Parallel Programming Models for Space Systems

(ESA Contract No. 4000114391/15/NL/Cbi/GM)

Eduardo Quiñones (BSC), Paolo Gai (Evidence)

TEC-ED & TEC-SW Final Presentation Days
(Noordwijk, the Netherlands)
9th June 2016

Parallel Programming Models for Space Systems

- Proof of concept Innovation Triangle Initiative (ITI) project (TRL 3)
  - Demonstrate the potential benefits of using the OpenMP tasking model into the space domain in terms of **programmability, performance and time predictability**
- Participants
  - Barcelona Supercomputing Center (BSC) (Spain)
    - Eduardo Quiñones, eduardo.quinones@bsc.es
  - Evidence (Italy)
    - Paolo Gai, pj@evidence.eu.com

Agenda

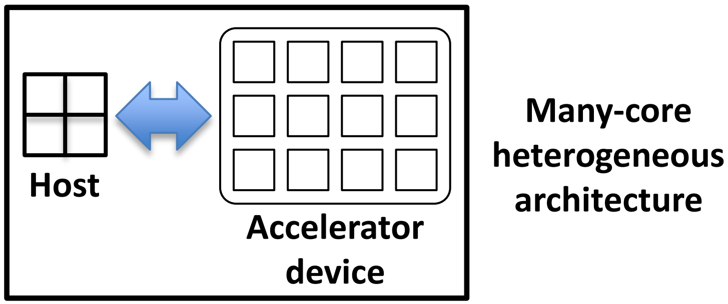
1. Introduction: Parallel programming models and OpenMP
2. Programmability and performance benefits
   – Parallelisation experiences with OpenMP: The tasking and acceleration execution models
   – Extensions required on non-POSIX Operating Systems
3. Time Predictability Benefits
   – OpenMP and Real-time embedded domain: A Possible Union?

Introduction

Parallel programming models and OpenMP

Parallel programming models

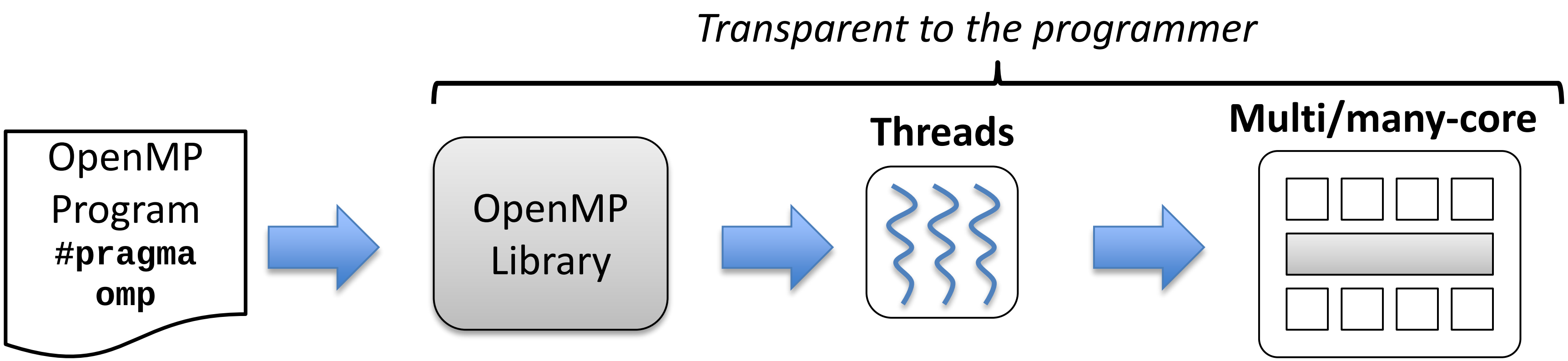
- An API to express application's parallelism (e.g. OpenMP, OpenCL)
  1. Parallel regions and synchronization mechanisms to guarantee the correct execution
  2. Couples host processor with acceleration devices in heterogeneous architectures



Why should you use them?

- High abstraction level for parallel programming hiding processor complexities
  - Mandatory to exploit the parallel computation capabilities of many-core architectures
- Become a vital element to provide the desired level of performance and programmability
- **This project focuses on OpenMP** (openmp.org)

Why OpenMP?



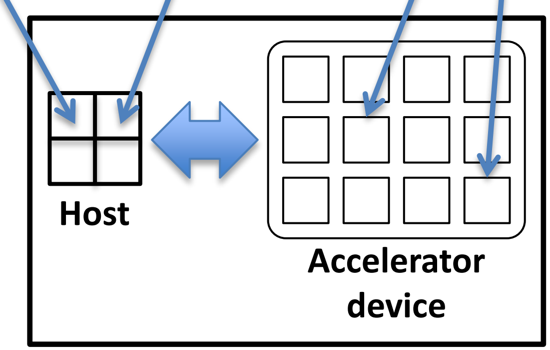
1. Supported by a large set of parallel architectures
   – Portable among parallel platforms
2. **Tasking model** expresses fine-grained and irregular
   – Code within a `task` executed by a *team of threads*
     • `depend` describe dependencies among tasks (`in, out, inout`)
3. **Accelerator model** offloads code and data to devices
   – The code within a `target` is executed within a device
     • Supports the `depend` clause integrating the acceleration and tasking model

OpenMP4.5 (released Nov 2015)

```
#pragma omp parallel num threads(2) {
#pragma omp task {// task T0
  p00 (x=0; y=0;)
  #pragma omp task depend(out:x,y) { p1 }// task T1
  p01
  #pragma omp target depend(in:x) map(to:x) { f1(); }// target task T2
  p02
  #pragma omp target depend(in:y) map(to:y) { f2(); }// target task T3
  p03
}}
```

1. A new team of 2 threads is created

2. Tasks are executed if available threads in the team



Many-core heterogeneous architecture

3. `f1()` and `f2()` execute on the accelerator device when...

4. ... T1 is completed

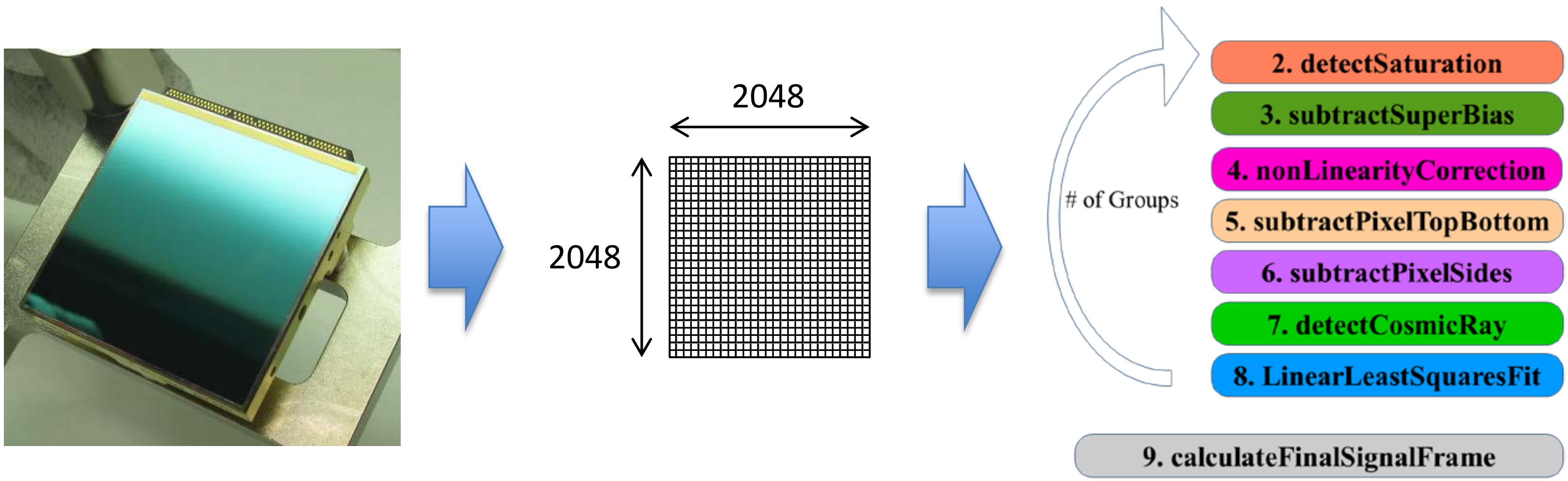
Performance and programability benefits

Parallelisation experiences of a space application with OpenMP:

The tasking and accelerator execution model

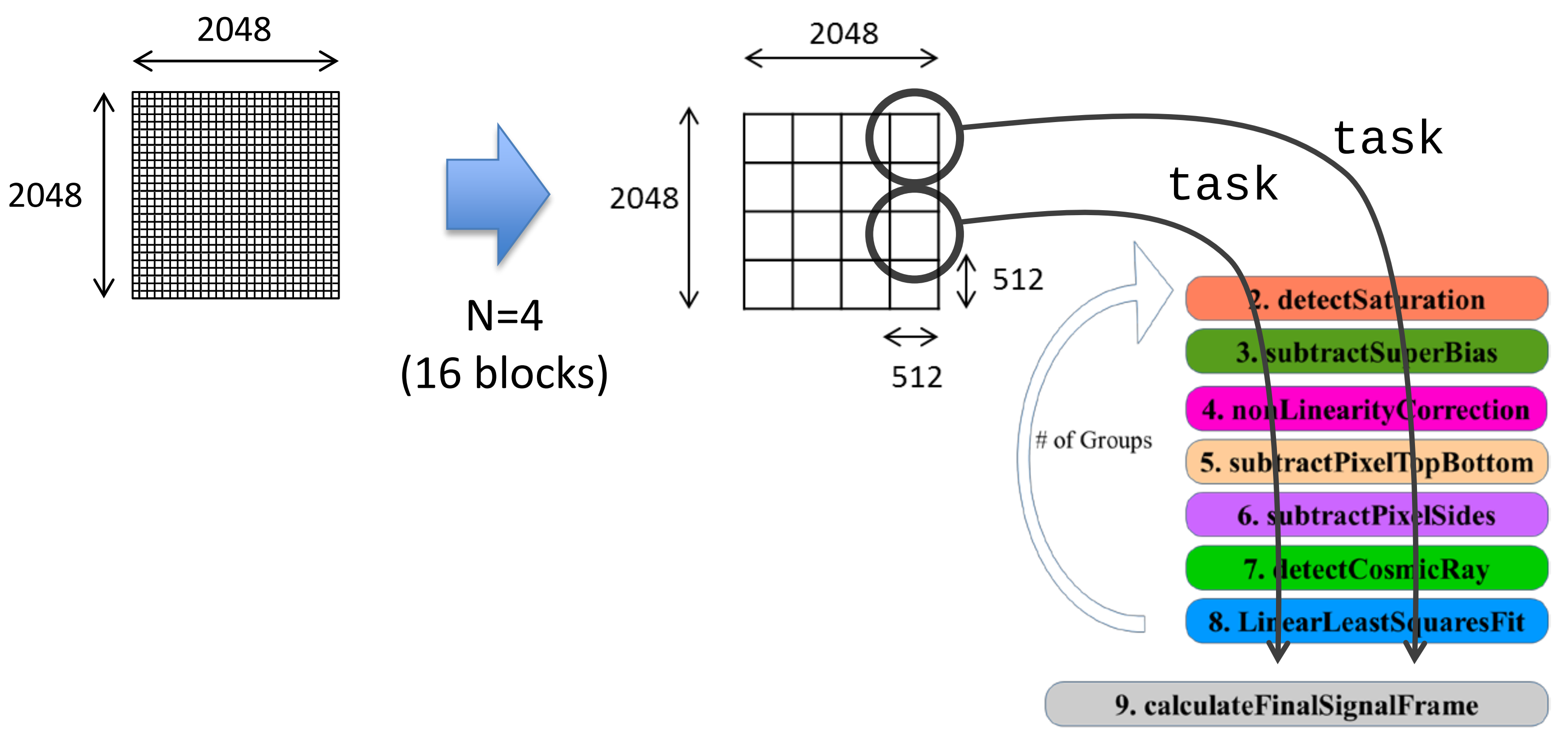
Application Case-study

- Pre-processing sampling for infra-red H2RG detectors
  - It processes sensor frames of 2048 x 2048 pixels through eight stages



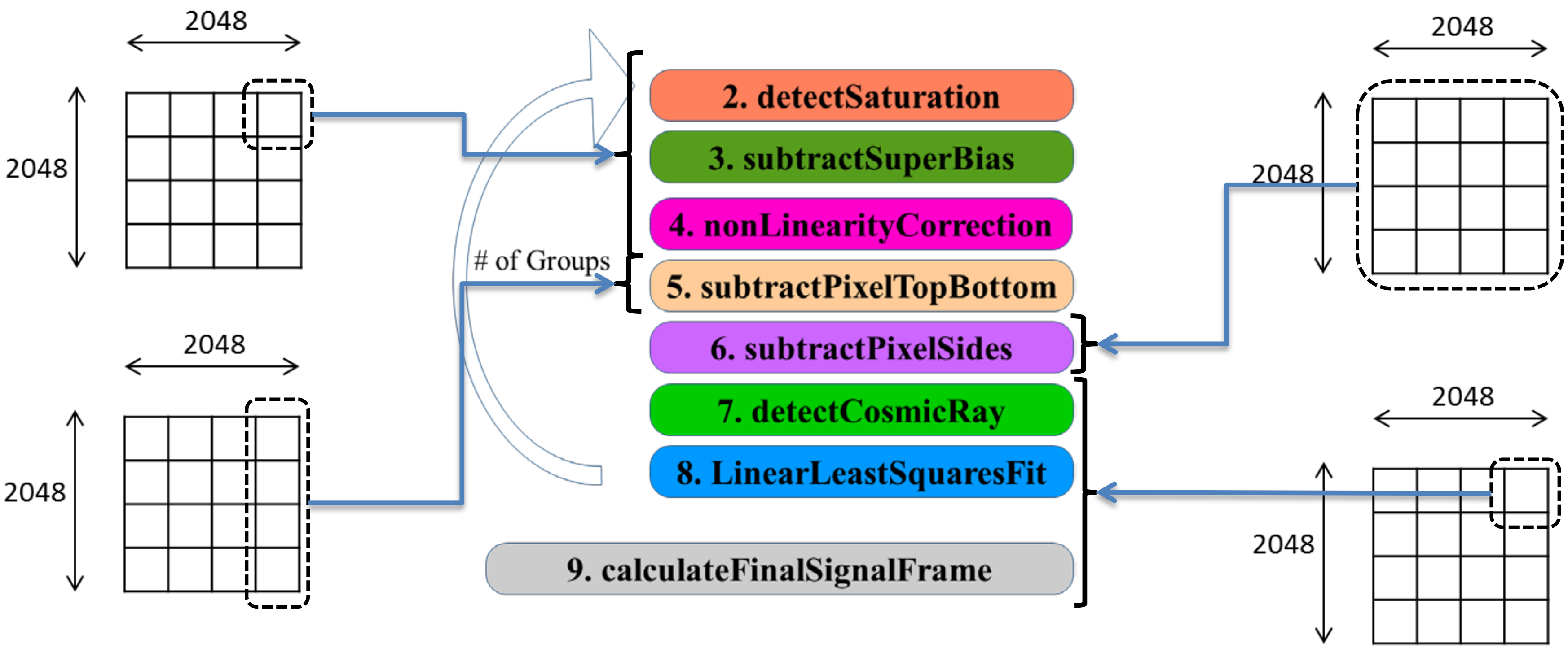
Parallelisation Strategy 1

- Divide the frame image in N x N blocks and process each in parallel (`task` directive)



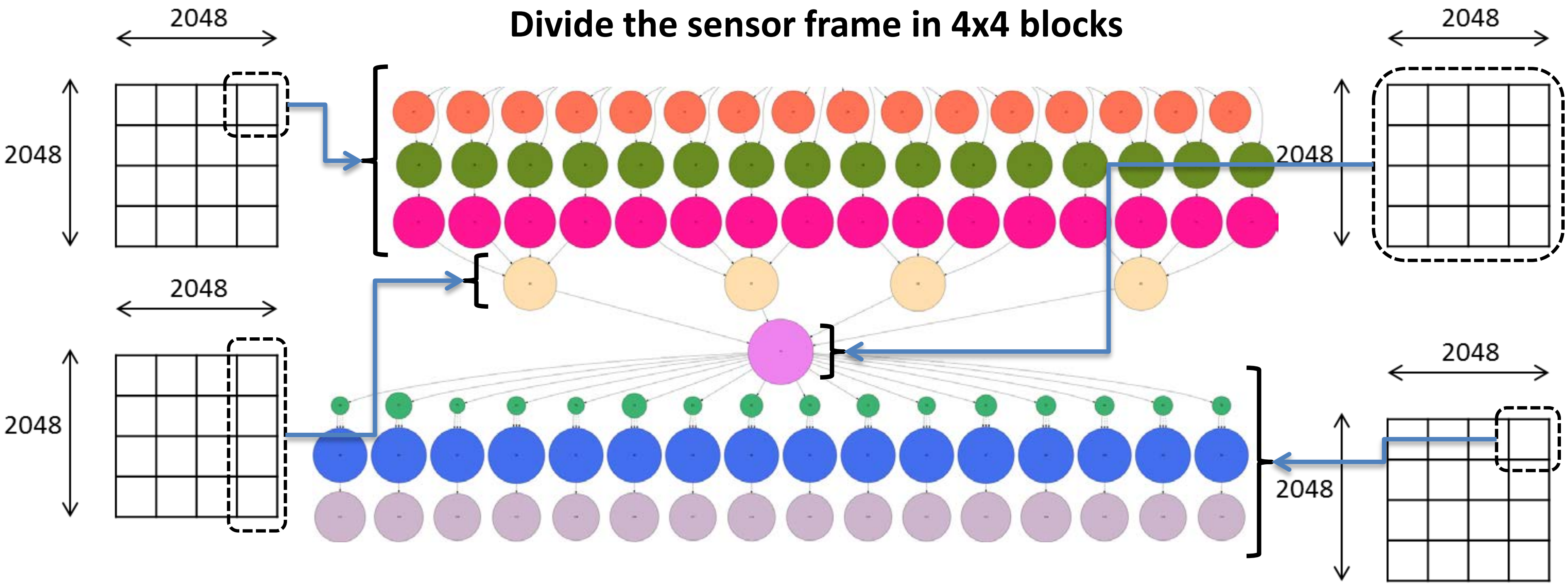
Parallelisation Strategy 1

- Data dependencies among application stages limits the level of parallelism



Application's Task Dependency Graph (TDG)

Divide the sensor frame in 4x4 blocks



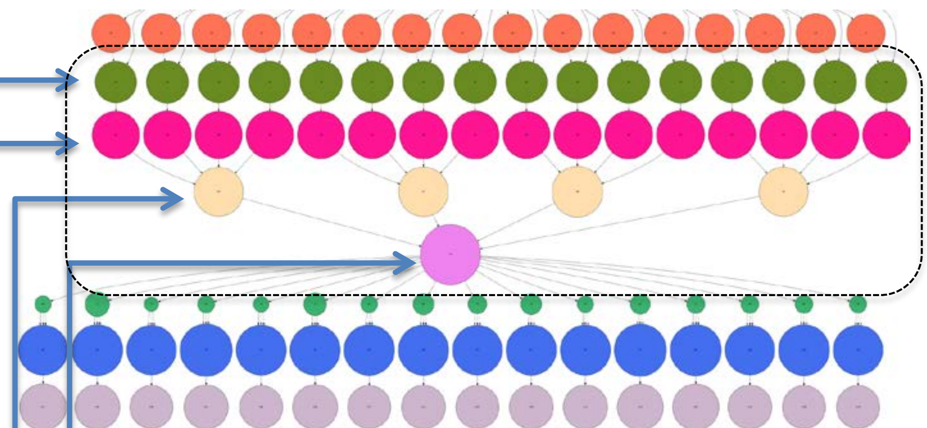
OpenMP Tasking Model

Original version

```
...
subtractSuperBias();
nonLinearityCorrectionPolynomial();
subtractReferencePixelTopBottom();
subtractReferencePixelSides();
...
```

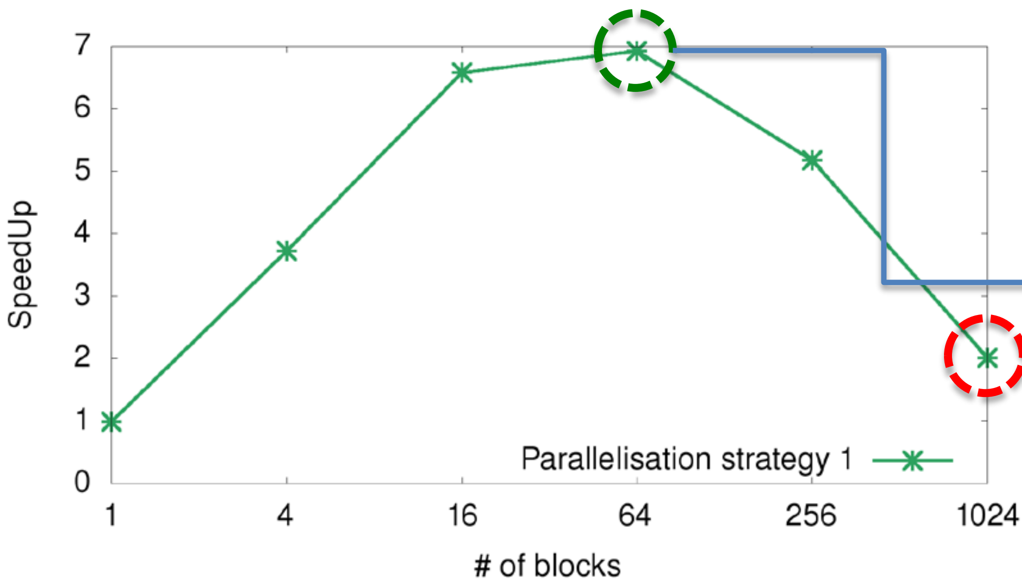
Parallel version

```
...
for (i=0; i < 4; i++)
  for (j=0; j < 4; j++)
#pragma omp task depend(inout:frame[i][j])
      subtractSuperBias(i,j);
for (i=0; i < 4; i++)
  for (j=0; j < 4; j++)
#pragma omp task depend(inout:frame[i][j])
    nonLinearityCorrectionPolynomial(i,j);
for (j=0; j < 4; j++)
#pragma omp task depend(in:frame[0][j] \\
                        in:frame[1][j] \\
                        in:frame[2][j] \\
                        in:frame[3][j])
  subtractReferencePixelTopBottom(j);
#pragma omp taskwait
subtractReferencePixelSides();
...
```



Application's Performance Speed-up

- Experiments on two Intel(R) Xeon(R) CPU E5-2670 processor, featuring 8 cores each and 20 MB L3
- OpenMP implemention from GNU-GCC (libgomp)



Number of blocks	Created tasks	Block size
1	31	2048x2048
4	114	1024x1024
16	436	512x512
64	1704	256x256
256	6736	128x128
1024	26784	64x64

Twofold reason:

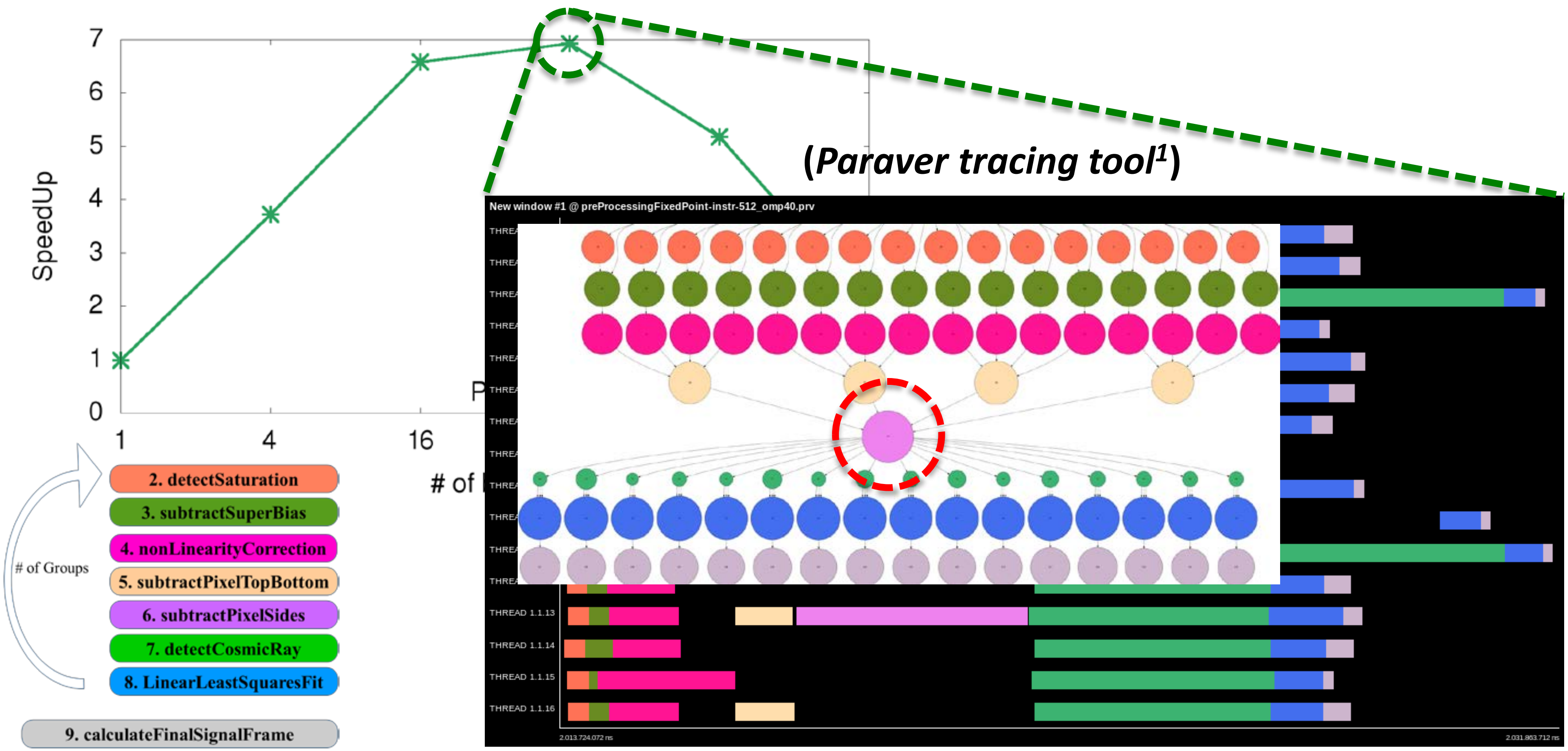
1. The run-time overhead due to a high number of tasks
2. The small data set upon which tasks operate

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EVIDENCE
EMBEDDING TECHNOLOGY

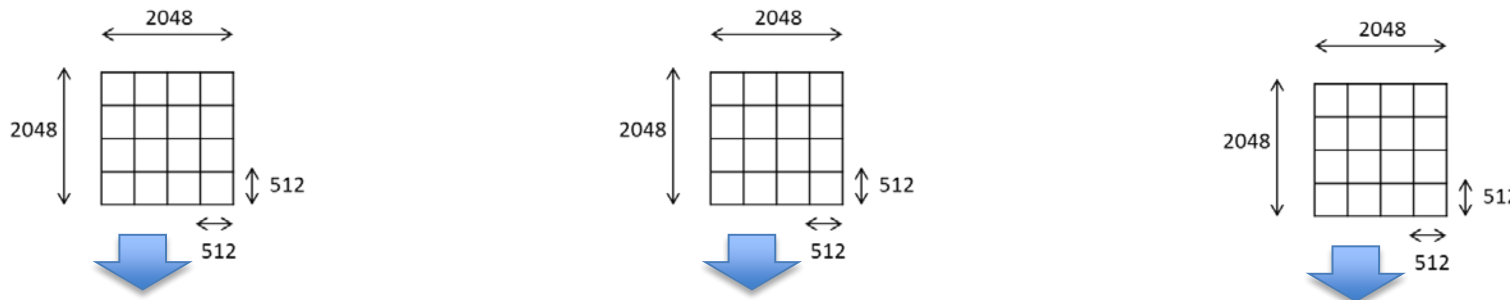
Performance Speed-up Analysis

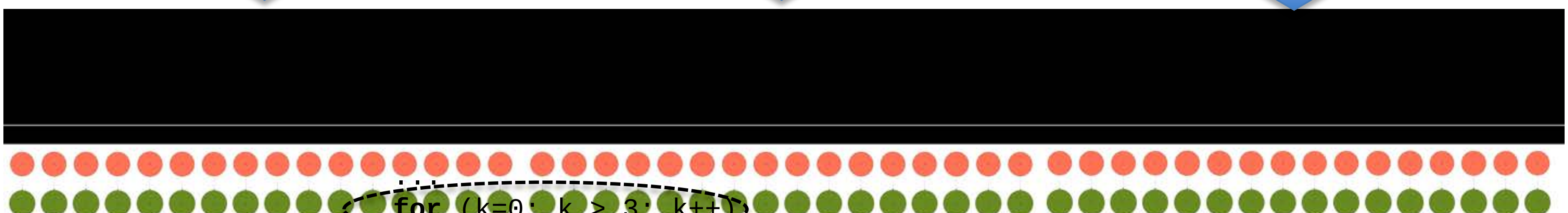
- Why only 7x in a 16-core architecture?



[1] http://www.bsc.es/computer-sciences/performance-tools/paraver

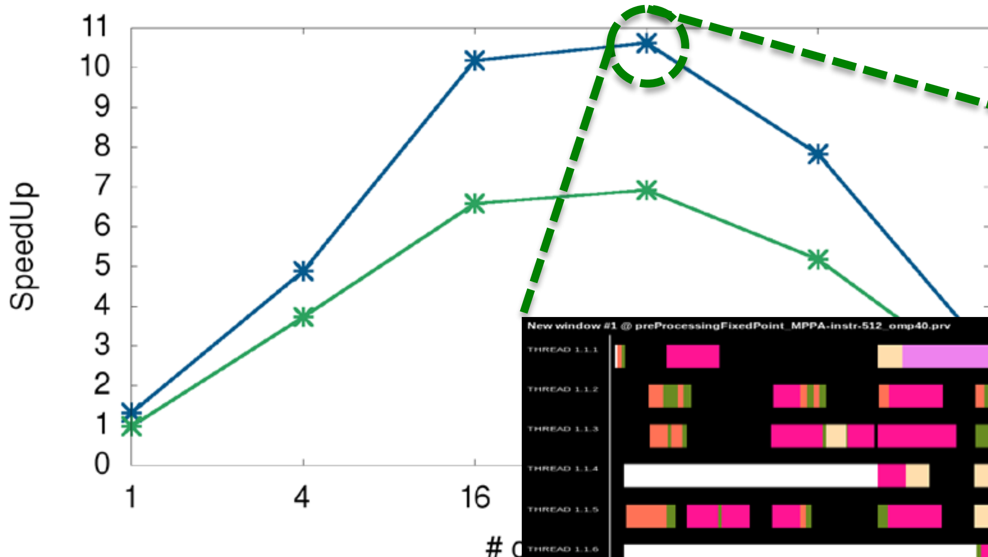
Tasking Model: Parallelisation Strategy 2





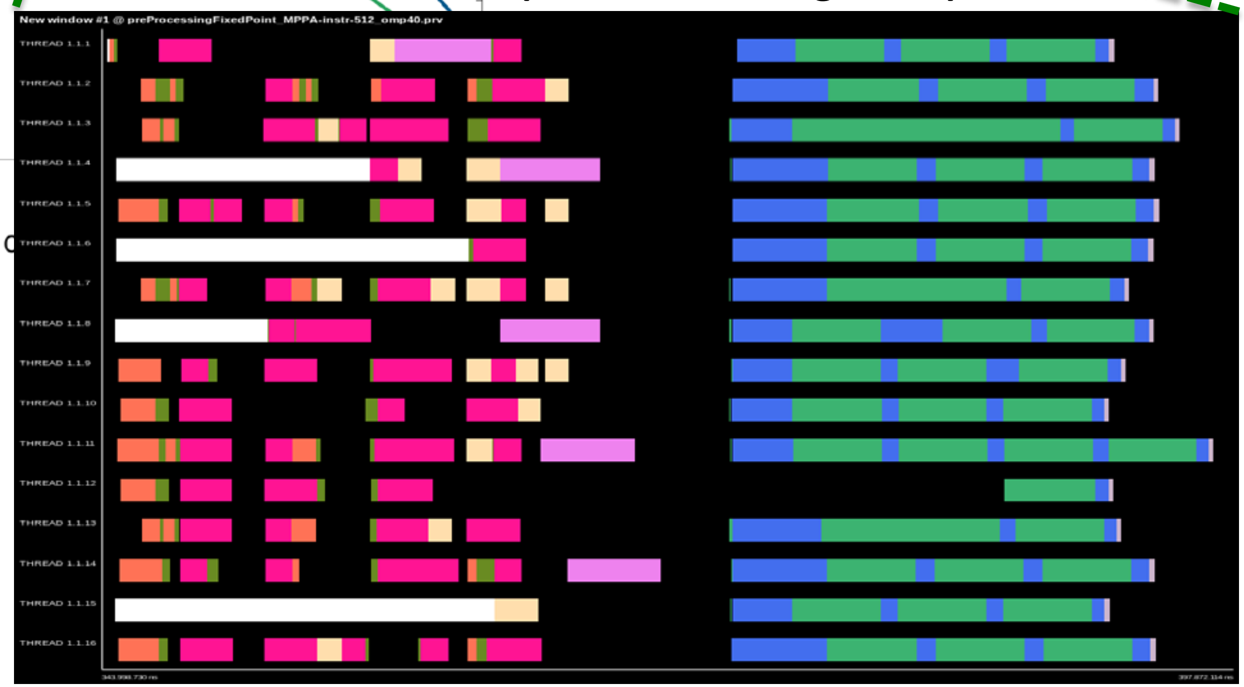
```
...
for (k=0; k > 3; k++)
  for (i=0; i < 4; i++)
    for (j=0; j < 4; j++)
#pragma omp task depend(inout:frame[k][i][j])
    nonLinearityCorrectionPolynomial(k,i,j);
for (k=0; k > 3; k++)
  for (j=0; j < 4; j++)
#pragma omp task depend(in:frame[k][0][j] in:frame[k][1][j] \\
                        in:frame[k][2][j] in:frame[k][3][j] \\
                        out:frame[k])
    subtractReferencePixelTopBottom(k,j);
for (k=0; k > 3; k++)
#pragma omp task depend(inout:frame[k])
  subtractReferencePixelSides(k);
...
```

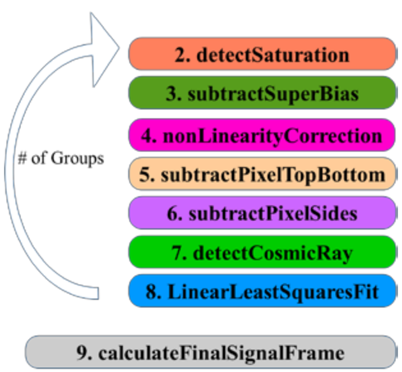
Application's Performance Speed-up



5 image frames processed simultaneously

(***Paraver tracing tool***[1])





[1] http://www.bsc.es/computer-sciences/performance-tools/paraver

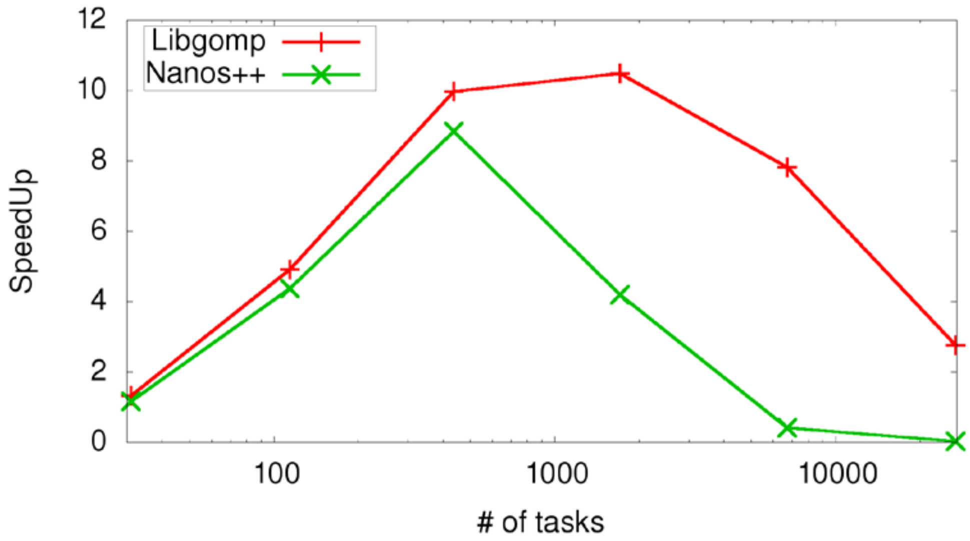
Tasking Model: 4-core Leon3

- Processor implemented on a FPGA running at 80 Mhz featuring a Leon3
- RTEMS v4.12 + *enable-smp* flag (not easy to discover that required) + GCC 6.0.0
- RTEMS APIs and OpenMP directives cannot be mixed
  - RTEMS part separated from the OpenMP part in two source files
  - Replaced the standard `main` entry point by the *RTEMS Init task*

	Sequential Time (seconds)	Parallel Time (seconds)	Speed-up
Parallelisation strategy 1	284.863	76.543	**3.7x**
Parallelisation strategy 2	284.863	74.357	**3.8x**

Tasking Model: Run-time Overhead

- The run-time overhead may completely dominate when the workload computed by tasks is small
  - Data dependencies are managed through a complex data structure (hash table)

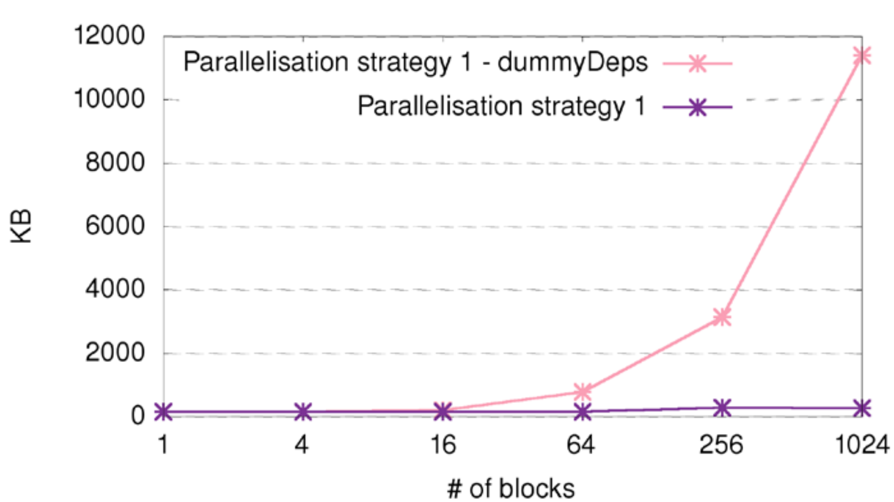


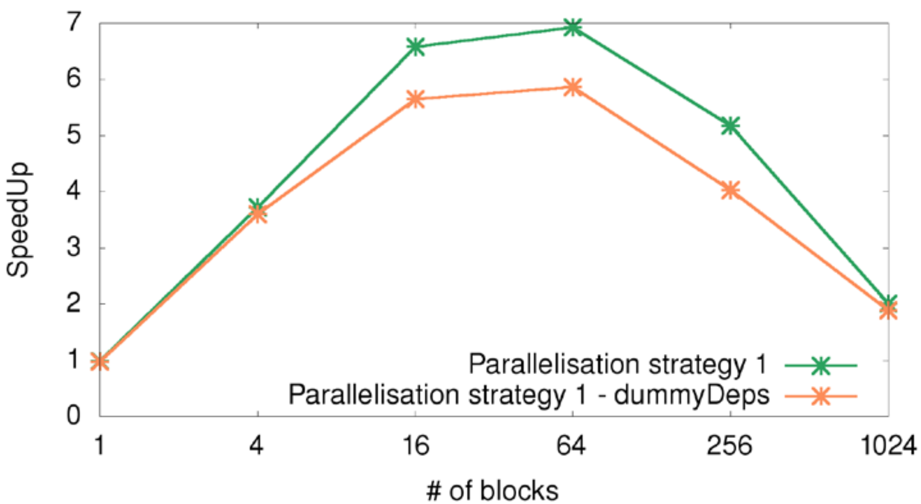
Nanos++[1] captures dependencies among overlapped portions of arrays, introducing extra overhead when the number of tasks is very high

[1] A run-time targeting HP domain, https://pm.bsc.es/nanox

Tasking Model: Run-time Overhead

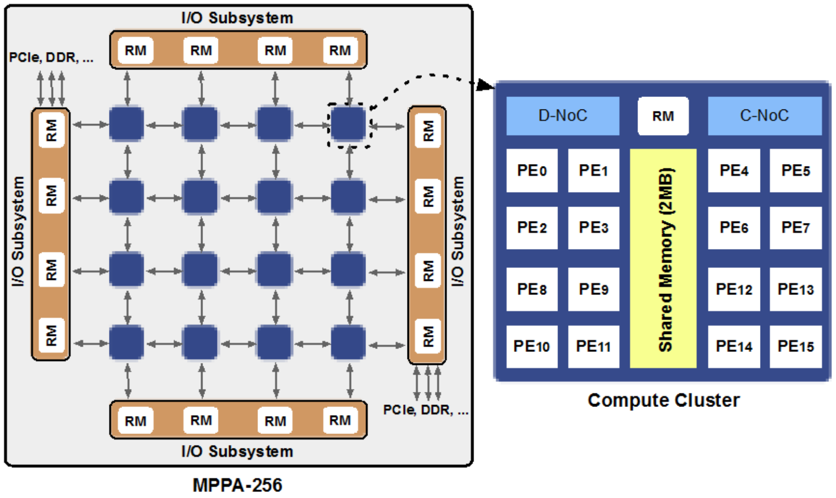
- The size of the hash table may significantly increase when using the `depend` clause intensively
  - *dummyDeps* replaces the `taskwait` directive with fake dependencies





Acceleration Model: 256 MPPA Kalray

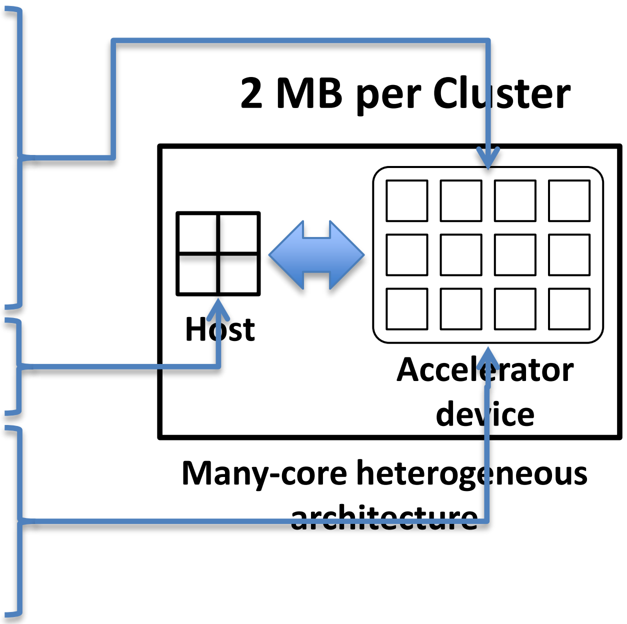
- 4 host subsystems (I/Os) featuring 4-core each
- 16 acceleration devices (clusters) featuring 16-cores each connected to a 2MB on-chip memory
  - The sensor frame already occupies 8 MB



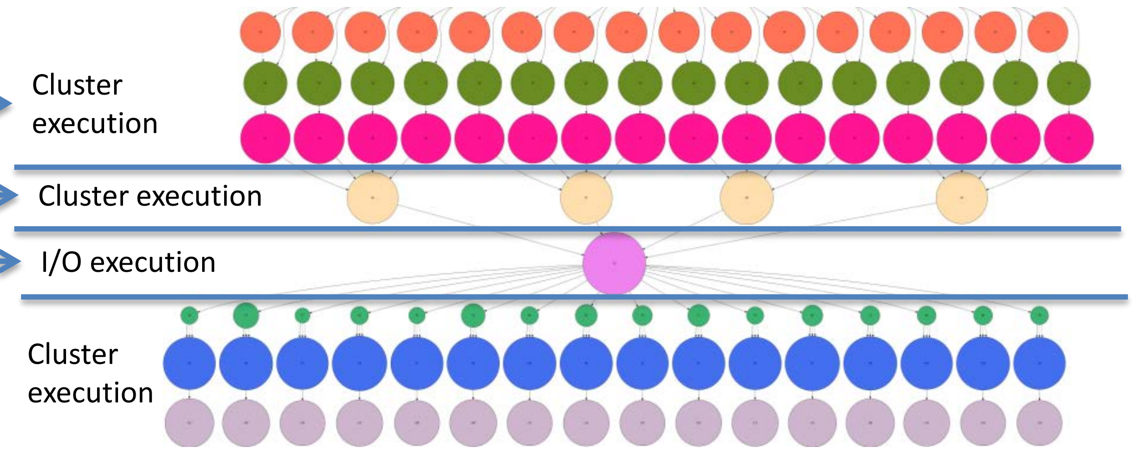
Computation distribution

- Application memory requirements when dividing 8 MB frame in 64 blocks (8x8)

Application Function	Sequential	Parallel
detectSaturation	8.5 MB	1MB (4 frame blocks)
subtractSuperBias	16 MB	
nonLinearityCorrection	8 MB	
subtractPixelTopBotton	8 MB	1 MB (8 frame blocks)
subtractPixelSides	8 MB	8 MB (complete frame)
detectCosmicRay	52 MB	1.5 MB (2 frame blocks)
linearLeastSquaresFit	48.5 MB	
calculateFinalSignalFrame	32 MB	



OpenMP `target` directive

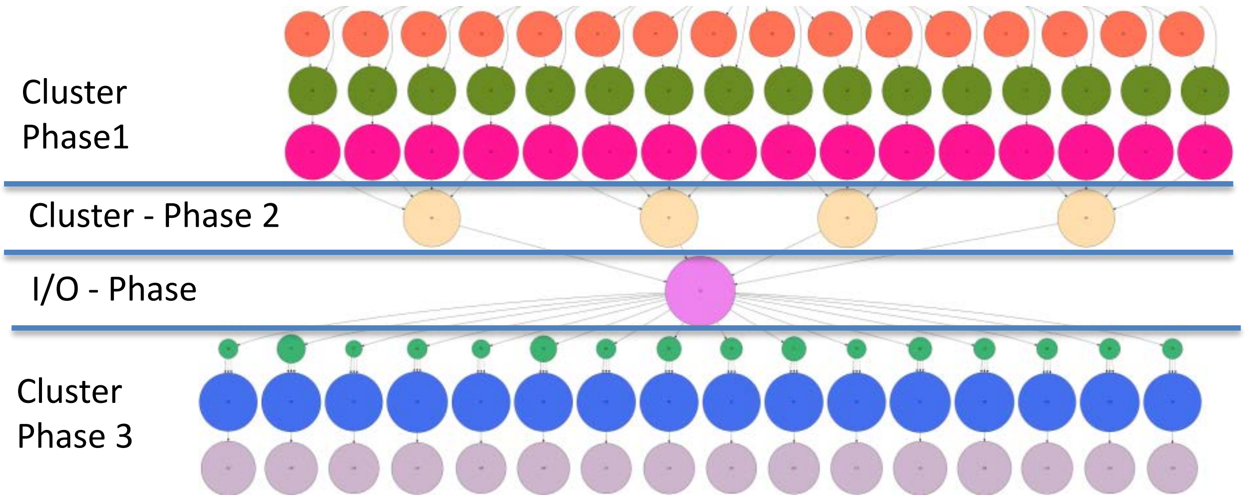


4 blocks sent

```
for(i=0;i<8;i++)
  for(j=0;j<8;j+=4)
#pragma omp target map(tofrom:frame[i][j],frame[i][j+1],frame[i][j+2],frame[i][j+3])
                   depend(out:frame[i][j],frame[i][j+1],frame[i][j+2],frame[i][j+3])
                   nowait {
  detectSaturation (i,j); subtractSuperBias(i,j); nonLinearityCorrectionPolynomial(i,j);
}
for(i=0;i<8;i++)
#pragma omp target map(tofrom:frame[i][0],frame[i][1],frame[i][2],frame[i][3]
                           tofrom:frame[i][4],frame[i][5],frame[i][6],frame[i][7])
                   depend(in:frame[i][j],frame[i][1],frame[i][2],frame[i][3]
                              in:frame[i][4],frame[i][5],frame[i][6],frame[i][7])
                   nowait
  subtractReferencePixelTopBottom(j);
#pragma omp taskwait
subtractReferencePixelSides();
...
```

8 blocks sent

Application's Performance Speed-up



	Sequential Time (seconds)	Parallel Time (seconds)	Speed-up
CLUSTER-Phase 1	69.180	3.120	**22.17x**
CLUSTER-Phase 2	16.641	1.458	**11.41x**
IO-Phase	17.360	17.360	**1x**
CLUSTER-Phase 3	175.799	11.641	**15.10x**
Overall computation time	**278.980**	**33.579**	**8.3x**

OpenMP on a non-POSIX OS

- We evaluated the support that OpenMP libgomp run-time requires on non-POSIX OS
- ERIKA Enterprise http://erika.tuxfamily.org
  - Open-source automotive certified kernel
  - Minimal memory consumption (footprint of few KBs), run-time latencies and error-prone conditions
  - Configuration settings statically defined at compile-time
  - The set of services provided is very small and simple compared to POSIX standard

OpenMP on a non-POSIX OS

- We identified a minimal set of low-level primives to support an embedded and lightweight libgomp
  - Thread management primitives for job management
  - Synchronization primitives
  - Memory management primitives (standard libc)
- Additional footprint very limited

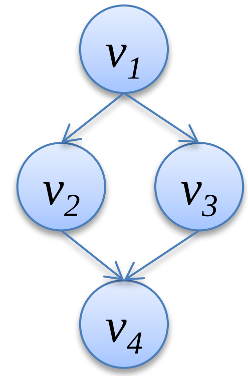
Extra Code footprint	Extra RAM usage per core
1024 – 2048 bytes	128 bytes for each core

Time Predictability Benefits

OpenMP and Real-time embedded domain:
A Possible Union?

OpenMP and Real-time Systems?

- DAG-based real-time scheduling models
  - System composed of a set of periodic directed acyclic graph (DAG) tasks: $G=(V, E)$
    - v_i in V is a job characterised with WCET
    - (v_i, v_j) in E means v_j cannot start until v_i finishes



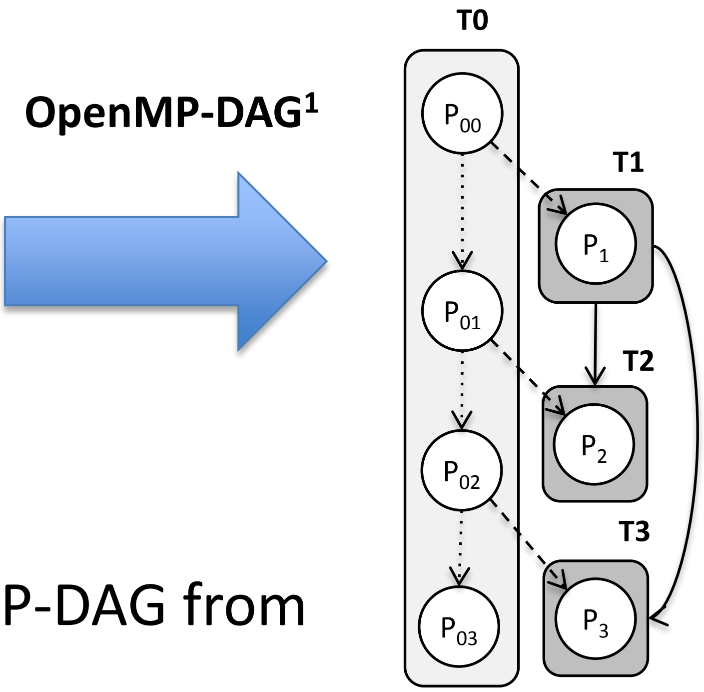
- OpenMP tasking model resembles the DAG-based representation

OpenMP	DAG-based real-time
OpenMP application	DAG-task (G)
`task` directive	Jobs in V
Synchronization directives ,task creation, control flow	Edges in E

Constructing the OpenMP-DAG

```
#pragma omp task {                          // T0
  p00
  #pragma omp task depend(out:x,y) {p1}// T1
  p01
  #pragma omp task depend(in:x) {p2}    // T2
  p02
  #pragma omp task depend(in:y) {p3}    // T3
  p03
}
```

OpenMP-DAG[1]



[1] Compiler methods to construct OpenMP-DAG from an OpenMP application available in

- Roberto E. Vargas, Sara Royuela, Maria A. Serrano, Xavier Martorell, Eduardo Quiñones, *A Lightweight OpenMP4 Run-time for Embedded Systems*, in AspDAC 2016
- P-SOCRATES FP7 project (www.p-socrates.eu)

Time Predictable OpenMP-DAG

- Timing characterization of nodes in DAG (OpenMP tasks)
  - Account for the potential execution time variation that hardware interferences can possibly introduce
    - **No interferences**: Tasks execute in isolation in one core
    - **“Some” level of interferences**[1]: Tasks execute in parallel interfering among each other
    - The **“maximum” level of interferences**[2]: Tasks execute with benchmarks designed to stress hardware resources (opponents)

[1] Gabriel Fernandez, Javier Jalle, Jaume Abella, Eduardo Quiñones, Tullio Vardanega and Francisco J Cazorla, *Resource Usage Templates and Signatures for COTS Multicore Processor,* In DAC 2015
[2] Mikel Fernandez, Roberto Gioiosa, Luca Fossati, Marco Zulianello, Eduardo Quiñones, Francisco J. Cazorla, *Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain*, in EMSOFT 2012

Time Predictable OpenMP-DAG

- Computation of **application's** worst case response time bound
  - Static allocation approaches based on sub-optimal heuristics[1]
  - Work-conserving dynamic schedulers (as implemented in the Libgomp OpenMP run-time)[2]

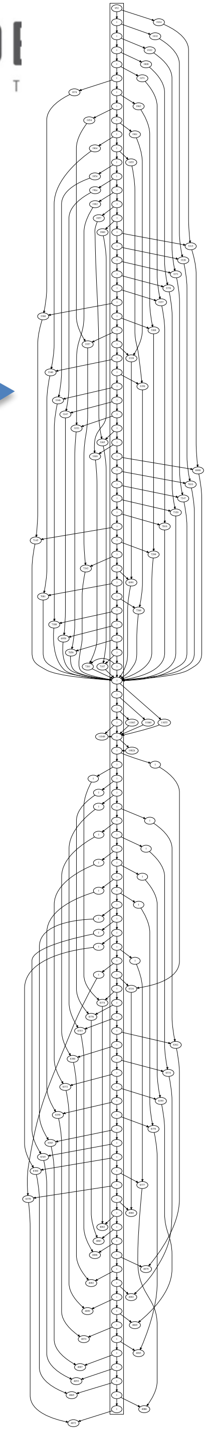
$$R^{ub} = \mathrm{len}(G) + \frac{1}{m}\left(\mathrm{vol}(G) - \mathrm{len}(G)\right)$$

[1] K. E. Raheb, C. T. Kiranoudis, P. P. Repoussis, and C. D. Tarantilis. *Production scheduling with complex precedence constraints in parallel machines*, In Computing and Informatics 2012

[2] Maria A. Serrano, Alessandra Melani, Roberto Vargas, Andrea Marongiu, Marko Bertogna and Eduardo Quiñones, *Timing Characterization of OpenMP4 Tasking Model*, in CASES 2015

Time Predictability of the infra-red application

- Extract the OpenMP-DAG of parallelisation strategy 1 with 16 blocks
- WCET computed measuring tasks in isolation (no interferences) and adding a 40% (safety margin)
  - 4-core Intel Core i7-4600U at 2.1 GHz
- Run-time and OS overhead not considered



Measured execution time	Dynamic Scheduler	Static Allocation				
		LPT	SPT	LNSNL	LNS	LRW
96	131	117	120	117	117	118

(in milliseconds)

Time Predictable OpenMP-DAG

- Computation of **system's** worst case response time bound
  - OpenMP tasking model resembles the DAG-based limited preemption scheduler[1]

OpenMP	DAG-based Limited preemption
OpenMP application	Real-time task
Task-parts	Non-preemptive regions
Task Scheduling Points	Preemption points

$$R_k^{ub} \leftarrow L_k + \frac{1}{m}\left(vol(G_k) - L_k\right) + \left\lfloor \frac{1}{m}\left(I_k^{lp} + I_k^{hp}\right) \right\rfloor$$

[1] Maria A. Serrano, Alessandra Melani, Marko Bertogna and Eduardo Quiñones, *Response-Time Analysis of DAG Tasks under Fixed Priority Scheduling with Limited Preemptions*, in DATE 2016

Summary and Conclusions

- OpenMP can be effectively used to developed future parallel real-time embedded systems
  - Easy to program and express parallelism
  - Good performance and time predictable
  - Supported by a large set of parallel architecture
- Performance speed-up of an OpenMP parallel version of the space application
  - 11x on a 16-core Intel(R) Xeon(R) CPU E5-2670
  - 3.8x on a quad-core Leon3
  - 8.3x on a 256 MPPA Kalray (heterogeneous architecture)

Summary and Conclusions

- Evaluate the run-time overhead in terms of speed-up degradation and memory usage
- Investigate the use of OpenMP (libgomp) in non-supported POSIX OS
- Evaluate the response time analysis of the application under
  - Work-conserving dynamic scheduler
  - Static allocation approaches

Future work: OpenMP in space

1. The timing analysis of OpenMP not completed
   - No sound and trustworthy timing analysis method to compute the WCET of jobs in the DAG
     - The run-time overhead must be take into consideration
   - Current timing and scheduling techniques only assumes homogeneous architectures
     - Scheduling techniques for efficient (and predictable) host/device computation and data transfer are missing
     - The construction of the OpenMP-DAG does not include the accelerator model
2. Response time analysis of system composed of multiple OpenMP applications
3. Efficient and lightweight OpenMP run-time for embedded applications

Acknowledgments

- FP7 P-SOCRATES project, www.p-socrates.eu

P-SOCRATES
Parallel Software Framework for Time-Critical many-core Systems

- Sebastian Huber, from Embedded-brains
- ESA colleagues: Luca Fossati, Marcel Verhoef, Athanasios Tsiodras