Title: [D7_D9_D10] Abstract, Executive Summary and Final Report

	NAME AND FUNCTION	DATE	SIGNATURE
Prepared by	**Franck Wartel** *Data processing & On Board Software* *TLS – TESOD5* *(Airbus DS, Toulouse)*		
Prepared by	**Sara Royuela,** **Adrian Munera,** **Eduardo Quinones** *Predictable Parallel Computing group* *(Barcelona Supercomputing Center)*		
Authorized by	**Matthieu Couderc** *Data processing & On Board Software* *TLS – TESOD5* *(Airbus DS, Toulouse)*		

This document has been written in the scope of ESA project HP4S
High Performance Parallel Payload Processing for Space
[ITI study Ref. 44, ESA-ESTEC Contract N° 4000124124/18/NL/CRS]

Export Control Information

This document contains EU or/and US Export Controlled technology (data):

YES ☐ NO ☒

If **YES**:

1/ European/French/German regulation controlled content

☐ Technology contained in this document is controlled by the European Union in accordance with dual-use regulation 428/2009 under Export Control Classification Number [x**E**xx]. ***(1)***

☐ Technology contained in this document is controlled by Export Control regulations of French Munitions List under Export Control Classification Number [MLXX or AMAXX]. ***(1)***

☐ Technology contained in this document is controlled by Export Control regulations of the German Foreign Trade and Payments Regulation (AWV) as per Part I of Annex AL (Export List) to AWV under Export Control Classification Number [MLXX or AMAXX]. ***(1)***

2/ US Regulation controlled content

☐ Technology contained in this document is controlled under Export Control Classification Number [x**E**xxx] by the U.S. Department of Commerce - Export Administration Regulations (EAR). ***(1)***

☐ Technology contained in this document is controlled under Export Control Classification Number [x**E**xxx] by the U.S. Department of State - Directorate of Defense Trade Controls - International Traffic in Arms Regulations (ITAR). ***(1)***

3/ Other Regulation(s)

☐ Technology contained in this document is controlled under Export Control Classification Number [x**E**xxx] by the [XXX].

(1) See applicable export control license/authorization/exception in Delivery Dispatch Note.

Dissemination is only allowed to legal or natural persons with right to know who are covered by an appropriate export license/authorization/exception.

Document change log

ISSUE	DATE	MODIFICATIONS	OBSERVATIONS
1.0	13-Dec-20		Initial draft issue for Review

Reference Documents

ALIAS	TITLE	REFERENCE	ISSUE	DATE
SoW	[HP4S] Study Statement of Work	ESA-TRP-TECTI-SOW-004127	1.0	27/01/2017
Proposal	[HP4S] Study proposal	TSES4.PC.VC.737741.17	1.0	21/04/2017
Contract	[HP4S] Study Contract	4000124124/18/NL/CRS	1.0	05/06/2018
[D1]	System needs and perspective for use of OpenMP in payload data processing applications	ADST-TN-1000485161	1.0	26/11/2018
[D2]	OpenMP evaluation plan	ADST-TN-1000485166	1.0	26/11/2018
[D4]	Use Case N°1 design report	ADST-TN-1000583806	1.3	12/2020
[D6]	Evaluation Report	ADST-TN-1000885198	1.0	12/2020
[D6.AppendixC]	ICPP 2020 paper	[D6_AppendixC]ICPP_2020.pdf	-	06/2020

Table of Contents

Figures

Tables

1 ABSTRACT

Next generation space missions will require more capable computers in order to implement either advanced navigation and control algorithms needed to increase the spacecraft autonomy and agility or on the payload side with complex scientific payload data pre-processing algorithms. The advent of low-power multi- and many-cores processor architectures provides critical real-time embedded systems with an unprecedented performance level, opening the door to implement more intelligent systems. These architectures however, pressure the software development process, as applications must be parallelised in order to exploit the huge performance capabilities they offer.

The complexity of parallel programming has already been identified as a major challenge in general purpose computing, and it is now exacerbated in the critical embedded systems domain, due to the non-functional properties that these systems must fulfilled, e.g. functional safety and time predictability.

The majority of architectures incorporate parallel programming models in their software development kits with the objective of easing programmability and exploiting performance capabilities. These models provide an interface that entails a level of abstraction enabling to expose parallelism while hiding the complexity of the processor architecture. Parallelism is accomplished by the definition of independent execution units and synchronization mechanisms that guarantee the correct control- and data-flow.

The decision on what parallel programming model is used directly affects portability. While high level models can be compiled and executed on different architectures supporting the same model without modifying the application, low level APIs (e.g. Pthreads) may require some tuning before they can be ported to another machine.

Overall, parallel programming models are a fundamental element to exploit the huge performance capabilities offered by the newest parallel heterogeneous architectures.

The purpose of this ITI is to demonstrate the benefits of using one of the most well-known parallel programming models, i.e. ***OpenMP***, for the development of parallel space applications, in terms of performance, programmability and portability.

Two main goals are identified:

- **G1 - Improve overall system performance** by exploiting the most advanced parallel embedded architectures targeting the space domain
- **G2 Improve the parallel programming productivity** by reducing the initial development efforts of systems based on parallel architectures,

By selecting and porting two representative payload processing use cases to OpenMP parallel programming model, and evaluating their deployment on two high-end computing devices, GR740 in the radiation hardened components family and latest Kalray Coolidge MPPA as COTS, the project execution successfully demonstrated that usage of OpenMP parallel programming could facilitate the development, and analysis of parallel real-time space applications.

Development framework is completely based on open source solutions. Cross compiler for the selected targets is mainstream GNU GCC and includes OpenMP 4.5 runtime and open source observability tools provided by Barcelona Supercomputing Center (Paraver and Extrae) were ported from HPC mainstream to the selected hardware targets, and exercised through the selected software use cases.

2 EXECUTIVE SUMMARY

The principle behind OpenMP (as well as many other parallel programming models) is that parallel computation is user-directed, but not fully controlled by the programmer. As a result, the parallel execution is delimited by the user and orchestrated by the runtime. This has the freedom to distribute the parallel execution among the computing resources provided by the underlying architecture in the most convenient way, with the objective of maximising performance speed-up.

Although this principle presents clear benefits in terms of programmability, portability and performance, it may also impact negatively on ensuring the safety properties of the system. The reason is that the developer in charge of guaranteeing functional and non-functional requirements has very little control on how the parallel execution is managed at run-time.

HP4S study defined two strategic end goals:

G1. Improve overall system performance.
Effectively master and exploit the most advanced parallel embedded architectures targeting the space domain.

G2. Improve the parallel programming productivity.
Reduce the development efforts of systems based on parallel architectures, while fulfilling system's functional and non-functional (time predictability) requirements.

These two goals derive in a set of technical measurable objectives listed hereafter:

O1. Facilitate the development, timing analysis and execution of parallel real-time space applications using the OpenMP parallel programming model.

O2. Evaluate the interest and porting effort of a list of homogeneous and heterogeneous foreseen COTS and RadHard hardware targets in the space domain with OpenMP programming model and framework.

O3. Adapt the OpenMP runtime libraries to ensure that the timing guarantees devised at analysis time can be guaranteed at deployment time.

O4. Evaluate state-of-the-art compiler techniques to guarantee that parallel OpenMP applications are functionally correct, and so no race conditions or deadlocks will occur.

O5. Demonstrate the portability benefits of the OpenMP parallel programming model.

The key capability to achieve the goals of high performance processing to support future missions is therefore clearly software parallelisation. Indeed, highly parallel computing devices may be useful only if the development of efficient programming techniques for such devices is achieved. The efficiency of programming techniques is twofold in our business where many software applications are specific to a given mission and therefore, globally non-recurring: the technical efficiency (i.e. the capability to use the device's resources efficiently)and the industrial efficiency (mastering and lowering the programmer's efforts for software development and verification).

This study raises the TRL for on-board software parallelisation on the two selected targets – currently at the "proof of concept" level (TRL3). For these targets, usage of laboratory representative implementation allows to reach TRL4.

The study is structured within four technical steps:

1) A **Preparatory phase** dedicated to the inventory of possible application use cases that would make sense for being evaluated during this study as well as an in-depth analysis of the available many/multi-cores processing targets an associated resources w.r.t. their suitability for the use cases and for supporting an OpenMP framework. This phase results in the selection of 2 specific use cases.

2) A **Development phase** dedicated to the preparation of the OpenMP framework on the selected targets and to the parallelisation work on the two selected use cases.

3) An **Evaluation phase** dedicated to the actual porting of the parallelised use cases and benchmarks on the targets and evaluation according to the criteria's defined during the preparatory phase.

4) A **Synthesis phase** dedicated to the analysis of all collected evaluation results and a synthesis for all the study outcomes as well as recommendations for the way forward with respect to parallel processing and multi/many cores targets.

From the implementation perspective, the main contributors to the project are Airbus Defence and Space and Barcelona Supercomputing Center, briefly introduced hereafter.

AIRBUS Defence and Space is a world leader as satellite prime contractor answering needs of space programmes in the various fields of space activities: Telecommunications, Earth observation, Space Science, Navigation, Launch vehicle, manned space flights. This expertise is covering a wide range of activities, from the complete system including both space and ground segments down to equipment level. It includes in particular the prime contracting of Telecommunications, Earth observation and Science satellites and experiments. At its location in the southwest of France, the Toulouse site specialises in satellite prime contracting, design, assembly integration and test for communications, Earth observation and science satellites. Its expertise also extends to avionics, optical instruments, on board software, ground systems and space-based geo-intelligence and telecommunications services. The Data processing and On-Board software advanced studies team involved in this project has a very high level of expertise on topics such as high performance algorithm mapping on software execution platform and its impact on software architectures.

Barcelona Supercomputing Center, established in 2005, serves as the National Supercomputing facility in Spain. The Center hosts MareNostrum, one of the most powerful supercomputers in Europe. The mission of the BSC-CNS is to research, develop and manage information technologies in order to facilitate scientific progress. The BSC-CNS not only strives to become a first-class research center in supercomputing, but also in scientific fields that demand high performance computing resources such as the Life and Earth Sciences. Following this approach, the BSC-CNS has brought together a critical mass of top-notch researchers, high performance and embedded computing experts and cutting-edge supercomputing and high-end embedded technologies in order to foster multidisciplinary scientific collaboration and innovation. At an international level, the BSC-CNS has already participated in an impressive number of activities, including the 6th, 7th and H2020 Framework Programs of the European Commission. In the parallel programming domain, BSC has a long tradition investigating parallel programming paradigms and intelligent runtime systems to increase the productivity when programming HPC and embedded systems to effectively exploit performance out of the target architecture (from embedded many-core processors to large-scale cluster systems, including both homogenous and heterogeneous systems that use accelerators like GPUs) for many years. It is worthy mention that BSC is member of the OpenMP Advisory Review Board (ARB), the non-profit corporation that owns the OpenMP brand, oversees the OpenMP specification and produces and approves new versions of the specification

3 PREPARATORY PHASE

The preparatory phased purpose was to refine and summarize the actual needs, analyse the potential software solutions and associated challenges to build a solid plan for prototyping and evaluation on relevant targets.

3.1 System Needs and perspective for use of OpenMP

3.1.1 Needs of current and future payload data processing applications

There are typically two different categories of on-board data processing platforms in spacecraft. Some computers are responsible for spacecraft or instrument control and computers, others are responsible for the on-board processing of instruments data, more often located on the Payload segment.

For both categories, we foresee that next generation space missions will require more capable computers, with respect to what we can implement today using proven technologies like the LEON processor: On the control side, spacecraft autonomy and agility need to be improved thanks to new sensors data acquisition and processing for in-situ real-time usage, such as for instance visual based navigation. On the payload side, scientific data pre-processing algorithms as well as more robotics for in-orbit/on-Planet operations are foreseen with, in the long term, the foreseeable usage of artificial intelligence.

Two tracks are identified by Airbus Defence and space, largely shared with other European space Industrial for achieving such goal, and supported by the technology roadmap of the European Space Agency and national agencies:

1) The development of new processing architectures on new rad-hard technologies such as, among others, the recent GR740 and HPDP with the STM-65nm and the future Dahlia/Brave-Ultra targeted for the new STM 28nm FD-SOI (will) provide higher running frequencies with more efficient processing devices. On this technology track, developments of High performance computers are on-going which all require the capability to efficiently develop software on Multi-Core devices.
2) The development of mitigation techniques enabling the use of commercial of the shelves processors (COTS) both as FPGA and micro-processors in the space environment. This way has been supported and matured through the ESA “COTS Based Computers” studies in which Airbus Defence and Space was strongly involved. For instance, this track which is more easily applicable for the LEO orbit has enabled the new Airbus product line “PureLine” based on Automotive quality grade parts.

Both tracks have their specific advantages, drawbacks and limitations which make them both to be tackled with respect to the different categories of missions and markets we face. But a common characteristic is that both will lead to the use of devices enabling parallelisation of the processing. In this context, the emergence of complex computing architecture embedding many processing cores on single device is a real opportunity to drastically increase the on-board processing capability for both of these technology development tracks.

3.1.2 Hardware solutions for high performance data processing

A wide range of devices are relevant for satisfying these needs:

- **FPGAs (Field-Programmable Gate Arrays)**

FPGAs use configurable logic blocks containing LUTs, FFs, DSPs and block RAM along with a configurable routing matrix to implement a wide range of applications. They tend to outperform with streamed algorithms.

- **DSPs (Digital Signal Processors)**
  DSPs are optimized for single thread signal processing applications which mainly use typical signal processing operations. DSPs tend to perform these operations using fewer instructions than GPPs, and with lower power consumption. Some DSPs have a SIMD (Single Instruction, Multiple Data) architecture giving them Data Level Parallelism (DLP) capabilities.

- **GP GPUs (General Purpose Graphics Processing Units)**
  There is a wide variety of GPU architectures, but they usually consist of up to 32 cores, with each core containing up to hundreds of processing units. The cores have their own instruction stream and the same stream is shared by the processing units of a core. The cores have a private, local memory and they communicate through the GPU shared memory.

- **Many-core processors**
  Many-core processors are designed to reach high throughputs using Thread Level Parallelism (TLP). Usually, the cores are quite small in order to fit a high number of them on a chip with a reasonable power consumption.

- **Multi-core GPPs (General Purpose Processors).**
  GPPs outperform in single-threaded applications and mostly take advantage of ILP (Instruction Level Parallelism) thanks to complex pipelined and superscalar architectures. This parallelization is implicit since it is automatically managed by the hardware. GPPs rely on cache memories to reduce the latency of memory transfers. Multi-core architectures make it possible to run some threads in parallel.

Each device has its own advantages but it is important to realize that every design is unique and there is no universal rule for choosing among these devices. Some systems use a combination of devices to implement the overall application. To choose the most suitable hardware solution for a specific algorithm, several items such as cost, speed, flexibility, power and optimization as well as the design environment (team's skill, design tools, licensing…) should be taken into account.

The following table summarizes the characteristics of the main processing devices (note that there can be a lot of variability within a type of processing platform and that this table only provides trends).

Table 1: Processing platforms comparison

Device	Parallelism	Gate Reuse And Time Sharing	Flexibility/ Programmability	Absolute Power Consumption
FPGA (SRAM)	High	Some	Moderate	Moderate
DSP	Some	Moderate	High	Low
GP GPU	High	High	High	High
Many-core	High	High	High	High
Multi-core GPP	Moderate	High	High	Moderate

3.1.3 Parallel programming frameworks

The complexity of parallel programming has already been identified as a major challenge in general purpose computing, and it is now exacerbated in the critical embedded systems domain, due to the non-functional properties that these systems must fulfil, e.g. functional safety and time predictability.

OS's and RTOS's already provide all the services required to designing a parallel application. Nonetheless, reaching a high level of parallelism by only using the services of the OS is hard as it requires finding a good balance between the loads of the different cores with very rudimentary tools.

The majority of architectures incorporate parallel programming models in their software development kits with the objective of easing programmability and exploiting performance capabilities. These models provide an interface that entails a level of abstraction enabling to expose parallelism while hiding the complexity of both the processor architecture and the OS. Parallelism is accomplished by the definition of independent execution units and synchronization mechanisms that guarantee the correct control- and data-flow.

OpenMP presents the following advantages over its competitors:

- Different evaluations demonstrate that OpenMP delivers performance and efficiency tantamount to highly tunable models such as TBB, CUDA, OpenCL, and MPI. Regarding low-level libraries such as Pthreads, it offers multiples advantages such as: a) robustness without sacrificing performance, and b) OpenMP does not lock the software to a specific number of threads.
- The code can be compiled as a single-threaded application by either disabling support for OpenMP or assigned a single computing unit (i.e., OpenMP thread), thus easing debugging.
- OpenMP has a large and experienced community that has constantly reviewed and augmented the language for the past 20 years achieving great expressiveness. OpenMP is the de-facto shared-memory programming model standard. The language defines a very powerful tasking model that allows expressing fine-grained, both regular and irregular, and highly-dynamic task parallelism, augmented with features to express task dependencies. The latest specification of OpenMP also incorporates new features that facilitate the coupling of a main host processor to accelerator devices by defining both synchronous and asynchronous communication between them.
- OpenMP is widely implemented by several chip and compiler vendors (e.g. GNU, Intel, IBM, Kalray, Texas Instruments).
- Typically, ensuring functional correctness in languages that are not developed for such purpose is not straightforward. Nonetheless, OpenMP static correctness techniques are quite mature and simple compared to other parallel programming models which are not designed with correctness in mind (e.g. the low level Pthreads API or the Message Passing Interface, MPI).

Overall, OpenMP is a good candidate to be used in the real-time embedded domain in general, and in the space domain specifically by virtue of its benefits.

3.1.4 Correctness and time predictability considerations

Space applications and embedded software in general, are driven by real time constraints and the need to validate that the deadlines defined for a piece of software are met. These constraints are linked to the performance of the system and in this context the focus is rather on worst case

performance than average performance. Controlling the way the software is executed on the hardware platform is crucial when it comes to validating the timing behavior of the complete application. It is worth noting that some space applications have softer requirements on timing constraints.

The principle behind OpenMP (as well as many other parallel programming models) is that parallel computation is user-directed, but not fully controlled by the programmer. As a result, the parallel execution is delimited by the user and orchestrated by the runtime. This makes it possible to distribute the parallel execution among the computing resources provided by the underlying architecture in the most convenient way, while maximizing speed-up.

Although this principle presents clear benefits in terms of programmability, portability and performance, it may also impact negatively on ensuring the safety properties of the system. The reason is that the developer in charge of guaranteeing functional and non-functional requirements has very little control on how the parallel execution is managed at run-time.

There is therefore an urgent necessity to ensure that decisions taken at run-time maintain the guarantees of system correctness endorsed during design and implementation. Hence, a parallel framework (considering both compiler and runtime) targeting a critical real-time embedded system must ensure that processor resource allocation, either static or dynamic, maintains the response time analysis performed at analysis time. Additionally, such a framework must also ensure the functional correctness of the parallel execution safeguarding it from data races and deadlocks.

OpenMP offers two ways to synchronize threads: via directives (master and synchronization constructs such as critical and barrier), and via runtime routines (lock routines such as `omp set lock` and `omp unset lock`). Although both mechanisms may introduce deadlocks, the latter is much more error-prone because these routines work in pairs. The former may cause deadlocks if various critical constructs with the same name are nested. The latter may cause errors in the following situations: attempt to access an uninitialized lock, attempt to unset a lock owned by another thread or attempt to set a simple lock that is in the locked state and is owned by the same task. Moreover, OpenMP introduces the concept of nestable locks, which differ from the regular locks in that they can be locked repeatedly by the same task without blocking.

Another important phenomenon to be considered are race conditions that appear in a concurrent execution when two or more threads simultaneously access the same resource and at least one of them is a write. This situation is not acceptable for a safety-critical environment since the results of the algorithm are non-deterministic. The problem of detecting data races in a program is NP-hard. On account of this, a large variety of static, dynamic and hybrid data race detection techniques have been developed over the years.

On the one hand, dynamic tools extract information from the memory accesses of specific executions. Despite this, there exist algorithms capable of finding at least one race when races are present, as well as not reporting false positives. On the other hand, static tools still seek a technique with no false negatives and minimal false positives. Current static tools have been proved to work properly on specific subsets of OpenMP such as having a fixed number of threads or using only affine constructs. A more general approach can be used to determine the regions of code that are definitely non-concurrent. Although it is not an accurate solution, it does not produce false negatives, which is paramount in the safety-critical domain. Therefore, the previously mentioned techniques can be combined to deliver conservative and fairly accurate results.

Finally, resiliency is a crucial feature in the safety-critical domain. However, OpenMP does not prescribe how implementations must react to erroneous (or unexpected) situations. In that

respect, if the error is produced by the environment, users may want to define what recovery method needs to be executed. Several approaches have been proposed with the aim of adding resiliency mechanisms to OpenMP. There are four different strategies for error handling: exceptions, error codes, call-backs and directives.

3.2 OpenMP Evaluation Plan

Based on the definition of those requirements, a work plan for applicability of OpenMP on different targets was established, so as to answer the project goals objectives and provide crucial pieces of information amongst which :

- a measurable view of the performance of high demanding algorithms with two different multiple cores targets using an OpenMP implementation
- feedback and experience on the efficiency of OpenMP to parallelize application
- some practical view on the portability offered by the OpenMP framework

A set of criteria was defined and applied to select software and hardware to be exercised.

3.2.1 Use Cases selection

Based on a set of criteria not detailed in this public report but main objective was to optimize the add value/project effort ratio, two image processing use cases were selected.

3.2.1.1 HRGEO use case

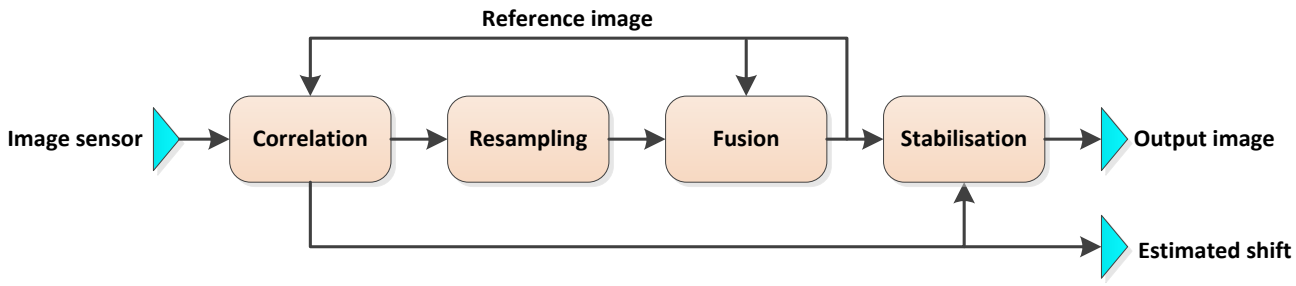


Figure 1: HRGEO Algorithm Overview

The complete algorithm is divided into several computation stages. The sensor image is a RGB image with the three composites values for each pixel (no bayering is considered here).

The algorithm includes image registration, resampling and fusion. The goal of the algorithm is to improve the S/N ratio on a static image by merging multiple images from a stream coming out of a more or less steady camera. The algorithm creates and manages a geometric model which fits the image displacement. It accumulates the successive images into a single reference image after compensating for the displacement with a sub-pixel accuracy.

The algorithm considers chips (fraction of the whole image) and searches the maximum correlation between the chip from the reference image (accumulation) and the chip from the current image along both dimensions and within the exploration range. The geometric model considers only translation. When all the chips displacements have been estimated then the average displacement over all the chips is computed and applied to the resampling function which

moves the accumulated image (reference) in the same position as the current image. This function resamples the reference image by using a bank of convolution filters. Together, the correlation and the resampling stages perform the registration. The fusion stage applies a configurable gain to the current image and adds it to the reference image.

This algorithm while not being the state of the art of this type of processing in Airbus Defence and Space is representative of a complex algorithm implementation process.

The implementation uses fixed-point representation for image correlation and keeps only a few operations in double precision.

3.2.1.2 Adaptive Mirror use case

Future earth observation satellites with high resolution requirements will need primary mirrors of huge size. Due to their big size, they will suffer thermoelastic deformations. Adaptive optics allows fixing these defaults thanks to a deformable mirror actuated by a myriad of piezo actuators. The wave front defaults are measured on board by a Shack-Hartmann analyzer with its image sensor.

Wavefront sensors aim at analyzing the shape of an incident beam's wavefront in order to identify aberrations caused by light traveling through individual optics or optical assemblies.

Shack-Hartmann wavefront sensors achieve this by dividing the beam into an array of discrete intensity points using a microlens array. To measure the wavefront of a beam, the light is aligned so that it is normally incident on the microlens array at the front of the wavefront sensor. Each lens collects the light filling its aperture and forms a single focal spot on the CMOS camera sensor, which is located at the focal plane of the microlens array. If the wavefront is planar, all focal spots are centered directly behind each respective lens, coincident with the optical axis of each. The result is a regularly spaced grid of spots on the camera sensor, as illustrated in Figure 1Figure 2. These spot locations are called reference spot positions, and they compose the reference spot field.

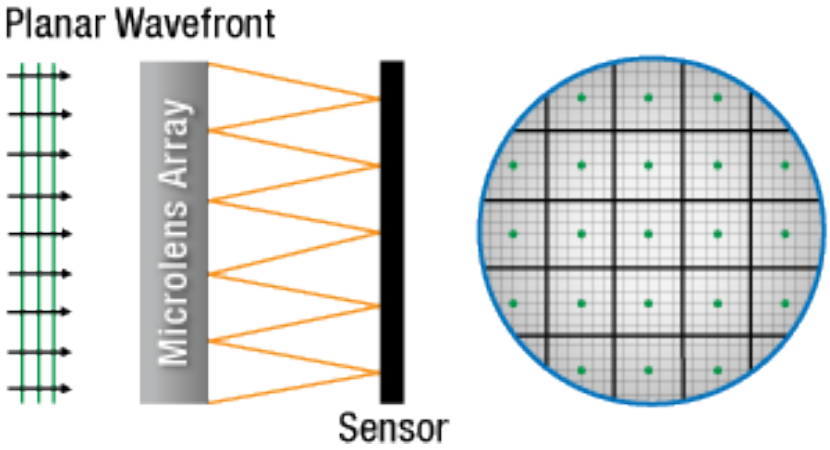


Figure 2: Planar Wavefront (credit *https://www.thorlabs.com*)

Error! Reference source not found. Figure 3. By comparing the locations of the spots in the measured spot field with those in the reference spot field, the shape of the wavefront can be calculated, and this is the aim of the image processing algorithm of this use case.

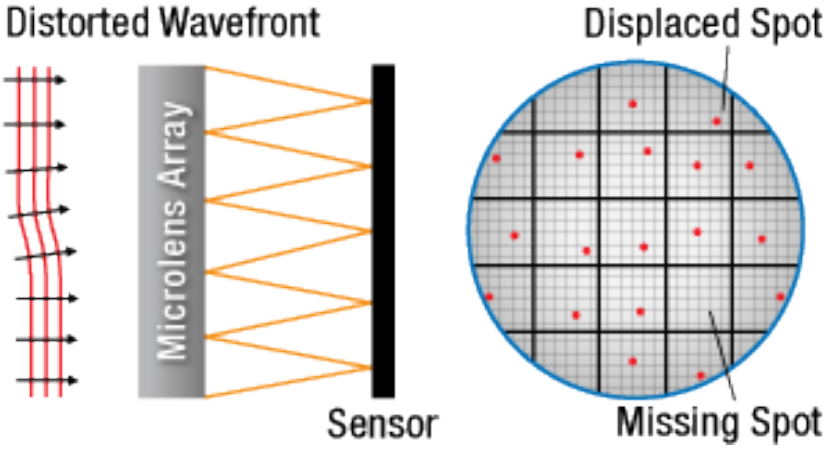


Figure 3: Distorted WaveFront (credit *https://www.thorlabs.com*)

For robustness reasons and to have several measurements on several points on the mirror, 3 detectors provide the images. It has been demonstrated in previous studies that the processing of one detector could fit in a LEON processor. The control of the piezo actuators would require another LEON processor. In the use case of this study, we integrate the processing of the 3 detectors actuators on the same multicore processor.

The algorithm is composed of two steps. It is applied on a 12x12 matrix of lenses for 3 detectors that are independent. But not all lenses must be processed, only the ones with a light intensity above a certain threshold. The light intensity is a constant, thus the number of lenses to be processed can be defined beforehand. 104 lenses have to be processed in the current implementation.

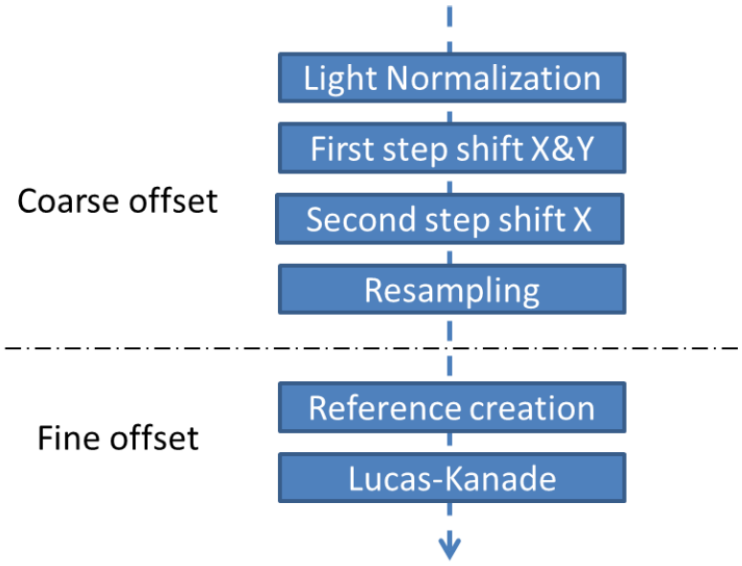


Figure 4: MIRROR application processing steps

As illustrated in Figure 4, the image processing partition is composed of two parts: one that computes a first rough estimate of the shifts and interpolates the image with this shift; and another one that uses the interpolated image to compute a very precise shift.

The coarse offset is composed of a normalization of the images with the light matrix, then the sliders algorithm is applied on all valid lenses, finally a bilinear interpolation applies the shift computed for the next step. The sliders are composed of two steps of convolution, but the second step is executed only if the first shift is bigger than 0.5 on the X axis. For the reference batch provided with the algorithm, it happens on 26% of the lenses and represents an increase in the number of operations of 20% when it is applied. It means that the duration of the coarse offset has a small variability in duration due to the data processed.

The fine offset first computes the reference lens from all the other lenses then it uses the Lucas-Kanade algorithm to compute the shift between the reference lens and the others. The execution time of this algorithm does not depend on the data. This part of the algorithm cannot be started for a detector before all lenses (for the same detector) have been resampled by the previous step.

3.2.2 Hardware Targets Selection

Based on a set of criteria not detailed in this public report on purpose, two hardware targets were selected amongst the following candidates { GR740, HPDP, RC64 } for the radiation hardened targets and { Zynq, Zynq MPSoC, Kalray Coolidge MPPA } for the COTS ones.

The GR740, the Coolidge and the Zynq 7000/UltraScale+ appeared to be the targets the best suited to the HP4S study. This is mostly because they implement hardware cache coherency mechanisms, they support SMP and their software stack is based on open-source tools. On the other hand, the other two targets that are considered (the RC64 and the HPDP) have software stacks based on proprietary tools and a high effort would be required to implement a reduced OpenMP runtime on these targets.

Final selected ones GR740 and MPPA Coolidge are briefly presented hereafter:

3.2.2.1 GR740

The GR740 device is a radiation-hard quad-core fault-tolerant LEON4 SPARC V8 processor. The processor is organized around five AMBA AHB buses. For the purpose of this study, we are interested on those impacting on the parallel computation, i.e., the 128-bit Processor AHB bus and the 128-bit Memory AHB bus. The former connects the four LEON4 cores connected to the shared L2 cache. The latter connects the L2 cache and the main external memory interface (SDRAM) and attaches a memory scrubber (see Figure 19). The cache system is composed of 4KB L1 instruction and data caches per core, and a 2MB L2 unified cache shared among cores. The L1 data caches have one valid bit per cache line and implements a write-through policy with a double-word write-buffer. Data may exist in both the L1 and L2 caches, or only in L1 or L2. The cache system implements an AMBA AHB master to load and store data to/from the caches. A bus-snooping on the AHB bus maintains cache coherency for L1 data caches. The L2 cache works as an AHB-to-AHB bridge, caching the data that is read or written via the bridge. A front-side AHB interface is connected to the Processor AHB bus, while a backend AHB interface is connected to the Memory AHB bus.

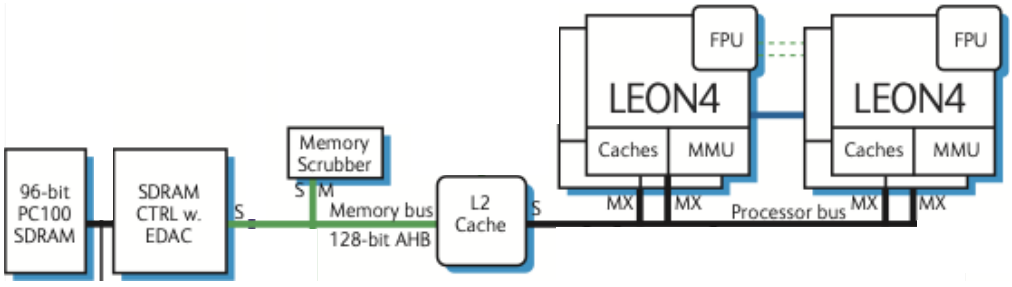


Figure 5: Memory hierarchy of the GR740 processor

Software Environment

GR740 supports RTEMS-SMP version 5.x, a POSIX-based operating system targeting the embedded computing domain. RTEMS-SMP implements all the primitives needed to support parallel execution, including the pthread library upon which the implementation of OpenMP in GNU-GCC (named *libgomp*) is based on. Concretely, the selected RTEMS-SMP for GR740 supports GCC 7.2 This version implements the OpenMP specification 4.5, which will be the one considered in this project. ESA is working towards a qualifiable RTEMS-SMP based on 5.x, and a evaluation is based on beta version of RTEMS-SMP 5.1 supporting GR740 released and incorporated in Cobham Gaisler RCC SDK.

3.2.2.2 MPPA Coolidge

The MPPA Coolidge is a none radiation-hardened 80-core processor architecture from Kalray. The *Coolidge* architecture, which is the evolution of the *Bostan* architecture, is organized in 5 clusters, each featuring 16 cores and a local memory of 4MB that can be configured as a L2 cache and/or scratchpad (see Figure 6). Each VLIW core includes a coherent 16KB L1 instruction and data cache and a Memory Management Unit (MMU). Cache coherence among L2 caches from different clusters is supported as well for a complete SMP configuration. By doing so, a single application can be spawned across the 80-cores. Coolidge can be configured as an AMP machine as well at cluster level, guaranteeing isolation across clusters as defined by ISO26262 ASIL B and C criticality levels. Coolidge is available since Q1-2020.

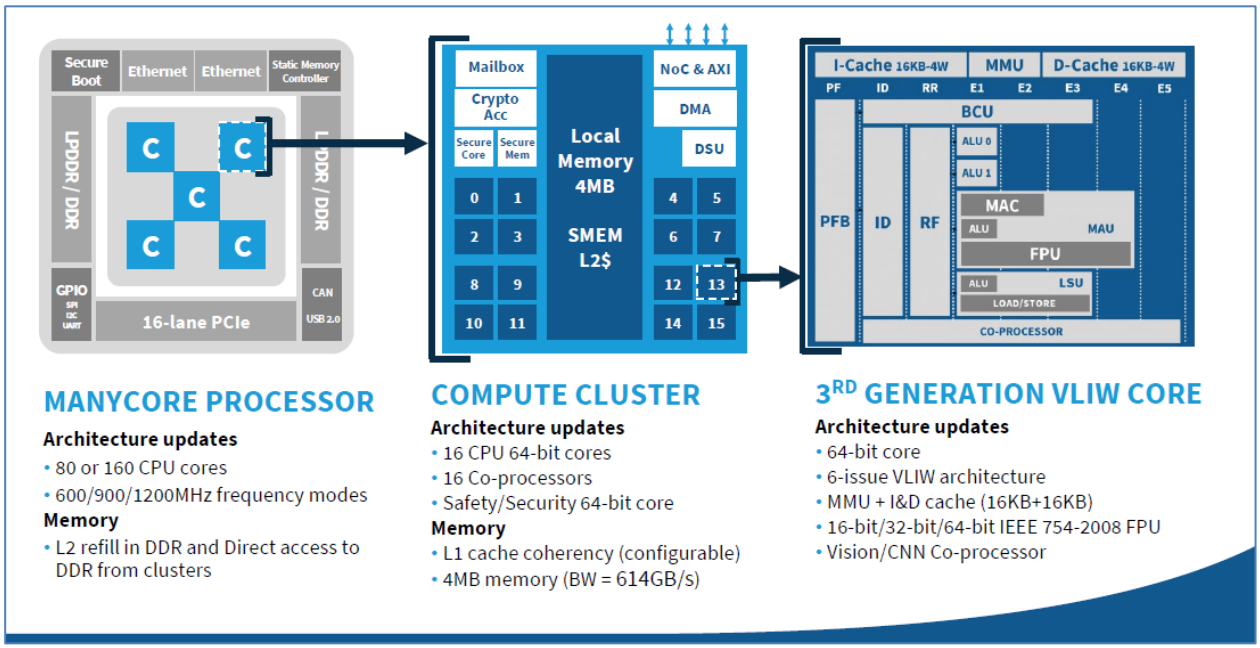


Figure 6: MPPA Coolidge architecture

Software Environment

Coolidge provides a linux-based environment build on GCC 7.5 and supporting OpenMP v4.5 at clustering level. The offloading mechanisms are based on the native MPPA API, and Coolidge also supports OpenCL, allowing to manage clusters as OpenCL Compute Units.

4 IMPLEMENTATION PHASE

4.1 Use case software porting to OpenMP methodology

The implementation was divided into two stages for this study:

- Porting for first sequential execution on the targets.
- Parallelization of the applications with OpenMP (low optimization effort).

The sequential version of application serves as a reference for the considered evaluation metrics, such as memory occupation and of course execution time performance speed up with the parallelized version.

A single parallel implementation is performed for each use-case and the efficiency of this implementation is evaluated on the different targets to evaluate portability capabilities of OpenMP.

The methodology followed to parallelize the use-cases is presented below:

A. Identifying the hotspots of the algorithm

The hotspots can be identified by theoretical analysis of the algorithm or empirically, by profiling the sequential program. Initial measurements and profiling executed on x86_64 benefited from standard HPC tools through GNU gprof x86_64 and Intel VTune Analyzer.

This allowed selecting the main functions to parallelize for each use case, yielding 99.8% of the sequential execution time for HRGEO application and 90% for the adaptive mirror one.

The theoretical speed-up achieved by parallelizing the hotspots is provided by Amdhal's law recalled hereafter.

$$S_N = \frac{1}{\frac{F_{parallel}}{N} + F_{sequential}}$$

- S_N is the theoretical speed-up of the whole program with N cores.
- $F_{parallel}$ is the fraction of time consumed by the regions considered for parallelization.
- $F_{sequential}$ is the fraction of time consumed by the sequential regions ($F_{sequential} = 1 - F_{parallel}$).

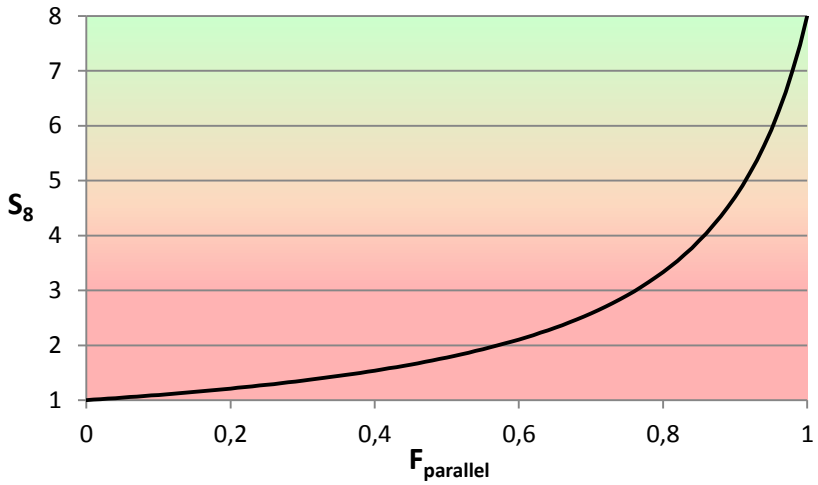


Figure 7: Amdhal's criteria for an 8-core target

B. Choosing an appropriate parallelization strategy for each hotspot

Selecting a parallelization strategy consists in identifying the appropriate type of parallelism and the appropriate OpenMP constructs to implement it. With a good parallelization strategy, it is possible to achieve high performance with minor modifications to the serial code.

The different types of parallelism are:

- ***Data parallelism***, where each thread executes the same task on different sets of data.
- ***Task parallelism***, where each thread executes a different task. This can be used to parallelize recursive algorithms or to create pipelines.

Both data and task parallelism can be implemented with different OpenMP constructs:

- ***Worksharing constructs***: The work is divided into work items which are distributed over a team of threads, either statically or dynamically. The following constructs are used to divide the work into work items:
  - The `loop` constructs **`do`** and **`for`** (data-parallel constructs).
  - The `sections` construct (task-parallel construct).
- ***Tasking constructs:*** Although writing task-parallel programs with worksharing constructs is possible, it is often inconvenient. Tasking constructs have been introduced to provide a convenient syntax and more flexibility for task parallelism. With tasking constructs, tasks are generated by threads whenever they encounter a tasking directive. The generated tasks can be assigned to available threads or they can be queued for deferred execution. The following constructs are used to generate tasks:
  - The `task` construct (task-parallel construct).
  - The `taskloop` construct (data-parallel construct).

To summarize, worksharing constructs are suited to data parallelism (do/for loop constructs) while tasking constructs implement task and data parallelism more conveniently and with more flexibility.

C. Applying parallelization constructs

The parallelization constructs are applied using compilation pragmas.

For both worksharing and tasking constructs, the parallel threads must be created by declaring a parallel region (with the parallel construct) in which a team of OpenMP threads is created. The parallel construct clause num_threads can be used to locally define the number of threads in the team. Then, the appropriate worksharing and/or tasking constructs can be applied in the parallel region. If not specified default parameters are applied based on OMP environment variables, such as max number of threads, or thread scheduling polcies.

D. Applying synchronization and data management clauses

The synchronization and data management clauses are applied to the different worksharing and tasking constructs. Synchronization points can also be added at specific points of the program.

E. Checking the acceleration achieved with the parallelization

For a region parallelized with N cores, the execution time should be theoretically N times smaller than with a single core. If this acceleration factor is not reached, the program is likely to suffer from one of the issue listed in the next point. Depending on the application, achieving an acceleration factor of N can be very challenging and a lower acceleration should be targeted. Under certain conditions, achieving an accelerator factor higher than N is also possible. This phenomenon, named super-speedup, occurs when processor resources are used more efficiently when the program is parallelized, e.g. when the data-set fits within the L1 cache.

F. Investigating the potential issues when the targeted acceleration is not met

The following points can be responsible for low performances:

i. Poor memory accesses due to an inefficient use of cache memory.

To maximize the cache efficiency, each thread should be able to work on independent sets of data and the cache spatial and temporal locality principles should be taken into account. Achieving this may require source code modifications.

ii. Overhead due to synchronization, scheduling or context switching.

Several solutions can be considered depending on the origin of the overhead. For instance:

- Avoid coarse-grain synchronization mechanisms, e.g. barrier or taskwait constructs.
- Use fine-grain synchronization mechanisms by means of the depend task clause
- Merging some tasks to increase the granularity (size) of tasks.

iii. Unbalanced workload across threads.

Several solutions can be considered depending on the origin of the unbalance. For instance:

- Using dynamic scheduling in case of loop constructs.
- Using fine-grained parallelization instead of coarse-grained parallelization

4.2 Porting the compilation tool chain and observability tools

Compilation tool chain already supporting OpenMP 4.5 on both targets through *RCC 1.3-rc6* and *ACE SDK 4.1.0* respectively on GR740 and Kalray Coolidge targets.

There are two main additional tools used during this evaluation: Extrae which is used to instrument code and Paraver which displays the data collected by Extrae. Both have been

developed and are maintained as open-source project by the Barcelona Supercomputing Centre (BSC). Extrae is a rich tool which takes advantages of reusing packages from others developers' community.

It supports natively CUDA, OpenCL and OpenMP environments. In its basic version it allows to measure time and the number of instructions executed. To get memory usage information (which could be very interesting for optimization) then the PAPI library should be used. PAPI provides a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors but the usage of this library requires a lot of re-compilation, including the kernel.

Extrae is enabled by linking with its tracing libraries, which inserts probes inside the functions of the HPC framework (OpenMP, OpenCL, …) used for the implementation. Extrae uses an XML configuration file at runtime. This configuration file is used to enable or disable some profiling features, and to set some parameters. Once the program has been executed, the profiling data can be displayed and analysed with Paraver.

Paraver does not have to be installed on the same target used for the measurement. For this study, Paraver is installed on the x86 host and the trace files are copied from the targets to this host for visualization.

Extrae adaptation to the embedded targets consisted in replacing some of the techniques applied on regular HPC context to comply with the embedded constraints, main changes are:

- Intercepting calls in a static environment
- Managing POSIX dependence
- Retrieving function names
- Traces generation
- Support for hardware counters
- Static environment definition

Ported extrae intends to keep benefiting from all the features of the HPC mainstream version, however ports realized during this study present some limitations due to their alpha version and bounded effort. IT is however important to highlight that main features such as manual, automatic instrumentation as well as hardware counter automatic gathering are fully functional.

One of the main change is related to automatic instrumentation. Originally based on the LD_PRELOAD dynamic calls interception at runtime, it had to be adapted for embedded targets using symbol wrapping at compile time using linker flags, as illustrated in **Error! Reference source not found.**.

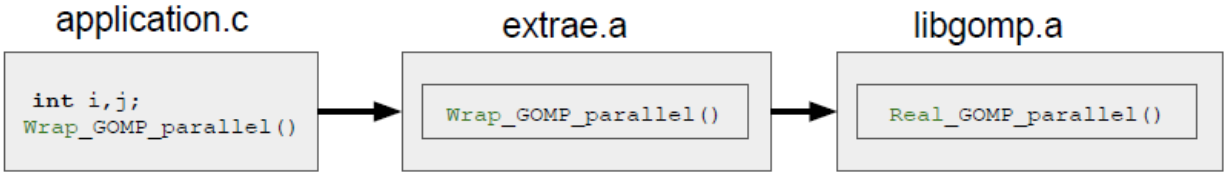


Table 2 describes the (super-set of) methods that have to be wrapped based on the OpenMP constructs used when using the sparc-gaisler-rtems5-gcc compiler version 7.2. This relation has been obtained by compiling the source code with the mentioned compiler, and then analysing the generated binary.

Table 2 - Available Methods that can be intercepted through wrapping on GR740

OpenMP directive	OpenMP clauses	Wrapped libgomp symbols (*–wrap*)
parallel	-	GOMP_parallel GOMP_barrier
single	-	GOMP_single
task	-	GOMP_task
for	-	-
	schedule(static)	-
	schedule(dynamic)	GOMP_loop_dynamic_start GOMP_loop_dynamic_next GOMP_loop_end
	schedule(guided)	GOMP_loop_guided_start GOMP_loop_guided_next GOMP_loop_end
	schedule(runtime)	GOMP_loop_runtime_start GOMP_loop_runtime_next GOMP_loop_end
	schedule(static) ordered	GOMP_loop_ordered_static_start GOMP_loop_ordered_static_next GOMP_loop_end
	schedule(dynamic) ordered	GOMP_loop_ordered_dynamic_start GOMP_loop_ordered_dynamic_next GOMP_loop_end
	schedule(guided) ordered	GOMP_loop_ordered_guided_start GOMP_loop_ordered_guided_next GOMP_loop_end
	schedule(runtime) ordered	GOMP_loop_ordered_runtime_start GOMP_loop_ordered_runtime_next GOMP_loop_end
parallel for	-	-
	schedule(static)	-
	schedule(dynamic)	GOMP_parallel_loop_dynamic GOMP_loop_end_nowait
	schedule(guided)	GOMP_parallel_loop_guided GOMP_loop_end_nowait
	schedule(runtime)	GOMP_parallel_loop_runtime GOMP_loop_end_nowait

Another modification lies in the hardware counter gathering process originally based on PAPI library. While PAPI library is natively available on Kalray Coolidge MPPA, hardware counters function was enhanced to also benefit from GR740 L4STAT counters, through L4STAT driver.

4.3 Ported OpenMP framework

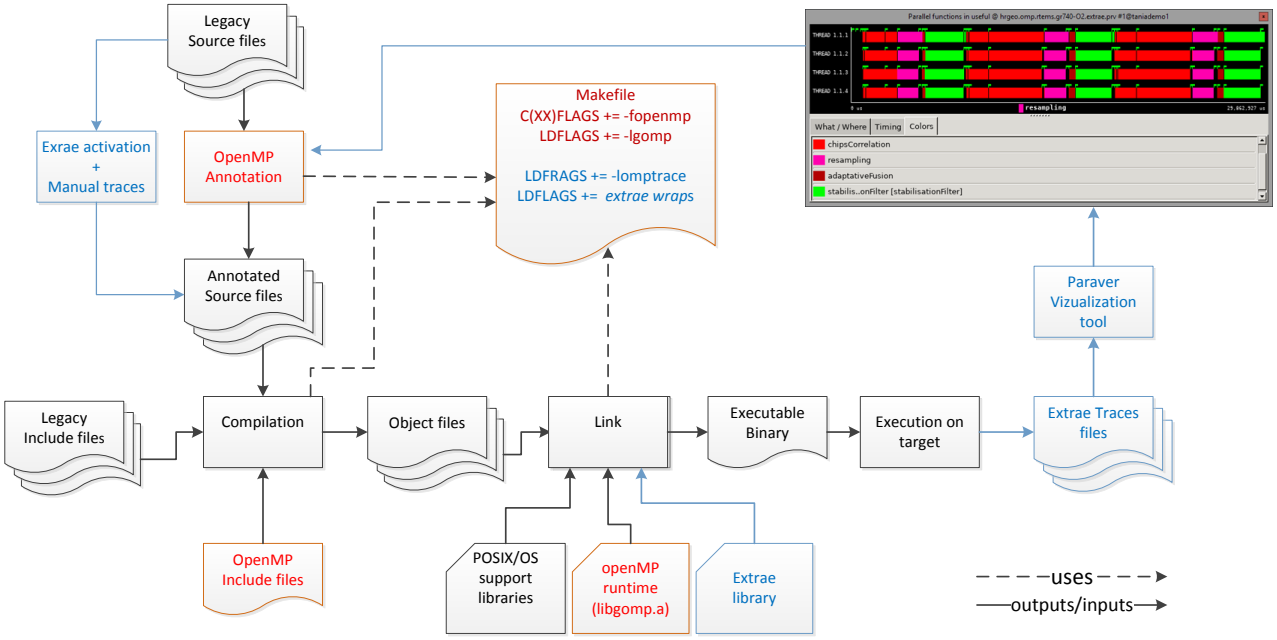


Figure 8; OpenMP/Extrae Framework workflow overview

As illustrated in Figure 8, on the hypothesis that an POSIX operating system is already in place for the sequential version, the current application building process is only lightly impacted in order to introduce OpenMP and Extrae support.

The main changes are indeed limited to:

- OpenMP parallelization (in red):
  - Annotation of the source code through #omp pragmas
  - Introduction `-fopenmp` flag at compilation time and import of omp related include files
  - Introduction of the runtime library through `-libgomp` at link time
- Extrae instrumentation (in blue):
  - Source code modification to initialize tracing and potentially add manual tracing events
  - Introduction of the extrae library through `-libomptrace` linker flag

With Extrae instrumentation enabled execution on target allows to retrieve OpenMP runtime useful information such as thread dispatches and overall load balancing, which can be naturally be used to tune and iteratively improve the omp annotations or explore other parallelization parameters. Hardware counters can as well be used to detect hardware resource usage bottlenecks or interferences that eventually might be mitigated by algorithm implementation tuning, and can be applied as well on the omp with one single thread mimicking sequential behaviour.

4.4 Final Use Case parallelization strategy

For both HRGEO and MIRROR application use cases results presented in this document in section **Error! Reference source not found.** and **Error! Reference source not found.** comply with the following rules:

- Data parallelism model is applied
- OMP ***schedule*** is set to ***static*** for determinism, with unspecified chunk-size.
- ***collapse*** clauses are used as fine grain optimization for load balancing
- OMP parallel loops are declared ***ordered*** so as to allow Extrae observability, and force instrumented runtime calls. This is not expected to induce any major overhead as the parallel sections do not contain any actual ordered statements. Without the ***ordered*** pragma, gcc might bypass gomp library and Extrae wrapping functions and limit the observability.
- Functional legacy C code remained untouched, and parallelization of actual algorithmic parts only consisted in ***#pragma omp*** annotations. No modifications of Legacy code were applied to improve performance.
- Code discrepancies between sequential and parallelized versions are limited to:
    - wrapper functions for I/Os
    - adaptation for OS integration and configuration
    - OpenMP runtime environment initialization
    - Extrae initialization and manual instrumentation

An actual example of such OpenMP annotations for one of the use case parallelized loop is given below.

```
97  #pragma omp parallel firstprivate(refImageLowL, refImageHighL, refImageLowC, refImageHighC, curImageLowL, curImageHighL,
    curImageLowC, curImageHighC, search_h, search_w, ssd_w)
98  {
99  #pragma omp for ordered shedule(static) collapse(2)
100   for(int32_t ligne = refImageLowL; ligne < refImageHighL; ligne ++)
101   {
102     for(int32_t colonne = refImageLowC; colonne < refImageHighC; colonne ++)
103     {
104       T[(ligne - refImageLowL) * IMG_WIDTH + colonne - refImageLowC] = refImage[ligne * IMG_WIDTH + colonne].R + refImage[ligne *
          IMG_WIDTH + colonne].G + refImage[ligne * IMG_WIDTH + colonne].B;
105     }
106   }
```

Figure 9: Common annotation applied to all data parallelized loops

4.4.1 HRGEO application

Profiling of the sequential version of HRGEO determined that 99% of computation time was spent in the function highlighted in blue in Figure 10, and then parallelized with OpenMP.

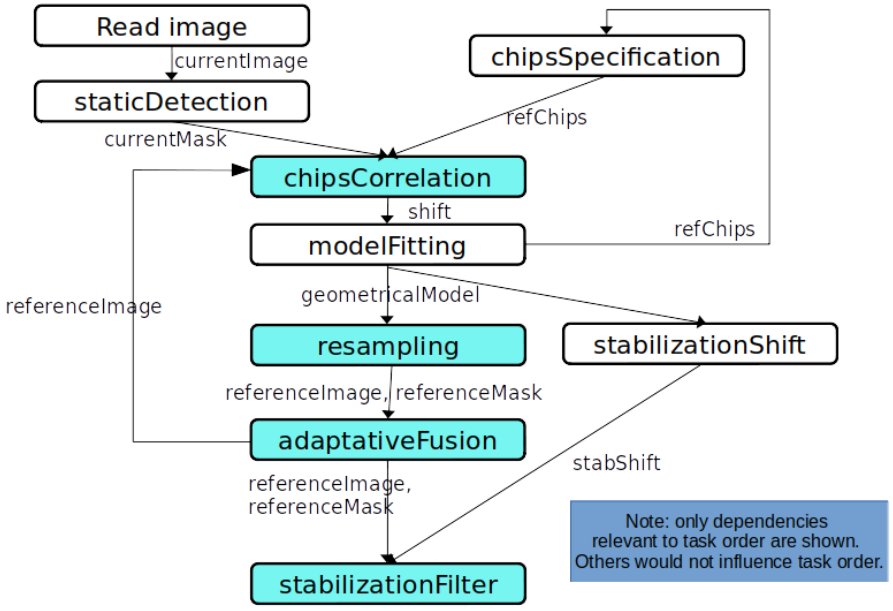


Figure 10: HRGEO algorithm phases executing in parallellel

4.4.2 Mirror application

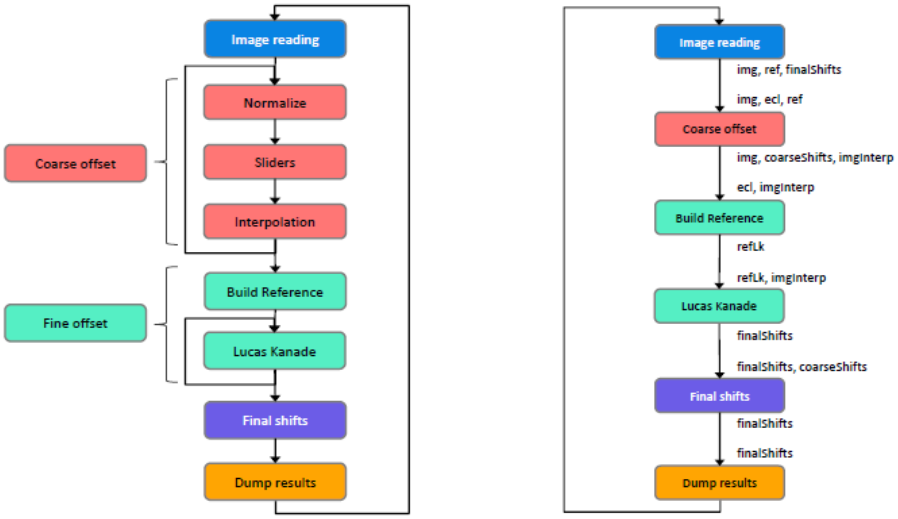


Figure 11: Mirror applicaton overview

As illustrated in the Mirror application consists of a main loop that, after a pre-loop initialization, iterates over a series of images. For each image, it performs four successive erent steps, as depicted in Figure 1a, that proceed as follows:

- The first step, *Image reading*, loads an image as a matrix of *NB_LENS*NB_LENS*SIZE_X*SIZE_Y* elements (where *NB_LENS = 12* and *SIZE_X = SIZE_Y= 34*).

- The second step, *Coarse offset*, iterates over the two most signicant dimensions of the matrix, (*NB_LENS * NB LENS*). Each iteration executes three actions: `normalize`, `sliders`, and `interpolation`. These steps end up traversing the two less significant dimensions of each element, SIZE X _ SIZE Y).

- The third step, `Fine offset`, performs two actions: `buildReference` and `lucasKanade`. These phases work on a different matrix of *NB_LENS * NB_LENS * SIZE_INTER_X * SIZE_INTER_Y* elements. While the first action is just called once, the second one iterates over the two most significant dimensions, calling the actual kernel for each element. In both cases, all elements of the matrix are visited *NB_LENS * NB_LENS * SIZE X * SIZE Y*.

- The fourth step, `Final shifts`, generates the final results.

- The fifth step, `Dump results`, dumps the results in a file.

Evaluation focused on the evaluation of parallelization of second step `Coarse Offset` and `lucasKanade` subpart of `Fine offset` step, totalizing 90 % of the sequential execution time.

An infructuous attempt not detailed in this report of parallelizing the `buidRef` offset led to the conclusion of either an non acceptable overhead induced by openMP runtime vs small footprint of the function. Reducing the overhead would require modification of the code so as not to jeopardize computanioanl correctness and such a modification was not applied as we wanted to measure the naïve potential gain without modifying the legacy code.

5 EVALUATION PHASE

5.1 Test Setups

5.1.1 GR740

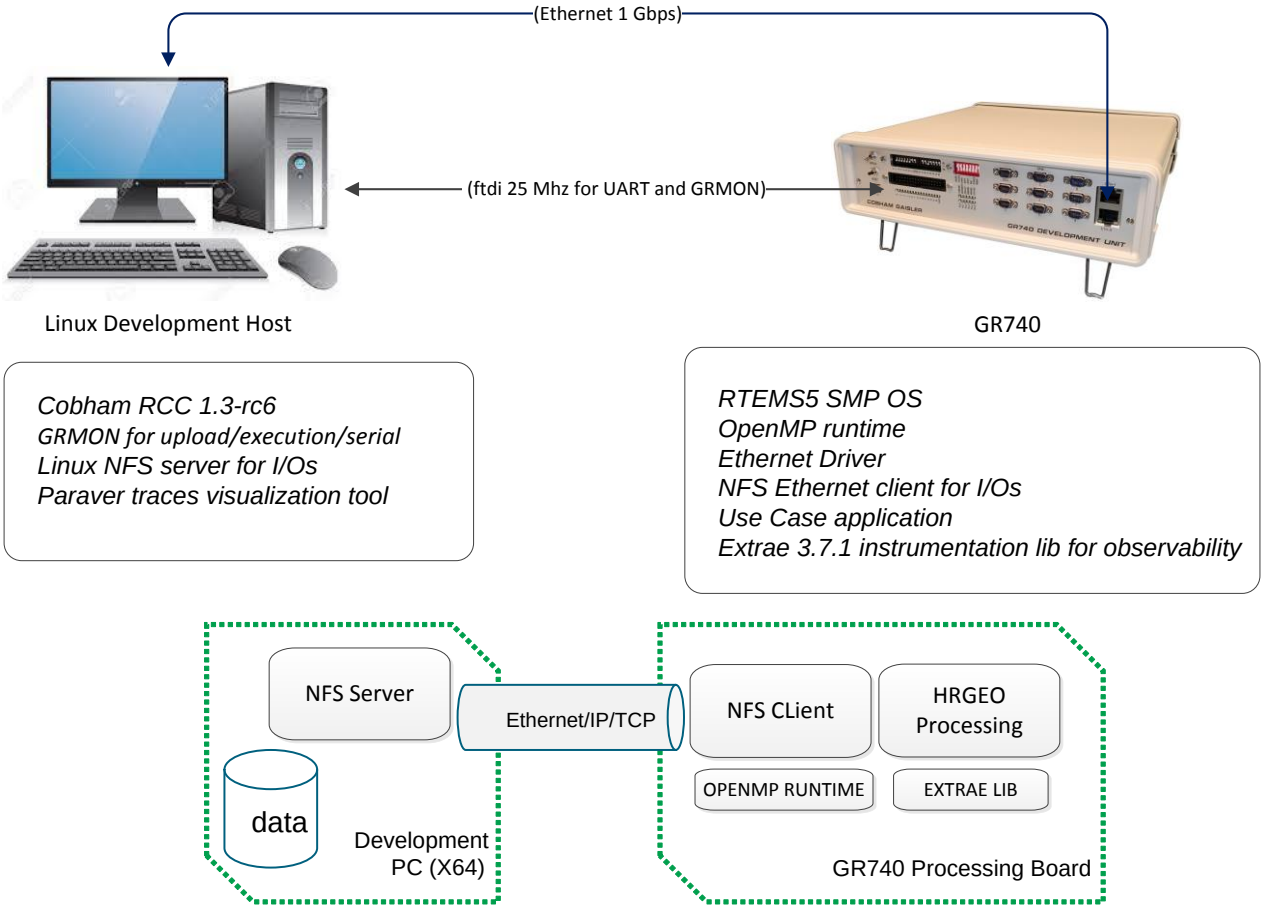


Figure 12: GR740 test setup

As illustrated in Figure 12, the test setup is mainly composed of a GR-CPCI-GR740 Quad-Core LEON4FT Development Board, connected to a Linux host development machine.

I/Os consist of input data file either directly linked with the application or retrieved from development host through NFS over a 1Gbps Ethernet link. This NFS share is also used to retrieve both functional and instrumentation outputs.

The software stack on the development host is composed of :

- RCC 1.3-rc6 : cross compiler for LEON4 target with RTEMS 5 OS, based on gcc 7.2.0
- grmon Pro 2.0.98 : used for executable load and serial output
- nfs-kernel-service : basic nfs client to support a remote file system

- BSC Paraver trace visualization tool for Linux, version 4.8.1.

The software stack on the embedded platform is composed of :

- RTEMS 5 SMP OS with POSIX support, with Ethernet and NFS support
- Actual Use Case algorithm linked
- OpenMP 4.5 runtime
- Extrae 3.7.1 instrumentation library from BSC ported to GR740
-

5.1.2 KALRAY TEST SETUP

The Kalray test setup is based on the MPPA® DEV4 workstation which is a complete X86 – MPPA® based environment, packaged with Kalray S/W, for easy benchmarking and development of accelerated systems.

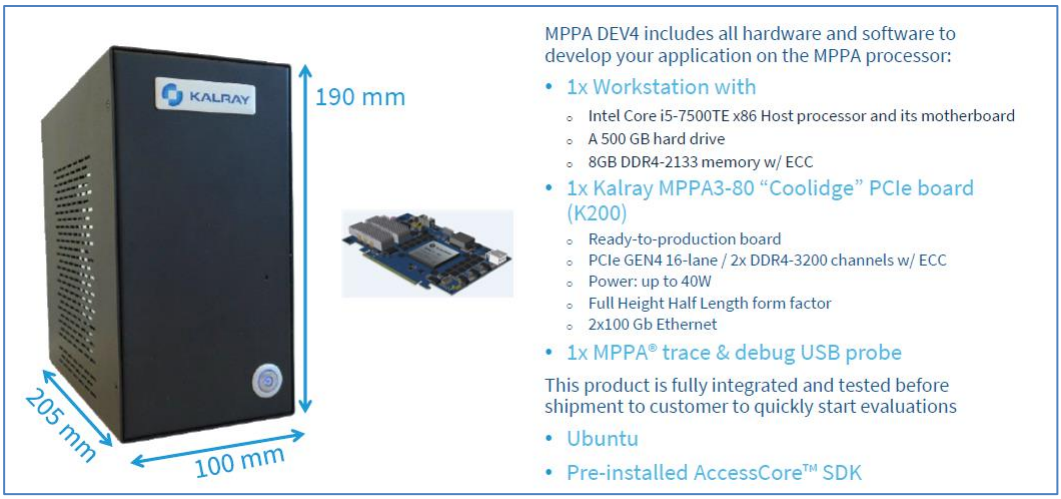
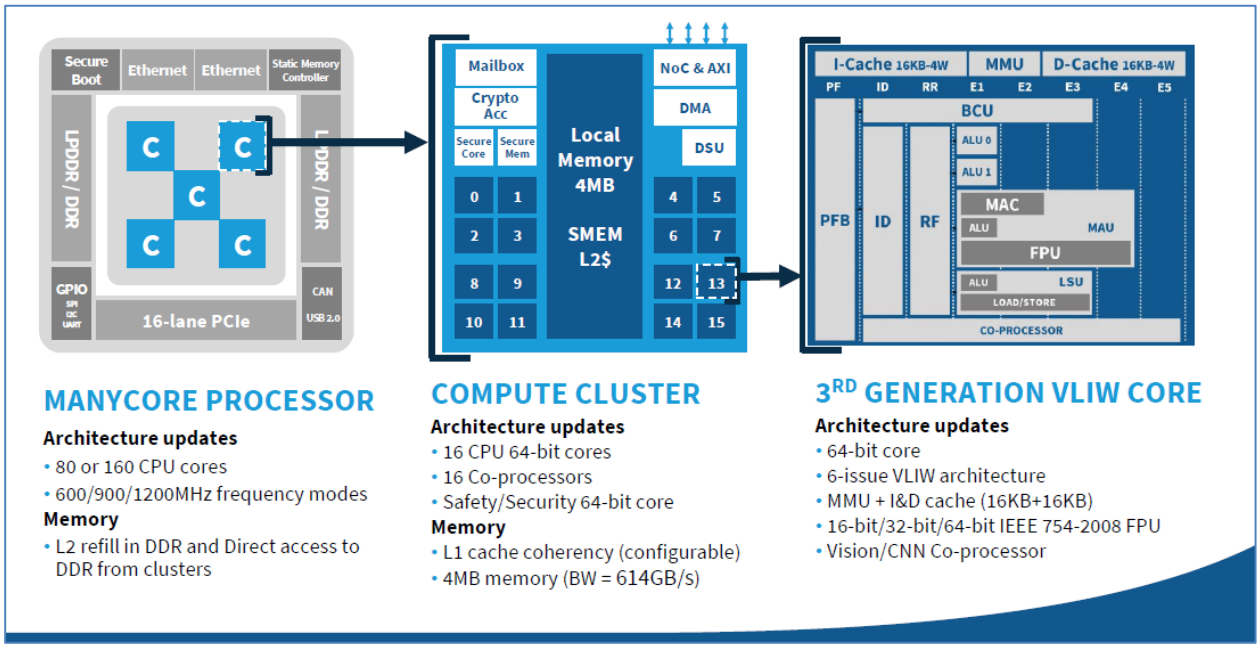


Figure 13: MPPA® DEV 4

This development is composed of a x86_64 host part running a standard Linux distribution and a KONIC200 PCIe Programmable accelerator card, hosting the Coolidge MPPA device.



Two modes of operation were exploited during the study. The first one is the JTAG mode on single cluster illustrated in Figure 14. It consists on the generation of a standalone executable intended to be run and parallelized on a single cluster on top of the Kalray Cluster OS. Deployment of the executable on target relies a JTAG link with PCIe acceleration for executable or binary blobs transfer to/from MPPA® Coolidge target. The JTAG mode also offer semi-hosting capabilities allowing to access host file system through regular file manipulation calls, with inherent

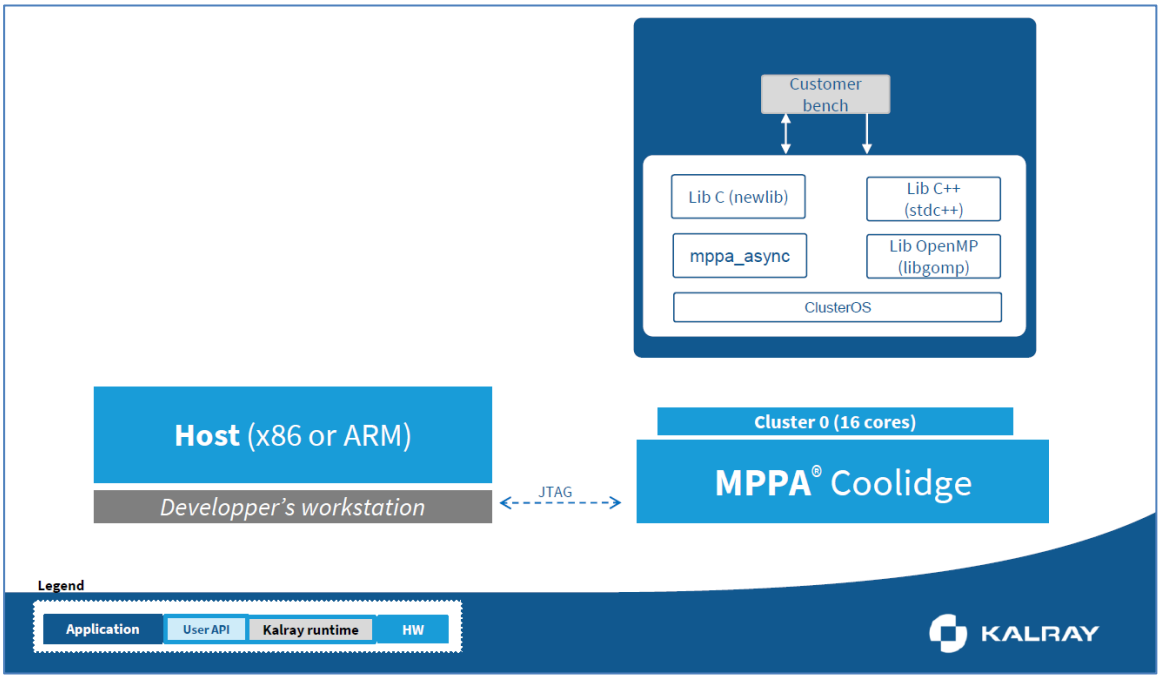


Figure 14: Using JTAG on one Cluster Only

Second mode of operation explored is OpenCL offloading where the Coolidge manycore is considered as an accelerator. The kernels that will run in the accelerator must be compiled in isolation with the OpenCL compiler, and then linked with the host application part. All the communication is done through PCIe.

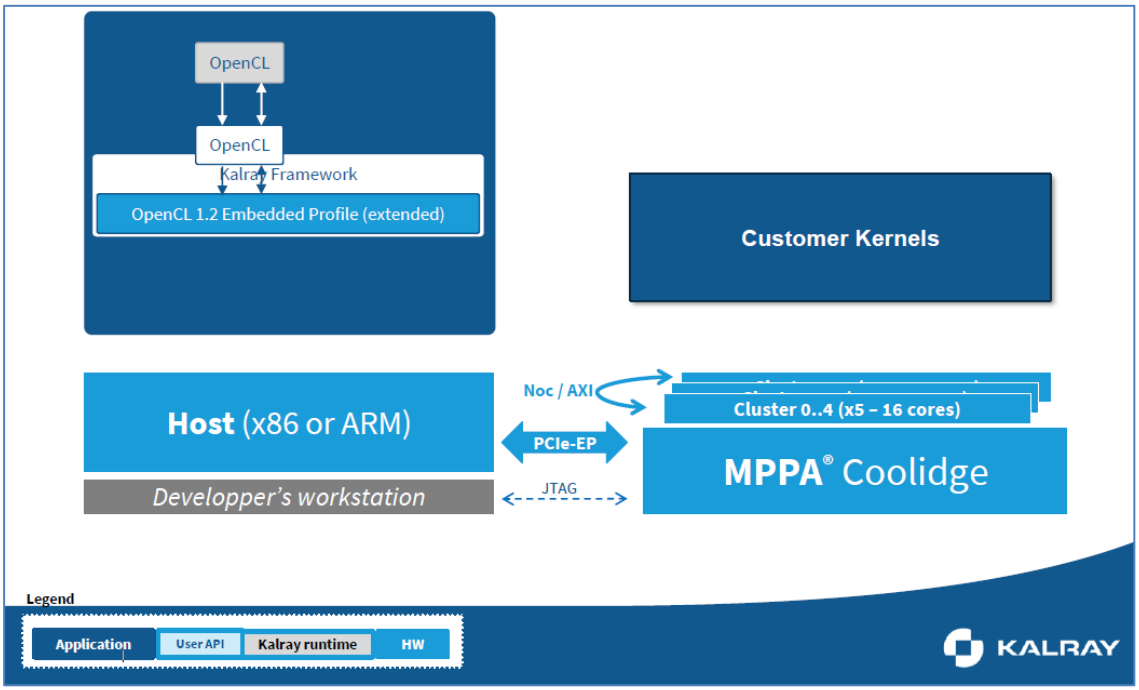


Figure 15: OpenCL offloading using MPPA® as an accelerator

The software stack on the host platform is composed of Kalray SDK ACE 4.1.0, based on gcc 7.5.0, and same x86 4.8.1 Paraver version than the one used for GR740.

On the embedded part the stack is composed of ClusterOS, which is a light OS optimized for MPPA® cluster, with openMP 4.5 support and multithreading support through pThread POSIX, as well as a port of Extrae 3.7.1 to the Coolidge architecture.

As far as application mapping is concerned we selected amongst the authorized configurations, the one maximizing size of L2 cache:

- The local cluster SMEM is split in 2 parts: a 2MB TCM plus a 2MB level-2 cache
- ClusterOS and its execution kernel and user stack are stored in SMEM
- User code and associated data sections, including openMP and instrumentation libraries are stored in global DDR external to the Cluster

This is mapping would be more likely to be optimized to improve performance, but was deemed sufficient to perform the openMP deployment and performance measurements relative to equivalent sequential version with similar memory mapping.

Additionally it allowed to have a first evaluation of the performance of the cluster programmed as a generic GPPU, which was not so trivial with previous generation of the MPPA®.

5.2 Test Scenario

Test scenario for both HRGEO and MIRROR use cases execution on target follows an incremental approach.

First the algorithm is executed in its pure sequential form to obtain a reference baseline.

Then algorithm is executed with OpenMP runtime integrated but only 1 core mainly to measure open MP overhead in terms executable size overhead **Error! Reference source not found.**and execution time overhead.

Then the number of exploited cores are incrementally increased to measure actual multicore performance gain thanks to parallelization.
The selected multicore scheme are:

- 1,2,4 cores for GR740 target
- 1,4,8,16 cores for Kalray Coolidge target

Finally OpenMP Extrae is integrated on th scenario with the maximum number of cores active to check the observability mechanism and verify proper and efficient parallelization of the code.

For each of those tests functional correctness of parallelized code is verified by comparing outputs against expected ones for the predefined set of inputs.

For each use case the procedure is

1. Provide reference inputs
2. Execute algorithm
3. Retrieve Performance measurements
4. Retrieve algorithm output
5. Retrieve Traces and Generate

5.3 Quantitative metrics

As set of measurable metrics were captured during the evaluation:

1) OpenMP and Instrumentation executable size overhead
2) Parallelized version speed up, based on local figure of merit and overall speed up vs Amdahl's law
3) Functional Correctness

5.3.1 Executable Sizes overhead

Section present the parallel executable sizes ratio against sequential version, in terms of code and data sections overhead.

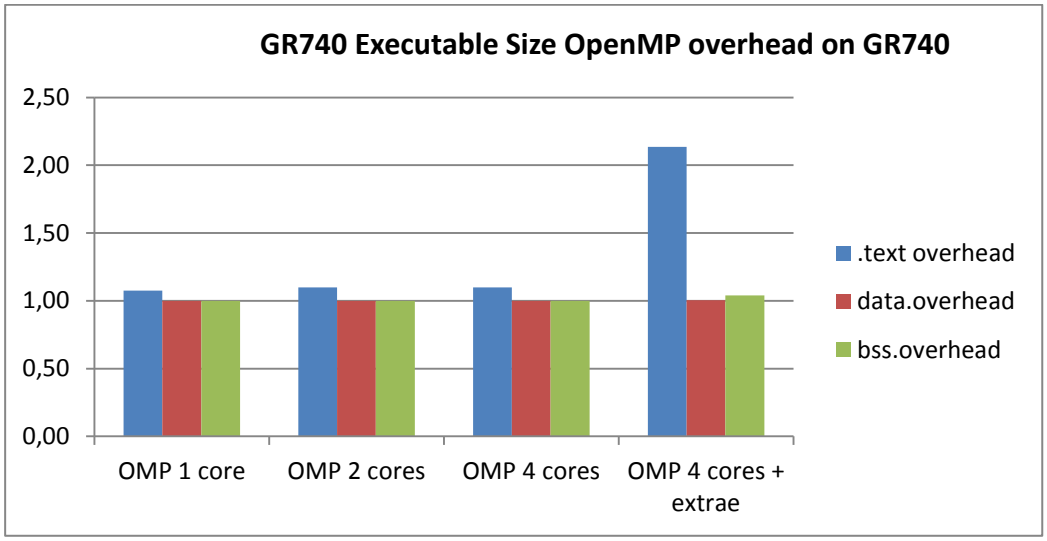


Figure 16: HRGEO executable size overhead on GR740

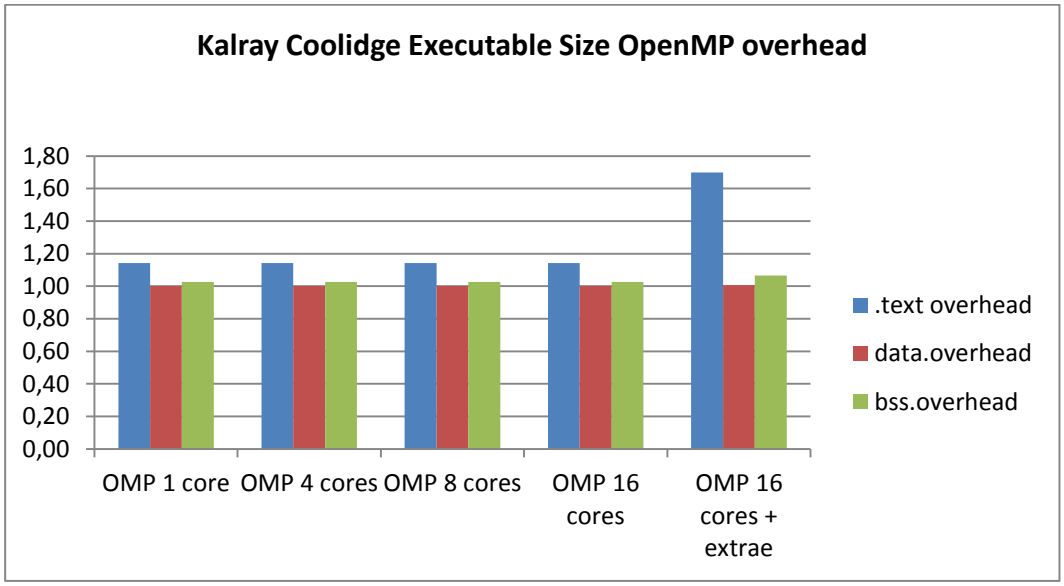


Figure 17: HRGEO executable size overhead on KALRAY

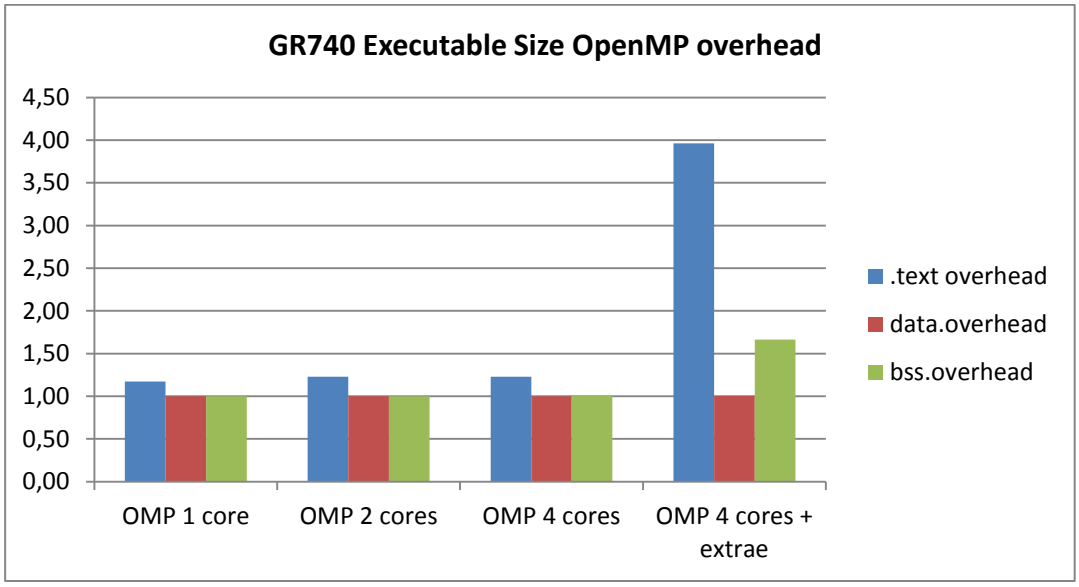


Figure 18: MIRROR executable size overhead on GR740

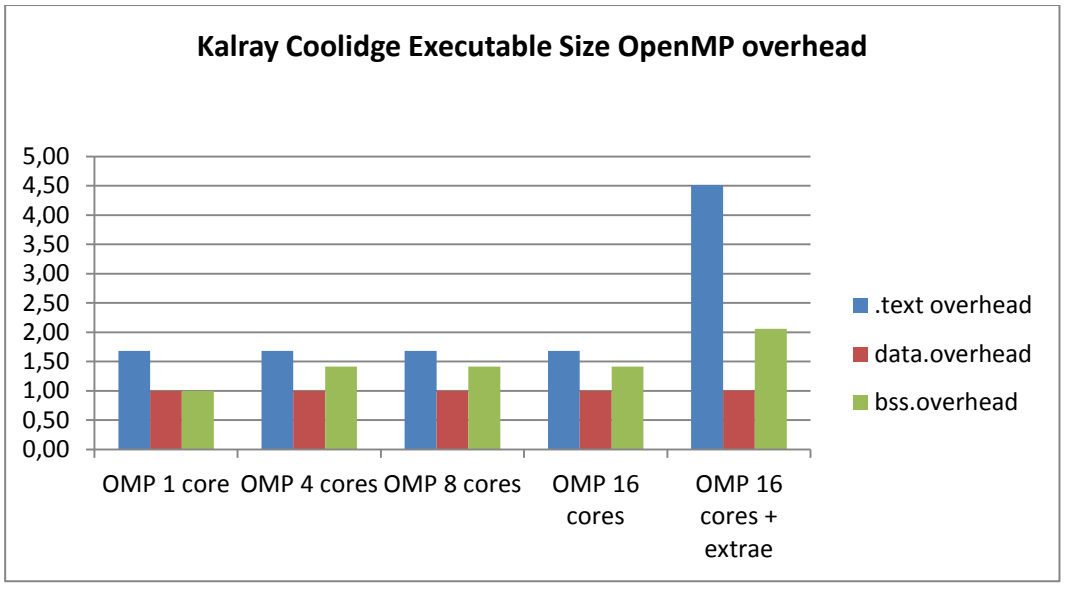
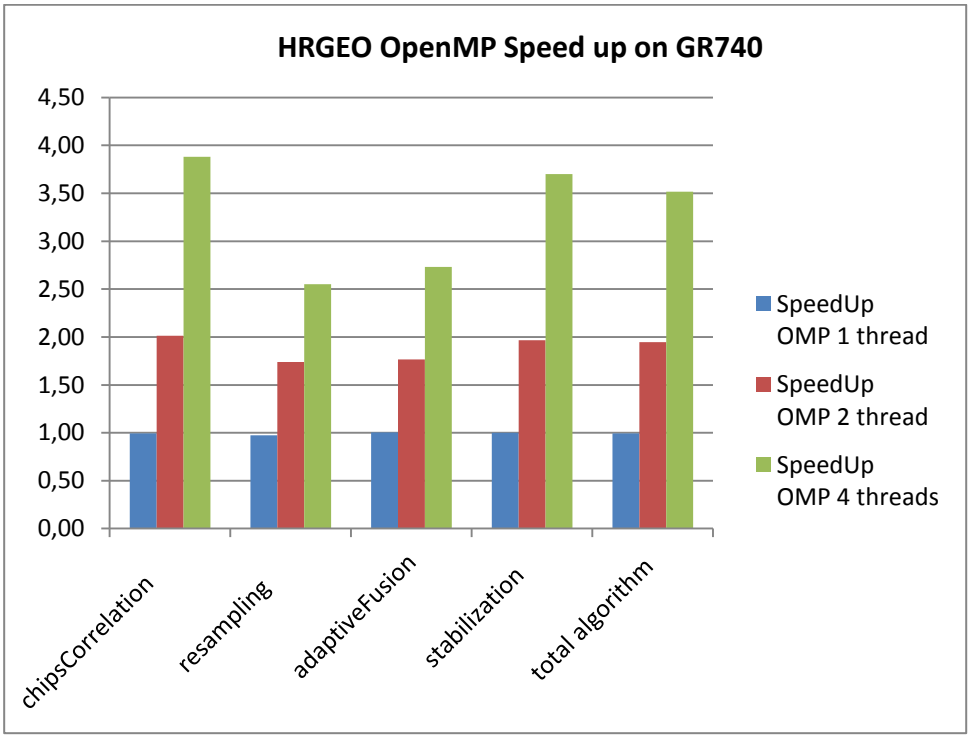


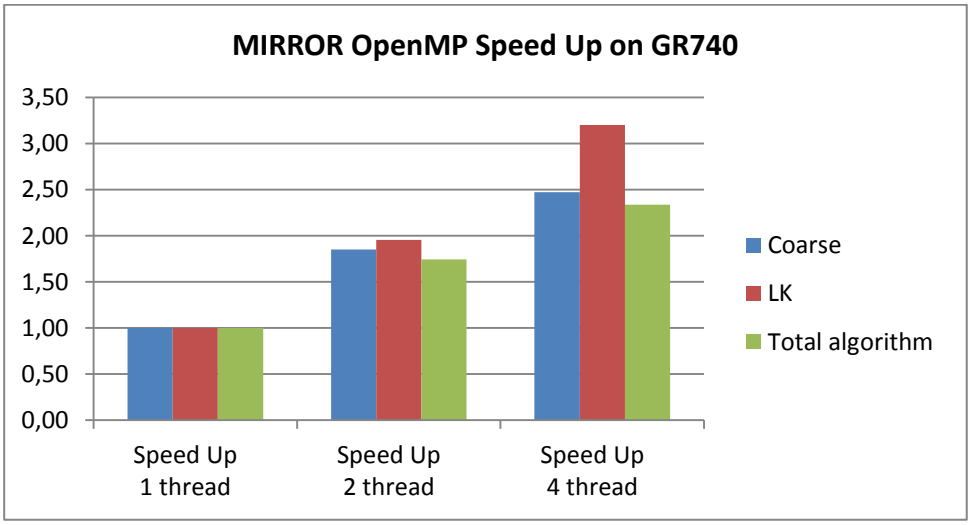
Figure 19: MIRROR executable size overhead on KALRAY

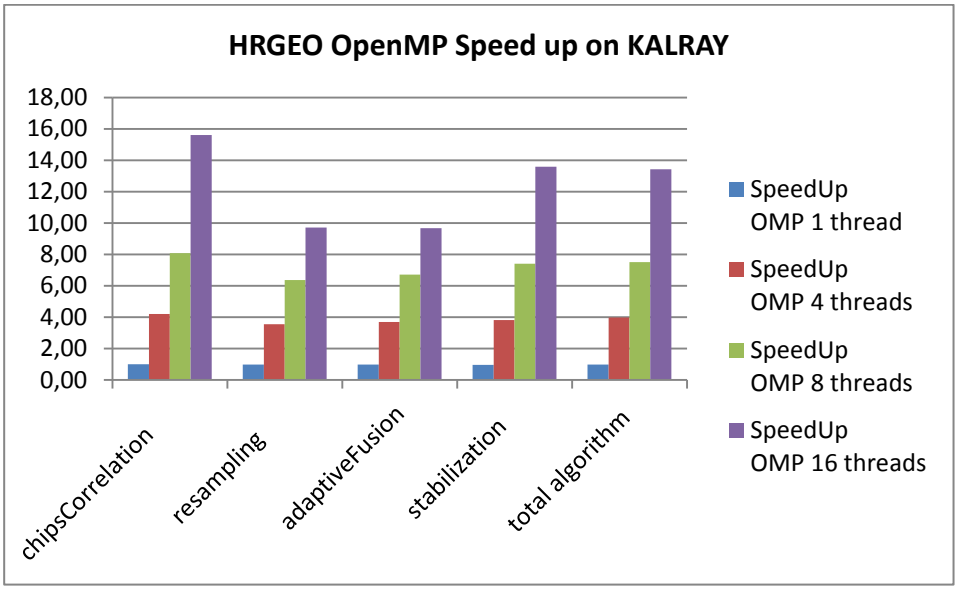
For all use cases and use cases the overhead of both runtime and instrumentation library is compatible with actual amount of memory available on embedded targets, with an absolute overhead of ~100KiB of rthe openMP runtime and ~1MIB for the Extrae library.

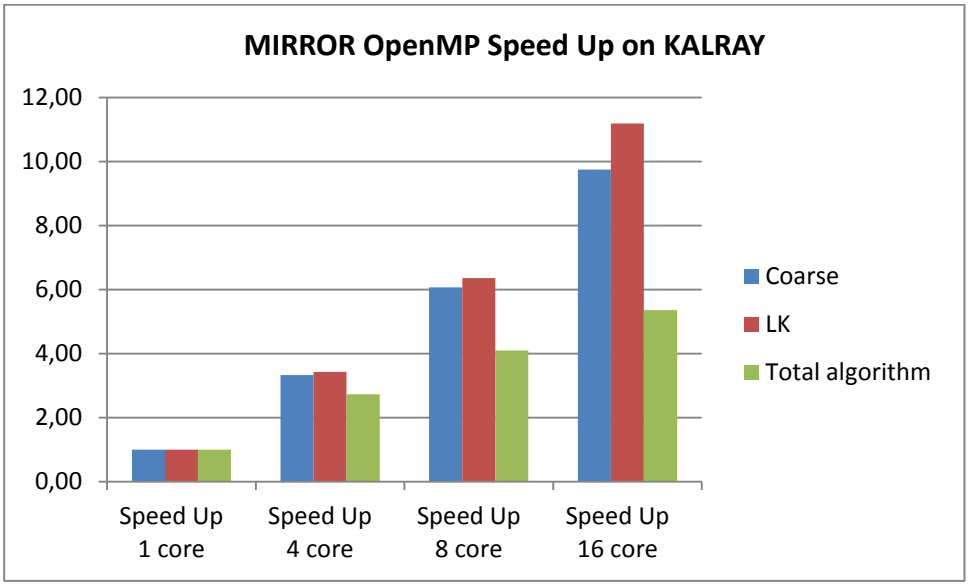
5.3.2 Speed Up

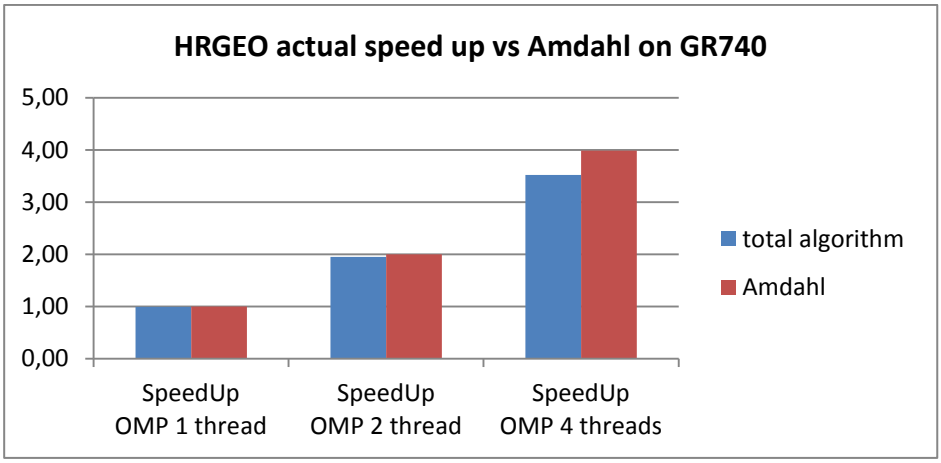
This section presents the measure figure of merit for each individual parallelized loop as well as the overall total algorithm time speed up, including remaining sequential phases, to be compared to theoretical execution time obtained with Amdahl's law.

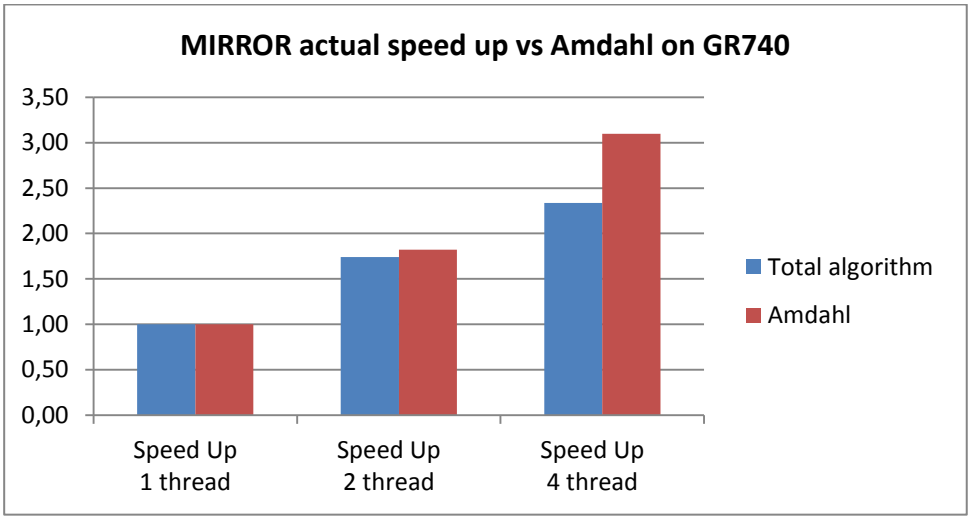


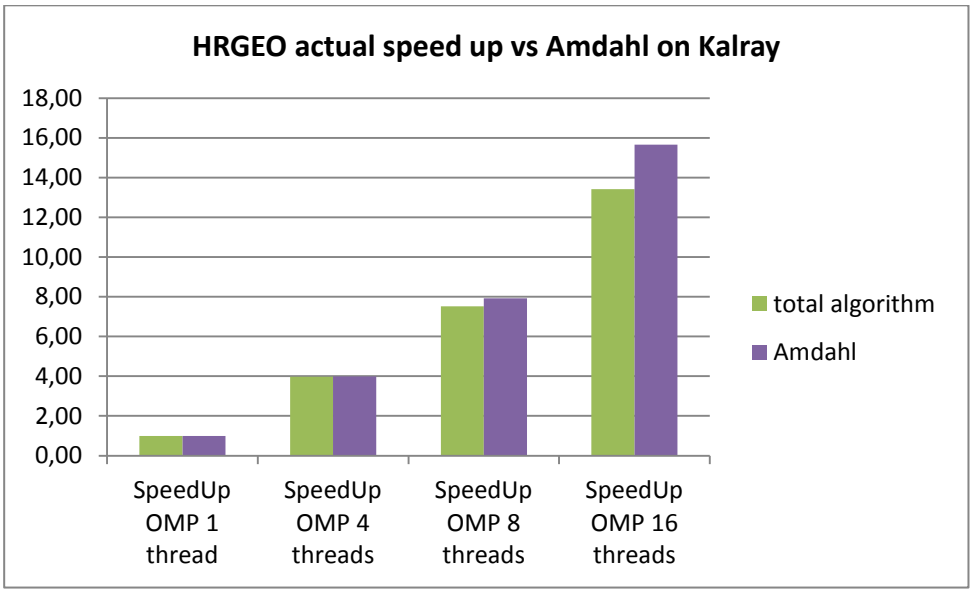


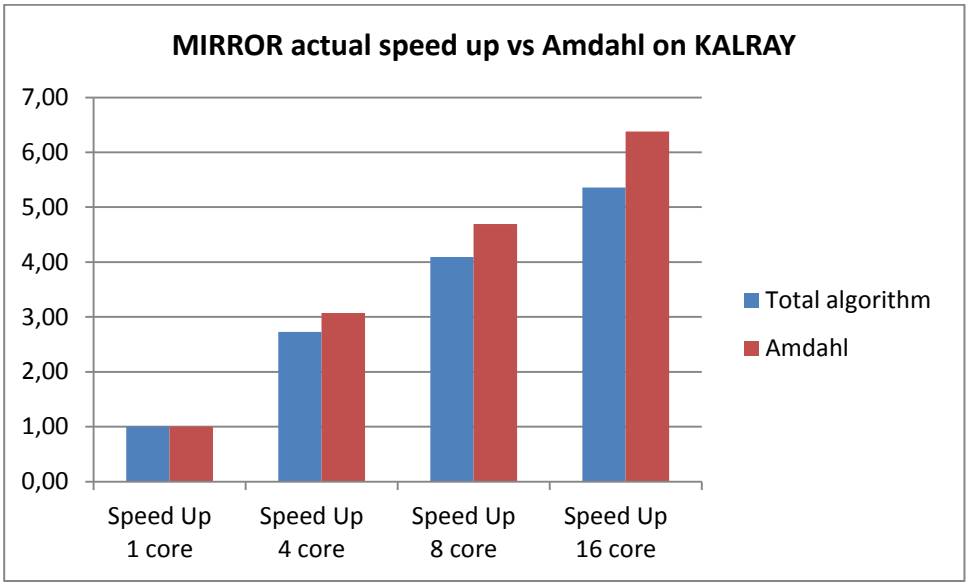












5.3.3 Functional Correctness

All tests presented correct outputs, checked based on md5sum signature.

5.4 Qualitative metrics on Extrae exploitation

During the project we evaluated the benefit of Extrae instrumentation and Paraver visualization to provide better visibility and understanding of parallel execution of the algorithm on actual target, providing crucial pieces of information in the efficient usage of the multicore resources.

Two examples regarding load balancing and hardware counters exploitation on HRGEO use case on GR740 targets are presented hereafter.

5.4.1 Load Balancing and scheduling

During HRGEO initial parallelization no fine tuning was used and static scheduling was applied, resulting in unbalanced execution illustrated below where fourth core starves.

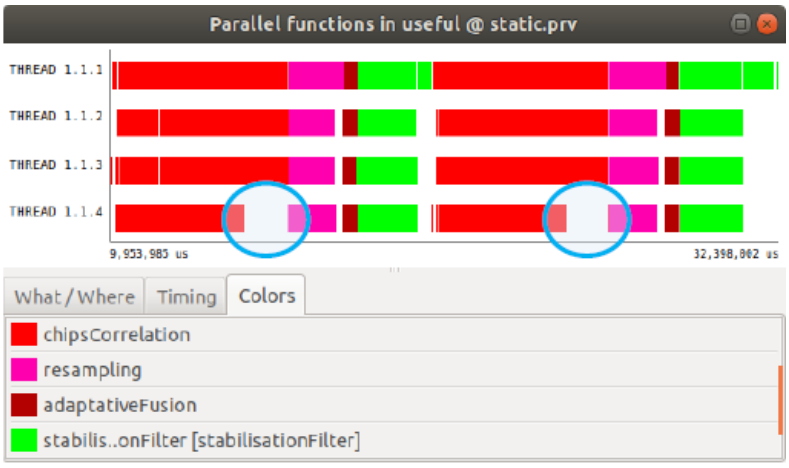


Figure 20: Unbalanced workload with static scheduling

Zooming and activating event flags visualization allow to visualize the activity and highlight the four parallelized loops composing the `chipCorrelation` function, spotting the fourth loop and highlighting additional unbalance at the early stage of the processing.

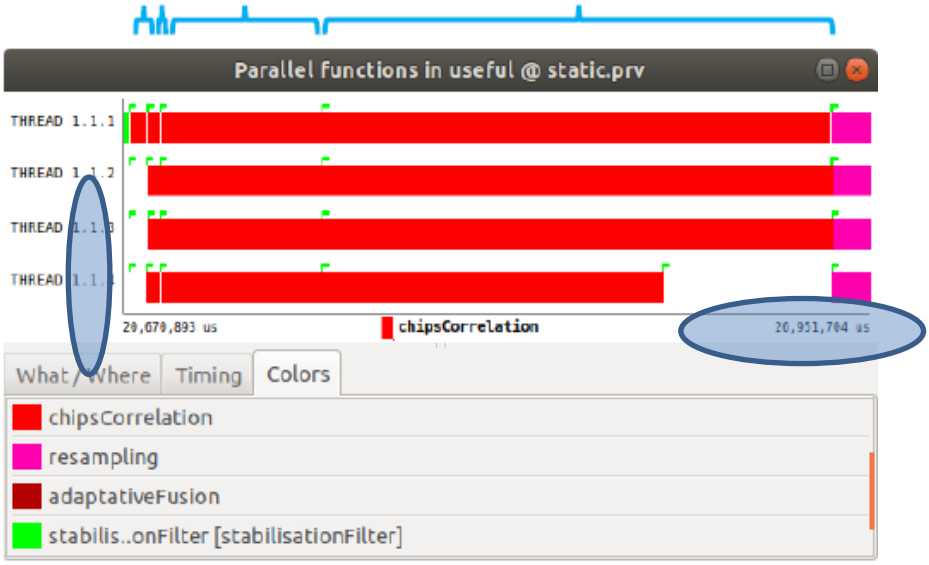


Figure 21: Detailed view of the unbalanced execution

Analysis of the last loop code shows that it traverses a really small iteration space and static partitioning cannot divide the space perfectly into 4 threads, so the last one gets less workload. The processing consists of two nested loops and only outer one is actually parallel. As the loops are perfectly nested they can be collapsed so as to increase the iteration space and better balance the chunks split.

Figure 22: Load balancing thanks to the collapse clause

Finally as some threads do not get enough work in the first loop of chipsCorrelation, a switch to guided schedule could help by forcing a mode dynamic scheduling with big chunks at the beginning then small at the end.

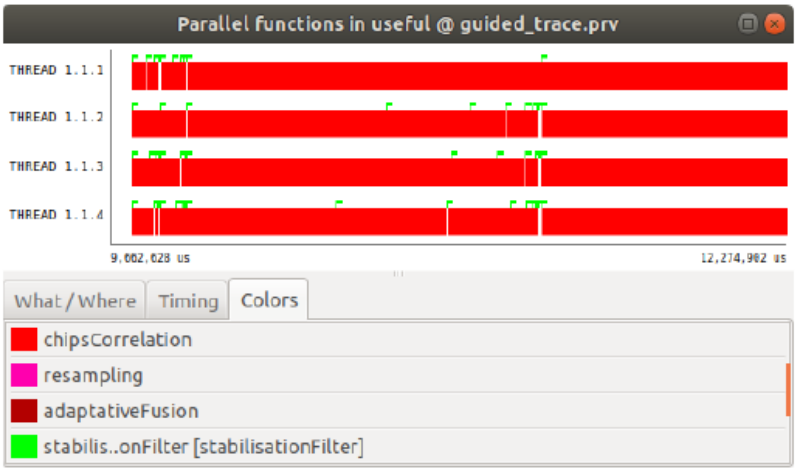


Figure 23: Final optimal parallelization of legacy code

Initial omp annotation: `#pragma parallel for schedule(static)`
Final omp annotation: `#pragma parallel for schedule(guided) collapse(2)`

5.4.2 Hardware Counters

Hardware counters are very useful to measure actual performance, and potential underperformance of the executed code. Extrae instrumentation can be used to retrieve such counters automatically alongside OpenMP runtime calls or periodically through configurable sampling.

Figure 24 gives an example of (IPC) Instruction Per Cycle metric automatically captured for the HRGEO use case.

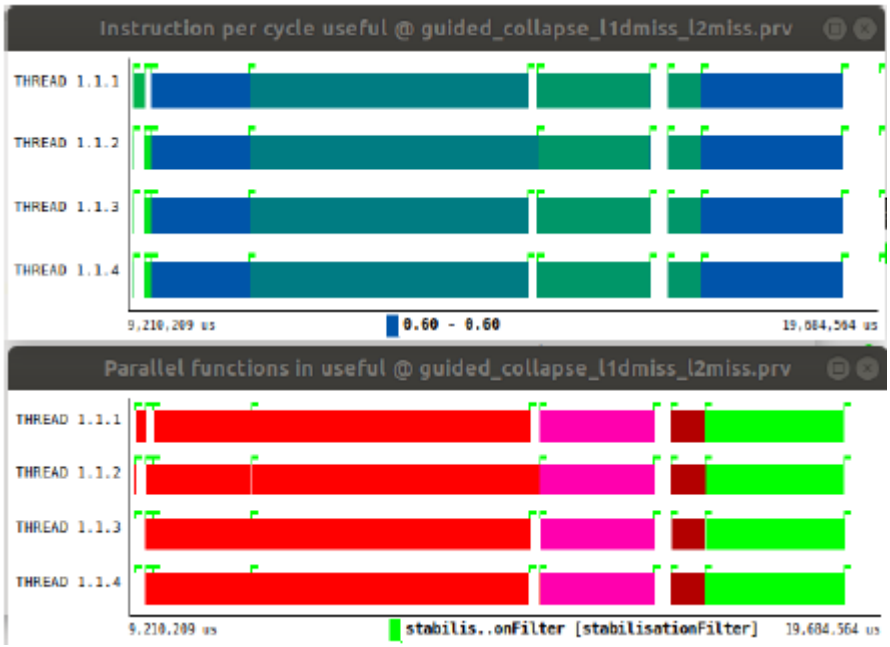


Figure 24: HRGEO use case IPC (light green=low IPC, dark blue=high IPC)

Here we can see that *resampling* (pink) and *adaptiveFusion* (dark red) present poor IPC metrics. We can use additional counters to try to explain the poor IPC, as an example by retrieving cache misses counters. This is illustrated in Figure 25, which highlights from top to bottom, L1 data cache miss ratio, L2 cache miss ratio and IPC, with color code being the higher the darker.

Figure 25: Correlating IPC with L1 and L2 cache misses

The first low IPC region can be explained by high L1D cache miss ration (~50) and descent L2 miss ratio (~5) , while the second can be explain by high L2 cache miss ratio (~20). This conveniently give some precious hints to try to track area of improvement in the algorithm detailed implementation.

6 SUCCES CRITERIA

Success Criteria	GR740	KALRAY
[Criteria#1] OpenMP runtime execution time overhead	PASSED	PASSED
[Criteria#2] OpenMP runtime code / data size overhead	PASSED	PASSED
[Criteria#3] Extrae library timing overhead	PASSED	PASSED
[Criteria#4] Extrae timing measurement accuracy	PASSED	PASSED
[Criteria#5] Multicore achieved speed up	PASSED	PASSED
[Criteria#6] Multicore efficient exploitation, explored with Extrae instrumentation	PASSED	PASSED
[Criteria#7] Functional Correctness of parallelized code	PASSED	PASSED
[Criteria#8] Initial Instrumentation Effort	PASSED	PASSED
[Criteria#9] Porting effort when switching to a new target	PASSED	PASSED
[Criteria#10] Observability through instrumentation	PASSED	PASSED

	Rationale
Criteria#1	Execution time overhead is absolutely acceptable in the order of magnitude of a few percents on single core execution, and only one blocking point was found on Mirror use case when trying to parallelize one of the remaining sequential execution contributors
Criteria#2	Executable size overhead is in the order of 5% for the considered use case which are not that big. Absolute overhead size is in the order of magnitude of a hundred of kbytes
Criteria#3	Extrae execution time overhead is very low
Criteria#4	Extrae measurement is as accurate as the available Performance Monitor or core timers can be. So as precise as what would be added through manual source instrumentation, with the advantage of not modifying the source code for those openMP runtime calls that can be automatically trapped thanks to wrapping
Criteria#5	Achieved speed up is very good, following Amdahl's considering the amount proportion of the algorithm actually parallelized. Instrumentation provides a convenient way to retrieve performance counters, and internal insights, in order to help profiling and support investigation of the areas with limited speedup
Criteria#6	Load balancing amongst core is quite easy to monitor thanks to Extrae instrumentation, allowing to tune parallelization annotations appropriately
Criteria#7	No errors were detected during tests: expected functional behavior with parallelized version of the algorithm was preserved.
Criteria#8	Initial parallelization effort was quite low, limited to a few `omp pragma` added to the code. Most of the coding effort was dedicated to wrapping of the algorithm to the target, operating system integration and configuration, and I/O management, rather than actual parallelization.
Criteria#9	Annotations or functional algorithm code was not modified from a target to another, only specific target init or I/O management functions.
Criteria#10	Extrae provides good automatic observables, user friendly way to defined additional manual events, and assistance for automatic accurate Hardware Counters retrieval.

7 CONCLUSION AND FUTURE WORK

Both selected targets SDKs were already supporting a mature openMP runtime, demonstrating the ecosystem is ready on both for radiation hardened and COTS components perspective. Porting based on a GNU GCC mainstream compiler is completely in line with current state of the art and golden rules applied in the scope of embedded software development as far as compiler choice is concerned..

The initial parallelization effort of legacy code revealed to be low, as expected, thanks to non-intrusive openMP annotations scheme. Selected annotations used to reach the decent performance improvement presented in this report were very limited in number and complexity, and selected so as to preserve determinism from one execution to another. An opposite choice could have been taken so as to introduce some more random dynamic scheduling schemes to capitalize on complexity to avoid deterministic worst cases.

The porting effort when switching from the first radiation hardened hardware target to the second COTS other proved to be inexistent as openMP annotations remained strictly unchanged. This is applicable as well to Extrae activation and manual instrumentations. The only customization consisted in openMP environment, .i.e. number of available cores.

Open source Extrae observability library and its associated Paraver visualization tool both provided by BSC and so far mainly targeting HPC mainstream world, were successfully ported to GR740 and MPPA Coolidge ManyCore. They provided all the expected features to sustain efficient profiling, verification and parallelization tuning. The tooling required a reasonable learning phase, is well documented and offers a standardized hardware agnostic interface allowing focusing on the actual data providing added value to the final end user mainly in the form of graphical intuitive load dispatch representations, timing information, and hardware counters. Such knowledge is mandatory to ensure efficient usage of multi and many cores, and support production delay shortening.

However it is important to remind that despite we demonstrated during this evaluation phase that the selected algorithms can be efficiently parallelized and observed, this is only valid for those two selected representative algorithms, and of course cannot prove applicable to any legacy code. We also found some limitation in the strategy for some of the functions where the ideal speed up cannot be reached either due to openMP overhead or due to actual hardware resource usage bottlenecks. While mitigation of the detected poor IPCs performance in well balanced parallel phase remain to be defined, we however confirmed that all possible observables would be available through the evaluated OpenMP framework to detect such corner cases and support such analysis.

As a result this evaluation comforts the idea that OpenMP could and should be seriously considered in the scope of future R&D multicore roadmap, as well as for rapid prototyping in advanced studies, especially when considering new space approaches.

As stated before HP4S defined two strategic end goals:

> ***G1.*** *Improve overall system performance. Effectively master and exploit the most advanced parallel embedded architectures targeting the space domain.*
>
> ***G2****. Improve the parallel programming productivity. Reduce the development efforts of systems based on parallel architectures, while fulfilling system's functional and non-functional (time predictability) requirements*

Those two goals derived in a set of technical measurable objectives, listed hereafter:

O1. Facilitate the development, timing analysis and execution of parallel real-time space applications using the OpenMP parallel programming model.

O2. Evaluate the interest and porting effort of a list of homogeneous and heterogeneous foreseen COTS and RadHard hardware targets in the space domain with OpenMP programming model and framework.

O3. Adapt the OpenMP runtime libraries to ensure that the timing guarantees devised at analysis time can be guaranteed at deployment time.

O4. Evaluate state-of-the-art compiler techniques to guarantee that parallel OpenMP

O5. Demonstrate the portability benefits of the OpenMP parallel programming model.

The outcomes of the previous preparatory and implementation phase, completed by the experiments results obtained during the evaluation phase and presented in the report allow us to state that **O1**, **O2**, and **O5** are achieved while **O3** and **O4** remain open for future work.

Main foreseen follow up activities are suggested hereafter:

- *Extrae/Paraver improvements and industrialization*

While *Paraver* tool could be deemed quite mature, the *Extrae* library porting resulting from the HP4S project can still be matured in terms of minor bugs or additional features but also in term of process. One of the current main show stoppers for industrial exploitation would relate to trace dumping, which could benefit from modern debug tracing features available on modern SoCs and Manycore and final *Paraver* report generation which would have to be somehow deported to the development host.

- *OpenMP Qualification strategy and development process impacts*

Parallelization with OpenMP while being quite seamless regarding source code is far more impacting when considering generated object code and final executable. While it does only marginally impacts the high level TS validation strategy it clearly jeopardizes some of the well-established steps starting with unit testing and coverage measurement Adoption of new programming models will more likely require to adapt our current processes, and rely on a combination of current process enhanced with state-of-the-art compiler and runtime correctness techniques so as to demonstrate functional safety of the parallelized software and absence a data races (i.e. the definition of data-sharing attributes and synchronization mechanisms) or deadlocks (i.e. locking routines and synchronization mechanisms).

- *OpenMP Time Predictability and WCET estimation for parallelized software*

Obtained results with static scheduling proved to be quite stable and deterministic, however there is a necessity to ensure that decisions taken at run-time maintain the guarantees of system correctness endorsed during design and implementation. Hence, a parallel framework (considering both compiler and runtime) targeting a critical real-time embedded system must ensure that processor resource allocation, either static or dynamic, maintains the response time analysis performed at analysis time. Additionally, such a framework must also ensure the functional correctness of the parallel execution safeguarding it from data races and deadlocks as mentioned in previous point.

- *OpenMP Offloading features*

Current study did not explore the openMP offloading capabilities, and only considered an homogeneous model where the thread that executes the implicit parallel region executes on the same processing device. An implementation may support other target devices. If supported, one or more devices are available to the host device for offloading code and data. Each device has its own threads that are distinct from threads that execute on another device. Threads cannot migrate from one device to another device. The execution model becomes host-centric such that the host device offloads target regions to target devices. This might be of particular interest in the scope of a standard interfacing and offloading to FPGA hardware accelerators IPs, potentially exploring benefits from partial reconfiguration, exploitation of additional clusters on MPPA Coolidge manycore, or exploitation of IA accelerators.

- *Extension of the study to ARM MPSoCs, and heterogeneous targets*
- *Exploration of usage of OpenMP on "non-friendly" target, such as ones with no hardware cache coherency assistance*

8 ACRONYMS AND ABBREVIATIONS

Specific acronyms and abbreviations used in this document are given below.

ALU	Arithmetic-Logic Unit
API	Application Programming Interface
APU	Application Processing Unit
BSC	Barcelona Supercomputing Center
COTS	Commercial Of The Shelf
CPU	Central Processing Unit
FDIR	Fault Detection, Isolation and Recovery
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
GPGPU	General Purpose Graphic Processing Unit
GPPU	General Purpose Processing Unit
GSD	Ground Sampling Distance
HPC	High Performance Computing
HW	Hardware
HW IP	HardWare Intellectual Property (a custom logic function implemented in HW)
IPC	Instructions per Cycle
MPPA®	Massively Parallel Processor Array
MPI	Message Passing Interface
MPSoC	Multi-Processor SoC
NFS	Network File System
OS	Operating System
PAPI	Performance API
PE	Processing Element
RGB	Red Green Blue
SIMD	Single Instruction Multiple Data
S/N	Signal/Noise
SoC	System On Chip
SSE	Streaming SIMD Extension
SW	SoftWare
TCM	Tightly Coupled Memory
VLIW	Very Long Instruction Word