Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

DLB: Dynamic Load Balancing Library

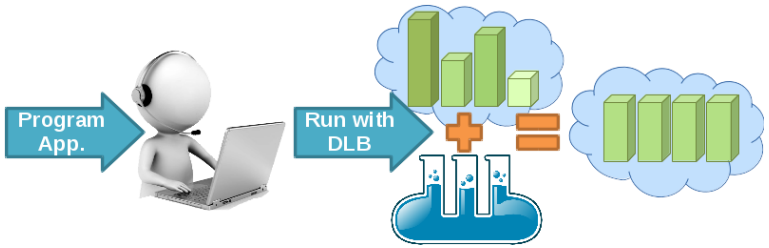
Marta Garcia-Gasulla
Victor Lopez

November 2022

Tutorial

Dynamic Load Balancing - DLB

- Our objectives:
  - Address all sources of imbalance
    - Fine Grain, dynamic…
    - How?
      - Detect imbalance at runtime
      - React immediately
  - Real product for HPC
    - Use common programming model/environment
      - MPI + OpenMP
  - Transparent to the application
    - Runtime library

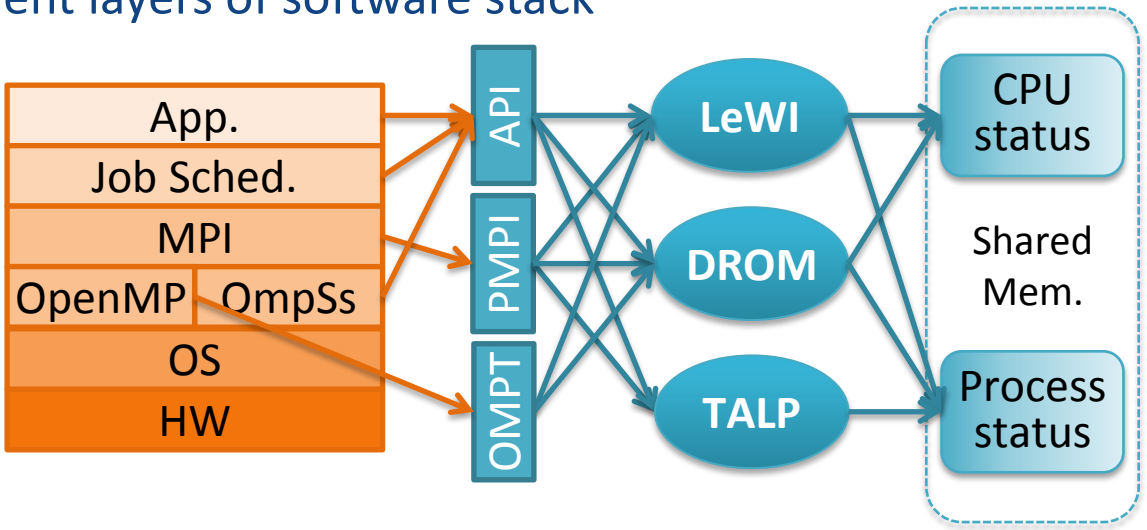


DLB structure

- Three modules:
  - LeWI: For fine grain load balancing
  - DROM: For coarse grain resource management
  - TALP: For performance measurement

Three modules, integrated but independent

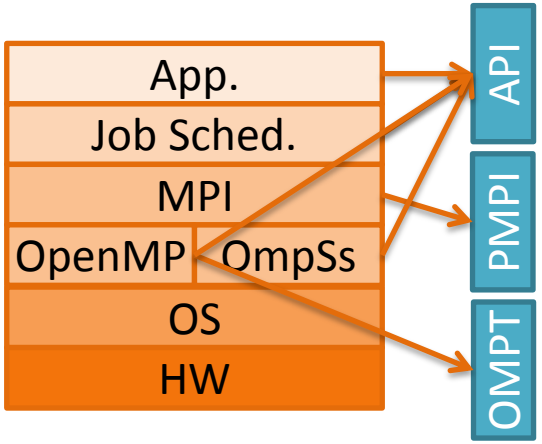
- Common infrastructure
  - Integration with different layers of software stack
  - API
  - Shared memory



DLB integration

- Integration with different layers:
  - Application: DLB offers a user-level API
  - Job Scheduler: DLB offers an API for resource managers
  - MPI: PMPI interception
  - OpenMP:
    - OMPT interception (few OpenMP runtimes support it)
    - API added by user
    - OpenMP free agent threads **NeW!**
  - OmpSs: Runtime integration with DLB library, the OmpSs runtime calls the DLB library

DLB offers mechanisms to be transparent to the application or user. DLB API is useful for advanced users that have a good knowledge of their application.



DLB: Main concepts

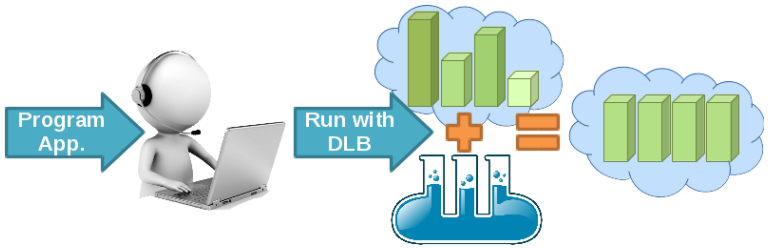
- **CPU (core)**: Minimum computing unit acknowledged by DLB, where one thread (and only one at the same time) can run.
- **Idle CPU**: A CPU that is not being used to do useful computation.
- **Owner**: Process that owns a CPU. A process owns the resources where it is started. A CPU can only be owned by one process at the same time.
- **Lend**: When the owner of a CPU is not using it, the CPU can be lent to the system. When a CPU is lent, a process that it is not its owner can use it.
- **Claim**: When the owner of a CPU wants to use it after lending it, the owner can claim the CPU.
- **Ask for Resources**: A process of the system can ask DLB for idle CPUs to speed up its execution.

LeWI
Lend When Idle

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

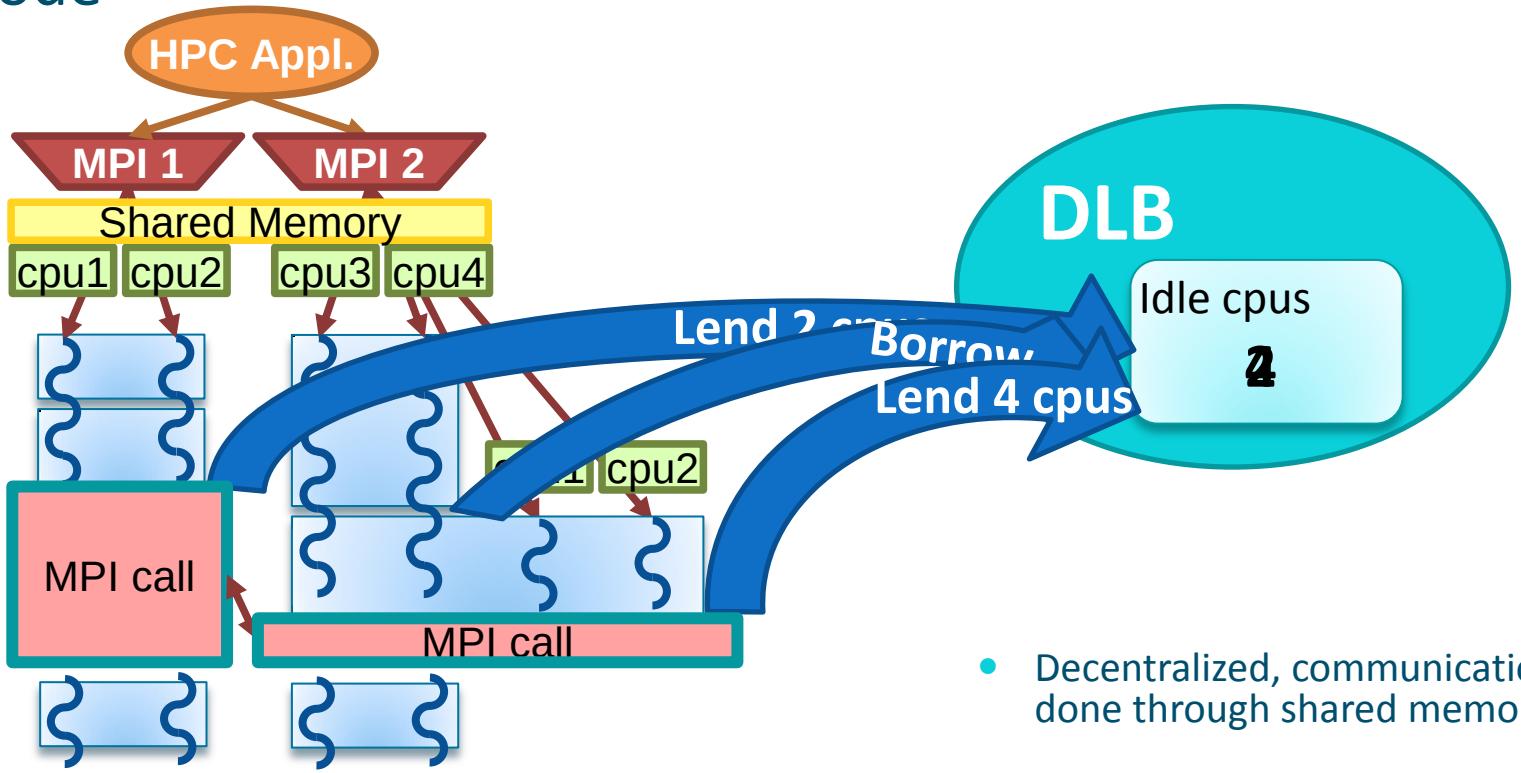
Lend When Idle (LeWI)

- Our objectives:
  - Address all sources of imbalance
    - Fine Grain, dynamic…
    - How?
      - Detect imbalance at runtime
      - React immediately
  - Real product for HPC
    - Use common programming model/environment
      - MPI + OpenMP
  - Transparent to the application
    - Runtime library



Lend When Idle (LeWI)

- **The idea:** Use computational resources of a process when not using them to speed up another process in the same node



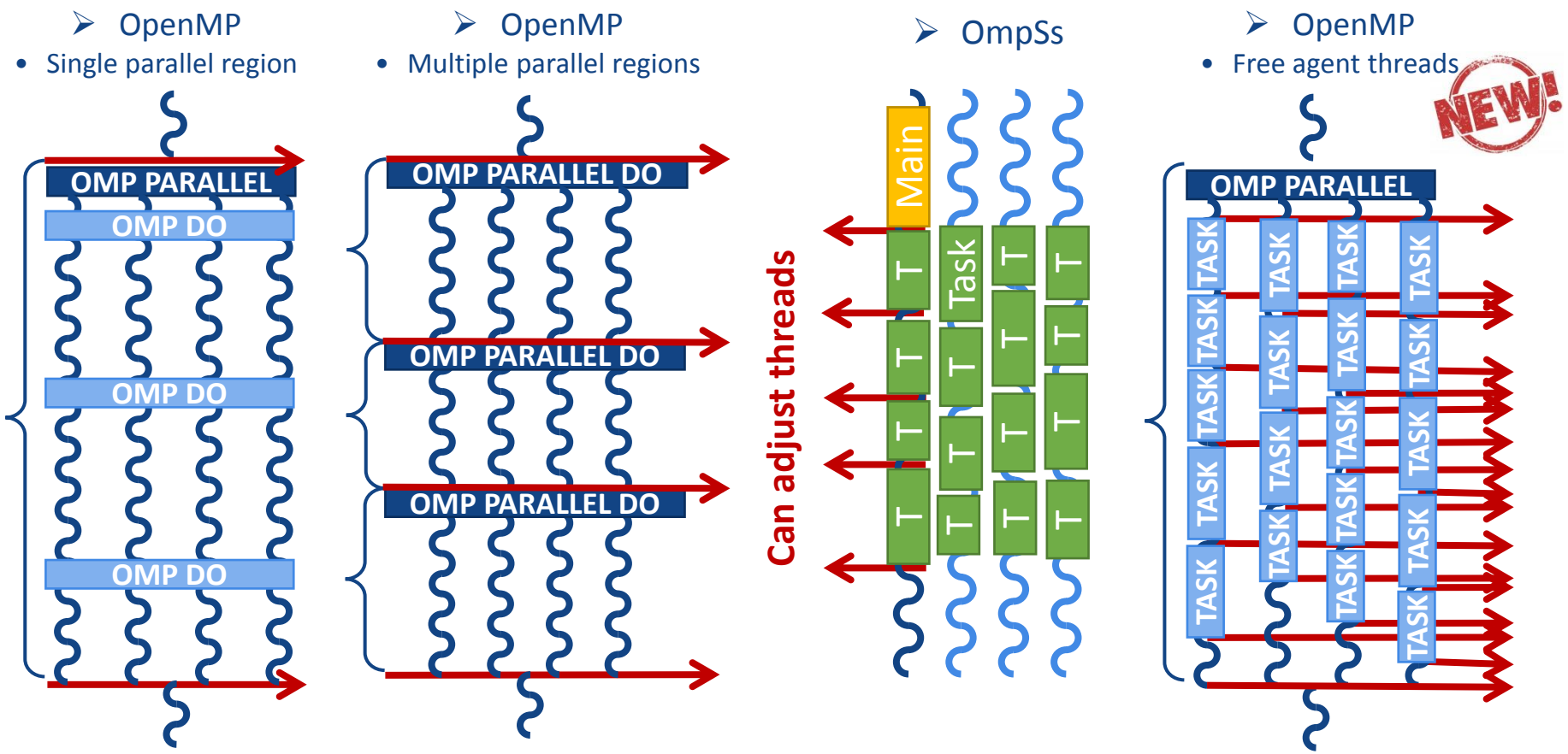
- Decentralized, communication is done through shared memory

LeWI a runtime balancing algorithm for nested parallelism.
In Proceedings of International Conference of Parallel Processing (ICPP 2009).

LeWI: Implementation

- **MPI interception:** Use standard PMPI interception avoids recompiling, DLB can be used with LD_PRELOAD.
- **Second level of parallelism**: We need shared memory parallelism. Current version supports OmpSs and OpenMP.
  - Must be malleable. Lack of malleability limits performance
    - Malleability can be limited by the programming model or the application.
  - **OpenMP**: Three levels/options of integration:
    - **API**: Works with any OpenMP implementation, must modify the code and link with DLB.
    - **OMPT**: Only works with OpenMP implementations implementing OMPT.
    - **Free agents**: Works preloading our own version of OpenMP implementation (LLVM based)
  - **OmpSs:** Fully integrated with DLB, high malleability.

LeWI: Malleability vs. Programming Model

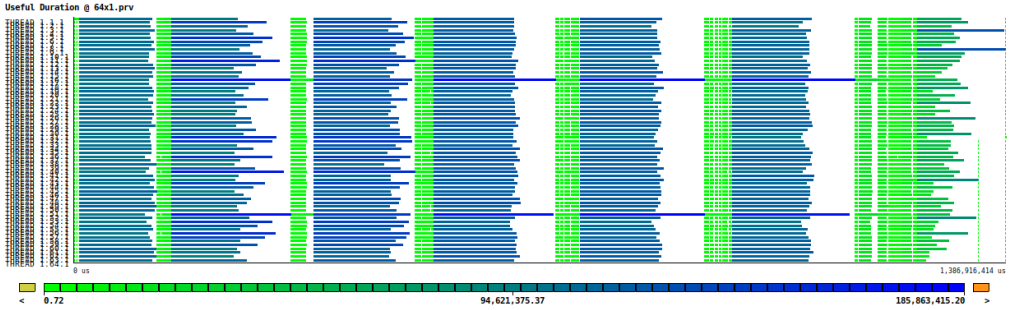


→ Number of threads can be changed

OpenMP/OmpSs Summary

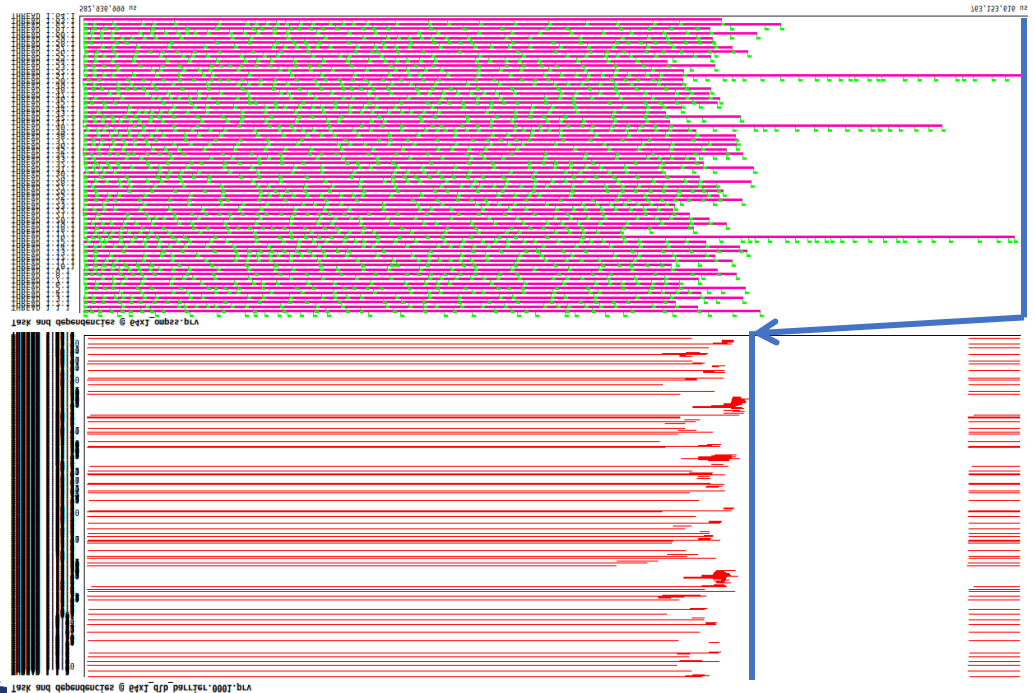
	OpenMP	OpenMP + OMPT	OpenMP + free agent	OmpSs
Requirements	None	Intel OpenMP 2018 / LLVM 8.0 or greater	Own LLVM OpenMP runtime	
CPU binding	No supported	Rebind through OMPT	Yes	Yes
Malleability	Only outside parallel regions	Only outside parallel regions	Malleability through tasking	Yes
Integration	Add DLB_borrow before each parallel region + LD_PRELOAD	LD_PRELOAD	LD_PRELOAD	Transparent, integrated through OmpSs

Success Story 1: ParMMG



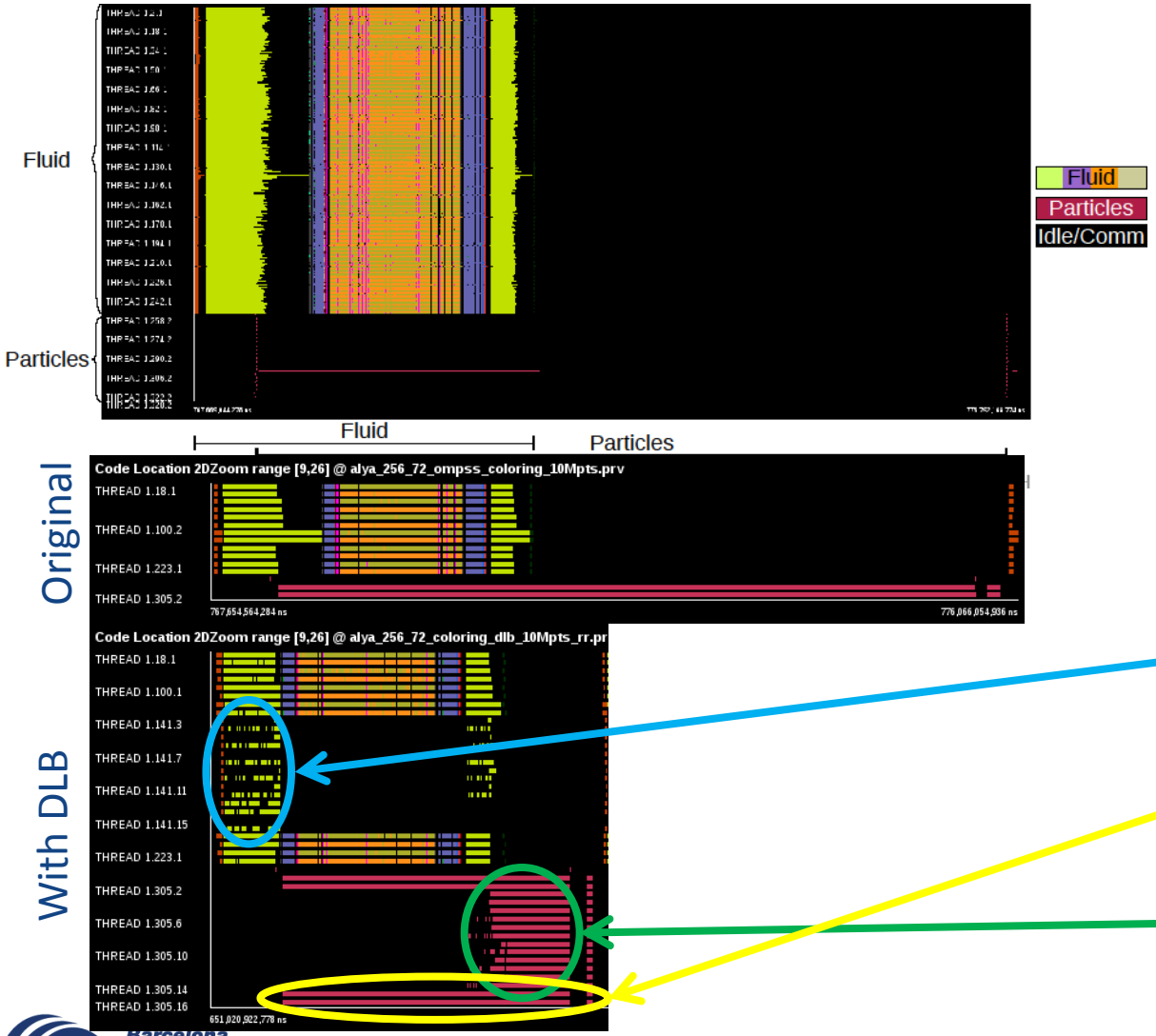
- Parallel mesh adaptation of 3D volume meshes. High imbalance and changing between iterations. Pure MPI code

- Added OmpSs parallelization to one loop + 1 call to DLB API.



- 1.2x Speedup overall execution.

Success Story 2: Alya coupled codes



- Respiratory simulation coupling 2 codes:
  - Fluid + particle tracking
  - 256 MPI ranks Fluid
  - 72 MPI ranks Particles
- Zoom in 1 node with DLB
- 3 Actions:
  - Load Balance Fluid
  - Load Balance Particles
  - Load Balance 2 codes

LeWI API

- int DLB_Enable(void);
  - Enable DLB and all its features in case it was previously disabled otherwise it has no effect
- int DLB_Disable(void);
  - Disable DLB actions for the calling process.
- int DLB_SetMaxParallelism(int max);
  - Set the maximum number of resources to be used by the calling process.
- int DLB_UnsetMaxParallelism(void);
  - Unset the maximum number of resources to be used by the calling process.
- int DLB_Lend(…);
  - Lend current CPUs
- Int DLB_Reclaim(…);
  - Reclaim CPUs owned by the process
- int DLB_AcquireCpu(…);
  - Acquire a specific CPU, equivalent to Reclaim if the process it the owner and to Borrow if not.
- int DLB_Borrow(…);
  - Borrow possible CPUs registered on DLB
- int DLB_Return(…);
  - Return claimed CPUs of other processes
- int DLB_Barrier(void);
  - Barrier between processes in the node

These functions have 4 different versions:
- **void**: Any CPU.
- **int CPU_Id**: the specified CPU
- **int num_CPUs:** the amount of CPUs indicated
- **CPUmask**: the CPUs indicated in the mask

DROM
Dynamic Resource Ownership Management

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

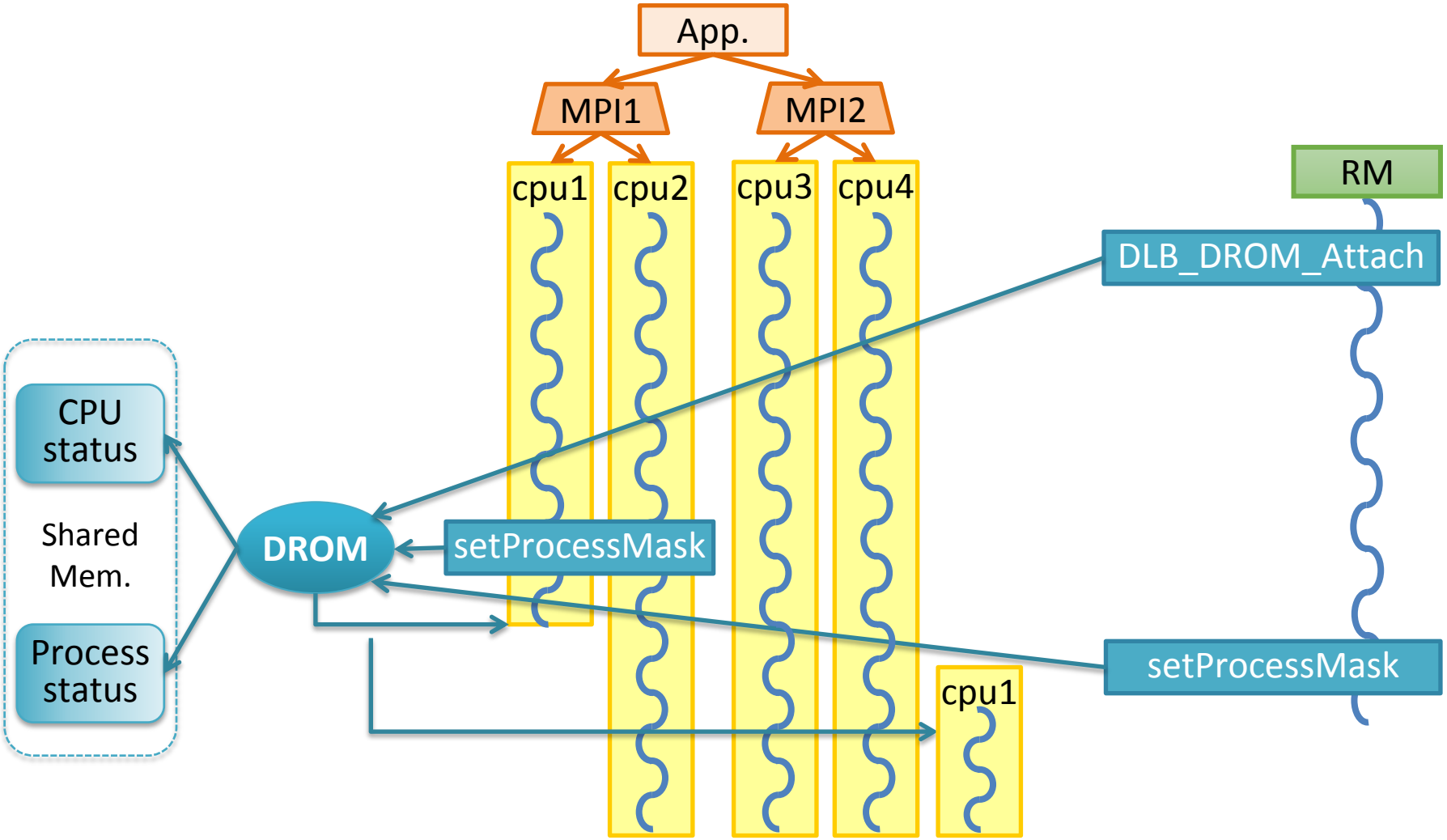
DROM: Dynamic Resource Ownership Management

- API for superior entity
  - Job Scheduler
  - Resource manager
  - User

- Allow to change the **owner** of resources (CPUs)
  - By the process
  - By an external entity (Resource manager)

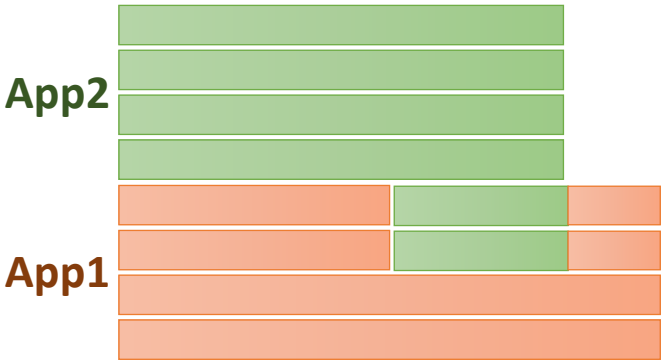
DROM: Enabling Efficient and Effortless Malleability for Resource Managers.
In Proceedings of International Conference on Parallel Processing (ICPP 2018)

DROM: Dynamic Resource Ownership Management

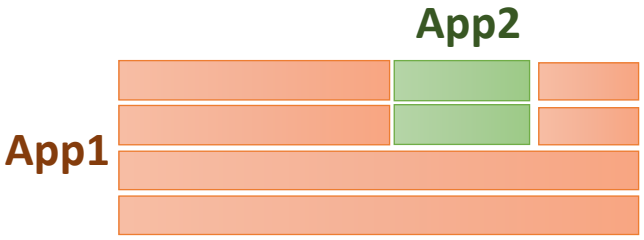


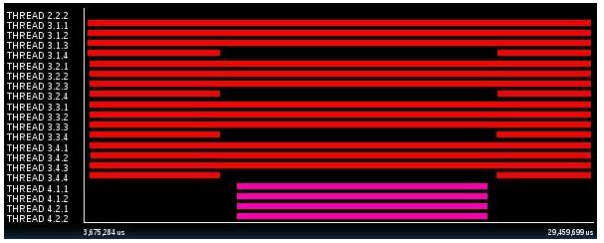
DROM: Use cases

- A) User:
  Increase priority to App2

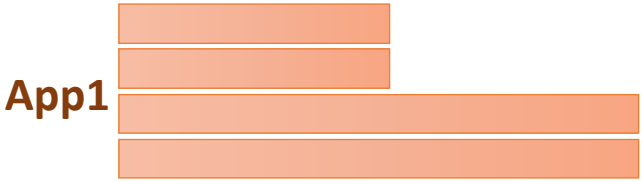


- B) Job Scheduler:
  Run High priority App2 in resources assigned to App1

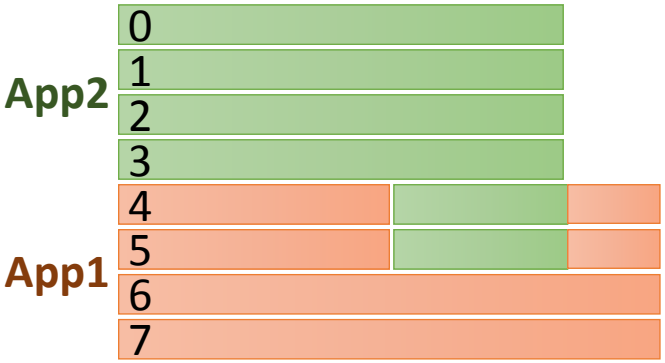




- C) App1: Release 2 CPUs because not using efficiently

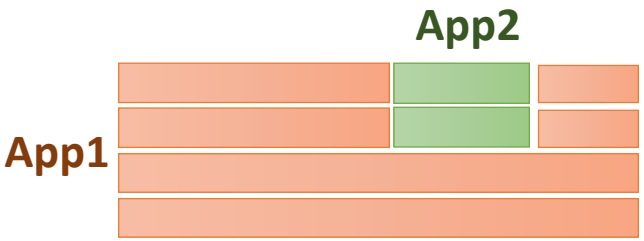


DROM: How to



➢ A)

```
$> dlb_taskset -p pid_app2 -c 0-5
```



➢ B)

```
$> dlb_taskset -c 0,1 ./App2
```

➢ C)

```
DLB_DROM_SetProcessMask(my_pid, [0,0,1,1]);
```

DROM API

- int DLB_DROM_Attach(void);
  - Attach current process to DLB system as DROM administrator
- int DLB_DROM_Detach(void);
  - Detach current process from DLB system
- int DLB_DROM_GetNumCpus(int *ncpus);
  - Get the number of CPUs in the node
- int DLB_DROM_GetPidList(int *pidlist, int *nelems, int max_len);
  - Get the list of running processes registered in the DLB system
- int DLB_DROM_GetProcessMask(int pid, dlb_cpu_set_t mask, dlb_drom_flags_t flags);
  - Get the process mask of the given PID
- int DLB_DROM_SetProcessMask(int pid, const_dlb_cpu_set_t mask, dlb_drom_flags_t flags);
  - Set the process mask of the given PID
- int DLB_DROM_PostFinalize(int pid, dlb_drom_flags_t flags);
  - Unregister a process from the DLB system

TALP
Tracking Application Live Performance

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

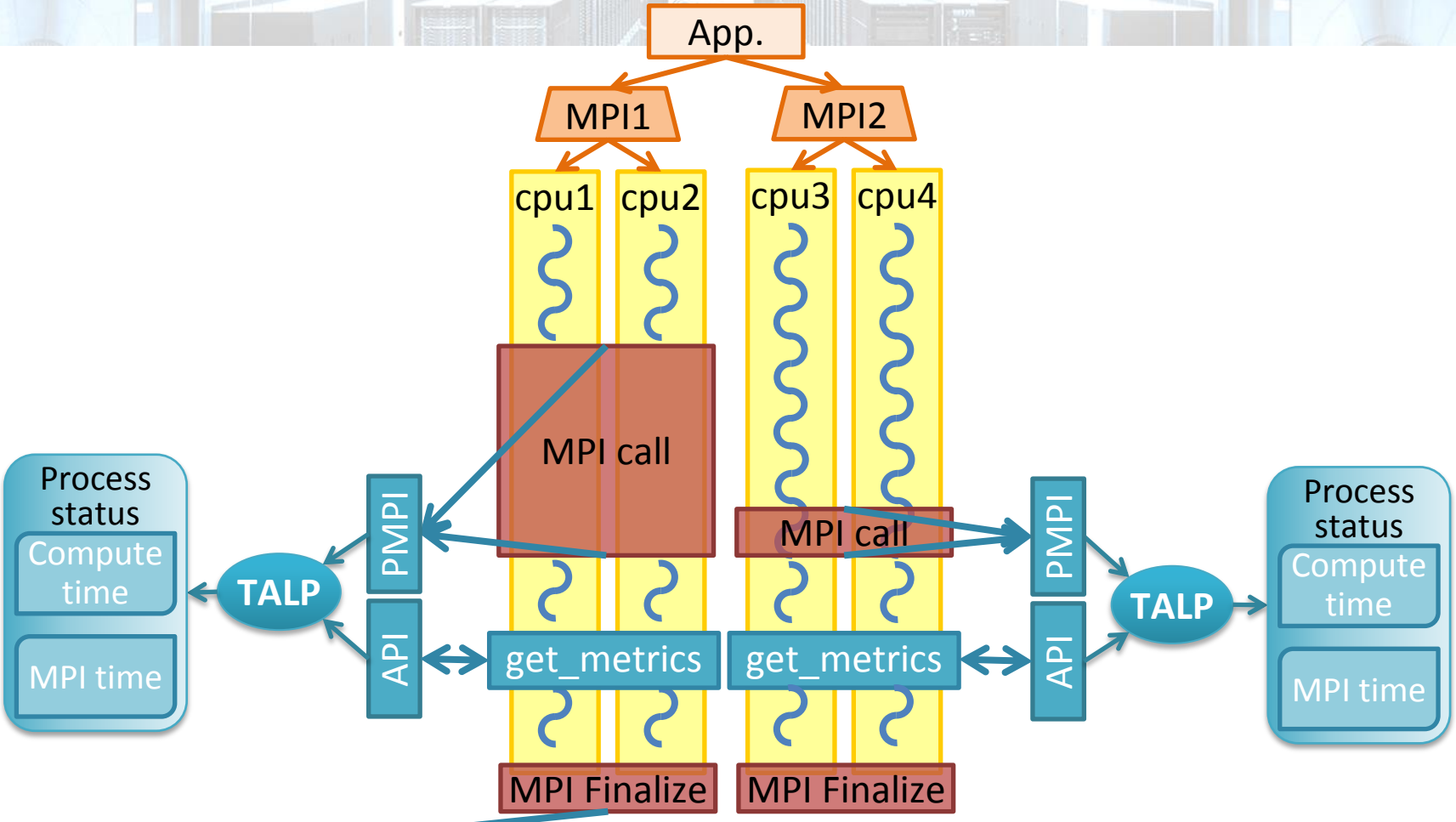
TALP: Tracking Application Live Performance

- Profiling tool with:
  - Low overhead
  - Report POP metrics
  - API to obtain metrics at runtime
  - API to instrument code and profile regions of code

- Current version profiles MPI performance

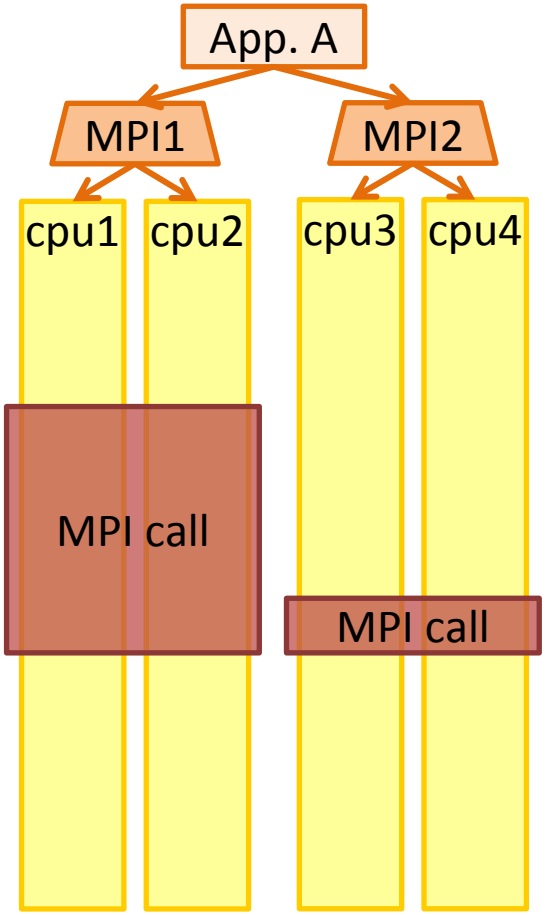
TALP a lightweight tool to Unveil Parallel Efficiency of Large Scale Executions. *In Proceedings of Performance Engineering, Modelling, Analysis, and Visualization Strategy (Permavost 2021).*

TALP

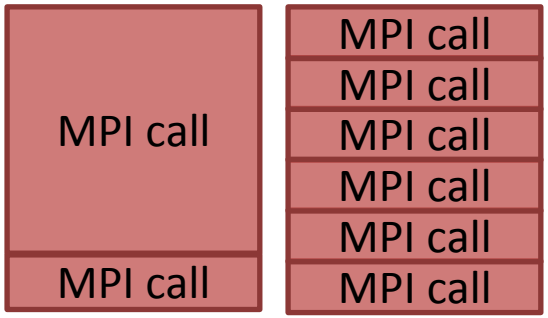



```
######### Monitoring Region App Summary #########
### Name:                       MPI Execution
### Elapsed Time :              2.66 s
### Parallel efficiency :       0.70
###    - Communication eff. :   0.98
###    - Load Balance :         0.72
###       - LB_in :             0.72
###       - LB_out:             1.00
```

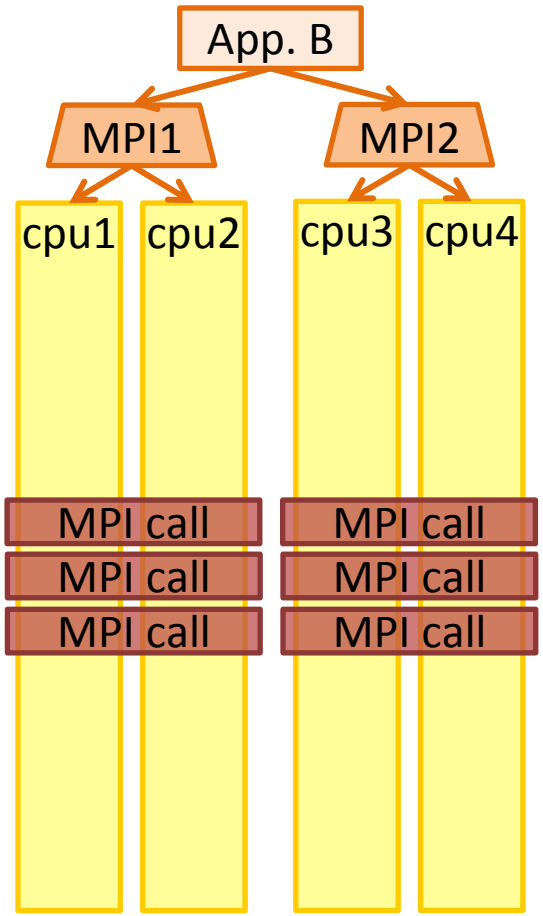
Why is more than “yet another profiling tool”?



- 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



Using TALP

At finalization

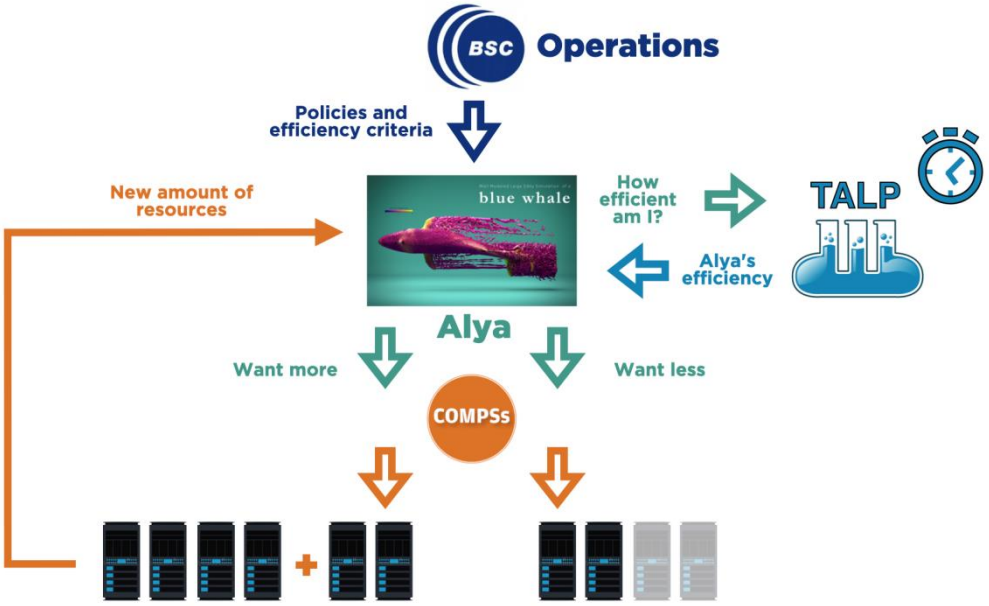
```
DLB_ARGS=" --talp [--talp-summary=pop-metrics]"
env LD_PRELOAD="$DLB_LIBS/libdlb_mpi.so" ./app
```

```
######### Monitoring Region App Summary #########
### Name:                       MPI Execution
### Elapsed Time :              2.66 s
### Parallel efficiency :       0.70
###    - Communication eff. :   0.98
###    - Load Balance :         0.72
###       - LB_in :             0.72
###       - LB_out:             1.00
```

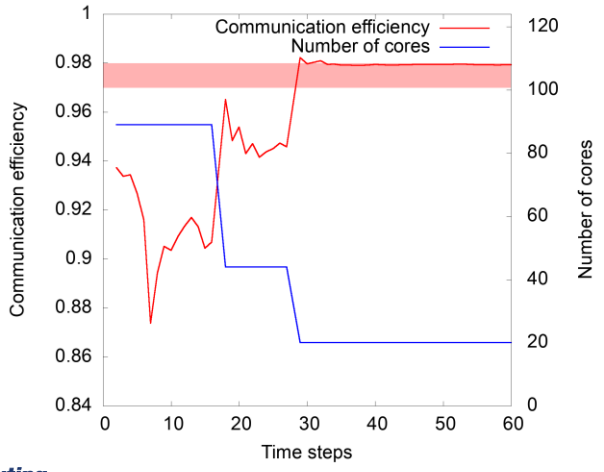
At runtime

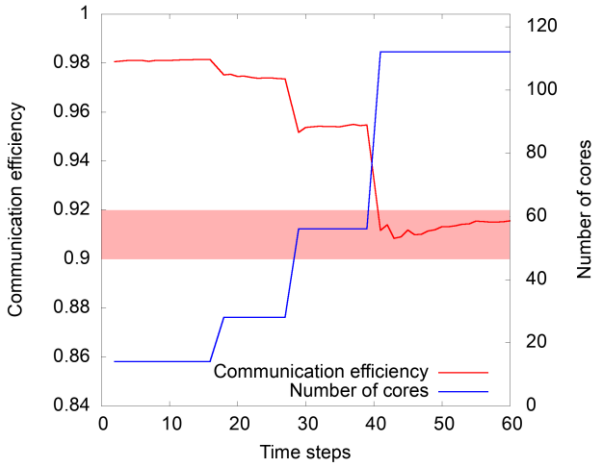
```
# include < dlb_talp .h >
...
// Register a new region or obtain an existing handler
dlb_monitor_t * monitor = DLB_MonitoringRegionRegister
("Name");
// Start TALP monitoring region
DLB_MonitoringRegionStart( monitor );
...
// Stop TALP monitoring region
DLB_MonitoringRegionStop( monitor );
...
// Print a report by standard output
DLB_MonitoringRegionReport( monitor );
...
// Manually obtain some metrics from the monitor
int64_t elapsed = monitor->elapsed_time;
int64_t elapsed_use =monitor->elapsed_computation_time;
float comm_eff = ( float ) elapsed_use / elapsed ;
```

Success story 3: Malleable simulation



- Application that adjust number of resources used based on measures by TALP





About DLB

- Current stable: Version 3.2 (2022-03-16)
  - LeWI
  - DROM
  - TALP

- Free Download under LGPL-v3 license: https://pm.bsc.es/dlb-downloads
- User Guide: https://pm.bsc.es/ftp/dlb/doc/user-guide/

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