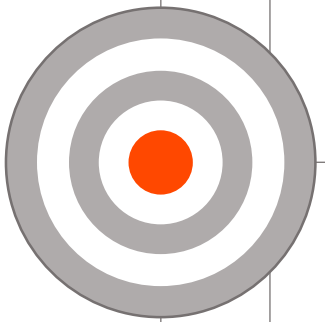
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Computer Sciences Department: strategic developments

November 2020

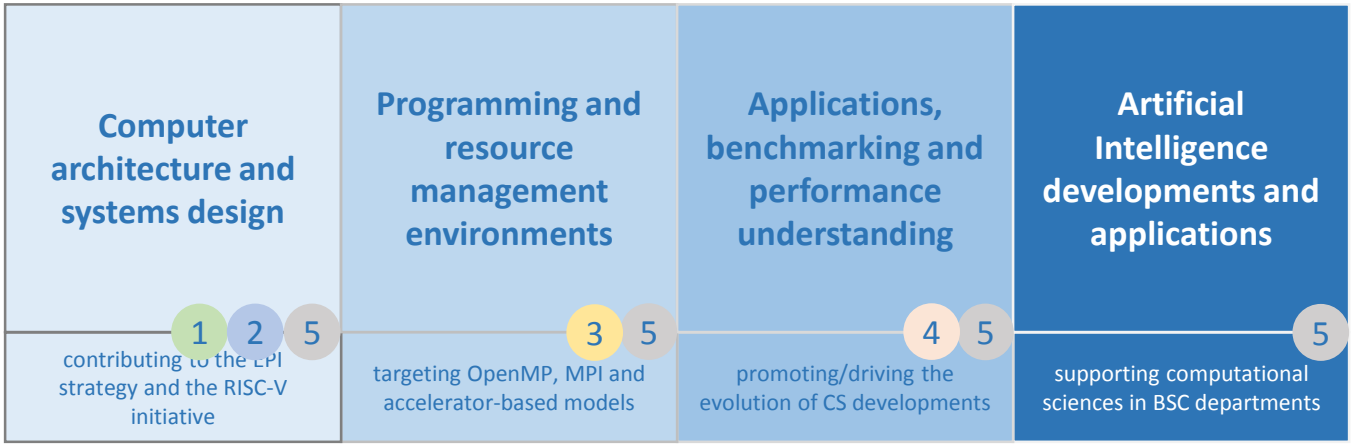
Supercomputing SC'20

Department objectives

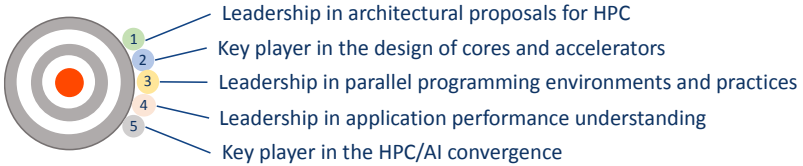


1. **Leadership in architectural proposals for HPC**
   - Runtime-aware architecture innovations
   - Multicore and accelerator architectures in real-time environments
2. **Key player in the design of cores and accelerators**
   - European Processor Initiative
   - New team with solid knowledge on low-level system design
3. **Leadership in the evolution and standardization of parallel programming environments and practices**
   - Inter-operability and resource malleability across different runtime layers
4. **Leadership in methodologies and tools for application/system behaviour understanding**
   - Proof-of-concept and best practices towards improving applications scalability and performance
5. **Key player in the HPC/AI convergence**
   - Novel algorithms, system software and architectural support
   - Adoption of AI technologies in future simulation frameworks

Main research lines



Contributing to Department's objectives:



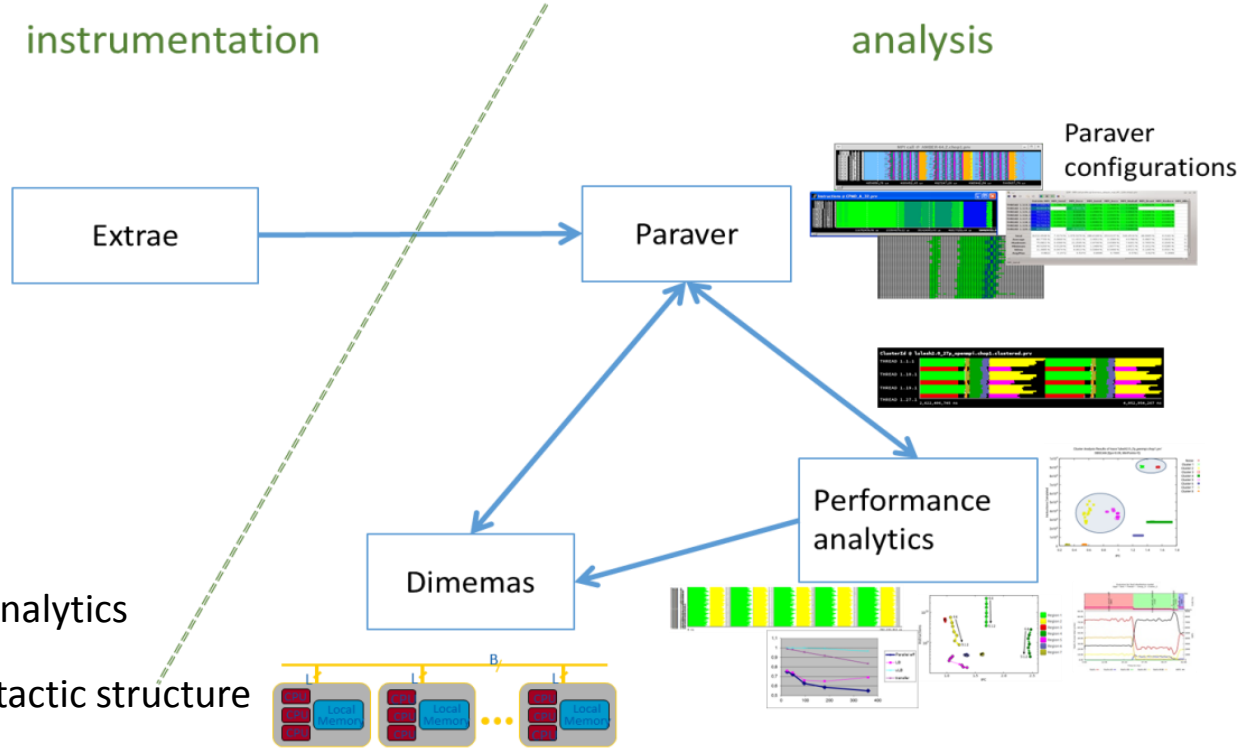
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Performance tools and methodologies

For further information please visit
http://tools.bsc.es and https://www.pop-coe.eu

BSC Performance Tools

- Since 1991
- Based on traces
- Open source
- Focus
  - Detail, variability, flexibility
  - Key factors
  - Visual analysis
  - Intelligence: Performance Analytics
  - Behavioral structure vs. syntactic structure



BSC Tools – what's new?

- Extrae extensions with updated support to OMPT and GASPI instrumentation
- Extrae prototypes supporting OpenACC and extending Burst mode to OpenMP
- Paraver easy-to-use features (extended hints, sessions management, automatic tutorials download)
- Improvements in Paraver timelines (colors management, what-where function line)
- Improvements in Paraver tables (columns ordering, independent object selection)
- Robust Basic Analysis module with support to MPI+X hybrid codes, burst mode and I/O efficiencies

POP Center of Excellence

- A **Centre of Excellence**
  - On **Performance Optimisation and Productivity**
  - Promoting **best practices in parallel programming**

- Providing **Services since Oct 2015**
  - Precise understanding of application and system behaviour
  - Suggestion/support on how to refactor code in the most productive way

Do not guess about your code performance, measure it with POP

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

HLRS

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

nag®

RWTH AACHEN UNIVERSITY

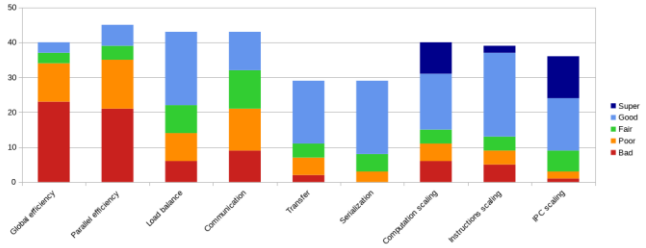
UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

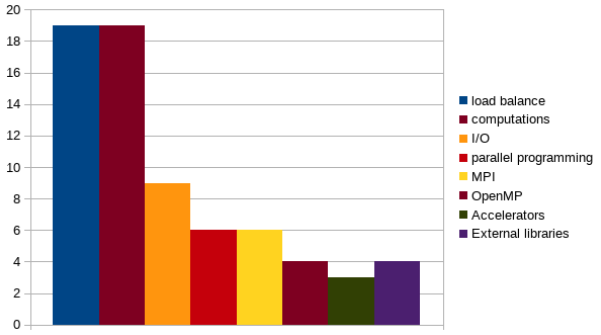
Teratec

Some POP numbers

- More than **300 free services**

- Close to **100 codes improved with our support**
- **95% of the users satisfied** with our work
- Around **5 training events per year**





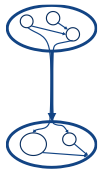
Contact us:	https://www.pop-coe.eu pop@bsc.es @POP_HPC youtube.com/POPHPC

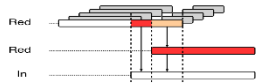
OmpSs-2 and TAMPI

For further information please visit
https://pm.bsc.es/dlb

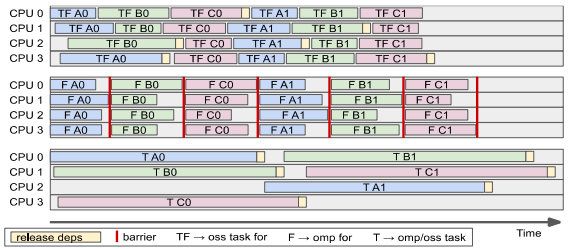
Data-flow programming model

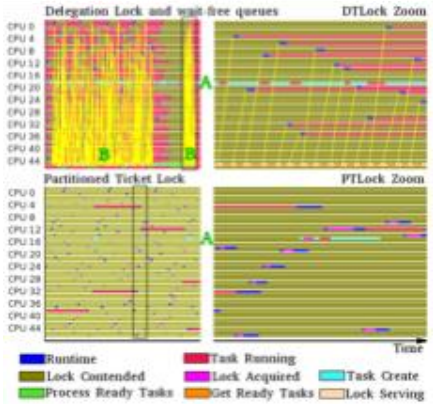
- **Advanced dependency system**
  - *in/inout*, *concurrent*, *commutative*, weak, multideps, scalar & array reductions
- **Optimized for many-core processors**
  - Scalable scheduler based o a novel Delegation Lock
  - Wait-free dependency system
  - Work sharing tasks (*task for*)
- **Good integration with communication and storage APIs**
  - TAMPI (MPI)
  - TAGASPI (GASPI)
  - TASIO (Linux) & TASPDK (Intel SPDK)
- **Native integration with Dynamic Load Balancing (DLB) library**

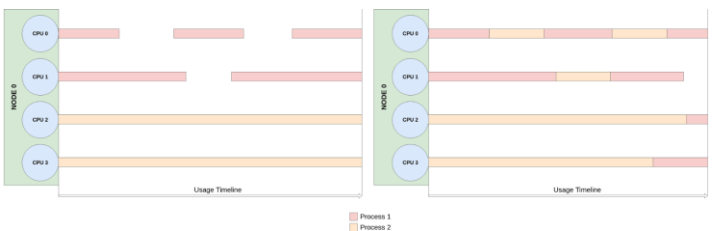


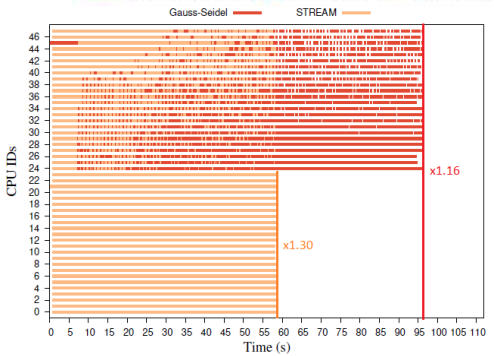



```
int data[N] = {};

#pragma oss task reduction(+: data[0;N/2 + 1])
{ ... }

#pragma oss task reduction(+: data[N/2 - 1;N/2 + 1])
{
    foo(data);
}

// pragma oss task in(data[N/2-2;N])
{ ... }

// pragma oss taskwait
```









Barcelona Supercomputing Center
Centro Nacional de Supercomputación

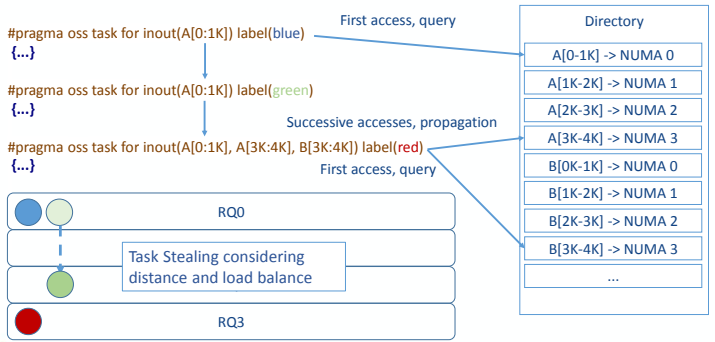
SC 2020 Release

https://github.com/bsc-pm/ompss-2-releases

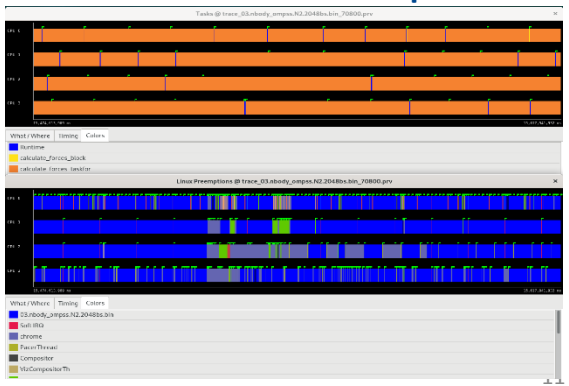
New features

- New LLVM compiler with support for all OmpSs-2 features expect device)
- New user friendly config file
- Support for NUMA systems: data distribution policies, locality-aware scheduler and data tracking system
- Tracing support for kernel and user events
- Enhanced runtime support for hyperthreading
- Enhanced performance for systems with weak memory models (ARM and Power)
- Array reductions in CUDA

OmpSs-2 NUMA-Aware System



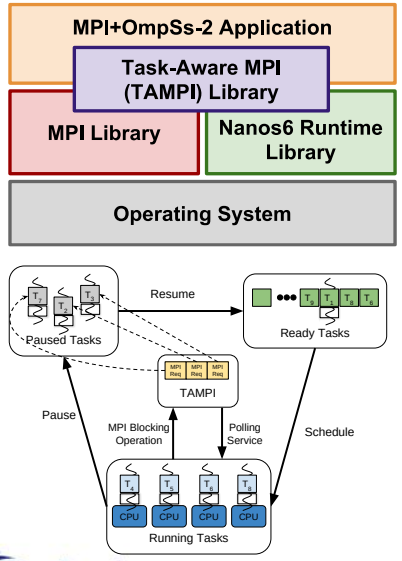
New unified kernel and user space tracing

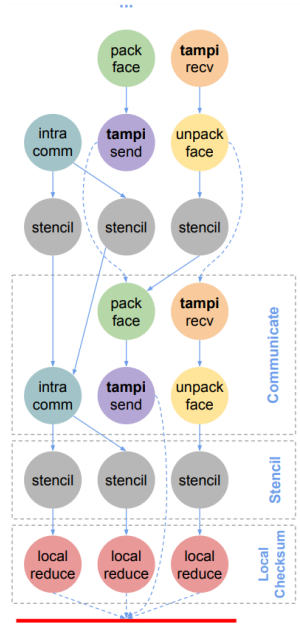


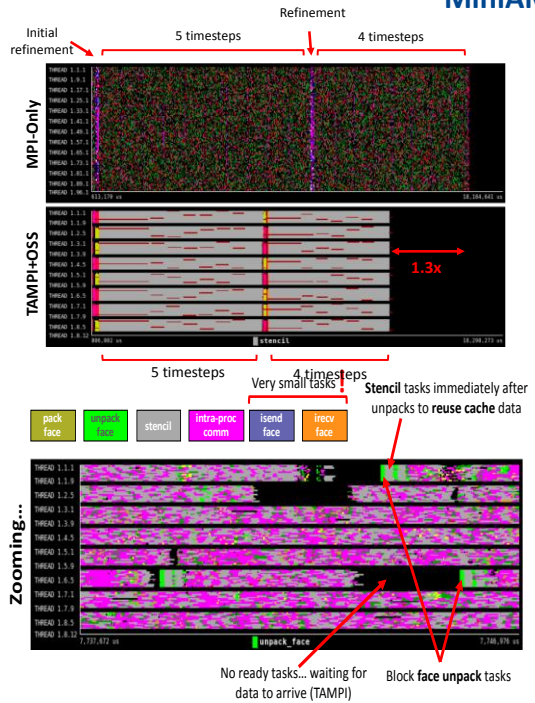
Task-Aware MPI (TAMPI)

https://github.com/bsc-pm/tampi

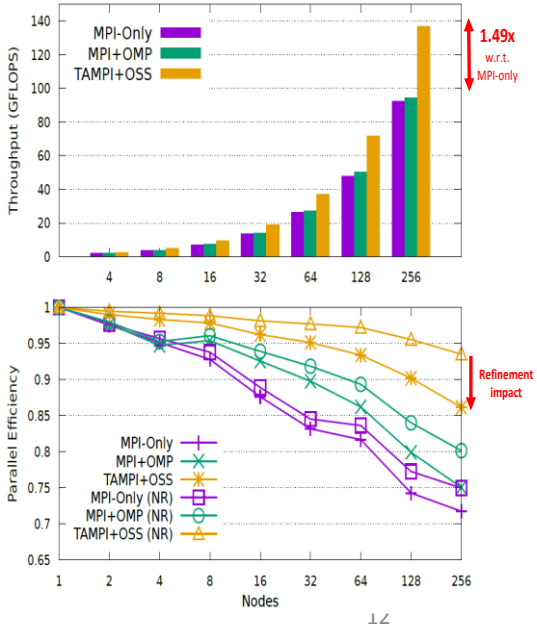
- TAMPI makes easier hybrid programming combining OpenMP/OmpSs-2 and MPI
- It can be used on top of any MPI implementation (Intel, MPICH, ParastationMPI, etc)
- Support of blocking, non-blocking on one-sided (WiP) MPI primitives inside tasks







MiniAMR benchmark



Dynamic Load Balancing (DLB)

For further information please visit
https://pm.bsc.es/dlb

- **DLB** is a dynamic library transversal to the different layers of the **HPC** software stack
- **Objective**: Maximize the utilization of computational resources inside a node

- **DLB** offers different levels of integration:
  - **Transparent** to the application
  - **API** for application fine tuning
  - **API** for runtimes and programming models

Integrated with:
- MPI
- OpenMP
- OmpSs
- Slurm

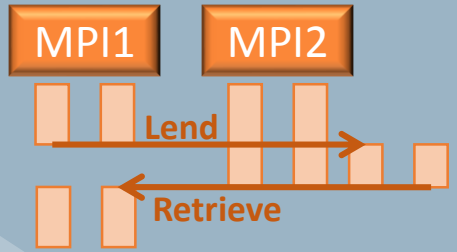
- Since 2012
- Current stable release DLB 2.1
- Available under LGPLv3
- `https://github.com/bsc-pm/dlb.git`

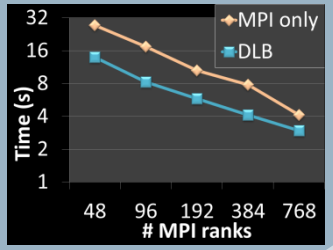
- **`https://pm.bsc.es/dlb`**
  - ✓ Documentation
  - ✓ Downloads
  - ✓ User guide
  - ✓ Tutorial
  - ✓ Publications
  - ✓ Contact

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

LeWI

- Load balance hybrid applications
- Redistribute computational resources at shared memory level





DROM

- Re-assign computational resources at runtime between processes

- **API** for resource managers
  - Prioritize applications
  - Allow interactive visualization
- **API** for applications
  - Release resources

TALP

- Collect application performance metrics at runtime

```
### Monitoring Region App Summary ###########
### Name:                          MPI Execution
### Parallel efficiency :          0.87
###   - Communication eff. :       1.00
###   - Load Balance :             0.87
###       - LB_in :                0.87
###       - LB_out:                1.00
```

- Application summary of efficiencies
  - At finalization and at runtime
  - Whole execution and user defined regions
- **API** for monitoring efficiencies at runtime

DLB has three modules **independent** and **compatibles**

OmpSs2 + OpenACC

For further information please contact
antonio.pena@bsc.es

Why OmpSs-2 + OpenACC?

	Coding Prod.	Perf.
CUDA	🔴	🟢
OpenACC	🟢	🔴
OpenACC + CUDA	🟡	🟢
OmpSs + CUDA	🟡	🟢
OmpSs + OpenACC	🟢	🟡
OmpSs + OpenACC + CUDA	🟢	🟢

EPEEC PRODUCTIVITY AT EXASCALE

```
#pragma oss task device(openacc) in(…) out(…)
void my_function_kernel(args);

void my_function_kernel(args) {
#pragma acc kernels
  …
}
```

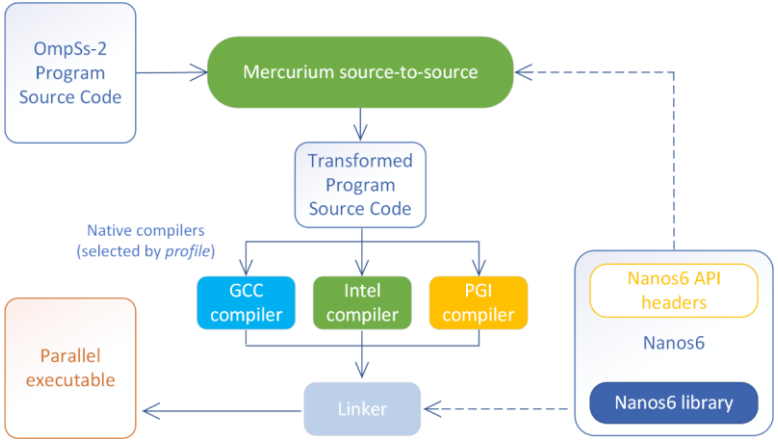
Each function call will become a task invocation → SO EASY!

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

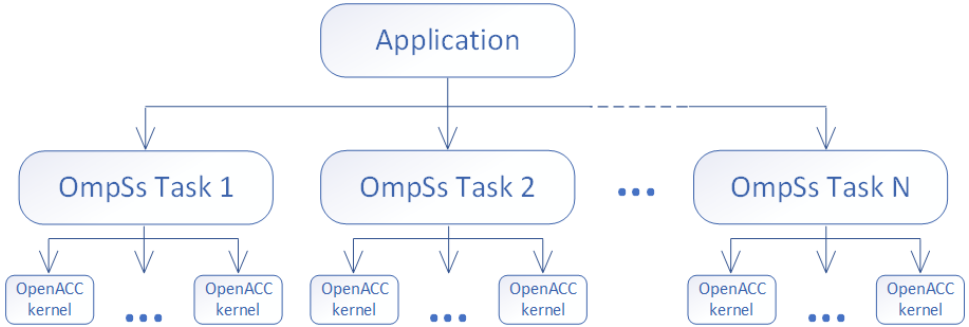
OmpSs-2 + OpenACC Interoperability

(Currently available as of OmpSs-2 v2.4)

- **Combining the programming models:**
  - The user is expected to use only compute constructs from the OpenACC model;
  - No data transfers clauses
  - No async (asynchronous behavior is implied in tasking; managed automatically by OmpSs-2)
  - No executables (initialization, device management)



Compiler Flow

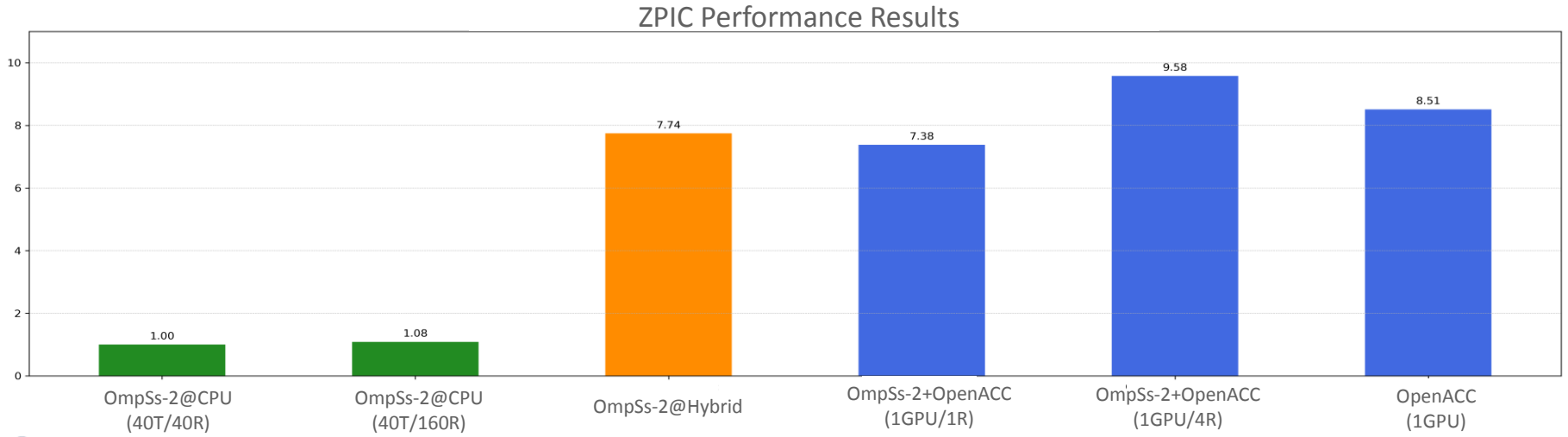


Programing Model Hierarchy

- **Hierarchy of Programming Models**
  - OmpSs-2 task-parallel programming model is combined with OpenACC data-parallel programming model
  - Scientific applications can be broken down to parallel tasks, in turn those tasks can be suited for data-parallel accelerator execution
  - OmpSs-2 runtime can be launching multiple OpenACC regions concurrently, from independent tasks running on different CPU threads

Experimental evaluation

- CTE-POWER cluster based on IBM Power9 processors
  - 2 x IBM Power9 8335-GTH @ 2.4GHz (3.0GHz on turbo, 20 cores and 4 threads/core, total 160 threads per node)
  - 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2.
- ZPIC: 2D Electromagnetic particle-in-cell (https://github.com/nlg550/ZPIC_OmpSs2)
- Speedup over pure OpenACC version achived by taking advantge of more OmpSs-2 parallel tasks that can be overlaped to hide latency.
- Code complexity remains low, programmers only concentrates on compute construct while OmpSs-2 handles task scheduling and synchronization



T: Threads, **R**: Compute Region Decomposition, **Hybrid**: Regions partitioned between CPU and GPU

Note: Regions are assigned to OmpSs-2 task, pure OpenACC can not support this feature

OmpSs@FPGA

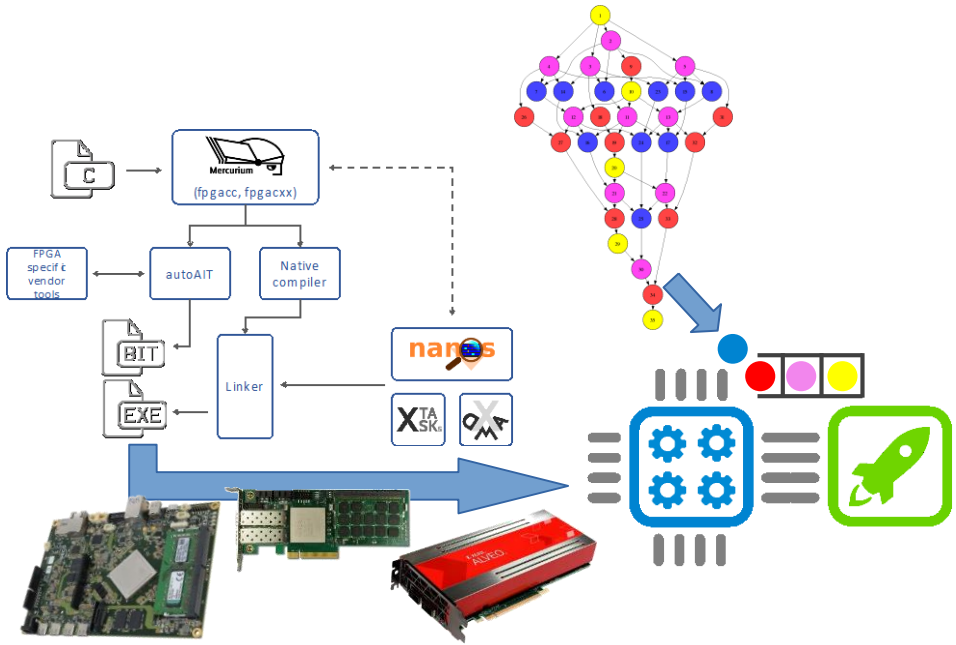
For further information please contact

pm.bsc.es

OmpSs@FPGA

- Allows easy programmability (pragma based offload) and usability (one-click compilation with autoAIT) of FPGAs

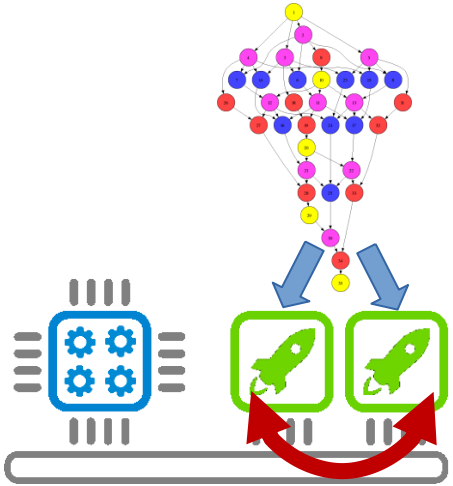
```
#pragma omp target device(fpga)
#pragma omp task copy_inout([BS]a)
void update_fpga(int *a, int val, size_t BS) {
  for (size_t i=0; i<BS; ++i) a[i] += val;
}

void update_blocked(int *a, int val, size_t LEN, size_t BS) {
  for (size_t i=0; i<LEN; i+=BS)
    update_fpga(a+i, val, BS);
}

int main(...) {
  int *a = (int *)malloc(NUM_ELEMENTS*sizeof(int));
  update_blocked(a,2020,NUM_ELEMENTS,NUM_ELEMENTS_BLOCK);
  #pragma omp taskwait
}
```



eOmpSs@FPGA

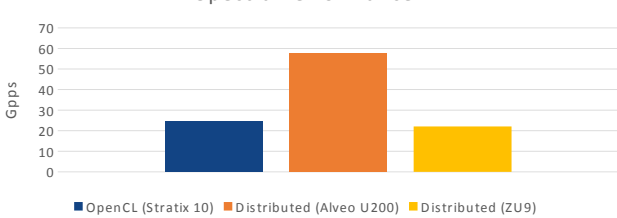
- An evolution of OmpSs@FPGA classical offload model distributes the control among the computing elements embedding a HW runtime in the FPGA

```
#pragma omp target device(fpga)
#pragma omp task inout([BS]a)
void update_fpga(int *a, int val, size_t BS) {
  for (size_t i=0; i<BS; ++i) a[i] += val;
}
#pragma omp target device(fpga)
#pragma omp task inout([LEN]a) inout([LEN/BS]index)
void update_blocked(int *a,int *index,int val,size_t LEN,size_t
BS) {
  for (size_t i=0; i<(LEN/BS); i++])
    update_fpga(a+BS*index[i], val, BS);
  #pragma omp taskwait
}
int main(...) {
  int *a = (int *)malloc(NUM_ELEM*sizeof(int));
  int *index=(int*)malloc(NUM_ELEM/NUM_ELEM_BLOCK*sizeof(int));
  update_blocked(a, index, 2020, NUM_ELEM, NUM_ELEM_BLOCK);
  #pragma omp taskwait
}
```

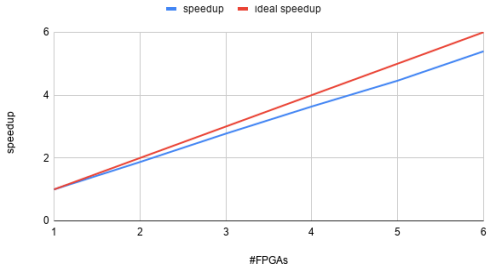


Allows FPGA2FPGA direct control & communication

Improves FPGA Performance over centralized models



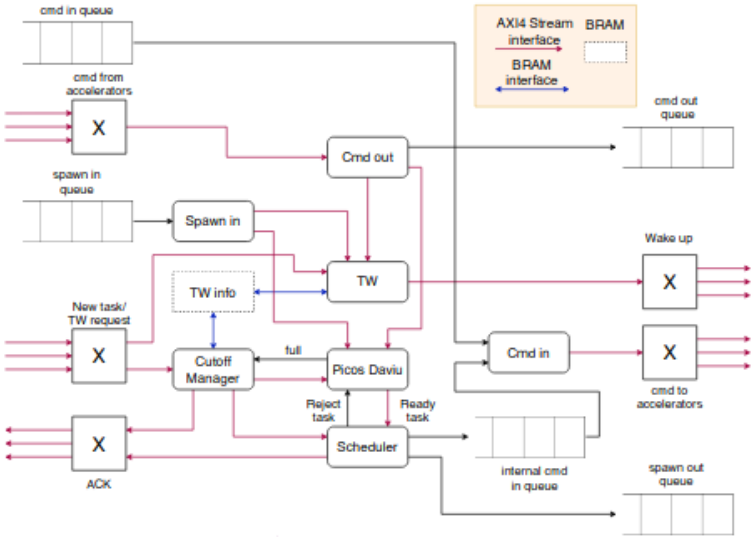
N-Body scalability for multiple FPGAs



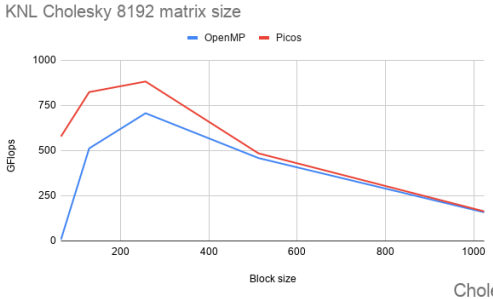
Picos OmpSs manager

- Tasks and dependences management HW support for many-core architectures

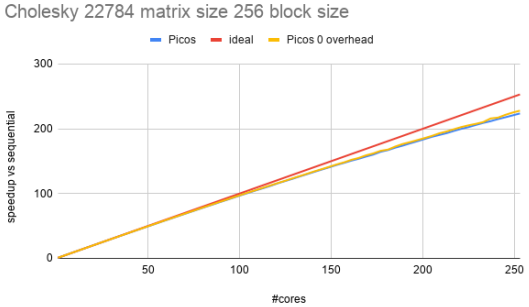
New improved HW support



Better absolute performance than alternative software runtimes for existing architectures



Nearly ideal scalability for future designs

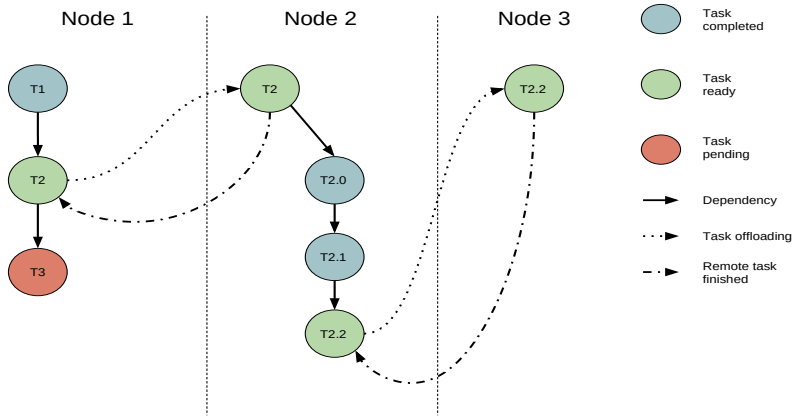


OmpSs@cluster

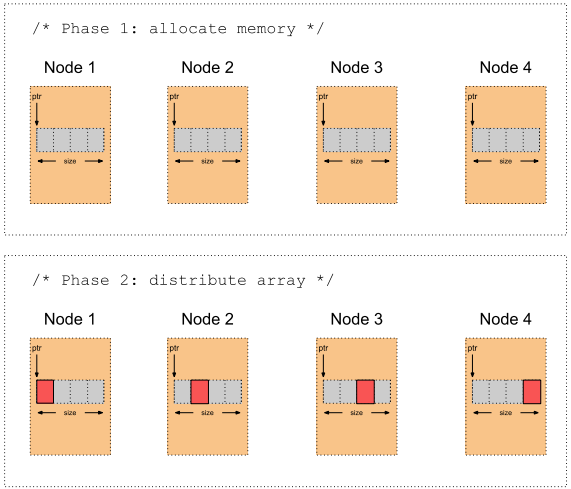
For further information please contact
paul.carpenter@bsc.es

OmpSs-2@cluster

- OmpSs-2 programming model for distributed memory
- Nanos6 runtime transparently offloads tasks among nodes
  - Scheduling, dependencies and data copies handled by the runtime system
- Interoperable with MPI and DLB (dynamic load balancing)
- Common virtual memory layout among all nodes in application MPI rank



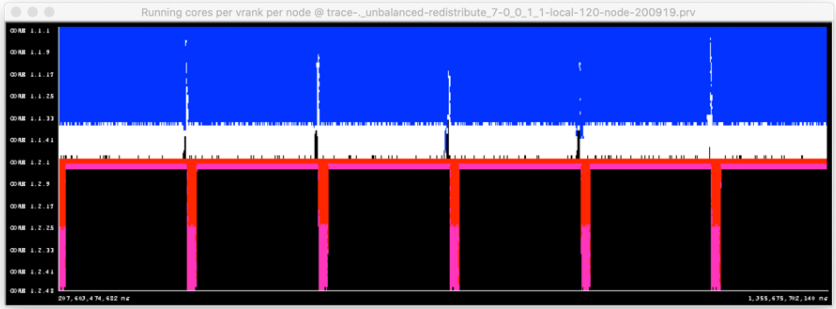
Transparent task offloading and data transfers



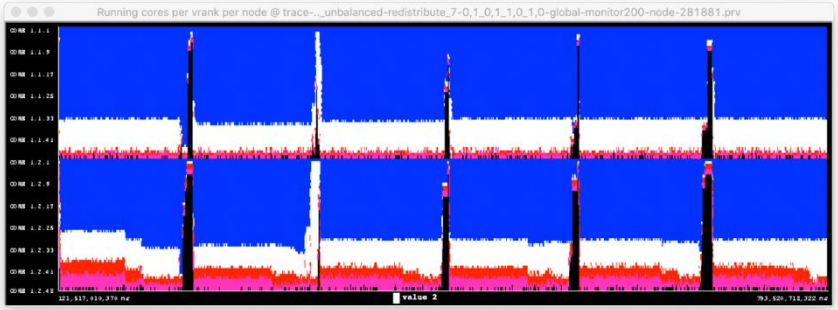
Common virtual memory layout

OmpSs-2@cluster: advantages

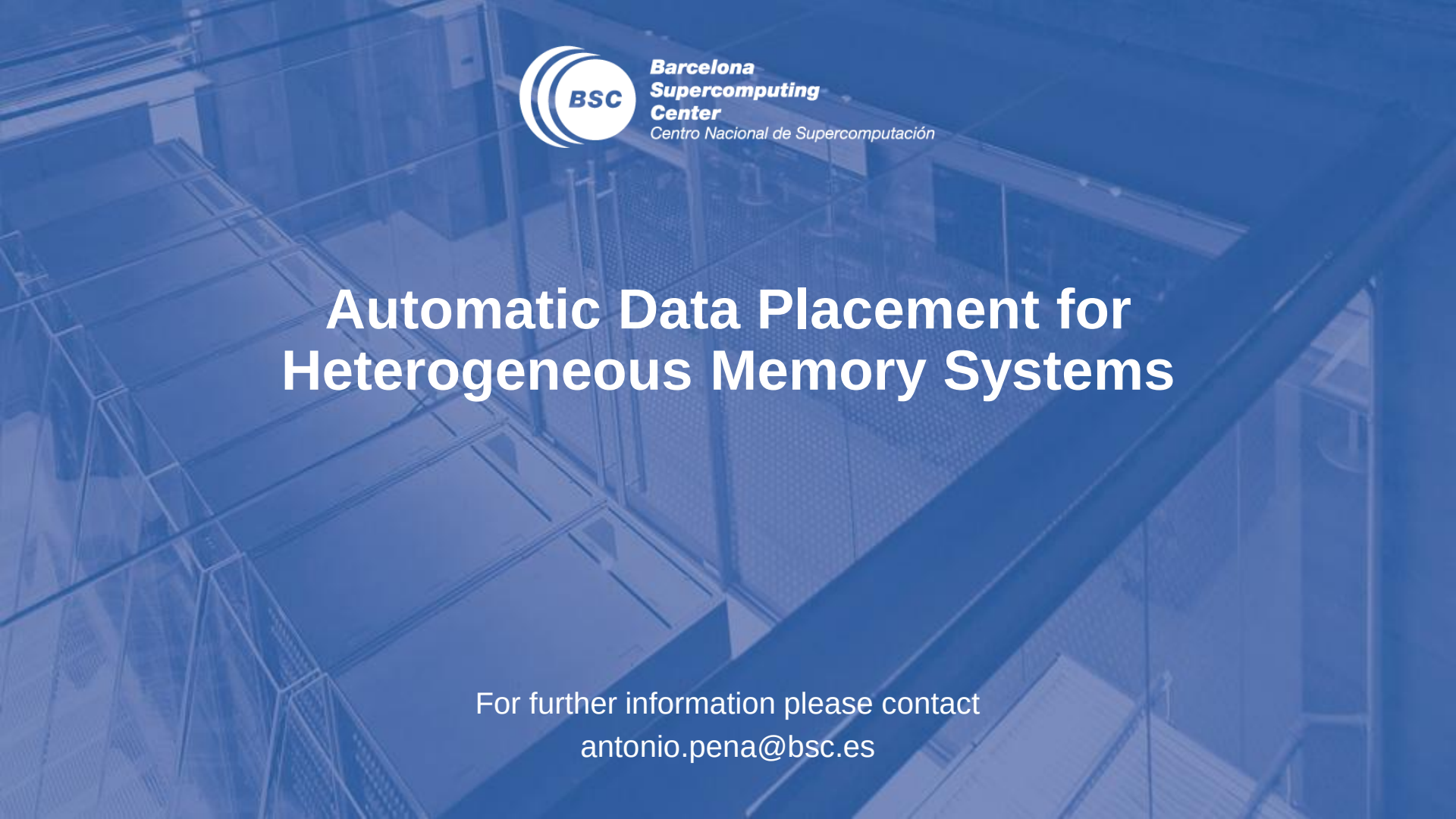
- Ease of programming
  - OmpSs-2 task annotations
- Efficient
  - Task hierarchies supported using weak dependencies
- Run OmpSs-2 application across multiple nodes
- Cross-node DLB (dynamic load balancing) in MPI+OmpSs-2 application



MPI + OmpSs-2: imbalanced load on two nodes



MPI + OmpSs-2@cluster: balanced load on two nodes

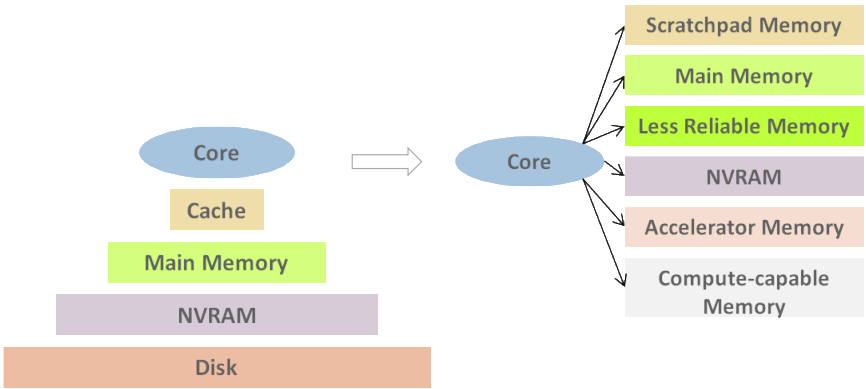


Automatic Data Placement for Heterogeneous Memory Systems

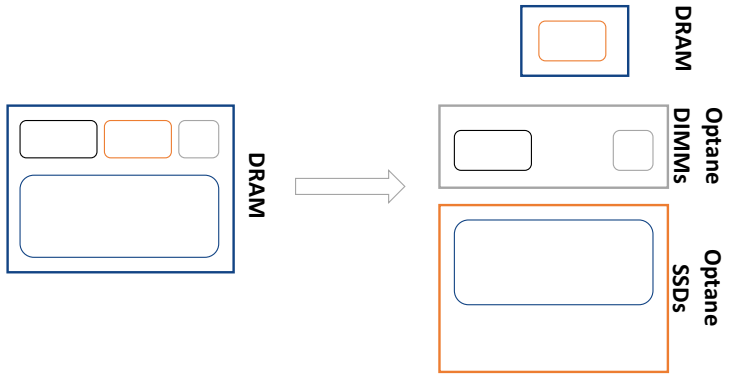
For further information please contact
antonio.pena@bsc.es

Heterogeneous Memory Systems

Bring memories as 1st class citizens



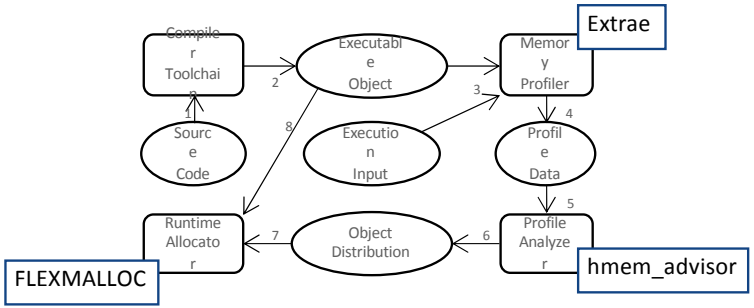
Distribute memory objects



Methodology (Intel Optane PMem)

Methodology

1. Profile
2. Assess optimal distribution
3. **Run *unmodified* binary**



Evolved version of:

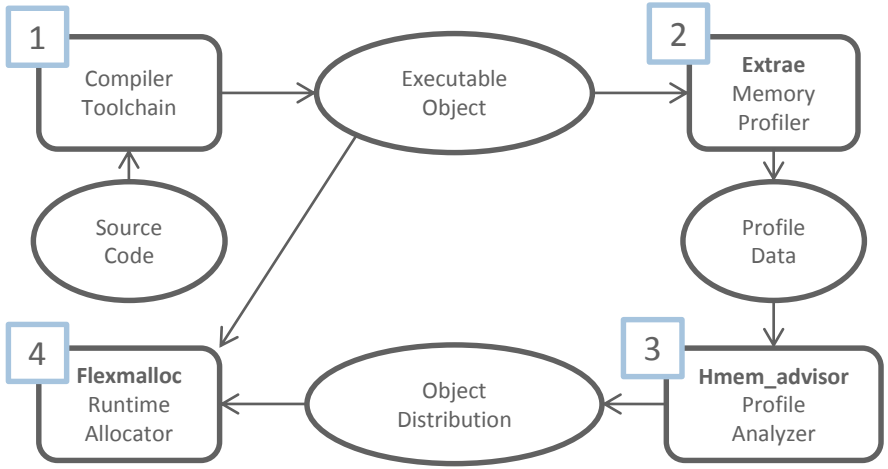
A. J. Peña and P. Balaji, “Toward the efficient use of multiple explicitly managed memory subsystems”, IEEE Cluster 2014

- LAMMPS
  - **Keep performance w/ DRAM reduction vs. Memory Mode @16GB**
  - Even at 1/4th DRAM size!
- OpenFOAM
  - **The gain in performance w.r.t. Memory Mode is higher as simulations get longer**

This work is done under the Intel-BSC Exascale Laboratory

Automatic data placement for heterogeneous memory systems

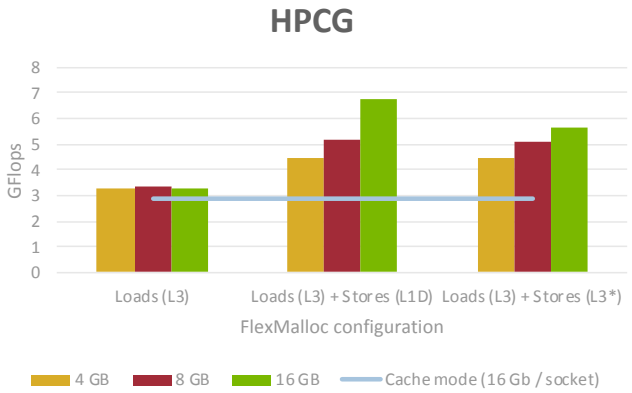
- Evaluate performance of explicitly managed memory tiers vs. cache-like HW-managed alternatives
- **Application-level automatic data placement Framework**
  - Ecosystem of tools
  - Dynamic memory allocation granularity (objects)
  - Offline object distribution based on initial profiling execution

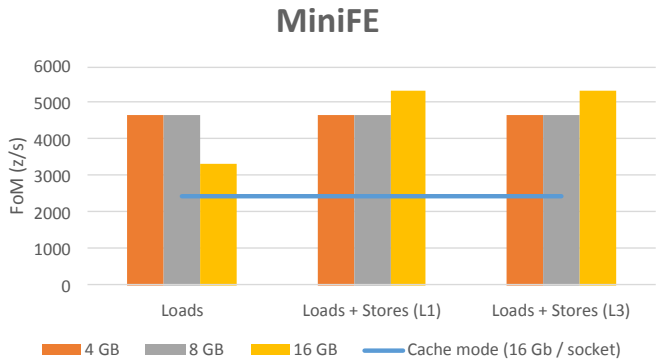


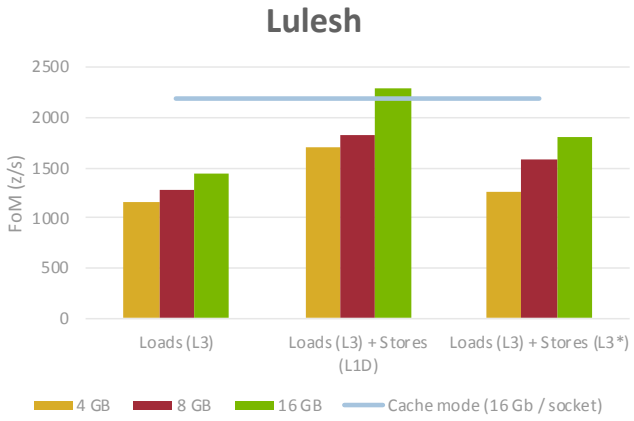
- **Summary of the workflow**:
  1. Compile application with the flags to generate debugging information. No source code changes are required
  2. Profiling execution to collect per-object cache behavior data (e.g. misses , avg. access time, …)
  3. Assess the optimal distribution of the different objects among the available memory subsystems
  4. Execute with interposition library that automatically places each object to the corresponding memory subsystem

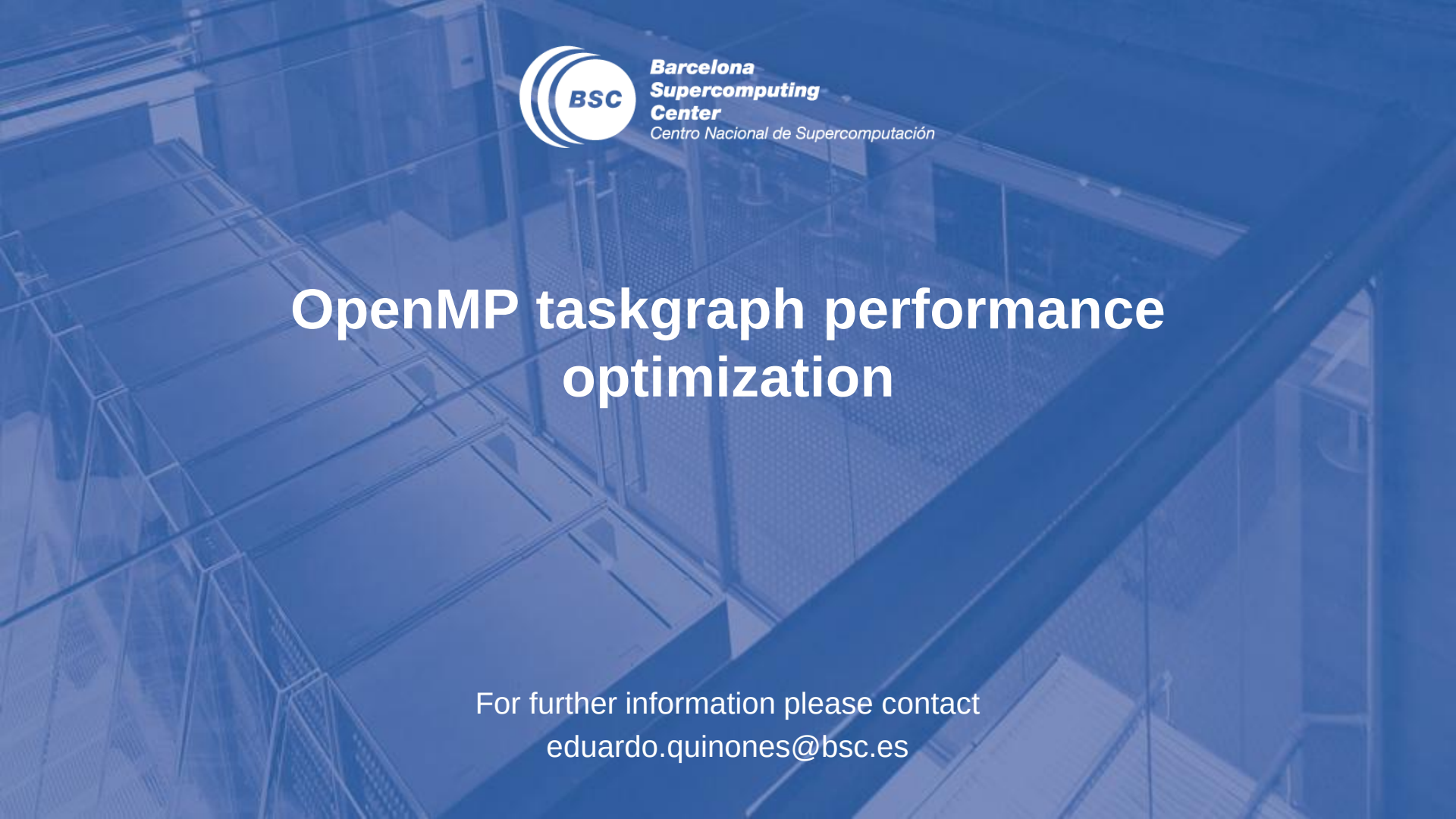
Experimental evaluation

- Server with DDR4 + OptaneDC persistent memory DIMMs
  - Intel Xeon Platinum 8260L CPU @ 2.30GHz
  - Intel software stack (compiler, MPI)
- Data distributions based on different profiling data and several amounts of DRAM available
  - Memory-mode baseline uses 16GB of DRAM as HW-managed cache
- Up to around 2x speedup over baseline for HPCG and MiniFE
  - Even using only ¼ of DRAM w.r.t memory-mode
- In other cases we are within negligible performance improvement/degradation w.r.t memory-mode









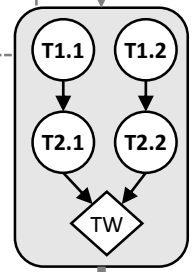
OpenMP taskgraph performance optimization

For further information please contact
eduardo.quinones@bsc.es

OpenMP taskgraph for Performance Optimization

The task-based OpenMP user code is replaced by an optimised execution of the corresponding **Task Dependency Graph (TDG)** on the targeted device, i.e., **SMP** and **GPU**

```
void foo() {
  #pragma omp parallel
  #pragma omp single
  {
    int a[2];
    #pragma omp taskgraph
    {
      for(int i=0;i<2;i++) {
/*T1*/  #pragma omp task depend(out:a[i])
        a[i]=0;
/*T2*/  #pragma omp task depend(in:a[i])
        a[i]++;
      }
      #pragma omp taskwait
    }
  }
}
```




```
void foo() {
  #pragma omp parallel
  #pragma omp single
  {
    int a[2];
    execute_TDG(a);
  }
}
```

OpenMP

NVIDIA CUDA

```
void foo() {
  // Creation of the kernel node T1.1
  cudaGraphNode_t node_T1_1;
  cudaKernelNodeParams nodeArgs_T1_1={0};
  nodeArgs_T1_1.func = (void *) f;
  void *kernelArgs_T1_1[3] = {&Ah[1]};
  nodeArgs_17.kernelParams =(void * * ) kernelArgs_17 ;
  cudaGraphAddKernelNode(&node_17,graph[0],NULL, 0,&nodeArgs_17);
  // Creation of the kernel node T2.1
  ...
}
```

TDG representation of the `taskgraph` region optimised for **OpenMP SMP** and **CUDA graphs**

Benefits

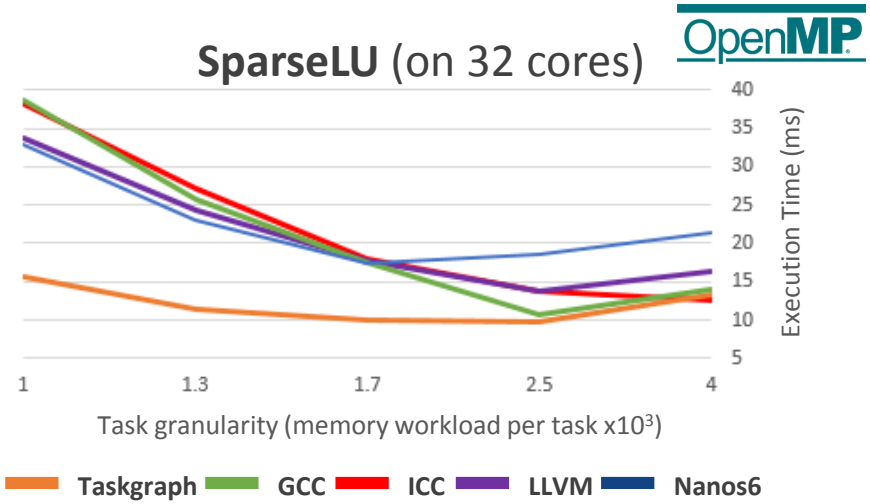
1. The runtime task structure becomes much more lighter, allowing to **exploit finer-grain parallelism**
2. The OpenMP TDG representation can be transformed to other graph-based API, e.g., CUDA graphs, **enhancing programmability**

OpenMP taskgraph for Performance Optimization

OpenMP taskgraph in SMP

Exploit **finer-grain parallelism** by reducing:

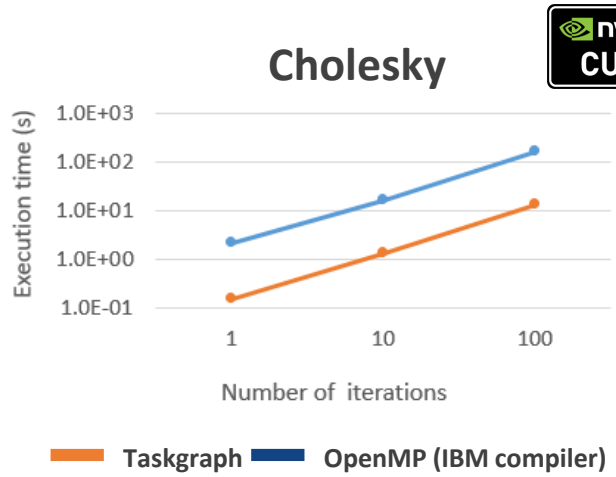
- Contention on runtime structures
- Overhead of the OpenMP tasking model management



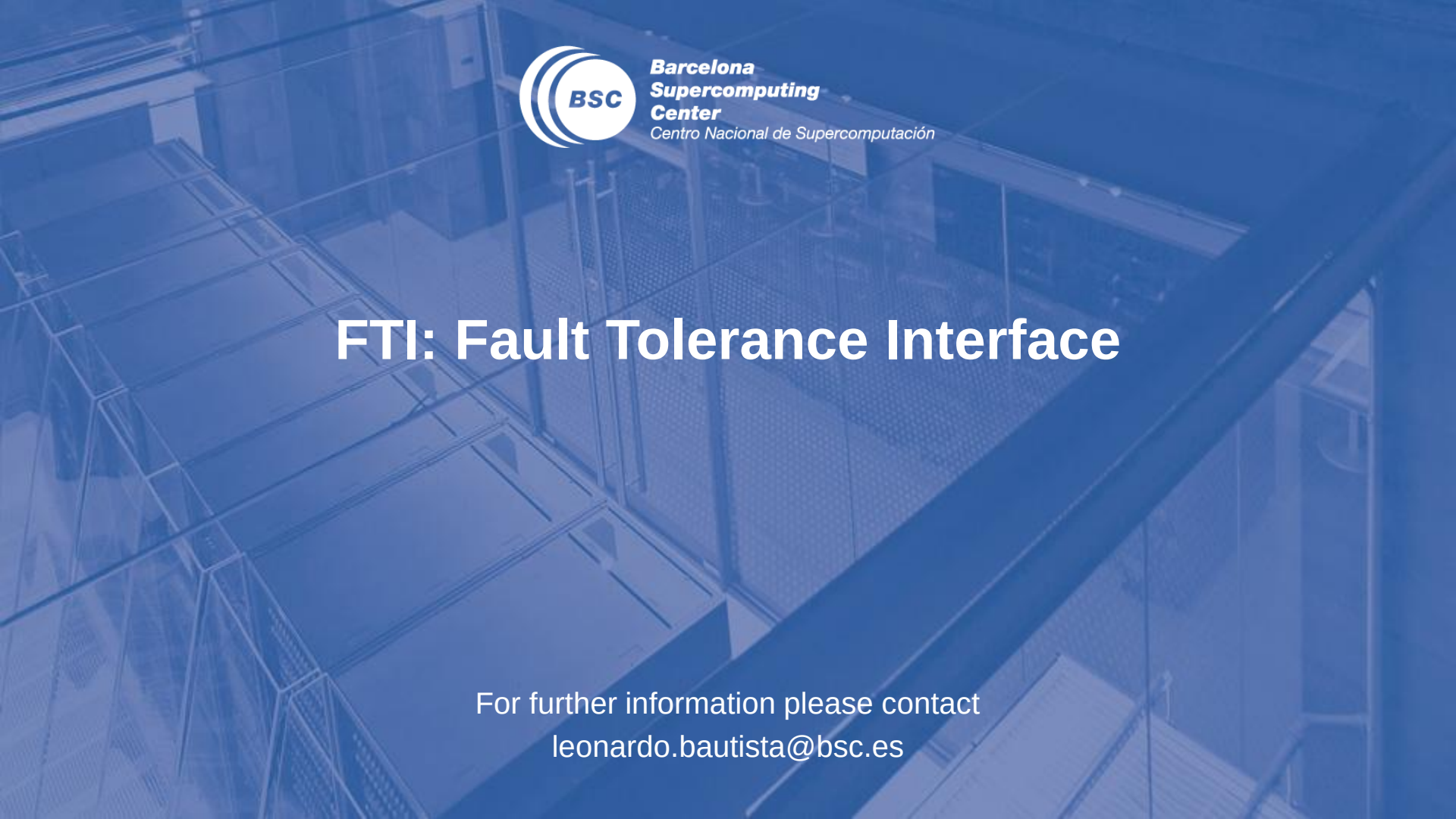
Marenostum 4 node: Two sockets Intel Xeon Platinum 8160 CPU, 24 cores each, @2.10GHz, 32KB L1 and 1MB L2, private to each socket

OpenMP taskgraph to CUDA graph

- Exploit the define-once-run-repeatedly execution model
- Enhanced programmability:
  - ***+6 lines of code*** added with OpenMP taskgraph vs. ***+15500 lines of code*** for the CUDA graph definition



NVIDIA V100, 16 GB HBM2, CUDA v10.1



FTI: Fault Tolerance Interface

For further information please contact
leonardo.bautista@bsc.es

Fault Tolerance Interface (FTI)

Multilevel Checkpointing library

- 4 levels of performance / reliability
- Fast Asynchronous Checkpointing
- Transparent Support for GPUs
- IO modes: POSIX, HDF5, MPI-IO
- Support for both N-N and N-1 ckpt.
- Elastic Recovery with less/more ranks
- Differential Dynamic Checkpointing
- Extentions to complement ABFT
- Incremental Checkpointing

Local Storage: SSD, PCM, NVM.
Fastest checkpoint level.
Low reliability, transient failures.

Partner Copy: Ckpt. Replication.
Fast copy to neighbor node.
It tolerates single node crashes.

RS Encoding: Ckpt. Encoding.
Slow for large checkppoints.
Reliable, multiple node crashes.

File System: Classic Ckpt.
Slowest of all levels.
The most reliable. Power outage.

New release! FTI v1.5 “Rabat”:https://github.com/leobago/fti/releases/tag/1.5
Full Documentation: https://fault-tolerance-interface.readthedocs.io/en/latest/

Fault Tolerance Interface (FTI)

Fast Asynchronous Checkpointing (**SC'11**)

Checkpointing and Failure Prediction (**IPDPS'13**)

Checkpointing and Power Management (**PMBS'14**)

Introspective Analysis and Checkpointing (**IPDPS'16**)

Extentions for ABFT and partial restart (**FTXS'18**)

Differential Dynamic Checkpointing (**CCGrid'19**)

Transparent Support for GPUs (**CCGrid'20**)

Elastic Recovery with less/more ranks (**HiPC'20**)*

*(to appear)

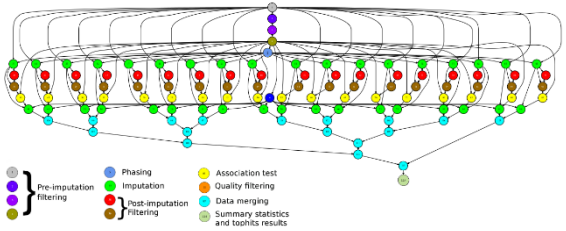
Programming with PyCOMPSs/COMPSs

For further information please visit

compss.bsc.es

Programming with PyCOMPSs/COMPSs

- Task-based programming model
- Distributed computing platforms: clusters, clouds, and container infrastructures
  - Docker, Singularity and Mesos
- Python, C/C++, Java interfaces
- Builds a task graph at runtime that express potential concurrency
- Parallelism and data transfer handled by runtime
- Support for parallel tasks: MPI, OpenMP, threads
- Other features: task contraints, stream data, IO tasks

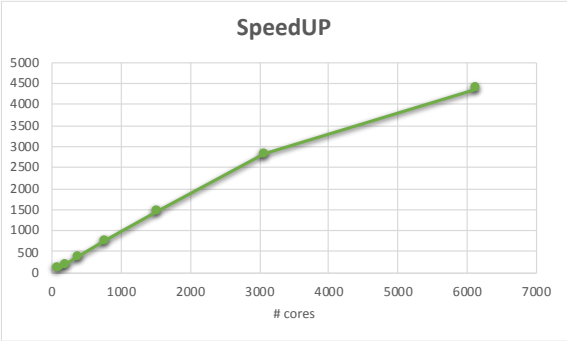



```
@task(c=INOUT)
def multiply(a, b, c):
    c += a*b
```

```
initialize_variables()
for i in range(MSIZE):
    for j in range(MSIZE):
    for k in range(MSIZE):
            multiply (A[i][k], B[k][j],
C[i][j])
compss_barrier()
```

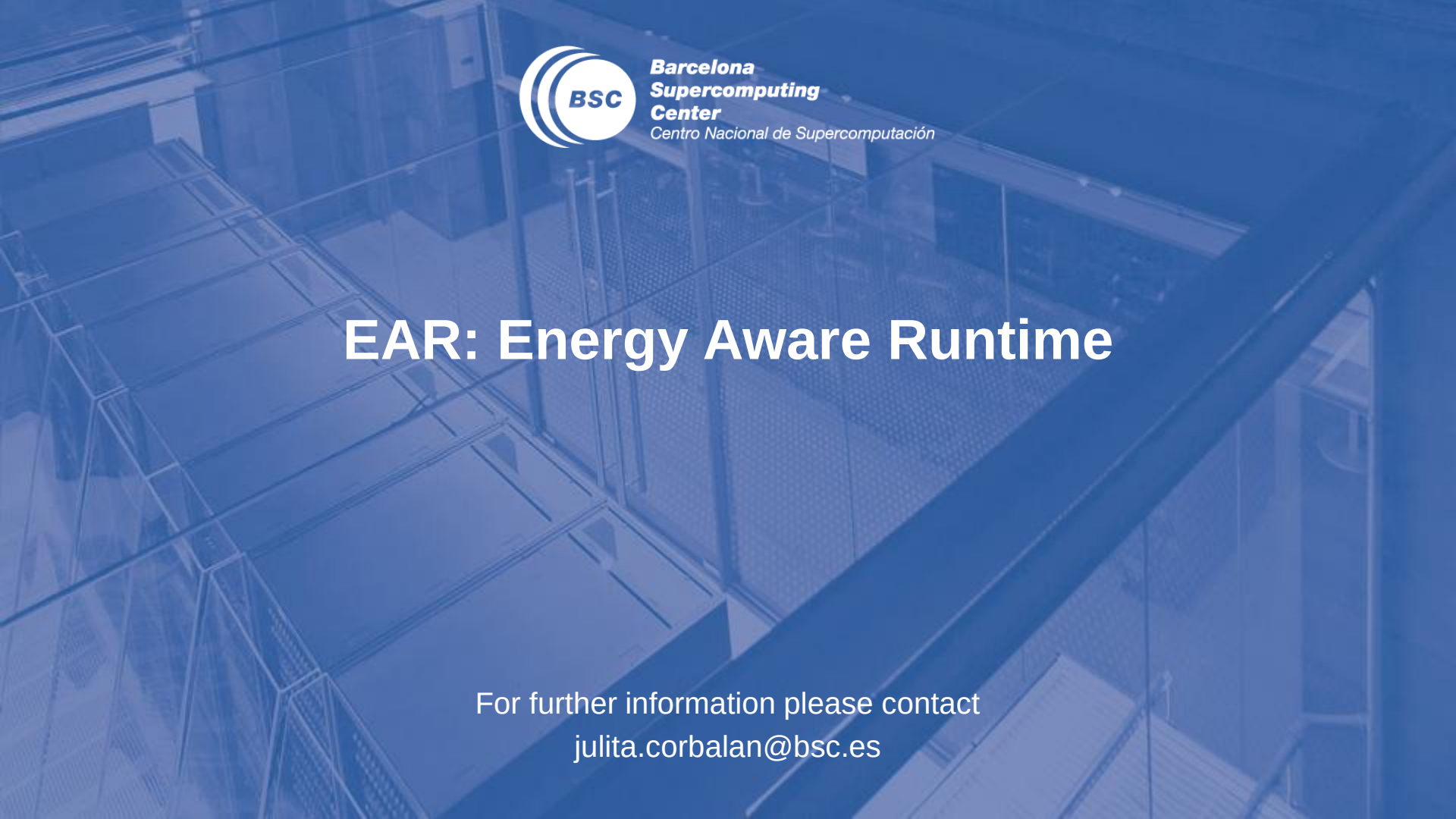
Programming with PyCOMPSs/COMPSs

- Good scalability in real problems



Execution time and speedup of Multi-Level Monte Carlo algorithm executed in up to 128 nodes of MareNostrum 4 (ExaQUte project)

- **COMPSs version 2.8: New features**
  - New reduction clause to support user-defined reductions. The runtime implements a locality aware algorithm that reduces inter-node transfers
  - New @container annotation to support tasks deployed in containers
  - Other extensions and bug fixing



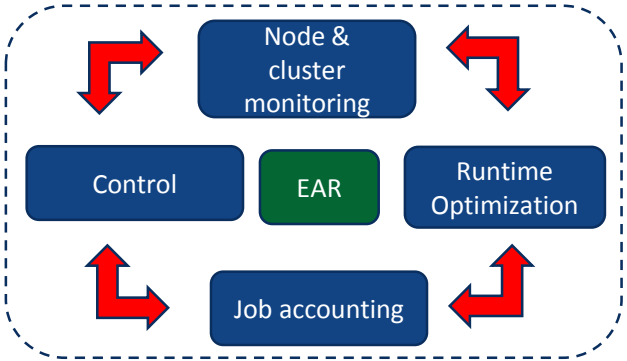
EAR: Energy Aware Runtime

For further information please contact
julita.corbalan@bsc.es

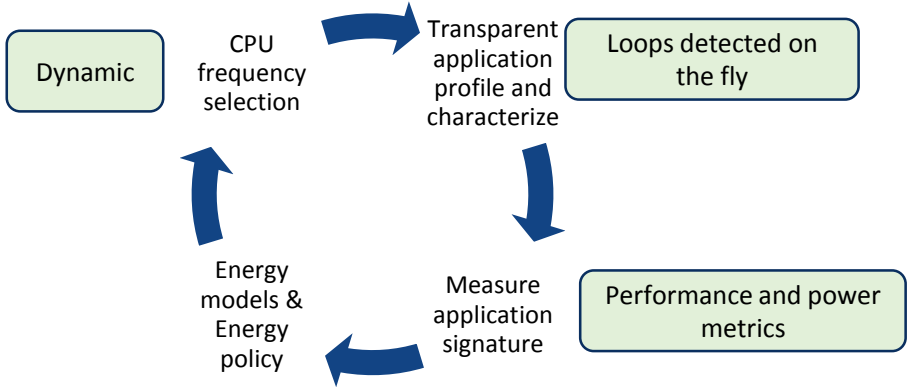
EAR: Energy Aware Runtime

The System Software tool for energy optimization, monitoring, control and accounting

- BSC – Lenovo collaboration project since 2016



Energy management framework



EAR runtime

- Power monitoring extensible through plugins. EAR DB
- Flexible and configurable
- Heterogenous cluster support
- Cluster energy limits

- **100% Transparent to users and Runtime optimization**
- **Neither user input nor historic information**
- Energy models and policies as plugins
- SLURM plugin for Job submission. Intel MPI and OpenMPI

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

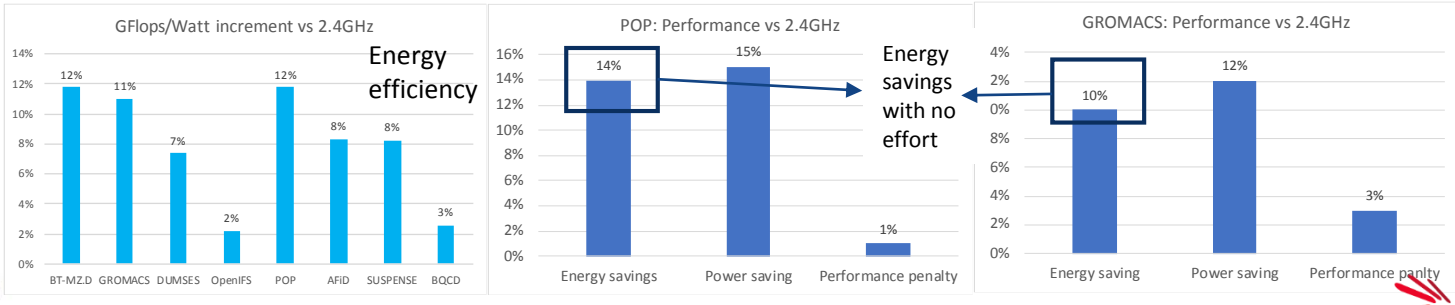
Lenovo

EAR

EAR: Energy Aware Runtime

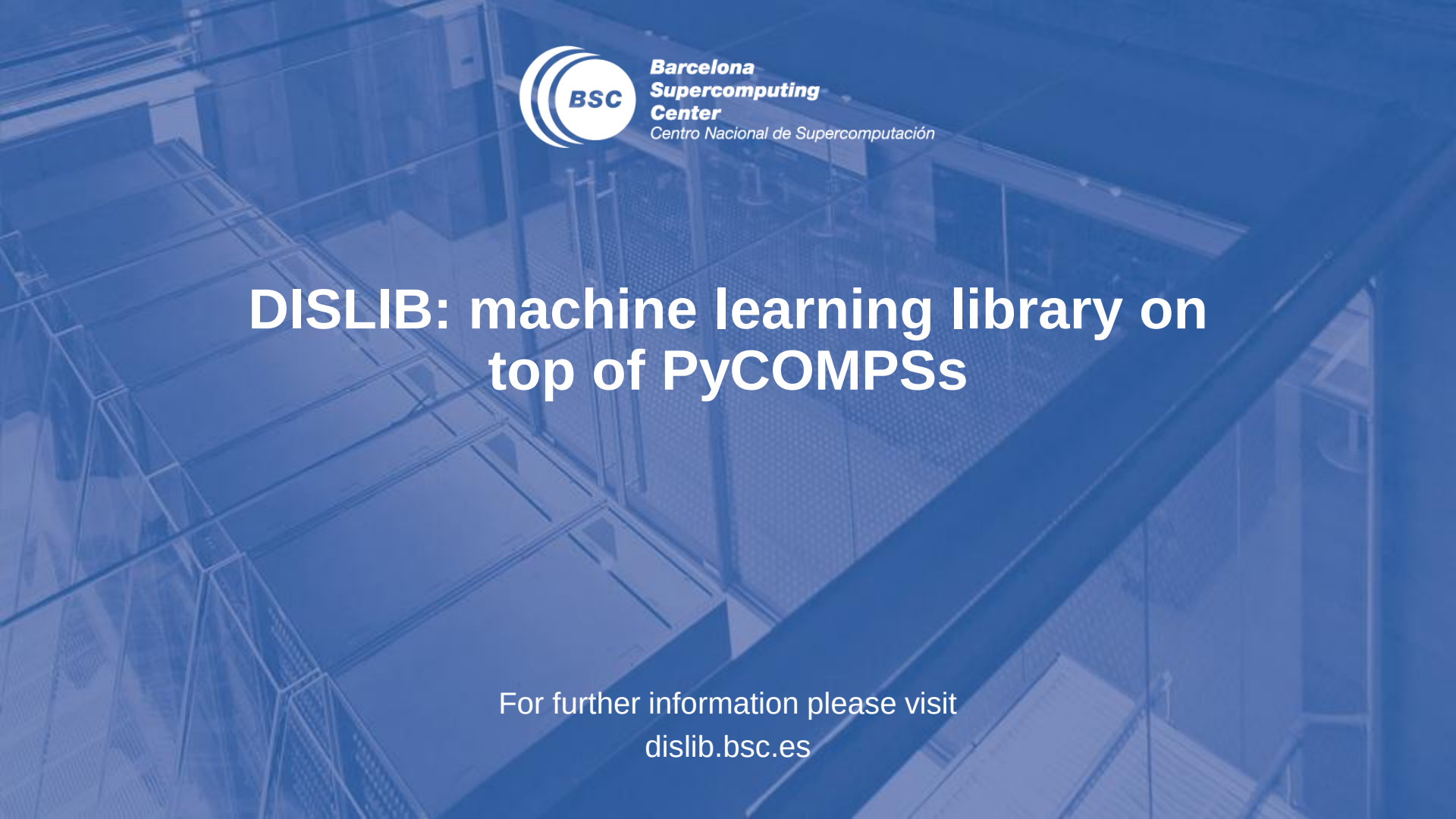
More info…

- **BSC Contact:** Julita Corbalan **– BSC (julita.corbalan@bsc.es)**
- Operational since August 2019 at LRZ on **SuperMUC NG 6480 nodes system**
- **Open Source & Licence:** BSD-3 license for individual/non-commercial use . EPL-1.0 license for commercial use
- Download: https://gitlab.bsc.es/ear_team/ear/-/tree/EAR_3.3
- Professional services through EAS BSC-UPC spin-off: www.eas4dc.com
- **What's next**: NVIDIA and AMD support. Powercap.



10% of improvement in energy efficiency

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



DISLIB: machine learning library on top of PyCOMPSs

For further information please visit

dislib.bsc.es

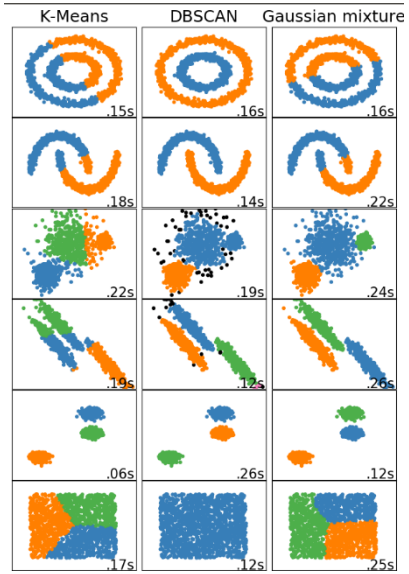
Dislib: parallel machine learning

dislib: Collection of machine learning algorithms developed on top of PyCOMPSs

- Unified interface, inspired in scikit-learn (fit-predict)
- Based on a distributed data structure (ds-array)
- Unified data acquisition methods
- Parallelism transparent to the user – PyCOMPSs parallelism hidden
- Open source, available to the community

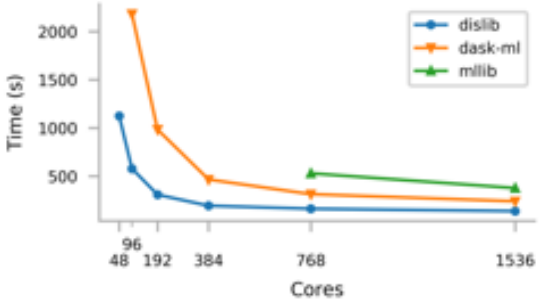
Provides multiple methods:

- data initialization
- Clustering
- Classification
- Model selection, ...

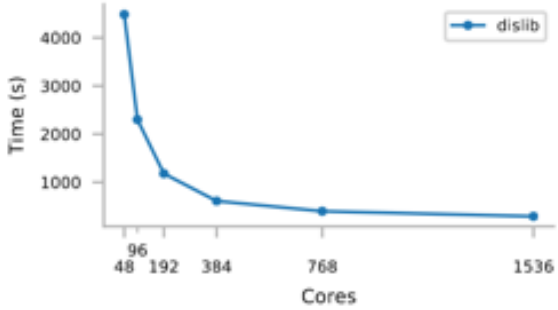


Dislib: parallel machine learning

- Good scalability in cluster execution. For very large sizes, dislib can obtain results while competitors fail to finish the execution



Kmeans:
500 million samples
100 features



Kmeans:
2 billion samples
100 features

- **Dislib version 0.6.0: new features**
  - New methods: Multivariate linear regression, SVD (Singular Value Decomposition), PCA using SVD, ADMM Lasso, Daura clustering
  - New ds-array operators, matmul, kronecker product and rechunk methods for of ds-arrays
  - Other improvements and bug-fixing

Numerical libraries using OmpSs

For further information please contact
marc.casas@bsc.es

Numerical Libraries: Simple and efficient parallel codes

- Dense Linear Algebra: Our code transformations leverage data locality while keeping code simplicity.

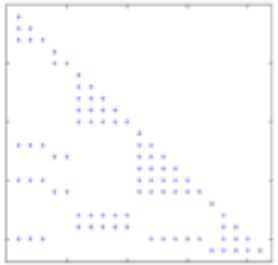
Algorithm 1 *baseline* dtrsm

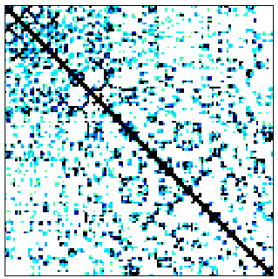
```
1: numRows = M                                          ▷ There are a total of M tiles in B
2: for t = 1 in numRows do
3:     #pragma omp task in(A_tt) inout(B_t)
4:     B_t = TRSM(B_t, A_tt)
5:     for n = t + 1 in numRows do
6:         #pragma omp task in(A_nt, B_t) inout(B_n)
7:         B_n -= A_nt B_t                              ▷ dgemm tasks to partially update the tiles of B below t
8:     end for
9: end for
```

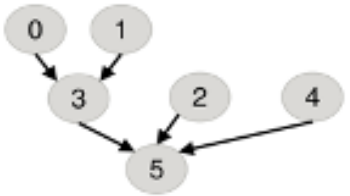
Algorithm 2 *dgemm-fuse* dtrsm

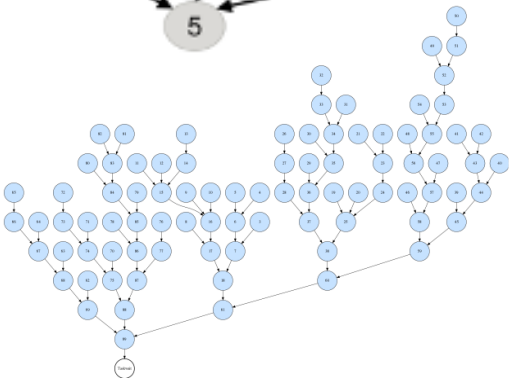
```
1: numRows = M                                          ▷ There are a total of M tiles in B
2: for t = 1 in numRows do
3:     #pragma omp task in(A_tt) inout(B_t)
4:     B_t = TRSM(B_t, A_tt)
5:     #pragma omp task in(A_[t+1:numRows]t, B_t) inout(B_[t+1:numRows])   ▷ Fused dgemm task
6:     for n = t + 1 in numRows do
7:         B_n -= A_nt B_t
8:     end for
9: end for
```

- Sparse Linear Algebra: We leverage advanced parallel constructs to express sparsity-driven dependencies.



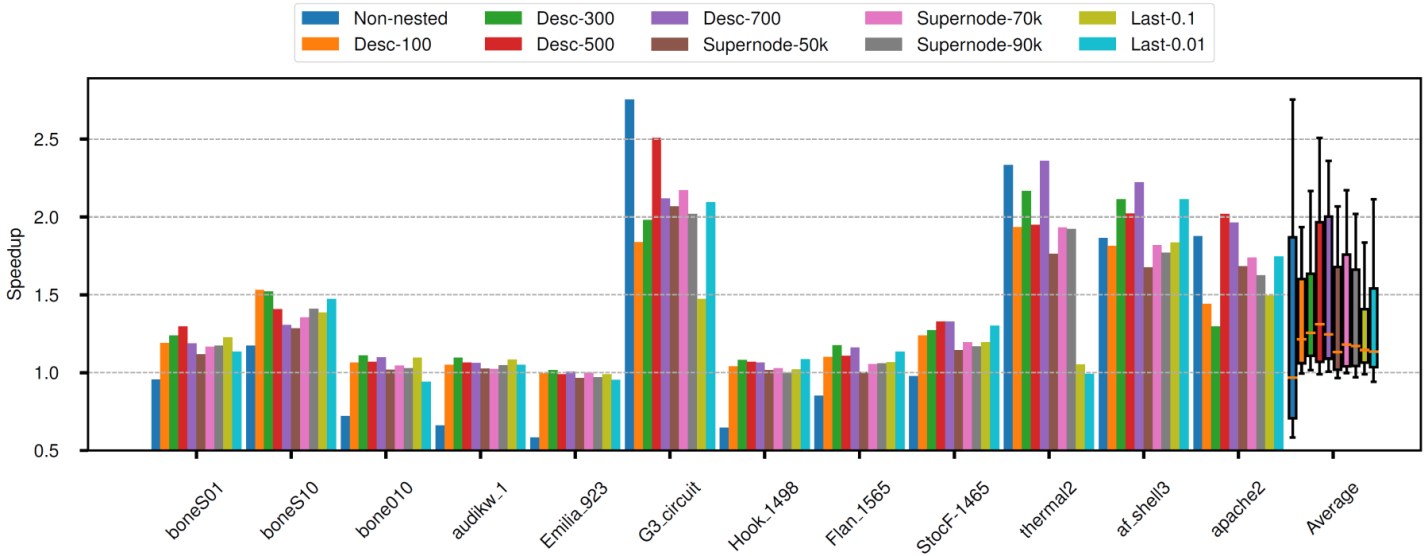






Numerical Libraries: Simple and efficient parallel codes

- We implement flexible schemes to dynamically control the amount of concurrency taking into account parallel workloads features.

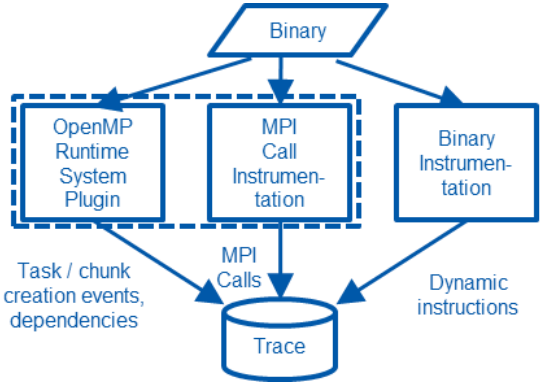


Multi-level Simulation Approach (MUSA)

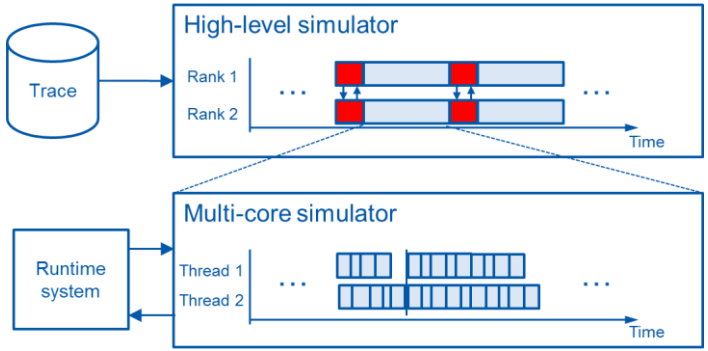
For further information please contact
marc.casas@bsc.es

Multi-level Simulation Approach (MUSA)

- Trace driven
  - High-level Trace
    - MPI events
    - OpenMP/OmpSs runtime system activity
  - Dynamic Instructions Trace
    - x86, Arm, RISC-V



- Simulation
  - MPI activity is simulated using the DIMEMAS model
  - Computation phases are simulated considering both
    - OpenMP/OmpSs runtime system activity
    - Shared-memory multi-core architecture
  - Simulation Speed: ~10MIPS, 100 times faster than gem5



Architecture Component	Architecture Parameters	Application Performance Analysis per HW Component
Vector Processing Unit (VPU)	Processing unit frequency Vector Unit Throughput Vector Register Size Vector Register File Size	Instruction Vectorization Stalled Cycles at the VPU Level Consumed Power at the VPU
Cache Hierarchy	# of Cache Levels Cache Storage Capacity Cache Associativity Cache Line Size Cache Replacement and Promotion	Hit/Miss Ratios Data Reuse at the Cache Write Backs Consumed Power per Cache
Main Memory	Memory Bandwidth # of in-flight requests Memory Controller	Memory Bandwidth Congestion # of in-flight requests Consumed Power at MM

From VPU to Memory

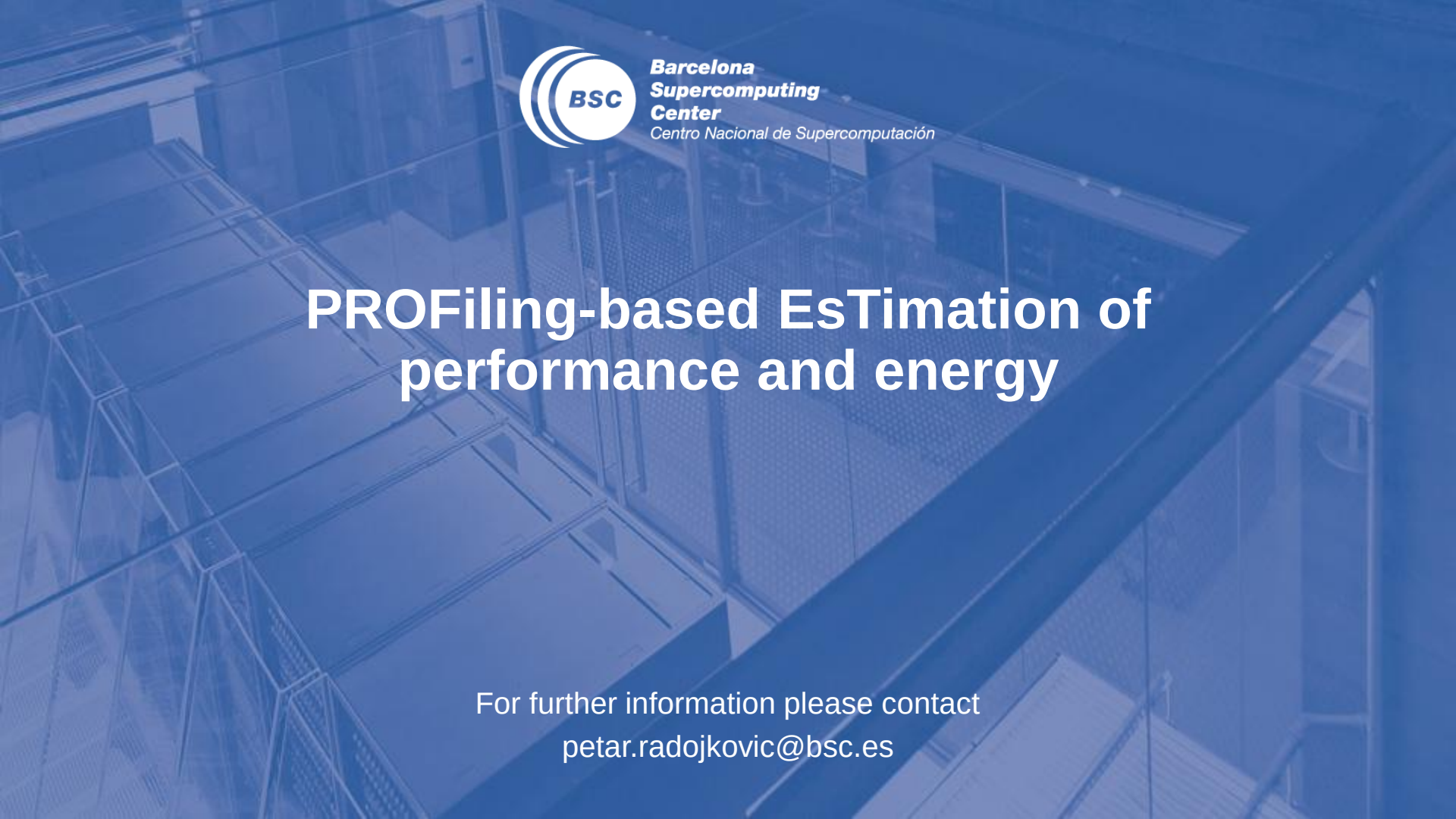
From HW to SW

From Coarse- to Fine-Grain Performance Analysis

Architecture Component	Architecture Parameters	Application Performance Analysis per HW Component
Vector Processing Unit (VPU)	Processing unit frequency Vector Unit Throughput Vector Register Size Vector Register File Size	Instruction Vectorization Stalled Cycles at the VPU Level Consumed Power at the VPU
Cache Hierarchy	# of Cache Levels Cache Storage Capacity Cache Associativity Cache Line Size Cache Replacement and Promotion	Hit/Miss Ratios Data Reuse at the Cache Write Backs Consumed Power per Cache
Main Memory	Memory Bandwidth # of in-flight requests Memory Controller	Memory Bandwidth Congestion # of in-flight requests Consumed Power at MM

Coarse-grain Application Performance Analysis
Execution Time
Power Consumption
Data Movement
Memory Access Pattern
Data Reuse

From HW to SW



PROFiling-based EsTimation of performance and energy

For further information please contact
petar.radojkovic@bsc.es

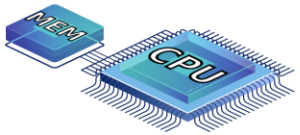
PROFET: PROFiling-based EsTimation of performance and energy

SIMULATE (MAIN) MEMORY

Problem

* System: CPUs + memory devices
* We have to simulate CPUs (although we would like to avoid it)

* Memory devices covered:
  - DRAMSim3
  - Ramulator



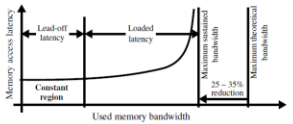
* Simulating CPUs - a lot of problems:
  - OOO mechanism
  - Prefetching
  - Obsolete CPU models
  - Time consuming

Let's think one more time

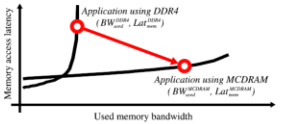
* Memory simulation: *memory access latency* = ***f****(memory system, memory traffic)*
* Our idea: Use memory bandwidth-latency curve

* CPU simulation: *performance* = ***f****(memory access latency)*
* Our idea: Can we profile the application running on an actual platform

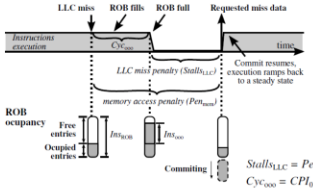
Bandwidth-latency curve showing how the mem. access latency depends on the used memory bandwidth



High-level view of the transition from DDR4 to high-bandwidth MCDRAM memory on the KNL platform



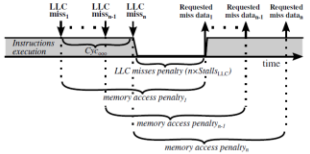
In OOO processors, LLC misses overlap with the execution of the instructions independent of the missing data



$Stalls_{LLC} = Pen_{mem} - Cyc_{ooo}$

$Cyc_{ooo} = CPI_0 \times Ins_{ooo}$

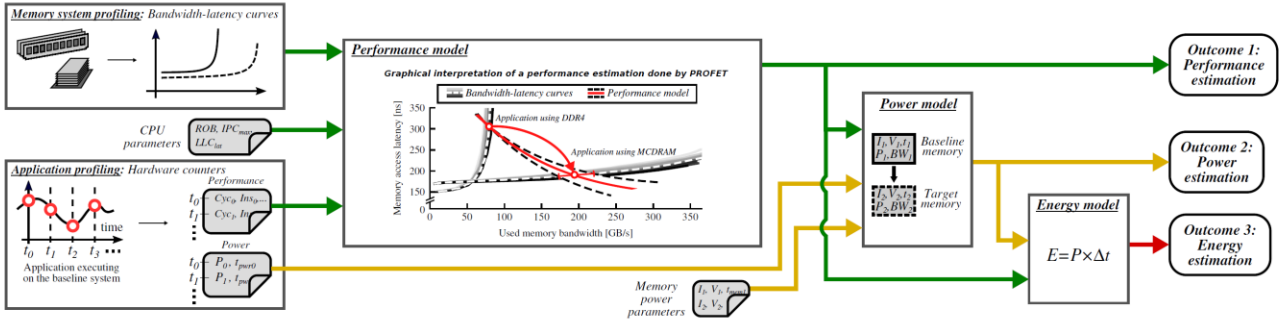
Overlapping LLC misses in an OOO processor: the penalty of a single miss is divided by a number of concurrent LLC misses



$Stalls_{LLC} = \frac{1}{MLP} \times (Pen_{mem} - Cyc_{ooo})$

$= \frac{1}{MLP} \times (Pen_{mem} - CPI_0 \times Ins_{ooo})$

PROFET: PROFiling-based EsTimation of performance and energy



MODEL EVALUATION

Experimental environment

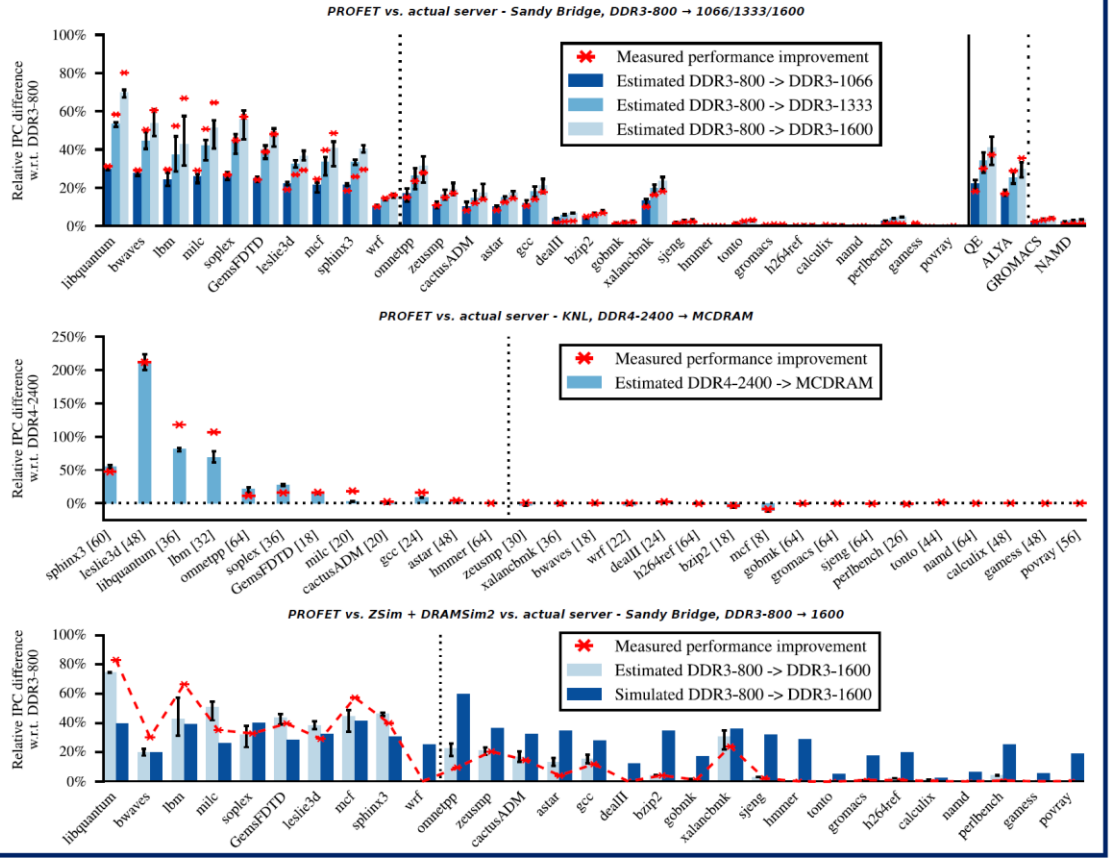
* Sandy Bridge-EP E5-2670: DDR3-800/1066/1333/1600
* Knights Landing Xeon Phi 7230: DDR4-2400 + MCDRAM
* Comparing to simulations: ZSim + DRAMSim2
* SPEC CPU2006 + 4 UEABS HPC applications

Conclusions

* Sandy Bridge evaluations:
  - Narrow estimation ranges
  - Average difference from measured values:
    1.8%, 3.8% and 5.1% for the high-bandwidth benchmarks,
    1%, 1.3% and 1.6% over the low-bandwidth benchmarks
  - Average error of power and energy estimations below 3%

* Knights Landing evaluations:
  - Narrow estimation ranges
  - Average difference from measured values:
    7% for the high-bandwidth benchmarks,
    1.6% over the low-bandwidth benchmarks

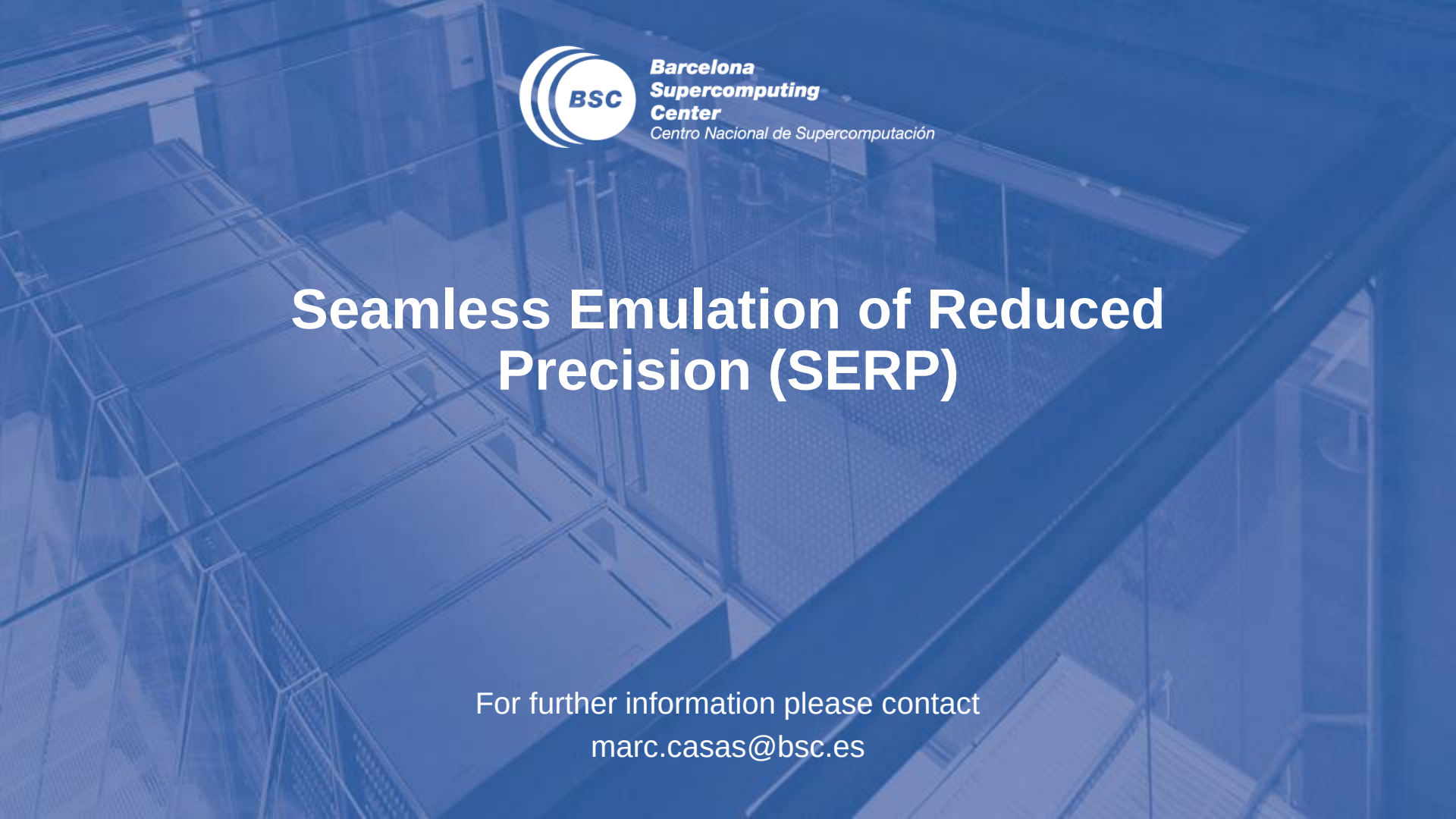
* PROFET vs. ZSim + DRAMSim2 evaluations:
  - PROFET prediction follows the trend of the actual measurements
  - Average difference from measured values:
    PROFET 3.6%
    Simulator 15.7%
  - PROFET is three orders of magnitude faster than ZSim+DRAMSim2



FURTHER READING

PROFET: Modeling System Performance and Energy Without Simulating the CPU
Proceedings of the ACM on Measurement and Analysis of Computing Systems - SIGMETRICS, Article 34 (June 2019)
Milan Radulovic, Rommel Sánchez Verdejo, Paul Carpenter, Petar Radojković, Bruce Jacob, and Eduard Ayguadé

PROFET source code with examples: **https://github.com/bsc-mem/PROFET** (BSD-3 license)

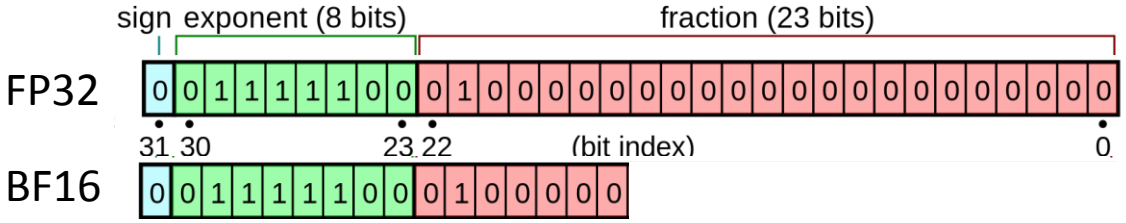


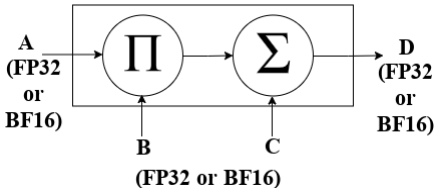
Seamless Emulation of Reduced Precision (SERP)

For further information please contact
marc.casas@bsc.es

Seamless Emulation of Reduced Precision (SERP)

- SERP: A binary analysis tool based on PIN to instrument and analyze scientific workloads.
  - SERP intercepts all instructions executed by these frameworks.
  - SERP rounds the operands of some Floating-Point32 (FP32) instructions to BFloat16 (BF16) using the Rounding to Nearest Even (RNE) algorithm.
  - The cost of operands rounding is reduced by:
    - using vectorization intrinsics.
    - avoid redundant rounding of instructions of the same Basic Block.

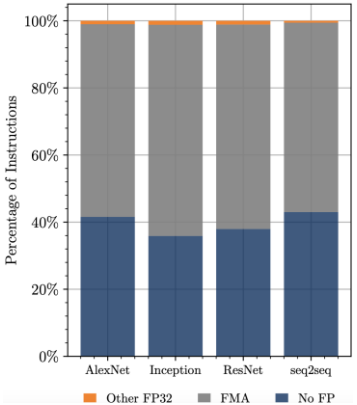




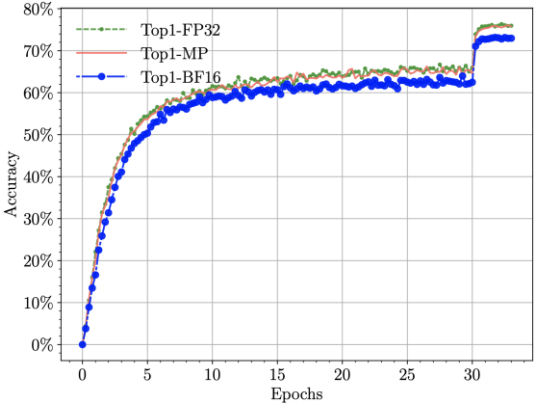
$$D = A \cdot B + C$$

SERP allows precise accuracy analysis

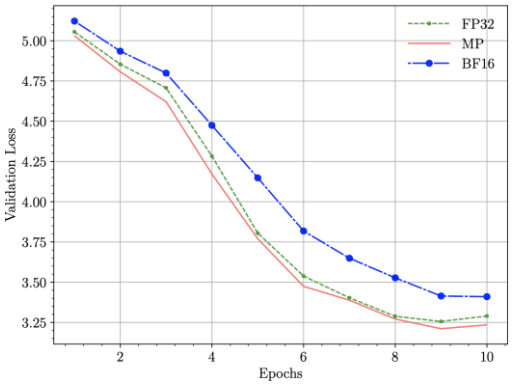
- SERP enables the analysis of reduced-precision formats (E.g. BFloat16) on relevant workloads like DNN training.
- SERP can be easily combined with PIN-based tools like Sniper to provide performance estimations.



Instruction Breakdown



Static Techniques on ResNet-50



Static Techniques on *seq2seq*

High-resolution Deep Learning

For further information please contact
dario.garcia@bsc.es

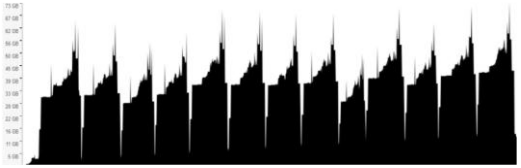
High-resolution Deep Learning

- **High-Resolution (HR)** image processing is a matter of life and death (medical imaging, autonomous driving, etc.)
- **Variable-Shaped (VS)** image processing is how real world data works (each device generates images at different resolution and aspect ratio)
- **Current DL Software is inadequate** (tensor size limits, batch size limitations, batch normalization inconsistencies, excessive padding, etc.)
- **Current DL Hardware is inadequate** (limited GPU memories, poor CPU efficiency, benchmarks based on low-resolution tasks, etc.)

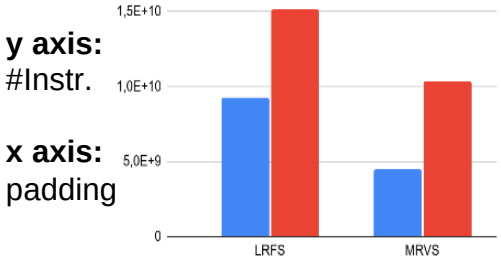
Insights into HR-VS Deep Learning

- We need a *SW* benchmark to assess methods
  - **MAMe dataset** (37K HR-VS artwork images)
  - `https://hpai.bsc.es/MAMe-dataset/`
  - Baseline results: HR > LR

- We need a *HW* benchmark to assess performance
  - **Low** res. vs **Mid**. res. vs **High** res.
  - Memory needs vary up to 40% among batches
  - Num. instructions goes down with padding
  - Some architectures do best in HR, some in LR
  - Benchmark to be released soon



y axis: Memory **x axis:** batches

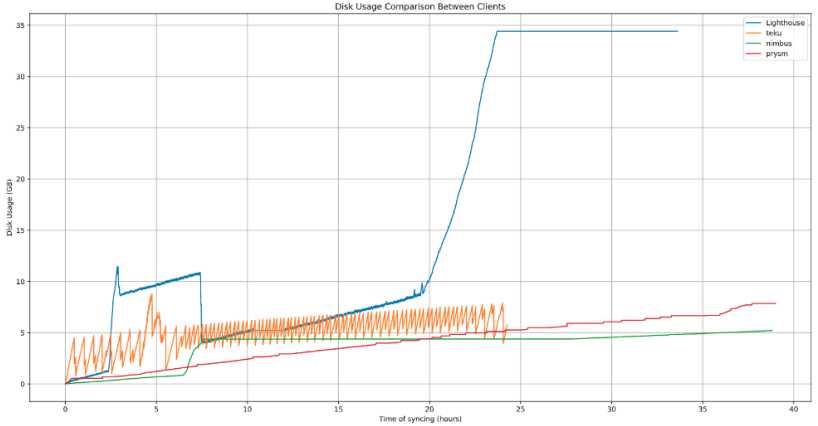


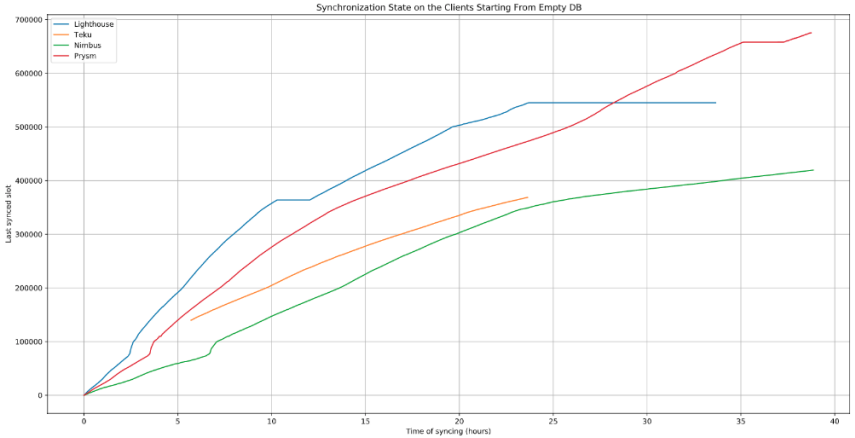
Blockchain research

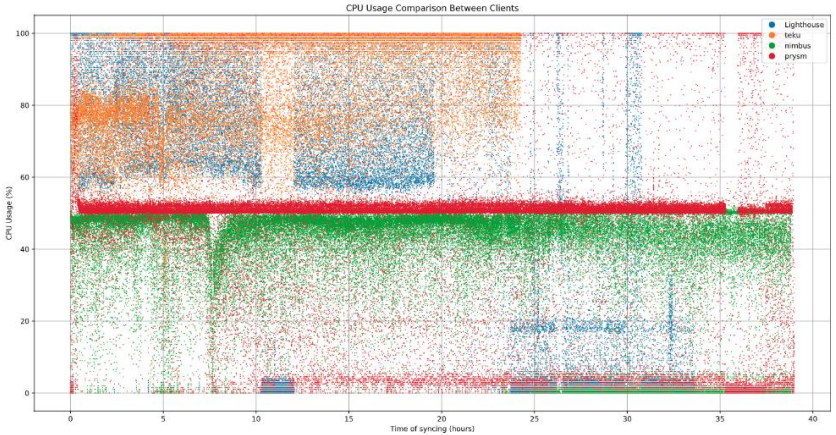
For further information please contact
leonardo.bautista@bsc.es

Blockchain Research - ETH2 Monitor

- Ethereum 2.0 – Sharding
- Testing network robustness
- Monitoring clients in testnet
- Chain Data analytics
- Blockchain Simulations







Barcelona Supercomputing Center
Centro Nacional de Supercomputación

www.bsc.es