BSC Tools hands-on

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

Extrae main features

- Platforms
  - Intel, AMD, Cray, BlueGene, MIC, ARM, Android, Fujitsu Sparc, RISC-V …
- Parallel programming models
  - MPI, OpenMP, pthreads, OmpSs, CUDA, HIP, OpenACC, OpenCL, GASPI, Java, Python …
- Performance Counters
  - Using PAPI interface
- Link to source code
  - Callstack at MPI routines
  - OpenMP outlined routines
  - Selected user functions
- Periodic sampling (time-based, counters-based, PEBS)

- User events (Extrae API)

No need to recompile nor 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
- Compiler instrumentation (`-finstrument-functions`)
- Static link (PMPI, Extrae API)

Getting your first trace

- Provided folder **`/lrz/sys/courses/vihps/2024/material/bsctools`** contains:
  - Sample application compiled with default Intel 2021.4 + Intel MPI 2019 toolchain (`lulesh2.0`)
  - Jobscripts to execute and trace (`job.cmd, trace.sh`)
  - Configuration of the tracing tool (`extrae.xml`)
  - Already generated tracefiles (`traces/*.{pcf,prv,row}`)

- Copy this folder and you are ready to follow this hands-on tutorial

```
cm2> cp -r /lrz/sys/courses/vihps/2024/material/bsctools $HOME
```

Using Extrae in 3 steps

1. **Adapt** your job submission script

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: Example of a standard jobscript (without tracing)

```
#!/bin/bash

#SBATCH -o ./%x.%j.%N.out
#SBATCH -e ./%x.%j.%N.out
#SBATCH -D ./
#SBATCH --get-user-env
#SBATCH --clusters=cm2_tiny
#SBATCH --ntasks=27
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=27
#SBATCH --time=00:05:00
#SBATCH --reservation=hhps1s24

module load slurm_setup

mpiexec -n ${SLURM_NTASKS} ./lulesh2.0 -i 10 -s 65 -p
```

Request resources

Run the program

Step 1: Modify the job script to load Extrae

```
cm2> cat $HOME/bsctools/extrae/job.cmd
```

```
#!/bin/bash

#SBATCH -o ./%x.%j.%N.out
#SBATCH -e ./%x.%j.%N.out
#SBATCH -D ./
#SBATCH --get-user-env
#SBATCH --clusters=cm2_tiny
#SBATCH --ntasks=27
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=27
#SBATCH --time=00:05:00
#SBATCH --reservation=hhps1s24

module load slurm_setup
module use /lrz/sys/courses/vihps/2024/modulefiles
module load extrae

export TRACE_NAME=lulesh2.0_27p_1N.prv

mpiexec -n ${SLURM_NTASKS} ./trace.sh ./lulesh2.0 -i 10 -s 65 -p

${EXTRAE_HOME}/bin/mpi2prv -f TRACE.mpits -o ${TRACE_NAME}
```

Run with Extrae

Final trace generation

```
#!/bin/bash

export EXTRAE_CONFIG_FILE=./extrae.xml

# Keep this if your app uses MPI, remove otherwise
export EXTRAE_SKIP_AUTO_LIBRARY_INITIALIZE=1

# Select tracing library
export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitrace.so

# Run the program
$*
```

What to trace?

Choose 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" if code is Fortran and default lib misses MPI activity

Step 2: Extrae XML configuration

```
cm2> cat $HOME/bsctools/extrae/extrae.xml
```

```
<mpi enabled="yes">
  <counters enabled="yes" />
  <comm-calls enabled="yes" />
</mpi>

<openmp enabled="yes“>
  <locks enabled="no" />
  <taskloop 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)

```
cm2> cat $HOME/bsctools/extrae/extrae.xml
```

```
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="0">
      PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L2_DCM,PAPI_L3_TCM
    </set>
    <set enabled="yes" domain="all" changeat-time="0">
      RESOURCE_STALLS:RS,RESOURCES_STALLS:ROB,
      RESOURCE_STALLS:SB
    </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="no"
       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
(Very handy!)

Step 3: Run it!

- Submit your job as usual

```
cm2> cd $HOME/bsctools/extrae
cm2> sbatch job.cmd
```

- Once finished (check `squeue -M cm2_tiny -u $USER`) the trace is in the same folder (3 files):

```
cm2> ls
… lulesh2.0_27p_1N.pcf   lulesh2.0_27p_1N.prv   lulesh2.0_27p_1N.row
```

- Any trouble? There's a trace already generated under folder `bsctools/traces`
- Now copy it to your laptop and let's look into it!

```
laptop> scp <USER>@lxlogin1.lrz.de:bsctools/extrae/*.{pcf,prv,row} .
```

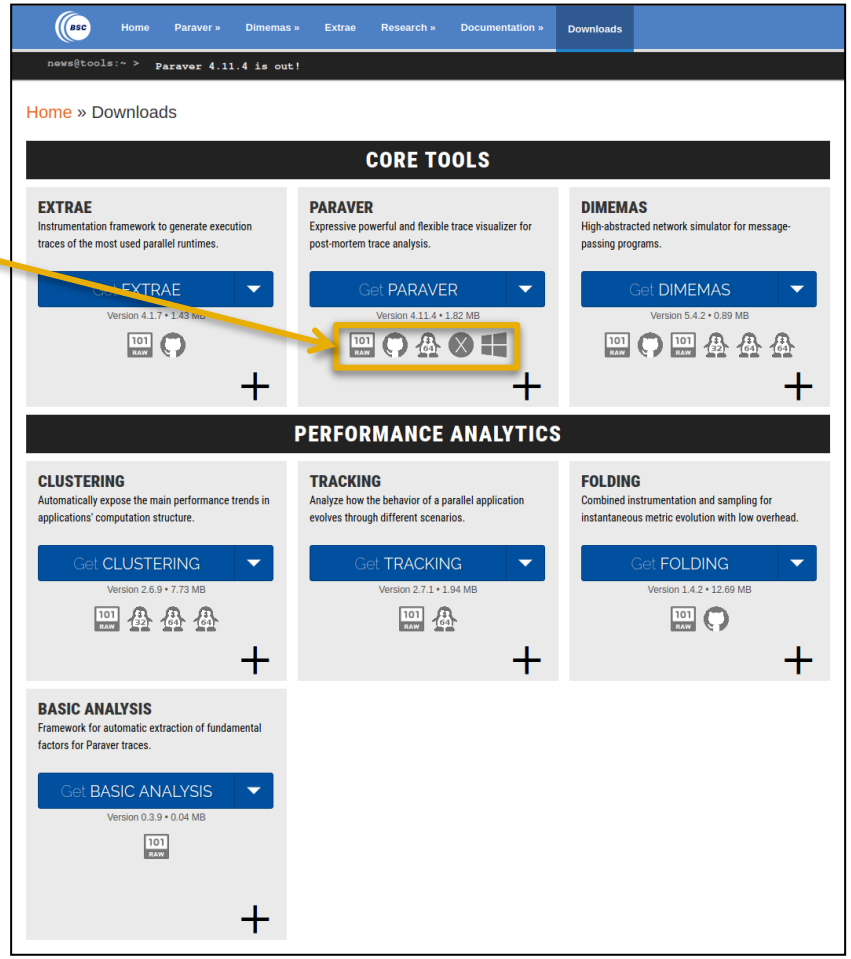
Install Paraver in your laptop

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

wxparaver-4.11.4-win.zip

wxparaver-4.11.4-mac.zip

wxparaver-4.11.4-Linux_x86_64.tar.gz



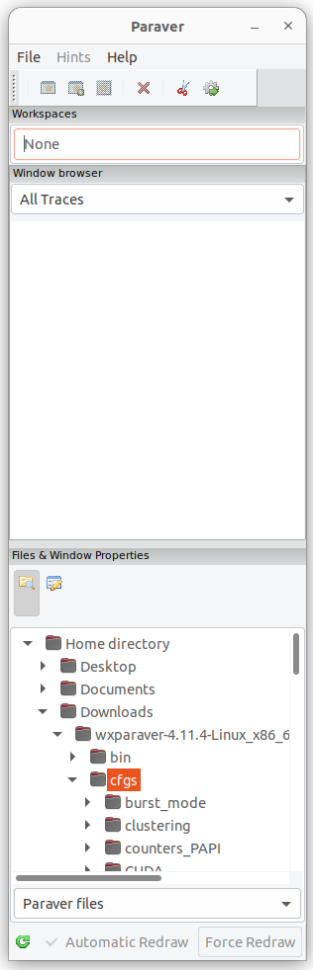
Install Paraver in your laptop

- Start Paraver:
  - Linux:

```
laptop> tar xf wxparaver-4.11.4-Linux_x86_64.tar.bz2
laptop> wxparaver-4.11.4-Linux_x86_64/bin/wxparaver
```

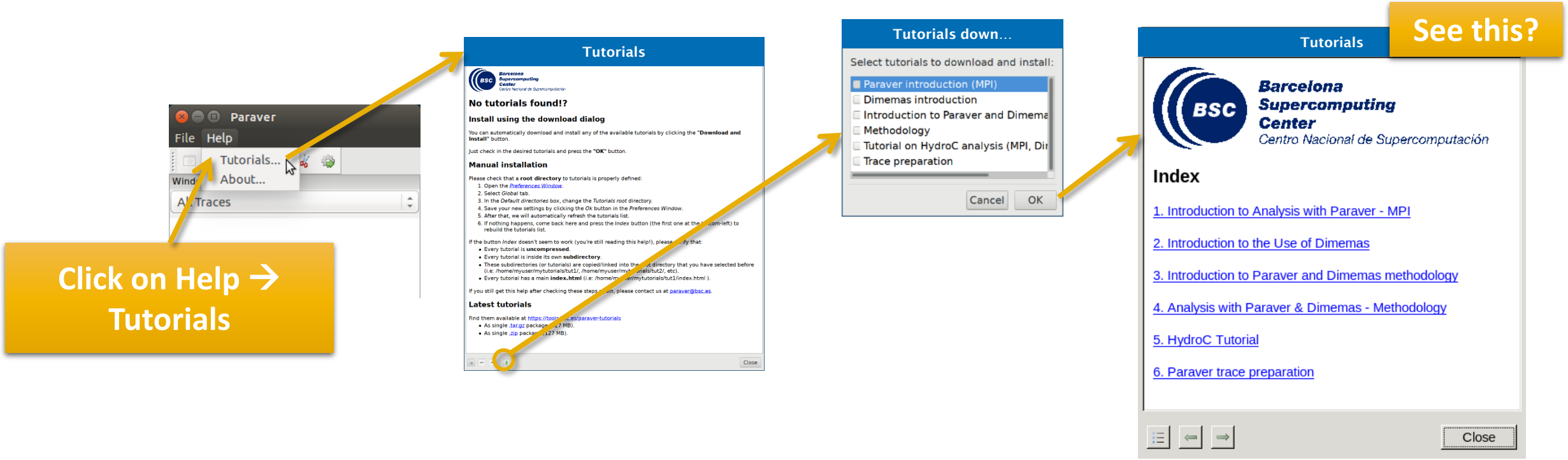
  - Windows: Unzip & double-click on `wxparaver-4.11.4-win/wxparaver.bat`
  - Mac: Unzip & double-click on `wxparaver.app`
  - Any issue? Remotely from CoolMUC-2:

```
laptop> ssh -Y <USER>@lxlogin1.lrz.de
cm2> module use /lrz/sys/courses/vihps/2024/modulefiles
cm2> module load paraver
cm2> wxparaver
```



Install Paraver tutorials

- Download Tutorial #3 – Introduction to Paraver and Dimemas methodology



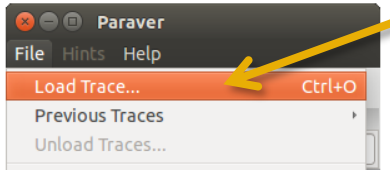
First steps of analysis

- Download application trace (3 files: prv, pcf, row; backup under `bsctools/traces`):

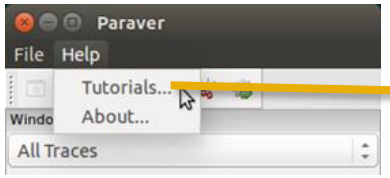
```
laptop> scp <USER>@lxlogin1.lrz.de:bsctools/extrae/lulesh2.0_27p_1N.* .
```

- Load the trace in Paraver:

Click on File → Load Trace → Browse to "lulesh2.0_27p_1N.prv"



- Follow Tutorial #3:



Measure the parallel efficiency

- Click on "mpi_stats.cfg" → 3 numbers to quickly describe the efficiency of your code
  - Start paying more attention to values below 85%

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) distrib
configuration file cfgs/papi/2dh_useful_instructions
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
look at the time distribution and correlate both views.

- To measure the serial regions performance look at the IP

Parallel efficiency (Avg 89.61%)

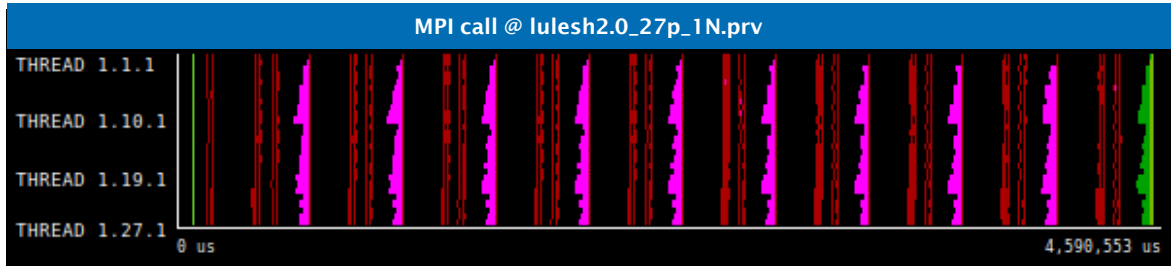
Comm efficiency (Max 99.03%)

Load balance (Avg/Max 90%)

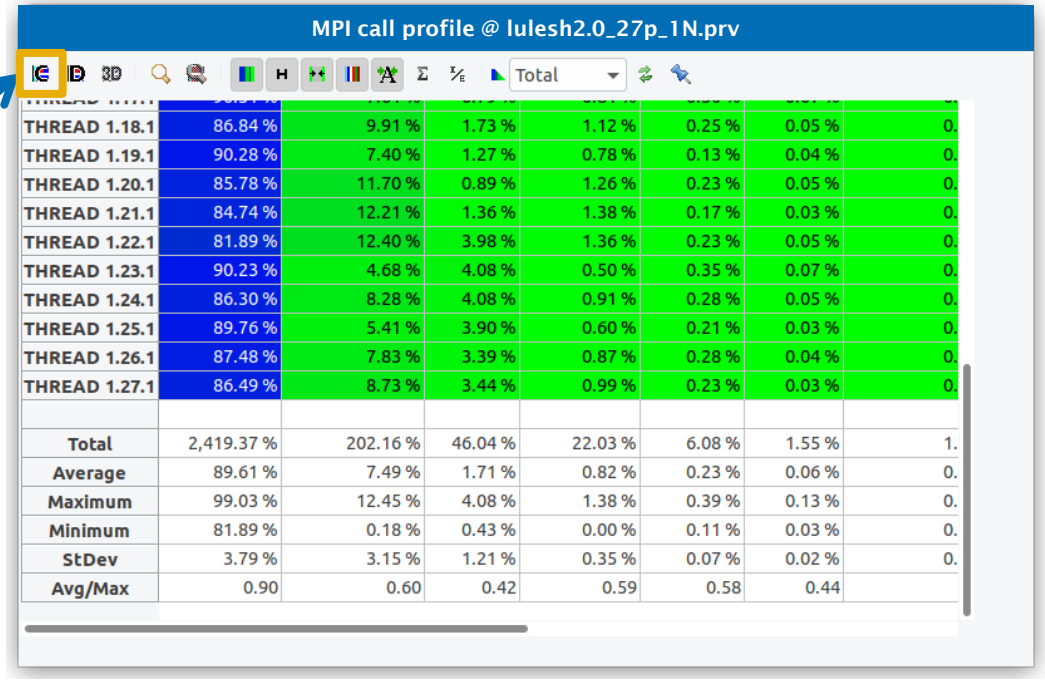
MPI call profile @ lulesh2.0_27p_1N.prv

	Outside MPI	MPI_Allreduce	MPI_Wait	MPI_Reduce	MPI_Isend	MPI_Irecv	MPI_Comm_rank	MPI_Barrier	MPI_Init	MPI_Waitall	MPI_Comm_size	MPI_Finalize
THREAD 1.1.1	99.03 %	0.18 %	0.54 %	0.00 %	0.11 %	0.05 %	0.04 %	0.02 %	0.00 %	0.01 %	0.01 %	0.00 %
THREAD 1.2.1	97.20 %	1.15 %	1.15 %	0.16 %	0.16 %	0.07 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.3.1	92.22 %	5.38 %	1.54 %	0.58 %	0.14 %	0.04 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.4.1	92.99 %	4.64 %	1.56 %	0.48 %	0.17 %	0.06 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.5.1	93.27 %	4.84 %	0.92 %	0.52 %	0.26 %	0.09 %	0.05 %	0.03 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.6.1	88.89 %	8.10 %	1.75 %	0.88 %	0.22 %	0.06 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.7.1	89.11 %	8.60 %	1.08 %	0.93 %	0.14 %	0.04 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.8.1	89.59 %	8.45 %	0.67 %	0.89 %	0.23 %	0.06 %	0.06 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.9.1	86.80 %	10.67 %	1.02 %	1.17 %	0.19 %	0.04 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.10.1	86.83 %	11.05 %	0.55 %	1.22 %	0.18 %	0.06 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.11.1	85.17 %	12.45 %	0.56 %	1.37 %	0.26 %	0.08 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.12.1	93.06 %	4.68 %	1.39 %	0.50 %	0.21 %	0.06 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.13.1	94.20 %	4.34 %	0.58 %	0.43 %	0.27 %	0.08 %	0.04 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.14.1	91.43 %	6.81 %	0.43 %	0.72 %	0.39 %	0.13 %	0.06 %	0.00 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.15.1	88.14 %	8.11 %	2.36 %	0.91 %	0.28 %	0.09 %	0.06 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.16.1	91.33 %	6.56 %	1.04 %	0.70 %	0.20 %	0.05 %	0.04 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.17.1	90.31 %	7.61 %	0.79 %	0.81 %	0.30 %	0.07 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.18.1	86.84 %	9.91 %	1.73 %	1.12 %	0.25 %	0.05 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.19.1	90.28 %	7.40 %	1.27 %	0.78 %	0.13 %	0.04 %	0.04 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.20.1	85.78 %	11.70 %	0.89 %	1.26 %	0.23 %	0.05 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.21.1	84.74 %	12.21 %	1.36 %	1.38 %	0.17 %	0.03 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.22.1	81.89 %	12.40 %	3.98 %	1.36 %	0.23 %	0.05 %	0.04 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.23.1	90.23 %	4.68 %	4.08 %	0.50 %	0.35 %	0.07 %	0.04 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.24.1	86.30 %	8.28 %	4.08 %	0.91 %	0.28 %	0.05 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.25.1	89.76 %	5.41 %	3.90 %	0.60 %	0.21 %	0.03 %	0.04 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.26.1	87.48 %	7.83 %	3.39 %	0.87 %	0.28 %	0.04 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
THREAD 1.27.1	86.49 %	8.73 %	3.44 %	0.99 %	0.23 %	0.03 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
Total	2,419.37 %	202.16 %	46.04 %	22.03 %	6.08 %	1.55 %	1.31 %	0.54 %	0.43 %	0.28 %	0.21 %	0.02 %
Average	89.61 %	7.49 %	1.71 %	0.82 %	0.23 %	0.06 %	0.05 %	0.02 %	0.02 %	0.01 %	0.01 %	0.00 %
Maximum	99.03 %	12.45 %	4.08 %	1.38 %	0.39 %	0.13 %	0.06 %	0.03 %	0.02 %	0.01 %	0.01 %	0.00 %
Minimum	81.89 %	0.18 %	0.43 %	0.00 %	0.11 %	0.03 %	0.04 %	0.00 %	0.00 %	0.01 %	0.01 %	0.00 %
StDev	3.79 %	3.15 %	1.21 %	0.35 %	0.07 %	0.02 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Avg/Max	0.90	0.60	0.42	0.59	0.58	0.44	0.87	0.77	0.96	0.81	0.85	0.79

Focus on the iterative part

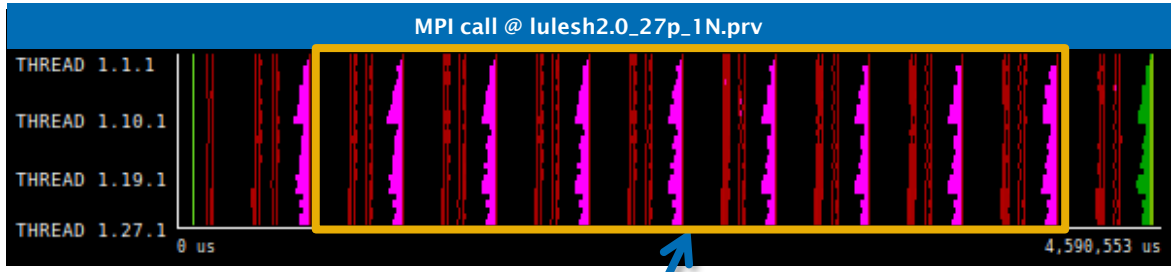


Click on
"Open Control Window"

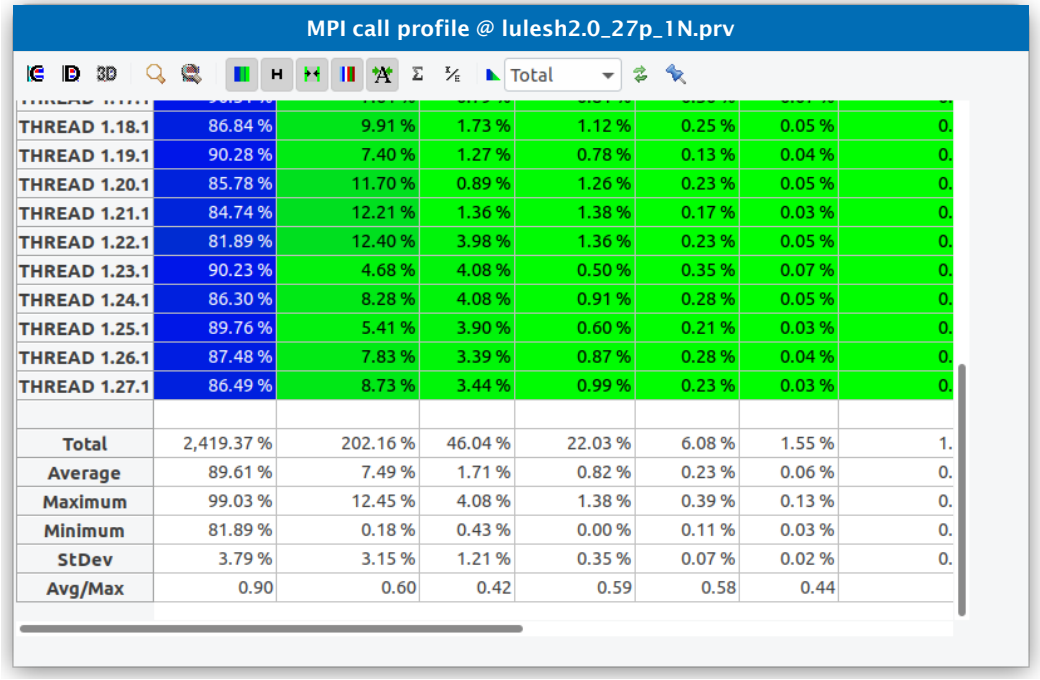


THREAD 1.18.1	86.84 %	9.91 %	1.73 %	1.12 %	0.25 %	0.05 %	0.
THREAD 1.19.1	90.28 %	7.40 %	1.27 %	0.78 %	0.13 %	0.04 %	0.
THREAD 1.20.1	85.78 %	11.70 %	0.89 %	1.26 %	0.23 %	0.05 %	0.
THREAD 1.21.1	84.74 %	12.21 %	1.36 %	1.38 %	0.17 %	0.03 %	0.
THREAD 1.22.1	81.89 %	12.40 %	3.98 %	1.36 %	0.23 %	0.05 %	0.
THREAD 1.23.1	90.23 %	4.68 %	4.08 %	0.50 %	0.35 %	0.07 %	0.
THREAD 1.24.1	86.30 %	8.28 %	4.08 %	0.91 %	0.28 %	0.05 %	0.
THREAD 1.25.1	89.76 %	5.41 %	3.90 %	0.60 %	0.21 %	0.03 %	0.
THREAD 1.26.1	87.48 %	7.83 %	3.39 %	0.87 %	0.28 %	0.04 %	0.
THREAD 1.27.1	86.49 %	8.73 %	3.44 %	0.99 %	0.23 %	0.03 %	0.
Total	2,419.37 %	202.16 %	46.04 %	22.03 %	6.08 %	1.55 %	1.
Average	89.61 %	7.49 %	1.71 %	0.82 %	0.23 %	0.06 %	0.
Maximum	99.03 %	12.45 %	4.08 %	1.38 %	0.39 %	0.13 %	0.
Minimum	81.89 %	0.18 %	0.43 %	0.00 %	0.11 %	0.03 %	0.
StDev	3.79 %	3.15 %	1.21 %	0.35 %	0.07 %	0.02 %	0.
Avg/Max	0.90	0.60	0.42	0.59	0.58	0.44	

Focus on the iterative part

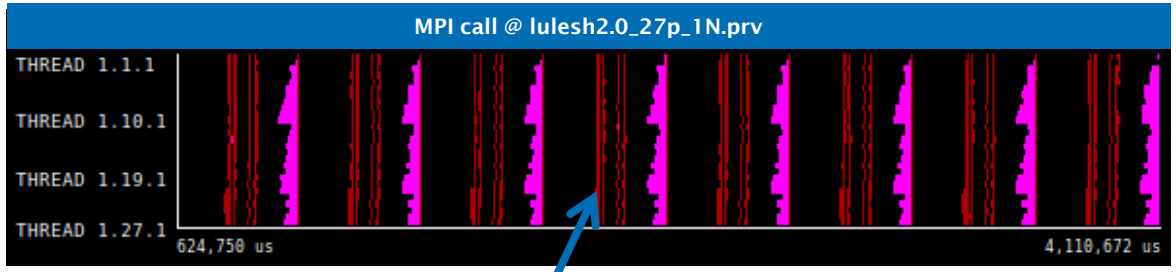


Drag & drop on this area
to zoom on the iterative region

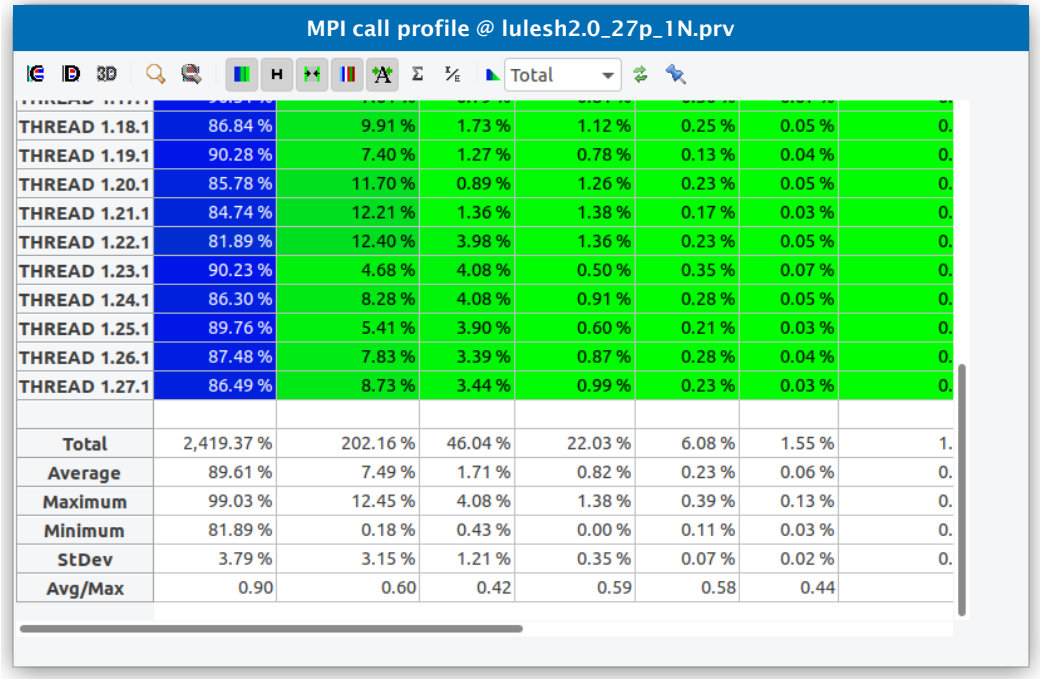


THREAD 1.18.1	86.84 %	9.91 %	1.73 %	1.12 %	0.25 %	0.05 %	0.
THREAD 1.19.1	90.28 %	7.40 %	1.27 %	0.78 %	0.13 %	0.04 %	0.
THREAD 1.20.1	85.78 %	11.70 %	0.89 %	1.26 %	0.23 %	0.05 %	0.
THREAD 1.21.1	84.74 %	12.21 %	1.36 %	1.38 %	0.17 %	0.03 %	0.
THREAD 1.22.1	81.89 %	12.40 %	3.98 %	1.36 %	0.23 %	0.05 %	0.
THREAD 1.23.1	90.23 %	4.68 %	4.08 %	0.50 %	0.35 %	0.07 %	0.
THREAD 1.24.1	86.30 %	8.28 %	4.08 %	0.91 %	0.28 %	0.05 %	0.
THREAD 1.25.1	89.76 %	5.41 %	3.90 %	0.60 %	0.21 %	0.03 %	0.
THREAD 1.26.1	87.48 %	7.83 %	3.39 %	0.87 %	0.28 %	0.04 %	0.
THREAD 1.27.1	86.49 %	8.73 %	3.44 %	0.99 %	0.23 %	0.03 %	0.
Total	2,419.37 %	202.16 %	46.04 %	22.03 %	6.08 %	1.55 %	1.
Average	89.61 %	7.49 %	1.71 %	0.82 %	0.23 %	0.06 %	0.
Maximum	99.03 %	12.45 %	4.08 %	1.38 %	0.39 %	0.13 %	0.
Minimum	81.89 %	0.18 %	0.43 %	0.00 %	0.11 %	0.03 %	0.
StDev	3.79 %	3.15 %	1.21 %	0.35 %	0.07 %	0.02 %	0.
Avg/Max	0.90	0.60	0.42	0.59	0.58	0.44	

Recalculate efficiency of iterative region

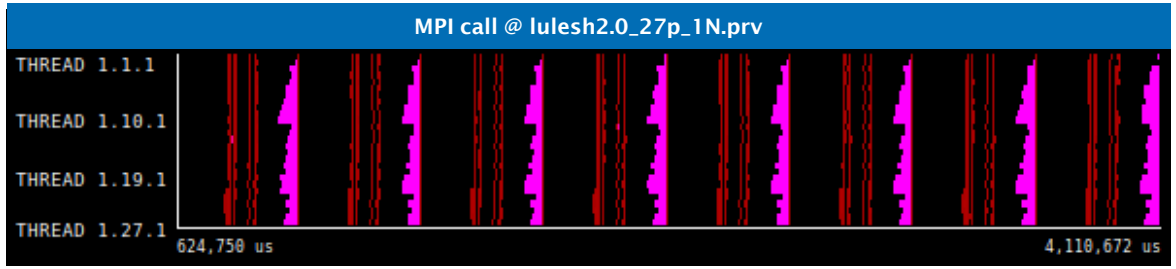


Right click → Copy



THREAD 1.18.1	86.84 %	9.91 %	1.73 %	1.12 %	0.25 %	0.05 %	0.
THREAD 1.19.1	90.28 %	7.40 %	1.27 %	0.78 %	0.13 %	0.04 %	0.
THREAD 1.20.1	85.78 %	11.70 %	0.89 %	1.26 %	0.23 %	0.05 %	0.
THREAD 1.21.1	84.74 %	12.21 %	1.36 %	1.38 %	0.17 %	0.03 %	0.
THREAD 1.22.1	81.89 %	12.40 %	3.98 %	1.36 %	0.23 %	0.05 %	0.
THREAD 1.23.1	90.23 %	4.68 %	4.08 %	0.50 %	0.35 %	0.07 %	0.
THREAD 1.24.1	86.30 %	8.28 %	4.08 %	0.91 %	0.28 %	0.05 %	0.
THREAD 1.25.1	89.76 %	5.41 %	3.90 %	0.60 %	0.21 %	0.03 %	0.
THREAD 1.26.1	87.48 %	7.83 %	3.39 %	0.87 %	0.28 %	0.04 %	0.
THREAD 1.27.1	86.49 %	8.73 %	3.44 %	0.99 %	0.23 %	0.03 %	0.
Total	2,419.37 %	202.16 %	46.04 %	22.03 %	6.08 %	1.55 %	1.
Average	89.61 %	7.49 %	1.71 %	0.82 %	0.23 %	0.06 %	0.
Maximum	99.03 %	12.45 %	4.08 %	1.38 %	0.39 %	0.13 %	0.
Minimum	81.89 %	0.18 %	0.43 %	0.00 %	0.11 %	0.03 %	0.
StDev	3.79 %	3.15 %	1.21 %	0.35 %	0.07 %	0.02 %	0.
Avg/Max	0.90	0.60	0.42	0.59	0.58	0.44	

Recalculate efficiency of iterative region



- In this case… same computation-to-communication ratio ➔ Negligible differences

- **It's good practice! ➔** Efficiency usually lowers due to disregarded long computations during initialization / finalization

- **Which is the lowest? ➔** Direct next analysis steps

Right click → Paste → Time

Parallel efficiency (Avg 89.13%)

Comm efficiency (Max 99.01%)

Load balance (Avg/Max 90%)

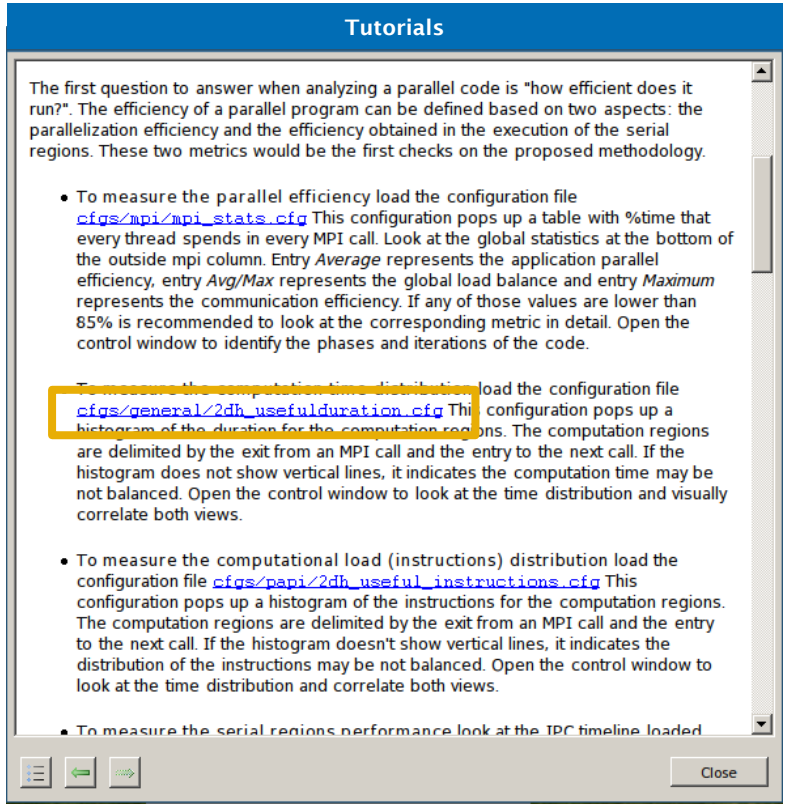
MPI call profile @ lulesh2.0_27p_1N.prv

Total

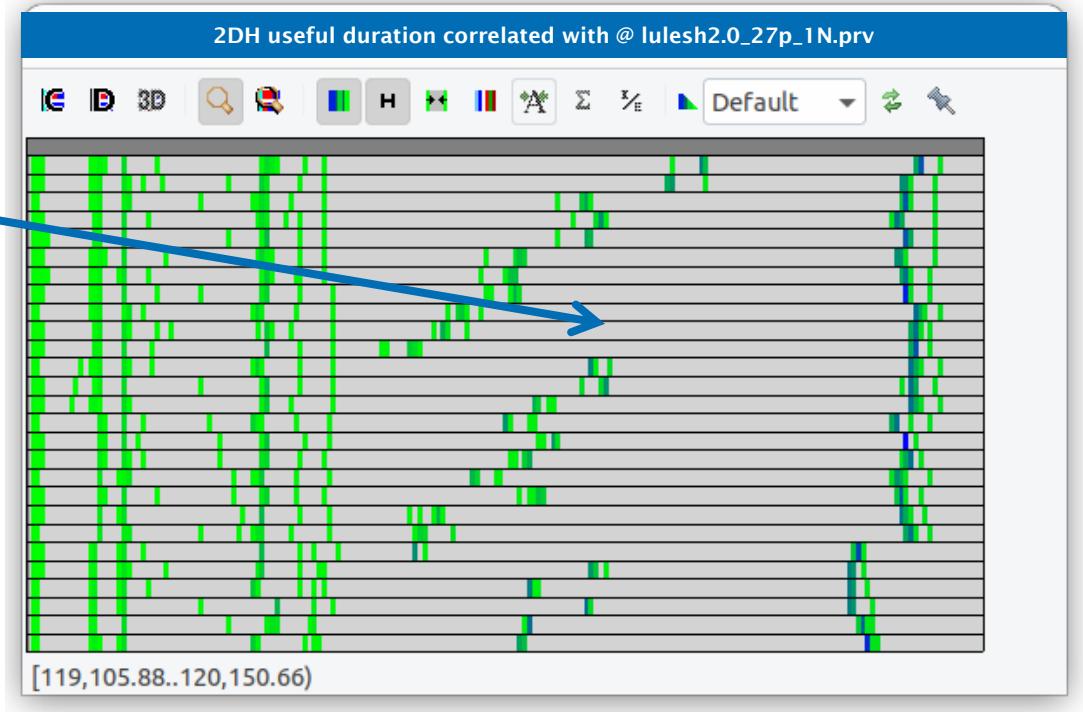
THREAD 1.18.1	86.09 %	11.76 %	1.79 %	0.25 %	0.05 %	0.05 %	0.
THREAD 1.19.1	89.95 %	8.60 %	1.23 %	0.13 %	0.03 %	0.04 %	0.
THREAD 1.20.1	85.18 %	13.61 %	0.89 %	0.23 %	0.05 %	0.04 %	0.
THREAD 1.21.1	83.91 %	14.47 %	1.35 %	0.17 %	0.03 %	0.05 %	0.
THREAD 1.22.1	[illegible]	14.59 %	4.01 %	0.23 %	0.04 %	0.04 %	0.
THREAD 1.23.1	89.83 %	5.49 %	4.21 %	0.35 %	0.06 %	0.04 %	0.
THREAD 1.24.1	85.68 %	9.78 %	4.16 %	0.29 %	0.04 %	0.04 %	0.
THREAD 1.25.1	89.35 %	6.41 %	3.95 %	0.21 %	0.03 %	0.04 %	0.
THREAD 1.26.1	87.03 %	9.21 %	3.38 %	0.28 %	0.04 %	0.05 %	0.
THREAD 1.27.1	85.93 %	10.29 %	3.46 %	0.23 %	0.03 %	0.05 %	0.
Total	2,406.50 %	237.40 %	46.96 %	6.12 %	1.51 %	1.25 %	0.
Average	89.13 %	8.79 %	1.74 %	0.23 %	0.06 %	0.05 %	0.
Maximum	99.01 %	14.59 %	4.21 %	0.40 %	0.13 %	0.06 %	0.
Minimum	81.08 %	0.23 %	0.42 %	0.11 %	0.03 %	0.04 %	0.
StDev	3.98 %	3.70 %	1.24 %	0.07 %	0.02 %	0.00 %	0.
Avg/Max	0.90	0.60	0.41	0.57	0.44	0.82	

"Low" load balance… should reflect on computation durations

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



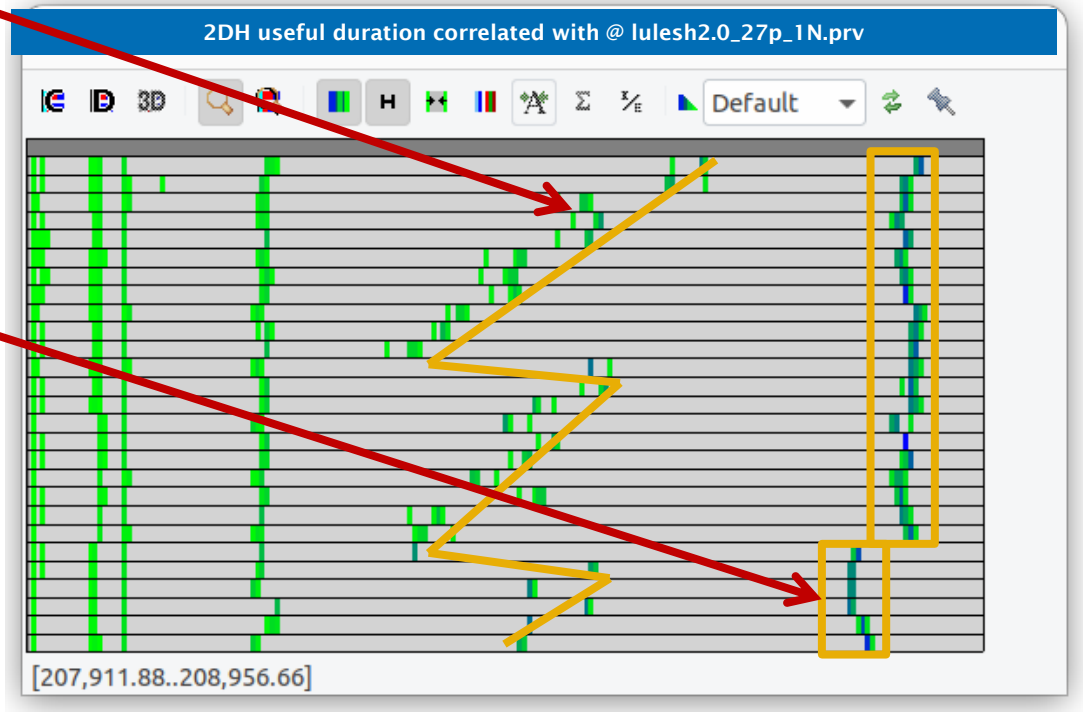
Right click → Paste → Time (Focus on iterative region)



Non-vertical lines reflect duration imbalance between processes

Lower MPI ranks are taking increasingly more time than higher ranks

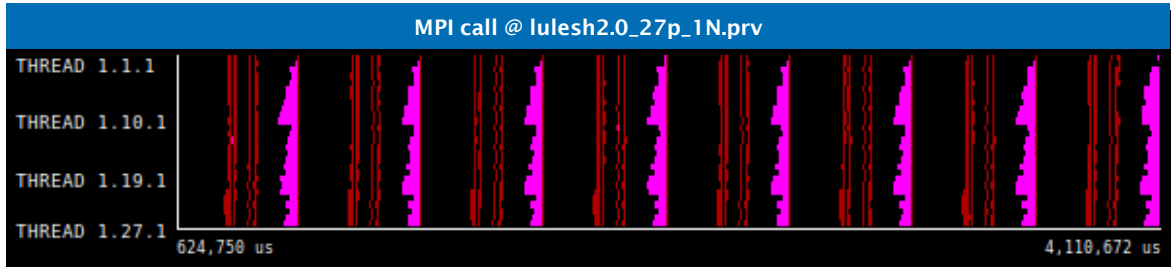
Longest computions in the whole program take less time on the last 6 ranks

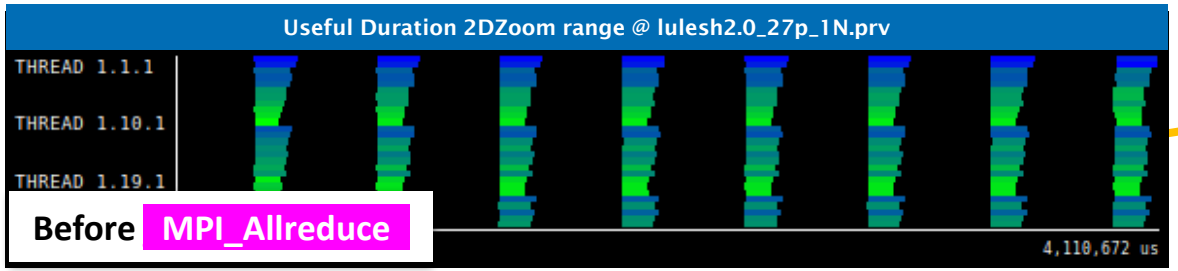


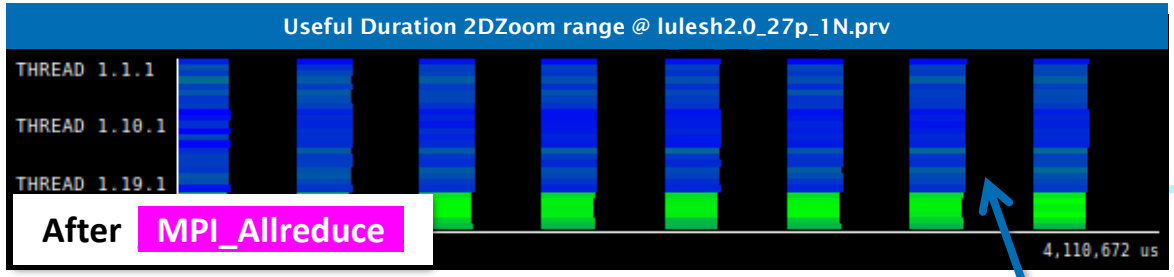
- Both effects seem "structured" rather than "random" because occur in groups of consecutive ranks

Where do the imbalances occur?

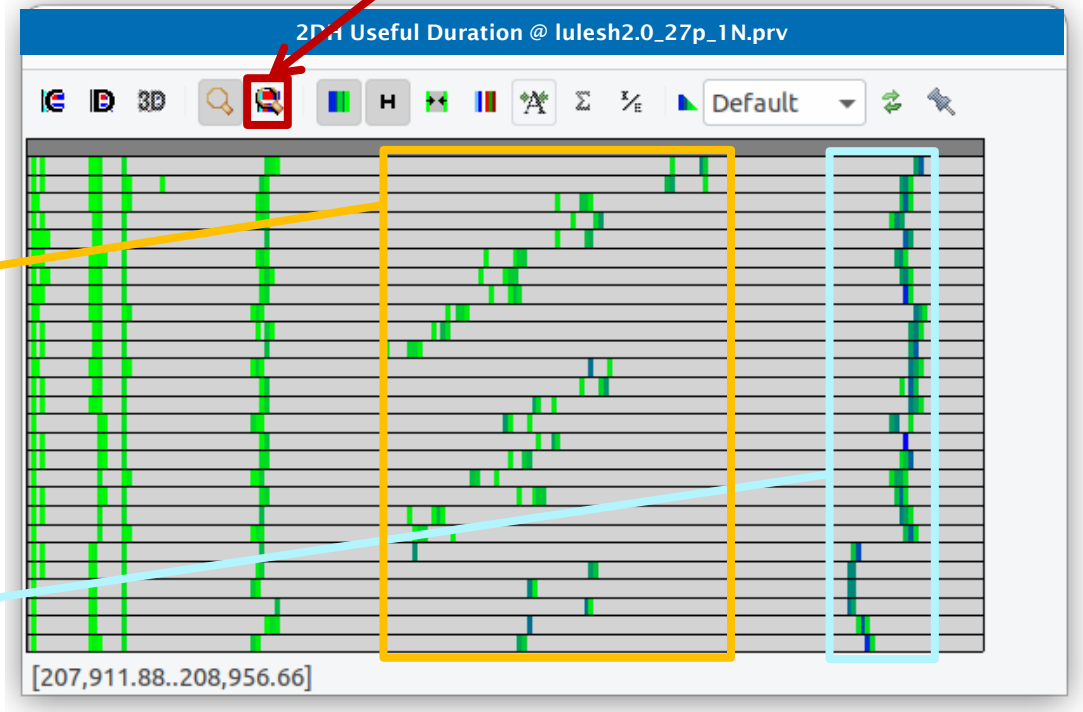
- **Slow** & **Fast** **simultaneously** → Imbalance paid later in MPI







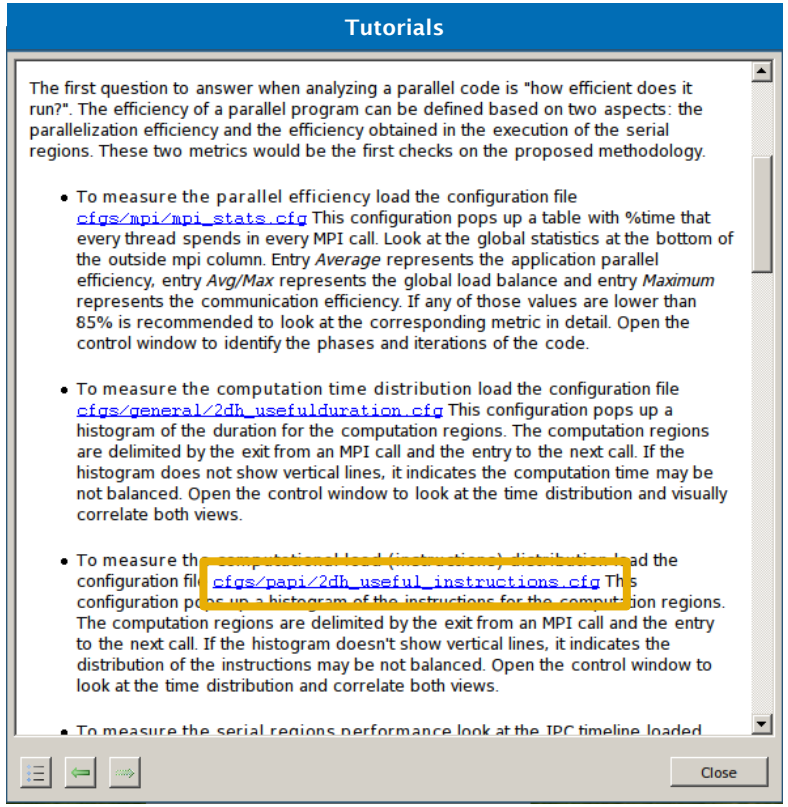
Click on "Open Filtered Control Window" → Select the interesting areas



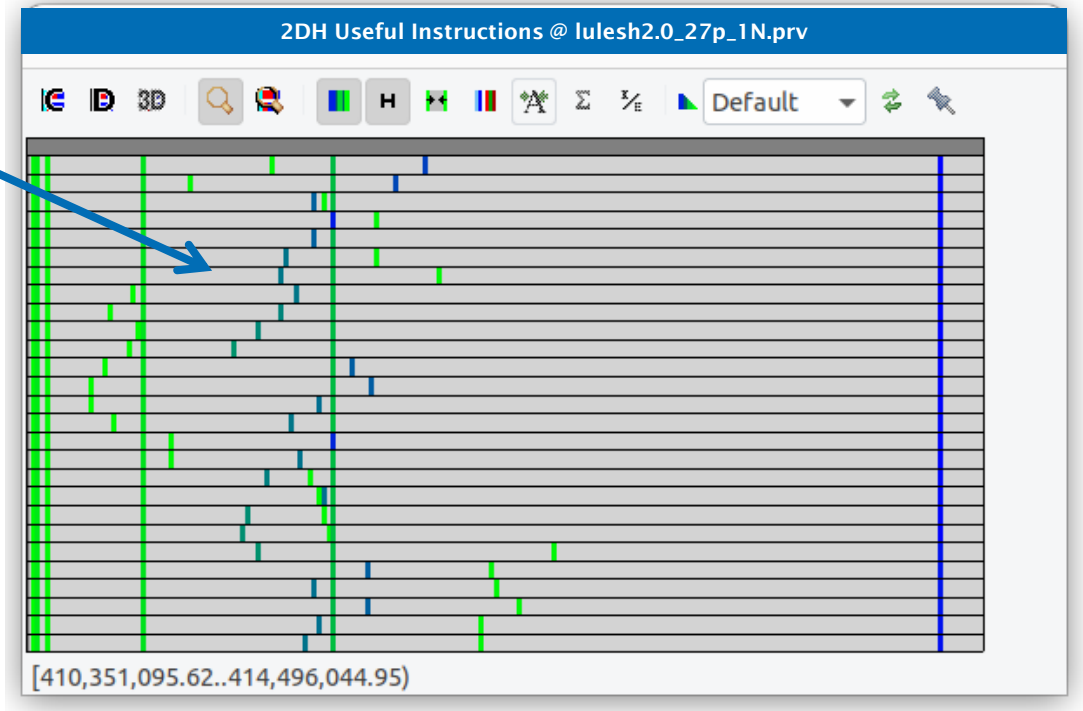
Plain colors? Right click → Fit Semantic Scale → Fit Both

Looking to blame: why the imbalances? Programmer's fault? System's fault?

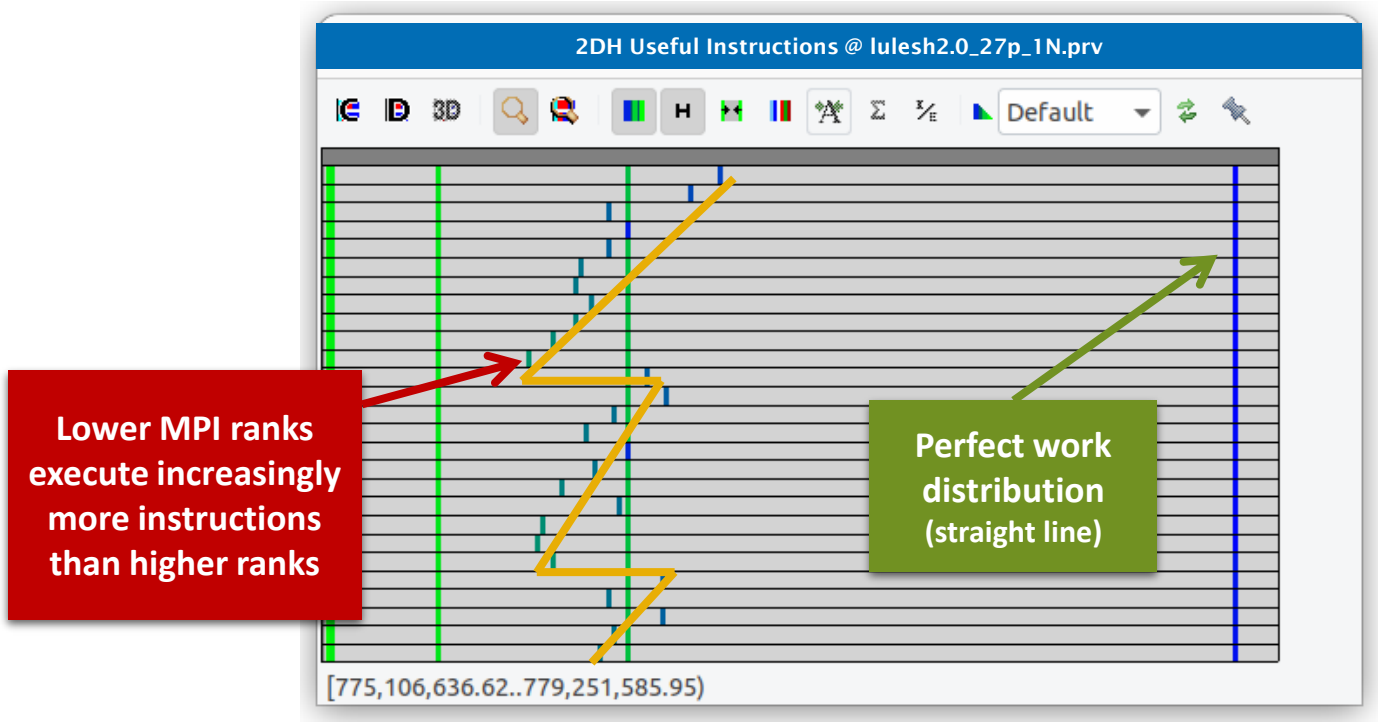
- Click on "2dh_useful_instructions.cfg" (3rd link) → Shows **amount of work in computing regions**



1. Right click → Paste → Time



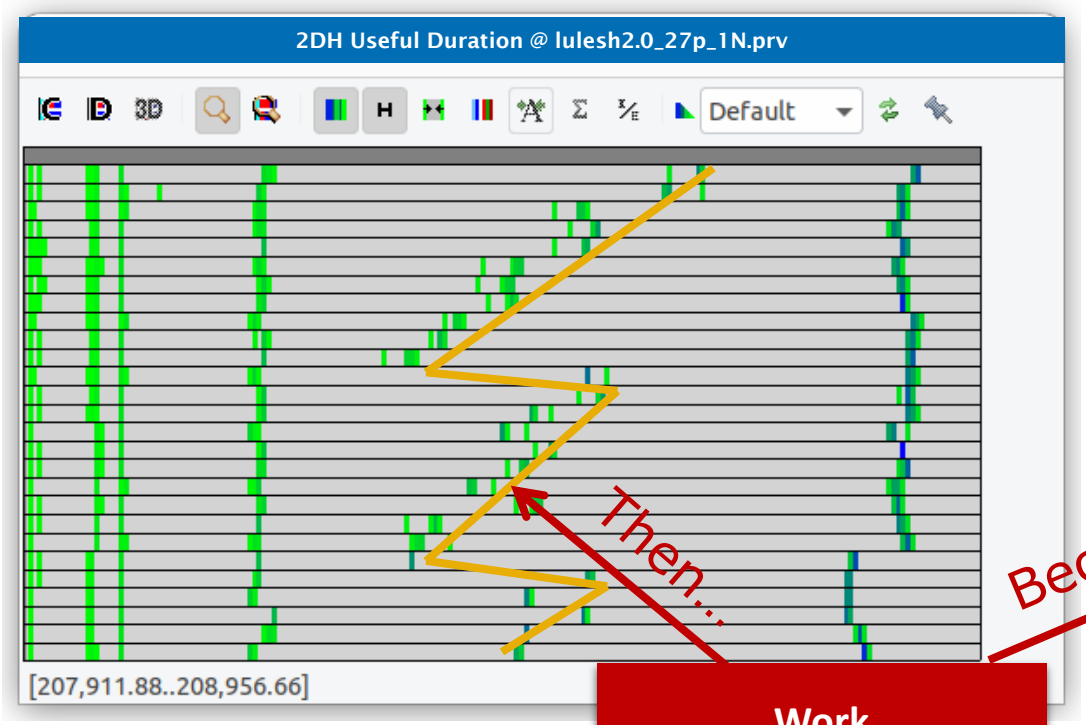
Non-vertical lines reflect work imbalance between processes



Visually correlate both time and instructions histograms

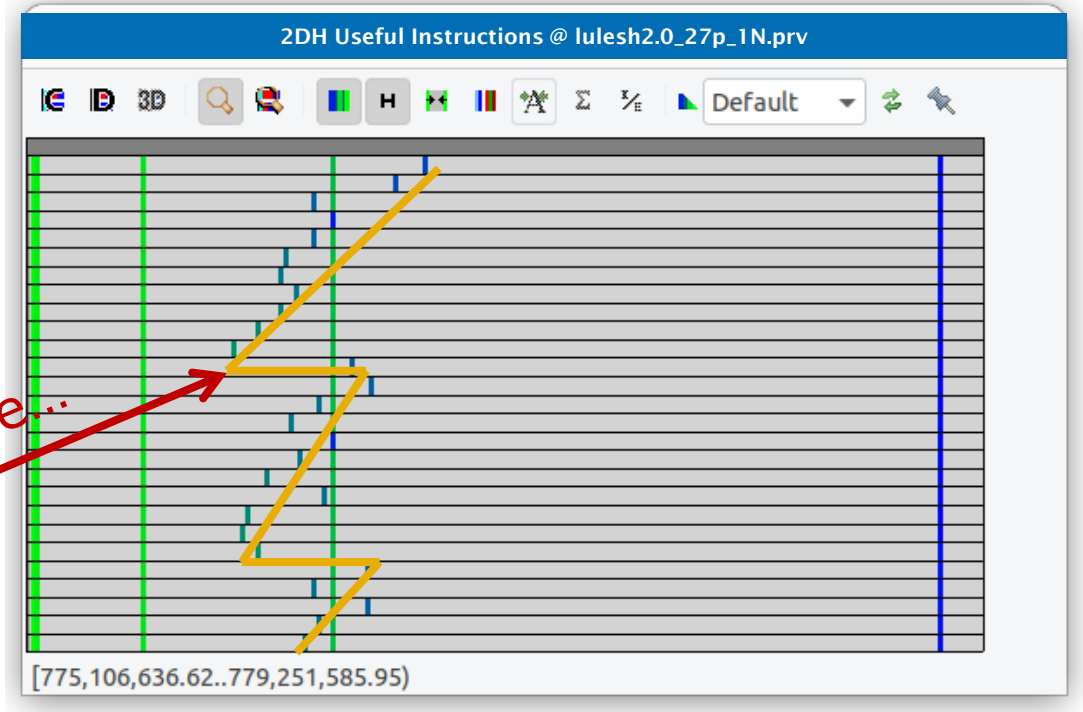
- **Similar shapes** → Work distribution determines time computing in the phase

Before **MPI_Allreduce**



Then…

Because…

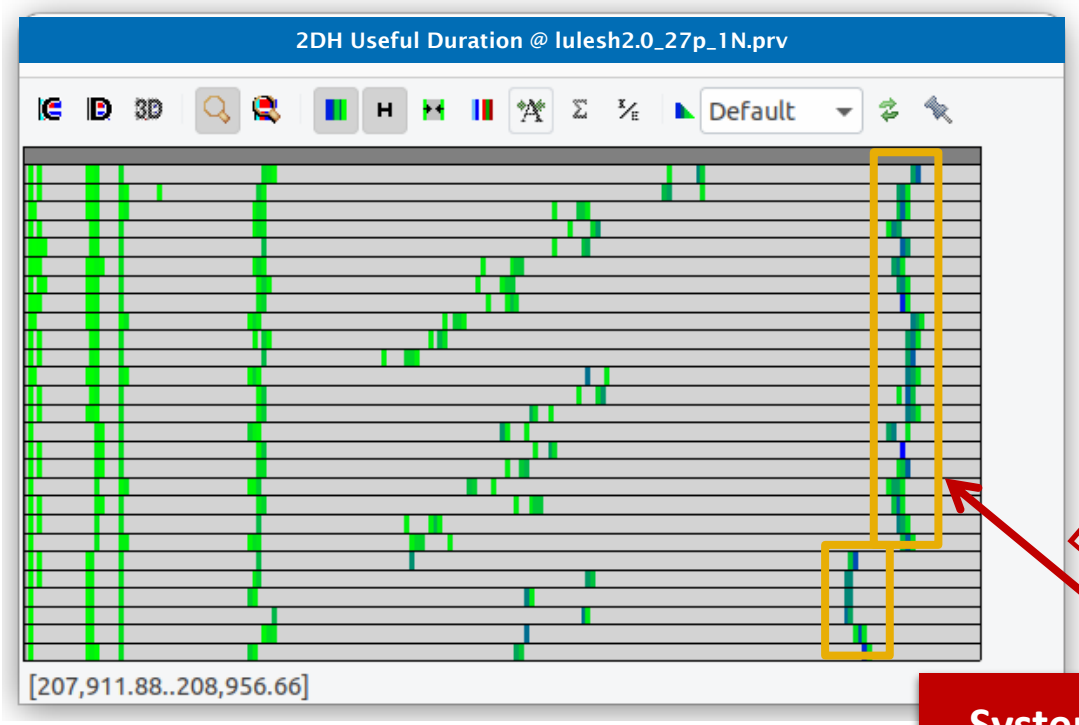


Work imbalance → Slow/fast processes

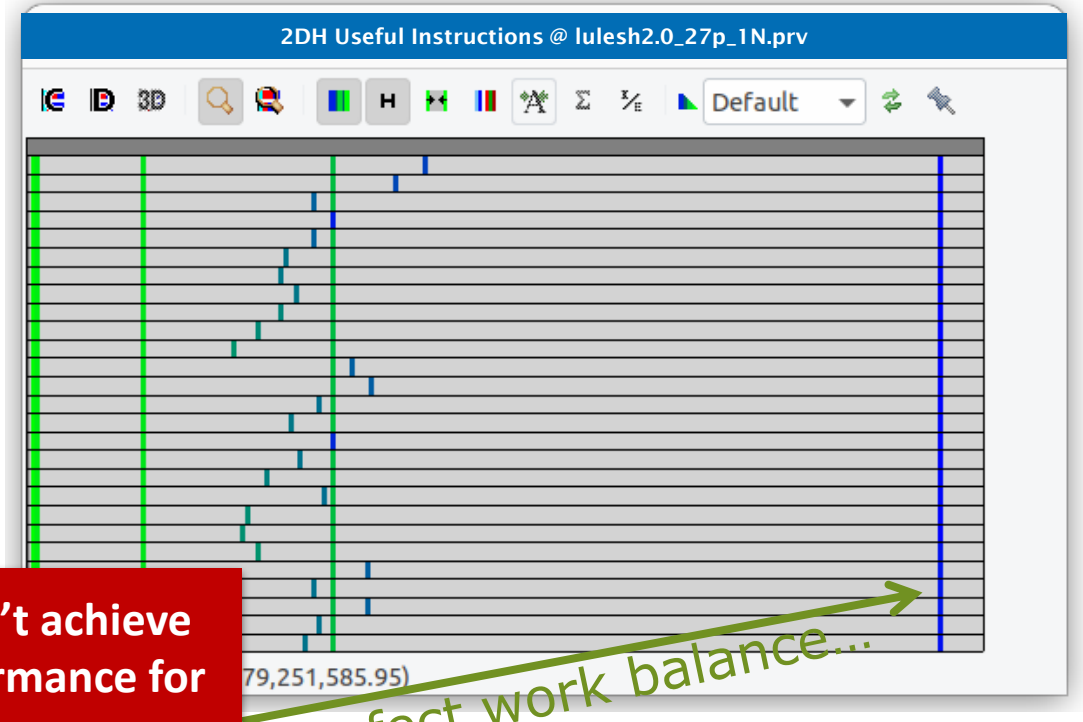
Visually correlate both time and instructions histograms

- **Different shapes** → System introduces performance variability in the phase

After **MPI_Allreduce**



But…



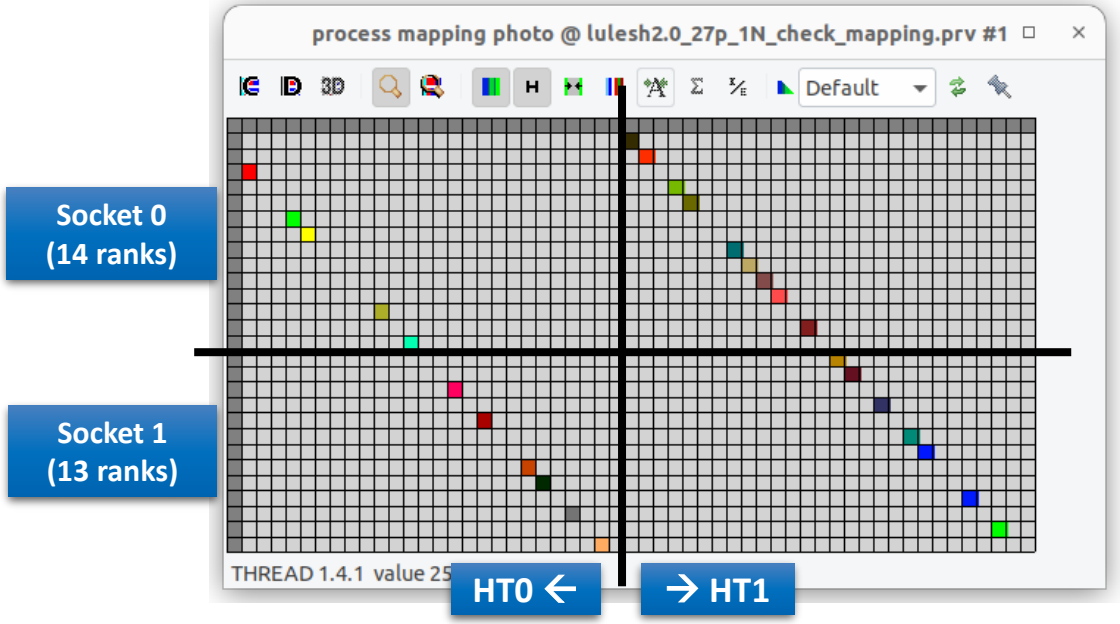
Perfect work balance…

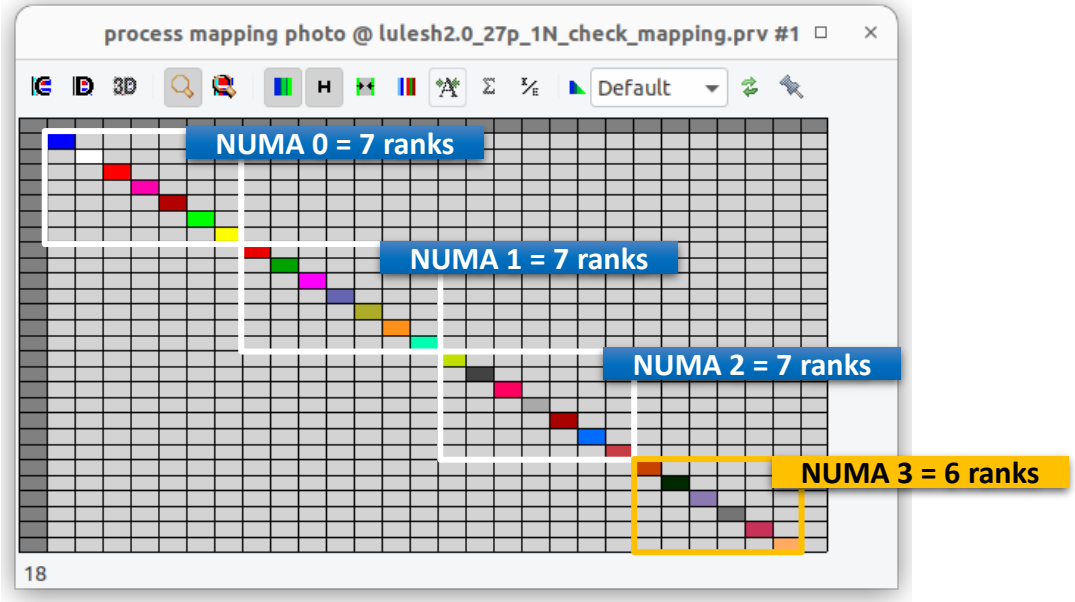
System can't achieve same performance for all processes

Impact of system's process mapping

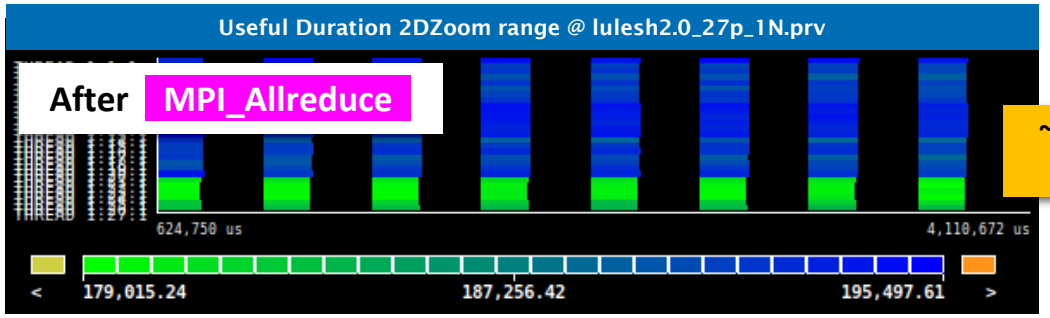
```
cm2> lscpu
Socket(s): 2
Core(s) per socket: 14
Thread(s) per core: 2
NUMA node0 CPU(s): 0-6,28-34
NUMA node1 CPU(s): 7-13,35-41
NUMA node2 CPU(s): 14-20,42-48
NUMA node3 CPU(s): 21-27,49-55
```

- 27 ranks is a tricky number to map… what can the system do?
  - **14/13 ranks per socket**
  - **No oversubscription on the same core**
  - **7/6 ranks per NUMA node**
    - **Last 6 MPI ranks fall on NUMA node 3, which is less populated than others**

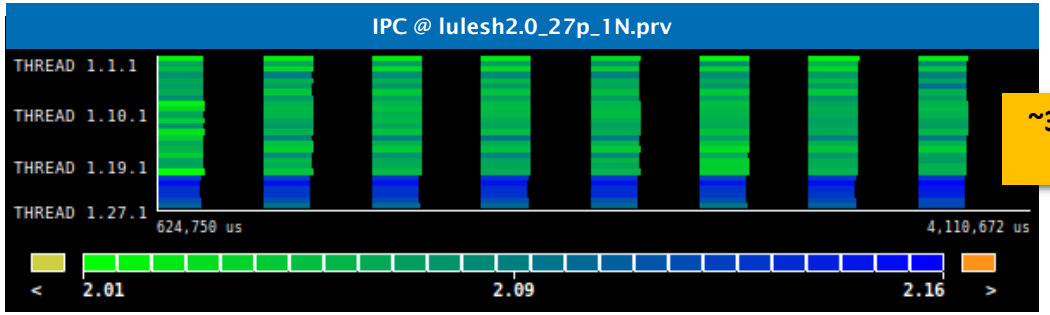




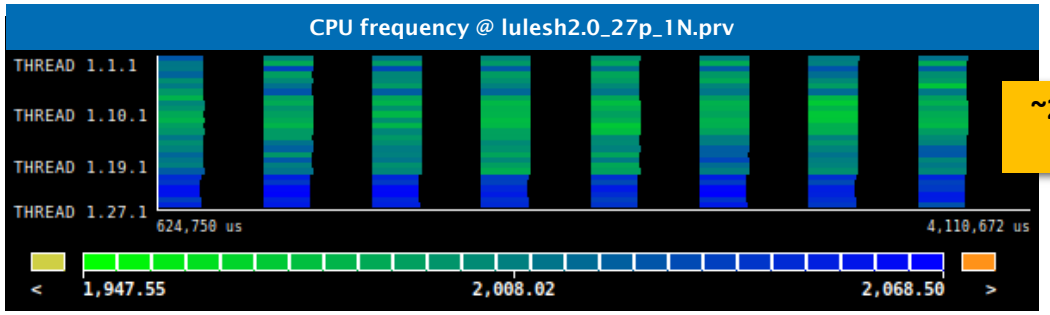
Impact of lower resource contention in NUMA node 3



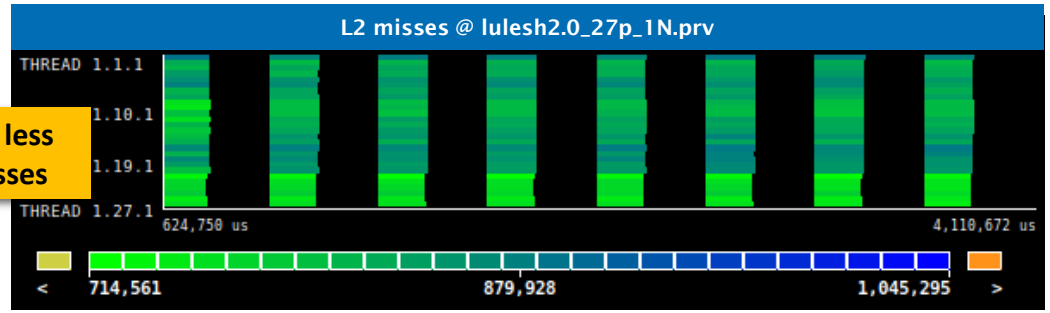
~5.8% less duration



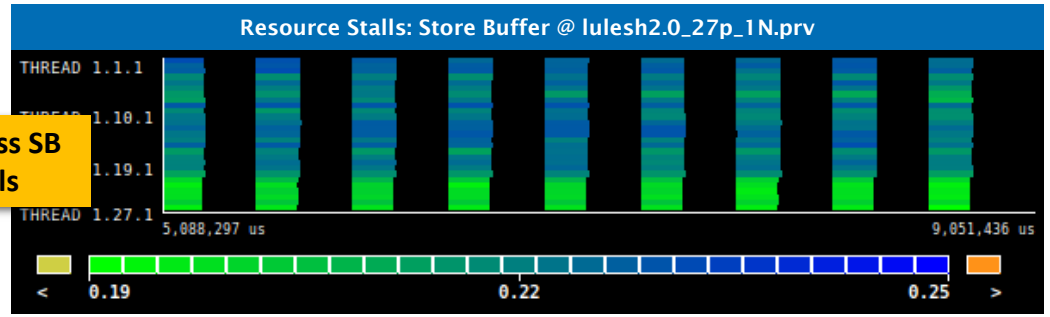
~3.4% more IPC



~2.45% more frequency

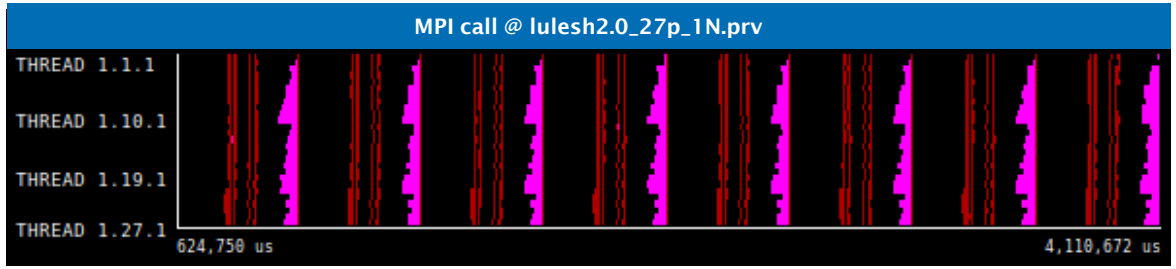


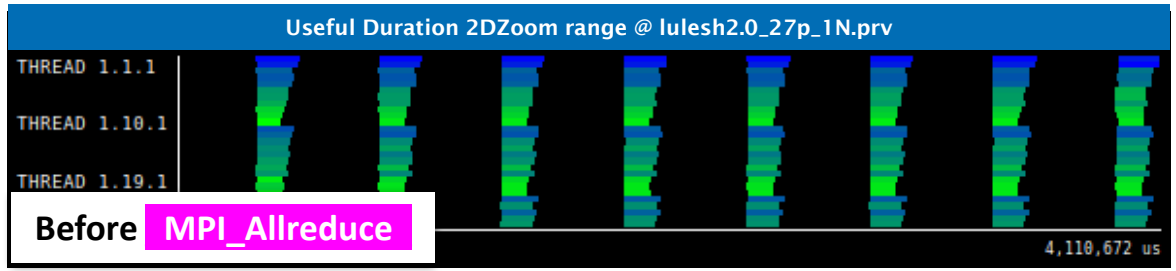
~8.6% less L2 misses

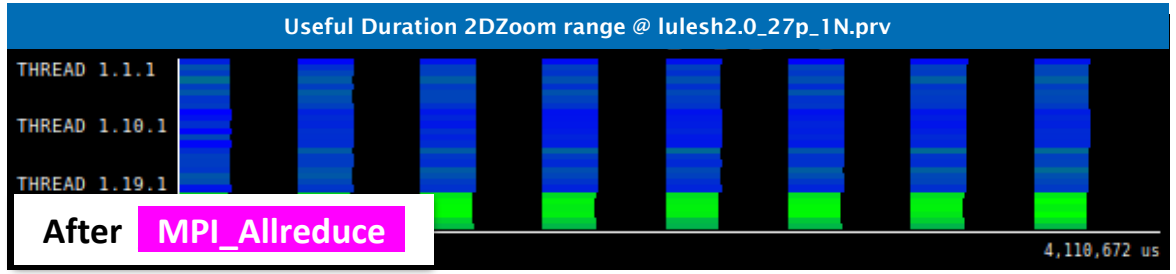


~3% less SB stalls

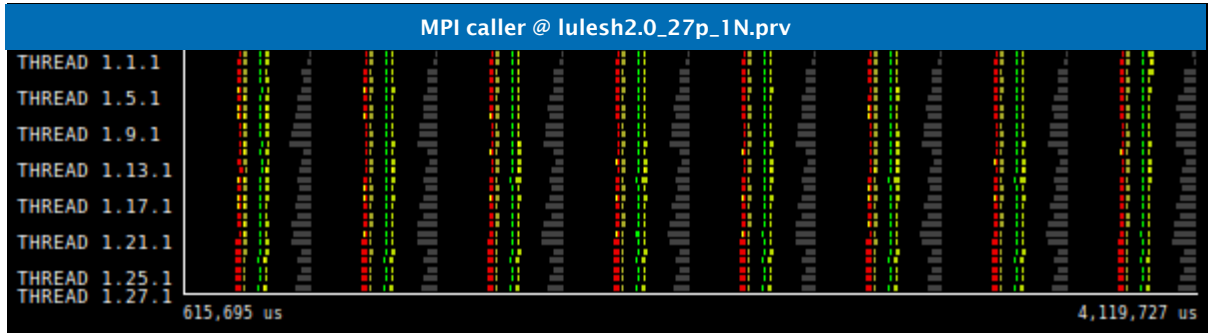
Where in the source code?

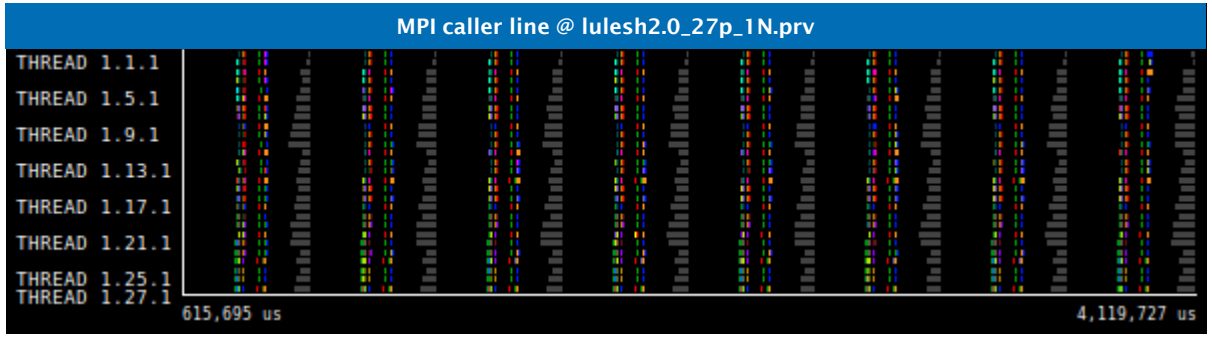






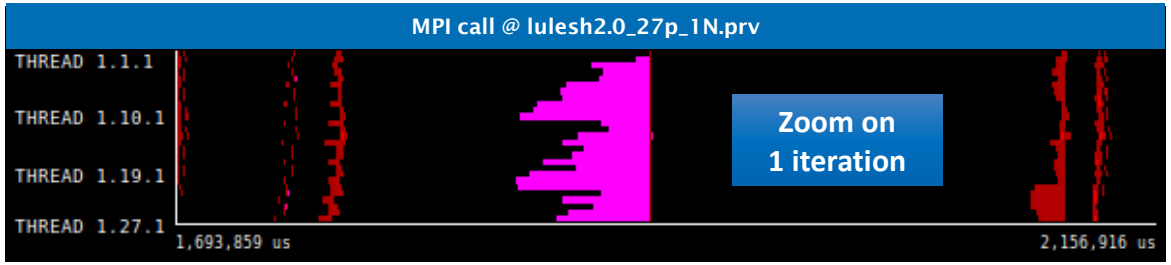
- Hints → Callstack references
  - **Views showing function name, file and line where calls to MPI occur**
  - **These bound the computing phases**

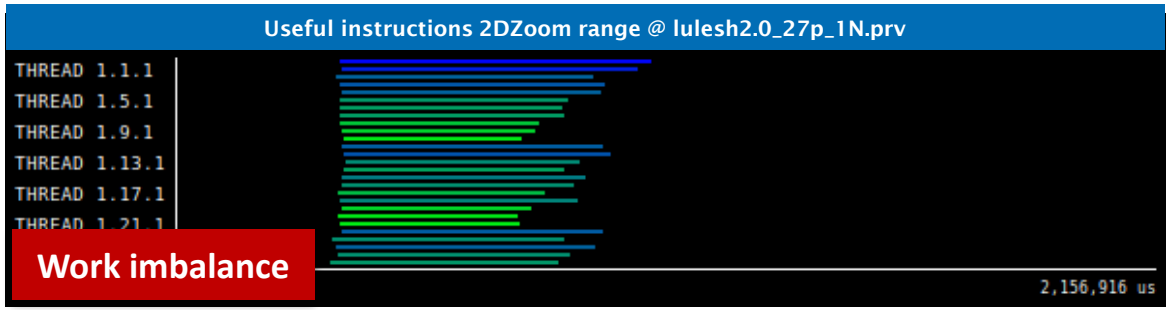


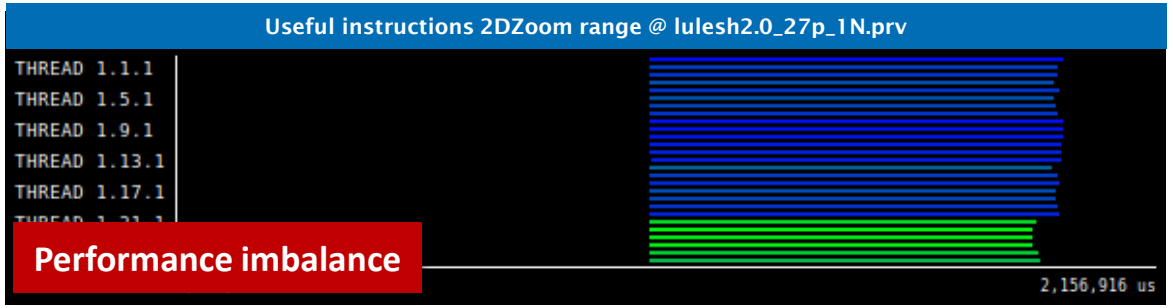


Where in the source code?

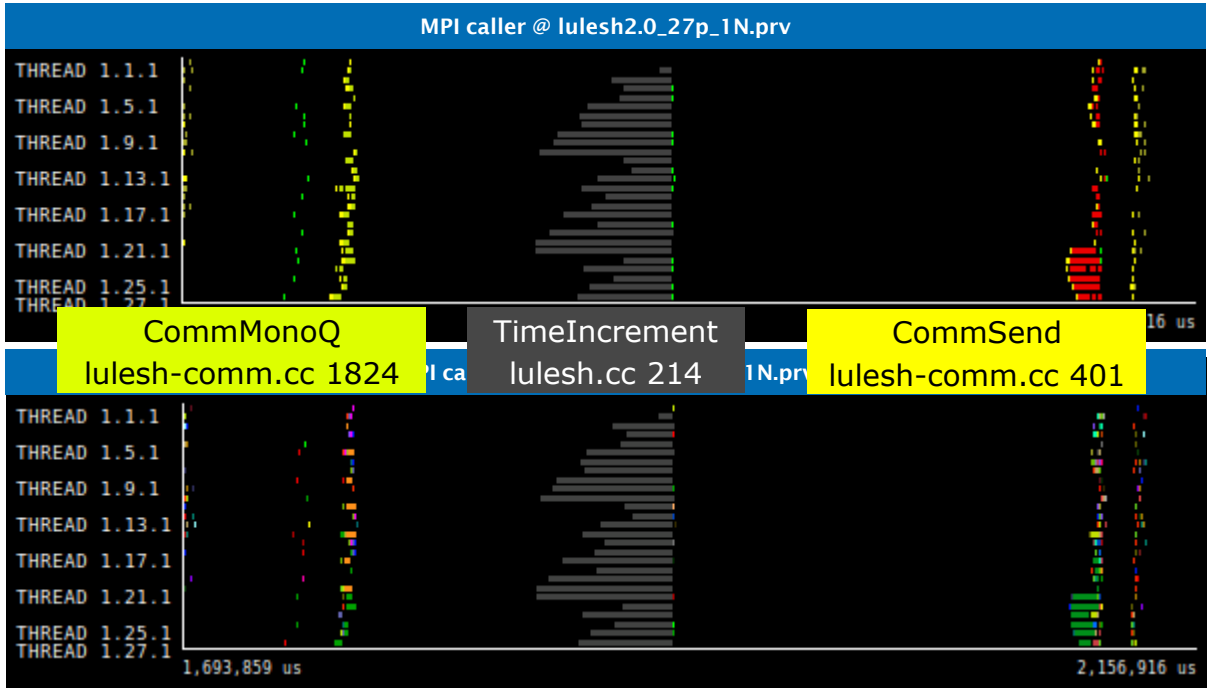
Remember to compile application with debug (-g) !



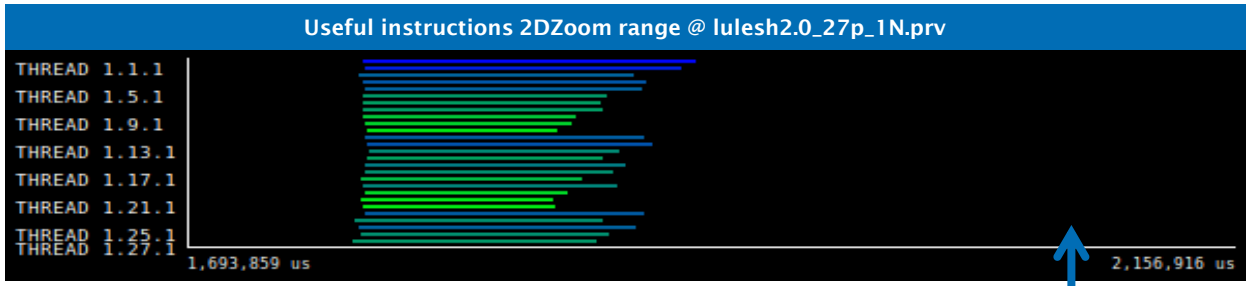




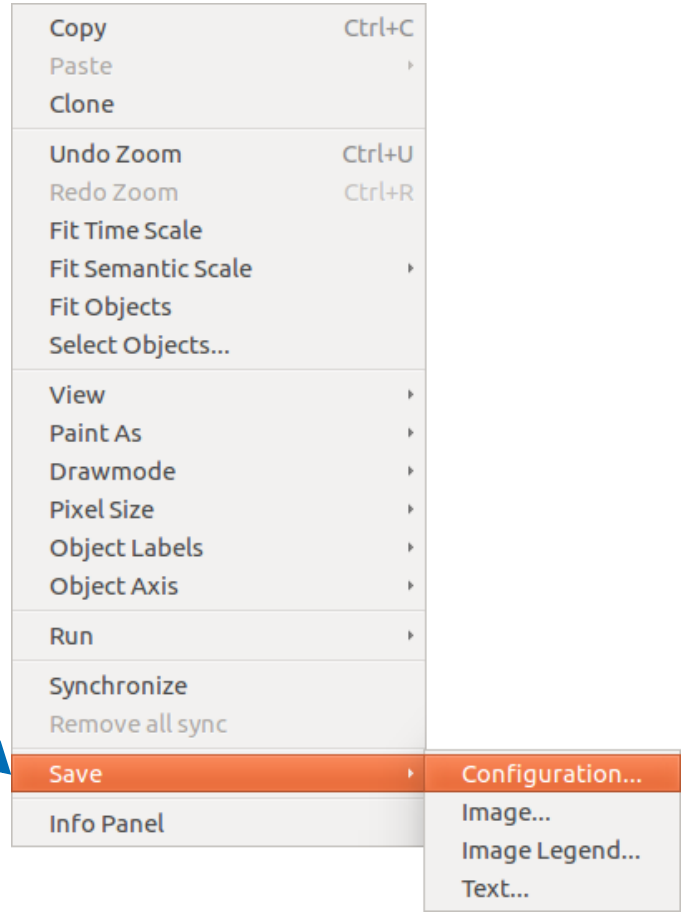
- Hints → Callstack references
  - **Views showing function name, file and line where calls to MPI occur**
  - **These bound the computing phases**



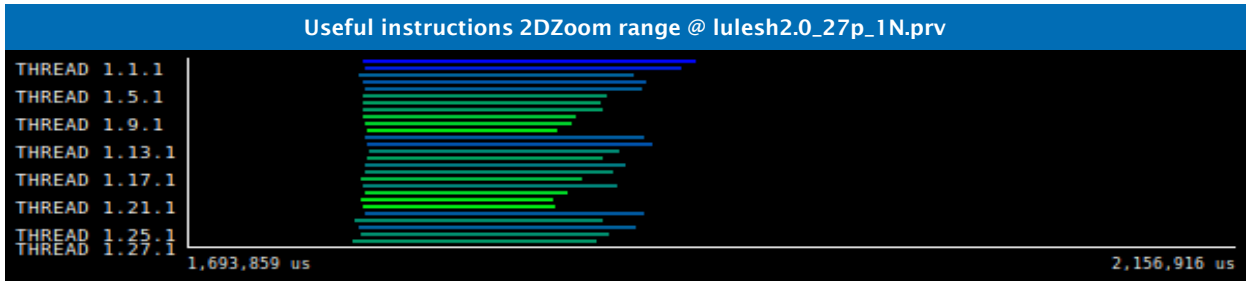
Save CFG's (method 1)



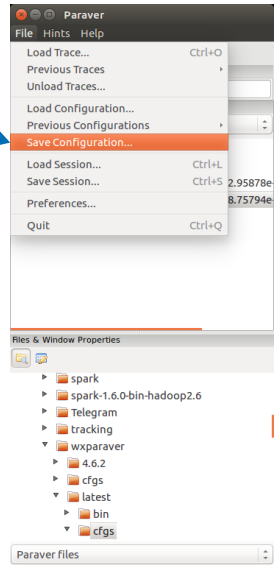
Right click on timeline

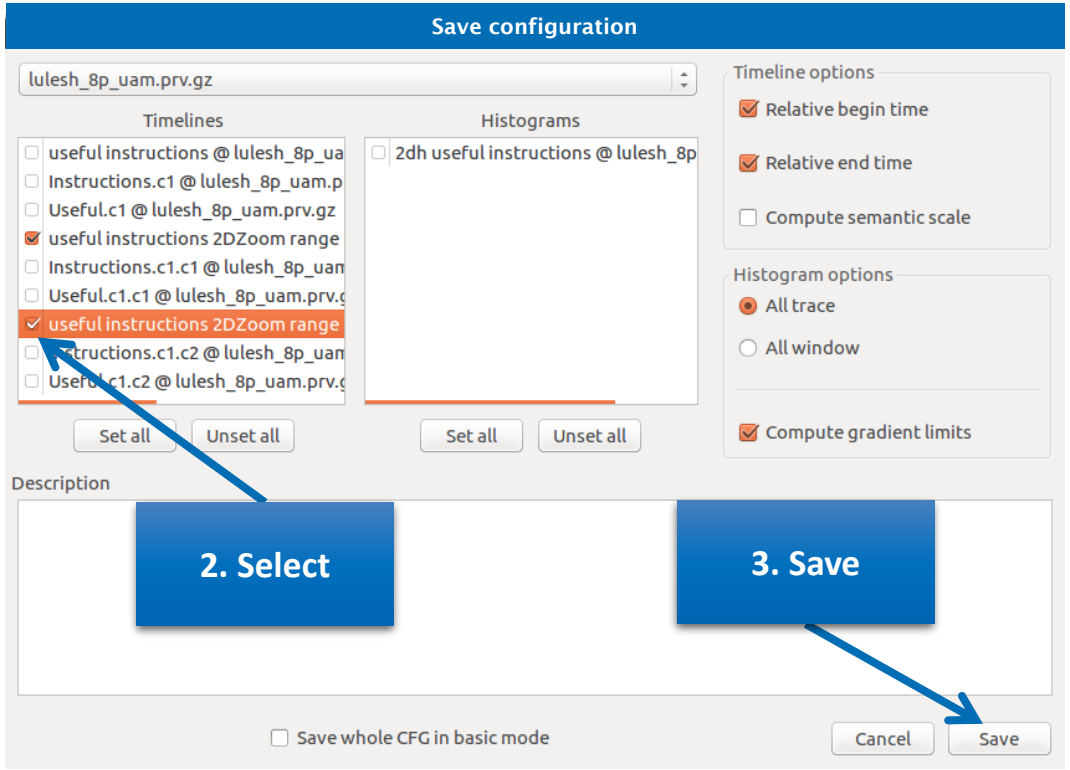


Save CFG's (method 2)



1. Main Paraver window



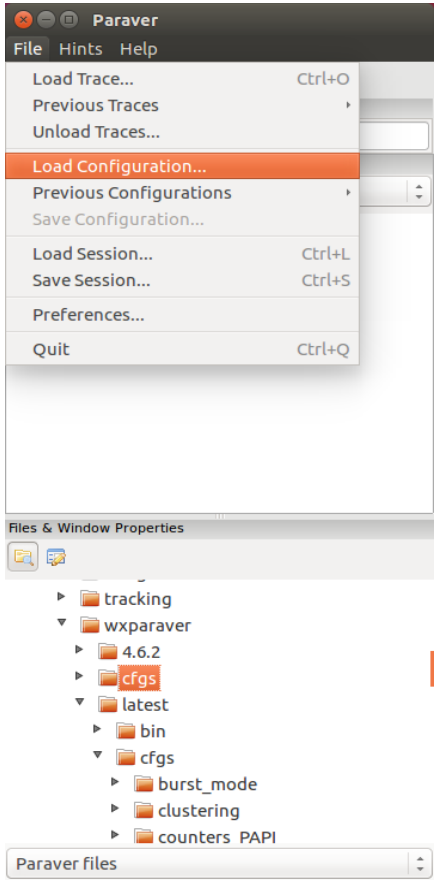


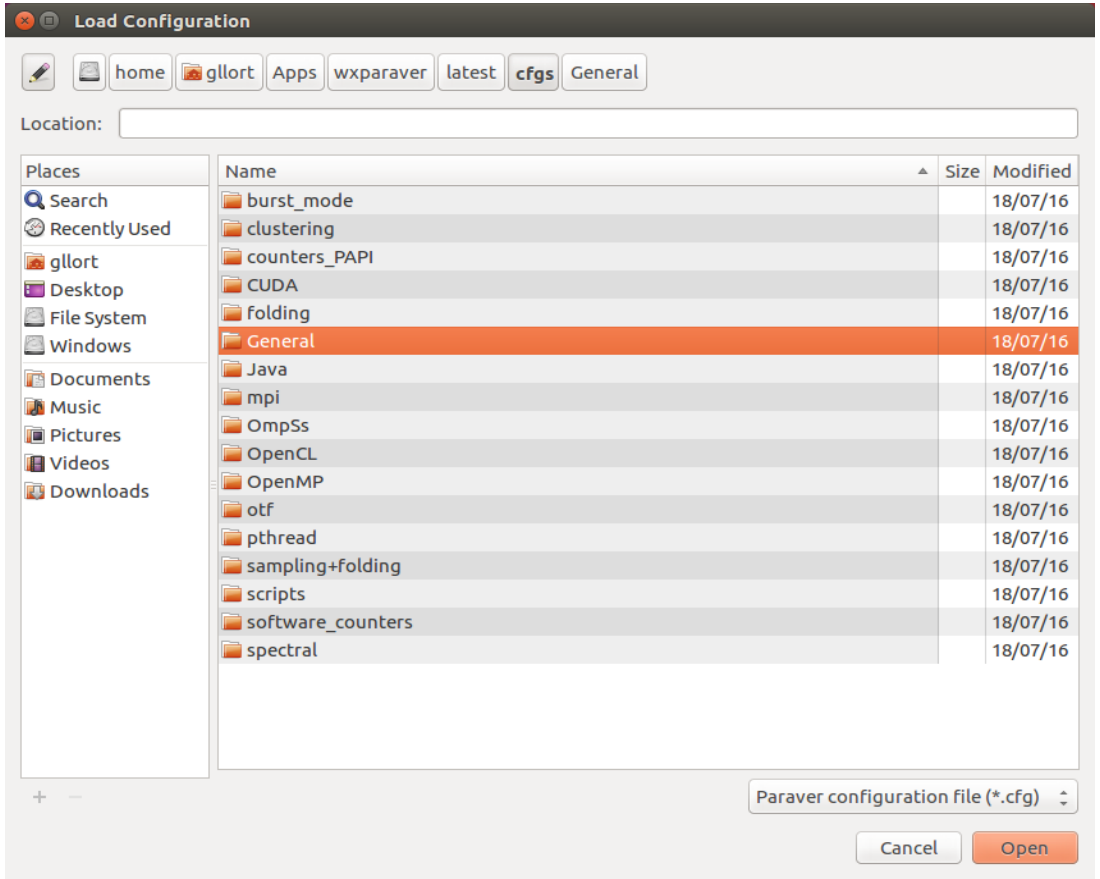
2. Select

3. Save

CFG's distribution

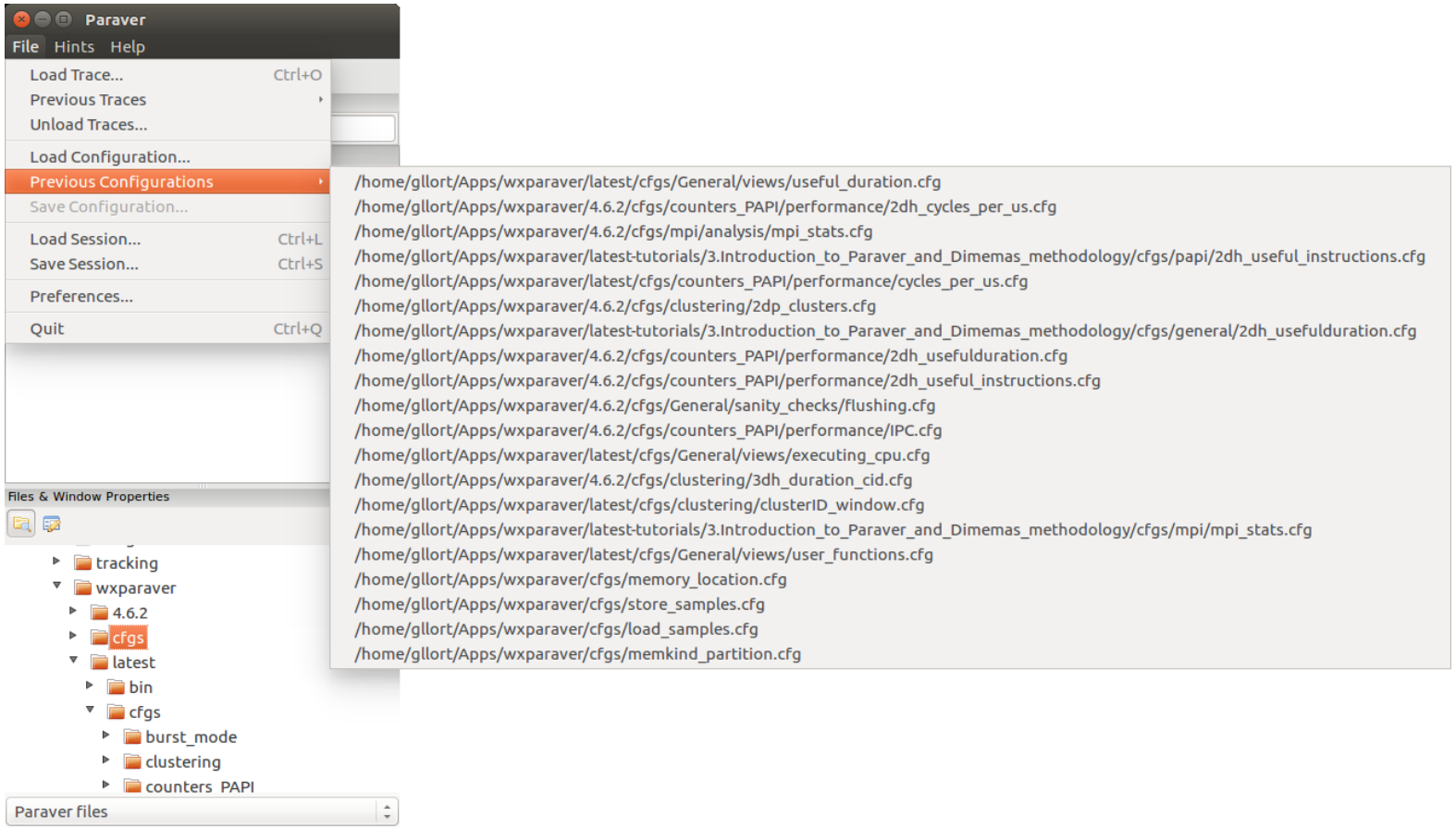
- Paraver comes with many included CFG's ➔ Apply any CFG to any trace!





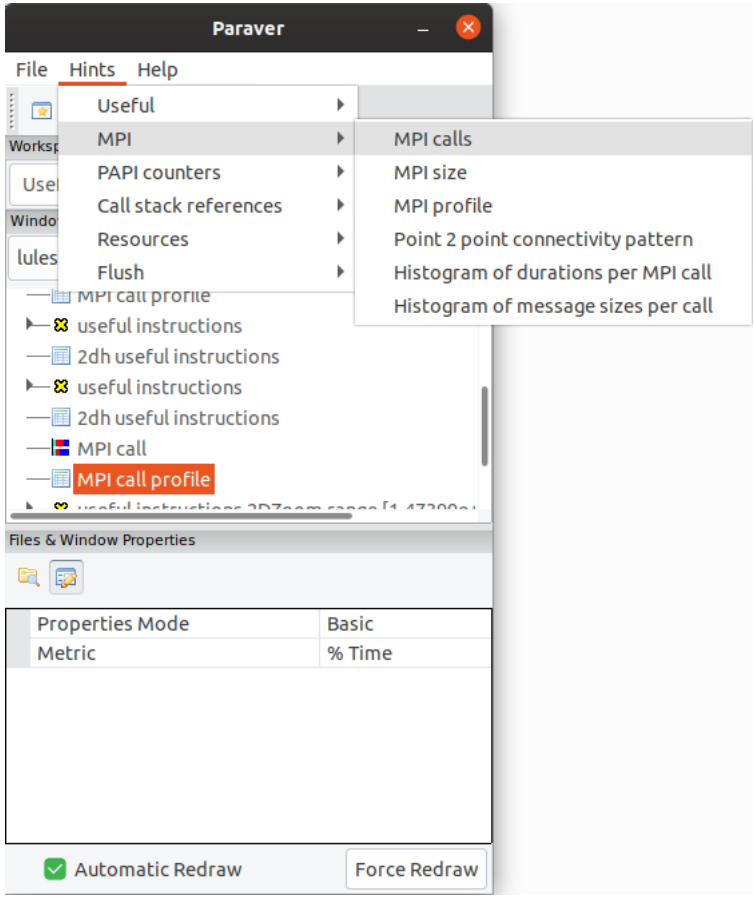
CFG's distribution

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



Hints: a good place to start!

- Suggested CFG's based on the contents of the trace



Do it on your code!

- Follow guidelines from slides 5-11 to get a trace of your own code
  - There are more examples of tracing scripts for different programming models at `$EXTRAE_HOME/share/examples`
- Follow guidelines from slides 19-34 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? Or IPC variations?
    - Where are things located in the source code?

Cluster-based analysis

Use clustering analysis

- Run the clustering tool

```
cm2> cd $HOME/bsctools/clustering
cm2> module use /lrz/sys/courses/vihps/2024/modulefiles
cm2> module load clustering-suite
cm2> BurstClustering -d cluster.xml \
        -i ../extrae/lulesh2.0_27p_1N.prv \
        -o lulesh2.0_27p_1N.clustered.prv
```

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

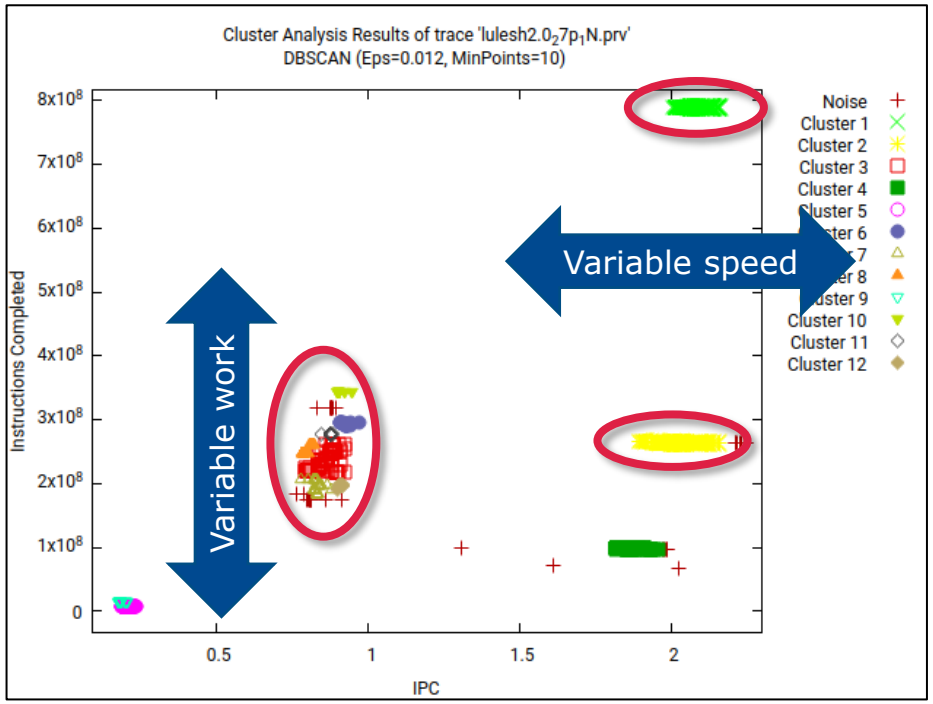
```
cm2> ls $HOME/bsctools/traces/lulesh2.0_27p_1N.prv
```

Cluster-based analysis

- Check the resulting scatter plot

```
cm2> gnuplot lulesh2.0_27p_1N.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**

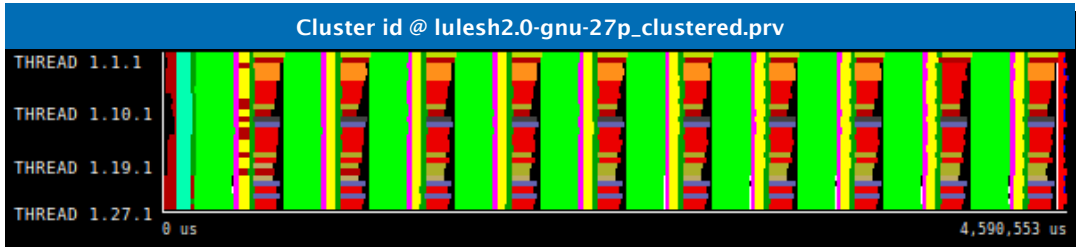


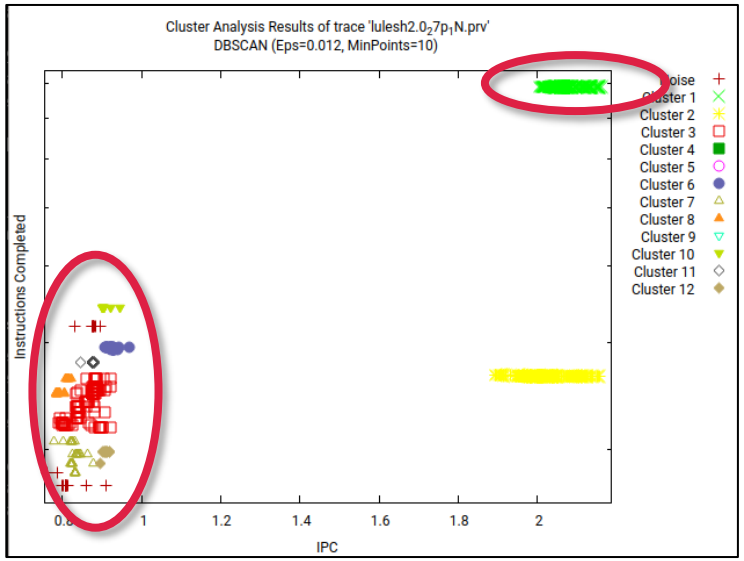
Correlating scatter plot and timeline

- Open the clustered trace with Paraver and look at the clusters

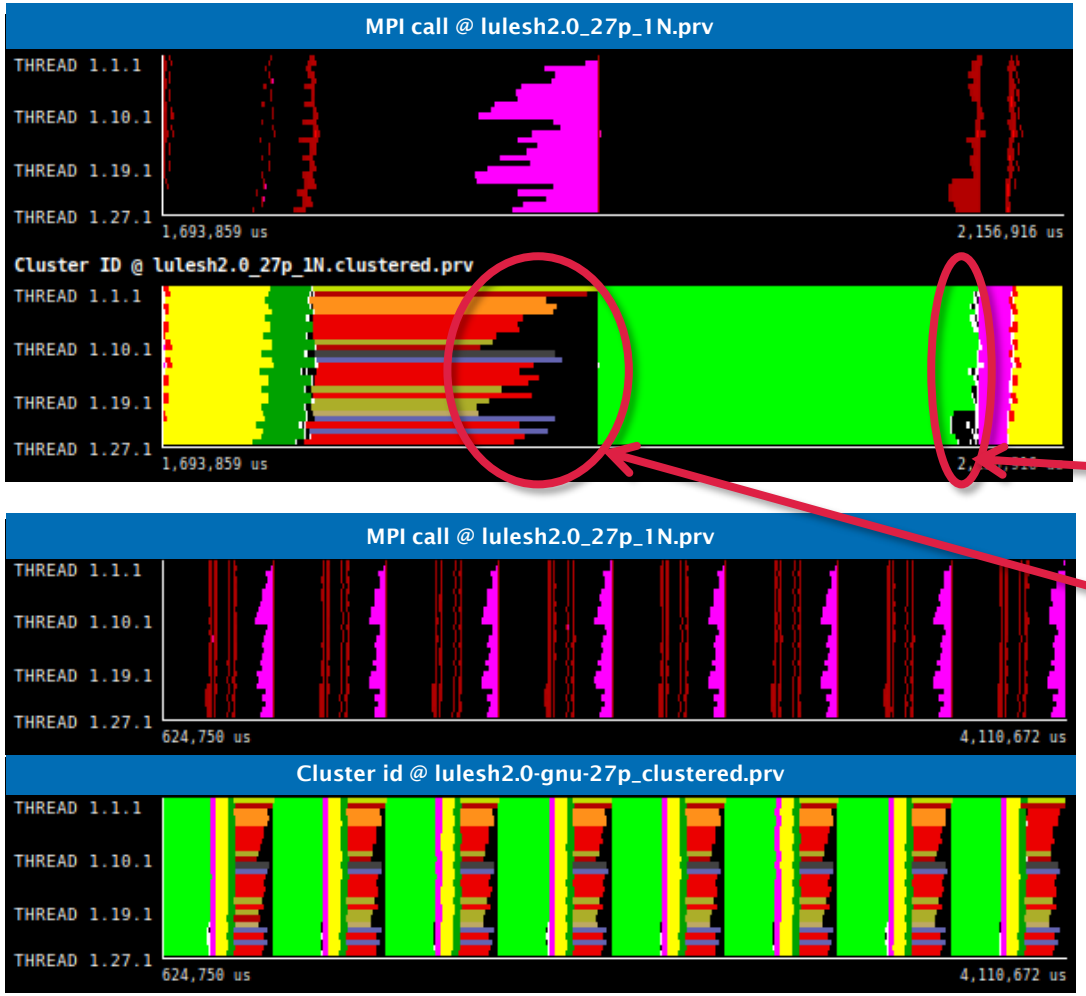
```
laptop> scp <USER>@lxlogin1.lrz.de:bsctools/clustering/*.{pcf,prv,row} .
laptop> <path-to>/bin/wxparaver ./lulesh2.0-intel_27p_clustered.prv
```

- Find where the elongated clusters occur over time
  - File → Load configuration
    → `$HOME/paraver/cfgs/clustering/clusterID_window.cfg`

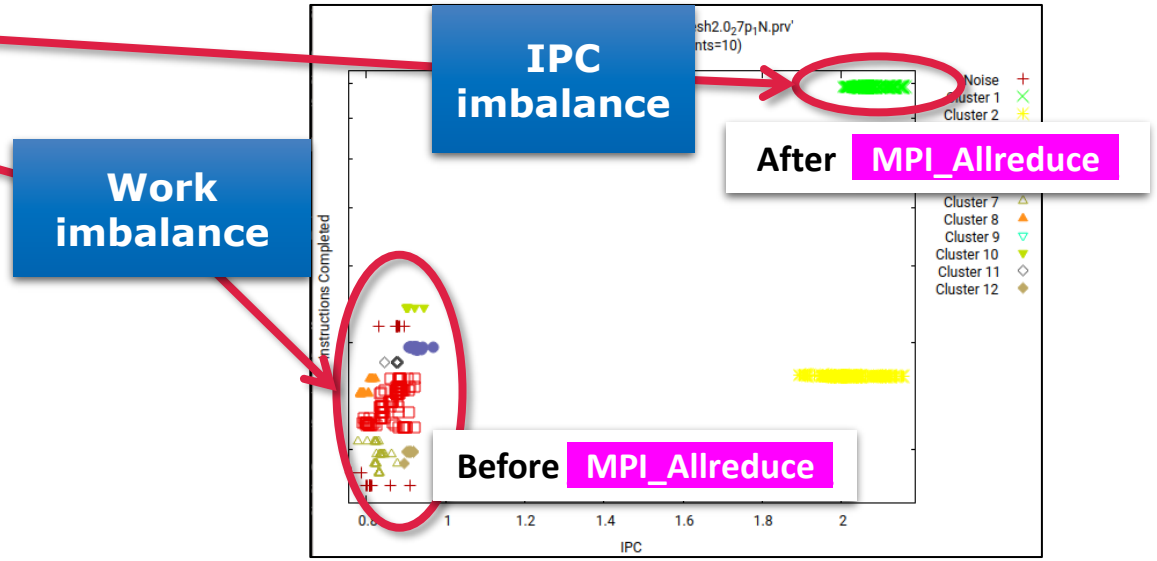




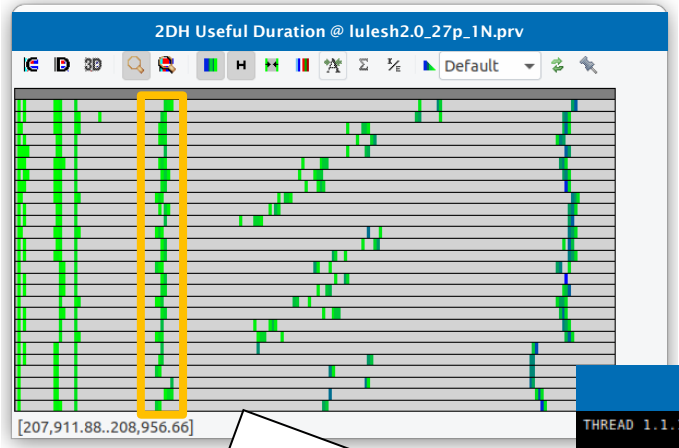
Correlating scatter plot and timeline

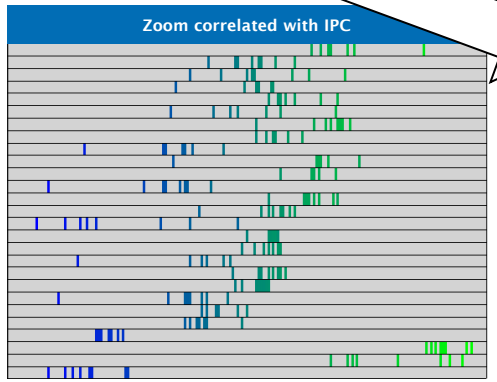


- Variable work / speed + Simultaneously at different processes = Imbalances
- Quickly exposed the same regions we identified before and the nature of their imbalance

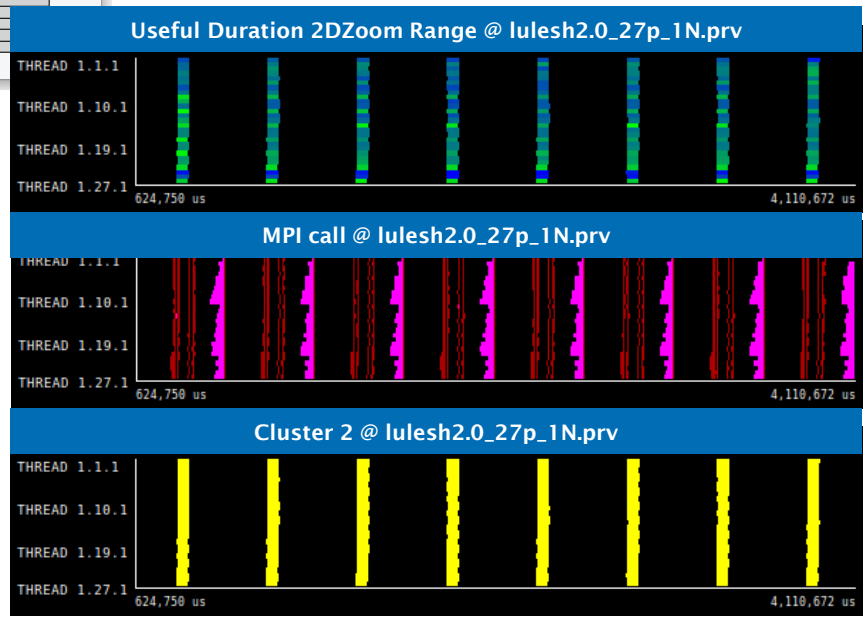


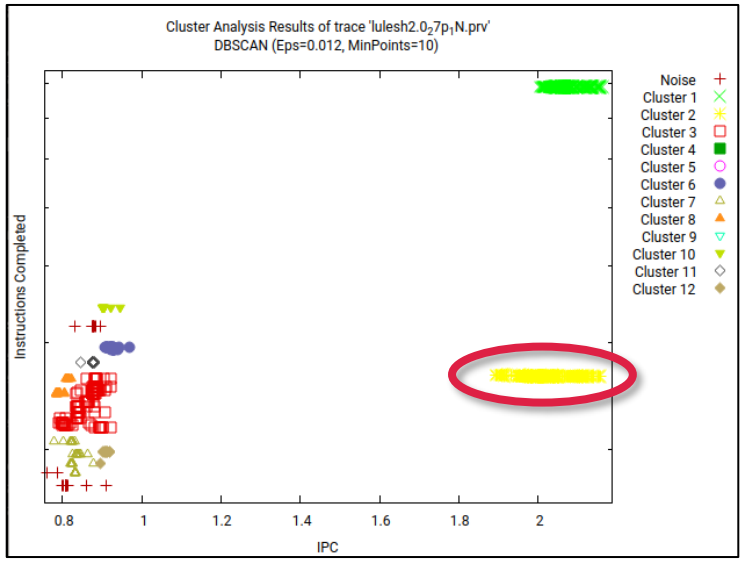
Correlating scatter plot and time distribution





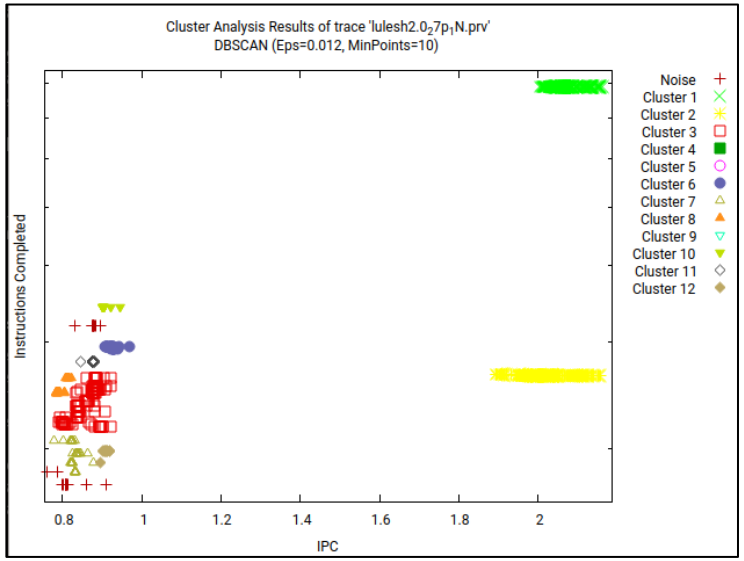
- Cluster 2 shows higher IPC variation than Cluster 1
  - 10% variation ranging from 1.89 to 2.12
- Can find this both ways, but clustering helped to quickly catch our eye and not overlook it

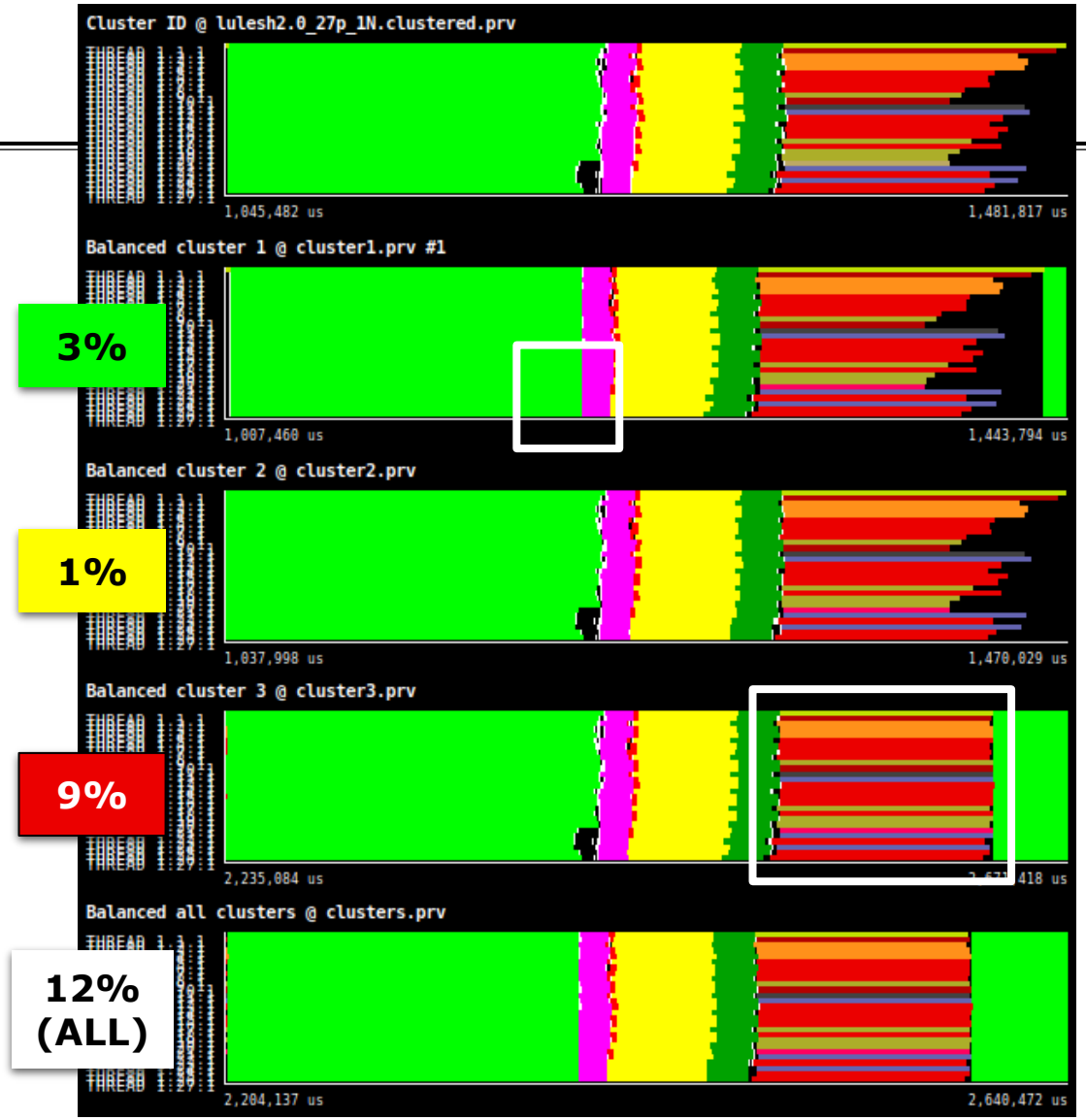




Should I care?

- Extrae + Clustering + Dimemas + Paraver
  - Simulate imbalanced clusters become balanced to their average duration
- Where do I put effort?
  - **Better fix the work imbalanced area!**



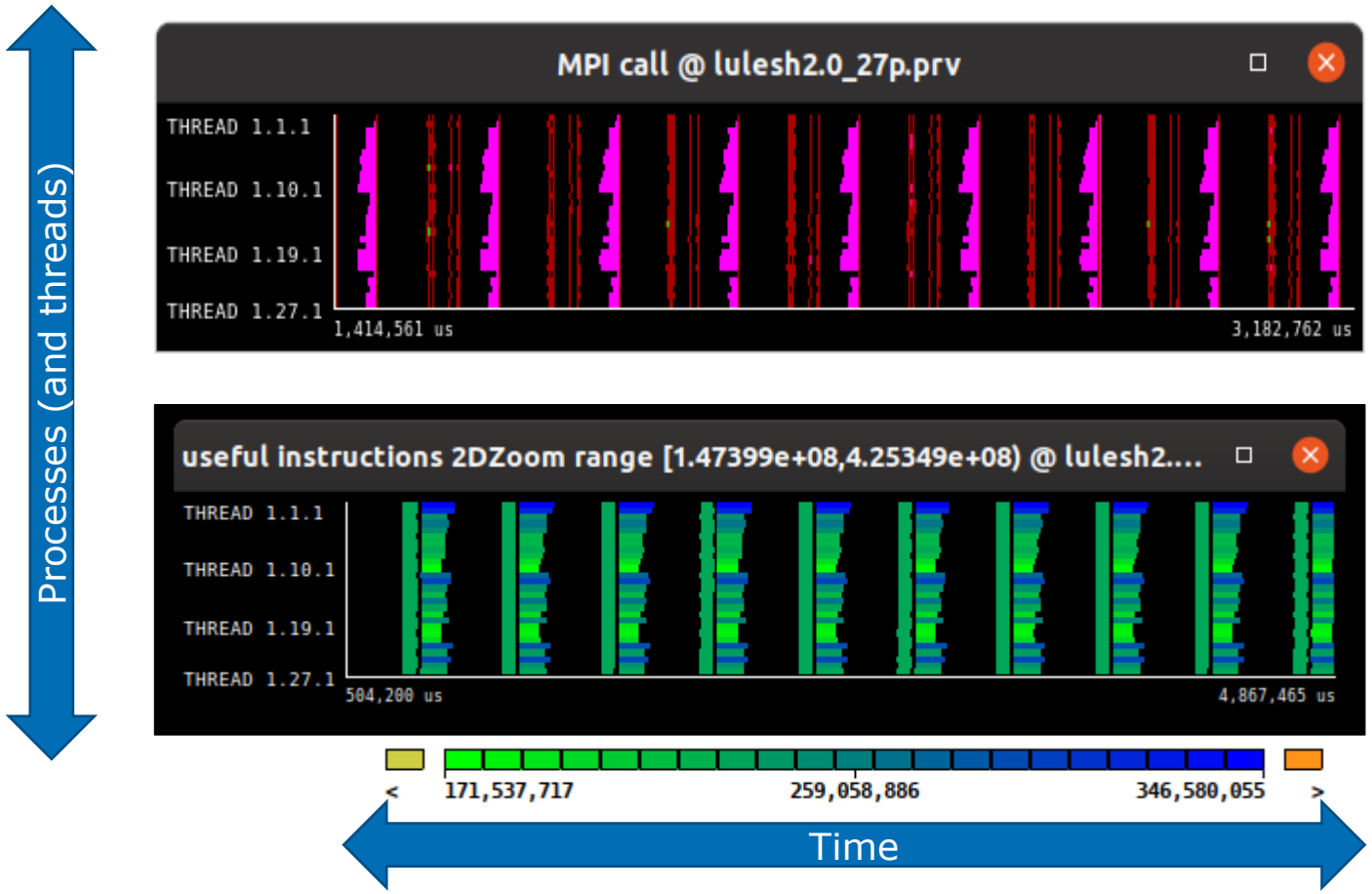


BSC Tools hands-on

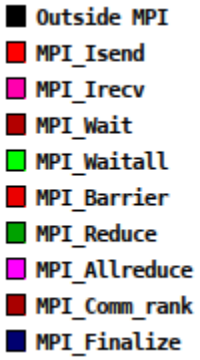
Germán Llort, Lau Mercadal
(tools@bsc.es)
Barcelona Supercomputing Center

Cheatsheet: 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)

Cheatsheet: 3 main views of Paraver (II)

- Table (Profile)

Categories (e.g. MPI calls)

Processes (and threads)

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

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

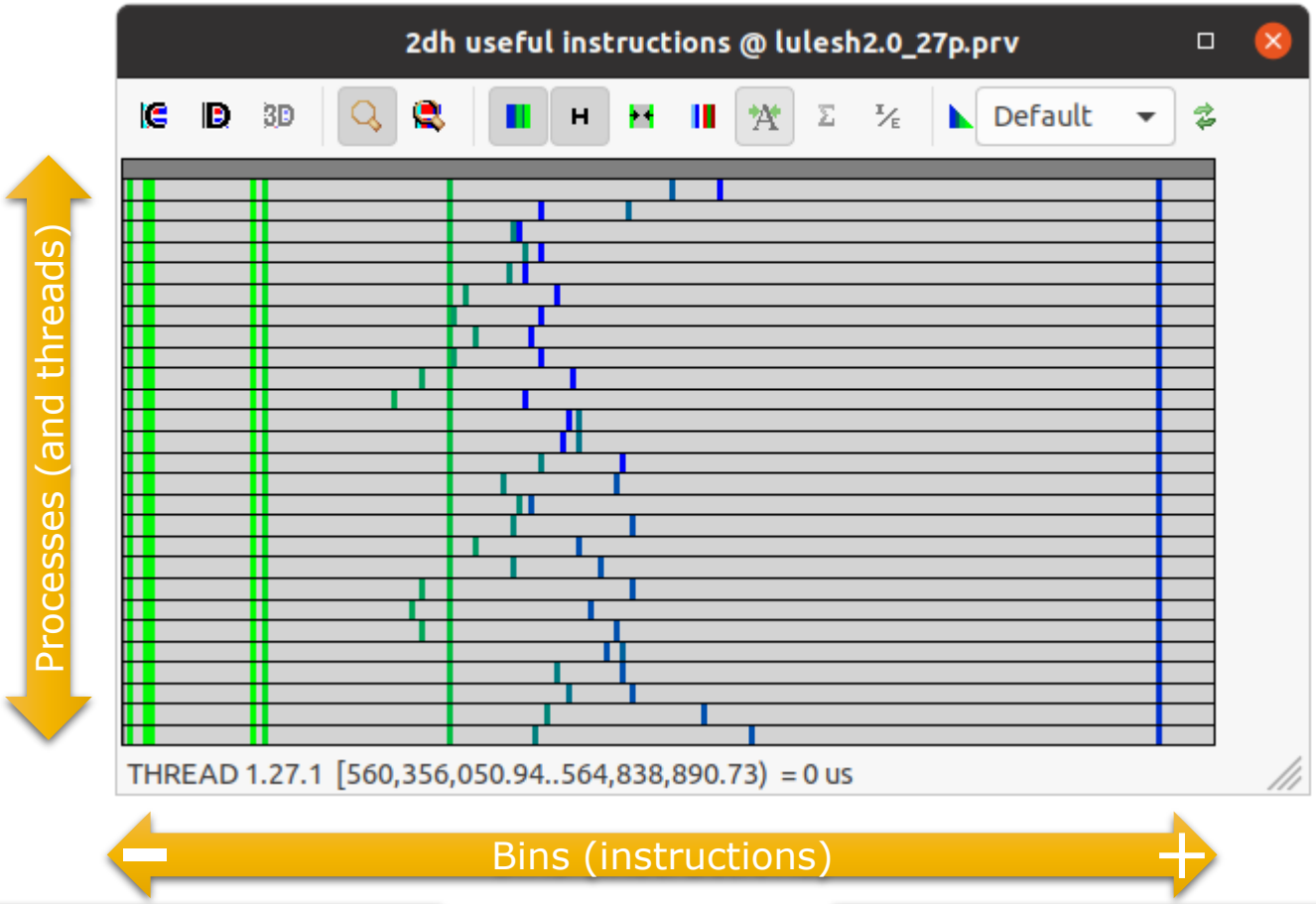
Cheatsheet: 3 main views of Paraver (III)

- Histogram

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

Gradient color represents low to high values of selected statistic (**time %**, # instances, etc.)

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



Computations with low count of instructions

Computations with high count of instructions