Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

DLB: Dynamic Load Balancing Library

Marta Garcia-Gasulla
Victor Lopez, (*Xavier Teruel*)

23rd November 2022

Huawei Training – On-line

Today’s Agenda

Day 3: Wednesday | Topic: Dynamic Load Balancing (DLB) | Trainers: Marta García, Victor Lopez, and Xavier Teruel (Best Practices for Performance and Programmability Group)

- Session 1 - **Load Balance** 8:00 - 9:30 Introduction to Load Balance and Hands-on measuring LB (Xavier Teruel, Victor Lopez)
- Session 2 - **DLB: LeWI** 9:50 - 11:20 Introduction and Hands on of DLB: LeWI (Marta Garcia, Victor Lopez)
- Session 3 **- DLB: DROM and TALP** 11:50 - 14:00 Introduction and Hands on of DLB: DROM and TALP (Marta Garcia, Victor Lopez)

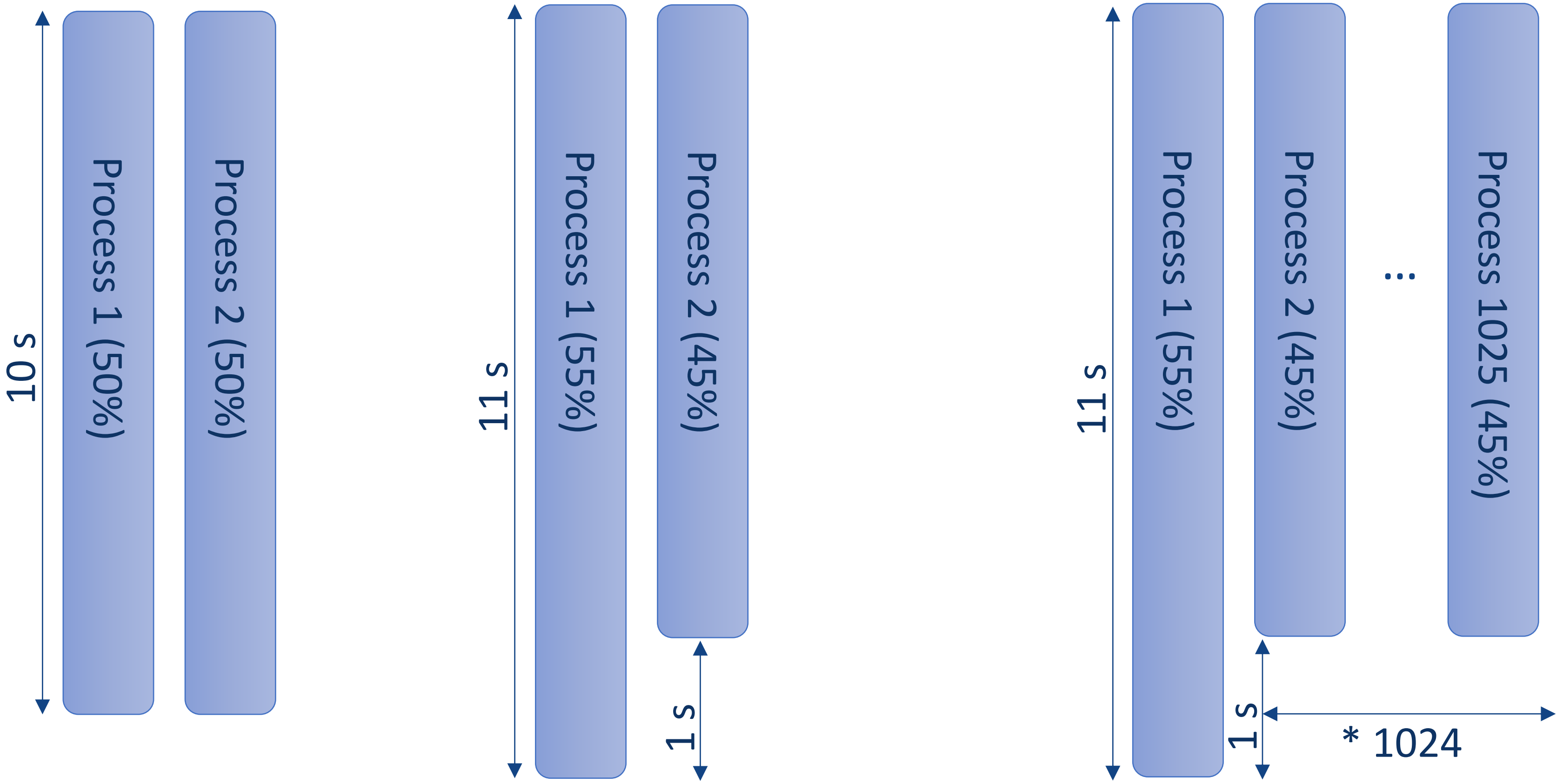
What is Load Imbalance

- Irregular distribution of load among resources.
  - Resources can be: computational, network, processing units…

- Our target: MPI load Imbalance
  - MPI is the standard de facto in HPC applications
  - MPI processes do not share data
    - Moving data around is expensive

Load Imbalance: Magnitude of the tragedy

- Ideal
- 5% difference
- 5% difference in 1024 processes



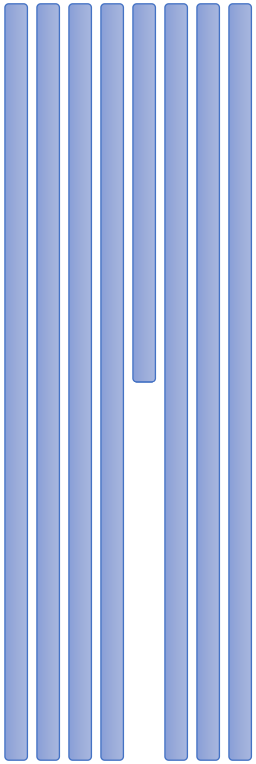
1s * 1024 CPUS = 1024 s = 17 minutes of CPU
17m * 10.000 time steps = **2.844 CPU hours**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

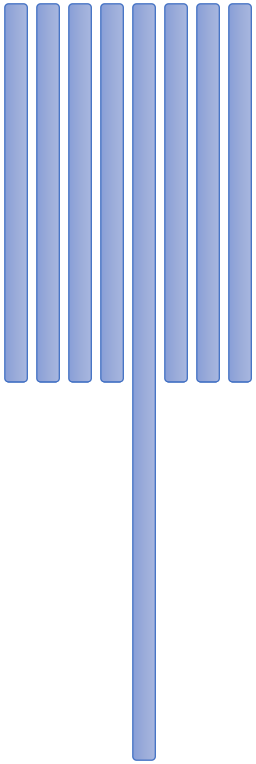
Load Imbalance: Measuring it

- Which application is more imbalanced? 🤔

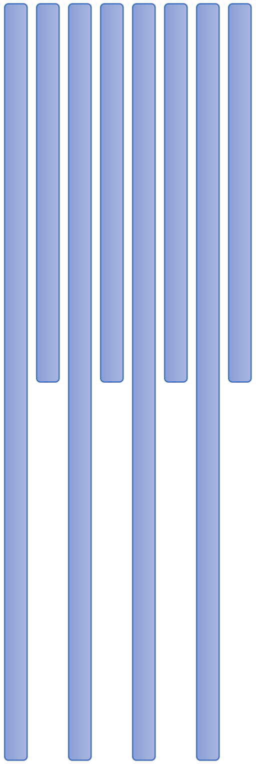
- A)



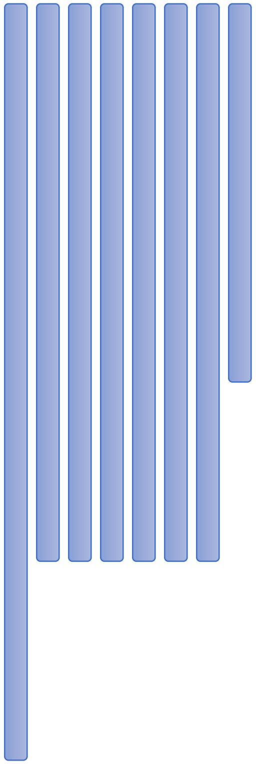
- B)



- C)



- D)



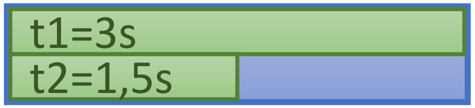
Load Imbalance: Measuring it

- ➢ Our focus is to make the most efficient use of computational resources

$$Load\ Balance = \frac{Useful\ CPU\ time}{Total\ used\ CPU\ time} =$$

$$= \frac{\sum_{n=1}^{numProcs}(t_n)}{Max_{n=1}^{numProcs}(t_n) * numProcs} = \frac{Average_{n=1}^{numProcs}(t_n)}{Max_{n=1}^{numProcs}(t_n)}$$

- $numProcs =$ number of MPI processes
- $t_n =$ execution time of process n
- $0 < LB < 1$
- $LB = 1$ → Perfect Load Balance)



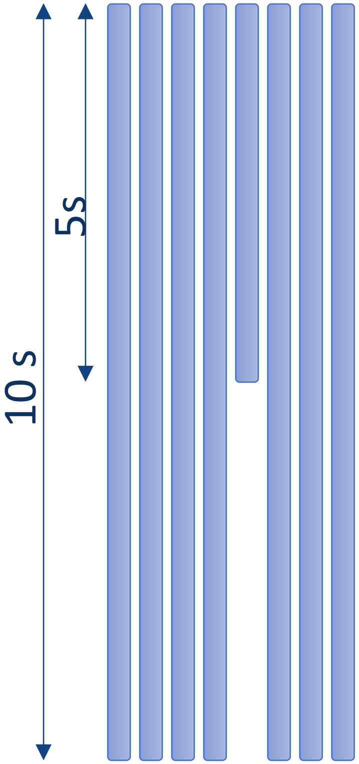
$$LB = \frac{\boxed{Useful = 3 + 1{,}5 = \mathbf{4,5}}}{\boxed{UsedCPU = 3 * 2 = \mathbf{6}}} = 0{,}75$$

Load Imbalance: Measuring it

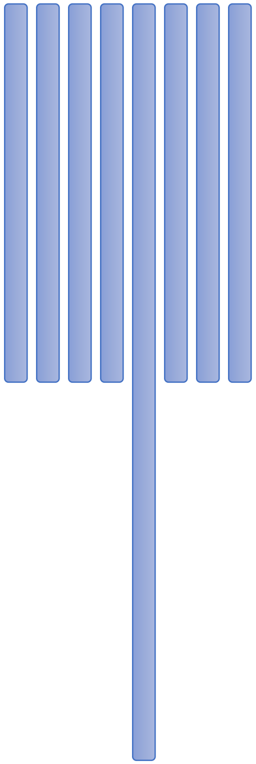
$$LB = \frac{useful\ CPU}{used\ CPU}$$

- Which application is more imbalanced?

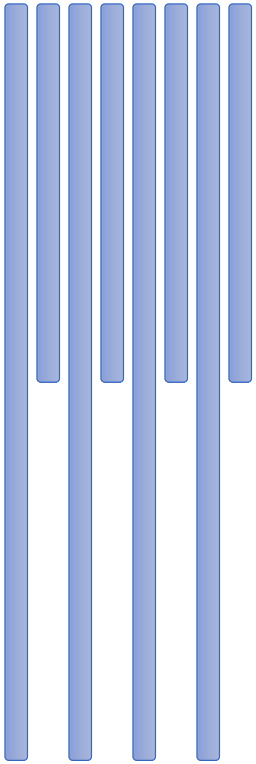
- A)



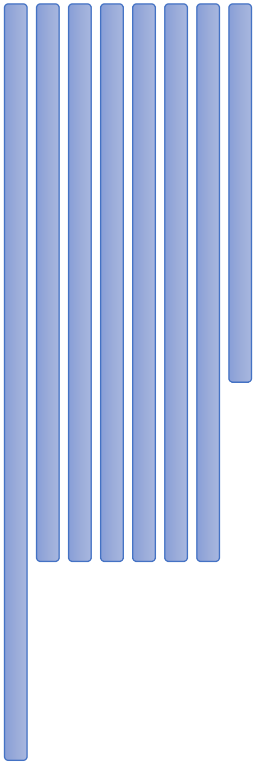
- B)



- C)



- D)



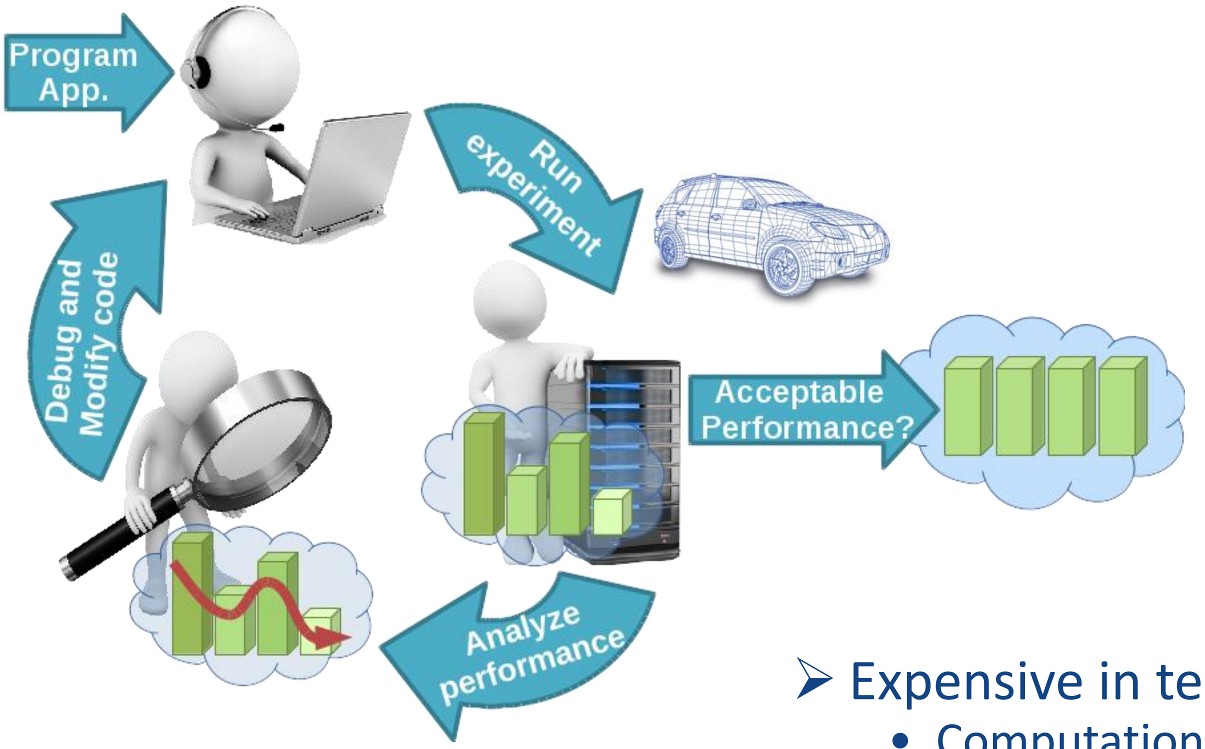
$$\frac{(7 * 10) + 5}{10 * 8} = 0{,}9375$$

$$\frac{(7 * 5) + 10}{10 * 8} = 0{,}5625$$

$$\frac{(4 * 10) + (4 * 5)}{10 * 8} = 0,75$$

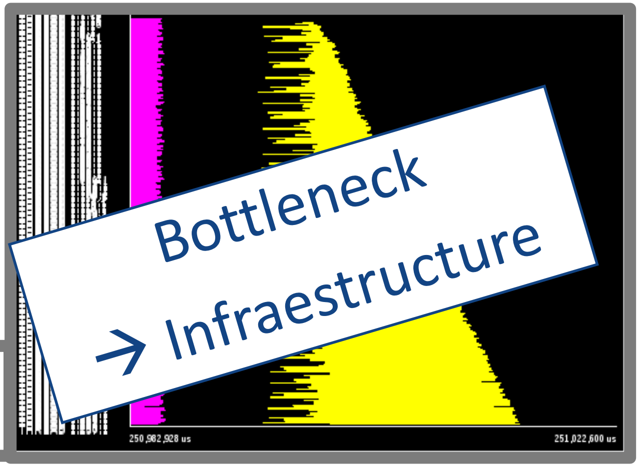
$$\frac{10 + 5 + (6 * 7{,}5)}{10 * 8} = 0{,}75$$

Load Imbalance: Solution from developers?



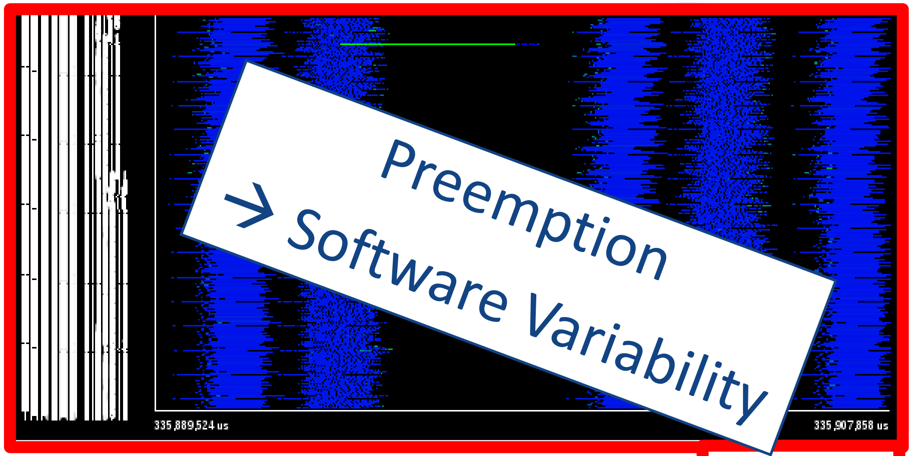
- ➢ Expensive in terms of:
  - Computational resources
  - Personal resources
- ➢ What happens if we change the input?
- ➢ And the hardware?
- ➢ Is it a real solution?

Load Imbalance: Where?



Bottleneck
→ Infraestructure

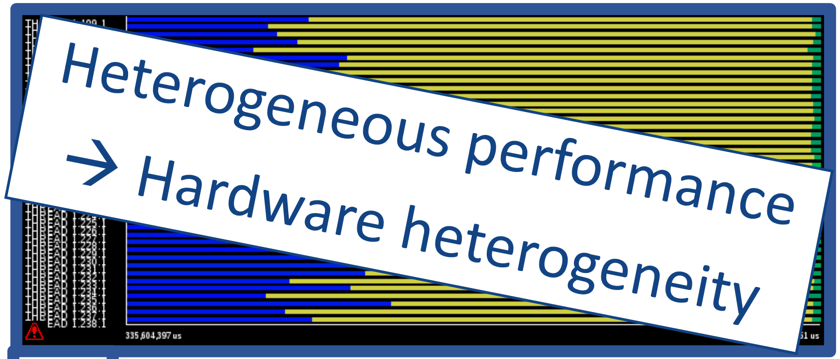
MPI calls



Preemption
→ Software Variability

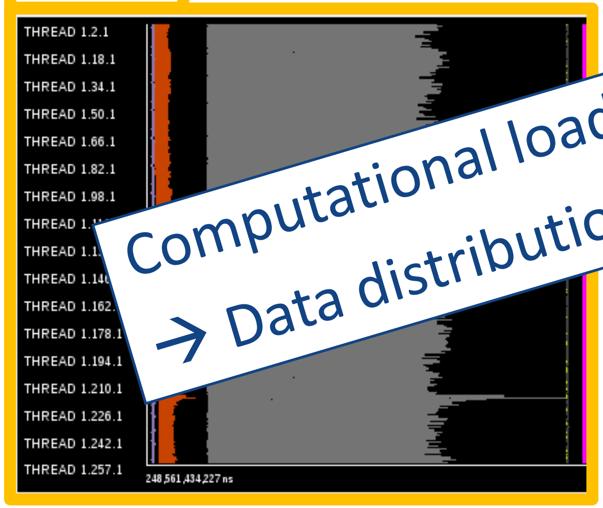
Cycles per µ second

Running function



Heterogeneous performance
→ Hardware heterogeneity

IPC



Computational load
→ Data distribution

Load Imbalance: Still searching for a solution…

- Different sources… different solutions
  - Data distribution
    - Redistribute → New Input, redistribute again?
  - Hardware heterogeneity
    - Tune specifically for architecture → New machine, tune again?
  - Infrastructure
    - Adapt code to infrastructure → New software or hardware, adapt again?
  - Software/Hardware variability
    - ???

- Our Solution: React when imbalance is happening
  - We can not fight it, lets adapt!
  - One solution to ~~rule~~ solve them all

Be water, my friend !!!

Bruce Lee

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

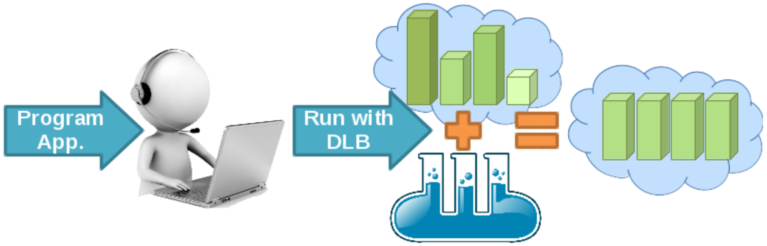
The DLB Library

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Dynamic Load Balancing - DLB

Our objectives:

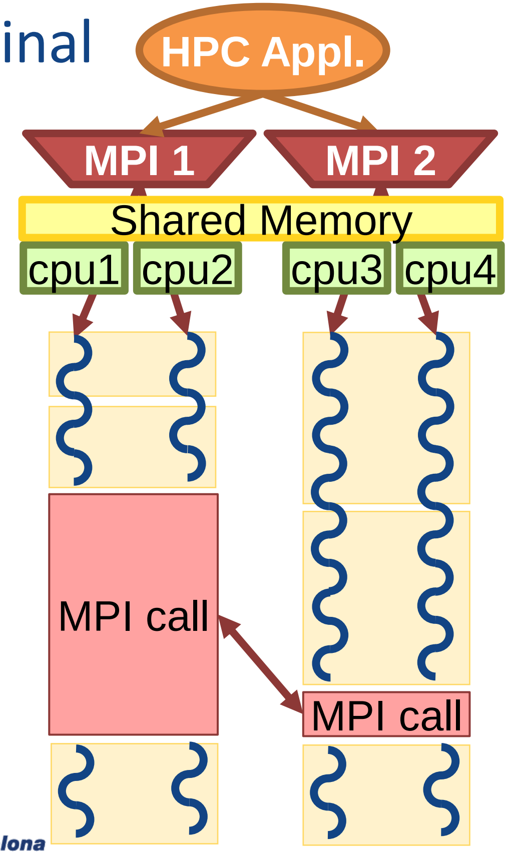
- Address all sources of imbalance
  - Fine Grain, dynamic…
  - How?
    - Detect imbalance at runtime
    - React immediately
- Real product for HPC
  - Use common programming model/environment
    - MPI + OpenMP
- Transparent to the application
  - Runtime library



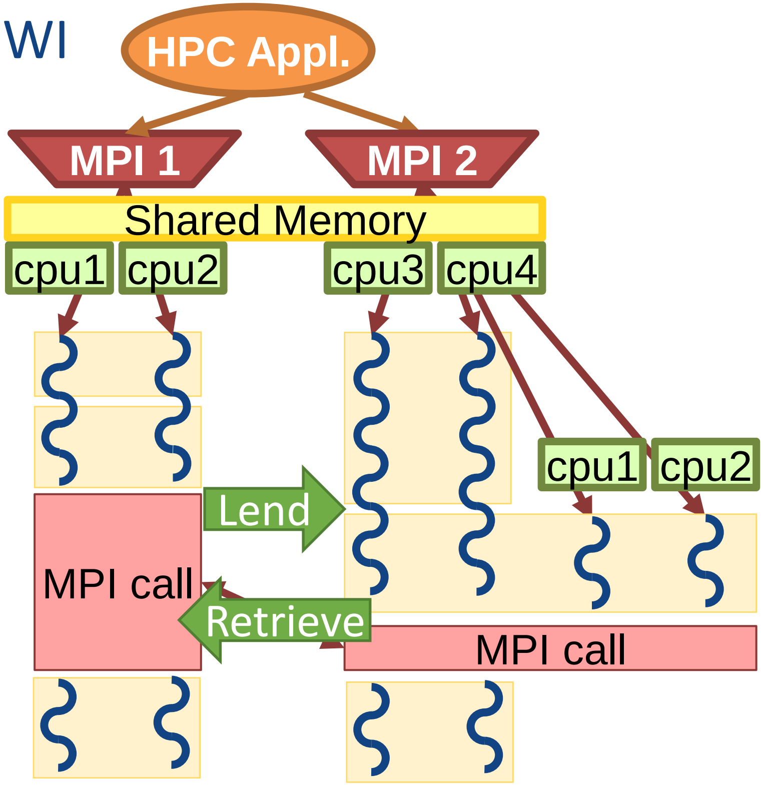
The idea: Lend When Idle (LeWI)

- Load balance MPI processes within a computational node
  - Use computational resources of a process when not using them to speed up another process in the same node

- Original

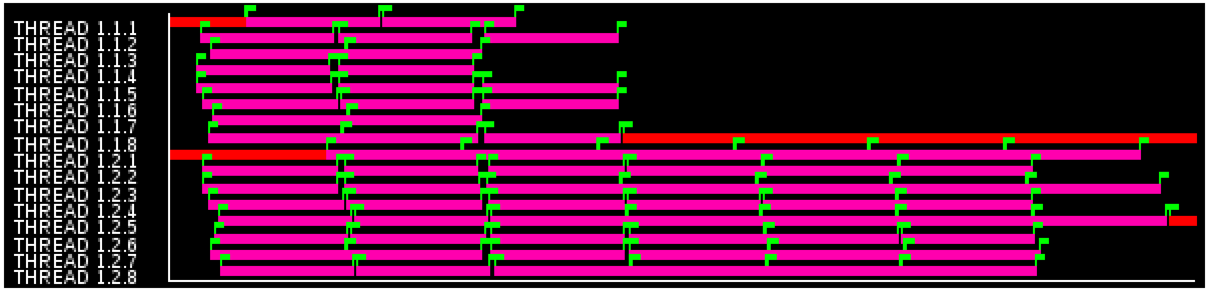


- LeWI



LeWI: A ~~image~~ trace is worth a thousand words

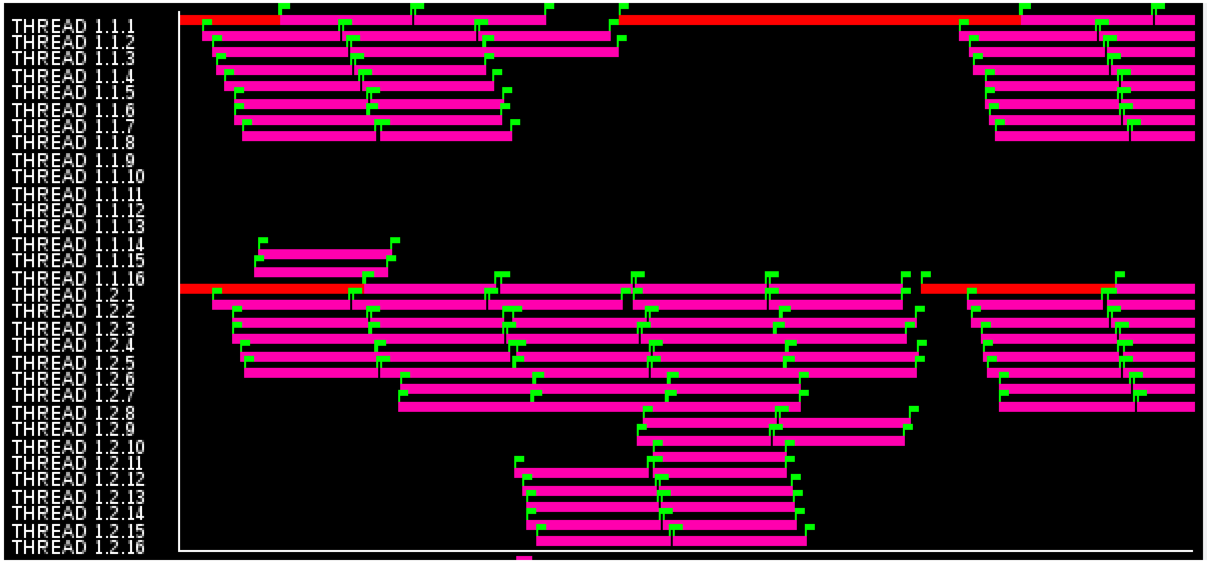
- Original:
  - 2x8



Computation

Communication

- With LeWI:
  - 2x8

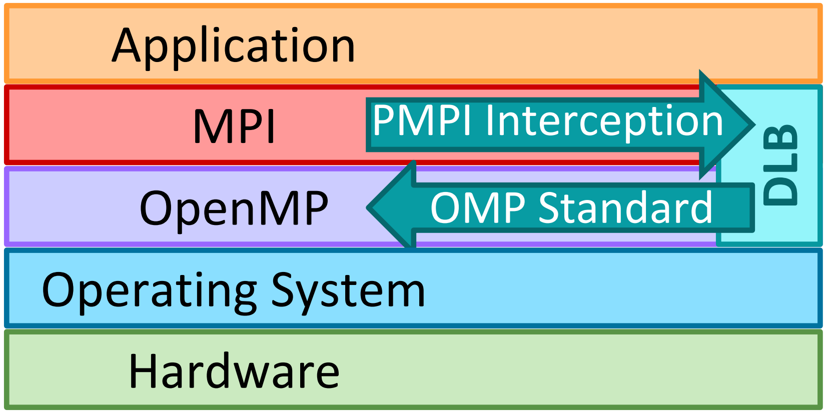


DLB: Main concepts

- **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.

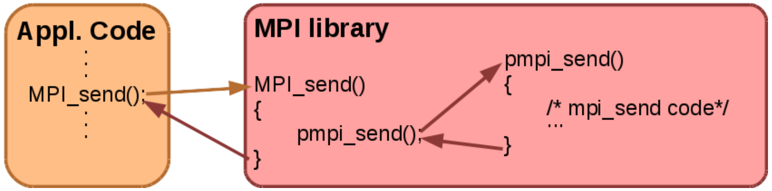
DLB: How?

- Runtime library: DLB
  - Transversal to different layers of the software stack
  - Using standard mechanisms whenever possible
    - Facilitate the adoption without modifying existing codes
  - MPI:
    - Intercept MPI calls using PMPI standard interface
  - OpenMP:
    - Use standard OpenMP API
    - omp_set_num_threads(x)

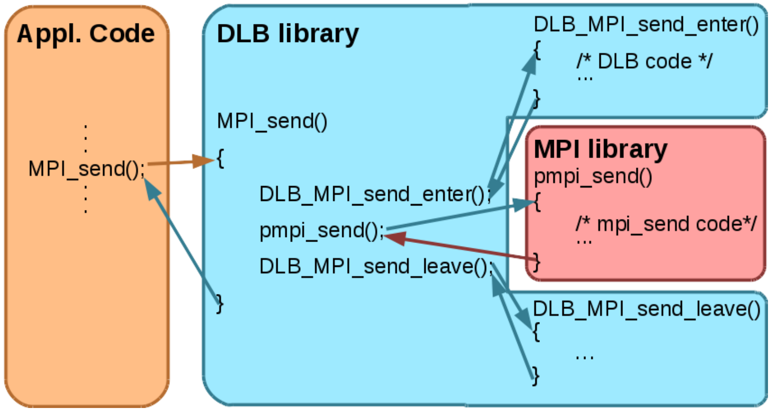


PMPI Interception

- PMPI: Profiling interface for MPI
  - MPI libraries implement an internal interface (PMPI) that implements the MPI call code

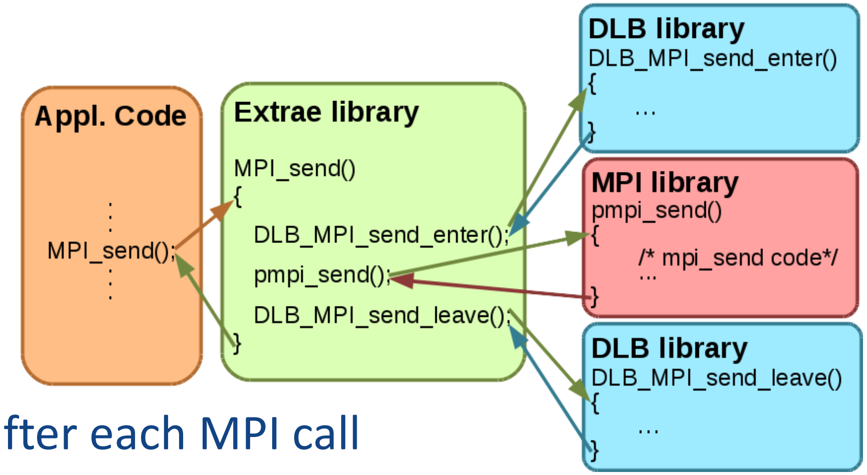


  - MPI calls can be redefined in a dynamic library
  - The intercepting library is loaded when starting the application
    - `export LD_PRELOAD = libdlb_mpi.so`
    - The dynamically loaded library has preference
  - Within the intercepted call the corresponding PMPI function must be called



PMPI Interception

- Using DLB and Extrae
  - Both use PMPI interface



- Integration:
  - Extrae intercepts MPI calls with PMPI
  - DLB API called from Extrae before and after each MPI call
  - DLB does not intercept MPI calls
    - `export LD_PRELOAD = libdlb_mpi_instr.so`

- And other profiling tools using PMPI?
  - We are studding using PnMPI
    - Allows n tools intercepting MPI
    - An order between them must be selected
    - All the tools must support PnMPI
    - So far no conflicts have been found… Future Work

MPI blocking mode

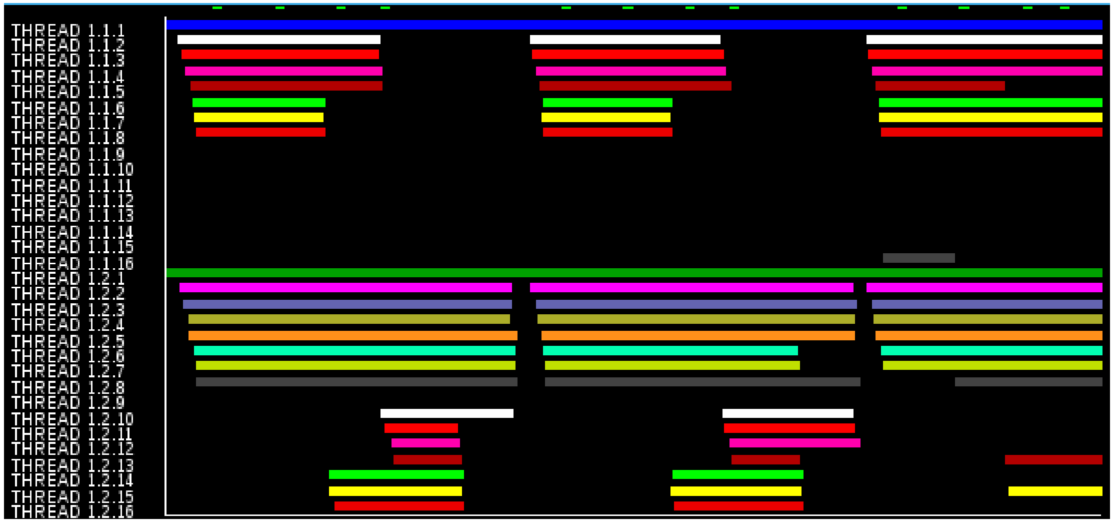
- MPI is greedy in the use of CPU
  - By default it will busy wait for messages/synchronizations to arrive
  - If the CPU is used by the MPI process waiting for the message we can not use it for doing useful computation by another thread.

- Different behavior for different MPI libraries 😱

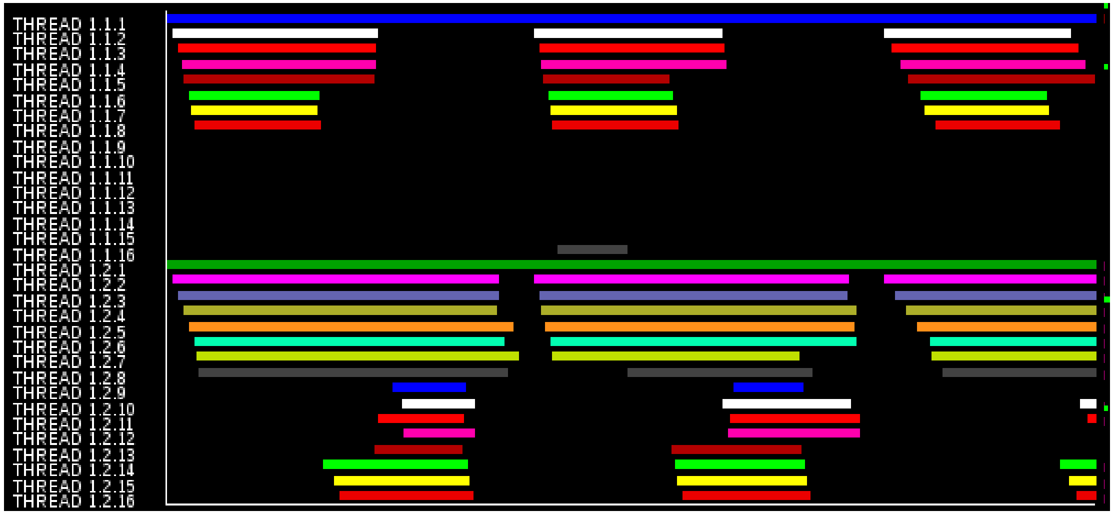
- We have two options:
  - Leave all the CPUs assigned to a process but one
    - `export DLB_ARGS += “--lewi-mpi=no”`
  - Tell MPI not to busy wait
    - `export I_MPI_WAIT_MODE=1`
    - `export DLB_ARGS += “--lewi-mpi”`

MPI blocking mode

- --lewi-mpi =no



- --lewi-mpi



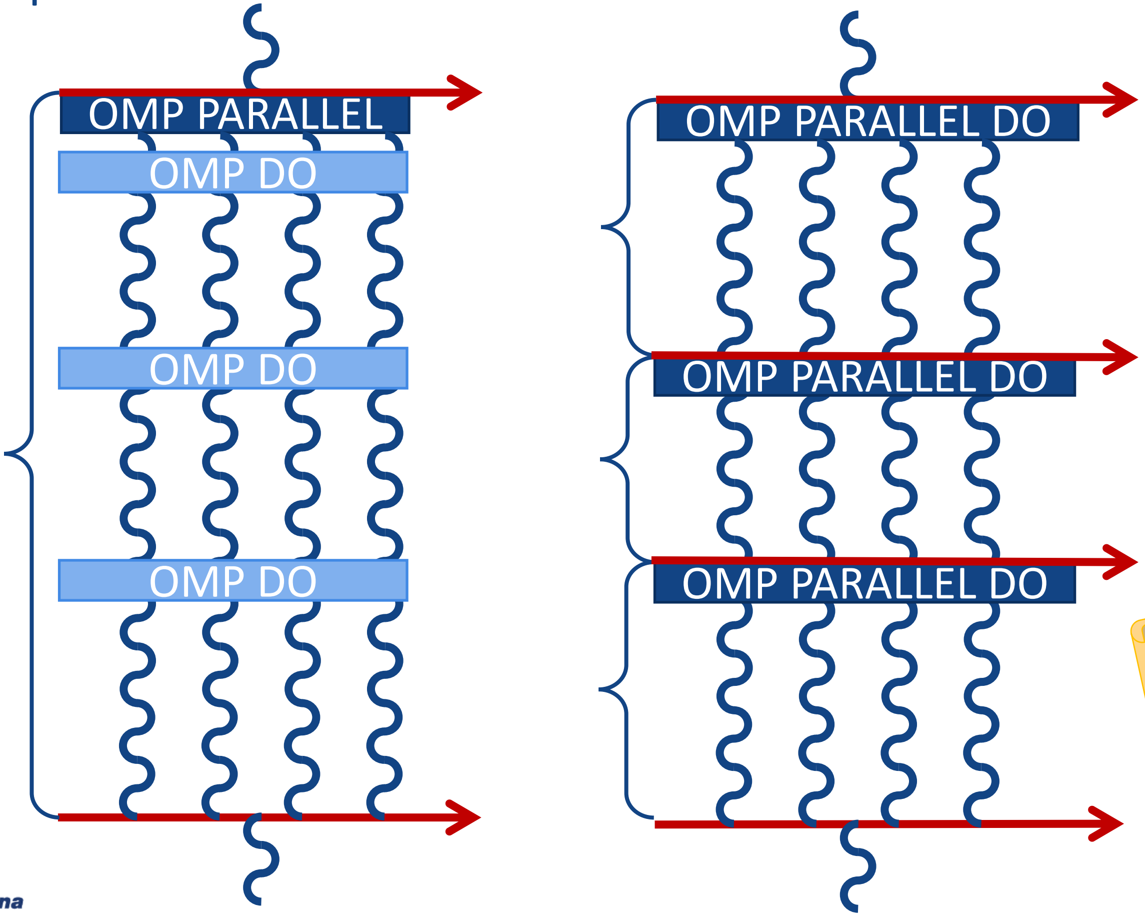
OpenMP: Malleability

- OpenMP is malleable, we can change number of threads
  - `omp_set_num_threads(int x)`
  - But only outside a parallel region

- But some programming practices can avoid malleability: 👎
  - Program in function of the thread Id
    - `omp_get_thread_num(int x)`
    - Fear if you see this call!
  - Do reductions “by hand”
    - Allocate memory in function of the number of threads and each one will reduce in its piece of data.
  - Avoid these practices please!

OpenMP: Malleability

- Use omp_set_num_threads(x)
  - It can only be called outside a parallel region (says the OpenMP standard)
  - Impact in DLB…



Mith#1: The overhead of opening/closing parallelism

OpenMP in DLB

- Add a call to `int DLB_Borrow(void)` before each `parallel`
- `int DLB_Borrow(void)` will check the system for idle CPUs and update the number of threads in case necessary

```
DLB_Borrow();
#pragma omp parallel do
for (i=0; i<n; i++){
    compute…
    …
}
```

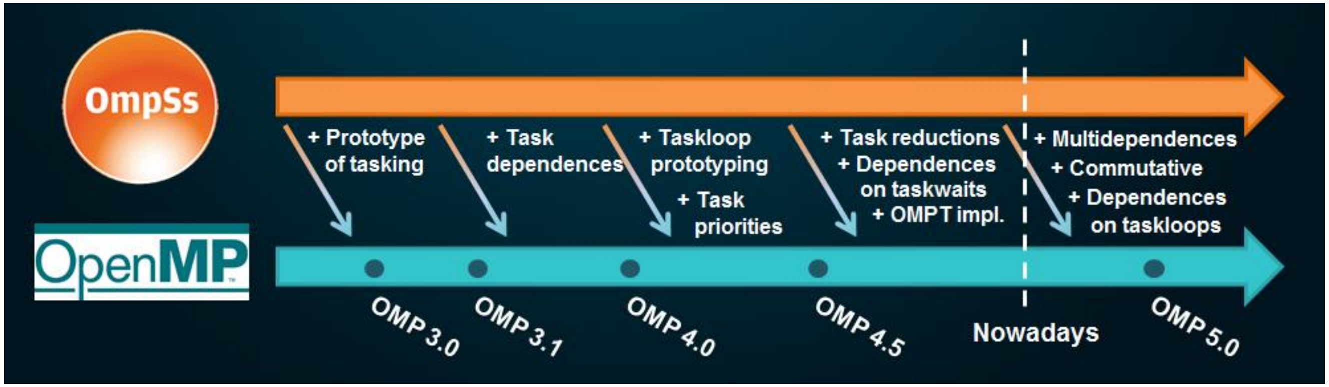
```
int DLB_Borrow(void){
    check_idle_cpus(x);
   set_omp_num_threads(x);
}
```

- This can be done by an automatic replacement in the code
- Latest news!
  - Working in using OMPT (tracing tool for OpenMP to appear in 5.0)
- Meanwhile…

Integration with Nanos++

- Nanos++: Parallel Runtime developed at BSC
  - Implements OpenMP 4.5 and OmpSs
  - Forerunner for OpenMP

- Mercurium: Source to source compiler developed at BSC
  - Generates code for Nanos++



Integration with Nanos++

- There is no need to modify the application at all
  - The runtime will call the DLB API where necessary to ask for resources or return them

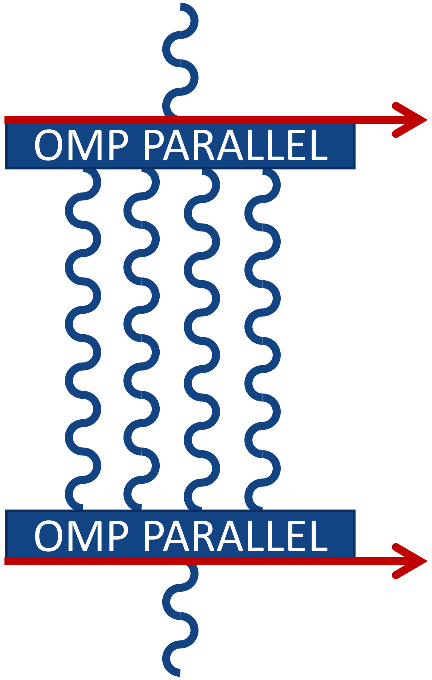
- Compile with Mercurium

- Run enabling DLB
  - Mandatory: `NX_ARGS+= “--enable-dlb --enable-block”`
  - Recommended: `NX_ARGS+= “--force-tie-master”`
  - In some cases: `NX_ARGS+= “--warmup-threads”`

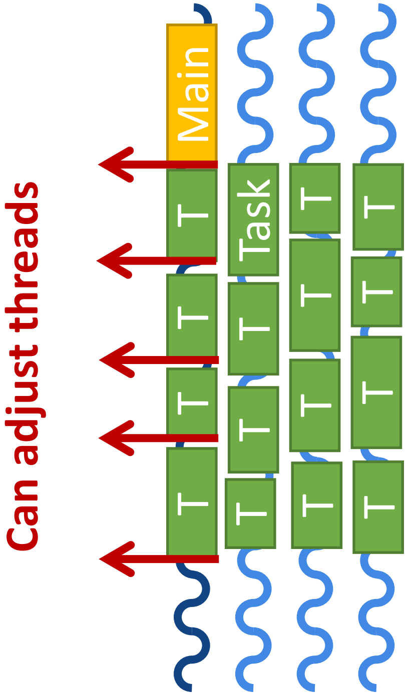
- Win! 💪

More malleability with OmpSs

- OpenMP (Fork-join model)

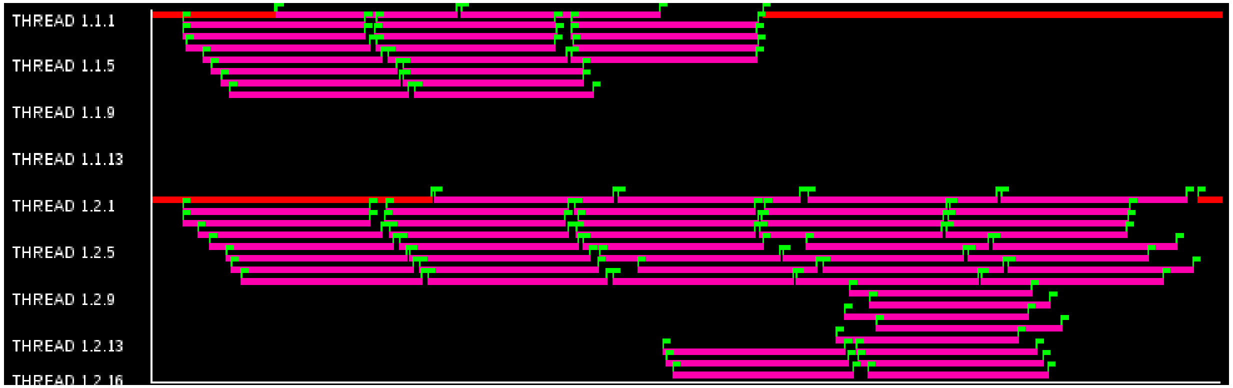


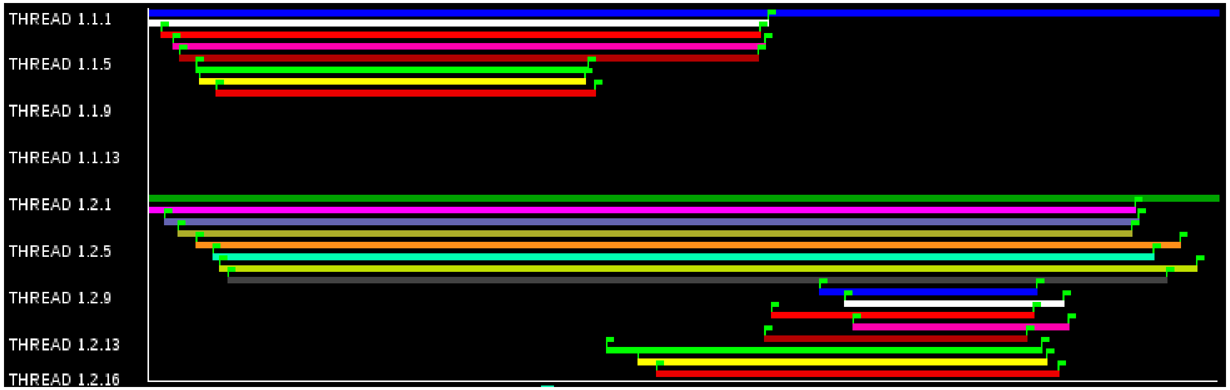
- OmpSs(Task based)



Integration with Nanos++

- Taking advantage of the integration and increased OmpSs malleability
  - Threads are autonomous
    - Fast response
    - The master thread is not a bottleneck
    - Benefit from imbalances at OmpSs level too





Summing up to use DLB…

- `export LD_PRELOAD = libdlb_mpi.so`

- If we want to use the CPU executing the MPI calls
  - `export I_MPI_WAIT_MODE=1`
  - `export DLB_ARGS += "--lend_mode=block"`
- else
  - `export DLB_ARGS += "--lend_mode=1CPU"`

- If we use Nanos++
  - `DLB_ARGS+= "--policy=auto_LeWI_mask"`
  - `NX_ARGS+= "--thread_manager=dlb"`
  - `NX_ARGS+= "--force-tie-master --warmup-threads"`
- else
  - Add `dlb_update_resources()` before each `#pragma omp parallel`
  - `DLB_ARGS+= "--policy=LeWI"`

Multiple Applications

- We can share CPUs between different applications running in the same node

- Do not need MPI

- Transparent to the user, works out of the box

DROM

Dynamic Resource Ownership Management

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

DROM: Dynamic Resource Ownership Management

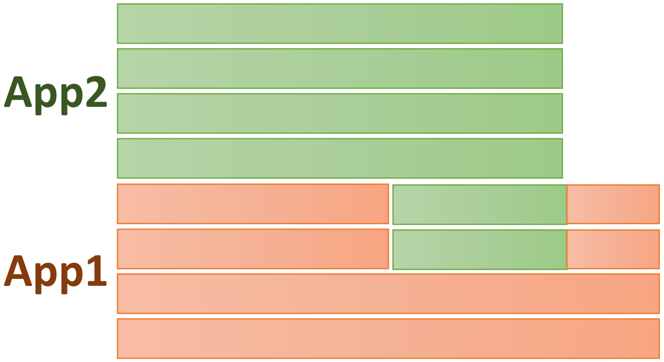
- API for superior entity
  - Job Scheduler
  - Resource manager
  - User

- Allow to change the assigned resources (CPUs) to a process

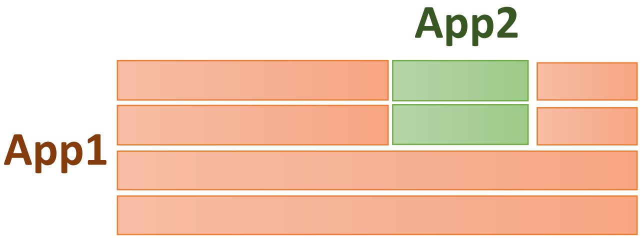
- Some possible use cases:
  - A) User wants to give more priority to one of the processes in the node
  - B) Job scheduler wants to start a high priority app. using the resources allocated for an other application
  - C) Application is not using the resources in a node efficiently (i.e the bottleneck is on another node) can free them to avoid accounting.

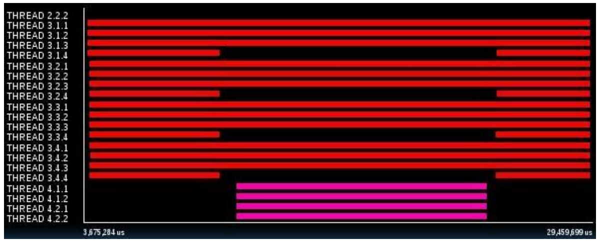
DROM: Use cases

- **A) User:**
  Increase priority to App2

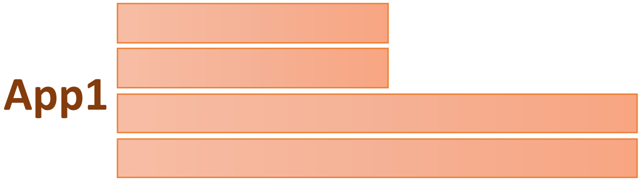


- **B) Job Scheduler:**
  Run High priority App2 in resources assigned to App1

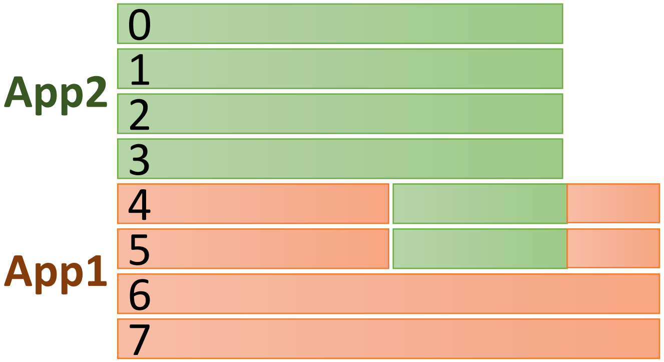




- **C) App1:** Release 2 CPUs because not using efficiently

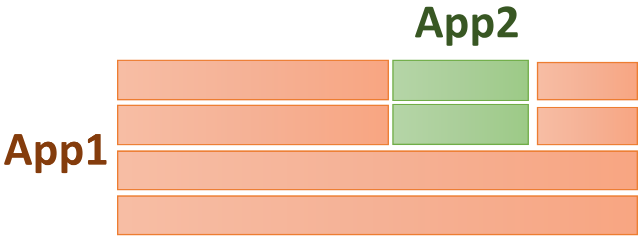


DROM: How to



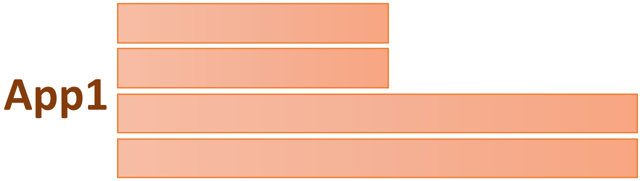
- A)

```
$> 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]);
```

About DLB

- Current stable version 1.3.1 (October 2017)

- New release 2.0 coming up for Christmas 2017 🎉
  - DROM
  - New asynchronous API
  - OMPT support

Work in Progress

- DROM
  - Implemented, evaluate performance
- OMPT
  - Enable use for any OpenMP runtime supporting OMPT (OpenMP 5.0)
  - Not “legal” according to the standard 🙊
- Study performance in many-core
  - i.e. Intel Xeon Phi KNL 256 threads
- Runtime Monitoring Tool
  - Monitor different levels and collect metrics
  - Offer an API to consult metrics during execution
- Load Balancing across containers
  - Studding feasibility, performance, issues and opportunities
  - Docker, Singularity…

Challenges

- Transversal to different layers, make the cooperate!!
  - MPI libraries are not willing to expose the non busy wait mode
    - They want all CPU cycles for them, but they are wasting them…
  - OS could help handling the cores? Giving priorities?

- Change mentality from “heroism programming” to trusting the runtime
  - Applications should stop doing things “by hand”
  - Let’s help them:
    - By addressing their needs and offering non intrusive solutions
    - By offering transversal solutions

- Malleability, malleability everywhere!!!
  - Application, Programming model, job scheduler…

FAQ

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

FAQ

- Why not “learn” and use previous redistribution?
- What about data locality?
- My application does not perform well with OpenMP
- What about load balance between nodes?
- Why not overload CPUS, it’s the same you do!
- How do you decide to which process CPUS go?
- I already have a load balancing algorithm within my application
- How do I know the different options in DLB?

Why not “learn” and use previous redistribution?

- There is a policy in DLB that does a “static” distribution of CPUs based in the load of each process
  - `--policy=WEIGHT`
    - Detects iterations, based in the MPI calls pattern
    - Computes an optimum distribution of CPUs
    - Applies it
  - Performance was much worse than LeWI → LeWI is more flexible
  - Code is deprecated

- Another policy that merge the functionality of WEIGHT and LeWI was implemented (Redistribute and Lend)
  - `--policy=RaL`
  - Performance was equal to the one obtained by LeWI

- We can recover these if we find the need

How do you decide to which process CPUS go?

- We do not decide it, it is first come, first served
- So far, our experience is: If there is a free CPU and some one willing to use it, do it.
- But… we might implement some accounting in the future if more actors come in… different apps, different users, different programming models…
- We DO decide which CPU to take first…

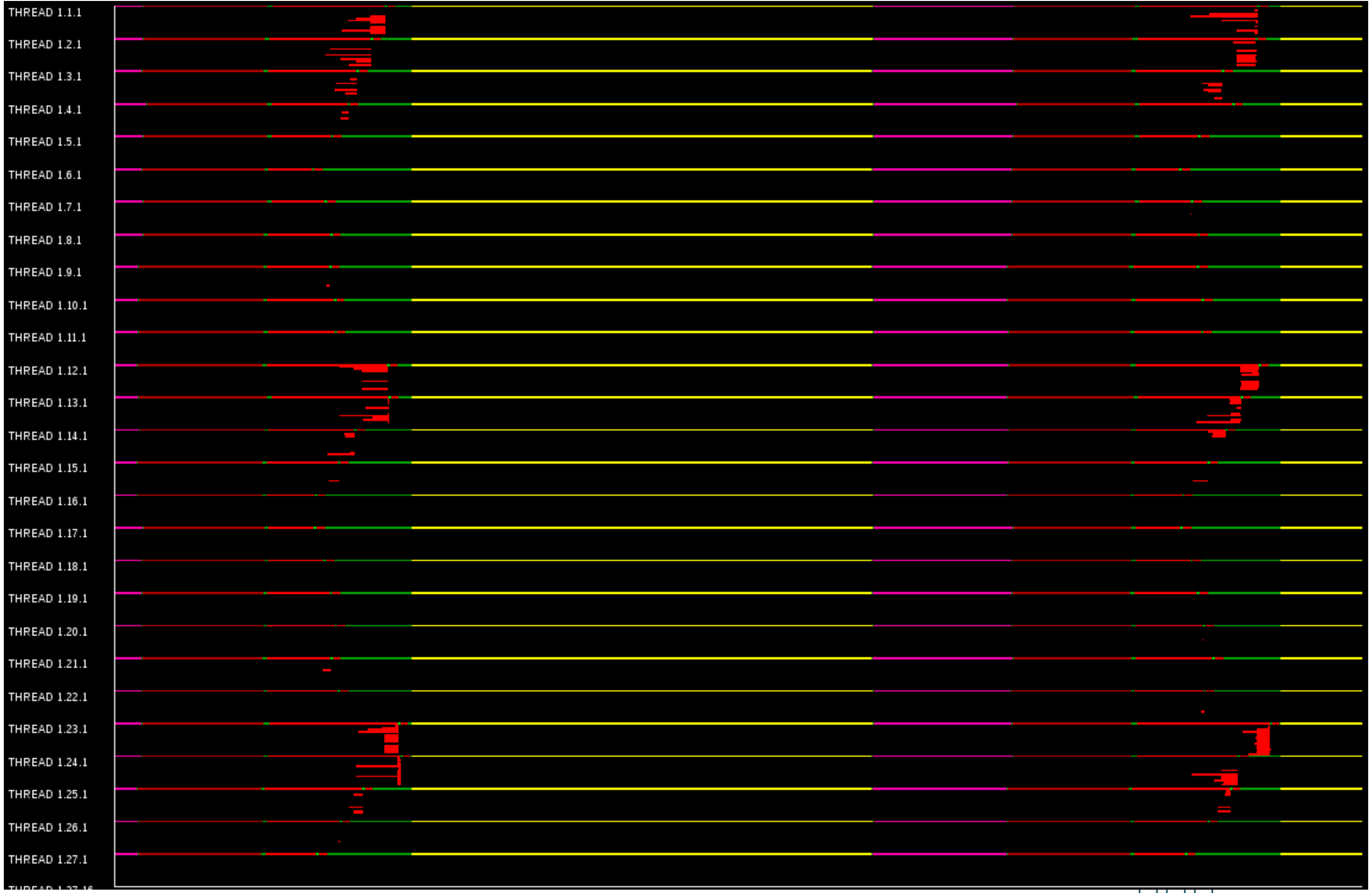
What about data locality?

- In some kernels spawning threads to another socket can have a penalty

- We can choose with flag `--priority` in DLB_ARGS environment variable which CPU a process will acquire **first** when asking for resources
  - none: Take the first free CPU, does not take into account topology
  - **affinity_first**: Take first CPUs that are "affine" to me, and then the others
  - affinity_full: Take first CPUs that are affine to me, take CPUs from another socket only if all the CPUs in that socket are free (meaning no body is running there)
  - affinity_only: Take only CPUs that are affine to me

My application does not perform well/it is not parallelized with OpenMP

- Don’t worry!
- In fact usually it is the best configuration… gives more flexibility to DLB

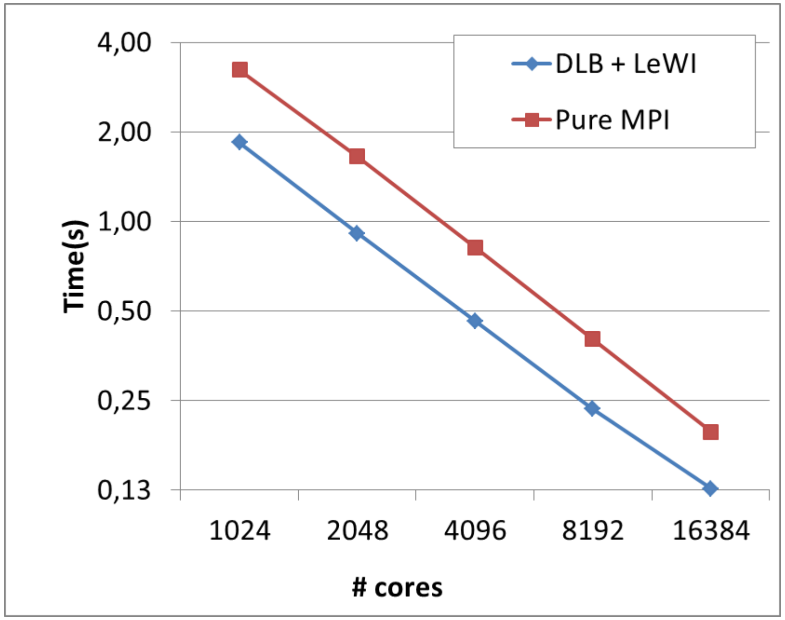


What about load balance between nodes?

- We do not have any solution for this yet

- It is a quite different problem
  - Big difference in granularity, moving data between nodes is expensive

- But… good news is…
  - We are achieving very good results by balancing inside the node even when running up to 1024 nodes

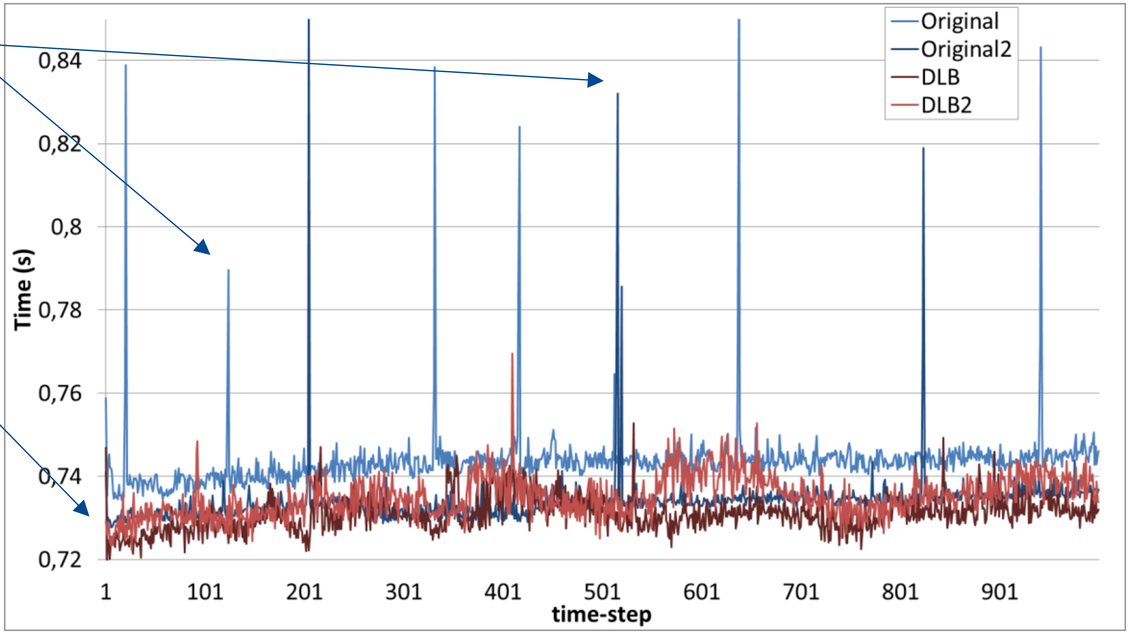


I already have a load balancing algorithm within my application

- Does it solve this?

- Fine grain + system noise
- Blue lines original application (2 different runs)
- Red lines same run with DLB (2 different runs)

Clearly visible spikes without DLB are absorbed by DLB



How do I know the different options in DLB?

- ➢ `[DLB_HOME]/bin/dlb --help`

```
The library configuration can be set using arguments
added to the DLB_ARGS environment variable.
All possible options are listed below:

--policy:               None                    [no, JustProf, LeWI, Map, WEIGHT, LeWI_mask,
auto_LeWI_mask, RaL]
--statistics:           no                      (bool)
--drom:                 no                      (bool)
--barrier:              no                      (bool)
--just-barrier:         no                      (bool)
--lend-mode:            1CPU                    [1CPU, BLOCK]
--verbose:
{api:microlb:shmem:mpi_api:mpi_intercept:stats:drom}
--verbose-format:       node:pid:thread         {node:pid:mpinode:mpirank:thread}
--trace-enabled:        yes                     (bool)
--trace-counters:       yes                     (bool)
--mask:                                         (string)
--greedy:               no                      (bool)
--shm-key:              7725                    (string)
--bind:                 no                      (bool)
--aggressive-init:      no                      (bool)
--priority:             affinity_first          [none, affinity_first, affinity_full,
affinity_only]
--debug-opts:                                   {register-signals:return-stolen}
```

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

DLB Hands-on

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Evironment (Nord3)

- 84 compute nodes

- Each node:
  - 2x E5–2670 SandyBridge-EP 2.6GHz cache 20MB 8-core
  - 16 cores per node divided into two sockets

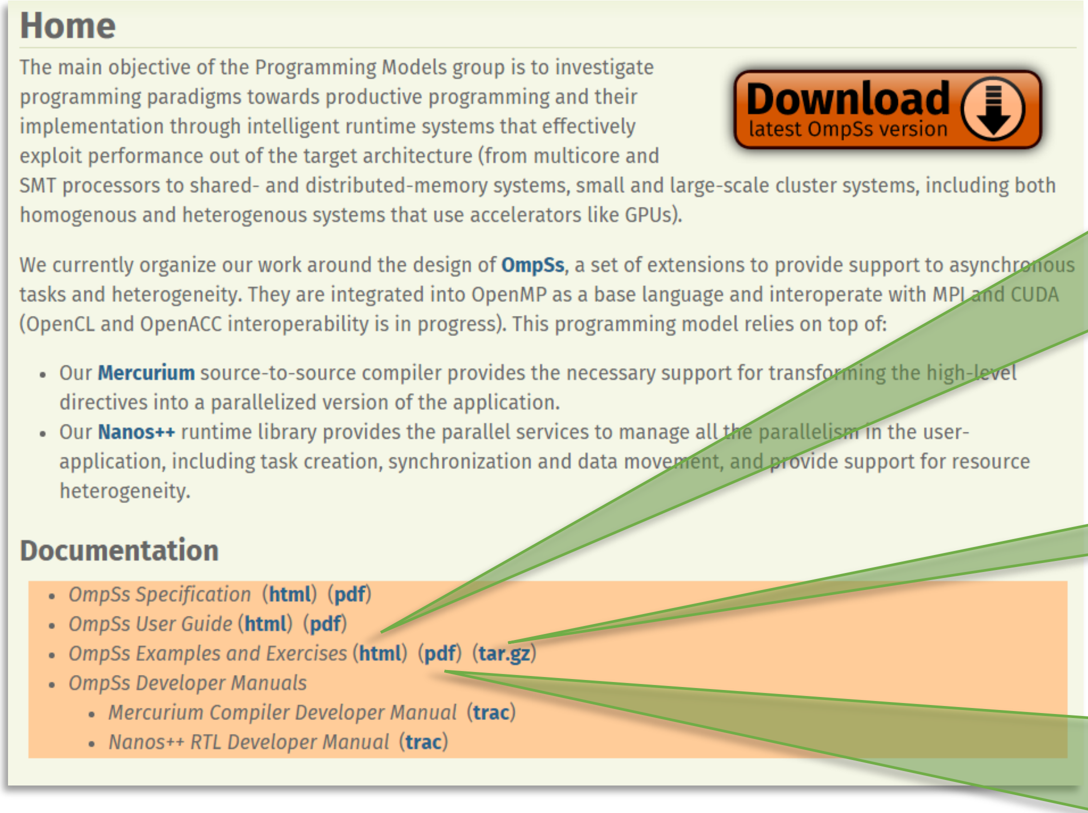
Account and Login Information

- Username and password
  - Username: nct010<your_id_here>
  - Password: OmpSsDLB.0<your_id_here>

- Example: for identifier 07, account information would be:
  - Username: nct01007
  - Password: OmpSsDLB.007

- Login in nord3:
  - ssh nct01007@nord1.bsc.es -x

Getting the examples package



Home

The main objective of the Programming Models group is to investigate programming paradigms towards productive programming and their implementation through intelligent runtime systems that effectively exploit performance out of the target architecture (from multicore and SMT processors to shared- and distributed-memory systems, small and large-scale cluster systems, including both homogenous and heterogenous systems that use accelerators like GPUs).

We currently organize our work around the design of **OmpSs**, a set of extensions to provide support to asynchronous tasks and heterogeneity. They are integrated into OpenMP as a base language and interoperate with MPI and CUDA (OpenCL and OpenACC interoperability is in progress). This programming model relies on top of:

- Our **Mercurium** source-to-source compiler provides the necessary support for transforming the high-level directives into a parallelized version of the application.
- Our **Nanos++** runtime library provides the parallel services to manage all the parallelism in the user-application, including task creation, synchronization and data movement, and provide support for resource heterogeneity.

Documentation

- *OmpSs Specification* (**html**) (**pdf**)
- *OmpSs User Guide* (**html**) (**pdf**)
- *OmpSs Examples and Exercises* (**html**) (**pdf**) (**tar.gz**)
- *OmpSs Developer Manuals*
  - *Mercurium Compiler Developer Manual* (**trac**)
  - *Nanos++ RTL Developer Manual* (**trac**)

pm.bsc.es/ompss-docs/examples/README.html

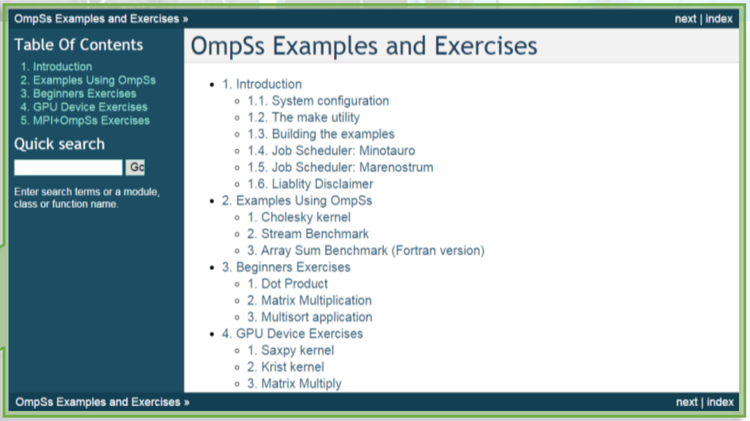


Table Of Contents

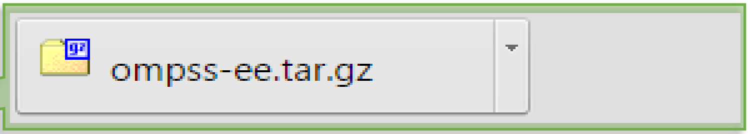
1. Introduction
2. Examples Using OmpSs
3. Beginners Exercises
4. GPU Device Exercises
5. MPI+OmpSs Exercises

Enter search terms or a module, class or function name.

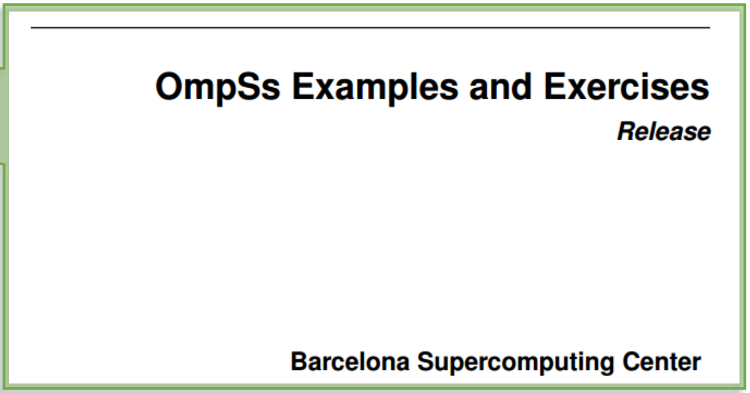
OmpSs Examples and Exercises

- 1. Introduction
  - 1.1. System configuration
  - 1.2. The make utility
  - 1.3. Building the examples
  - 1.4. Job Scheduler: Minotauro
  - 1.5. Job Scheduler: Marenostrum
  - 1.6. Liablity Disclaimer
- 2. Examples Using OmpSs
  - 1. Cholesky kernel
  - 2. Stream Benchmark
  - 3. Array Sum Benchmark (Fortran version)
- 3. Beginners Exercises
  - 1. Dot Product
  - 2. Matrix Multiplication
  - 3. Multisort application
- 4. GPU Device Exercises
  - 1. Saxpy kernel
  - 2. Krist kernel
  - 3. Matrix Multiply

pm.bsc.es/ompss-docs/examples/ompss-ee.tar.gz



pm.bsc.es/ompss-docs/examples/OmpSsExamples.pdf



OmpSs Examples and Exercises

Release

Barcelona Supercomputing Center

- Exercise scripts in *.html* and *.pdf* formats
- A single package including all source files
- Simple to configure, compile and execute

Starting the hands-on

```
$> cp /apps/PM/ompss-ee.tar.gz .

$> tar -xzf ompss-ee.tar.gz

$> cd ompss-ee

$> source configure.sh

$> cd 05-ompss+dlb
```

05-ompss+dlb

- Subfolders include different benchmarks and examples
  - `pils`: (Parallel ImbaLance Simulator) Synthetic benchmark to simulate different imbalance patterns
  - `lulesh`: Benchmark from LLNL, represents a typical hydrocode, like ALE3D
  - `lub`: LU matrix decomposition by blocks
  - `pils-multiapp`: Example for a multi application situation

Inside each folder…

➢ To build:

- `$> make`

➢ We can see…

- `[app]-p` → Binary for performance
- `[app]-i` → Binary for tracing
- `[app]-d` → Binary for debugging

- `run_once.sh` → For running/obtaining trace if one run
- `trace.sh` → Auxiliary script for tracing
- `multi_run.sh` → To run several executions and compare execution time