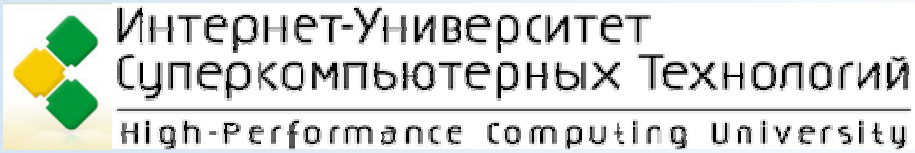
Основы параллельного программирования с использованием MPI

Лекция 7

Немнюгин Сергей Андреевич
Санкт-Петербургский государственный университет
кафедра вычислительной физики

snemnyugin@mail.ru



Лекция 7

Аннотация

В этой лекции мы познакомимся с управлением коммуникаторами. Затем нам предстоит краткое знакомство с программными инструментами динамического анализа MPI-программ.

План лекции

- Управление коммуникаторами.
- Программные инструменты динамического анализа параллельных MPI-программ.

Коллективные обмены

Управление коммуникаторами

Создание коммуникатора — коллективная операция и соответствующая подпрограмма должна вызываться всеми процессами коммуникатора. Подпрограмма `MPI_Comm_dup` дублирует уже существующий коммуникатор `oldcomm`:

```
int MPI_Comm_dup(MPI_Comm oldcomm, MPI_Comm *newcomm)

MPI_Comm_dup(oldcomm, newcomm, ierr)
```

В результате вызова создается новый коммуникатор (`newcomm`) с той же группой процессов, с теми же атрибутами, но с другим контекстом.

Эта операция может применяться как к интра-, так и к интеркоммуникаторам.

Подпрограмма `MPI_Comm_create` создает новый коммуникатор (`newcomm`) из подмножества процессов (`group`) другого коммуникатора (`oldcomm`):

```
int MPI_Comm_create(MPI_Comm oldcomm, MPI_Group group, MPI_Comm *newcomm)

MPI_Comm_create(oldcomm, group, newcomm, ierr)
```

Вызов этой подпрограммы должны выполнить все процессы из старого коммуникатора, даже если они не входят в группу `group`, с одинаковыми аргументами.

Данная операция применяется только к интракоммуникаторам. Она позволяет выделять подмножества процессов со своими областями взаимодействия, если, например, требуется уменьшить «зернистость» параллельной программы. Побочным эффектом применения подпрограммы `MPI_Comm_create` является синхронизация процессов. Если одновременно создаются несколько коммуникаторов, они должны создаваться в одной последовательности всеми процессами.

Пример

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 char message[24];
 MPI_Group MPI_GROUP_WORLD;
 MPI_Group group;
 MPI_Comm fcomm;
 int size, q, proc;
 int* process_ranks;
 int rank, rank_in_group;
 MPI_Status status;

 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 printf("New group contains processes:");
 q = size – 1;
 process_ranks = (int*) malloc(q*sizeof(int));
 for (proc = 0; proc < q; proc++)
{
 process_ranks[proc] = proc;
 printf("%i ", process_ranks[proc]);
 }
```

```
printf("\n");
 MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
 MPI_Group_incl(MPI_GROUP_WORLD, q, process_ranks, &group);
 MPI_Comm_create(MPI_COMM_WORLD, group, &fcomm);

if (fcomm != MPI_COMM_NULL) {
 MPI_Comm_group(fcomm, &group);
 MPI_Comm_rank(fcomm, &rank_in_group);
 if (rank_in_group == 0) {
 strcpy(message, "Hi, Parallel Programmer!");
 MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
 printf("0 send: %s\n", message);
 }
else
{
 MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
 printf("%i received: %s\n", rank_in_group, message);
}

 MPI_Comm_free(&fcomm);
 MPI_Group_free(&group);
}
 MPI_Finalize();
 return 0;
}
```

Как работает эта программа

Пусть в коммуникатор `MPI_COMM_WORLD` входят `p` процессов.

Сначала создается список процессов, которые будут входить в область взаимодействия нового коммуникатора.

Затем создается группа, состоящая из этих процессов. Для этого требуются две операции. Первая определяет группу, связанную с коммуникатором `MPI_COMM_WORLD`.

Новая группа создается вызовом подпрограммы `MPI_Group_incl`.

Затем создается новый коммуникатор. Для этого используется подпрограмма `MPI_Comm_create`. Новый коммуникатор — `fcomm`.

В результате этих действий все процессы, входящие в коммуникатор `fcomm`, смогут выполнять операции коллективного обмена, но только между собой.

```
[nemnugin@pd00 ~]$ mpiexec -n 8 ./a.out
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
New group contains processes:0 1 2 3 4 5 6
0 send: Hi, Parallel Programmer!
2 received: Hi, Parallel Programmer!
1 received: Hi, Parallel Programmer!
6 received: Hi, Parallel Programmer!
3 received: Hi, Parallel Programmer!
5 received: Hi, Parallel Programmer!
4 received: Hi, Parallel Programmer!
[nemnugin@pd00 ~]$
```

В следующем примере процессы разбиваются на две группы. Одна содержит процессы с чётными рангами, а другая – с нечётными.

```
#include "stdio.h"
#include "mpi.h"

void main(int argc, char *argv[])
{
        int num, p;
        int Neven, Nodd, members[6], even_rank, odd_rank;
        MPI_Group group_world, even_group, odd_group;

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

        Neven = (p + 1)/2;
        Nodd = p - Neven;
        members[0] = 2;
        members[1] = 0;
        members[2] = 4;
```

```
        MPI_Comm_group(MPI_COMM_WORLD, &group_world);
        MPI_Group_incl(group_world, Neven, members, &even_group);
        MPI_Group_excl(group_world, Neven, members,  &odd_group);
        MPI_Barrier(MPI_COMM_WORLD);
        if(num == 0) {
         printf("Number of processes is %d\n", p);
         printf("Number of odd processes is %d\n", Nodd);
         printf("Number of even processes is %d\n", Neven);
         printf("members[0] is assigned rank %d\n", members[0]);
         printf("members[1] is assigned rank %d\n", members[1]);
         printf("members[2] is assigned rank %d\n", members[2]);
         printf("\n");
         printf("     num     even      odd\n");
        }

        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Group_rank(even_group, &even_rank);
        MPI_Group_rank( odd_group,  &odd_rank);
        printf("%8d %8d %8d\n",num, even_rank, odd_rank);
        MPI_Finalize();
}
```

```
[nemnugin@pd00 ~]$ mpiexec -n 4 ./a.out
Number of processes is 4
Number of odd processes is 2
Number of even processes is 2
members[0] is assigned rank 2
members[1] is assigned rank 0
members[2] is assigned rank 4

     Iam     even      odd
       0        1   -32766
       2        0   -32766
       1   -32766        0
       3   -32766        1
[nemnugin@pd00 ~]$
```

Подпрограмма `MPI_Comm_free` помечает коммуникатор `comm` для удаления:

```
int MPI_Comm_free(MPI_Comm *comm)

MPI_Comm_free(comm, ierr)
```

Обмены, связанные с этим коммуникатором, завершаются обычным образом, а сам коммуникатор удаляется только после того, как на него не будет активных ссылок. Данная операция может применяться к коммуникаторам обоих видов (интра- и интер-).

Сравнение двух коммуникаторов (comm1) и (comm2) выполняется подпрограммой MPI_Comm_compare:

```
int MPI_Comm_compare(MPI_Comm comm1, MPI_Comm comm2, int *result)

MPI_Comm_compare(comm1, comm2, result, ierr)
```

Результат ее выполнения result — целое значение, которое равно MPI_IDENT, если контексты и группы коммуникаторов совпадают; MPI_CONGRUENT, если совпадают только группы;
MPI_UNEQUAL, если не совпадают ни группы, ни контексты.

В качестве аргументов нельзя использовать пустой коммуникатор MPI_COMM_NULL.

К числу операций управления коммуникаторами можно отнести операции MPI_Comm_size и MPI_Comm_rank. Они позволяют, в частности, распределить роли между процессами в модели "главная задача — подчиненные задачи" (master-slave).

С помощью подпрограммы `MPI_Comm_set_name` можно присвоить коммуникатору `comm` строковое имя `name`:

```
int MPI_Comm_set_name(MPI_Comm com, char *name)

MPI_Comm_set_name(com, name, ierr)
```

`MPI_Comm_get_name` возвращает `name` — строковое имя коммуникатора `comm`:

```
int MPI_Comm_get_name(MPI_Comm comm, char *name, int *reslen)

MPI_Comm_get_name(comm, name, reslen, ierr)
```

Имя представляет собой массив символьных значений, длина которого должна быть не менее `MPI_MAX_NAME_STRING`. Длина имени — выходной параметр `reslen`.

Доступ к удаленной группе, связанной с интеркоммуникатором `comm`, можно получить, обратившись к подпрограмме:

```
int MPI_Comm_remote_group(MPI_Comm comm, MPI_Group *group)

MPI_Comm_remote_group(comm, group, ierr)
```

Ее выходным параметром является удаленная группа `group`.

Подпрограмма `MPI_Comm_remote_size` определяет размер удаленной группы, связанной с интеркоммуникатором `comm`:

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

MPI_Comm_remote_size(comm, size, ierr)
```

Ее выходной параметр `size` — количество процессов в области взаимодействия, связанной с коммуникатором `comm`.

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 анализ многопоточных Java-приложений.

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 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

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

Запуск файла для трассировки MPI-приложения

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.

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

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.

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.

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

Счетчики

Сбор данных о производительности процессора

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

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 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!

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

Intel® Trace Analyzer

Intel® Trace Analyzer

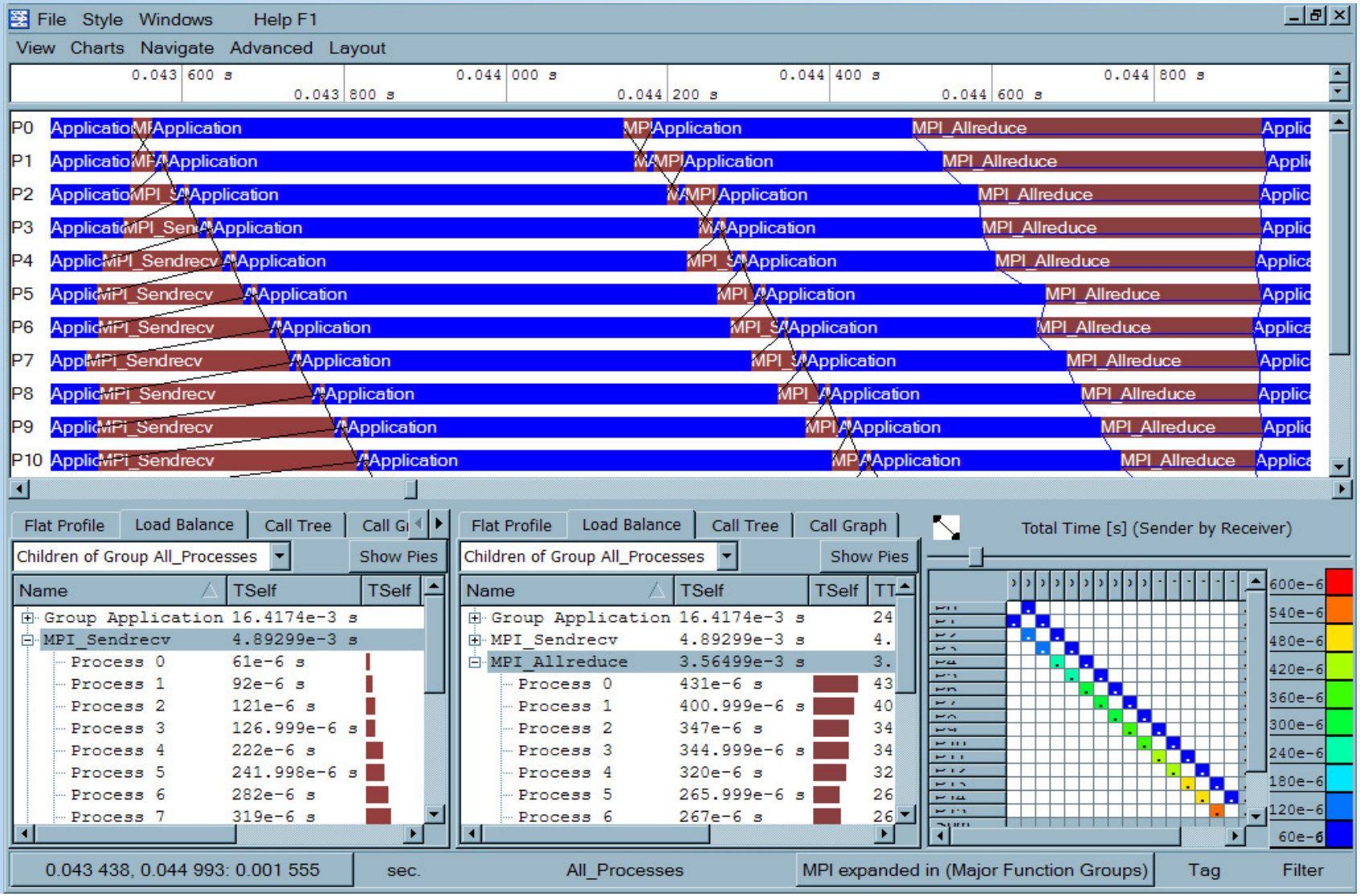
Запуск анализатора:

```
#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.

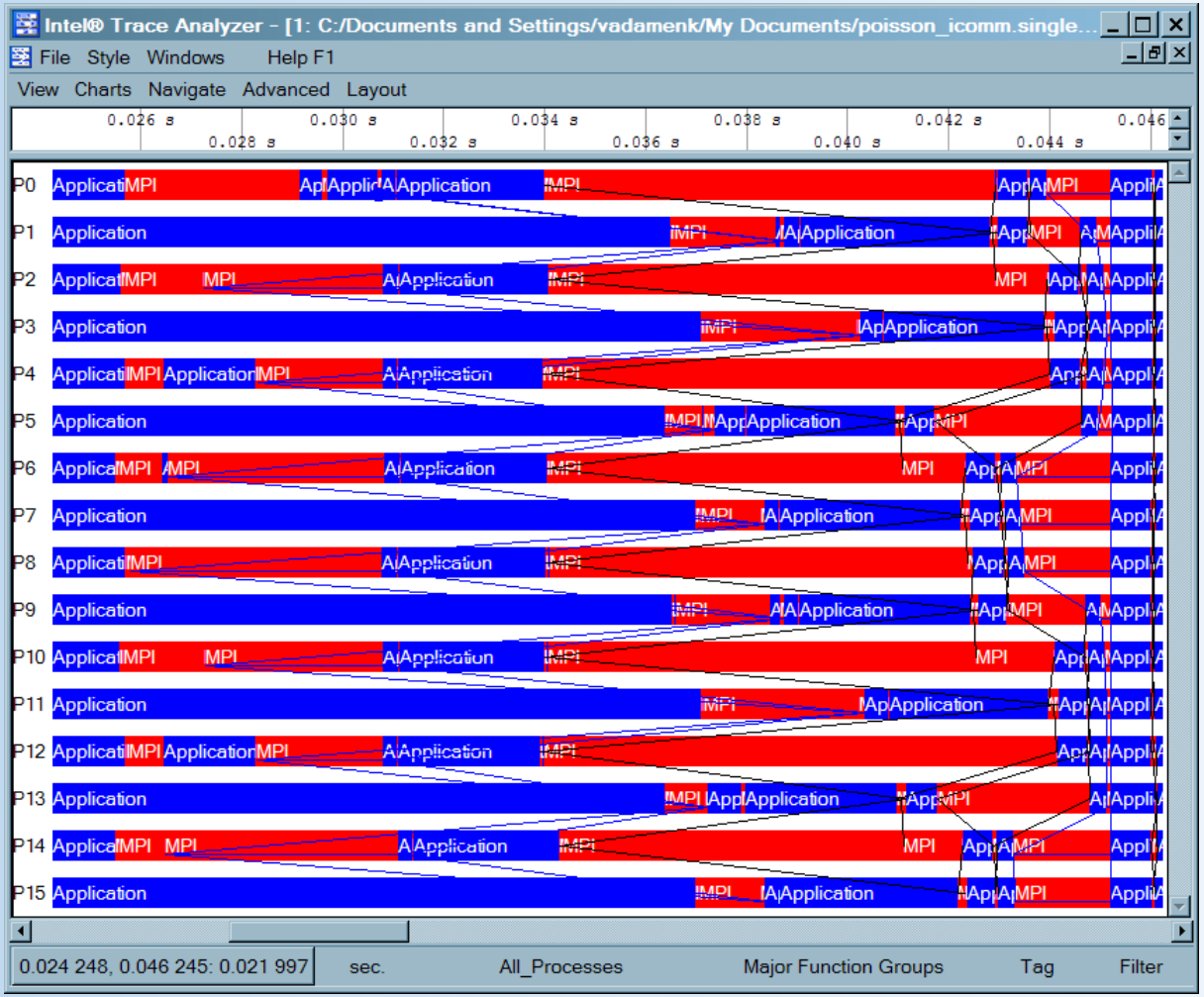
Process Aggregation
Объединение результатов трассировки по группам процессов.

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.

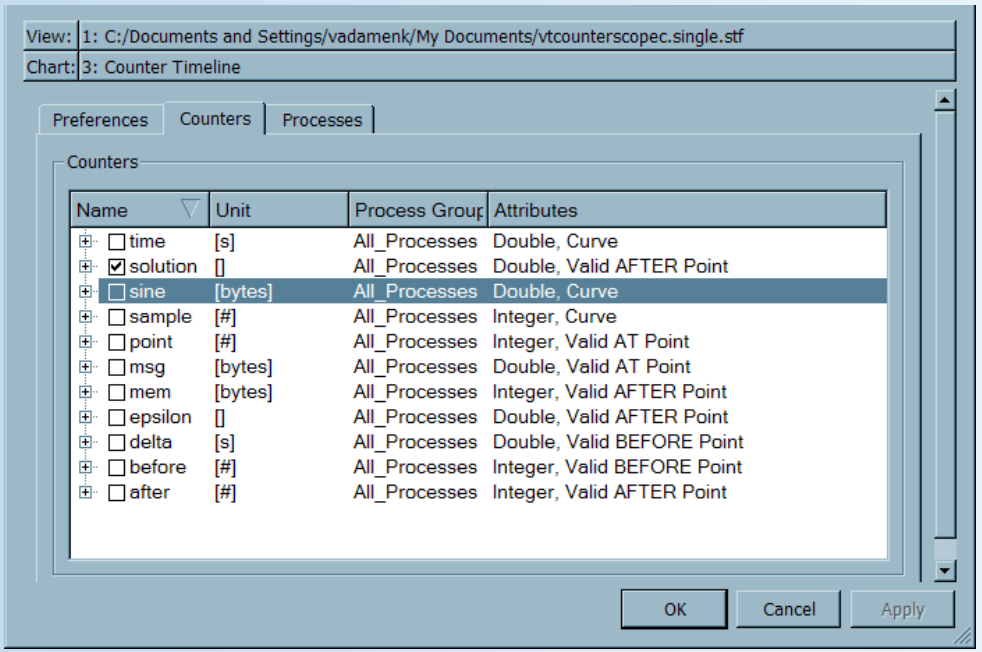
Чёрными линиями отображаются операции двухточечного обмена.

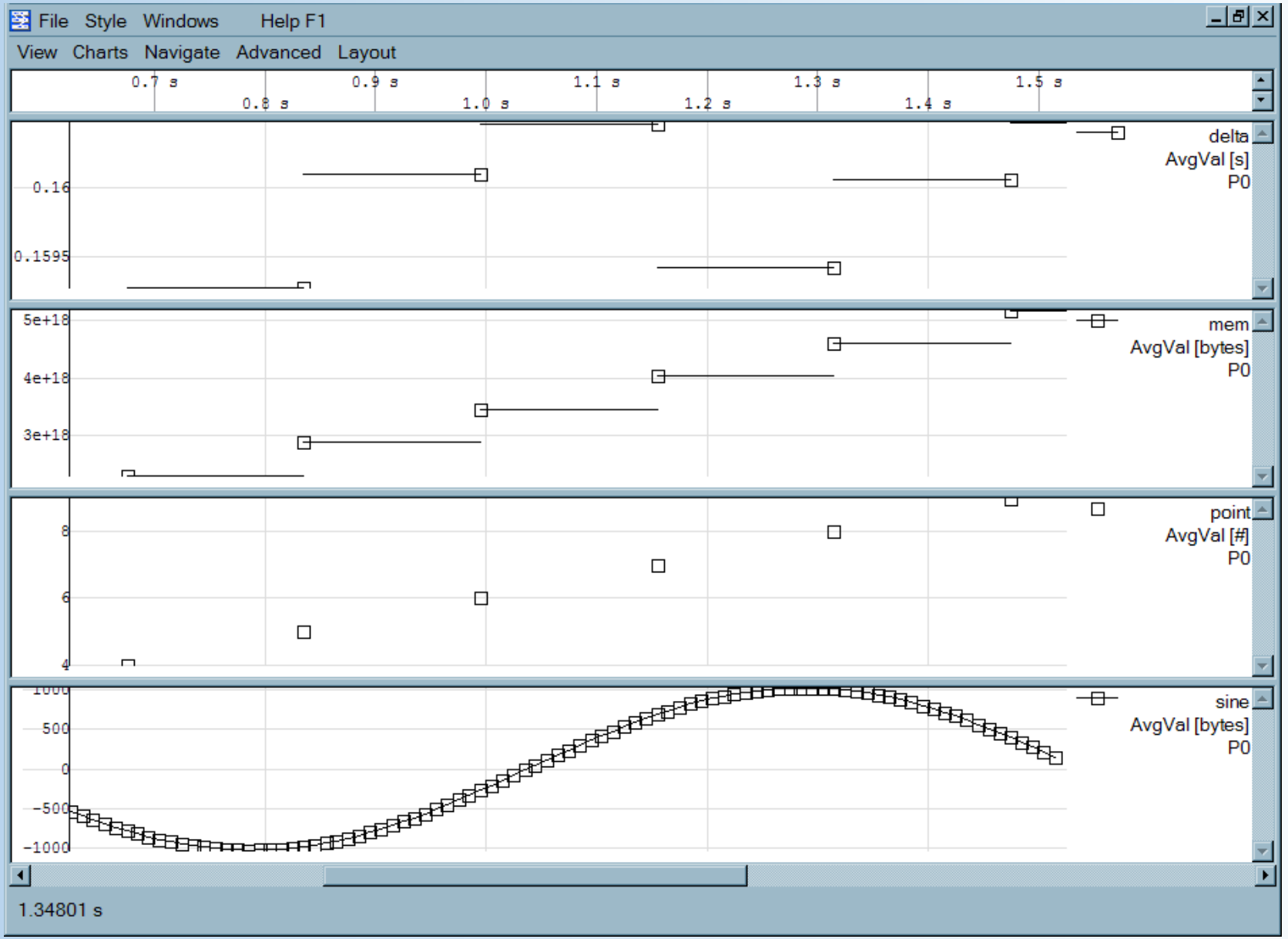
Синими линиями отображаются операции коллективного обмена.

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
0.16
0.1595
delta
AvgVal [s]
P0
5e+18
4e+18
3e+18
mem
AvgVal [bytes]
P0
8
6
4
point
AvgVal [#]
P0
1000
500
0
-500
-1000
sine
AvgVal [bytes]
P0
1.34801 s

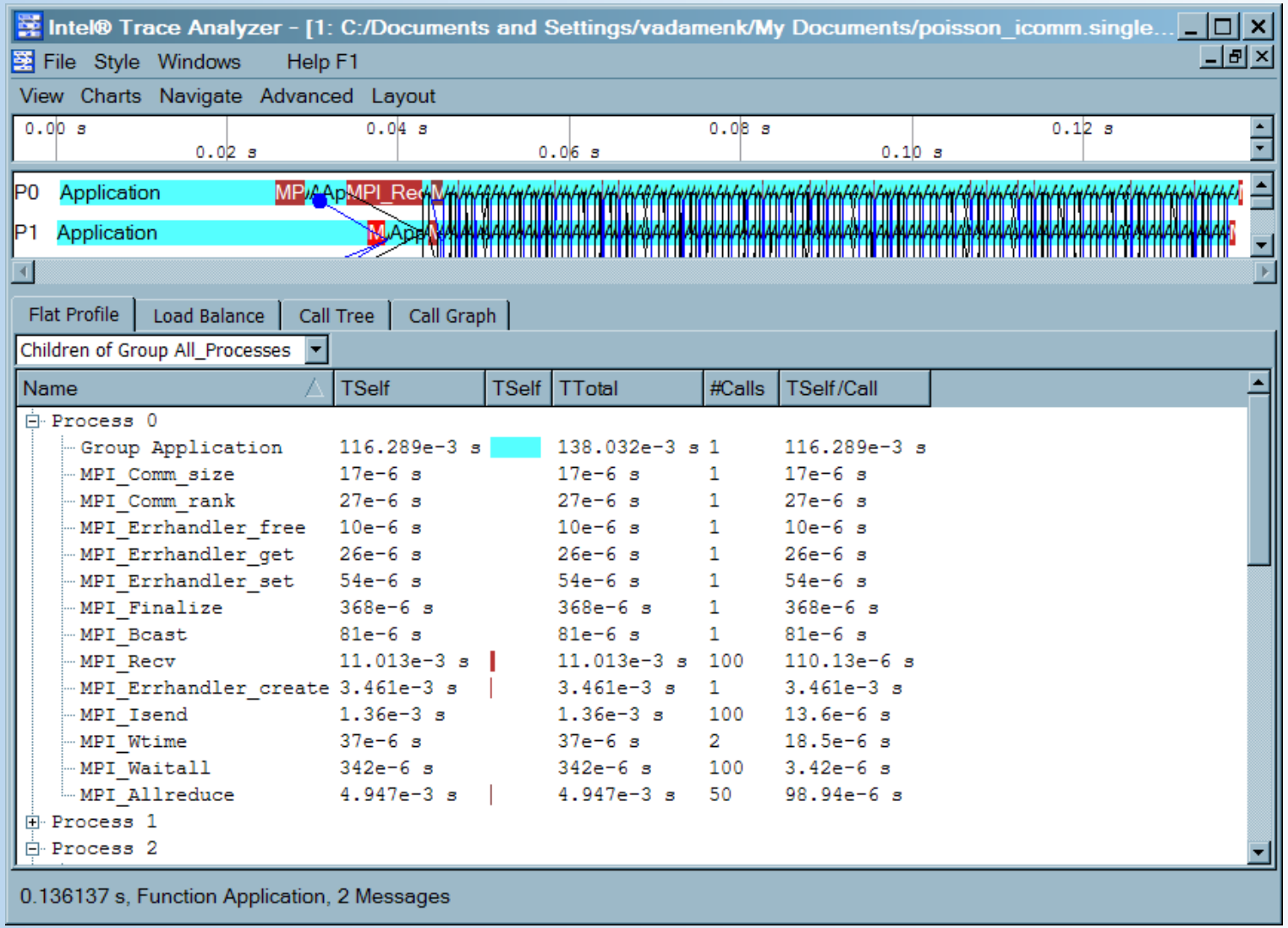
Профиль функций

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

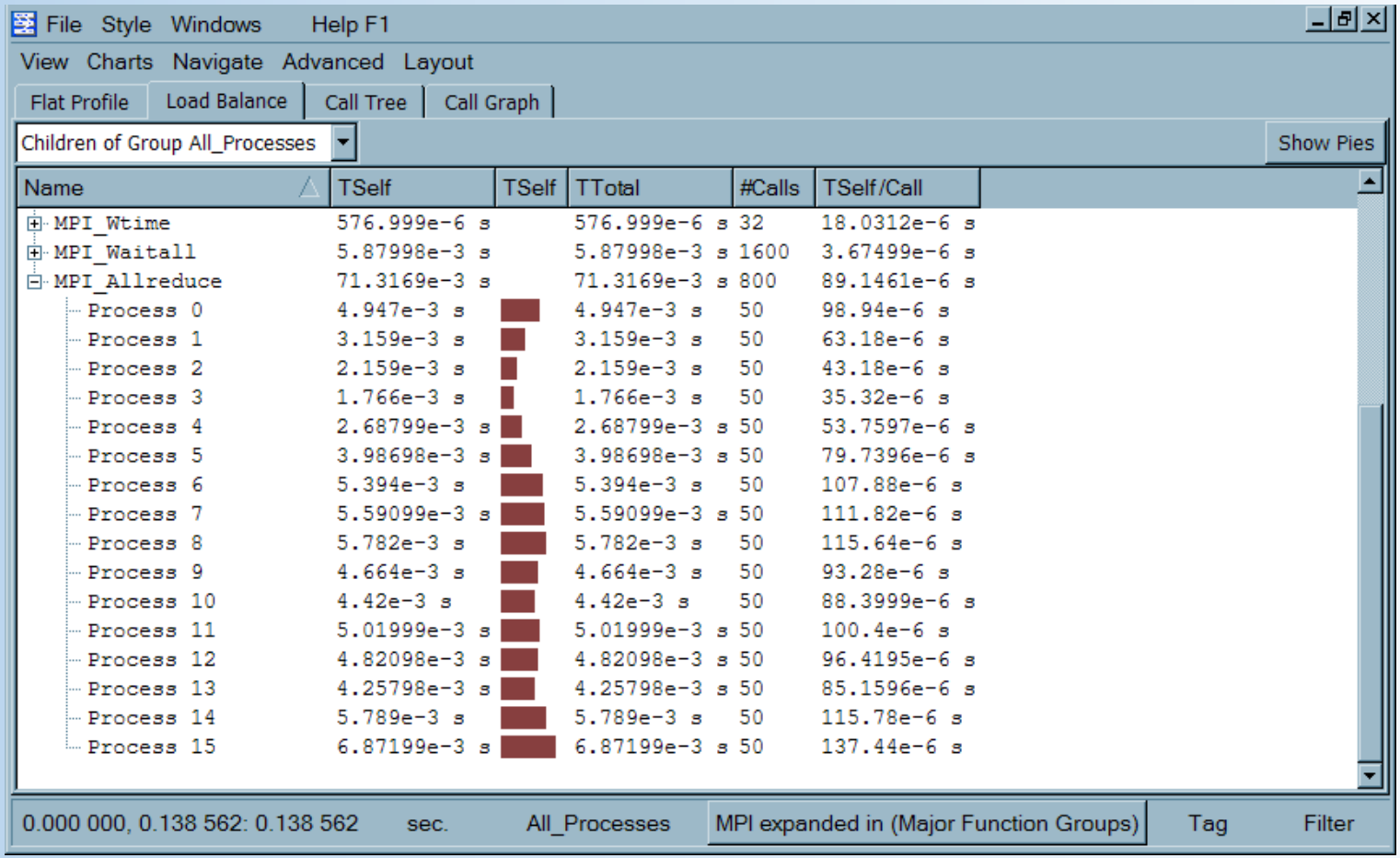
Содержит вкладки:

- 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).

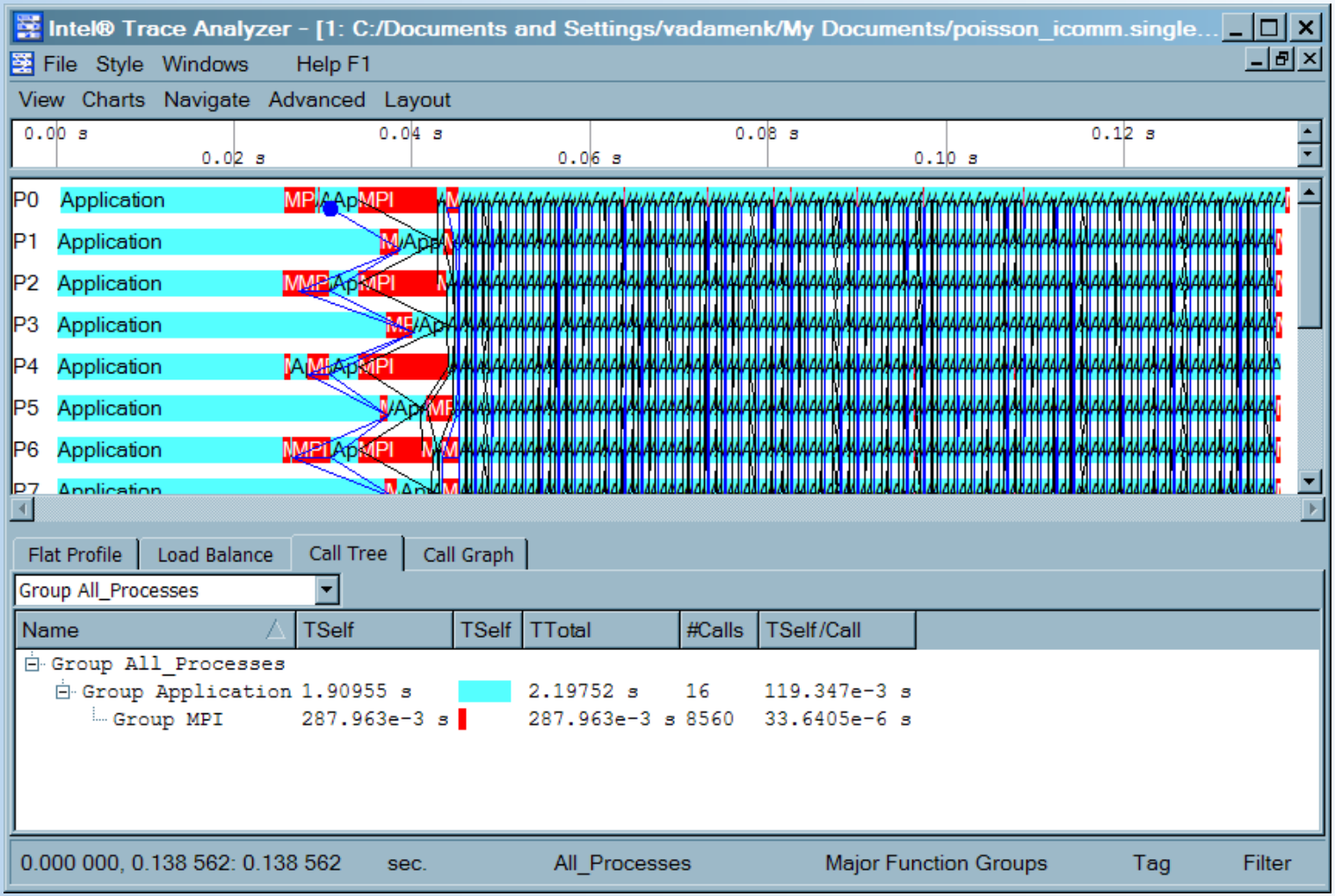
«Flat Profile»



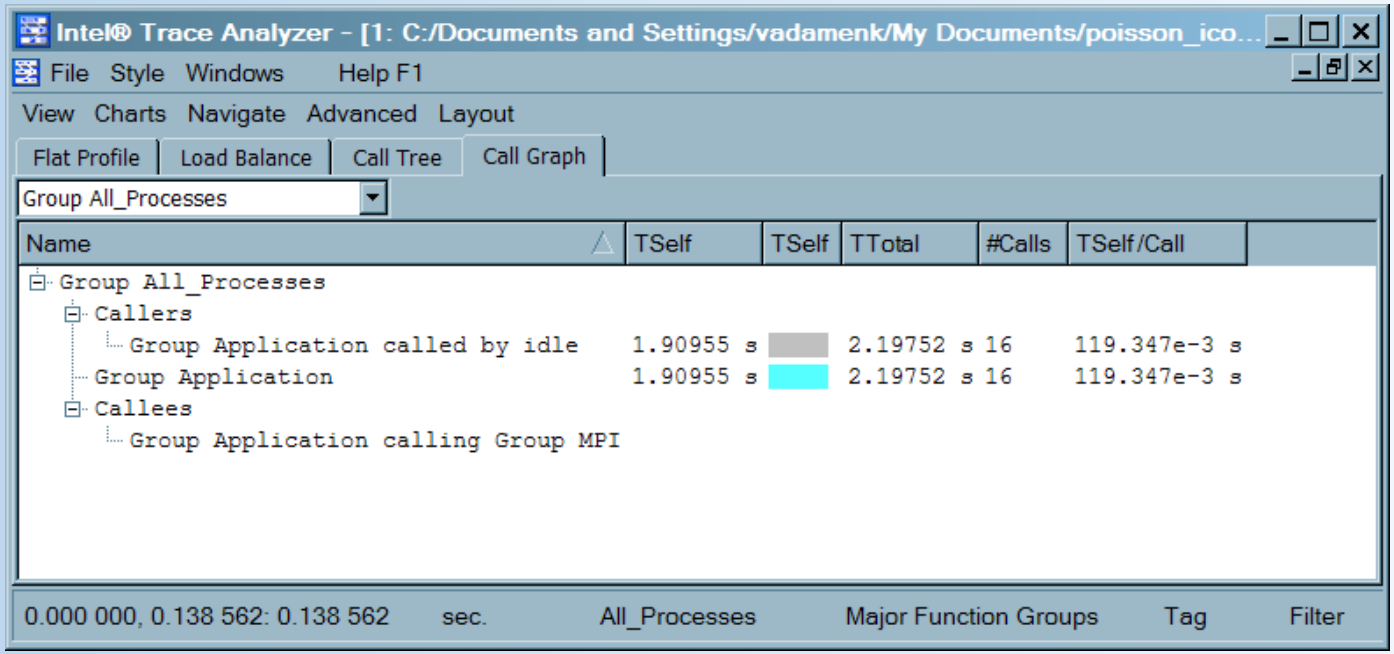
«Load Balance»



«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
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.

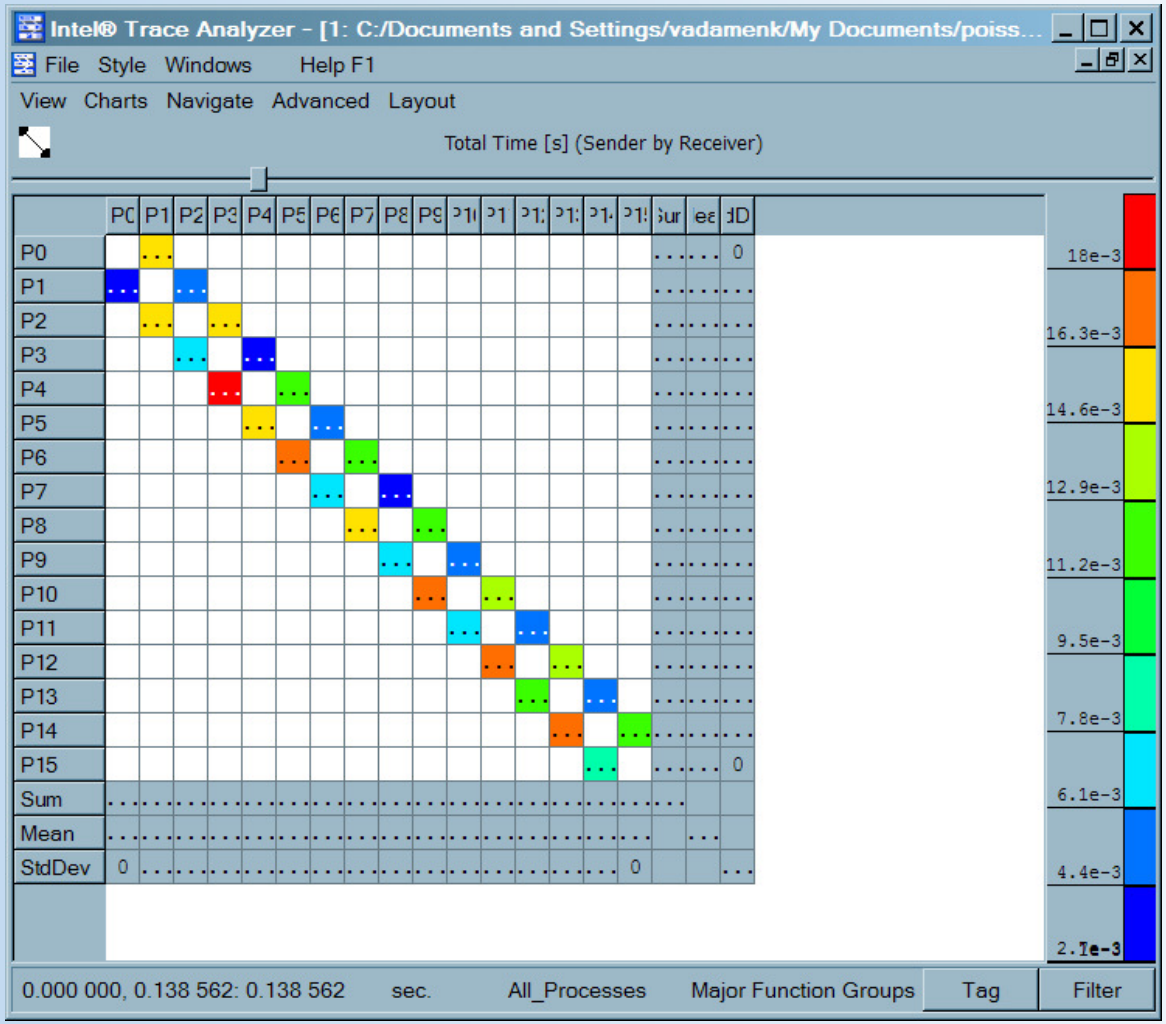
Строки матрицы соответствуют процессам-источникам сообщений.

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.

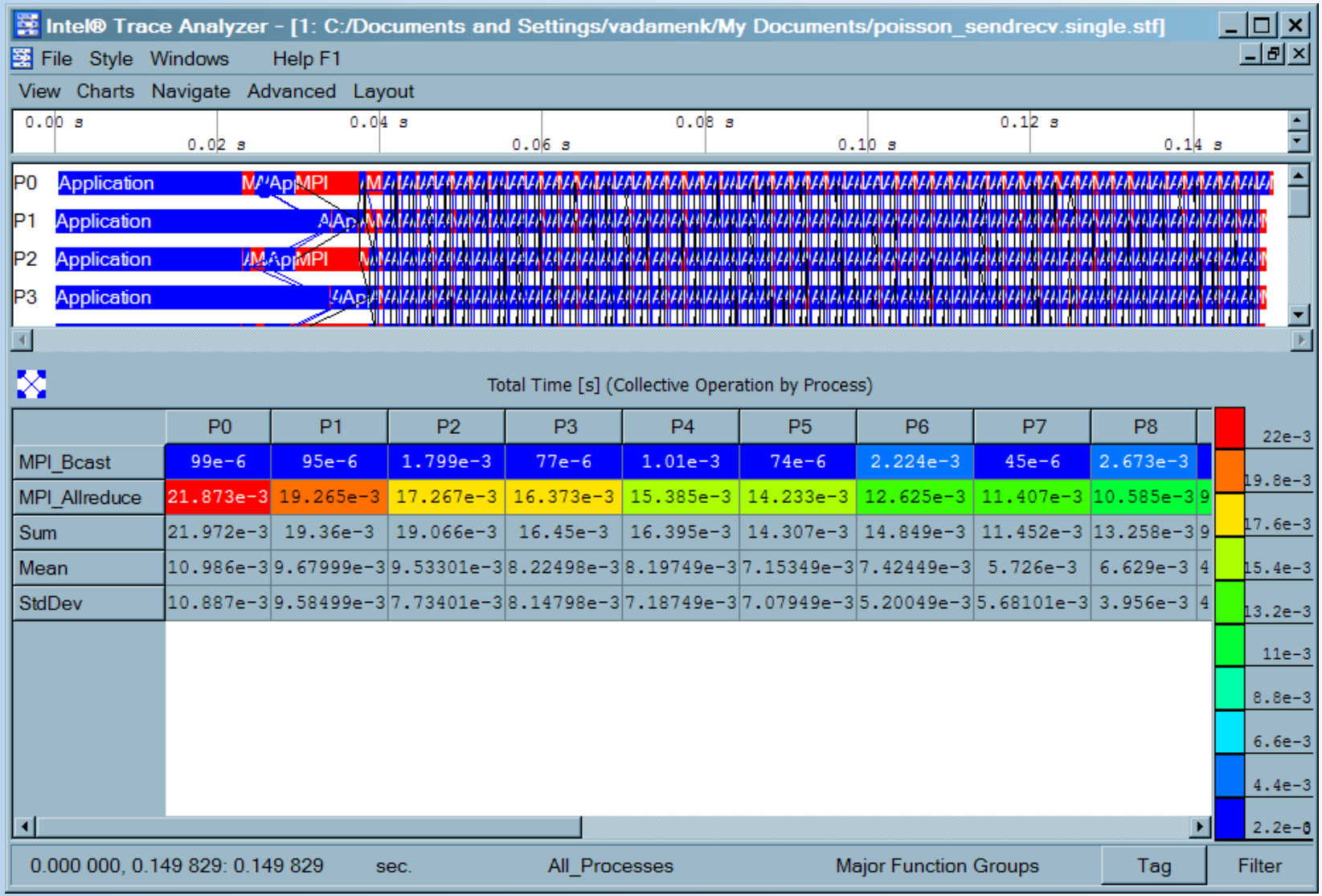
Строки матрицы соответствуют типам коллективных операций.

Столбцы матрицы соответствуют процессам-участникам обменов.

Каждая ячейка содержит суммарное время, затраченное на коммуникации между соответствующими процессами.

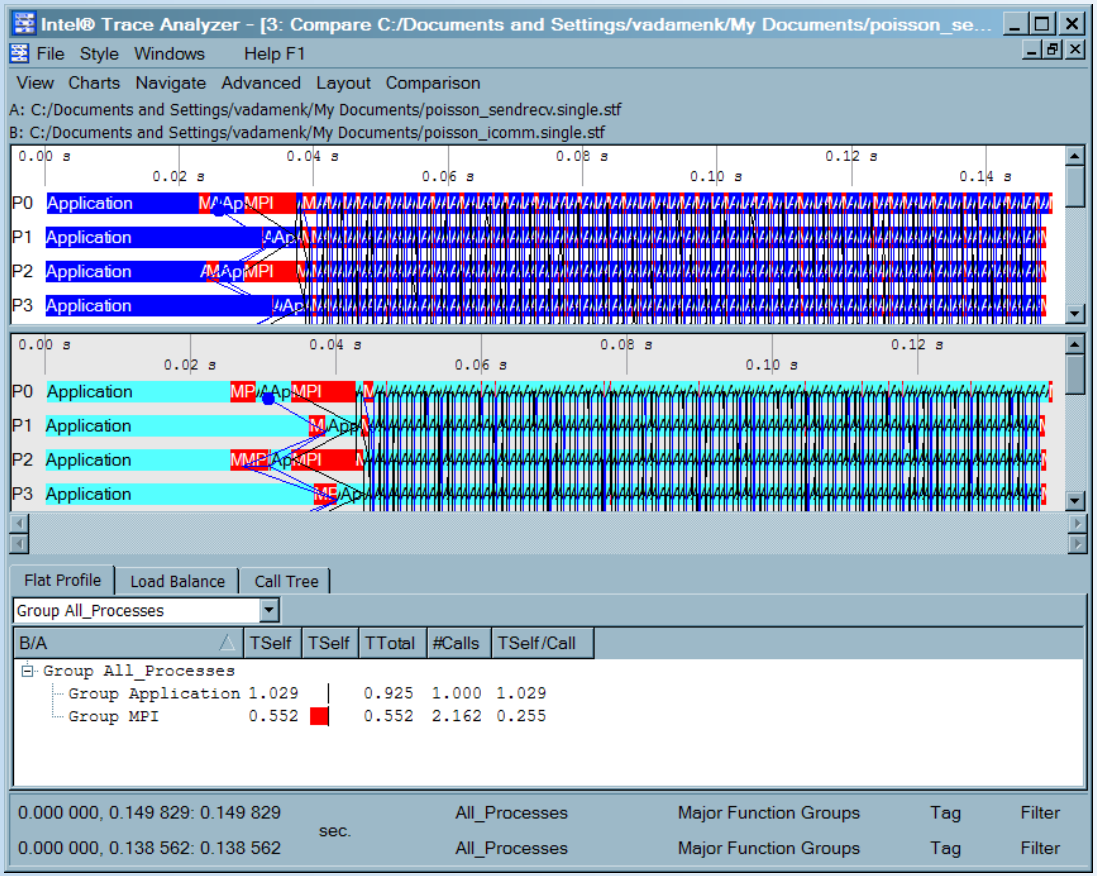
Приводятся также статистические характеристики по строкам и по столбцам.

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:

- управление коммуникаторами;
- программные инструменты динамического анализа параллельных программ.