Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

Extrae Hands-On

✉ tools@bsc.es

07/2020

DEEP-EST Early Access Programme

Extrae features

- Platforms
  - Intel, Cray, BlueGene, MIC, ARM, Android, Fujitsu Sparc …
- Parallel programming models
  - MPI, OpenMP, pthreads, OmpSs, CUDA, OpenCL, Java, Python …
- Performance Counters
  - Using PAPI interface
- Link to source code
  - Callstack at MPI routines
  - OpenMP outlined routines
  - Selected user functions (Dyninst)
- Periodic sampling
- User events (Extrae API)

No need to recompile or relink!

How does Extrae work?

- Symbol substitution through LD_PRELOAD
  - Specific libraries for each combination of runtimes
    - MPI
    - OpenMP
    - OpenMP+MPI
    - …

Recommended

- Dynamic instrumentation
  - Based on Dyninst (developed by U.Wisconsin / U.Maryland)
    - Instrumentation in memory
    - Binary rewriting
- Alternatives
  - Static link (i.e., PMPI, Extrae API)

Extrae on DEEP-EST (I)

- Log-in to DEEP-EST:

```
laptop$ ssh –Y <USER>@deep-fz.juelich.de
```

- Extrae is available via modules for 4 toolchains… choose yours!

1. GCC compiler with ParaStation MPI

```
deep$ module use $OTHERSTAGES
deep$ ml Stages/Devel-2019a
deep$ ml GCC/8.3.0
deep$ ml ParaStationMPI/5.4.6-1
deep$ ml Extrae
```

2. Intel compiler with Intel MPI

```
deep$ module use $OTHERSTAGES
deep$ ml Stages/Devel-2019a
deep$ ml Intel/2019.3.199-GCC-8.3.0
deep$ ml IntelMPI/2018.5.288
deep$ ml Extrae
```

Extrae on DEEP-EST (II)

3. Intel compiler with ParaStationMPI

```
deep$ module use $OTHERSTAGES
deep$ ml Stages/Devel-2019a
deep$ ml Intel/2019.5.281-GCC-8.3.0
deep$ ml ParaStationMPI/5.4.6-1
deep$ ml Extrae
```

4. Intel compiler with ParaStationMPI (MPI multithreaded support)

```
deep$ module use $OTHERSTAGES
deep$ ml Stages/Devel-2019a
deep$ ml Intel/2019.5.281-GCC-8.3.0
deep$ ml ParaStationMPI/5.4.6-1-mt
deep$ ml Extrae
```

Getting your first trace

- Provided package **lulesh-example.tar.xz** contains:
  - README
  - Application (lulesh2.0)
  - Jobscripts to execute and trace (job.slurm, trace.sh)
  - Configuration of the tracing tool (extrae.xml)
  - Already generated trace (trace/*.{pcf,prv,row})

- Copy this package to DEEP-EST cluster and uncompress into **your own /work/<project>/<user> folder**

Using Extrae in 3 steps

1. **Adapt** your job submission scripts

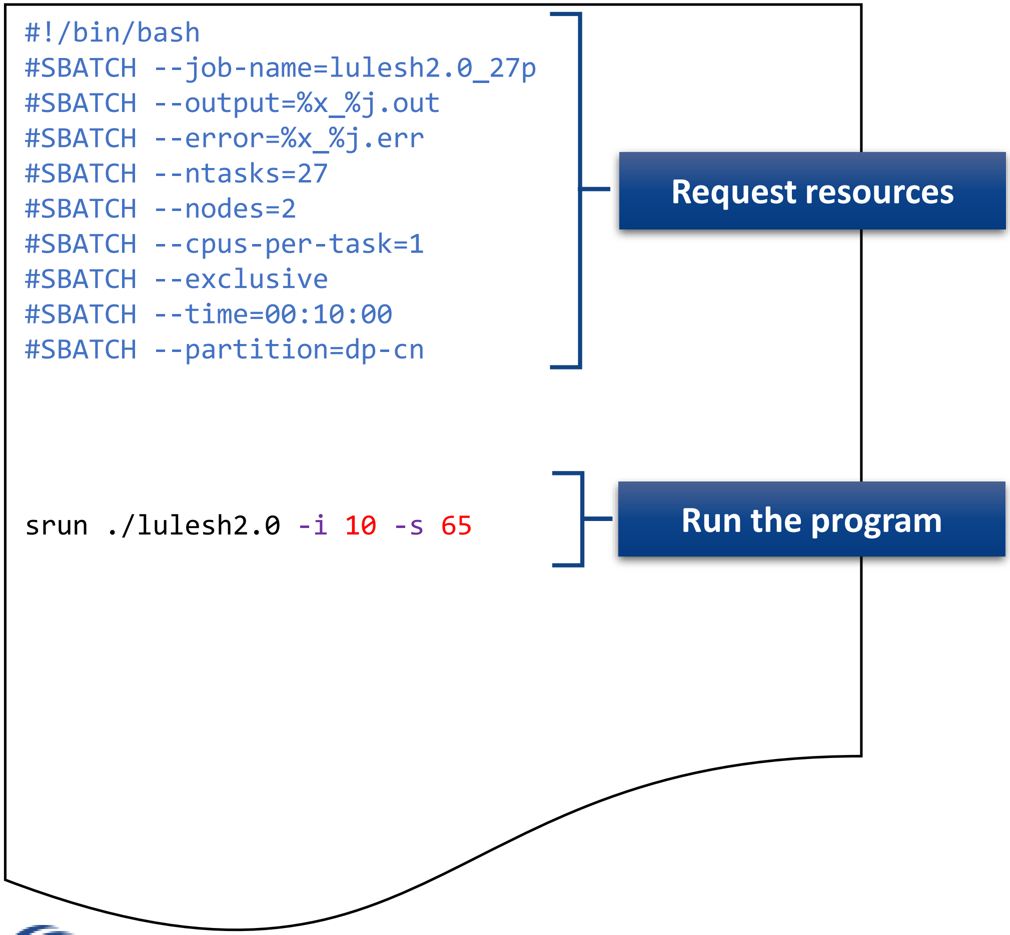
2. **Configure** what to trace
   - XML configuration file
   - Example configurations at `$EXTRAE_HOME/share/example`

3. **Run** it!

- For further reference check the **Extrae User Guide:**
  - https://tools.bsc.es/doc/html/extrae
  - Also distributed with Extrae at `$EXTRAE_HOME/share/doc`

Step 1: Adapt the job script to load Extrae

- Example of a standard jobscript (without tracing)



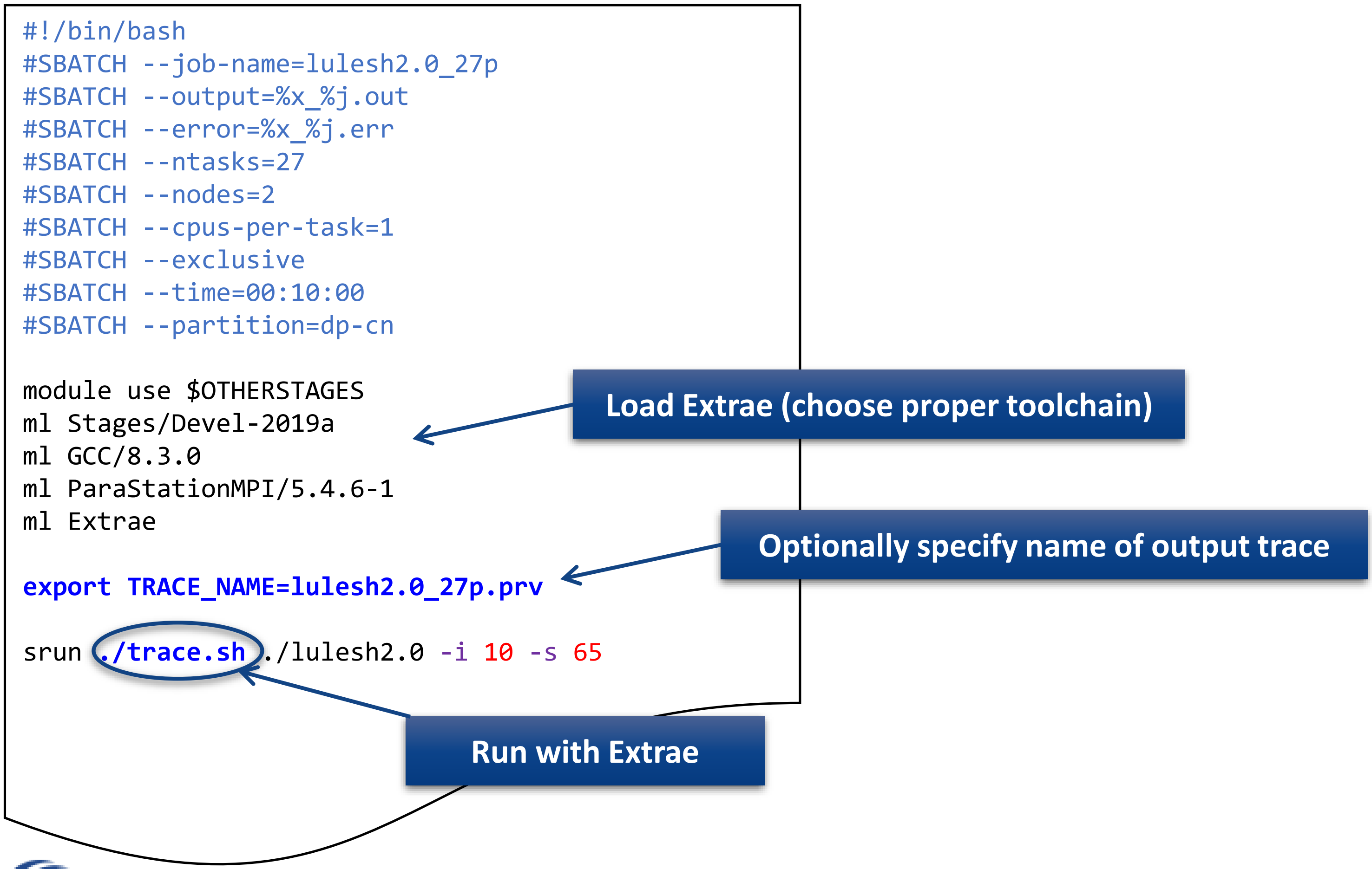
```
#!/bin/bash
#SBATCH --job-name=lulesh2.0_27p
#SBATCH --output=%x_%j.out
#SBATCH --error=%x_%j.err
#SBATCH --ntasks=27
#SBATCH --nodes=2
#SBATCH --cpus-per-task=1
#SBATCH --exclusive
#SBATCH --time=00:10:00
#SBATCH --partition=dp-cn



srun ./lulesh2.0 -i 10 -s 65
```

Request resources

Run the program

Step 1: Adapt the job script to load Extrae

- Jobscript modified to load Extrae



```
#!/bin/bash
#SBATCH --job-name=lulesh2.0_27p
#SBATCH --output=%x_%j.out
#SBATCH --error=%x_%j.err
#SBATCH --ntasks=27
#SBATCH --nodes=2
#SBATCH --cpus-per-task=1
#SBATCH --exclusive
#SBATCH --time=00:10:00
#SBATCH --partition=dp-cn

module use $OTHERSTAGES
ml Stages/Devel-2019a
ml GCC/8.3.0
ml ParaStationMPI/5.4.6-1
ml Extrae

export TRACE_NAME=lulesh2.0_27p.prv

srun ./trace.sh ./lulesh2.0 -i 10 -s 65
```

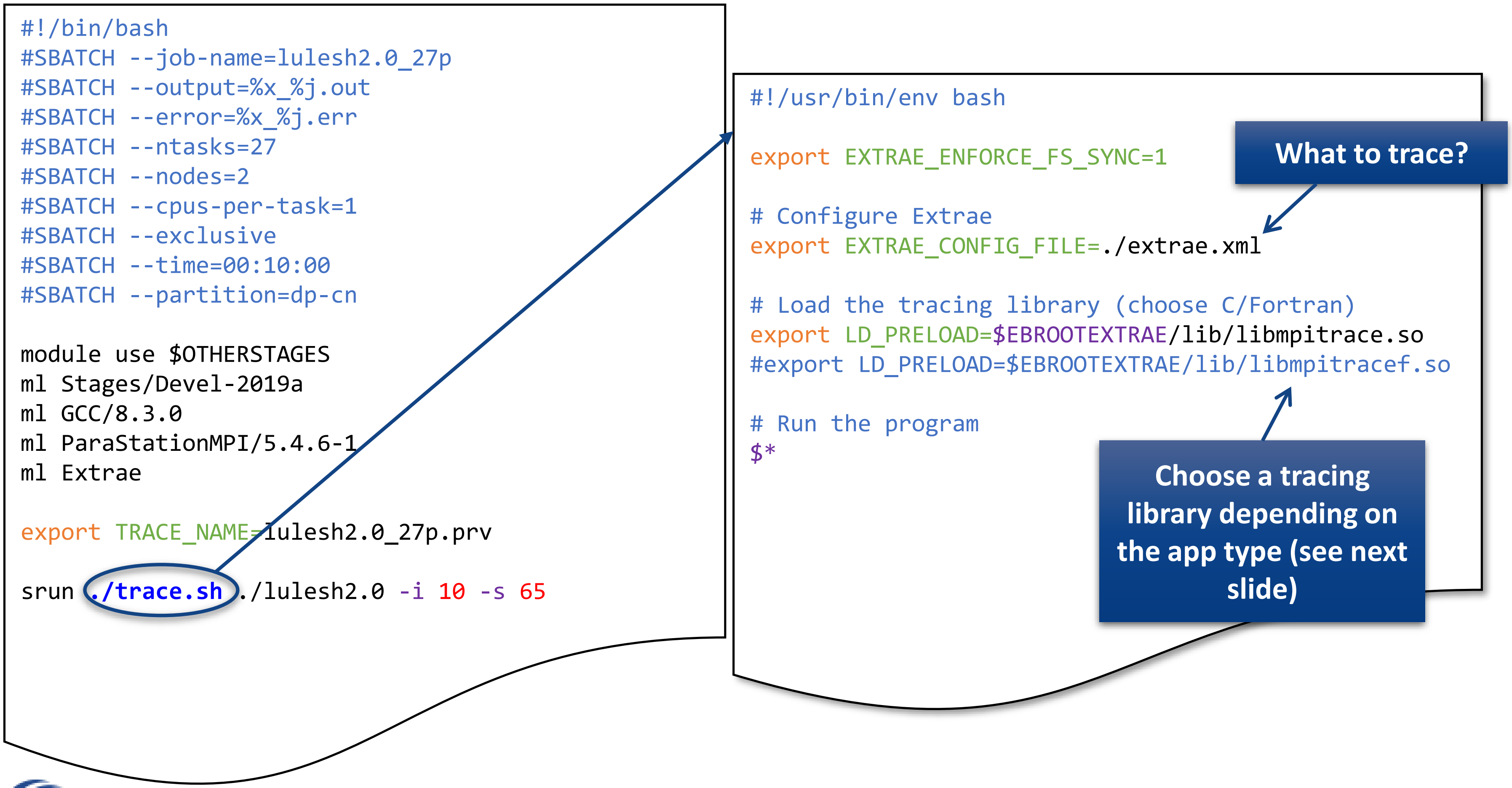
Load Extrae (choose proper toolchain)

Optionally specify name of output trace

Run with Extrae

Step 1: Adapt the job script to load Extrae

- Tracing launcher helper script (trace.sh)



```
#!/bin/bash
#SBATCH --job-name=lulesh2.0_27p
#SBATCH --output=%x_%j.out
#SBATCH --error=%x_%j.err
#SBATCH --ntasks=27
#SBATCH --nodes=2
#SBATCH --cpus-per-task=1
#SBATCH --exclusive
#SBATCH --time=00:10:00
#SBATCH --partition=dp-cn

module use $OTHERSTAGES
ml Stages/Devel-2019a
ml GCC/8.3.0
ml ParaStationMPI/5.4.6-1
ml Extrae

export TRACE_NAME=lulesh2.0_27p.prv

srun ./trace.sh ./lulesh2.0 -i 10 -s 65
```

```
#!/usr/bin/env bash

export EXTRAE_ENFORCE_FS_SYNC=1

# Configure Extrae
export EXTRAE_CONFIG_FILE=./extrae.xml

# Load the tracing library (choose C/Fortran)
export LD_PRELOAD=$EBROOTEXTRAE/lib/libmpitrace.so
#export LD_PRELOAD=$EBROOTEXTRAE/lib/libmpitracef.so

# Run the program
$*
```

What to trace?

Choose a tracing library depending on the app type (see next slide)

Step 1: Which tracing library?

- Choose depending on the application type

Library	Serial	MPI	OpenMP	pthread	CUDA
libseqtrace	✓				
libmpitrace[f][1]		✓			
libomptrace			✓		
libpttrace				✓	
libcudatrace					✓
libompitrace[f] [1]		✓	✓		
libptmpitrace[f] [1]		✓		✓	
libcudampitrace[f] [1]		✓			✓

[1] include suffix “f” in Fortran codes

Step 2: Extrae XML configuration

```
deep$ vi extrae.xml
```

```
<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>
```

Trace the MPI calls
(What's the program doing?)

Trace the call-stack
(Where in my code?)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Step 2: Extrae XML configuration (II)

```
deep$ vi extrae.xml
```

```
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution=“1">
    <set enabled="yes" domain="all" changeat-time="500000us">
      PAPI_TOT_INS,PAPI_TOT_CYC
    </set>
  </cpu>
  <network enabled="no" />
  <resource-usage enabled="no" />
  <memory-usage enabled="no" />
</counters>
```

Select which HW counters are measured

(How’s the machine doing?)

Step 2: Extrae XML configuration (III)

```
deep$ vi extrae.xml
```

```
<buffer enabled="yes">
  <size enabled="yes">5000000</size>
  <circular enabled="no" />
</buffer>

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

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

Trace buffer size
(Flush/memory trade-off)

Additional sampling
(Want more details?)

Automatic post-processing to generate the Paraver trace

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Step 3: Run it!

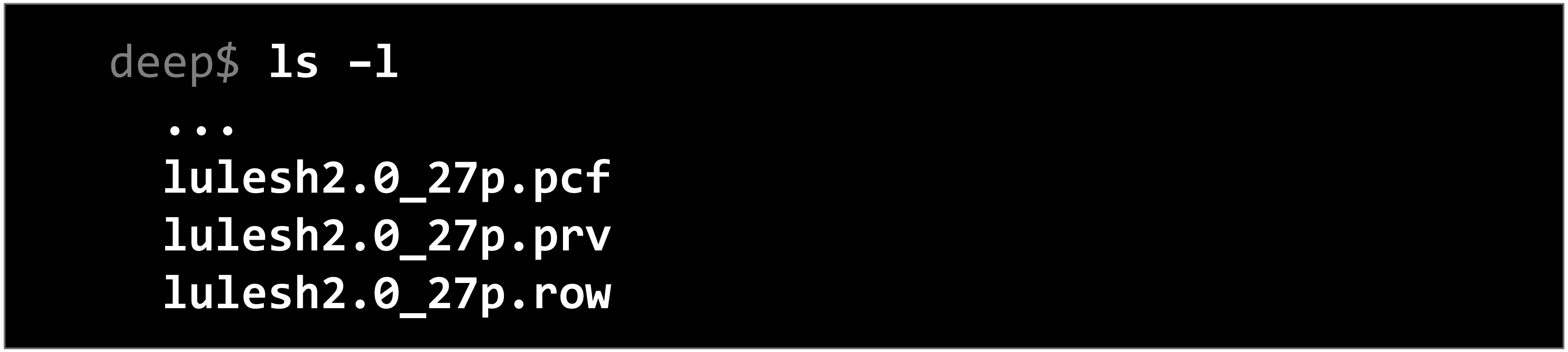
- Submit your job as usual

```
deep$ sbatch job.slurm
```

- **REMEMBER!** Run job from your /work folder (NOT IN HOME!)

All done! Check your resulting trace

- Once finished (check with “squeue”) you will have the trace (3 files):



```
deep$ ls -l
   ...
   lulesh2.0_27p.pcf
   lulesh2.0_27p.prv
   lulesh2.0_27p.row
```

- Any trouble? There’s a trace already generated under the “trace” folder

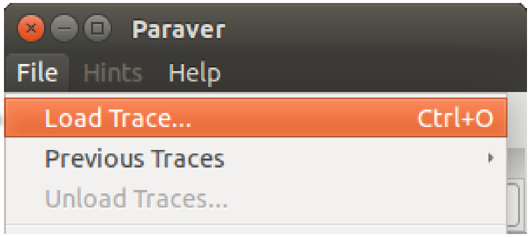
- Now let’s look into it !

First steps of analysis

- Copy the trace to your computer

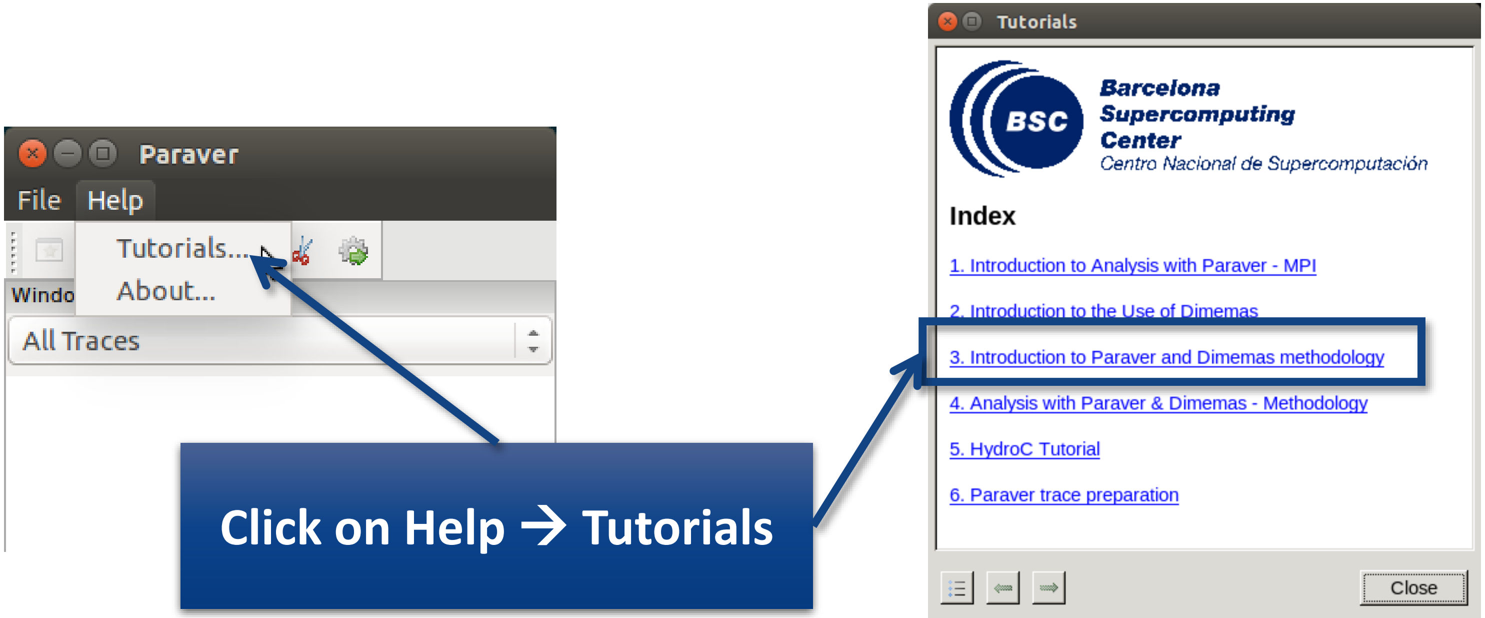
- Load the trace with Paraver

Click on File → Load Trace → Browse to “lulesh2.0_27p.prv”



First steps of analysis

- Follow Tutorial #3
  - Introduction to Paraver and Dimemas methodology



Click on Help → Tutorials

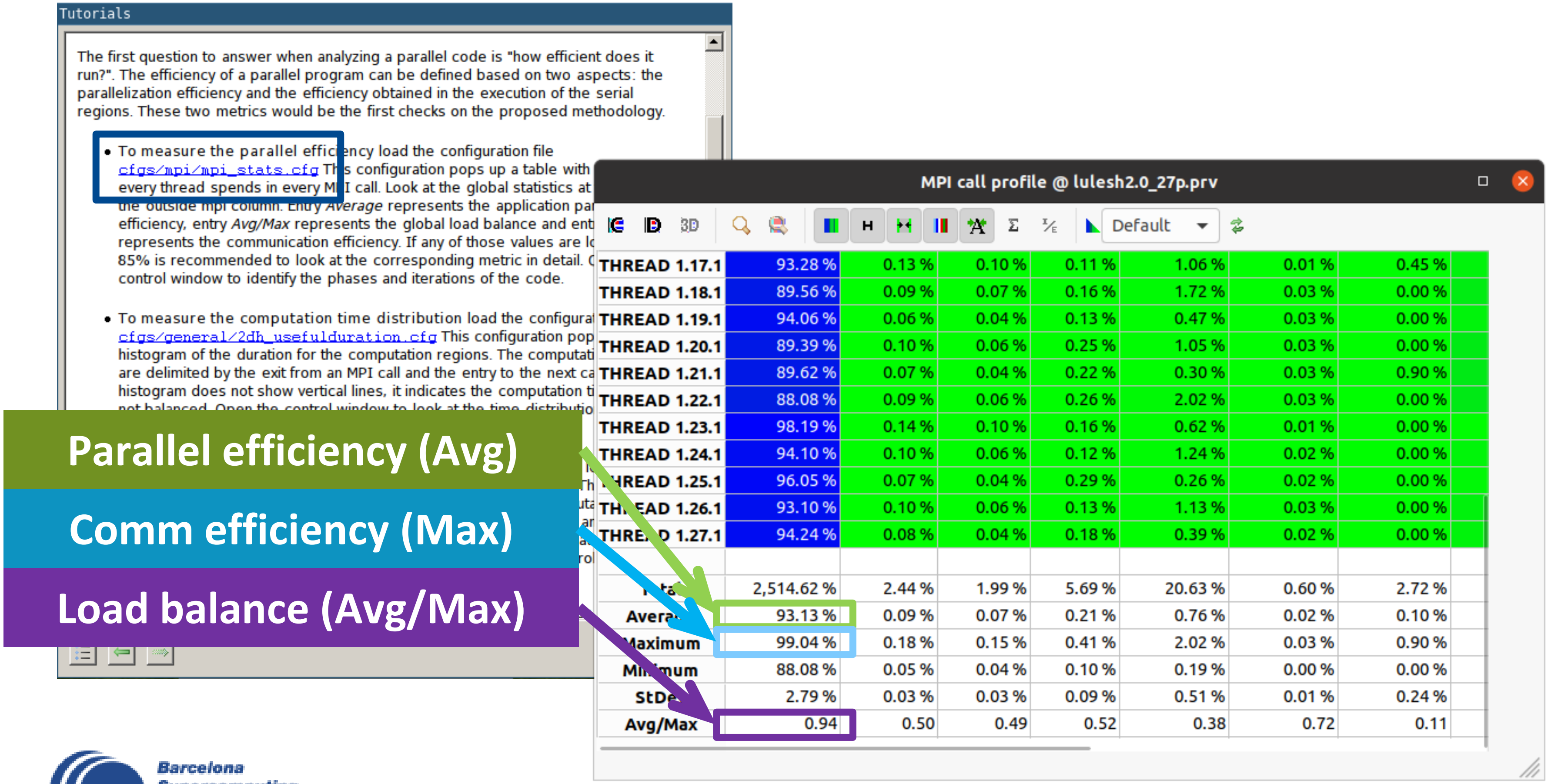
Paraver
File
Help
Tutorials...
About...
Windo
All Traces
Tutorials
Barcelona Supercomputing Center
Centro Nacional de Supercomputación
Close

Index

1. Introduction to Analysis with Paraver - MPI
2. Introduction to the Use of Dimemas
3. Introduction to Paraver and Dimemas methodology
4. Analysis with Paraver & Dimemas - Methodology
5. HydroC Tutorial
6. Paraver trace preparation

Measure the parallel efficiency

- Click on “mpi_stats.cfg”
  - Check the **Average** for the column labeled “**Outside MPI**”



The first question to answer when analyzing a parallel code is "how efficient does it run?". The efficiency of a parallel program can be defined based on two aspects: the parallelization efficiency and the efficiency obtained in the execution of the serial regions. These two metrics would be the first checks on the proposed methodology.

- To measure the parallel efficiency load the configuration file cfgs/mpi/mpi_stats.cfg This configuration pops up a table with every thread spends in every MPI call. Look at the global statistics at the outside mpi column. Entry *Average* represents the application par efficiency, entry *Avg/Max* represents the global load balance and ent represents the communication efficiency. If any of those values are lo 85% is recommended to look at the corresponding metric in detail. O control window to identify the phases and iterations of the code.
- To measure the computation time distribution load the configurat cfgs/general/2dh_usefulduration.cfg This configuration pop histogram of the duration for the computation regions. The computati are delimited by the exit from an MPI call and the entry to the next ca histogram does not show vertical lines, it indicates the computation ti not balanced. Open the control window to look at the time distributio

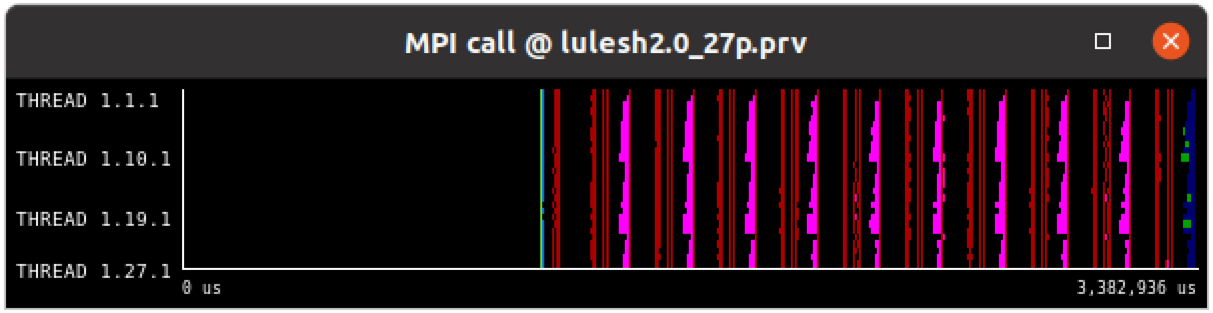
THREAD 1.17.1	93.28 %	0.13 %	0.10 %	0.11 %	1.06 %	0.01 %	0.45 %
THREAD 1.18.1	89.56 %	0.09 %	0.07 %	0.16 %	1.72 %	0.03 %	0.00 %
THREAD 1.19.1	94.06 %	0.06 %	0.04 %	0.13 %	0.47 %	0.03 %	0.00 %
THREAD 1.20.1	89.39 %	0.10 %	0.06 %	0.25 %	1.05 %	0.03 %	0.00 %
THREAD 1.21.1	89.62 %	0.07 %	0.04 %	0.22 %	0.30 %	0.03 %	0.90 %
THREAD 1.22.1	88.08 %	0.09 %	0.06 %	0.26 %	2.02 %	0.03 %	0.00 %
THREAD 1.23.1	98.19 %	0.14 %	0.10 %	0.16 %	0.62 %	0.01 %	0.00 %
THREAD 1.24.1	94.10 %	0.10 %	0.06 %	0.12 %	1.24 %	0.02 %	0.00 %
THREAD 1.25.1	96.05 %	0.07 %	0.04 %	0.29 %	0.26 %	0.02 %	0.00 %
THREAD 1.26.1	93.10 %	0.10 %	0.06 %	0.13 %	1.13 %	0.03 %	0.00 %
THREAD 1.27.1	94.24 %	0.08 %	0.04 %	0.18 %	0.39 %	0.02 %	0.00 %
Total	2,514.62 %	2.44 %	1.99 %	5.69 %	20.63 %	0.60 %	2.72 %
Average	93.13 %	0.09 %	0.07 %	0.21 %	0.76 %	0.02 %	0.10 %
Maximum	99.04 %	0.18 %	0.15 %	0.41 %	2.02 %	0.03 %	0.90 %
Minimum	88.08 %	0.05 %	0.04 %	0.10 %	0.19 %	0.00 %	0.00 %
StDev	2.79 %	0.03 %	0.03 %	0.09 %	0.51 %	0.01 %	0.24 %
Avg/Max	0.94	0.50	0.49	0.52	0.38	0.72	0.11

Parallel efficiency (Avg)

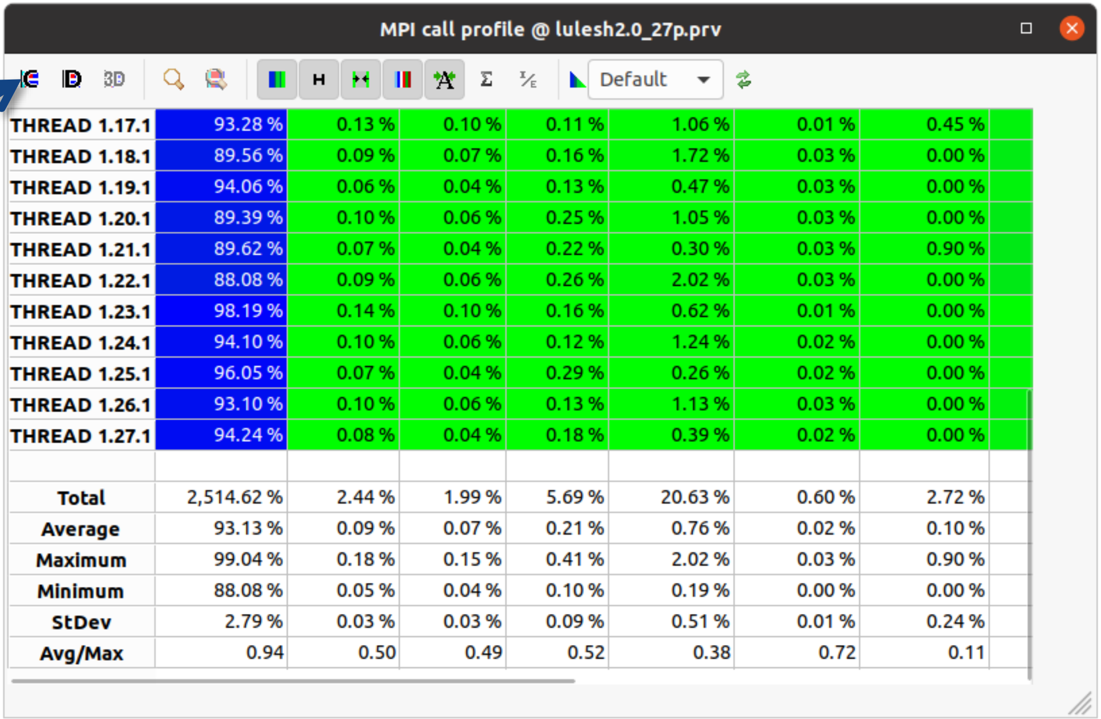
Comm efficiency (Max)

Load balance (Avg/Max)

Focus on the iterative part

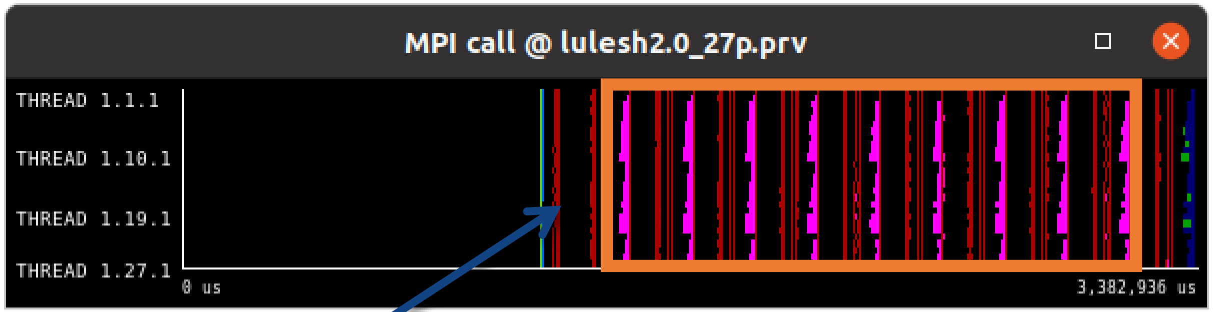

MPI call @ lulesh2.0_27p.prv

Click on Open Control Window

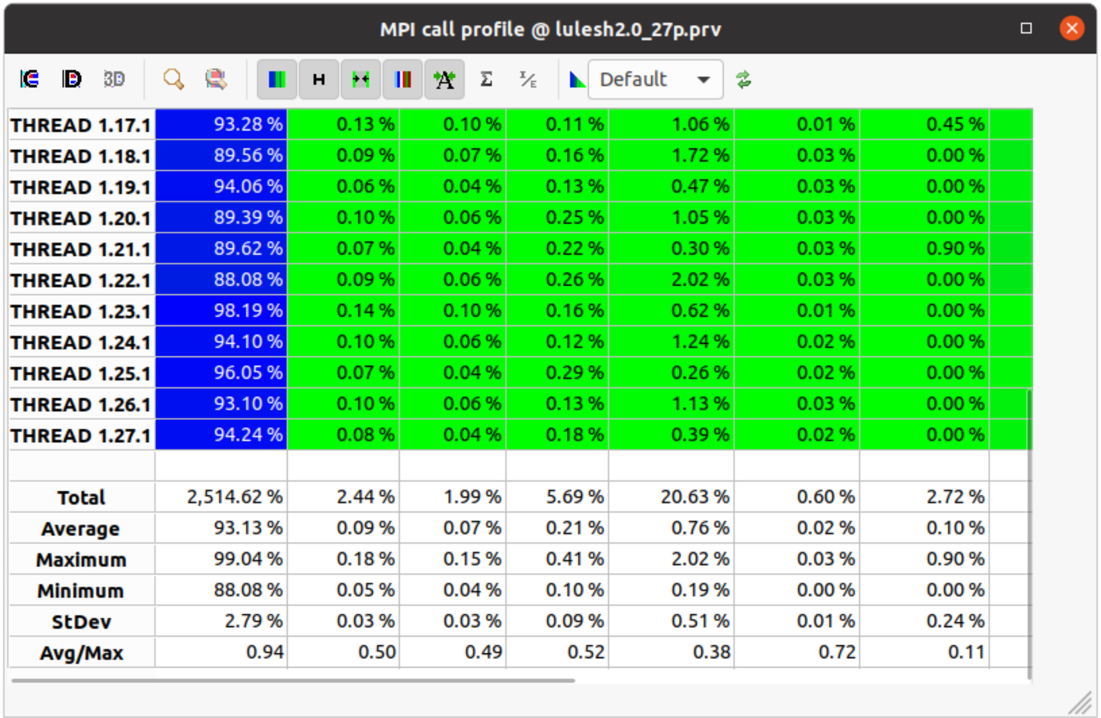

MPI call profile @ lulesh2.0_27p.prv

THREAD 1.17.1	93.28 %	0.13 %	0.10 %	0.11 %	1.06 %	0.01 %	0.45 %
THREAD 1.18.1	89.56 %	0.09 %	0.07 %	0.16 %	1.72 %	0.03 %	0.00 %
THREAD 1.19.1	94.06 %	0.06 %	0.04 %	0.13 %	0.47 %	0.03 %	0.00 %
THREAD 1.20.1	89.39 %	0.10 %	0.06 %	0.25 %	1.05 %	0.03 %	0.00 %
THREAD 1.21.1	89.62 %	0.07 %	0.04 %	0.22 %	0.30 %	0.03 %	0.90 %
THREAD 1.22.1	88.08 %	0.09 %	0.06 %	0.26 %	2.02 %	0.03 %	0.00 %
THREAD 1.23.1	98.19 %	0.14 %	0.10 %	0.16 %	0.62 %	0.01 %	0.00 %
THREAD 1.24.1	94.10 %	0.10 %	0.06 %	0.12 %	1.24 %	0.02 %	0.00 %
THREAD 1.25.1	96.05 %	0.07 %	0.04 %	0.29 %	0.26 %	0.02 %	0.00 %
THREAD 1.26.1	93.10 %	0.10 %	0.06 %	0.13 %	1.13 %	0.03 %	0.00 %
THREAD 1.27.1	94.24 %	0.08 %	0.04 %	0.18 %	0.39 %	0.02 %	0.00 %
Total	2,514.62 %	2.44 %	1.99 %	5.69 %	20.63 %	0.60 %	2.72 %
Average	93.13 %	0.09 %	0.07 %	0.21 %	0.76 %	0.02 %	0.10 %
Maximum	99.04 %	0.18 %	0.15 %	0.41 %	2.02 %	0.03 %	0.90 %
Minimum	88.08 %	0.05 %	0.04 %	0.10 %	0.19 %	0.00 %	0.00 %
StDev	2.79 %	0.03 %	0.03 %	0.09 %	0.51 %	0.01 %	0.24 %
Avg/Max	0.94	0.50	0.49	0.52	0.38	0.72	0.11

Focus on the iterative part



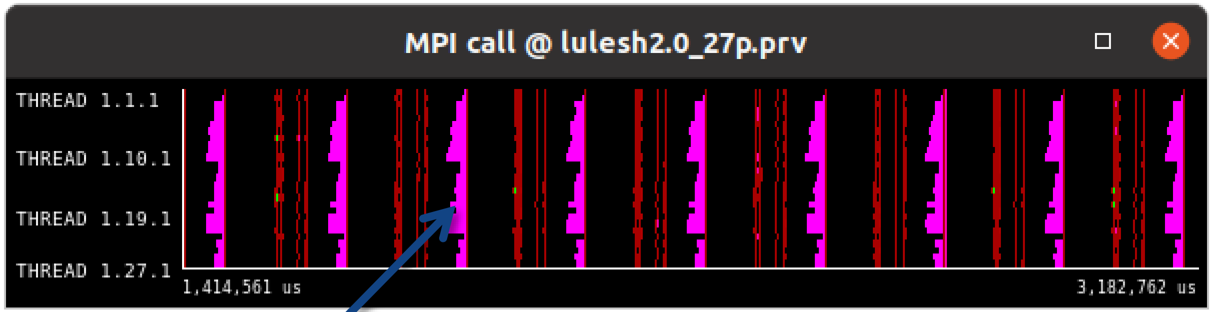
Drag & drop on this area to zoom on the iterative region



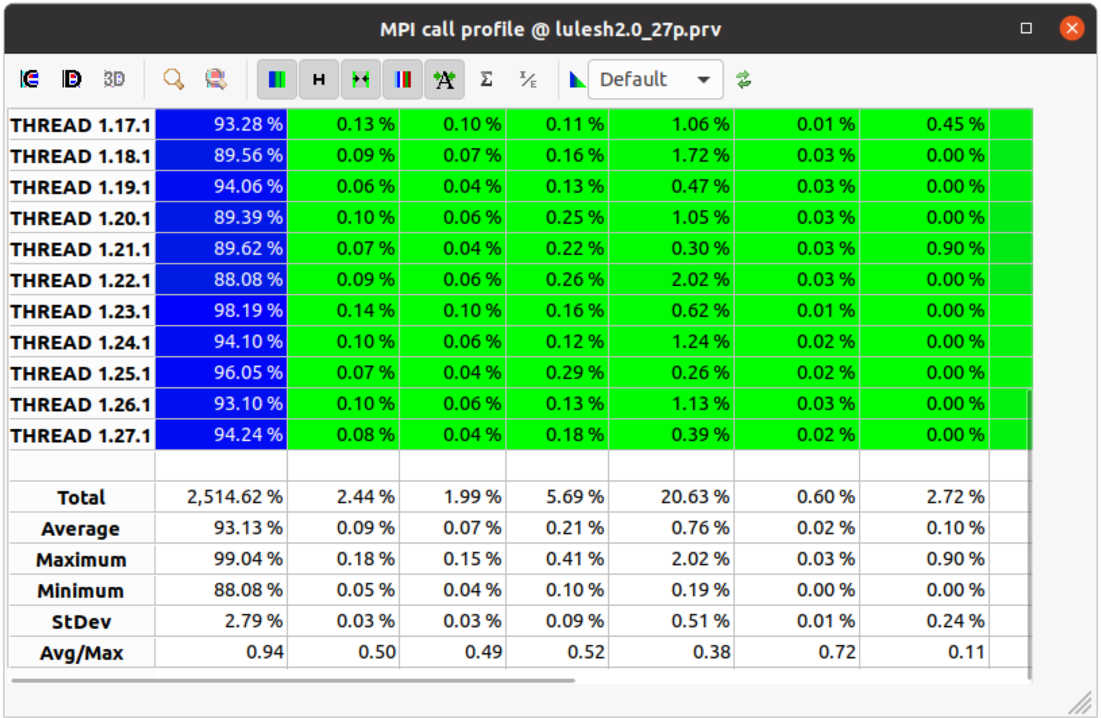
THREAD 1.17.1	93.28 %	0.13 %	0.10 %	0.11 %	1.06 %	0.01 %	0.45 %
THREAD 1.18.1	89.56 %	0.09 %	0.07 %	0.16 %	1.72 %	0.03 %	0.00 %
THREAD 1.19.1	94.06 %	0.06 %	0.04 %	0.13 %	0.47 %	0.03 %	0.00 %
THREAD 1.20.1	89.39 %	0.10 %	0.06 %	0.25 %	1.05 %	0.03 %	0.00 %
THREAD 1.21.1	89.62 %	0.07 %	0.04 %	0.22 %	0.30 %	0.03 %	0.90 %
THREAD 1.22.1	88.08 %	0.09 %	0.06 %	0.26 %	2.02 %	0.03 %	0.00 %
THREAD 1.23.1	98.19 %	0.14 %	0.10 %	0.16 %	0.62 %	0.01 %	0.00 %
THREAD 1.24.1	94.10 %	0.10 %	0.06 %	0.12 %	1.24 %	0.02 %	0.00 %
THREAD 1.25.1	96.05 %	0.07 %	0.04 %	0.29 %	0.26 %	0.02 %	0.00 %
THREAD 1.26.1	93.10 %	0.10 %	0.06 %	0.13 %	1.13 %	0.03 %	0.00 %
THREAD 1.27.1	94.24 %	0.08 %	0.04 %	0.18 %	0.39 %	0.02 %	0.00 %
Total	2,514.62 %	2.44 %	1.99 %	5.69 %	20.63 %	0.60 %	2.72 %
Average	93.13 %	0.09 %	0.07 %	0.21 %	0.76 %	0.02 %	0.10 %
Maximum	99.04 %	0.18 %	0.15 %	0.41 %	2.02 %	0.03 %	0.90 %
Minimum	88.08 %	0.05 %	0.04 %	0.10 %	0.19 %	0.00 %	0.00 %
StDev	2.79 %	0.03 %	0.03 %	0.09 %	0.51 %	0.01 %	0.24 %
Avg/Max	0.94	0.50	0.49	0.52	0.38	0.72	0.11

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Recalculate efficiency of iterative region

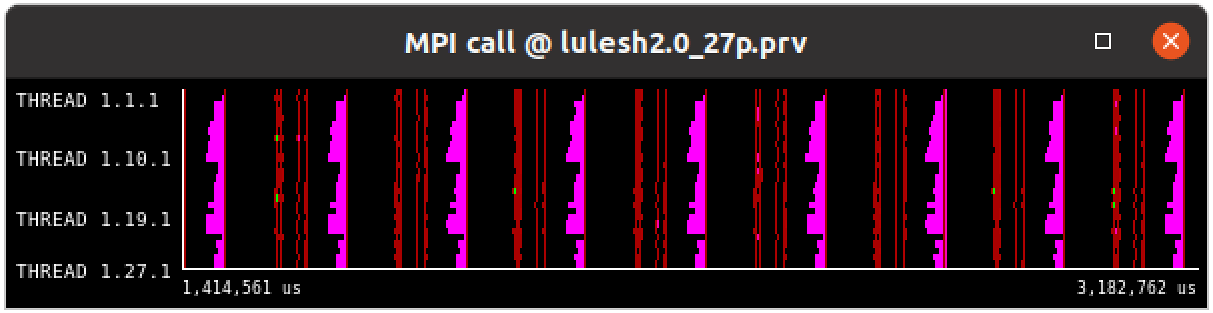


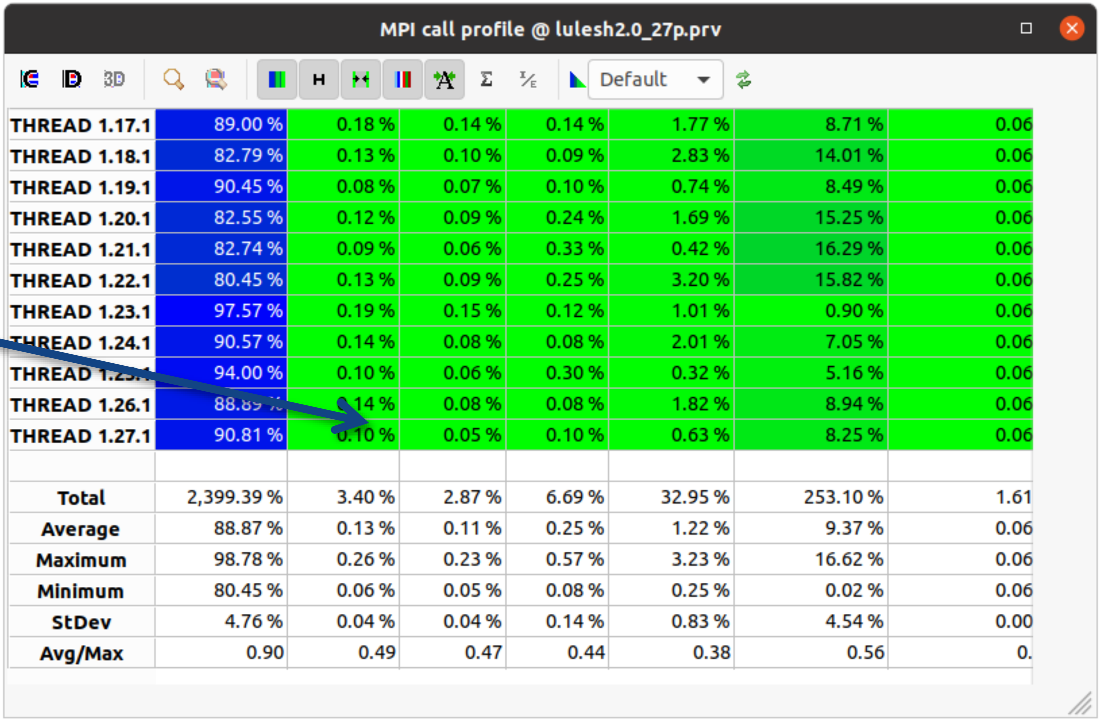
Right click → Copy



THREAD 1.17.1	93.28 %	0.13 %	0.10 %	0.11 %	1.06 %	0.01 %	0.45 %
THREAD 1.18.1	89.56 %	0.09 %	0.07 %	0.16 %	1.72 %	0.03 %	0.00 %
THREAD 1.19.1	94.06 %	0.06 %	0.04 %	0.13 %	0.47 %	0.03 %	0.00 %
THREAD 1.20.1	89.39 %	0.10 %	0.06 %	0.25 %	1.05 %	0.03 %	0.00 %
THREAD 1.21.1	89.62 %	0.07 %	0.04 %	0.22 %	0.30 %	0.03 %	0.90 %
THREAD 1.22.1	88.08 %	0.09 %	0.06 %	0.26 %	2.02 %	0.03 %	0.00 %
THREAD 1.23.1	98.19 %	0.14 %	0.10 %	0.16 %	0.62 %	0.01 %	0.00 %
THREAD 1.24.1	94.10 %	0.10 %	0.06 %	0.12 %	1.24 %	0.02 %	0.00 %
THREAD 1.25.1	96.05 %	0.07 %	0.04 %	0.29 %	0.26 %	0.02 %	0.00 %
THREAD 1.26.1	93.10 %	0.10 %	0.06 %	0.13 %	1.13 %	0.03 %	0.00 %
THREAD 1.27.1	94.24 %	0.08 %	0.04 %	0.18 %	0.39 %	0.02 %	0.00 %
Total	2,514.62 %	2.44 %	1.99 %	5.69 %	20.63 %	0.60 %	2.72 %
Average	93.13 %	0.09 %	0.07 %	0.21 %	0.76 %	0.02 %	0.10 %
Maximum	99.04 %	0.18 %	0.15 %	0.41 %	2.02 %	0.03 %	0.90 %
Minimum	88.08 %	0.05 %	0.04 %	0.10 %	0.19 %	0.00 %	0.00 %
StDev	2.79 %	0.03 %	0.03 %	0.09 %	0.51 %	0.01 %	0.24 %
Avg/Max	0.94	0.50	0.49	0.52	0.38	0.72	0.11

Recalculate efficiency of iterative region



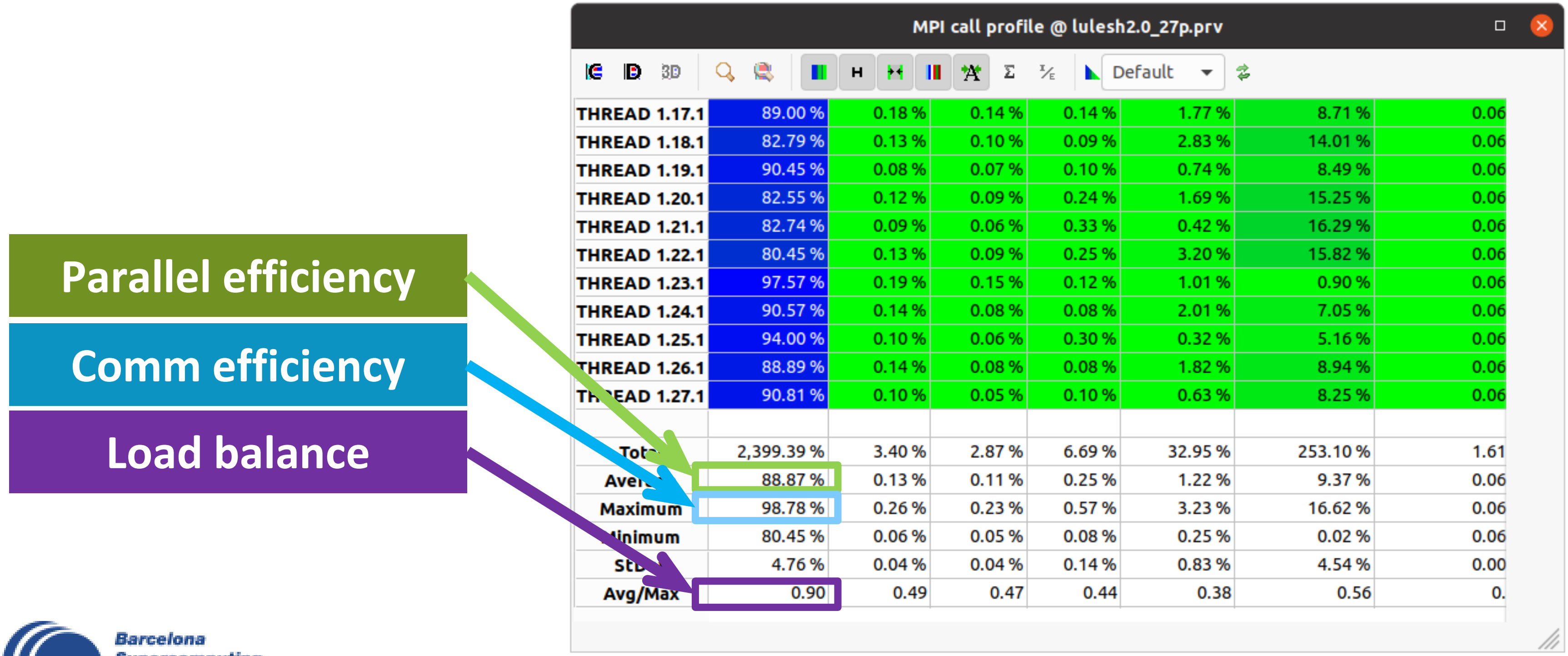


THREAD 1.17.1	89.00 %	0.18 %	0.14 %	0.14 %	1.77 %	8.71 %	0.06
THREAD 1.18.1	82.79 %	0.13 %	0.10 %	0.09 %	2.83 %	14.01 %	0.06
THREAD 1.19.1	90.45 %	0.08 %	0.07 %	0.10 %	0.74 %	8.49 %	0.06
THREAD 1.20.1	82.55 %	0.12 %	0.09 %	0.24 %	1.69 %	15.25 %	0.06
THREAD 1.21.1	82.74 %	0.09 %	0.06 %	0.33 %	0.42 %	16.29 %	0.06
THREAD 1.22.1	80.45 %	0.13 %	0.09 %	0.25 %	3.20 %	15.82 %	0.06
THREAD 1.23.1	97.57 %	0.19 %	0.15 %	0.12 %	1.01 %	0.90 %	0.06
THREAD 1.24.1	90.57 %	0.14 %	0.08 %	0.08 %	2.01 %	7.05 %	0.06
THREAD 1.25.1	94.00 %	0.10 %	0.06 %	0.30 %	0.32 %	5.16 %	0.06
THREAD 1.26.1	88.89 %	0.14 %	0.08 %	0.08 %	1.82 %	8.94 %	0.06
THREAD 1.27.1	90.81 %	0.10 %	0.05 %	0.10 %	0.63 %	8.25 %	0.06
Total	2,399.39 %	3.40 %	2.87 %	6.69 %	32.95 %	253.10 %	1.61
Average	88.87 %	0.13 %	0.11 %	0.25 %	1.22 %	9.37 %	0.06
Maximum	98.78 %	0.26 %	0.23 %	0.57 %	3.23 %	16.62 %	0.06
Minimum	80.45 %	0.06 %	0.05 %	0.08 %	0.25 %	0.02 %	0.06
StDev	4.76 %	0.04 %	0.04 %	0.14 %	0.83 %	4.54 %	0.00
Avg/Max	0.90	0.49	0.47	0.44	0.38	0.56	0.

Right click → Paste → Time

Efficiency of iterative region

- 3 numbers to quickly describe the efficiency of your code
  - Parallel efficiency → % of time my program is computing (100% is perfect)
  - Comm efficiency → At least 1 process can finish all communications in 100 - Maximum % of the program's time (100% is perfect)
  - Load balance → Ratio of slow/fast processes (1 is perfectly balanced)
  - Any value below 85% (0.85)? Pay attention there…



Parallel efficiency

Comm efficiency

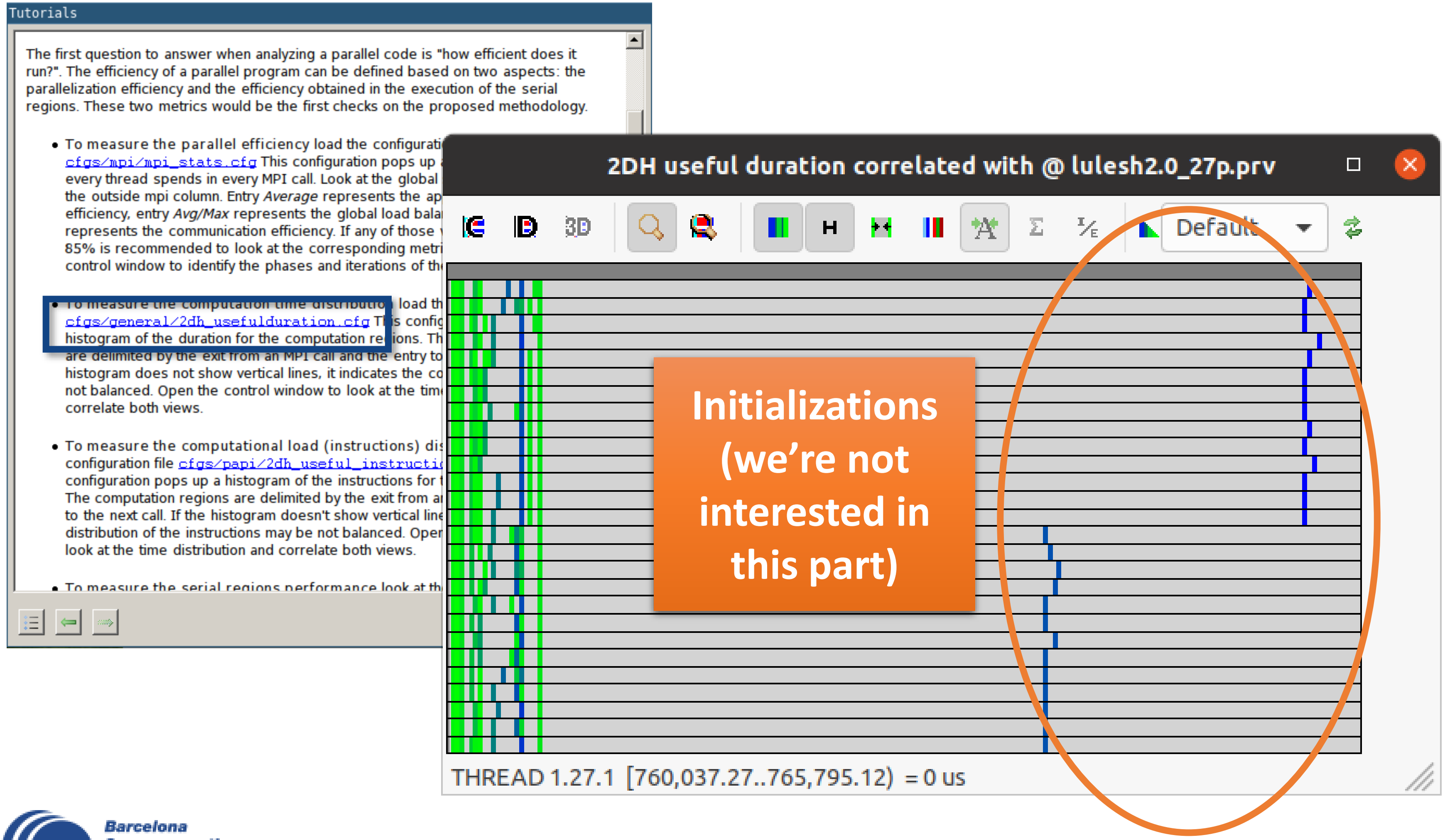
Load balance

THREAD 1.17.1	89.00 %	0.18 %	0.14 %	0.14 %	1.77 %	8.71 %	0.06
THREAD 1.18.1	82.79 %	0.13 %	0.10 %	0.09 %	2.83 %	14.01 %	0.06
THREAD 1.19.1	90.45 %	0.08 %	0.07 %	0.10 %	0.74 %	8.49 %	0.06
THREAD 1.20.1	82.55 %	0.12 %	0.09 %	0.24 %	1.69 %	15.25 %	0.06
THREAD 1.21.1	82.74 %	0.09 %	0.06 %	0.33 %	0.42 %	16.29 %	0.06
THREAD 1.22.1	80.45 %	0.13 %	0.09 %	0.25 %	3.20 %	15.82 %	0.06
THREAD 1.23.1	97.57 %	0.19 %	0.15 %	0.12 %	1.01 %	0.90 %	0.06
THREAD 1.24.1	90.57 %	0.14 %	0.08 %	0.08 %	2.01 %	7.05 %	0.06
THREAD 1.25.1	94.00 %	0.10 %	0.06 %	0.30 %	0.32 %	5.16 %	0.06
THREAD 1.26.1	88.89 %	0.14 %	0.08 %	0.08 %	1.82 %	8.94 %	0.06
THREAD 1.27.1	90.81 %	0.10 %	0.05 %	0.10 %	0.63 %	8.25 %	0.06
Tot	2,399.39 %	3.40 %	2.87 %	6.69 %	32.95 %	253.10 %	1.61
Aver	88.87 %	0.13 %	0.11 %	0.25 %	1.22 %	9.37 %	0.06
Maximum	98.78 %	0.26 %	0.23 %	0.57 %	3.23 %	16.62 %	0.06
Minimum	80.45 %	0.06 %	0.05 %	0.08 %	0.25 %	0.02 %	0.06
StD	4.76 %	0.04 %	0.04 %	0.14 %	0.83 %	4.54 %	0.00
Avg/Max	0.90	0.49	0.47	0.44	0.38	0.56	0.

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Computation time distribution

- Click on “2dh_usefulduration.cfg” (2nd link) → Shows **time computing**



Tutorials

The first question to answer when analyzing a parallel code is "how efficient does it run?". The efficiency of a parallel program can be defined based on two aspects: the parallelization efficiency and the efficiency obtained in the execution of the serial regions. These two metrics would be the first checks on the proposed methodology.

- To measure the parallel efficiency load the configurati
  cfgs/mpi/mpi_stats.cfg This configuration pops up
  every thread spends in every MPI call. Look at the global
  the outside mpi column. Entry *Average* represents the ap
  efficiency, entry *Avg/Max* represents the global load bala
  represents the communication efficiency. If any of those
  85% is recommended to look at the corresponding metri
  control window to identify the phases and iterations of th

- To measure the computation time distributio load th
  cfgs/general/2dh_usefulduration.cfg This config
  histogram of the duration for the computation regions. Th
  are delimited by the exit from an MPI call and the entry to
  histogram does not show vertical lines, it indicates the co
  not balanced. Open the control window to look at the time
  correlate both views.

- To measure the computational load (instructions) dis
  configuration file cfgs/papi/2dh_useful_instructio
  configuration pops up a histogram of the instructions for
  The computation regions are delimited by the exit from a
  to the next call. If the histogram doesn't show vertical line
  distribution of the instructions may be not balanced. Oper
  look at the time distribution and correlate both views.

- To measure the serial regions performance look at th

2DH useful duration correlated with @ lulesh2.0_27p.prv

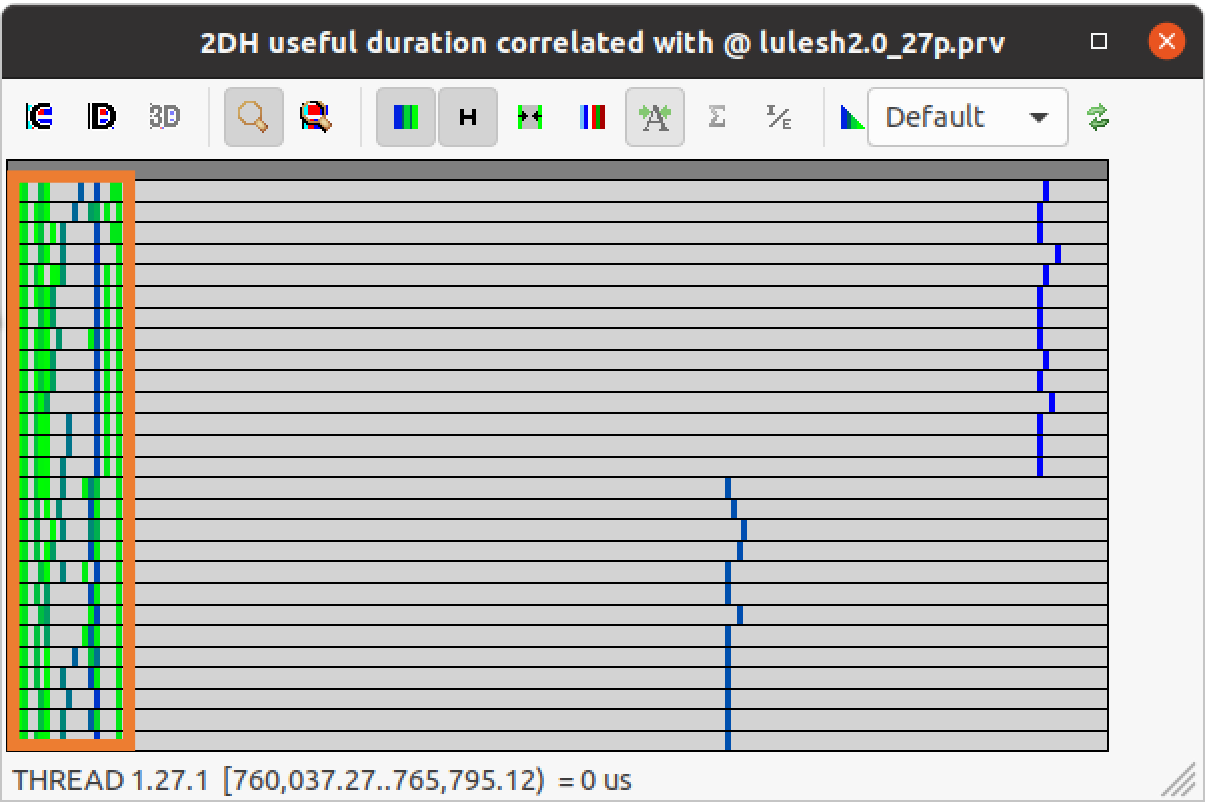
THREAD 1.27.1 [760,037.27..765,795.12) = 0 us

Initializations (we’re not interested in this part)

Focus on the iterative part

- Click on “2dh_usefulduration.cfg” (2nd link) → Shows **time computing**

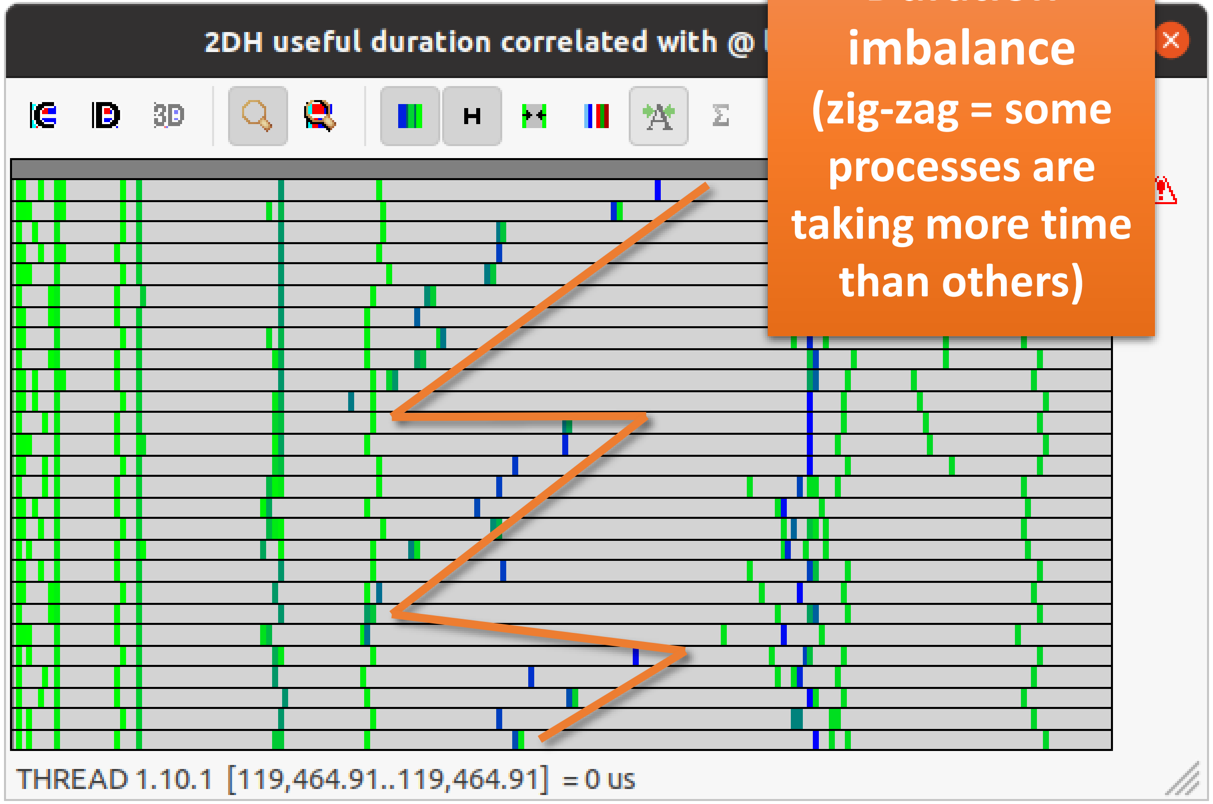
Drag & drop on this area to skip the initializations



THREAD 1.27.1 [760,037.27..765,795.12) = 0 us

Computation time distribution

- Click on “2dh_usefulduration.cfg” (2nd link) → Shows **time computing**

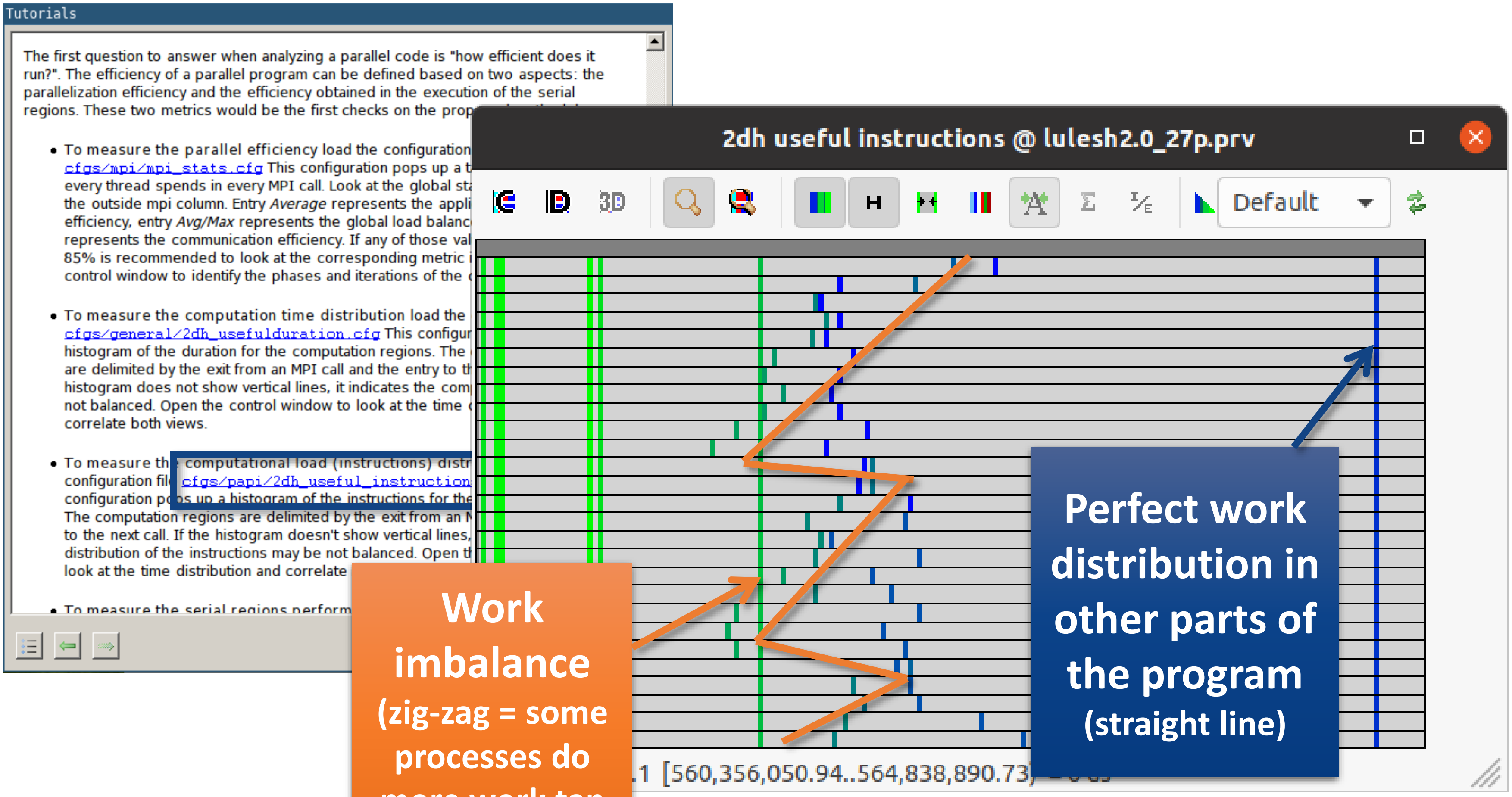


THREAD 1.10.1 [119,464.91..119,464.91] = 0 us

Duration imbalance (zig-zag = some processes are taking more time than others)

Computation load distribution

- Click on “2dh_useful_instructions.cfg” (3rd link) → Shows **amount of work**



The first question to answer when analyzing a parallel code is "how efficient does it
run?". The efficiency of a parallel program can be defined based on two aspects: the
parallelization efficiency and the efficiency obtained in the execution of the serial
regions. These two metrics would be the first checks on the prop

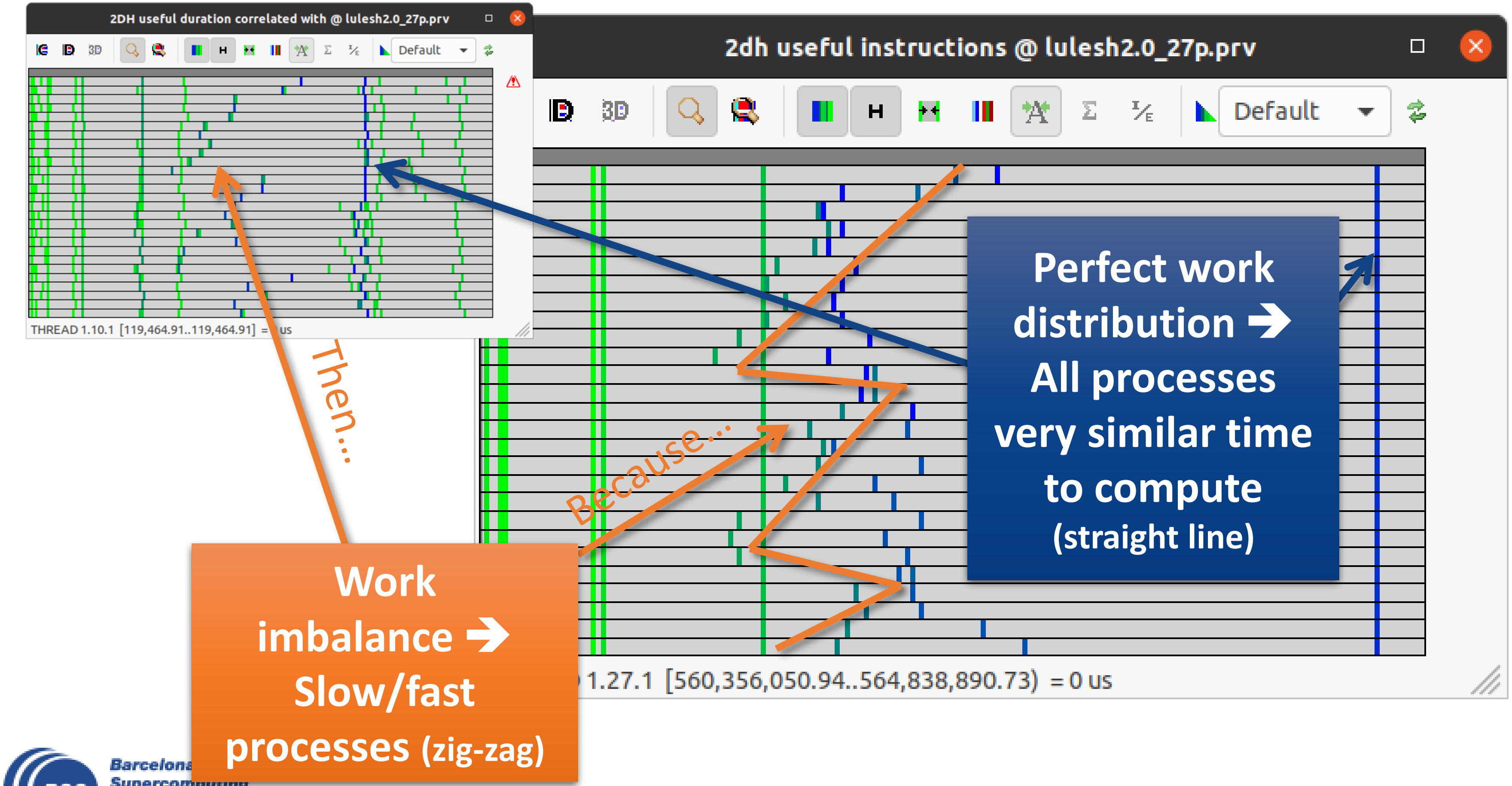
- To measure the parallel efficiency load the configuration
  cfgs/mpi/mpi_stats.cfg This configuration pops up a t
  every thread spends in every MPI call. Look at the global sta
  the outside mpi column. Entry *Average* represents the appli
  efficiency, entry *Avg/Max* represents the global load balanc
  represents the communication efficiency. If any of those val
  85% is recommended to look at the corresponding metric i
  control window to identify the phases and iterations of the
- To measure the computation time distribution load the
  cfgs/general/2dh_usefulduration.cfg This configur
  histogram of the duration for the computation regions. The
  are delimited by the exit from an MPI call and the entry to th
  histogram does not show vertical lines, it indicates the com
  not balanced. Open the control window to look at the time
  correlate both views.
- To measure the computational load (instructions) distr
  configuration file cfgs/papi/2dh_useful_instruction
  configuration pops up a histogram of the instructions for the
  The computation regions are delimited by the exit from an M
  to the next call. If the histogram doesn't show vertical lines,
  distribution of the instructions may be not balanced. Open th
  look at the time distribution and correlate
- To measure the serial regions perform

Work imbalance (zig-zag = some processes do more work tan others)

Perfect work distribution in other parts of the program (straight line)

Computation load distribution

- Comparing the two histograms → **Similar shapes** → Work distribution determines time computing



2DH useful duration correlated with @ lulesh2.0_27p.prv

THREAD 1.10.1 [119,464.91..119,464.91] = us

2dh useful instructions @ lulesh2.0_27p.prv

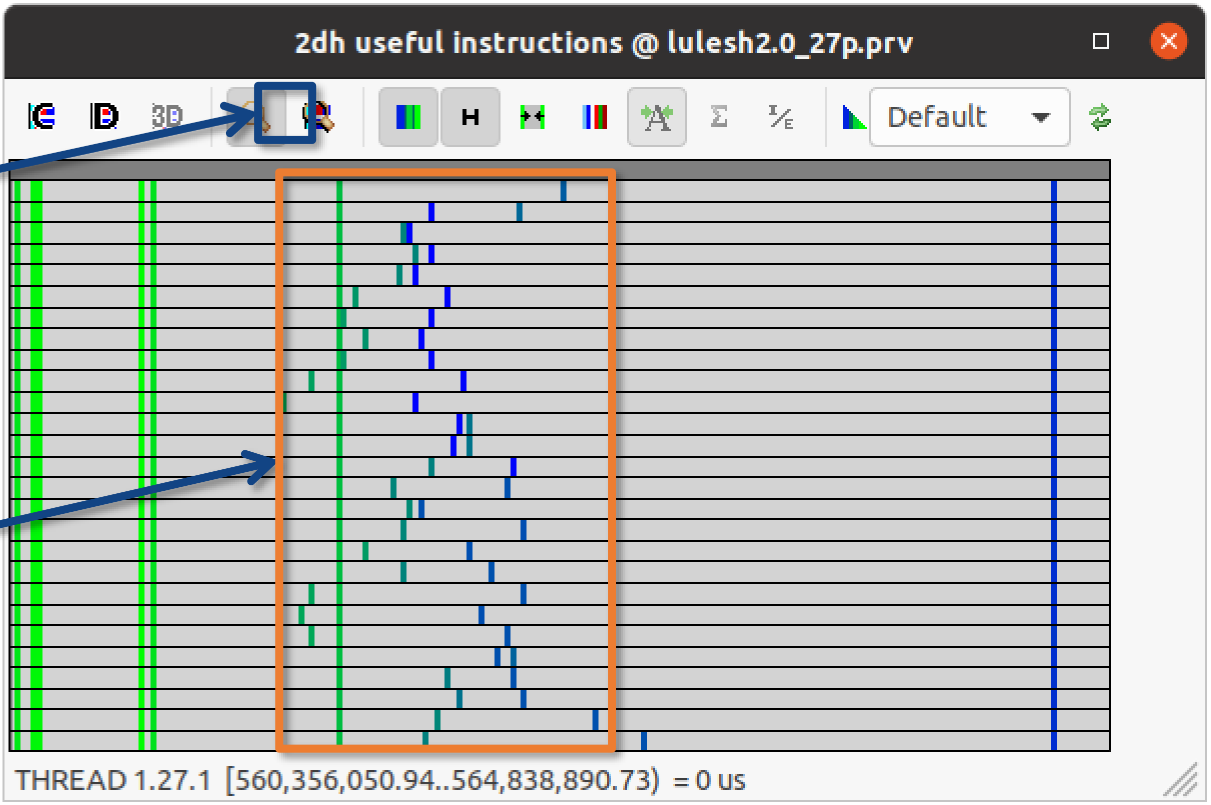
1.27.1 [560,356,050.94..564,838,890.73) = 0 us

Perfect work distribution → All processes very similar time to compute (straight line)

Work imbalance → Slow/fast processes (zig-zag)

Where does this happen?

- Go from the table to the timeline



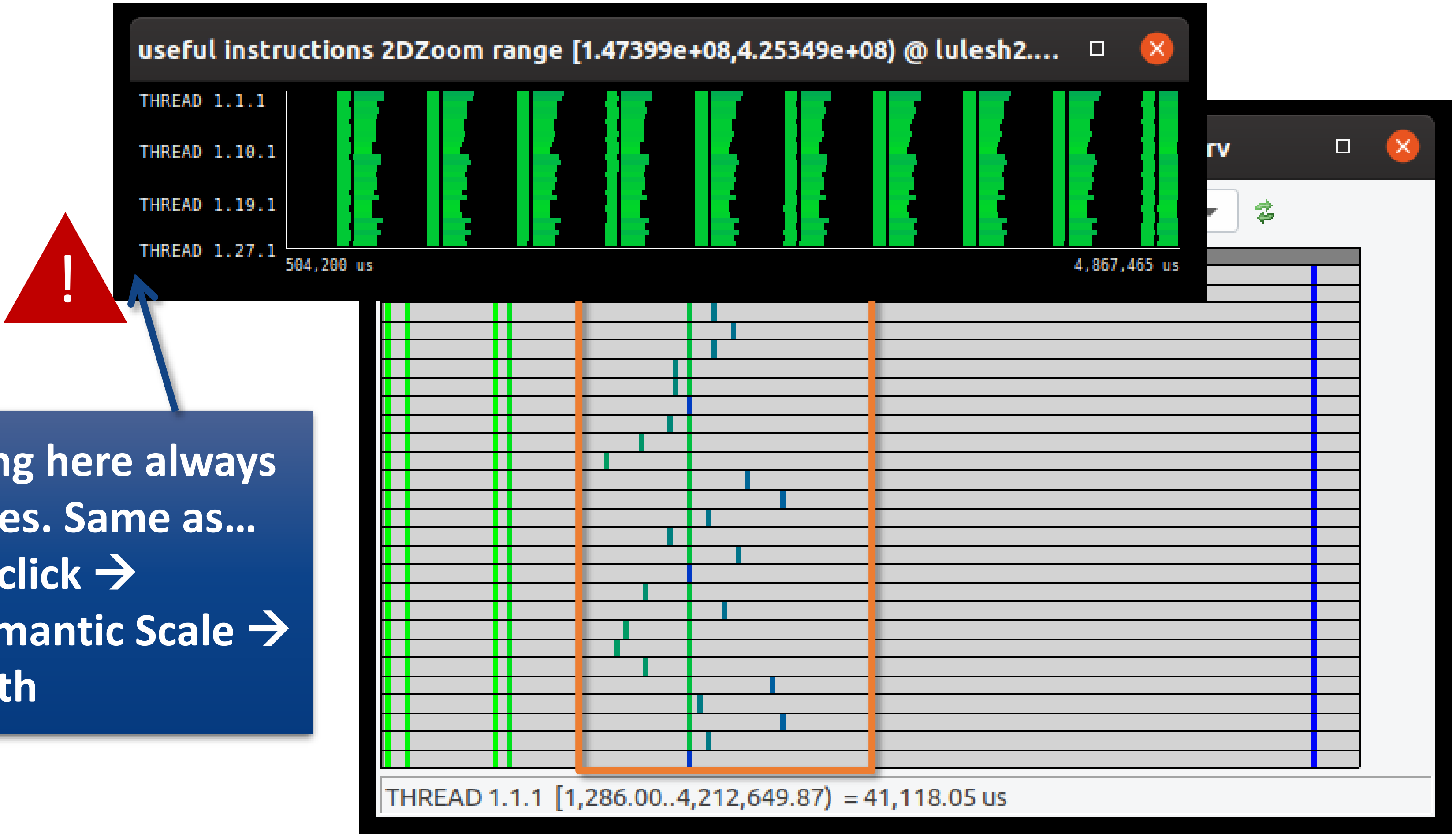
THREAD 1.27.1 [560,356,050.94..564,838,890.73) = 0 us

1. Click on "Open Filtered Control Window"

2. Select this area (drag-and-drop)

Where does this happen?

- Go from the table to the timeline



useful instructions 2DZoom range [1.47399e+08,4.25349e+08) @ lulesh2....

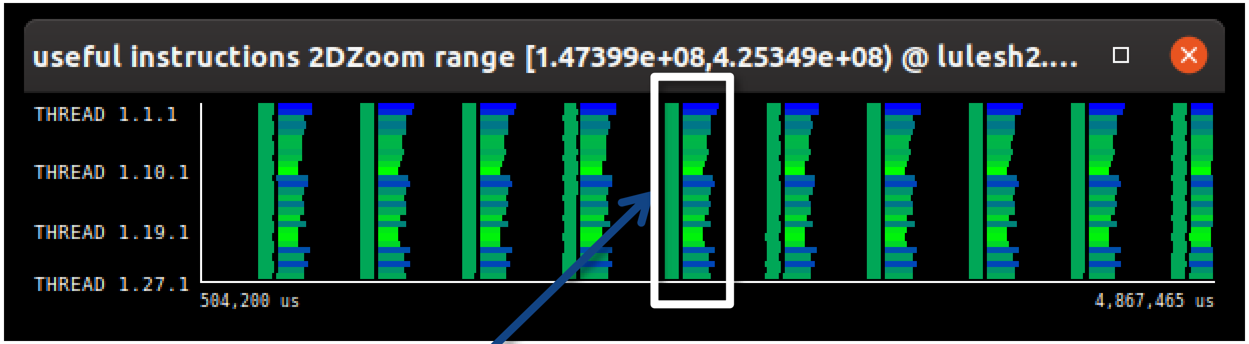
THREAD 1.1.1 [1,286.00..4,212,649.87) = 41,118.05 us

Clicking here always rescales. Same as…
Right click →
Fit Semantic Scale →
Fit Both

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Where does this happen?

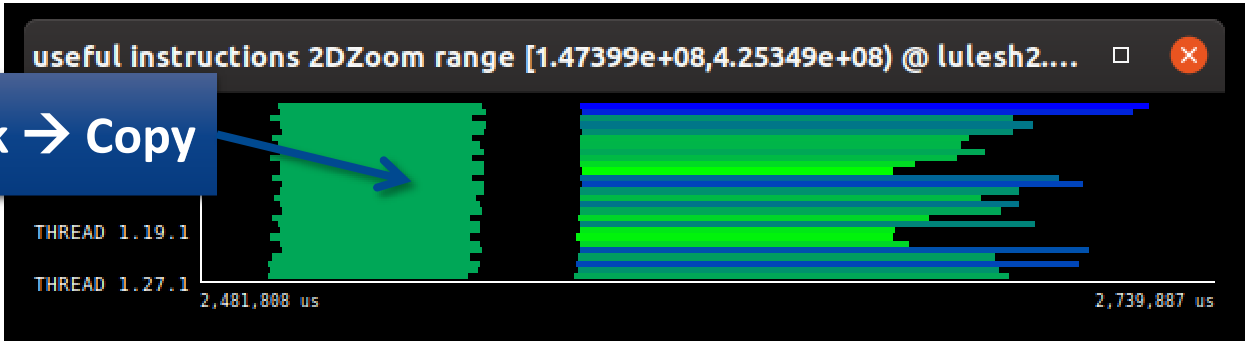
- Slow & Fast at the same time? → Imbalance


useful instructions 2DZoom range [1.47399e+08,4.25349e+08) @ lulesh2....

Zoom into
1 of the iterations
(by drag-and-dropping)

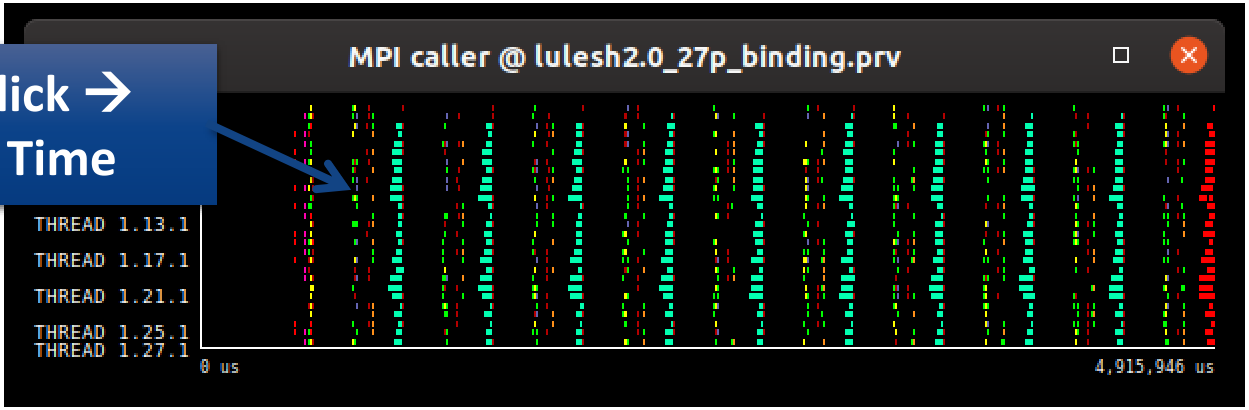
Where does this happen?

- **Slow** & **Fast** at the same time? → Imbalance


useful instructions 2DZoom range [1.47399e+08,4.25349e+08) @ lulesh2....
THREAD 1.19.1
THREAD 1.27.1
2,481,888 us
2,739,887 us

1. Right click → Copy

- **Hints → Call stack references → Caller function**

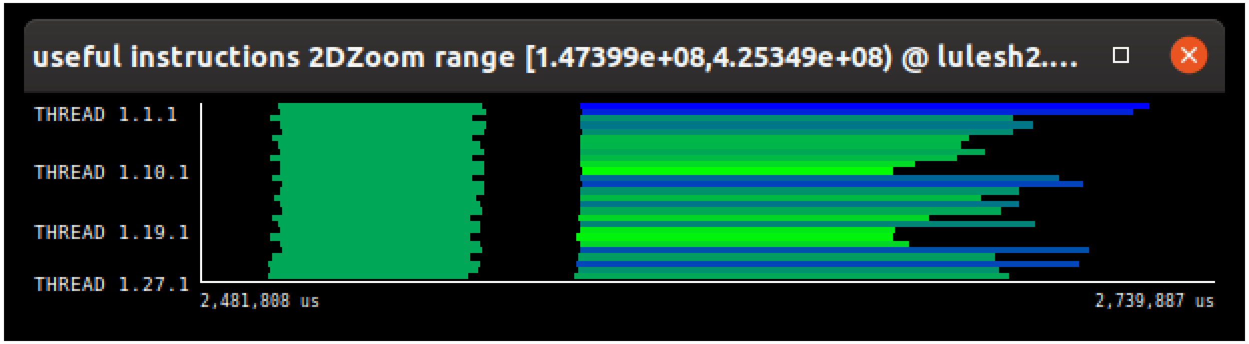


2. Right click → Paste → Time

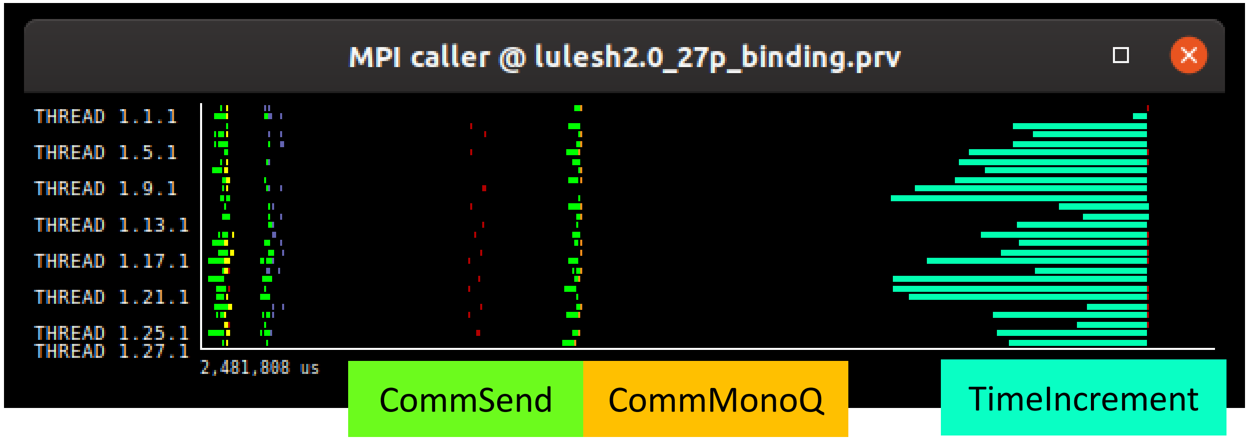
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Where does this happen?

- Slow & Fast at the same time? → Imbalance

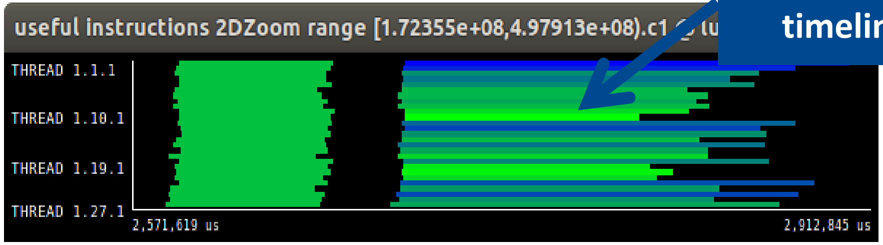


- **Hints → Call stack references → Caller function**

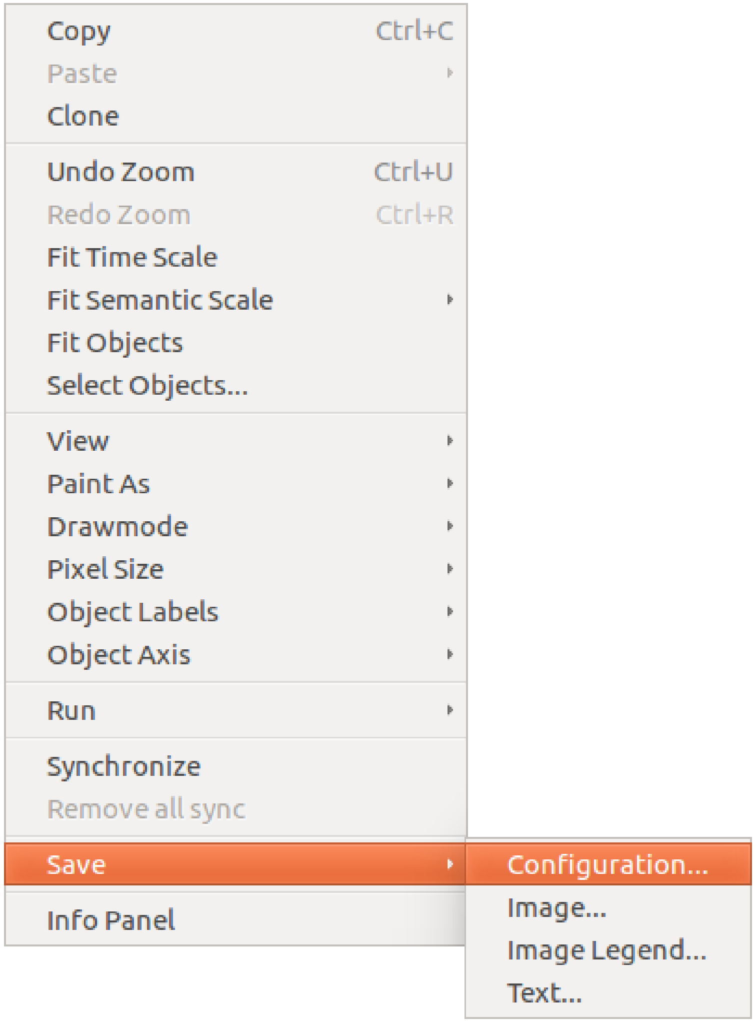


CommSend CommMonoQ TimeIncrement

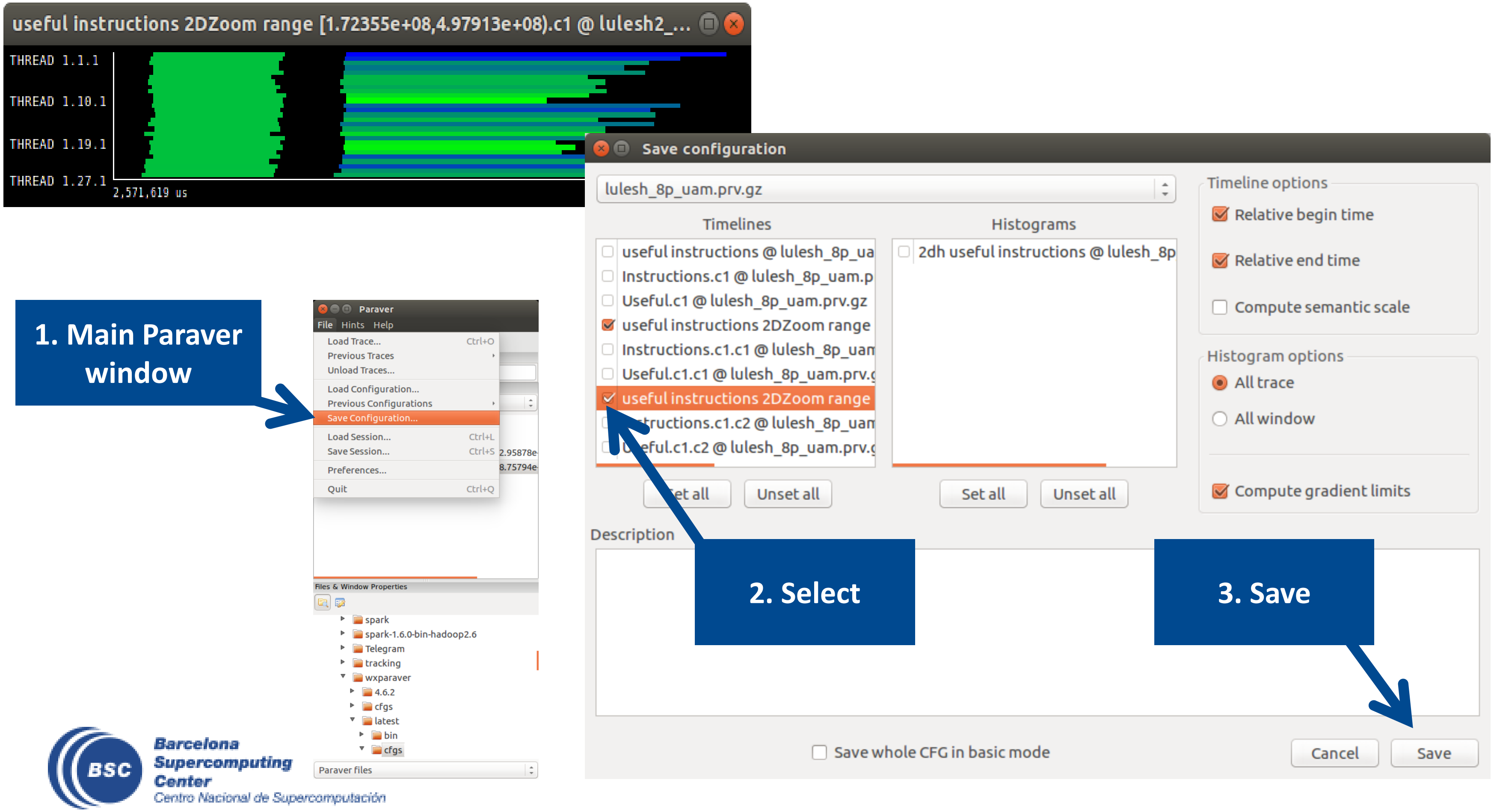
Save CFG's (method 1)



Right click on timeline



Save CFG's (method 2)



useful instructions 2DZoom range [1.72355e+08,4.97913e+08).c1 @ lulesh2_...

1. Main Paraver window

Save configuration

lulesh_8p_uam.prv.gz

Timelines
- □ useful instructions @ lulesh_8p_ua
- □ Instructions.c1 @ lulesh_8p_uam.p
- □ Useful.c1 @ lulesh_8p_uam.prv.gz
- ☑ useful instructions 2DZoom range
- □ Instructions.c1.c1 @ lulesh_8p_uam
- □ Useful.c1.c1 @ lulesh_8p_uam.prv.g
- ☑ useful instructions 2DZoom range
- □ Instructions.c1.c2 @ lulesh_8p_uam
- □ Useful.c1.c2 @ lulesh_8p_uam.prv.g

Histograms
- □ 2dh useful instructions @ lulesh_8p

Timeline options
- ☑ Relative begin time
- ☑ Relative end time
- □ Compute semantic scale

Histogram options
- ● All trace
- ○ All window
- ☑ Compute gradient limits

Description

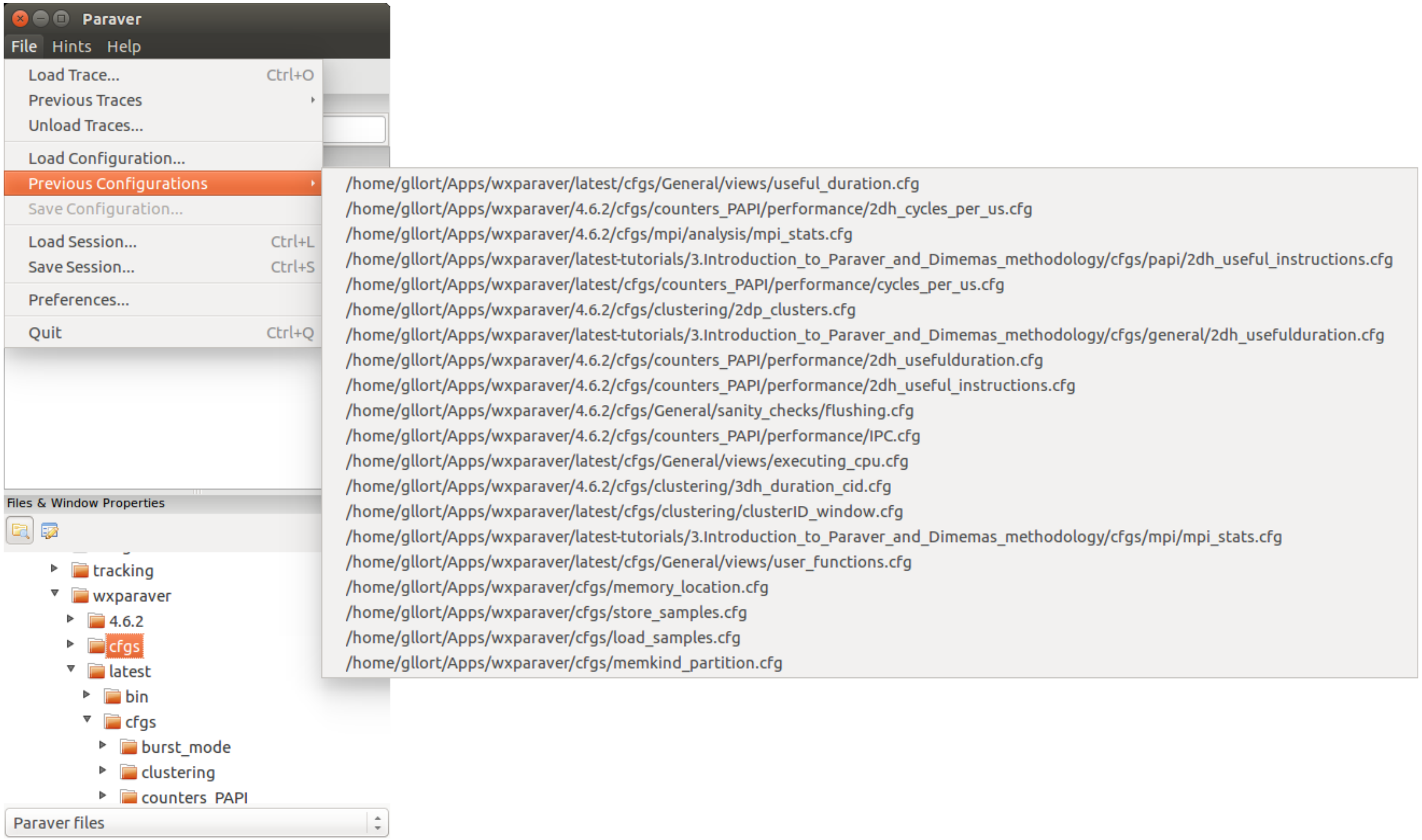
□ Save whole CFG in basic mode

2. Select

3. Save

CFG's distribution

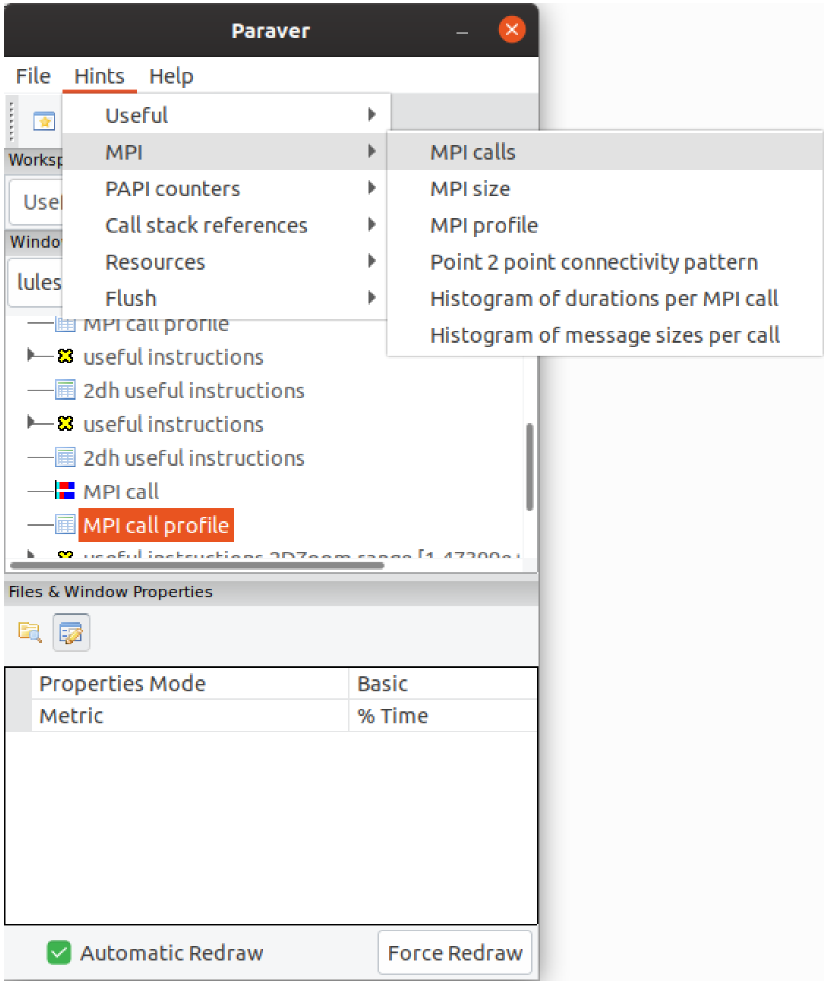
- Paraver comes with many included CFG's ➔ Apply any CFG to any trace!



/home/gllort/Apps/wxparaver/latest/cfgs/General/views/useful_duration.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/counters_PAPI/performance/2dh_cycles_per_us.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/mpi/analysis/mpi_stats.cfg
/home/gllort/Apps/wxparaver/latest-tutorials/3.Introduction_to_Paraver_and_Dimemas_methodology/cfgs/papi/2dh_useful_instructions.cfg
/home/gllort/Apps/wxparaver/latest/cfgs/counters_PAPI/performance/cycles_per_us.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/clustering/2dp_clusters.cfg
/home/gllort/Apps/wxparaver/latest-tutorials/3.Introduction_to_Paraver_and_Dimemas_methodology/cfgs/general/2dh_usefulduration.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/counters_PAPI/performance/2dh_usefulduration.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/counters_PAPI/performance/2dh_useful_instructions.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/General/sanity_checks/flushing.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/counters_PAPI/performance/IPC.cfg
/home/gllort/Apps/wxparaver/latest/cfgs/General/views/executing_cpu.cfg
/home/gllort/Apps/wxparaver/4.6.2/cfgs/clustering/3dh_duration_cid.cfg
/home/gllort/Apps/wxparaver/latest/cfgs/clustering/clusterID_window.cfg
/home/gllort/Apps/wxparaver/latest-tutorials/3.Introduction_to_Paraver_and_Dimemas_methodology/cfgs/mpi/mpi_stats.cfg
/home/gllort/Apps/wxparaver/latest/cfgs/General/views/user_functions.cfg
/home/gllort/Apps/wxparaver/cfgs/memory_location.cfg
/home/gllort/Apps/wxparaver/cfgs/store_samples.cfg
/home/gllort/Apps/wxparaver/cfgs/load_samples.cfg
/home/gllort/Apps/wxparaver/cfgs/memkind_partition.cfg

Hints: a good place to start!

- Paraver suggests CFG's based on the contents of the trace



Properties Mode	Basic
Metric	% Time

Do it on your code!

- Follow guidelines from slides 7-16 to your own code to get a trace
    - There are more examples of tracing scripts for different programming models under $EBROOTEXTRAE/share/examples

- Follow guidelines from slides 17-34 to conduct an initial analysis
    - The usual suspects:
        - Parallel Efficiency is low? Load balance issues?
        - Imbalances in the durations of computations?
        - Are these caused by work imbalance?