BSC Tools
Hands-On

Germán Llort, Judit Giménez
(tools@bsc.es)
Barcelona Supercomputing Center

Installing Paraver in your computer

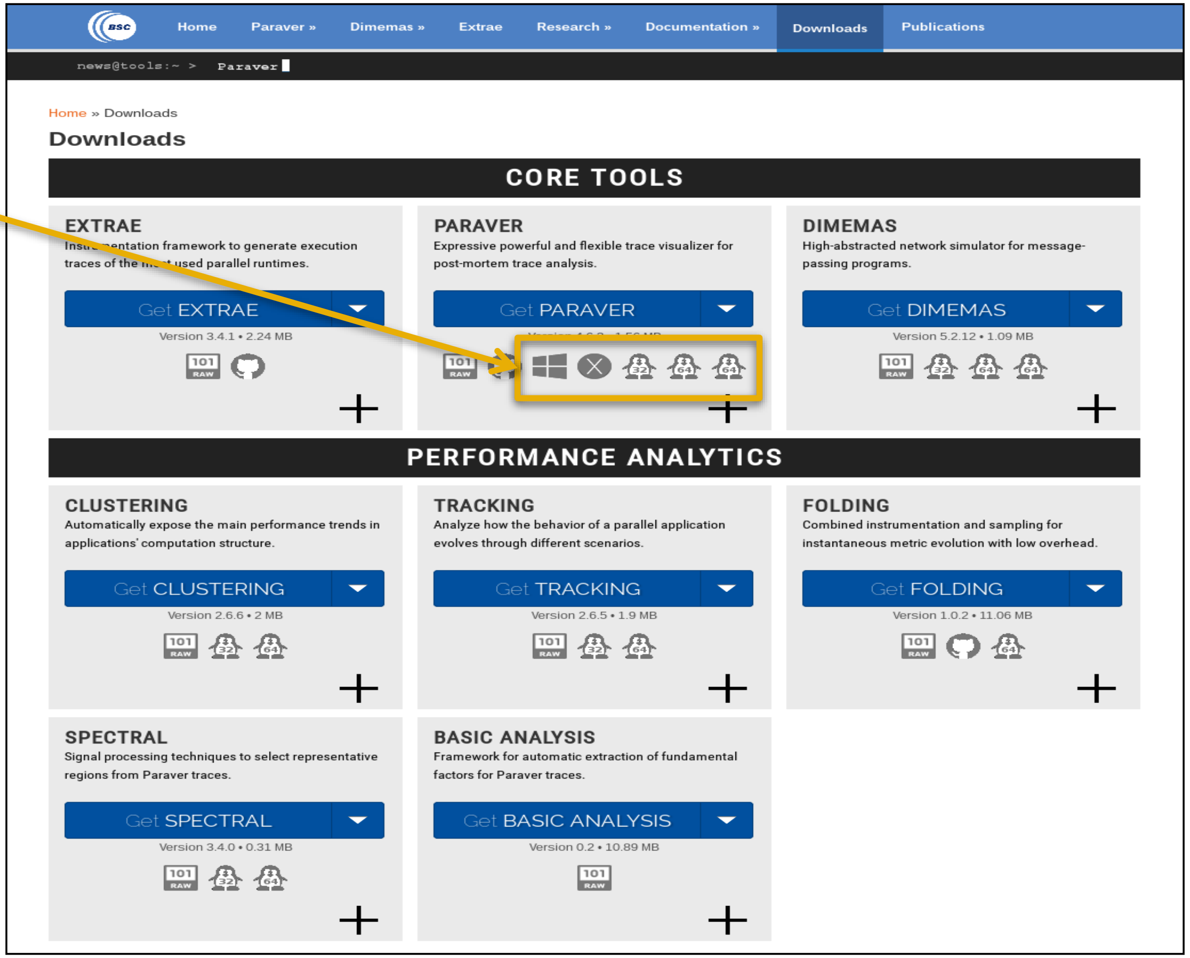
Install Paraver in your laptop

- Download a binary for your OS
  - https://tools.bsc.es/downloads

wxparaver-4.8.2-win.zip

wxparaver-4.8.2-mac.zip

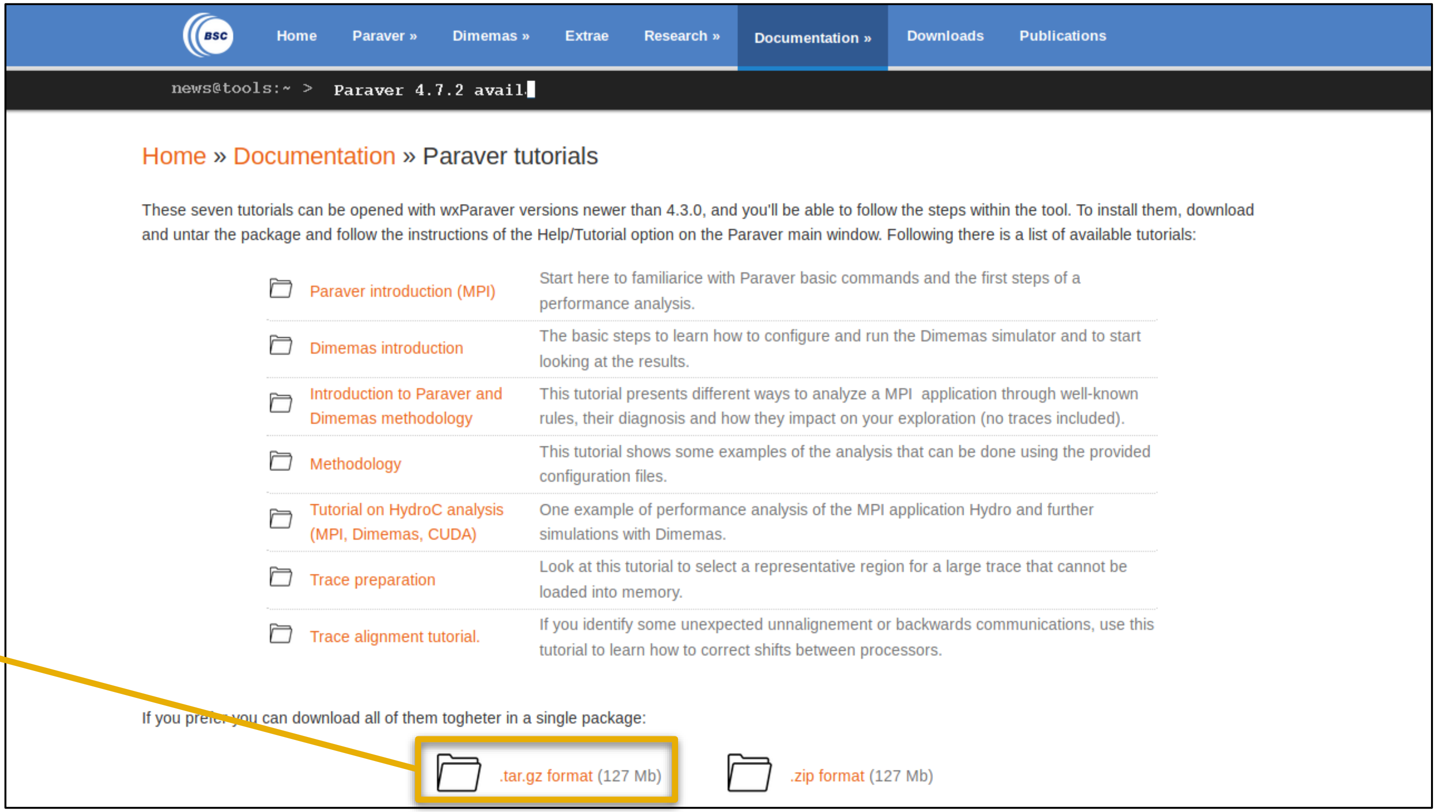
wxparaver-4.8.2-Linux_i686.tar.gz (32bit)
wxparaver-4.8.2-Linux_x86_64.tar.gz (64bit)



Install Paraver tutorials

- Download tutorials archive
  - https://tools.bsc.es/paraver-tutorials

All tutorials



Install Paraver (III)

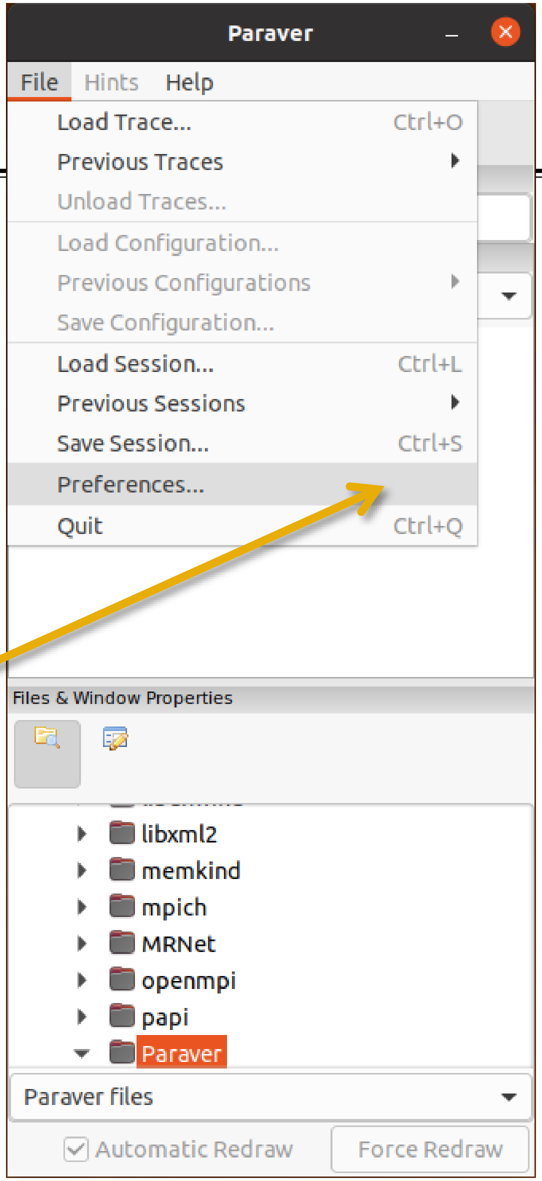
- Uncompress downloaded packages
- Rename the folders:
  - wxparaver-4.8.2-* → paraver
  - paraver-tutorials-20150526 → tutorials
- Start Paraver locally:
  - Linux: Run the command:

```
laptop$ paraver/bin/wxparaver
```

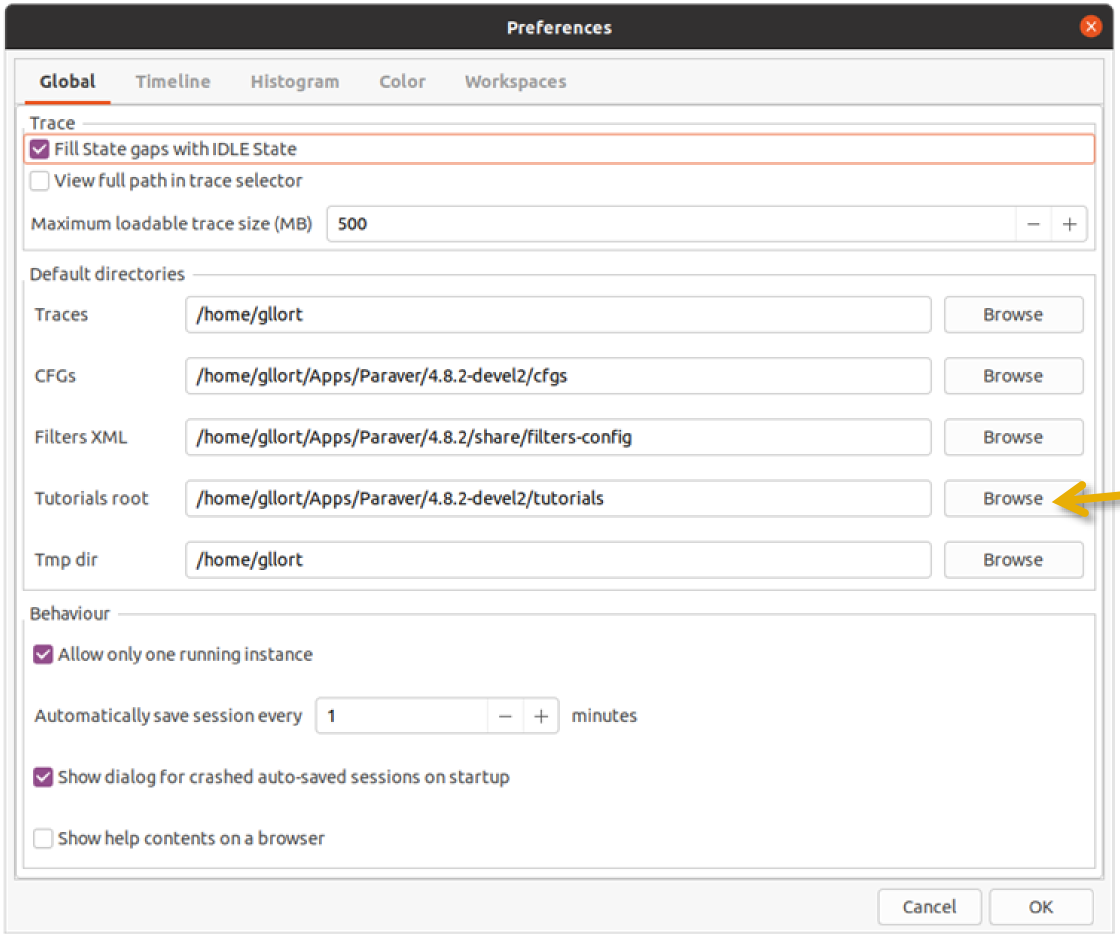
  - Windows: Double-click on paraver/wxparaver.exe
  - MAC: Double click on paraver/wxparaver.app
- Any issue? Start Paraver in HAWK:

```
hawk> module load bsc_tools
hawk> wxparaver &
```

- Open File → Preferences



Install Paraver (IV)

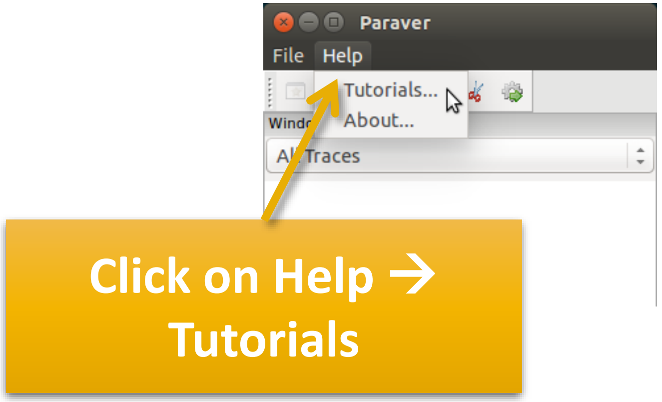


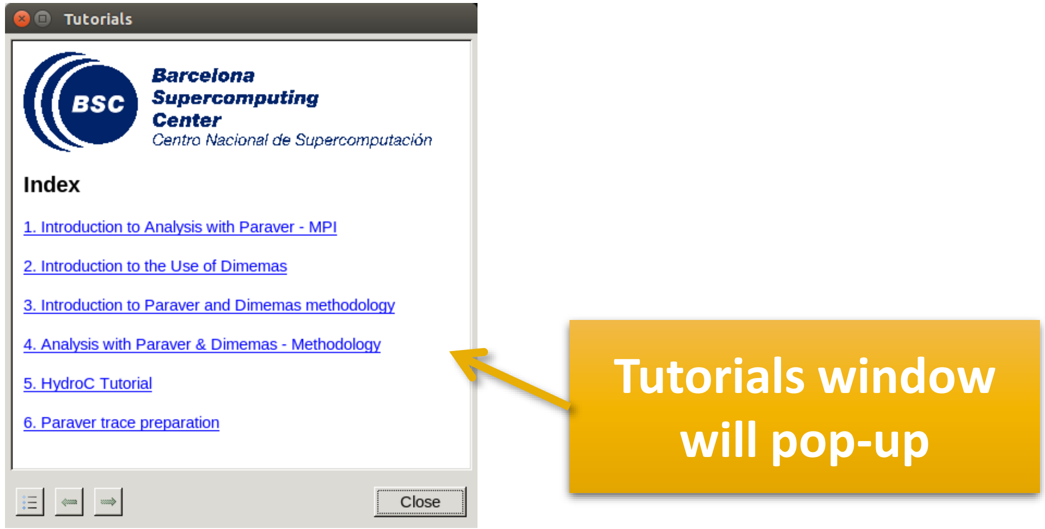
- Setup the "Tutorials root" pointing to your folder "tutorials"

Click Browse and select your folder "tutorials"

Install Paraver (IV)

- Check tutorials are properly installed



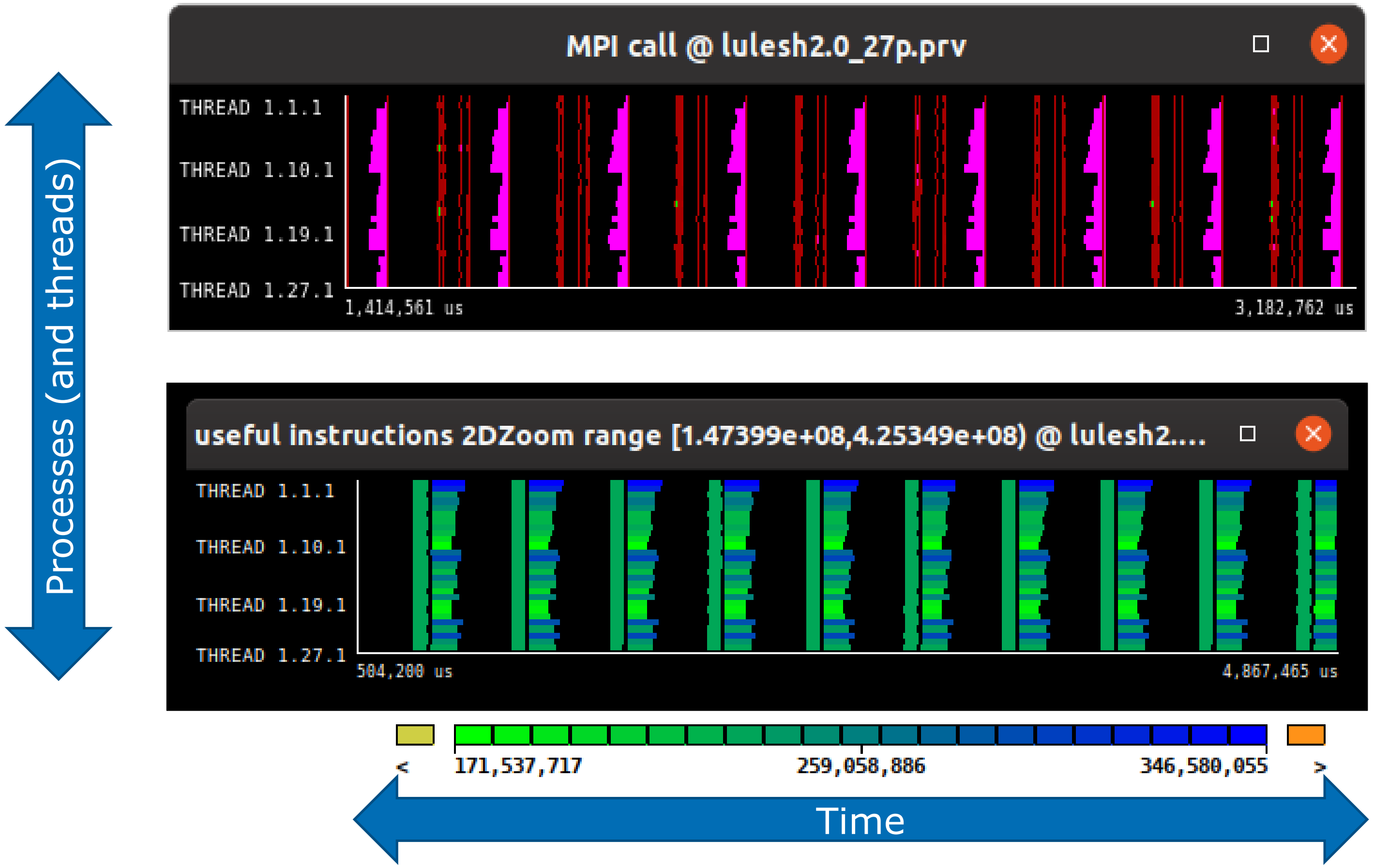


- Follow these tutorials by clicking on the hyperlinks and reading the explanations. When you click on a link, one or more views will open.

Brief introduction to Paraver

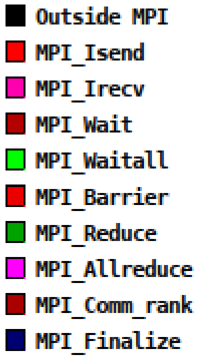
3 main views of Paraver (I)

- Timeline



Code color
(e.g. 1 color for each MPI call)

Gradient color
(e.g. from low #instructions to high #instructions)



3 main views of Paraver (II)

- Table (Profile)

Categories (e.g. MPI calls)

Processes (and threads)

MPI call profile @ lulesh2.0_27p.prv

	Outside MPI	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Barrier	MPI_Reduce	MPI
THREAD 1.1.1	99.04 %	0.05 %	0.06 %	0.35 %	0.30 %	0.03 %	0.00 %	
THREAD 1.2.1	97.37 %	0.07 %	0.08 %	0.20 %	0.82 %	0.03 %	0.00 %	
THREAD 1.3.1	93.79 %	0.05 %	0.05 %	0.22 %	0.52 %	0.03 %	0.02 %	
THREAD 1.4.1	93.93 %	0.07 %	0.08 %	0.17 %	0.61 %	0.03 %	0.00 %	
THREAD 1.5.1	93.75 %	0.11 %	0.11 %	0.38 %	0.19 %	0.01 %	0.00 %	
THREAD 1.6.1	91.64 %	0.08 %	0.08 %	0.10 %	0.74 %	0.02 %	0.00 %	
THREAD 1.7.1	91.24 %	0.06 %	0.05 %	0.16 %	0.42 %	0.03 %	0.11 %	
THREAD 1.8.1	91.93 %	0.08 %	0.08 %	0.16 %	0.76 %	0.03 %	0.00 %	
THREAD 1.9.1	91.20 %	0.06 %	0.05 %	0.14 %	0.59 %	0.02 %	0.50 %	
THREAD 1.10.1	90.47 %	0.08 %	0.07 %	0.33 %	0.37 %	0.03 %	0.00 %	
THREAD 1.11.1	89.19 %	0.12 %	0.11 %	0.35 %	0.30 %	0.01 %	0.70 %	
THREAD 1.12.1	95.80 %	0.09 %	0.07 %	0.19 %	0.83 %	0.03 %	0.00 %	
THREAD 1.13.1	96.04 %	0.12 %	0.10 %	0.41 %	0.33 %	0.01 %	0.00 %	
THREAD 1.14.1	94.61 %	0.18 %	0.15 %	0.15 %	0.35 %	0.00 %	0.00 %	
THREAD 1.15.1	93.28 %	0.13 %	0.10 %	0.10 %	1.21 %	0.01 %	0.00 %	
THREAD 1.16.1	91.67 %	0.09 %	0.07 %	0.26 %	2.01 %	0.03 %	0.00 %	
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	

Summary

The table can display a variety of statistics (e.g. % of time, # of calls, etc.) with gradient coloring showing from low values to high values

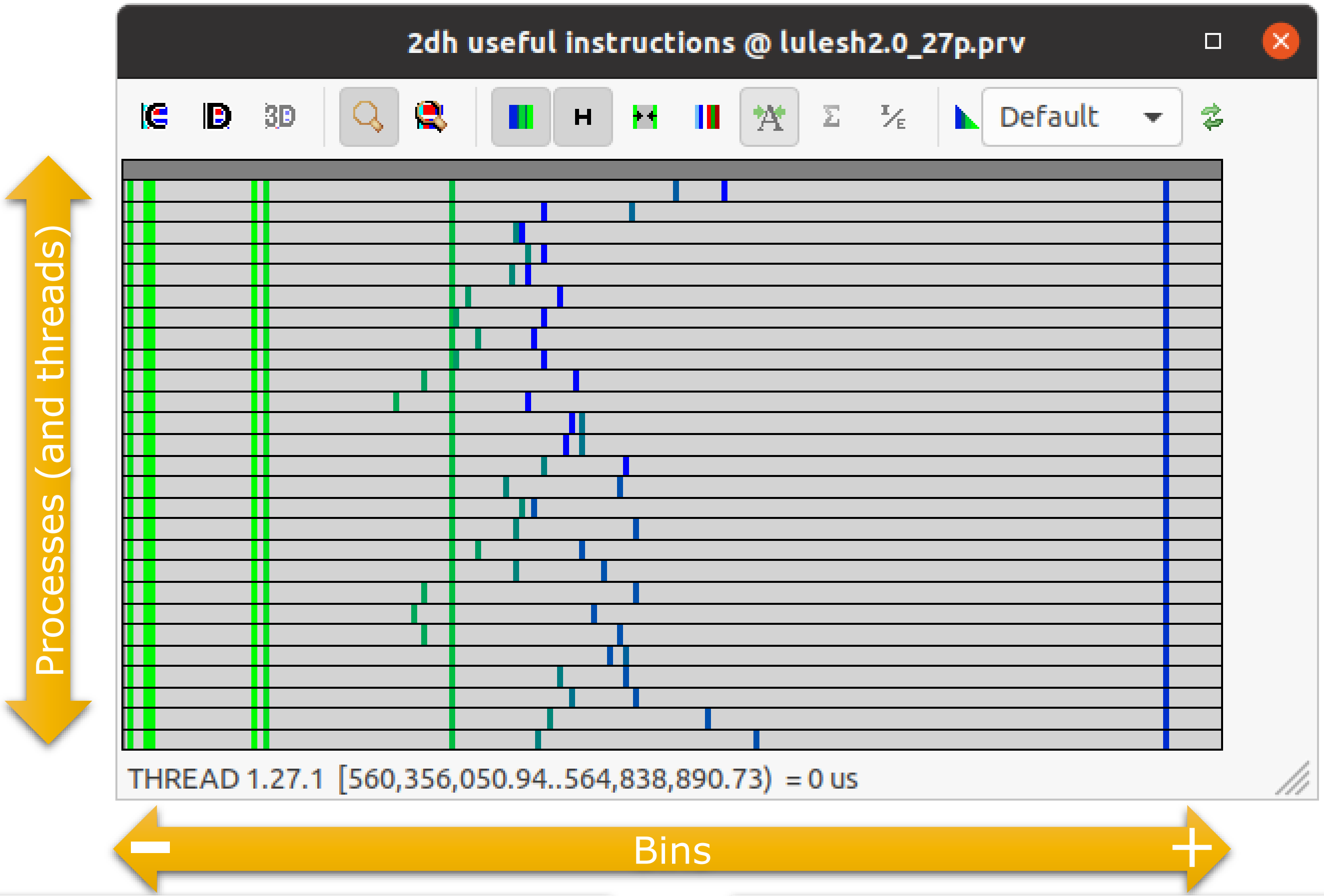
3 main views of Paraver (III)

- Histogram

Displays continuous metrics (e.g. **instructions executed**, duration of computations, bytes sent/received, etc.)

Gradient color represents if the selected statistic (% of time, # of calls, etc.) for each behavior is high or low

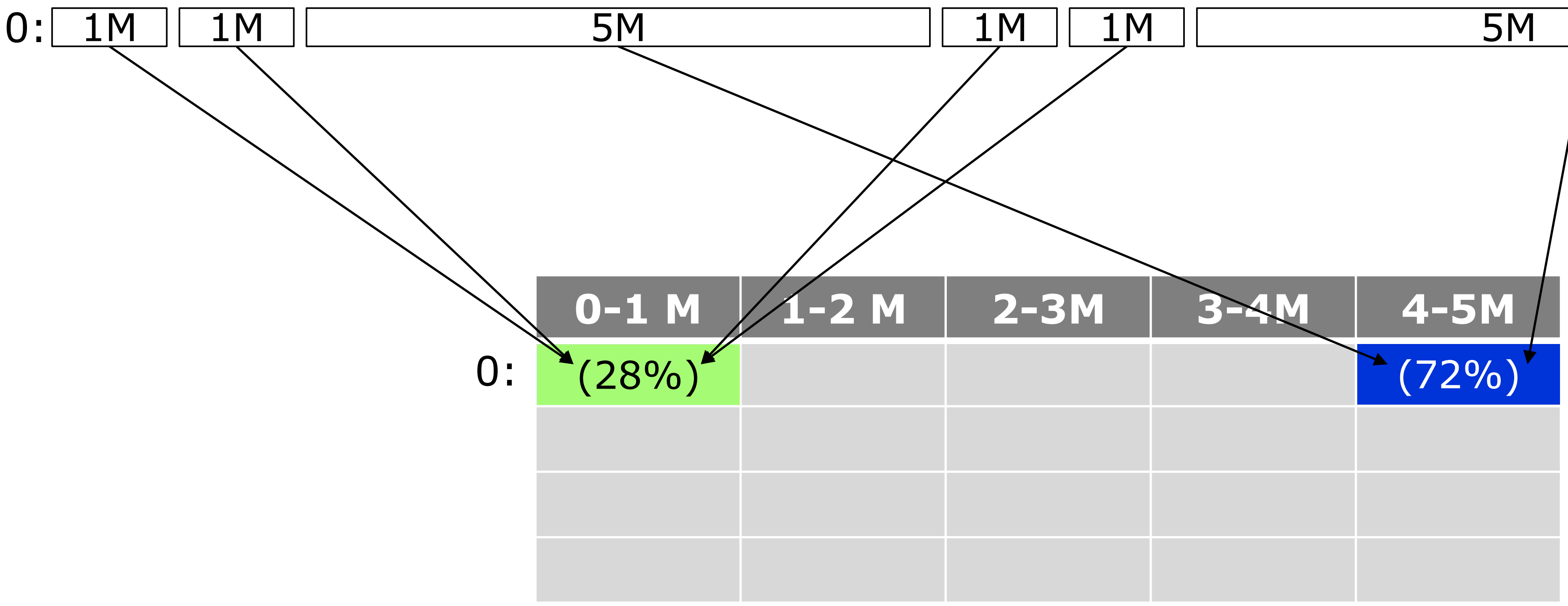
General tip: straight lines are good (all processes show same behavior), while variabilities usually indicate imbalances



Data on the left ➔ Program records low values of the metric

Data on the right ➔ Program records high values of the metric

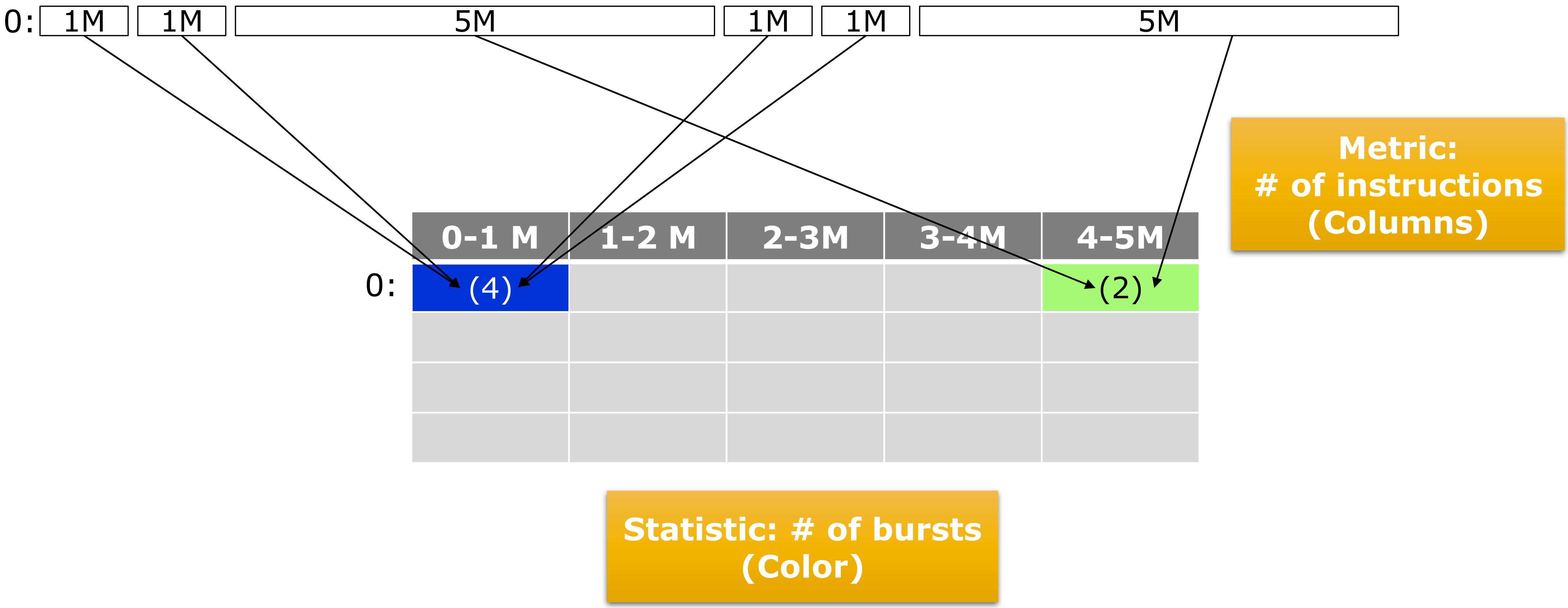
Histogram: position, metric and statistic … wait, what!?



Metric:
# of instructions
(Columns)

Statistic: % of time
(Color)

Histogram: position, color, metric and statistic … wait, what!?



Histogram: position, color, metric and statistic … wait, what!?

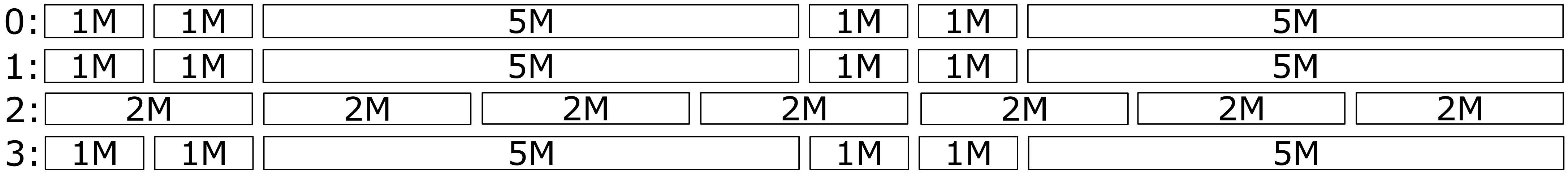
0:	1M	1M	5M	1M	1M	5M
1:	1M	1M	5M	1M	1M	5M
2:	1M	1M	5M	1M	1M	5M
3:	1M	1M	5M	1M	1M	5M

	0-1 M	1-2 M	2-3M	3-4M	4-5M
0:	(4)				(2)
1:	(4)				(2)
2:	(4)				(2)
3:	(4)				(2)

Metric:
# of instructions
(Columns)

Statistic: # of bursts
(Color)

Histogram: position, color, metric and statistic … wait, what!?



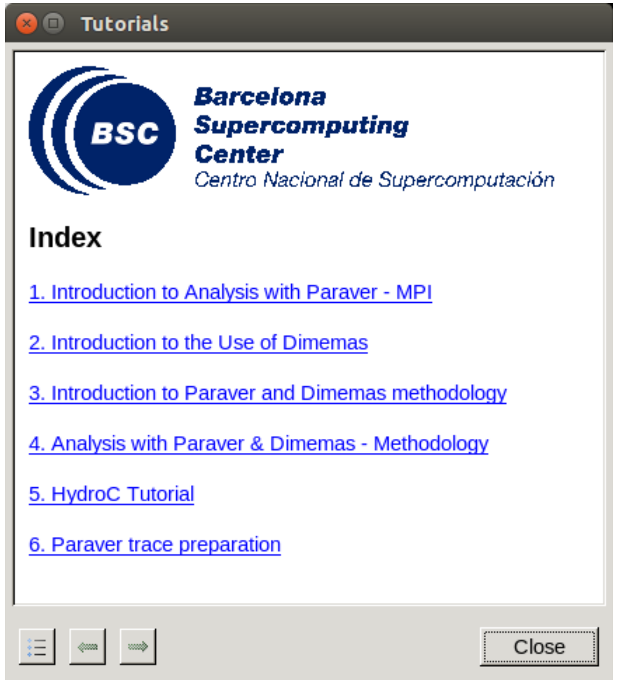
	0-1 M	1-2 M	2-3M	3-4M	4-5M
0:	(4)				(2)
1:	(4)				(2)
2:			(7)		(2)
3:	(4)				(2)

Metric: # of instructions (Columns)

Statistic: # of bursts (Color)

First steps with Paraver

- Follow tutorial number…
  - 1 → Explains basic navigation with the tool
  - 3 → Basic analysis methodology (first 4 bullets, Clustering and Dimemas part not covered today!)
  - 5 → Analysis methodology applied to a real application



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

No need to recompile nor relink!

- User events (Extrae API)

Extrae Overheads

	Average values
Recording 1 Event	150 – 200ns
1 Event + PAPI hardware counters	750 – 1000ns
1 Event + Callstack (1 level)	1µs
1 Event + Callstack (6 levels)	2µs

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 HAWK

- Log-in to HAWK:

```
laptop$ ssh -Y <USER>@hawk.hww.hlrs.de
```

- Extrae is available via modules for 2 MPI versions… choose yours!
  - Cray MPT (default)

```
hawk$ module load mpt
hawk$ module load extrae
```

  - OpenMPI

```
hawk$ module load openmpi
hawk$ module load extrae
```

Getting your first trace

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

```
hawk$ cp -r /lustre/cray/ws9/2/ws/hpcjgrac-tw35/BSC/tools-material $HOME
```

- Provided folder **`tools-material`** in `/lustre/cray/ws9/2/ws/hpcjgrac-tw35/BSC` contains:
  - Test application compiled for OpenMPI (`lulesh2.0_openmpi`)
  - 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`)
  - A copy of this slides

Using Extrae in 3 steps

1. **Adapt** your job submission scripts

2. **Configure** what to trace
   - XML configuration file
   - More 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/sh

#PBS -l walltime=00:10:00
#PBS -l select=1:node_type=rome:mpiprocs=27
#PBS -j oe

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

# Load required modules
module load openmpi



# Run on 27 processes
mpirun -np 27 ../apps/lulesh2.0_openmpi -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/sh

#PBS -l walltime=00:10:00
#PBS -l select=1:node_type=rome:mpiprocs=27
#PBS -j oe

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

# Load required modules
module load openmpi

export TRACE_NAME=lulesh2.0_openmpi_27p.prv

# Run on 27 processes
mpirun -np 27 ./trace.sh ../apps/lulesh2.0_openmpi
  -i 10 -s 65 -p
```

Specify name of output trace (optional)

Run with Extrae

Step 1: Adapt the job script to load Extrae

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

```
!/bin/sh

#PBS -l walltime=00:10:00
#PBS -l select=1:node_type=rome:mpiprocs=27
#PBS -j oe

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

# Load required modules
module load openmpi

export TRACE_NAME=lulesh2.0_openmpi_27p.prv

# Run on 27 processes
mpirun -np 27 ./trace.sh ./apps/lulesh2.0_openmpi
  -i 10 -s 65 -p
```

```
#!/bin/sh

# Extrae configuration file
export EXTRAE_CONFIG_FILE=./extrae.xml

# Select tracing library (depends on runtime & lang)

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

# Run the application
$*
```

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 – most relevant settings

```
<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 – most relevant settings (II)

```
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="5s">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM
    </set>
    <set enabled="yes" domain="all" changeat-time="5s">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_FP_OPS
    </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?)

```
> papi_avail
```

← List of available counters (PAPI)

```
> papi_best_set
```

← Make groups of compatible counters (Extrae)

```
<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

```
hawk$ qsub -q R_tw job.pbs
```

All done! Check your resulting trace

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

```
hawk$ ls -l
  ...
  lulesh2.0_openmpi_27p.pcf
  lulesh2.0_openmpi_27p.prv
  lulesh2.0_openmpi_27p.row
```

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

- Now let's look into it!

Analysing a trace with Paraver

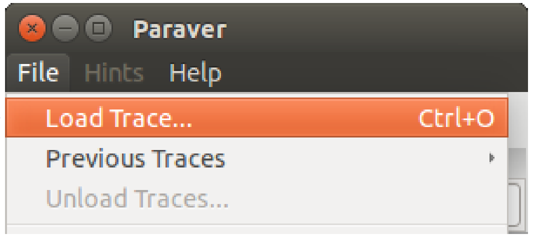
First steps of analysis

- Copy the trace to your computer (scp, WinSCP, etc.)

```
laptop$ scp <USER>@hawk.hww.hlrs.de:tools-material/extrae/*.{prv,pcf,row} .
```

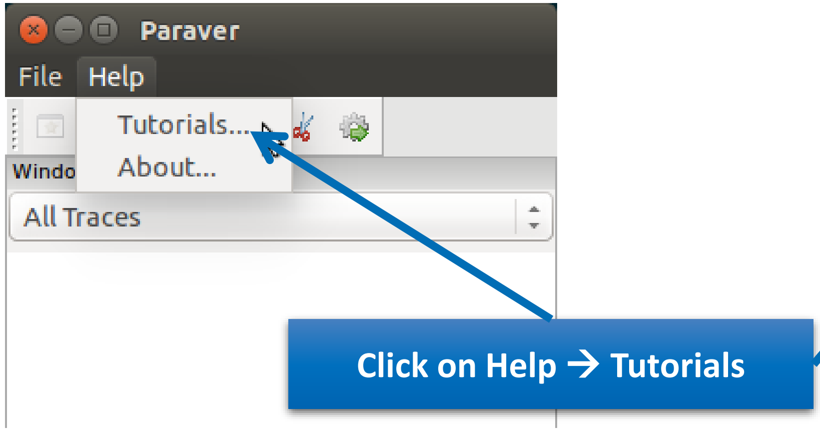
- Load the trace with Paraver

Click on File → Load Trace → Browse to "lulesh2.0_openmpi_27p.prv"



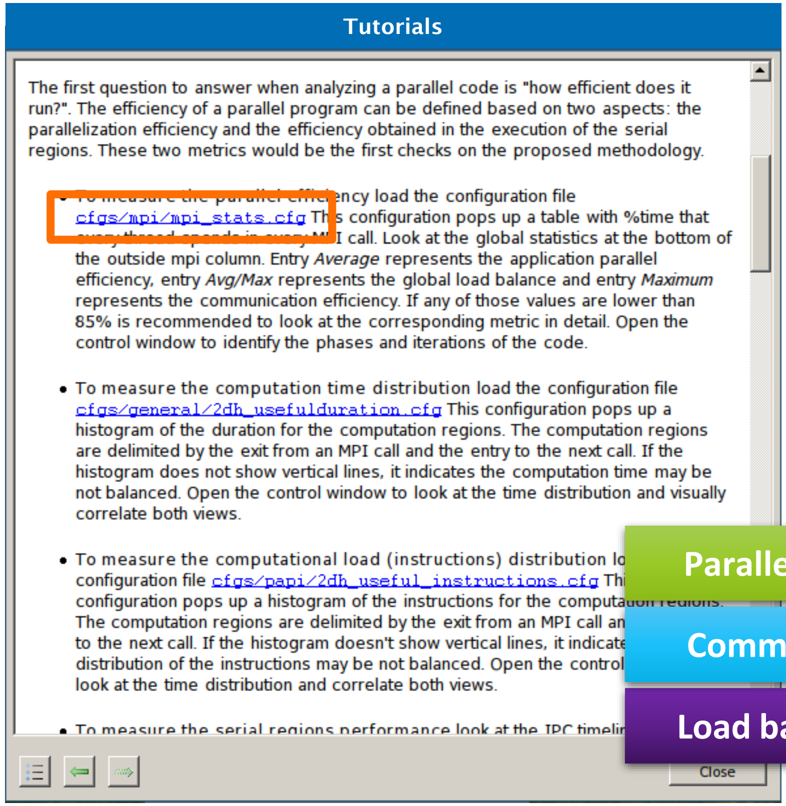
First steps of analysis

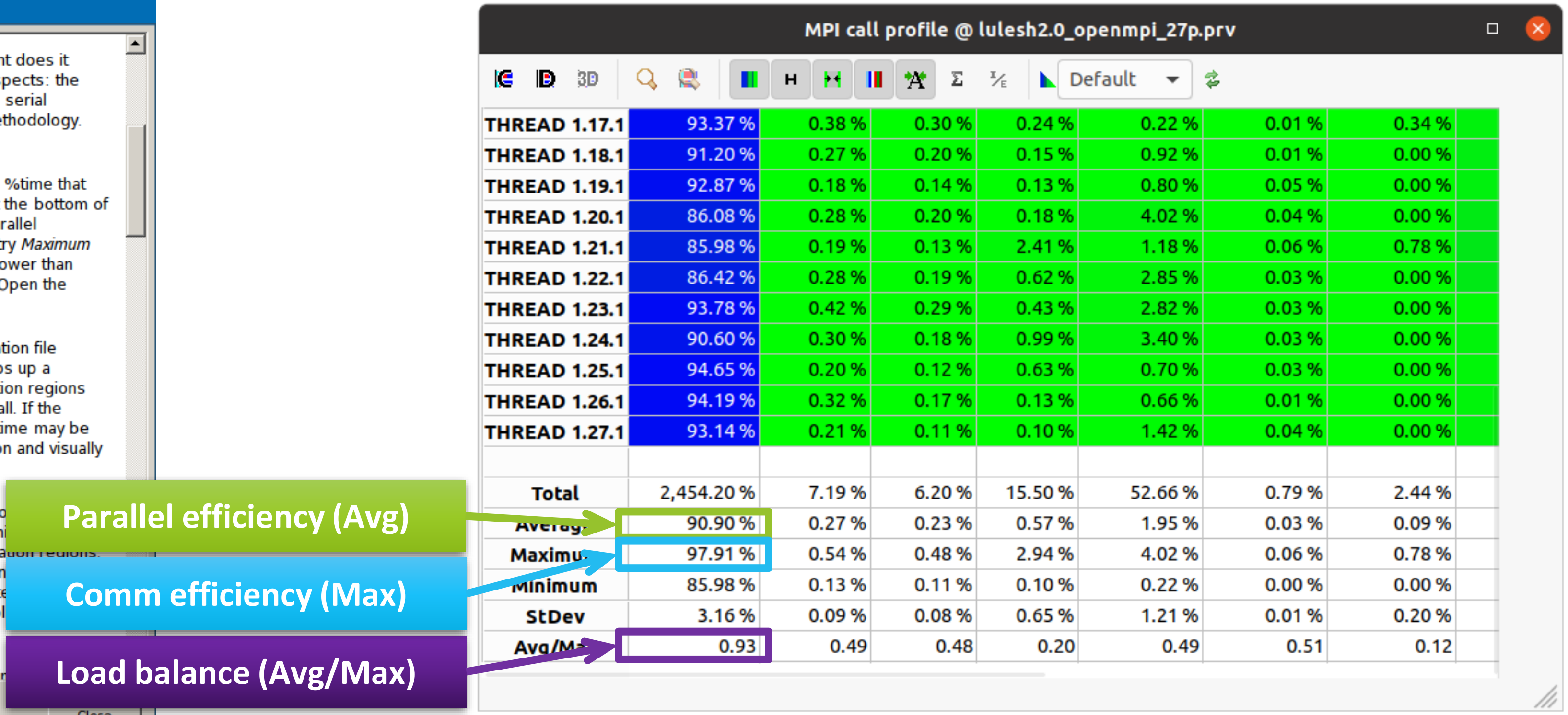
- Follow Tutorial #3
  - Introduction to Paraver and Dimemas methodology



Measure the parallel efficiency

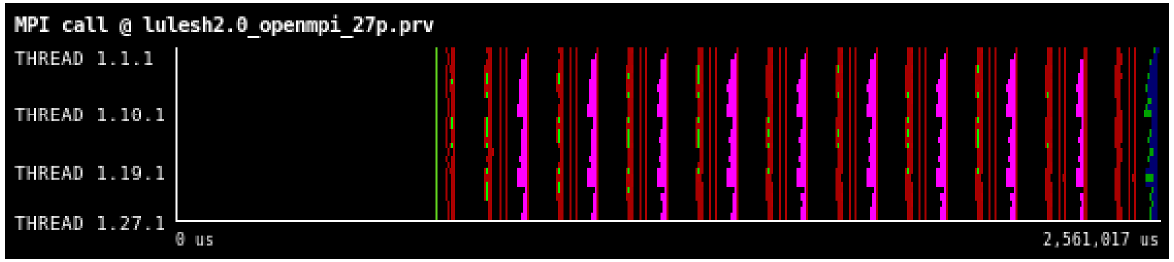
- Click on “mpi_stats.cfg”
  - Check the **Average** for the column “**Outside MPI**”



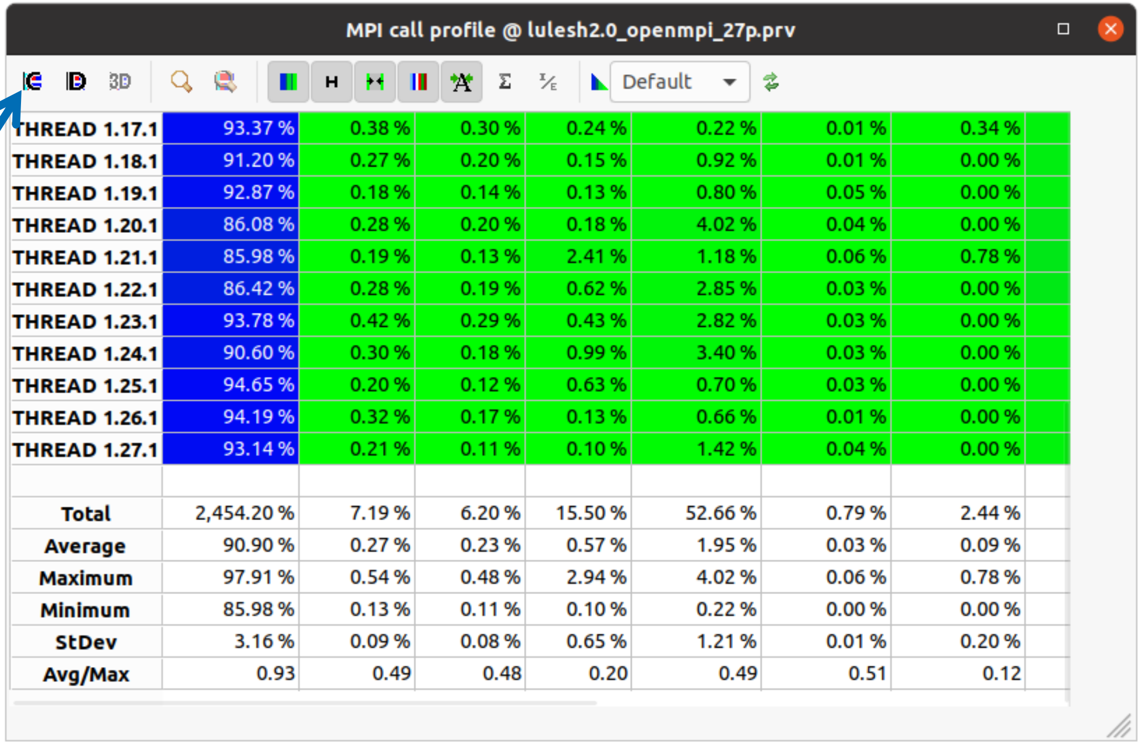


THREAD 1.17.1	93.37 %	0.38 %	0.30 %	0.24 %	0.22 %	0.01 %	0.34 %	
THREAD 1.18.1	91.20 %	0.27 %	0.20 %	0.15 %	0.92 %	0.01 %	0.00 %	
THREAD 1.19.1	92.87 %	0.18 %	0.14 %	0.13 %	0.80 %	0.05 %	0.00 %	
THREAD 1.20.1	86.08 %	0.28 %	0.20 %	0.18 %	4.02 %	0.04 %	0.00 %	
THREAD 1.21.1	85.98 %	0.19 %	0.13 %	2.41 %	1.18 %	0.06 %	0.78 %	
THREAD 1.22.1	86.42 %	0.28 %	0.19 %	0.62 %	2.85 %	0.03 %	0.00 %	
THREAD 1.23.1	93.78 %	0.42 %	0.29 %	0.43 %	2.82 %	0.03 %	0.00 %	
THREAD 1.24.1	90.60 %	0.30 %	0.18 %	0.99 %	3.40 %	0.03 %	0.00 %	
THREAD 1.25.1	94.65 %	0.20 %	0.12 %	0.63 %	0.70 %	0.03 %	0.00 %	
THREAD 1.26.1	94.19 %	0.32 %	0.17 %	0.13 %	0.66 %	0.01 %	0.00 %	
THREAD 1.27.1	93.14 %	0.21 %	0.11 %	0.10 %	1.42 %	0.04 %	0.00 %	
Total	2,454.20 %	7.19 %	6.20 %	15.50 %	52.66 %	0.79 %	2.44 %	
Averag	90.90 %	0.27 %	0.23 %	0.57 %	1.95 %	0.03 %	0.09 %	
Maximu	97.91 %	0.54 %	0.48 %	2.94 %	4.02 %	0.06 %	0.78 %	
Minimum	85.98 %	0.13 %	0.11 %	0.10 %	0.22 %	0.00 %	0.00 %	
StDev	3.16 %	0.09 %	0.08 %	0.65 %	1.21 %	0.01 %	0.20 %	
Avg/M	0.93	0.49	0.48	0.20	0.49	0.51	0.12	

Focus on the iterative part

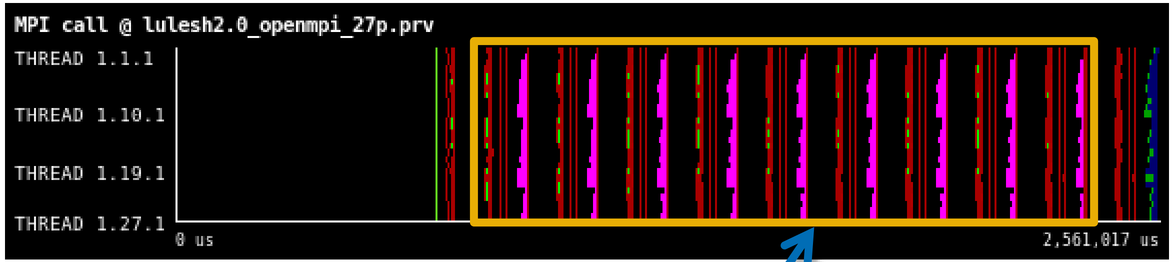


Click on “Open Control Window”



THREAD 1.17.1	93.37 %	0.38 %	0.30 %	0.24 %	0.22 %	0.01 %	0.34 %
THREAD 1.18.1	91.20 %	0.27 %	0.20 %	0.15 %	0.92 %	0.01 %	0.00 %
THREAD 1.19.1	92.87 %	0.18 %	0.14 %	0.13 %	0.80 %	0.05 %	0.00 %
THREAD 1.20.1	86.08 %	0.28 %	0.20 %	0.18 %	4.02 %	0.04 %	0.00 %
THREAD 1.21.1	85.98 %	0.19 %	0.13 %	2.41 %	1.18 %	0.06 %	0.78 %
THREAD 1.22.1	86.42 %	0.28 %	0.19 %	0.62 %	2.85 %	0.03 %	0.00 %
THREAD 1.23.1	93.78 %	0.42 %	0.29 %	0.43 %	2.82 %	0.03 %	0.00 %
THREAD 1.24.1	90.60 %	0.30 %	0.18 %	0.99 %	3.40 %	0.03 %	0.00 %
THREAD 1.25.1	94.65 %	0.20 %	0.12 %	0.63 %	0.70 %	0.03 %	0.00 %
THREAD 1.26.1	94.19 %	0.32 %	0.17 %	0.13 %	0.66 %	0.01 %	0.00 %
THREAD 1.27.1	93.14 %	0.21 %	0.11 %	0.10 %	1.42 %	0.04 %	0.00 %
Total	2,454.20 %	7.19 %	6.20 %	15.50 %	52.66 %	0.79 %	2.44 %
Average	90.90 %	0.27 %	0.23 %	0.57 %	1.95 %	0.03 %	0.09 %
Maximum	97.91 %	0.54 %	0.48 %	2.94 %	4.02 %	0.06 %	0.78 %
Minimum	85.98 %	0.13 %	0.11 %	0.10 %	0.22 %	0.00 %	0.00 %
StDev	3.16 %	0.09 %	0.08 %	0.65 %	1.21 %	0.01 %	0.20 %
Avg/Max	0.93	0.49	0.48	0.20	0.49	0.51	0.12

Focus on the iterative part



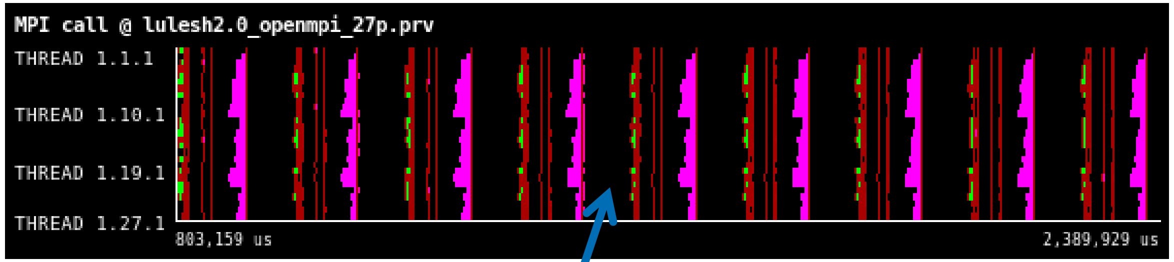
Drag & drop on this area to zoom on the iterative region

MPI call profile @ lulesh2.0_openmpi_27p.prv

Default

THREAD 1.17.1	93.37 %	0.38 %	0.30 %	0.24 %	0.22 %	0.01 %	0.34 %
THREAD 1.18.1	91.20 %	0.27 %	0.20 %	0.15 %	0.92 %	0.01 %	0.00 %
THREAD 1.19.1	92.87 %	0.18 %	0.14 %	0.13 %	0.80 %	0.05 %	0.00 %
THREAD 1.20.1	86.08 %	0.28 %	0.20 %	0.18 %	4.02 %	0.04 %	0.00 %
THREAD 1.21.1	85.98 %	0.19 %	0.13 %	2.41 %	1.18 %	0.06 %	0.78 %
THREAD 1.22.1	86.42 %	0.28 %	0.19 %	0.62 %	2.85 %	0.03 %	0.00 %
THREAD 1.23.1	93.78 %	0.42 %	0.29 %	0.43 %	2.82 %	0.03 %	0.00 %
THREAD 1.24.1	90.60 %	0.30 %	0.18 %	0.99 %	3.40 %	0.03 %	0.00 %
THREAD 1.25.1	94.65 %	0.20 %	0.12 %	0.63 %	0.70 %	0.03 %	0.00 %
THREAD 1.26.1	94.19 %	0.32 %	0.17 %	0.13 %	0.66 %	0.01 %	0.00 %
THREAD 1.27.1	93.14 %	0.21 %	0.11 %	0.10 %	1.42 %	0.04 %	0.00 %
Total	2,454.20 %	7.19 %	6.20 %	15.50 %	52.66 %	0.79 %	2.44 %
Average	90.90 %	0.27 %	0.23 %	0.57 %	1.95 %	0.03 %	0.09 %
Maximum	97.91 %	0.54 %	0.48 %	2.94 %	4.02 %	0.06 %	0.78 %
Minimum	85.98 %	0.13 %	0.11 %	0.10 %	0.22 %	0.00 %	0.00 %
StDev	3.16 %	0.09 %	0.08 %	0.65 %	1.21 %	0.01 %	0.20 %
Avg/Max	0.93	0.49	0.48	0.20	0.49	0.51	0.12

Recalculate efficiency of iterative region



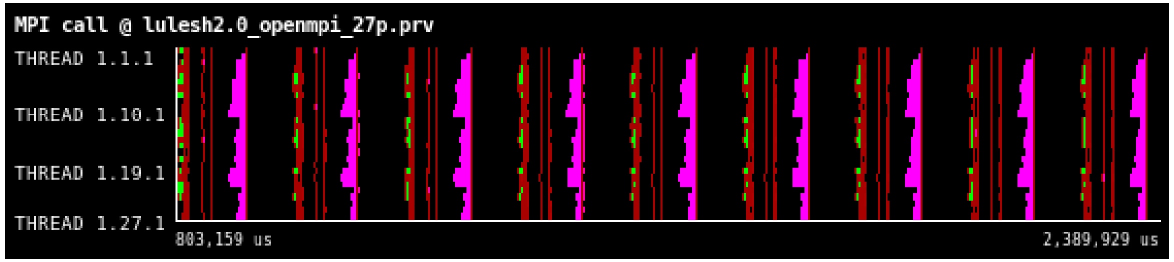
Right click → Copy

MPI call profile @ lulesh2.0_openmpi_27p.prv

Default

THREAD 1.17.1	93.37 %	0.38 %	0.30 %	0.24 %	0.22 %	0.01 %	0.34 %
THREAD 1.18.1	91.20 %	0.27 %	0.20 %	0.15 %	0.92 %	0.01 %	0.00 %
THREAD 1.19.1	92.87 %	0.18 %	0.14 %	0.13 %	0.80 %	0.05 %	0.00 %
THREAD 1.20.1	86.08 %	0.28 %	0.20 %	0.18 %	4.02 %	0.04 %	0.00 %
THREAD 1.21.1	85.98 %	0.19 %	0.13 %	2.41 %	1.18 %	0.06 %	0.78 %
THREAD 1.22.1	86.42 %	0.28 %	0.19 %	0.62 %	2.85 %	0.03 %	0.00 %
THREAD 1.23.1	93.78 %	0.42 %	0.29 %	0.43 %	2.82 %	0.03 %	0.00 %
THREAD 1.24.1	90.60 %	0.30 %	0.18 %	0.99 %	3.40 %	0.03 %	0.00 %
THREAD 1.25.1	94.65 %	0.20 %	0.12 %	0.63 %	0.70 %	0.03 %	0.00 %
THREAD 1.26.1	94.19 %	0.32 %	0.17 %	0.13 %	0.66 %	0.01 %	0.00 %
THREAD 1.27.1	93.14 %	0.21 %	0.11 %	0.10 %	1.42 %	0.04 %	0.00 %
Total	2,454.20 %	7.19 %	6.20 %	15.50 %	52.66 %	0.79 %	2.44 %
Average	90.90 %	0.27 %	0.23 %	0.57 %	1.95 %	0.03 %	0.09 %
Maximum	97.91 %	0.54 %	0.48 %	2.94 %	4.02 %	0.06 %	0.78 %
Minimum	85.98 %	0.13 %	0.11 %	0.10 %	0.22 %	0.00 %	0.00 %
StDev	3.16 %	0.09 %	0.08 %	0.65 %	1.21 %	0.01 %	0.20 %
Avg/Max	0.93	0.49	0.48	0.20	0.49	0.51	0.12

Recalculate efficiency of iterative region



Right click → Paste → Time

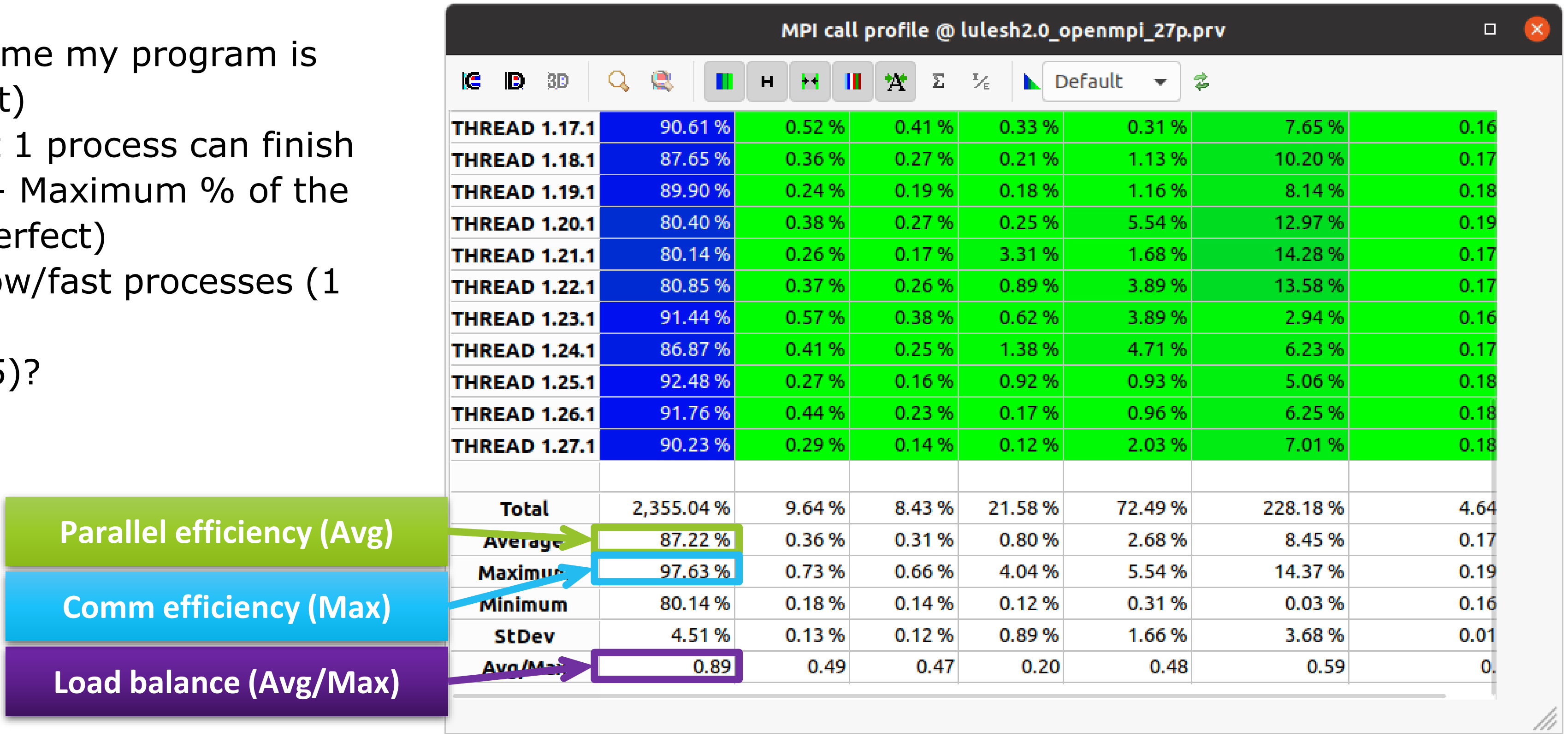
MPI call profile @ lulesh2.0_openmpi_27p.prv

Default

THREAD 1.17.1	93.37 %	0.38 %	0.30 %	0.24 %	0.22 %	0.01 %	0.34 %
THREAD 1.18.1	91.20 %	0.27 %	0.20 %	0.15 %	0.92 %	0.01 %	0.00 %
THREAD 1.19.1	92.87 %	0.18 %	0.14 %	0.13 %	0.80 %	0.05 %	0.00 %
THREAD 1.20.1	86.08 %	0.28 %	0.20 %	0.18 %	4.02 %	0.04 %	0.00 %
THREAD 1.21.1	85.98 %	0.19 %	0.13 %	2.41 %	1.18 %	0.06 %	0.78 %
THREAD 1.22.1	86.42 %	0.28 %	0.19 %	0.62 %	2.85 %	0.03 %	0.00 %
THREAD 1.23.1	93.78 %	[illegible]	0.29 %	0.43 %	2.82 %	0.03 %	0.00 %
THREAD 1.24.1	90.60 %	0.30 %	0.18 %	0.99 %	3.40 %	0.03 %	0.00 %
THREAD 1.25.1	94.65 %	0.20 %	0.12 %	0.63 %	0.70 %	0.03 %	0.00 %
THREAD 1.26.1	94.19 %	0.32 %	0.17 %	0.13 %	0.66 %	0.01 %	0.00 %
THREAD 1.27.1	93.14 %	0.21 %	0.11 %	0.10 %	1.42 %	0.04 %	0.00 %
Total	2,454.20 %	7.19 %	6.20 %	15.50 %	52.66 %	0.79 %	2.44 %
Average	90.90 %	0.27 %	0.23 %	0.57 %	1.95 %	0.03 %	0.09 %
Maximum	97.91 %	0.54 %	0.48 %	2.94 %	4.02 %	0.06 %	0.78 %
Minimum	85.98 %	0.13 %	0.11 %	0.10 %	0.22 %	0.00 %	0.00 %
StDev	3.16 %	0.09 %	0.08 %	0.65 %	1.21 %	0.01 %	0.20 %
Avg/Max	0.93	0.49	0.48	0.20	0.49	0.51	0.12

Efficiency of iterative region

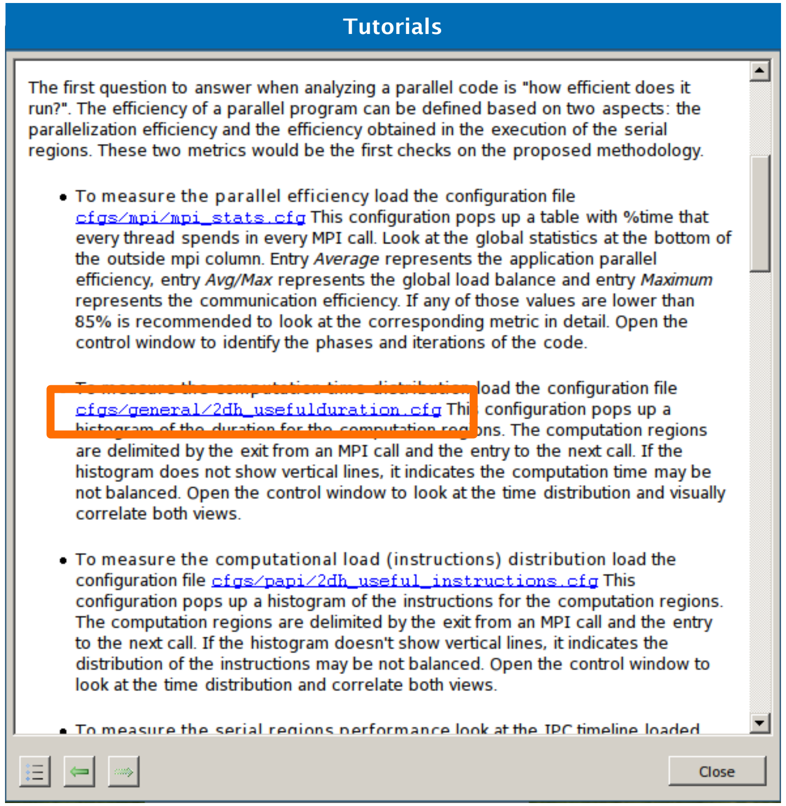
- 3 numbers to quickly describe the efficiency of your code
  - Parallel efficiency → % of time my program is computing (100% is perfect)
  - Comm efficiency → At least 1 process can finish all communications in 100 - Maximum % of the program's time (100% is perfect)
  - Load balance → Ratio of slow/fast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

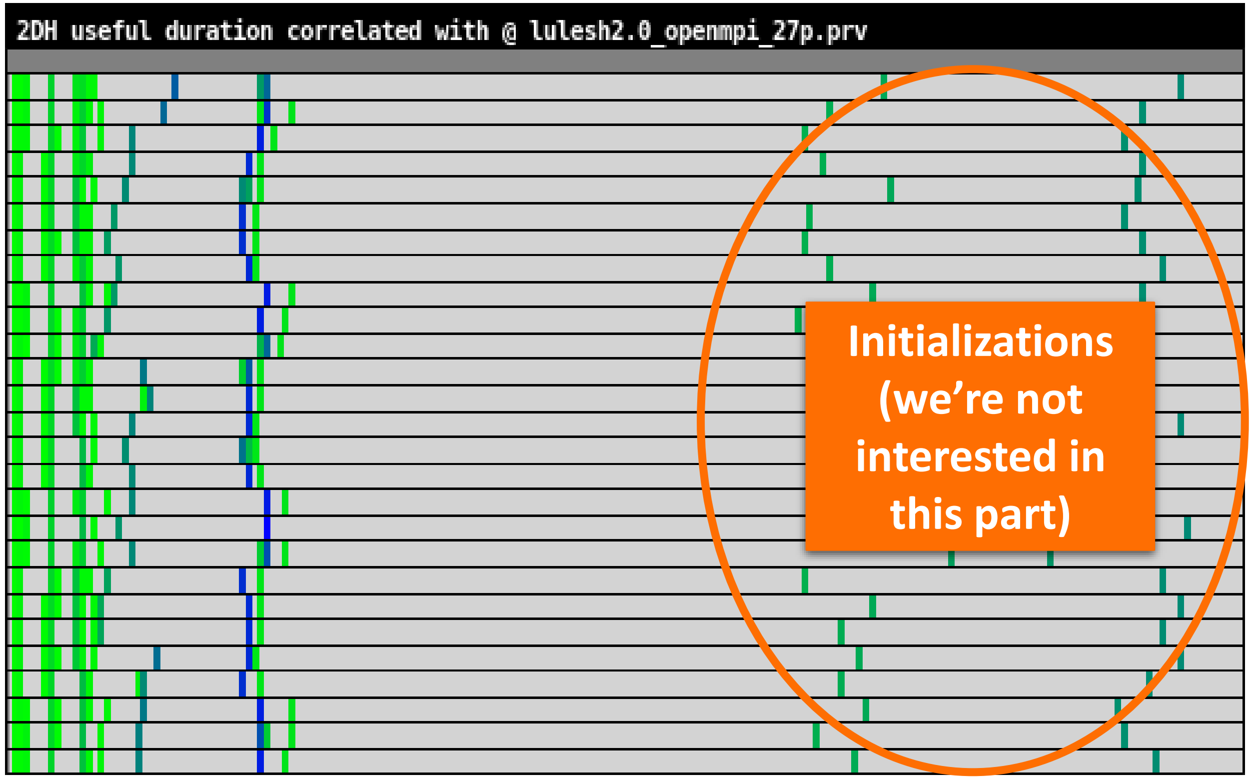


THREAD 1.17.1	90.61 %	0.52 %	0.41 %	0.33 %	0.31 %	7.65 %	0.16
THREAD 1.18.1	87.65 %	0.36 %	0.27 %	0.21 %	1.13 %	10.20 %	0.17
THREAD 1.19.1	89.90 %	0.24 %	0.19 %	0.18 %	1.16 %	8.14 %	0.18
THREAD 1.20.1	80.40 %	0.38 %	0.27 %	0.25 %	5.54 %	12.97 %	0.19
THREAD 1.21.1	80.14 %	0.26 %	0.17 %	3.31 %	1.68 %	14.28 %	0.17
THREAD 1.22.1	80.85 %	0.37 %	0.26 %	0.89 %	3.89 %	13.58 %	0.17
THREAD 1.23.1	91.44 %	0.57 %	0.38 %	0.62 %	3.89 %	2.94 %	0.16
THREAD 1.24.1	86.87 %	0.41 %	0.25 %	1.38 %	4.71 %	6.23 %	0.17
THREAD 1.25.1	92.48 %	0.27 %	0.16 %	0.92 %	0.93 %	5.06 %	0.18
THREAD 1.26.1	91.76 %	0.44 %	0.23 %	0.17 %	0.96 %	6.25 %	0.18
THREAD 1.27.1	90.23 %	0.29 %	0.14 %	0.12 %	2.03 %	7.01 %	0.18
Total	2,355.04 %	9.64 %	8.43 %	21.58 %	72.49 %	228.18 %	4.64
Average	87.22 %	0.36 %	0.31 %	0.80 %	2.68 %	8.45 %	0.17
Maximum	97.63 %	0.73 %	0.66 %	4.04 %	5.54 %	14.37 %	0.19
Minimum	80.14 %	0.18 %	0.14 %	0.12 %	0.31 %	0.03 %	0.16
StDev	4.51 %	0.13 %	0.12 %	0.89 %	1.66 %	3.68 %	0.01
Avg/Max	0.89	0.49	0.47	0.20	0.48	0.59	0.

Computation time distribution

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

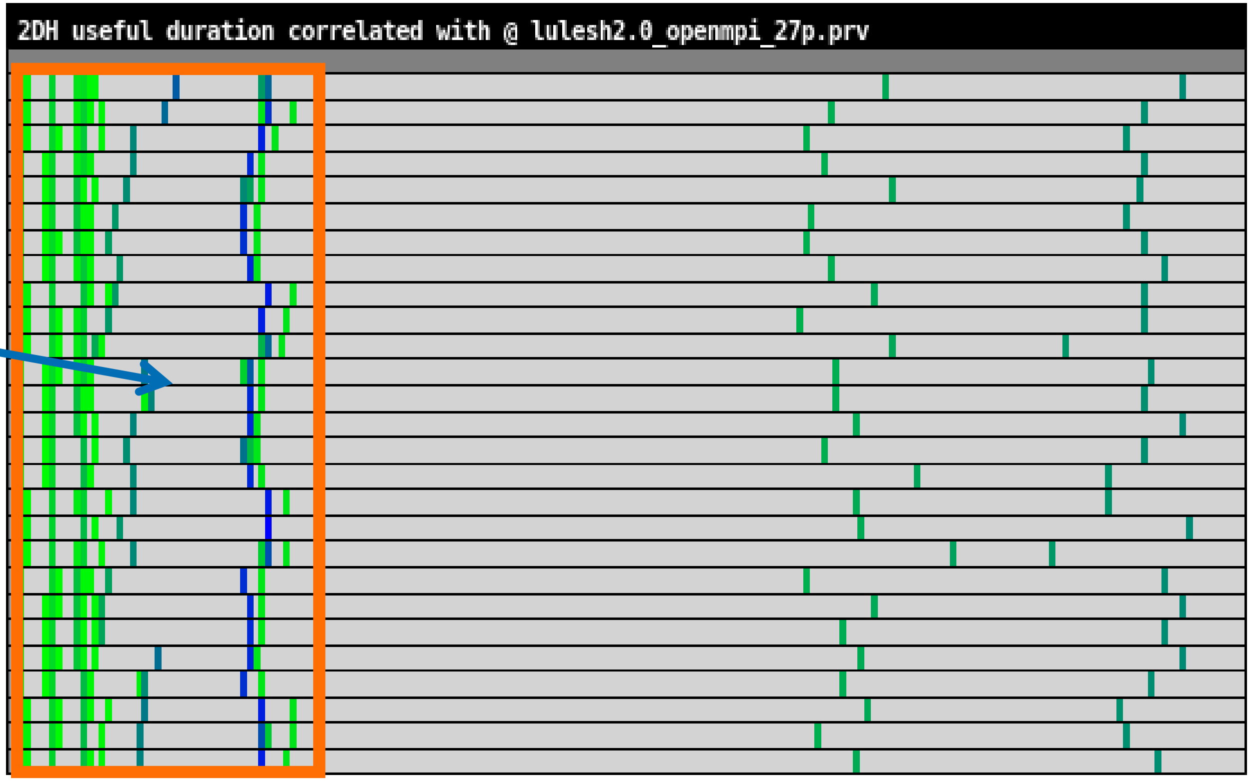




Focus on the iterative part

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

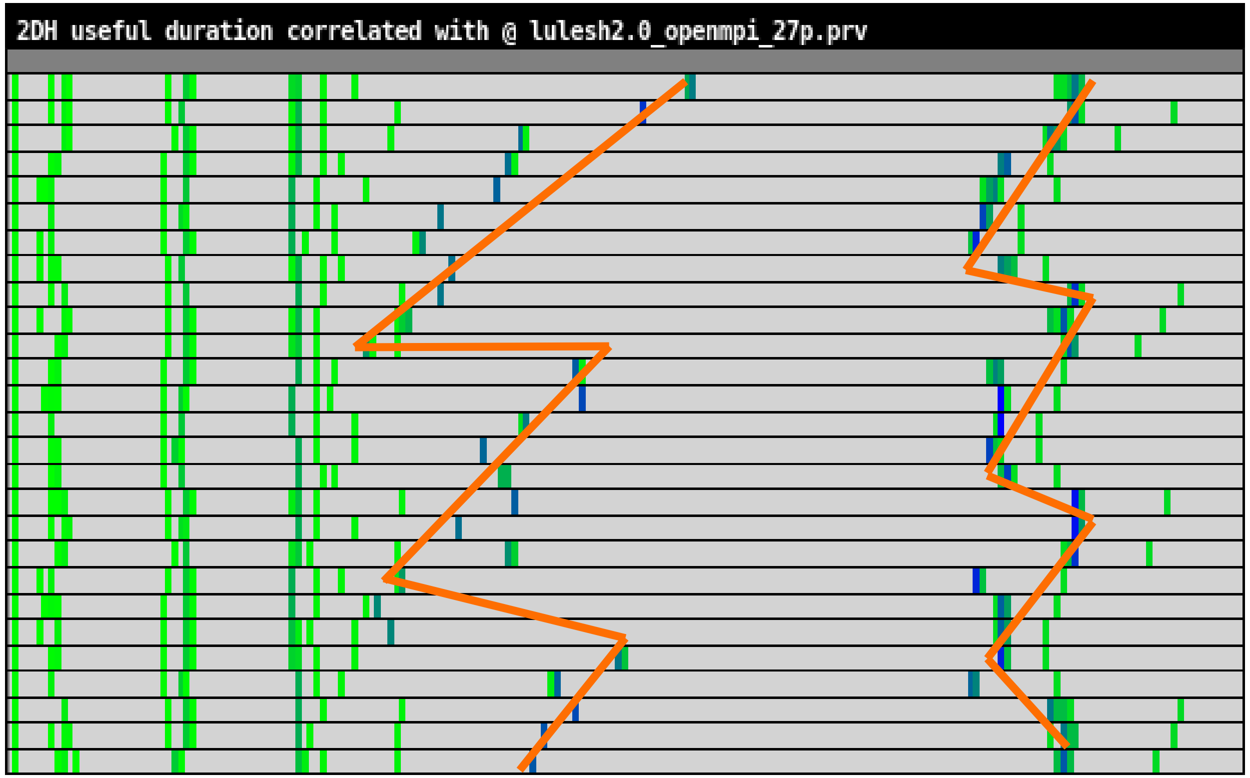
Drag & drop on this area
to skip initializations



Computation time distribution

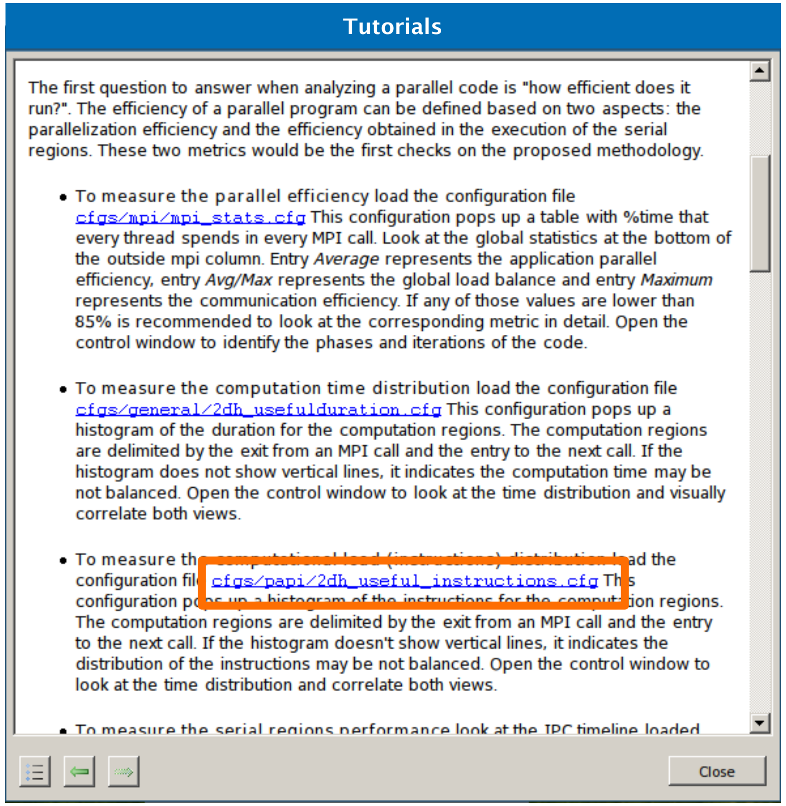
- Click on “2dh_usefulduration.cfg” (2nd link) → Shows **time computing**

Duration imbalance
(zig-zag = the more to the right, the more time a process takes)



Computation load distribution

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

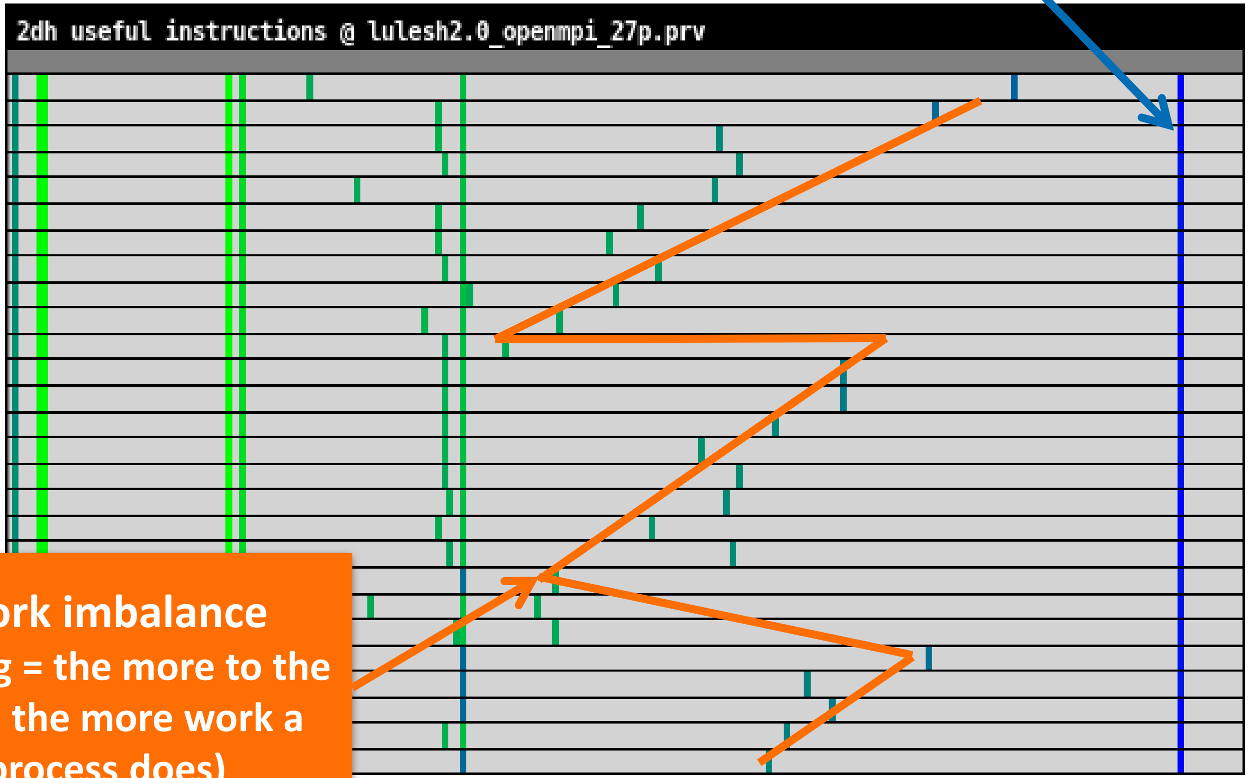

Tutorials

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 configuration file cfgs/mpi/mpi_stats.cfg This configuration pops up a table with %time that every thread spends in every MPI call. Look at the global statistics at the bottom of the outside mpi column. Entry *Average* represents the application parallel efficiency, entry *Avg/Max* represents the global load balance and entry *Maximum* represents the communication efficiency. If any of those values are lower than 85% is recommended to look at the corresponding metric in detail. Open the control window to identify the phases and iterations of the code.
- To measure the computation time distribution load the configuration file cfgs/general/2dh_usefulduration.cfg This configuration pops up a histogram of the duration for the computation regions. The computation regions are delimited by the exit from an MPI call and the entry to the next call. If the histogram does not show vertical lines, it indicates the computation time may be not balanced. Open the control window to look at the time distribution and visually correlate both views.
- To measure the computational load (instructions) distribution load the configuration file cfgs/papi/2dh_useful_instructions.cfg This configuration pops up a histogram of the instructions for the computation regions. The computation regions are delimited by the exit from an MPI call and the entry to the next call. If the histogram doesn't show vertical lines, it indicates the distribution of the instructions may be not balanced. Open the control window to look at the time distribution and correlate both views.
- To measure the serial regions performance look at the IPC timeline loaded

Close

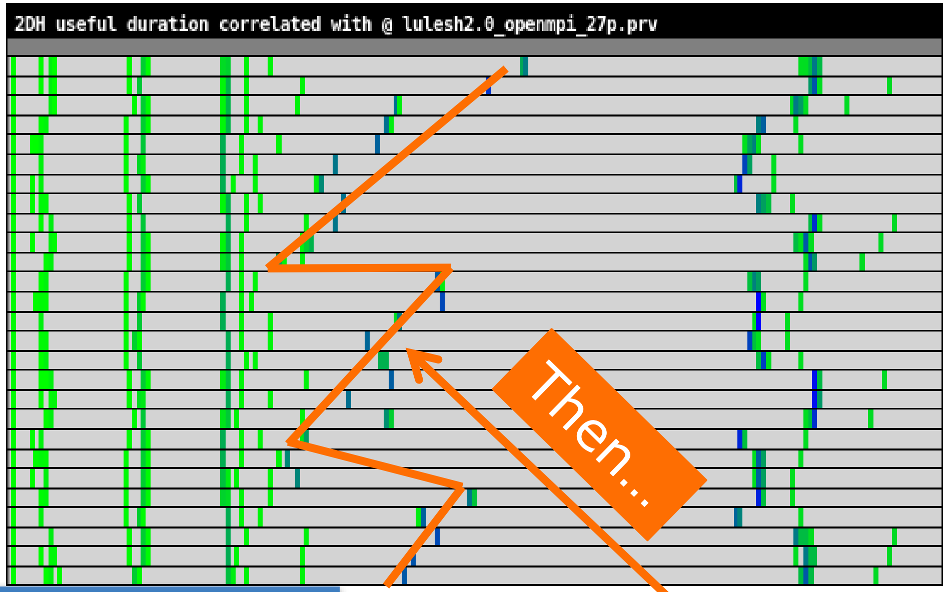
Perfect work distribution in other parts of the program (straight line)



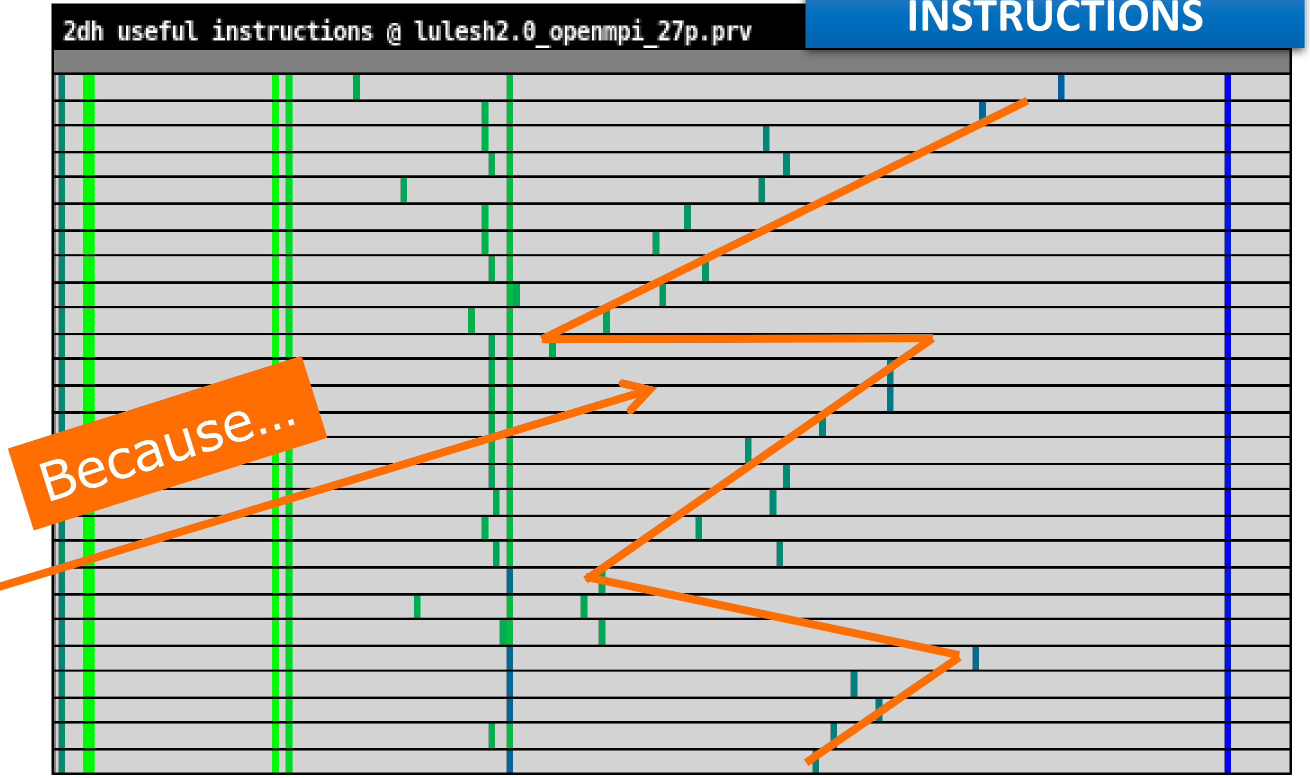
Work imbalance (zig-zag = the more to the right, the more work a process does)

Computation load distribution

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



DURATION

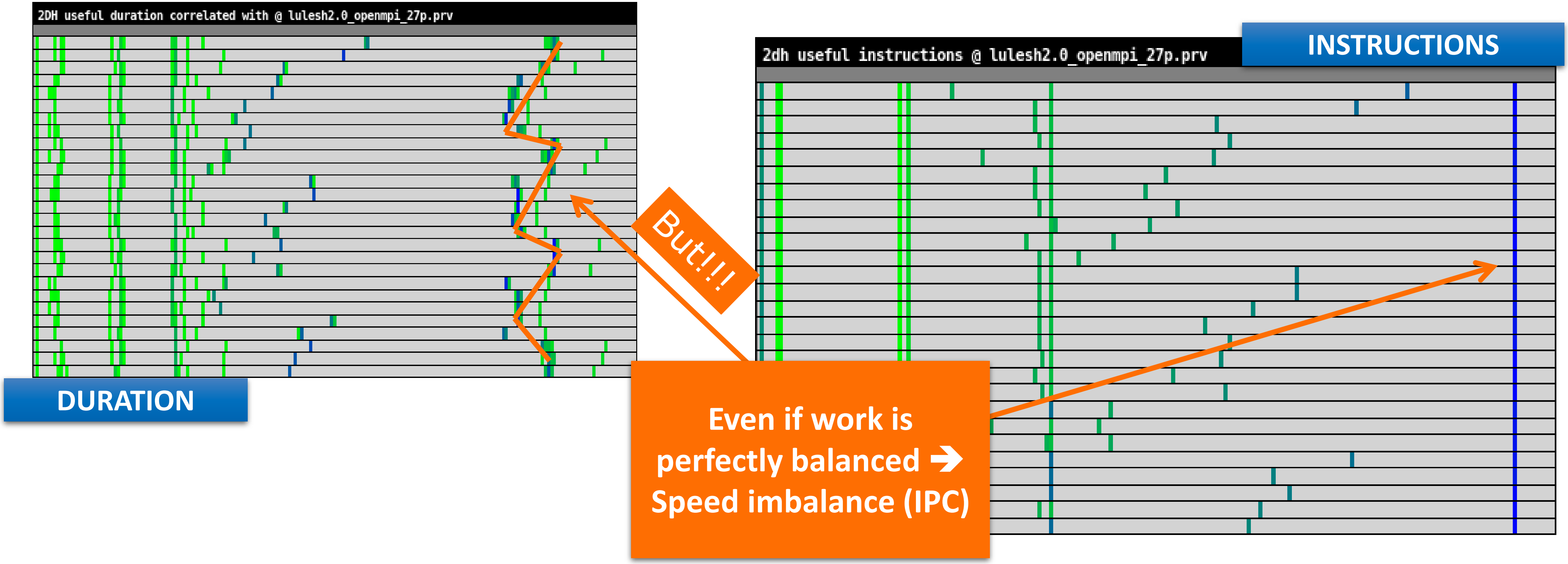


INSTRUCTIONS

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

Computation load distribution

- Comparing two histograms → **Not similar shapes** → Balanced workload executes at variable speed

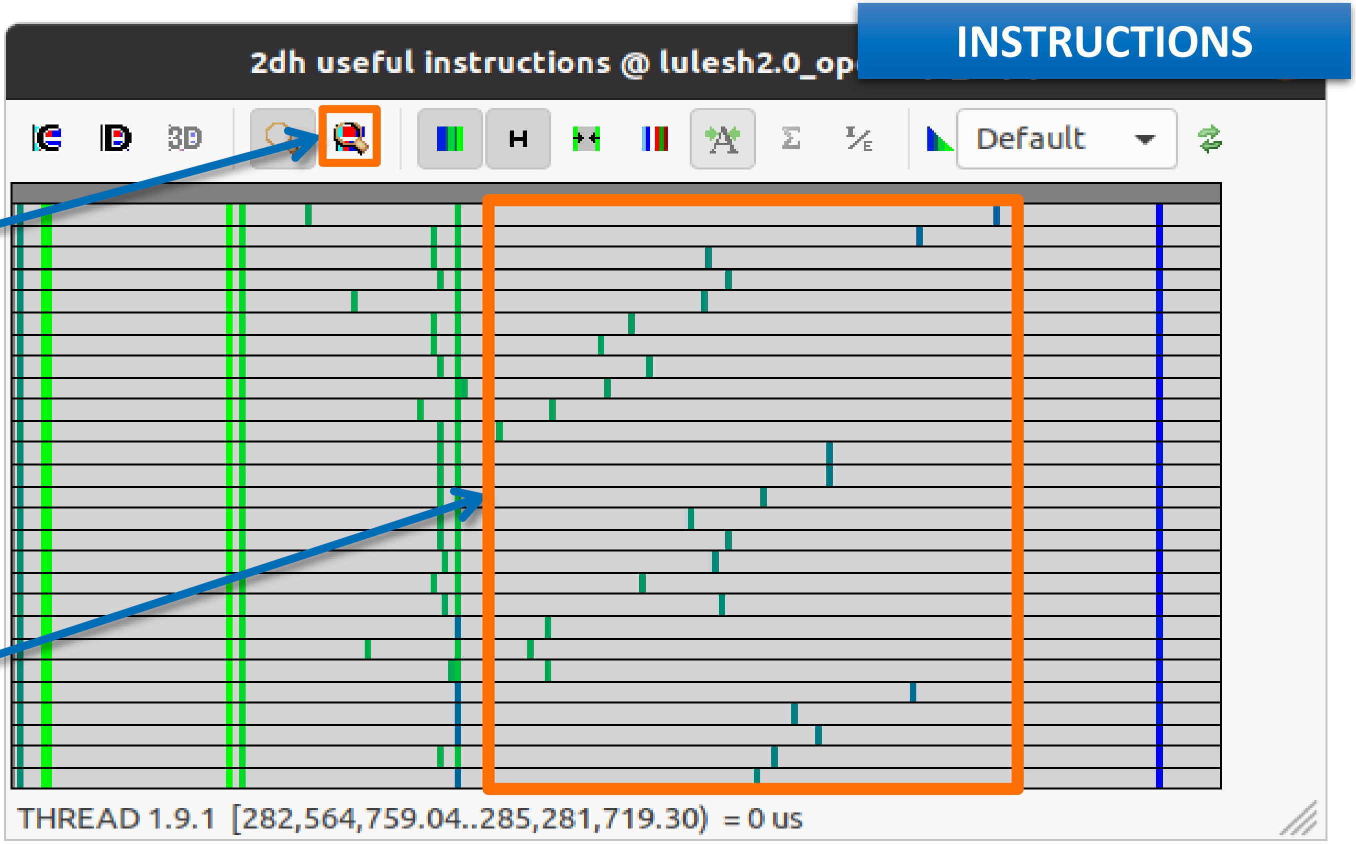


Where does this happen?

- Go from the table to the timeline

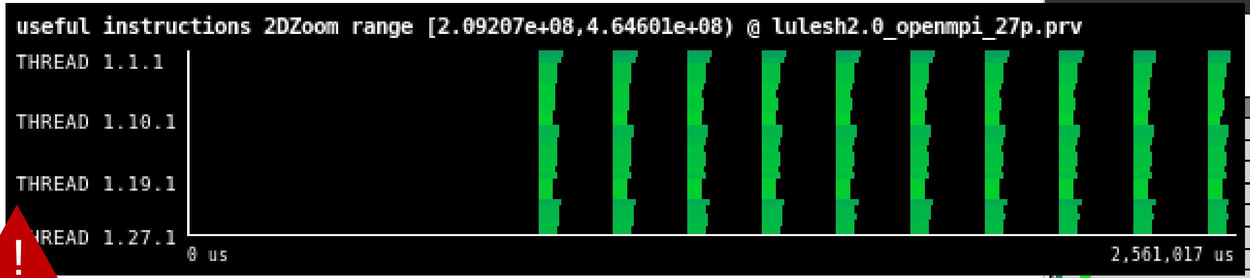
1. Click on "Open Filtered Control Window"

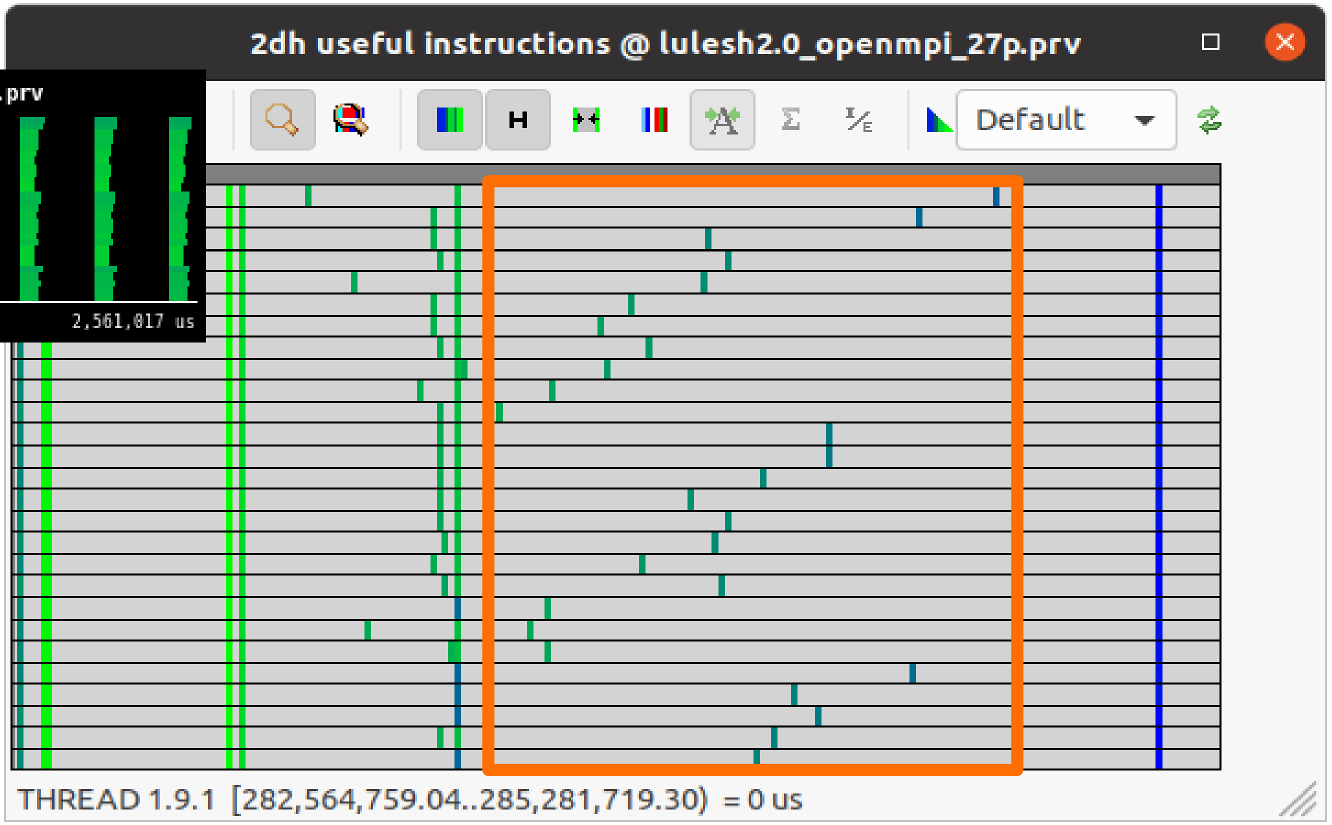
2. Select this area (drag-and-drop)



Where does this happen?

- Go from the table to the timeline

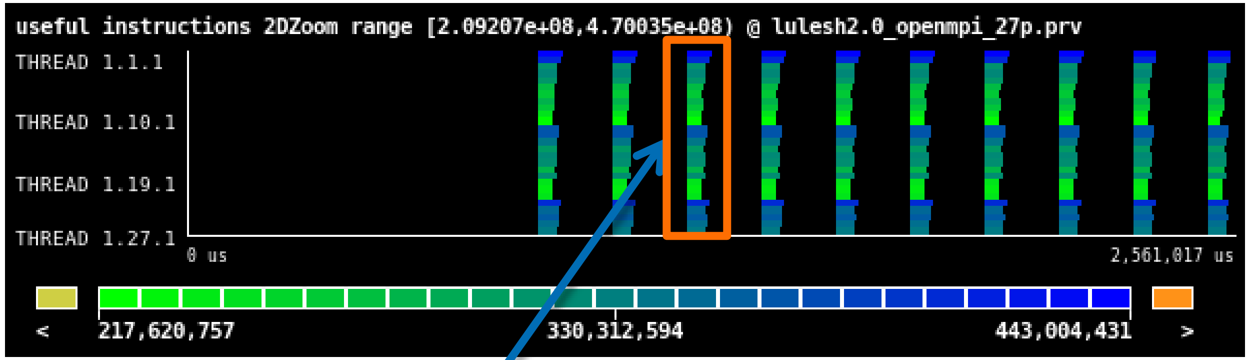




Clicking here always rescales the gradient. Same as…
Right click →
Fit Semantic Scale → Fit Both
(do this when colors are too similar)

Where does this happen?

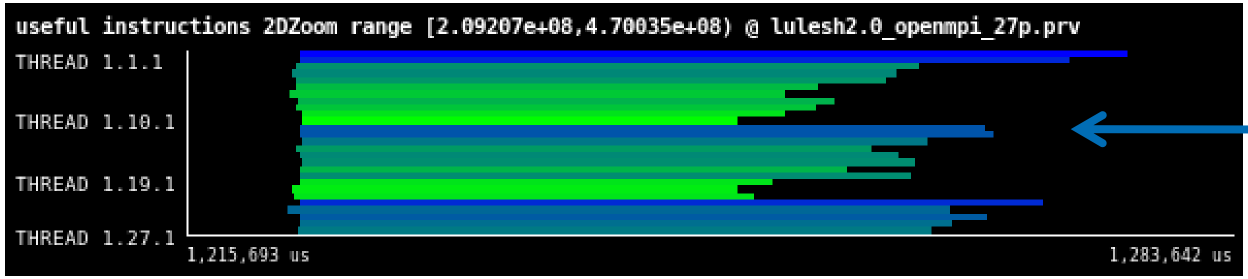
- Slow & Fast at the same time? → Imbalance



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

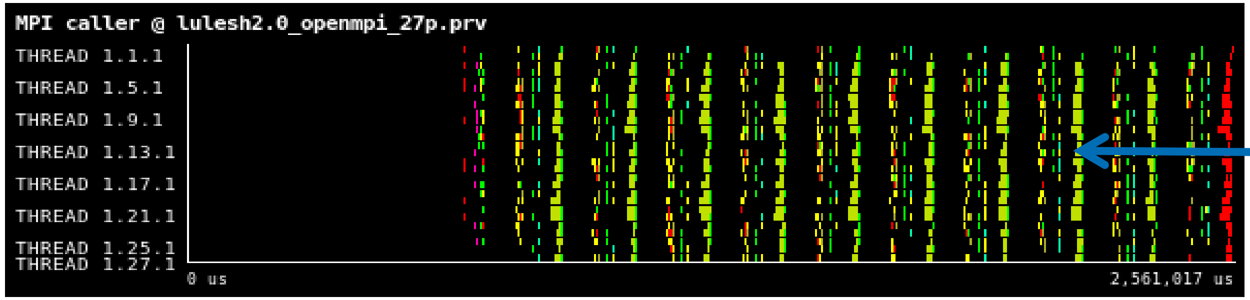
Where does this happen?

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



1. Right click → Copy

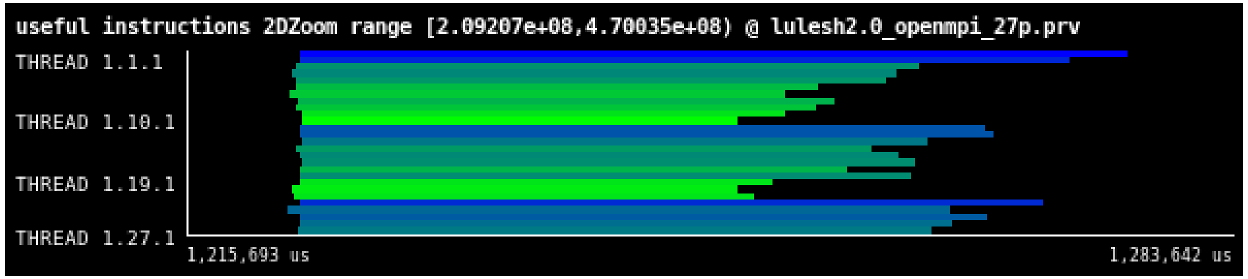
- **Hints → Call stack references → Caller function**



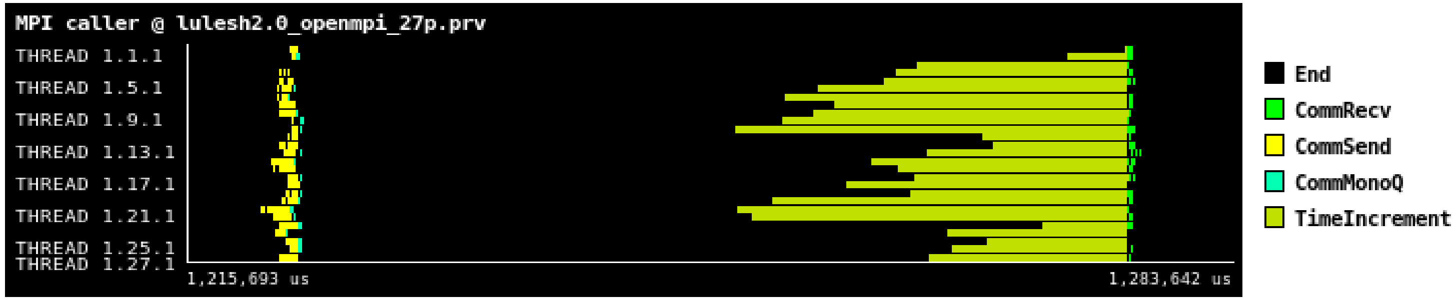
2. Right click → Paste → Time

Where does this happen?

- Slow & Fast at the same time? → Imbalance

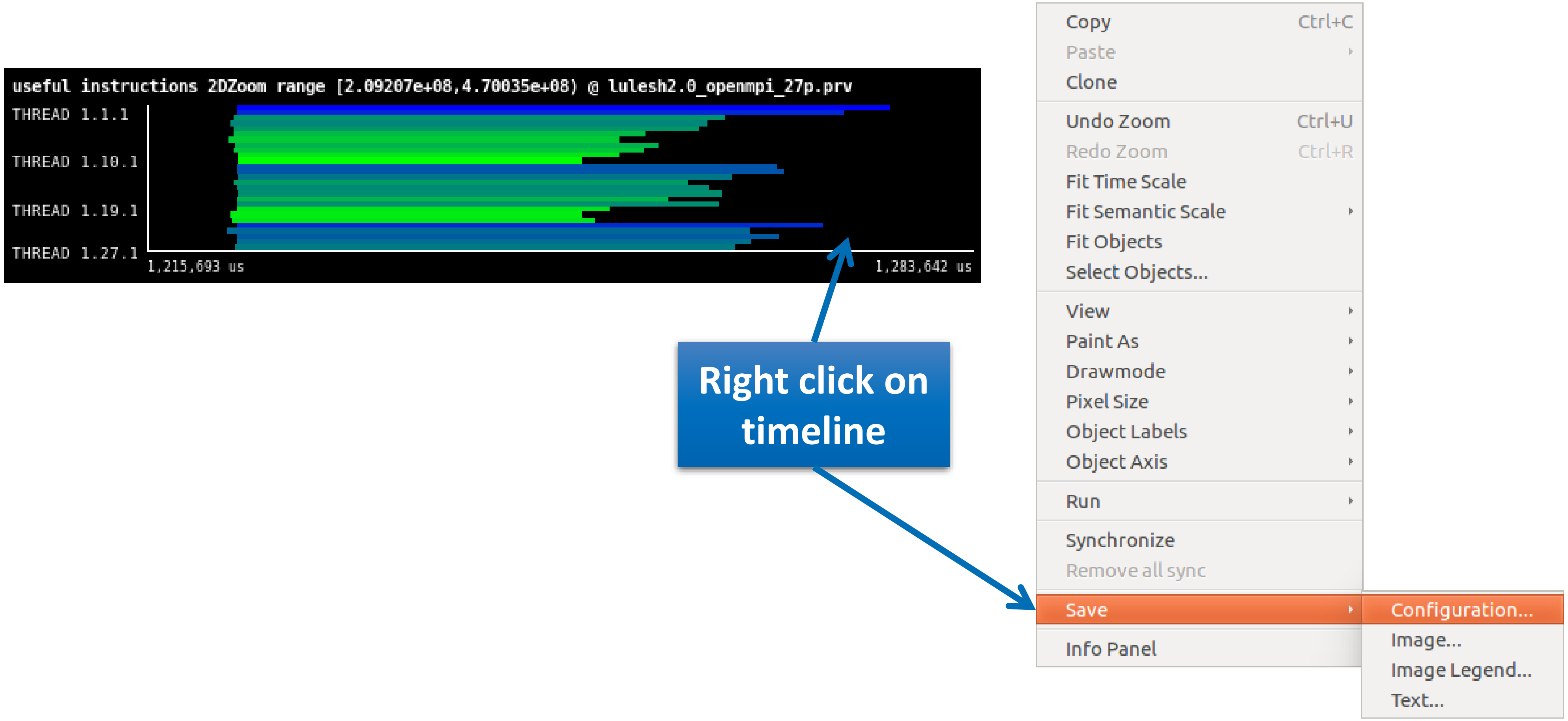


- **Hints → Call stack references → Caller function**

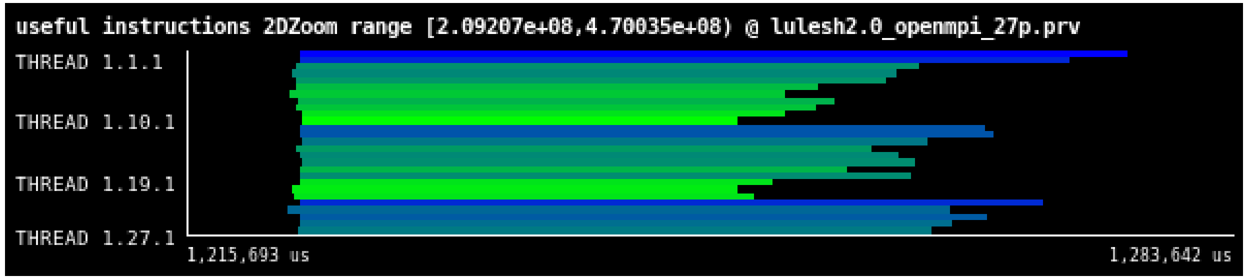


User code with imbalanced workload between `CommMonoQ` & `TimeIncrement`
(you can hover the mouse to see the legend over the colors)

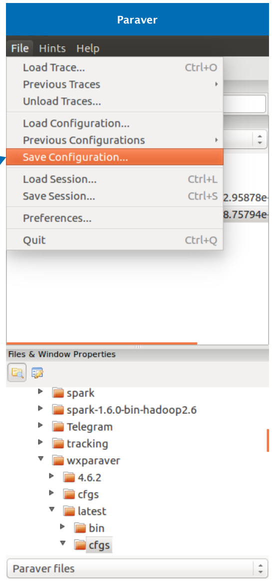
Save CFG's (method 1)

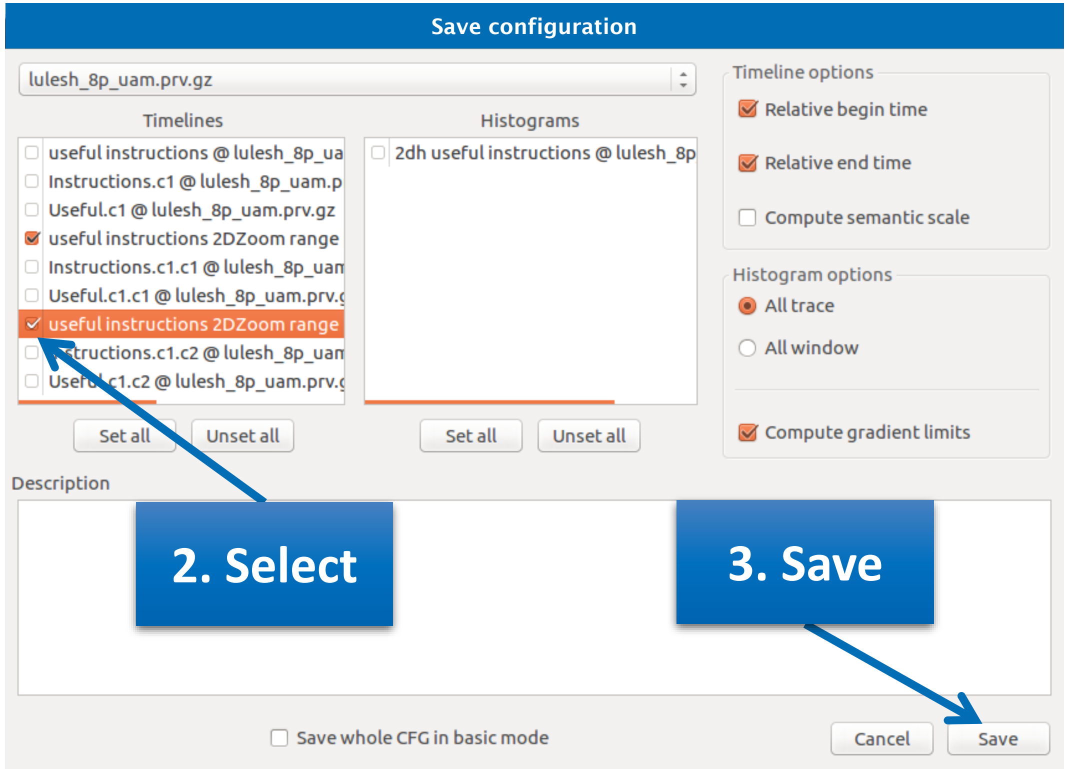


Save CFG's (method 2)



1. File → Save Configuration



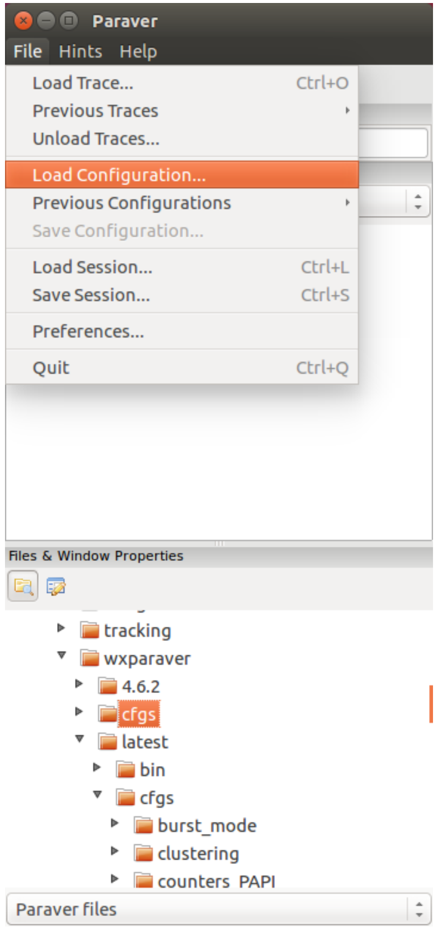


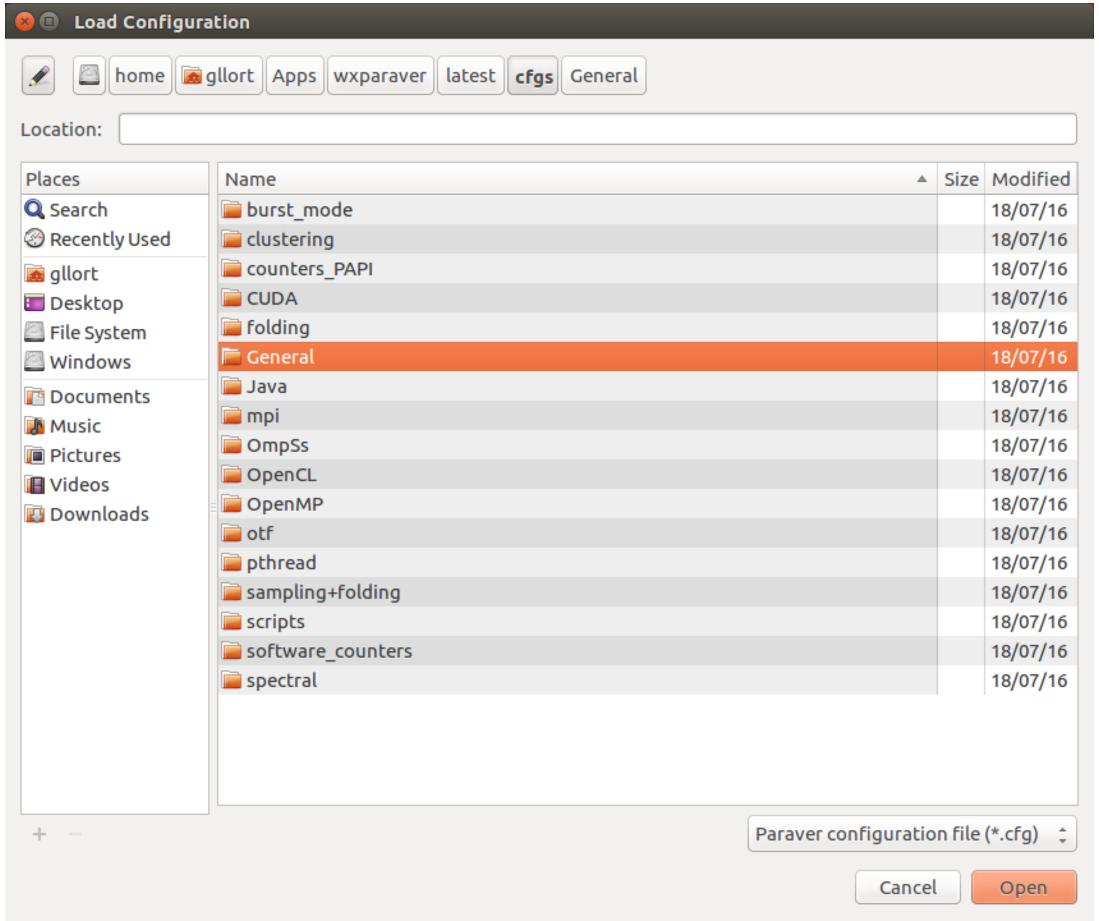
2. Select

3. Save

CFG’s distribution

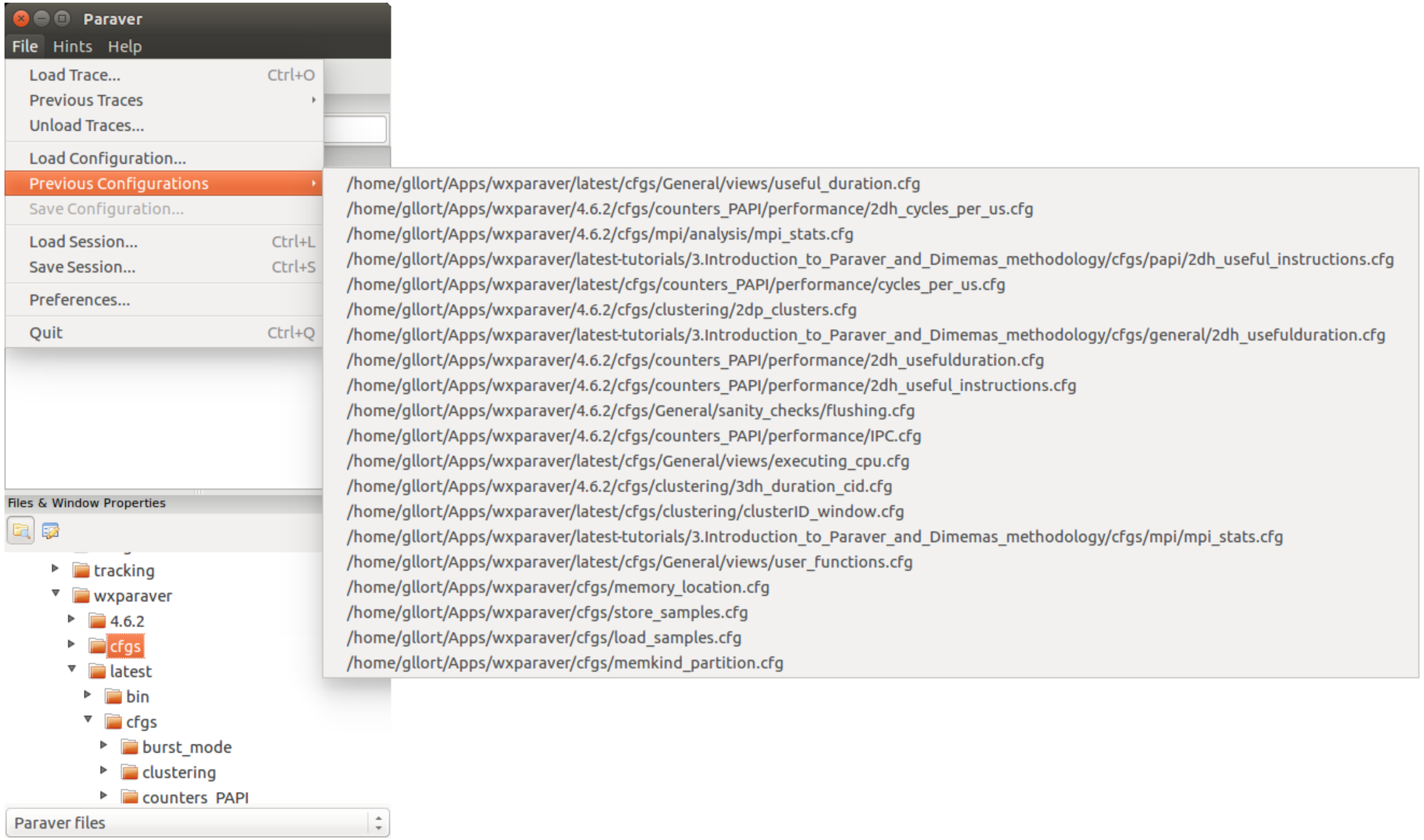
- Paraver comes with many included CFG’s: File ➔ Load Configuration ➔ Apply any CFG to any trace!





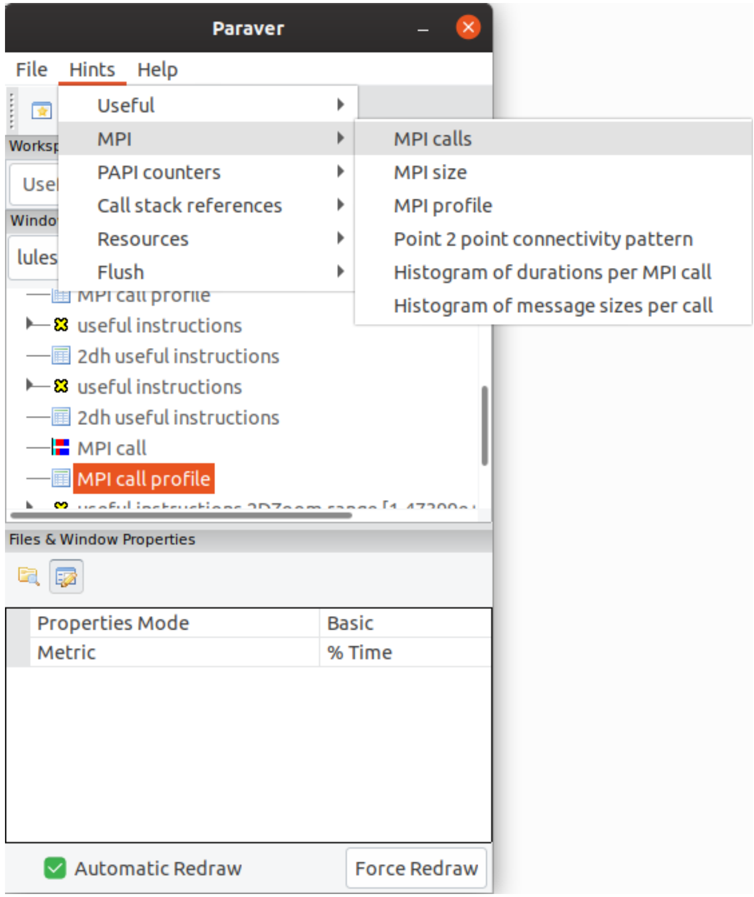
CFG’s distribution

- Recently opened CFG’s are always at hand: File ➔ Previous Configurations



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 7-15 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 17-35 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

```
hawk> cd $HOME/tools-material/clustering
hawk> cat ./run_clustering.sh
hawk> ./run_clustering.sh
```

Have a look, it's just a wrapper for the tool `BurstClustering`

- If you didn't get your own trace, you can edit `run_clustering.sh` and change the input trace to use a prepared one from:

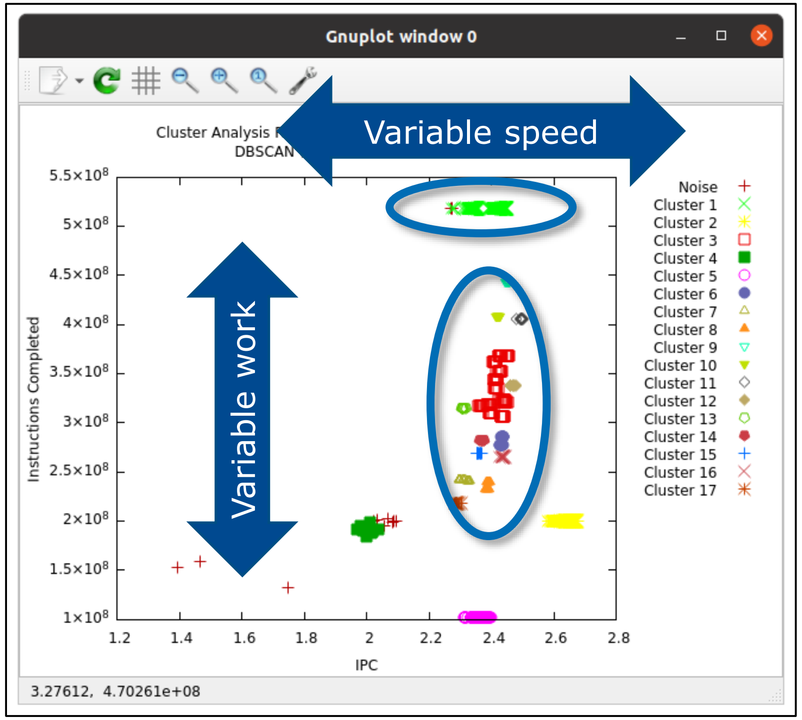
```
-i ../traces/lulesh2.0_openmpi_27p.prv
```

Cluster-based analysis

- Check the resulting scatter plot

```
hawk> gnuplot lulesh2.0_openmpi_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**

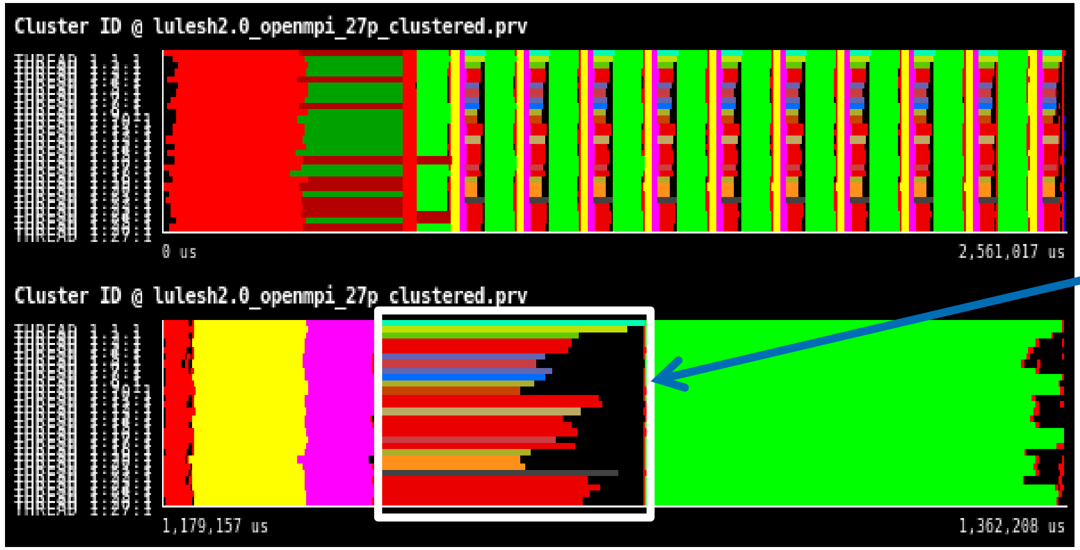


Correlating scatter plot and time distribution

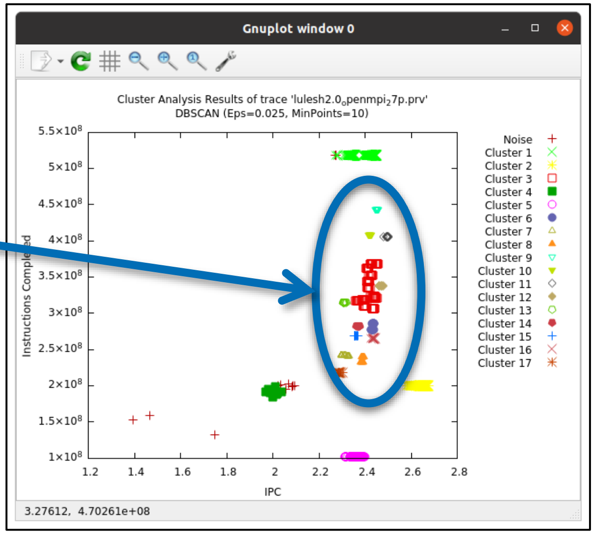
- Copy the clustered trace to your laptop and open it with Paraver:

```
laptop> $HOME/paraver/bin/wxparaver <path-to>/lulesh2.0_openmpi_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

Germán Llort, Judit Giménez
(tools@bsc.es)
Barcelona Supercomputing Center