Using Multi-Core Architectures to Execute High Performance-Oriented Real-Time Applications

Ch. Aussaguès[a], *E. Ohayon*[a], *K. Brifault*[a] and Q.V. Dinh[b]

[a]CEA LIST, Embedded Real-Time Systems Laboratory
[b]Dassault Aviation

ParMA

Parallel Programming for **M**ulti-core **A**rchitectures
www.parma-itea2.org

International Parallel Computing Conference (ParCo'09)

Outline

//ParMA

- Context: parallelism in embedded real-time system design

- Extension targeting multi-cores

- Experiment & early results

- Future works

cea list

digiteo labs

- We have to cope with the same problems than HPC to **bridge the gap** between parallelism expressed in design (modularity) and those available in multicore-based architectures (performances)
  - Express potential parallelism in programming models
    - Independently of any architecture
  - Map the potential parallelism to the real one available in the underlying hardware
    - From single-core implementation to **multi-core** ones
    - In a **scalable** and transparent manner for the developer
  - Taking into account additional embedded requirements (real-time constraints, …)

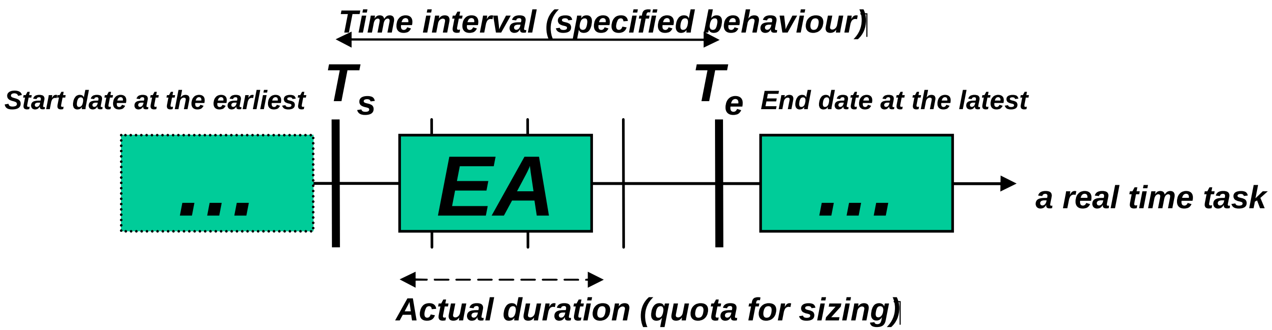
The OASIS tool chain and real-time kernel

- Objective: to allow design and implementation of safety-critical multitasking applications with **advanced real time functionalities**
  - To guarantee responses in specified times
  - To ensure **predictable** and **reproducible** behaviors
  - The application behavior is **deterministic**

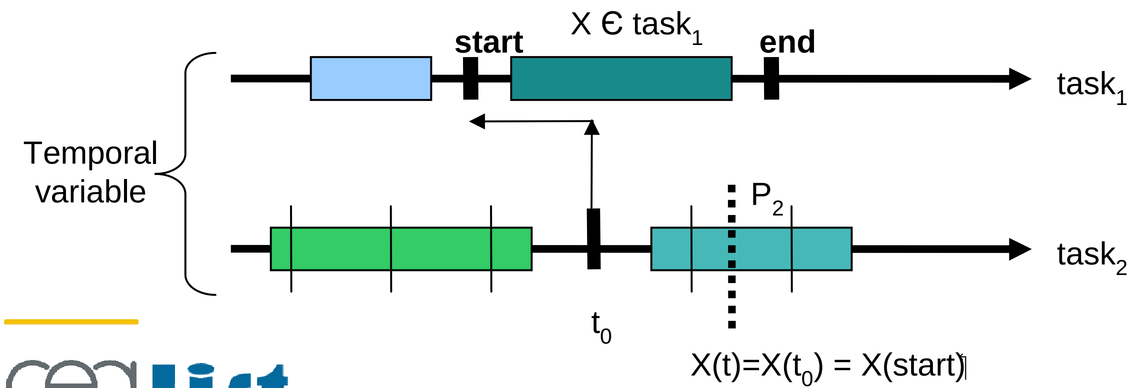
- Means
  - Code generation tools for real-time multitasking systems
    - Compilation (*semi-formal language ΨC including ANSI C*)
    - Automatic **code generation** & sizing
    - Dedicated **code & data segmentation**, link edition (*with MMU tables*)
  - Safety-oriented kernel
    - Generic, **time-triggered** and **safety-oriented**
    - Current targets are Motorola 68040/60, **Intel IA32**, ARM, S12XE

Some basics on OASIS

- Time-Triggered Multitasking programming & execution model
  - All the processing of a task take place between two known instants of the TT system (not necessarily consecutive)
  - All the data transfers between “tasks” are done at the transition between two elementary actions (EA)



- Temporal consistency of exchanged data
  - Each data protected and has only one producer
  - Data on which works a processing are not sensitive between the beginning date and the ending date



- Strict observation principle
- Implicit & automatic update at defined instants
- Automatic sizing & protected safe buffers

The OASIS SMP kernel – key points

ParMA

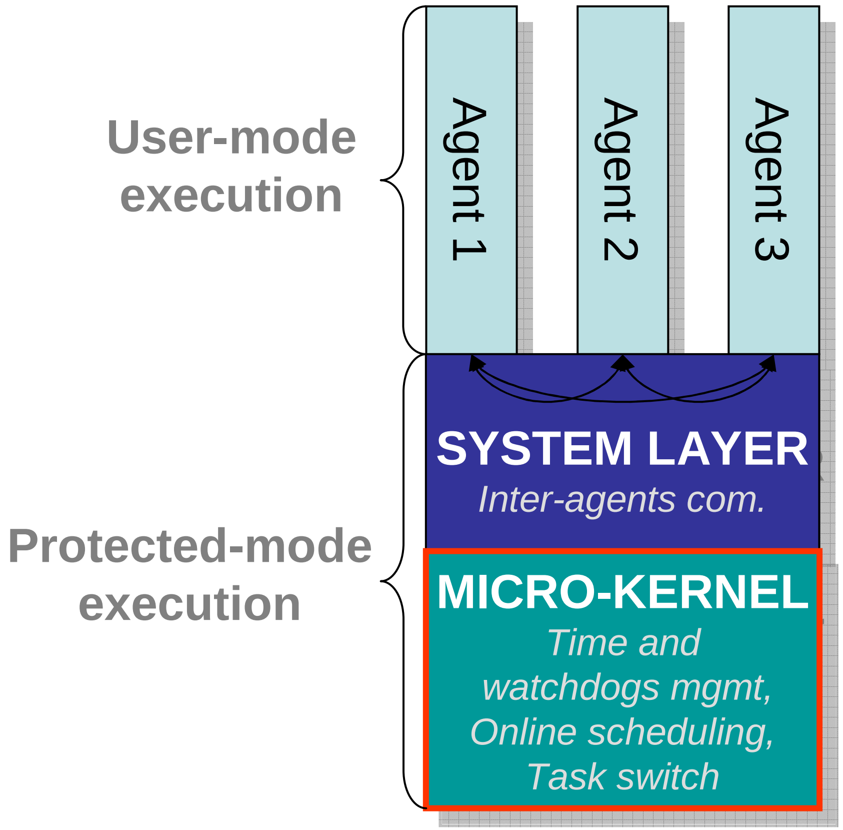
- **Simultaneous execution of different real-time tasks** (*agents*) **on several cores** (*up to 16*)

- ***No change to the OASIS programming model***
  - An application compiled for single-core OASIS kernel can be used *directly* with the SMP kernel
  - Tasks of the app. are *dynamically distributed* between available cores

- **Modification of the kernel**
  - *Detection and setup of available cores* during kernel boot
  - *Dynamic and transparent mapping* of real-time tasks to cores at run-time
  - In conformance with the OASIS time-triggered execution model
    - Guarantee of *time consistency* between cores
    - Global *Earliest-Deadline-First* scheduling policy

- **Agents code**:
  - No shared memory or communication outside the System Layer
- **System Layer code**:
  - Already lock-free interruptible code
  - Requires memory fences
- **Main difficulty: the μK**
  - Single-core arch: atomic section
  - *Masked IT: not sufficient on SMP arch.*

Which code can be executed simultaneously ?



- **Design choice: allow most of the micro-kernel code to be executed on any CPU**
  - ***Aim: balance system load between available cores***
  - **Asymmetric approach for time mgmt:** only CPU0 can handle timer ITs → additional system load for
    - *global time update management*
    - *watchdog setup and handling*
  - **But scheduler and context-switch ops. can be run on any CPU**

- **SMP micro-kernel features:**
  - **Lock-free** time management algorithm
  - **Lock-free** context-switching procedure
  - **Atomic** scheduler – *fair spinlock (active wait) protection*

- Use case from Dassault Aviation: real-time tracking of targets using particle clouds simulations:
  - requires a **safe and real-time** design & execution environment
  - *contains parallelizable code from data and control points of view*

- So far, purely sequential C-code

Benchmarking with DA's tracking application – experiment (1/3)

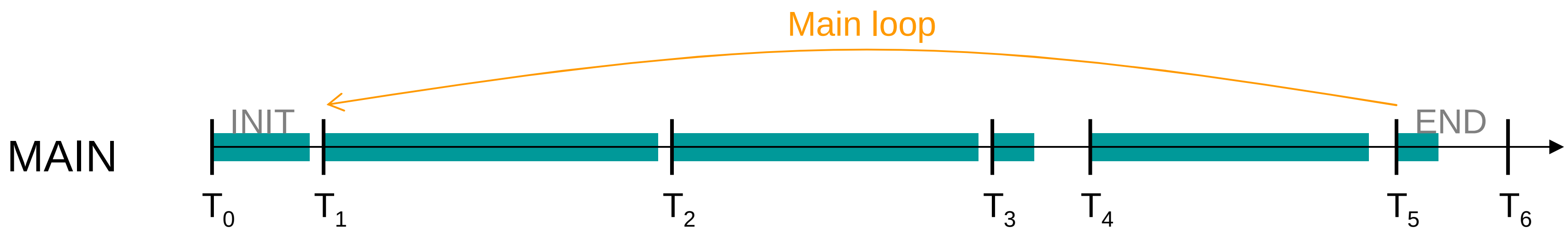
- **Keypoint**: use SMP OASIS version on a *n*-core arch. to improve performances
  - Use **multiple communicating tasks** to execute the code
  - 1 master task communicating with (*n*-1) slave tasks helping occasionally for time-consuming operations

- **Cons**: overhead due to inter-task communications (*no shared memory between tasks*)
- **Pros**: allows **parallel execution** of selected portions of code on several cores

Benchmarking with DA's tracking application – experiment (2/3)

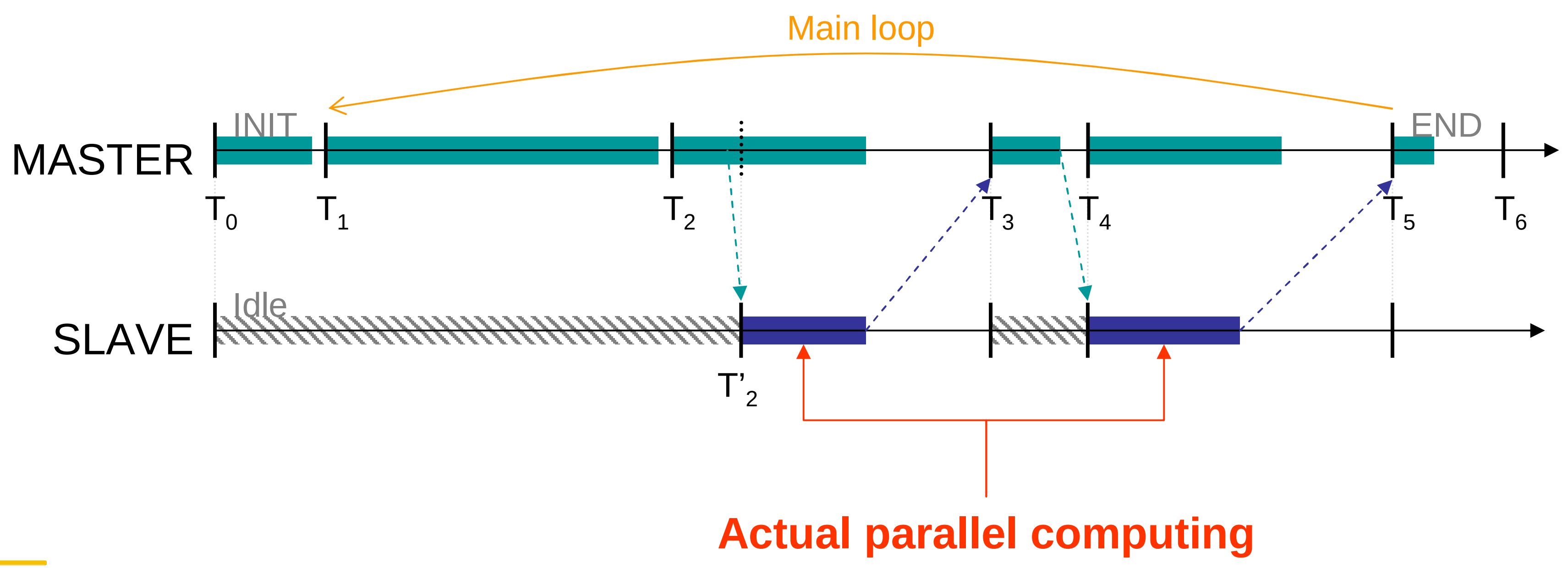
- Work done with DA: translation of the tracking algorithm pure C code to ΨC in **2 steps**:

1. Directly in **one real-time task** in ΨC (reference step), defining main real-time constraints
    - Computation on **70.000** particles



ParMA

2. Into **one master task, slave tasks and their communication** interfaces in ΨC
Application structure: **1 master task + 1 slave task**
   - **35.000** particles per task
   - **4 temporal sync. points** (*messages exchanges*) per loop



cea list

digiteo labs

Early results

- **Same real-time behavior & results in 1-task and 2-tasks versions**

- In computation code portions excluding communications, **average 30% performance enhancement**, up to the optimum (50%)

- Global performances: **impact due to inter-tasks communications**
  - **Slight performance improvement** when the caches are empty (*e.g. first loop: 18% faster*)
  - Communication overhead when the cache is filled (*max. 5%*)

- **Predictable** overhead due to
  - 1 particle size of ~ 40 bytes,
  - 1 communication between tasks = 3 memory moves or more + 2 MMU switches + 2 CPU mode switches (*necessary to ensure full memory segmentation and safety*)

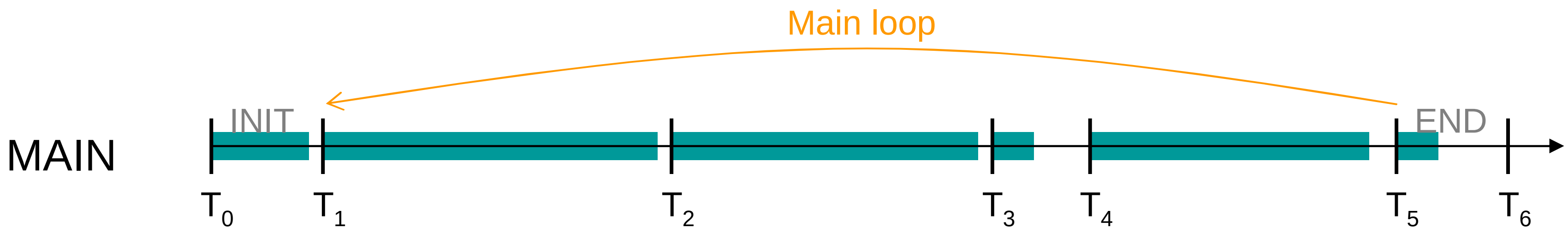
➔ *It paves the way* for the next step…

2nd step: multi-threaded OASIS version

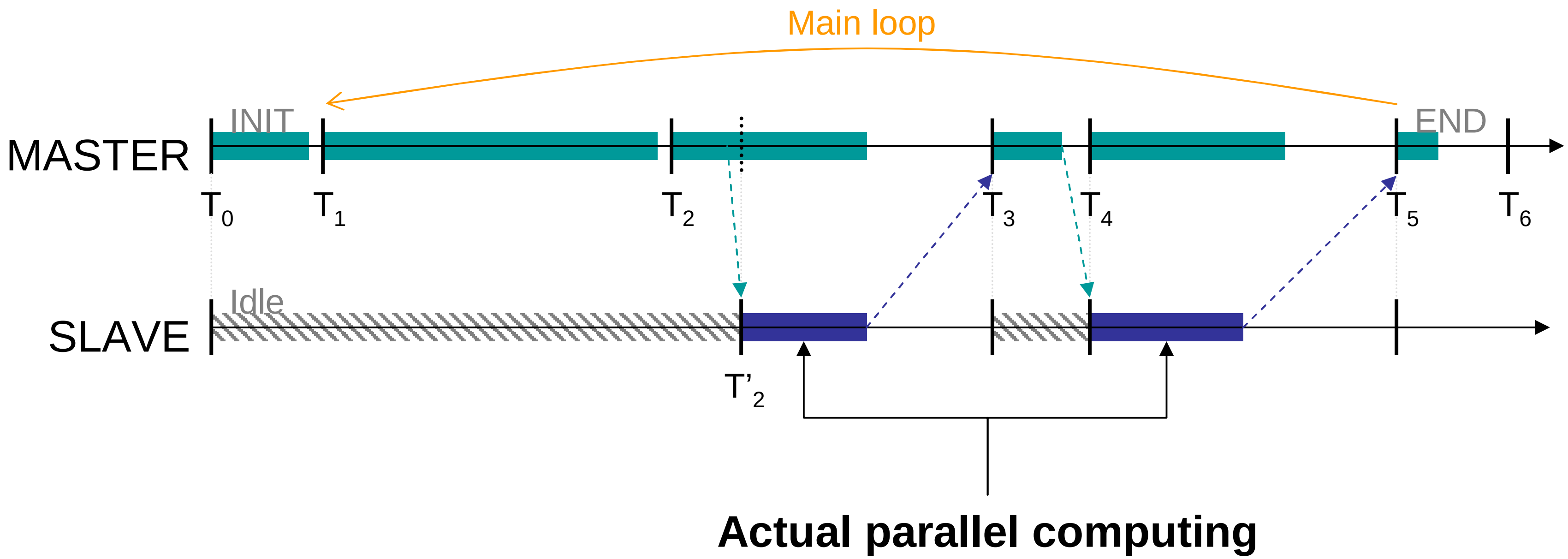
ParMA

- Inside a real-time task: **data-parallelism** management (streaming-like)
  - To allow simultaneous execution of portions of code (e.g. functions) on different data chunks in a temporal window
- Split/Join extensions to the OASIS programming model
  - Threads are executed within a single real-time task, i.e. within one memory context → ***saves context-switch time***
  - Execution threads share their memory → ***much faster comm.***
- On-going modifications of *the entire tool chain*
  - ΨC parser and code generator (*e.g. new keywords to instantiate and terminate threads*)
  - Micro-kernel: handling of thread "contexts"

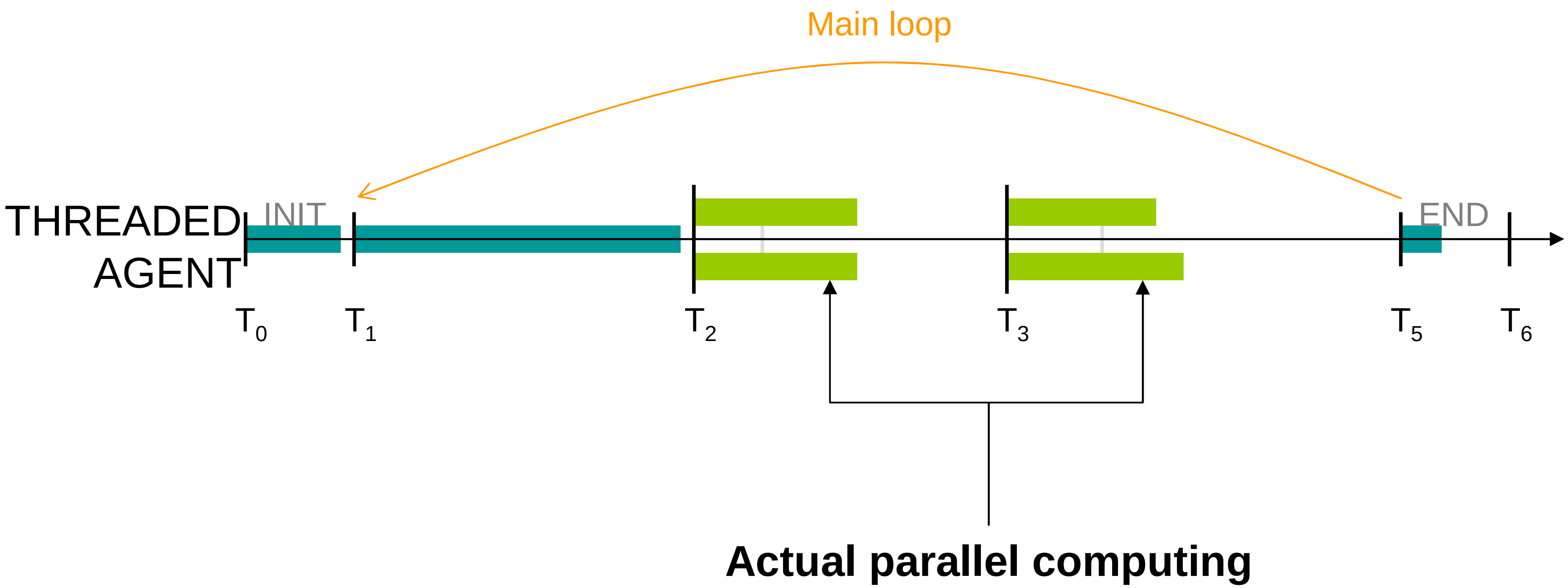
Sequential (single agent) version



Parallel master/slave version



Parallel threaded version



Less synchronization points: no more time slots dedicated to communication purposes

Summary

- Extension of OASIS environment to support **multicore** execution of high performance-oriented real-time applications
  - In a **transparent** manner
    - Same application running on single- and multi-core
      - Same real-time behaviour
      - But allowing integration of more real-time tasks
  - In an **efficient** manner
    - Exploiting parallelism at kernel level
    - Adding data-parallelism at design level

Thank you for your attention

Any questions?