Towards realistic core-failure-resilient scheduling and analysis

Borislav Nikolić and Konstantinos Bletsas
{borni, ksbs}@isep.ipp.pt

(1) The scheduling problem

- **Goal:** Meeting all task deadlines on a multicore platform even when a core suddenly fails and is rendered unusable.
- **Model:** When a core fails, whichever task was running there is killed but its deadline must still be met.

(2) The baseline approach

- Global fixed-priority scheduling.
- Generalisation of two simple but “faulty” ideas:
  - Full task replication (Resource-inefficient!)
  - Restart task upon failure (May be too late!)
- For each job by task $\tau_i$, release a *copy job* after a time offset $O_i$, relative to the main job.
  - smaller $O_i$: more redundant execution.
  - bigger $O_i$: harder to meet deadline.
  - Optimal $O_i$: the biggest value that allows provably meeting deadlines in every case.

(3) Critical sections

- Provisions for resource sharing under some adaptation of an existing protocol are needed.
- But what happens if a task dies while executing a critical section?

- Transaction semantics (with COMMIT and ROLLBACK) appear as an appropriate solution.

(4) Indirect resource sharing

- With job copies, *all* resources suddenly become shared (between the main job and its copy!)
- Possible solutions:
  - Critical sections everywhere (inefficient).
  - Code-level analysis also considering $O_i$, in order to rule out some access hazards.

(5) Implementation aspects

- Facility for detecting/handling core failures.
- Facility for launching, tracking and terminating jobs early.
- Incorporation of overheads into the analysis, taking into account the actual implementation.