



COMP SUPERSCALAR

COMPSS Tracing Manual

VERSION: 1.4

April 15, 2016



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

This manual only provides information about the COMPSs tracing system. Specifically, it provides how to run COMPSs applications with tracing (using the Extrae tool¹) and how to visualize, interpret and analyse the obtained traces (using the Paraver tool²).

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/> .

¹For more information: <https://www.bsc.es/computer-sciences/extrae>

²For more information: <https://www.bsc.es/computer-sciences/performance-tools/paraver>]

Contents

1	COMP Superscalar (COMPSs)	1
2	Tracing	2
2.1	Basic Mode	2
2.2	Advanced Mode	8
3	Visualization	12
3.1	Trace Loading	12
3.2	Configurations	12
3.3	View Adjustment	12
4	Interpretation	16
5	Analysis	17
5.1	Graphical Analysis	17
5.2	Numerical Analysis	18
	Appendices	22
A	PAPI: Hardware Counters	22
B	Paraver: configurations	23

List of Figures

1	Basic mode tracefile for a k-means algorithm visualized with compss_runtime.cfg	7
2	Advanced mode tracefile for a testing program showing the total completed instructions	10
3	Paraver menu	13
4	Trace file	13
5	Paraver view adjustment: Fit window	14
6	Paraver view adjustment: View Event Flags	14
7	Paraver view adjustment: Show info panel	15
8	Paraver view adjustment: Zoom configuration	15
9	Paraver view adjustment: Zoom configuration	15
10	Trace interpretation	16
11	Caption.	17
12	Caption.	18
13	Caption.	18
14	Caption.	19
15	Paraver Menu - New Histogram	19
16	Hmmpfam histogram	19
17	Paraver histogram options menu	20
18	Hmmpfam histogram with the number of bursts	21

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 Tracing

COMPSs Runtime has a built-in instrumentation system to generate post-execution tracefiles of the applications' execution. The tracefiles contain different events representing the COMPSs master state, the tasks' execution state, and the data transfers (transfers' information is only available when using NIO adaptor), and are useful for both visual and numerical performance analysis and diagnosis. The instrumentation process essentially intercepts and logs different events so it will add an overhead to the execution time of the application.

Traces of the application are generated with *Extrac* and the resulting tracefiles are visualized with *Paraver*. Both tools are developed and maintained by the Performance Tools team of the *BSC* and are available on its webpage <http://www.bsc.es/computer-sciences/performance-tools>.

For each worker node and the master, *Extrac* keeps track of the events in an intermediate format file (with *.mpit* extension). At the end of the execution, all intermediate files are gathered and merged with *Extrac's mpi2prv* command in order to create the final tracefile, a *Paraver* format file (*.prv*). See the visualization section 3 in this manual for further information about the *Paraver* tool.

When instrumenting the application *Extrac* will output several messages corresponding to the tracing initialization, intermediate files' creation, and the merging process.

At present time, COMPSs tracing features two execution modes:

Basic, aimed to COMPSs applications developers

Advanced, aimed to COMPSs developers and users with access to its source code or custom installations

Next sections describe the information provided by each mode and how to use them.

2.1 Basic Mode

This mode is aimed to COMPSs' apps users and developers. Instruments computing threads and some management resources providing information about tasks' executions, data transfers and hardware counters if PAPI is available (see PAPI counters appendix A for more info).

2.1.1 Usage

In order to activate basic tracing one needs to provide one of the following arguments to the execution command:

- `-t`
- `-tracing`
- `-tracing=basic`
- `-tracing=true`

Examples given:

```
runcompss --tracing application_name application_args
```

Figure 1 was generated as follows:

```
runcompss \  
  --lang=java \  
  --tracing \  
  --classpath=/path/to/jar/kmeans.jar \  
  kmeans.KMeans
```

When tracing is activated, additional extrae output is generated helping the user ensure that instrumentation is indeed turned on and that no problems were found. On basic mode this is the output users should see when tracing is working correctly:

```
*** RUNNING JAVA APPLICATION KMEANS  
Resolved: /path/to/jar/kmeans.jar:  
  
----- Executing kmeans.Kmeans  
-----  
  
Extrae: WARNING!  
Extrae: WARNING! XML parser version and property 'xml-parser-id' do not match. Check the XML  
       file. Trying to proceed...  
Extrae: WARNING!  
Extrae: xml-parser-id found 'Id: xml-parse.c 3682 2015-11-26 14:32:27Z harald $' when expecting '  
       Id: xml-parse.c 3918 2016-03-11 14:59:01Z harald $'.  
Welcome to Extrae 3.3.0 (revision 3966 based on extrae/trunk)  
Extrae: Parsing the configuration file (/opt/COMPSS/Runtime/scripts/user/../../configuration/xml/  
       tracing/extrae.basic.xml) begins  
Extrae: Tracing package is located on /opt/COMPSS/Dependencies/extrae/  
Extrae: Generating intermediate files for Paraver traces.  
Extrae: Warning! change-at-time time units not specified. Using seconds  
Extrae: PAPI domain set to ALL for HWC set 1  
Extrae: HWC set 1 contains following counters < PAPI_TOT_INS (0x80000032) PAPI_TOT_CYC (0  
       x8000003b) PAPI_L2_DCM (0x80000002) PAPI_L3_TCM (0x80000008) > - never changes  
Extrae: Warning! change-at-time time units not specified. Using seconds  
WARNING: IT Properties file is null. Setting default values  
[ API] - Deploying COMPSS Runtime v1.4 (build 20160412-1147.r2040)  
[ API] - Tracing is activated  
[ API] - Starting COMPSS Runtime v1.4 (build 20160412-1147.r2040)  
...  
...  
...  
merger: Output trace format is: Paraver  
merger: Extrae 3.3.0 (revision 3966 based on extrae/trunk)  
mpi2prv: Assigned nodes < Marginis >  
mpi2prv: Assigned size per processor < <1 Mbyte >  
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000000.mpit is object 1.1.1 on node Marginis  
       assigned to processor 0
```



```

mpi2prv: File set-0/TRACE@Marginis.0000001904000000000001.mpit is object 1.1.2 on node Marginis
assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000002.mpit is object 1.1.3 on node Marginis
assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.00000019800000001000000.mpit is object 1.2.1 on node Marginis
assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.00000019800000001000001.mpit is object 1.2.2 on node Marginis
assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.00000019800000001000002.mpit is object 1.2.3 on node Marginis
assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.00000019800000001000003.mpit is object 1.2.4 on node Marginis
assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.00000019800000001000004.mpit is object 1.2.5 on node Marginis
assigned to processor 0
mpi2prv: Time synchronization has been turned off
mpi2prv: A total of 9 symbols were imported from TRACE.sym file
mpi2prv: 0 function symbols imported
mpi2prv: 9 HWC counter descriptions imported
mpi2prv: Checking for target directory existence... exists, ok!
mpi2prv: Selected output trace format is Paraver
mpi2prv: Stored trace format is Paraver
mpi2prv: Searching synchronization points... done
mpi2prv: Time Synchronization disabled.
mpi2prv: Circular buffer enabled at tracing time? NO
mpi2prv: Parsing intermediate files
mpi2prv: Progress 1 of 2 ... 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75%
80% 85% 90% 95% done
mpi2prv: Processor 0 succeeded to translate its assigned files
mpi2prv: Elapsed time translating files: 0 hours 0 minutes 0 seconds
mpi2prv: Elapsed time sorting addresses: 0 hours 0 minutes 0 seconds
mpi2prv: Generating tracefile (intermediate buffers of 838848 events)
This process can take a while. Please, be patient.
mpi2prv: Progress 2 of 2 ... 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75%
80% 85% 90% 95% done
mpi2prv: Warning! Clock accuracy seems to be in microseconds instead of nanoseconds.
mpi2prv: Elapsed time merge step: 0 hours 0 minutes 0 seconds
mpi2prv: Resulting tracefile occupies 991743 bytes
mpi2prv: Removing temporal files... done
mpi2prv: Elapsed time removing temporal files: 0 hours 0 minutes 0 seconds
mpi2prv: Congratulations! ./trace/kmeans.Kmeans_compss_trace_1460456106.prv has been generated.
[ API] - Execution Finished
Extrae: Tracing buffer can hold 100000 events
Extrae: Circular buffer disabled.
Extrae: Warning! <dynamic-memory> tag will be ignored. This library does support instrumenting
dynamic memory calls.
Extrae: Warning! <input-output> tag will be ignored. This library does support instrumenting I/O
calls.
Extrae: Dynamic memory instrumentation is disabled.
Extrae: Basic I/O memory instrumentation is disabled.
Extrae: Parsing the configuration file (/opt/COMPSS/Runtime/scripts/user/../../configuration/xml/
tracing/extrae.basic.xml) has ended
Extrae: Intermediate traces will be stored in /home/kurtz/compss/tests_local/app10
Extrae: Tracing mode is set to: Detail.
Extrae: Successfully initiated with 1 tasks and 1 threads

```

It contains diverse informations about the tracing, for example Extrae version used (3.3.0), the XML configuration file used (extrae_basic.xml), the amount of threads instrumented (objects through 1.1.1 to 1.2.5), available hardware counters (PAPI_TOT_INS (0x80000032) ... PAPI_L3_TCM (0x80000008)) or the name of the generated tracefile (./trace/kmeans.Kmeans_compss_trace_1460456106.prv). When using NIO communications adaptor with debug activated, the log of each worker will also contain the Extrae initialization information.

2.1.2 Instrumented Threads

Basic traces instrument the following threads:

- Master node (3 threads)
 - COMPSs runtime
 - Task Dispatcher
 - Access Processor
- Worker node (1 + Computing Units)
 - Main thread
 - Number of threads available for computing

2.1.3 Information Available

The basic mode tracefiles contain three kinds of information:

Events, marking diverse situations such as the runtime start, tasks' execution or synchronization points.

Communications, showing the transfers and requests of the parameters needed by COMPSs tasks.

Hardware counters, of the execution obtained with Performance API (see PAPI counters appendix A)

2.1.4 Trace Example

Figure 1 shows an the tracefile generated by the execution of a k-means clustering algorithm. Each timeline contains information of a different resource and each event is named on the legend. Depending on the number of computing threads specified for each worker, the number of timelines will vary. However the following threads are always shown:

Master - Thread 1.1.1,

this timeline shows the actions performed by the main thread of the COMPSs application

Task Dispatcher - Thread 1.1.2,

shows information about the state and scheduling of the tasks to be executed.

Access Processor - Thread 1.1.3,

all the events related with the tasks' parameters management, such as dependencies or transfers are shown on this thread.

Worker X Master - Thread 1.X.1,

repeated for each available resource, this thread is the master of each worker, handling the computing resources and transfers. All data events of the worker, such as requests, transfers and receivals are marked on this timeline (when using the appropriate configurations).

Worker X Computing Unit Y - Thread 1.X.Y

repeated as many times as computing threads has the worker X. Shows the actual tasks execution information.



Figure 1: Basic mode tracefile for a k-means algorithm visualized with compss_runtime.cfg

2.2 Advanced Mode

This mode is aimed at more advanced COMPSs' users such as developers who want to customize further the information provided by the tracing or need rawer information like pthreads calls or Java garbage collection.

N.B.: The extra information provided by the advanced mode is only available on the workers when using NIO adaptor.

2.2.1 Usage

In order to activate the advanced tracing add the following option to the execution:

- `-tracing=advanced`

Examples given:

```
runcompss --tracing=advanced application_name application_args
```

Figure 2 was generated as follows:

```
runcompss \  
  --lang=java \  
  --tracing=advanced \  
  --classpath=/path/to/jar/kmeans.jar \  
  kmeans.KMeans
```

When advanced tracing is activated the configuration file reported on the output will be `extrae_advanced.xml`.

```
*** RUNNING JAVA APPLICATION KMEANS  
...  
...  
...  
Welcome to Extrae 3.3.0 (revision 3966 based on extrae/trunk)  
Extrae: Parsing the configuration file (/opt/COMPSs/Runtime/scripts/user/../../configuration/xml/  
tracing/extrae_advanced.xml) begins
```

This is the default file used for advanced tracing. However, advanced users can modify it in order to customize the information provided by Extrae. The configuration file is read first by the master on the *runcompss* script. When using NIO adaptor for communication the configuration file is also read when each worker is started (on *persistent_worker.sh* or *persistent_worker_starter.sh* depending on the execution environment).

If the default file is modified the changes will affect the master, and also the workers when using NIO. Modifying the scripts which turn on the master and the workers is possible to achieve different instrumentations for master/workers. However, not all

Extrae available XML configurations will work with COMPSs, some of them will make the runtime and/or workers crash so modify them at your own discretion and risk. More information about instrumentation XML configurations on Extrae User guide at: <https://www.bsc.es/computer-sciences/performance-tools/trace-generation/extrae/extrae-user-guide>.

2.2.2 Instrumented Threads

Advanced mode instruments all the pthreads created during the application execution. It contains all the threads shown on basic traces plus extra ones used to call command-line commands, I/O streams managers and all actions which create a new process. Due to the temporal nature of many of this threads they may contain little information or appear just at specific parts of the execution pipeline.

2.2.3 Information Available

The advanced mode tracefiles contain the same information as the basic ones:

Events, marking diverse situations such as the runtime start, tasks' execution or synchronization points.

Communications, showing the transfers and requests of the parameters needed by COMPSs tasks.

Hardware counters, of the execution obtained with Performance API (see PAPI counters appendix A)

2.2.4 Trace Example

Figure 2 shows the total completed instructions for a sample program executed with the advanced tracing mode. Note that the thread - resource correspondence described on the basic trace example is no longer static and thus cannot be inferred. Nonetheless, they can be found thanks to the named events shown in other configurations such as `compss_runtime.cfg`.



Figure 2: Advanced mode tracefile for a testing program showing the total completed instructions

For further information about *Extræ* 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 *Extrac* 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 argument.

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

3.2 Configurations

To see the different events, counters and communications that the runtime generates, diverse configurations are available with the COMPSs installation. To open one of them, go to the “Load Configuration” option in the main window and select “File”. The configuration files are under the following path for the default installation */opt/COMPSs/Dependencies/paraver/cfgs/*. A detailed list of all the available configurations can be found in Appendix B.

The following guide uses the *compss_tasks.cfg* as an example to illustrate the basic usage of *Paraver*. After accepting the load of the configuration file, another window will appear to show the view. Figures 3 and 4 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 4 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



Figure 3: Paraver menu



Figure 4: Trace file

- View Event Flags: This will put a flag whenever an event starts/ends.
 - Right click on the trace window
 - Chose the option View / Event Flags



Figure 5: Paraver view adjustment: Fit window



Figure 6: Paraver view adjustment: 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.
 - Right click on the trace window
 - Check the Info Panel option
 - Select the Colors tab in the 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 7: Paraver view adjustment: Show info panel



Figure 8: Paraver view adjustment: Zoom configuration



Figure 9: 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 10: 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.

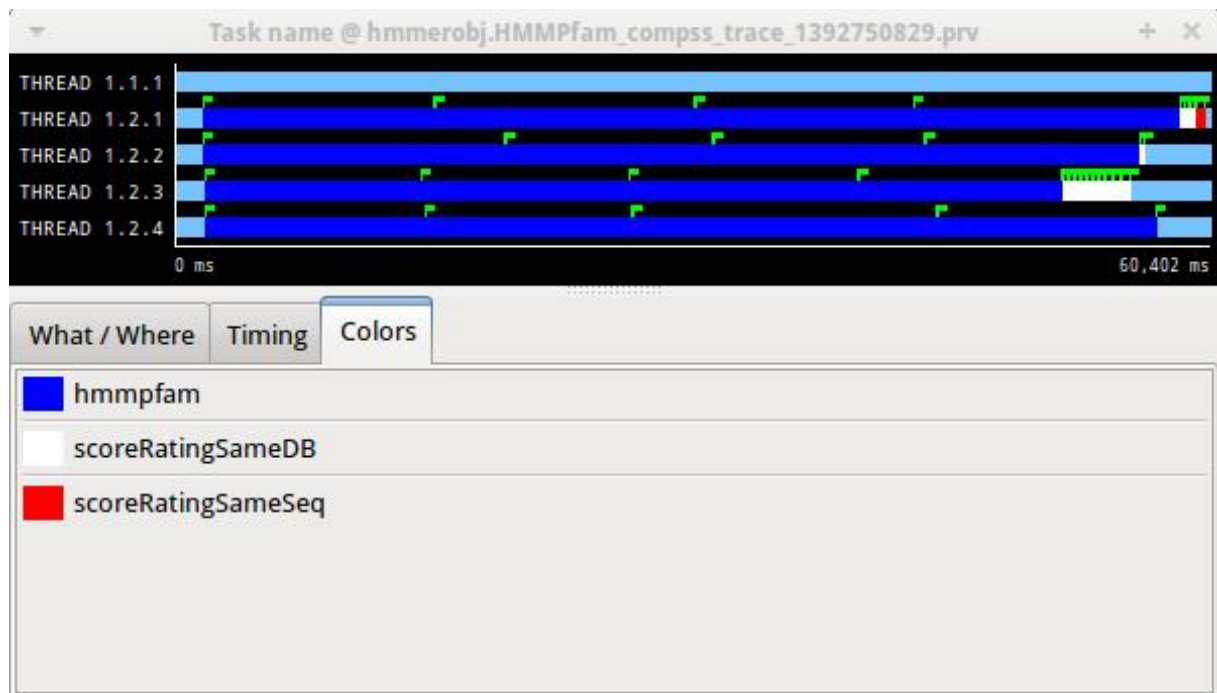


Figure 11: Caption.

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 12: Caption.



Figure 13: 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 14: Caption.



Figure 15: Paraver Menu - New Histogram

New Histogram #1 @ hmmerobj.HMMPfam_compss_trace_1392912552.prv			
	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 16: 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.

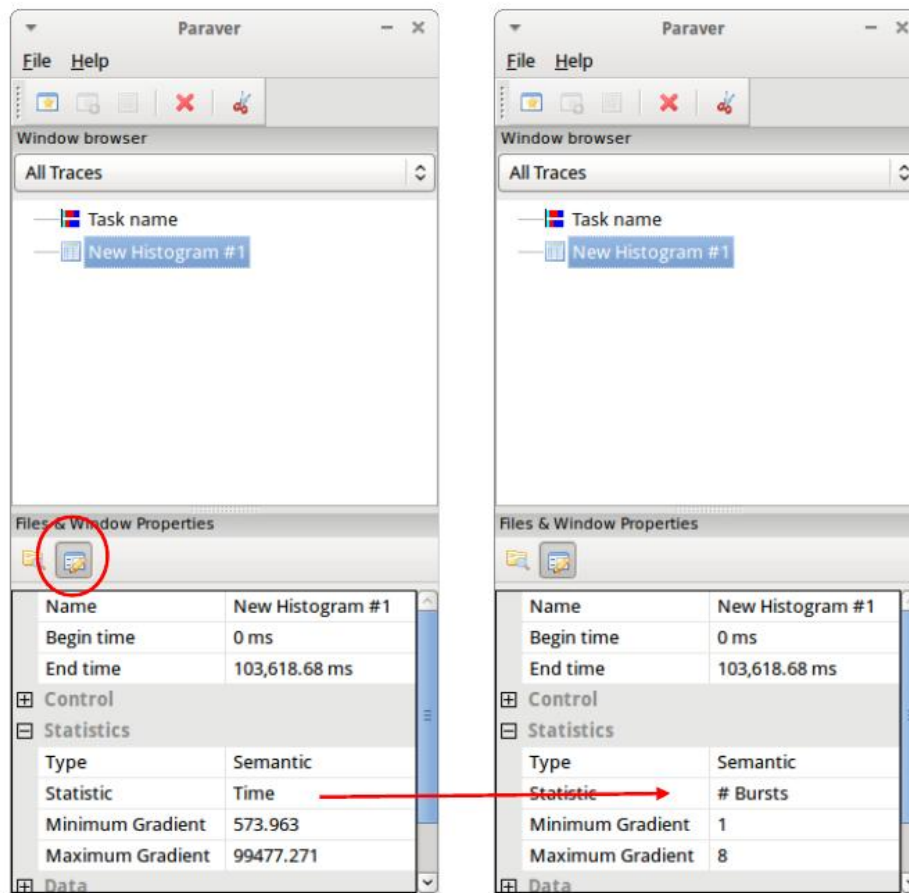
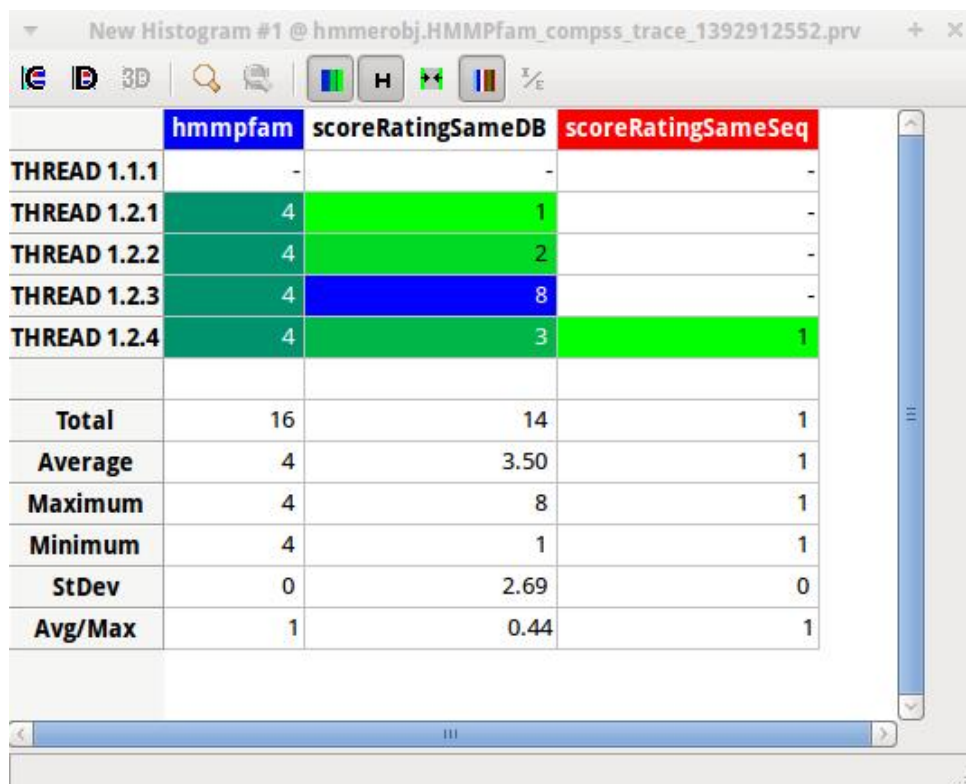


Figure 17: Paraver histogram options menu

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 18: Hmmpfam histogram with the number of bursts

Appendices

Appendix A PAPI: Hardware Counters

The applications instrumentation supports hardware counters through the performance API (PAPI). In order to use it, PAPI needs to be present on the machine before installing COMPSs.

During COMPSs installation it is possible to check if PAPI has been detected in the Extrae config report:

```
Package configuration for Extrae 3.3.0 based on extrae/trunk rev. 3966:
```

```
-----  
Installation prefix: /opt/COMPSs/Dependencies/extrae  
Cross compilation: no
```

```
...  
...  
...
```

```
Performance counters: yes  
Performance API: PAPI  
PAPI home: /usr  
Sampling support: yes
```

PAPI installation and requirements depend on the OS. On Ubuntu 14.04 it is available under `textitpapi-tools` package; on OpenSuse `textitpapi` and `textitpapi-dev`. For more information check https://icl.cs.utk.edu/projects/papi/wiki/Installing_PAPI

Please find more details about the COMPSs framework at [https://www.bsc.es/computer-sciences/
grid-computing/comp-superscalar](https://www.bsc.es/computer-sciences/grid-computing/comp-superscalar)