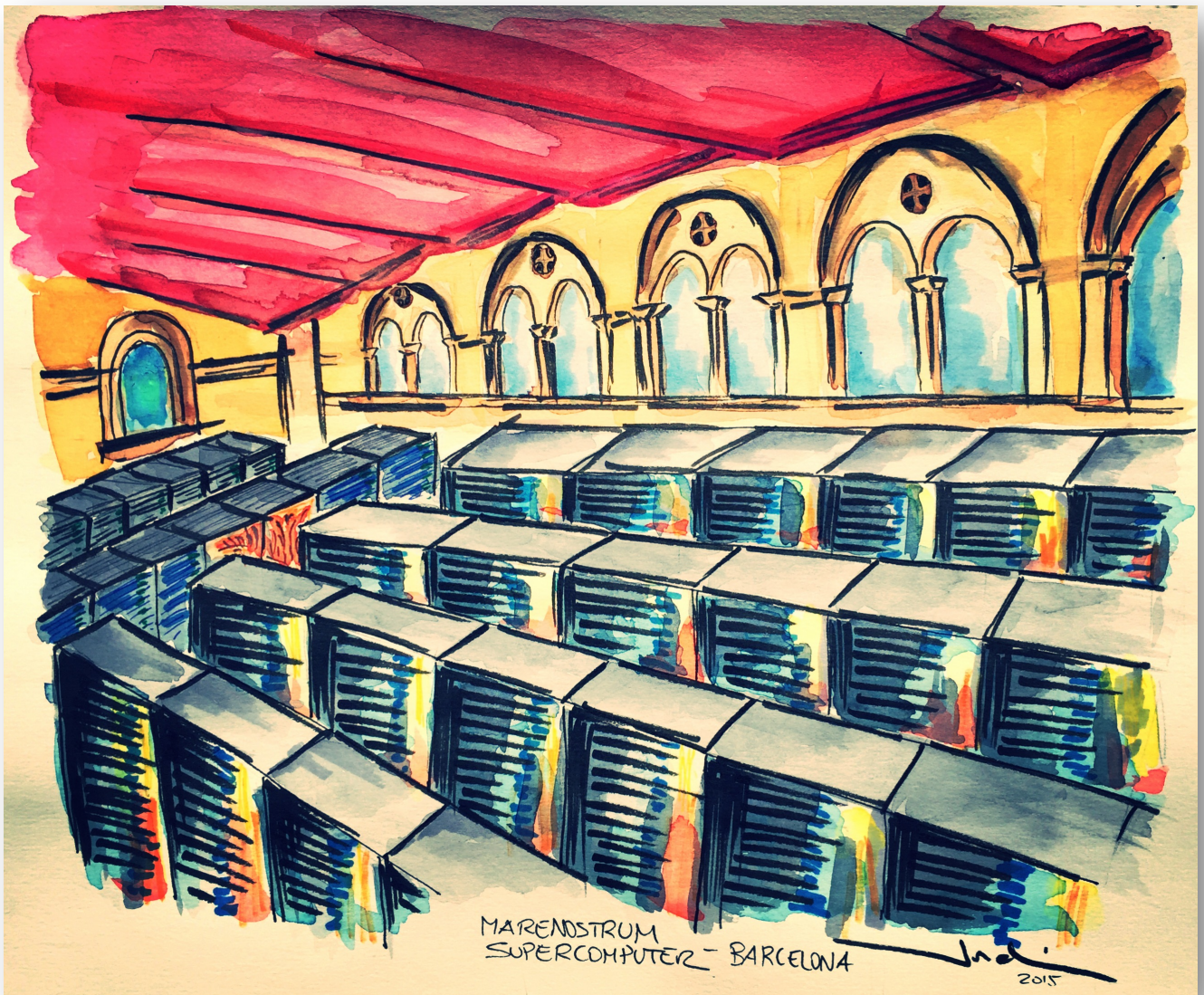


Hands-on 7:

Analysing Jobs with BSC 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 7

Analysing Jobs with BSC Tools

SUPERCOMPUTERS ARCHITECTURE

Master in innovation and research in informatics

(Specialization High Performance Computing)

UPC Barcelona Tech & Barcelona Supercomputing Center

Version 2.0 - 30 September 2015

Autors: Jordi Torres, jordi.torres@bsc.es



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 | Deep Analysis using Paraver | 3 |
| 2.1 | Goal | 3 |
| 2.2 | Case Study | 3 |
| 2.3 | Support files..... | 4 |
| 2.4 | Traces generation..... | 4 |
| 2.5 | Paraver analysis | 6 |
| 3 | Lab Report..... | 16 |

1 Hands-on description

In this Hands-On the student will practise the use of PAVAVER in order to discover the detailed behavior of our parallel program in order to understand their performance.

2 Deep Analysis using Paraver

2.1 Goal

As we introduced in a previous hands-on, Paraver was developed to respond to the need for 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. We will use this tool to analyze some of the jobs executed during hands-on 5.

2.2 Case Study

We will center our analysis in table 5.4 from exercise 8 in Hands-on 5. Remember that we explored the behavior of our program with different numbers of threads per node (e.g. 4,8,16). We can use the job directive

```
#BSUB -R "span[ptile=number]"
```

that indicates the number of processes assigned to a node.

Remember that we concluded that the time to execute our program (with problem size $N=8192$) with 16 threads has a very different performance behavior if we execute all 16 threads in one node ($ptile=16$) or we execute the threads in two nodes ($ptile=8$, 8 threads per node).

We want to discover more details about these results. For this purpose we will generate a couple of traces.

In order to have a small size problem to help the learning process, we suggest using the version of matrix-vector multiplication introduced in the previous hands-on x100 instead of x1000.

2.3 Support files

You can download the files required for this hands-on7 (and tar -xvzf SA-MIRI-Hands-on7.tar.gz) from:

<http://www.jorditorres.org/wp-content/uploads/2012/03/SA-MIRI-hands-on7.tar.gz>

2.4 Traces generation

Exercise 1: Generate the two traces.

Be sure to compile the correct version of the code (x100 as we suggest):

```
mpicc -o mvmmmpi mvmmmpi.c
```

We suggest using the two LSF examples to generate your traces:

a) Trace with ptile=16:

```
#!/bin/bash

#BSUB -J "SA-MIRI-extrae"
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -W 00:55
#BSUB -x
#BSUB -R"span[ptile=16]"
#BSUB -n 16

module load extrae
APP="mvmmmpi"

SIZE=8192

Mpirun ./trace.sh ./APP $SIZE
mv ./APP.pcf ./APP-$SIZE-$LSB_MAX_NUM_PROCESSORS.ptile16.pcf
mv ./APP.row ./APP-$SIZE-$LSB_MAX_NUM_PROCESSORS.ptile16.row
mv ./APP.prv ./APP-$SIZE-$LSB_MAX_NUM_PROCESSORS.ptile16.prv
```

b) Trace with ptile=8:

```
#!/bin/bash

#BSUB -J "SA-MIRI-extrae"
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -W 00:55
#BSUB -x
#BSUB -R"span[ptile=8]"
#BSUB -n 16

module load extrae
APP="mvmmmpi"

SIZE=8192

mpirun ./trace.sh ./APP $SIZE
mv ./APP.pcf ./APP-$SIZE-$LSB_MAX_NUM_PROCESSORS.ptile8.pcf
mv ./APP.row ./APP-$SIZE-$LSB_MAX_NUM_PROCESSORS.ptile8.row
mv ./APP.prv ./APP-$SIZE-$LSB_MAX_NUM_PROCESSORS.ptile8.prv
```

Before submitting the jobs be sure to have the appropriate files in the local directory:

```
bsc31991@login1:~/Hands-on7> ls -ltr
total 144
-rwxr-xr-x 1 bsc31991 bsc31 mvmmmpi
-rwxr-xr-x 1 bsc31991 bsc31 trace.sh
-rw-r--r-- 1 bsc31991 bsc31 extrae.xml
-rw-r--r-- 1 bsc31991 bsc31 job.extrae.8192.ptile16.lsf
-rw-r--r-- 1 bsc31991 bsc31 job.extrae.8192.ptile8.lsf
```

Check that your jobs finished correctly. You can inspect the *.out file. You have to find that *"mpi2prv: Congratulations! mvmmmpi.prv has been generated."* and *"Successfully completed"* at the end of the file.

```
bsc31991@login1:~/Hands-on7> ls -ltr mvmmmpi*
mvmmmpi-8192-16.ptile8.prv
mvmmmpi-8192-16.ptile8.pcf
mvmmmpi-8192-16.ptile8.row
mvmmmpi-8192-16.ptile16.prv
mvmmmpi-8192-16.ptile16.pcf
mvmmmpi-8192-16.ptile16.row
```


2.5 Paraver analysis

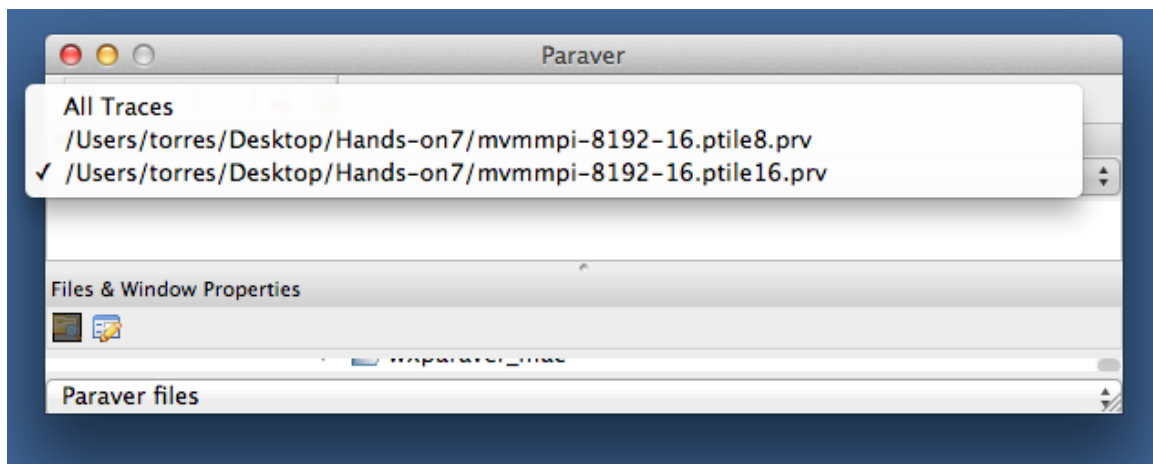
Exercise 2: Copy the trace files in your local computer and run paraver.

Remember that in the previous hands-on we installed Paraver in our local computer. Now it is time to move those paraver files generated in Marenostum into the same directory where you have installed paraver. As we practised in the first hands-on we can use the “scp” command:

```
torres$ scp bsc31991@mn1.bsc.es:Hands-on7/mvmmmpi* .
mvmmmpi-8192-16.ptile16.pcf      100% 3027   3.0KB/s   00:00
mvmmmpi-8192-16.ptile16.prv     100% 65KB  64.6KB/s   00:00
mvmmmpi-8192-16.ptile16.row     100% 475   0.5KB/s   00:00
mvmmmpi-8192-16.ptile8.pcf     100% 3027   3.0KB/s   00:00
mvmmmpi-8192-16.ptile8.prv     100% 65KB  64.5KB/s   00:00
mvmmmpi-8192-16.ptile8.row     100% 484   0.5KB/s   00:00
```

2.5.1 Load the traces

The first thing to do is load the two traces:



2.5.2 Global view: Timelines with MPI calls

Remember that in order 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. We will do the analysis of our traces with prebuild configuration files already used in the previous Hands-on and included in the downloadable tar file.

We will start with the *mpi-call.cfg* configuration file. Remember that for this view the horizontal axis represents time, from the start time of the application on the left of the window to the end time on the right. For every thread, the colors represent the MPI call or black(blue depending of the paraver version) when doing user level computation outside of MPI.

Remember that 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).

Exercise 3: Load the “mpi-call.cfg” configuration file to obtain the timeline with MPI calls for both traces. Identify the different MPI calls involved in each process. Identify the process 0. Synchronize the two timelines and comment on the results. Include the views in answers in the following questions.

To synchronize the two timelines, just right-click and select Copy on the source (ptile16) window and then on the target window (ptile8) right-click and select Paste → Size and Paste → Time. Both windows will then be of the same size and same part of the trace. If you put one above the other there is a one to one correspondence between points in vertical.

2.5.3 Focus of analysis: Computation part

Remember that in Table 5.4 we only measure the computation time between “MPI_Allgather” and “MPI_Gather “ (see the code in Annex of Hands-on 5):

```
.
.
MPI_Allgather(local_x, local_n, MPI_DOUBLE, x, local_n, MPI_DOUBLE, comm);

if (my_rank == 0) { gettimeofday(&start_time, NULL); }

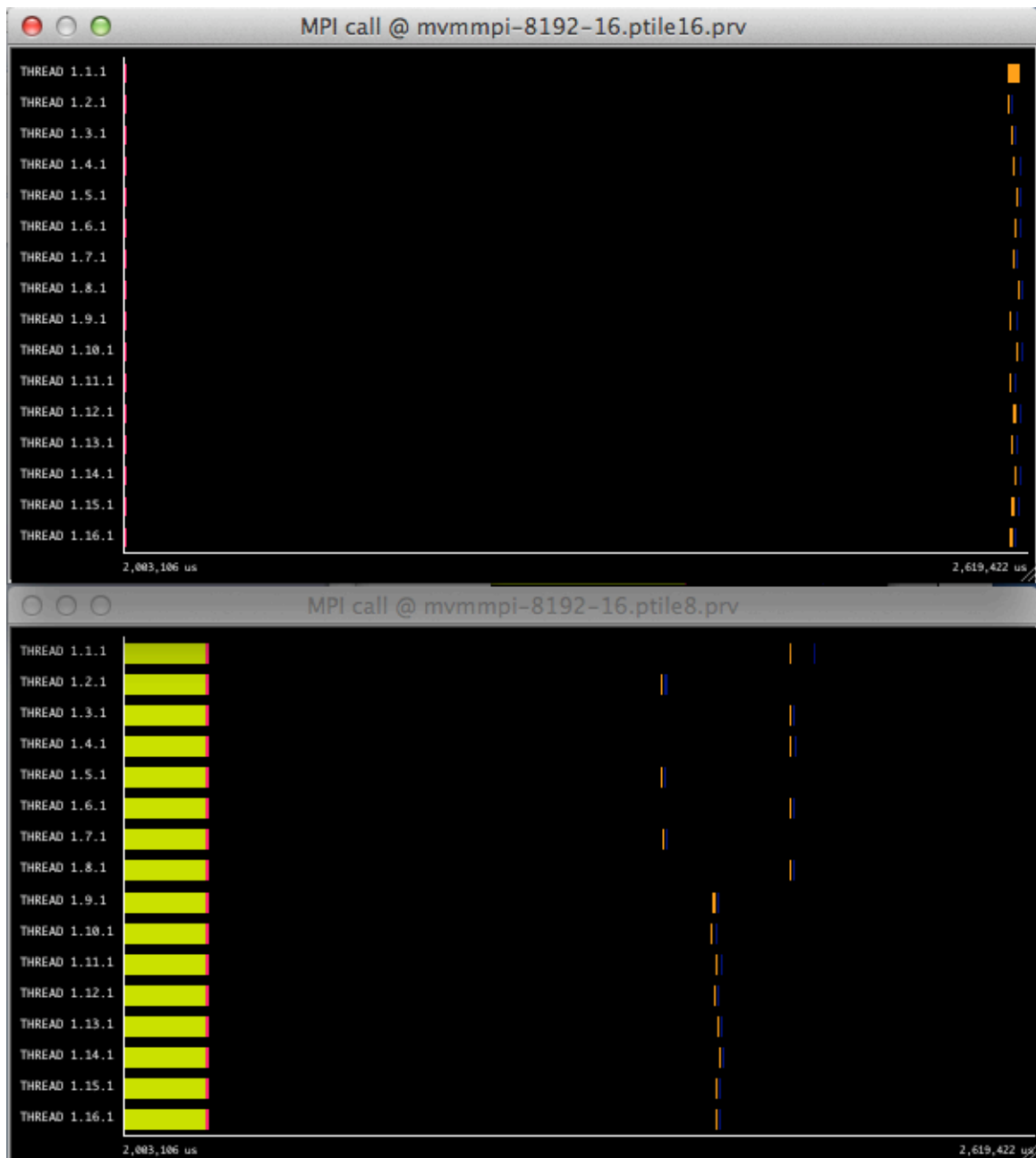
    for (noreal_i = 0; noreal_i < 100; noreal_i++){ // x100 iterations
        for (local_i = 0; local_i < local_m; local_i++) {
            local_y[local_i] = 0.0;
            for (j = 0; j < n; j++)
                local_y[local_i] += local_A[local_i*n+j]*x[j];
        }
    }

if (my_rank == 0) { gettimeofday(&end_time, NULL); print_times(); }

//Vector Gather
MPI_Gather(local_y, local_m, MPI_DOUBLE, vec, local_m, MPI_DOUBLE, 0, comm);
.
.
.
```

Exercise 4: Use the zoom to identify and synchronize this part in both views. What do you observe?

Left click 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.




2.5.4 Average statistic to identify problems


Although the application is balanced, previous timelines show different execution times for this part of the code. How can we do a more in depth analysis?

Remember from the previous hands-on that histograms are a very useful view to analyze the behavior of programs. To get a histogram of continuous valued metrics for these traces we can use the configuration file *2dh_useful_duration.cfg* shown in the previous hands-on that includes the parts of code that are not MPI calls.

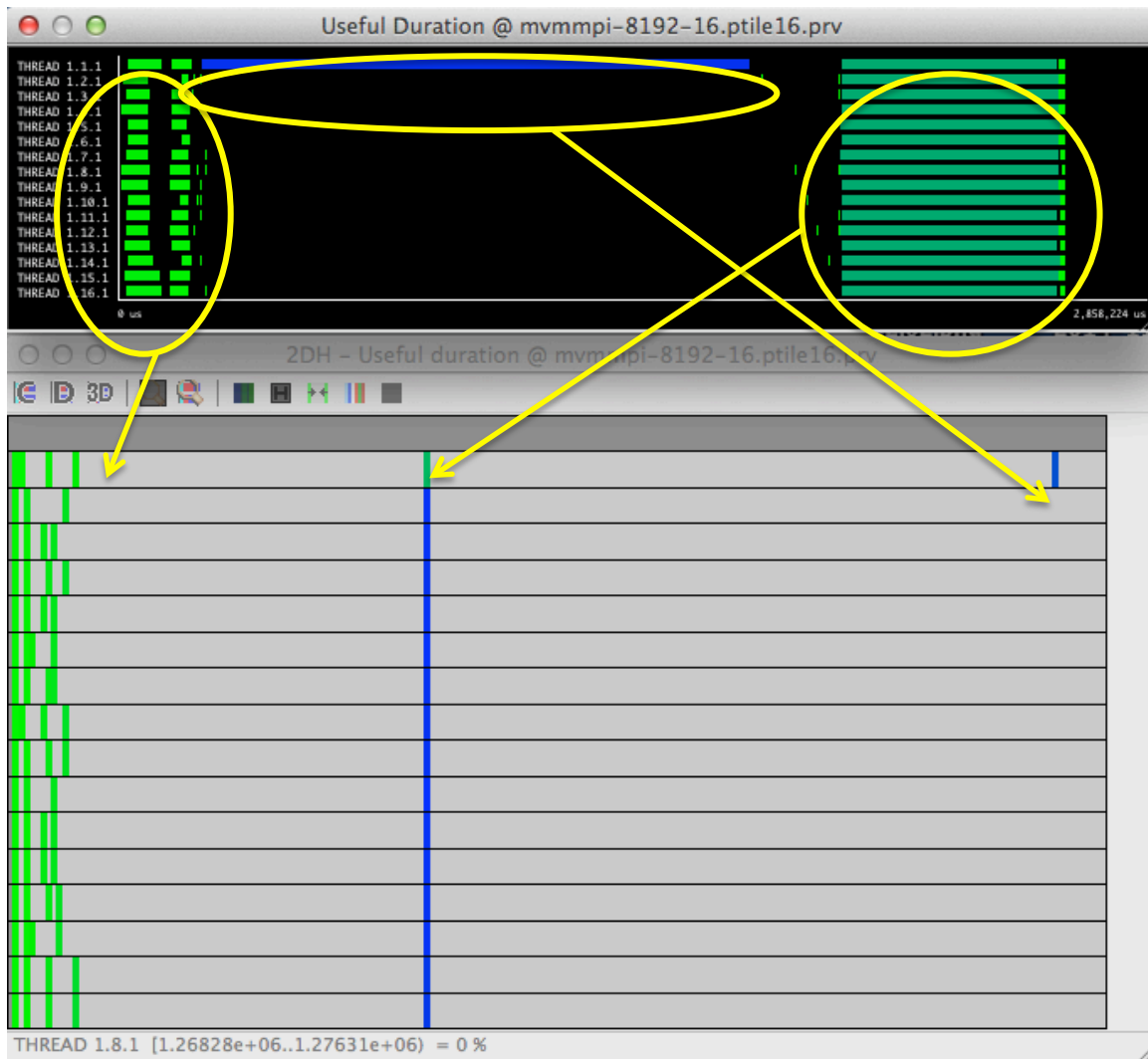
After load configuration file *2dh_useful_duration.cfg*, a table pops up with one row per thread where the X axis represents bins of a histogram (bin of 2000 microseconds, the *Delta* value in the *Control* section of the Main Window). The bins on the left of the table represent short durations, those on the right represent long duration. Every bin is colored with the percentage of time that process spent in a computation phase of the length corresponding to the bin column. Remember that 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.


Exercise 5: Load *2dh_useful_duration.cfg* configuration file and discuss the results for both traces.

IMPORTANT: If a warning like  appears in the window (top right corner) you can select *Auto Fit Control Scale* from the Right Button menu, it means that the values are not correctly presented.

In order to understand this table a little bit click on the *control window icon* on one of the traces (e.g. ptile=16 trace). 

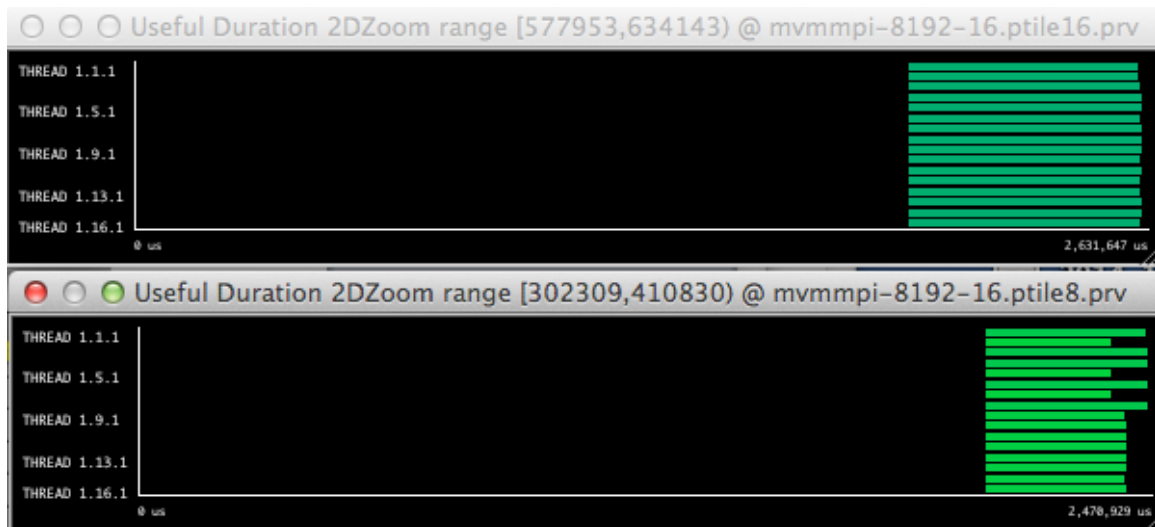
It is easy to identify the main parts as it is indicated in the following figure.



Because we are interested in analysing only the matrix-vector multiplication code part (not communication part), we can click on the *Open Filtered Control Window* icon  and then left-click and drag to select the columns in *2DH_Useful_duration* window that represent this part of the code.

Exercise 6: Click the *Open Filtered Control Window* icon and select the columns that represent the vector-matrix multiplication core. Regarding the time required for each process,

- Compare the time required in each execution
- What happens with threads 1,3,4,6 and 8?

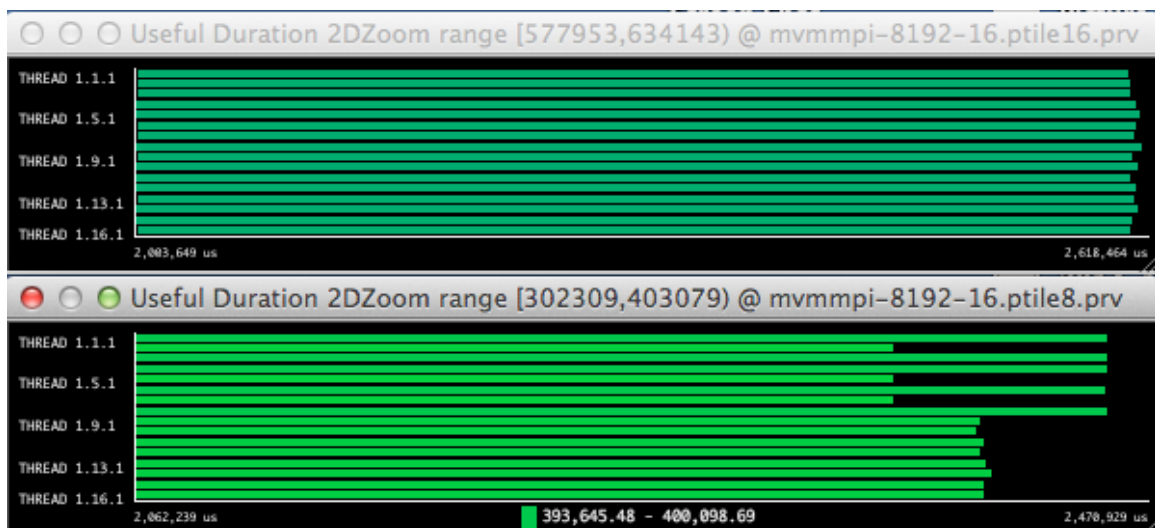


2.5.5 Cache misses

Remember we assumed that the problem is related with the memory access. Let's check it.

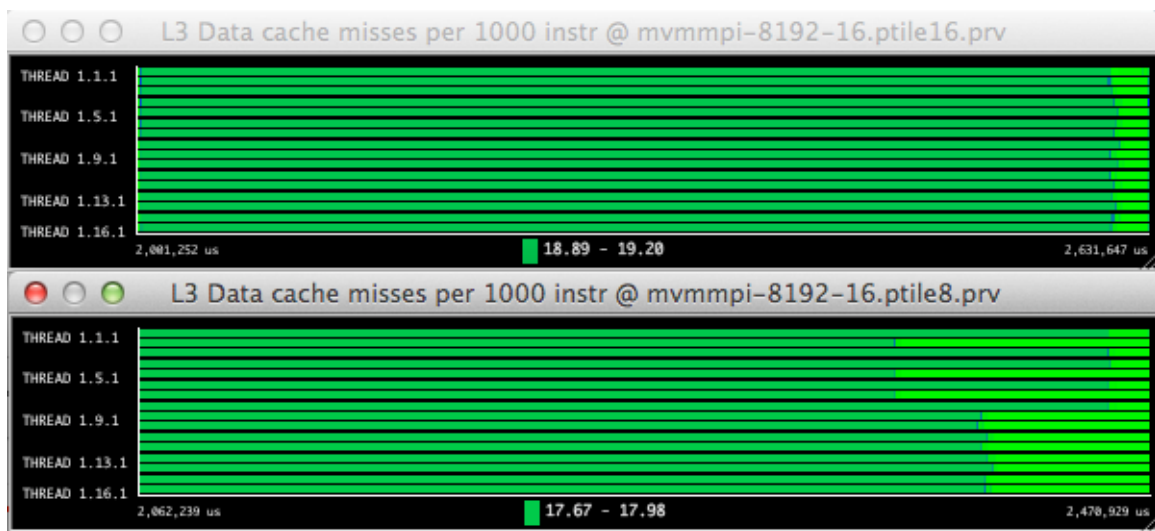
One way to analyze it is to consider the L3 cache misses. A L3 miss means that the processor needs to go to the main memory to obtain the required parameter with a higher latency.


The first step is to select the part of the time line that we are interested in to analyze in more depth using the zoom tool and synchronize both windows.



Now, when we have a window with the bursts that focus our analysis, we can load the *L3_miss_ratio.cfg* configuration file that shows a timeline with the L3 cache miss ratio (L3 misses per 1000 instructions) in each interval between events.

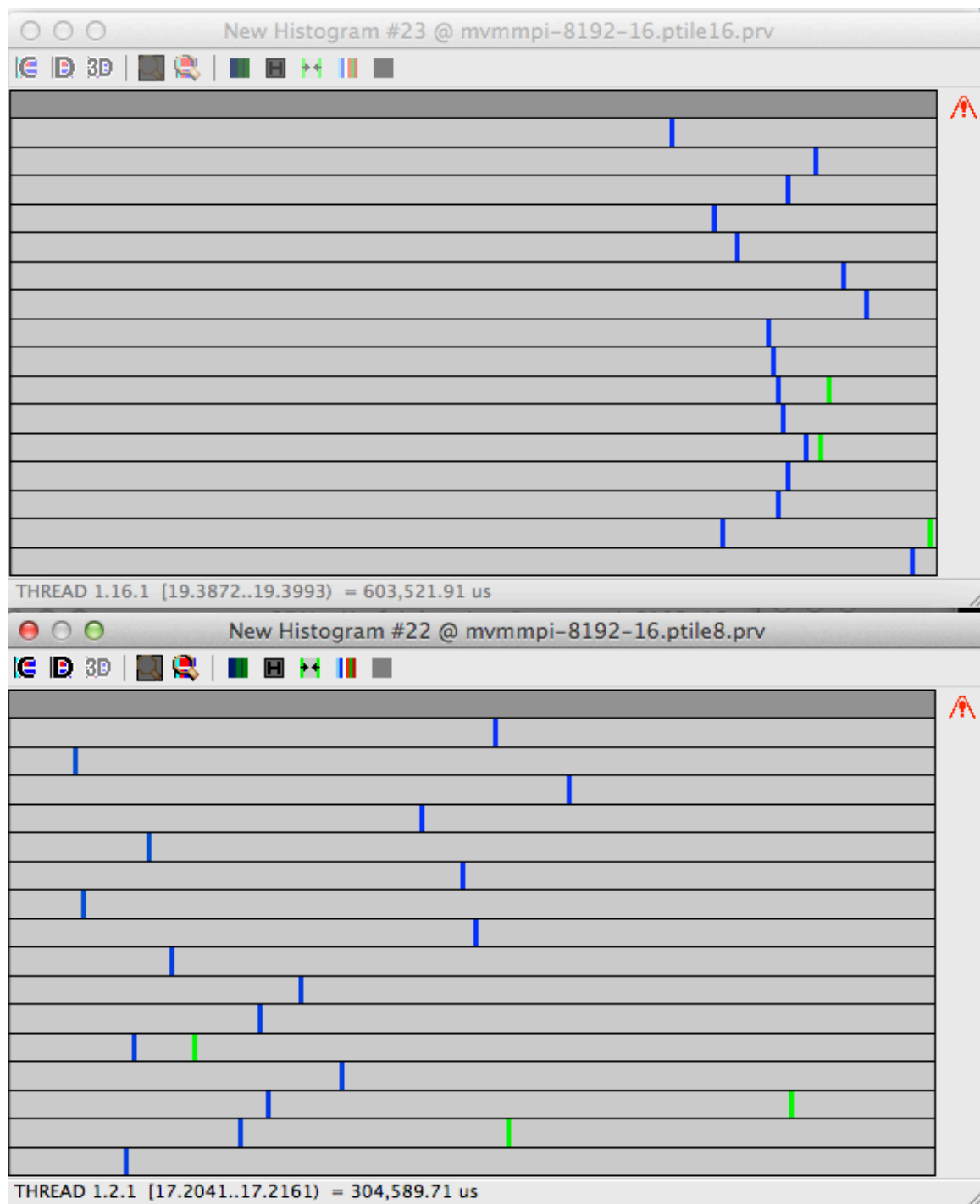
Exercise 7: Load *L3_miss_ratio.cfg* configuration file and generate two views .



As you can see it is difficult to see the difference. We suggest a 2D view, as shown in the previous hands-on for the desired region in the trace. Go to the main window and click on the New Histogram icon . Click on the magnifying glass icon at the top of the icons column on the right of the window to switch between numerical and zoomed out display.

It is clear that the blue binds represent the L3 misses of the matrix multiplication section (between two MPI events).

We can zoom and we can synchronize both windows in order to compare the two traces.



Using the mouse, we can select some of the point and see the numerical value. We can observe significant differences, e.g. process 15 in ptile=16 trace have 19,39 misses (per 1000 instructions) versus process 1 in ptile=8 trace that presents 17,20 misses (per 1000 instructions). We can see that L3 cache misses in pthread=16 trace are higher. Also we can see that in ptile=8 trace the duration time of processes are not equal, especially the first eight, that are executed in one of the servers (remember that with ptile=8 the maximum number of threads per server are 8). It seems that the default scheduling of the threads applied by the system are different in the two servers. We can use some flags such as

```
--map-by core --bind-to socket
```

to control and modify the default scheduling. Processor affinity, or CPU pinning enables the binding and unbinding of a process to a CPU or socket. However beyond the scope of this hands-on.

Exercise 8: Create the 2D views and analyze them. Comment on the results. Are they similar to the view shown in the views included in this section?

Exercise 9 (OPCIONAL): Do the same analysis with the L2 misses (the configuration file is included in the tar file).

2.5.6 Conclusions

In this project we have shown by example the need for analysis tools like Paraver. We can conclude that timing the process 0 with the gettimeofday is an approximation of the real behavior of our code.

3 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.