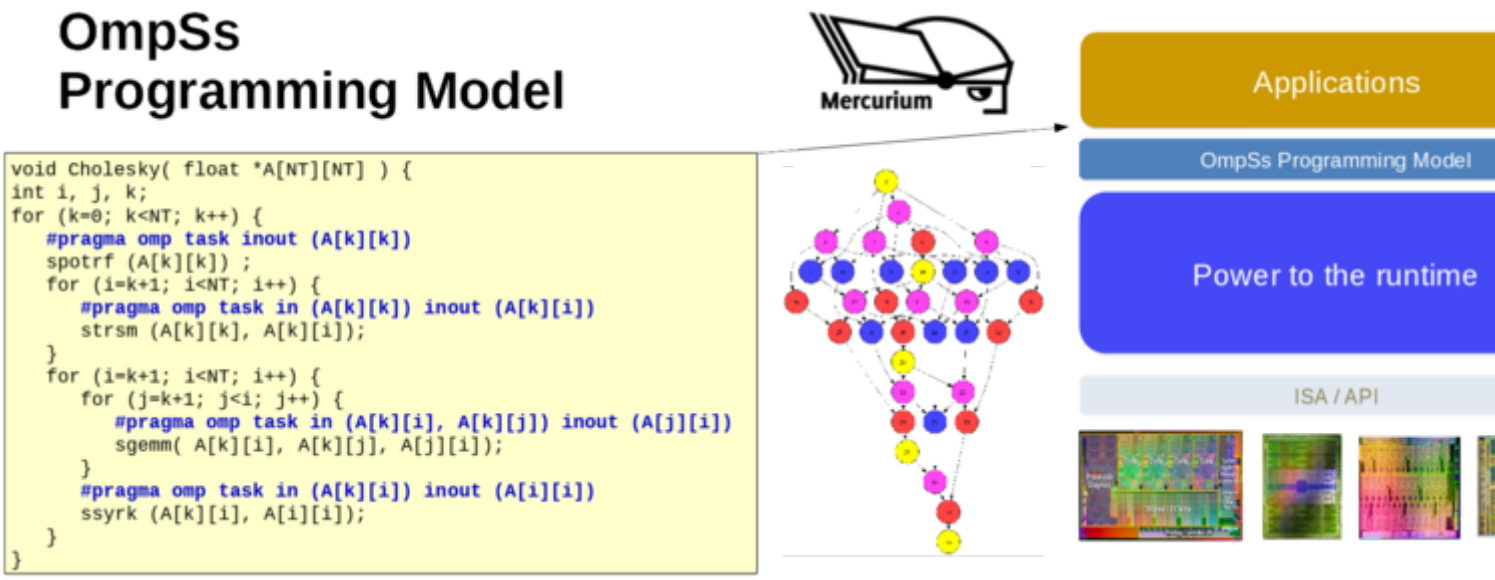
The OmpSs Programming Model

OmpSs is an effort to integrate features from the **StarSs** programming model developed by **BSC** into a single programming model, including support for asynchronous parallelism and heterogeneity on devices like GPUs and FPGAs.

Summary

Asynchronous parallelism is enabled in **OmpSs** by the use **data-dependencies** between the different tasks of the program. To support **heterogeneity** a new construct is introduced: the target construct.

In OmpSs the task construct also allows the annotation of function declarations or definitions in addition to *structured-blocks*. When a function is annotated with the task construct each invocation of that function becomes a task creation point. Note that only the execution of the function itself is part of the task not the evaluation of the task arguments (which are *firstprivatized*). Another restriction is that the task is not allowed to have any return value, that is, the return must be void.



Data dependences

The **task** construct allows to express data-dependencies using the clause in (standing for input, equivalent to OpenMP depend(in:var-list)), **out** (standing for output, equivalent to OpenMP depend(out:var-list)) and inout(standing for input/output, equivalent to OpenMP depend(inout:var-list)) clauses to this end. They allow to specify for each task in the program what data a task is waiting for and signaling is readiness. Note, that whether the task really uses that data in the specified way its the programmer responsibility. Each time a new task is created its in and out dependencies are matched against of those of existing tasks. If a dependency, either RaW, WaW or WaR, is found the task becomes a successor of the corresponding tasks. This process creates a task dependency graph at runtime. Tasks are scheduled for execution as soon as all their predecessor in the graph have finished (which does not mean they are executed immediately) or at creation if they have no predecessors.

The task construct is extended with the concurrent clause. The concurrent clause is an special version of the inout clause where the dependencies are computed with respect to in, out and inout but not with respect to other concurrent clauses. As it relaxes the synchronization between tasks the programmer must ensure that either the task can executed concurrently or that additional synchronization is used. While the reduce tasks can be executed concurrently between them they are still ordered with respect to the generate task but as the reduce tasks can run simultaneously the atomic construct needs to be used to ensure proper synchronization.

The use of the data dependency clauses allow the execution of tasks from the multiple iterations at the same time. All dependency clauses allow extended *lvalues* from those of C/C++. Two different extensions are allowed: **array sections** allow to refer to multiple elements of an array (or *pointee* data) in single expression and **shaping expressions** allow to recast pointers into recover the size of dimensions that could have been lost across calls.

The **taskwait** construct is also extended with the **on** clause. This clause allows to wait only on the tasks that produces some data in the same way as in clause. It suspends the current task until all previous tasks with an out over the expression are completed.

Heterogeneity

The intent of the target construct is to specify that a given element can be run in a set of devices. The target construct can be applied to either a task construct or a function definition. In the future we will allow to allow it to work on worksharing constructs.

Objectives

OmpSs main goal is to act as a forefront and nursery of ideas for a data-flow task-based programming model so these ideas can ultimately be incorporated in the **OpenMP** industrial standard.

Barcelona Supercomputing Center - Centro Nacional de Supercomputación

---