A Model-driven development framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimisation

Technology behind the AMPERE SW framework:

HPC programming models for predictable parallel performance

Run-time support: Resiliency

Sara Royuela, Barcelona Supercomputing Center (BSC)

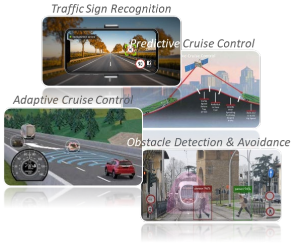
Final Event – AMPERE Project Webinar

27 June 2023 - HIPEAC

The AMPERE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871669

Programming multi-cores

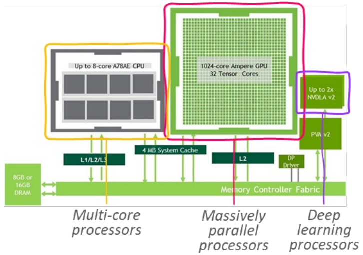
Applications



High-performance *and* ***non-functional requirements***
(time predictability, resilience and energy)
fulfilled at ***all stages of the development*** *process*

Design → ***Coding*** → ***Analysis*** → ***Execution***

Hardware



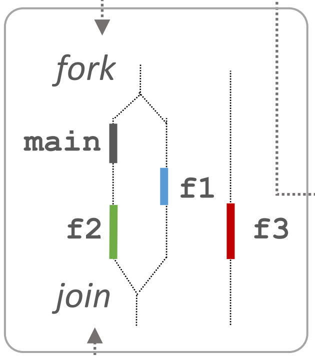
Parallel programming models

1. **Mandatory for SW productivity in terms of**
   - ***Programmability***: Abstractions to describe parallelism while hiding HW complexities
   - ***Portability***: Compatibility with multiple Software Development Kits (SDKs) and Hardware (HW) platforms
   - ***Performance***: Efficiently exploit parallel capabilities of HW
2. **Efficient offloading to HW acceleration devices for an energy-efficient parallel execution**

Parallel programming with OpenMP tasks

- API based on compiler directives, runtime routines and environment variables
- For shared-memory system + accelerators
- Build on top of C/C++ and FORTRAN

Open parallelism



Close parallelism

<u>Sequential version</u>

```
void main() {
  int x,y;
  f1(&x,&y);
  f2(x);
  f3(y);
}
```

Executes on the host (f1, f2)

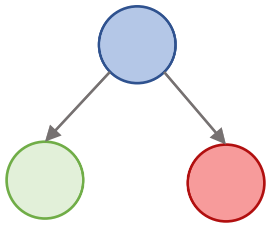
Executes on the accelerator (f3)

<u>OpenMP version</u>

```
void main() {
  #pragma omp parallel
  #pragma omp single
  {
     int x,y;
     #pragma omp task depend(out:x,y)
     { f1(&x,&y); }
     #pragma omp task depend(in:x)
     { f2(x); }
     #pragma omp target map(to:y) depend(in:y)
     { f3(y); }
  }
} // Implicit barrier
```

Task Dependency Graph

(TDG)



Support for non-functional requirements in AMPERE

Performance

Resilience

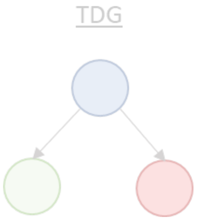
Model-to-code transformation

OpenMP

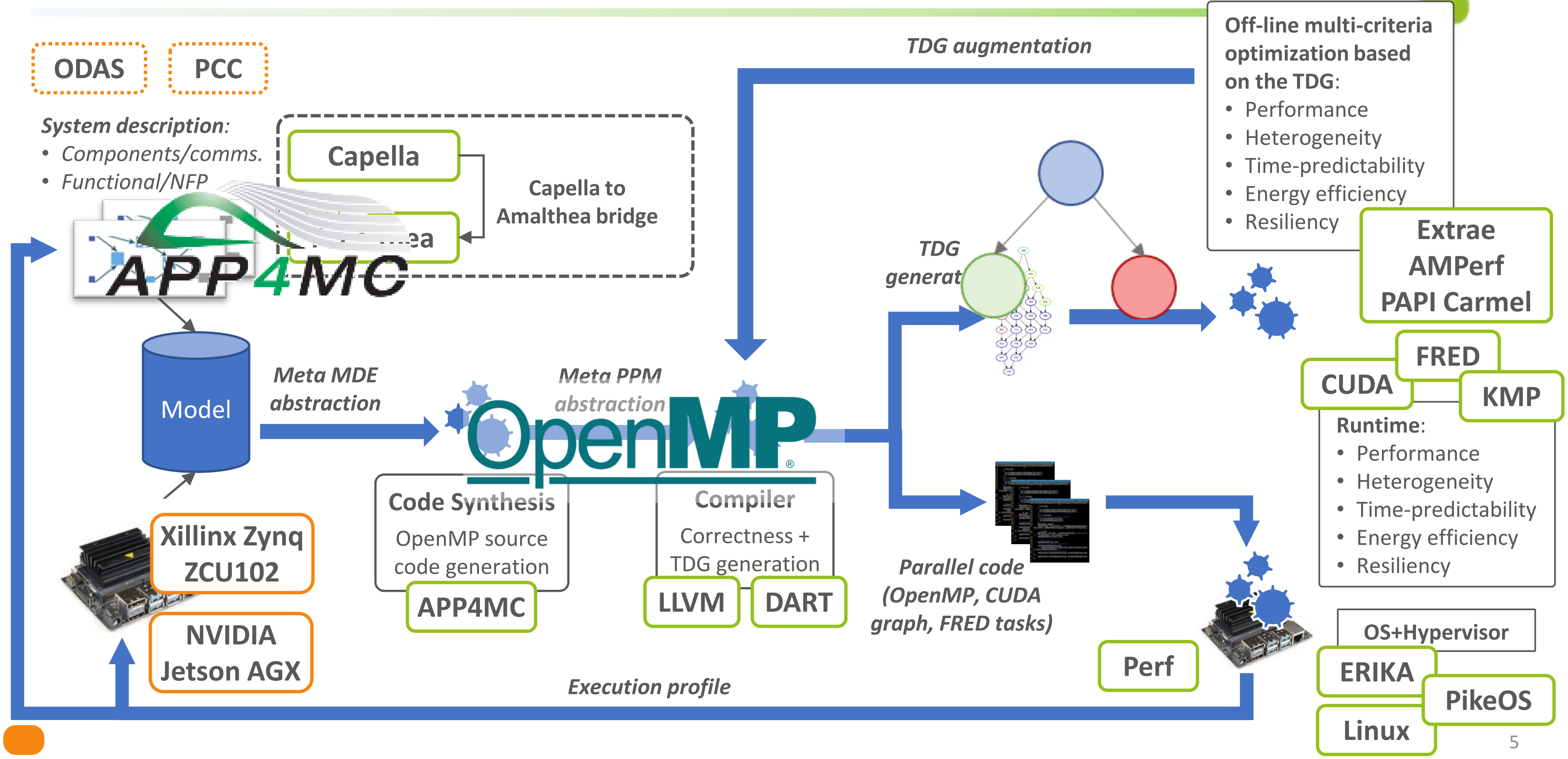
Time predictability

Energy

Multi-criteria optimization



AMPERE ecosystem workflow



Opportunities for parallelism with AMALTHEA

- AMALTHEA model (version 2.1.0)
  - Software
    - Runnables (5)
      - read_image
      - convert_image
      - analysisA
      - analysisB
      - merge_results
    - Labels (3)
      - Image
      - ResultsA
      - ResultsB
    - Tasks (1)
      - PeriodicTask
        - Activity Graph
          - call read_image
          - call convert_image
          - call analysisA
          - call analysisB
          - call merge_results

Model-to-code transformation for performance

AMALTHEA MODEL

- AMALTHEA model (version 2.1.0)
  - Software
    - Runnables (4)
      - read_and_convert
        - Activity Graph
          - read Image
          - write ResultsA
      - analysisA
      - analysisB
      - merge_results
    - Labels (3)
      - Image
      - ResultsA
      - ResultsB
    - Tasks (1)
      - PeriodicTask
        - Activity Graph
          - "Parallel" -> (Boolean) true
          - call read_and_convert
          - call analysisA
          - call analysisB
            - analysisB.variantType ◄– device_omp
          - call merge_results

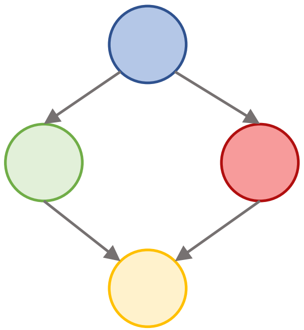
- analysisA
  - Local Labels
  - Activity Graph
    - <> Switch
      - case: "host_omp"
        - condition: OR
        - read Image
        - Ticks
        - write ResultsA
      - case: "device_omp"
        - condition: OR
        - read Image
        - Ticks
        - write ResultsA

OPENMP CODE

```
#pragma omp parallel
#pragma omp single
#pragma omp taskgraph
{
   #pragma omp task depend(out: Image)
   read_and_convert();
   #pragma omp task depend(in: Image) \
                    depend(out: ResultsA)
   analysisA();
   #pragma omp target depend(in: Image) ) \
                      depend(out: ResultsB) \
                      map(to: Image) \
                      map(from: ResultsB)
   analysisB();
   #pragma omp task depend(in: ResultsA, \
                               ResultsB)
   merge_results();
}
```

The OpenMP Taskgraph framework

```
#pragma omp parallel
#pragma omp single
#pragma omp taskgraph
{
   #pragma omp task depend(out: Image)
   read_and_convert();
   #pragma omp task depend(in: Image) \
                    depend(out: ResultsA)
   analysisA();
   #pragma omp target depend(in: Image) ) \
                      depend(out: ResultsB) \
                      map(to: Image) \
                      map(from: ResultsB)
   analysisB();
   #pragma omp task depend(in: ResultsA, \
                              ResultsB)
   merge_results();
}
```



TDG: Representation of the parallel nature of an OpenMP task-based region

- Includes all the information for functional and non-funcional correctness
  - **Parallel units** and **synchronization** dependencies
  - **Characterization of the execution** of parallel units (e.g., time, energy, memory accesses)

Enables performance optimizations

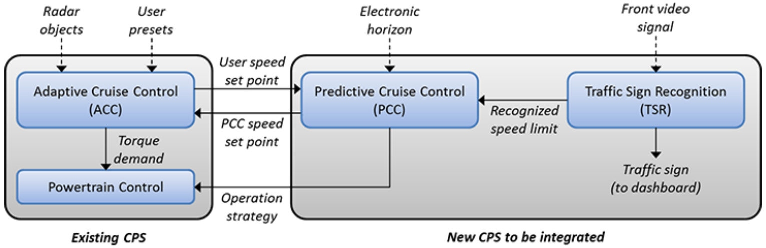
- **Parallel orchestration** fully driven by the runtime based on the TDG:
  - **Avoid context switching**
  - **Reduce** the number of **instructions**
- Avoid **contention** on shared resources (e.g., task ready queues)
- Reduce the **overhead** of the runtime:
  - Task creation (tasks can be preallocated or reused across TDG executions)
  - Dependencies resolution is no longer needed

Enables static analysis techniques

- **Correctness** of the parallelization (e.g., race free)
- **Timing analysis** for predictable execution

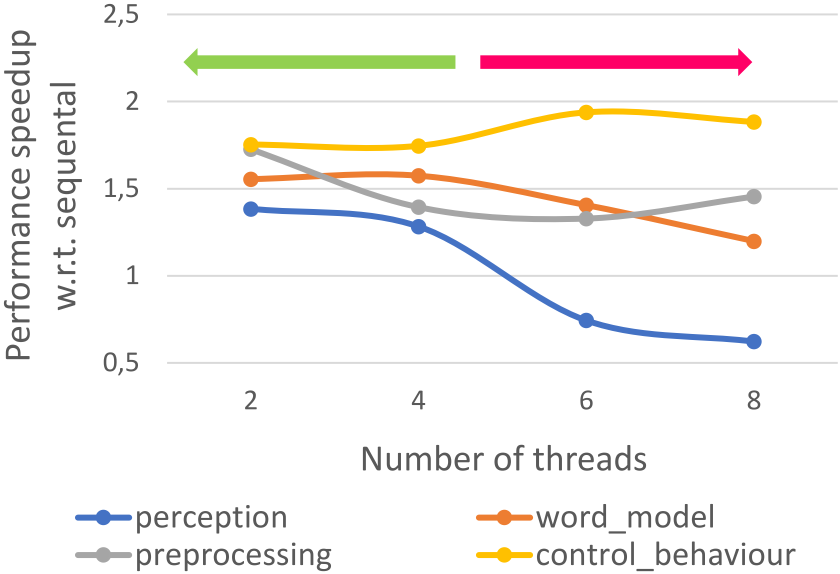
Taskgraph: A Low Contention OpenMP Tasking Framework. C. Yu, S. Royuela, E. Quiñones. Transactions of Parallel and Distributed Systems (TPDS). 2023.

Performance evaluation on the PCC use case (CPU)



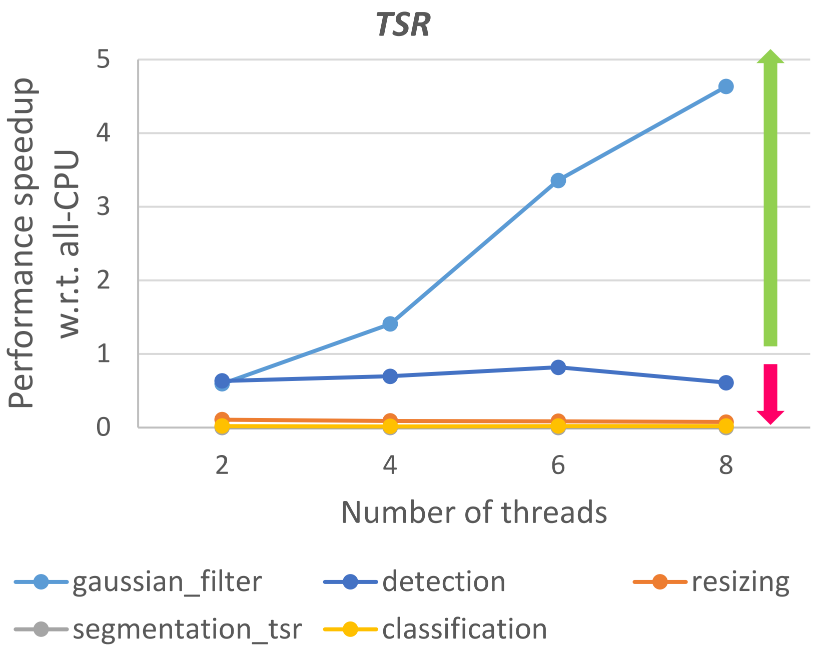
	AMALTHEA tasks	Tasks w. inter-runnable parallelism	Granularity
ACC	**6**	**6**	**~10^4 µs**
ECM	22	22	~10^1 µs
PCC	3	2	~10^1 µs
TSR	8	1	~10^1-10^2 µs

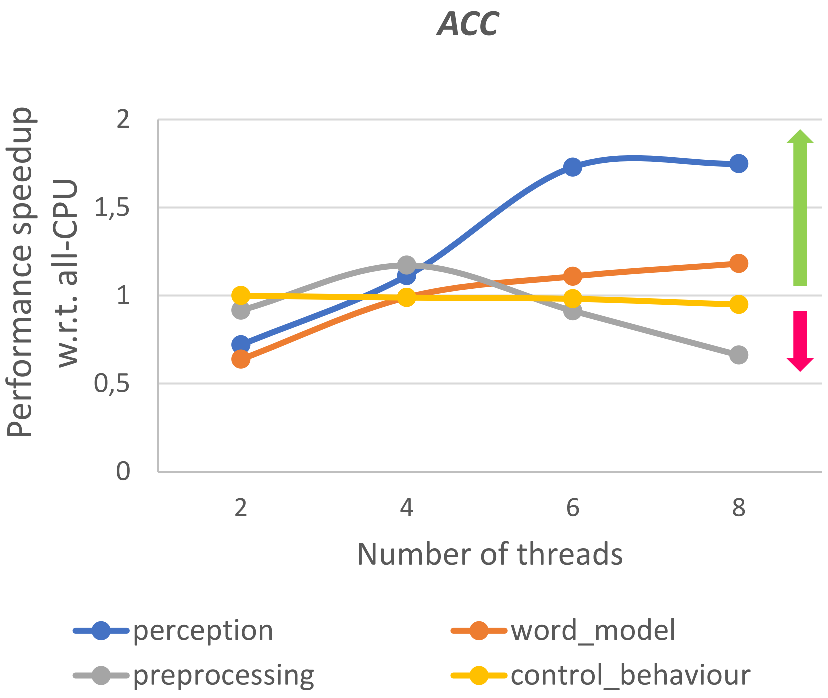
Performance speedup parallelizing the ACC component with 2 to 8 OpenMP threads



Performance evaluation on the PCC use case (GPU)

Performance speedup parallelizing the ACC component with 2 to 8 OpenMP threadsand sending TSR to the GPU





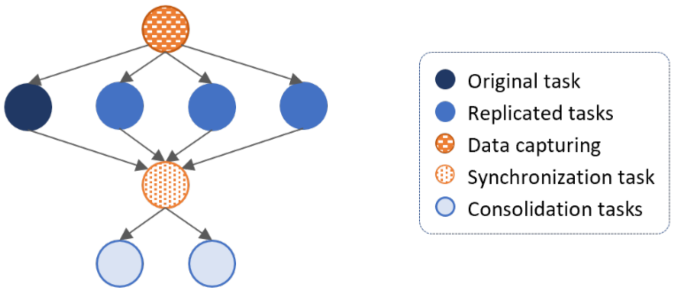
Resilience through software replication

- **Replication based on ASIL/SIL**
- **Parametrization:**
  - In the clause:
    - *Number* of replicas
    - *Variable:function* tuple used to check the results
    - *Type* of replication: *spatial*, *temporal* or *spatial_temporal* defines the type of replication.
  - At compilation time:
    - *MooN safety architecture*

Generated code:

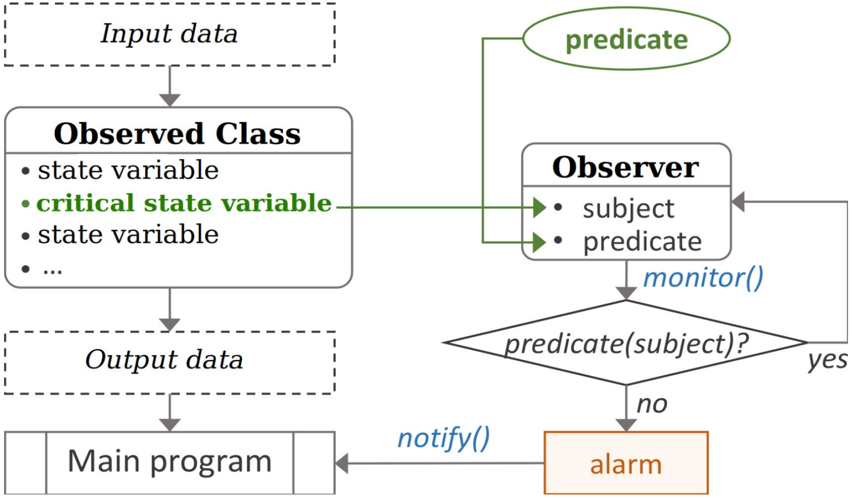
```
int consolidation_function(int* a_original, int* a_replicated) {
    return (*a_original == *a_replicated);
}

void foo (void) {
    int a = ...;
    #pragma omp task replicated(3, (a:consolidation))
    {...}
}
```

TDG (spatial replication):

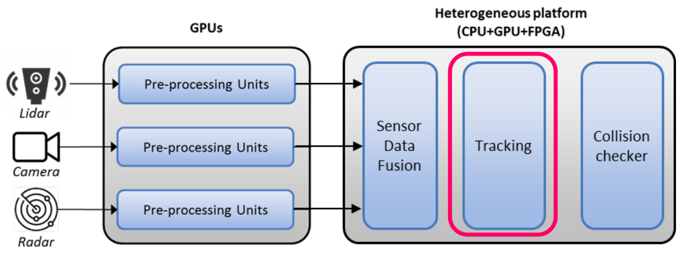


Resilience through proactive monitoring

- **General** and **lightweight software technique** for proactive monitoring based on the **observer design pattern**.
- **Main features:**
  - Early detection of transient software faults, to avoid **silent errors** that may lead to system malfunctioning
  - **Critical internal variables** can be monitored by external code in a **minimally coupled** fashion
  - **Correctness-checking mechanisms** can use predicates implemented as external functions

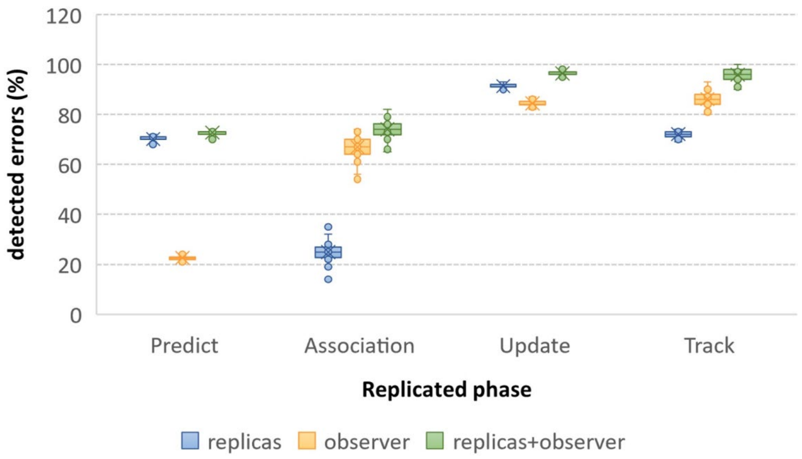


Replication evaluation

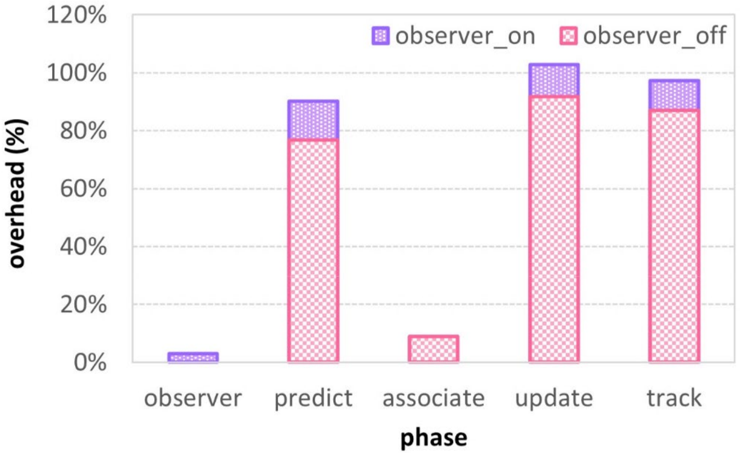


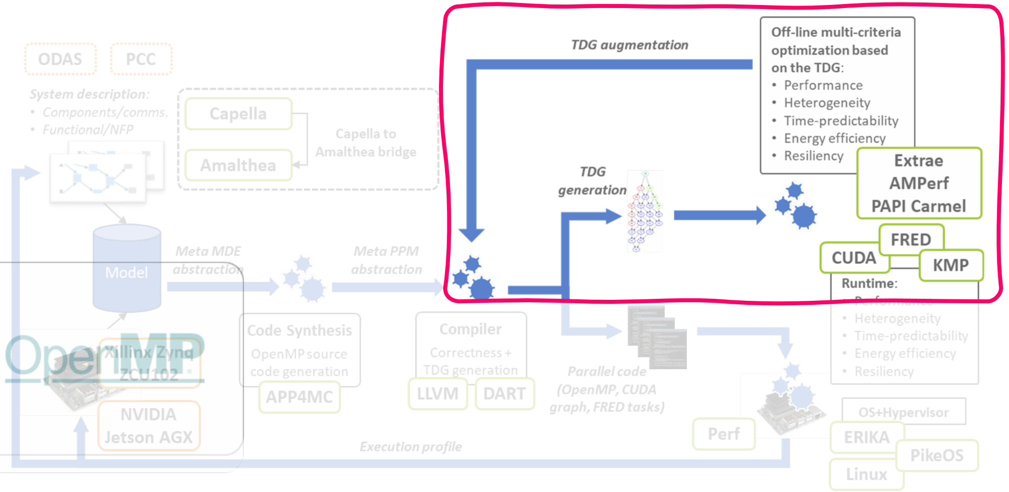
- 3 phases, i.e., *predict*, *association*, and *update*, included in the *track* phase
- Different data-sets, i.e., scattered, crowded, and inflated

Accuracy



Overhead





Performance

Resilience

Model-to-code transformation

Time predictability

Energy

Multi-criteria optimization

A Model-driven development framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimisation

The AMPERE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871669