An Optimized Task-Based Programming Model for Embedded Many-core Computing Platforms

Giuseppe Tagliavini

Andrea Marongiu

Luca Benini

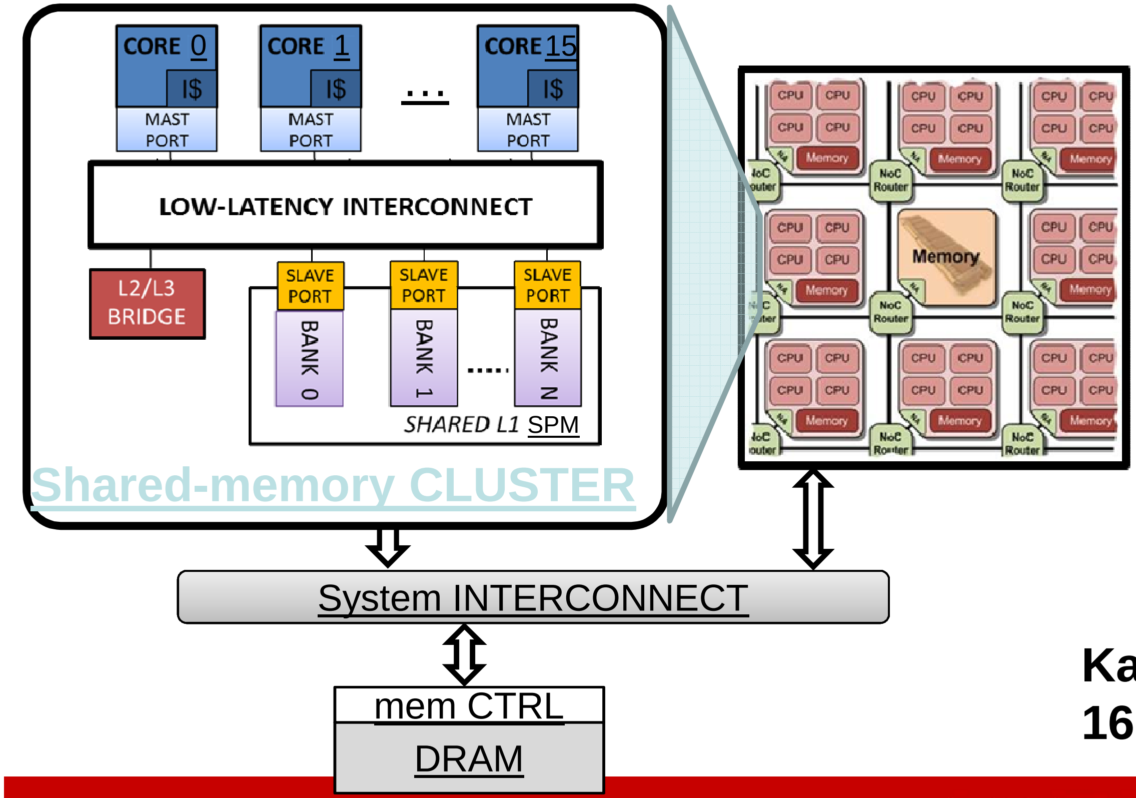
Contact*: giuseppe.tagliavini@unibo.it*

IWES 2017, Rome

- **Introduction**
- OpenMP tasking model
- Main contributions
- Experimental results
- Conclusion

Many-core accelerators…

- ***Many-core accelerators*** are a promising solution for energy- efficient embedded computing systems
- **Clustered parallel accelerators** → multiple clusters that are equipped with processing units tightly-coupled with a shared low-latency L1 scratchpad memory.



Kalray MPPA
16 cluster → 16 core = 256 core

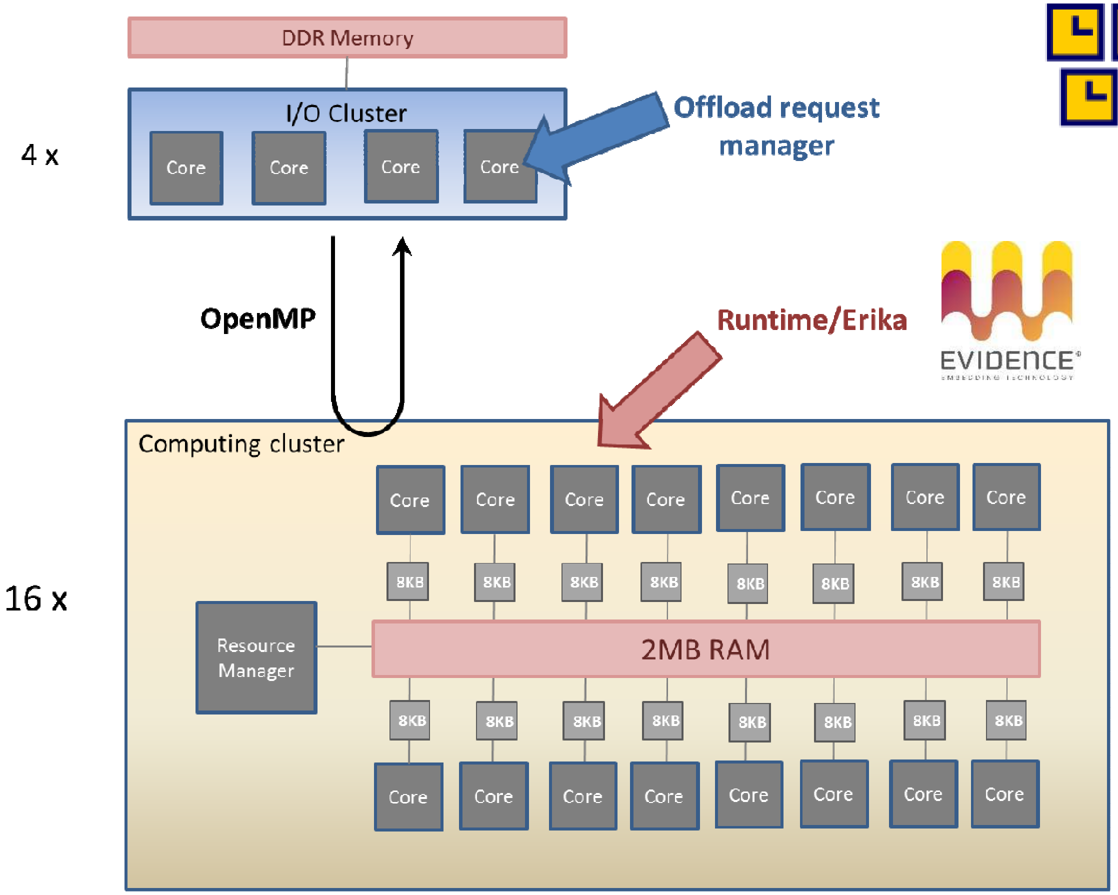
… with proper SW support

- **Clustered many-core designs offer tremendous GOps/Watt, and parallel potential…**
  - ..but **extracting peak performance** at application level remains hard
- **Traditional form of parallelism exploited in large systems is data-parallelism**
  - e.g, loop based
- **New applications expose irregular/structured parallelism**
  - Often, more levels (**nested parallelism**)

- Need for **programming abstractions** to support parallelism in an elastic/dynamic way
- **Flexible and scalable solution → OFFLOADING + TASKING**

Offload model

P-SOCRATES



Main requirement:
PREDICTABILITY

- Introduction
- **OpenMP tasking model**
- Main contributions
- Experimental results
- Conclusion

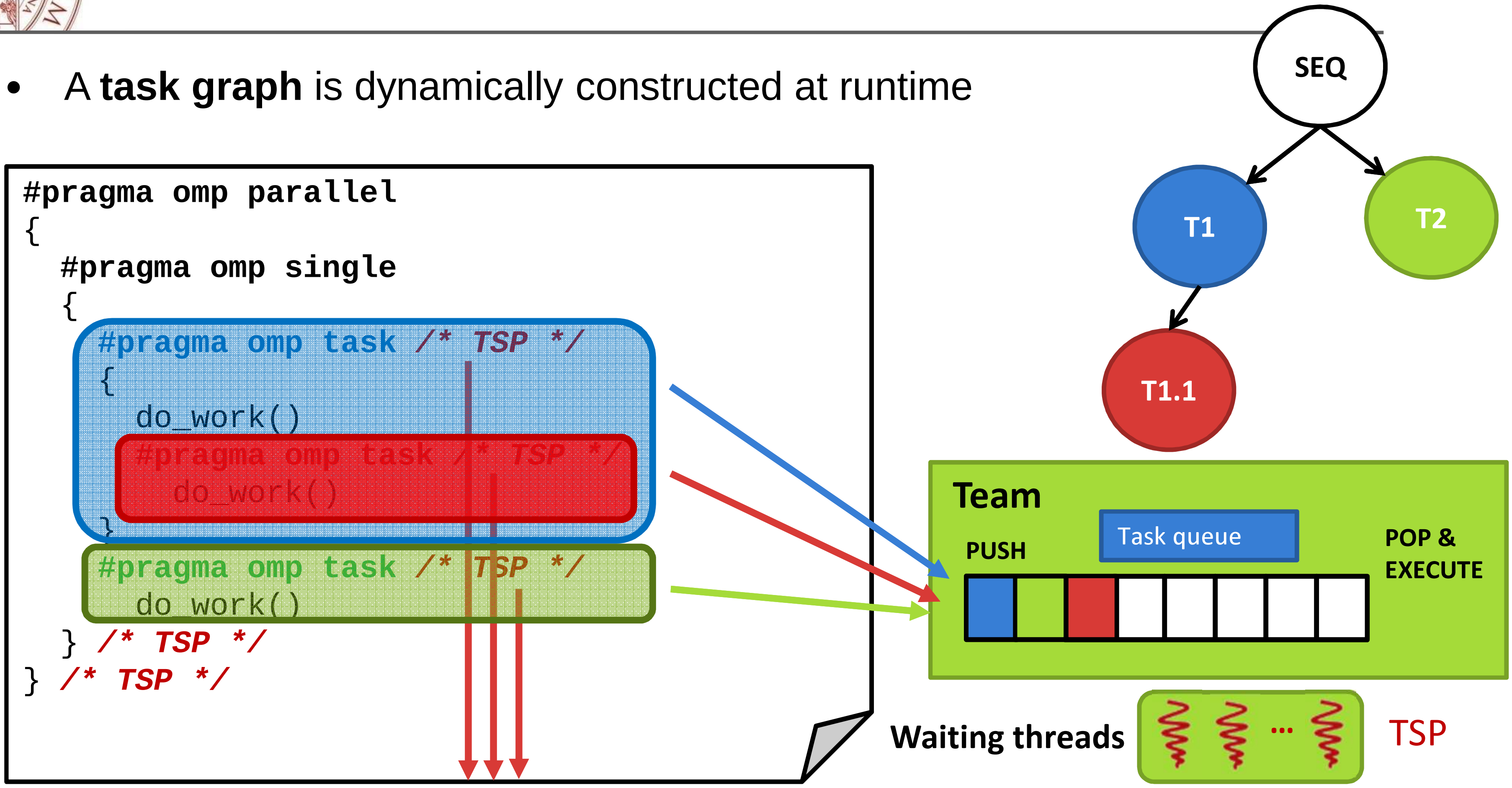
OpenMP tasking

We propose a fully compliant implementation of OpenMP tasking for embedded parallel accelerator with ultra-low overhead, higher performance and higher predictability compare to current OpenMP implementations

- **Why OpenMP?**
  - Widely adopted programming model for shared memory systems
  - Several implementation for embedded system are available
  - Simple pragma-based programming interface

OpenMP tasking model

- A **task graph** is dynamically constructed at runtime



```
#pragma omp parallel
{
  #pragma omp single
  {
    #pragma omp task /* TSP */
    {
      do_work()
      #pragma omp task /* TSP */
        do_work()
    }
    #pragma omp task /* TSP */
      do_work()
  } /* TSP */
} /* TSP */
```

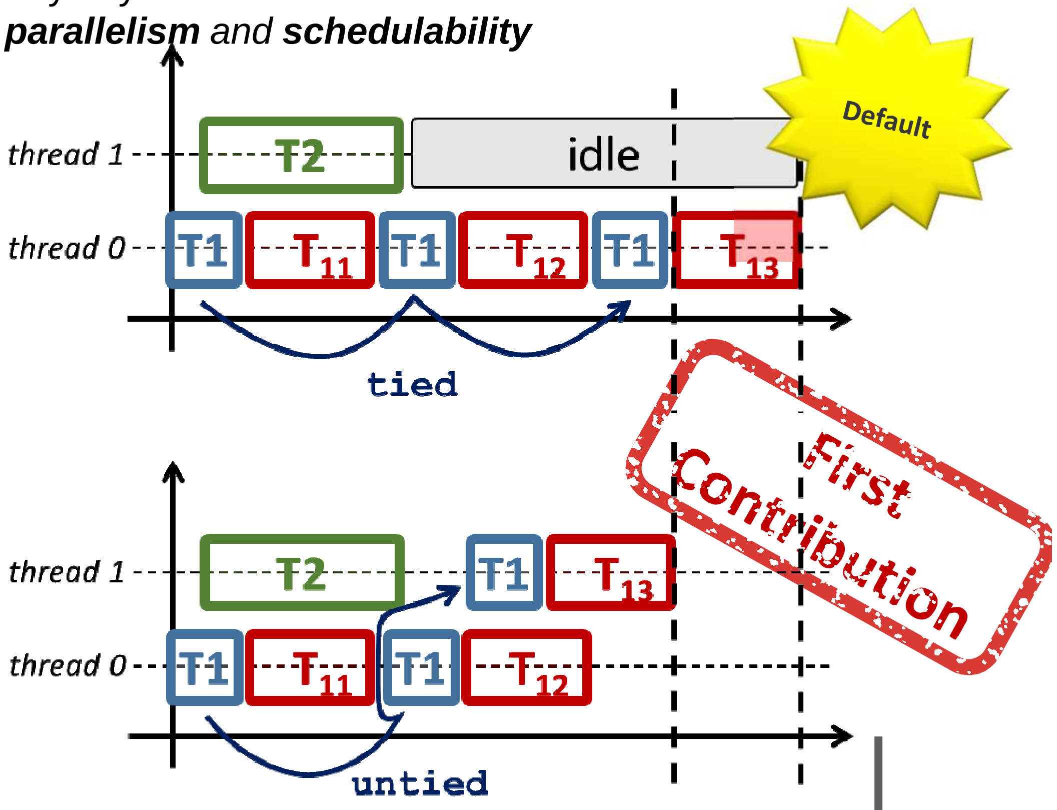
OpenMP defines *task scheduling points* (TSP) in a program, where the encountering task can be suspended and the hosting thread can be rescheduled to a different task.

- Introduction
- OpenMP tasking model
- **Main contributions**
- Experimental results
- Conclusion

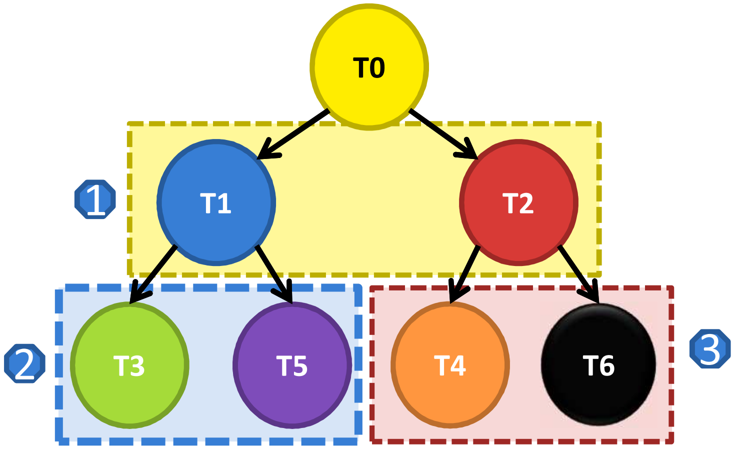
Task types

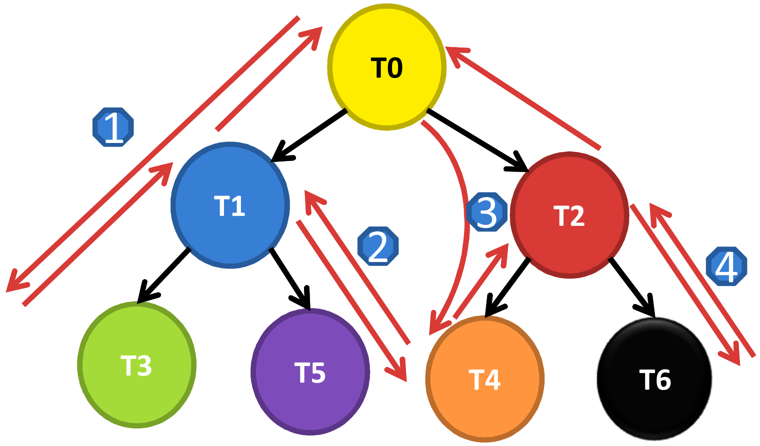
- **Tied task (*default*)**
  - If suspended, it can later only be resumed by the same thread that originally started it
  - Trade-off between ease of programming and scheduling flexibility
- **Untied task**
  - If suspended, they can later be resumed by any thread
  - *Significantly increasing the **achievable parallelism** and **schedulability***

```
#pragma omp parallel
{
  #pragma omp single
  {
    #pragma omp task /* TSP */
    {
      do_work()
      #pragma omp task /* TSP */
        do_work()
    }
    #pragma omp task /* TSP */
      do_work()
  } /* TSP */
} /* TSP */
```



Task scheduling





Default

- **Breadth-first scheduling (B**
  - The parent task creates all the children tasks and pushes them in the working queue continuing the execution until the end of task
  - Tends to be more demanding in terms of memory

Second Contribution

- **Work-first scheduling (WFS)**
  - Suspends the parent task and start execution of the new task
  - Lower demands of memory
  - Better data locality → follow the path of the original sequential program
  - Needs untied tasks

Cost of tasking

Two key issues must be addressed for runtimes based on tasking:

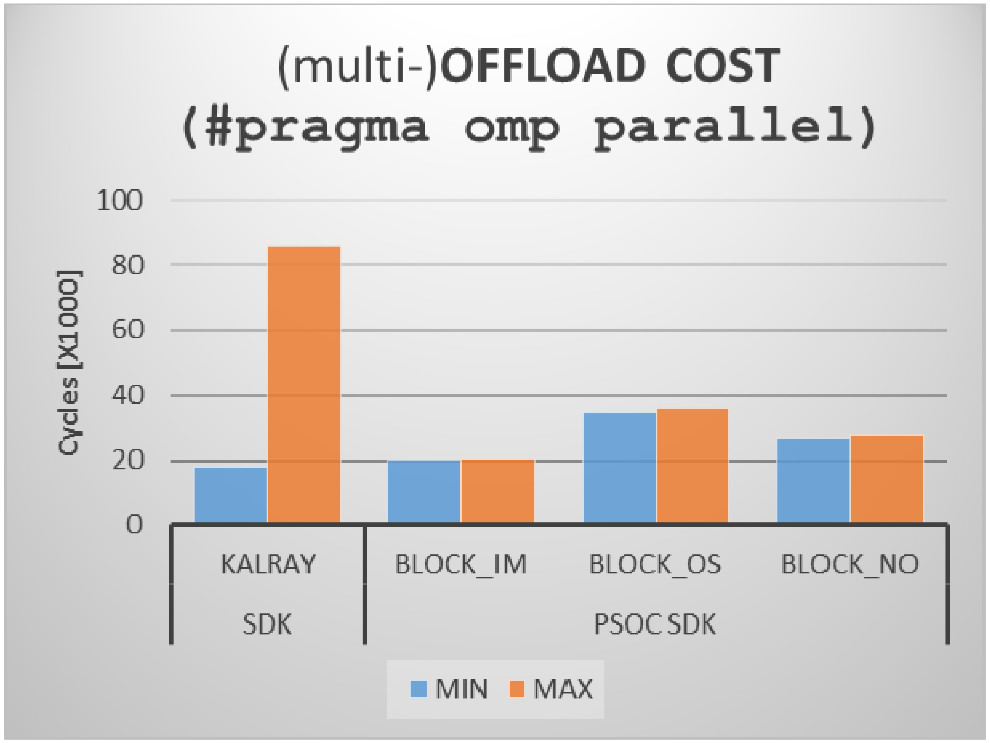
Third Contribution

- **Time overheads**
  - The applicability of the tasking approach to embedded many-core accelerators is often limited to **coarse-grained tasks**
  - The runtime must support **fine-grain tasks** to exploit in a efficient way parallel workloads

- **Space overheads**
  - In resource-constrained systems that are based on **space-limited scratchpad memory,** is very important having RTE with a **low memory footprint** to leave as much as possible memory to the application data

- Introduction
- OpenMP tasking model
- Main contributions
- **Experimental results**
- Conclusion

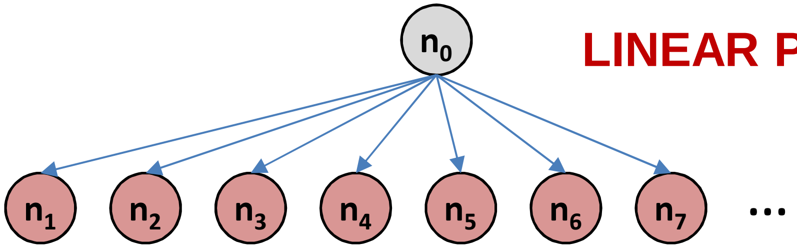
Offload on ERIKA/Kalray MPPA



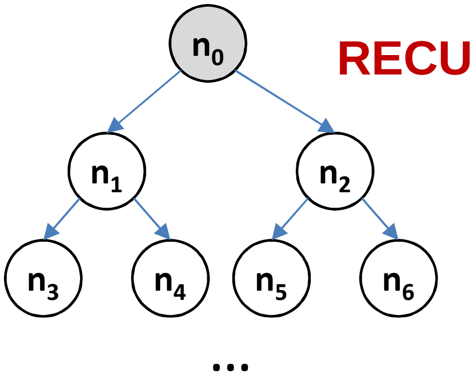
Fairly high cost for offload startup on clusters (parallel). Main reason is management of non-coherent caches

- Different implementation of synchronization primitives
  - **BLOCK_IMMEDIATE** the condition is checked in a busy waiting loop;
  - **BLOCK_OS** informs the OS that the OpenMP thread is “idle”. The OS can then block this thread and schedule another one in the ready queue;
  - **BLOCK_NO** (LIMITED PREEMPTION) informs the OS that the OpenMP thread has reached a TASK SCHEDULING POINT. If a higher-priority thread is found in the ready queue it gets scheduled.

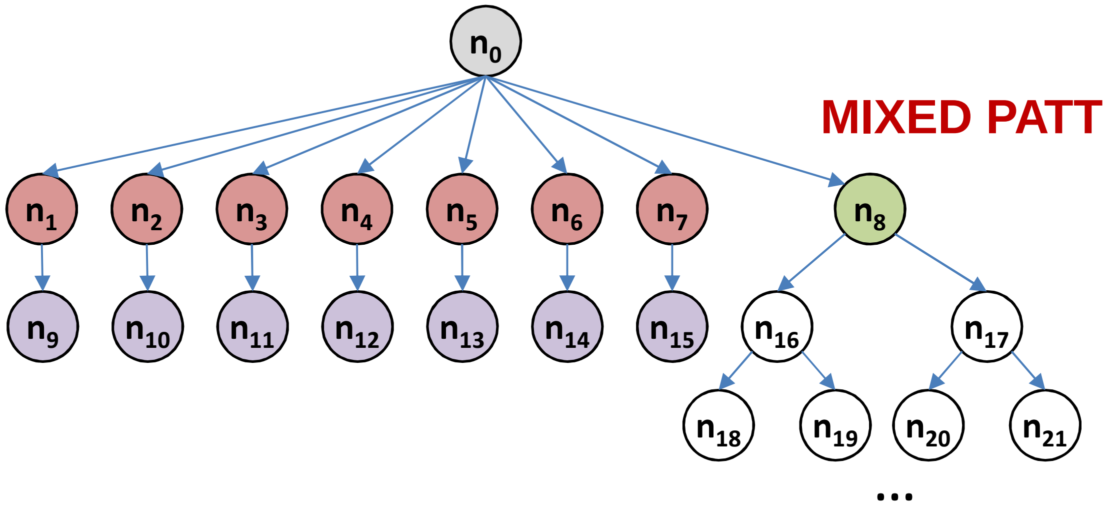
Synthetic benchmarks



LINEAR PATTERN

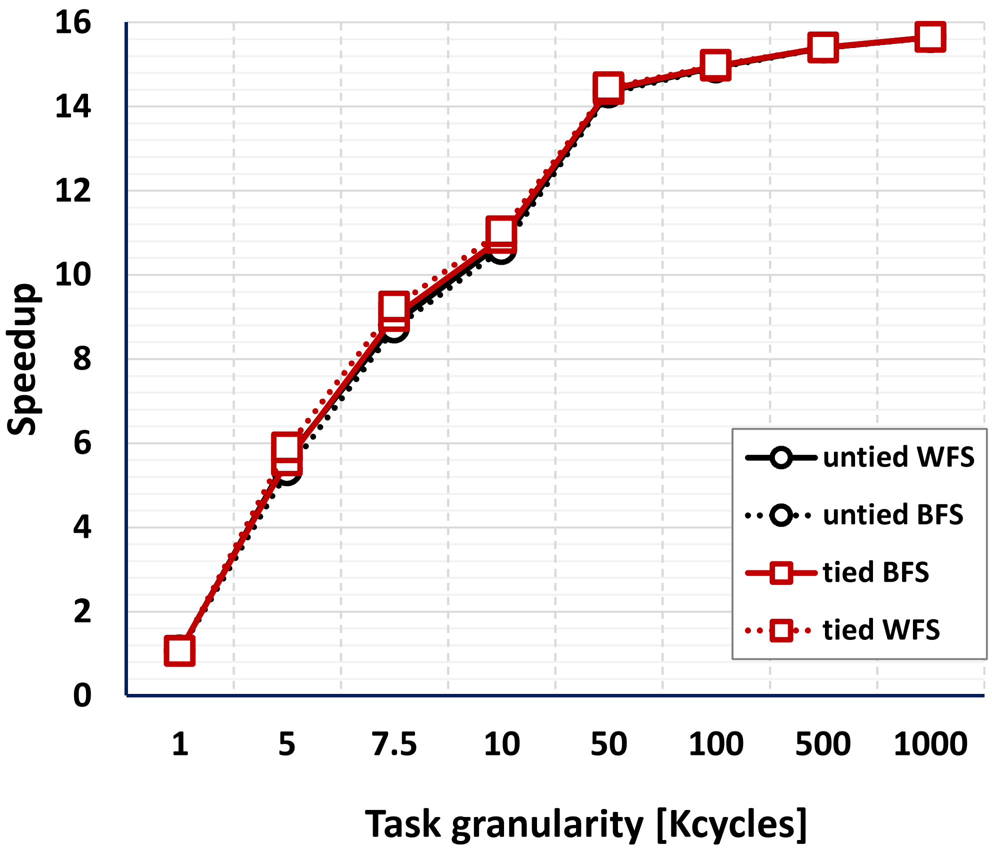


RECURSIVE PATTERN



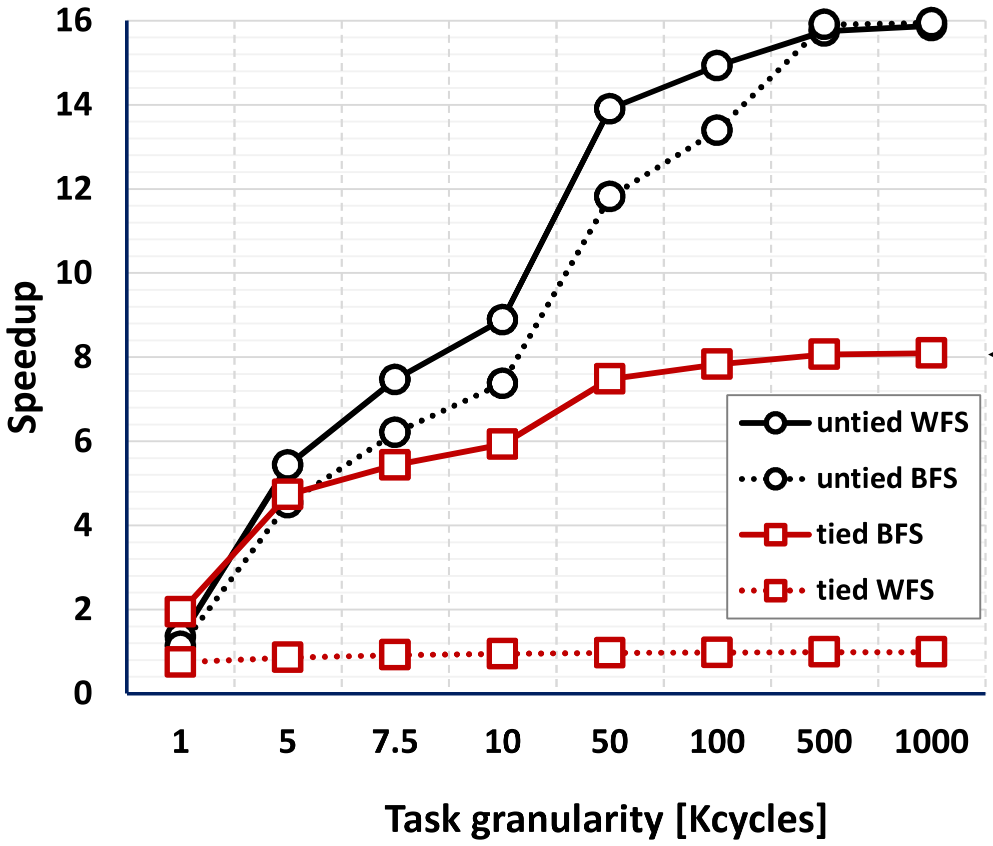
MIXED PATTERN

TIED vs UNTIED: linear



No relevant difference between WFS / BFS and tied/untied tasks!

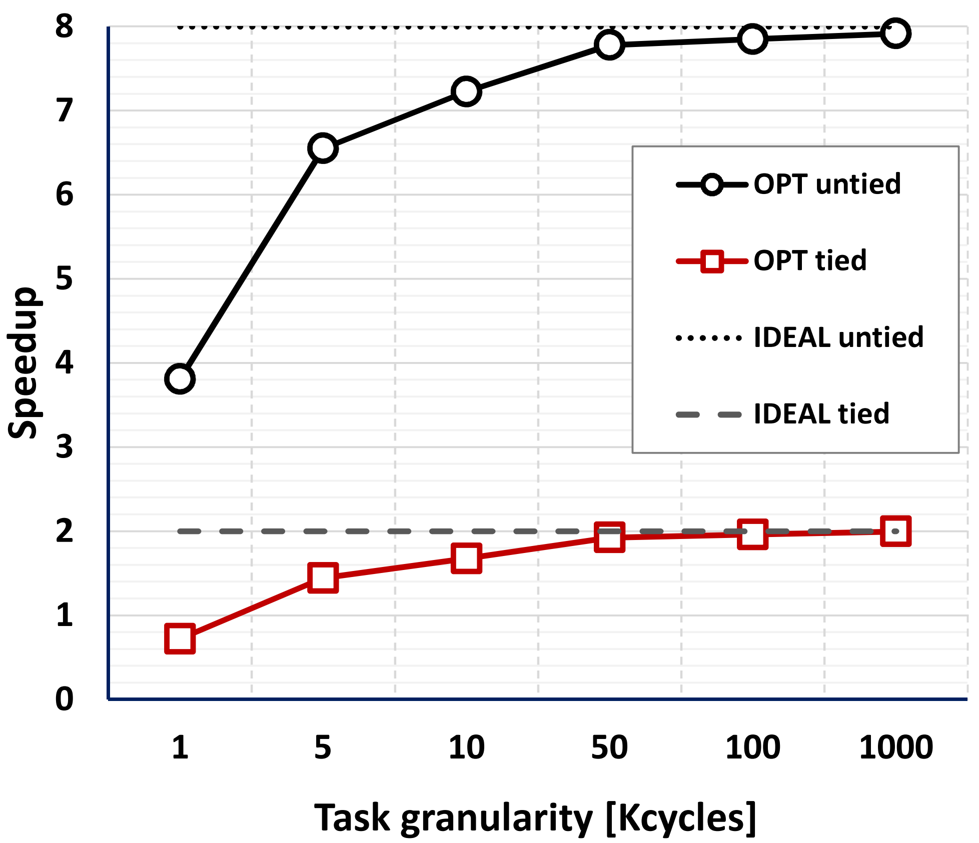
TIED vs UNTIED: recursive



???

Untied tasks with WFS achieve the maximum speedup

TIED vs UNTIED: mixed



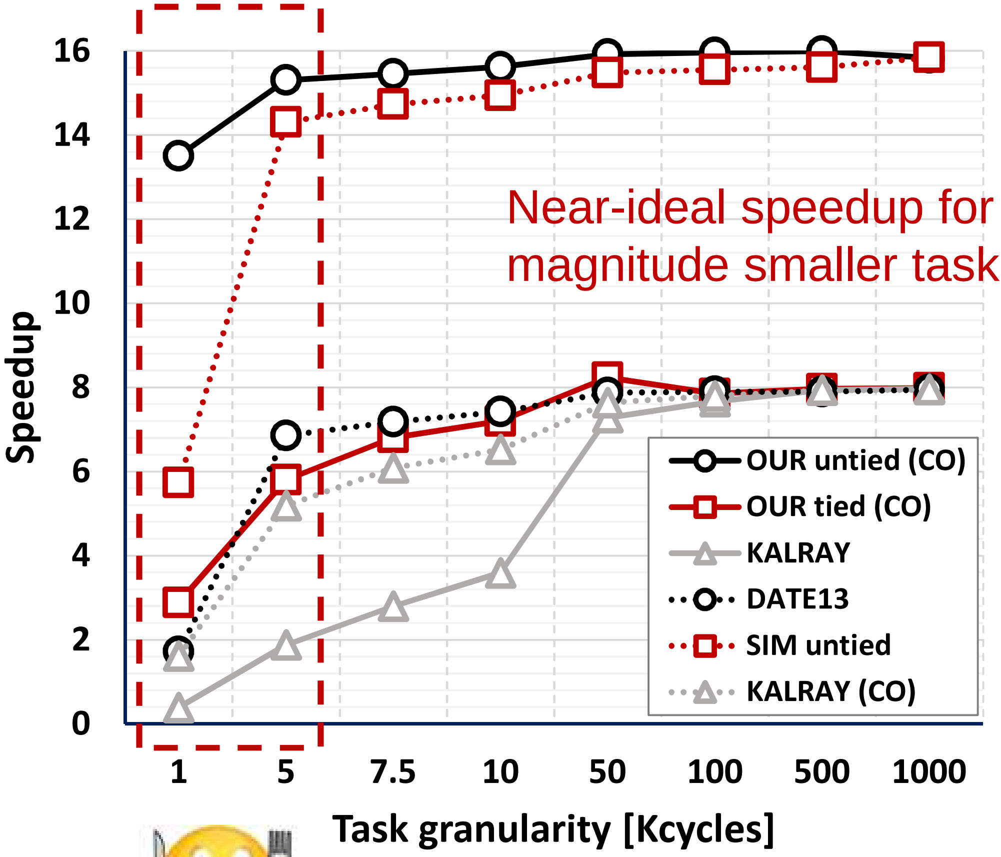
???

?????????

Using tied tasks, 14 cores are allocated to execute the linear part of the application → 7 are blocked by the taskwait directive

Comparison with other embedded runtimes (recursive pattern)

Low-granularity tasks



Near-ideal speedup for one order of magnitude smaller tasks

- Introduction
- OpenMP tasking model
- Main contributions
- Experimental results
- **Conclusion**

Where we are, where we are going

- Optimized runtime for OpenMP tasking
  - Support of **untied tasks** based on lightweight co-routines
  - Data structure policies to **reduce memory footprint**
  - Allocation policies to reduce **task creation time**
  - *Cut-off policies* to **reduce execution time**

- Work in progress and evolutions:
  - Impact of tasking on alternative **architectural templates**
  - Offload on **heterogeneous platforms**
  - Integration with **alternative programming models** (OpenCL, OpenVX, CUDA, …)

Questions? Ideas?

Contact: ***giuseppe.tagliavini@unibo.it***

<u>Work supported by EU-funded ERC advanced project</u>

Multitherman