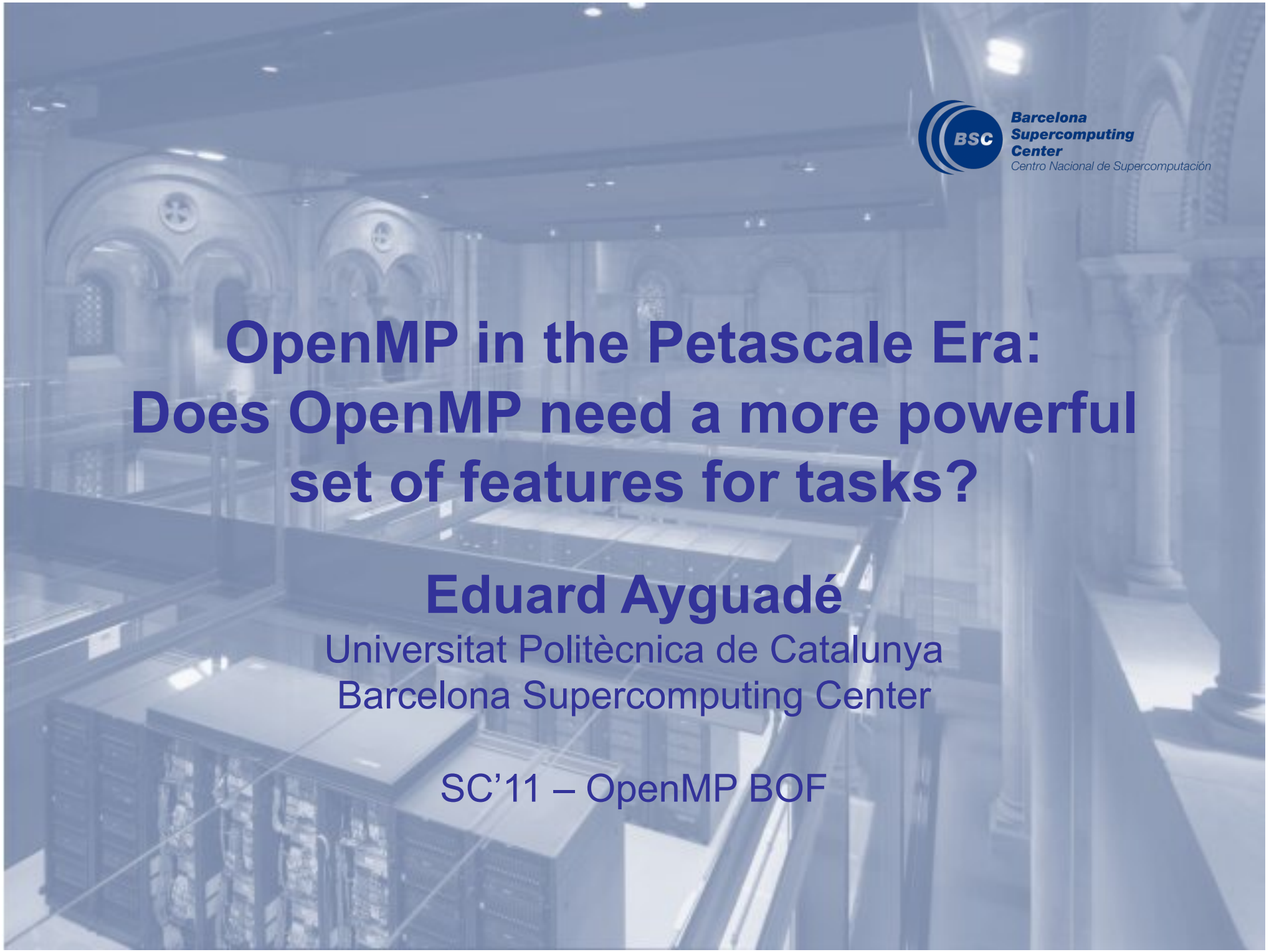

OpenMP in the Petascale Era: Does OpenMP need a more powerful set of features for tasks?

Eduard Ayguadé
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center

SC'11 – OpenMP BOF

Q1: Share your experiences with tasking in OpenMP

- Teaching @ Computer Science School (UPC), Barcelona (Spain)
  - "Parallelism" (3rd course undergraduate) and "Algorithms and Parallel Programming Models" (master degree): "shape" vs. "reshape" minds
  - Tasking is the natural way for expressing parallelism for the algorithms they are used to write (lists, trees, graphs, …). Loop worksharing presented as a compact way to express tasks coming out of loops with granularity control

Q1: Share your experiences with tasking in OpenMP

- Teaching @ Computer Science School (UPC), Barcelona (Spain)
  - "Parallelism" (3rd course undergraduate) and "Algorithms and Parallel Programming Models" (master degree): "shape" vs. "reshape" minds
  - Tasking is the natural way for expressing parallelism for the algorithms they are used to write (lists, trees, graphs, …). Loop worksharing presented as a compact way to express tasks coming out of loops with granularity control
- OmpSs as the programming model used in current EU projects (hybrid MPI)
  - TEXT: Scalapack, PLASMA, SPECFEM3D, LBC, CPMD PSC, PEPC, LS1 Mardyn, Asynchronous algorithms, Microbenchmarks
  - Montblanc: YALES2, EUTERPE, SPECFEM3D, MP2C, BigDFT, PEPC, SMMP, QuantumESPRESSO, ProFASI, COSMO, BQCD
  - Other initiatives: GROMACS, GADGET, WRF, …

Q1: Share your experiences with tasking in OpenMP

- Teaching @ Computer Science School (UPC), Barcelona (Spain)
  - "Parallelism" (3rd course undergraduate) and "Algorithms and Parallel Programming Models" (master degree): "shape" vs. "reshape" minds
  - Tasking is the natural way for expressing parallelism for the algorithms they are used to write (lists, trees, graphs, …). Loop worksharing presented as a compact way to express tasks coming out of loops with granularity control
- OmpSs as the programming model used in current EU projects (hybrid MPI)
  - TEXT: Scalapack, PLASMA, SPECFEM3D, LBC, CPMD PSC, PEPC, LS1 Mardyn, Asynchronous algorithms, Microbenchmarks
  - Montblanc: YALES2, EUTERPE, SPECFEM3D, MP2C, BigDFT, PEPC, SMMP, QuantumESPRESSO, ProFASI, COSMO, BQCD
  - Other initiatives: GROMACS, GADGET, WRF, …

Bottom up and being in total control

Fork join, data parallel, explicit data placement

Top down, potentials and hints rather than how-to's,

Tools for taskification, performance prediction and debugging

Q2: Opinion on the importance of tasking (now and future)

- **Exploitation of unstructured parallelism**
  - Not just loop/data parallelism

Q2: Opinion on the importance of tasking (now and future)

- **Exploitation of unstructured parallelism**
  - Not just loop/data parallelism
  - What do we need for Tera/Exa? More asynchrony, avoid global synchronizations and let the **runtime** orchestrate tasks based on dependences detected at runtime
  - Large amounts of lookahead: instantiate work even if it can not be executed now

Q2: Opinion on the importance of tasking (now and future)

- **Exploitation of unstructured parallelism**
  - Not just loop/data parallelism
  - What do we need for Tera/Exa? More asynchrony, avoid global synchronizations and let the **runtime** orchestrate tasks based on dependences detected at runtime
  - Large amounts of lookahead: instantiate work even if it can not be executed now

- **Locality optimizations / latency tolerance**
  - Let the **runtime** do optimizations that are hard for programmers: reuse, prefetch, overlap data transfers (MPI/OpenMP, OpenMP/accelerator), …

Q2: Opinion on the importance of tasking (now and future)

- **Exploitation of unstructured parallelism**
  - Not just loop/data parallelism
  - What do we need for Tera/Exa? More asynchrony, avoid global synchronizations and let the **runtime** orchestrate tasks based on dependences detected at runtime
  - Large amounts of lookahead: instantiate work even if it can not be executed now
- **Locality optimizations / latency tolerance**
  - Let the **runtime** do optimizations that are hard for programmers: reuse, prefetch, overlap data transfers (MPI/OpenMP, OpenMP/accelerator), …
- **Handling resource heterogeneity**
  - Tasks encapsulating work to be offloaded to accelerators
  - Compatibility with proprietary low level technologies (lot of efforts devoted here!)
  - Let the **runtime** make decisions about scheduling (core/accelerator/…): autotuning, dynamic resource allocation and load balancing

Q3: Present your ideas for enhancing it

- Programmer giving information to the runtime (“hints”) to the runtime and not coding “howtos”

Q3: Present your ideas for enhancing it

- Programmer giving information to the runtime (“hints”) to the runtime and not coding “howtos”

```
# pragma omp task [ input (...)] [ output (...)] [ inout (...)] [ concurrent (...)]
 { function or code block }
```

Annotation of function declarations or definitions

To compute dependences

To allow concurrent execution of commutative tasks (reductions)

Q3: Present your ideas for enhancing it

- Programmer giving information to the runtime (“hints”) to the runtime and not coding “howtos”

```
# pragma omp task [ input (...)] [ output (...)] [ inout (...)] [ concurrent (...)]
 { function or code block }
```

Annotation of function declarations or definitions

To compute dependences

To allow concurrent execution of commutative tasks (reductions)

Task implementation for a GPU device
The compiler parses CUDA kernel invocation syntax

Support for multiple implementations of a task

```
# pragma omp target device ({ smp | cuda })        \
                  [ implements ( function_name )]   \
                  { copy_deps | [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] }
```

Ask the runtime to ensure data is accessible in the address space of the device

Q3: Present your ideas for enhancing it

- Programmer giving information to the runtime (“hints”) to the runtime and not coding “howtos”

```
# pragma omp task [ input (...)] [ output (...)] [ inout (...)] [ concurrent (...)]
 { function or code block }
```

Annotation of function declarations or definitions

To compute dependences

To allow concurrent execution of commutative tasks (reductions)

Task implementation for a GPU device
The compiler parses CUDA kernel invocation syntax

Support for multiple implementations of a task

```
# pragma omp target device ({ smp | cuda })        \
                  [ implements ( function_name )]   \
                  { copy_deps | [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] }
```

Ask the runtime to ensure data is accessible in the address space of the device

```
#pragma omp taskwait [on (...)] [noflush]
```

Relax consistency to main program

Wait for sons or specific data availability

Q3: Present your ideas for enhancing it (cont.)

- Better **control of the threads in the team**

  #pragma omp parallel vs. #pragma omp parallel parallel_threads(n)
  #pragma omp single

- … and also for the implicit parallel region
  `OMP_NUM_THREADS=n` and `OMP_PARALLEL_TASKS=m`

Q3: Present your ideas for enhancing it (cont.)

- Better **control of the threads in the team**

  #pragma omp parallel vs. #pragma omp parallel parallel_threads(n)
  #pragma omp single

- … and also for the implicit parallel region
  `OMP_NUM_THREADS=n` and `OMP_PARALLEL_TASKS=m`

- **Task aggregation**:
  - In recursive programs `final` and `mergeable` already here
  - In unbounded loops with task no solution yet

Want to try OmpSs?

Visit us @ booth 235

Download @ pm.bsc.es