МО ВВС ИВМиМГ СО РАН

Основы параллельного программирования

Лабораторная работа

Профилирование MPI программ

2024

1. Описание

1.1. Введение.

1.1.1. Цели профилирования программ

Написание эффективной параллельной программы имеет существенно более высокую трудоемкость по сравнению с написанием эффективной последовательной программы. Перед тем как начать повышать производительность программы, т.е. оптимизировать ее, необходимо выявить ее узкие места, критические участки, которые потребляют большую часть времени исполнения. Оптимизировать нужно именно их. Оптимизация других частей программы не только приводит к неэффективной трате времени разработчика программы, так как результаты такой оптимизации будут мало заметны, но, что еще хуже, они делают программу более запутанной, усложняя ее отладку и сопровождение.

Существенную помощь в выявлении таких критических участков параллельной программы может оказать ее профилирование. Профилирование – анализ программы в момент ее исполнения. При профилировании производится сбор статистики о частоте вызова функций, времени, затрачиваемом на их исполнение, размере требуемой памяти. Полученная при профилировании информация в первую очередь используется для оптимизации программ.

1.1.2. Методы профилирования программ

Существуют различные способы реализации профайлеров. Главные из них:

1) статистические профайлеры (в регулярные моменты по прерыванию времени происходит определение текущей исполняемой функции),
2) профайлеры с инструментированием (в код каждой функции добавляются команды, учитывающие количество обращений к ней и время ее исполнения).

Postmortem profiling. Один из возможных методов использования результатов профилирования параллельной программы основан на генерации трассировочных файлов при исполнении программы и их последующего анализа и визуализации с целью выявления частей программы, занимающих большую часть времени исполнения.

1.2. Инструменты профилирования MPI программ

1.2.1. MultiProcessing Environment

MultiProcessing Environment (MPE) - инструмент для анализа производительности MPI программ. Он основан на генерации трассировочного файла в процессе исполнения программы и его последующем анализе (postmortem profiling).

MPE содержит следующие компоненты:
1) набор профилирующих библиотек для сбора информации о поведении MPI программ;

2) скрипт для компиляции MPI программ с поддержкой MPE (*mpecc* для программ на C);
3) визуализатор трассировочных файлов Jumpshot;
4) утилиты по работе с трассировочными файлами;
5) библиотека проверки коллективных операций и типов данных;
6) генератор исходных текстов функций интерфейса MPI с поддержкой профилирования:
7) подпрограммы настройки отладчика.

Установка MPE под Ubuntu Linux с MPICH2 для программ на C состоит из следующих шагов:
1) Со страницы *http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm* скачивается и распаковывается архив с исходными текстами *mpe2.tar.gz (ссылка на сайте использует протоколом ftp, если не получается скачать пробуйте http: http://ftp.mcs.anl.gov/pub/mpi/mpe/mpe2.tar.gz, также имеется копия на сайте кафедры по адресу https://ssd.sscc.ru/sites/default/files/content/attach/343/mpe2.tgz)*.
2) запускается утилита configure:
./configure --with-mpicc=mpicc --disable-f77
3) запускается компиляция и сборка MPE:
make all
4) запускается скрипт установки:
sudo make install

После успешной установки становится доступной команда компиляции MPI программ с поддержкой профилирования mpecc. Для сборки MPI программы, генерирующей трассировочный файл **в режиме автоматического профилирования** можно использовать команду:
mpecc -mpilog -o <имя_файла> <имя_файла>.c

Для сборки MPI программы, генерирующей трассировочный файл **в режиме настраиваемого (customized) профилирования** можно использовать команду:
mpecc -log -o <имя_файла> <имя_файла>.c

C++: Если вы программируете в C++, то вам следует в скрипте mpecc заменить mpicc на mpicxx или другим образом обеспечить вызов соответствующего компилятора из скрипта mpecc.

Скомпилированная программа запускается так же, как и обычная MPI программа (скриптами mpirun, mpiexec). В процессе исполнения программы в режиме автоматического профилирования для каждой вызываемой MPI функции с помощью *MPI_Wtime* замеряется время перед вызовом и после него. MPE формирует трассировочный файл с расширением *clog2* (если не указано иное с помощью переменных среды окружения), в котором отмечаются времена вызовов MPI функций. Суммарное время исполнения каждой функции накапливается для каждого MPI процесса и сохраняется в файл при вызове *MPI_Finalize*.

Для настраиваемого профилирования в программе необходимо в явном виде произвести инициализацию и деинициализацию профайлера (функции *MPE_Init_log*, *MPE_Finish_log*) получить идентификаторы событий (*MPE_Log_get_event_number*),

описать состояния (промежутки между начальным и конечным событиями), которые необходимо исследовать (*MPE_Describe_state*) и сообщать о наступлении события (*MPE_Log_event*). Пример программы с встроенным кодом для настраиваемого профилирования приведен на листинге 1. Вычисления в программе – фиктивные, они имитируются задержкой с помощью системного вызова *sleep*.

```
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>
#include <mpe.h>

main(int argc, char **argv){
int rankNode, sizeCluster;
int evtid_beginPhase1, evtid_endPhase1;
int evtid_beginPhase2, evtid_endPhase2;

MPI_Init(&argc, &argv);

MPE_Init_log();

MPI_Comm_rank(MPI_COMM_WORLD, &rankNode);
MPI_Comm_size(MPI_COMM_WORLD, &sizeCluster);

evtid_beginPhase1 = MPE_Log_get_event_number();
evtid_endPhase1 = MPE_Log_get_event_number();
evtid_beginPhase2 = MPE_Log_get_event_number();
evtid_endPhase2 = MPE_Log_get_event_number();

printf("Starting, number of nodes: %d, rank of node: %d\n", sizeCluster, rankNode);

MPE_Describe_state(evtid_beginPhase1, evtid_endPhase1, "Phase1", "red");
MPE_Describe_state(evtid_beginPhase2, evtid_endPhase2, "Phase2", "blue");

MPE_Log_event(evtid_beginPhase1, rankNode, (char*)0);
printf("Phase 1, number of nodes: %d, rank of node: %d\n", sizeCluster, rankNode);
sleep((rankNode + 1) * 10);
MPE_Log_event(evtid_endPhase1, rankNode, (char*)0);

MPE_Log_event(evtid_beginPhase2, rankNode, (char*)0);
printf("Phase 2, number of nodes: %d, rank of node: %d\n", sizeCluster, rankNode);
sleep(((1 - rankNode) + 1) * 10);
MPE_Log_event(evtid_endPhase2, rankNode, (char*)0);

printf("Shutting down, number of nodes: %d, rank of node: %d\n", sizeCluster, rankNode);
MPE_Finish_log("tutor.3.clog2");
```

```
MPI_Finalize();
}
```

Листинг 1. Программа с измерением времени выполнения ее двух фаз вычислений.

1.2.2. Jumpshot

Пакет Jumpshot служит для обработки и визуализации трассировочных файлов в форматах *slog2*, *clog2* и прочих.

Установка Jumpshot под Ubuntu Linux состоит из следующих шагов:
1) Со страницы загрузок проекта Performance Visualization for Parallel Programs http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm скачивается и распаковывается архив с исходными текстами *slog2sdk.tar.gz (ссылка на странице использует протокол ftp, если не получается скачать файл, попробуйте http: ftp.mcs.anl.gov/pub/mpi/slog2/slog2sdk.tar.gz)*.
2) запускается утилита configure:
./configure
3) запускается скрипт установки:
sudo make install

Установка Jumpshot под Win32 состоит из следующих шагов:
1) Установить на машину Java 2 Runtime Environment.
2) Со страницы загрузок проекта Performance Visualization for Parallel Programs http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm скачивается и распаковывается архив *slog2rte.tar.gz (ссылка на странице использует протокол ftp, если не получается скачать, пробуйте http: ftp.mcs.anl.gov/pub/mpi/slog2/slog2rte.tar.gz* .

Запуск среды визуализации осуществляется с помощью команды *jumpshot* в Linux или *jumpshot_launcher.jar* в поддиректории *lib* в Win32. При первом запуске пользователю предлагается создать каталог с настройками. Далее в диалоге выбора файла необходимо выбрать сформированный в результате прогона MPI программы *clog2* файл (см. рис. 1).

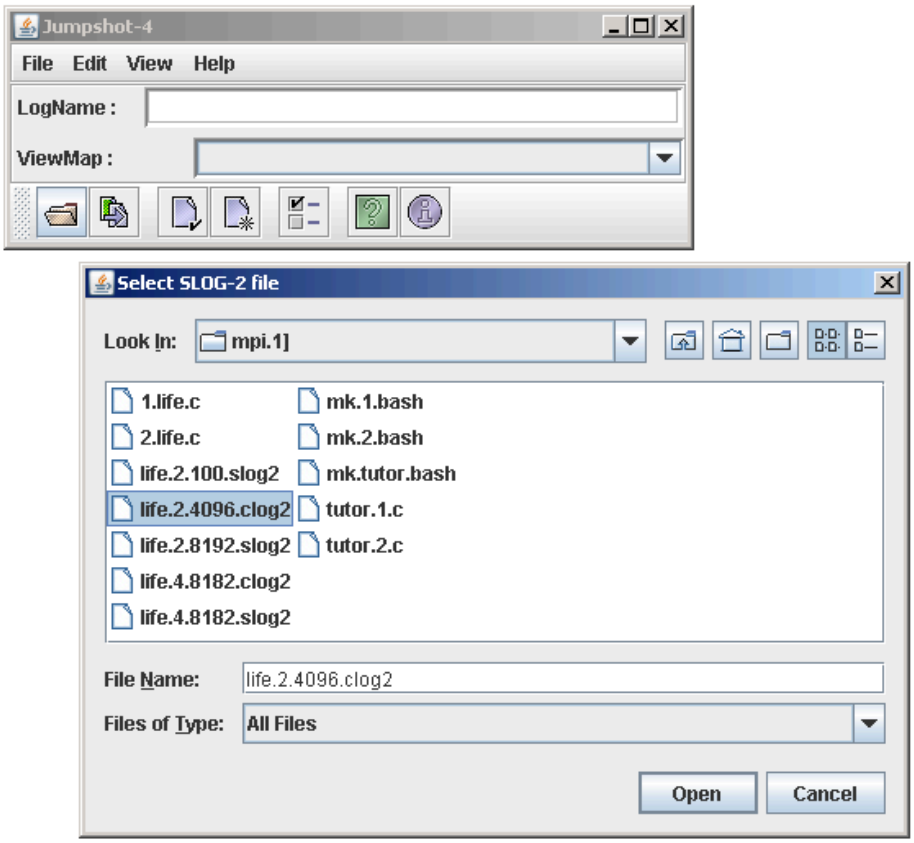

Рис. 1. Выбор трассировочного файла в JumpShot

Среда предложит сконвертировать его в формат *slog2* (см. рис. 2), необходимо выбрать *Yes*. Далее, в диалоге конвертации (см. рис. 3) следует выбрать *Convert*, а после ее успешного завершения - *OK*.

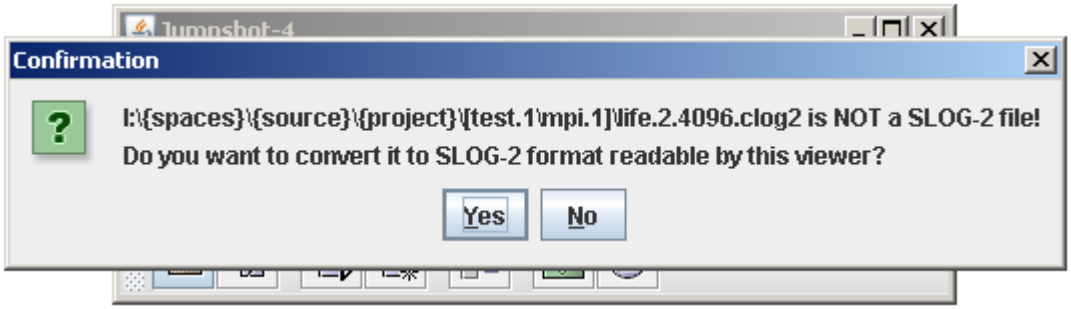

Рис. 2. Запрос на конвертацию трассировочного файла clog2 в slog2.

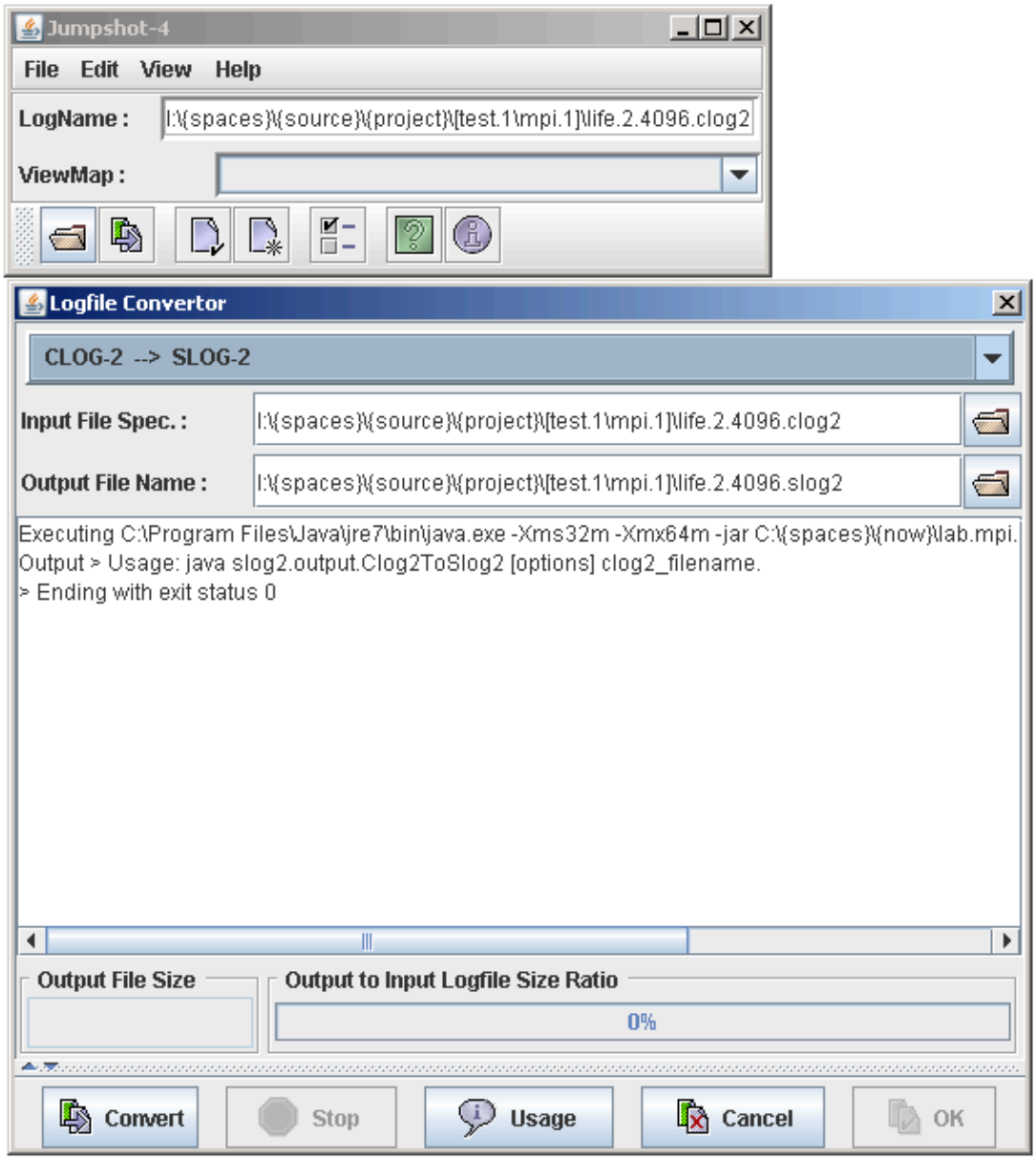

Рис. 3. Диалог конвертации трассировочного файла clog2 в slog2.

После конвертации открываются окна *Legend* с настройками отображения статистики для разных событий и состояний, например для разных функций MPI и *TimeLine*, в котором в графическом виде показаны доли времени, затрачиваемые на операции MPI каждым из MPI процессов в различные периоды исполнения.

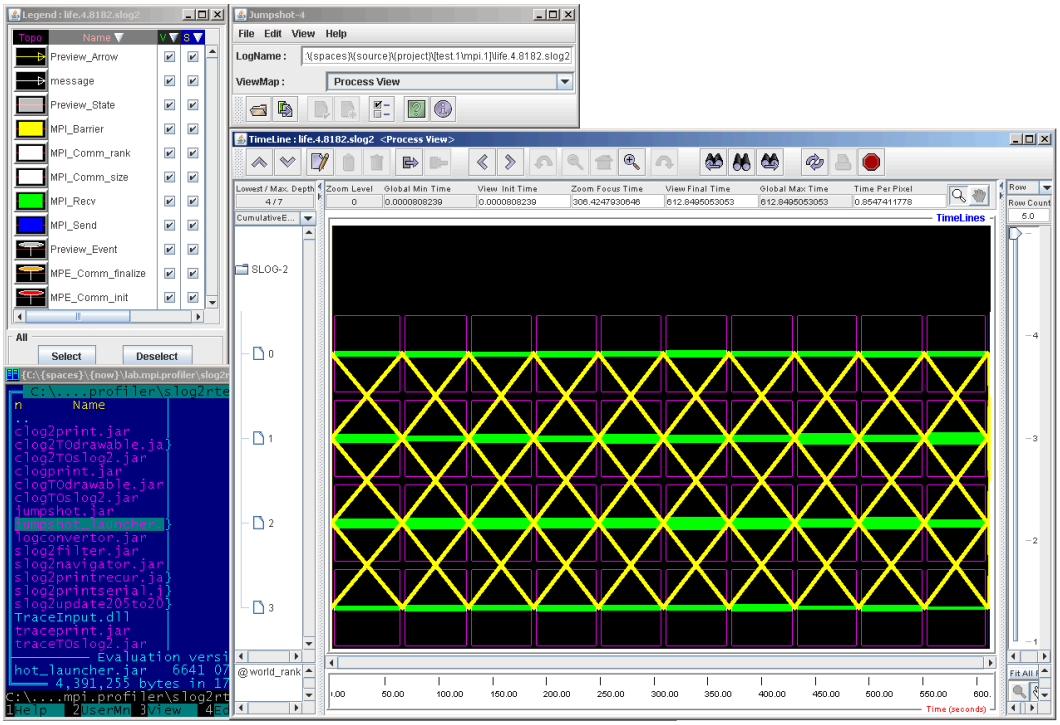

Рис. 4. Общий вид окон *Legend* и *Timeline*.

С помощью *Zoom* можно получить детальную статистику по времени исполнения с точностью до отдельных вызовов MPI функций (см. рис. 5).

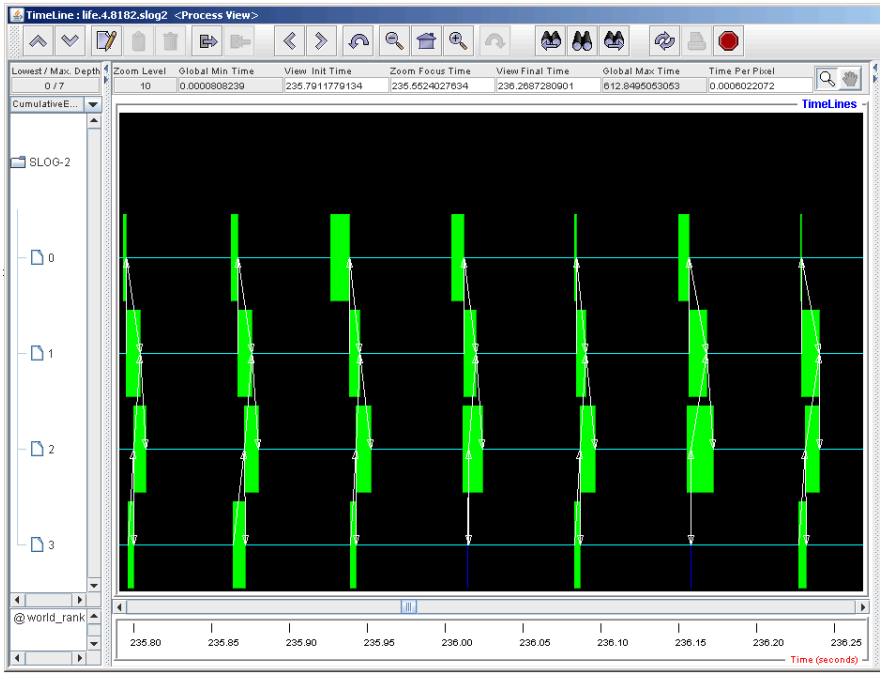

Рис. 5. Просмотр с большой детализацией для малых промежутков времени в окне *TimeLine*.

Для получения обобщенной информации о времени исполнения функций на интересующем временном интервале, в окне *TimeLine* с помощью контекстной

клавиши мыши выбирается соответствующий интервал, а в появившемся диалоговом окне *Duration Info Box* нажимается кнопка *Statistics*, после чего создается окно *Histogram for the duration* (см. рис. 6).

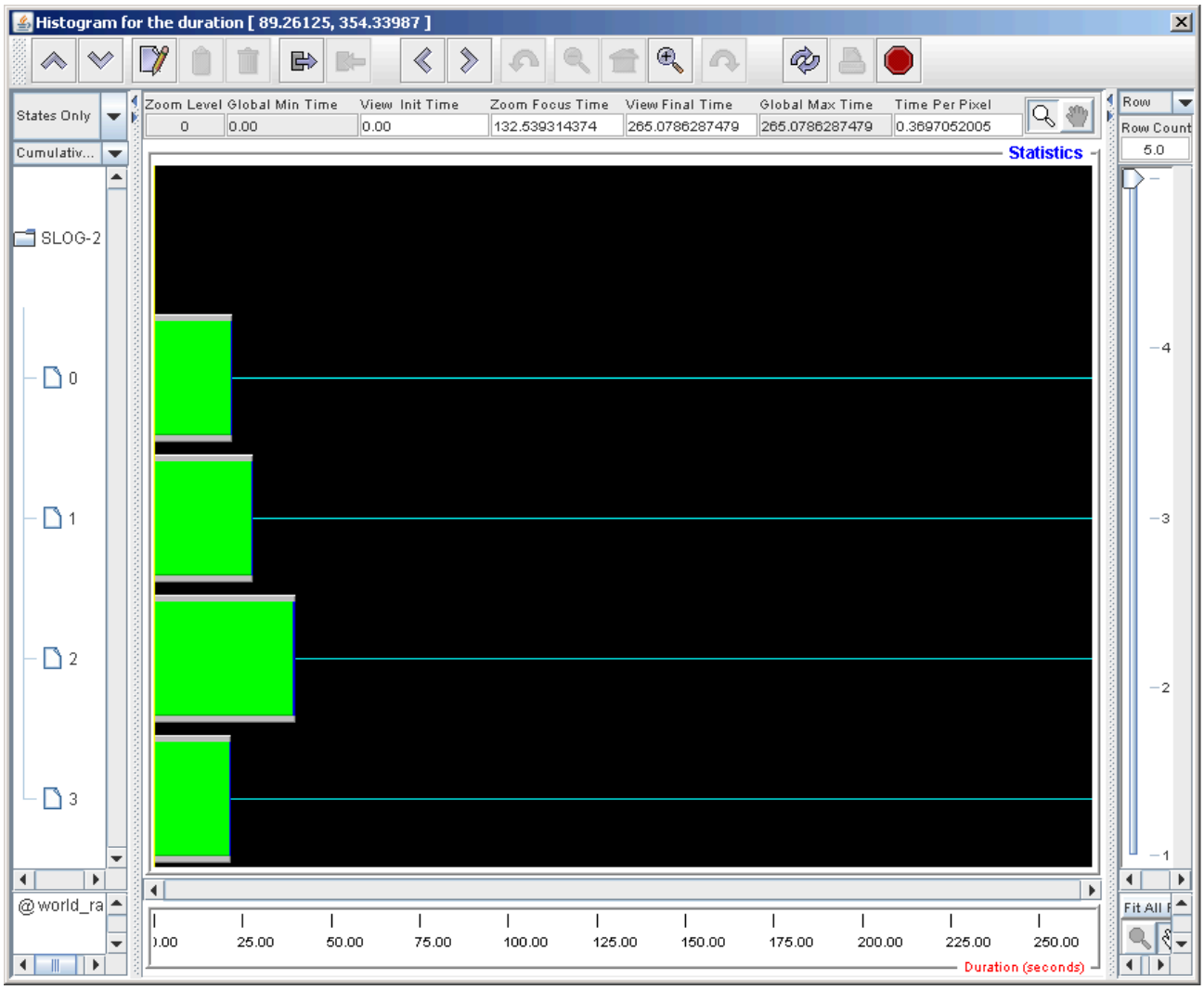

Рис. 6. Окно *Histogram for the duration*

Показ результатов настраиваемого профилирования происходит аналогично (при открытии соответствующего трассировочного файла). Пример визуализации программы листинга 1 приведен на рис. 7.

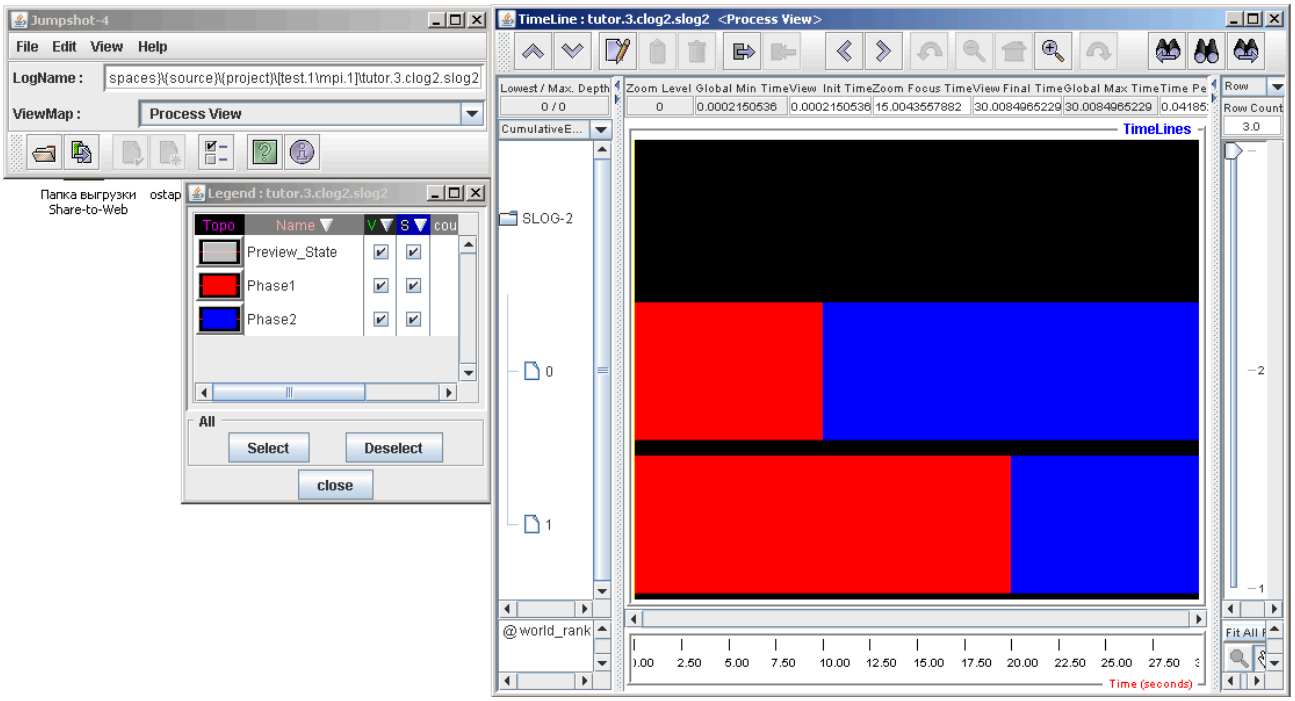

Рис. 7. Показ результатов настраиваемого профилирования

2. Задания

1) Установить MPE и Jumpshot.
2) Скомпилировать одну (из реализованных в предыдущих лабораторных работах программу, например, программу решения системы линейных уравнений методом МПИ или методом Гаусса) с использованием MPE. Запустить и собрать статистику в режиме автоматического профилирования для нескольких различных размеров матрицы с использованием двух и четырех процессов. По собранной статистике оценить долю времени работы счетной части и коммуникационной части в зависимости от размера массива. По данным измерений построить график.
3) Модифицировать текст выбранной программы для профилирования в настраиваемом режиме. Выбрать интересующие фазы счета, время которых планируется измерить. Например, если выбрана программа системы линейных уравнений методом Гаусса, то можно выделить фазы прямого хода и обратного хода. Скомпилировать модифицированную программу и собрать для нее статистику при выбранных размере матрицы и количестве MPI процессов.
4) (дополнительное задание) Установить пробную версию пакета Allinea MAP и произвести профилирование выбранной программы с его помощью.

3. Контрольные вопросы

1) Что такое профилирование? Для чего нужно профилирование?
2) Какие существуют методы реализации профайлеров? Какова специфика профилирования в MPI?
3) Какие есть средства профилирования MPI программ, какой набор пользовательских функций они реализуют?

4. Ссылки

[1] Проект MPE: http://www.mcs.anl.gov/research/projects/perfvis/
[2] Jumpshot: http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/
[3] MPICH2: http://www.mcs.anl.gov/research/projects/mpich2/
[4] Chan A., Gropp W., Lusk E. - User's Guide for MPE: Extensions for MPI Programs. - https://ftp.mcs.anl.gov/pub/mpi/mpeman.pdf