17 September 2025

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

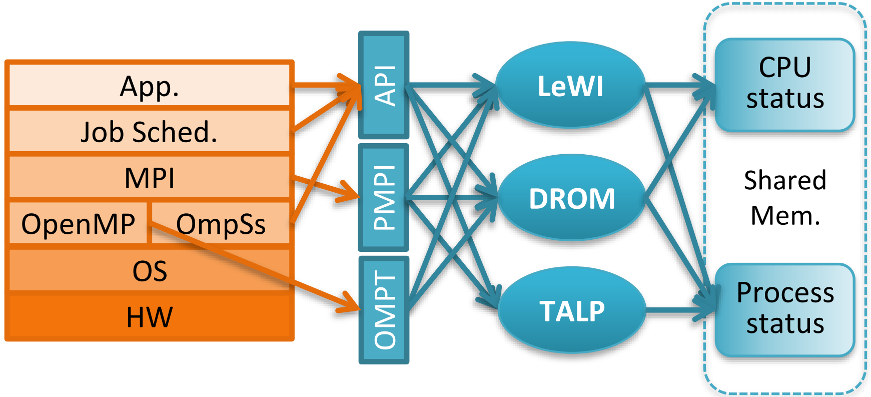
TALP
Tracking Application Live Performance

Marta Garcia-Gasulla

DLB library structure

- Dynamic Library
- Three modules:
  - LeWI: For fine grain load balancing
  - DROM: For coarse grain resource management
  - TALP: For performance measurement
- Common infrastructure
  - Integration with different layers of software stack
  - API
  - Shared memory

Three modules, integrated but independent

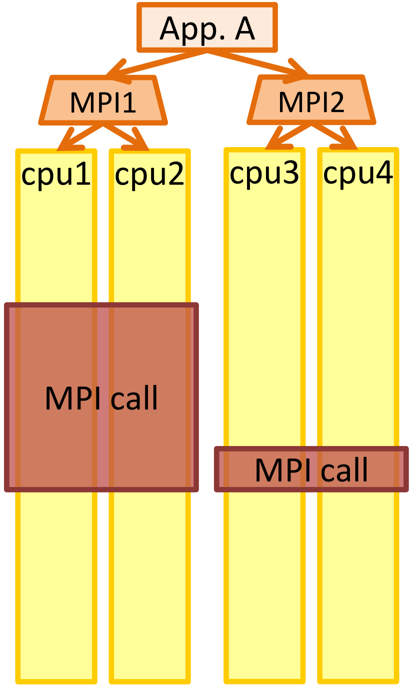


TALP: Tracking Application Live Performance

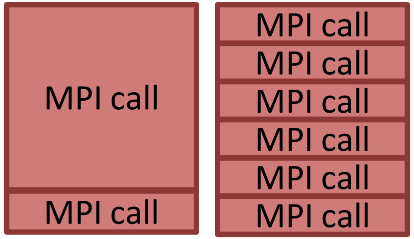
- Profiling tool with:
  - Low overhead
  - Report POP metrics
  - API to obtain metrics at runtime
  - API to instrument code and profile regions of code
- Version 3.6.0-beta1
  - MPI efficiency metrics
  - Hardware counters (cycles, instructions, IPC, and frequency)
  - OpenMP metrics
  - TALP-pages (CI/CD integration)
  - GPU efficiency metrics (NVIDIA devices)
- Work in progress
  - GPU efficiency metrics (AMD devices)
  - GPU computational metrics
  - TALP-pages support for GPU metrics

TALP a lightweight tool to Unveil Parallel Efficiency of Large Scale Executions. *In Proceedings of Performance Engineering, Modelling, Analysis, and Visualization Strategy (Permavost 2021).*

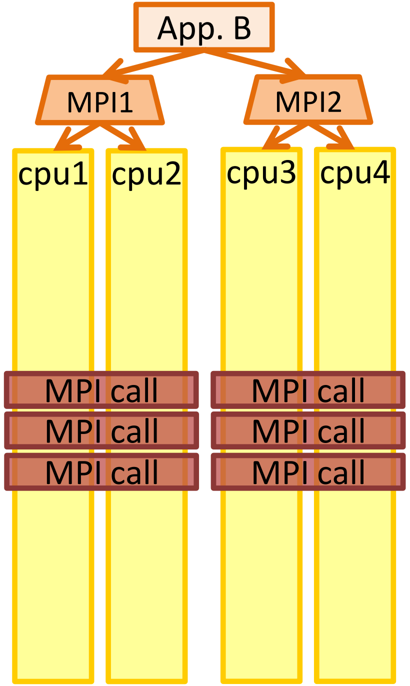
Why is more than “yet another profiling tool”?



- ➢ A profiler will report same “issue” while both cases have very different problems.



- ➢ TALP will report a low Load Balance for App A and a low Communication efficiency for App B

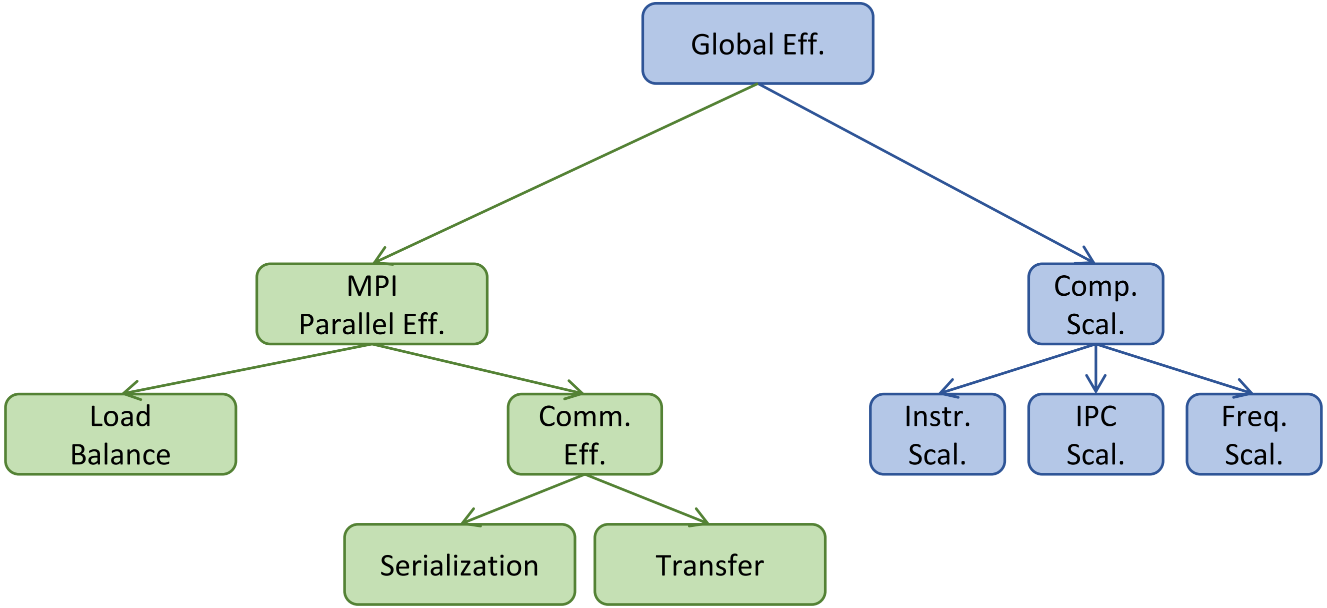


TALP Metrics

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MPI POP metrics

All parent metrics are the product of their child metrics

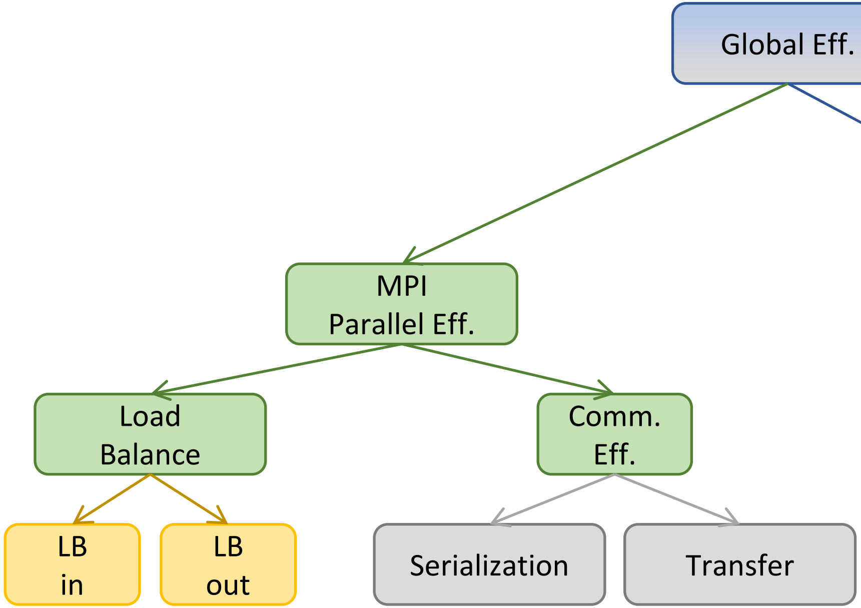


🟦 Scalability metrics computed based on a reference run

🟩 Efficiency metrics

TALP MPI metrics

All parent metrics are the product of their child metrics



Metrics not computed by TALP

Scalability metrics computed with several TALP runs

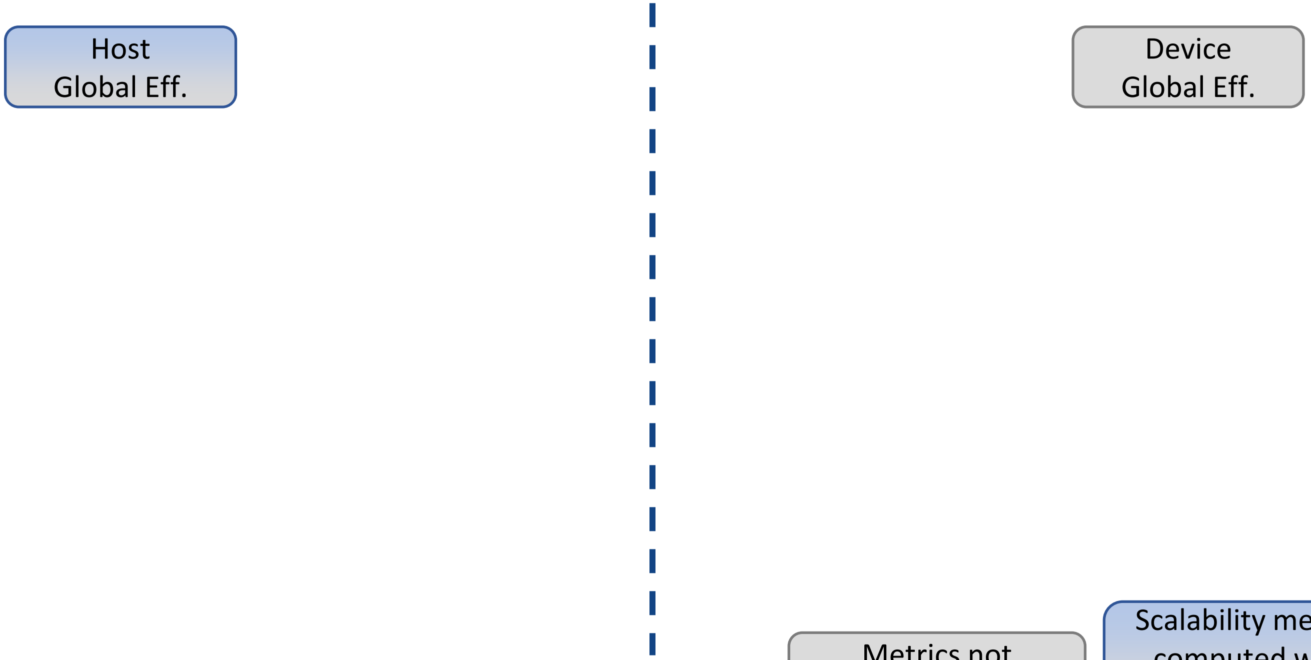
Additional metrics

Efficiency metrics

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

TALP GPU metrics

All parent metrics are the product of their child metrics


Host Global Eff.

Device Global Eff.

Metrics not computed by TALP

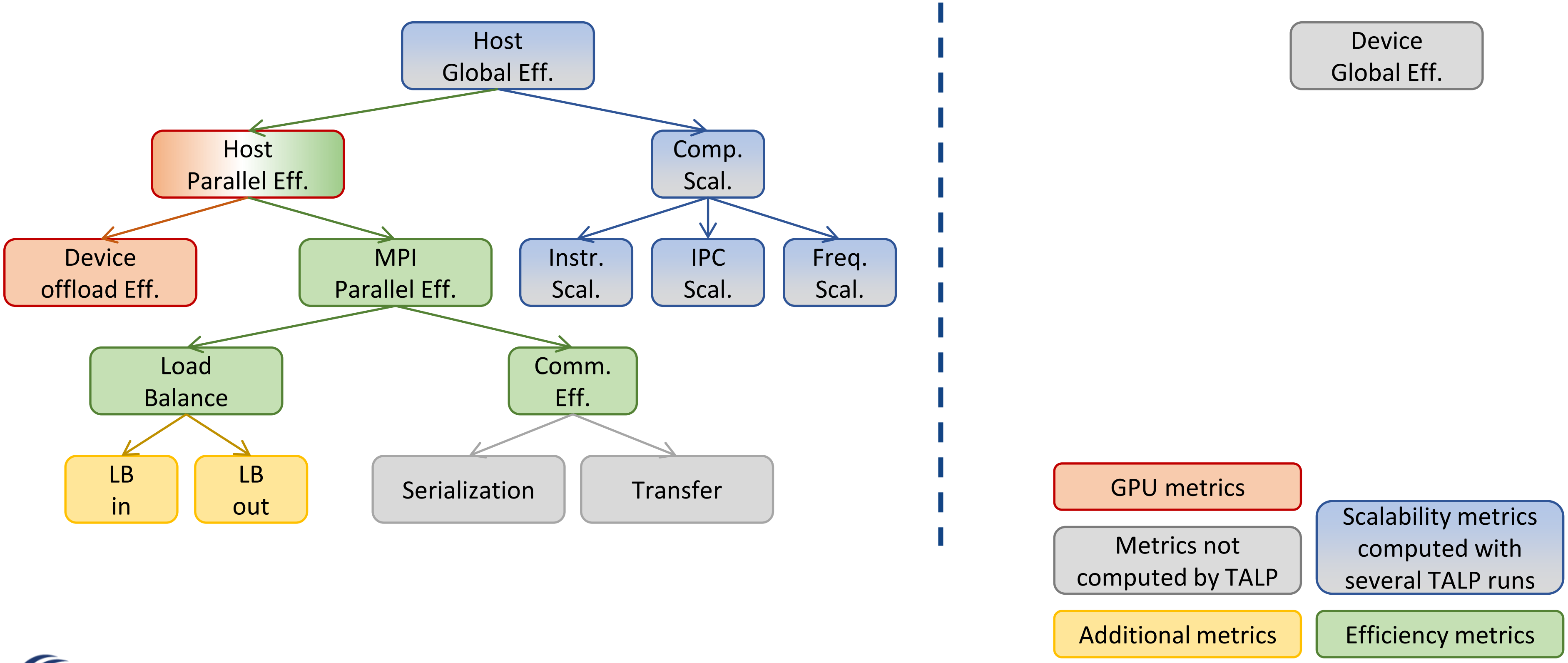
Scalability metrics computed with several TALP runs

Additional metrics

Efficiency metrics

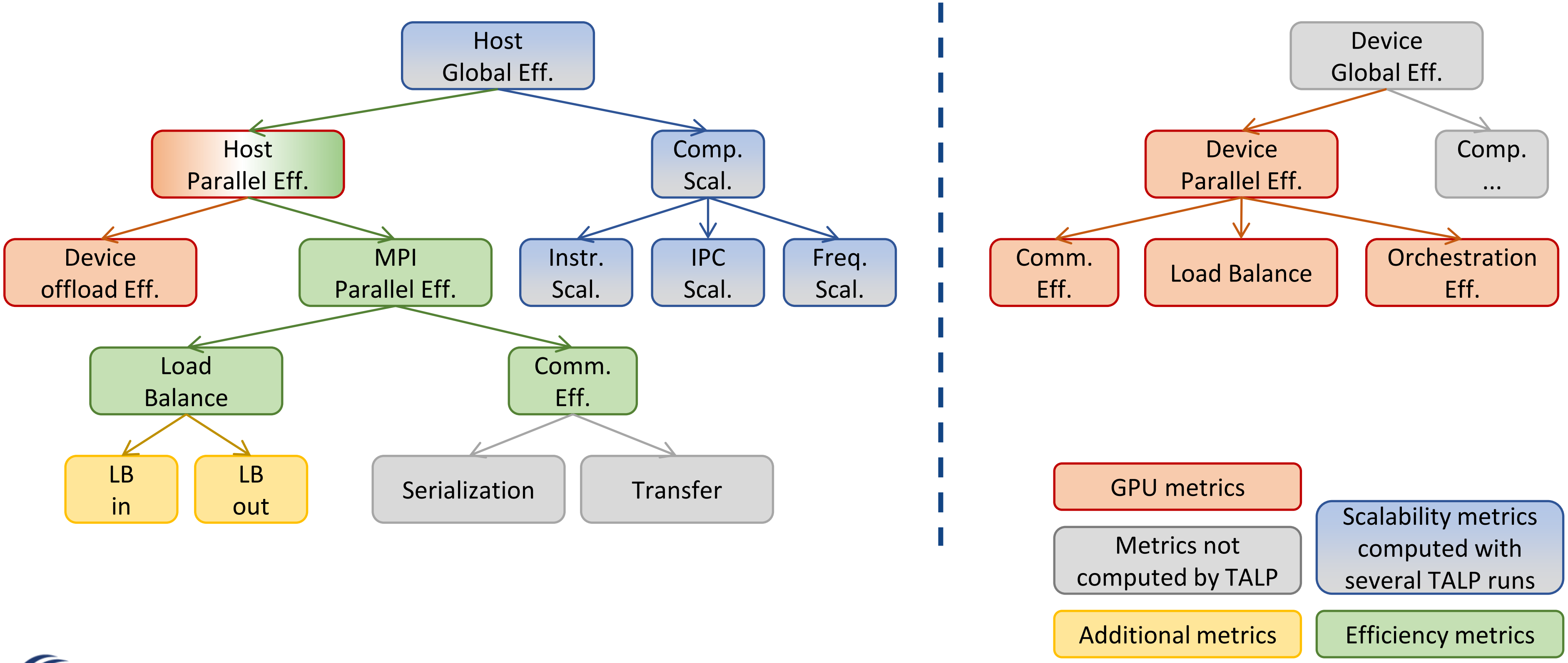
TALP GPU metrics

All parent metrics are the product of their child metrics



TALP GPU metrics

All parent metrics are the product of their child metrics



Host Global Eff.

Host Parallel Eff.

Comp. Scal.

Device offload Eff.

MPI Parallel Eff.

Instr. Scal.

IPC Scal.

Freq. Scal.

Load Balance

Comm. Eff.

LB in

LB out

Serialization

Transfer

Device Global Eff.

Device Parallel Eff.

Comp. ...

Comm. Eff.

Load Balance

Orchestration Eff.

🟥 GPU metrics

⬜ Metrics not computed by TALP

🟦 Scalability metrics computed with several TALP runs

🟨 Additional metrics

🟩 Efficiency metrics

TALP GPU metrics

- Host and Device Efficiencies are “unrelated”
  - We can measure them separately
- Device Global Efficiency divided in
  - Device Parallel Efficiency
  - Computation (Sc./Eff.) → WiP
- We consider one GPU as a single resource
- All arrows are multiplicative

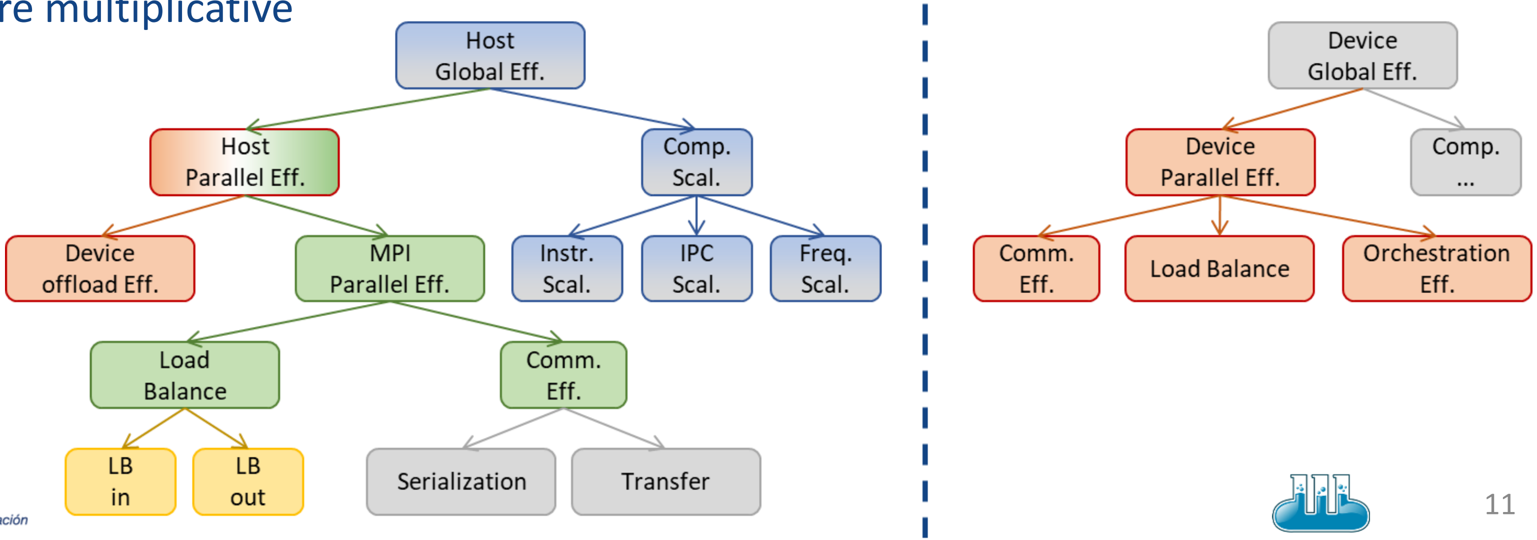
GPU metrics

Metrics not computed by TALP

Scalability metrics computed with several TALP runs

Additional metrics

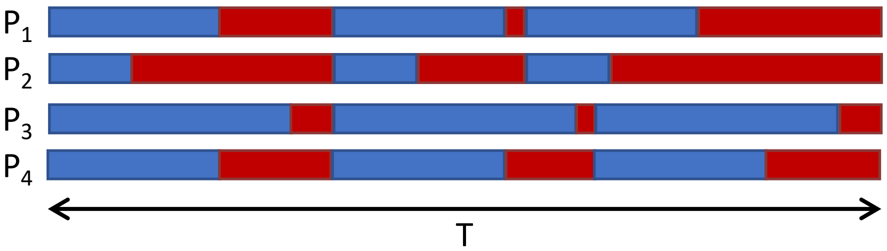
Efficiency metrics



Computing
MPI TALP Metrics

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

States



The status of a process is simplified to two states:

🟦 Useful

🟥 Not Useful

One process correspond to one core.
Core as the resource unit

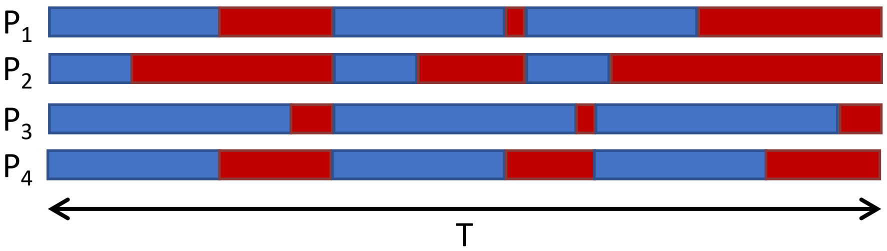
We define:

- C_i as the useful time of P_i
- T as the total elapsed time
- P as the number of processes

Parallel Efficiency

🟦 Useful

🟥 Not Useful



We define:

- C_i as the useful time of P_i
- T as the total elapsed time
- P as the number of processes

$$Par.Eff. = \frac{\sum_{i=1}^{P} c_i}{T*P} = \frac{\text{🟦}}{\text{🟦} + \text{🟥}}$$

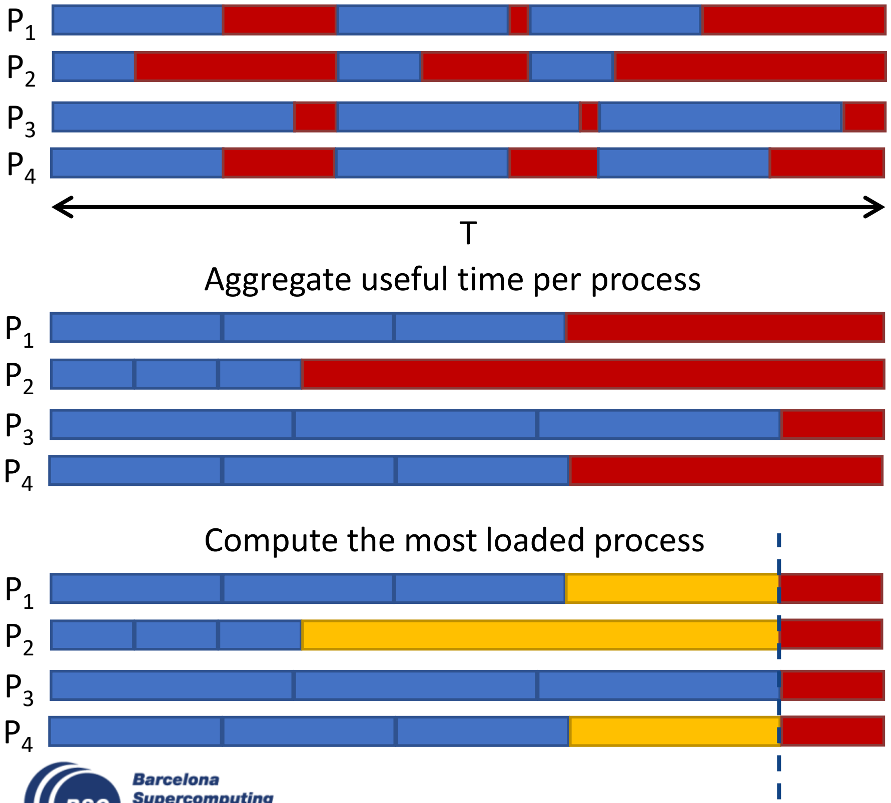
Quantifies how much time the resources are used to do useful work

Load Balance

🟦 Useful

🟥 Not Useful

Quantifies how much time the resources are idle due to one process having more work than the others



Aggregate useful time per process

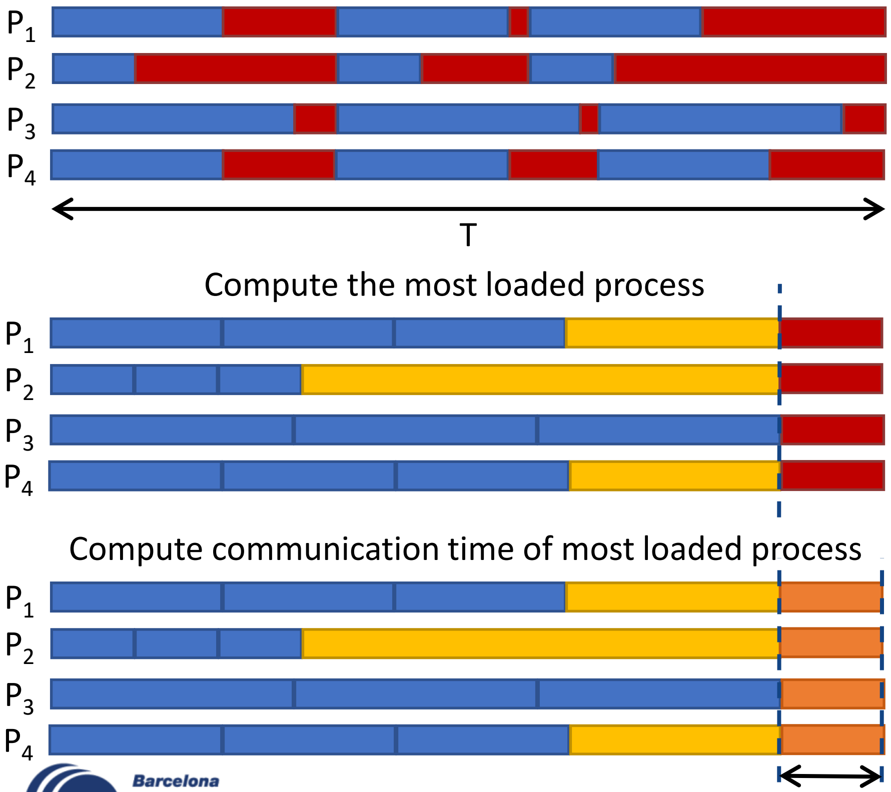
Compute the most loaded process

$$LB = \frac{\sum_{i=1}^{P} c_i}{\max(\,c_i) * P} = \frac{\text{🟦}}{\text{🟦} + \text{🟨}}$$

Communication Efficiency

🟦 Useful

🟥 Not Useful



Compute the most loaded process

Compute communication time of most loaded process

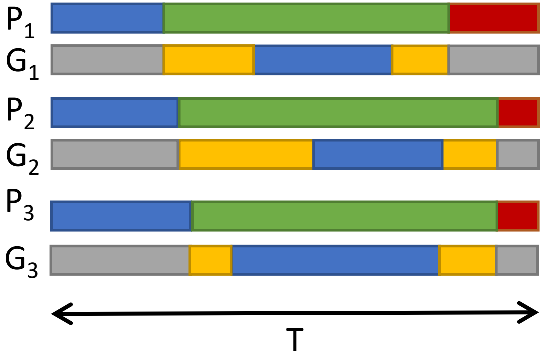
Quantifies how much time the resources are idle due to the non-immediate nature of communications

$$Comm = \frac{\max(c_i)}{T} = \frac{\text{🟦} + \text{🟨}}{\text{🟦} + \text{🟧} + \text{🟨}}$$

Computing TALP GPU metrics

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

States



P = num procs

N = num GPUs

T = Total elapsed time

C_i = Useful time of CPU i

W_i = Wait time of CPU i

G_i = Useful time of GPU i

3 states for the host (CPU):

- Useful
- Waiting 4 GPU
- MPI

Whenever the CPU is not doing useful work because it is managing the GPU. Inside CUDA API: launching kernels, sending data to GPU, waiting for data…

3 states for the device (GPU):

- Useful
- Data transfer
- Idle

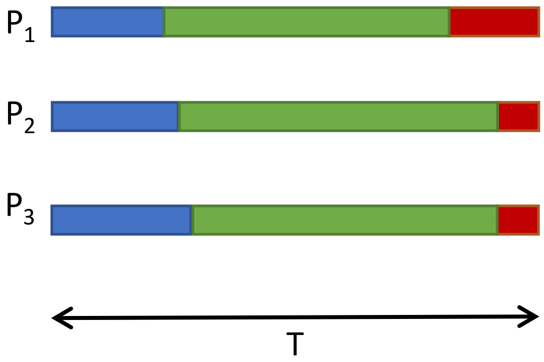
Whenever the GPU is waiting for data communication: GPU-GPU, or GPU-CPU. If communication is overlapped with computation, it is considered computation.

Host Metrics

🟦 Useful

🟩 Waiting 4 GPU

🟥 MPI



- To compute MPI metrics we consider 🟩 Waiting 4 GPU as useful
- First, we blame MPI, then, Device offload

P = num procs

T = Total elapsed time

C_i = Useful time of CPU i

W_i = Wait time of CPU i

$$Par.Eff. = \frac{\sum_{i=1}^{P} c_i}{T*P} = \frac{\text{🟦}}{T*P}$$

$$MPI\ Par.Eff. = \frac{\sum_{i=1}^{P} c_i + \sum_{i=1}^{P} w_i}{T*P} = \frac{\text{🟦} + \text{🟩}}{T*P}$$

$$Device\ offload\ Eff. = \frac{\sum_{i=1}^{P} c_i}{\sum_{i=1}^{P} c_i + \sum_{i=1}^{P} w_i} = \frac{\text{🟦}}{\text{🟦} + \text{🟩}}$$

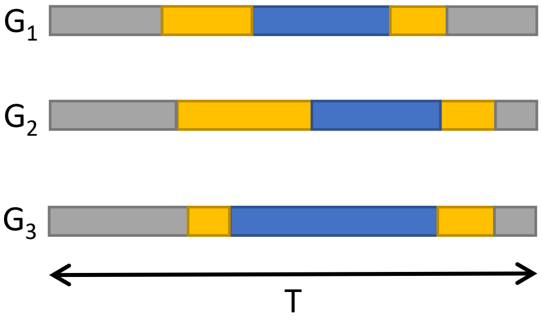
Quantifies how much time the resources are idle due the use of devices

Device Parallel Efficiency

🟦 Useful

🟨 Data move

⬜ Idle



N = num GPUs

T = Total elapsed time

G_i = Useful time of GPU i

D_i = Time waiting for data in GPU i

$$Par.Eff. = \frac{\sum_{i=1}^{N} G_i}{T*N} = \frac{\blacksquare}{T*N}$$

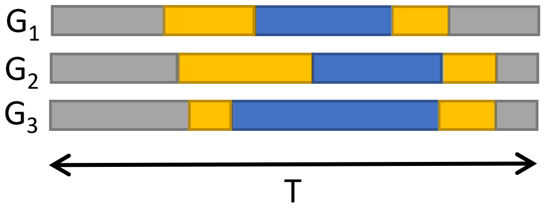
Quantifies how much time the devices are used to do useful work

Device Load Balance

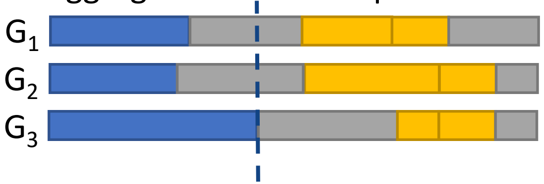
🟦 Useful

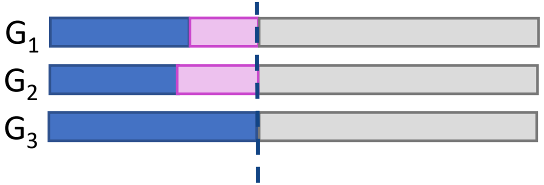
🟨 Data move

⬜ Idle



Aggregate useful time per device





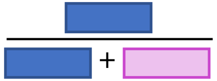
N = num GPUs

T = Total elapsed time

G_i = Useful time of GPU i

D_i = Time waiting for data in GPU i

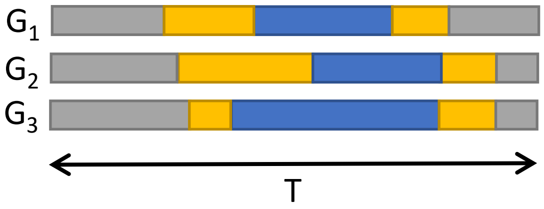
$$LB = \frac{\sum_{i=1}^{N} G_i}{\max(G_i) * N} =$$



Quantifies how much time the devices are idle due to one device spending more time in useful work than others

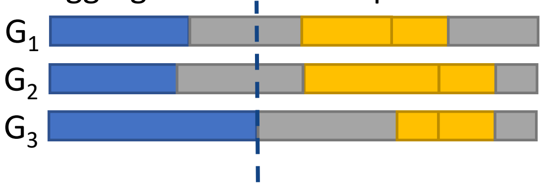
Device Communication Efficiency

🟦 Useful
🟨 Data move
⬜ Idle



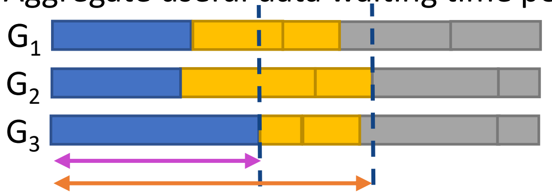
N = num GPUs
T = Total elapsed time
G_i = Useful time of GPU i
D_i = Time waiting for data in GPU i

Aggregate useful time per device



$$\text{Comm. } Eff. = \frac{\max(G_i)}{\max(G_j + D_j))} = \frac{\longleftrightarrow}{\longleftrightarrow}$$

Aggregate useful data waiting time per device



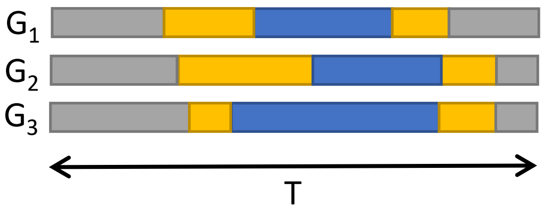
Quantifies how much time the devices are busy due to data movements

Device Orchestration Efficiency

Useful

Data move

Idle



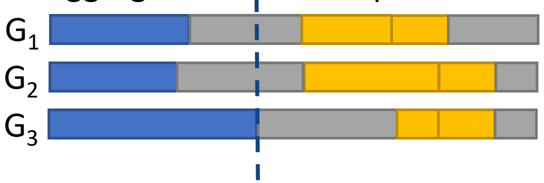
N = num GPUs

T = Total elapsed time

G_i = Useful time of GPU i

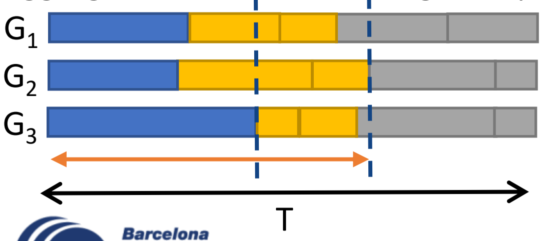
D_i = Time waiting for data in GPU i

Aggregate useful time per device



$$Orch.Eff. = \frac{\max(G_i + D_j)}{T} = \frac{\longleftrightarrow}{T}$$

Aggregate useful data waiting time per device



Quantifies how much time the devices are idle because there is no pending work to do

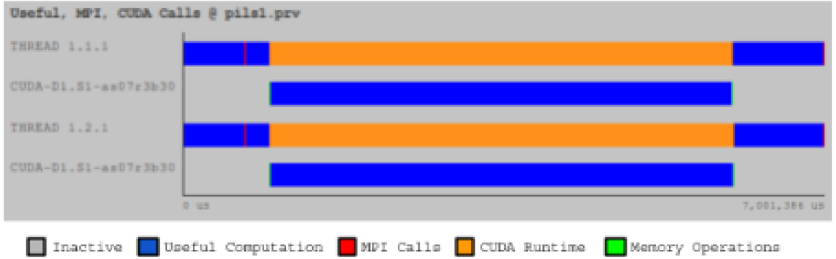
Summary of metrics explanation

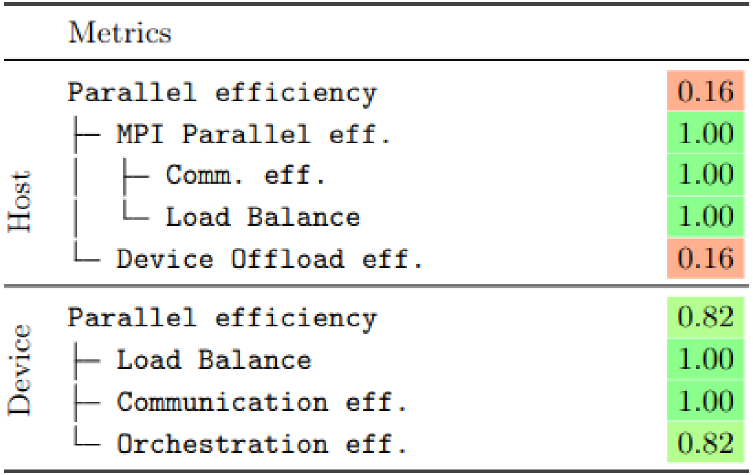
- Host:
  - **Device offload Eff.:** Inefficiency due to use of accelerator.
    - Includes: offloading work, waiting for kernels, waiting for data or sending data to accelerator.
    - Will be very low for applications that “only” use the device.
- Device:
  - **Orchestration Eff.**: Inefficiency due to lack of available work to do.
    - Includes: waiting for work from host, dependencies between kernels…
    - Low value indicates the GPU is not efficiently used because lack of work to do
  - **Communication Eff.**: Inefficiency due to data movement not instantaneous.
    - Includes: waiting for data from host, sending data to host, sending data to other accelerator, NCCL comm, MPI CUDA aware communication.
  - **Load Balance:** Inefficiency due to not all the GPUs computing the same amount of time
    - Does not differentiate between GPUs from the same process or different processes
    - Maybe in the future we can add child metrics similar to LB_in and LB_out
  - **Computation**: How well the resources inside the accelerator are being used.
    - TBD: streams, warps, occupancy, instructions, tensor core use….

TALP GPU metrics examples

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Most work offloaded to GPU. CPUs well balanced, GPUs well balanced

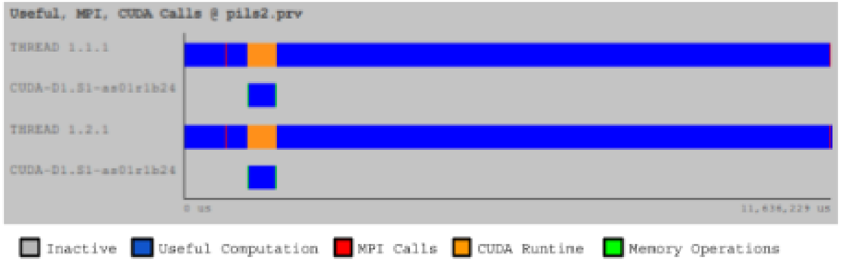


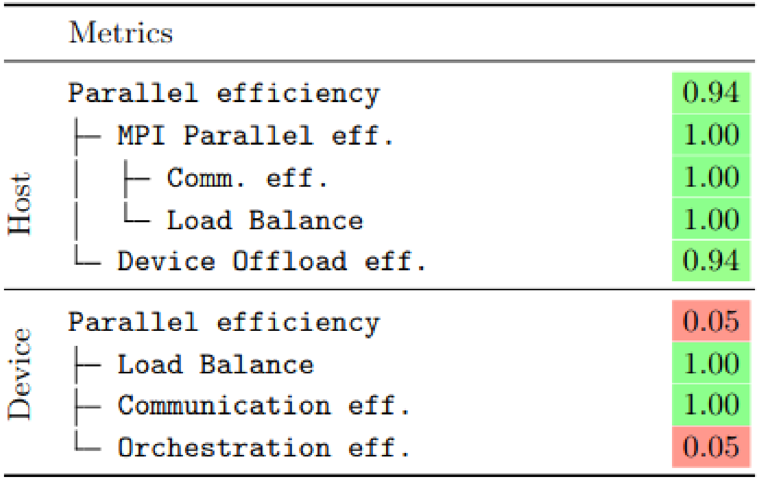


	Metrics	
Host	Parallel efficiency	0.16
	├ MPI Parallel eff.	1.00
	│ ├ Comm. eff.	1.00
	│ └ Load Balance	1.00
	└ Device Offload eff.	0.16
Device	Parallel efficiency	0.82
	├ Load Balance	1.00
	├ Communication eff.	1.00
	└ Orchestration eff.	0.82

- ➢ Low Offload efficiency indicates CPUs are only used to offload work to GPUs
- ➢ Good device efficiency, GPUs are used efficienty

Few work offloaded to GPU.
CPUs well balanced, GPUs well balanced

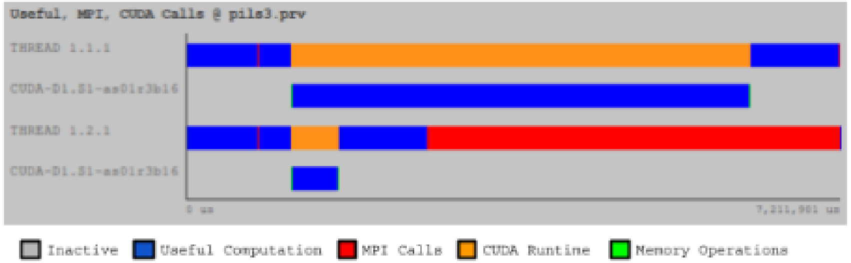


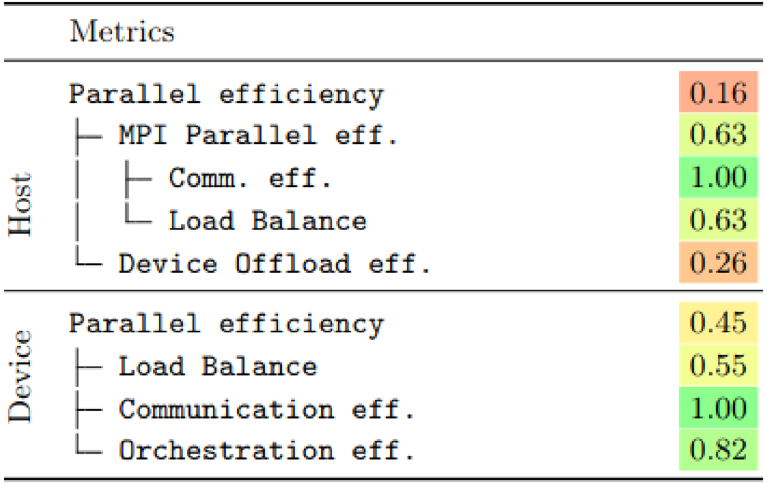


	Metrics	
Host	Parallel efficiency	0.94
	├─ MPI Parallel eff.	1.00
	│ ├─ Comm. eff.	1.00
	│ └─ Load Balance	1.00
	└─ Device Offload eff.	0.94
Device	Parallel efficiency	0.05
	├─ Load Balance	1.00
	├─ Communication eff.	1.00
	└─ Orchestration eff.	0.05

- Good host efficiency
  - CPUs are used efficiently
- Low orchestration efficiency indicates GPUs are not used efficiently because not enough work is offloaded to them

Load imbalance between GPUs. CPUs well balanced

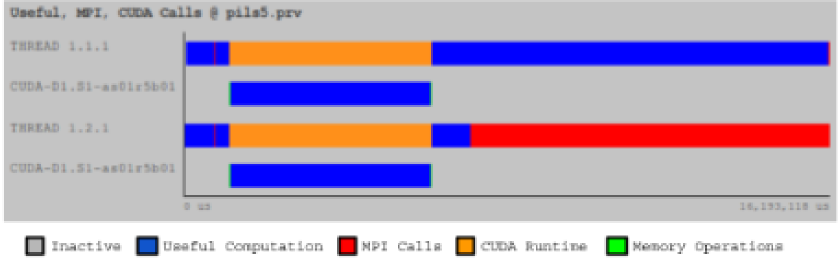


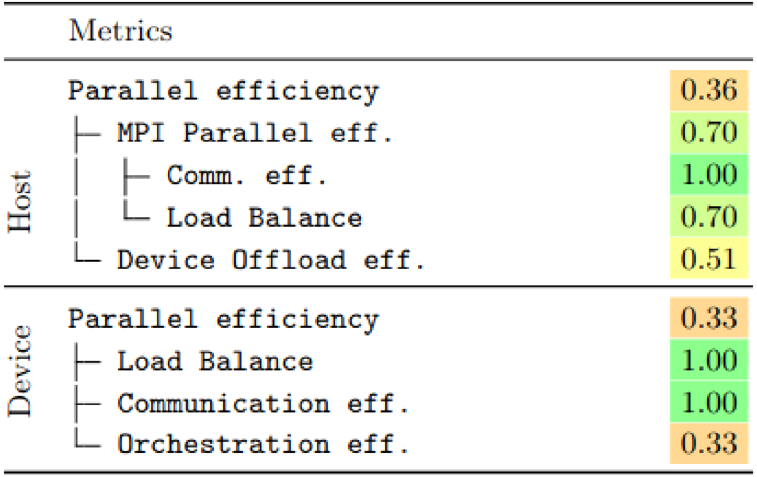


	Metrics	
Host	Parallel efficiency	0.16
	├─ MPI Parallel eff.	0.63
	│ ├─ Comm. eff.	1.00
	│ └─ Load Balance	0.63
	└─ Device Offload eff.	0.26
Device	Parallel efficiency	0.45
	├─ Load Balance	0.55
	├─ Communication eff.	1.00
	└─ Orchestration eff.	0.82

- Low MPI Load Balance
  - Indicates one process has more work to do than the other
- Low Offload efficiency
  - Indicates CPUs are not being used while waiting for GPUs
- Low device Load Balance
  - Indicates GPUs are not well balanced

GPUs well balanced
CPUs unbalanced

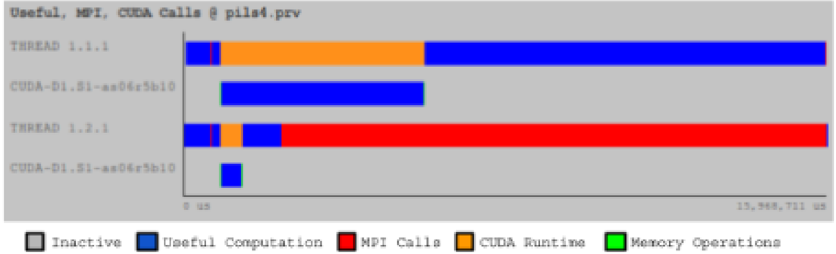


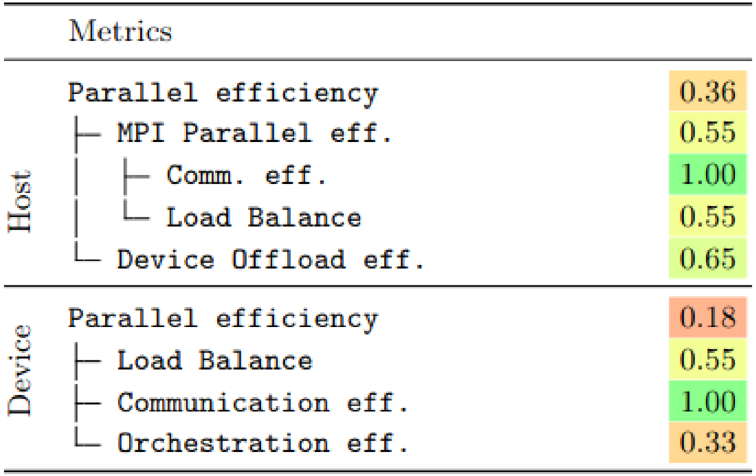


	Metrics	
Host	Parallel efficiency	0.36
	├ MPI Parallel eff.	0.70
	│ ├ Comm. eff.	1.00
	│ └ Load Balance	0.70
	└ Device Offload eff.	0.51
Device	Parallel efficiency	0.33
	├ Load Balance	1.00
	├ Communication eff.	1.00
	└ Orchestration eff.	0.33

- Low MPI Load Balance
  - Indicates there is load imbalance between processes
- Low offload efficiency
  - Indicates CPUs are not working while waiting for GPUs
- Low orchestration efficiency
  - Indicates not enough work is being offloaded to GPUs

GPUs unbalanced. CPUs unbalanced

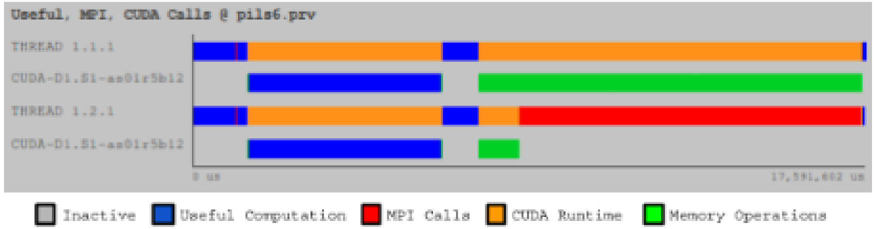




	Metrics	
Host	Parallel efficiency	0.36
	├─ MPI Parallel eff.	0.55
	│ ├─ Comm. eff.	1.00
	│ └─ Load Balance	0.55
	└─ Device Offload eff.	0.65
Device	Parallel efficiency	0.18
	├─ Load Balance	0.55
	├─ Communication eff.	1.00
	└─ Orchestration eff.	0.33

- Low MPI Load Balance
  - Indicates there is load imbalance between processes
- Low offload efficiency
  - Indicates CPUs are not working while waiting for GPUs
- Low device Load Balance
  - Indicates GPUs are not well balanced
- Low orchestration efficiency
  - Indicates not enough work is being offloaded to GPUs

CPUs well balanced. GPUs well balanced. Data movement in one of the processes



	Metrics	
Host	Parallel efficiency	0.06
	├─ MPI Parallel eff.	0.72
	│ ├─ Comm. eff.	1.00
	│ └─ Load Balance	0.72
	└─ Device Offload eff.	0.09
Device	Parallel efficiency	0.31
	├─ Load Balance	1.00
	├─ Communication eff.	0.36
	└─ Orchestration eff.	0.86

- Low MPI load balance
  - Indicates imbalance between processes
- Low offload efficiency
  - Indicates CPUs are not being used while waiting for GPUs
- Low device communication efficiency
  - Indicates data movement is limiting the use of the GPUs

TALP Use cases

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

TALP use cases

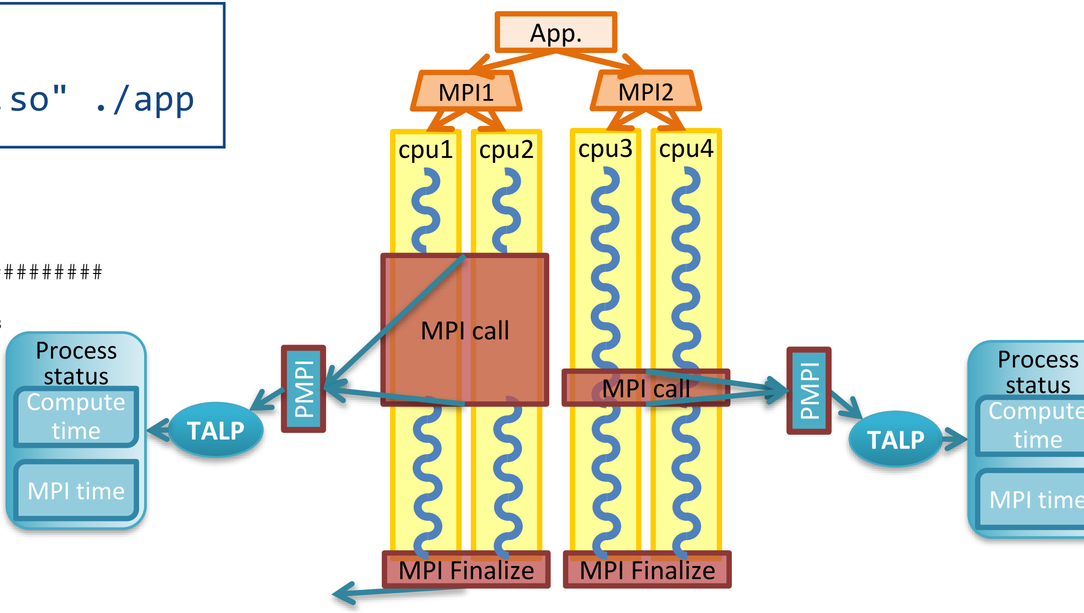
- Use cases examples:
  - Transparent use for the user
  - With user defined regions
  - Getting metrics at runtime
  - TALP pages: Continuous Performance Monitoring

TALP: Transparent use for the user

```
DLB_ARGS=" --talp"

env LD_PRELOAD="$DLB_LIBS/libdlb_mpi.so" ./app
```

```
DLB[...]: ############### Monitoring Region POP Metrics ###############
DLB[...]: ### Name:                                      Global
DLB[...]: ### Elapsed Time:                              31.76 s
DLB[...]: ### Parallel efficiency:                       0.70
DLB[...]: ###  - MPI Parallel efficiency:                0.70
DLB[...]: ###     - Communication efficiency:            1.00
DLB[...]: ###     - Load Balance:                        0.70
DLB[...]: ###        - In:                               0.70
DLB[...]: ###        - Out:                              1.00
DLB[...]: ###  - OpenMP Parallel efficiency:             0.84
DLB[...]: ###     - Load Balance:                        1.00
DLB[...]: ###     - Scheduling efficiency:               1.00
DLB[...]: ###     - Serialization efficiency:            0.84
DLB[...]: ### Computational metrics:
DLB[...]: ###  - Average useful IPC:                     0.59
DLB[...]: ###  - Average useful frequency:               2.95 GHz
DLB[...]: ###  - Number of instructions:                 1.55E+11
```



No knowledge from the user needed
Metrics reported transparently at finalization

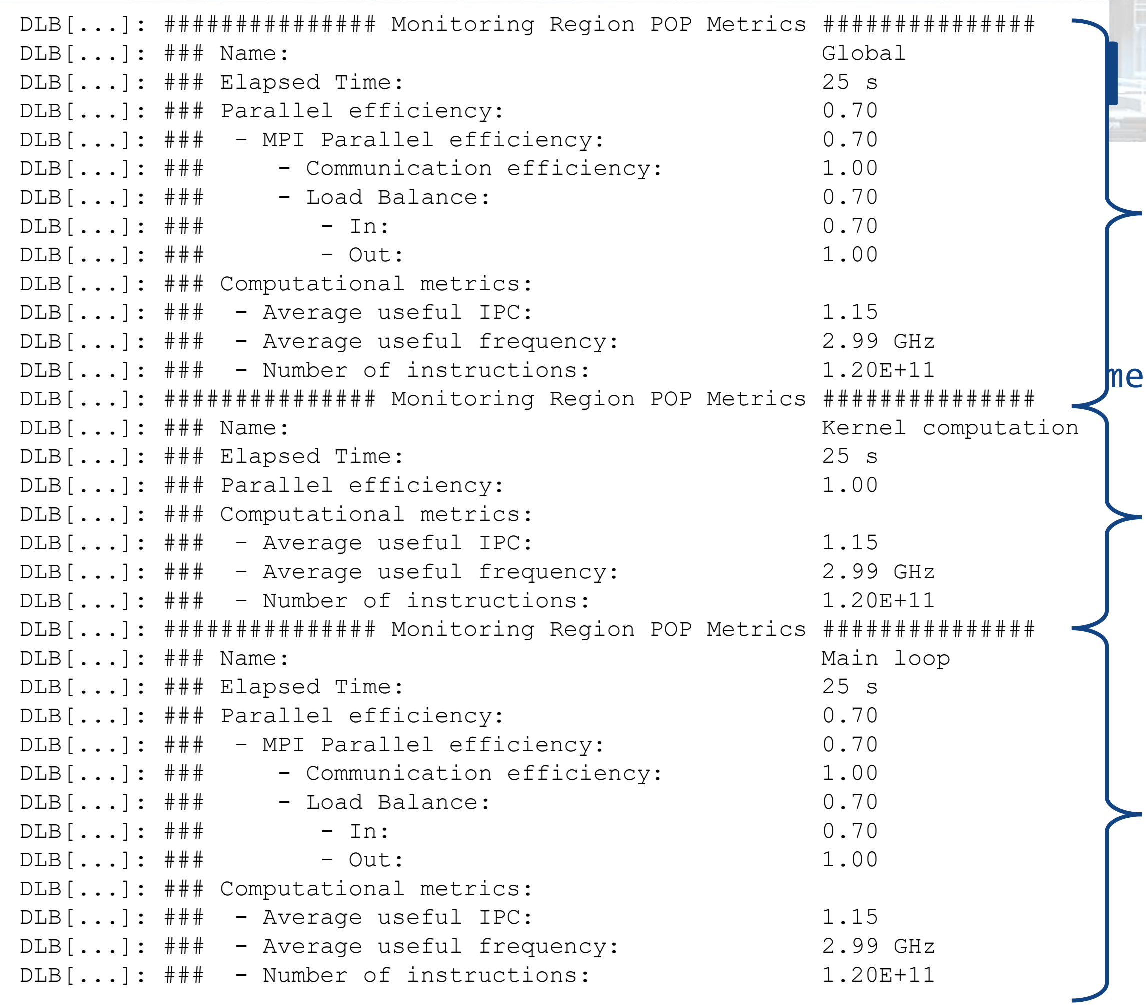
TALP: With user defined regions

```
# include < dlb_talp.h >
...
// Register a new region or obtain an existing handler
dlb_monitor_t * monitor = DLB_MonitoringRegionRegister (“Name”);
// Start TALP monitoring region
DLB_MonitoringRegionStart( monitor );
...
// Stop TALP monitoring region
DLB_MonitoringRegionStop( monitor );
...
```

Code modification needed
Metrics reported by region

l regions

```
# incl
...
// Reg
dlb_mc                                  me”);
// Sta
DLB_Mc
...
// Stc
DLB_Mc
...
```



```
DLB[...]: ############### Monitoring Region POP Metrics ###############
DLB[...]: ### Name:                                     Global
DLB[...]: ### Elapsed Time:                             25 s
DLB[...]: ### Parallel efficiency:                      0.70
DLB[...]: ###   - MPI Parallel efficiency:              0.70
DLB[...]: ###     - Communication efficiency:           1.00
DLB[...]: ###     - Load Balance:                       0.70
DLB[...]: ###       - In:                               0.70
DLB[...]: ###       - Out:                              1.00
DLB[...]: ### Computational metrics:
DLB[...]: ###   - Average useful IPC:                   1.15
DLB[...]: ###   - Average useful frequency:             2.99 GHz
DLB[...]: ###   - Number of instructions:               1.20E+11
DLB[...]: ############### Monitoring Region POP Metrics ###############
DLB[...]: ### Name:                                     Kernel computation
DLB[...]: ### Elapsed Time:                             25 s
DLB[...]: ### Parallel efficiency:                      1.00
DLB[...]: ### Computational metrics:
DLB[...]: ###   - Average useful IPC:                   1.15
DLB[...]: ###   - Average useful frequency:             2.99 GHz
DLB[...]: ###   - Number of instructions:               1.20E+11
DLB[...]: ############### Monitoring Region POP Metrics ###############
DLB[...]: ### Name:                                     Main loop
DLB[...]: ### Elapsed Time:                             25 s
DLB[...]: ### Parallel efficiency:                      0.70
DLB[...]: ###   - MPI Parallel efficiency:              0.70
DLB[...]: ###     - Communication efficiency:           1.00
DLB[...]: ###     - Load Balance:                       0.70
DLB[...]: ###       - In:                               0.70
DLB[...]: ###       - Out:                              1.00
DLB[...]: ### Computational metrics:
DLB[...]: ###   - Average useful IPC:                   1.15
DLB[...]: ###   - Average useful frequency:             2.99 GHz
DLB[...]: ###   - Number of instructions:               1.20E+11
```

Main region

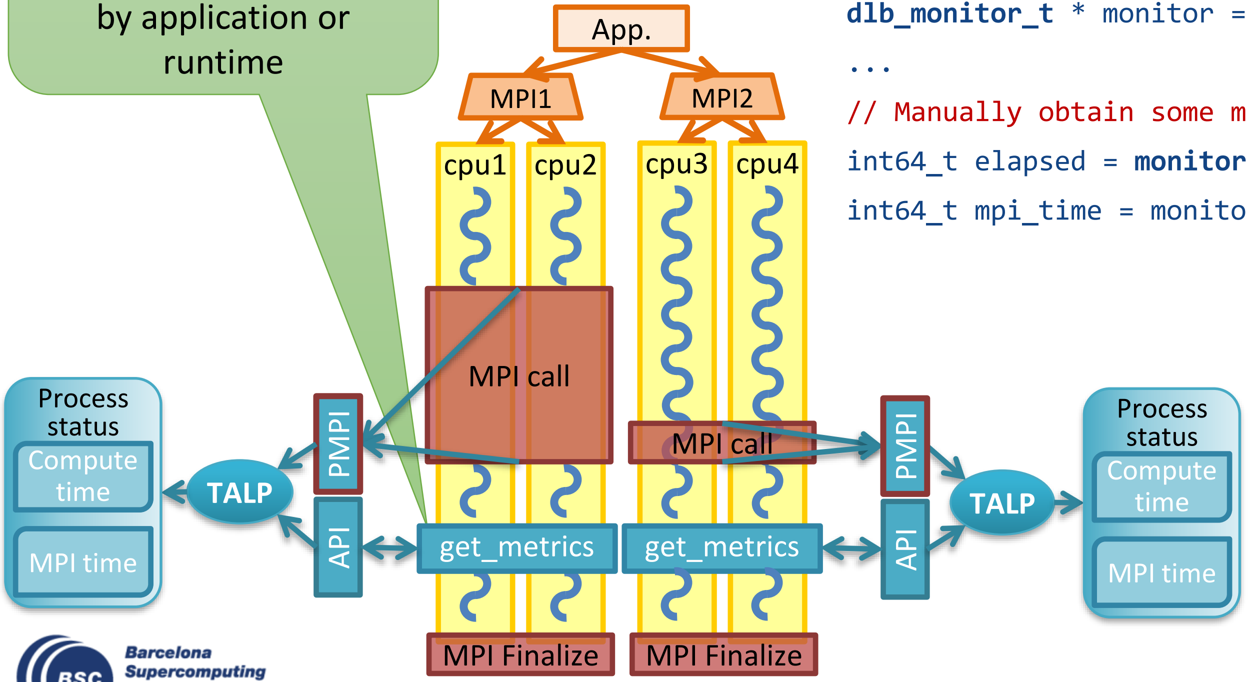
Region without MPI code

Region with MPI code

TALP: getting metrics at runtime

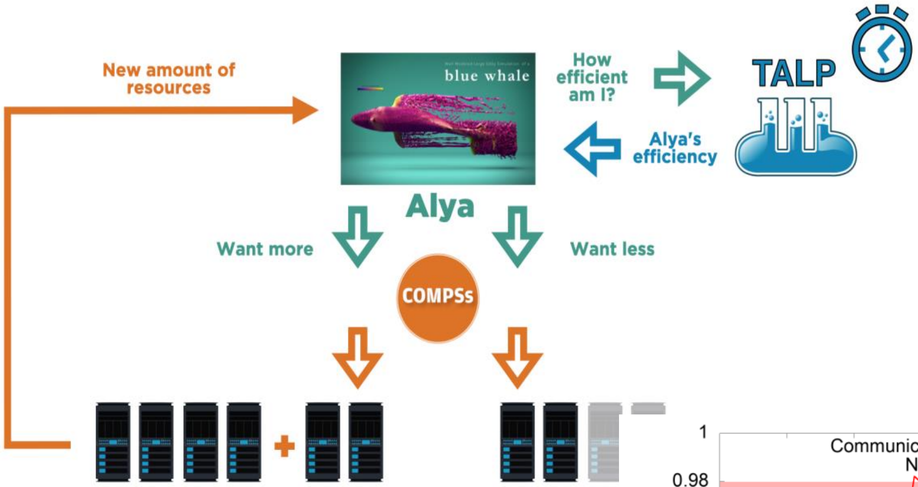
At this point, informed decisions to improve efficiency can be taken by application or runtime

```
# include < dlb_talp.h >
...
// Register a new region or obtain an existing handler
dlb_monitor_t * monitor = DLB_MonitoringRegionRegister (“Name”);
...
// Manually obtain some metrics from the monitor
int64_t elapsed = monitor->elapsed_time;
int64_t mpi_time = monitor->mpi_time
```



Enable dynamic resource management, load balancing or malleability

Success story 3: Malleable simulation



New amount of resources

blue whale

How efficient am I?

TALP

Alya's efficiency

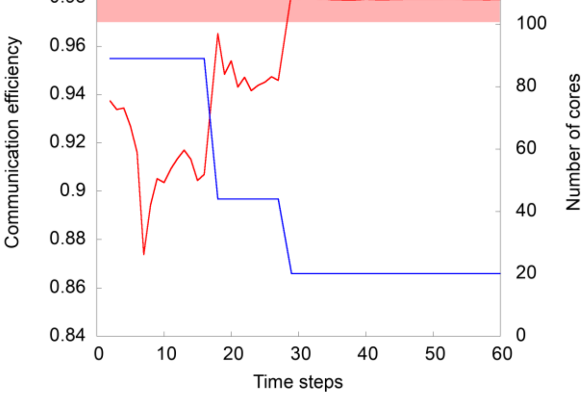
Alya

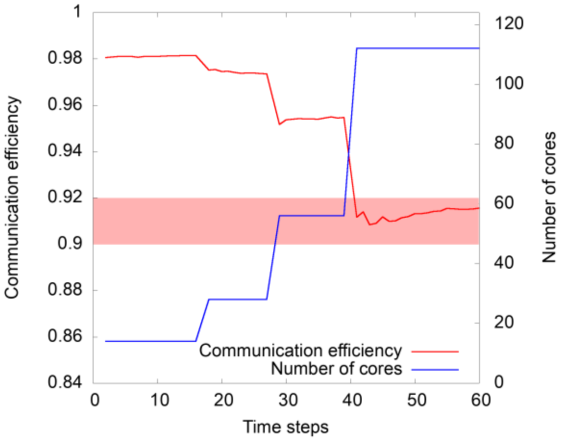
Want more

Want less

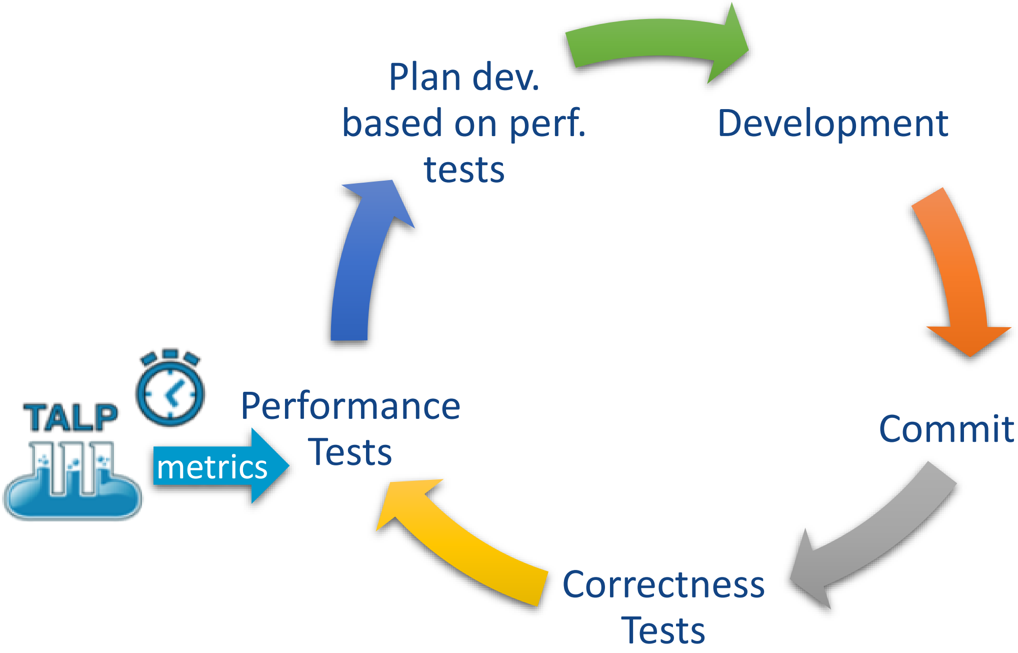
COMPSs

- ➢ Application obtain efficiency metrics through TALP
- ➢ Adjust resources accordingly using COMPSs
- ➢ Target efficiency reached after some iterations





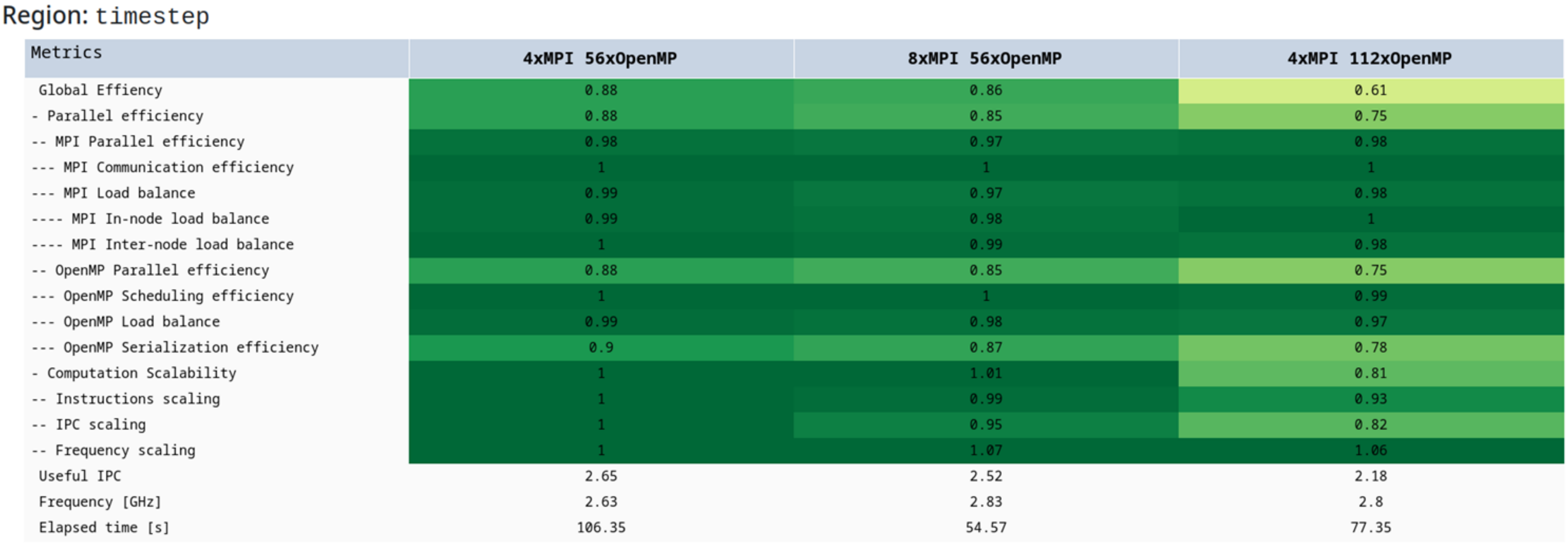
TALP pages: Continuous Performance Monitoring



Integrate in CI/CD platform to detect performance issues added

- Two visualization modes available:
  - **Scaling efficiency** tables with POP-like metrics (to gain **insight**)
  - **Metrics evolution** plots of POP-like metrics (to track **regression**)

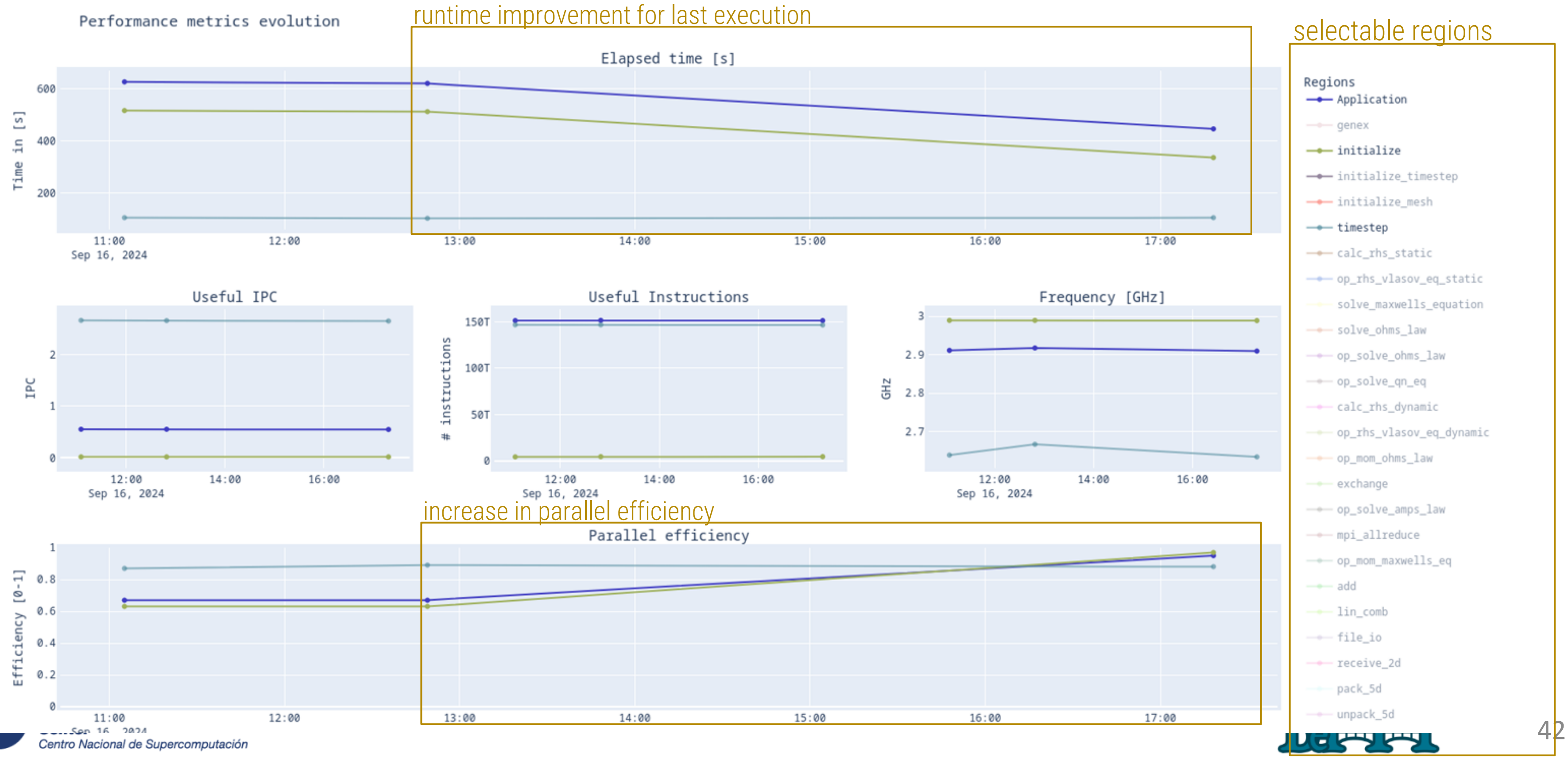
TALP pages: Scaling efficiency



Region: `timestep`

Metrics	4xMPI 56xOpenMP	8xMPI 56xOpenMP	4xMPI 112xOpenMP
Global Effiency	0.88	0.86	0.61
- Parallel efficiency	0.88	0.85	0.75
-- MPI Parallel efficiency	0.98	0.97	0.98
--- MPI Communication efficiency	1	1	1
--- MPI Load balance	0.99	0.97	0.98
---- MPI In-node load balance	0.99	0.98	1
---- MPI Inter-node load balance	1	0.99	0.98
-- OpenMP Parallel efficiency	0.88	0.85	0.75
--- OpenMP Scheduling efficiency	1	1	0.99
--- OpenMP Load balance	0.99	0.98	0.97
--- OpenMP Serialization efficiency	0.9	0.87	0.78
- Computation Scalability	1	1.01	0.81
-- Instructions scaling	1	0.99	0.93
-- IPC scaling	1	0.95	0.82
-- Frequency scaling	1	1.07	1.06
Useful IPC	2.65	2.52	2.18
Frequency [GHz]	2.63	2.83	2.8
Elapsed time [s]	106.35	54.57	77.35

TALP Pages: Metrics evolution



runtime improvement for last execution

selectable regions

increase in parallel efficiency

Summary

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Summary

- TALP is a lightweight tool to gather efficiency metrics
  - At finalization
    - Allows continuous performance monitoring
    - Integration with CI/CD systems (TALP-pages)
  - At runtime
    - Allows dynamic adjustment of execution
      - Enable load balancing
      - Dynamic resource management
  - Its low overhead allows its use in production runs
  - Does not store TB of data
  - Provides API to annotate regions and maximize the information gathered
  - Will indicate when detailed analysis using traces is needed

Summary

- Version 3.6.0-beta1 (2025-9)
  - MPI metrics – Fully supported
  - Hardware counters (Instructions, cycles, IPC) – Fully supported
  - GPU metrics
    - NVIDIA (CUDA and OpenACC) – available
    - AMD (HIP) – Under development
    - Computational metrics – Under development
  - OpenMP metrics – Under testing

- Download DLB (Free Download under LGPL-v3 license):
  - https://pm.bsc.es/dlb-downloads
  - https://github.com/bsc-pm/dlb/releases/tag/v3.6.0-beta1

  - User Guide: https://pm.bsc.es/ftp/dlb/doc/user-guide/
  - Hands-on: https://gitlab.pm.bsc.es/dlb/dlb-training

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

Thank you

marta.garcia@bsc.es

victor.lopez@bsc.es

https://pm.bsc.es/dlb