Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PPC Predictable Parallel Computing

Task-based Parallel Programming Models: The Convergence of High-Performance and Edge Computing Domains

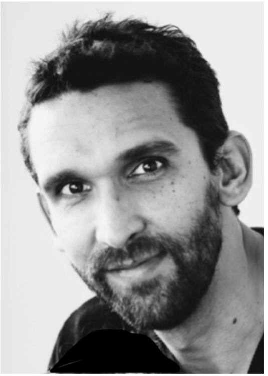
Eduardo Quiñones
{eduardo.quinones@bsc.es}

Rising STARS
www.risingstars-project.eu

AMLE Summer School 2023
September 20, 2023

EXTRACT
A distributed data-mining software platform for extreme data across the compute continuum

A bit of me…

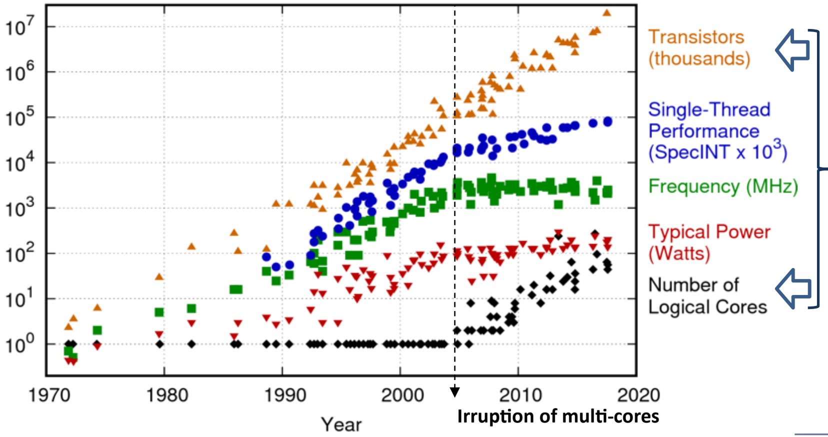


- PhD on Computer Science at Technical University of Catalonia (UPC) in 2009
- Team Leader of the Predictable Parallel Computing Research Group at Barcelona Supercomputing Center
  – Job positions available!! ;)
- Founder and CTO of TalpTech, a Startup company that provides edge computing solutions to precision agriculture

Agenda

1. The need of parallel programming models: OpenMP
2. Modelling a real-time system with OpenMP
3. Main Factors Impacting Parallel Execution
4. Runtime optimizacion for real-time systems

Heterogeneous and Parallel Computing



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Heterogeneous and Parallel computing becomes key to cope with performance requirements

Massively parallel systems that operate **as fast as possible**

Genomics

Weather

Big data

Avionics

Space

Automotive

Network of HW/SW components that **must** operate **correctly** in response to its inputs from both **functional** and **non-functional** perspectives

HPC Domain (~300W)

NVIDIA A100
(GPU-based)

AMD Instinct™ MI
(GPU-based)

Intel® Xeon® Series
(40-core)

AMD EPYC™ Series
(up to 64-core)

Embedded Domain (~10-20W)

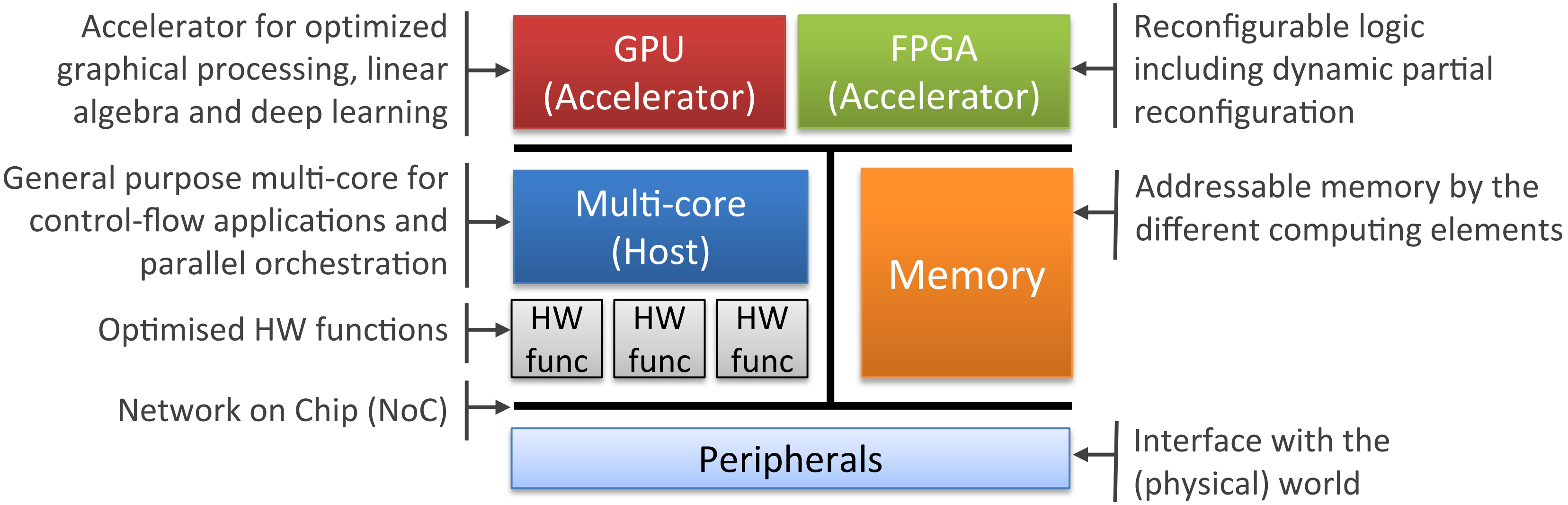
NVIDIA Jetson Family
(GPU-based)

Kalray MPPA Coolidge
(80-core fabric)

Xilinx Versal
(FPGA-based with DFX)

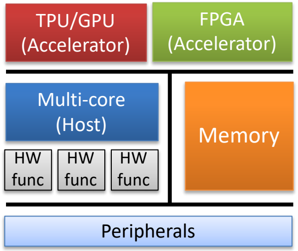
Heterogeneous and Parallel Computing

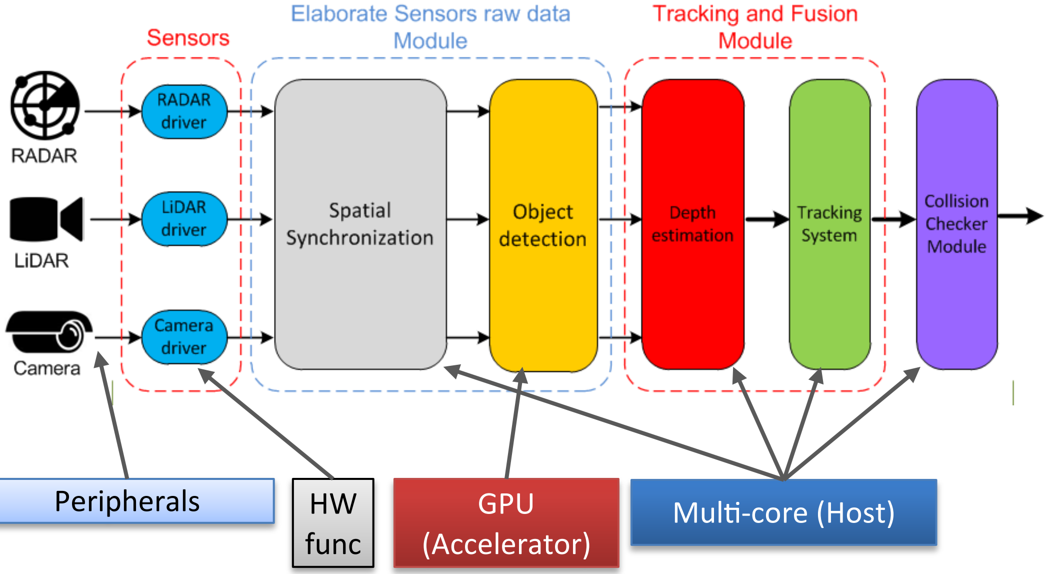
Host-centric paradigm: The parallel computation is orchestrated by the general-purpose multi-core



The example of the collision detection

Used in Adavanced Driving Assistant System (ADAS) and autonomous vehicles to identify objects (perception) and detect potential collisions





Heterogeneous and Parallel Computing

Performance: complex computations at high speed

Real-time: end-to-end response time within budget

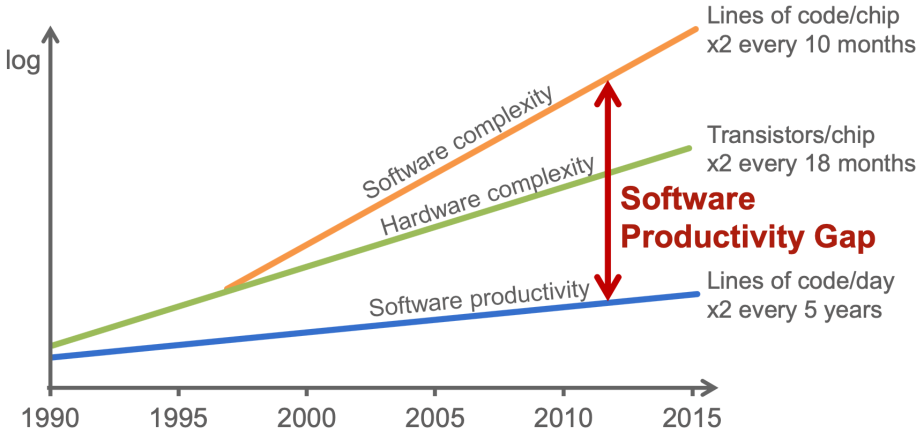
Power/Thermal: energy/temperature within budget

Safety: guarantee correctness and integrity of operation

Security: prevent external elements from affecting correctness and integrity

The SW Productivity Gap

1. *Efficiently exploit parallelism* and achieve the required performance
2. *Reason* about the functional and non-functional correctness



Source: ITRS & Hardware-dependent Software, Ecker et al., Springer

HPC Domain (~300W)		**Embedded Domain** (~10-20W)	
	**NVIDIA A100** (GPU-based)		**NVIDIA Jetson Family** (GPU-based)
	**AMD Instinct™ MI** (GPU-based)		**Kalray MPPA Coolidge** (80-core fabric)
	**Intel® Xeon® Series** (40-core)		**Xilinx Versal** (FPGA-based with DFX)
	**AMD EPYC™ Series** (up to 64-core)		

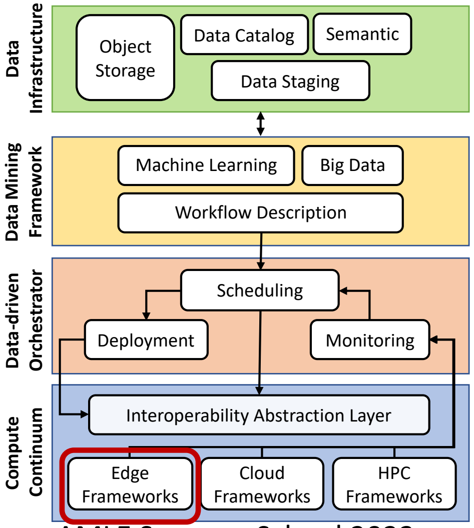
Parallel programming models are key!

OpenMP

Like a duck to water!

The Importance of Compute Continuum

- Set of computing resources to handle the complete lifecycle chain of data collected across highly distributed and heterogeneous sources
  - **Edge Computing** to reduce data transmission latencies, minimize security risks and provide data privacy, and reduce energy consumption
  - **HPC** to support massive parallel processing capabilities and acceleration features
  - **Cloud Computing** to provide highly scalable storage systems and on-demand analytics technologies



Parallel Programming Model Support

EXTRACT

A distributed data-mining software platform for extreme data across the compute continuum

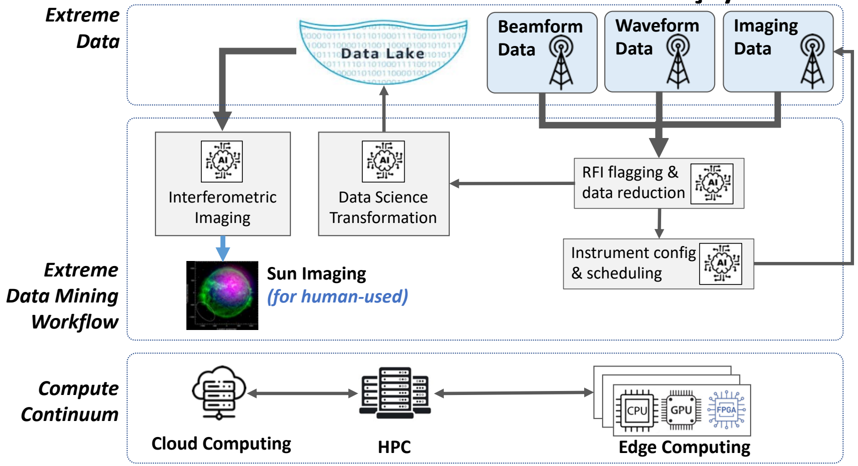
www.extract-project.eu

The Importance of Compute Continuum

- Set of computing resources to handle the complete lifecycle chain of data collected across highly distributed and heterogeneous sources
  - **Edge Computing** to reduce data transmission latencies, minimize security risks and provide data privacy, and reduce energy consumption
  - **HPC** to support massive parallel processing capabilities and acceleration features
  - **Cloud Computing** to provide highly scalable storage systems and on-demand analytics technologies

Nenufar radio telescopes at Nançay

Transient Astrophysics with a Square Kilometer Array Pathfinder (TASKA)



Parallel Programming Model Support

EXTRACT
A distributed data-mining software platform for extreme data across the compute continuum
www.extract-project.eu

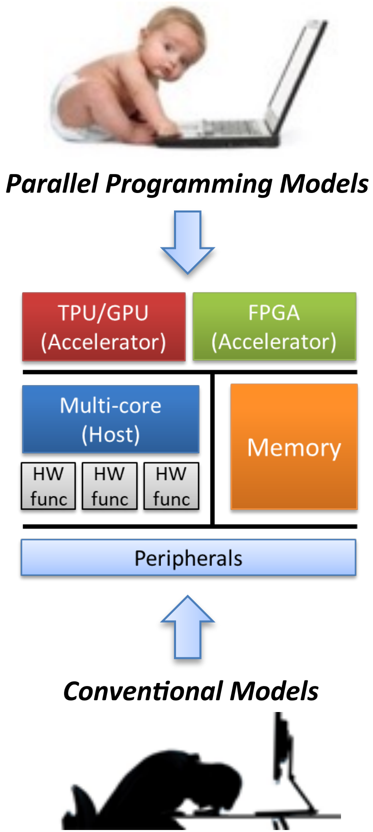
Parallel Programming Models for productivity

Parallel programming models …

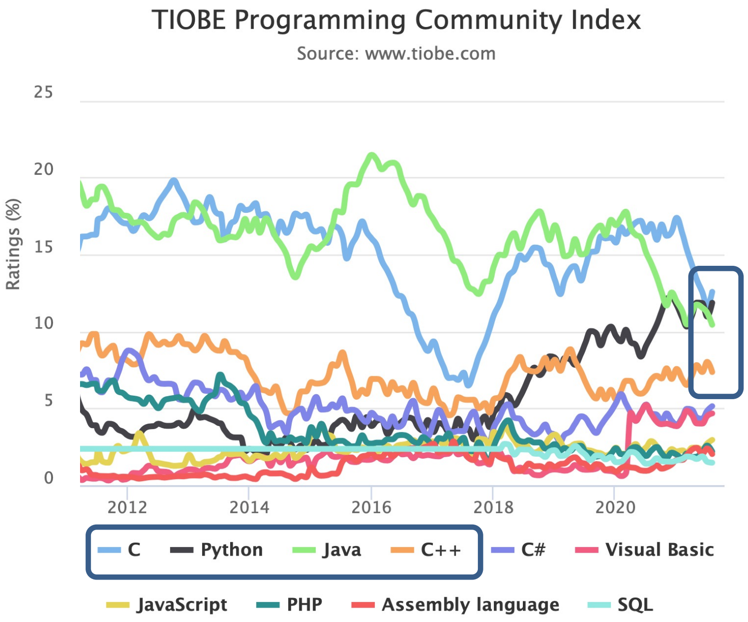
- expose *parallelism* in an easy way,
- *abstract* the complexities of the platform.

The objective is to provide **productivity**:

- *Programmability*. Simple yet flexible to define parallelism without considering architectural details
- *Portability*. Code is valid in different platforms
- *Performance*. Compiler and runtime mechanisms that exploit the performance of the platform



Parallel Programming Models for productivity



Model	Base Language	Type of PPM	Type of architect	Type of Parallelism
CUDA	C/C++, Python	HW-centric	NVIDIA GPU	Struct/ Unstruct
OpenCL	C/C++	App-centric	GPU/ FPGAs	Struct
OpenMP	C/C++	Parallel-centric	Shared mem	Struct/ **Unstruct**
Pthreads	C/C++	Parallel-centric	Shared mem	Unstruct
MPI	C/C++, Python	Parallel-centric	Distributed mem	Unstruct
COMPSs	C++, Java Python	Parallel-centric	Distributed mem	**Unstruct**
Spark	Java, Python	Parallel-centric	Distributed mem	Struct
Ray	C++,Java Python	Parallel-centric	Distributed mem	Unstruct

Parallel programming models supporting tasking

Our proposal: OpenMP

- **Mature language** constantly reviewed (last release Nov 2021, v5.2)
  - De-facto industrial *standard* in HPC for shared-memory systems.
  - Active research community with an **increasing interest** on the *embedded domain.*

- **Productivity**
  - *Performance*
    - Support for different types of in-node parallelism and accelerator devices.
    - Performance analysis tools.
  - *Portability*
    - Supported by many chip vendors (Intel, IBM, ARM, NVIDIA, TI, Gaisler, Kalray).
  - *Programmability*
    - Interoperability with other programming models (e.g., CUDA, OpenCL).
    - Allows incremental parallelization and can be easily compiled sequentially.

OpenMP tasking model

Sequential version

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

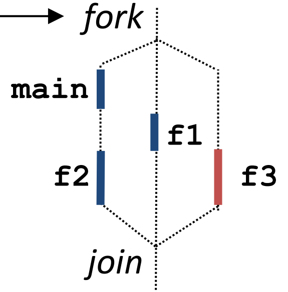
Executes on the host (f1, f2)

Executes on the accelerator (f3)

OpenMP version

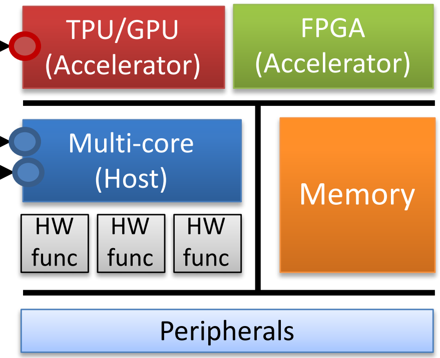
```
void main() {
    #pragma omp parallel
    #pragma omp single
    {
        int x,y;
        #pragma omp task depend(out:x,y)
        { f1(&x,&y); }
        #pragma omp task depend(in:x)
        { f2(x); }
        #pragma omp target map(to:y) depend(in:y)
        { f3(y); }
    }
}
```

1. Open parallelism



2. Tasks executed on the host

3. Tasks executed on the host and accelerator when f1 completes



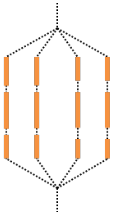
OpenMP tasking model

Expressiveness:

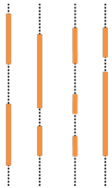
– Exposes *what* to do in parallel rather than *how* to do it
– The parallel framework orchestrates the execution

Support for different *types of parallelism*:

– *Structured*
  - regular patterns in the form of parallel loops
  - **`taskloop`** construct



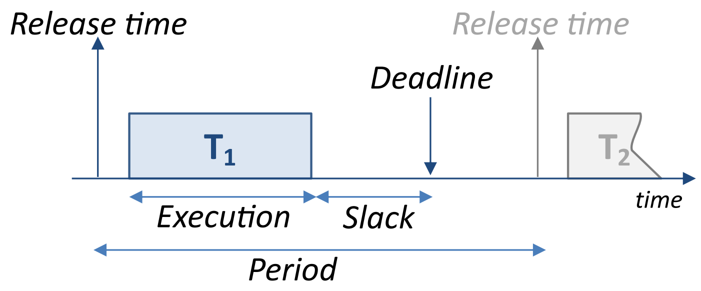
– *Unstructured*
  - irregular patterns that may change
  - **`task`** construct and **`depend`** clauses



Computation is not fully controlled by the programmer but by the parallel framework

"I'm a software engineer, so I can confirm it works by magic."

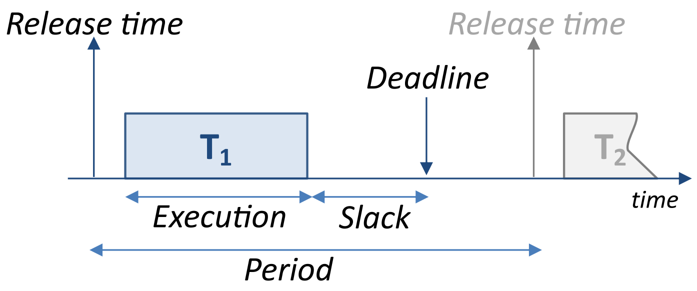
Modeling a RT system with OpenMP tasking



- **(Sub)system**: set of concurrent tasks
- **RT-Task (T_x)** → ***OpenMP tasks*** **(task)**
  - Recurrent: periodic (deadline/period), sporadic → ***Not supported***
  - Priority → ***(Prescriptive) Priorities*** **(priority)**
  - Preemption (non/limited/fully preemptive) → ***Task scheduling points*** **(taskyield)**
  - Fine-grain parallelism/heterogenous computation → ***Nested parallelism*** **(task/target)**

Modeling a RT system with OpenMP tasking

- **(Sub)system**: set of concurrent tasks



- **RT-Task (T_x)**
  - Recurrent: periodic (deadline/period), sporadic → ***Not supported***
  - Priority
  - Preemption (non/limited/fully preemptive)
  - Fine-grain parallelism/heterogenous computation

Time and event-based OpenMP tasks

Application-based control loop

No OpenMP runtime support needed

```
#pragma omp parallel
#pragma omp single nowait
while(1)
{
  if(get_time()%100) {
    #pragma omp task …
    rt_task_1();
  }
  …
  if(get_time()%200) {
    #pragma omp task …
    rt_task_N();
  }
}
```

Runtime-based control loop*

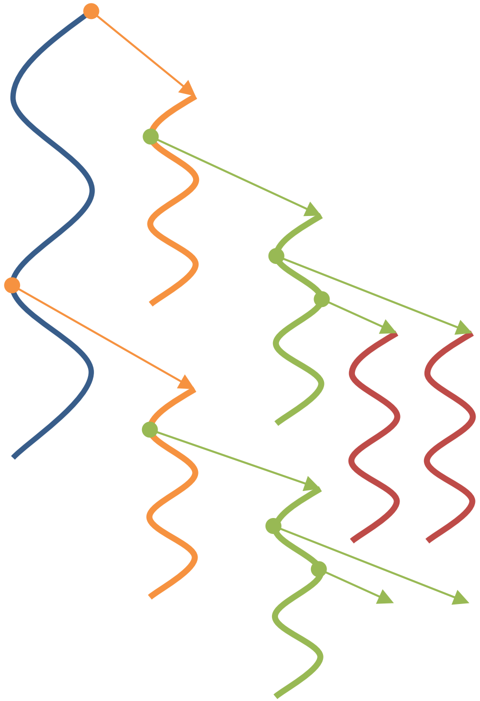
OpenMP runtime support

```
#pragma omp parallel
#pragma omp single nowait
{
  #pragma omp task event(periodic:100)
  rt_task_1();
  #pragma omp task event(sporadic:event1)
  rt_task_N();
  #pragma omp task event(sporadic:event2)
  rt_task_N();
}
```

* M.A. Serrano, S. Royuela, E. Quiñones, *Towards an OpenMP Specification for Critical Real-Time Systems*. IWOMP 2018.

Time and event-based OpenMP tasks

OpenMP runtime



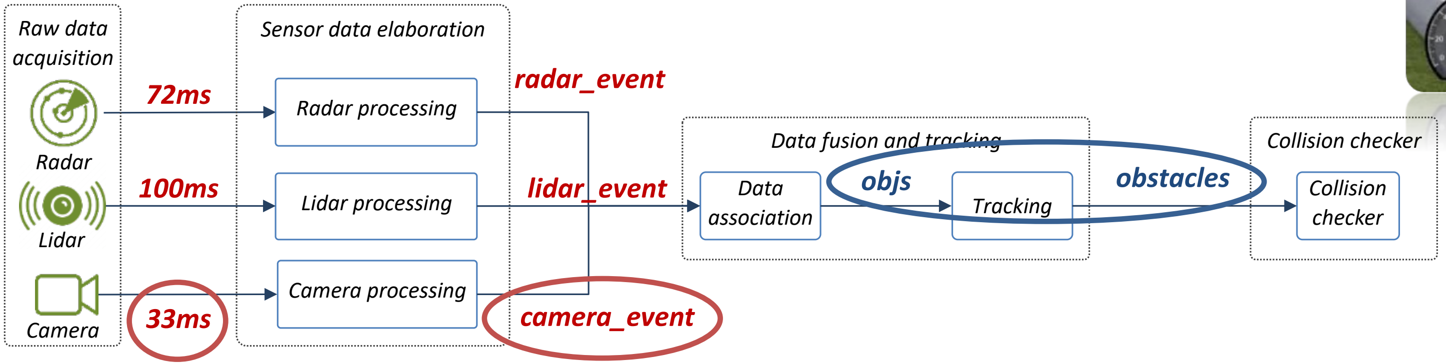
Runtime-based control loop*

OpenMP runtime support

```
#pragma omp parallel
#pragma omp single nowait
{
  #pragma omp task event(periodic:100)
  rt_task_1();
  #pragma omp task event(sporadic:event1)
  rt_task_N();
  #pragma omp task event(sporadic:event2)
  rt_task_N();
}
```

* M.A. Serrano, S. Royuela, E. Quiñones, *Towards an OpenMP Specification for Critical Real-Time Systems*. IWOMP 2018.

Automotive example



```
#pragma omp task event(periodic:33)
camera_processing();
omp_fulfill_event(camera_event);
… // Processing all sensors
#pragma omp task event(sporadic:camera_event) depend(out:objs)
data_association();
#pragma omp task depend(in:objs, out:obstacles)
tracking();
#pragma omp task depend(in:obstacles)
collision_checker();
```

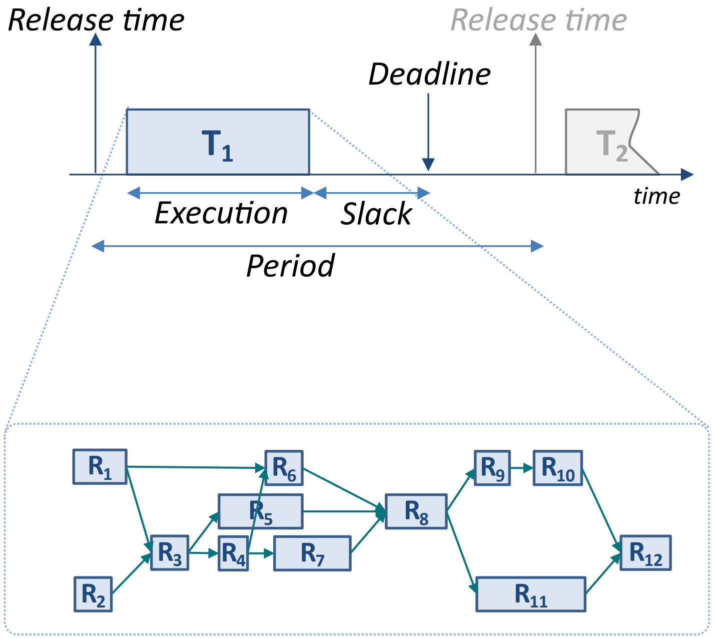
New instance of the (persistent) task every **33ms**

New instance of the (persistent) task every time **camera_event** *occur*

New instance of the tasks when **objs** *and* **obstacles** *data dependencies are honored*

Modeling a real-time system

- **(Sub)system**: set of concurrent tasks
- **RT-Task (T_x)**
  - Recurrent: periodic (deadline/period), sporadic
  - Priority
  - Preemption (non/limited/fully preemptive)
  - Fine-grain parallelism/heterogenous computation (nested parallelism)
    - Described as functionalities (Rx)
    - Execution time (WCET)
    - Accesses labels



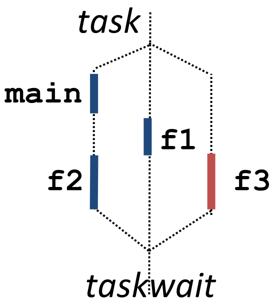
Task Dependency Graph (TDG)

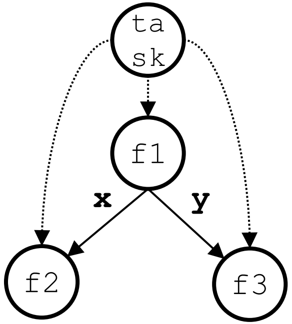
Main Factors Impacting Parallel Execution: TDG

1. Parallel structure of the application (including data usage): **<u>Task Dependency Graph</u> (TDG)**
2. The execution and memory model: The **<u>Runtime Scheduler</u>** responsible of mapping task to parallel units

```
#pragma omp task event(periodic:33)
{
  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);
  #pragma omp taskwait
}

#pragma omp task event(sporadic:object_event)
…
```



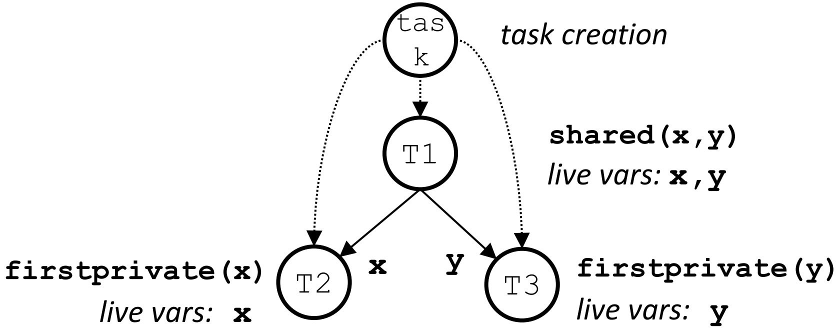


Task Dependency Graph (TDG)

A representation of the parallel nature of a given OpenMP region, extracted by means of compilation and runtime methods [1]

- Includes all the information for functional and non-funcional correctness
  - **Parallel units** and **synchronization** dependencies
  - **Liveness analysis of variables** and **data-sharings** involved in the parallel execution
- Independent from the targeted parallel platform (but can include HW dependent information)
  - **Execution characterisation** of parallel units (e.g., time, energy, memory behaviour)

```
#pragma omp task event(periodic:33)
{
  int x,y;
  #pragma omp task depend(out:x,y) // T1
  f1(&x,&y);
  #pragma omp task depend(in:x) // T2
  f2(x);
  #pragma omp target map(to:y) depend(in:y) //T3
  f3(y);
  #pragma omp taskwait
}
```



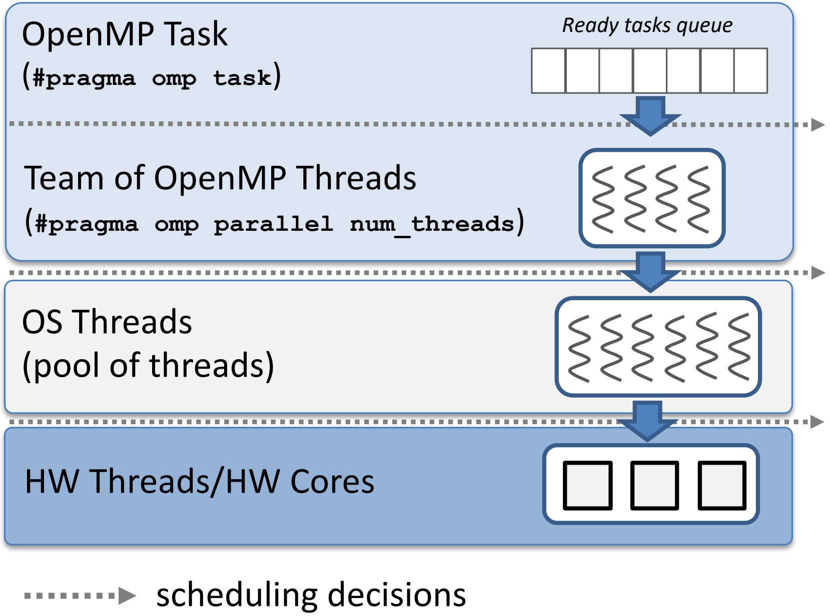
[1] *Supported by LLVM*

Time behavior of OpenMP tasks

Timing behaviour depends on the **mapping** between **parallel units** to **computing resources**

In the scope of OpenMP:

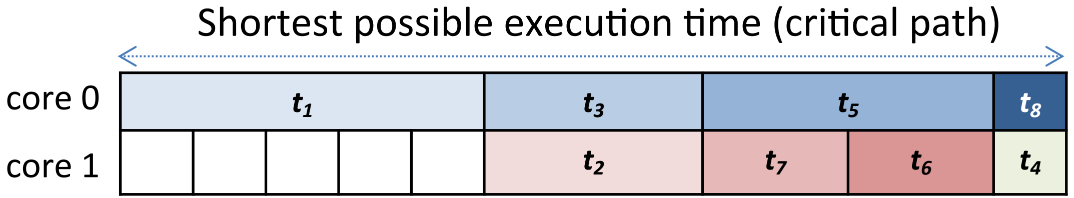
1. Parallel structure of the application
   - ✓ TDG
2. Scheduler(s) responsible of mapping OpenMP tasks to cores/accelerators
   - ✓ Fix OpenMP threads to HW threads: OMP_PLACES, OMP_PROC_BIND
   - ✓ Fix tasks to threads: *tied* tasks

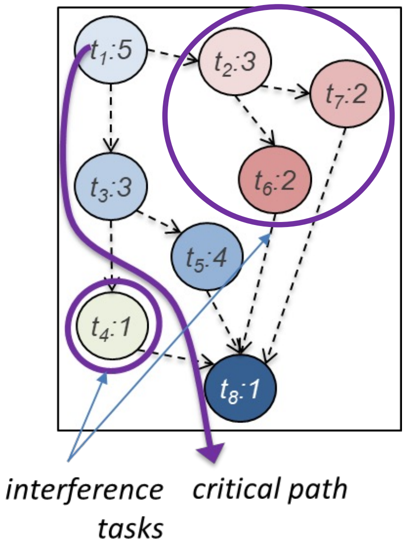


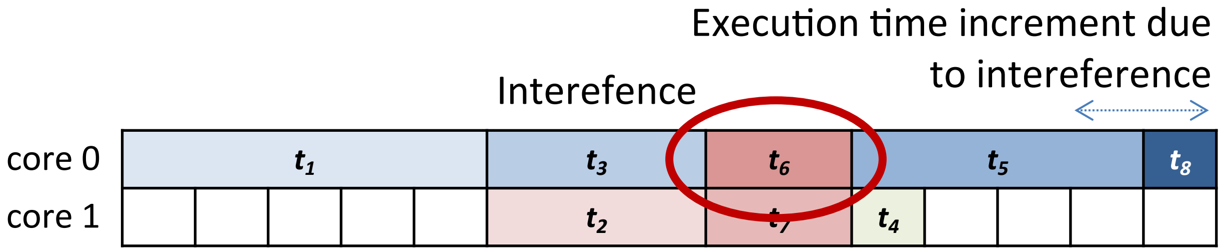
Time predictability

The execution time of a TDG is determined by:

1. Execution of OpenMP tasks within the **critical path**
2. **Interferences** of the rest of OpenMP tasks on the critical path
3. **Interferences** on HW/SW resources with other applications

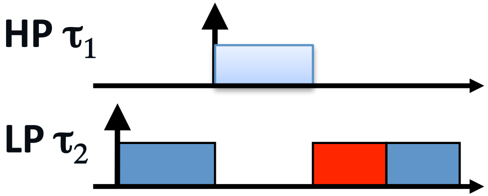




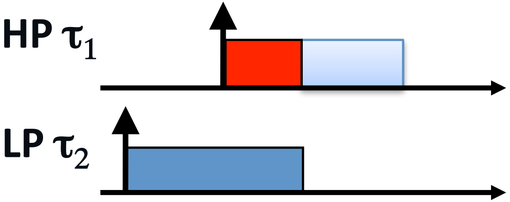


Preemption strategy

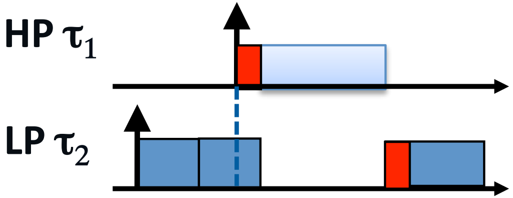
There exist three preemption strategies:



Fully preemptive scheduling



Non-preemptive scheduling



Limited preemption (cooperative) scheduling

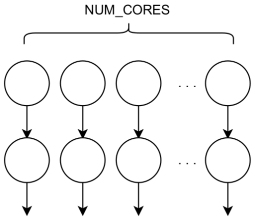
OpenMP tasks define a **limited preemption scheduling strategy**

- Preemptions occur at predefined points (a.k.a. *task scheduling points*)
- A priority level can be set to OpenMP tasks

The implementation of the runtime scheduler is *implementation-defined*

OpenMP Tasking Model Overhead

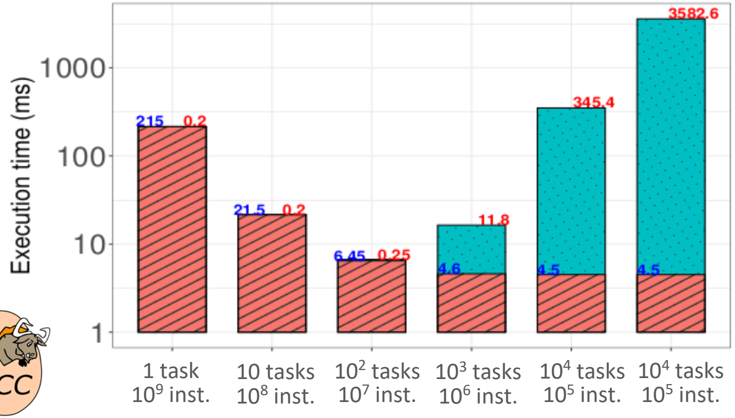
```
for (int i=0; i<nTasks; ++i) {
  #pragma omp task depend(out:a[i%nCores])
  cpu_bound_fn();
}
```



$Computation = \frac{serial\_time}{\#cores}$

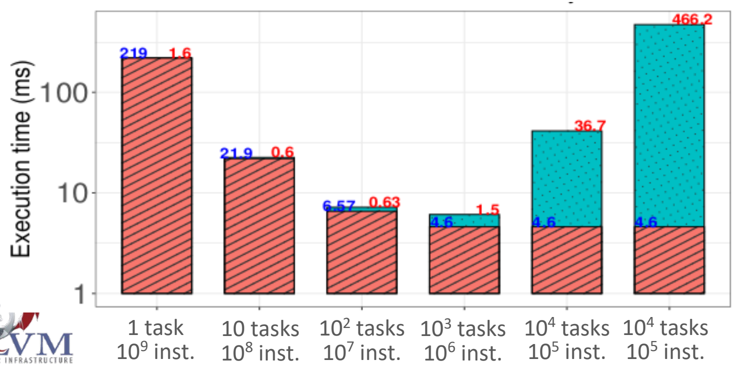
$Overhead = Total\_time - Computation$

Fixed workload: $\uparrow$ *tasks* $\Rightarrow \downarrow$ *granularity*



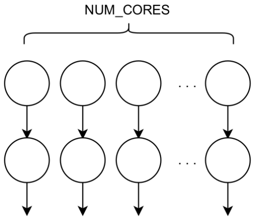
Causes:

1. Contention on shared resources
   - Less pronounced in LLVM due to distributed task queues and fine-grained locking
2. Parallel orchestration
   - Task creation and synchronization



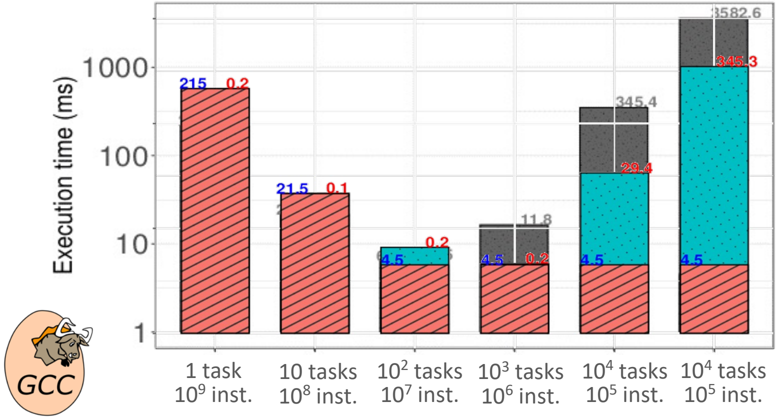
Leveraging the Task Dependency Graph

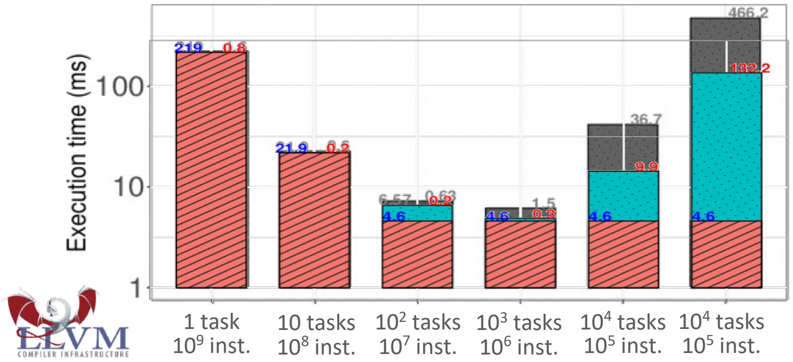
```
#pragma omp taskgraph
for (int i=0; i<nTasks; ++i) {
  #pragma omp task depend(out:a[i%nCores])
  cpu_bound_fn();
}
```



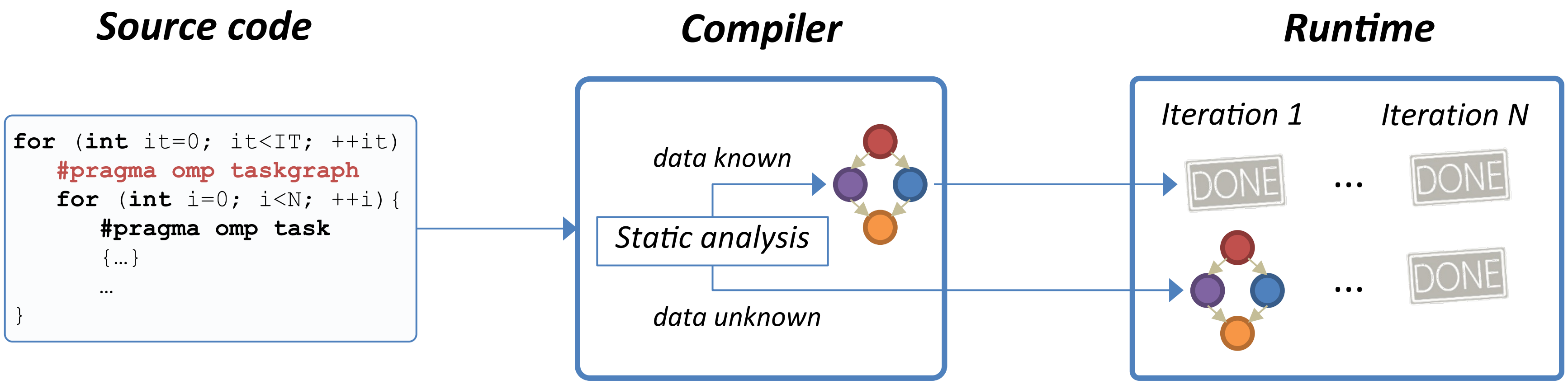
The **`taskgraph`**

- Reduces/eliminates overhead due to task orchestration and dependency resolution
- Used to instantiate and orchestrate tasks



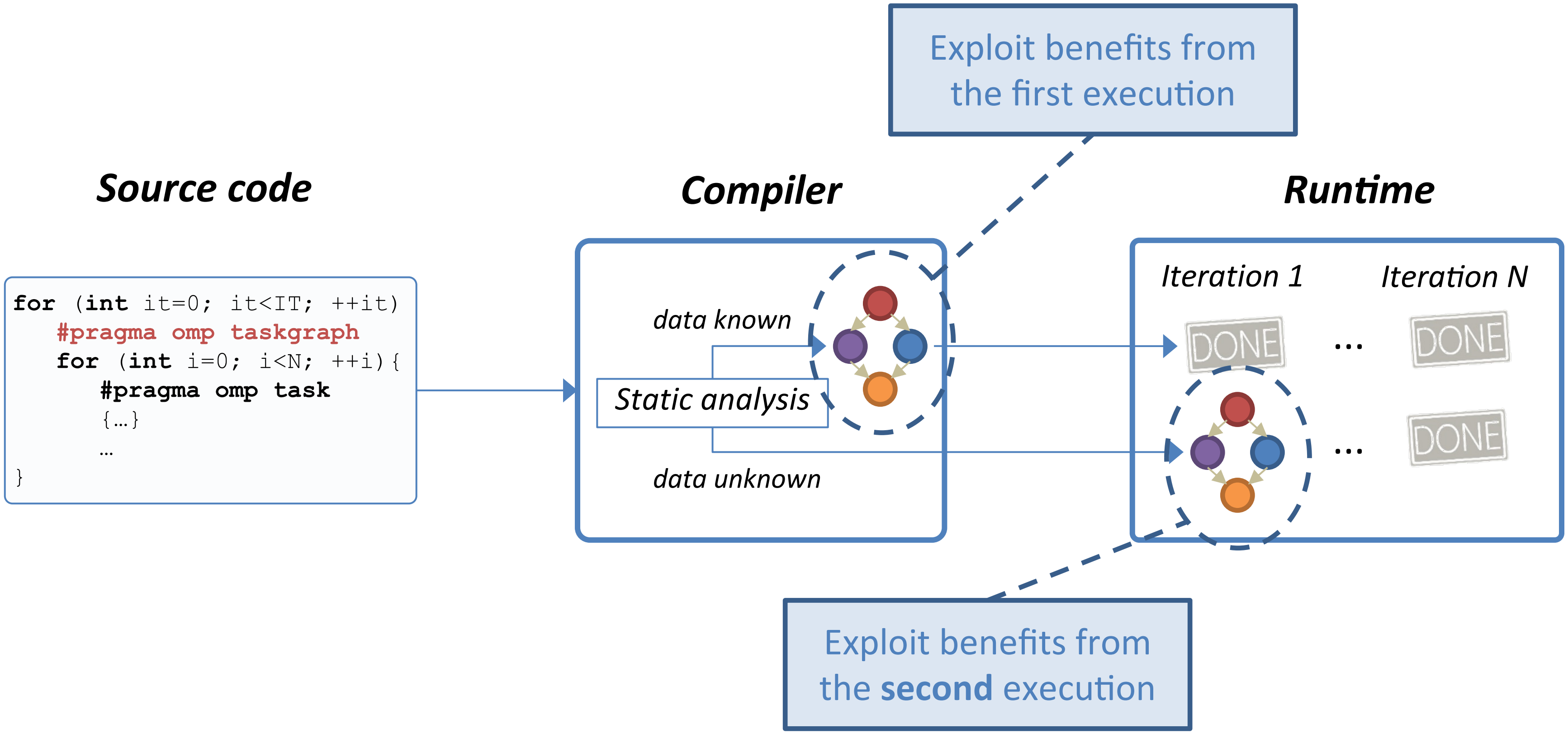


TDG-driven framework



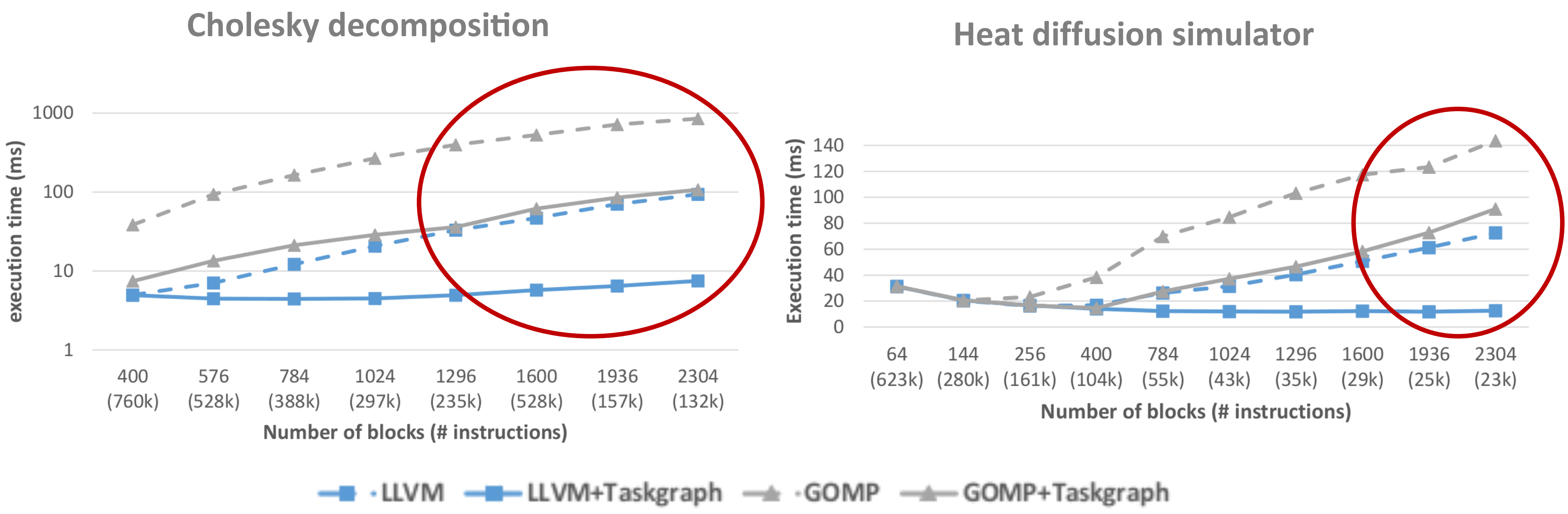
Chenle, Y, Royuela, S, and Quiñones, E. **Enhancing OpenMP Tasking Model: Performance and Portability,** In *International Workshop on OpenMP (IWOMP)*. 2021.
Chenle, Y, Royuela, S, and Quiñones, E. **Taskgraph: A Low Contention OpenMP Tasking Framework,** In https://arxiv.org/abs/2212.04771. 2022.

TDG-driven framework

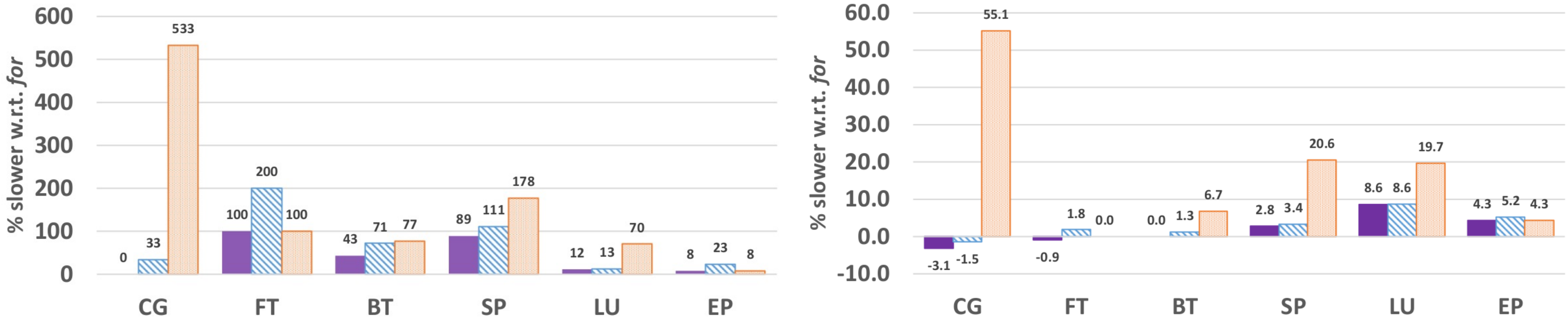


Chenle, Y, Royuela, S, and Quiñones, E. **Enhancing OpenMP Tasking Model: Performance and Portability,** In *International Workshop on OpenMP (IWOMP)*. 2021.
Chenle, Y, Royuela, S, and Quiñones, E. **Taskgraph: A Low Contention OpenMP Tasking Framework,** In https://arxiv.org/abs/2212.04771. 2022.

Optimizing Parallel Execution using the TDG



Can tasking replace threading?



(a) Problem size W

(a) Problem size C

Optimal TDG   Taskgraph   Taskloop

Overhead of Taskgraph when replacing original for with taskloop, with OMP_NUM_THREADS=48 (lower is better).

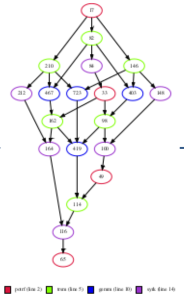
→ Penalization when number of iterations is low (small problem size)

→ Speedup close to optimal when recording is amortized

Interoperability with low-level libraries

OpenMP → CUDA Graphs

```
for (k=0; k<NB; k++) {
  #pragma omp target depend(inout: Ah[k][k])
  potrf(Ah[k][k], ts, ts);
  for (i=k+1; i<NB; i++) {
    #pragma omp task depend (in: Ah[k][k]) \
                     depend(inout: Ah[k][i])
    trsm(Ah[k][k], Ah[k][i], ts, ts);
  }
  for (l=k+1; l<NB; l++){
    for (j=k+1; j<l; j++){
      #pragma omp task depend(in: Ah[k][l]) \
                       depend(in: Ah[k][j]) \
                       depend(inout: Ah[j][l])
      gemm(Ah[k][l], Ah[k][j], Ah[j][l], ts, ts);
    }
    #pragma omp task depend(in: Ah[k][l]) \
                     depend(inout: Ah[l][l])
    syrk(Ah[k][l], Ah[l][l], ts, ts);
  }
}
```



static

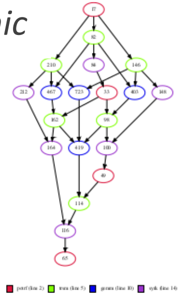
```
...
cudaGraphNode_t node_17 ;
cudaKernelNodeParams nodeArgs_17 = { 0 } ;
nodeArgs_17.func = (void *) potrf;
void * kernelArgs_17[3] = {&Ah[1][1], &ts , &ts} ;
nodeArgs_17.kernelParams = (void **) kernelArgs_17;
cudaGraphAddKernelNode(&node_17, graph[0], NULL,
                       0, &nodeArgs_17);
cudaGraphNode_t node_82 ;
cudaHostNodeParams nodeArgs_82 = {0} ;
nodeArgs_82.func = (void *) trsm;
void * hostArgs_82[4] = {&Ah[1][1], &Ah[1][1], &ts, &ts};
nodeArgs_82.kernelParams = (void **) hostArgs_82;
cudaGraphAddHostNode(&node_82, graph[0], &node_17,
                     1, &nodeArgs_82);
...
```

dynamic

```
cudaGraph_t graph;
cudaGraphExec_t instance;
If(!graphCreated){
  cudaStreamBeginCapture(stream, ...);
  ... // kernel calls
  cudaStreamEndCapture(stream, &graph);

  cudaGraphInstantiate(&instance, graph, NULL, NULL, 0);

  graphCreated=true;
}
cudaGraphLaunch(instance, stream);
cudaStreamSynchronize(stream);
```



Chenle, Y, Royuela, S, and Quiñones, E. **OpenMP to CUDA graphs: a compiler-based transformation to enhance the programmability of NVIDIA devices**, In *SCOPES*. 2020.

Interoperability with low-level libraries

OpenMP → CUDA Graphs

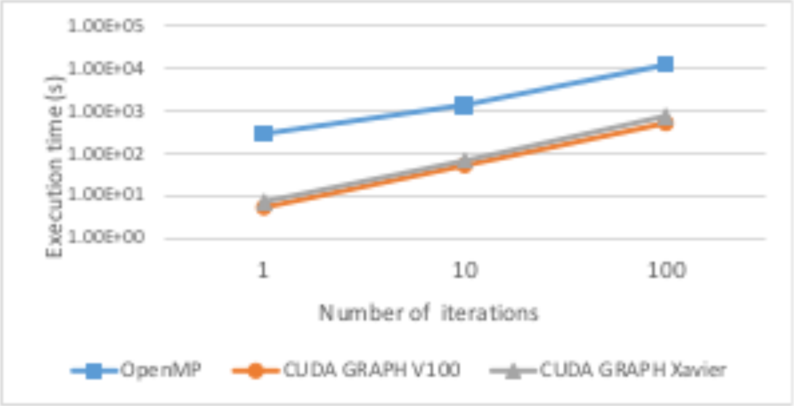
❑ Goals:

- ✓ Enhance *performance* (NVIDIA devices)
- ✓ Maintain *programmability*

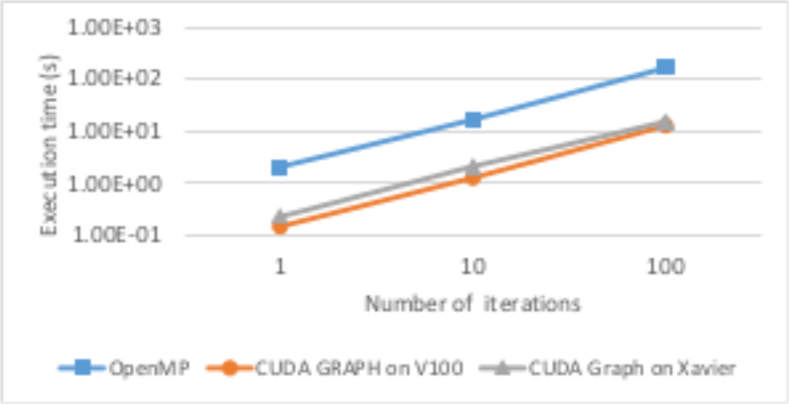
❑ Results:

- – OpenMP synchronizations take longer than CUDA graphs

saxpy

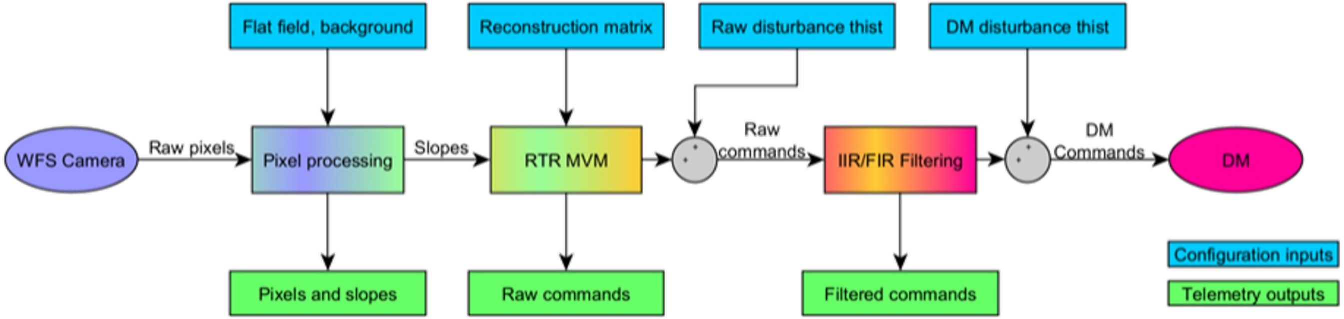


cholesky



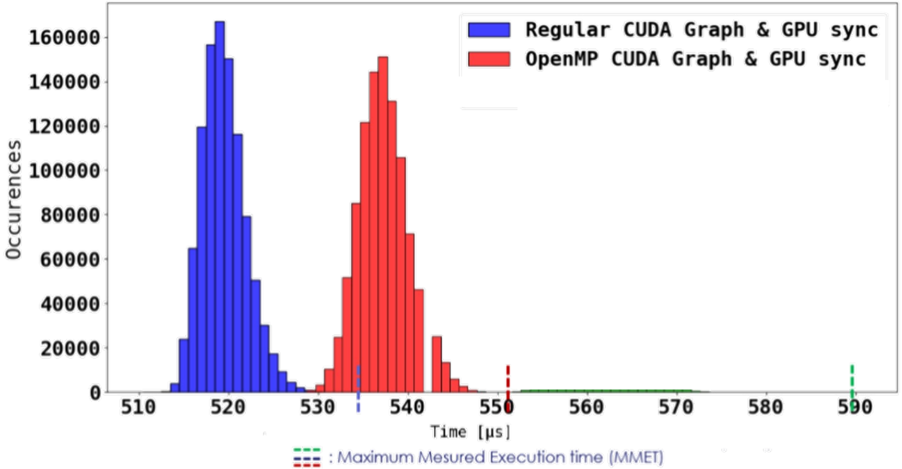
Chenle, Y, Royuela, S, and Quiñones, E. **OpenMP to CUDA graphs: a compiler-based transformation to enhance the programmability of NVIDIA devices,** In *SCOPES*. 2020.

Taskgraph for Adaptive Optics



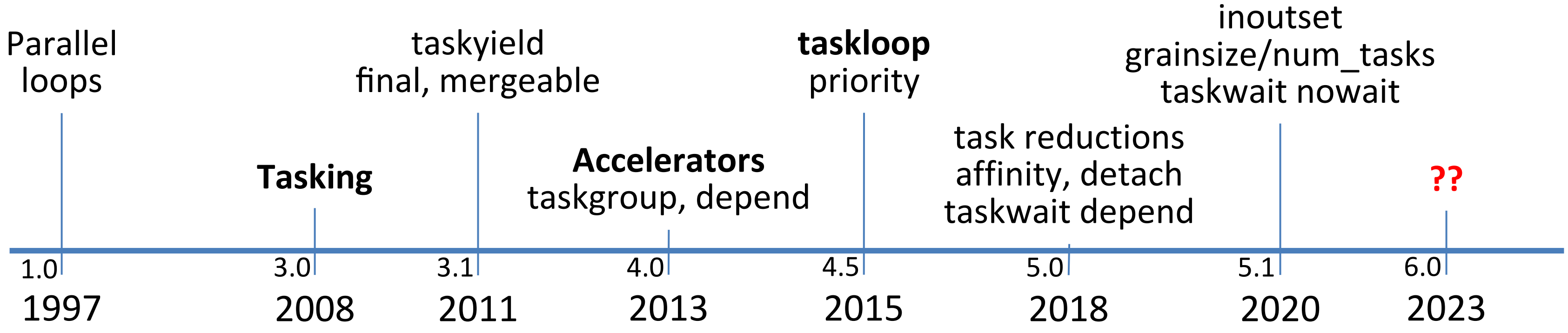
Simplified code:

```
#pragma omp parallel
#pragma omp single
for (n_iters)
  #pragma omp taskgraph
  {
    #pragma omp task depend(…)
    WFS_camera();
    #pragma omp task depend(…)
    pixel_processing();
    #pragma omp task depend(…)
    RTR_MVM();
  }
```



What happens with the OpenMP tasking model?

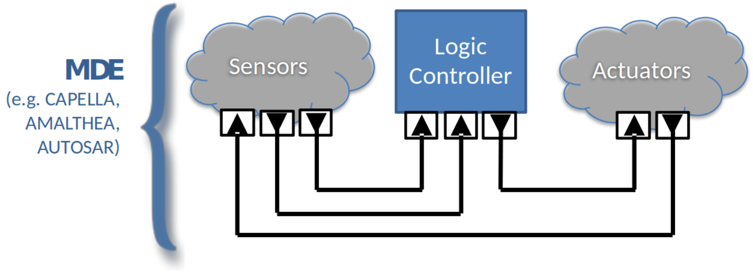
The specification clearly moves from threading to tasking (host and device):



Proposal: a new directive to expose a region that can be represented as a TDG and event-driven model

```
#pragma omp taskgraph
#pragma omp task event…
```

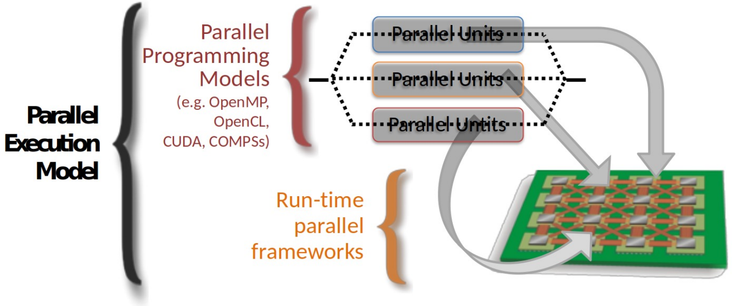
Interoperability with DSMLs



Model Driven Engineering (MDE)

1. Construction of complex systems
2. **Formal verification** of FR and NFR with **composability**
3. **Correct-by-construction paradigm** thourgh code generation
   - Suitable for single-core execution or very limited multi-core support

Gap between the MDE used for CPS and the PPM supported by parallel platforms



Parallel Programming Models

1. Mandatory for **SW productivity**
   - Programmability: Parallel abstraction hiding HW complexities
   - Portability: Compatibility multiple HW platforms
   - Performance: Efficient exploitation of HW parallel capabilities
2. **Efficient offloading** to HW acceleration devices

Interoperability with DSMLs: Automotive

1. Exploit parallelism within OpenMP (host and target) tasks
2. Exploit heterogeneity through specializations

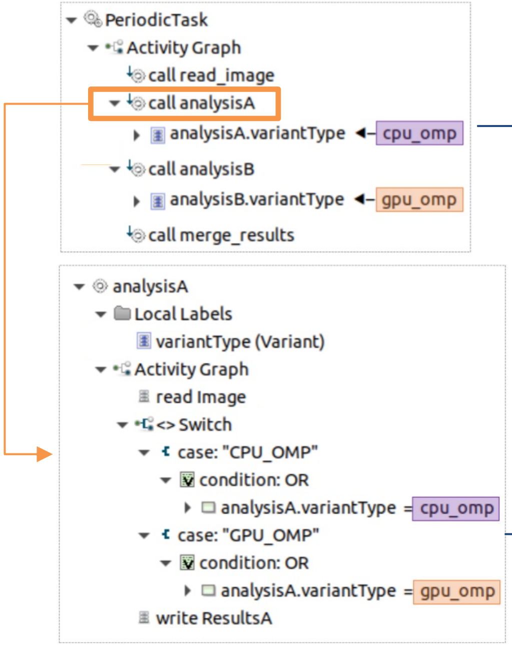
Amalthea

BOSCH

APP4MC

Automatic code generation

OpenMP




```
void PeriodicTask() {
  #pragma omp parallel
  #pragma omp single
  {
     #pragma omp task depend(out:Image)
     { read_image(); }
     #pragma omp task depend(in:Image) depend(out:ResultsA) cpu_omp
     { analysisA(); }
     #pragma omp target depend(in:Image) depend(out:ResultsA)\
                 map(to:Image) map(from:ResultsA) gpu_omp
     { analysisB(); }
     #pragma omp task depend(in:ResultsA, ResultsB)
     { read_image(); }
  }
}
```

```
void analysisA_gpu() { … }


#pragma omp declare variant(analysisA_gpu)\
              match(construct={target})\
              implementation={extension(gpu_omp)}
void analysisA() {…}
```

Home-take message

1. Real-time systems **requires parallel computation** to cope with the performance requirements of the most advanced functionalities, and…
2. … current task-based parallel programming models allows to **reasoning about functional correctness and time predictability** while removing from developers the responsibility of managing the complexity of parallel execution
3. **OpenMP** provides the level of productivity required while allowing reasoning about the functional and non-functional requirements across the **compute continuum**

OpenMP in the scope of embedded systems

Eduardo Quiñones

Head of the Predictable Parallel Computing Research Group

{eduardo.quinones@bsc.es}

Journée thématique GdR SOC2 – IRT Saint Exupéry :
« Calcul haute performance pour les systèmes embarqués »

March 14, 2023