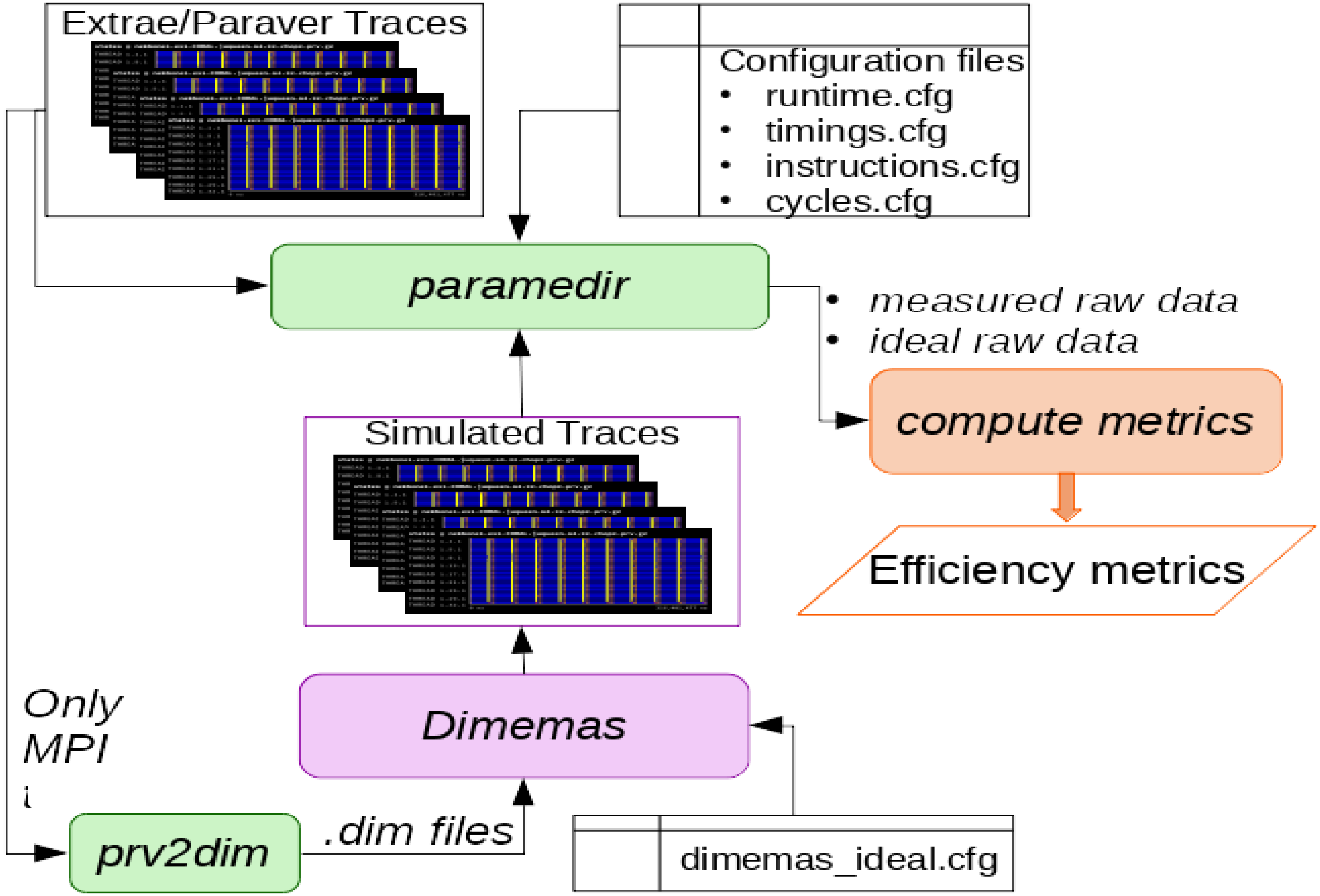
BasicAnalysis

Sandra Mendez
(tools@bsc.es)
Barcelona Supercomputing Center

BasicAnalysis Workflow



BasicAnalysis

There is no installation required. Just copy the content of package into your preferred location and add such directory to the PATH environment variable.

Requirements:
paramedir and **Dimemas** being installed and available through the PATH environment variable.

- **paramedir** available at https://tools.bsc.es/paraver
- **Dimemas** available at https://tools.bsc.es/dimemas

Add them to the PATH environment variable with:

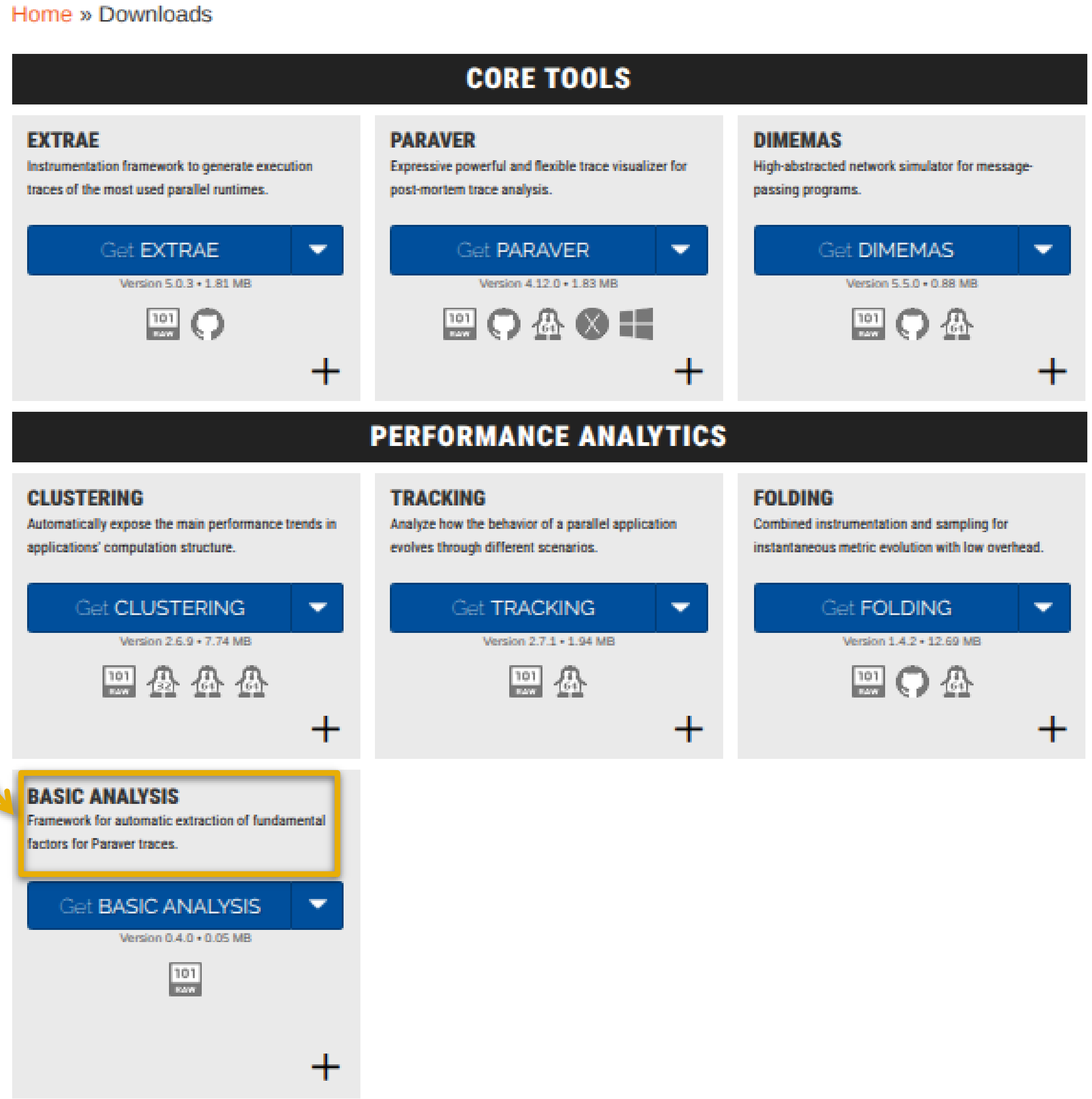
```
export PATH=<paraver-install-dir>/bin:$PATH
export PARAVER_HOME=<paraver-install-dir>
export PATH=<dimemas-install-dir>/bin:$PATH
export DIMEMAS_HOME=<dimemas-install-dir>
```

Usage:

```
modelfactors.py <list-of-traces>
```

Download BasicAnalysis in your laptop

- Download the package from
  - https://tools.bsc.es/downloads



BasicAnalysis on MareNostrum5 (I)

BasicAnalysis is available via modules…

```
@alogin2 ~]$ module av basicanalysis
```

```
@alogin2 ~]$ module load basicanalysis
```

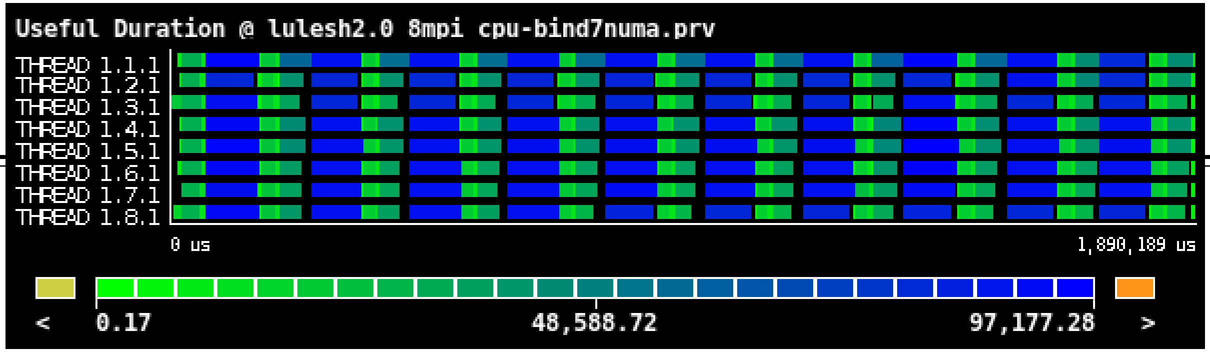
```
@alogin2 ~]$ modelfactors.py --help
```

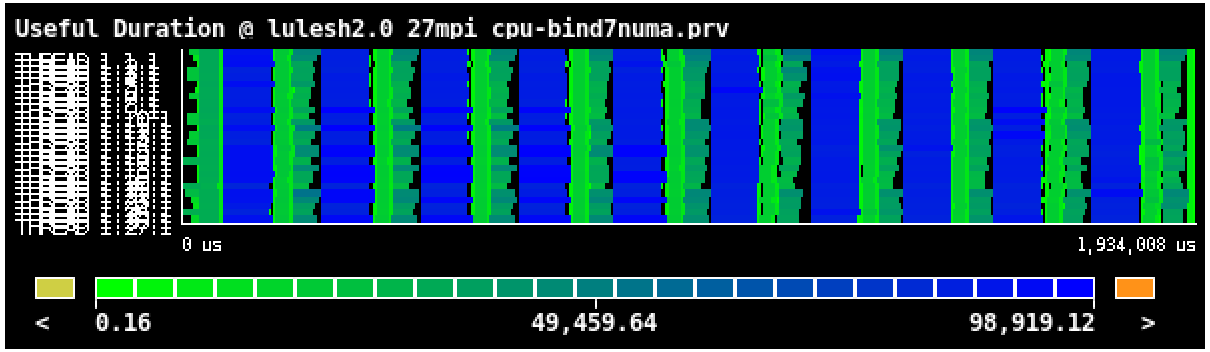
```
@alogin2 basicanalysis-output-mpi]$ modelfactors.py ../traces/lulesh/MPI/*numa.prv
```

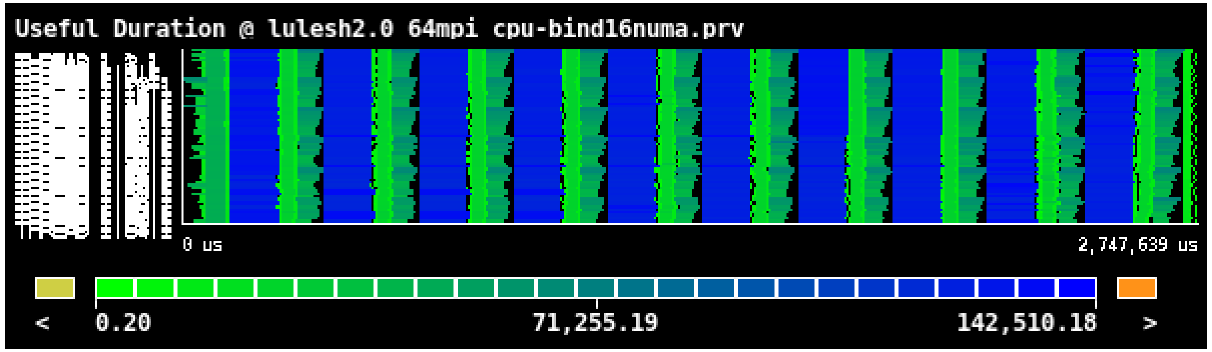
Lulesh Structure

- Lulesh only MPI version run in MN5.
- 10 iterations
- Weak Scaling test (8, 27 and 64 MPI processes)

Let us obtain the POP performance metrics for these traces using the BasicAnalysis tool.







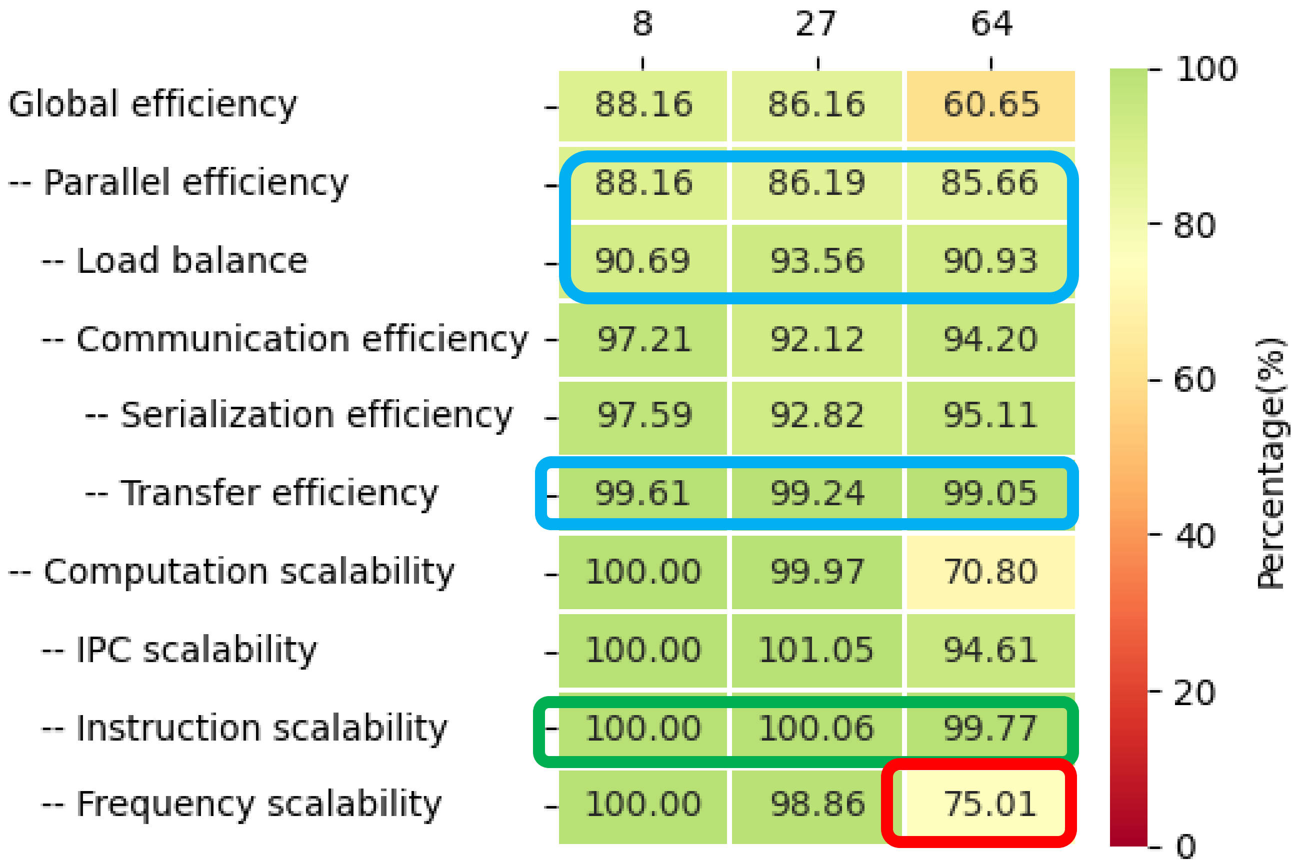
Lulesh MPI - BasicAnalysis Output – POP Metrics

- Output shown in a table
  - Rows: Metrics
  - Columns: Different traces
- With colored cells as a heat map
- What to look for?
  - Low values
  - Trend
  - High values

	8	27	64
Global efficiency	88.16	86.16	60.65
-- Parallel efficiency	88.16	86.19	85.66
-- Load balance	90.69	93.56	90.93
-- Communication efficiency	97.21	92.12	94.20
-- Serialization efficiency	97.59	92.82	95.11
-- Transfer efficiency	99.61	99.24	99.05
-- Computation scalability	100.00	99.97	70.80
-- IPC scalability	100.00	101.05	94.61
-- Instruction scalability	100.00	100.06	99.77
-- Frequency scalability	100.00	98.86	75.01

Percentage(%): 0, 20, 40, 60, 80, 100

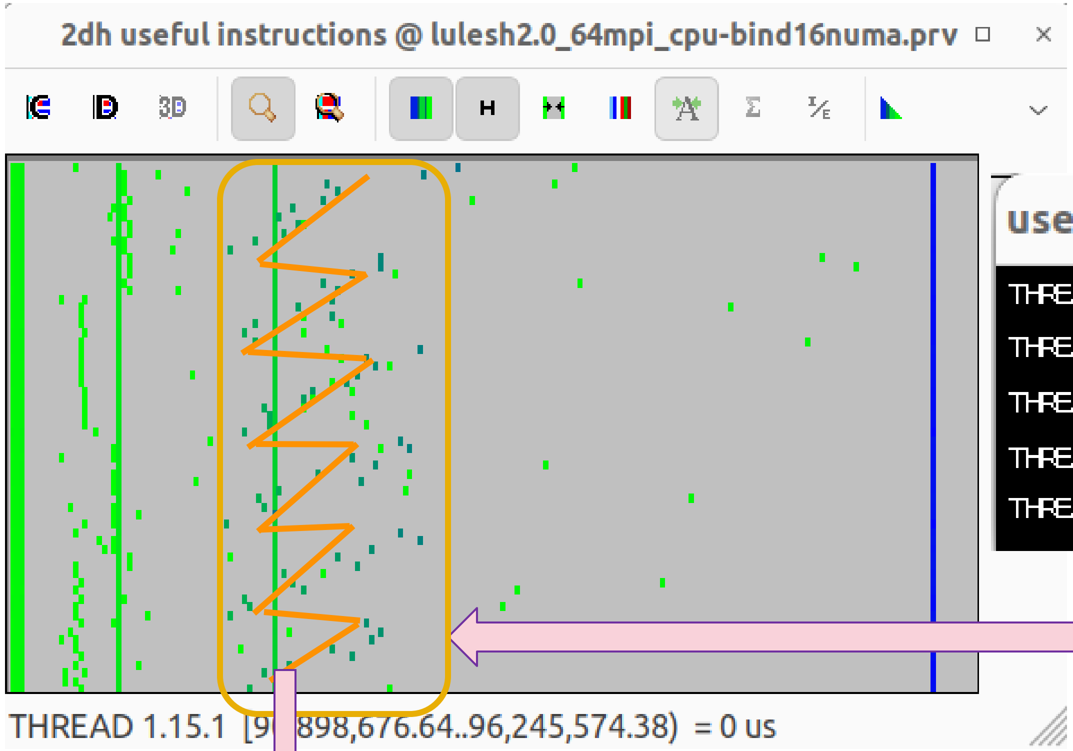
Lulesh MPI - BasicAnalysis Output – POP Metrics

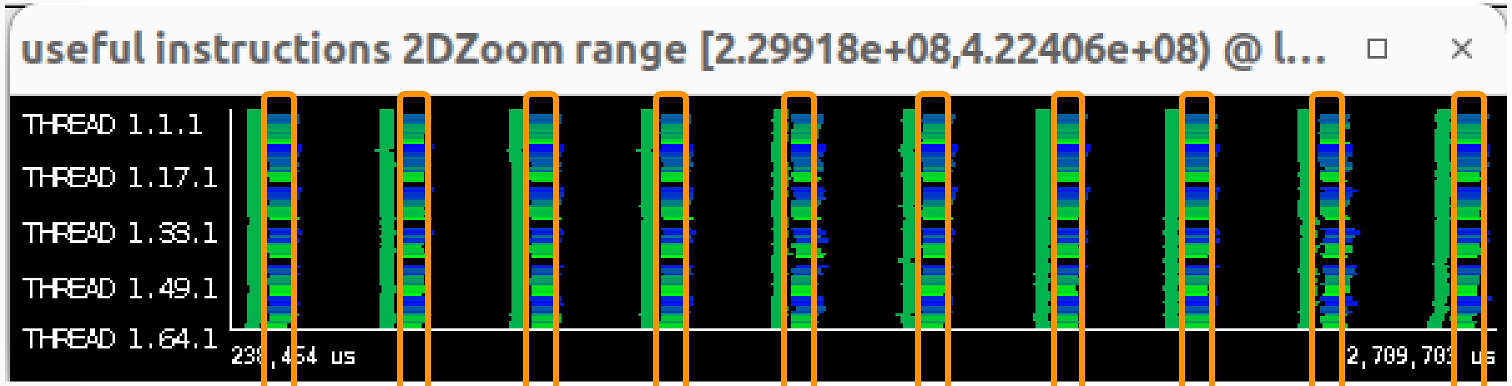


- **Parallel Efficiency** decreases mqinly due to **Load Balance (90% -> 90%)**.
- **Transfer** time remains nearly constant at around 99%.
- **Frequency** drops at 64 MPI processes because the sockets are more fully utilized (all NUMA domains active)

Load Balance Efficiency

	8	27	64
-- Load balance	90.69	93.56	90.93



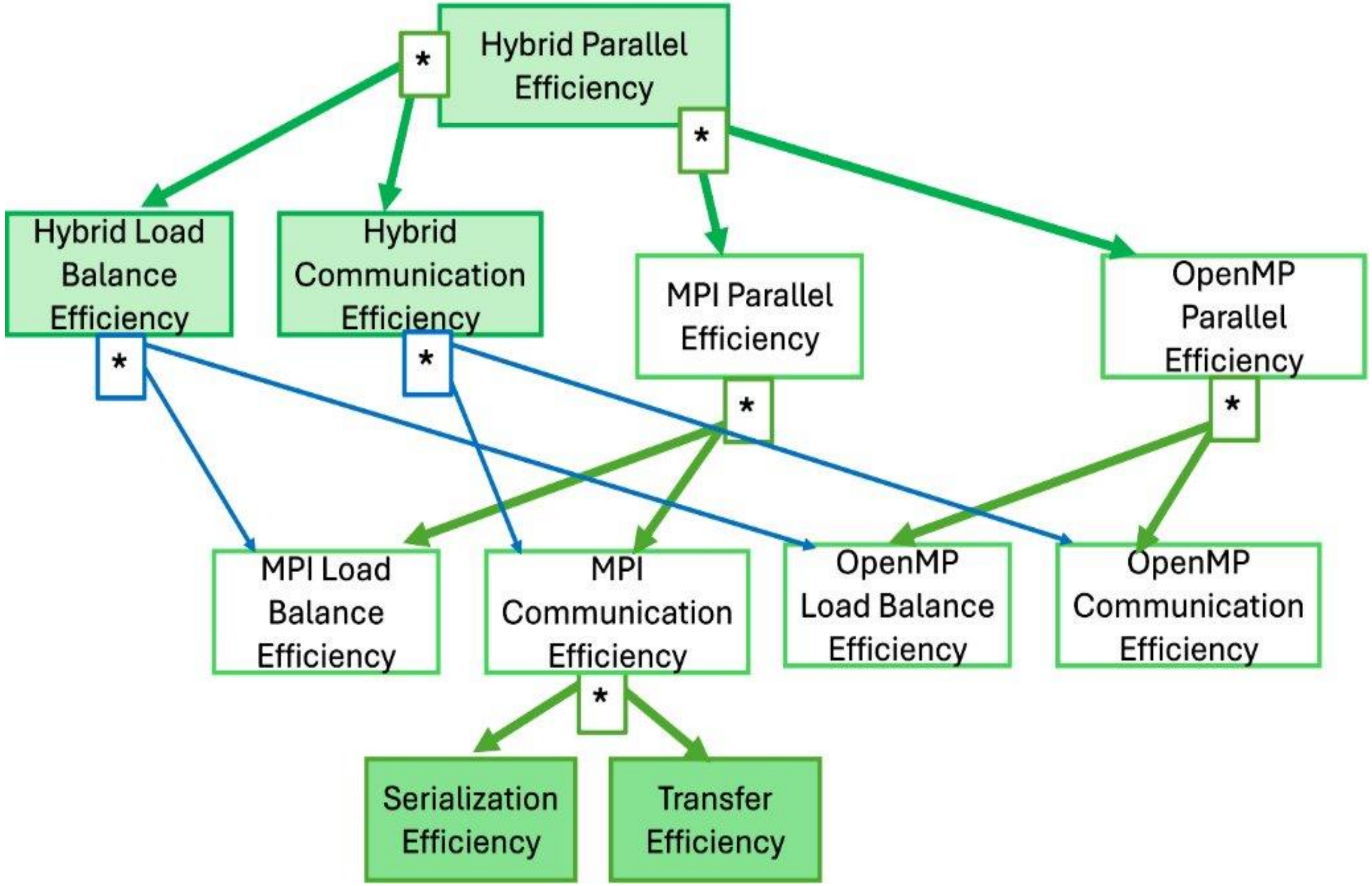


This is the part where the imbalance occurs.

Imbalance in instructions is reflected in the useful duration.

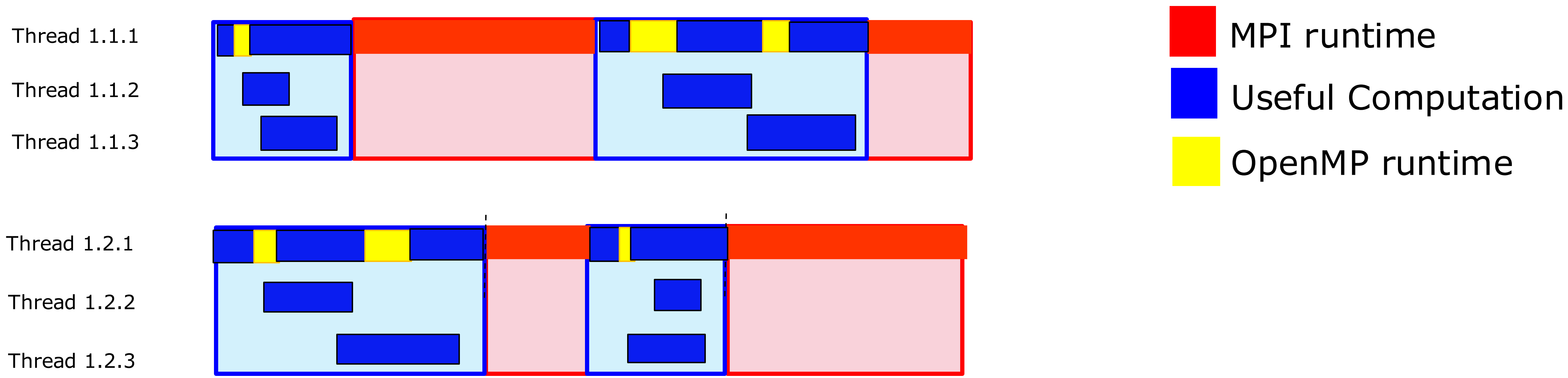
POP Efficiency hybrid MPI+OMP with BasicAnalysis

Hybrid Parallel Efficiency (MPI+OMP)



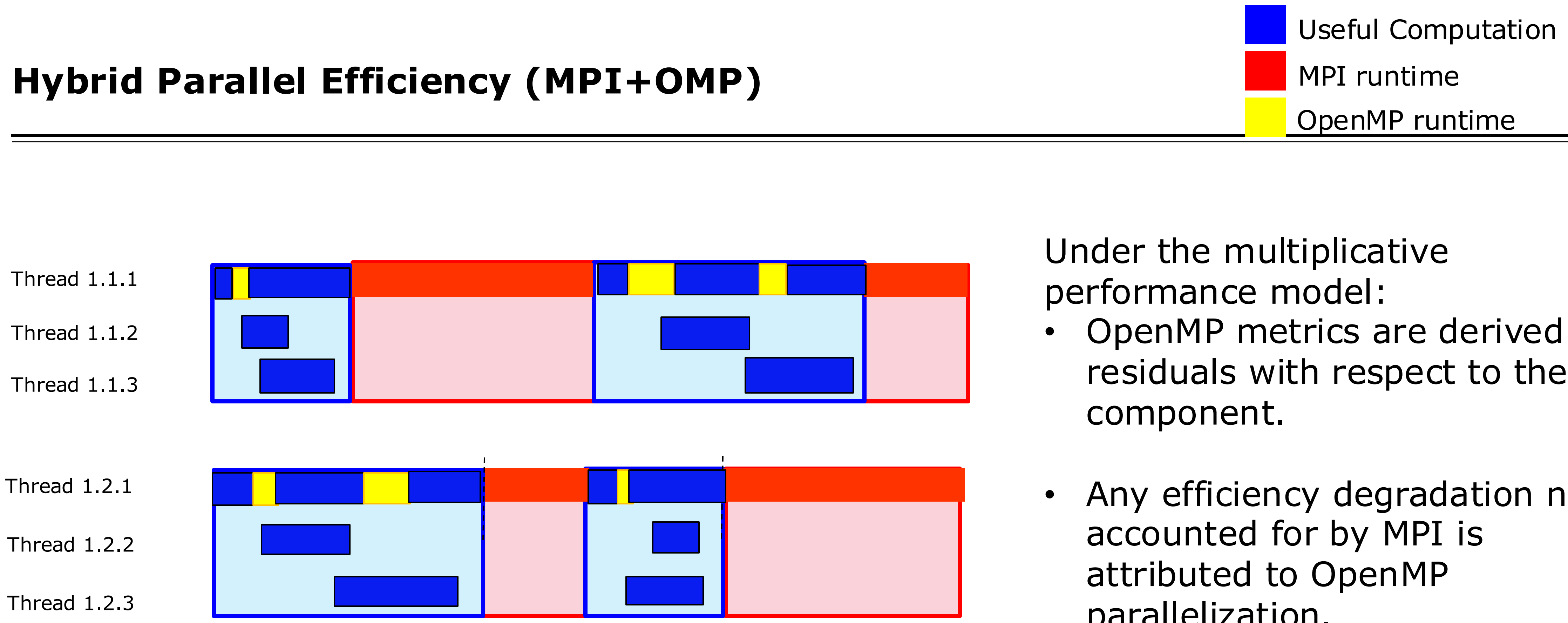
- Efficiency computed as percentage of time outside the two parallel runtimes
- Useful as a first step to distinguish between Load Balance and Communication. But how to dig down?
  - Efficiencies are mixing inefficiencies from MPI and OpenMP -> Need to distribute the blame between the two programming models.
- Global Load Balance and Communication concepts can be mapped to any parallel programming paradigm.
- The efficiencies at hybrid level collapse the contributions from the two programming models.

Hybrid Parallel Efficiency (MPI+OMP)



- Hierarchical codes where MPI is the upper level.
- From the MPI point of view, OpenMP runtime is useful (like computations)

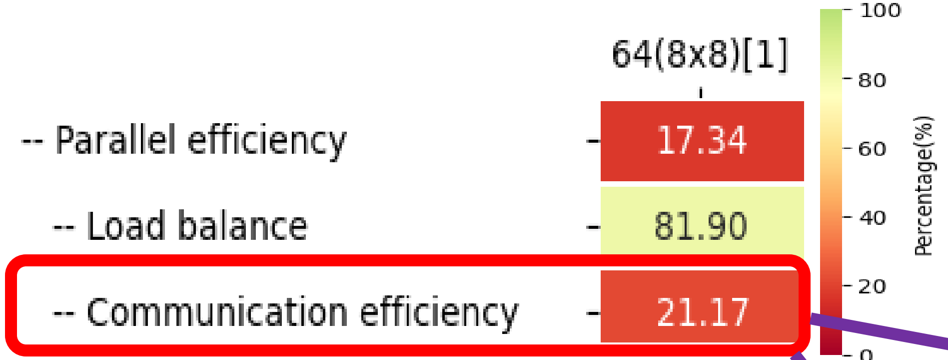
Hybrid Parallel Efficiency (MPI+OMP)

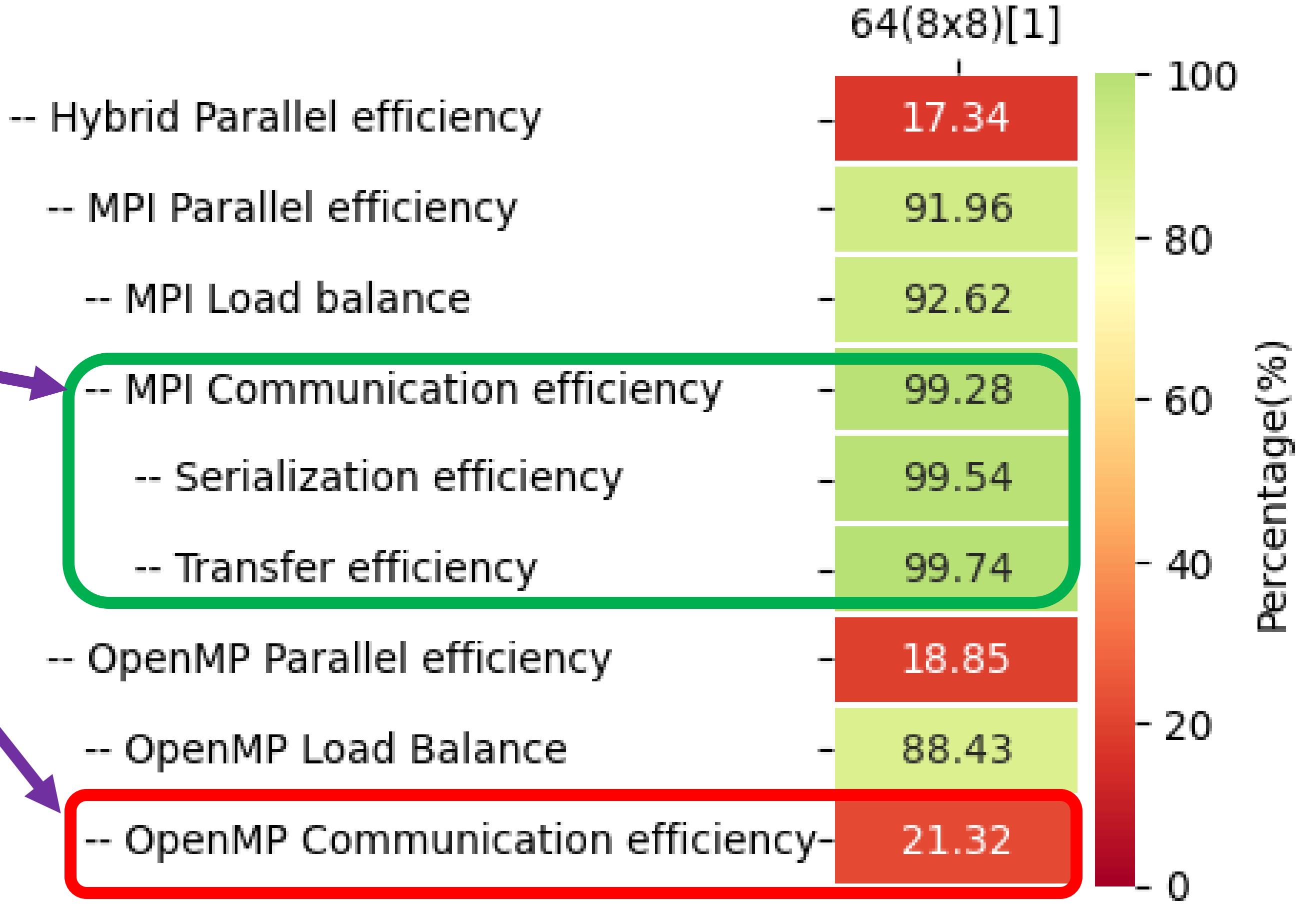


Under the multiplicative performance model:

- OpenMP metrics are derived as residuals with respect to the MPI component.
- Any efficiency degradation not accounted for by MPI is attributed to OpenMP parallelization.
- Inefficiencies measured in regions without MPI activity are assigned to the OpenMP component.

Lulesh MPI+OMP





- The model points to low communication efficiency. However, in OpenMP, what should be understood as "communication"?

In our model, it represents overhead introduced by the OpenMP runtime.

Lulesh MPI+OMP – OpenMP Communication Efficiency

Computing from MPI processes point of view

	Outside MPI
THREAD 1.1.1	2,600,712.23 us
THREAD 1.2.1	2,751,730.45 us
THREAD 1.3.1	2,495,340.64 us
THREAD 1.4.1	2,637,581.04 us
THREAD 1.5.1	2,465,857.52 us
THREAD 1.6.1	2,580,549.29 us
THREAD 1.7.1	2,411,772.92 us
THREAD 1.8.1	2,446,410.92 us

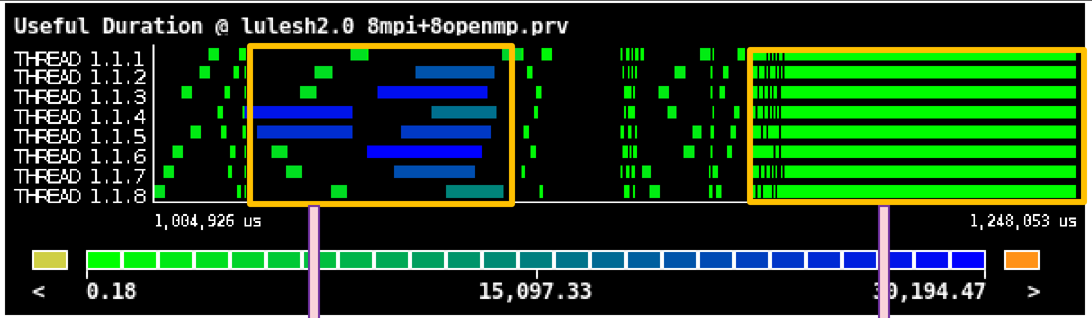
Idle	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O
2,523.75 us	470,508.62 us	-	54,075.87 us	2,053,070.36 us	20,533.63 us
-	359,481.58 us	122,090.91 us	54,791.64 us	2,192,341.28 us	23,025.04 us
4,877.06 us	425,727.97 us	39,855.37 us	49,071.82 us	1,957,629.78 us	18,178.64 us
1,362.87 us	357,568.55 us	121,347.91 us	49,943.60 us	2,085,655.68 us	21,702.42 us
3,007.68 us	424,334.04 us	41,282.63 us	47,396.14 us	1,929,802.53 us	20,034.49 us
2,762.57 us	362,409.15 us	122,584.86 us	47,258.43 us	2,025,219.46 us	20,314.82 us
3,231.19 us	420,968.22 us	41,738.29 us	44,420.05 us	1,881,557.68 us	19,857.47 us
1,720.02 us	404,525.45 us	63,369.13 us	47,569.33 us	1,907,930.07 us	21,296.92 us

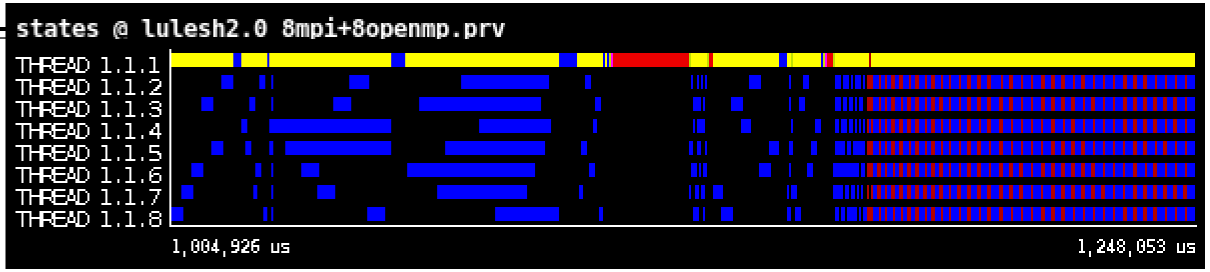
How much of the Outside MPI time is spent in Scheduling and Fork/Join operations?

78.94%
79.67%
78.45%
79.07%
78.26%
78.48%
78.02%
77.99%

Master Threads: Most of the time is spent in OpenMP scheduling and fork/join operations.

Lulesh MPI+OMP – OpenMP Communication Efficiency (1 iteration)





Average Burst Time

	Idle	Running	Scheduling and Fork/Join
THREAD 1.1.1	-	856.37 us	12,794.34 us
THREAD 1.1.2	10,979.81 us	8,111.44 us	-
THREAD 1.1.3	11,982.02 us	16,153.75 us	-
THREAD 1.1.4	7,847.03 us	22,356.23 us	-
THREAD 1.1.5	6,686.09 us	24,097.64 us	
THREAD 1.1.6	11,471.24 us	16,919.92 us	
THREAD 1.1.7	14,497.83 us	12,380.03 us	
THREAD 1.1.8	12,397.05 us	6,221.79 us	

	Idle	Running	Synchronization	Scheduling and Fork/Join
THREAD 1.1.1	-	9.94 %	2.14 %	79.25 %
THREAD 1.1.2	80.81 %	15.18 %	4.01 %	-
THREAD 1.1.3	77.81 %	18.20 %	3.99 %	-
THREAD 1.1.4	74.20 %	22.25 %	3.55 %	-
THREAD 1.1.5	71.68 %	24.57 %	3.75 %	-
THREAD 1.1.6	77.49 %	18.82 %	3.69 %	-
THREAD 1.1.7	79.49 %	15.26 %	5.25 %	-
THREAD 1.1.8	83.74 %	12.86 %	3.40 %	-

	Idle	Running	Synchronization	Scheduling and Fork/Join
THREAD 1.1.1	-	5.73 us	121.87 us	80.91 us
THREAD 1.1.2	153.18 us	10.32 us	237.89 us	-
THREAD 1.1.3	153.83 us	10.04 us	233.59 us	-
THREAD 1.1.4	156.41 us	10.00 us	202.35 us	-
THREAD 1.1.5	154.11 us	10.22 us	227.84 us	-
THREAD 1.1.6	155.65 us	9.95 us	212.37 us	-
THREAD 1.1.7	147.39 us	10.18 us	310.82 us	-
THREAD 1.1.8	156.70 us	10.10 us	197.48 us	-

Master thread 1: Time is mainly spent in OpenMP scheduling and fork/join.
Worker threads: Most of the execution time is idle.

Very fine-grained region (avg burst time <11µs)

Getting a MPI+CUDA trace with Extrae

Step 1: Adapt the scripts to instrument MPI + CUDA binaries

- Changes in jobscript, launcher and configuration file (`extrae/MPI+CUDA/{trace.sh,extrae.xml}`)

jobscript

```
!/usr/bin/env bash

#SBATCH --job-name=cloverleaf_cuda
#SBATCH --output=%x.%j.out
#SBATCH --error=%x.%j.err
#SBATCH --ntasks=4
#SBATCH --cpus-per-task=20
#SBATCH --gres=gpu:4
#SBATCH --time=00:10:00
#SBATCH --qos=acc_debug
#SBATCH --exclusive
#SBATCH --constraint-perfparanoid

ml gcc/11.4.0
ml openmpi/4.1.5-gcc
ml cuda/12/2

export TRACE_NAME=cloverleaf.prv

srun ./trace.sh ./cloverleaf
```

4 tasks, one for each CUDA device and 20 cores per task

trace.sh

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

module load extrae

export EXTRAE_CONFIG_FILE=./extrae.xml
# For MPI+CUDA apps
- export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitrace.so
+ export LD_PRELOAD=${EXTRAE_HOME}/lib/libcudampitrace.so

## Run the desired program
$* --device $PMI_RANK
```

Tracing library for MPI + CUDA

Adds multi-device support

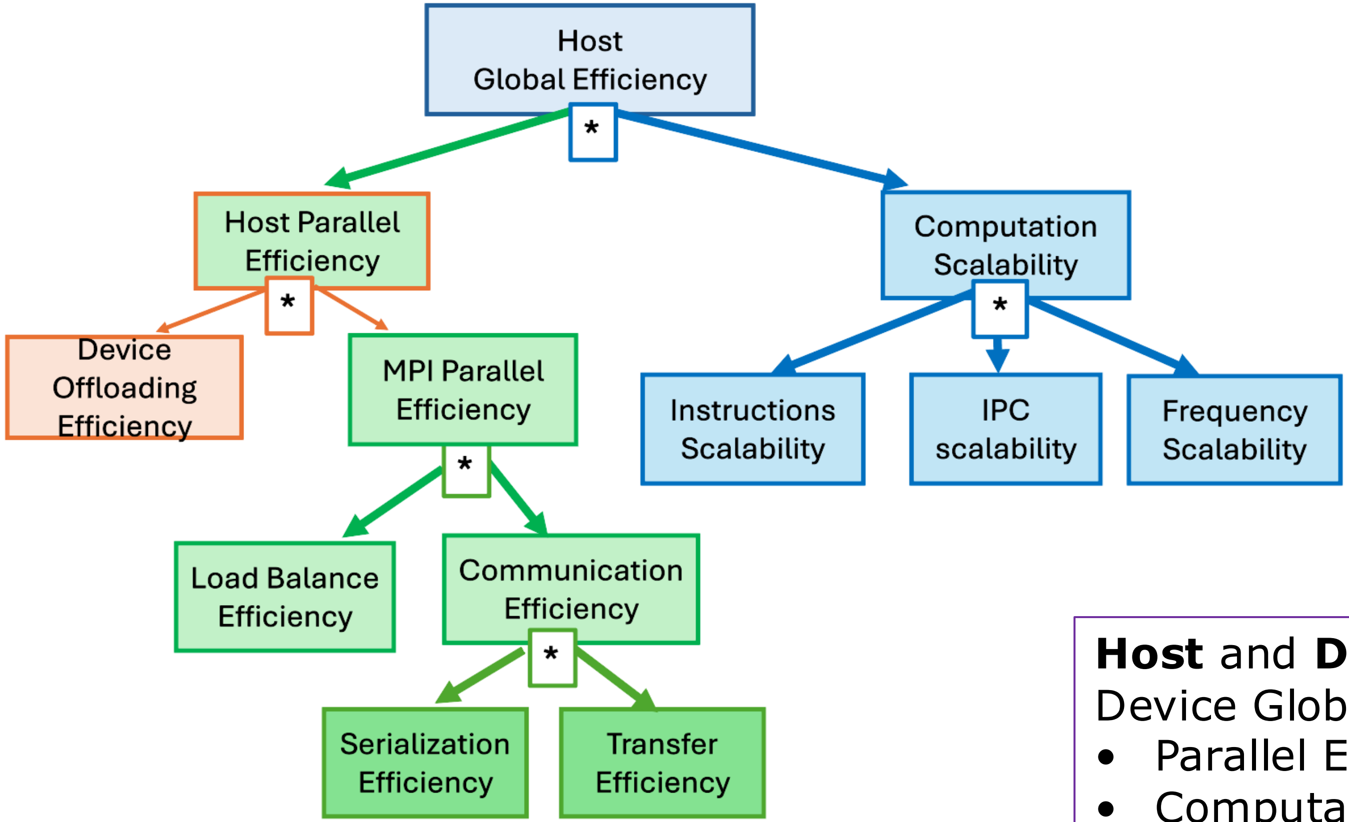
extrae.xml

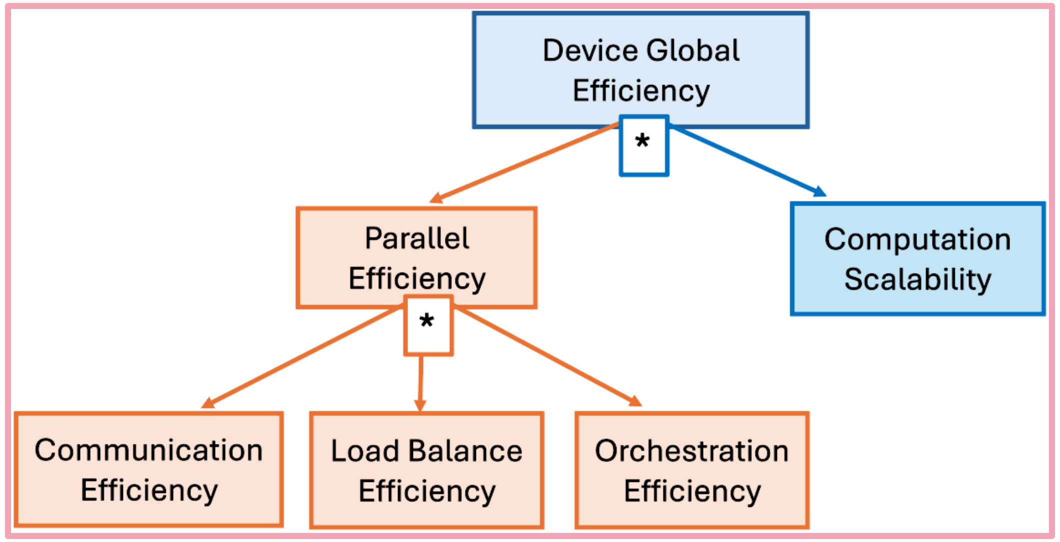
```
- <cuda enabled="no" />
+ <cuda enabled="yes" />
```

Enables CUDA instrumentation

Hybrid Efficiency Metrics (MPI+CUDA) with BasicAnalysis

Metrics for MPI+CUDA (GPU)





Host and **Device** Global Efficiency (not multiplicative)
Device Global Efficiency divided in:

- Parallel Efficiency
- Computation Scalability (Only reference computation time)

GPU as a single resource
All arrows are multiplicative

Metrics for MPI+CUDA (GPU)

Host

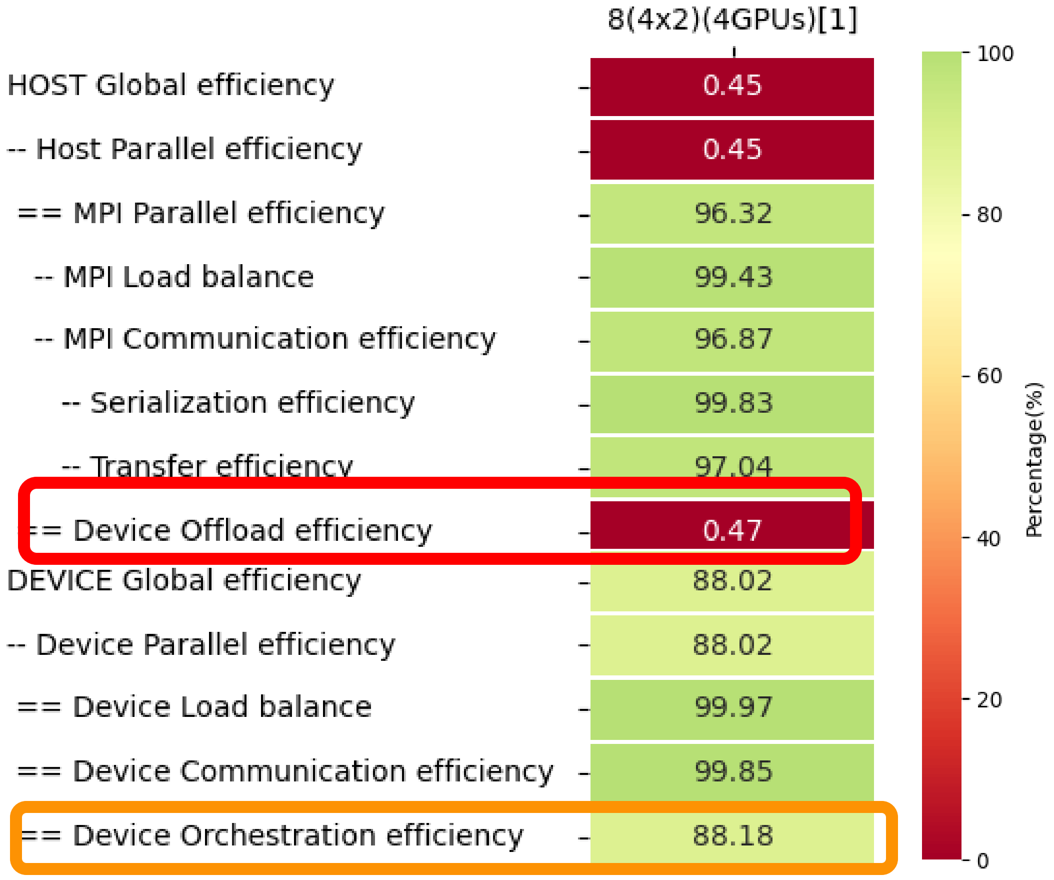
- **Device Offload Efficiency**: Inefficiency on the host side due to the use of an accelerator.
  - Includes: offloading work to the device, waiting for kernel execution, and waiting for data transfers to or from the accelerator.
  - expected to be low for applications that primarily execute on the device, reflecting limited host-side activity.

Device

- **Orchestration Efficiency**: Inefficiency due to a lack of available work on the accelerator.
  - Includes: idle time caused by waiting for work from the host, kernel launch dependencies, and insufficient concurrency.
  - A low value indicates that the accelerator is underutilized due to work starvation.
- **Communication Efficiency**: Inefficiency due to non-instantaneous data movement involving the accelerator.
  - Includes: waiting for data transfers from or to the host, communication between accelerators, NCCL communication, and CUDA-aware MPI communication.
- **Load Balance Efficiency**: Inefficiency caused by unequal computation time across accelerators.
  - This metric captures load imbalance at the device level and does not distinguish between accelerators belonging to the same process or different processes.
- **Computation Scalability**: How well the resources inside the accelerator are being used.

TBD: streams, warps, occupancy, instructions, tensor core use.

Metrics for MPI+CUDA (GPU) - Cloverleaf



- **Device Offload Efficiency**: Inefficiency due to use of accelerator.

$$Device\_Offload\_Eff = \frac{\sum_{i=1}^{P} Useful_i}{\sum_{i=1}^{P} OutsideMPI_i}$$

- **Orchestration Efficiency**: Inefficiency due to lack of available work to do.

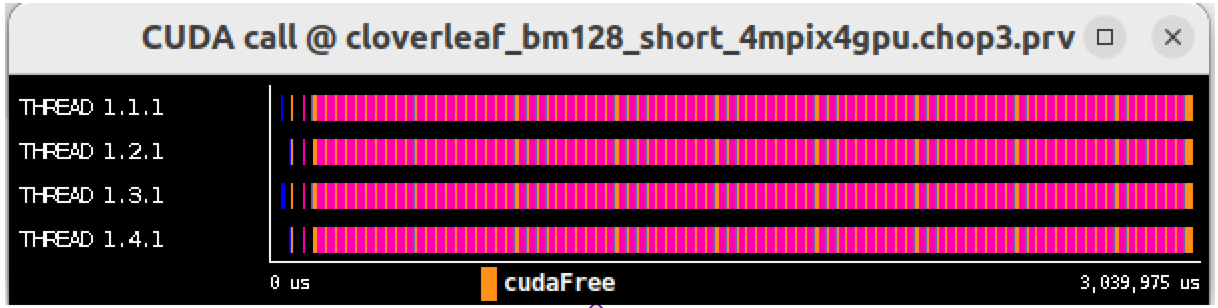
$$Device\_Orch\_Eff = \frac{\max_{d \in [1,D]} (Useful_d + MemTransfer_d)}{T}$$

Metrics for MPI+CUDA (GPU) - Cloverleaf Device Offload Efficiency

$$Device\_Offload\_Eff = \frac{\sum_{i=1}^{P} Useful_i}{\sum_{i=1}^{P} OutsideMPI_i}$$

Useful Computation

THREAD 1.1.1	17,064.79 us
THREAD 1.2.1	11,003.73 us
THREAD 1.3.1	12,116.84 us
THREAD 1.4.1	14,508.98 us
Num. Cells	4
Total	54,694.34 us



Outside MPI

THREAD 1.1.1	2,910,449.90 us
THREAD 1.2.1	2,944,962.29 us
THREAD 1.3.1	2,919,970.86 us
THREAD 1.4.1	2,937,136.20 us
Num. Cells	4
Total	11,712,519.25 us

Most of the host-side execution time is spent in calls to the accelerator. **97% of the Outside MPI time is spent in CUDA calls**.

Total time spent in cuda calls = 11,383,812.9 us

	cudaLaunch	cudaMemcpy	cudaDeviceSynchronize	cudaMalloc	cudaFree
THREAD 1.1.1	166,167.12 us	199,390.98 us	2,452,784.47 us	2,746.22 us	19,053.48 us
THREAD 1.2.1	177,340.30 us	199,337.08 us	2,451,128.06 us	2,571.65 us	21,357.35 us
THREAD 1.3.1	177,225.38 us	199,486.83 us	2,451,018.35 us	2,639.29 us	20,563.01 us
THREAD 1.4.1	165,230.73 us	199,391.78 us	2,455,493.14 us	2,507.24 us	18,380.43 us
Num. Cells	4	4	4	4	4
Total	685,963.53 us	797,606.68 us	9,810,424.02 us	10,464.41 us	79,354.27 us

Metrcis for MPI+CUDA (GPU) - Cloverleaf Device Orchestration Efficiency

$$Device\_Orch\_Eff = \frac{\max_{d \in [1,D]} (Useful_d + MemTransfer_d)}{T}$$

		Running	Memory transfer	Idle	Not created
	CUDA-D1.S1-as06r4b08	2,676,978.09 us	2,214.66 us	327,693.61 us	33,088.47 us
	CUDA-D2.S1-as06r4b08	2,677,739.51 us	2,265.86 us	297,908.05 us	62,061.42 us
max	CUDA-D3.S1-as06r4b08	2,678,376.92 us	2,285.24 us	322,572.98 us	36,739.69 us
	CUDA-D4.S1-as06r4b08	2,677,607.54 us	2,175.33 us	298,739.42 us	61,452.55 us

Where is the remaining execution time spent?

Orchestration efficiency is mainly affected by idle time with a 10.61%, with a smaller contribution from the not-created state (before stream is registered) with a 1.21%.

BasicAnalysis

Sandra Mendez
(tools@bsc.es)
Barcelona Supercomputing Center