

# Research Topic: Hardware Transactional Memory

PhD student:

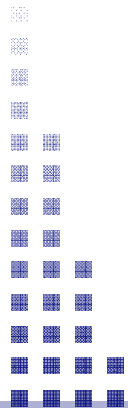
J. Rubén Titos

Advisors:

Manuel E. Acacio

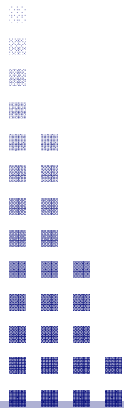
José M. García

{rtitos,meacacio,jmgarcia}@ditec.um.es



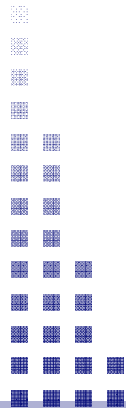
# Context: Transactional Memory

- Paradigm shift toward multicore architectures
- Difficulty of Parallel Programming (PP)
  - The eternal tradeoff: Performance vs Correctness&Productivity
- TM Programming Model
  - Transaction-based Synchronization
    - Programmer declares critical regions `begin_xact/commit_xact`
    - ...but doesn't worry about guaranteeing atomicity (vs. Locking)
- TM system (HW, SW or hybrid)
  - Concurrently executes critical regions (optimistic synchronization)
  - Dynamically detects and resolves conflicts
- Goal: Programmability + Performance
  - Programming effort: as coarse-grain locking
  - Performance: as fine-grain locking (if conflicts are rare)



# Challenges

- How much HW support for TM?
  - Make PP easier by investing a part of the transistor budget
  - TM exposes a tricky interaction between HW and SW
- Conflicts:
  - What if they are not so uncommon?
    - TM's goal: PP also for the common programmer
      - Key to success of CMP systems (everybody can fully exploit cores)
      - Large transactions are to be expected (coarse grain tasks)
      - Result: high(er) frequency of data conflicts
  - → TM systems should deal with conflicts efficiently
    - Offer good performance regardless of conflict frequency
- Lack of transactional workloads
  - What is the common case?



# Our proposal

- Enhance TM basic mechanisms in HW:
  - Faster conflict detection and resolution
    - Reduce the number of hops needed for detection
  - Reduce abort penalty
    - Smarter recovery schemes
  - Flexible version management policies (Eager & Lazy)
    - Supported in HW, controlled by SW
  - Preserve atomicity while allowing some violations
    - Take full advantage of the checkpoint/rollback support

