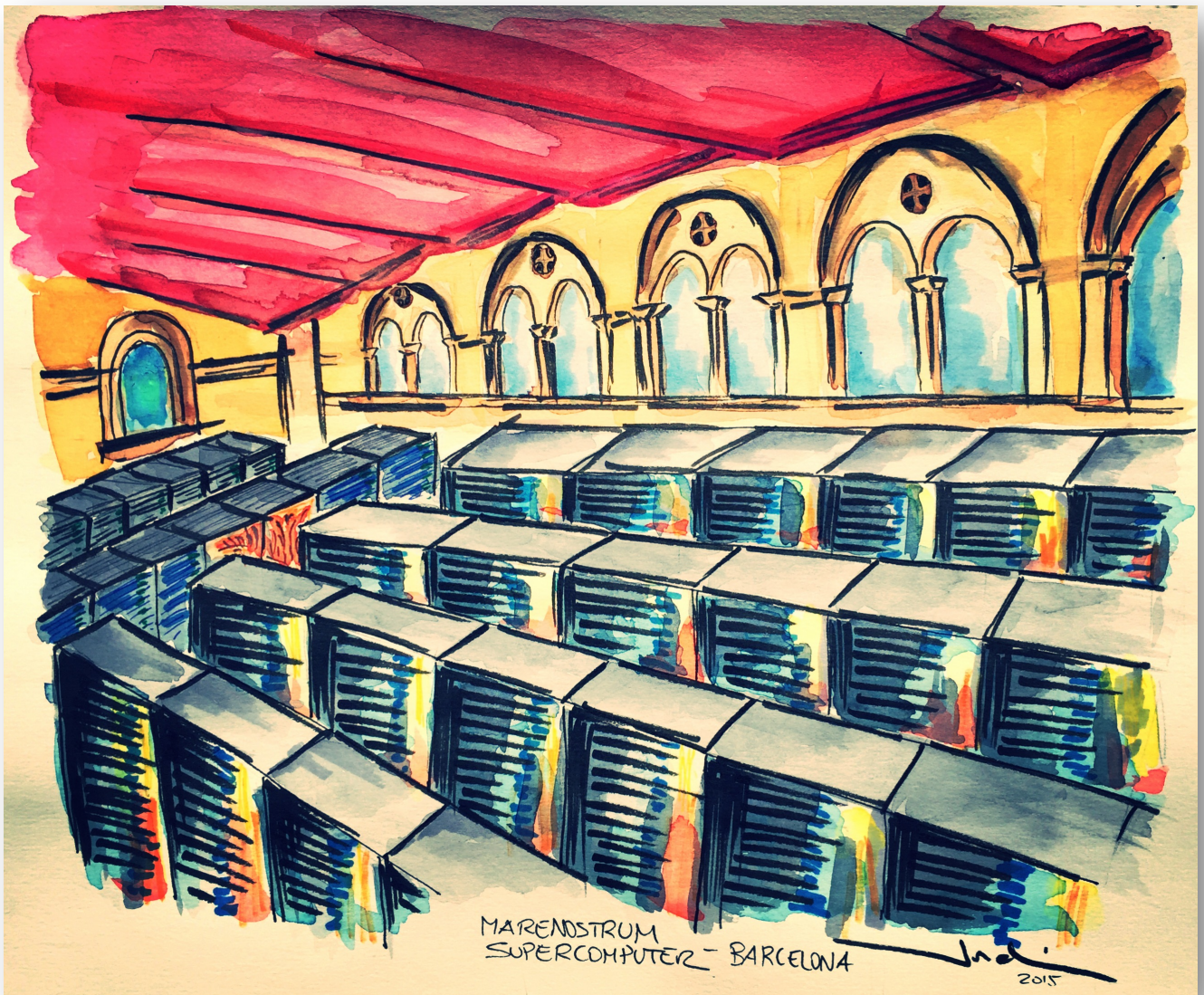


# Hands-on 6: Advanced Performance Tools



## SUPERCOMPUTERS ARCHITECTURE

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS

Barcelona, Fall 2015



UNIVERSITAT POLITÈCNICA  
DE CATALUNYA  
BARCELONATECH



Barcelona  
Supercomputing  
Center  
Centro Nacional de Supercomputación

## **Hands-on 6**

# **Advanced Performance Tools**

**SUPERCOMPUTERS ARCHITECTURE**

**Master in innovation and research in informatics**

**(Specialization High Performance Computing)**

**UPC Barcelona Tech & Barcelona Supercomputing Center**

Version 2.1 - 30 September 2015

Professor: Jordi Torres [jordi.torres@bsc.es](mailto:jordi.torres@bsc.es) [www.JordiTorres.eu](http://www.JordiTorres.eu)



This work is licensed under a [Creative Commons Attribution Share Alike 3.0 Unported License](https://creativecommons.org/licenses/by-sa/3.0/).

## Table of Contents

1	Hands-on description.....	3
2	BSC Tools .....	3
3	Paraver .....	4
3.1	Overview .....	4
3.2	Configuration files .....	4
3.3	Trace file .....	5
3.4	Tool structure .....	5
3.5	Install the Paraver binaries in your laptop.....	6
3.6	Configure the Paraver package (tutorials) .....	6
4	Introduction to Analisis with Paraver .....	7
4.1	Trace.....	7
4.2	Timelines: Navigation and Basic concepts .....	7
4.3	Main Paraver Window.....	9
5	Obtaining useful information.....	11
5.1	Instructions per cycle .....	11
5.2	Cache misses ratio .....	12
5.3	Duration of computation and message size.....	13
6	Profiles.....	13
6.1	2D Analyzer .....	13
6.2	Data Window .....	14
7	Histograms .....	15
8	Extræ .....	17
9	Quick step by step trace generation recipe in Marenostum .....	19
9.1	Module load extræ .....	19
9.2	trace.sh script .....	19
9.3	extræ.xml file.....	20
9.4	LSF file.....	22
9.5	Case Study .....	23
10	Lab Report.....	23

## 1 Hands-on description

This Hands-on will introduce the student to Paraver, a powerful performance visualization and analysis tool based on traces that can be used to analyse any information that is expressed on its input trace format.

In this hands-on you will also find an introductory guide to get execution traces in Marenostum. The tracing tool Extrae supports many different tracing mechanisms, programming models and configurations.

## 2 BSC Tools

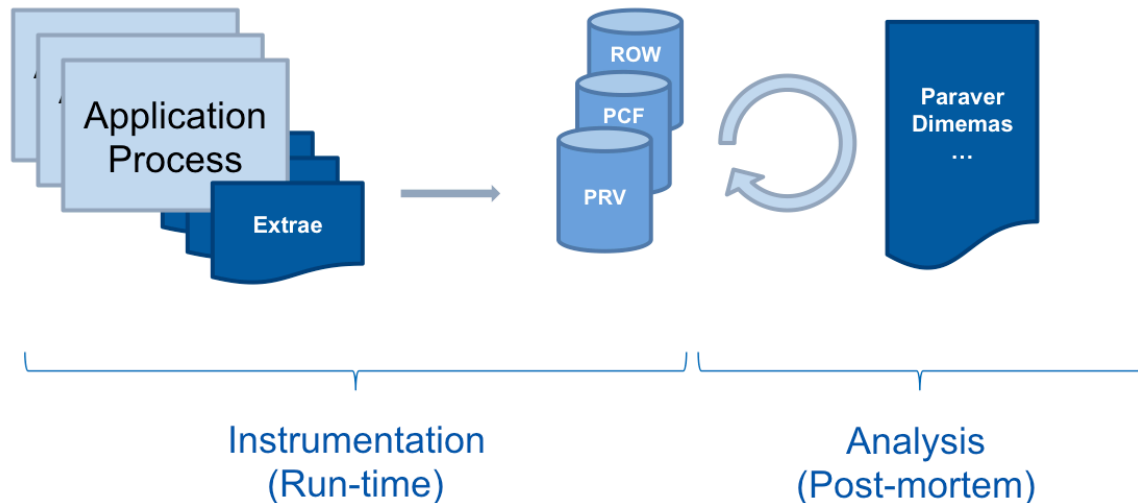
Parallel architectures enable us to target more complex and ambitious problems each year. But in many cases, the achieved performance is far from what the theoretical values promised us. Performance analysis tools allow application developers to identify and characterize the inefficiencies that caused a poor performance. BSC consider that this analysis must be the first step towards the optimization of an application. Optimizing without a previous analysis could be like driving without directions as it could mean wasting efforts improving parts of the code that were not the real performance bottlenecks.

Performance analysis is, in some sense, still an art where the experience and intuition of the analyst drives the analysis and determines the quality of the results. For this reason the definition of methodologies and procedures that would simplify and facilitate the process of extracting information from the performance data is very important.

The set of performance tools under BSC tools umbrella, named also CEPBA-Tools, is currently comprised by:

- Paraver: A very powerful performance visualization and analysis tool based on traces that can be used to analyse any information that is expressed on its input trace format.
- Dimemas: Simulation tool for the parametric analysis of the behaviour of message-passing applications on a configurable parallel platform.
- Extrae: Set of programs and libraries to generate or translate Paraver and Dimemas traces. BSC have packages for instrumenting different programming models under different platforms.
- Utilities: Set of small programs to process Paraver traces to cut, summarize, translate or accumulate the performance data. They can be used independently but they are also integrated within Paraver.

The basic workflow is represented in the next figure:



## 3 Paraver

### 3.1 Overview

Paraver was developed to respond to the need to have a qualitative global perception of the application behavior by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems. Expressive power, flexibility and the capability of efficiently handling large traces are key features addressed in the design of Paraver. The clear and modular structure of Paraver plays a significant role towards achieving these targets.

Its analysis power is based on two main pillars. First, its trace format has no semantics; extending the tool to support new performance data or new programming models requires no changes to the visualizer, just to capture such data in a Paraver trace. The second pillar is that the metrics are not hardwired on the tool but programmed. To compute them, the tool offers a large set of time functions, a filter module, and a mechanism to combine two time lines. This approach allows displaying a huge number of metrics with the available data.

### 3.2 Configuration files

To capture the experts knowledge, any view or set of views can be saved as a Paraver configuration file (.cfg files). After that, re-computing the view with new



data is as simple as loading the saved file. With the Paraver distribution you will download a set of configuration files that you can use for analyzing your codes.

### 3.3 *Trace file*

A trace file has three types of records:

- **State:** Record associating a state value for one thread during a time interval. Paraver associates no semantics to the encoding of the state field.
- **Event:** This record represents a punctual event that happens on one object. It is encoded in two integers (type and value). Paraver associates no semantics to the encoding of these fields.
- **Relation** (communication): event relating two objects in two points in time. Actually represent a pair of events in two different threads and represents a causal relationship between the event happening in the first thread and the second.

### 3.4 *Tool structure*

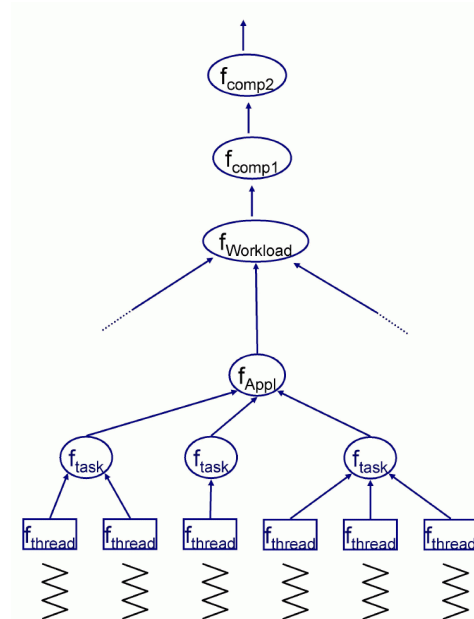
To extract the information from the trace records in a flexible and powerful way, Paraver is structured in three levels of modules: Filter, Semantic and Representation.

The whole operation of Paraver corresponds to a demand driven evaluation, where the Representation module inquires the semantic module for values and this one asks for the records it needs to the filter.

The Filter module has direct access to the trace and provides to the Semantic module a partial view of the tracefile.

The Semantic module is the key of the expressive power of Paraver. Its role is to generate a numeric value (semantic value) for each object to be represented. For each object, the semantic value is a function of time that is computed from the records that correspond to the object following the corresponding object model structure.

For both programming model objects (workload, application, task and thread) and resources model objects (system, node and CPU) the semantic value is hierarchically computed according to the general object model structure. This means that for an object of type thread, the semantic value will be computed by processing the records of that thread. For an object of type task, all the semantic values of its constituent threads will be combined by a task function. Similarly, an application function will be applied to all the semantic values of its constituent tasks.



### 3.5 Install the Paraver binaries in your laptop

You can download Paraver for linux x86 & x86-64, windows and mac, from the BSC-CNS official web page <http://www.bsc.es/computer-sciences/performance-tools/downloads>

The Paraver package is in compressed tar format. To unpackage them execute from the desired target directory the following command line: `gunzip | tar -xvf -`. If you web browser is configured to automatically uncompress the file, you will need to execute `tar -xvf` (use the command file to find the package type).

### 3.6 Configure the Paraver package (tutorials)

After opening the Paraver program the first step is to set-up the tutorials directory. Go into Help > Tutorials, click on Preferences Window. Edit Tutorials root to the directory that contains the tutorial (tools-material/packages/Tutorials).

Now you can open the window Help → Tutorials. In this hands-on lab we will follow the links from the tutorial “Introduction to Analisis with Paraver”. You can use the links in this tutorial in order to open the files (optional).

## 4 Introduction to Analysis with Paraver

### 4.1 Trace

For this lab we will use a paraver trace already created by Extrae. The trace content an execution of the Weather Research & Forecasting Model (<http://www.wrf-model.org>) executed in Marenstrum using data from Iberian Peninsula. The program is written in MPI and executed using 128 threads.

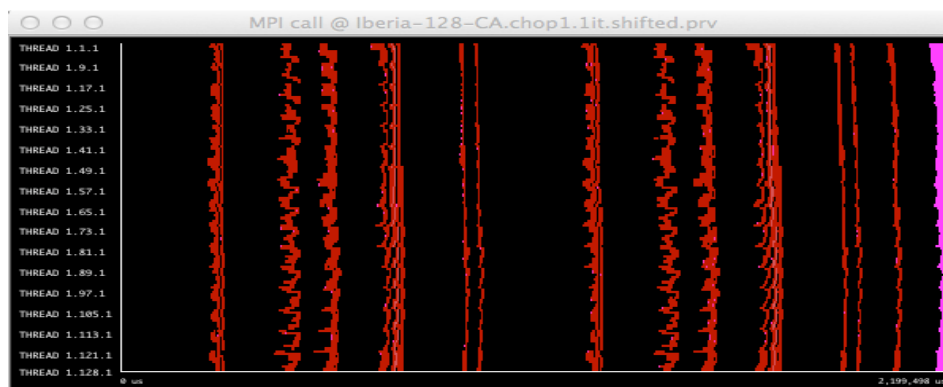
In order to load the trace, go to the main menu, select File → Load Trace..., and open the file *Iberia-128-CA.chop1.1it.shifted.prv*.

**Exercise 1:** (a) Download and install Paraver. (b) Configure the Paraver package (tutorials). (c) Load [Iberia-128-CA.chop1.1it.shifted.prv](#) trace.

Warning: Depending the platform it is sometimes necessary to remove the file `./lib/paraver-kernel/libc.so.6`

### 4.2 Timelines: Navigation and Basic concepts

Let us go through some basic navigation and analysis functionalities.



Select File → Load Configuration..., and select `mpi_call.cfg`.

A display window will appear with the timeline of which MPI call is being executed at each point in time by each process. The horizontal axis represents time, from the start time of the application on the left of the window to the end time at the right. For every thread, the colors represent the MPI call or black when doing user level computation outside of MPI.



Moving the mouse over the display window you will see at the bottom of the window the name of the MPI call (the label corresponding to the pointed color)

Double clicking anywhere inside the window will open the Info Panel with the textual information of the function value in the selected point. Right click inside the window and select the option Info Panel to hide it.

Click with the Left Button of the mouse to select the starting time of the zoomed view, drag the mouse over the area of interest, and release the mouse to select the end time of the zoomed view. Undo Zoom and Redo Zoom commands are available on the Right Button menu. You can do and undo several levels of zooming.

The *Control-Zoom* option will let you select a subset of threads. This is useful when analyzing runs with many processes and you want to concentrate on a few of them. Hold down the “*Control*” or “*CTRL*” key on the keyboard, and using the mouse, identify a rectangular area by clicking on top left corner of the desired area with the left mouse button and dragging and releasing the button on the bottom right corner. The *Control-Zoom* option is not possible for MAC users. In this case you should click the Right Button menu and select “Select objects” option.

Right-click on the window, and select the *View → Event Flags* checkbox. Flags appear at the entry and exit points to the MPI calls. Depending on the scale, displaying flags may help differentiate whether there is one or many MPI calls at a given zoom level.

To measure time between any two points in the trace: Use the Shift-Zoom combination to activate the timing. The time and the interval between the two selected points of the trace is displayed in the *Timing* tab of the *Info Panel*.

**Exercise 2:** Load [mpi\\_call.cfg](#) configuration file. Practice the previous outlined actions. What is the duration of last MPI\_Allreduce call of object THREAD 1.1.1?

In Paraver every timeline window represents a single metric (MPI call, useful IPC,...) for all selected processes and time span. It is possible to synchronize two timelines by making them display the exact same processes and time span. For doing so, just right-click and select *Copy* on the source (reference) window and then on the target window right-click and select *Paste → Size* and *Paste → Time*. Both windows will then be of the same size and represent different views (metrics) for the same part of the trace. If you put one above the other there is a one to one correspondence between points in vertical. The *Paste Time* lets you copy only the time scale from the first window to the target one. The *Paste Default Special* is a shortcut for copying size, time and objects (threads). The

*Paste Semantic Scale* lets you apply to the target window the same vertical scale the source window had.

**Exercise 3:** Practice how to synchronize windows

### **4.3 Main Paraver Window**

The main Paraver Window will keep all the information about the traces and configurations loaded. The top part keeps information of all the traces opened (you can load different traces and select one of them). After that you will see a window with the information of all the configurations loaded. If you select one of them, in the bottom part of the windows you will find the information associated with this configuration file. You can change the parameters at different levels: the filter module (that offers a partial view of the trace) and the Semantic module (that generates a numeric value for each object to be represented).

Paraver

Window browser

/Users/torres/Documents/DADES/2014.TOPS/SA-MIRI/7.LAB.TOOLS/PR.PAR...

- MPI call
  - Useful IPC
    - Instructions per cycle
      - Useful
  - L3 Data cache misses per 1000 instr
    - L3 Data cache misses

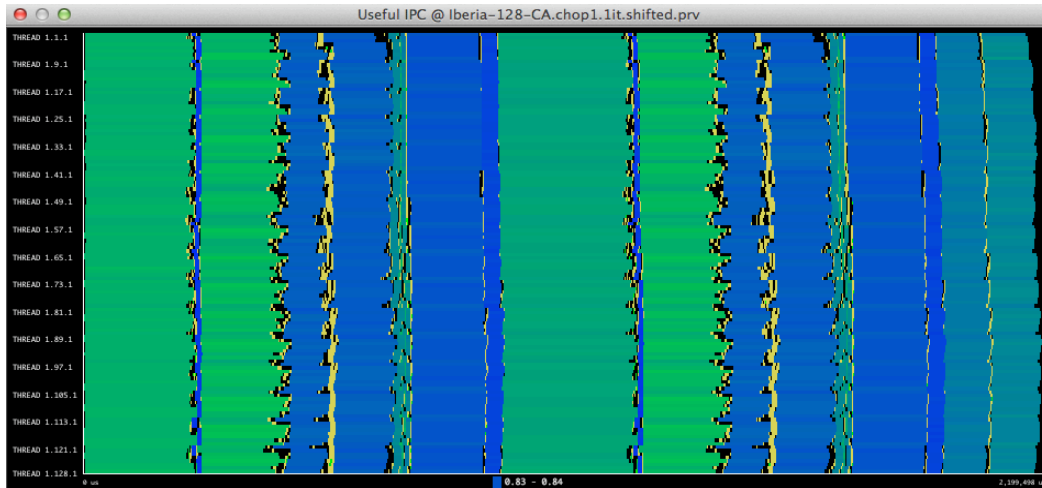
Files & Window Properties

Name	MPI call
Begin time	0 us
End time	691,334.46 us
Semantic Minimum	0
Semantic Maximum	70
Level	Thread
Time unit	Microseconds
Filter	
Communications	
Events	
Semantic	
Top Compose 1	As Is
Top Compose 2	As Is
Compose Thread	As Is
Thread	Last Evt Val

## 5 Obtaining useful information

### 5.1 Instructions per cycle

Load configuration file [useful\\_IPC.cfg](#). From the main menu, select *File* → *Load Configuration...*, and select the configuration file specified.



This configuration file shows a timeline with the instructions per cycle (IPC) achieved in each interval of useful computation.

Right-click on the window, select *Info Panel*, and select the *Color* tab to see the actual coloring scheme and scale for this window. The IPC function of time for each process is represented as a gradient between light green representing a low IPC value and dark blue representing a high IPC value. This view shows in black in the regions where processes are in MPI in order to let us focus on the actual useful computation parts.

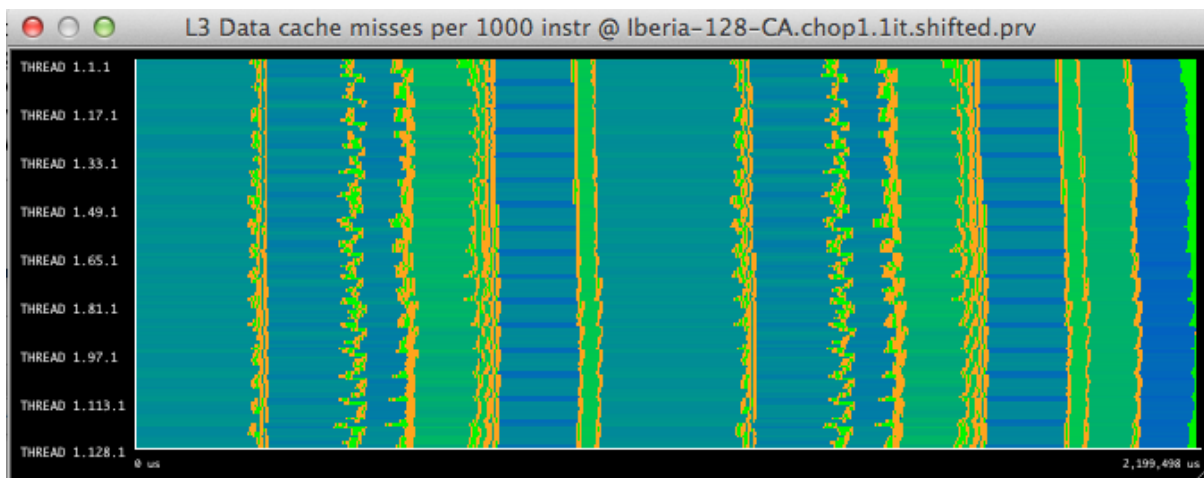
Use the *Control-Zoom* to select a few processes for the whole duration of the trace. Select the *Paint as* option in the window menu (right click) and select *Function Line*. You will see a display of the IPC for the selected processes as a function of time. Both the color and function of time view present the same information. The color scheme is inherently more scalable. Click with the Left Button on any point in the window. It will list in textual form the actual value of IPC at the point selected (an vicinity if time scale too coarse). The text display will be in the *What/Where* tab of the *Info Panel*.

You can change the vertical scale (Y scale) for the function of time representation as well as for the color encoding. To manually control the scale, use the *Semantic Maximum* and *Semantic Minimum* fields in the Main Paraver Window (in the bottom part of the window *Files & Windows Properties*). Just keep in mind

that values outside the specified range will be truncated in the function of time display and will be assigned a different color (orange) in the color display. To automatically fit the scale to the whole dynamic range of the function, inside the window, right-click on the window, and select *Fit Semantic Scale* → *Fit Both* (or *Fit Maximum* or *Fit Minimum*).

## 5.2 Cache misses ratio

Load the configuration [L2missratio.cfg/L3missratio.cfg](#). (depending on the version of PARAVR there are a typographic error in the file name). This configuration file shows a timeline with the L3 cache miss ratio (L3 cache misses per 1000 instructions) in each interval between MPI events.



You can observe that the L3 cache miss ratio is higher in the communication phases than in the long computation phases. If Y scale is not enough to display the whole dynamic range of L3 cache miss ratios some areas can appear as orange (above the Semantic Minimum value) (in this situation a red triangle in the lower left corner can appear). By default the *Semantic Maximum* value in the Main Window is set to 0.5. Again, just keep in mind that values outside the specified range will be truncated in the function of time display and will be assigned a different color (orange) in the color display. You can automatically set the Semantic Minimum and Maximum values to have a linear gradient display for the whole range by right-clicking and selecting *Fit Semantic Scale* → *Fit Both*. In this case the original semantic scale does show the difference between the computation phases.

**Exercise 4:** What is the *Semantic Maximum* value in the Main Window after *fit semantic scale*?

### 5.3 Duration of computation and message size

Load [useful\\_duration.cfg](#) to analyze the duration of a computation burst between an exit from MPI and the next entry. The function is valued to 0 (black) in the regions where processes are in MPI. Load [p2p\\_size.cfg](#) to see the message sizes for each message sent along the time axis.

**Exercise 5:** Where are the major computation phases? Are they correctly balanced across processors?

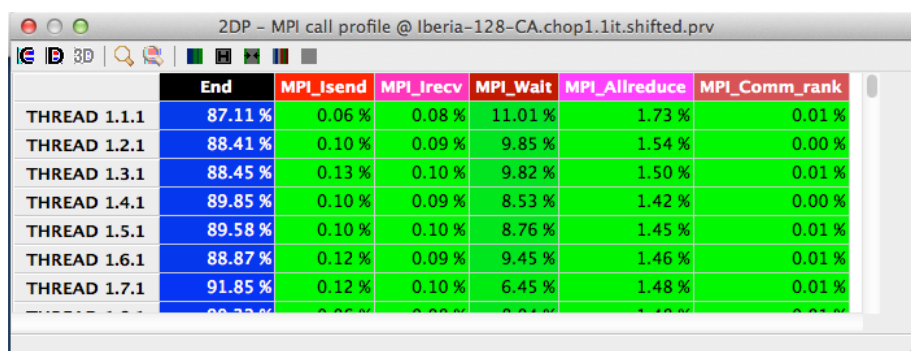
## 6 Profiles

The above analysis went directly to the detailed timeline, but a less detailed averaged statistic can often be sufficient to identify problems and gives a summarized view of the behavior of an application.

### 6.1 2D Analyzer

Paraver provides one mechanism to obtain such profiles for the desired region of a trace. We call it the *2D Analyzer* as it is a very flexible mechanism to generate tables of summarized statistics.

In order to use it load configuration file [mpi\\_stats.cfg](#).




2DP - MPI call profile @ Iberia-128-CA.chop1.1it.shifted.prv

	End	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Allreduce	MPI_Comm_rank
THREAD 1.1.1	87.11 %	0.06 %	0.08 %	11.01 %	1.73 %	0.01 %
THREAD 1.2.1	88.41 %	0.10 %	0.09 %	9.85 %	1.54 %	0.00 %
THREAD 1.3.1	88.45 %	0.13 %	0.10 %	9.82 %	1.50 %	0.01 %
THREAD 1.4.1	89.85 %	0.10 %	0.09 %	8.53 %	1.42 %	0.00 %
THREAD 1.5.1	89.58 %	0.10 %	0.10 %	8.76 %	1.45 %	0.01 %
THREAD 1.6.1	88.87 %	0.12 %	0.09 %	9.45 %	1.46 %	0.01 %
THREAD 1.7.1	91.85 %	0.12 %	0.10 %	6.45 %	1.48 %	0.01 %

A table pops up with one row per thread and one column per MPI call. The first column (End – back color) corresponds to the time outside MPI. Each entry in the table tells the percentage of time the corresponding thread has been inside the



specific call. The table shows the global perception of the profile for all MPI calls and processes.


Click on the magnifying glass  at the top of the icons column on the right of the window to switch between numerical and zoomed out display.

To see a different statistic change the *Statistic selector* in the Main Window (expand the *Statistics* section in the *Window Properties*, if necessary). Interesting options at this time may be:

- *Time*: to get the accumulated time each process has spent in each MPI call.
- *#Bursts*: To count the number of invocations to each call.
- *Average Burst Time*: to compute the average duration of the call.

**Exercise 6:** What are the number of MPI\_Wait invocations by thread 1?

## 6.2 Data Window

All the above statistics are computed based on a single timeline window, which we call the **Control Window** and which can be popped up by clicking on the control window icon  in the top left corner of the window.

In this example, you will see that it is the same MPI call view we had loaded before. The values of the control window determine to which column is a given statistic accumulated/accounted.


The statistic inside a cell can actually be performed on a different window, that we call the **Data Window**. For example, if you select the *Average Value* statistic and you select as the data window the Instructions per cycle window we loaded before, the entry will report the average IPC within the specific MPI call.

To change the Data Window, expand the *Data* section (if necessary) in the *Main Window*, and change the *Window* value by clicking on the value and selecting a window. Change it back to the MPI Call window and to the %Time statistic.

To apply the analysis to a subset of the trace, zoom on any of the timelines to the time region you are interested in. Right-click and select *Copy* on this window and right-click and select *Paste* → *Time* on the table. The analysis will be repeated just for the selected time interval.

**Exercise 7:** Apply the analysis to a subset of the trace

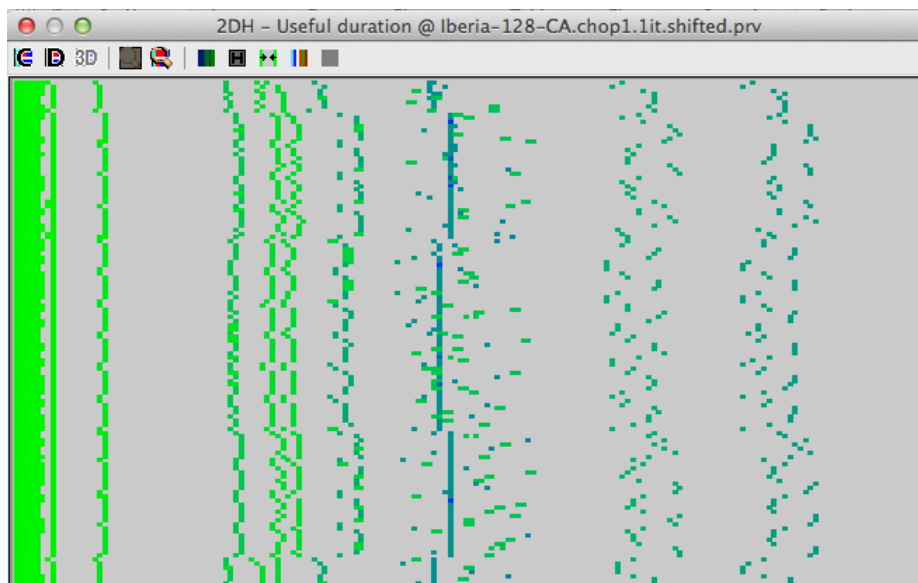
If you want to focus only on the actual MPI call columns (discarding the first column) change the *Control: Minimum* on the Main Window. If the whole table appears as green, you can rescale the gradient coloring scheme by right-clicking in the window and selecting *Autofit Data Gradient*.

**Exercise 8:** Click on the *Open Filtered Control Window* icon  from the top of the Profile Window, and then left-click and drag to select one or two columns in the 2D table. What does the new timeline window created show?

## 7 Histograms

Histograms are a very useful view to analyze the behavior of programs as we discussed in a previous theory class. In Paraver, the same mechanism used to compute profiles can be used to compute histograms.

To get a histogram of continuous valued metrics for this trace load configuration file [2dh\\_useful\\_duration.cfg](#). A table pops up with one row per thread.



The X axis represents bins of a histogram. In this case, every pixel represents a bin of 2000 microseconds (*Delta* value in the *Control* section of the Main Window). The pixels on the left of the table represent short durations, those on the right represent large duration.

Every pixel is colored with the percentage of time that process spent in a computation phase of the length corresponding to the pixel column. The color encoding is light green for a low percentage, dark blue for a high percentage (low

and high are defined by the Minimum and Maximum values in the *Control* section of the *Main Window*). The pixel is colored in grey when there is no value in the corresponding range

Ideally on a balanced SPMD application you would expect vertical lines, representing the different computation bursts are of exactly the same duration for all processes. You can see that in this example there is a certain variance between processors (vertical bands rather than lines).

If you move the cursor over the colored pixels, the bottom of the window will show the actual range represented by the column of the pixel and the actual value (percentage in this case) for that pixel. If you want to see the numerical values of the table click on the magnifying glass at the top of the column of icons at the right of the window. You will see the individual columns, for each of the range of durations it represents is show at the top.

It is possible to change the statistic so that the actual value represented is total time, average duration of the bursts or average value of other metric (such as “IPC”). For example, in the *Statistics* section of the *Main Window*, you can change the statistic selector to *Average value* and select *Instructions Per Cycle* as the data window. You can change the range of the coloring scheme in the *Statistics: Minimum Gradient* and *Statistics: Maximum Gradient* fields in the *Main Window*.

Another useful pece of information is the message size in point to point calls. If you load [3dh\\_msgsize\\_p2pcall.cfg](#), the window actually shows the histogram for a specific MPI call, *MPI\_Isend* in this case as shown at the *3D* section of the *Main Window*. The Statistic (color) represented is the number of messages of that size sent. You can see that there are smaller than large messages.

**Exercise 9:** Change the *Plane* selector in the *3D* section of the *Main Window* to see the histogram of message sizes for a different MPI call. (not required answer in the report for this exercise)

## 8 Extrae

As we introduced in previous sections, we could use Extrae to get execution traces in Marenostum. In this section you will find an introductory guide. For detailed explanations and advanced options, please check the complete Extrae User Guide in the BSC Tools web site : <http://www.bsc.es/computer-sciences/performance-tools/trace-generation/extrae/extrae-user-guide>

The most recent stable version of Extrae is always located at:

```
/apps/CEPBATTOOLS/extrae/latest/default/64
```

This package is compatible with the default MPI runtime in Marenostum (OpenMPI). Packages corresponding to older versions and enabling compatibility with other MPI runtimes (IntelMPI, MVAPICH) can be respectively found under this directory structure:

```
/apps/CEPBATTOOLS/extrae/<choose-version>/<choose-runtime>/64
```

In order to trace an execution, you have to load the module extrae and write a script that sets the variables to configure the tracing tool. Let's call this script trace.sh. It must be executable (chmod +x ./trace.sh). Then your job needs to run this script before executing the application.

Example for MPI jobs:

```
#!/bin/bash
#BSUB -n 128
#BSUB -o output_%J.out
#BSUB -e output_%J.err
#BSUB -R "span[ptile=16]"
#BSUB -J job_name
#BSUB -W 00:10

module load extrae

mpirun ./trace.sh ./app.exe
```

Example for threaded (OpenMP or pthreads) jobs:

```
#!/bin/bash
#BSUB -n 1
#BSUB -oo output_%J.out
#BSUB -eo output_%J.err
#BSUB -J job_name
#BSUB -W 00:10

module load extrae

./trace.sh ./app.exe
```

Example of trace.sh script:

```
#!/bin/bash

export EXTRAE_CONFIG_FILE=./extrae.xml
export LD_PRELOAD=${EXTRAE_HOME}/lib/<tracing-library>
$*
```

Where:

- EXTRAE\_CONFIG\_FILE points to the Extrae configuration file. Editing this file you can control the type of information that is recorded during the execution and where the resulting trace file is written, among other parameters. By default, the resulting trace file will be written into the current working directory. Configuration examples can be found at: \${EXTRAE\_HOME}/share/examples
- <tracing-library> depends on the programming model the application uses:

Job Type	Tracing library	An example to get started
MPI	libmpitrace.so (C codes) libmpitracef.so (Fortran Codes)	MPI/ld-preload/job.lsf
OpenMP	libomptrace.so	OMP/run_ldpreload.sh
Pthreads	libpttrace.so	PTHREAD/README
OmpSs	-	OMPSS/job.lsf
Sequential job (manual instrumentation)	libseqtrace.so	SEQ/run_instrumented.sh
*Automatic instrumentation of user functions and parallel runtime calls	-	SEQ/run_dyninst.sh

\* Jobs that make explicit calls to the Extrae API do not load the tracing library via LD\_PRELOAD, but link with the libraries instead.

\*\* Jobs using automatic instrumentation via Dyninst neither load the tracing library via LD\_PRELOAD nor link with it. For more advanced options, please check the complete Extrae User Guide: <http://www.bsc.es/computer-sciences/performance-tools/trace-generation/extrae/extrae-user-guide>.

Extrae have support for other programming models and their combinations: Serial, MPI, OpenMP, pthread, SMPss, nanos/OMPss, CUDA, OpenCL, Java.

## 9 Quick step by step trace generation recipe in Marenostrium

In this section we will describe all the steps required to generate traces. In the next hands-on we will analyze these traces in more detail with Paraver.

### 9.1 *Module load extrae*

First of all we need to be sure that the environment module `extrae` will be correctly loaded. Check it with the following command:

```
$ module load extrae
```

### 9.2 *trace.sh script*

The `trace.sh` script file should contains:

```
#!/bin/bash

#Workaround for tracing in MN3, make TMPDIR point to an existing dir
if [ ! -z "${TMPDIR}" ]; then
  export TMPDIR=$TMPDIR/extrae
  mkdir -p $TMPDIR
fi

export EXTRAE_CONFIG_FILE=./extrae.xml
export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitrace.so # For C apps

## Run the desired program
$*
```

*Note: the tracing library for C is `/lib/libmpitrace.so`.*



### 9.3 *extrae.xml* file

In the previous `trace.sh` file we point `./extrae.xml` as a `EXTRAE_CONFIG_FILE`. Remember that with this file we can control the type of information that is recorded during the execution and where the resulting trace file is written, among other parameters. Enclosed you will find the configuration file that can be used in order to obtain the required information for analyzed before codes.

```
<?xml version='1.0'?>

<trace enabled="yes"
home="/apps/CEPBATOOLS/extrae/2.5.2/openmpi/64"
initial-mode="detail"
type="paraver"
xml-parser-id="Id: xml-parse.c 2327 2013-11-22 11:47:07Z harald $"
>

  <mpi enabled="yes">
    <counters enabled="yes" />
  </mpi>

  <openmp enabled="yes">
    <locks enabled="no" />
    <counters enabled="yes" />
  </openmp>

  <pthread enabled="no">
    <locks enabled="no" />
    <counters enabled="yes" />
  </pthread>

  <callers enabled="yes">
    <mpi enabled="yes">1-3</mpi>
    <sampling enabled="no">1-5</sampling>
  </callers>

  <user-functions enabled="no" list="/home/bsc41/bsc41273/user-functions.dat" exclude-
automatic-functions="no">
    <counters enabled="yes" />
  </user-functions>

  <counters enabled="yes">
    <cpu enabled="yes" starting-set-distribution="1">
      <set enabled="yes" domain="all">
        PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM, PAPI_L2_DCM, PAPI_L3_TCM
        <sampling enabled="no" frequency="100000000">PAPI_TOT_CYC</sampling>
      </set>
      <set enabled="yes" domain="user" changeat-globalops="5">
        PAPI_TOT_INS,PAPI_FP_INS,PAPI_TOT_CYC
      </set>
    </cpu>
  </counters>

  <network enabled="no" />
  <resource-usage enabled="no" />
  <memory-usage enabled="no" />
</counters>
```

```

<storage enabled="no">
  <trace-prefix enabled="yes">TRACE</trace-prefix>
  <size enabled="no">5</size>
  <temporal-directory enabled="yes">/scratch</temporal-directory>
  <final-directory enabled="yes">/gpfs/scratch/bsc41/bsc41273</final-directory>
>
  <gather-mpits enabled="no" />
</storage>

<buffer enabled="yes">
  <size enabled="yes">500000</size>
  <circular enabled="no" />
</buffer>

<trace-control enabled="no">
  <file enabled="no" frequency="5M">/gpfs/scratch/bsc41/bsc41273/control</file>
>
  <global-ops enabled="no"></global-ops>
  <remote-control enabled="no">
    <signal enabled="no" which="USR1"/>
  </remote-control>
</trace-control>

<others enabled="no">
  <minimum-time enabled="no">10M</minimum-time>
</others>

<bursts enabled="no">
  <threshold enabled="yes">500u</threshold>
  <mpi-statistics enabled="yes" />
</bursts>

<cell enabled="no">
  <spu-file-size enabled="yes">5</spu-file-size>
  <spu-buffer-size enabled="yes">64</spu-buffer-size>
  <spu-dma-channel enabled="no">2</spu-dma-channel>
</cell>

<sampling enabled="no" type="default" period="50m" variability="10m" />

<dynamic-memory enabled="no" />

<input-output enabled="no" />

<merge enabled="yes"
  synchronization="default"
  tree-fan-out="16"
  max-memory="512"
  joint-states="yes"
  keep-mpits="yes"
  sort-addresses="yes"
  overwrite="yes"
/>

</trace>

```

## 9.4 LSF file

Enclosed you will find a LSF example file that you can use as a template to generate your LSF file.

```
#!/bin/bash
#BSUB -J MPI_MAT_VEC_MUL
#BSUB -n 64
#BSUB -W 00:55
#BSUB -oo output_%J.out
#BSUB -eo output_%J.err
#BSUB -R"span[ptile=16]"
#BSUB -x

# set application and parameters
echo $PWD
module load extrae
module list

APP="mpi_mat_vect_mult"
SIZE=4096
NP=64

mpirun -np $NP ./trace.sh ./APP $SIZE > out-$APP-$SIZE-$NP

mv ./APP.pcf ./APP-$SIZE-$NP.pcf
mv ./APP.row ./APP-$SIZE-$NP.row
mv ./APP.prv ./APP-$SIZE-$NP.prv
```

Notice that extrae generates the paraver files (.prv, .pcf, .row) using the default name. Be careful not to overwrite files.

And finally ...

```
$ bsub -q debug <mat_vec.mpi.extrae.lsf
```

In the standard out file you will find information about the monitoring process. The last lines should say “mpi2prv: Congratulations! \$APP.prv has been generated”.

## 9.5 Case Study

**Exercise 10 (OPTIONAL) (2/10 points):** Generate a paraver file for one of the executions in table 5.4 from exercise 8 in Hands-on 5. Open the trace with paraver and capture some views.

## 10 Lab Report

You have one week to deliver a report with the answers to the exercises.

*Acknowledgement: Part of this hands-on is based on BSC Paraver tutorials. Thank you to Judit Gimenez from Tools department at BSC and Miguel Bernabeu from BSC Operations department for his invaluable help preparing this hands-on.*