Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Universitat Politècnica de Catalunya
BARCELONA TECH

OpenMP tasking model: from the standard to the classroom

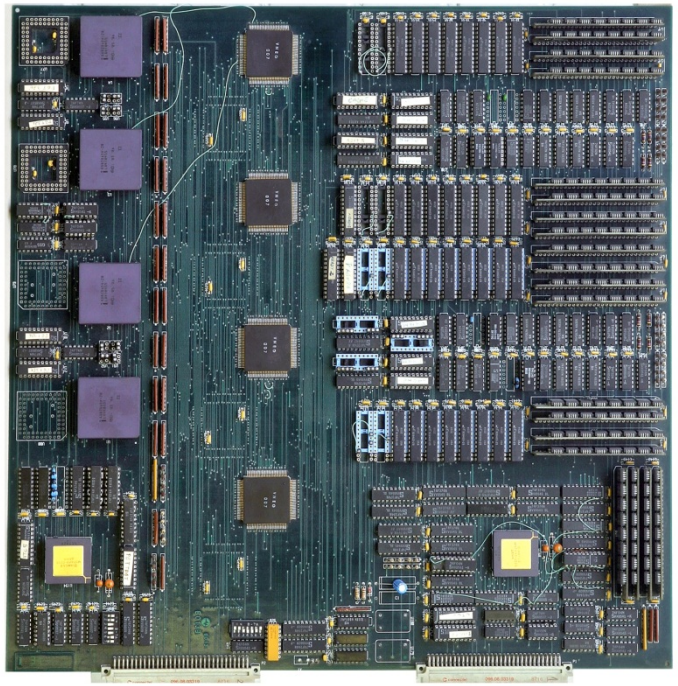
Eduard Ayguadé

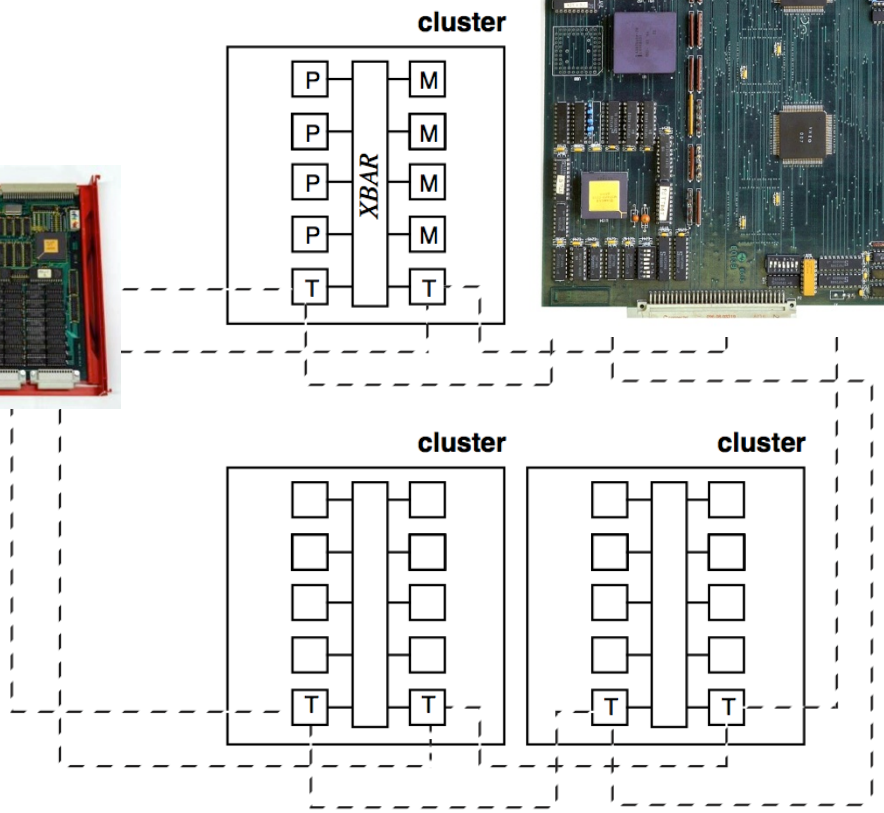
Dep. Arquitectura de Computadors, U. Politècnica de Catalunya
Computer Sciences Dep., Barcelona Supercomputing Center

EXCELENCIA
SEVERO
OCHOA

Madrid. March 30, 2017

ICT386 multiprocessor architecture, late 80s





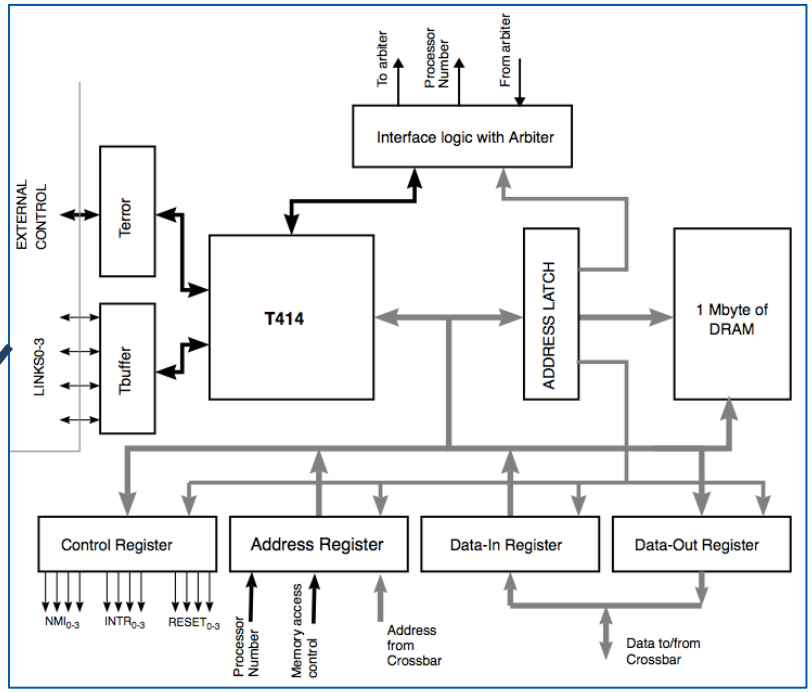
4 x CPU module
- i386/i387
- no private cache memory

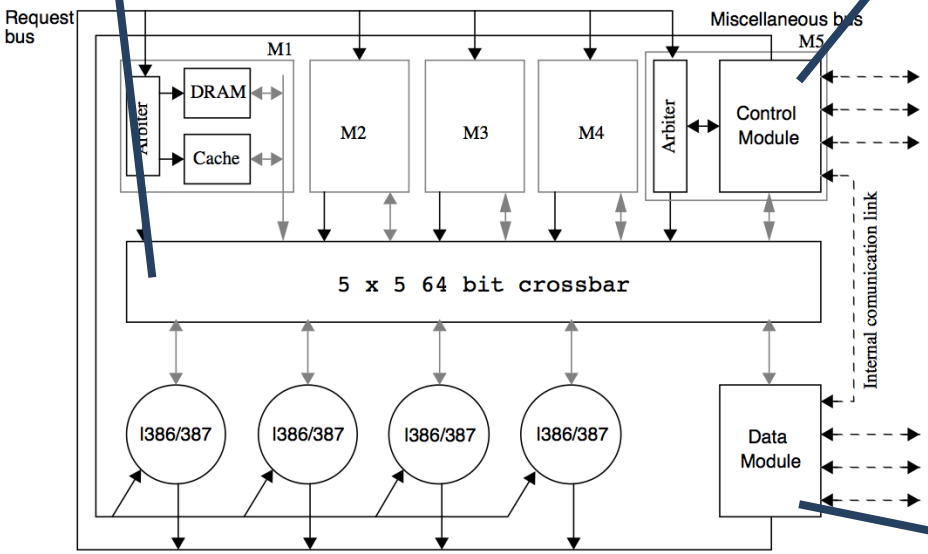
4 x Memory module
- 4 MB DRAM (100 ns)
- 64 KB **shared cache**
- 25 ns, direct-mapped

ICT386 multiprocessor architecture, late 80s

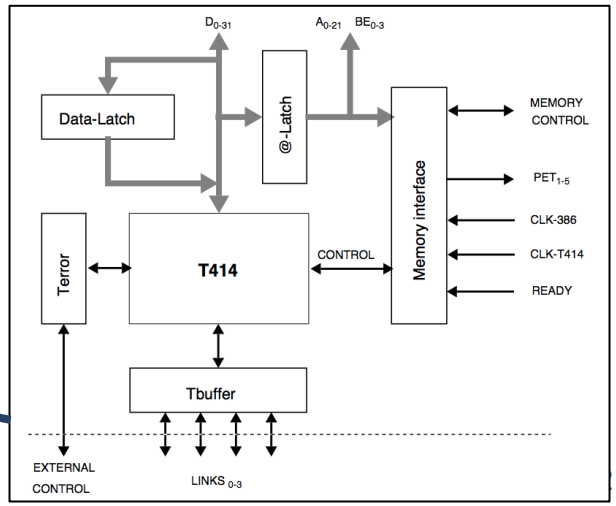
Crossbar 5x5
Gate array design
10 ns transfer

Intellligent memory
Memory mapped
1 MB DRAM
Functions: management, **synchronization and scheduling**





Programmable NIC

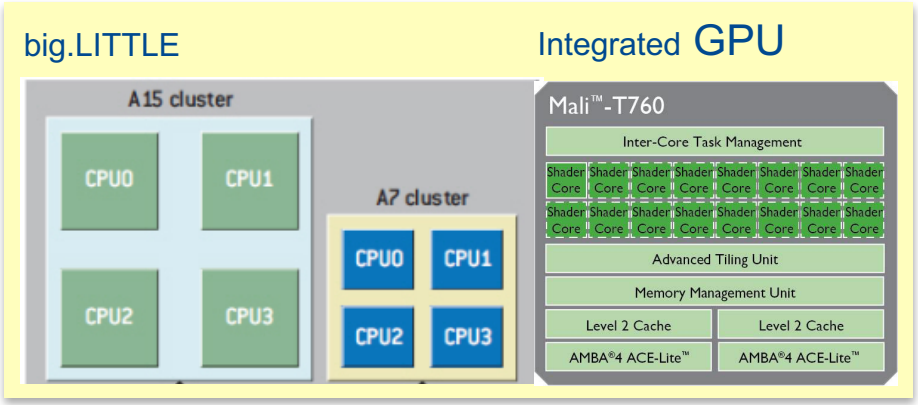


And the wall has become higher and higher …

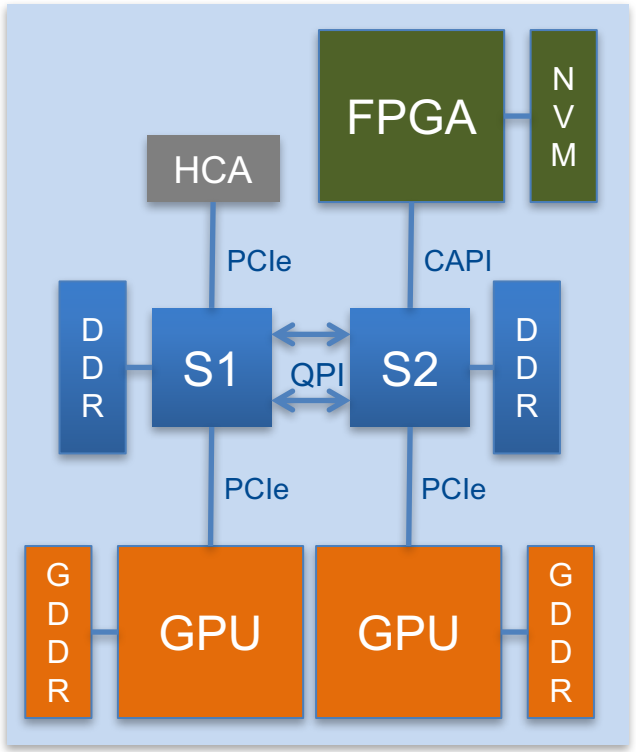
Rank	Name	Site	Computer	Total Cores	Accelerator/ Co-Proc. Cores
1	Sunway TaihuLight	National Supercomputing Center in Wuxi	Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway	10649600	0
2	Tianhe-2 (MilkyWay-2)	National Super Computer Center in Guangzhou	TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P	3120000	2736000
3	Titan	DOE/SC/Oak Ridge National Laboratory	Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x	560640	261632
4	Sequoia	DOE/NNSA/LLNL	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	1572864	0
5	Cori	DOE/SC/LBNL/NERSC	Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect	622336	0

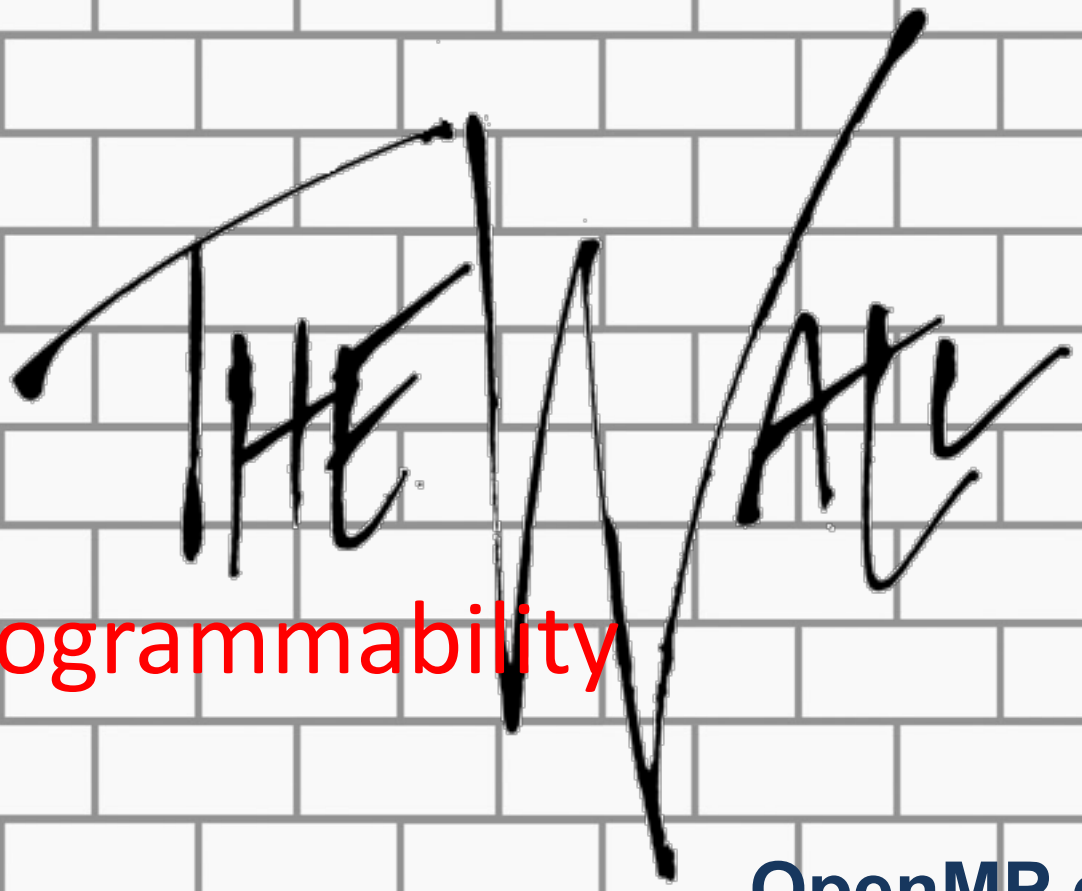
Top5 November 2016 list

Socket



Node





Programmability

OpenMP evolution and OmpSs influence

Runtime opportunities

Architecture opportunities

Teaching opportunities

1979 Pink Floyd's The Wall album,
Cray-1 delivering 80 Mflops, single processor

The vision

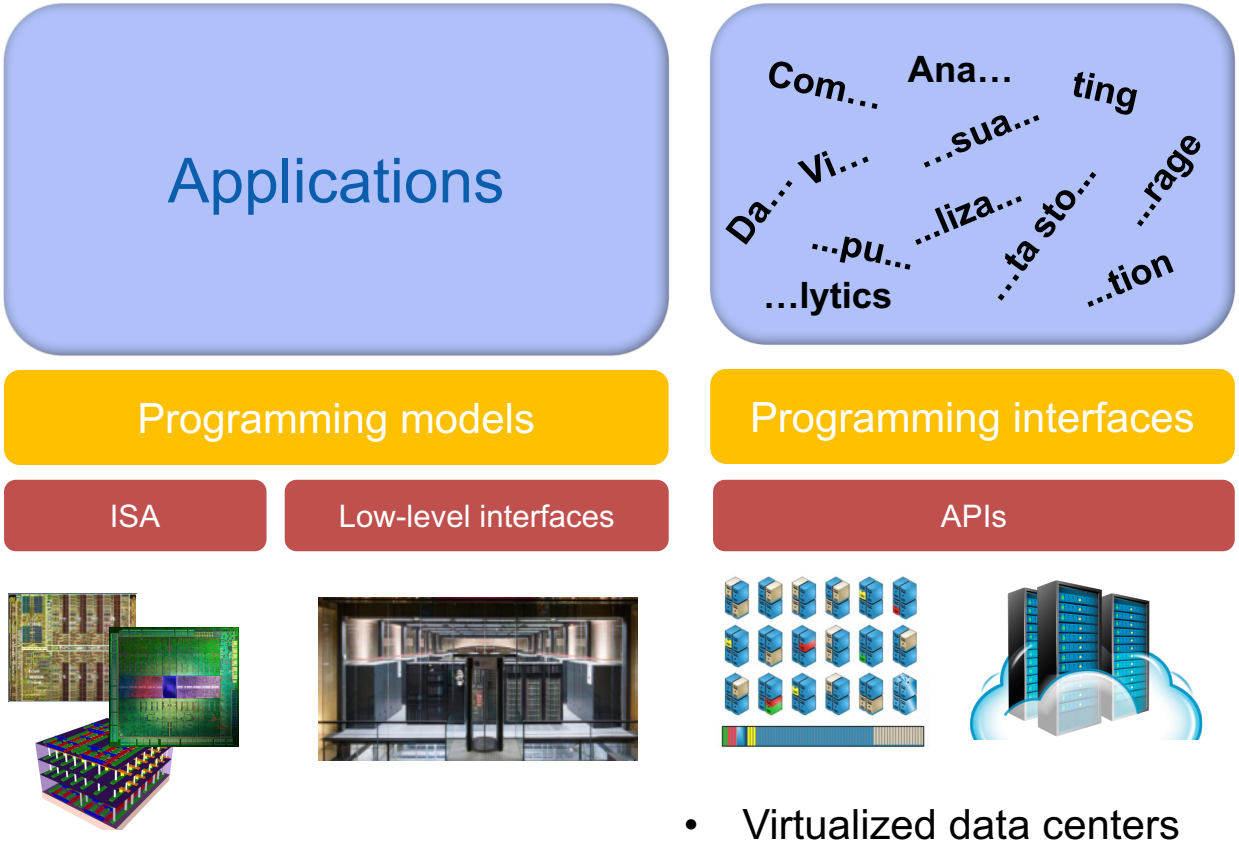
New **application domains**

Programmability and **porting/inter-operability**
- Legacy codes
- ”Legacy” mental models and habits

ISA/PM **leaks** and **variability/faults/errors** everywhere
- Programmer exposed
- Divergence between mental models and actual behavior

Plethora of architectures
- Heterogeneity
- Hierarchies

New **usage practices**
- Analytics and visualization
- New data models
- Interactive supercomputing, response time



- Virtualized data centers
- Cloud

The vision

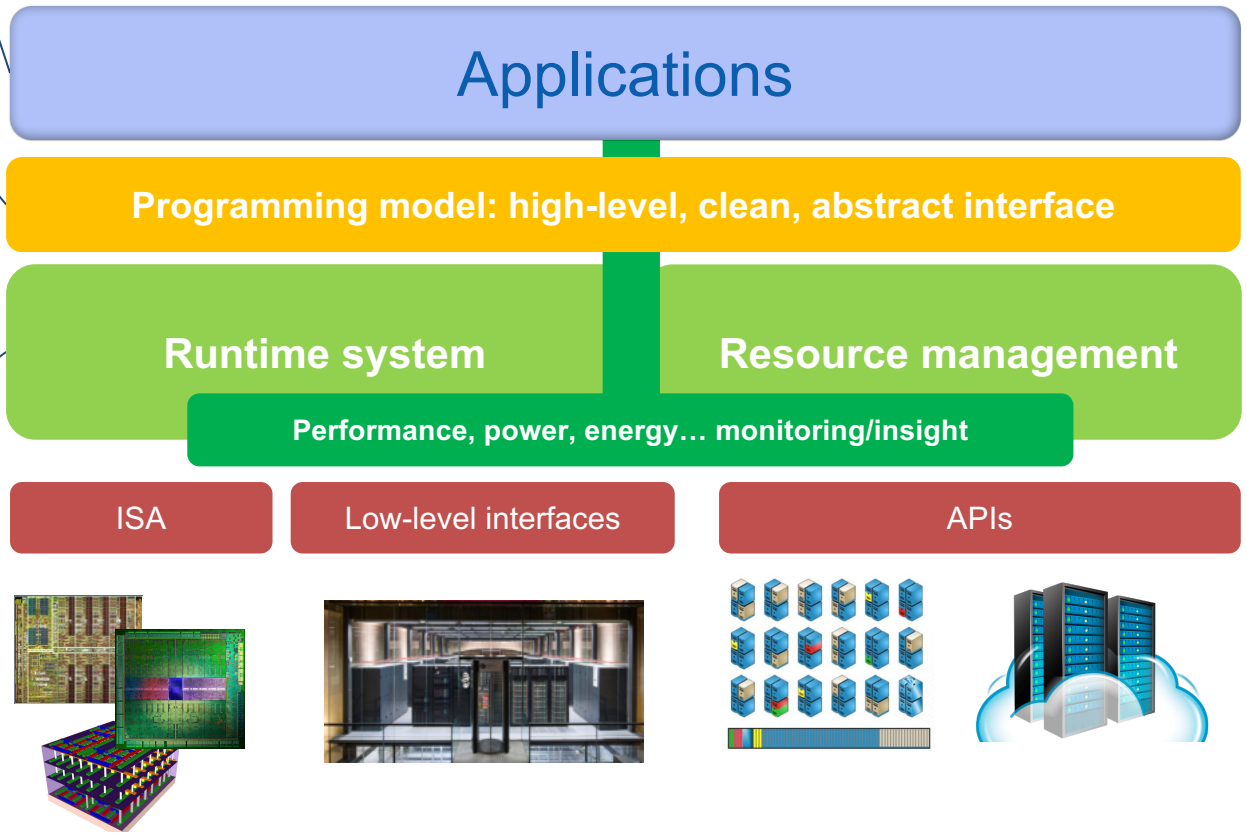
Program logic independent of computing platform

General purpose
Task based
Single address space

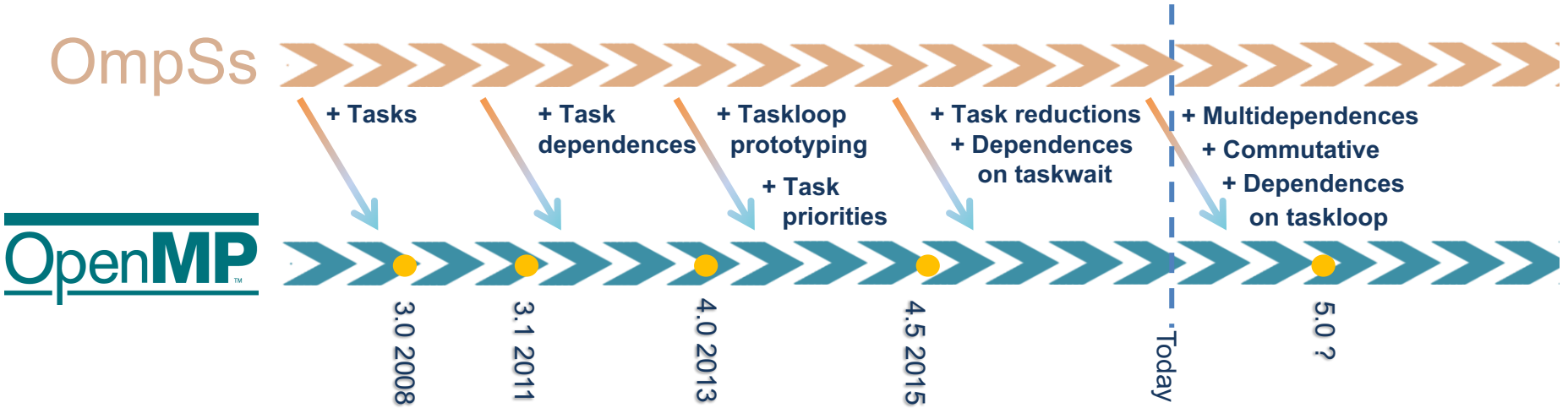
Intelligent runtime, parallelization, distribution, interoperability

Resource management

Performance- and power-driven optimization/tuning



The OmpSs forerunner for OpenMP



- Proposing ideas …
  - IWOMP workshop is the ideal scenario
  - Participation in the OpenMP language committee and sub-committees
- Demonstrated with …
  - Prototype implementation: Mercurium compiler and Nanos runtime
  - Benchmark/application suite
- Lots of discussions and final polishing

OmpSs influencing tasking model evolution (1)

- The big change …

task-based model, nesting of tasks (3.0)

```
#pragma omp task firstprivate(list) final(expr) priority(value)
{ code block }
```

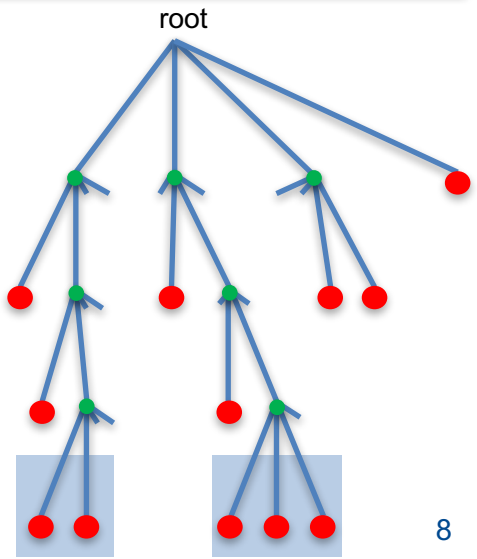
task generation cut-off (3.1)

task priorities (4.5)

```
#pragma omp taskwait
```

task generation waits for child tasks to finish (3.0)

- From loop-based model to task-based model
  - Dynamically allocated data structures (lists, trees, …)
  - Recursion

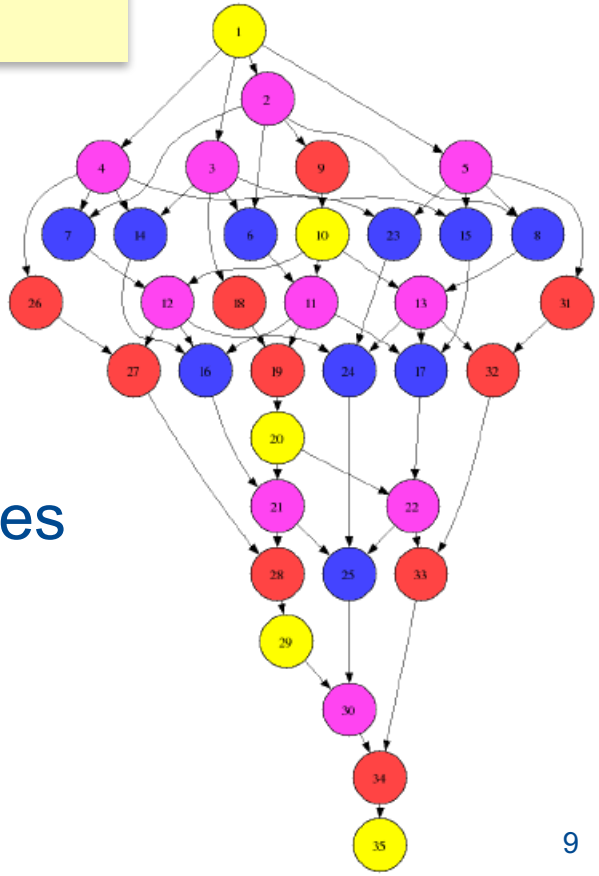


OmpSs influencing tasking model evolution (2)

- Directionality of task arguments

directionality for task arguments (4.0)

```
#pragma omp task depend(in: list, out: list, inout: list)
{ code block }
```



- Dependences between tasks computed at runtime
- Tasks executed as soon as all dependences are satisfied

OmpSs influencing tasking model evolution (3)

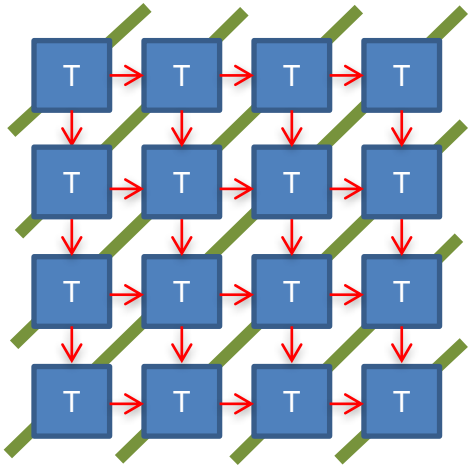
- Tasks become chunks of iterations of one or more associated loops

```
#pragma omp taskloop [num_tasks(value) | grainsize(value)]
   for-loops
```

tasks out of loops and granularity control (4.5)

- Dependences between tasks generated in the same or different taskloop construct under discussion

```
#pragma omp taskloop block(2) grainsize(B, B)
        depend(in : block[i-1][j], block[i][j-1])
        depend(in : block[i+1][j], block[i][j+1])
        depend(out: block[i][j])
for (i=1; i<n i++)
    for (j=1; j<n;j++)
        foo(i, j);
```



OmpSs influencing tasking model evolution (4)

- Reduction operations on task and taskloop not yet supported, currently in discussion

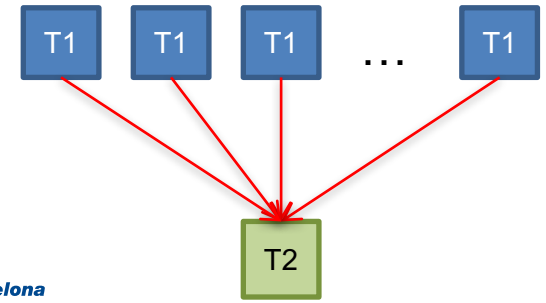
```
#pragma omp taskgroup reduction(+: res)
{
  while (node) {
    #pragma omp task in_reduction(+: res)
    res += node->value;

    node = node->next;
  }
}
```

```
#pragma omp taskloop grainsize(BS) reduction(+: res)
for (i = 0; i < N; ++i) {
  res += foo(v[i]);
}
```

OmpSs influencing tasking model evolution (5)

- The number of task dependences have to be constant at compile time → multidependences proposal under discussion

```
for (i = 0; i < l.size(); ++i) {
  #pragma omp task depend(inout: l[i])                    // T1
  { ... }
}

#pragma omp task depend(in: {l[j], j=0:l.size()}) // T2
for (i = 0; i < l.size(); ++i) {
  ...
}
```

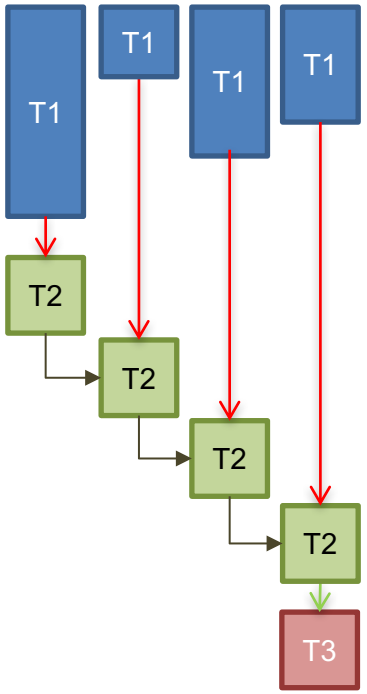


Defines an iterator with a range of values that only exists in the context of the directive. Equivalent to:
`depend(in: l[0], l[1], ..., l[l.size()-1])`

OmpSs influencing tasking model evolution (6)

- New dependence type that relaxes the execution order of commutative tasks but keeping mutual exclusion between them

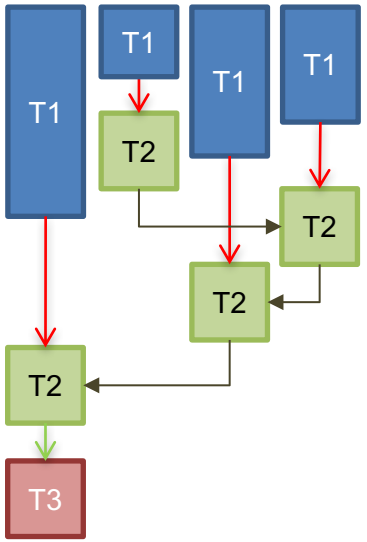
```
for (i = 0; i < l.size(); ++i) {
  #pragma omp task depend(out: l[i])
  // T1
}

for (i = 0; i < l.size(); ++i) {
   #pragma omp task shared(x) depend(in: l[i])
                              depend(inout: x)
   // T2
}

#pragma omp task shared(x) depend(in: x)          // T3
for (i = 0; i < l.size(); ++i) {
  ...
}
```



OmpSs influencing tasking model evolution (6)

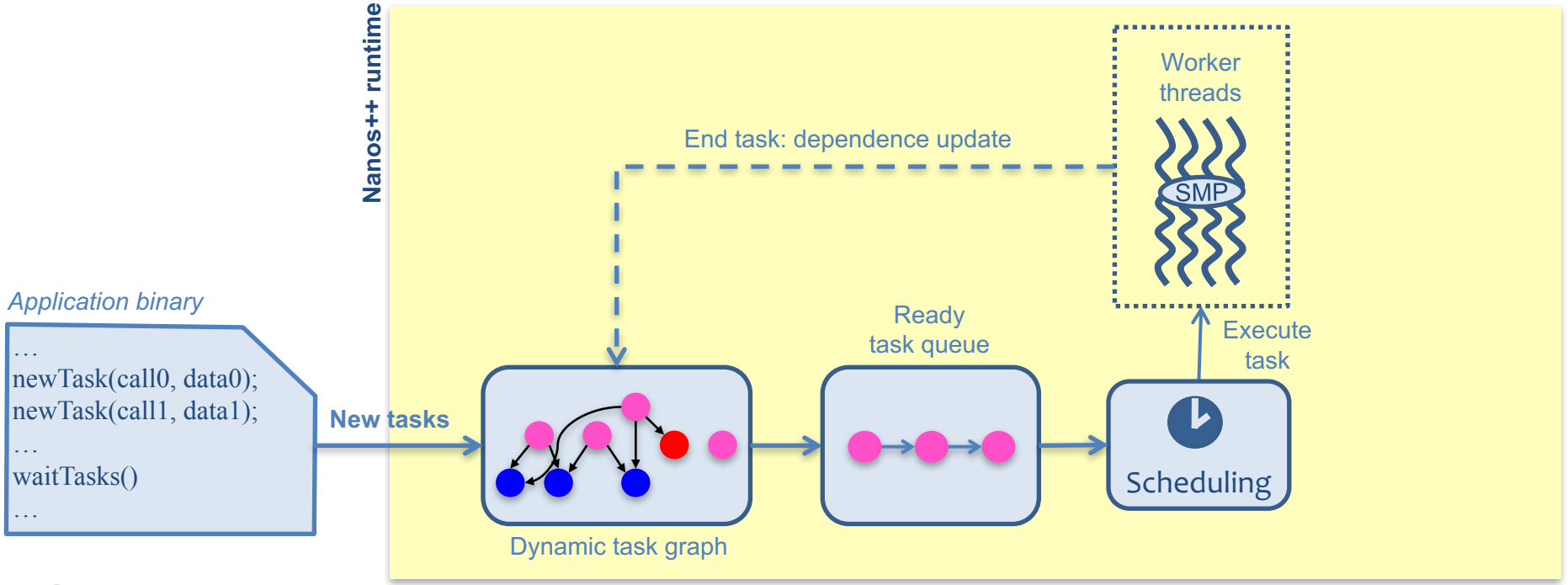
- New dependence type that relaxes the execution order of commutative tasks but keeping mutual exclusion between them

```
for (i = 0; i < l.size(); ++i) {
  #pragma omp task depend(out: l[i])
  // T1
}

for (i = 0; i < l.size(); ++i) {
   #pragma omp task shared(x) depend(in: l[i])
                              depend(commutative: x)
   // T2
}

#pragma omp task shared(x) depend(in: x)          // T3
for (i = 0; i < l.size(); ++i) {
  ...
}
```



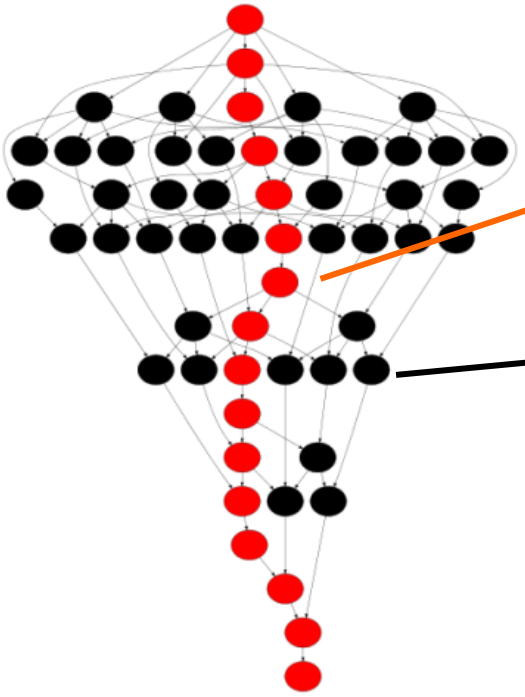
OmpSs: runtime opportunities (1)

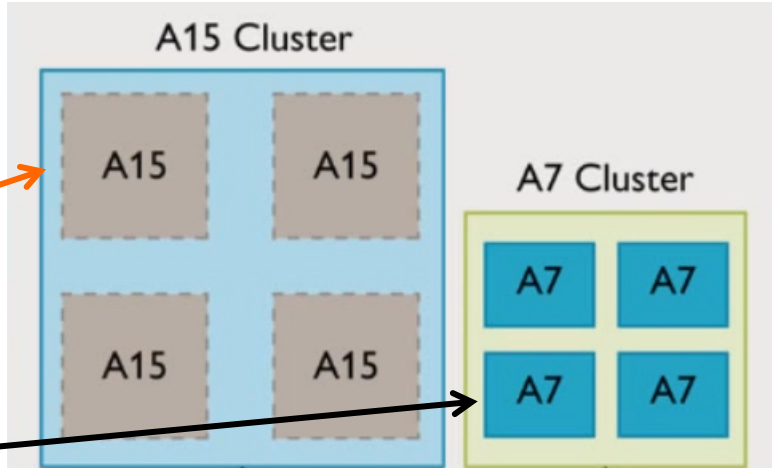
- Runtime parallelization
  - Compute data dependences among tasks → dynamic task graph
  - Dataflow execution of tasks → ready task queue
  - Look-ahead: task instantiation can proceed past non-ready tasks (with opportunities to find distant parallelism)

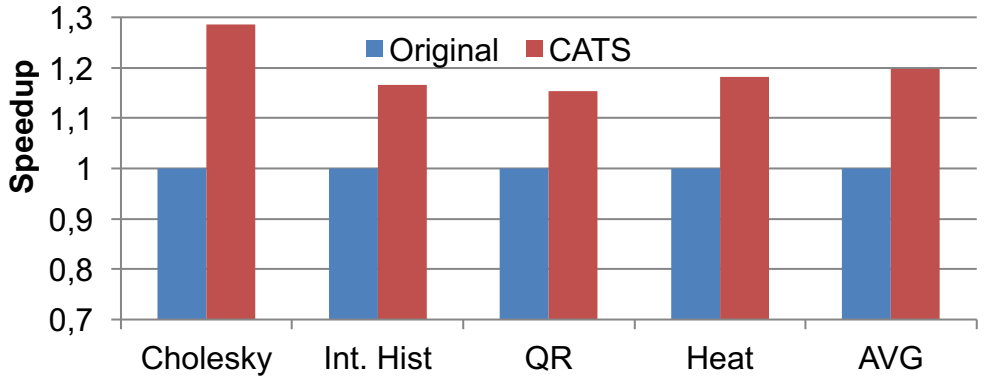


OmpSs: runtime opportunities (2)

- Criticallity-aware scheduler: managing heterogeneous SoC
  - Tasks in the critical path detected at runtime → execute in faster cores
  - Non-critical tasks executed in slower, more power-efficient cores
  - Work-stealing for load balancing



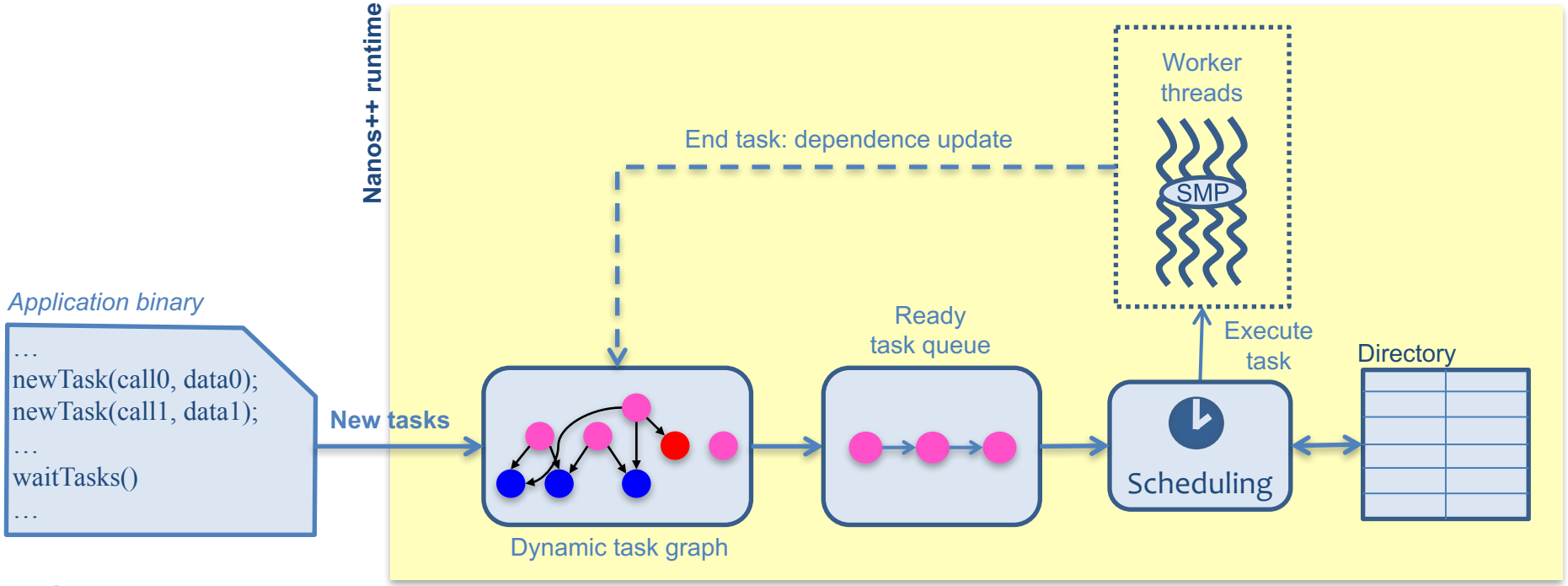




"Criticality-Aware Dynamic Task Scheduling on Heterogeneous Architectures". K. Chronaki et al. ICS-2015

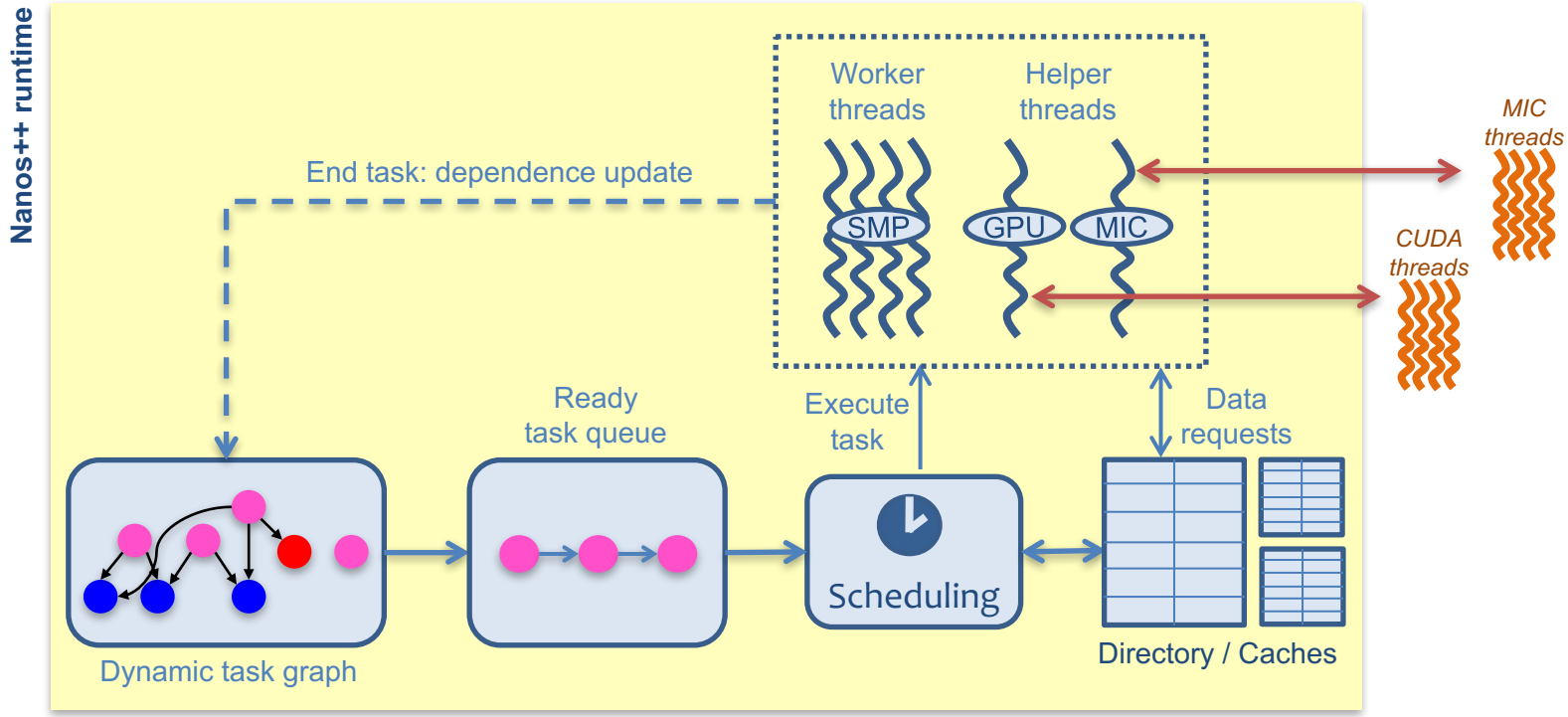
OmpSs: runtime opportunities (3)

- Runtime parallelization
- Directory with per-core arguments utilization
  - Locality-aware task scheduling strategies
  - Driving argument prefetching



OmpSs: runtime opportunities (4)

- Management of address spaces and coherence
  - Directory@master: identifies for every region reference where data is
  - Per-device software cache: translate host to device address spaces
  - Use of device specific mechanisms to move data, if needed
  - Locality-aware scheduling when multiple devices exist



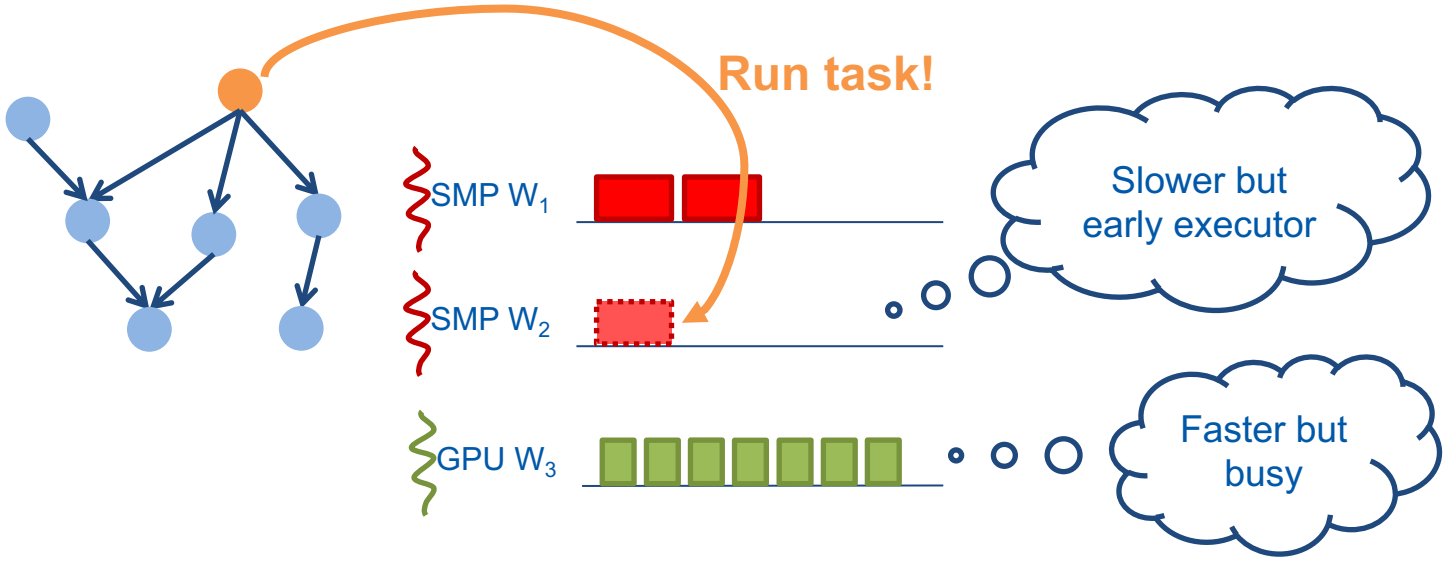
"Self-Adaptive OmpSs Tasks in Heterogeneous Environments". J. Planas et al., IPDPS 2013.

Example: nbody

```
#pragma omp target device(smp) copy_deps
#pragma omp task depend(out: [size]out), in: [npart]part)
void comp_force (int size, float time,  int npart, Part* part, Particle *out, int gid);

#pragma omp target device(fpga) ndrange(1,size,128) copy_deps implements (comp_force)
#pragma omp task depend(out: [size]out), in: [npart]part)
__kernel void comp_force_opencl (int size, float time,  int npart, __global Part* part,
                                                    __global Part* out, int gid);

#pragma omp target device(cuda) ndrange(1,size,128) copy_deps implements (comp_force)
#pragma omp task depend(out: [size]out), in: [npart]part)
__global__ void comp_force_cuda (int size, float time,  int npar, Part* part, Particle *out, int gid);



void Particle_array_calculate_forces(Particle* input, Particle *output, int npart, float time) {
   for (int i = 0; i < npart; i += BS )
        comp_force (BS, time, npart, input, &output[i], i);
}
```

OmpSs: runtime opportunities (5)

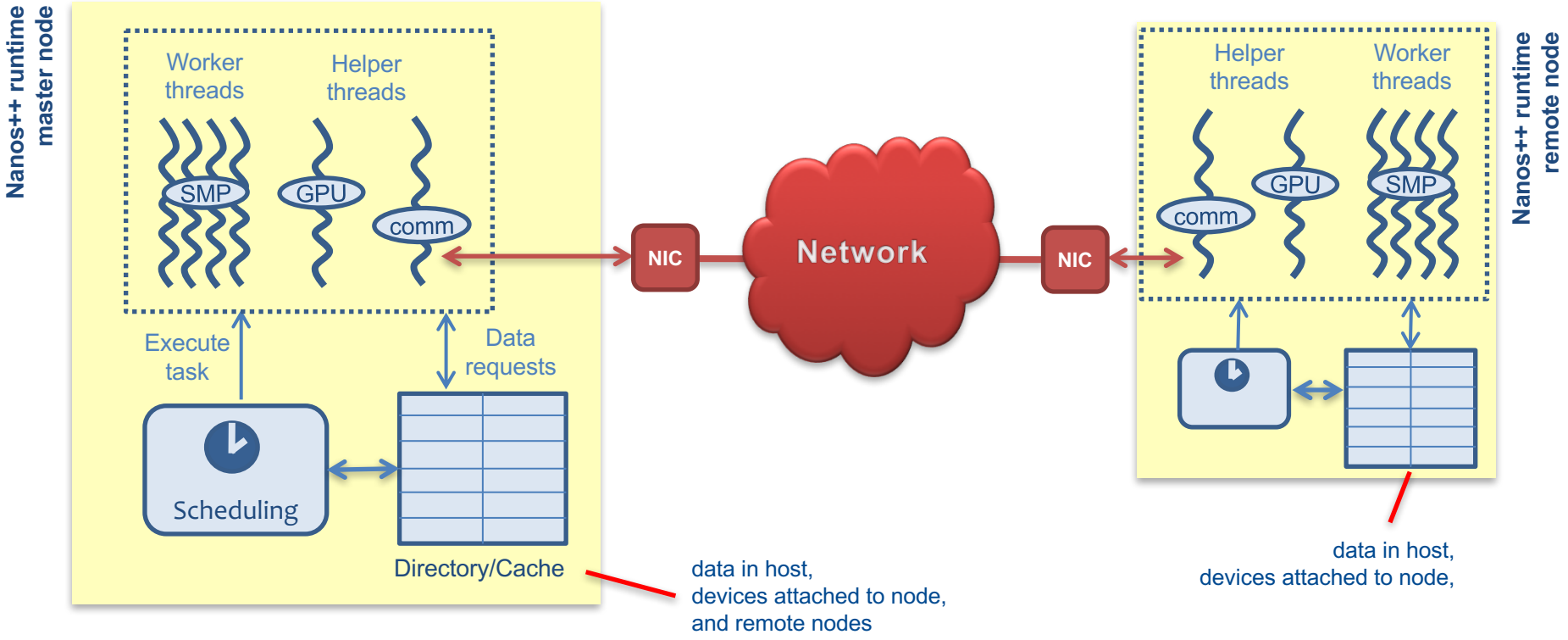
- Support for multiple implementations for same task
  - Versioning scheduler:
    - Monitoring (learning) phase and scheduling phase
    - Most suitable implementation is selected each time depending on monitoring, simple model and OmpSs worker's work load



"Self-Adaptive OmpSs Tasks in Heterogeneous Environments". J. Planas et al., IPDPS 2013.

OmpSs: runtime opportunities (6)

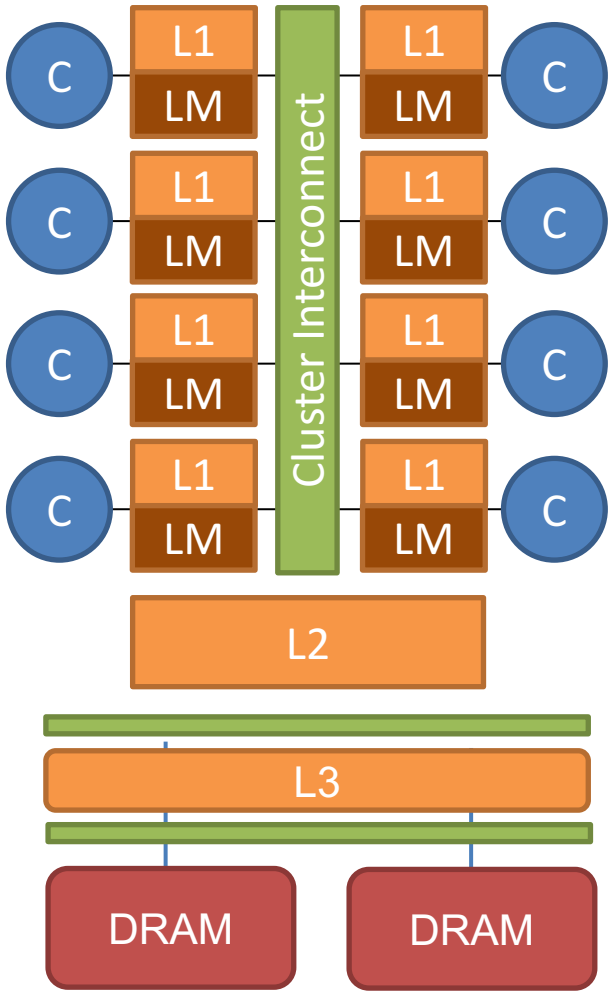
- At cluster …
  - Task creation starts at master node, outermost tasks may be executed on remote nodes and spawn inner tasks
  - Locality-aware scheduler, overlap transfers/computation, work stealing



"Productive Cluster Programming with OmpSs". J. Bueno et al. EuroPar 2011.
"Productive Programming of GPU Clusters with OmpSs". J. Bueno et al., IPDPS 2012.

OmpSs: architecture opportunities (1)

- Novel hybrid cache/local memory
  - Cache: irregular accesses
  - Local memory (energy efficient, less coherence traffic): strided accesses
- Runtime-assisted prefetching
  - Task inputs and outputs mapped to the LMs
  - Overlap with runtime
  - Double buffering between tasks
- 8.7% speedup, 14% reduction in power, 20% reduction in network-on-chip traffic



"Runtime-Guided Management of Scratchpad Memories in Multicores." Ll. Alvarez et al. Ç PACT-2015

OmpSs: architecture opportunities (3)

- Task dependence manager to speedup task management and dependence tracking

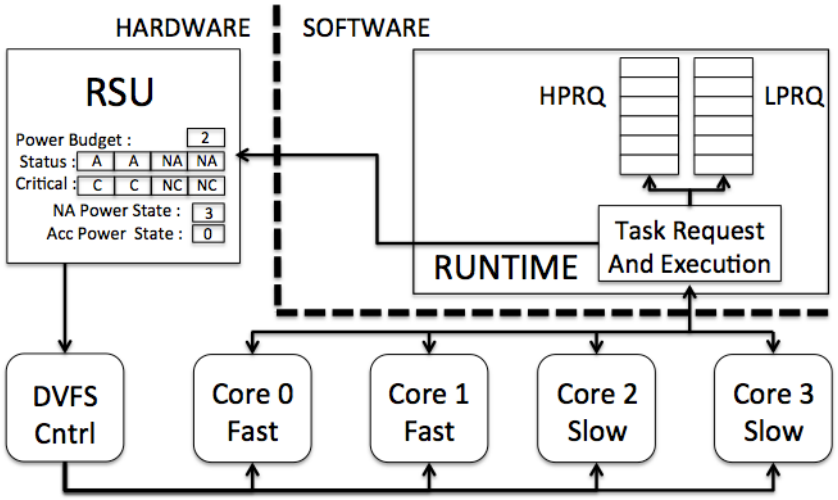
  X. Tan et al. Performance Analysis of a Hardware Accelerator of Dependency Management for Task-based Dataflow Programming models. ISPASS 2016.
  X. Tan et al. General Purpose Task-Dependence Management Hardware for Task-based Dataflow Programming Models. IPDPS 2017.

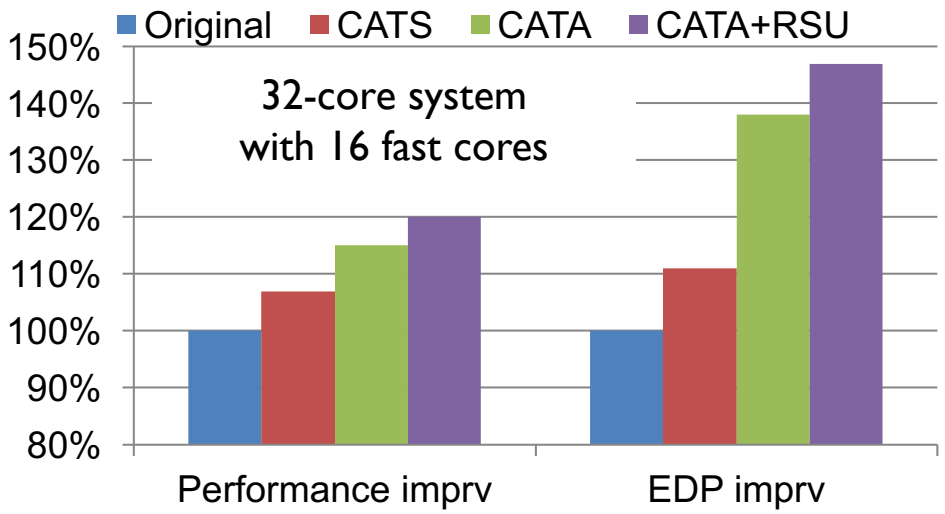
- Hardware support for reductions
  - Privatization of variables to speedup reduction → Significant overhead when performing reductions on a vector or matrix block.
  - HW support to perform simple reduction operations (+, -, *, max, min) in the last-level cache or in memory → In-memory computation

  J. Ciesko et al. Supporting Adaptive Privatization Techniques for Irregular Array Reductions in Task-parallel Programming Models. IWOMP 2016.

OmpSs: architecture opportunities (2)

- CATA: criticallity-aware task acceleration
  - Runtime drives per-core DVFS reconfigurations meeting a global power budget
  - Hardware Runtime Support Unit (RSU) to minimize overheads that grow with the number of cores



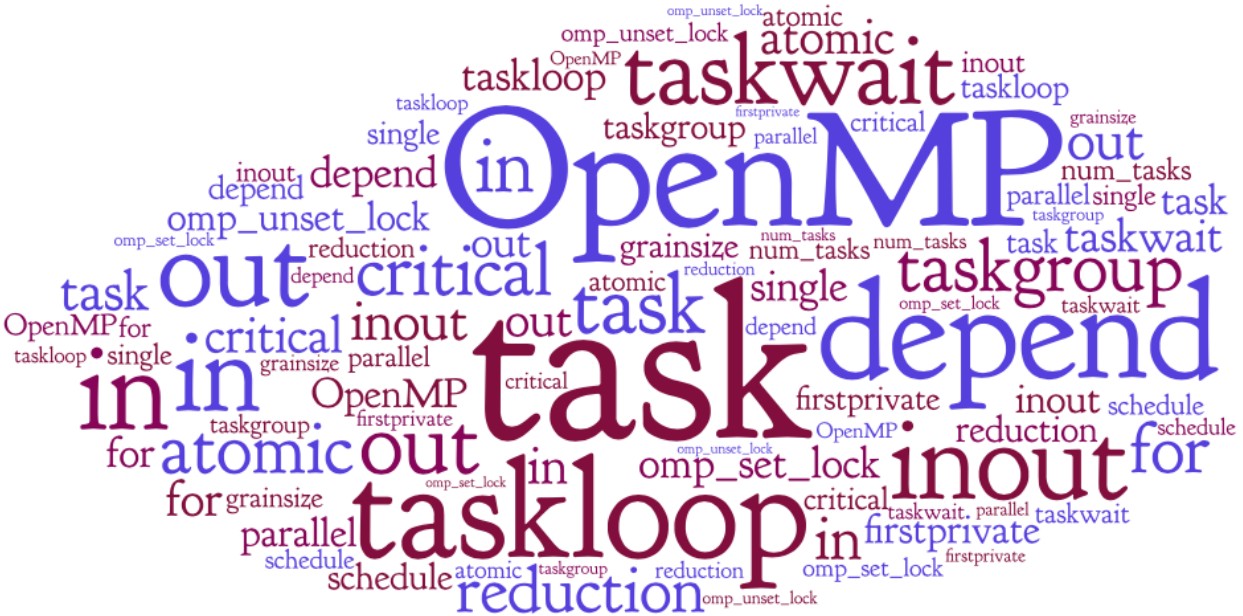


E. Castillo et al., CATA: Criticality Aware Task Acceleration for Multicore Processors. IPDPS 2016.

Teaching opportunities (1)

- Two courses in Computer Science BSc @ FIB (UPC)
  - PAR – Parallelism: 5th semester, mandatory course
  - PAP – Parallel Architectures and Programming: optional, computer engineering specialization

- OpenMP task-based programming @ node level



Teaching opportunities (2)

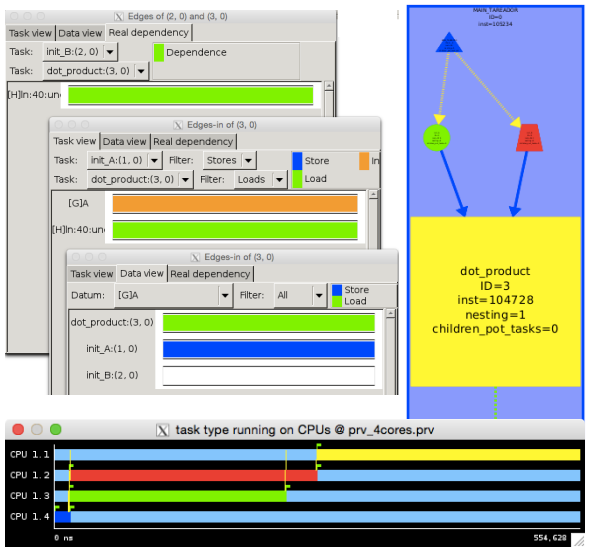
Syllabus

- Tasks, task graph, parallelism
- Overheads and model: task creation, synchronization, data sharing
- Task decomposition: linear, iterative, recursive
- Task ordering and data sharing constraints
- The magic of the architecture: cache coherency and synchronizatio support
- Data decomposition and influence of memory

OpenMP tasks at its heart, going towards task-only parallel programming

- Embarrassingly parallel, recursive task decomposition, task/data decomposition, and branch and bound
- Tareador tool for the exploration of task decomposition strategies

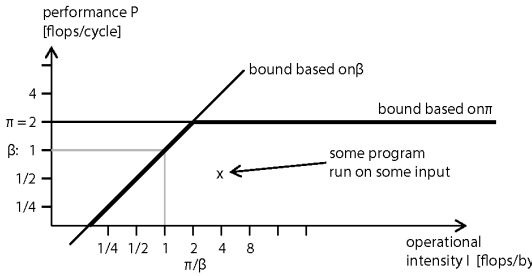
Innovation:
(pseudo-) flipped-classroom model



Teaching opportunities (2)

Syllabus

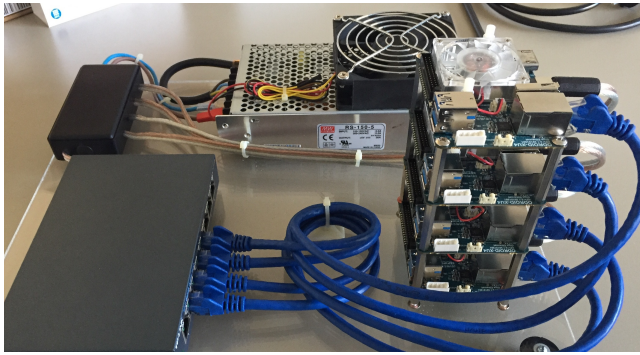
- Compiler and runtime support to OpenMP
- Pthreads API and lib intrinsics
- Designing an HPC cluster: components and interconnect, roofline model for flops/bw
- Message-passing interface MPI



PAP

OpenMP for node programming, with MPI towards cluster programming

- OpenMP tasking review
- Implementing a runtime to support OpenMP
- Building a 4xOdroid cluster: hard and soft installation: the effect of heterogeneity
- Programming the 4xOdroid cluster with MPI: Stencil with Jacobi



Innovation: puzzle work in groups when designing cluster


Thanks
https://www.bsc.es