Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OpenMP tasking:

Extensions and optimizations for performance, predictability and resilience

Sara Royuela
sara.royuela@bsc.es

PPC
Predictable Parallel Computing

OpenMP User's Monthly Telecon

October 28, 2022

Predictable Parallel Computing in OpenMP

Dr. Eduardo Quiñones
Team leader

Chenle Yu
PhD candidate

Adrian Munera
PhD candidate

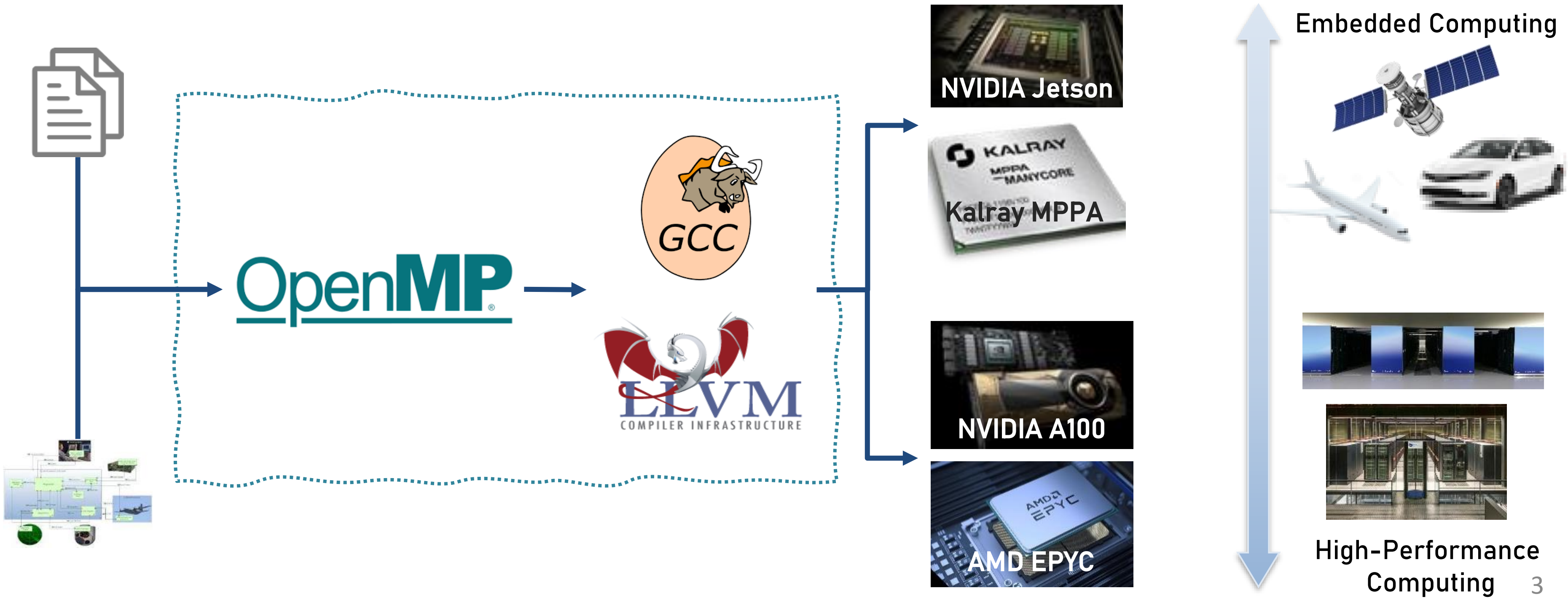
Dr. Sara Royuela
Senior Researcher

www.ampere-euproject.eu

PPC in the scope of OpenMP

Specification, **compiler** and **runtime** support in OpenMP

targeting **performance**, **predictability** and **resilience** in multiple domains



Today

- The overhead of tasking
- The Task Dependency Graph
  - Performance
  - Memory consumption
  - Interoperability with CUDA graphs
- The RISING Stars and the AMPERE project

Tasking overheads

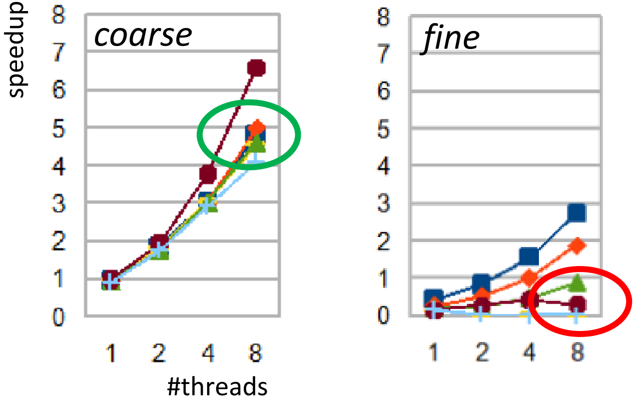
Podobas, A., Brorsson, M., and Faxén, K. F.
In *3rd workshop on programmability issues for multi-core computers*.
A comparison of some recent task-based parallel programming models. 2010.

OpenMP 3.1

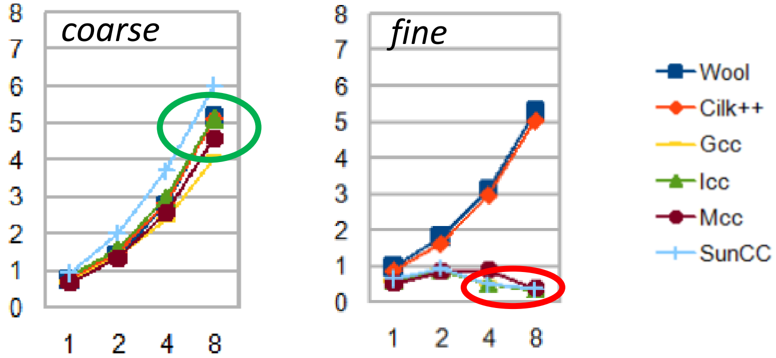
✓ Motivation for tasking: focus on the exposing parallelism rather than figuring out how to fit in a specific machine.

✗ Real limitations: fine granularities and deep cut-offs introduce too much overhead, reducing potential speed up.

Multisort



Strassen



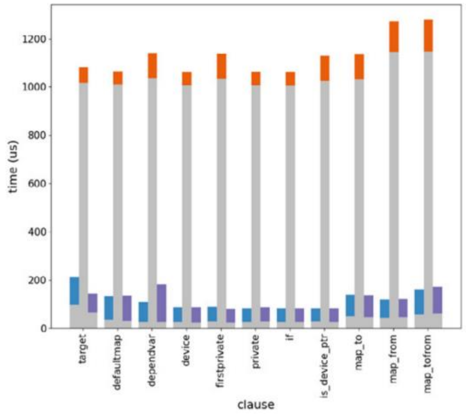
Tasking overheads

Diaz, J.M., Friedline, K., Pophale, S., Bernholdt, D.E., and Chandreasekaran, S. Parallel Computing. **Analysis of OpenMP 4.5 offloading in implementations: correctness and overhead**. 2019.

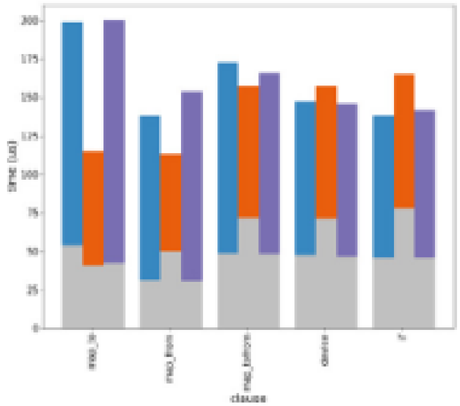
OpenMP 4.0

<u>Motivation for tasking</u>: implement offloading capabilities to exploit accelerator devices.

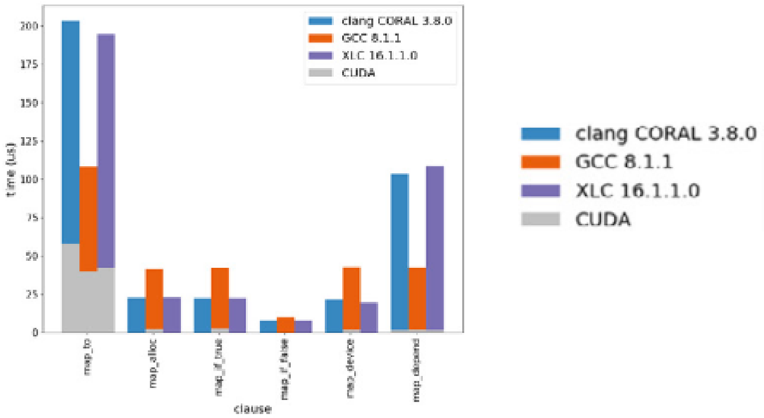
<u>Real limitations</u>: code transformation might not be optimized, e.g., memory allocations, synchronizations.



target



target data



target enter data

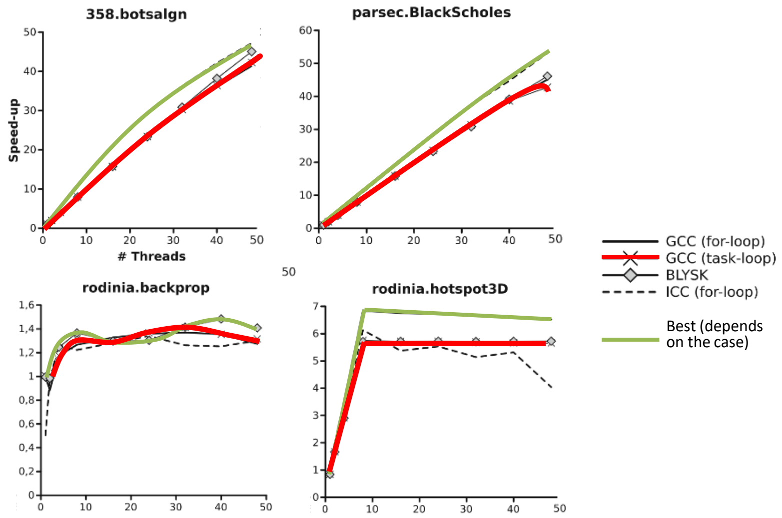
Tasking overheads

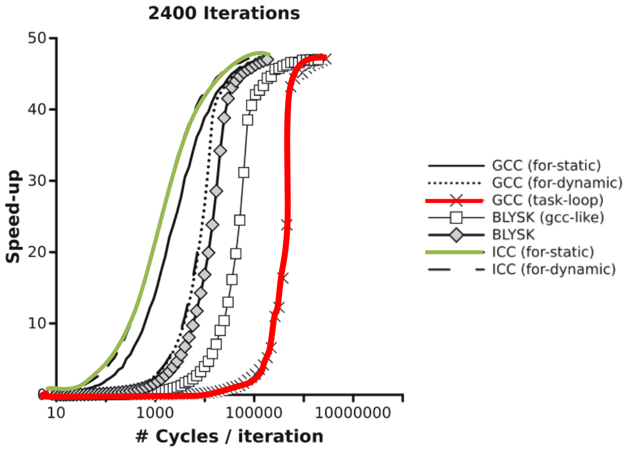
Podobas, A., and Karlsson, S.
In *International Workshop on OpenMP*.
Towards unifying OpenMP under the task-parallel paradigm. 2016.

OpenMP 4.5

Motivation for tasking: taskloop could eliminate the need for thread-parallelism.

Real limitations: the implementation is crucial and determining granularity becomes a challenge.





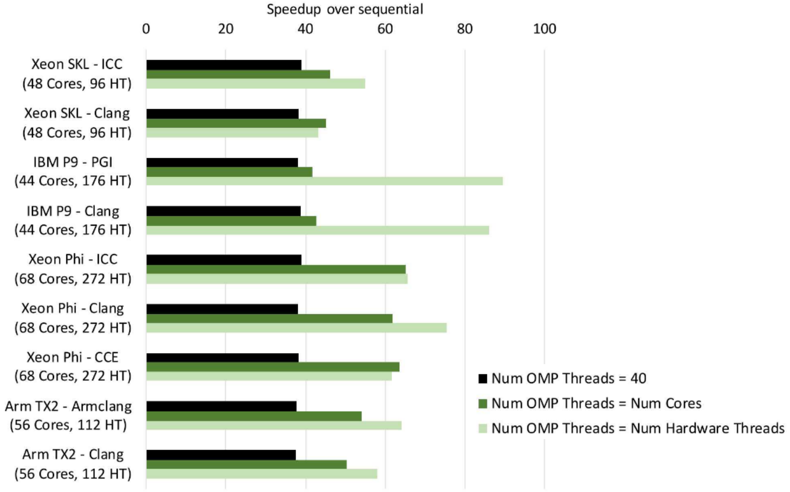
Tasking overheads

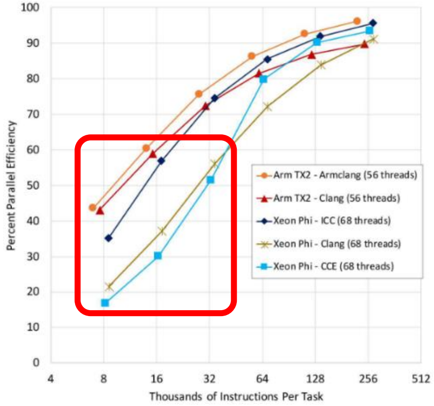
Olivier, S.L.. In *International Workshop on OpenMP*.
Evaluating the Efficiency of OpenMP Tasking for Unbalanced Computation on Diverse CPU Architectures. 2020.

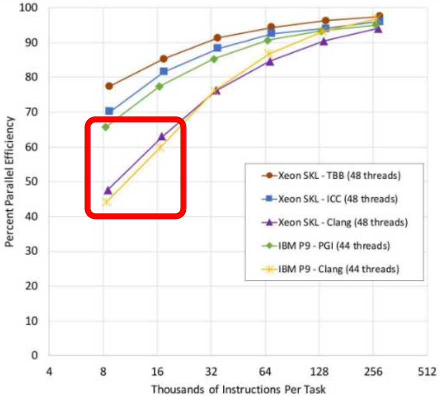
OpenMP 5.0

<u>Motivation for tasking</u>: many different extensions that have changed the internals of the implementations.

<u>Real limitations</u>: fine granularity provides poor efficiency 17% - 77%; acceptable granularity from tens of ms.

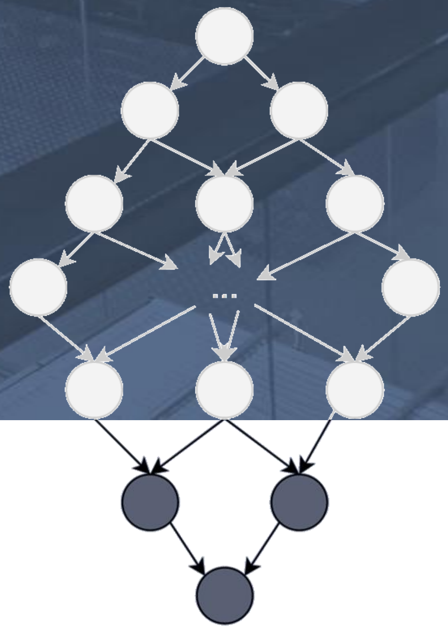






Where are we

- Tasking is convenient to expose parallelism
- Implementation overheads limit its use for:
    - Fine grained parallelism
    - Loop parallelism
    - Accelerator devices



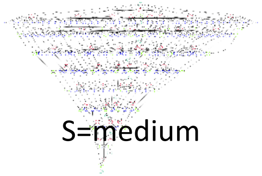
Let's capitalize on the Task Dependency Graph!

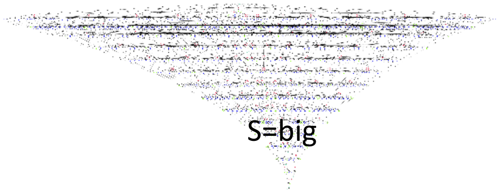
Leveraging the Task Dependency Graph

A region of code that can be fully represented as a TDG:

1. *Taskified* region:
   a) All computations are enclosed in tasks.
   b) Non-taskified code has no side effects on the tasks (e.g., induction variables in loops).

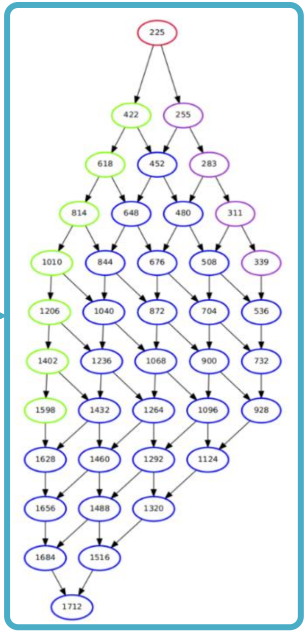
2. TDG shape:
   a) Shape does not change across TDG executions.
   b) Provide information for recomputing the TDG (e.g., a clause with the variables shaping the TDG).


S=medium


S=big

Pedestrian Detector

```
#pragma omp parallel
#pragma omp single
{
  for (i=0; i<N_ITER; ++i) {
    for (by=0; by < BY; by+=BS) {
      for (bx=0; bx < BX; bx+=BS) {
        if (bx==0 && by==0) {
          #pragma omp task depend(…)
          {…}
        } else if (by==0) {
          #pragma omp task depend(…)
          {…}
        } else if (bx==0) {
          #pragma omp task depend(…)
          {…}
        } else {
          #pragma omp task depend(…)
          {…}
        }
      }
    }
  }
}
```



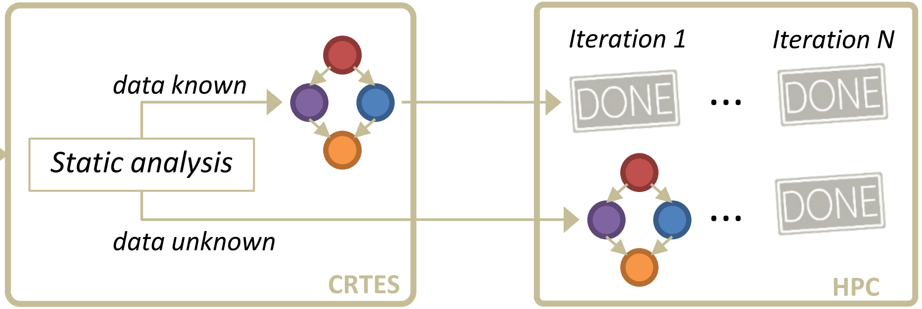
TDG-driven framework

- **Goals:** Reduce overhead due to task orchestration and dependency resolution
- **Methodology:** Eliminate the execution of user code to instantiate tasks

Source code

```
for (int it=0; it<IT; ++it)
    #pragma omp taskgraph
    for (int i=0; i<N; ++i){
        #pragma omp task
        {…}
        …
}
```

Compiler ***Runtime***



How to define it:

→ User-defined: `taskgraph` directive

→ Automatically detected by the compiler:

  → Via analysis

  → `taskloop`

When to generate it:

At compile-time (CRTES)

At run-time (HPC)

Chenle, Y, Royuela, S, and Quiñones, E. **Enhancing OpenMP Tasking Model: Performance and Portability,** In *International Workshop on OpenMP (IWOMP)*. 2021.

Compiler transformations

```
for (int it=0; it<IT; ++it)
   #pragma omp taskgraph
   for (int i=0; i<N; ++i){
      #pragma omp task
      {…}
      …
}
```

1. Meet conditions / taskgraph region

Yes

2. CFG
3. Loop unroll / Constant prop.
4. Dominator tree
5. Dependencies resolution

Done

TDG generated at compile-time

```
struct kmp_record_info
   kmp_tdg_1[N_inst] = {…}
```

```
for (int it=0; it<IT; ++it)
  execute_TDG();
}
```

Not done

TDG generated at run-time

```
struct kmp_record_info
   kmp_tdg_1[N_inst] = {…}
```

```
for (int it=0; it<IT; ++it)
  if (first_time)
    record_TDG();
  else
    execute_TDG();
}
```

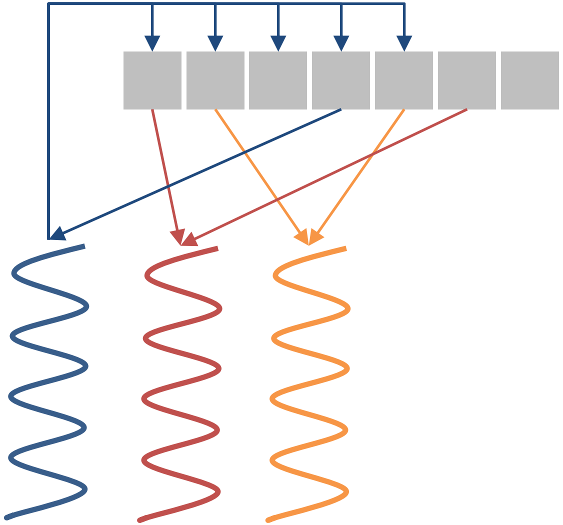
No

No taskgraph

```
for (int it=0; it<IT; ++it)
  instantiate_task();
  …
}
```

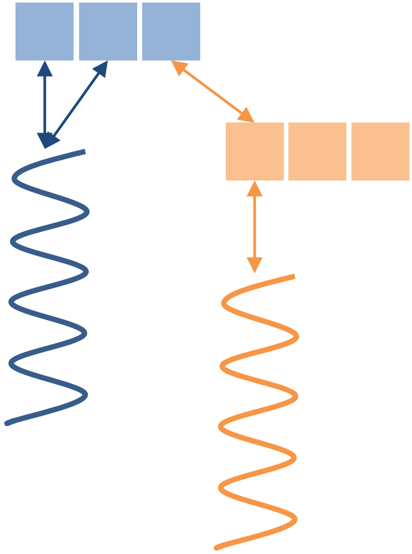
Runtime execution: LLVM vs. GCC

One thread pushes to a single queue, from which all threads pull.



Vanilla

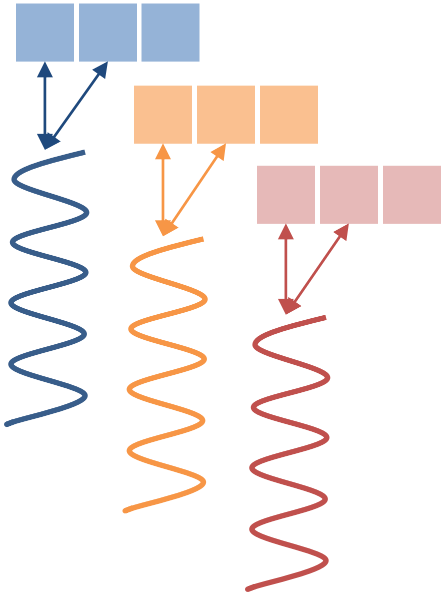
One thread pushes to its queue and the rest steal work from it.



TDG

Each thread pushes and pulls from its own queue.

Work-stealing is allowed.

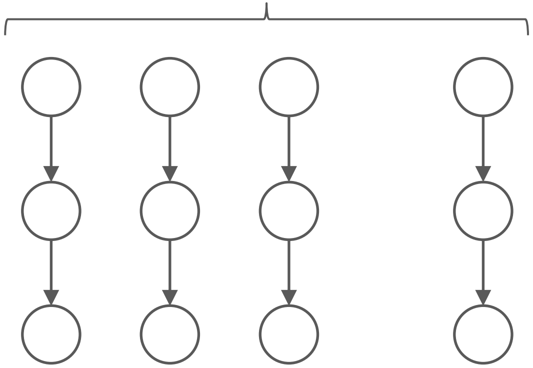


Can we reduce overhead?

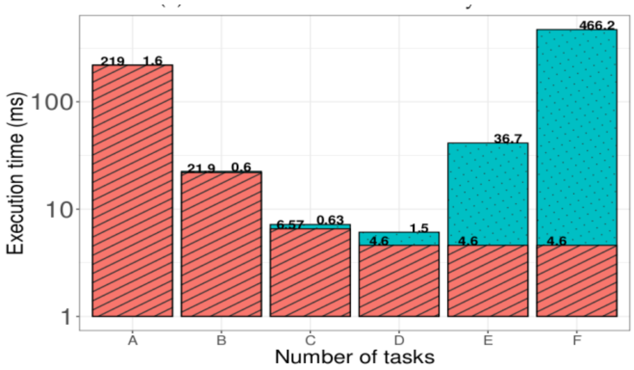
Synthetic

```
#pragma omp parallel
#pragma omp single
{
  for (int i=0; i<N_Tasks; ++i) {
    int index = I % N_Cores;
    #pragma omp task depend(out:deps[index])
    fn();
  }
}
```

N cores



Task orchestration overhead



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

$Overhead = Total\_time - Computation$

A = 1 task of 10^9 inst., B = 10 tasks of 10^8 inst., C = 10^2 tasks of 10^7 inst., D = 10^3 tasks of 10^6 inst., E = 10^4 tasks of 10^5 inst., F = 10^5 tasks of 10^4 inst.

- ✓ Reduce #instructions needed to orchestrate tasks
- ✓ Alleviate contention

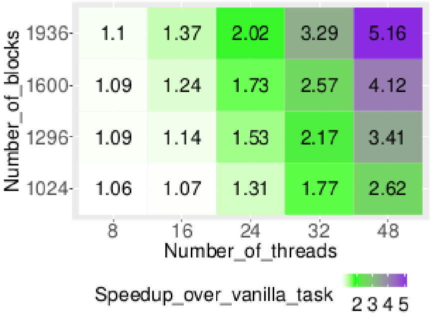
#tasks	10^0	10^1	10^2	10^3	10^4	10^5
Vanilla	1.6	0.6	0.6	1.5	36.7	466.2
Taskgraph	**0.2**	**0.1**	**0.2**	**0.3**	**9.9**	**132.2**

Scalability

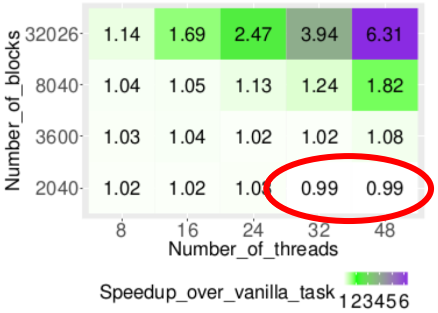
Speed-up of *TDG-driven* execution compared to vanilla *task* and *taskloop* implementations using different number of threads and different task granularities

Explicit tasks

Heat propagation simulation

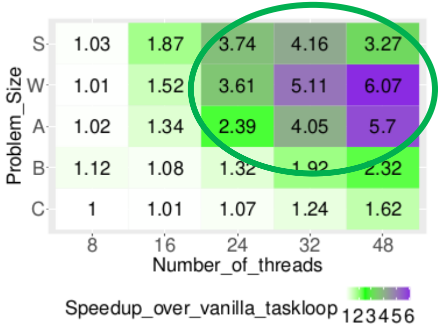


HoG Object detector

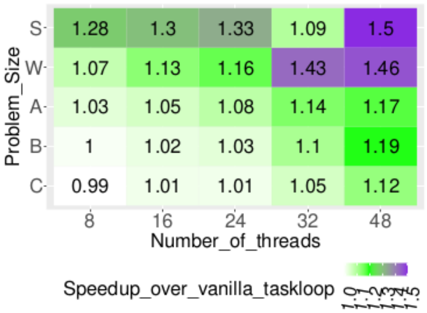


Taskloop

NAS_CG



NAS_LU



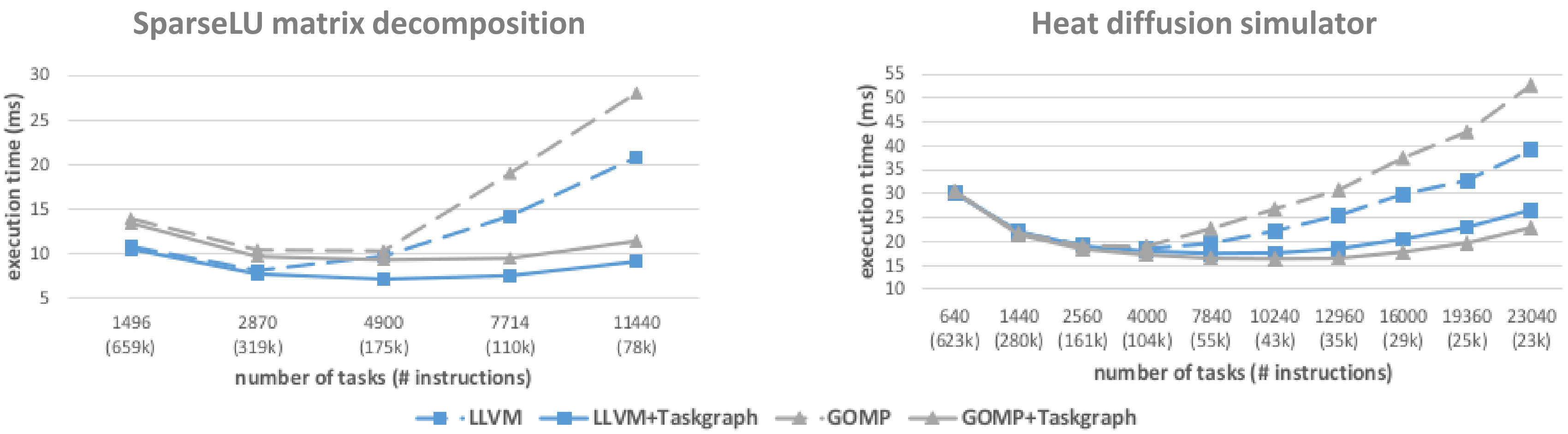
→ Negligible negative impact

→ Considerable positive impact for fine granularities and high thread contention

Chenle, Y, Royuela, S, and Quiñones, E. **Taskgraph: A low contention OpenMP tasking framework,** Submitted to *IEEE Transactions on Parallel and Distributed Systems*. 2022.

Portability

Speed-up of *TDG-driven* execution compared to vanilla GCC and LLVM implementations using different task granularities



→ Coarse grained tasks provide comparable results

→ Fine grained tasks show certain stability

→ Benefits are portable across compilers/RTLs

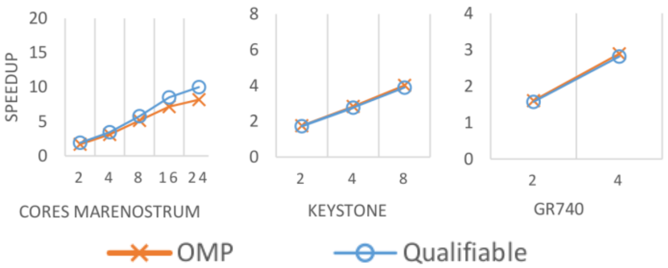
Memory management

- **Goals:**
  1. Avoid dynamic allocation of task structures
  2. Reduce and bound the memory requirements of the OpenMP RTL
- **Methodology:**
  1. Compiler: static generation of the required task structures
  2. Runtime: *lazy task creation* (task created when dependencies fulfilled)

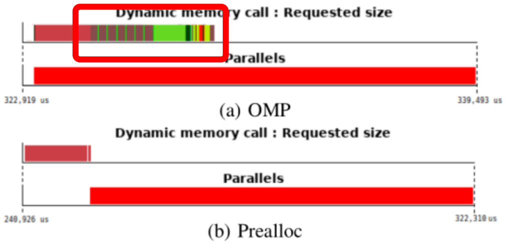
GCC

Space-Time Adaptive Processing

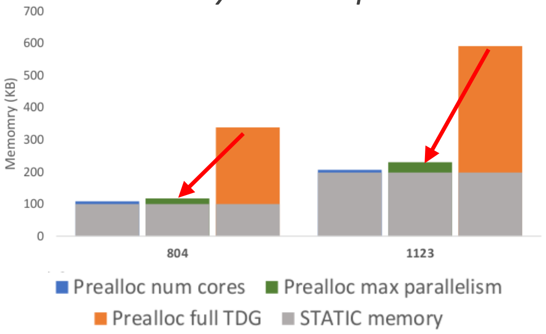
Performance speedup



Use of dynamic memory



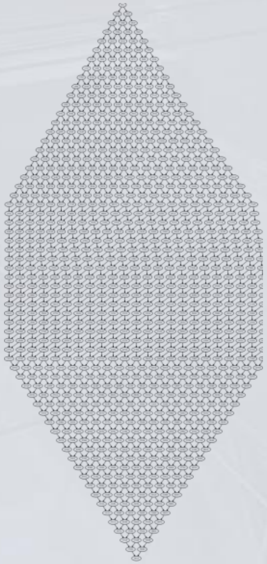
Memory consumption



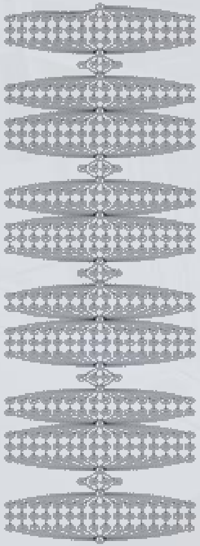
Munera, A, Royuela, S, and Quiñones, E. **Towards a qualifiable OpenMP framework for embedded systems,** In *Design, Automation & Test in Europe (DATE)*. 2020.

Applicability

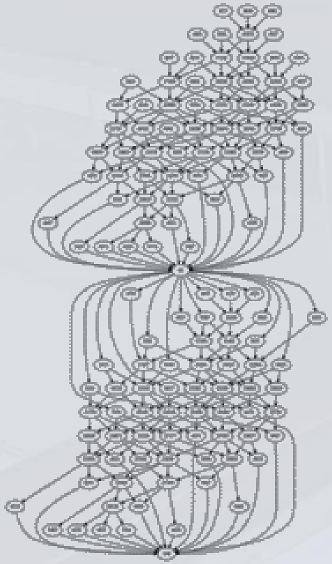
Pedestrian detector (automotive)



Infra-red sensor processing (space)



3D Path Planning (avionics)



Cholesky Factorization (HPC)

Many applications can be represented as a TDG!

The RISING stars project

- Enable a versatile and efficient **data acquisition** providing interoperability between different programming models (OpenMP, CUDA)
- Expose **data acquisition/transfer** in the **programming model**
- Introduce **real-time** oriented features in the **programming model** to define periodicity, preemption, migration, and allocation.
- Use cases: Adaptive optics, adaptive beamforming, the Square Kilometer Array and Space Situational Awareness.

Rising STARS

RISE International Network for Solutions Technologies and Applications of Real-time Systems

www.risingstars-project.eu

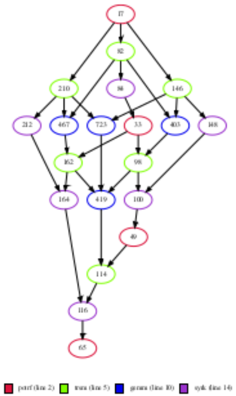
Interoperability

Rising STARS
RISE International Network for Solutions Technologies and Applications of Real-time Systems

OpenMP

```
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);
  }
}
```

TDG



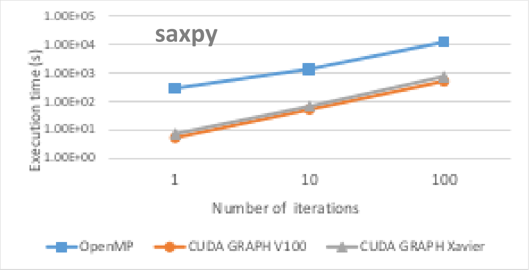
CUDA Graphs

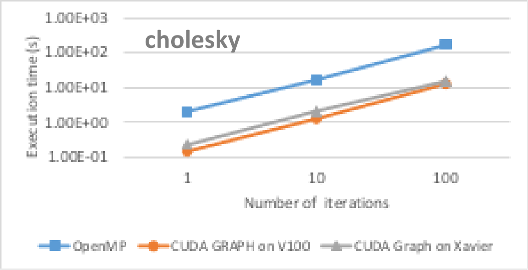
```
...

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);

...
```





OpenMP synchronizations take longer than CUDA graphs

Chenle, Y, Royuela, S, and Quiñones, E. **OpenMP to CUDA graphs: a compiler-based transformation to enhance the programmability of NVIDIA devices,** In *international Workshop on Software and Compilers for Embedded Systems (SCOPES)*. 2020.

CUDA memory management strategies

Rising STARS

RISE International Network for Solutions Technologies and Applications of Real-time Systems

Faster when there is no need for unified memory

Execution time in ms.

App	Graph nodes	Unified memory + prefetch	Unified memory (non-prefectch)	Zero copy	cudaMemcpy	cudaMemcpyAsync
Vector addition	**1024**	739	865	687	448	435
	32	592	738	595	344	343
Saxpy	**1024**	658	916	538	441	441
	32	459	572	452	286	255
Nbody	**1024**	6080	6058	6464	6041	6094
	32	762	775	806	764	765

Faster execution when reducing the number of nodes

Pre-fetching reduces page faults

The AMPERE project

- Use of **Domain Specific Modeling Languages** and **high-level synthesis methods** for building *correct-by-construction* systems.
- Use OpenMP to provide the **performance** needed to develop complex Cyber-Physical Systems:
  - Predictive Cruise Control (automotive)
  - Obstacle Detection and Avoidance System (railway)
- Provide mechanisms to guarantee **non-functional requirements**: time predictability, resilience and energy consumption.

AMPERE

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

www.ampere-euproject.eu

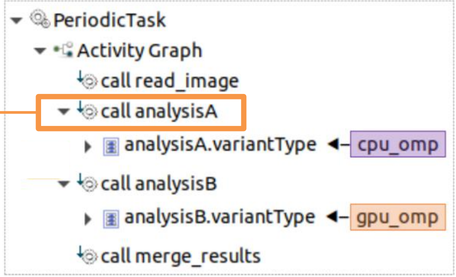
AMALTHEA DSML to OpenMP

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

APP4MC — *Automatic code generation* → OpenMP

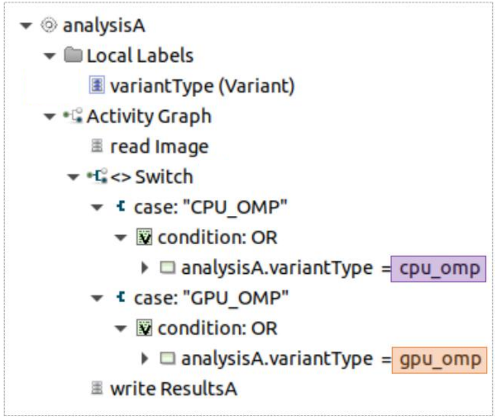
Goals:

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



OS task

```
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(); }
  }
}
```



OpenMP task

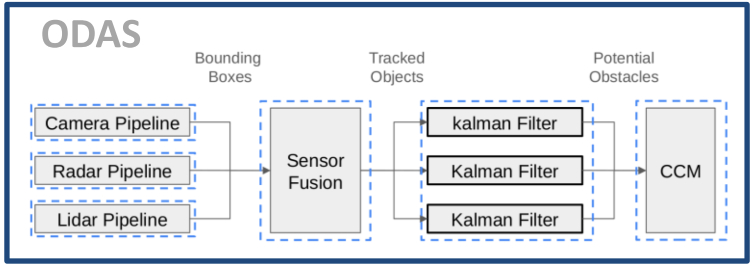
```
void analysisA_gpu() { … }


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

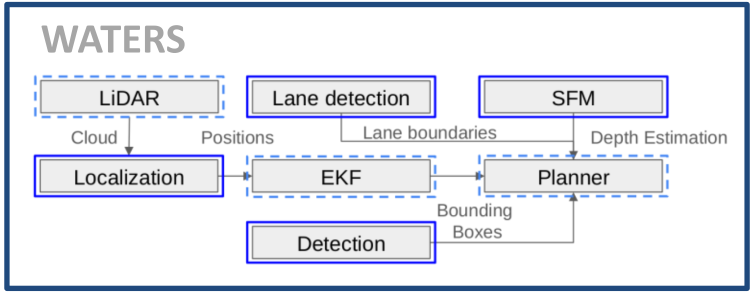
https://gitlab.bsc.es/ampere-sw/wp2

AMALTHEA DSML to OpenMP: Performance

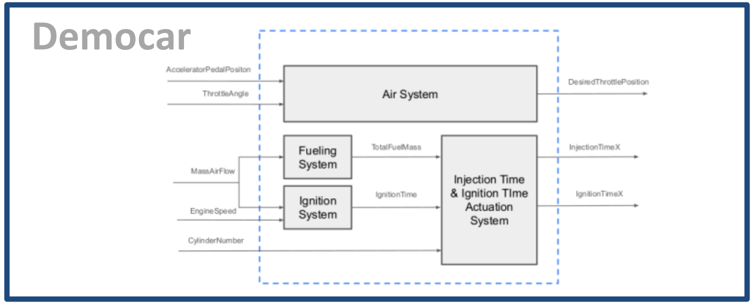
ODAS



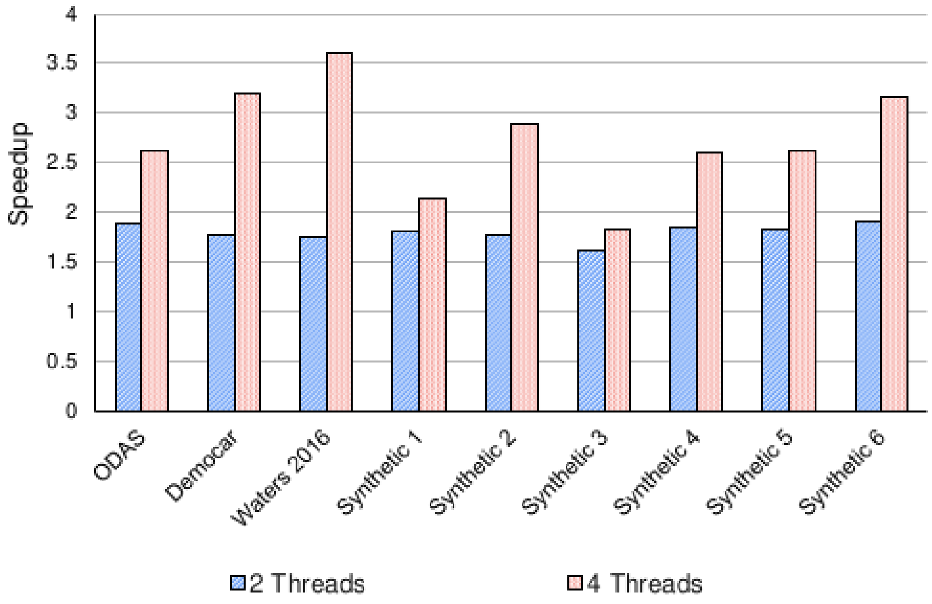
WATERS



Democar



NVIDIA Jetson TX2 board
with a GPU and a 4-core ARM CPU



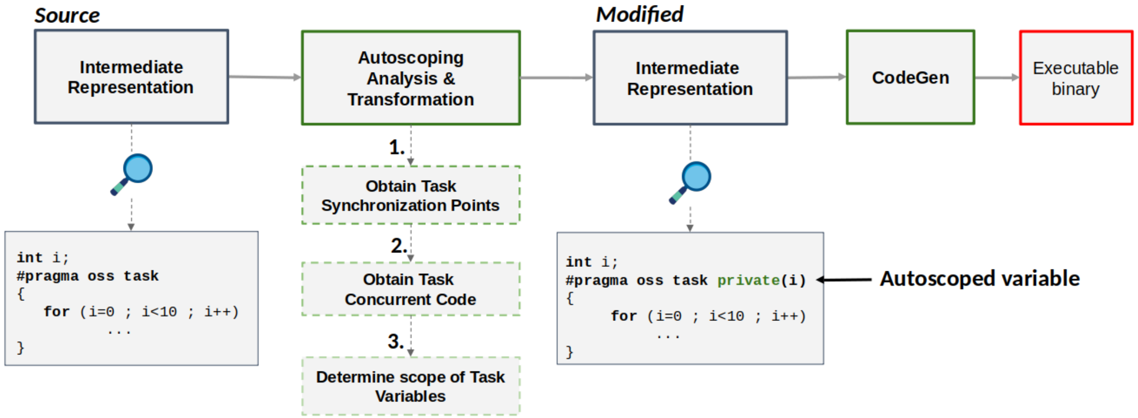
Correctness analysis for OpenMP

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

Goals:

1. Detect/resolve race conditions
2. Detect/correct wrong synchronizations (task dependencies, memory fences)
3. Detect inconsistencies in the data-sharing attributes

LLVM Compiler pipeline to define/correct data-sharing attributes:



A. Munera, S Royuela, R Ferrer, R Peñacoba, E Quiñones. **Static analysis to enhance the programmability and performance in OmpSs-2,** In *ISC*. 2020.
S. Royuela, R Ferrer, D Caballero, X Martorell. **Compiler analysis for OpenMP tasks correctness,** In *CF*. 2015.
S. Royuela, A Duran, X Martorell. **Compiler automatic discovery of OmpSs task dependencies,** In *LCPC*. 2012.
S. Royuela, A Duran, C Liao, DJ Quinlan. **Auto-scoping for OpenMP tasks,** In *IWOMP*. 2012.

Static analysis for OpenMP time predictability

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

GOAL: Assess predictability of OpenMP to allow schedulability analysis

Requirements:

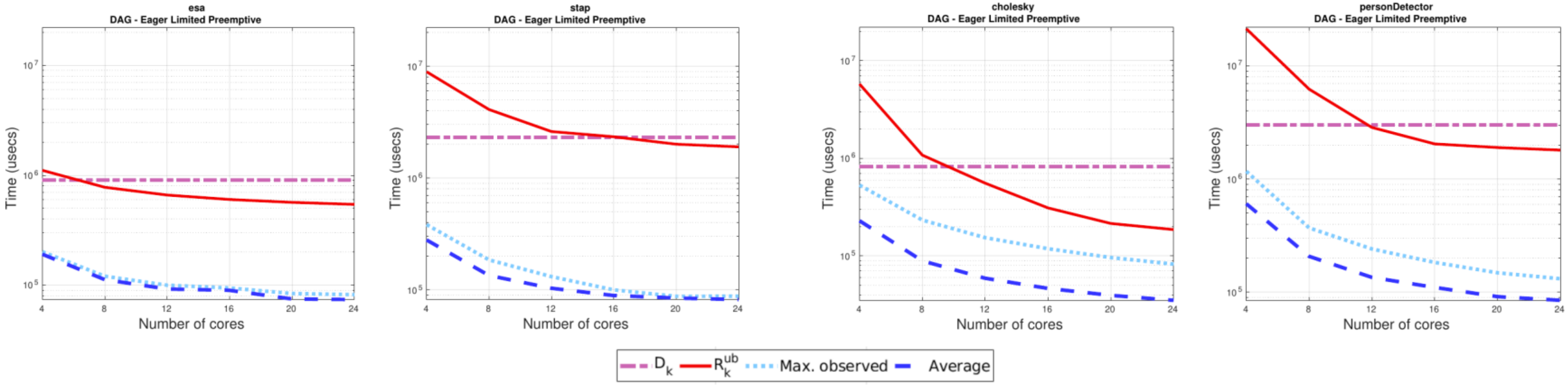
1. *Work-conserving* scheduler for non-pessimistic WCRT analysis
2. Prescriptive *task priorities* to support fixed priority schedulers
3. Prescriptive implementation of *Task Scheduling Points* to allow limited preemptive scheduling
   - *Taskyield*, to alleviate pessimism and enhance schedulability

Response time upper bound (R_k^{ub}):

$$R_k^{ub} \leftarrow len(G_k) + \frac{1}{m}(vol(G_k) - len(G_k)) + \frac{1}{m}(I_k^{hp} + I_k^{lp})$$

R_k^{ub} bounds the maximum observed execution time

Real-time system schedulable with at least 16 cores



M. A. Serrano, et al. **Response-time analysis of DAG tasks under fixed priority scheduling with limited preemptions,** In *DATE*. 2016.
M. A. Serrano, et al. **An analysis of lazy and eager limited preemption approaches under DAG-based global fixed priority scheduling,** In *ISORC*. 2017.

Resilience with OpenMP

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

GOAL: Task-level replication for fault-detection

Augmented OpenMP code

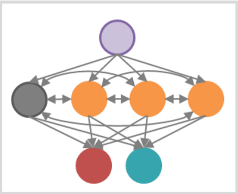
```
#pragma omp task redundancy(
    spatial|temporal|spatial_temporal,
    3, /*n replicas*/
    var1:func1,
    var2:func2)
{/*code*/}
```

Data capturing

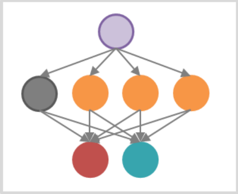
Replicated tasks

Consolidation function / variable to check

TDG with replication



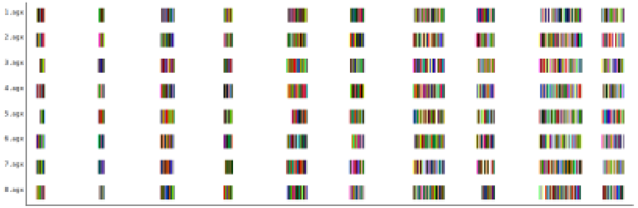
Temporal:
OpenMP mutexinoutset between replicas

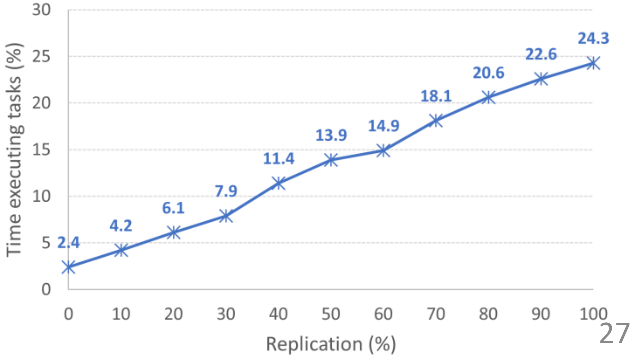


Spatial:
Each replica in a different core

Engine control management system

- TDG1 (T10, T20, T50)
- TDG2 (T10, T50)
- TDG3 (T10, T20)
- TDG4 (T10)





O. Tahan, et al. **Using dynamic task level redundancy for OpenMP fault tolerance,** In *ISC*. 2020.

PPC publications

Optimizations

1. *C. Yu, S. Royuela, E. Quiñones*, **Enhancing OpenMP tasking model: Performance and portability**, In IWOMP 2021.
2. *A. Munera, S. Royuela, E. Quiñones*, **Towards a qualifiable OpenMP framework for embedded systems**, In DATE 2020.
3. *R. E. Vargas, S. Royuela, M. A. Serrano, X. Martorell, E. Quiñones,* **A Lightweight OpenMP4 Run-time for Embedded Systems**, In ASP-DAC 2016.

Interoperability

4. *C. Yu, S. Royuela, E. Quiñones,* **OpenMP to CUDA graphs: A compiler-based transformation to enhance the programmability of NVIDIA devices**, In SCOPES 2020.
5. *S Royuela, L.M. Pinho, E. Quiñones*, **Enabling Ada and OpenMP runtimes interoperability through template-based execution**, In JSA 2020.
6. *S. Royuela, L.M. Pinho, E. Quinones,* **Converging Safety and High-performance Domains: Integrating OpenMP into Ada,** In DATE 2018.

Functional safety

7. *S. Royuela, A. Duran, M. A. Serrano, E. Quiñones,* **A functional safety OpenMP for critical real-time embedded systems,** In IWOMP 2017.
8. *S. Royuela, X. Martorell, E. Quinones, L. M. Pinho,* **OpenMP Tasking Model for Ada: Safety and Correctness,** In AEiC 2017.
9. *S. Royuela, R. Ferrer, D. Caballer, X. Martorell*, **Compiler analysis for OpenMP tasks correctness**, In CF 2015.

Predictability and CRTES

10. *M. A. Serrano, S. Royuela, E. Quiñones.* **Towards an OpenMP Specification for Critical Real-time Systems.** In IWOMP 2018.
11. *M. A. Serrano, A. Melani, S. Kehr, M. Bertogna, E. Quiñones,* **An Analysis of Lazy and Eager Limited Preemption Approaches under DAG-based Global Fixed Priority Scheduling,** In ISORC 2017.
12. *M. A. Serrano, A. Melani, M. Bertogna, E. Quiñones,* **Response-Time Analysis of DAG Tasks under Fixed Priority Scheduling with Limited Preemptions,** In DATE 2016.
13. *M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, E. Quiñones,* **Timing Characterization of OpenMP4 Tasking Model,** In CASES 2015.

Projects and collaborations

AMPERE
www.ampere-euproject.eu

Rising STARS
www.risingstars-project.eu

P-SOCRATES

esa

AIRBUS DEFENCE & SPACE

DENSO

BOSCH

THALES

High-performance OpenMP tasking

Sara Royuela
sara.royuela@bsc.es

OpenMP User's Monthly Telecon

October 28, 2022