BSC Tools
Hands-On

Lau Mercadal
(tools@bsc.es)
Barcelona Supercomputing Center

Getting a trace with Extrae

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
    - …

- Dynamic instrumentation
  - Based on Dyninst (developed by U.Wisconsin / U.Maryland)
    - Instrumentation in memory
    - Binary rewriting

- Alternatives
  - Static link (i.e., PMPI, Extrae API)

Getting your first trace

- Provided folder **`tools-material`** in `/work/ta002/shared/bsctools` contains:
  - Application compiled for the GNU PrgEnv (`lulesh2.0-gnu`)
  - Jobscripts to execute and trace (`job.pbs`, `trace.sh`)
  - Configuration of the tracing tool (`extrae.xml`)
  - Already generated tracefiles (`traces/*.{pcf,prv,row}`)
  - Clustering analysis configuration file (`cluster.xml`)

- Copy this folder to your `$HOME` and you are ready to follow this hands-on tutorial

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 --login

#PBS -l select=2
#PBS -l walltime=00:10:00
#PBS -A ta002
#PBS -N lulesh2.0-gnu-27p

#Change to directory the job was submitted from
cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=1

export TMPDIR=${PBS_O_WORKDIR}

# Run on 27 processes, with 7 processes per socket
aprun -n 27 -S 7 ./lulesh2.0-gnu -i 10 -s 65 -p
```

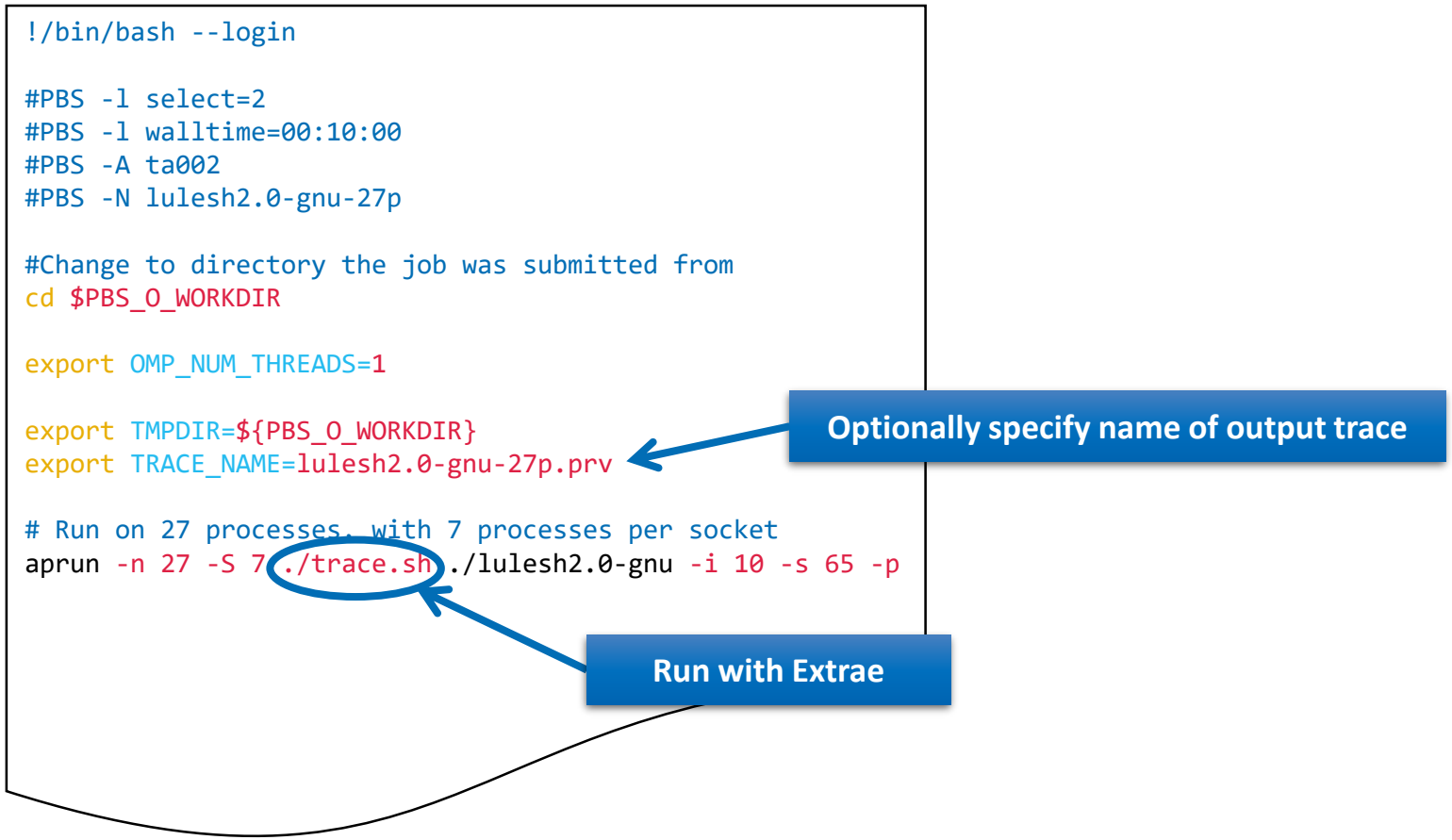
Request resources

Run the program

Step 1: Adapt the job script to load Extrae

- Jobscript modified to load Extrae (`extrae/job.pbs`)

```
!/bin/bash --login

#PBS -l select=2
#PBS -l walltime=00:10:00
#PBS -A ta002
#PBS -N lulesh2.0-gnu-27p

#Change to directory the job was submitted from
cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=1

export TMPDIR=${PBS_O_WORKDIR}
export TRACE_NAME=lulesh2.0-gnu-27p.prv

# Run on 27 processes, with 7 processes per socket
aprun -n 27 -S 7 ./trace.sh ./lulesh2.0-gnu -i 10 -s 65 -p
```



Step 1: Adapt the job script to load Extrae

- Tracing launcher helper script (`extrae/trace.sh`)

```
!/bin/bash --login

#PBS -l select=2
#PBS -l walltime=00:10:00
#PBS -A ta002
#PBS -N lulesh2.0-gnu-27p

#Change to directory the job was submitted from
cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=1

export TMPDIR=${PBS_O_WORKDIR}
export TRACE_NAME=lulesh2.0-gnu-27p.prv

# Run on 27 processes, with 7 processes per socket
aprun -n 27 -S 7 ./trace.sh ./lulesh2.0-gnu -i 10 -s 65 -p
```

```
#!/bin/bash

export
EXTRAE_HOME=/work/ta002/shared/bsctools/extrae/3.8.0-
PrgEnv-gnu_cray-mpich_7.5.5

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

## Run the desired 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] add suffix "f" in Fortran codes

Step 2: Extrae XML configuration

```
<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>
```

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

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

Step 2: Extrae XML configuration (II)

```
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="0">
      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?)

```
<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>
```

Extrae buffer size
(Flush/memory trade-off)

Additional sampling
(Want more details?)

Automatic post-processing to generate the Paraver trace

Step 3: Run it!

- Submit your job as usual

```
archer$ qsub -q R7133965 job.pbs
```

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

All done! Check your resulting trace

- Once finished (check with "`qstat -u $USER`") you will have the trace (3 files):

```
archer$ ls -l
  ...
  lulesh2.0-gnu-27p.pcf
  lulesh2.0-gnu-27p.prv
  lulesh2.0-gnu-27p.row
```

- Any trouble? There's a trace already generated under the "`traces`" folder

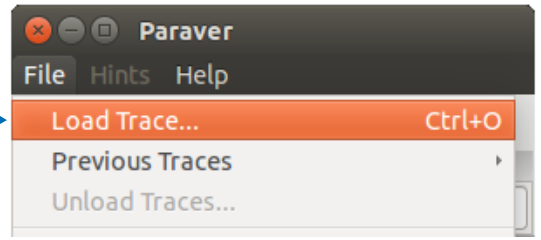
- Now let's look into it!

Analysing a trace with Extrae

First steps of analysis

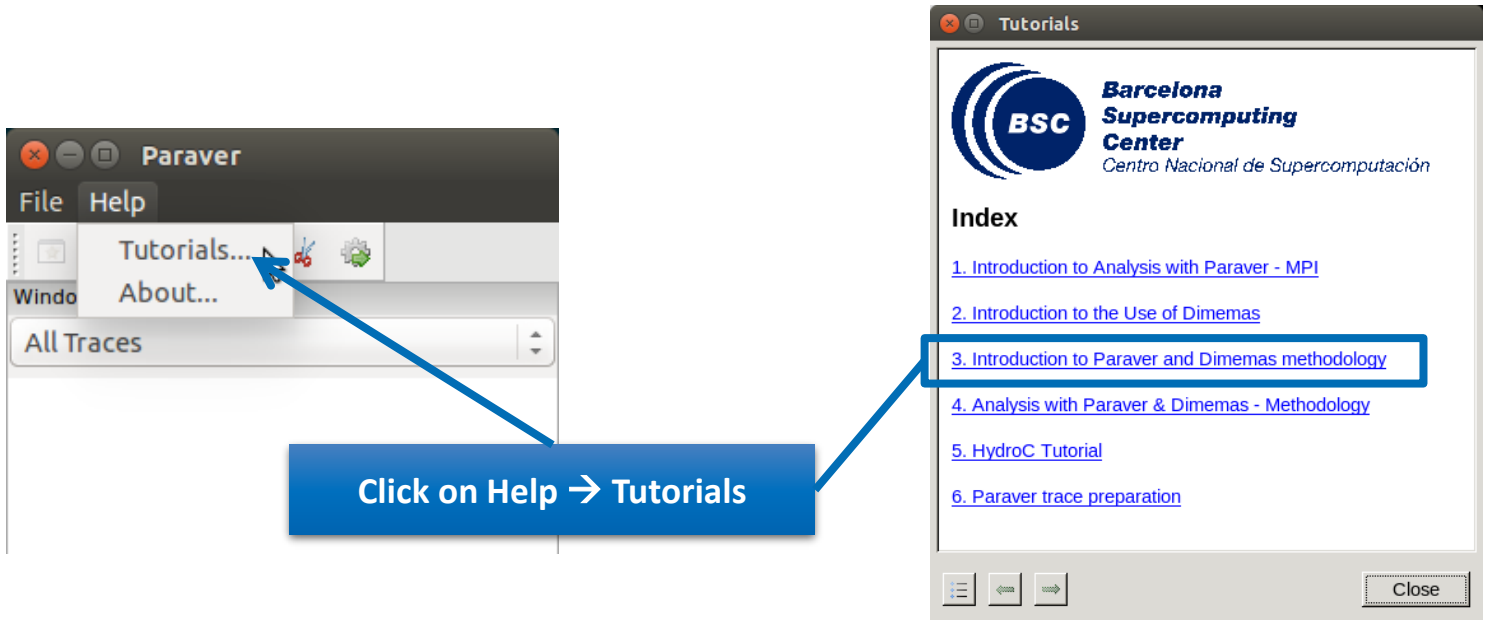
- Copy the trace to your computer
- Load the trace with Paraver

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



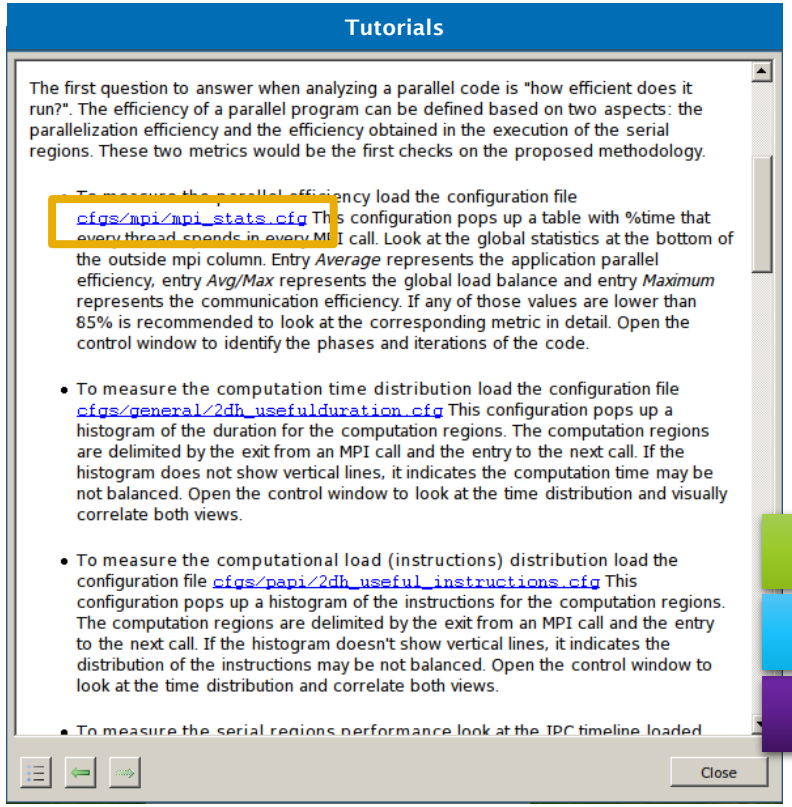
First steps of analysis

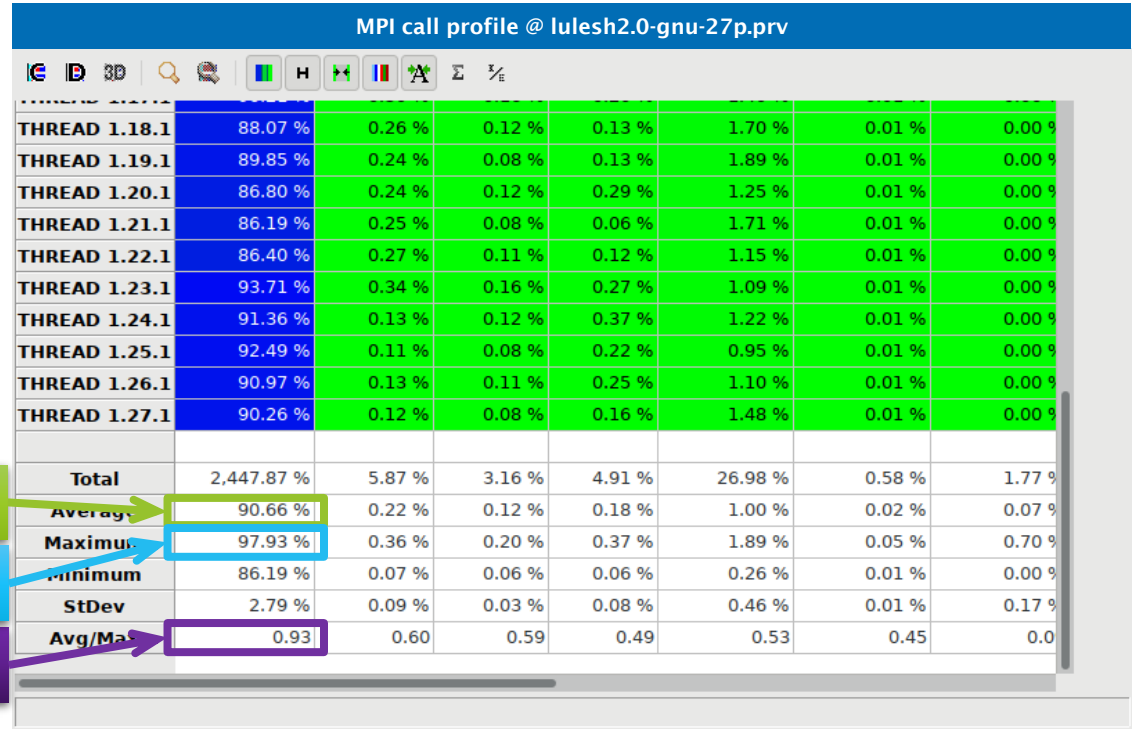
- Follow Tutorial #3
  - Introduction to Paraver and Dimemas methodology



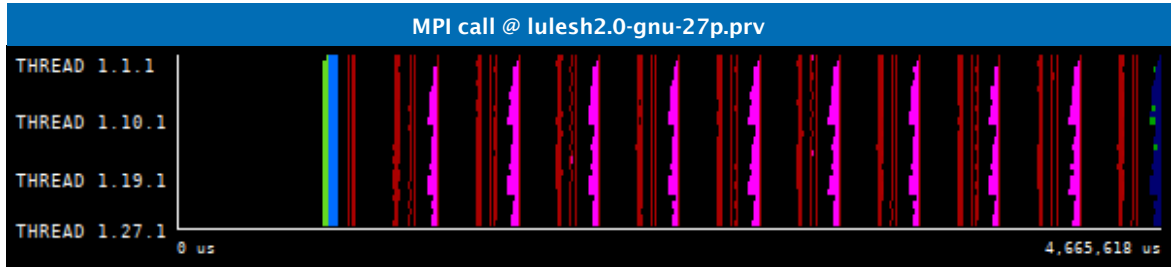
Measure the parallel efficiency

- Click on "mpi_stats.cfg"
  - Check the **Average** for the column labelled "**Outside MPI**"





Focus on the iterative part

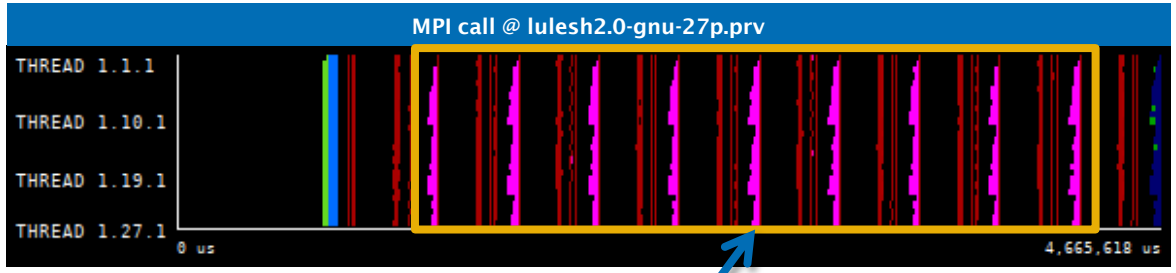


Click on
Open Control Window

MPI call profile @ lulesh2.0-gnu-27p.prv

THREAD 1.18.1	88.07 %	0.26 %	0.12 %	0.13 %	1.70 %	0.01 %	0.00 %
THREAD 1.19.1	89.85 %	0.24 %	0.08 %	0.13 %	1.89 %	0.01 %	0.00 %
THREAD 1.20.1	86.80 %	0.24 %	0.12 %	0.29 %	1.25 %	0.01 %	0.00 %
THREAD 1.21.1	86.19 %	0.25 %	0.08 %	0.06 %	1.71 %	0.01 %	0.00 %
THREAD 1.22.1	86.40 %	0.27 %	0.11 %	0.12 %	1.15 %	0.01 %	0.00 %
THREAD 1.23.1	93.71 %	0.34 %	0.16 %	0.27 %	1.09 %	0.01 %	0.00 %
THREAD 1.24.1	91.36 %	0.13 %	0.12 %	0.37 %	1.22 %	0.01 %	0.00 %
THREAD 1.25.1	92.49 %	0.11 %	0.08 %	0.22 %	0.95 %	0.01 %	0.00 %
THREAD 1.26.1	90.97 %	0.13 %	0.11 %	0.25 %	1.10 %	0.01 %	0.00 %
THREAD 1.27.1	90.26 %	0.12 %	0.08 %	0.16 %	1.48 %	0.01 %	0.00 %
Total	2,447.87 %	5.87 %	3.16 %	4.91 %	26.98 %	0.58 %	1.77 %
Average	90.66 %	0.22 %	0.12 %	0.18 %	1.00 %	0.02 %	0.07 %
Maximum	97.93 %	0.36 %	0.20 %	0.37 %	1.89 %	0.05 %	0.70 %
Minimum	86.19 %	0.07 %	0.06 %	0.06 %	0.26 %	0.01 %	0.00 %
StDev	2.79 %	0.09 %	0.03 %	0.08 %	0.46 %	0.01 %	0.17 %
Avg/Max	0.93	0.60	0.59	0.49	0.53	0.45	0.0

Focus on the iterative part

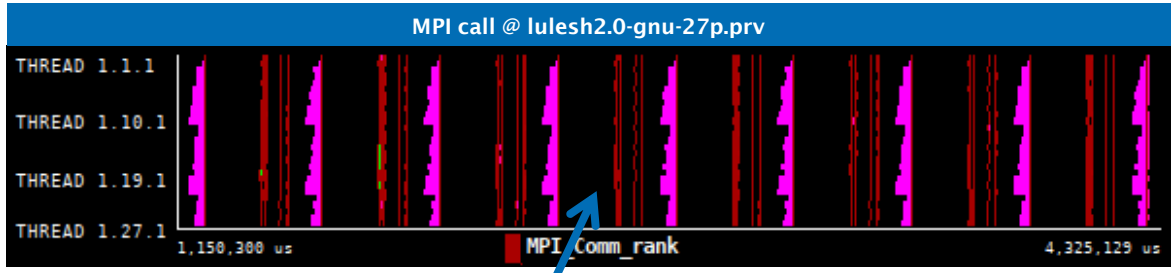


Drag & drop on this área
to zoom on the iterative region

MPI call profile @ lulesh2.0-gnu-27p.prv

THREAD 1.18.1	88.07 %	0.26 %	0.12 %	0.13 %	1.70 %	0.01 %	0.00 %
THREAD 1.19.1	89.85 %	0.24 %	0.08 %	0.13 %	1.89 %	0.01 %	0.00 %
THREAD 1.20.1	86.80 %	0.24 %	0.12 %	0.29 %	1.25 %	0.01 %	0.00 %
THREAD 1.21.1	86.19 %	0.25 %	0.08 %	0.06 %	1.71 %	0.01 %	0.00 %
THREAD 1.22.1	86.40 %	0.27 %	0.11 %	0.12 %	1.15 %	0.01 %	0.00 %
THREAD 1.23.1	93.71 %	0.34 %	0.16 %	0.27 %	1.09 %	0.01 %	0.00 %
THREAD 1.24.1	91.36 %	0.13 %	0.12 %	0.37 %	1.22 %	0.01 %	0.00 %
THREAD 1.25.1	92.49 %	0.11 %	0.08 %	0.22 %	0.95 %	0.01 %	0.00 %
THREAD 1.26.1	90.97 %	0.13 %	0.11 %	0.25 %	1.10 %	0.01 %	0.00 %
THREAD 1.27.1	90.26 %	0.12 %	0.08 %	0.16 %	1.48 %	0.01 %	0.00 %
Total	2,447.87 %	5.87 %	3.16 %	4.91 %	26.98 %	0.58 %	1.77 %
Average	90.66 %	0.22 %	0.12 %	0.18 %	1.00 %	0.02 %	0.07 %
Maximum	97.93 %	0.36 %	0.20 %	0.37 %	1.89 %	0.05 %	0.70 %
Minimum	86.19 %	0.07 %	0.06 %	0.06 %	0.26 %	0.01 %	0.00 %
StDev	2.79 %	0.09 %	0.03 %	0.08 %	0.46 %	0.01 %	0.17 %
Avg/Max	0.93	0.60	0.59	0.49	0.53	0.45	0.0

Recalculate efficiency of iterative region

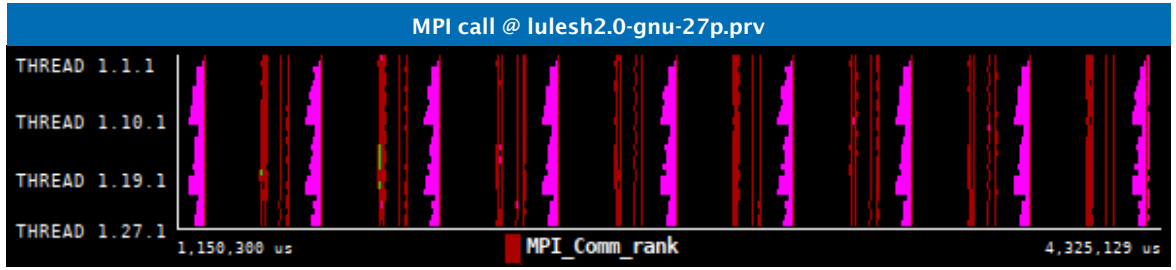


Right click → Copy

MPI call profile @ lulesh2.0-gnu-27p.prv

THREAD 1.18.1	88.07 %	0.26 %	0.12 %	0.13 %	1.70 %	0.01 %	0.00 %
THREAD 1.19.1	89.85 %	0.24 %	0.08 %	0.13 %	1.89 %	0.01 %	0.00 %
THREAD 1.20.1	86.80 %	0.24 %	0.12 %	0.29 %	1.25 %	0.01 %	0.00 %
THREAD 1.21.1	86.19 %	0.25 %	0.08 %	0.06 %	1.71 %	0.01 %	0.00 %
THREAD 1.22.1	86.40 %	0.27 %	0.11 %	0.12 %	1.15 %	0.01 %	0.00 %
THREAD 1.23.1	93.71 %	0.34 %	0.16 %	0.27 %	1.09 %	0.01 %	0.00 %
THREAD 1.24.1	91.36 %	0.13 %	0.12 %	0.37 %	1.22 %	0.01 %	0.00 %
THREAD 1.25.1	92.49 %	0.11 %	0.08 %	0.22 %	0.95 %	0.01 %	0.00 %
THREAD 1.26.1	90.97 %	0.13 %	0.11 %	0.25 %	1.10 %	0.01 %	0.00 %
THREAD 1.27.1	90.26 %	0.12 %	0.08 %	0.16 %	1.48 %	0.01 %	0.00 %
Total	2,447.87 %	5.87 %	3.16 %	4.91 %	26.98 %	0.58 %	1.77 %
Average	90.66 %	0.22 %	0.12 %	0.18 %	1.00 %	0.02 %	0.07 %
Maximum	97.93 %	0.36 %	0.20 %	0.37 %	1.89 %	0.05 %	0.70 %
Minimum	86.19 %	0.07 %	0.06 %	0.06 %	0.26 %	0.01 %	0.00 %
StDev	2.79 %	0.09 %	0.03 %	0.08 %	0.46 %	0.01 %	0.17 %
Avg/Max	0.93	0.60	0.59	0.49	0.53	0.45	0.0

Recalculate efficiency of iterative region



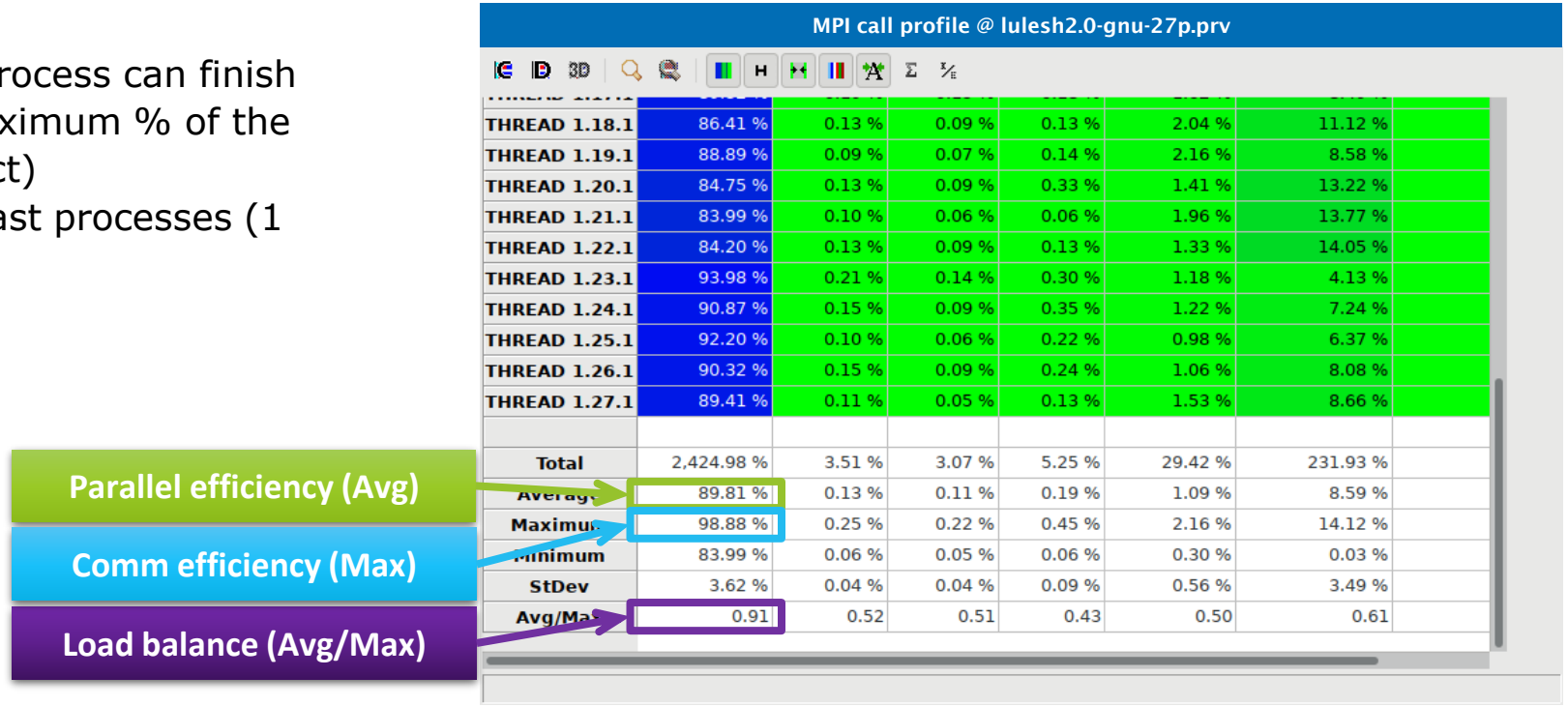
Right click → Paste → Time

MPI call profile @ lulesh2.0-gnu-27p.prv

THREAD 1.18.1	86.41 %	0.13 %	0.09 %	0.13 %	2.04 %	11.12 %
THREAD 1.19.1	88.89 %	0.09 %	0.07 %	0.14 %	2.16 %	8.58 %
THREAD 1.20.1	84.75 %	0.13 %	0.09 %	0.33 %	1.41 %	13.22 %
THREAD 1.21.1	83.99 %	0.10 %	0.06 %	0.06 %	1.96 %	13.77 %
THREAD 1.22.1	84.20 %	0.13 %	0.09 %	0.13 %	1.33 %	14.05 %
THREAD 1.23.1	93.98 %	0.21 %	0.14 %	0.30 %	1.18 %	4.13 %
THREAD 1.24.1	90.87 %	0.15 %	0.09 %	0.35 %	1.22 %	7.24 %
THREAD 1.25.1	92.20 %	0.10 %	0.06 %	0.22 %	0.98 %	6.37 %
THREAD 1.26.1	90.32 %	0.15 %	0.09 %	0.24 %	1.06 %	8.08 %
THREAD 1.27.1	89.41 %	0.11 %	0.05 %	0.13 %	1.53 %	8.66 %
Total	2,424.98 %	3.51 %	3.07 %	5.25 %	29.42 %	231.93 %
Average	89.81 %	0.13 %	0.11 %	0.19 %	1.09 %	8.59 %
Maximum	98.88 %	0.25 %	0.22 %	0.45 %	2.16 %	14.12 %
Minimum	83.99 %	0.06 %	0.05 %	0.06 %	0.30 %	0.03 %
StDev	3.62 %	0.04 %	0.04 %	0.09 %	0.56 %	3.49 %
Avg/Max	0.91	0.52	0.51	0.43	0.50	0.61

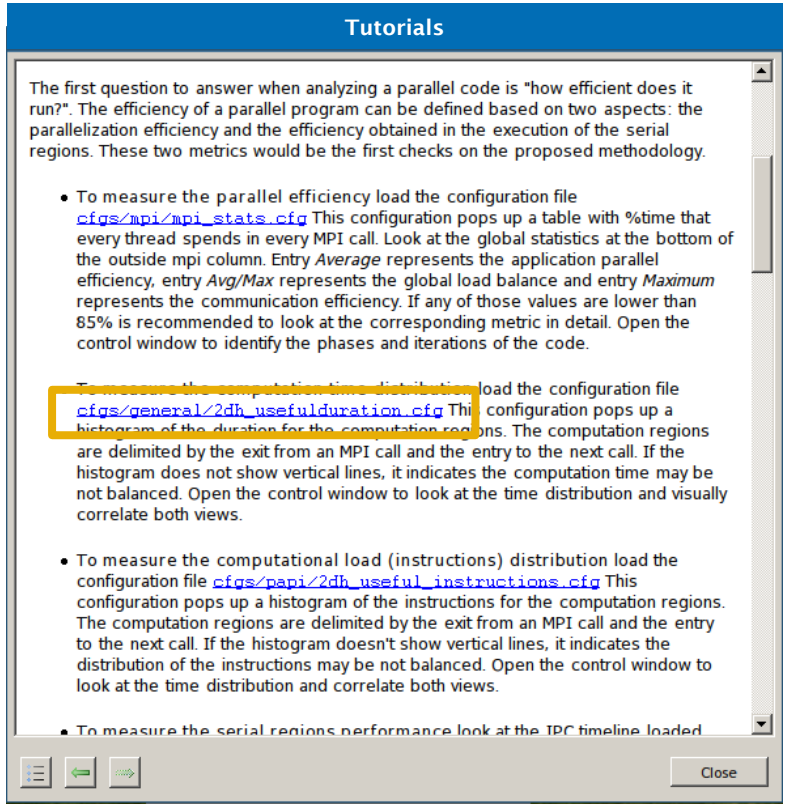
Efficiency of iterative region

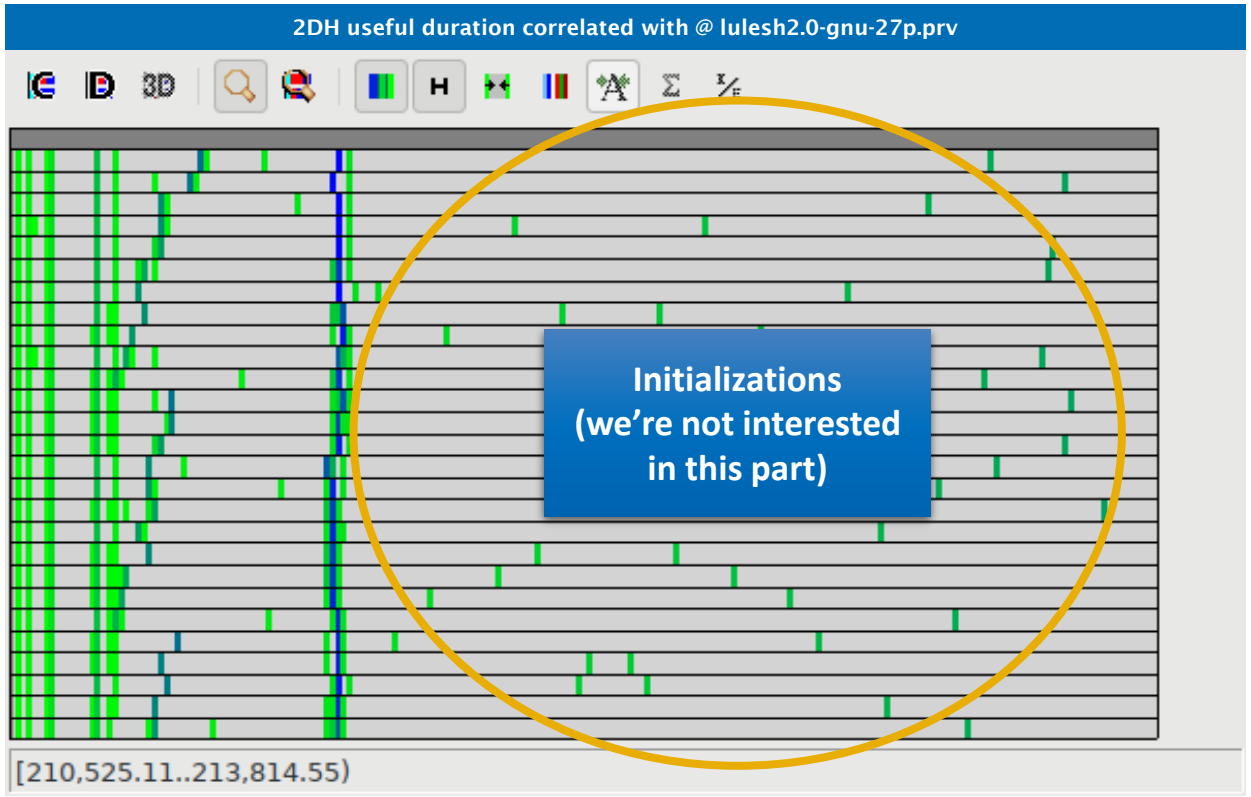
- 3 numbers to quickly describe the efficiency of your code
  - Parallel efficiency → % of time my program is computing (100% is perfect)
  - Comm efficiency → At least 1 process can finish all communications in 100 - Maximum % of the program's time (100% is perfect)
  - Load balance → Ratio of slow/fast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



Computation time distribution

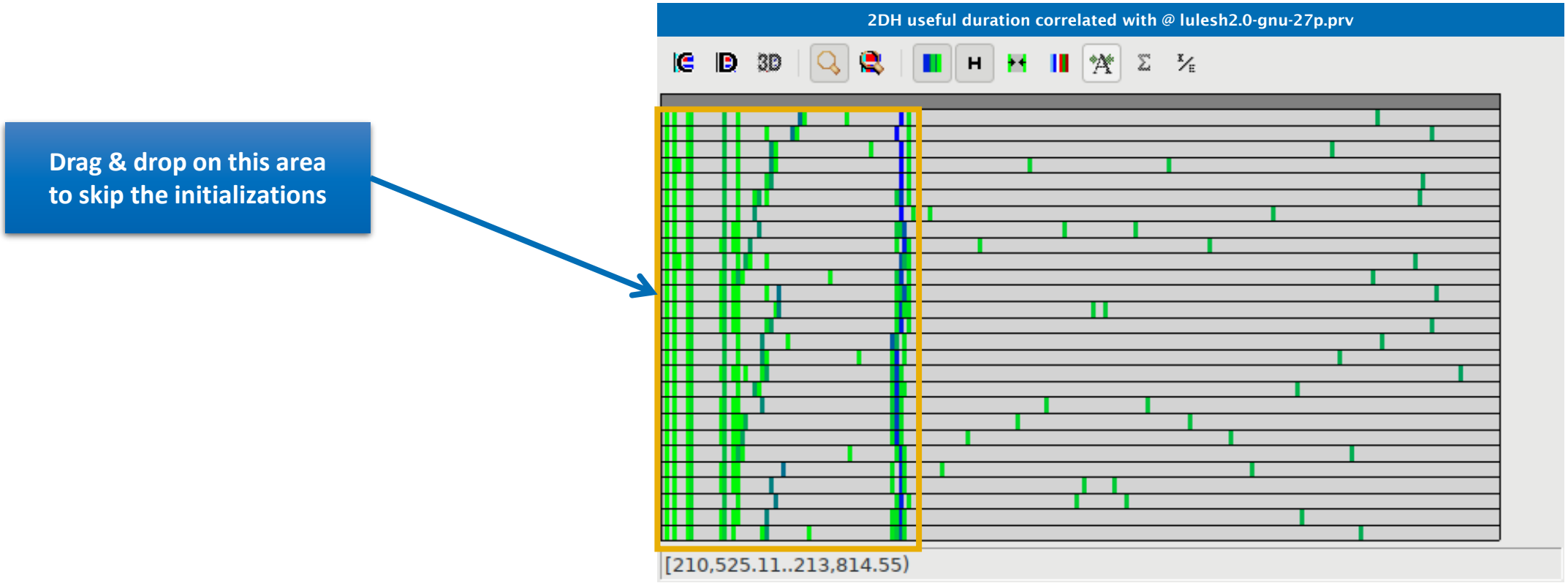
- Click on "2dh_usefulduration.cfg" (2nd link) → Shows **time computing**





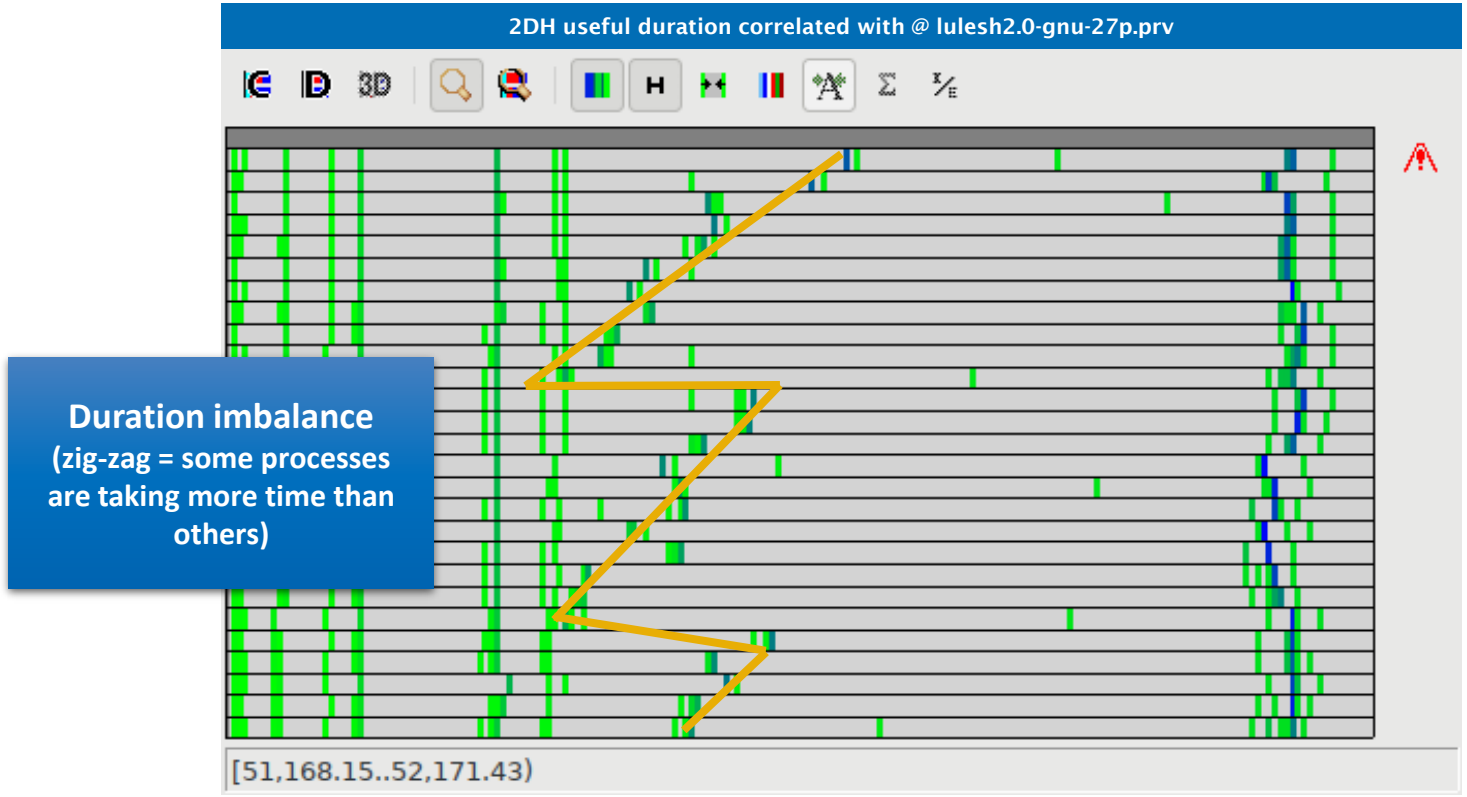
Focus on the iterative part

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



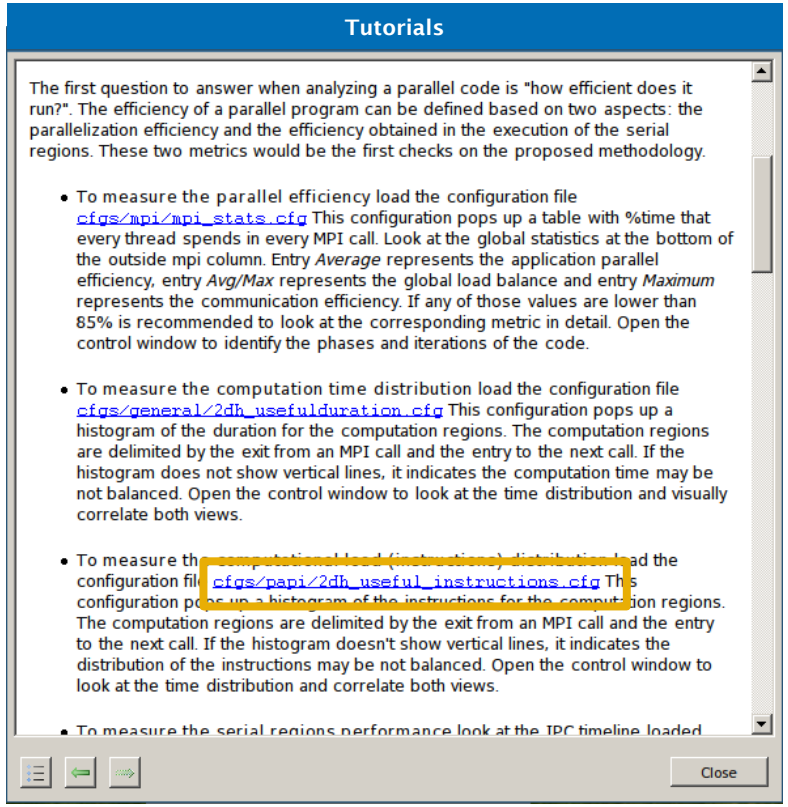
Computation time distribution

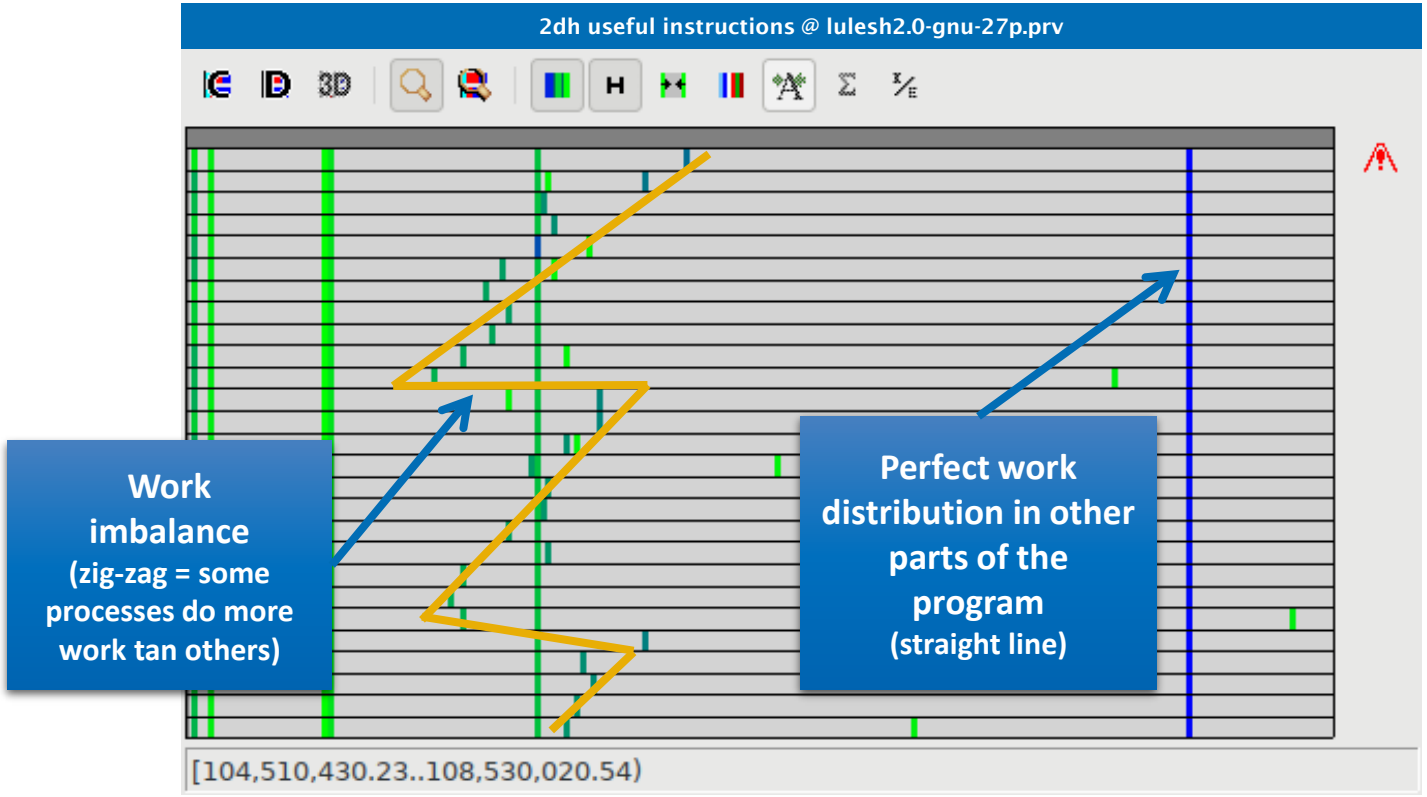
- Click on "2dh_usefulduration.cfg" (2nd link) → Shows **time computing**



Computation load distribution

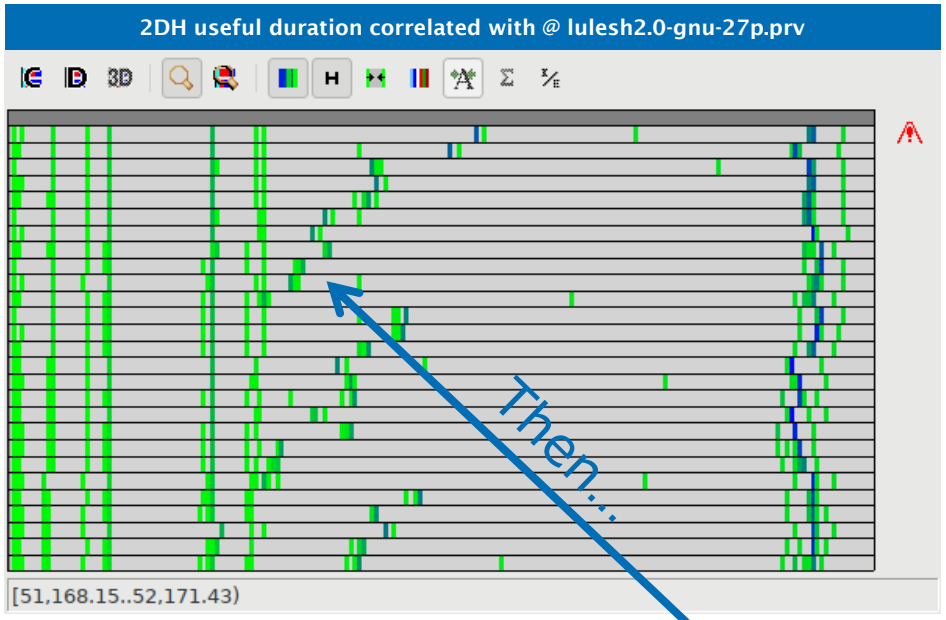
- Click on "2dh_useful_instructions.cfg" (3rd link) → Shows **amount of work**





Computation load distribution

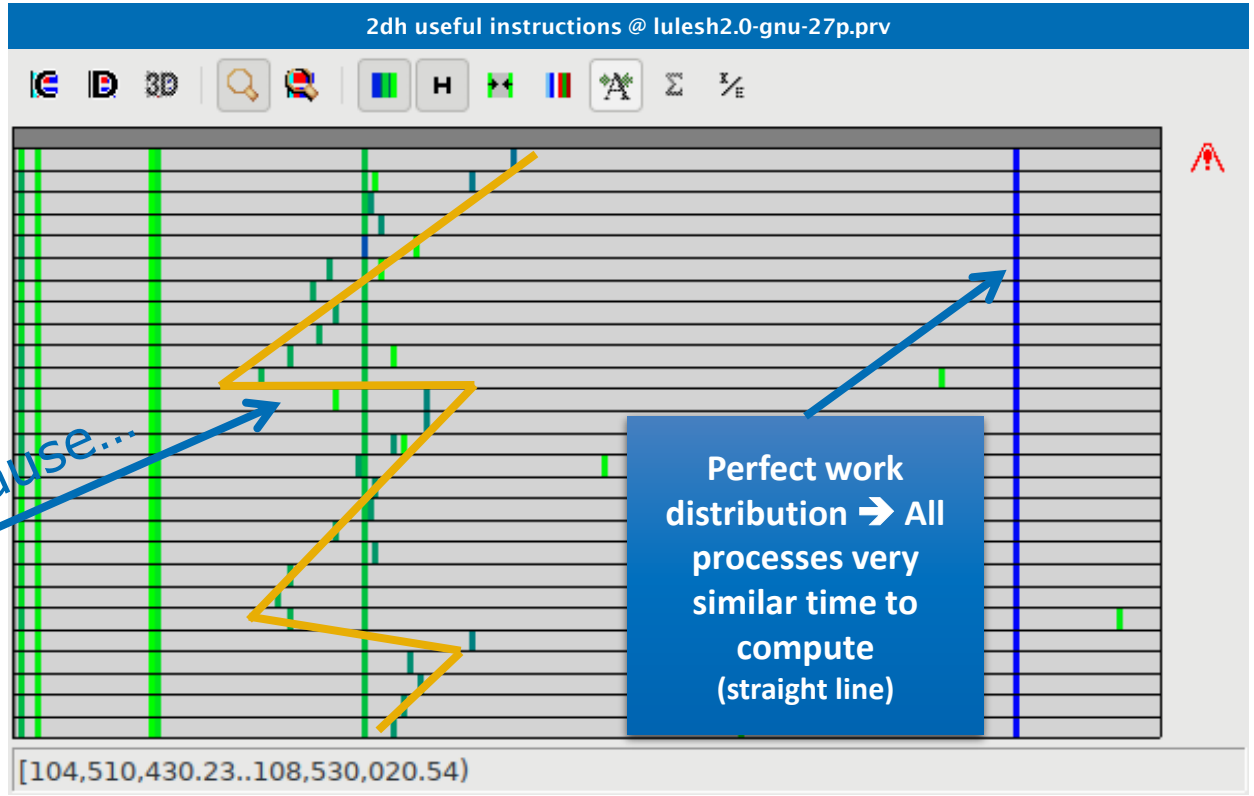
- Comparing the two histograms → **Similar shapes** → Work distribution determines time computing



Then...

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

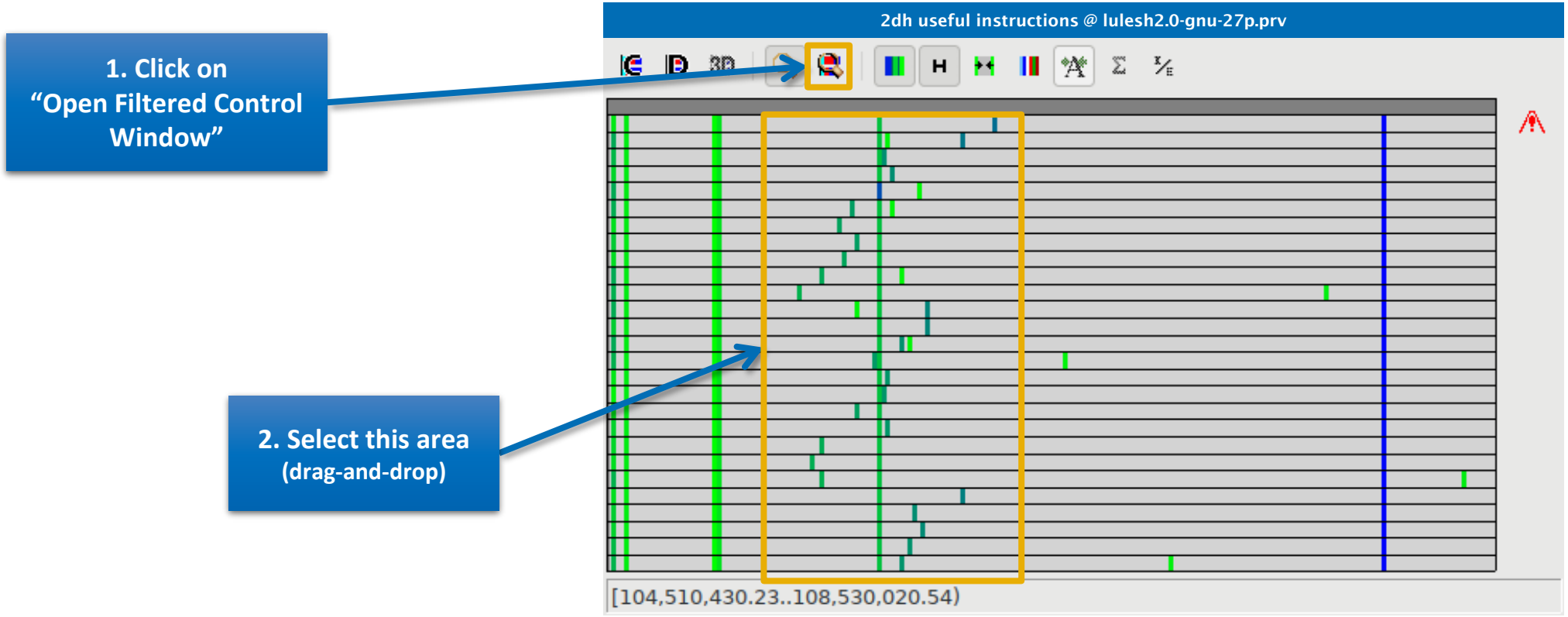
Because...



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

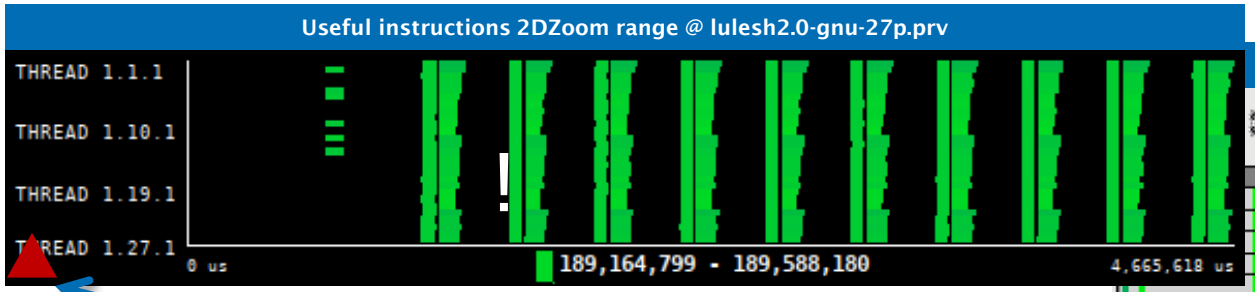
Where does this happen?

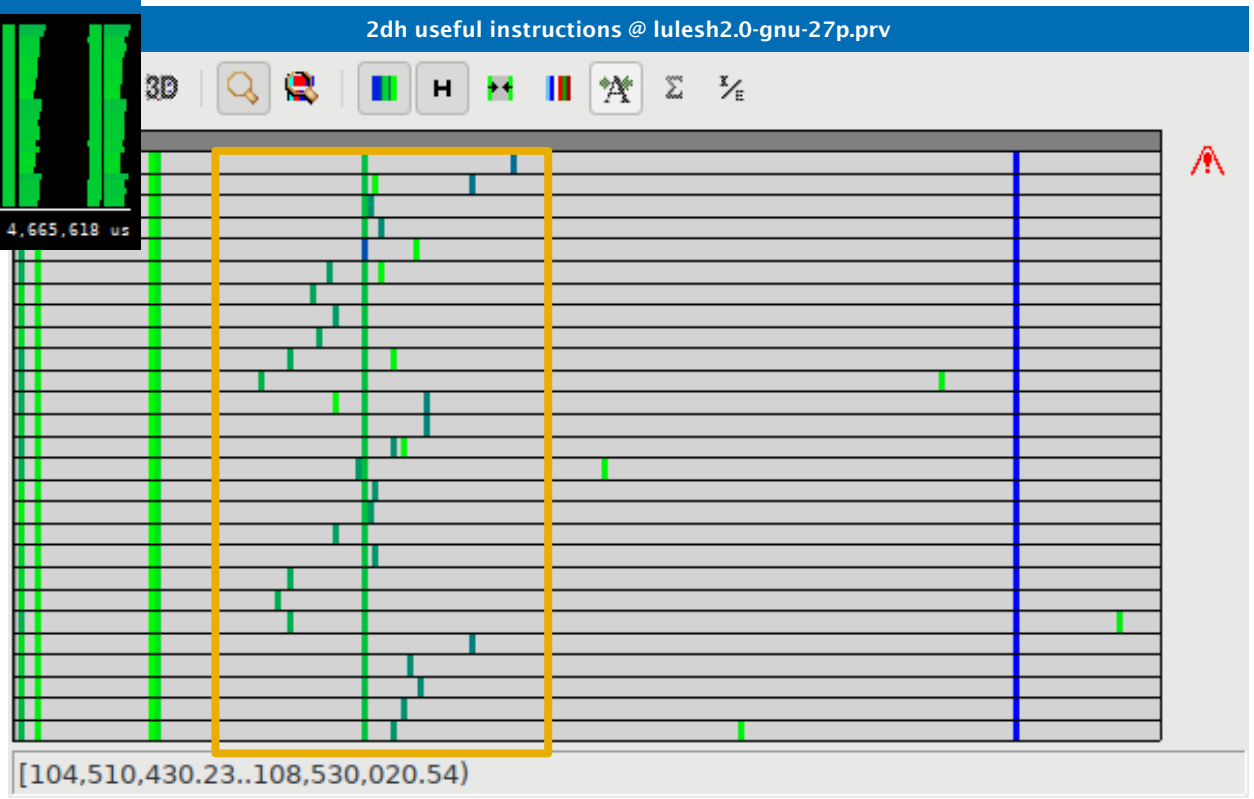
- Go from the table to the timeline



Where does this happen?

- Go from the table to the timeline

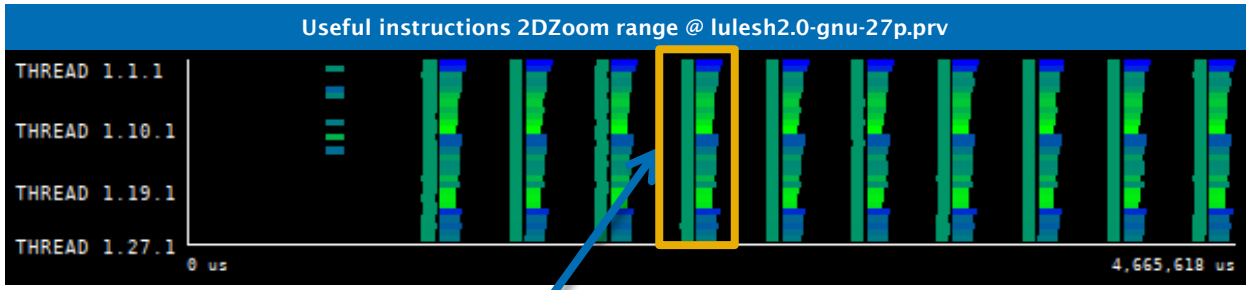




Clicking here always rescales.
Same as…
Right click →
Fit Semantic Scale → Fit Both

Where does this happen?

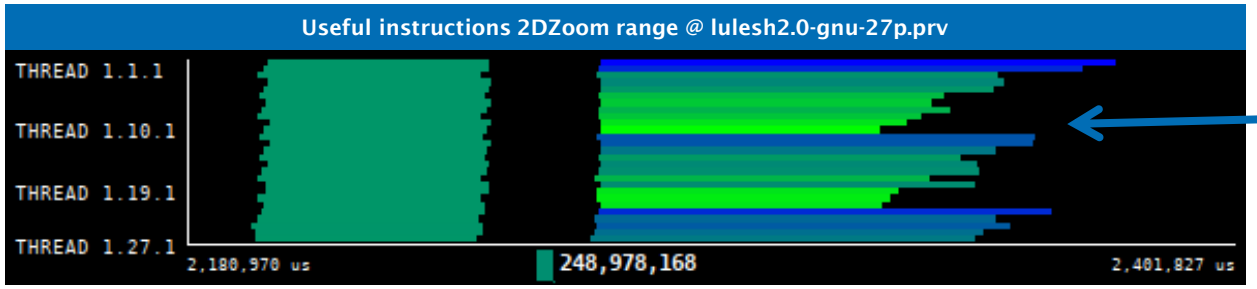
- **Slow** & **Fast** at the same time? → Imbalance



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

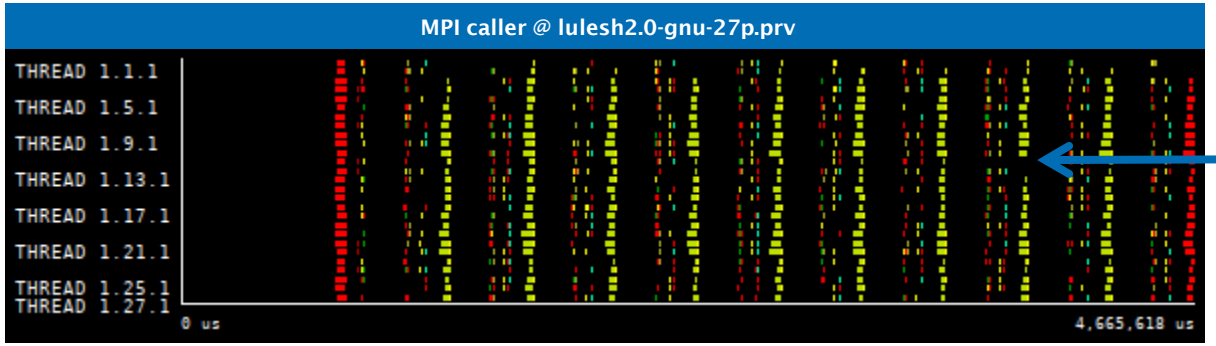
Where does this happen?

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



1. Right click → Copy

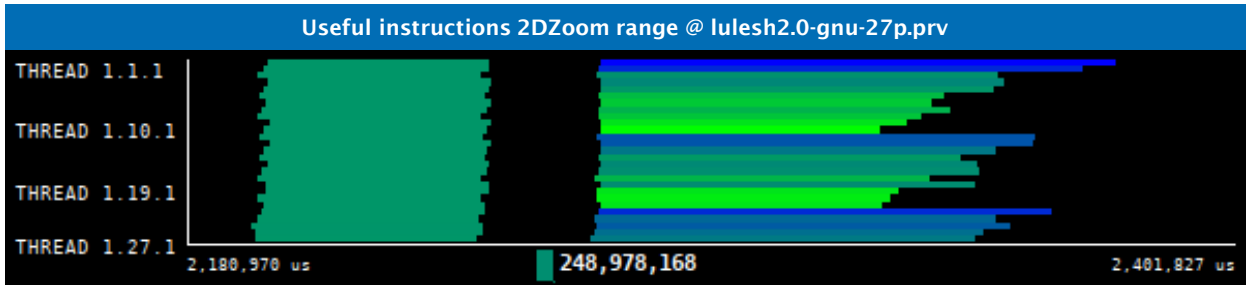
- **Hints → Call stack references → Caller function**



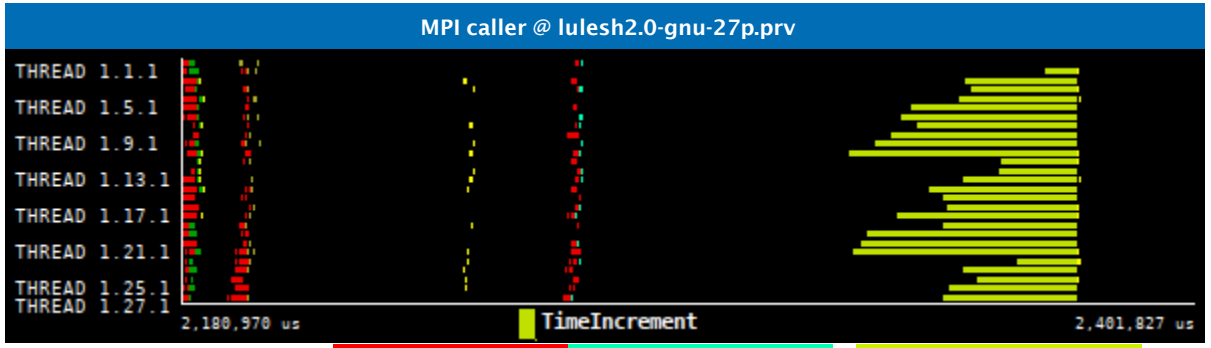
2. Right click → Paste → Time

Where does this happen?

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

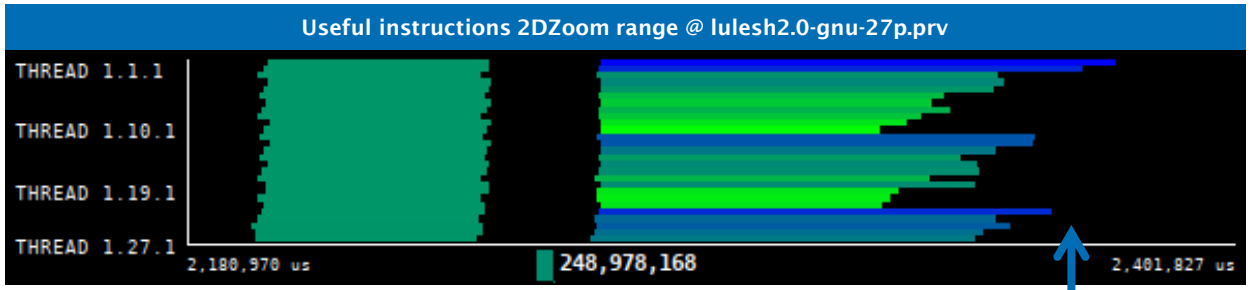


- **Hints → Call stack references → Caller function**

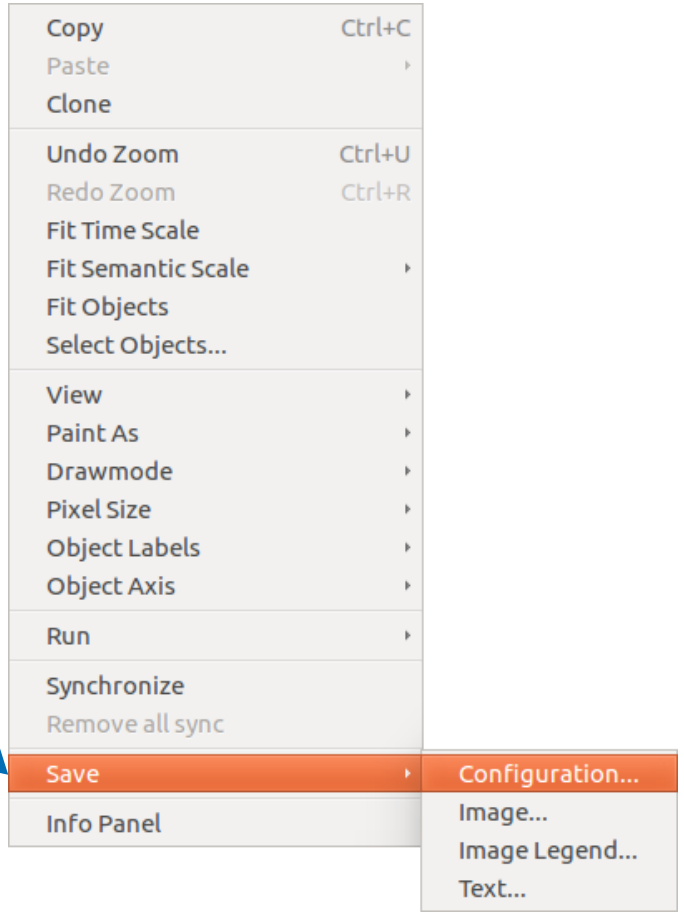


CommSend | CommMonoQ | TimeIncrement

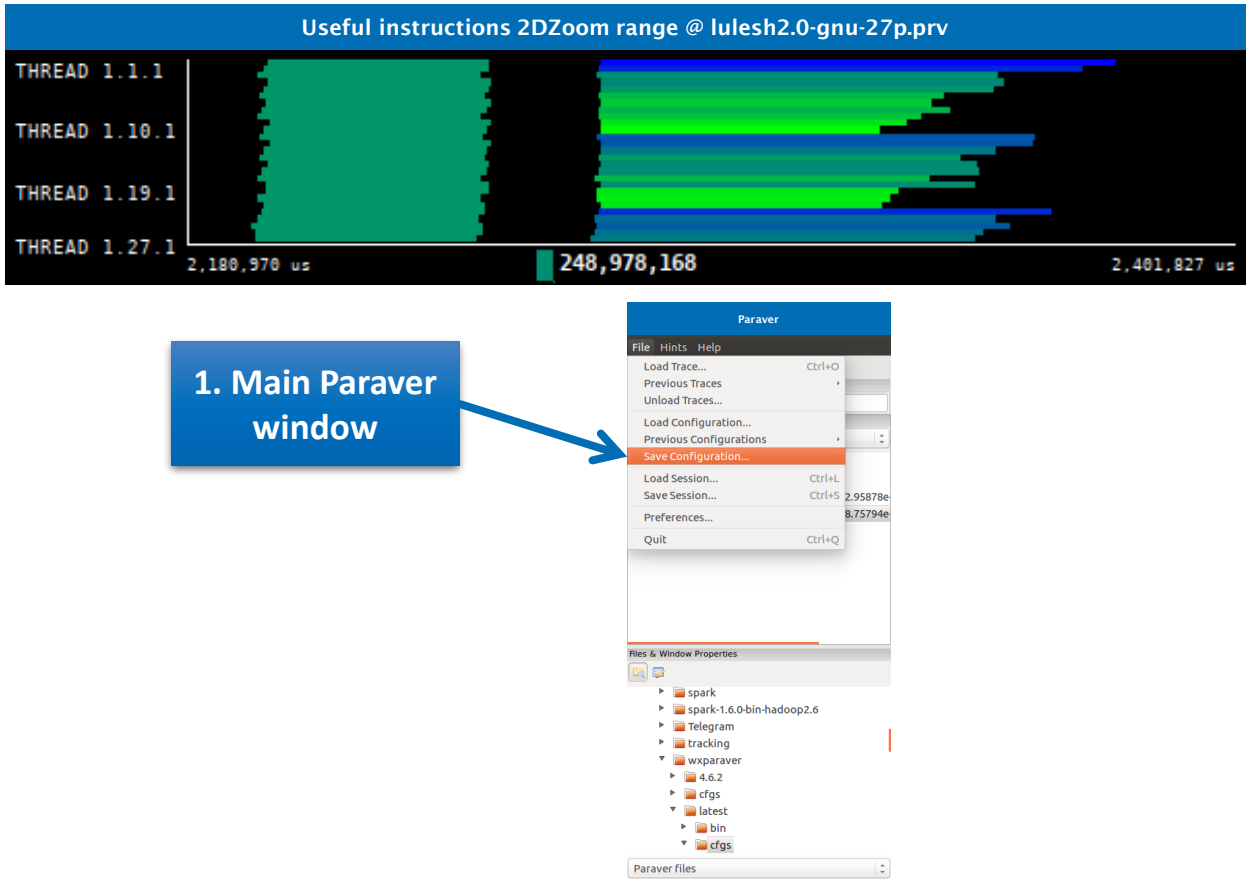
Save CFG's (method 1)

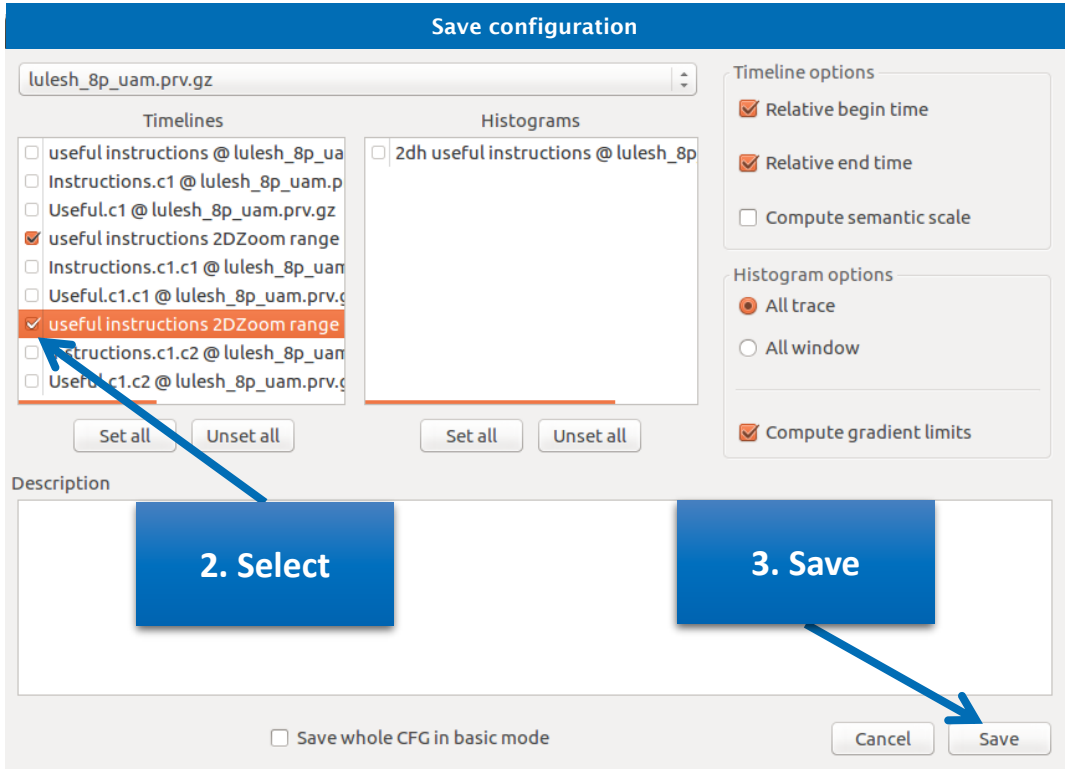


Right click on timeline



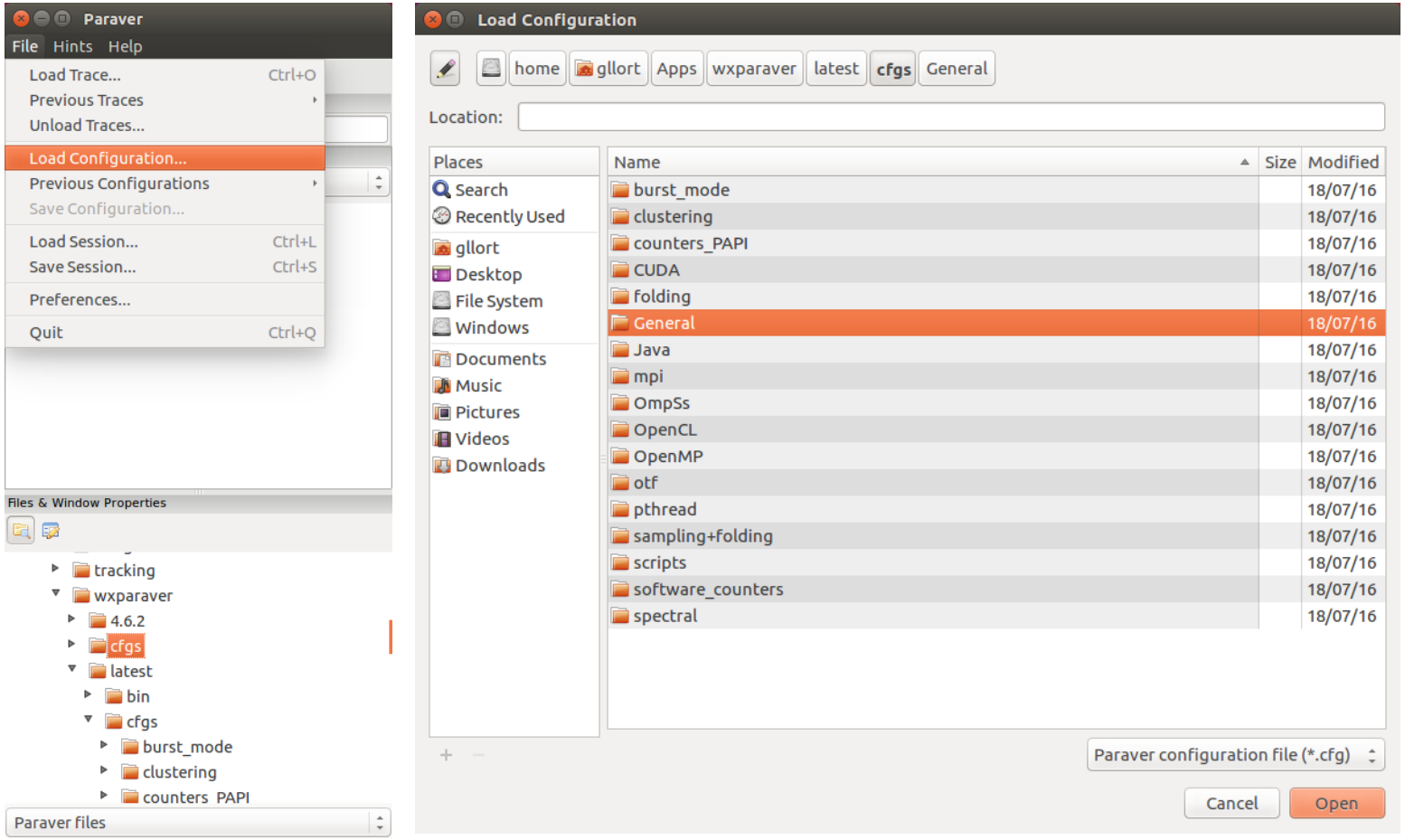
Save CFG's (method 2)





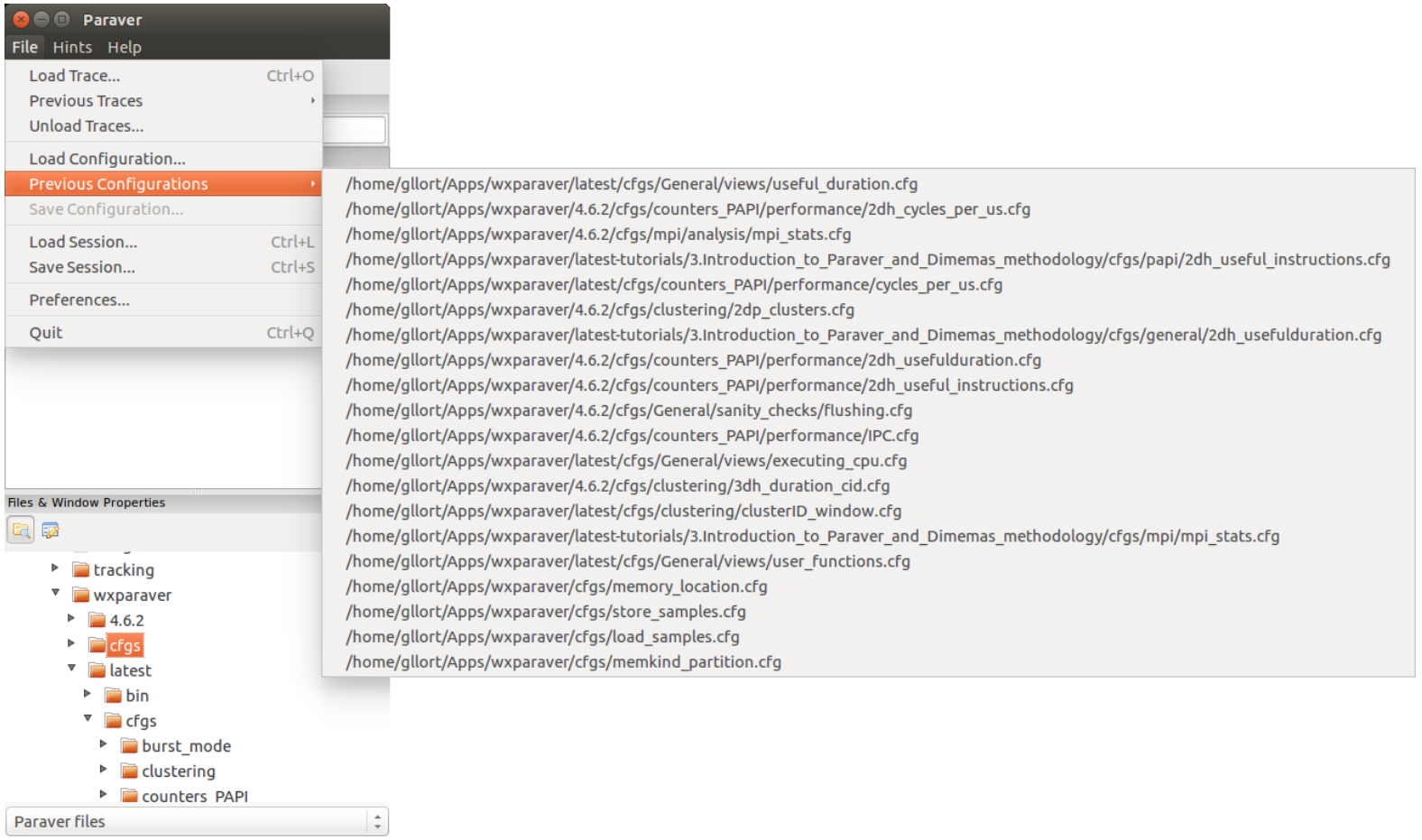
CFG's distribution

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



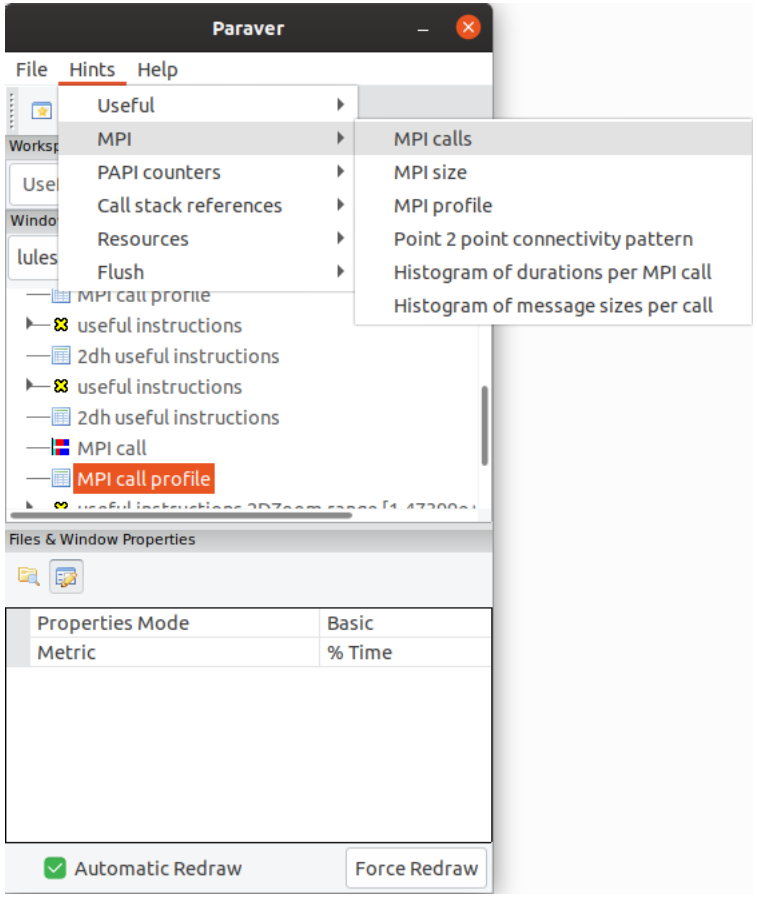
CFG's distribution

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



Hints: a good place to start!

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



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

`$EXTRAE_HOME/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?

Cluster-based analysis

Use clustering analysis

- Run clustering

```
archer> cd $HOME/tools-material/clustering
archer> $HOME/clustering/bin/BurstClustering \
        -d cluster.xml \
        -i ../extrae/lulesh2.0-gnu-27p.prv \
        -o lulesh2.0-gnu-27p_clustered.prv
```

- If you didn’t get your own trace, use a prepared one from:

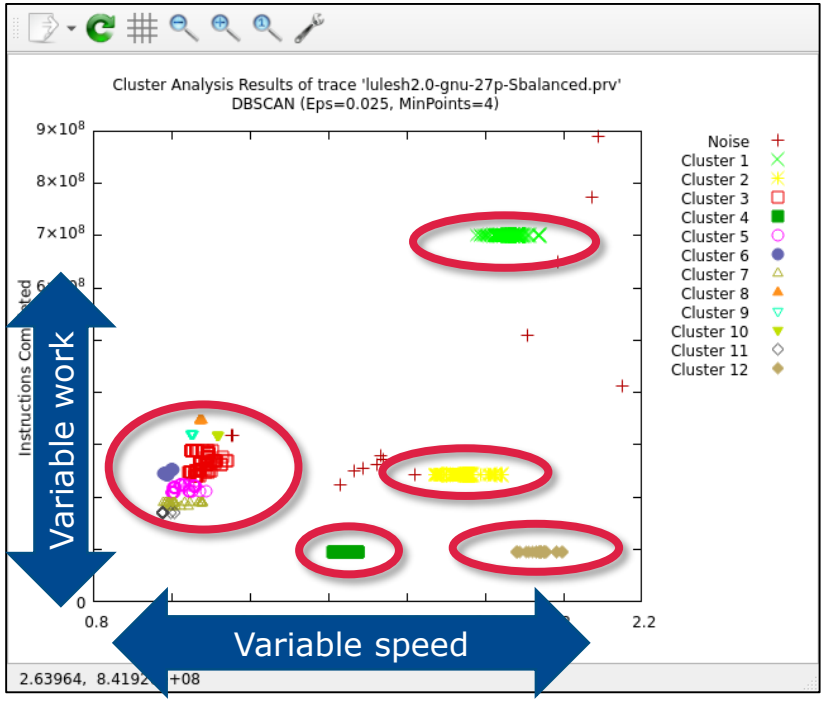
```
archer> ls $HOME/tools-material/traces/lulesh2.0-gnu-27p.prv
```

Cluster-based analysis

- Check the resulting scatter plot

```
laptop> gnuplot lulesh2.0_27p_clustered.IPC.PAPI_TOT_INS.gnuplot
```

- Identify main computing trends
- Work (Y) vs. Speed (X)
- Look at the clusters shape
  - Variability in both axes indicate **potential imbalances**

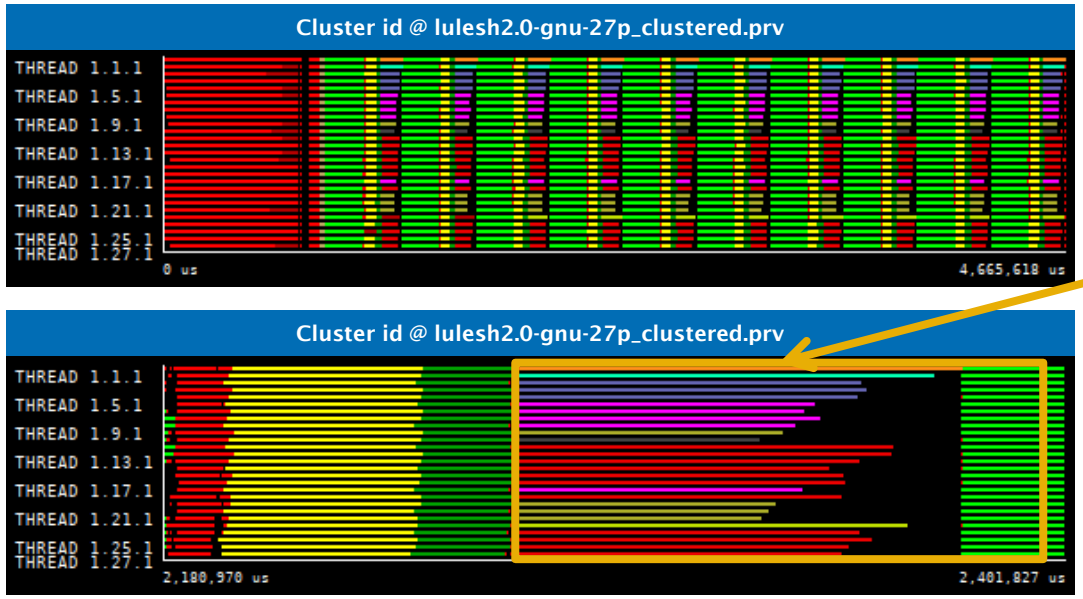


Correlating scatter plot and time distribution

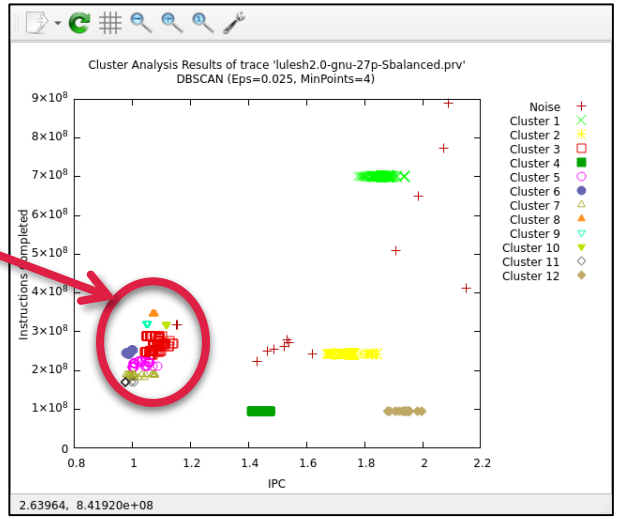
- Open the clustered trace with Paraver and look at it

```
laptop> $HOME/paraver/bin/wxparaver <path-to>/lulesh2.0-gnu-27p_clustered.prv
```

- Display the distribution of clusters over time
  - File → Load configuration → `$HOME/paraver/cfgs/clustering/clusterID_window.cfg`



Variable work / speed + Simultaneously @ different processes = Imbalances



BSC Tools
Hands-On

Lau Mercadal
(tools@bsc.es)
Barcelona Supercomputing Center