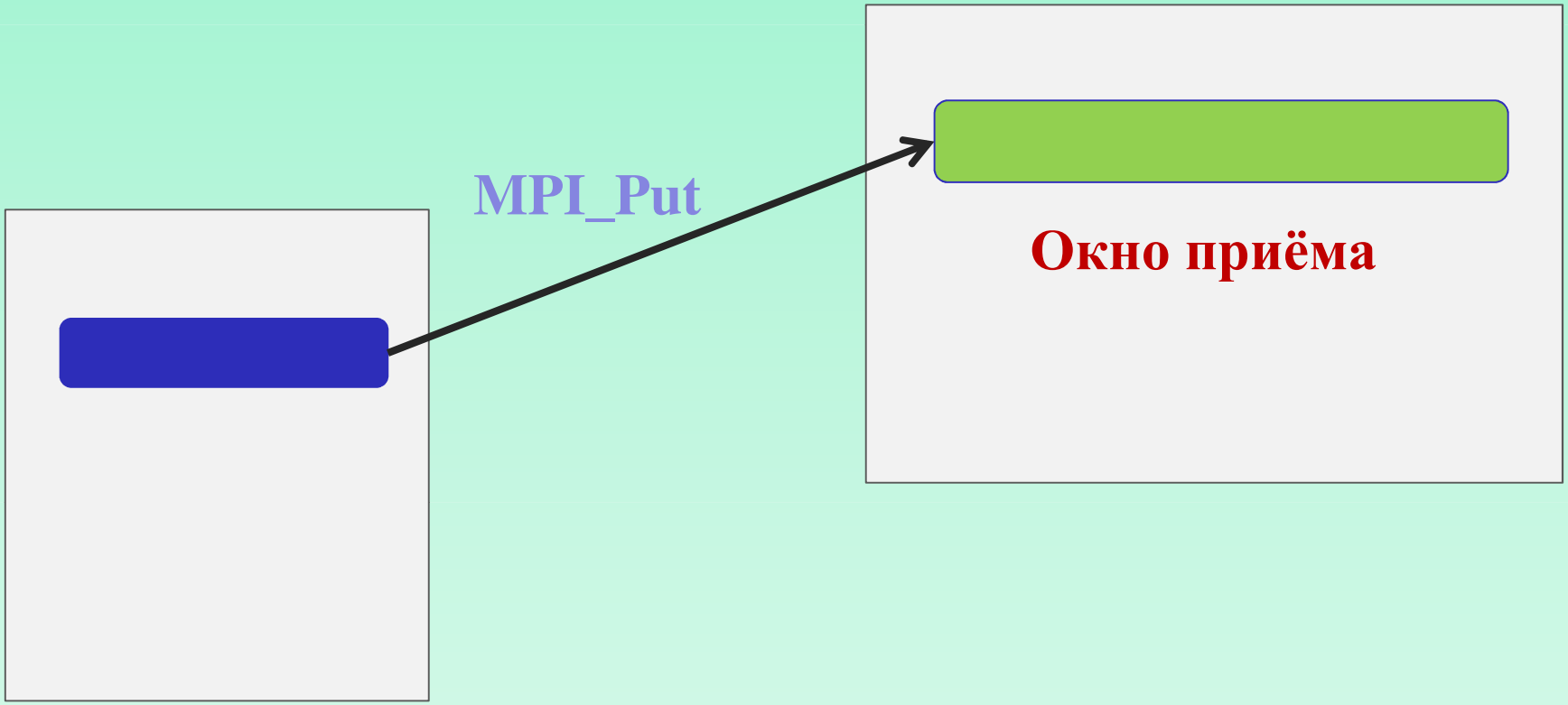
Односторонние обмены

Односторонние обмены основаны на механизме удалённого доступа к памяти (**RMA – Remote Memory Access**) и позволяют процессу, инициировавшему обмен, самостоятельно задать параметры обмена как для источника, так и для адресата сообщения.

Односторонние обмены используются в том случае, когда процесс «знает», какие данные другого процесса он должен модифицировать, а процесс-адресат сообщения этого не знает.
Стандартная схема обмена сообщениями в этом случае требует согласования действий отправителя и получателя сообщения, для чего могут потребоваться дополнительные затраты времени (например, на пересылку параметров обмена). При этом объединены функции *коммуникации* и *синхронизации*.

В односторонних обменах эти функции разделены.

Односторонний обмен возможен, если процесс создаёт «окно», доступное всем остальным процессам.



Окно создаётся (коллективным) вызовом подпрограммы:

```
int MPI_Win_create(void *base, MPI_Aint size, int disp_unit,
MPI_Info info, MPI_Comm comm, MPI_Win *win)

MPI_Win_create(base, size, disp unit, info, comm, win,
ierror)
```

Входные параметры:

- `base` – адрес окна;
- `size` – размер окна в байтах;
- `disp_unit` – масштабный множитель для вычисления смещений;
- `info` – информационный параметр;
- `comm` – коммуникатор.

Выходной параметр – `win` – окно.

Аннулировать окно можно вызовом подпрограммы:

```
int MPI_Win_free(MPI_Win *win)

MPI_Win_free(win, ierror)
```

Три операции одностороннего обмена:

1. `MPI_Put` – передача данных от отправителя в окно;
2. `MPI_Get` - передача данных из окна отправителю;
3. `MPI_Accumulate` – обновление окна получателя.

Это неблокирующие операции.

```
int MPI_Put(void *origin_addr, int origin_count, MPI_Datatype
Origin_datatype, int target_rank, MPI_Aint target_disp, int
target_count, MPI_Datatype target_datatype, MPI_Win win)

MPI_Put(origin_addr, origin_count, origin_datatype,
target_rank, target_disp, target_count, target_datatype, win,
ierror)
```

Входные параметры:

- `origin_addr` – адрес буфера отправки сообщения;
- `origin_count` – количество элементов в буфере отправки;
- `origin_datatype` – тип передаваемых данных;
- `target_rank` – ранг адресата;
- `target_disp` – смещение от начала окна приёма до буфера приёма;
- `target_count` – количество принимаемых данных;
- `target_datatype` – тип принимаемых данных;
- `win` – окно приёма.

При выполнении этой операции данные размещаются в буфере приёма по адресу

```
адрес_окна + смещение × disp_unit
```

При вызове подпрограммы `MPI_Get` данные копируются в обратном направлении – из памяти адресата в память «источника».

Синхронизация при выполнении односторонних обменов

Операции синхронизации:

- `MPI_Win_fence`
- `MPI_Win_start`, `MPI_Win_complete`, `MPI_Win_post`, `MPI_Win_wait`
- `MPI_Win_lock`, `MPI_Win_unlock`

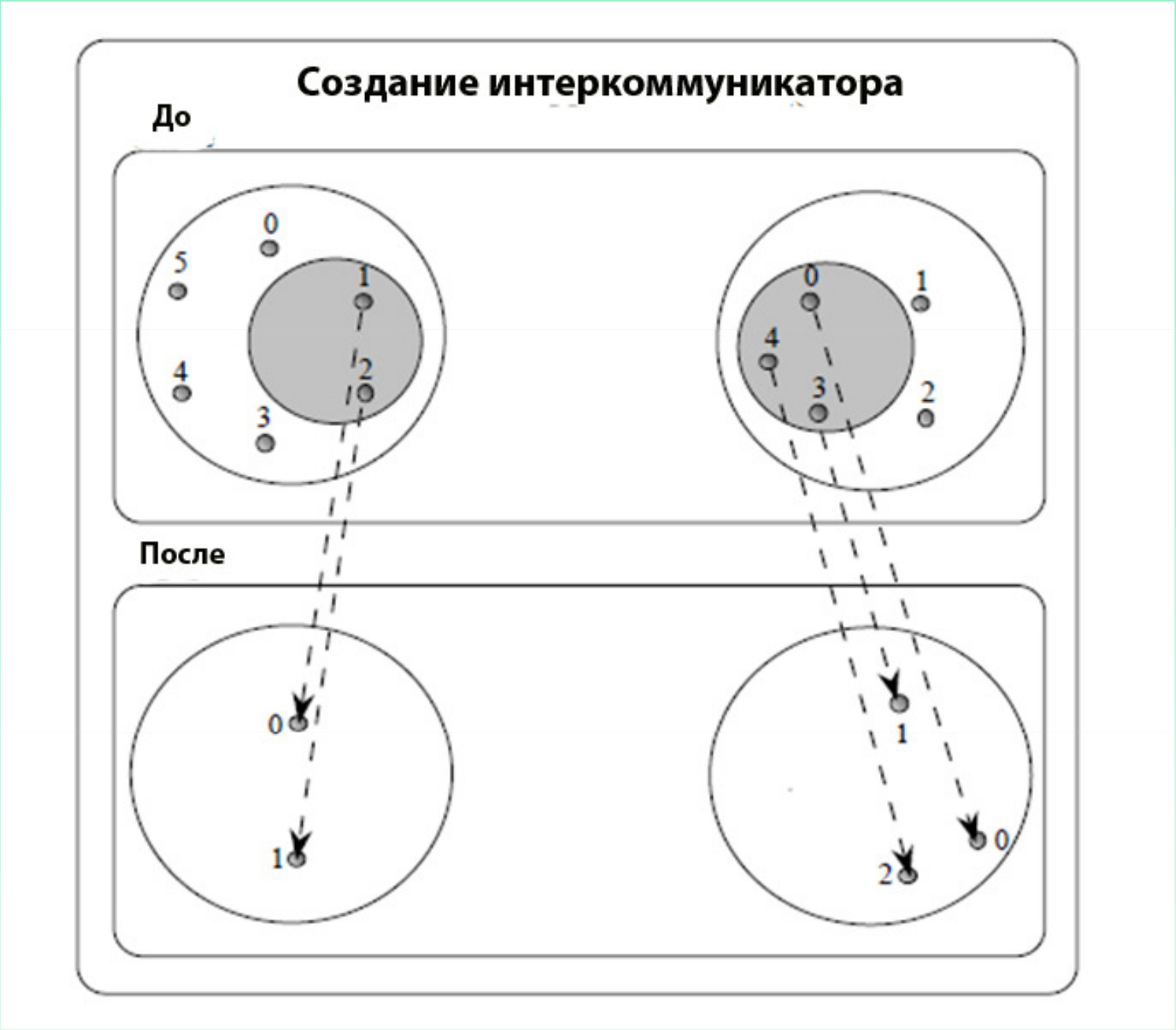
Пример

```
subroutine example(vec1, b, map, n, comm, p)
integer n, map(n), comm, p
real vec1(n), b(n)
integer sizeofreal, win, ierr
call MPI_Type_extent(MPI_REAL, sizeofreal, ierr)
call MPI_Win_create(b, n * sizeofreal, sizeofreal, & MPI_INFO_NULL, comm,
    win, ierr)
call MPI_Win_fence(0, win, ierr)
do i = 1, n
j = map(i) / p
k = mod(map(i), p)
call MPI_Get(vec1(i), 1, MPI_REAL, j, k, 1, MPI_REAL, win, & ierr)
end do
call MPI_Win_fence(0, win, ierr)
call MPI_Win_free(win, ierr)
end
```

Коллективные обмены

В MPI-2 расширены возможности коллективных обменов сообщениями.
Расширения заключаются в обобщении некоторых операций коллективного обмена на интеркоммуникаторы, введении дополнительных конструкторов интеркоммуникаторов, введении двух новых операций обмена – обобщённой операции «all-to-all» и операции исключающего сканирования. Есть и другие расширения.

Подпрограмма `MPI_Comm_create` может использоваться для создания интеркоммуникаторов.

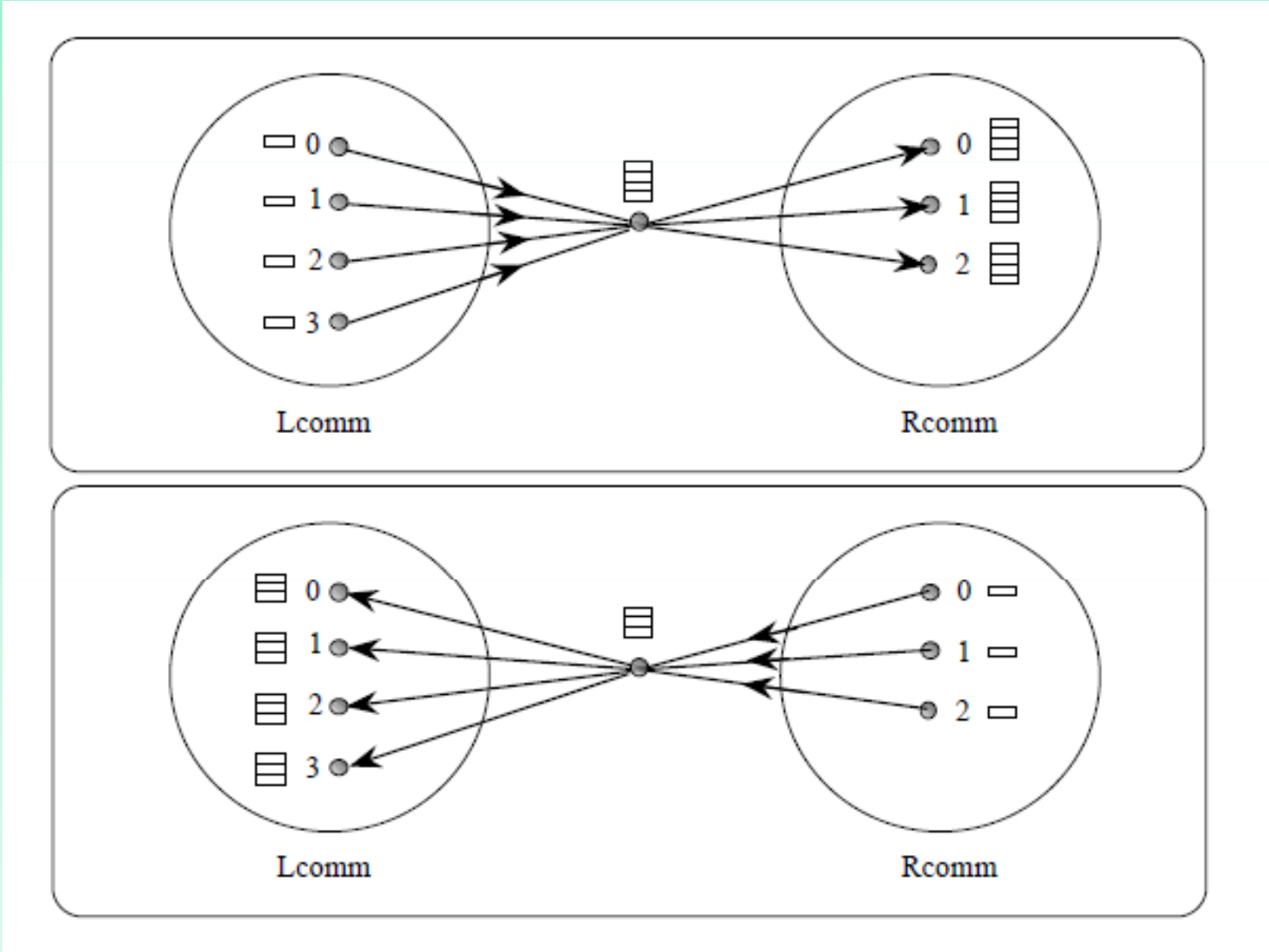

Создание интеркоммуникатора
До
0
5
1
4
2
3
0
1
4
3
2
После
0
1
1
2
0

Подпрограмма `MPI_Comm_split` может использоваться для расщепления интеркоммуникатора.

В MPI-1 коллективные обмены ограничены интракоммуникаторами.

В MPI-2 коллективные обмены могут выполняться и в интеркоммуникаторах.

Пример. Операция Allgather в интеркоммуникаторе



При выполнении коллективных обменов допускается использование одного буфера как для передачи, так и для приёма. **Только для интракоммуникаторов!**

В MPI-2 имеется обобщённая операция `MPI_Alltoallw`. Она допускает дифференцированное задание параметров `count`, `displacement`, `datatype`. Смещения задаются в байтах.

Внешние интерфейсы

Механизм внешних интерфейсов позволяет программисту добавить новую функциональность поверх базовой функциональности MPI.

Обобщённые запросы дают возможность определить новые неблокирующие операции.

При использовании стандартных запросов операции, связанные с ними, выполняются средой исполнения MPI и приложение на этот процесс не влияет.

При использовании обобщённых запросов «ответственность» за выполнение операции берёт на себя приложение. Оно сообщает MPI о завершении операции.

Операции с обобщёнными запросами:

- `MPI_Grequest_start`
- `MPI_Grequest_complete`

и некоторые другие.

Другие возможности MPI-2

Операции декодирования производных типов.

Ассоциирование пользовательской информации с полями структуры `status`.

Присвоение имён объектам MPI (например, коммуникаторам, окнам и др.).

MPI и многопоточность (`MPI_Init_thread`, `MPI_Thread_single`, `MPI_Thread_multiple` и др.).

Новые операции кеширования атрибутов.

Параллельные операции ввода-вывода (`MPI_File_open`, `MPI_File_close`, `MPI_File_read`, `MPI_File_write` и др.).

Поддержка многопоточности:

- `MPI_Init_thread`;
- `MPI_Is_thread_main`;
- `MPI_Query_thread`.

Отладка и профилирование параллельных MPI-программ

Отладка параллельных MPI-программ без использования специальных программных инструментов сложна и малоэффективна.

Существуют разные инструменты, среди них **jumpshot** – собственное средство отладки MPI.

Intel ® Trace Analyzer and Collector – это инструмент анализа, для которого характерно следующее:

- анализ выполняется на основе статистики, собранной во время выполнения программы;
- «инструментовка» исполняемого файла почти не влияет на производительность программы;
- анализ выполняется и для обменов сообщениями;
- поддерживается OpenMP и гибридная модель параллельного программирования MPI+OpenMP;
- поддерживается многопоточность Java.

Intel® Trace Collector

Intel® Trace Analyzer and Collector

Трассировка выполняется с помощью инструментовки приложения, то есть внедрения в него обращений к функциям, которые собирают статистику по различным событиям.

Виды инструментовки:
- бинарная;
- компиляторная;
- на уровне исходного кода.

При инструментовке на уровне исходного кода используется заголовочный файл VT.h (C/C++) или включаемый файл VT.inc (Fortran).

При инструментовке любого вида используются библиотеки Intel® Trace Collector.

Библиотеки Intel® Trace Collector

Библиотека	**Назначение**
libVTnull	«Заглушка»
libVT	Трассировка MPI-приложений и SHMEM-приложений
libVTfs	Безопасная трассировка MPI-приложений и SHMEM-приложений (результаты трассировки сохраняются даже после аварийного завершения приложения)
libVTmc	Проверка корректности
libVTcs	Трассировка распределённых приложений
VT_sample	Автоматическая трассировка счётчиков с PAPI и getrusage

Утилиты Intel® Trace Collector

Утилита	Назначение
stftool	Преобразование файлов с результатами трассировки
xstftool/expandvtlog.pl	Преобразование трассировочных файлов в читаемый формат
itcpin	Трассировка бинарных файлов без перекомпиляции

Сборка исполняемого файла для трассировки MPI-приложения

Linux

```
mpicc app.o -L$VT_LIB_DIR -lVT $VT_ADD_LIBS -o app
mpif77 app.o -L$VT_LIB_DIR -lVT $VT_ADD_LIBS -o app
```

Microsoft* Windows*

```
mpiicc app.obj /link /LIBPATH:%VT_LIB_DIR% VT.lib
mpiifort app.obj /link /LIBPATH:%VT_LIB_DIR% VT.lib
```

Запуск файла для трассировки MPI-приложения

Задать значения переменных окружения.

`VT_CONFIG`
путь к каталогу, где находится конфигурационный файл трассировки.

`VT_CONFIG_RANK`
ранг процесса (относительно MPI_COMM_WORLD), который должен считывать конфигурационный файл трассировки.

Приложение запускается как обычное MPI-приложение.

Файл трассировки формируется в памяти и сохраняется на диске после завершения вызова MPI_Finalize. Имя файла **<маршрутное имя исполняемого файла>.stf**

Аварийное завершение MPI-приложения приводит к потере результатов трассировки, если используется библиотека libVT.

Аварийное завершение MPI-приложения НЕ приводит к потере результатов трассировки, если используется библиотека libVTfs. В этом случае при аварийном завершении приложения его процессы «замораживаются» до того момента, когда на диске будет сохранён файл трассировки.

Фиксируются события:

Сигналы – внутренние (ошибки сегментации, ошибки операций с плавающей точкой) и внешние (SIGINT, SIGTERM). SIGKILL не фиксируется.
Преждевременное завершение – один или несколько процессов завершились до вызова MPI_Finalize.
Ошибки MPI – ошибки обменов, неправильно заданные параметры функций MPI.
Блокировки – фиксируются, если в течение некоторого времени процессы простаивают (находятся в состоянии вызова одной MPI-функции). Время простоя задаётся с помощью DEADLOCK-TIMEOUT.

Ошибки параллельного ввода-вывода фиксируются только в среде ОС Linux.

Intel® Trace Collector позволяет выполнять трассировку односторонних обменов.

Intel® Trace Collector позволяет выполнять трассировку приложений, работающих с общей памятью.

Intel® Trace Collector позволяет выполнять трассировку последовательных приложений (библиотека libVTcs). Для трассировки требуются вызовы VT_initialize() и VT_finalize().

«Фолдинг» (сокрытие информации) позволяет уменьшить количество трассировочной информации.

Компиляторная инструментовка запускается ключами:

Компиляторы Intel

`-tcollect [=библиотека]` (Linux)

`/Qtcollect [=библиотека]` (Windows)

Компиляторы GCC

`-finstrument-function` (Linux)

Бинарная инструментовка

`itcpin` выполняет подстановку библиотек ITC, их инициализацию, запись событий входа в функции и выхода из них.

```
itcpin [<ключи ITC>] -- <командная строка запуска приложения>
```

Если ключи не указаны, утилита выполняет проверку возможности инструментовки файла и определение способа такой инструментовки.

Ключ `--list` позволяет определить функции в исполняемом файле. Результат записывается в формате:
`<имя бинарного файла>:<имя исходного файла>:<имя функции>`

Трассировка

Ключ `--run` запускает приложение в режиме сбора статистики. По умолчанию, если в исполняемом файле содержатся вызовы MPI-функций, подключается библиотека libVT.

Ключ `--insert` используется для подключения определённой библиотеки.

На всех вычислительных узлах Collector должен быть установлен в каталогах с одинаковым маршрутным именем.

Ключ `--profile` используется для профилирования функций.

Каждая строка содержит:
- Поток или процесс.
- Функция.
- Получатель сообщения.
- Размер сообщения.
- Количество процессов.

Собирается следующая статистика:
- Количество обменов.
- Минимальное время выполнения, исключая время вызываемых функций.
- Максимальное время выполнения, исключая время вызываемых функций.
- Суммарное время выполнения, исключая время вызываемых функций.
- Минимальное время выполнения, включая время вызываемых функций.
- Максимальное время выполнения, включая время вызываемых функций.
- Суммарное время выполнения, включая время вызываемых функций.

В поле получателя сообщения записывается:
- `0xffffffff` для операций с файлами;
- `0xfffffffe` для коллективных операций.

В поле «Сообщение» записывается:

- `0xffffffff` для операций, не связанных с пересылкой сообщений (например, не функция MPI, MPI_Comm_rank и т.д.);

В поле «Количество процессов-участников» записывается отрицательное значение, если сообщение принимается.

Функция	**Записывается при сборе статистики**
MPI_Barrier	0
MPI_Bcast	Число рассылаемых байтов
MPI_Gather	Число отправленных байтов
MPI_Gatherv	Число отправленных байтов
MPI_Scatter	Число полученных байтов
MPI_Scatterv	Число полученных байтов
MPI_Allgather	Число отправленных+полученных байтов
MPI_Allgatherv	Число отправленных+полученных байтов
MPI_Alltoall	Число отправленных+полученных байтов
MPI_Alltoallv	Число отправленных+полученных байтов
MPI_Reduce	Число отправленных байтов
MPI_Allreduce	Число отправленных+полученных байтов
MPI_Reduce_Scatter	Число отправленных+полученных байтов
MPI_Scan	Число отправленных+полученных байтов

Сбор статистики с привязкой к исходному коду

Для сбора статистики с привязкой к исходному коду необходимо выполнить компиляцию с ключом генерации отладочной информации:

```
mpicc -g -c app.c
mpif77 -g -c app.f
```

Перед запуском программы следует установить значение переменной окружения VT_PCTRACE равным целому положительному числу (например, 5). Другой способ – указать параметры трассировки в конфигурационном файле:

```
# trace 4 call levels whenever MPI is used
ACTIVITY MPI 4
# trace one call level in all routines not mentioned
# explicitly; could also be for example, PCTRACE 5
PCTRACE ON
```

Счетчики

Сбор данных о производительности процессора

PAPI (Performance Application Programming Interface) – интерфейс, позволяющий собирать статистику с аппаратных счётчиков и системную статистику.

Поддержка PAPI реализована поверх ITC – в файле VT_sample.c.

Сбор статистики с аппаратных счётчиков включается с помощью опции конфигурации:

```
COUNTER <имя счётчика> ON
```

В имени счётчика допускается использование метасимвола *.

Системные счётчики (список неполон)

Название счётчика	Единицы измерения	Собираемая информация
RU_UTIME	Сек.	Время в режиме задачи
RU_STIME	Сек.	Время в режиме ядра
RU_MAXRSS	Байты	Максимальный размер резидентной части
RU_IXRSS	Байты	Суммарный размер разделяемой памяти
RU_MAJFLT	#	Ошибки страниц
RU_NSWAP	#	Выгрузки
RU_INBLOCK	#	Блочные операции ввода
RU_OUBLOCK	#	Блочные операции вывода
RU_MSGSND	#	Отправленные сообщения
RU_MSGRCV	#	Полученные сообщения
RU_NSIGNALS	#	Полученные сигналы

Системные счётчики (локальные, для узла)

Название счётчика	**Единицы измерения**	**Собираемая информация**
disk_io	кб/сек	Ввод-вывод на диск
net_io	кб/сек	Сетевой ввод-вывод (не включает транспортный уровень MPI)
cpu_ . . .	%	Средняя доля процессорного времени всех процессоров, проведенного в . . .
cpu_idle	%	. . . режиме простоя
cpu_sys	%	. . . режиме ядра
cpu_usr	%	. . . режиме задачи

Проверка корректности

ПРОВЕРКА КОРРЕКТНОСТИ

Проверка корректности включает:

- проверку переносимости;
- проверку нарушений стандарта MPI, не приводящие к немедленным фатальным последствиям, но проявляющиеся при переходе на другие платформы или к другим реализациям MPI;
- проверку ошибок в среде исполнения.

Проверка корректности реализована в библиотеке libVTmc.
По умолчанию результаты проверки корректности не фиксируются. Включить запись можно установкой флага CHECK-TRACING в файле конфигурации.

Проверка корректности требует дополнительных ресурсов!
Проверка корректности реализована только для Intel® MPI Library!

Как это делается:

1 вариант.
При запуске использовать переменную окружения LD_PRELOAD (только Linux):

```
mpiexec -genv LD_PRELOAD libVTmc.so -n ...
```

2 вариант.
Бинарная инструментовка.

3 вариант.
Пересборка приложения.

4 вариант.
Использование ключа –check для mpiexec (только Intel® MPI).

Сигнатуры ошибок

Локальные ошибки

Сигнатура	Описание
LOCAL:EXIT:SIGNAL	Процесс остановлен «фатальным» сигналом
LOCAL:EXIT:BEFORE_MPI_FINALIZE	Процесс завершён без вызова MPI_Finalize()
LOCAL:MEMORY:OVERLAP	Несколько операций MPI используют одну область памяти
LOCAL:MEMORY:ILLEGAL_MODIFICATION	Некорректная модификация данных
LOCAL:MEMORY:INACCESSIBLE	Буфер недоступен
LOCAL:MEMORY:ILLEGAL_ACCESS	Некорректный доступ к памяти, уже используемой MPI
LOCAL:MEMORY:INITIALIZATION	Проверка распределённой памяти
LOCAL:REQUEST:ILLEGAL_CALL	Неправильная последовательность вызовов
LOCAL:REQUEST:NOT_FREED	Избыточное количество запросов или завершение программы с отложенными обменами
LOCAL:BUFFER:INSUFFICIENT_BUFFER	Недостаточно памяти для буферизованной отправки сообщения

Глобальные ошибки

Сигнатура ошибки	Описание
GLOBAL:MSG/COLLECTIVE:DATATYPE:MISMATCH	Несогласованность типов
GLOBAL:MSG/COLLECTIVE:DATA_TRANSMISSION_CORRUPTED	Модификация данных во время передачи
GLOBAL:MSG:PENDING	Завершение программы до приёма всех сообщений
GLOBAL:DEADLOCK:HARD	Цикл процессов, ожидающих друг друга
GLOBAL:DEADLOCK:NO_PROGRESS	Возможна блокировка
GLOBAL:COLLECTIVE:OPERATION_MISMATCH	Процессы участвуют в разных коллективных операция
GLOBAL:COLLECTIVE:SIZE_MISMATCH	Данных больше или меньше, чем должно быть
GLOBAL:COLLECTIVE:REDUCTION_OPERATION_MISMATCH	Ошибка в операции приведения
GLOBAL:COLLECTIVE:INVALID_PARAMETER	Неправильные параметры коллективной операции

Инструментовка, определённая пользователем

Intel® Trace Collector позволяет пользователю описать процесс трассировки.
Инструментовка выполняется на уровне исходного кода.

Заголовочный файл VT.h для программ на C/C++.
Включаемый файл VT.inc для программ на Fortran.

Минимальный набор функций:
- `VT_initialize()` – инициализация;
- `VT_getrank()` – ID процесса;
- `VT_finalize()` – завершение.

Некоторые возможности пользовательской инструментовки:

- Инициализация, управление, завершение.
- Определение и запись положения в исходном тексте.
- Определение и запись функций и регионов.
- Определение изменения состояния.
- Определение и запись перекрытия областей видимости.
- Определение групп процессов.
- Определение и запись счётчиков.
- Определение событий коммуникации.

Intel® Trace Analyzer

Intel® Trace Analyzer

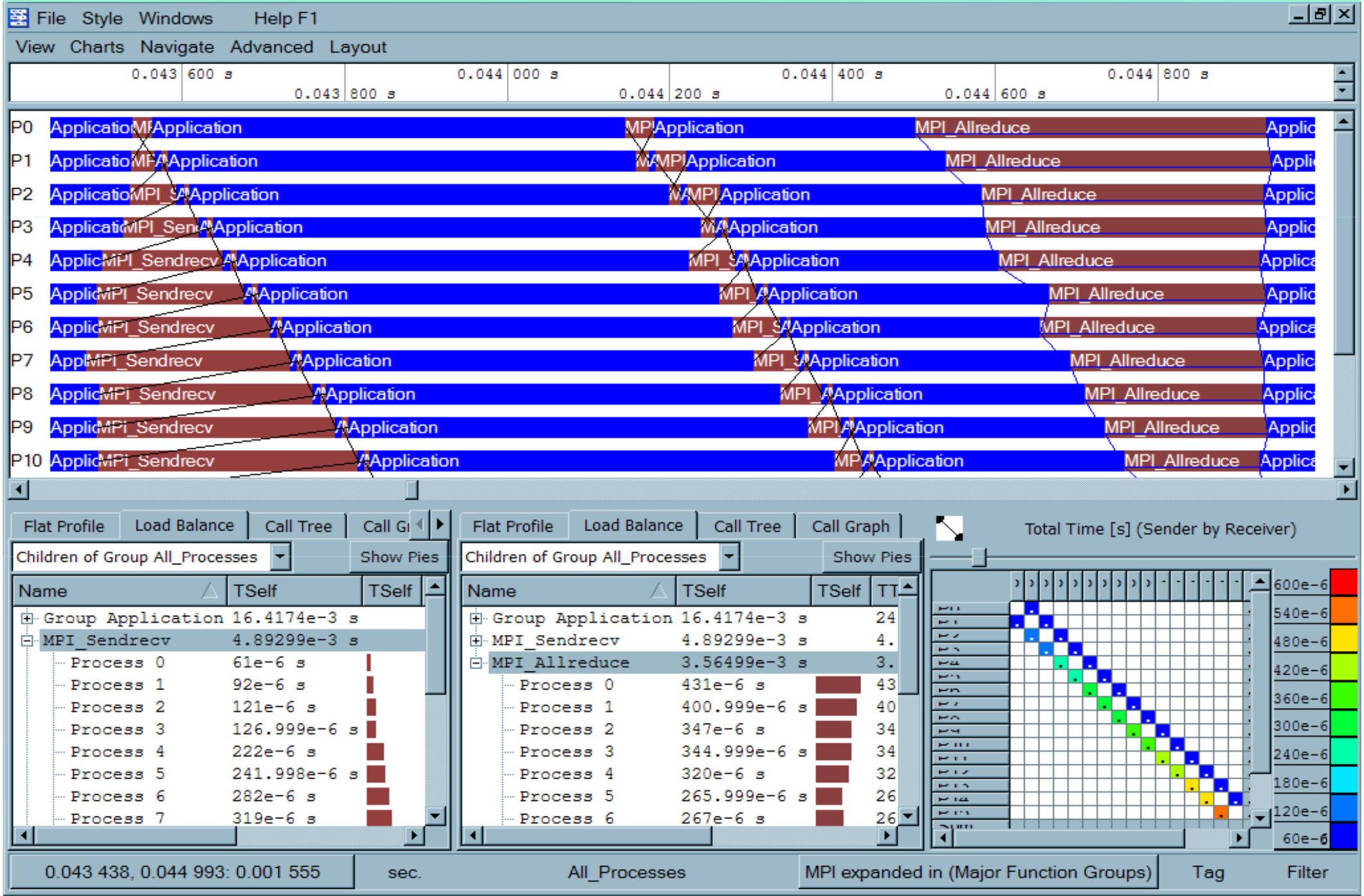
Запуск анализатора:

```
# traceanalyzer файл_трассировки.stf
```
(ОС Linux)

Start -> All Programs -> Intel Trace Analyzer (MS Windows)

Поддерживается как графический интерфейс, так и интерфейс командной строки.

Пример вывода результатов анализа



Интерфейс Intel® Trace Analyzer

Time Scale
Временная шкала.

Event Timeline
Временная диаграмма, на которой отображаются события, связанные с процессами параллельной программы.

Qualitative Timeline
Временная диаграмма, на которой отображаются атрибуты событий.

Quantitative Timeline
Временная диаграмма, на которой отображается поведение параллельной программы.

Function Profile
Диаграмма, на которой отображается информация, связанная с функциями программы.

Message Profile
Диаграмма, на которой отображается статистическая информация о двухточечных обменах.

Collective Operations Profile
Диаграмма, на которой отображается статистическая информация о коллективных обменах.

Tagging
Подсветка событий, удовлетворяющих условиям, заданным пользователем.

Filtering
Фильтрация событий, удовлетворяющих условиям, заданным пользователем.

Reset Tagging/Filtering
Откат операций маркировки и фильтрации.

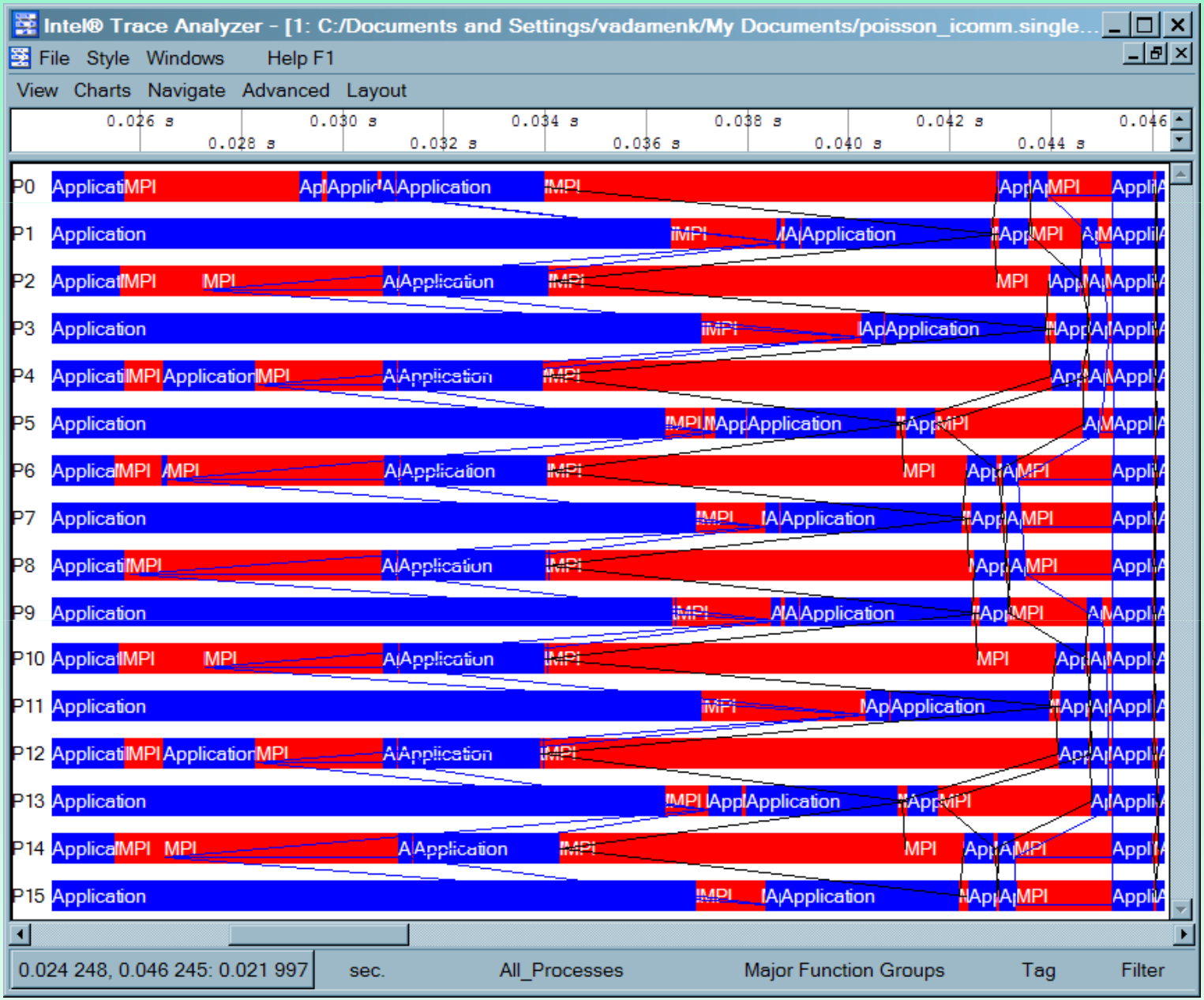
Process Aggregation
Объединение результатов трассировки по группам процессов.

Default Aggregation
Установка параметров аггрегации по умолчанию.

Show Process Group “Other” и **Show Function Group “Other”**
Отображение результатов по процессам и функциям, не попавшим в объединение.

Диаграммы

Диаграмма событий



Отображает активность индивидуальных процессов.

Горизонтальная ось – время.

Вертикальная ось – процесс.

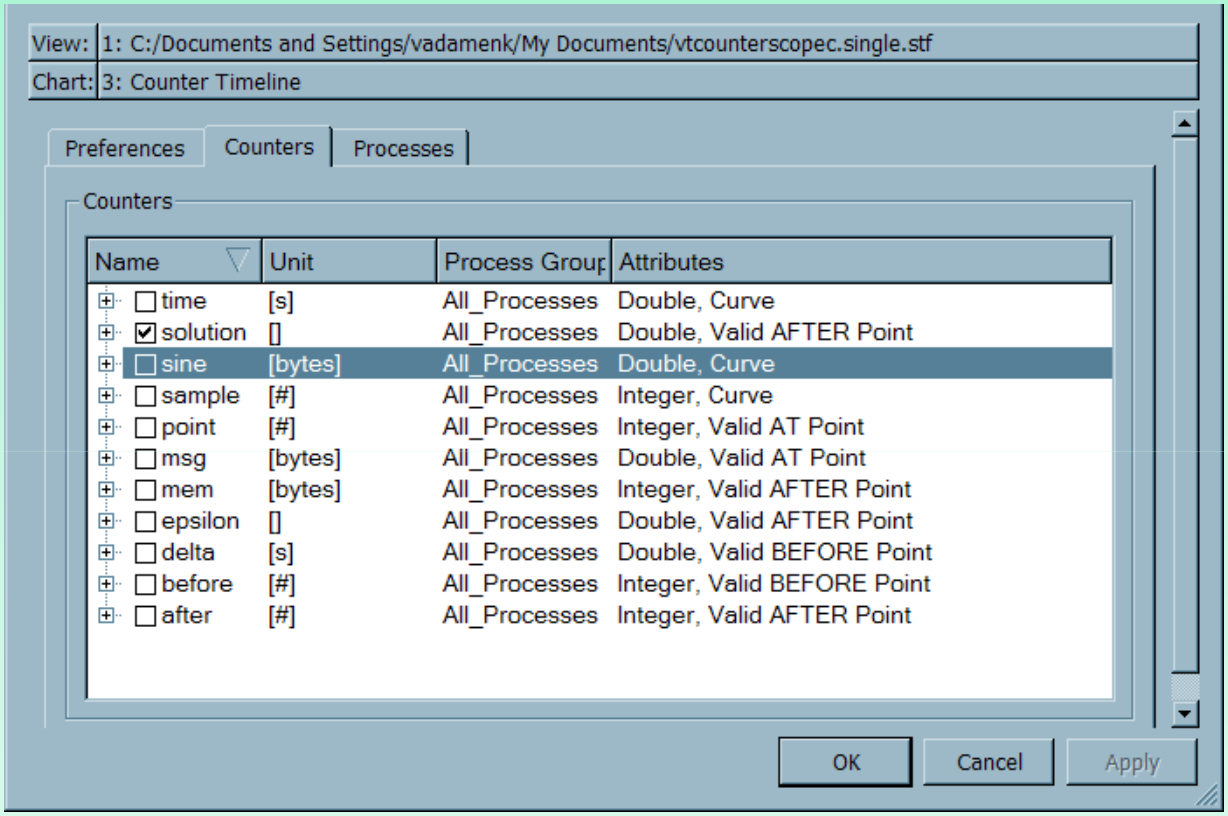
Чёрными линиями отображаются операции двухточечного обмена.

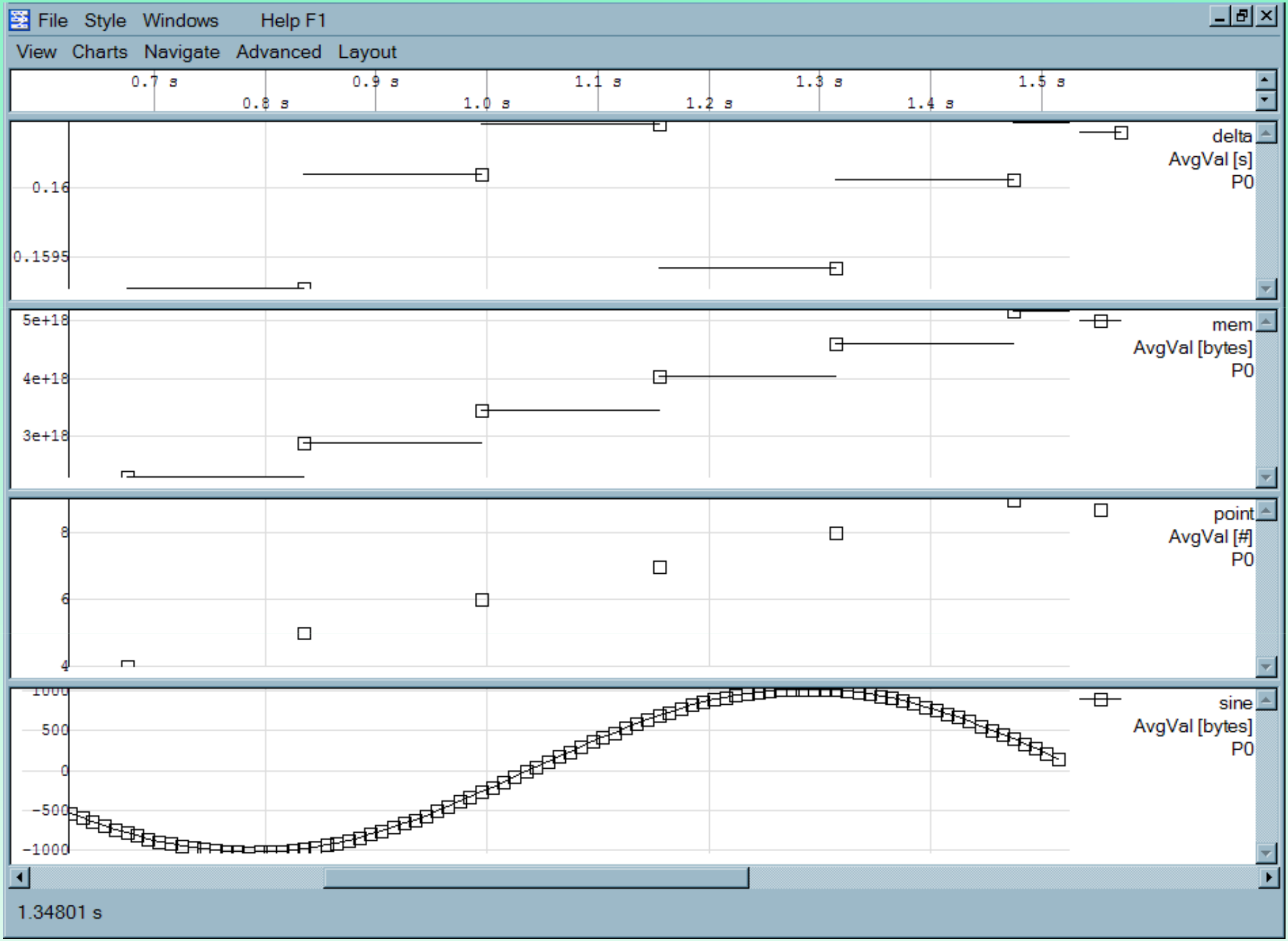
Синими линиями отображаются операции коллективного обмена.

Временной масштаб изменяется с помощью мыши.

«Диаграмма счётчиков» отображает значения счётчиков, сохранённые в файле трассировки.

Пример определения отображаемых счётчиков:




File Style Windows Help F1
View Charts Navigate Advanced Layout
0.7 s
0.8 s
0.9 s
1.0 s
1.1 s
1.2 s
1.3 s
1.4 s
1.5 s
delta
AvgVal [s]
P0
0.16
0.1595
mem
AvgVal [bytes]
P0
5e+18
4e+18
3e+18
point
AvgVal [#]
P0
8
6
4
sine
AvgVal [bytes]
P0
1000
500
0
-500
-1000
1.34801 s

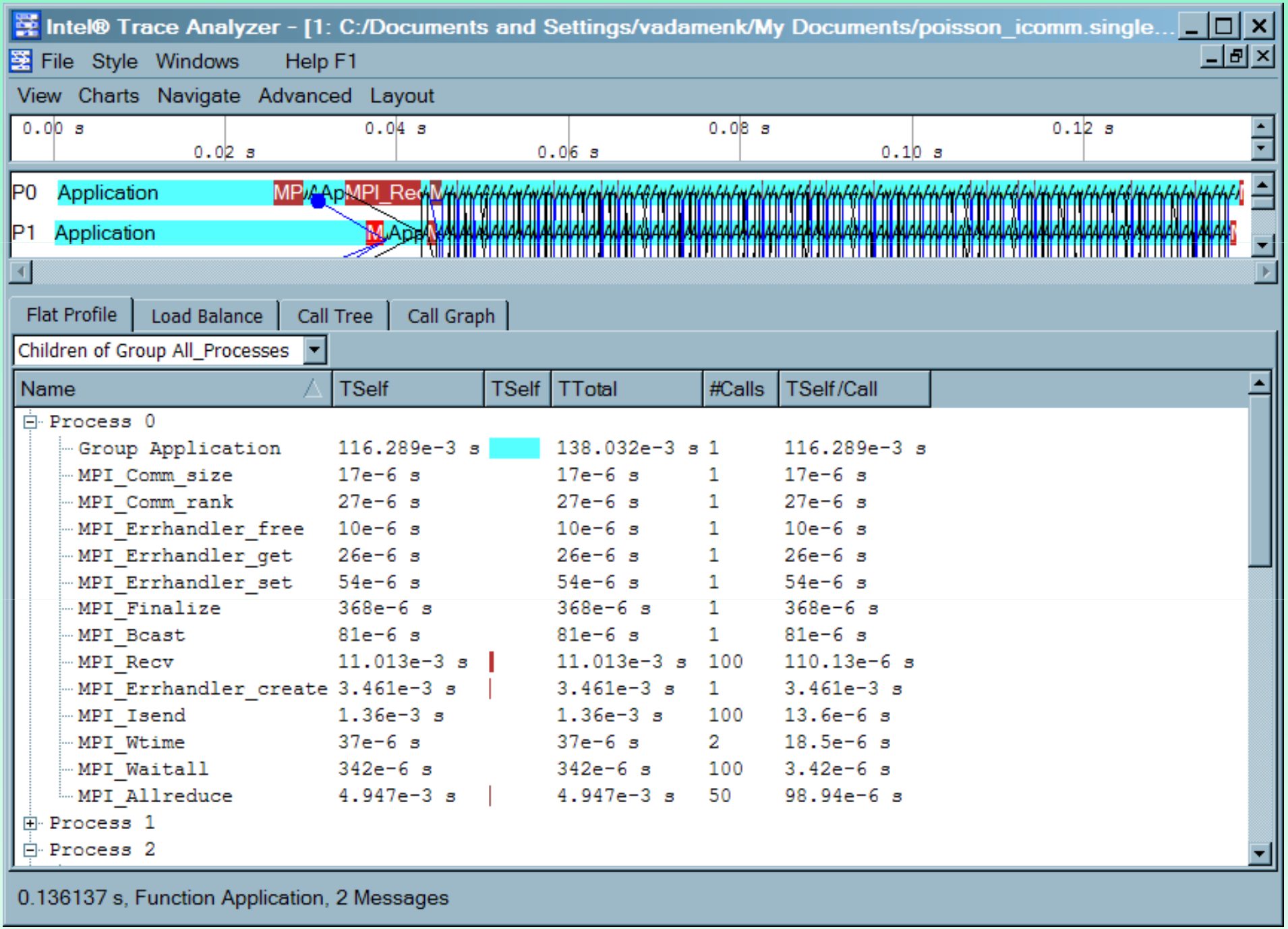
Профиль функций

Диаграмма **«The Function Profile»** отображает детальную информацию о производительности.

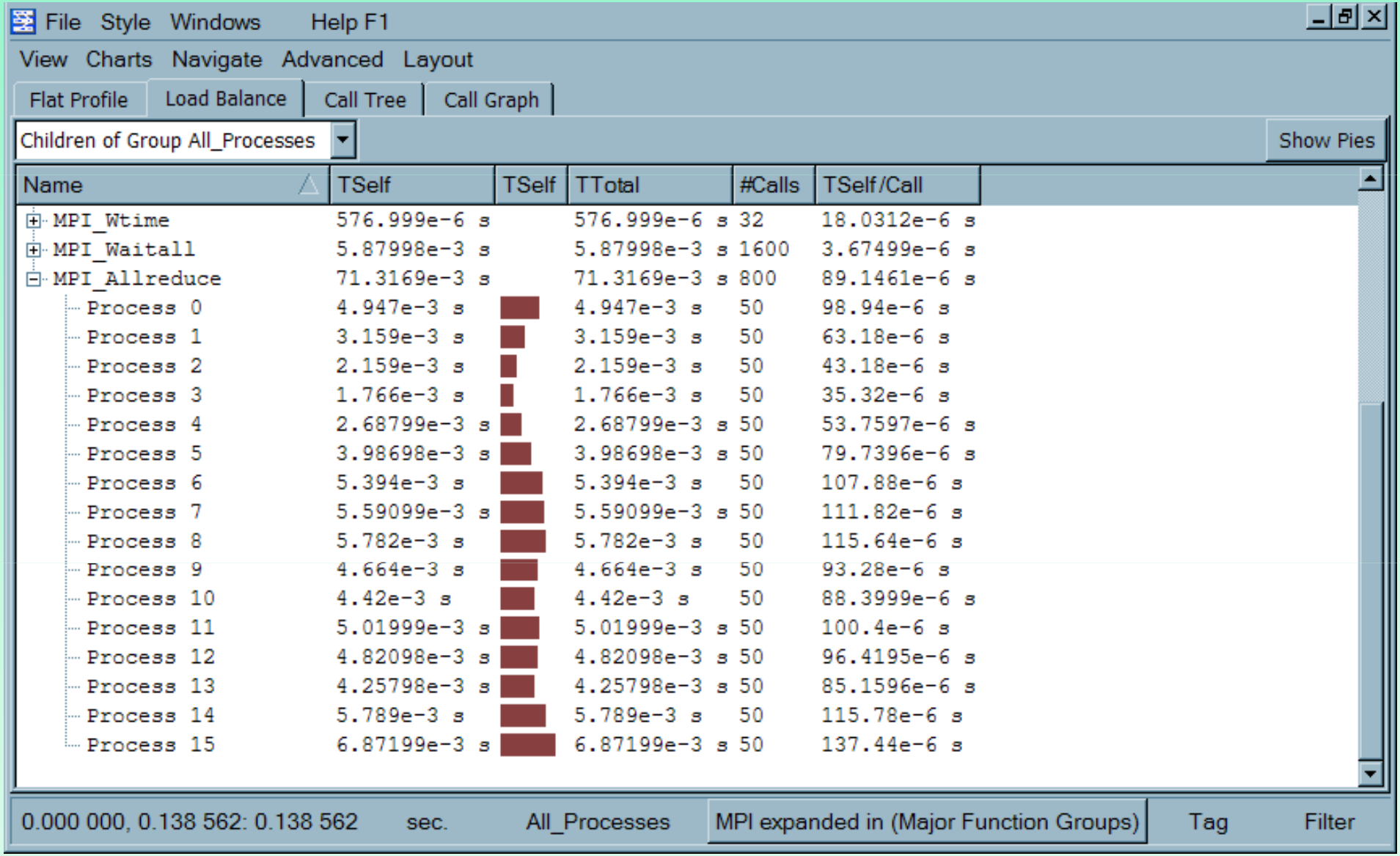
Содержит вкладки:

- Flat Profile – итоговая статистика по процессам.
- Load Balance – итоговая статистика по группам функций.
- Call Tree – последовательности вызовов.
- Call Graph – показывает небольшую часть графа вызовов (3 узла – центральная функция, вызывающая и вызываемая функции).

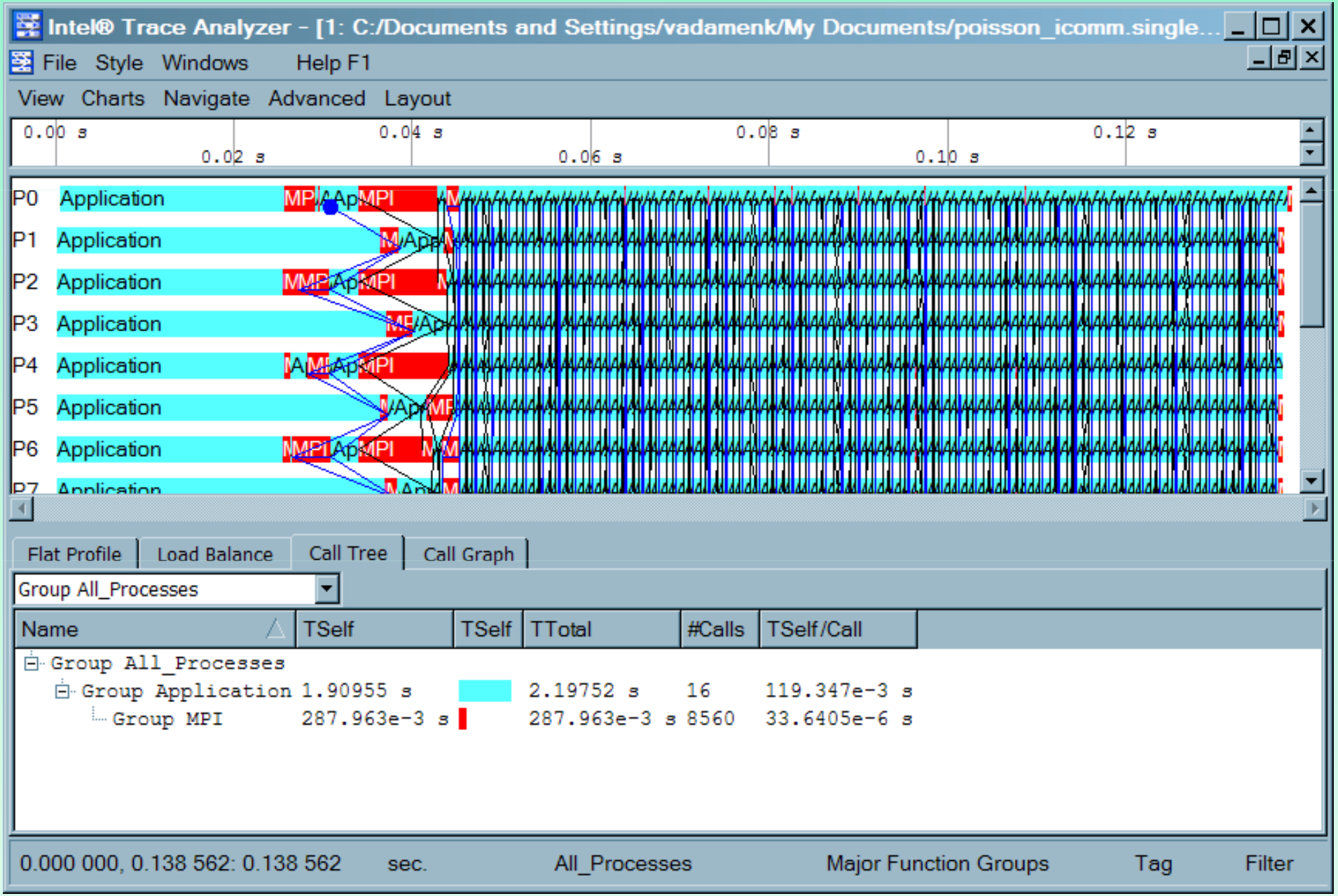
«Flat Profile»



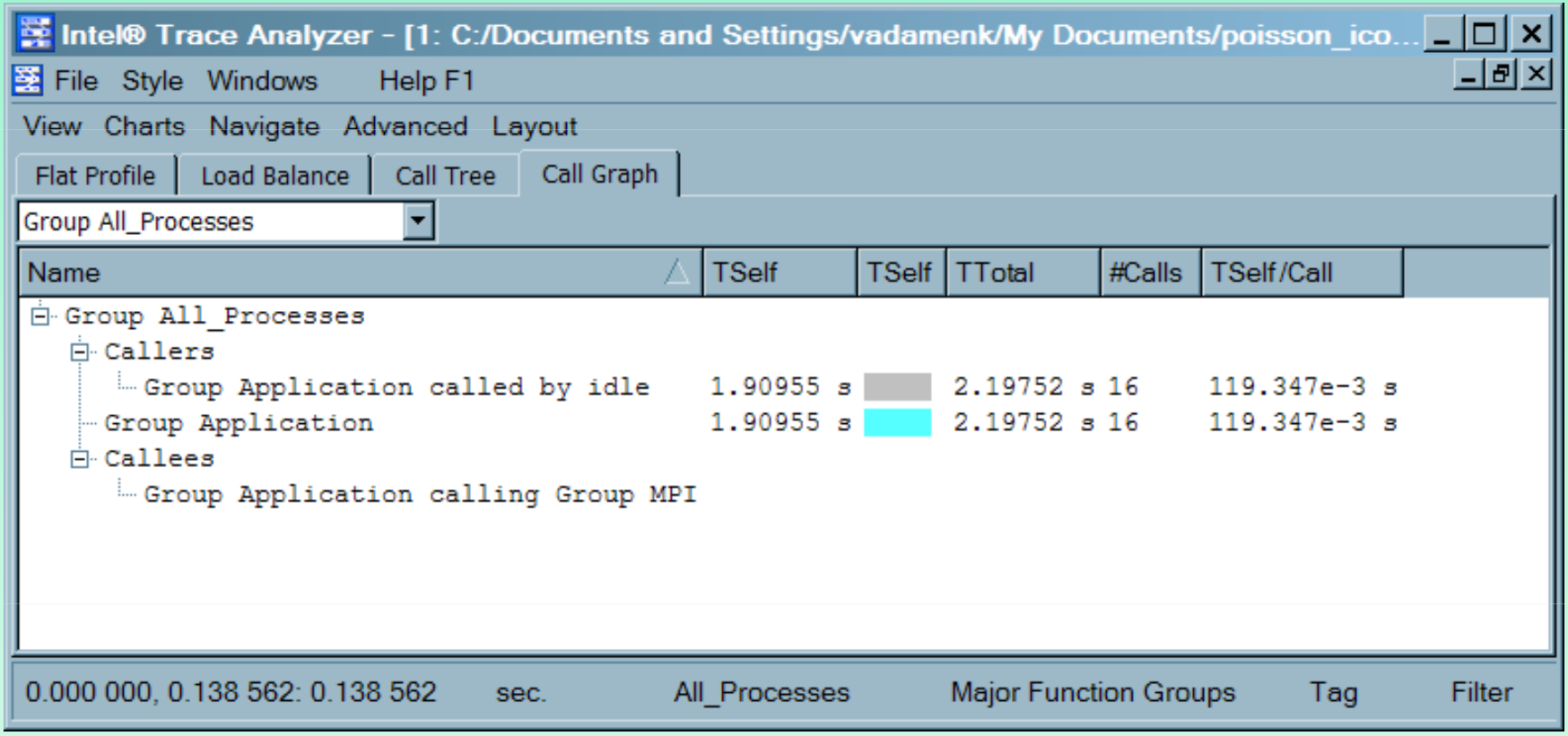
«Load Balance»



«Call Tree»


Intel® Trace Analyzer - [1: C:/Documents and Settings/vadamenk/My Documents/poisson_icomm.single...
File Style Windows Help F1
View Charts Navigate Advanced Layout
0.00 s
0.02 s
0.04 s
0.06 s
0.08 s
0.10 s
0.12 s
P0 Application
P1 Application
P2 Application
P3 Application
P4 Application
P5 Application
P6 Application
P7 Application
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
TTotal
#Calls
TSelf/Call
Group All_Processes
Group Application 1.90955 s 2.19752 s 16 119.347e-3 s
Group MPI 287.963e-3 s 287.963e-3 s 8560 33.6405e-6 s
0.000 000, 0.138 562: 0.138 562 sec.
All_Processes
Major Function Groups
Tag
Filter

«Call Graph»


Intel® Trace Analyzer - [1: C:/Documents and Settings/vadamenk/My Documents/poisson_ico...
File Style Windows Help F1
View Charts Navigate Advanced Layout
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name
TSelf
TSelf
TTotal
#Calls
TSelf/Call
Group All_Processes
Callers
Group Application called by idle 1.90955 s 2.19752 s 16 119.347e-3 s
Group Application 1.90955 s 2.19752 s 16 119.347e-3 s
Callees
Group Application calling Group MPI
0.000 000, 0.138 562: 0.138 562 sec.
All_Processes
Major Function Groups
Tag
Filter

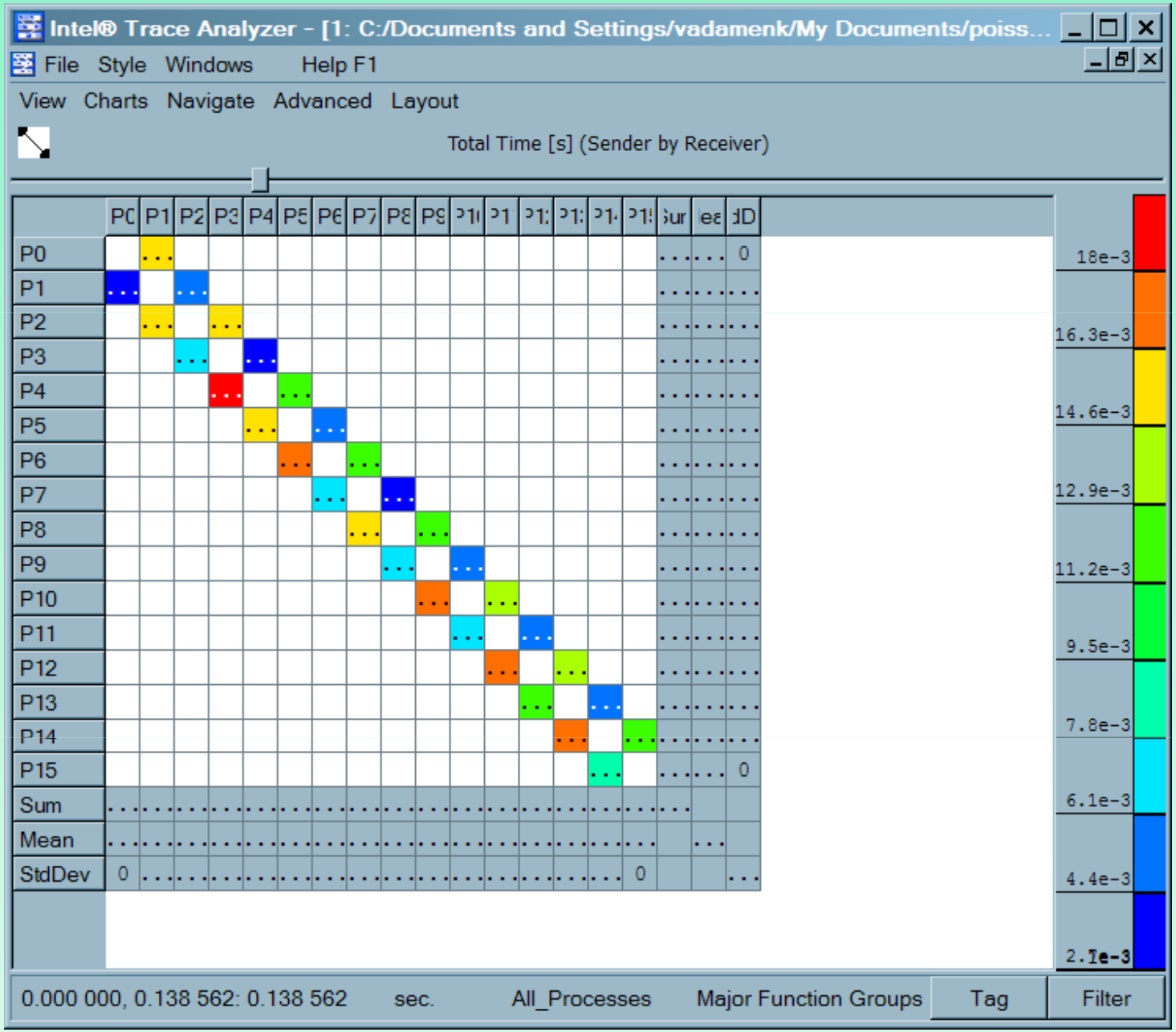
Профиль сообщений

Диаграмма **«The Message Profile»** отображает информацию об обменах в виде квадратной матрицы.

Строки матрицы соответствуют процессам-источникам сообщений.
Столбцы матрицы соответствуют процессам-приёмникам сообщений.
Каждая ячейка содержит суммарное время, затраченное на коммуникации между соответствующими процессами.

Приводятся также статистические характеристики по строкам и по столбцам.

Допускается группировка данных.


Intel® Trace Analyzer - [1: C:/Documents and Settings/vadamenk/My Documents/poiss...
File Style Windows Help F1
View Charts Navigate Advanced Layout
Total Time [s] (Sender by Receiver)
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 Sum Mean StdDev
18e-3
16.3e-3
14.6e-3
12.9e-3
11.2e-3
9.5e-3
7.8e-3
6.1e-3
4.4e-3
2.1e-3
0.000 000, 0.138 562: 0.138 562 sec.
All_Processes
Major Function Groups
Tag
Filter

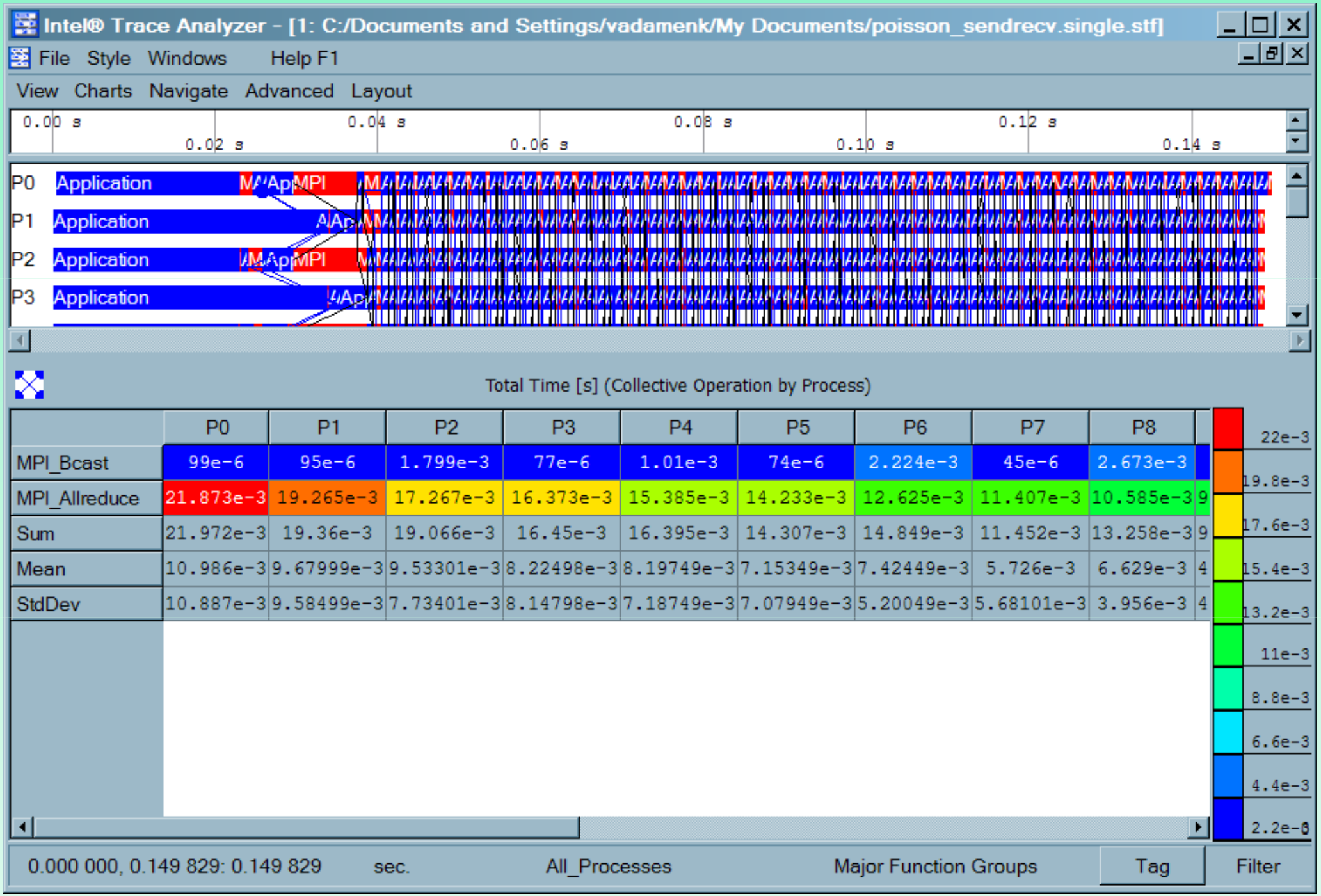
Профиль коллективных операций

Диаграмма **«The Collective Operations Profile»** отображает информацию о коллективных обменах в виде матрицы.

Строки матрицы соответствуют типам коллективных операций.
Столбцы матрицы соответствуют процессам-участникам обменов.
Каждая ячейка содержит суммарное время, затраченное на коммуникации между соответствующими процессами.

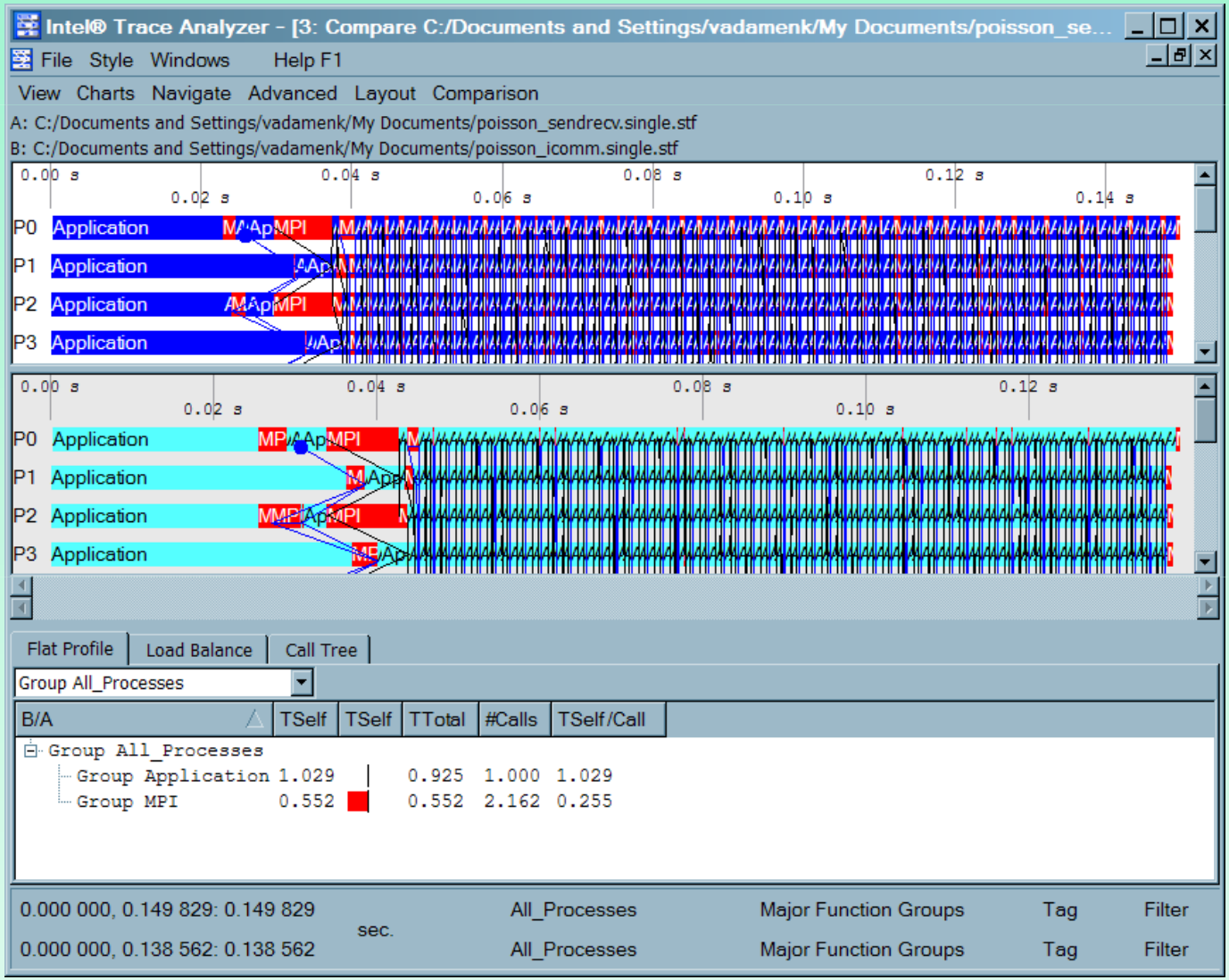
Приводятся также статистические характеристики по строкам и по столбцам.

Допускается группировка данных.


Intel® Trace Analyzer - [1: C:/Documents and Settings/vadamenk/My Documents/poisson_sendrecv.single.stf]
File Style Windows Help F1
View Charts Navigate Advanced Layout
0.00 s
0.02 s
0.04 s
0.06 s
0.08 s
0.10 s
0.12 s
0.14 s
P0 Application MPI
P1 Application
P2 Application MPI
P3 Application
Total Time [s] (Collective Operation by Process)
P0 P1 P2 P3 P4 P5 P6 P7 P8
MPI_Bcast 99e-6 95e-6 1.799e-3 77e-6 1.01e-3 74e-6 2.224e-3 45e-6 2.673e-3
MPI_Allreduce 21.873e-3 19.265e-3 17.267e-3 16.373e-3 15.385e-3 14.233e-3 12.625e-3 11.407e-3 10.585e-3 9
Sum 21.972e-3 19.36e-3 19.066e-3 16.45e-3 16.395e-3 14.307e-3 14.849e-3 11.452e-3 13.258e-3 9
Mean 10.986e-3 9.67999e-3 9.53301e-3 8.22498e-3 8.19749e-3 7.15349e-3 7.42449e-3 5.726e-3 6.629e-3 4
StdDev 10.887e-3 9.58499e-3 7.73401e-3 8.14798e-3 7.18749e-3 7.07949e-3 5.20049e-3 5.68101e-3 3.956e-3 4
22e-3
19.8e-3
17.6e-3
15.4e-3
13.2e-3
11e-3
8.8e-3
6.6e-3
4.4e-3
2.2e-3
0.000 000, 0.149 829: 0.149 829 sec.
All_Processes
Major Function Groups
Tag
Filter

Сравнение результатов трассировки

Окно просмотра **«The Comparison View»** (View Menu -> View -> Compare) позволяет сравнить данные из двух трассировочных файлов или из двух временных интервалов одного трассировочного файла.



Заключение

В этой лекции мы рассмотрели:

- динамический запуск процессов;
- операции одностороннего обмена;
- внешние интерфейсы;
- обобщённые коллективные обмены;
- другие возможности MPI-2;
- некоторые инструменты отладки и настройки MPI-приложений.