COMP Superscalar

COMPSs Tracing Manual

Version: 1.3

February 22, 2016

This manual only provides information about the COMPSs tracing system. Specifically, it provides how to run COMPSs applications with tracing (using the Extrae tool) and how to visualize, interpret and analyse the obtained traces (using the Paraver tool)

For further information about the application execution please refer to the *COMPSs User Manual: Application execution guide* available at `http://compss.bsc.es` .

For further information about the application development please refer to the *COMPSs User Manual: Application development guide* available at `http://compss.bsc.es/` .

Contents

List of Figures

List of Tables

1 COMP Superscalar (COMPSs)

COMP Superscalar (COMPSs) is a programming model which aims to ease the development of applications for distributed infrastructures, such as Clusters, Grids and Clouds. COMP Superscalar also features a runtime system that exploits the inherent parallelism of applications at execution time.

For the sake of programming productivity, the COMPSs model has four key characteristics:

- **Sequential programming:** COMPSs programmers do not need to deal with the typical duties of parallelization and distribution, such as thread creation and synchronization, data distribution, messaging or fault tolerance. Instead, the model is based on sequential programming, which makes it appealing to users that either lack parallel programming expertise or are looking for better programmability.

- **Infrastructure unaware:** COMPSs offers a model that abstracts the application from the underlying distributed infrastructure. Hence, COMPSs programs do not include any detail that could tie them to a particular platform, like deployment or resource management. This makes applications portable between infrastructures with diverse characteristics.

- **Standard programming languages:** COMPSs is based on the popular programming language Java, but also offers language bindings for Python and C/C++ applications. This facilitates the learning of the model, since programmers can reuse most of their previous knowledge.

- **No APIs:** In the case of COMPSs applications in Java, the model does not require to use any special API call, pragma or construct in the application; everything is pure standard Java syntax and libraries. With regard the Python and C/C++ bindings, a small set of API calls should be used on the COMPSs applications.

2 Trace Execution

COMPSs Runtime can generate a post-execution trace of the distributed execution of the application. This trace is useful for performance analysis and diagnosis.

A trace file may contain different events to determine the COMPSs master state, the task execution state or the file-transfers. Despite the fact that in the current release we do not support file-transfers, we intend to support them in a near future release.

During the execution of the application, an XML file is created at worker nodes to keep track of these events. At the end of the execution, all the XML files are merged to get a final trace file.

In the following sections we explain the command used for tracing, how the events are registered, in a process called instrumentation, how to visualize the trace file and make a good analysis of performance based on the data shown in the trace.

2.1 Trace Command

In order to obtain a post-execution trace file the option **-t** must be added to the runcompss command. Next we provide an example of the command execution with the tracing option enabled for the Hmmer java application.

```
compss@bsc:~$ runcompss -t --classpath=/home/compss/workspace_java/hmmerobj/jar/hmmerobj.
    jar
                  hmmerobj.HMMPfam
                  /sharedDisk/Hmmer/smart.HMMs.bin /sharedDisk/Hmmer/256seq
                  /home/compss/out.txt 2 8 -A 222
```

2.2 Application Instrumentation

The instrumentation is the process that intercepts different events of the application execution and keeps log of them. This will cause an overhead in the execution time of the application that the user should take into account, but the collected data will be extremely useful for performance analysis and diagnosis.

COMPSs Runtime uses the *Extrae* tool to dynamically instrument the application and the *Paraver* tool to visualize the obtained tracefiles. Both tools are developped at *BSC* and are available in its webpage `http://bsc.es` .

At the worker nodes, in background, *Extrae* keeps track of the events in an intermediate format file (with *.mpit* extension). Inside the master node, at the end of the execution, *Extrae* merges the intermediate files to get the final trace file, a *Paraver* format file (.prv). See the visualization section 3 in this manual for further information about the *Paraver* tool.

When instrumenting the application *Extrae* will output several messages. At the master node, *Extrae* will show up its initialization at the begining of the execution and the merging process and the paraver generation at the end of the execution. At the worker nodes *Extrae* will inform about the intermediate files generation every time a

task is executed. Next we provide a summary of the *stdout* generated by Hmmer java application execution with the trace flag enabled.

```
------------------ Executing hmmerobj.HMMPfam
     -------------------------

WARNING: IT Properties file is null. Setting default values
Welcome to Extrae 3.1.1rc (revision 3360 based on extrae/trunk)
Extrae: Warning! EXTRAE_HOME has not been defined!.
Extrae: Generating intermediate files for Paraver traces.
Extrae: Intermediate files will be stored in /home/compss/workspace_java/hmmerobj/jar
Extrae: Tracing buffer can hold 500000 events
Extrae: Tracing mode is set to: Detail.
Extrae: Successfully initiated with 1 tasks

Extrae: Warning! API tries to initialize more than once
Extrae:         Previous initialization was done by API

[  API]  -  Starting COMPSs Runtime v1.3 (build 20150821-1134.rnull)

...
...
...

[  API]  -  No more tasks for app 1
[  API]  -  Getting Result Files 1
[  API]  -  Execution Finished

Extrae: Intermediate raw trace file created : /home/compss/workspace_java/hmmerobj/jar/set-0/
     TRACE@bsc.0000031637000000000000.mpit
Extrae: Intermediate raw sym file created : /home/compss/workspace_java/hmmerobj/jar/set-0/
     TRACE@bsc.0000031637000000000000.sym
Extrae: Deallocating memory.
Extrae: Application has ended. Tracing has been terminated.

merger: Output trace format is: Paraver
merger: Extrae 3.1.1rc (revision 3360 based on extrae/trunk)

mpi2prv: Checking for target directory existance... exists, ok!
mpi2prv: Selected output trace format is Paraver
mpi2prv: Stored trace format is Paraver
mpi2prv: Parsing intermediate files
mpi2prv: Removing temporal files... done
mpi2prv: Congratulations! ./trace/hmmerobj.HMMPfam_compss_trace_1440151114.prv has been
     generated.

------------------------------------------------------------------------
```

For further information about *Extrae* please visit the following site:

`http://www.bsc.es/computer-science/extrae`

3 Visualization

Paraver is the *BSC* tool for trace visualization. Trace events are encoded in *Paraver* format (*.prv*) by the *Extrae* tool (see previous section). *Paraver* is a powerful tool that allows users to show many views of the trace data by means of different configuration files. Users can manually load, edit or create configuration files to obtain different trace data views.

In the following subsections we will explain how to load a trace file into *Paraver*, open the task events view by means of an already predefined configuration file, and how to adjust the view to properly display the data.

For further information about *Paraver* please visit the following site:

`http://www.bsc.es/computer-sciences/performance-tools/paraver`

3.1 Trace Loading

The final trace file in *Paraver* format (*.prv*) can be found at the *base log folder* of the application execution inside the trace folder. The fastest way to open it is calling directly the *Paraver* binary using the trace-file name as arugment.

```
compss@bsc:~$ wxparaver /home/compss/.COMPSs/hmmerobj.HMMPfam_01/trace/*.prv
```

3.2 Configuration File

In order to open a view with the task events of the application, an already predefined configuration file is provided. To open it, just go in the main window to the "Load Configuration" option in the menu "File". The configuration file is under the following path */opt/COMPSs/Dependencies/paraver/cfgs/tasks.cfg*. After accepting the load of the configuration file, another window will appear to show the view. Figures 1 and 2 show an example of this process.

3.3 View Adjustment

In a *Paraver* view, a red exclamation sign may appear on the bottom-left corner (see last Figure 2 in previous section). This means that some little adjustments must be done to view the trace correctly:

- Fit window: this will give a better color scale to identify events.
  - Right click on the trace window
  - Chose the option Fit Semantic Scale / Fit Both
- View Event Flags: This will put a flag whenever an event starts/ends.
  - Right click on the trace window

Figure 1: Paraver menu

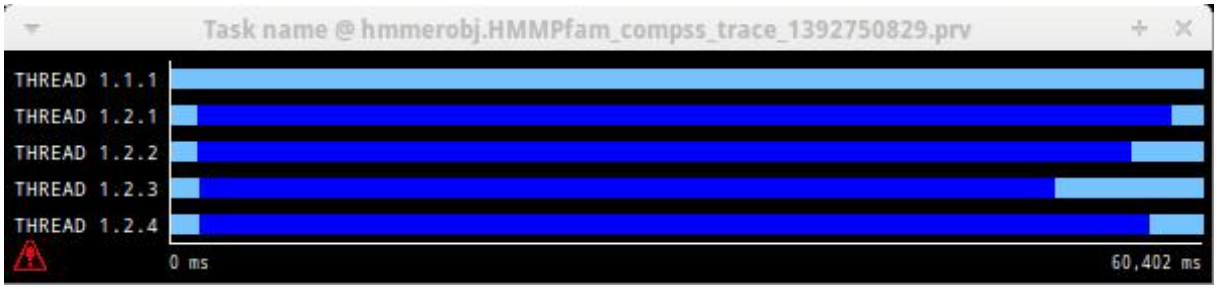


Figure 2: Trace file

  - Chose the option View / Event Flags

- Show Info Panel: This will show an information panel. In the tab “Colors” we can see the legend of the colors shown in the view.

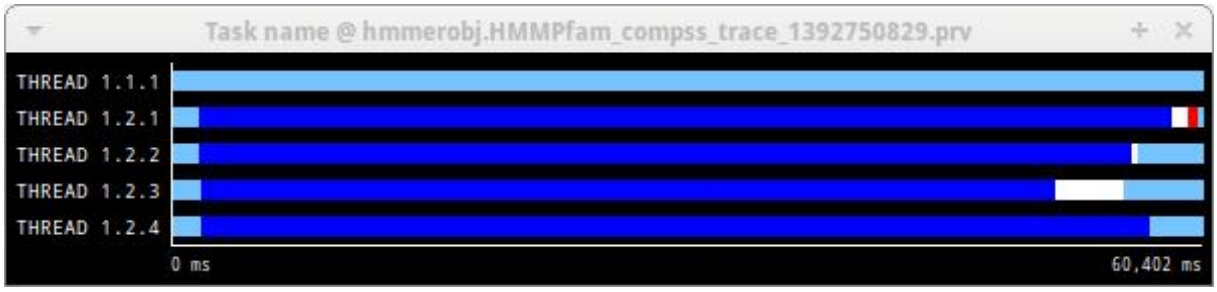


Figure 3: Paraver view adjustment: Fit window

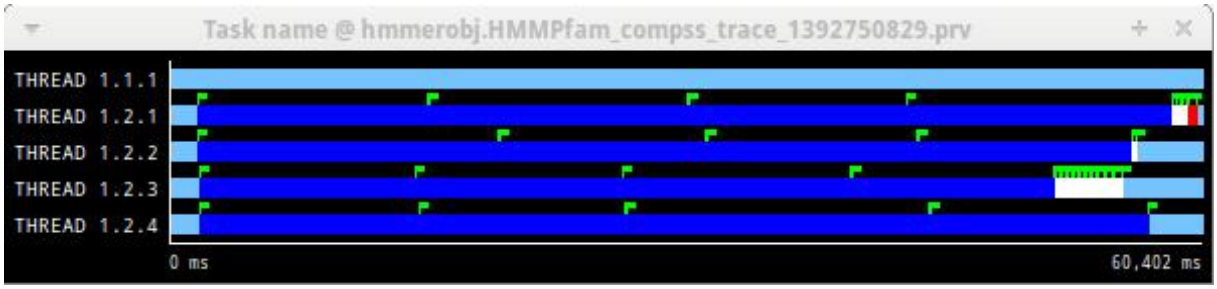


Figure 4: Paraver view adjustment: View Event Flags

– Right click on the trace window
– Check the Info Panel option
– Select the Colors tab in the panel

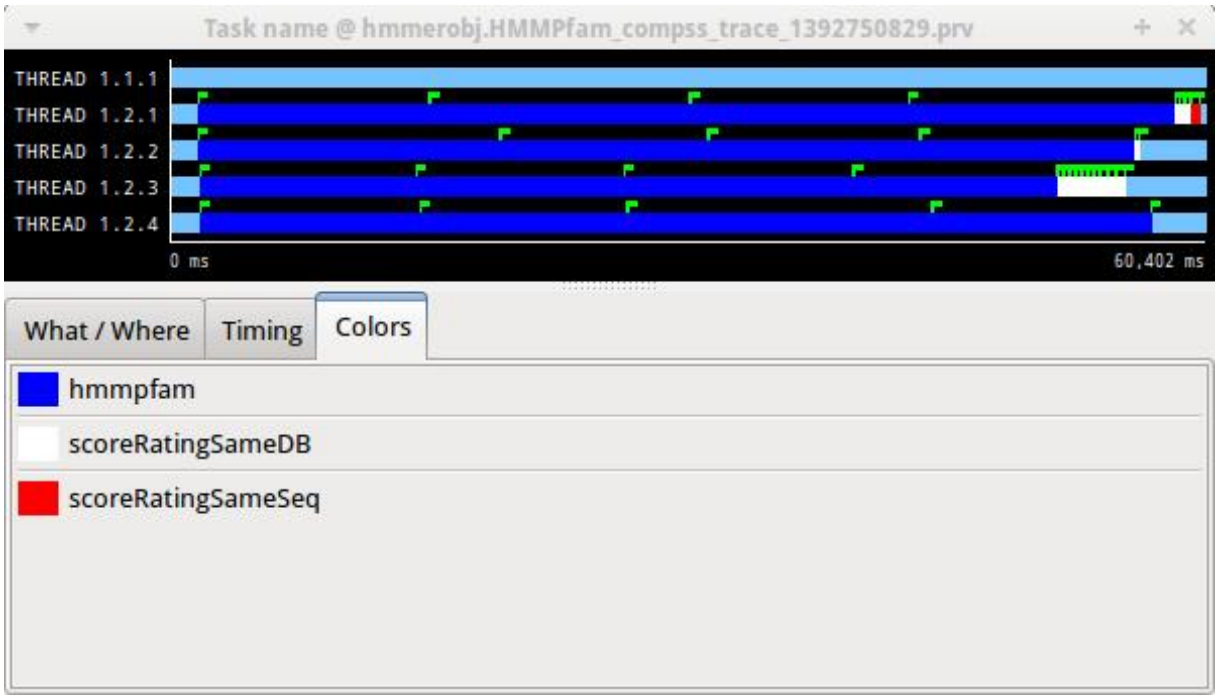


Figure 5: Paraver view adjustment: Show info panel

- Zoom: In order to understand a trace view better, sometimes it's a worth thing to zoom into it a little.
  - Select a region in the trace window to see that region in detail
  - And repeat the previous step as many times as needed
  - The undo-zoom option is in the right click panel

Figure 6: Paraver view adjustment: Zoom configuration

Figure 7: Paraver view adjustment: Zoom configuration

4 Interpretation

In this section we will explain how to interpret a trace view once it has been adjusted as described in the previous section.

- The trace view has in its horizontal axis the execution time and in the vertical axis one line for the master at the top, and below it, one line for each of the workers.
- In a line, the light blue color means idle state, in the sense that there is no event at that time.
- Whenever an event starts or ends a flag is shown.
- In the middle of an event, the line shows a different color. Colors are assigned depending on the event type.
- In the info panel the legend of assigned color to event type is provided.

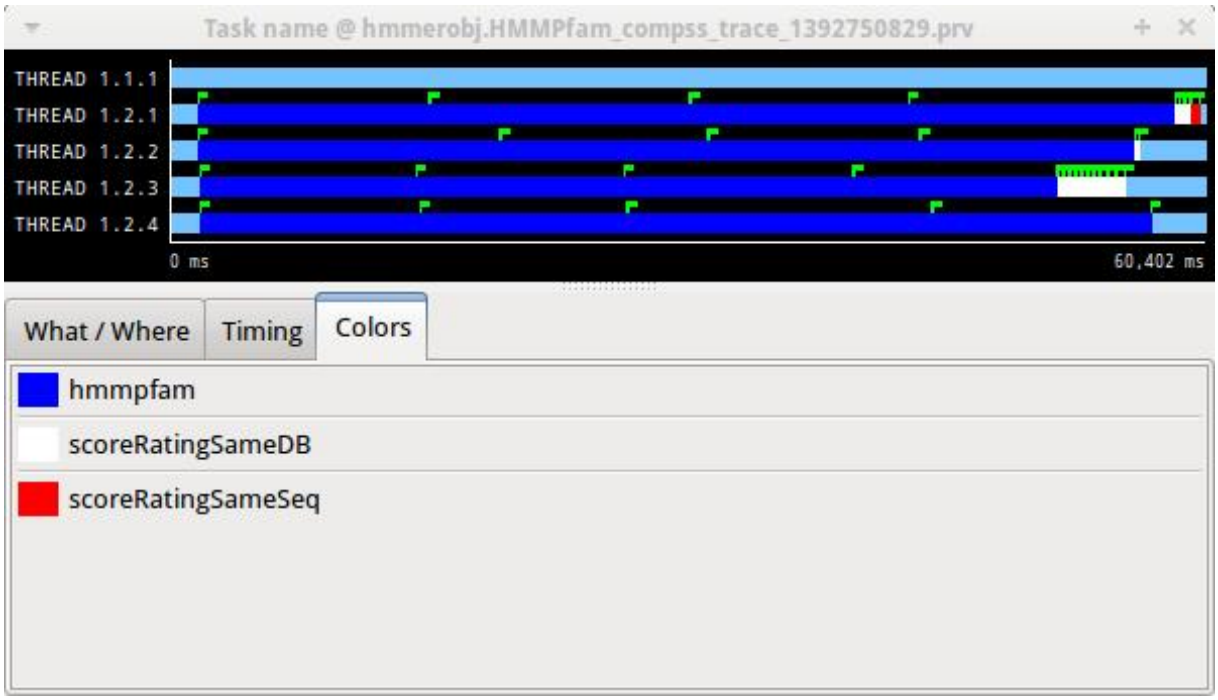


Figure 8: Trace interpretation

5 Analysis

In this section, we will give some tips to analyse a COMPSs trace from two different points of view: graphically and numerically.

5.1 Graphical Analysis

The main concept is that computational events, the task events in this case, must be well distributed among all workers to have a good parallelism, and the duration of task events should be also balanced, this means, the duration of computational bursts.

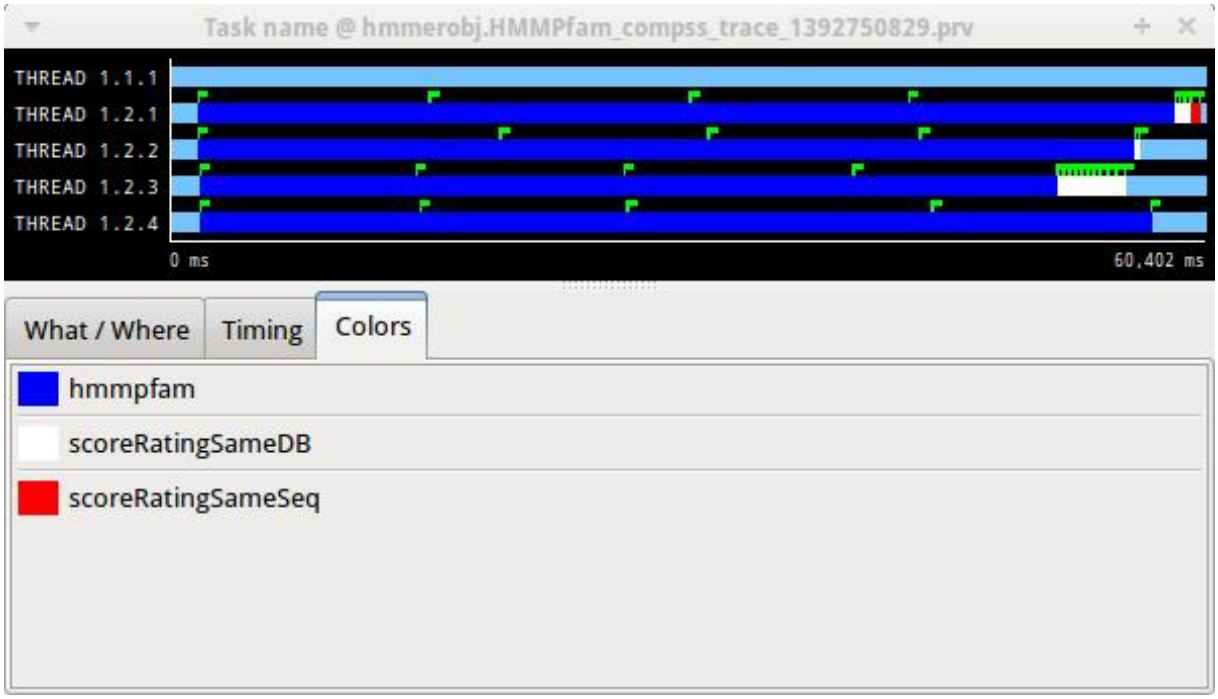


Figure 9: Caption.

In the previous trace view, all the tasks of type "hmmpfam" in dark blue appear to be well distributed among the four workers, each worker executes four "hmmpfam" tasks.

But some workers finish earlier than the others, worker 1.2.3 finish the first and worker 1.2.1 the last. So there is an imbalance in the duration of "hmmpfam" tasks. The programmer should analyse then whether all the tasks process the same amount of input data and do the same thing in order to find out the reason of such imbalance.

Another thing to highlight is that tasks of type "scoreRatingSameDB" are not equal distributed among all the workers. There are workers that execute more tasks of this type than the others. To understand better what happens here, let's take a look to the execution graph and also zoom in the last part of the trace.

There is only one task of type "scoreRatingSameSeq". This task appears in red in the trace (and in light-green in the graph). With the help of the graph we see that the "scoreRatingSameSeq" task has dependences on tasks of type "scoreRatingSameDB", in white (or yellow).

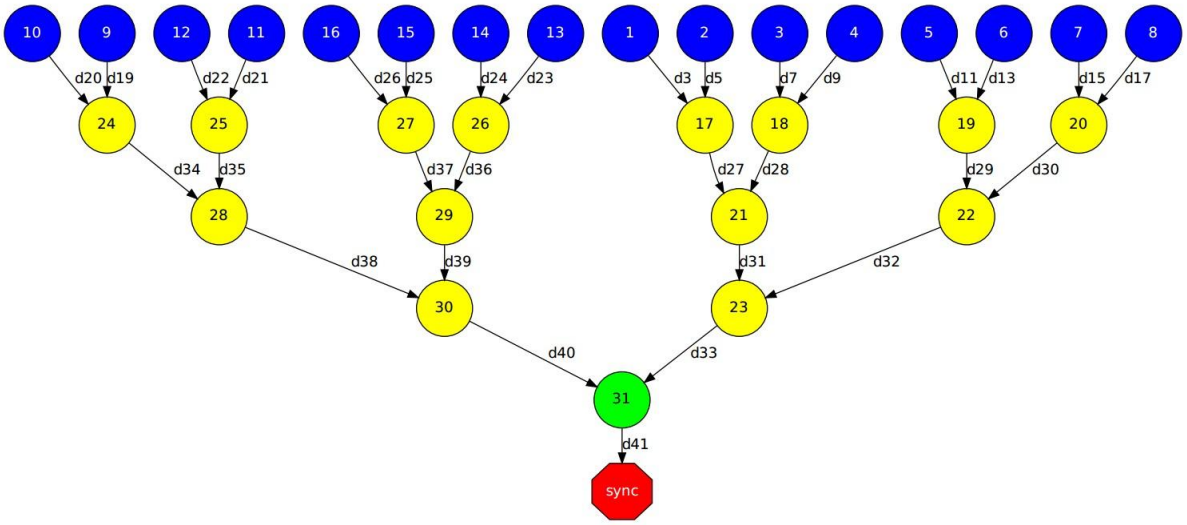


Figure 10: Caption.

Figure 11: Caption.

When the last task of type "hmmpfam" (in dark blue) ends, the last dependences are solved, and if we look at the graph, this means going across a path of three dependences of type "scoreRatingSameDB" (in yellow). And because of these are sequential dependences (one depends on the previous) no more than a worker can be used at the same time to execute the tasks. This is the reason of why the last three task of type "scoreRatingSameDB" (in white) are executed in worker 1.2.1 sequentially.

5.2 Numerical Analysis

Here we show another trace from a different parallel execution of the Hmmer program.

Paraver offers the possibility of having different histograms of the trace events. For it just click the "New Histogram" button in the main window and accept the default options in the "New Histogram" window that will appear.

After that, the following table is shown. In this case for each worker, the time spent executing each type of task is shown. Task names appear in the same color than in the trace view. The color of a cell in a row corresponding to a worker goes in a scale from a light-green for lower values to a dark-blue for higher ones. This conforms a color based histogram.

The previous table also gives, at the end of each column, some extra statistical in-

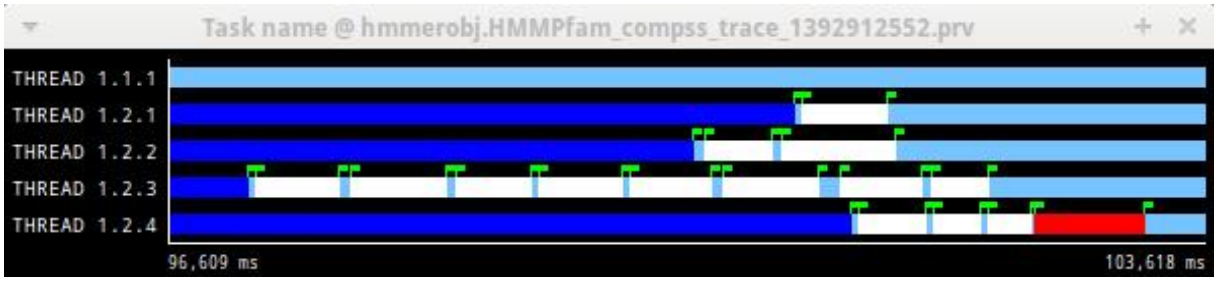
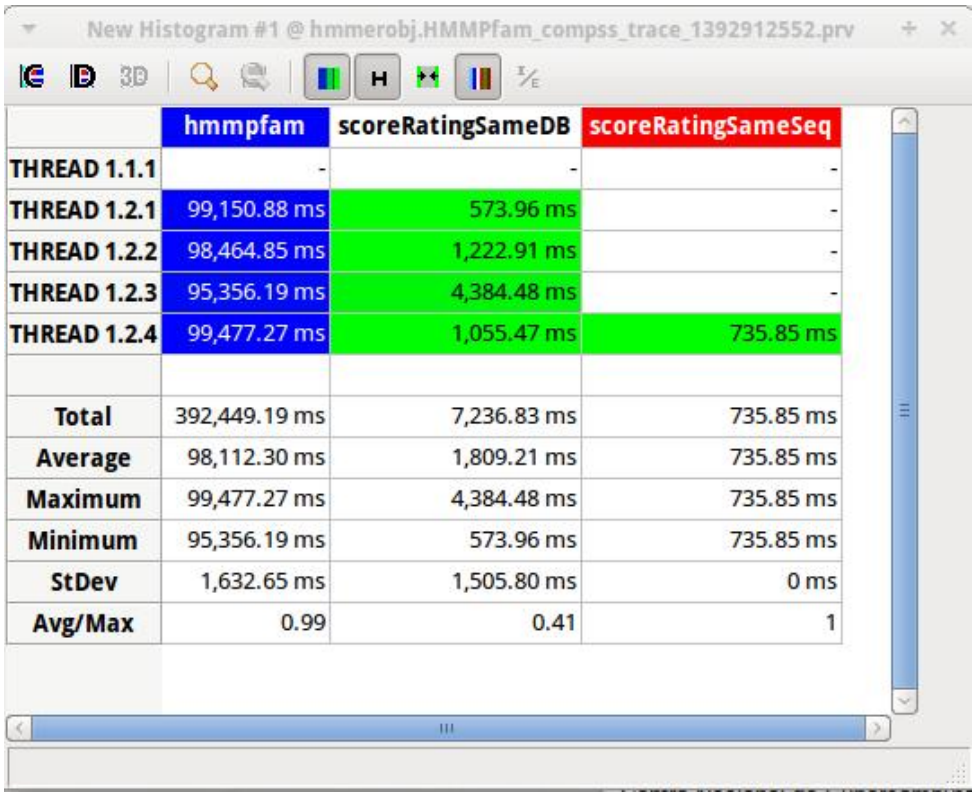


Figure 12: Caption.

Figure 13: Paraver Menu - New Histogram



	hmmpfam	scoreRatingSameDB	scoreRatingSameSeq
THREAD 1.1.1	-	-	-
THREAD 1.2.1	99,150.88 ms	573.96 ms	-
THREAD 1.2.2	98,464.85 ms	1,222.91 ms	-
THREAD 1.2.3	95,356.19 ms	4,384.48 ms	-
THREAD 1.2.4	99,477.27 ms	1,055.47 ms	735.85 ms
Total	392,449.19 ms	7,236.83 ms	735.85 ms
Average	98,112.30 ms	1,809.21 ms	735.85 ms
Maximum	99,477.27 ms	4,384.48 ms	735.85 ms
Minimum	95,356.19 ms	573.96 ms	735.85 ms
StDev	1,632.65 ms	1,505.80 ms	0 ms
Avg/Max	0.99	0.41	1

Figure 14: Hmmpfam histogram

formation for each type of tasks (as the total, average, maximum or minimum values, etc.).

In the window properties of the main window we can change the semantic of the statistics to see other factors rather than the time, for example, the number of bursts.

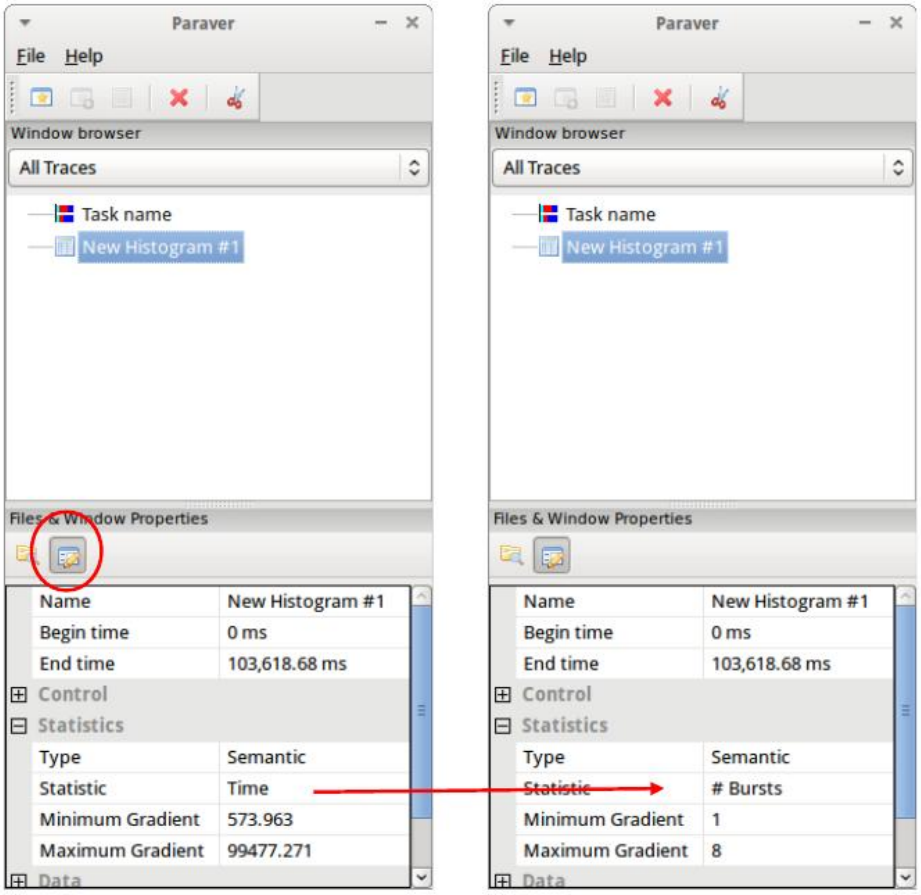
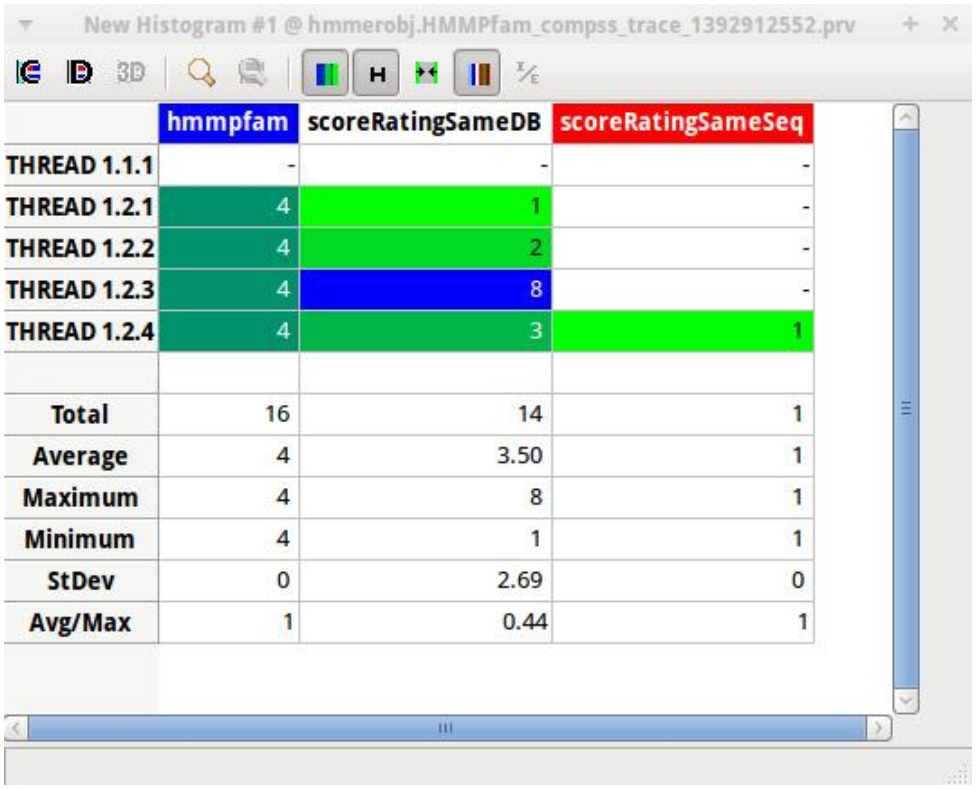


Figure 15: Paraver histogram options menu

In the same way as before, the following table shows for each worker the number of bursts for each type of task, this is, the number or tasks executed of each type. Notice the gradient scale from light-green to dark-blue changes with the new values.



	hmmpfam	scoreRatingSameDB	scoreRatingSameSeq
THREAD 1.1.1	-	-	-
THREAD 1.2.1	4	1	-
THREAD 1.2.2	4	2	-
THREAD 1.2.3	4	8	-
THREAD 1.2.4	4	3	1
Total	16	14	1
Average	4	3.50	1
Maximum	4	8	1
Minimum	4	1	1
StDev	0	2.69	0
Avg/Max	1	0.44	1

Figure 16: Hmmpfam histogram with the number of bursts

6 Other trace examples

To end this section, let's present some other examples of COMPSs traces. COMPSs traces can be much complex as the number of workers or tasks grows. Just to illustrate this, the following pictures show traces with a greater number of workers and tasks.

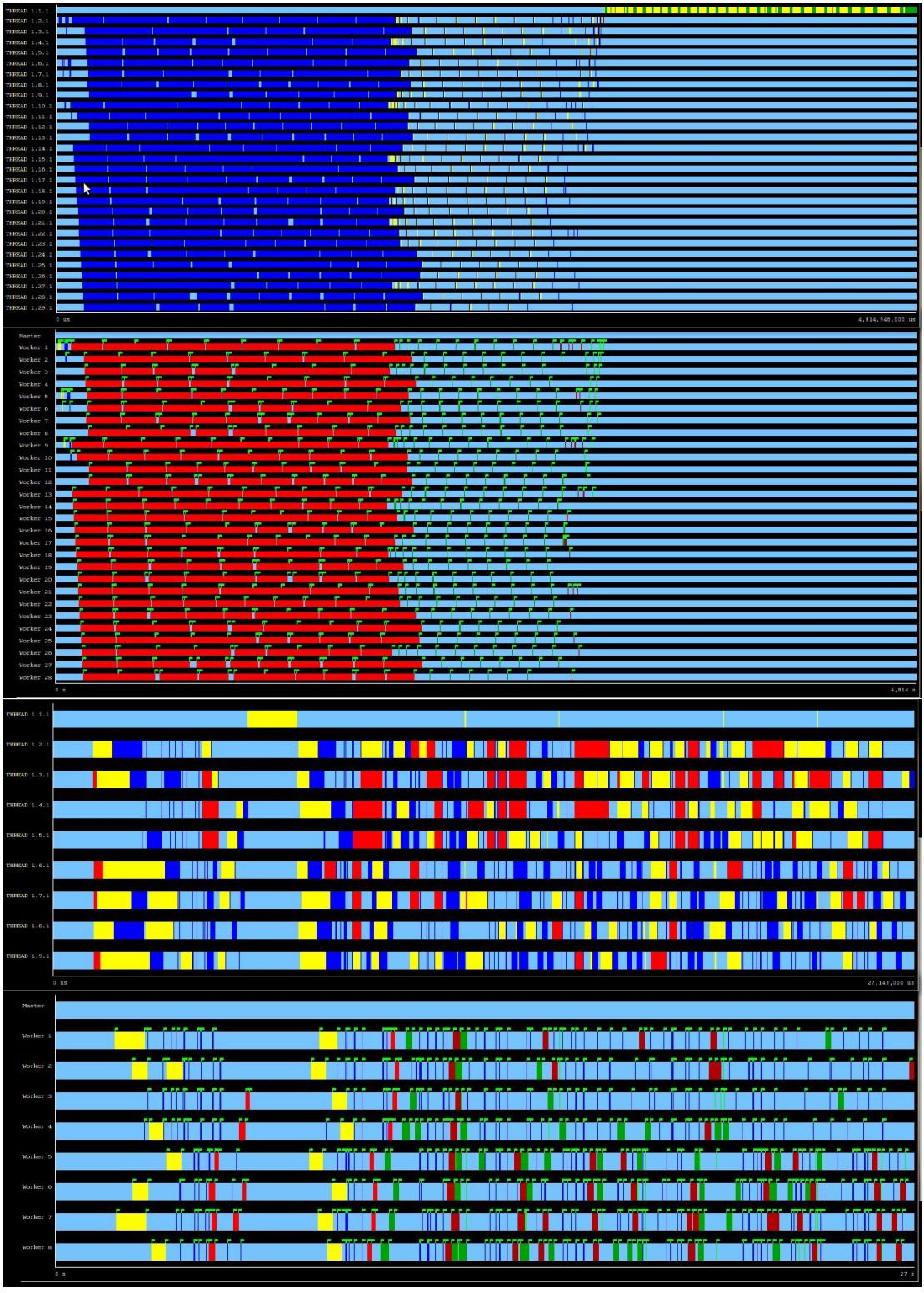


Figure 17: Examples of complex traces

Please find more details on the COMPSs framework at

http://compss.bsc.es