Grant Agreement number: 288574

Project acronym: **vIrtical**

Project title: SW/HW extensions for virtualized heterogeneous multicore platforms

Seventh Framework Programme

Funding Scheme: Collaborative project

FP7 -ICT -2011-7

Objective ICT-2011.3.4 Computing Systems

Start date of project: 15/07/2011 Duration: 36 months

D 1.1 *Acceleration, memory system opportunities, and virtualization requirements in selected industrial applications*

Due date of deliverable: Month 12

Actual submission date: Month 14

Organization name of lead beneficiary and contributors for this deliverable: THALES, UPV, UNIBO, STM
Work package contributing to the Deliverable: WP1

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

APPROVED BY:

Partners	**Date**
ALL	**September 4th 2012**

Index

Abstract

Embedded devices are becoming more and more present everywhere. Moreover mobile devices are becoming also more computationally powerful. These embedded architectures present new challenges since they execute several applications that must preserve security, allow sharing information in a coherent way, to be scalable and provide the required levels of performance, while at the same time they must be power efficient.

In this context, the vIrtical project focuses on these challenges and as a starting point, it tackles the characterization of applications targeted for the hardware platform developed in the project, that is, a heterogeneous multicore SoC. For this purpose several typical industrial applications and some computer vision kernels have been selected for characterization. From these applications, we have first analyzed acceleration opportunities in order to identify which computational kernels are best candidates for acceleration. Second, we have analyzed memory sharing patterns in order to exploit them to make the coherence protocols more scalable and power-efficient. Moreover we also have analyzed the compression opportunities offered by data moving between memory and cores in order to design a more power-efficient platform. Finally we also identify virtualization opportunities of industrial applications. As we show in this deliverable in the next sections, industrial applications exhibit acceleration, memory and virtualization opportunities that taken into consideration will enable the design of more efficient, scalable and secure heterogeneous multicore devices.

Glossary **(Please ensure that any acronyms used are clearly explained)**

CC: Common Criteria

CV: computer vision

EAL: Evaluation Assurance Level

NoC: Network on Chip

PBX: private branch exchange

QoS: Quality of Service

SoC: System on Chip

VoD: Video-on-Demand

VoIP: Voice over IP

Full Description of Deliverable content

1. Introduction

The use of computers has been extended to most areas of our everyday life. Embedded devices are becoming more and more present everywhere. In the Internet, computational load is moving from computers to servers, being clusters a very popular solution to provide the required computing power. Moreover mobile phones, PDAs and in general mobile devices are becoming also more computationally powerful since they are intended to perform complex computations locally. These embedded architectures present new challenges since they execute several applications that must preserve security, allow sharing information in a coherent way, to be scalable and provide the required levels of performance, while at the same time they must be power efficient.

In this context in the vIrtical project, we tackle these challenges, and as a starting point, we characterize applications targeted for the hardware platform developed in the project, that is, a heterogeneous muticore SoC. For this purpose Thales has provided us with a set of application examples covering the industrial needs. Moreover, some computer vision (CV) kernels have been also selected since scientific and industrial communities are showing a growing interest in developing this kind of algorithms on embedded systems. All the applications are described in Section 2. From these applications, we have first analyzed in Task 1.1 acceleration opportunities in order to identify which computational kernels are best candidates for acceleration (Section 3). Second in Task 1.3, we have analyzed memory sharing patterns (Section 4.2) in order to identify opportunities to be exploited to make the coherence protocols more suitable for embedded devices being more scalable and power-efficient. Moreover, also in Task 1.3 we have also analyzed the compression opportunities (Section 4.3) offered by data moving between memory and cores in the industrial applications, in order to design compression techniques that enable a more power-efficient platform. Finally, in Task 1.1 we have also identified virtualization opportunities of industrial applications.

2. Analyzed Applications

2.1. Applications for consumers

Smart set-top boxes often garner less attention than other connected CE devices like TVs and game consoles. However, there is a growing interest in generic Android based set-top boxes. The changing consumer landscape, however, could present a window of opportunity for smart set-top boxes as more consumers allocate entertainment budget, both monetary and time, to streaming media via the set top box. Today, we start to see on the market some examples how mobile contents could be streamed via a set top box or how to play android games in a set top box.

2.1.1. Android

From Wikipedia, Android is a Linux-based operating system for mobile devices such as smartphones and tablet computers. It is developed by the Open Handset Alliance, led by Google. Today, Android has a large community of developers writing applications ("apps") that extend the functionality of the devices. Developers write primarily in a customized version of Java. Apps can be downloaded from third-party sites or through online stores such as Google Play (formerly Android Market), the app store run by Google. In June 2012, there were more than 600,000 apps available for Android, and the estimated number of applications downloaded from Google Play was 20 billion.

The software stack is divided in four different layers, which include 5 different groups:

- The application layer
    - The Android software platform will come with a set of basic applications like browser, email client, SMS program, maps, calendar, contacts and many more. All these applications are written using the Java programming language. It should be mentioned that applications can be run simultaneously, it is possible to hear music and read an email at the same time. This layer will mostly be used by commonly cell phone users.
- The application framework
    - An application framework is a software framework that is used to implement a standard structure of an application for a specific operating system. With the help of managers, content providers and other services programmers it can reassemble functions used by other existing applications.
- The libraries
    - The available libraries are all written in C/C++. They will be called through a Java interface. These includes the Surface Manager (for compositing windows), 2D and 3D graphics, Media Codecs like MPEG-4 and MP3, the SQL database SQLite and the web browser engine WebKit.
- The runtime
    - The Android runtime consists of two components. First a set of core libraries, which provides most of the functionality available in the core libraries of the Java programming language. Second the virtual machine Dalvik, which operates like a translator between the application side and the operating system. Every application which

runs on Android is written in Java. As the operating system is not able to understand this programming language directly, the Java programs will be received and translated by the virtual machine Dalvik. The translated code can then be executed by the operating system.

- The kernel
  - A customized version of the Linux Kernel will be used by Android for its device drivers, memory management, process management and networking.

Android as shown in section 5.1.2 will be used in set top box or TV to open up the environment to the end users. This will allows the end user, on top of existing functionalities, to stream data, download apps, play games, use IP telephony etc. In this context, vIrtical project will explore during the second year of the project how to improve android performances by offloading code to the GPPA component or by using some compression techniques

2.2. Applications for telephony, data clustering and cryptography

Thales Communications & Security has provided 3 applications from 3 different domains:

1. Asterisk: PBX used in telephony.
2. KCluster: data clustering used in speech recognition.
3. OpenSSL: cryptography used in secure communications.

2.2.1. Asterisk

Asterisk is historically the first open source private branch exchange (PBX) for IP (Internet Protocol), digital and analogue interfaces. Asterisk was created in 1999, first version was released in 2004. The latest version is 10.0 released on 15 December 2011.

Asterisk offers a large set of PBX features like call forwarding, callee identification, conferences and voicemail as well as a variety of call center and operator grade services such as interactive voice response, text-to-speech and external call management via integrated APIs (Application Programming Interface). Many VoIP (Voice over IP), analogue and ISDN protocols are supported together with most of standard codecs. Asterisk is available under dual license: a GPL (General Public License) and a proprietary software license allowing personalized closed source distributions. It runs on Linux on x86 and PPC (Power PC) architectures. It supports many telephony interface cards allowing its use as a gateway and media compression cards allowing video and voice codec real-time conversion. It is maintained by Digium Incorporation based in Alabama, USA.

Asterisk is widely used by Thales Communications & Security, mainly for test purposes, but also as a gateway.

Asterisk based call generators are used during the validation of Thales telecommunication products. Automated test framework executes configured scenario and reports number of dropped or failed calls, average call establishment time, voice quality etc. Thanks to numerous Asterisk's interfaces, this test tool can be used on VoIP as well as analogue interfaces.

Asterisk is also used in several Thales gateways. Asterisk's features allow its use in two types of gateway: VoIP/analogue and VoIP/ISDN gateways. An example of a VoIP/analogue gateway is an integration of radio equipment in a VoIP network, where it's necessary to convert an analogue radio interface to an IP based one. Thales's VoIP/ISDN gateways integrate client specific value added services like QoS management.

Efficiency and scalability of Asterisk on multicore embedded platforms is of prime interest for Thales, as well as the capability to handle power-efficient switches between a large number of connections occurring in seldom cases and a small number of connections occurring at all times.

2.2.2. KCluster

At the heart of the KCluster application there is the well-known K-Means core algorithm which is done in one single stage. The variant here proposed does the split in two-stages in order to allow for parallelization of multiple K-Means algorithms in the second stage. K-Means is an algorithm designed to iteratively build a representative codebook from a training set of multi-dimensional data. The resulting codebook consists of a set of centroids, optimized to minimize a global distortion. The global distortion is computed as the cumulated distortion over all the training vectors using a nearest-neighbor criterion to select the representative centroid for each data vector.

K-Means principle

1. Initialization step: before entering the k-Means iterative optimization process, the centroids should be initialized. Various methods can be used for initialization depending on the available a priori information on the data. Here we only provide two types of initialization: random selection of vectors in the training set, and selection of the vectors which are the closest to the global mean of the training set.
2. Classification step: from an iteration to the other, once the centroids are updated (or initialized), all the training vectors are classified according to a nearest-neighbor rule. The training set is therefore partitioned in K classes associated to the corresponding K centroids.
3. Update step: once all the training vectors have been classified, the centroids are updated using the vectors in the corresponding class. The resulting codebook can then be re-optimized by going back to step 2), or considered as the final codebook depending on the termination step.
4. Termination step: the distortion is calculated as the cumulated distortion over all classes. One typical termination criterion is to stop the iterative optimization, when the relative distortion improvement is below a predefined threshold.

2.2.3. OpenSSL

OpenSSL is a well-known suite of open-source library and tools implementing cryptographic algorithms used for authentication and secure data transfers over networks. It is used by many services such as https and ssh. As indicated by its documentation, it implements the following cryptographic functions:

- Creation of RSA, DH and DSA key parameters,
- Creation of X.509 certificates, CSR and CRL,
- Calculation of message digests,
- Encryption and decryption with ciphers,
- SSL/TLS client and server tests,

Handling of S/MIME signed or encrypted mail.

2.3. Applications for Vision and Image Processing

Scientific and industrial communities are showing a growing interest in developing Computer Vision (CV) algorithms on high-end embedded systems. However, CV algorithms are well known for their high complexity and for being very resource-demanding. CV processing is indeed a computation-intensive task, dealing with huge amounts of data (an image can count several millions of pixels/bytes) and performing a large number of repetitive calculations over the whole image data set or part of it. Low-level data processing, like pixel-based operations, requires a large number of memory transactions and can quickly become a bottleneck if the system architecture is not tailored to such a kind of processing. Furthermore, higher-level algorithmic tasks (e.g., segmentation or recognition) can require complex iterative or recursive mathematical kernels that are quite demanding in terms of computational power. This is especially true in the embedded domain, where the complexity of processing nodes is limited by several constraints like power consumption, size and cost.

Accelerating those key computational kernel is thus paramount to speeding-up the execution time of more complex applications such as object motion tracking, object recognition and similar. We have identified three core kernels that are used by several vision applications to extract significant features from images:

1. SIFT (Scale-Invariant Feature Transform)
2. SURF (Speeded-Up Robust Features)
3. FAST + BRIEF (corner detection).

These kernels are used as a basic step in many applications in the field of computer vision: they work on raw images, and managing large amounts of raw data they basically turn out to be computation bottlenecks.

2.3.1. SIFT

SIFT **[Bay08]** is an algorithm that detects and describes image features. The computed features are mainly invariant to image scaling and rotation, and partially to change in illumination and camera viewpoint. SIFT uses a cascade filtering approach: it looks for some distinctive points (keypoints) in the image, then most of the candidate points are discarded at an early computation stage, in order to skip computationally intensive operations. The first stage of the algorithm creates a Gaussian pyramid that is a set of images which are derived from the original one applying a sequence of downsampling, upsampling and blurring filters. The algorithm provides three main configuration values to define this gaussian space, which are the number of octaves, the scale of first octave and the number of layers. Each octave is characterized by a different image scale, and default parameters imply a variation from twice the original size (octave scale 1) to one fourth of the original size (octave scale 2). The layers are the images actually contained in each octave: each layer, from 0 to 2 in the default implementation, corresponds to a increasing sigma value for gaussian blurring filter. The second stage computes the Difference of Gaussians (DoG) between adjacent layers in the same octave, and also the gradient of images, which is computed for each octave and layer. The third stage is the keypoint localization: local estrema in the DoG images are selected as candidate keypoints, considering neighboring pixels in both the selected image and the adjacent ones (which have been computed by the same layer image). Keypoints are filtered to remove edges, as they are redundant, and finally a non-maxima suppression is performed. The fourth stage assigns an orientation to each keypoint (sometimes more than one), using the local values of the gradient in the point area. The last stage is the creation of keypoint descriptors. The 16x16 region around each keypoint is described using a statistical analysis of local gradient orientations: orientation histograms are computed considering 4x4 subregions and 8 bins, then they are packed into 128-entries descriptors. The descriptors are finally normalized with the aim to minimize the effects of change in illumination. SIFT is one the most accurate algorithms for feature detection, but at the same time it is quite slow, so it is currently considered to be unsuitable for real-time applications, unless we consider small images or need a greater accuracy.

2.3.2. SURF

SURF **[Cal10]** is a detector and descriptor algorithm. Its approach is similar to SIFT but SURF uses faster techniques, and in some cases it is also more robust than SIFT. The detector stage is based on the determinant of a Hessian matrix, and considers both position and scale. Hessian matrices are computed by means of discrete box filters, that approximate second order Gaussian derivatives and can be computed using integral images. SURF uses 9x9 box filters at the lowest scale, referred as scale 1.2 (1.2 is the sigma parameter of the Gaussian filter). The scale space is analyzed by upscaling the filter size using bigger masks: this operation has a constant cost when using integral images, which in general enable fast computation of box type

convolution filters for big kernel sizes. The scale space is divided into octaves, which are subdivided into a constant number of levels. The default parameters of the algorithm provide 4 octaves, and 4 levels per octave.

The minimum increase of the mask size for two adjacent levels corresponds to 6 pixels, in order to guarantee a good Gaussian approximation. At the first octave, filters with sizes 15x15, 21x21 and 27x27 are applied. For each new octave, the filter size is doubled: the filter sizes for the second octave are 15x15, 27x27, 39x39, and 51x51. The last octave uses 51x51, 99x99, 147x147 and 195x195 kernels. To reduce the sampling intervals for the extraction of the interest points, the sampling step is doubled at each new octave, with minimum loss of accuracy.

In order to localize interest points, the maxima of calculated determinants are interpolated in scale space, and then a non-maximum suppression in a 3x3x3 neighborhood is applied.

The descriptor phase of SURF is similar to SIFT: a main orientation is first computed for the keypoint, and then orientation statistics are extracted for a region centered on the point itself. The size of this window is 20 times the image scale value, and it is divided in 4x4 subregions, each one internally divided in 4x4 areas. SURF computes orientations using Haar wavelet responses, which are invariant to changes in illumination and contrast. The final keypoint descriptor vector contains 64 elements.

2.3.3. FAST + BRIEF

FAST **[Ros10]** is a corner detection algorithm. It analyzes all points in the image, and compares the intensity value of each point p with all the sixteen points on the circle of radius 3 and center p; p is classified as a corner if there exists a set of contiguous pixels within the circle that are all brighter (minimum) or darker (maximum) of p (with a tolerance threshold). The number of contiguous pixels and the threshold value are both algorithm parameters; typical values are respectively 9 and 20.

All the versions of FAST assign a score to each detected corner, in order to enable a subsequent non-maximal suppression stage, with the aim to filter the corners which have an adjacent corner with higher score. Even if it can be used as a feature detector, FAST does not provide a step to generate keypoint descriptors. Nonetheless, extracted features can be described using a companion algorithm: a typical example is BRIEF, which produces compact binary descriptors.

BRIEF **[DBw³]** is an algorithm that computes keypoint descriptors using a method that produces efficient bit strings. The algorithm computes a small number of pairwise comparisons on a image patch centered on the keypoint, and then creates a bit vector as the final result of all the comparisons related to the same keypoint. This vector is finally packed in a bit string. At each step, BRIEF smooths the current image patch with a Gaussian filter, in order to reduce noise sensitivity and increase the stability and repeatability of the descriptors. The numbers of

pairwise comparisons considered by the authors (corresponding to a binary string of 16, 32 or 64 bytes) yield a good trade-off among speed, storage efficiency and recognition rate.

3. Acceleration opportunities

In this section we analyze the acceleration opportunities of the target applications described in the previous one.

The target platform template considered in the vIrtical project consists of an ARM-based host subsystem, plus accelerators of different nature (i.e., a programmable manycore, the GPPA, plus HW functional units, HWPUs), which can deliver tremendous peak performance, given that the target application can exploit it. In this section of the document we investigate the potential for different approaches to accelerating the target applications. We consider three different approaches, which are currently being adopted by vendors and/or research institutes

1. **Architecture Specific Optimizations**: this methodology consists of assessing the acceleration opportunities for an implementation of the target algorithm that is optimized to run on the ARM host processor only, taking advantage of specific architectural features (e.g., SIMD engine).
2. **Coarse-Grained Thread-Level Parallelization**: this approach aims at assessing the acceleration opportunities for an implementation of the target algorithm that parallelizes the workload among the small number of available cores in a typical ARM multi-core host subsystem. Coarse-grained threads are identified as a parallel unit of work from the application.
3. **Fine-Grained Data-Level Parallelization**: this methodology explores the potential for accelerating the target application on a large number of threads, by offloading it onto the GPPA. Thus we consider a different parallelization scheme which is fine-grained enough to keep busy a very large number of cores. We leverage loop-level parallelism to generate several fine-grained threads.

In particular, approaches 1 and 2 are suitable for the industrial applications described in Section 2.1, as explained in the following sections. Computer vision algorithms, on the other hand, are characterized by a remarkable workload, in particular when high definition images are considered. The traditional target of CV libraries is the desktop computing environment. Since embedded devices have not the same computing capabilities of desktop mainstream processors, the execution of such algorithms on mobile platforms often presents unsatisfying performance. However, CV workloads often exhibit fine-grained (i.e., pixel-level) parallelism, which makes them a suitable candidate for acceleration on the GPPA (approach 3).

We will thus describe the acceleration approaches for the two categories of applications in the two following sections.

3.1. Acceleration opportunities for telephony, data clustering and cryptography applications

3.1.1. Asterisk

As introduced in Section 2.1.1, Asterisk offers several PBX features for telephony services. New voice calls are handled by creating additional threads, thus it becomes straightforward to manage execution of multiple threads in parallel. Clearly, the degree of parallelism is directly dependent on the number of simultaneous calls occurring at the Asterisk host, thus we can only consider exploiting coarse-grained task-level parallelism for this application. This type of parallelism is well suited to run entirely on the host ARM system. A tasking programming model (e.g., OpenMP `task` directives) can be used to specify dynamically the creation of additional parallelism from the application.

3.1.2. KCluster

The KCluster application heavily relies on KMeans-based data clusterization. There are many approaches in literature discussing parallelization of KMeans for different systems (for a good survey see **[Zha06]**). The algorithms lend itself to different partitioning granularities: task parallel, data parallel and a mix of the two. In order to allow for parallelization of the KCluster application, we provide the following modified version of KMeans.

Modified K-Means

In the provided implementation, we consider a clustering algorithm combined several K-Means processes. A primary codebook (size L0) is first optimized on the whole training set. Then for each resulting centroid, we consider the set of vectors in the corresponding class as a new training set for secondary codebook (size L1). The secondary codebooks are iteratively optimized, and training data are re-allocated at the end of each iteration.

Parallelization

The optimization of the L0 secondary codebooks (of size L1) can be parallelized using a multicore processing platform. Generally the number of core processing units will be less than the number of L0 of codebooks, and the allocation of each optimization process should be done dynamically, since the duration of each optimization process can be different. It is approximated in the example code by the required number of iterations, in order to provide an example of dynamic allocation to the different cores.

This partitioning scheme, mostly task-based, is convenient for execution on the ARM subsystem. The load imbalance issue can be easily addressed by dynamically allocating parallel tasks to available processor, rather than statically determining a workload partitioning scheme.

3.1.3. OpenSSL

Cryptographic algorithms can be accelerated by using co-processing resources such as SIMD engines or specific cryptographic devices such as those present in PowerQUICC processors. Within the vIrtical project the first approach can be implemented by leveraging ARM NEON extensions. Dedicated HW blocks like those from the second approach could also be considered by implementing a specific HWPU (HW Processing Unit, see D1.2).
Also, a few approaches to parallelization of a class of encryption algorithms (AES, Rijndael) on many-core devices (i.e., GPUs) are available in literature **[Le10]**, which promise speedups of up to 7x, and which could be considered as a guideline to accelerating such algorithms on the GPPA in the vIrtical project.
More information on OpenSSL is available at www.openssl.org

3.2. Acceleration opportunities for Vision applications

While in literature there are several approaches to parallelizing applications for data clustering or cryptography, discussed in the previous section, for computer vision kernels there is less previous work available, which calls for a more detailed analysis. The source code for the computational kernels described in Section 1.2 is available from the open-source OpenCV (Open Source Computer Vision) library **[Stu11]**. OpenCV is a CV-oriented, cross-platform programming library, widely used by both academic and industrial partners. A pre-parallelized version of these kernels, which we use for approach 2 from the list above, comes from the MEVBench suite from University of Michigan **[Cle11]**.

The presentation of our study is divided into three independent sections, each one related to a different approach. To conduct our preliminary studies and to validate the above-mentioned methodologies prior to the definition and the availability of the vIrtical platform, we use the following approaches:

A. To estimate the behavior of code optimization strategies meant for the host system we use a Qualcomm DragonBoard **[OCVw³]** as a hardware platform. The DragonBoard is an advanced developer board, featuring a Snapdragon S3 APQ8060 SoC, which shares the following key architectural characteristics with the host system of the vIrtical platform:

   - An ARM-based dual-core host system, based on the same ARMv7 ISA of the Cortex A15
   - Support for the signal processing-oriented NEON instruction set extensions and floating-point VFPv3 extensions (the VeNum media processing engine)[1].
   - An Android-ready SDK, which allows us to readily execute OpenCV codes

[1] Qualcomm's NEON data paths are 128-bits wide. Since the ARM NEON is 64-bits wide, the Snapdragon S3 VeNum can issue the equivalent of two NEON instructions in parallel.

Table 1 summarizes the main architectural parameters for the Snapdragon S3 SoC.

B. To estimate acceleration opportunities for parallelization approaches leveraging programmable manycores (i.e., the GPPA – approach 3 from the list above) we either use a in-house, SystemC simulator of a cluster of the GPPA (see D1.2) or commodity GPU card. Specifically, we use a NVIDIA GeForce GTX 480 on a desktop machine equipped with a Intel Core i7920 CPU @ 2.67 GHz and 6GB DDR3.

Snapdragon S3	
Processor	
CPU	Dual Scorpion CPU
Frequency	1.2 GHz, per core
L1 Cache (I/D)	(32KB / 32KB) per core
L2 Cache	512 KB Shared
Memory	
Frequency	LPDDR2-333 (ISM)
Memory Size	1GB

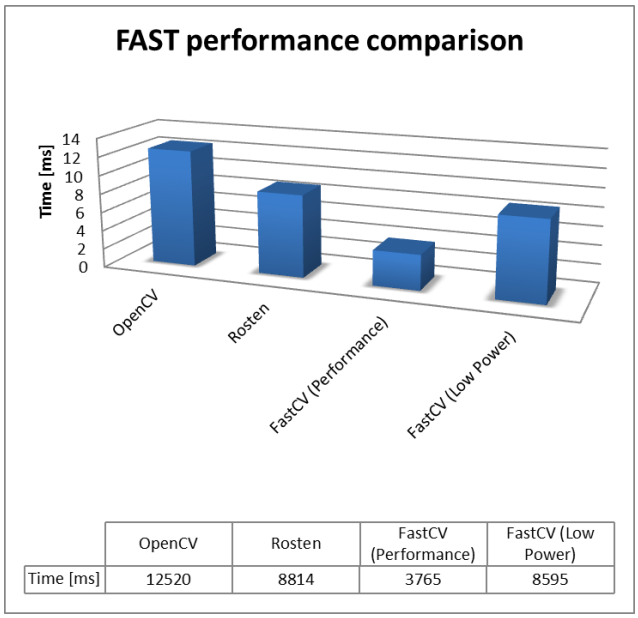
Table 1: Snapdragon S3 SoC hardware parameters.

3.2.1. Architecture Specific Optimizations

The first set of experiments is aimed at assessing the acceleration opportunities for architecture-specific optimizations on the host processor. We consider three different implementations of the FAST benchmark (see FAST + BRIEF):

- **OpenCV**: the original algorithm, contributed to OpenCV by the author, Edward Rosten. The algorithm has been substituted in OpenCV 2.3.1 with the optimized version (see below).
- **Rosten**: Machine-generated code, optimized for speed **[Ros10]**. It consists of an heuristic procedure derived from a machine learning tree.
- **FastCV**: the FAST algorithm from the FastCV library. FastCV **[QDNw³]** is a proprietary vision library distributed by Qualcomm providing a mobile-optimized computer vision library which includes the most frequently used vision processing functions for use across a wide array of ARM-based mobile devices. FastCV is designed for efficiency on all ARM-based processors, but is tuned to take advantage of Qualcomm's Snapdragon processor (S2 and above). In particular it uses vectorial instructions to perform multiple comparisons per cycle.

OpenCV and **Rosten** algorithms are compiled with the g++ compiler included in the standard NDK toolchain, with instruction set ARMv7, NEON support disabled and optimization level -O2. The **FastCV** library is distributed in binary form and can execute in two distinct operative modes: Performance mode (1), which enables the SIMD unit, and Low Power mode (2), which only runs on the processor.



	OpenCV	Rosten	FastCV (Performance)	FastCV (Low Power)
Time [ms]	12520	8814	3765	8595

Fig. 1: Comparison of FAST implementations.

Fig. 1 depicts the execution times (in milliseconds) of each FAST implementation. On the bottom part of the figure we report execution times in a table.

The results show that exploiting processor architecture-specific optimizations, notably the SIMD engine, the algorithm execution time can be sped-up up to 2.34 w.r.t. the standard version.

3.2.2. Coarse-Grained Thread-Level Parallelization

Besides using SIMD acceleration, another opportunity for improving the performance of the target applications on the host subsystem is parallelization. The small number of processors available on the multi-core processor clearly limits the number of threads that it is possible to exploit in the parallelization strategy. A parallel implementation of the target algorithms for general-purpose processors is provided in the freely-available, open-source MEVBench suite **[Cle11]**. We tuned the implementation of SIFT, SURF and FAST to run on Android systems. We consider a test image scaled in three different sizes:

- Small (352x288 pixels): 101376 pixels that require 2376 KB in RGB format and 792 KB in greyscale. 169 KB in PNG format. It is a standard size for mobile phone videos and images produced by very low-resolution digital cameras.
- Medium (640x480 pixels): 307200 pixels that require 7200 KB in RGB format and 2400 KB in greyscale. 465 KB in PNG format. It is a standard VGA format.
- HD (1920x1080 pixels): 2073600 pixels that require 48600 KB in RGB format and 16200 KB in greyscale. 1915 KB in PNG format. It is a standard full high definition format for video streams.

Note that the last configuration could only be tested with the FAST + BRIEF algorithm, since both SIFT and SURF run out of memory during the experiments.

Fig. 2 shows the execution cycle scaling when parallelizing the various applications among an increasing number of threads (on the x-axis). Cycle count (on the y-axis) for the various configurations is normalized to the value for single thread, small image.

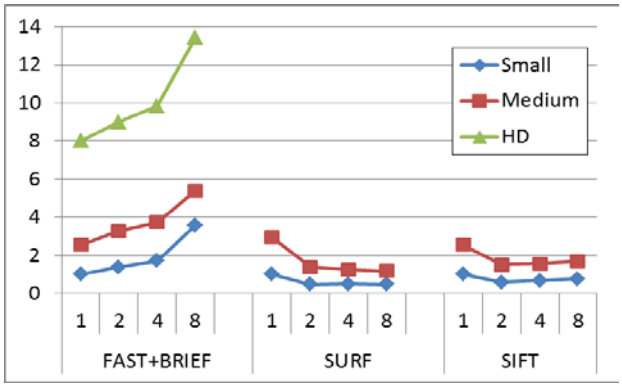


Fig. 2: Execution time (normalized) scaling for coarse-grained parallelization.

Regarding the FAST + BRIEF application, it is possible to see that there is basically no speed-up when increasing the number of threads. This is due to i) the coordination phase which is extremely time consuming and not parallelizable thus dominating the Amdahl law sequential part and ii) the parallel part which adds coordination overhead (barrier) which increases with the number of threads.

Things change a little for the remaining two applications, where coarse-grained parallelization can achieve some speed-ups. For SIFT, the maximum speed-up is obtained with two threads (i.e., as many threads as cores). With this configuration the workload is balanced, and the coordination phase has a negligible impact. When more threads are considered the total overhead increases with no additional benefits derived by hiding memory latency by means of thread switching.

The last application, SURF, lacks a final complete coordination phase. In more detail, each thread performs feature extraction on a partially overlapping part of the image, and the same feature can be extracted multiple times: a final matching step is omitted, and the coordination phase reports the sum of the feature extracted. This is not a real issue, as a subsequent step may discard the repeated values if needed. SURF benefits from the overlapping between computation and memory transfers, due to a different algorithmic approach and a lighter coordination phase. Thread switching partially hides the effects of memory latency when we consider the medium-sized image, and consequently the speed-up increases when using more than two threads (more threads than cores).

Overall, the results for these preliminary experiments show that i) coarse-grained parallelization schemes intended for general-purpose processors may perform poorly on embedded devices,

due to different hardware (memory, synchronization) and ii) overall, increasing the number of software threads beyond the core count does not bring additional benefits, on the contrary it may slow down.

3.2.3. Fine-Grained Data-Level Parallelization

The experiments presented in the previous sections demonstrate that the speedup that can be achieved on the host subsystem by means of parallelization and exploitation of SIMD engines is limited. The platform template that we target in the vIrtical project can be equipped with a many-core programmable accelerator, the GPPA (General Purpose Programmable Accelerator). Clearly, the parallelization scheme that needs to be adopted to take advantage of the GPPA is very different from the one used in the MEVBenchs. Indeed, if the number of processing cores is small, parallelization can generally be coarse-grained, where the amount of work is high enough to keep the cores busy and can tolerate synchronization overheads among a small number of workers. However, the same parallelization approach is bound to provide poor results for higher processor counts. It is therefore important to evaluate the benefits of a parallel implementation that is designed with scalability in mind in this scenario. We present in the following the parallelization strategy for each of the benchmarks.

3.2.3.1. SIFT

The SIFT algorithm consists of two main phases: features detection and feature description. We focus here on the most computation-intensive part, feature detection. A candidate feature (keypoint) is a pixel located at a specific image frequency scale that has either the minimum or the maximum value in relation to its neighborhood, defined by a 3x3 window located at the same scale space and at the upper and lower adjacent scales. The frequency band for each scale is obtained by using the difference of Gaussian (DoG) operation, which is computed by subtracting two identical images convolved by two different Gaussian kernels. Equation (1) defines the convolution operation for the first and the other subsequent scales, where I is the input image, K the Gaussian kernel and G the smoothed image.

$$\begin{aligned} G_0(x,y) &= K_0(x,y) * I(x,y) \\ G_{s+1}(x,y) &= K_{s+1}(x,y) * G_{s+1}(x,y) \end{aligned} \quad (1)$$

Equation (2) defines the difference operation (DoG), where D is the resulting image at a specific frequency scale defined by the kernel values (s).

$$D_i(x,y) = G_{s+1}(x,y) - G_s(x,y) \quad (2)$$

Accepting a keypoint as a feature is evaluated through three functions: location refinement, contrast check and edge responses. The location refinement is performed as shown in Equation (3), where I is the pixel location vector (x; y; s) (coordinates <x;y> and scale s). This equation performs an interpolation operation with the pixels found inside of the keypoint's neighborhood

(26 pixels), with ω added to the current keypoint position in order to produce its new location. The ω offset is also used to compute the keypoint contrast. If the result is smaller than a user-defined threshold then the keypoint is rejected.

$$\omega = -\left(\frac{\partial^2 D}{\partial l^2}\right)^{-1} \frac{\partial D}{\partial l} \tag{3}$$

Finally, principal curvature analysis is evaluated to reject keypoints that are located at poorly defined edges, which are consequently highly unstable to noise. This is particulary necessary for the difference of Gaussian function because most of the detected keypoints are located at edges. The bigger the principal curvature, the poorer is its edge. Equation (4) shows how the principal curvature is computed (and rejected if it is above a pre-established threshold).

$$\frac{\left(\frac{\partial D}{\partial x^2} + \frac{\partial D}{\partial y^2}\right)^2}{Det(H)} \leq Threshold \tag{4}$$

where

$$H = \begin{bmatrix} \frac{\partial D}{\partial x^2} & \frac{\partial D}{\partial x \partial y} \\ \frac{\partial D}{\partial y \partial x} & \frac{\partial D}{\partial y^2} \end{bmatrix} \tag{5}$$

Fig. 3 shows a block diagram representing the main operations of the SIFT algorithm.

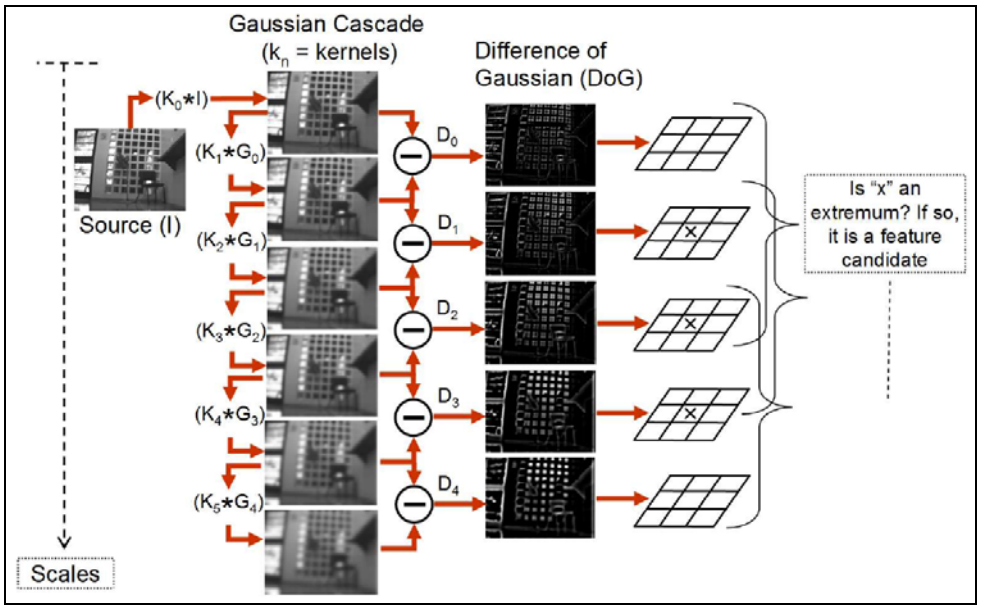


Fig. 3: Block diagram representing the main operations of SIFT.

We consider the following parallelization strategy, divided in four stages:

a) Image upsampling and downsampling

b) Image smoothing

c) Difference of Gaussians and gradient

d) Keypoint detection and orientation

The granularity of parallelization is in general decided based on hardware peculiarities, we describe here parameters chosen for a GPU prototype implementation. In a) we create an upscaled image (2:1) and two downscaled images (1:2), (1:4). The parallelization strategy employs as many parallel work units as pixels in each of the output images. Note that multiple work units can be in general grouped in a single parallel thread depending on the peculiarities of the target hardware. For GPUs employing the smallest work unit and the highest number of threads achieves the best results, so we create as many threads as pixels. Also note that scaling independent images can also be done in parallel.

In b) we apply Gaussian filtering as described by equations 1). In c) we compute the DoG (equation 2) and the gradient (), while in d) we accept/reject candidate features through the checks described by equations 3) and 4). The parallelization scheme for all these kernels follows the same pixel-level partitioning.

Fig. 4 shows the speedup results achieved on the target GPU for the SIFT-parallel algorithm, for increasing input image sizes (on the x-axis). We show three different bars. The blue bars represent the speedup obtained against the original OpenCV SIFT function running on the CPU when the whole algorithm is running as we described it above. The green bars represent the speedup obtained when we neglect the cost for moving data back and forth from the CPU space to the GPU space (e.g., what we could achieve if we could iterate the procedure over multiple input images and hide the transfer latency with double buffering techniques). Finally, the red bars show the speedup that can be obtained when we do not run the last kernel (keypoint detection).

Both green and red bars show improvements against the blue bars. In particular, the keypoint detection kernel proves to be the bottleneck in the scalability of the SIFT-parallel algorithm. The main performance blockers for this parallelization scheme are two:

i. the presence of conditional instructions, which are in particular used heavily in the keypoint detection stage and which is well known to be detrimental to GPGPU programs performance

ii. the very significant amount of data moved inside the kernels between global and local GPU memories. Figure 5 reports the size of memory transfers issued in each of the kernels in SIFT-parallel. It is possible to notice that overall there is a huge amount of data being transferred from global to local memory in GPU clusters. A better grouping of work units into parallel threads can help reducing the overall size of these transfers.

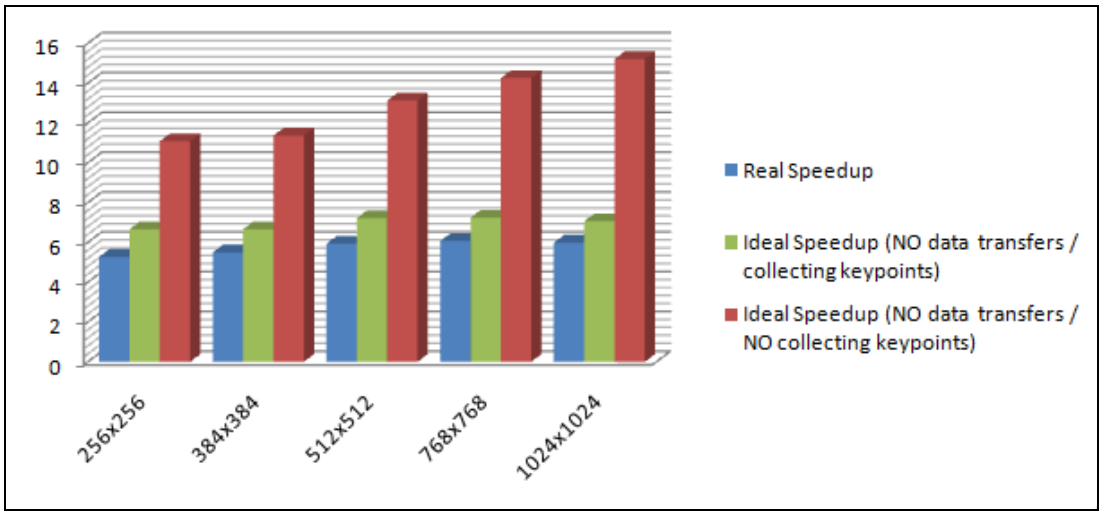


Fig. 4: Speedup results for the SIFT-parallel algorithm.

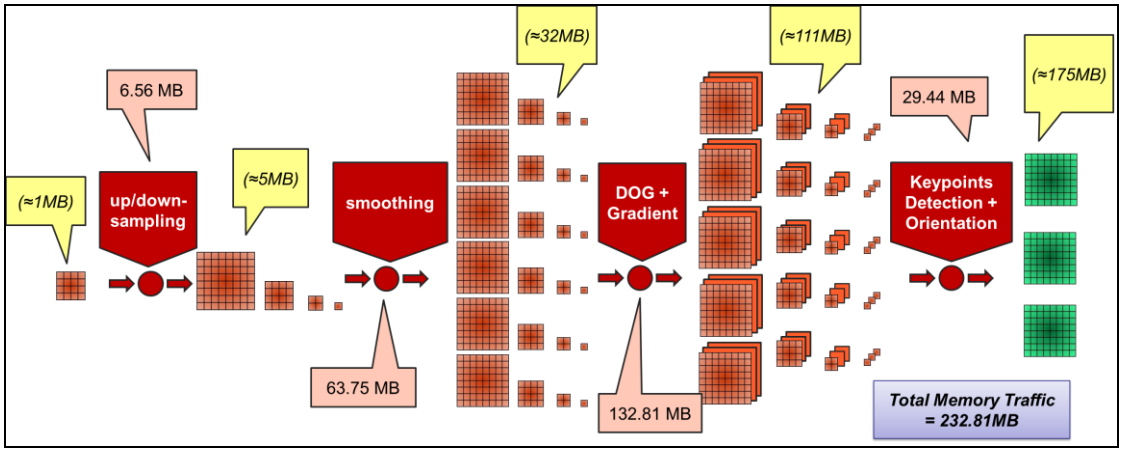


Fig. 5: Memory traffic due to data movements inside kernels in SIFT-parallel.

3.2.3.2. SURF

To understand the acceleration opportunities for the SURF application we analyze the implementation available in the OpenCV library running on top of Android 2.3.3 on the DragonBoard. We execute a series of tests on the sequence “freiburg1_desk”, extracted from the dataset described in **[Low04]**. From the original frame size (640x480) we obtained two scaled frame sets (400x300 and 320x240), that we used to assess the algorithm behavior. Figure 6 shows the flat profiles obtained through gprof and android-ndk-profiler. On the top part of the figure (a) we show the profile for frames containing from few tens up to two hundred features. On the bottom part of the figure (b) we show the profile for frames containing eight hundred to one thousand features. The functions implementing the three main stages of the SURF algorithm (see SURF) are:

- cv::SURFBuildInvoker::operator() where the pyramid of Gaussian-filtered images is created (see Algorithm 1)

- cv::SURFFindInvoker::operator() where maxima of the determinant of the Hessian matrix are computed
- cv::SURFInvoker::operator() where the descriptor of the extracted keypoints is computed.

It is possible to see that most of the time is spent in smoothing images through Gaussian filtering, as we already observed with SIFT. The second most important contribution to overall execution time is descriptor creation, which clearly has a linear dependence with the number of features found in the target image.

% time	cumulative seconds	self seconds	name
66.63	12.12	12.12	cv::SURFBuildInvoker::operator()...
24.79	16.63	4.51	cv::SURFInvoker::operator()...
4.29	17.41	0.78	void cv::resizeArea_ <...
2.97	17.95	0.54	cv::SURFFindInvoker::operator()...
0.44	18.03	0.08	cv::integral_8u32s(...
0.33	18.09	0.06	void cv::resizeAreaFast_ <...

(a)

% time	cumulative seconds	self seconds	name
47.64	32.00	32.00	cv::SURFInvoker::operator()...
39.08	58.25	26.25	cv::SURFBuildInvoker::operator()...
9.84	64.86	6.61	void cv::resizeArea_ <...
1.91	66.14	1.28	cv::SURFFindInvoker::operator()...
0.60	66.54	0.40	cv::phase(...
0.31	66.75	0.21	void cv::resizeAreaFast_ <...

(b)

Fig. 6: Execution time profile for SURF. a) profile from frames with up to 200 features. b) profile from frames with 800 to 1000 features.

Algorithm 1 SURF: filtraggio della piramide gaussiana.

Require: integral image: sum
Require: array of layers' info: $layers$
for all $l \in layers$ **do**
  for $i = 0 \rightarrow l$.octave.rows **do**
    $u \leftarrow i * l$.octave.step
    for $j = 0 \rightarrow l$.octave.cols **do**
      $v \leftarrow j * l$.octave.step
      $(dx, dy, dxy) \leftarrow \text{Haar}(sum, u, v, l.\text{pattern})$
      $l.\text{dets}(i, j) \leftarrow dx * dy - 0.81 * dxy * dxy$
      $l.\text{traces}(i, j) \leftarrow dx + dy$
    end for
  end for
end for

From these observations the parallelization strategy can be designed as follows:

- Keypoint detection: octaves are traversed sequentially in the original application, but for each of them the computation of the trace and determinant of the Hessian matrix can be executed in parallel for every point in that level, as well as the search for maxima and space-scale interpolation;
- Orientation: the dominant direction can be computed in parallel for every keypoint. Moreover, for each keypoint it is possible to parallelize wavelet filtering and response accumulation in the point of interest neighborhood;
- Descriptor computation: every sub-region of every descriptor can be independently extracted in parallel over multiple threads (synchronization is needed), to determine the values of the derivative in every key point of the sub-region. The following normalization stage can also be executed in parallel over multiple features.

3.2.3.3. FAST

FAST **[Ros10]** is a corner detection algorithm that first analyzes all points in the image, and compares the intensity value of each point p with all the sixteen points on the circle of radius 3 and center p (see ***Fig. 7)***. p is classified as a corner if there exists a set of contiguous pixels within the circle that are all brighter (minimum) or darker (maximum) of p (with a tolerance threshold). The number of contiguous pixels and the threshold value are both algorithm parameters; typical values are respectively 9 and 20.

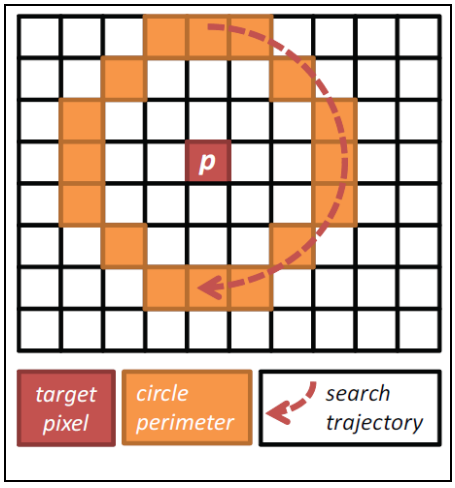


Fig. 7: FAST access pattern.

Given an N*M input image, the algorithm generates an output vector whose size is N*M*3, containing the coordinates of the corner points and a score. The latter is used in a subsequent non-maxima suppression stage, which merges multiple pixels belonging to the same corner. Finally, a keypoint detection pass detects relevant features. The core kernel performs most of the computation and it exhibits data-parallelism at the pixel level.

To estimate the acceleration opportunities for our parallel FAST algorithm we design a prototype implementation for a cluster of the GPPA, for which we developed a SystemC virtual platform. The block diagram for a cluster of the GPPA is shown in ***Fig. 8***.

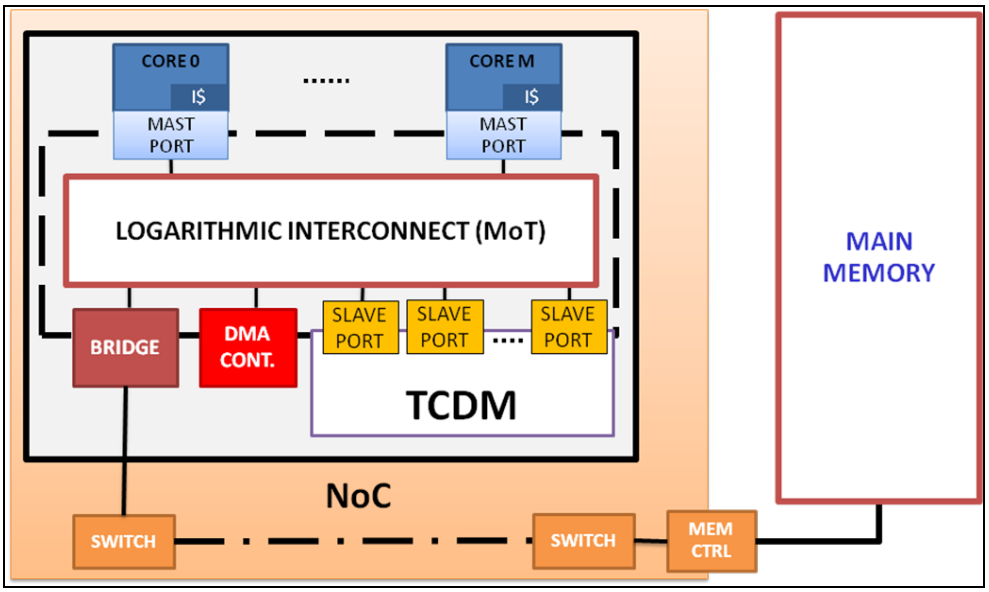


Fig. 8: Block diagram of the single-cluster GPPA used for the experiments.

A cluster consists of a configurable number (up to 16) of RISC32 processors with private instruction caches. Processors are interconnected through a low-latency, high-bandwidth logarithmic interconnect, and communicate through a fast multi-banked, multi-ported L1 scratchpad memory (Tightly-Coupled Data Memory – TCDM). The number of memory ports in the TCDM is equal to (a multiple of) the number of banks to allow concurrent accesses to different banks. Conflict-free TCDM accesses have two-cycles latency.

Scaling to larger core counts in this architectural template is achieved by interconnecting several clusters through a NoC, to which we also connect a memory controller to the main memory, where program code and data are originally stored. The table below summarizes the main architectural parameters for our virtual platform.

ARM v6 Cores	(up to) 16	TCDM banks	32
$I\$_i$ size	1 KB	TCDM size	256 KB
$I\$_i$ line	4 words	MAIN mem latency	50 cycles
t_{hit}	1 cycle	MAIN mem size	256 MB

To match these parameters, parallel units of work in FAST are designed to process an entire image row. We consider different sizes for the input images: 64x64, 128x128, 256x256 and 512x512 pixels. As such, the granularity of parallel work units doubles with the input size. However, due to the limited size of the L1 memory (TCDM) it is not possible to store the whole dataset therein. We thus split the image into stripes, and process them one after the other. We adopt a double buffering technique to overlap computation and DMA transfers from the global memory.

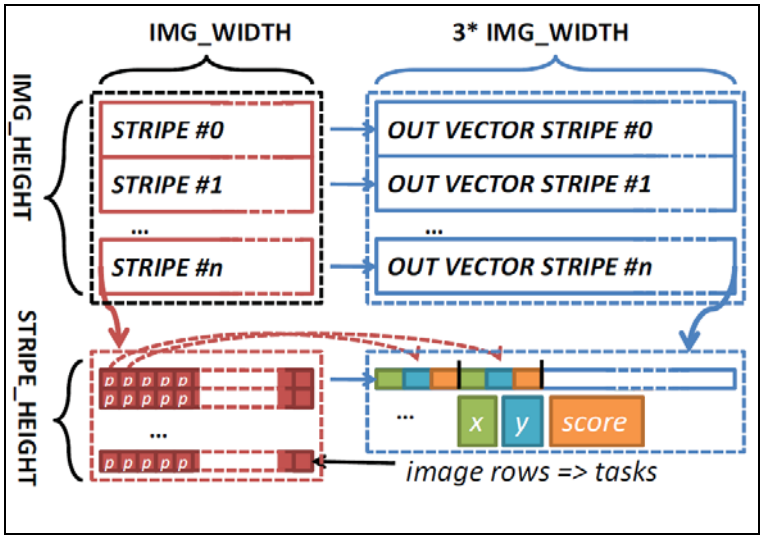


Fig. 9: Fast-parallel implementation.

The parallelization scheme is schematically represented in ***Fig. 9***. ***Fig. 10*** shows the speedup of the parallelized algorithm compared to the sequential version for different image sizes. A considerable speedup is achieved even for small images (11× for a 64x64 image, with each thread only processing 64 pixels) and we reach almost ideal speedup for bigger input sets (256x256 and 512x512), thus confirming the scalability of our parallelization scheme.

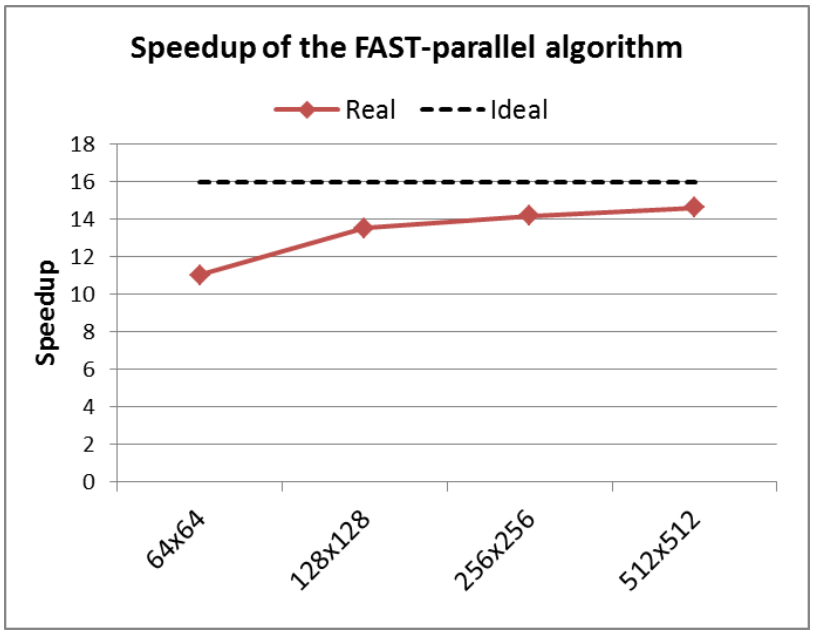


Fig. 10: Speedup of the FAST-parallel benchmark for different image sizes.

4. Memory system opportunities

Concerning the memory system opportunities, we have identified two different design issues in multicore systems that can help on obtaining the aforementioned target objectives described in

the introduction: cache coherence protocols, (that manage shared objects between cores of the CMP), and compression (for a power consumption and execution time efficient system).

The cache coherence problem arises when copies of the data are found at several private caches associated to the cores, being reachable at the same time by two or more cores, are modifiable by some of them. The cache coherence protocol must guarantee coherence of the data through the entire system which means deciding how and when a single core is granted permission to modify data and ensuring that subsequent readings of the written data by other cores will attain updated copies of the modified data (multiple readers). To do so, different cache coherence mechanisms can be applied. Commonly, these mechanisms introduce certain overhead in terms of either coherence traffic issued or storage resources required, which can significantly penalize performance, increasing the execution time of the running applications as well as power consumption. The current trend to increase the number of cores into CMP and MPSoC systems further aggravates this problem, jeopardizing the scalability of the coherence mechanisms. In brief, the cache coherence protocol does not scale, due to its resource overheads and its indirection when accessing data (access latency is increased).

On the other hand, a different approach to tackle the coherence problem has recently been proposed (**[Cues11]**, **[Har09]** and **[Kim10]**), consisting on removing coherence maintenance for those data objects (memory blocks) that do not need it, either because they are not shared (private to one core) or because they are shared but never written by any core. This approach requires the use of effective mechanisms to identify data blocks that do not need coherence maintenance, which in turn may introduce certain overhead. Anyway, the success and suitability of the selected coherence mechanism will strongly depend on both the architectural context of the system it is being applied to and the sharing patterns of the applications running on the system.

Therefore, as a first step in the exploration of cache coherence protocols in the vIrtical project, a detailed analysis of the sharing patterns of the applications to be supported is needed, in order to identify opportunities of applying one or another cache coherence mechanism together with different coherence optimization techniques.

At the same time, in the vIrtical project, reducing power consumption is of high relevance. In the final system a NoC will be developed connecting all the key components. When running the coherence protocol, the NoC will be in charge of transmitting large amounts of data, mainly memory blocks between memory resources (L1s, L2s, directory structures, memory controllers). Data transferred in the NoC can consume a significant percentage of the power of the system, more significant as the system will increase in size.

In several previous works, different data compression mechanisms are proposed and analyzed. In those works, the impact of NoC traffic in system power and execution overhead can be significantly reduced. In this deliverable we present a detailed analysis of the traffic generated by the analyzed applications over the NoC, in the context of the vIrtical project. The main aim is to identify compression opportunities so to select the best mechanisms that will optimize power reduction when focusing on the data traffic over the NoC. It is important to note that we focus on

NoC-data compression opportunities and not on storage compression opportunities. Thus, data is expected to be compressed and uncompressed when entering and leaving the NoC transmission respectively.

The remainder of this section is organized as follows: first we will give an overview of the capturing methodology used to obtain the data analyzed; secondly we will focus first on sharing patterns and later on, on compression opportunities detailing specific methodology particularities and showing our analysis results; finally some conclusions will be derived of the previous analysis and directions for next year research within the project for compression and coherence protocols will be given.

4.1. Analysis Methodology

Both sharing patterns and NoC-data compression opportunities must be analyzed under the same premises and thus they must share the applied methodology and tools. However, they also differ in several aspects. An analysis framework has been built in any case to make these analyses possible. In this section we show the main characteristics of such framework.

The steps followed to analyze both issues are similar as can be seen in ***Fig. 11***. First of all, in both cases, we trace memory accesses requested by the cores when running real applications. For the NoC-data compression analysis we perform a second step aimed at simulating the transfer over the NoC of data blocks included in the previously traced memory accesses over the NoC. Finally in both cases, we obtain statistics for the previously obtained traces.

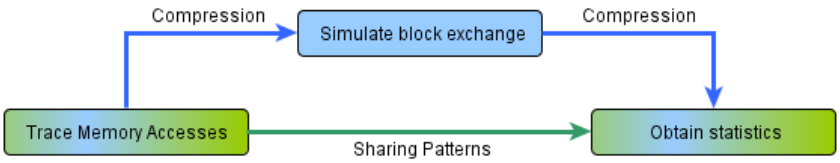


Fig. 11: Analysis Methodology Scheme.

4.1.1. Target System

The target system considered in the project is a heterogeneous multiprocessing system with a NoC connecting the main processing units, a GPPA to enhance performance and the main memory. The ARM architecture is defined as the target processing host, more precisely ARM's big.LITTLE architecture. It consists of one dual-core Cortex-A15 MPCore and one dual-core Cortex-A7 MPCore connected using the ARM CoreLink CCI-400 as described in Deliverable 1.2.

We use a simplified version made of a quad-core Cortex-A15 MPCore. The trade-off between complexity and accuracy makes this option more suitable. Conclusions obtained with the quad-core Cortex-A15 MPCore can be extrapolated to its big.LITTLE counterpart with acceptable precision. Furthermore, big.LITTLE has two different working modes, either only Cortex-A15 or

Cortex-A7 processors are awake or they are both working at the same time. When all processors are working at the same time they will not run a parallel application in both of them because they work at different frequencies. Since we are mainly concerned on parallel applications, the use of big.LITTLE is not mandatory.

Other characteristics of the simulated system for compression opportunities are:

- NoC with a 4x4 2D mesh where each core has been simulated as an individual node. In the target system the whole quad Cortex-A15 will be only one node, but in order to model the behavior of caches separating cores was a more suitable option.
- 4 L1 caches (one per core) each with 128 sets, 4 ways and a line size of 64 bytes.
- 1 L2 cache (shared by all the cores) with 512 sets, 16 ways and a line size of 64 bytes. Caches are inclusive (L1 caches' content is included in L2).
- 4 memory controllers located at the corners of the NoC.
- As for messages we have: control messages are 8 bytes long and data message are 72 bytes long; flit size is 4 bytes.
- The switching mechanism is virtual cut-through and the flow control is Stop&Go. The crossbar is allocated at packet level and supports collective communication (although at the moment no broadcast or multicast is being applied).

In ***Fig. 12*** we can see the simulation target system, with the ARM Cortex-A15, the GPPA and other potential components connected by the NoC. (The figure does not show the 4x4 mesh, instead it shows an irregular topology).

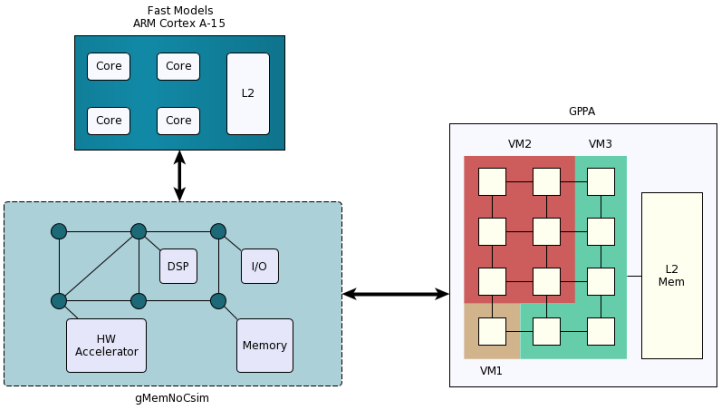


Fig. 12: Simulation target system

4.1.2. Simulation Tools

We use ARM FastModels together with our gMemNoCsim tool. Next we describe how these tools are used to model the target system.

ARM FastModels simulator provides out of the box programmer's view models of the ARM processors. It is thus both functionally accurate and easy to use since ARM processors models are already implemented as an Instruction Set Simulator. We use this simulator to model the quad-core Cortex-A15 MPCore part of the target system and to run on top of this the targeted applications.

The model of Cortex-A15 provided with FastModels is capable of running basic applications, but it does not cover all the requirements of an operating system, which is needed to evaluate and benchmark parallel applications. We thus use a more complex model also provided with FastModels (namely RTSM-VE Cortex-A15) that allows the simulation of both operating systems and applications. In this RTSM-VE model, as seen on ***Fig. 13***, the cores are connected directly to a Versatile Express platform through a 64-bits AXI bus. This platform includes the Motherboard Express uATX, which has been especially designed to support future generations of ARM processors, and the CoreTile Express daughterboard with the on-board DDR2 SDRAM. The motherboard provides the following features:

- Peripherals for multimedia or networking environments.
- All motherboard peripherals and functions are accessed through a static memory bus to simplify access from daughterboards.
- Consistent memory maps with different processor daughterboards simplify software development and porting.
- Supports FPGA and processor daughterboards to provide custom peripherals, or early access to processor designs, or production test chips.

The daughterboard contains the Cortex-A15 ARM processor model.

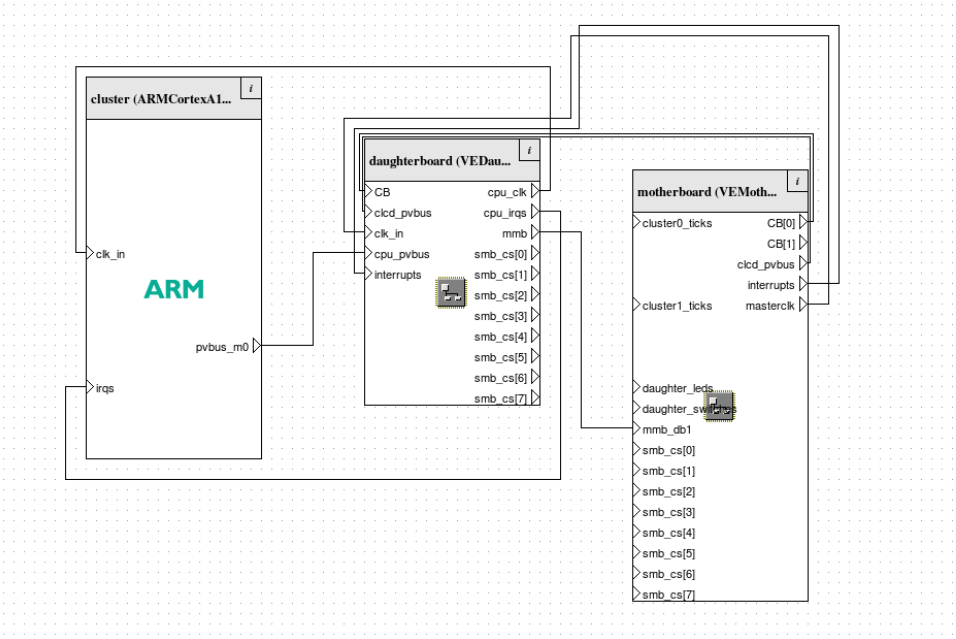


Fig. 13: ARM FastModels RTSM-VE model.

On the other hand, gMemNoCsim is a cycle accurate event-driven NoC simulator developed at UPV by the parallel architectures group (GAP), which allows high precision in modeling the key elements of NoCs, including all the key design aspects: as topologies, router, routing algorithms, schedulers and flow control mechanisms. In its current version, gMemNoCsim also models cache coherence protocols on top of the NoC.

gMemNoCsim has the capability of working with synthetic traffic, with real traces or it can be fed on-line with the output of a different simulator. In our case we feed gMemNoCsim with the memory access traces obtained by running the applications on Fast Models. Thus, gMemNoCsim simulates the memory hierarchy (converting memory accesses in memory block accesses flying between L2 cache and main memory over the NoC). This will provide the data needed to analyze compression opportunities.

gMemNoCsim has been enhanced, within the project, to add support for actual data exchange between all levels of the memory hierarchy. Also, tracing capability of gMemNoCsim has been provided to cope with the needs of compression analysis.

4.1.3. Trace Acquisition Methodology

The information that we expect to find on the trace files differs, depending on the focus of the analysis, whether the analysis is made in search of compression opportunities or for sharing pattern analysis. For sharing pattern analysis we need to capture all memory accesses from the cores. In turn, for memory compression opportunities, as communication between L1 and L2 is managed internally by ARM processors, we need to capture all communication between L2 and main memory. In the later we also need to know the memory contents in order to analyze the data transferred.

For sharing pattern analysis our point of interest is the parallel section of the applications, namely the section executed by several cores at the same time. To identify this section we explored the application code looking for the starting point and end point of threads. This was identified with THALES support. To delimit this section and make it recognizable by FastModels we introduce a special *nop* instruction on the application, which is available on the ARM Instruction Set. Therefore, tracing is started when the reserved instruction is fetched and a subsequent appearance of such an instruction determines that the tracing must finish.

On the other hand, compression involves not only shared data but also private data, so limiting the trace acquisition to the parallel section of the applications is no longer mandatory. Nevertheless, beyond that consideration, the traces obtained for sharing pattern analysis are suitable for compression analysis as well, so no different set of traces has been obtained.

FastModels supports the use of a Model Trace Interface (MTI) plug-in that permits us to consistently track the execution of the model. Through implementing an MTI plug-in for tracing memory accesses produced by the cores and adding it to the simulation we are able to trace exactly what we needed in the form that we required. The ARM simulator offers other alternative tracing methods but they would either modify the system behavior when using RTSM-VE model or be prohibitively time consuming.

MTI plug-in provides many different sources to trace, but the more verbose the trace obtained is and the more sources are involved, the more it slows down the simulation. Since it takes billions of instructions to boot a Linux system on FastModels, we need to deactivate the output and minimize the number of sources of the tracing until the starting point of the segment of interest

is detected. We have achieved an acceptable compromise solution by capturing only the instructions fetched by the cores until we reach the aforementioned special *nop*, and subsequently tracing loads, stores, and fetches until we get to the ending special *nop*.

The ARM Simulator provides programmer's view models with some limitations. On system simulators there is a trade-off between speed and accuracy: very accurate simulators lack in speed whilst fast simulators cannot be totally accurate. FastModels in particular opts for the execution speed thus lacking some features needed for our analysis, such as:

- Instruction timing: A processor issues a set of instructions (a.k.a a quantum) at the same point on the simulation time, and then waits some amount of time before executing the next quantum, being impossible to determine the right time each individual instruction is executed.
- Bus traffic: bus traffic has several optimizations that make it inaccurate. We were able to deactivate some of those optimizations but not all of them, at the expense of making the simulation far slower and still not accurate to the level required.
- It does not support out-of-order execution and write-buffers as architecturally defined: execution on FastModels is only an approximation to execution of architecture and it must be thus considered.

As mentioned above, traces must be obtained in the exchange of L2 and main memory to be used for compression opportunities analysis, including memory contents. Since we could not find a proper MTI source to trace at this point, we used the traces obtained from FastModels and feed them to gMemNoCsim. gMemNoCsim reproduces the communication between all different levels of the memory hierarchy translating loads and stores into coherent requests to main memory. In turn, main memory contents were obtained in FastModels through the use of CADI debug interface when the starting trace point was detected.

Finally, we describe the trace and memory files format. In the case of FastModels, we need to obtain traces with the information required for coherency modeling, including the core id (to characterize the number of sharers of the block), the address (to identify the block being accessed), the type of access (to classify the block as data or instructions and to discriminate writes from reads) and the data (for compression analysis). Traces obtained using gMemNoCsim must include address (to analyze whether or not address compression can be an interesting technique), and data (the entire block contents to analyze compression opportunities). Source and destination have been included for readability.

Traces obtained from ARM FastModels using the MTI plug-in is as follows:

```
core,address,type,data
0,001ea10c,l,b6f6713c
0,b6f6713c,f,e92d001f
0,bef1de68,s,0020787c
0,bef1de10,bs,bef1eef4,bef1ef10,bef1f984,001a0c9c,00000003
```

Where type refers to: l (load), s (store), f (fetch), and b applied to l or s (burst load-store). When a burst is detected the data field is extended to the total amount of data exchanged. Both addresses and data are coded in hexadecimal format.

Traces obtained from gMemNoCsim traces are as follows:

Address origin destination data_block
0x00207840 Memory L2Cache[0][481][0] 2030,69,0,0,1,88,0,0,0,0,52,3,1,4,0
0xdf04a040 L2Cache[0][129][9] Memory 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Where origin and destination can be either Memory or L2, including as indexes the node (in case several L2 are present), index and way. Addresses are coded in hexadecimal format and data is coded in decimal format.

Memory content obtained from FastModels is as follows:

Data
00000000
00000000
03e03dbf
801000c0
00000000

Memory is organized as a vector of words, which can be scanned translating address to line number. To keep a backup for subsequent executions, a copy of the memory file is made when starting to simulate the model on gMemNoCsim and all memory exchanges are made on the copy.

4.1.4. Applications

We have obtained traces with five of the many different algorithms in the OpenSSL suite, three hash algorithms (SHA1, SHA256 and SHA512), and two encryption algorithms (AES-128-ECB and AES-256-ECB). We chose these algorithms since they are the most relevant ones according to THALES point of view.

4.2. Sharing patterns analysis

As commented above, there exists several recent proposes that take advantage of memory block classification for different purposes, such as enhancing efficiency of directory caches, reducing coherence overhead or better taking advantage of NUCA caches. All of them are mainly based on the classification of blocks in private (P) and shared (S). Moreover, some others extend this classification to read (R) only and written (W).

So in this deliverable we propose to analyze blocks classifying them according to:

- PR (Private Read-only): Only one processor accesses the block. All accesses are loads. Thus, the block is private to the core and only that core reads the block but does not write it.
- PW (Private read-Write): Only one processor accesses the block. At least one access is a store. Thus, the block is private to the core and this core reads and writes that block.
- SR (Shared Read-only): At least two processors access the block. All accesses are loads. Thus, the block is shared by several cores but no one writes on that block.

- SW (Shared read-Write): At least two processors access the block. At least one access is a store. This is the most interesting mode as it requires coherence protocol support. In this mode the block is shared and is written by at least one core.

Considering this classification, the only blocks that actually need coherence maintenance are the SW ones and therefore we can take advantage of the fact that the remaining blocks do not need it, either because they are accessed by just one core or because they are only read by any number of cores. So special attention will be paid to SW blocks.

The classification schemes proposed in the literature have used different granularities: blocks (**[Hos11]** and **[Pug10]**) and pages (OS-based schemes, as can be seen in **[Cues11]**, **[Har09]** and **[Kim10]**), looking for a trade-off between detection accuracy and the required overhead. So our analysis is made with three different granularities based on blocks and pages as architecturally defined on ARM documentation: 64 bytes block, 4 Kbytes pages, and 64 Kbytes pages. This is interesting since coherency at such a level is easier to implement and manage. Working at page level also allows us to rely on the operating system to detect whether coherence needs to be applied or not, aiding to reduce the hardware overhead and complexity. On the other hand, the use of page level granularity allows us to analyze how critical is the block misclassification introduced with coarser grains. In addition, studies at page level granularity are intended to identify the viability of applying cache coherency at page level instead of block level.

Basically, we provide two kind of analysis. The first one, referred to as static analysis, is intended to counting the number of blocks included in each category. The second one, referred to as dynamic analysis, shows the number of accesses to blocks for each category. Both views complement each other and allow us to identify which categories of blocks are the most frequent and which ones are the most accessed.

The previous classification will help us later to identify which are the best optimization opportunities when developing the appropriate coherence protocol. As examples, in many works the producer-consumer access pattern (where one or more cores repeatedly write a certain core, which is read only by others cores different from the writer ones) could benefit from update-based protocols. In contrast, with invalidation-based protocols, this pattern induces a large latency penalty. Also, blocks with no write operations on them could benefit from a deactivation of the coherence protocol. In the next subsection we show the obtained results.

4.2.1. Analysis Results

All results shown are for the parallel phases of applications. The results presented are focused on the data blocks or pages. As can be seen on the ***Fig. 14*** and ***Fig. 15*** the proportion of data blocks and accesses is more relevant. Also, due to the fact that most of the instruction blocks are shared through all four cores, the classification between Private and Shared instruction blocks is less interesting in this case. Finally, we detected no interleaving between data and instruction blocks at different page granularities.

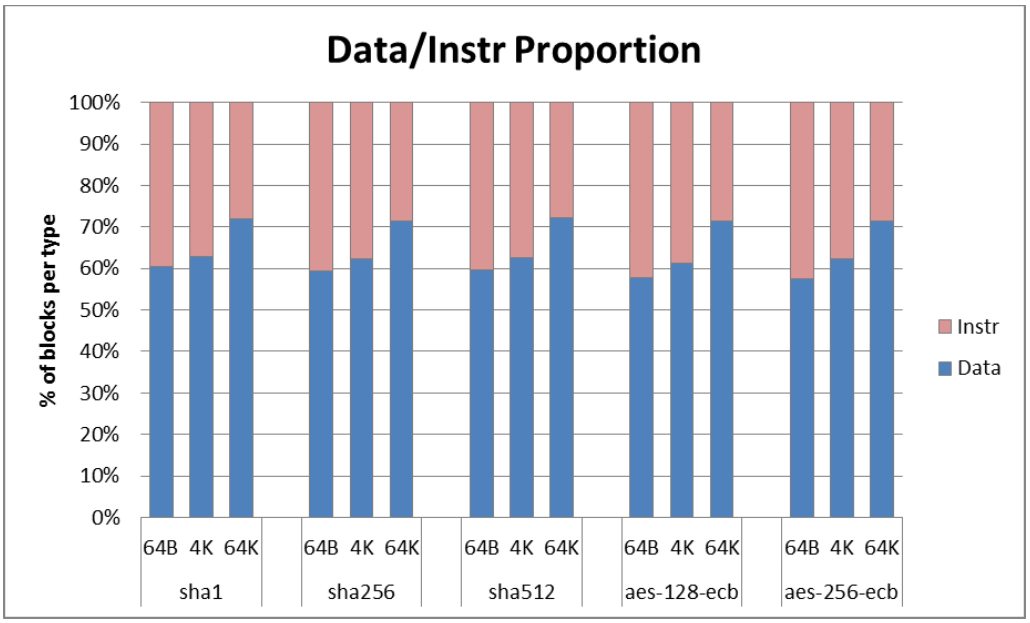


Fig. 14: Proportion of blocks distinguishing whether they are instruction or data blocks

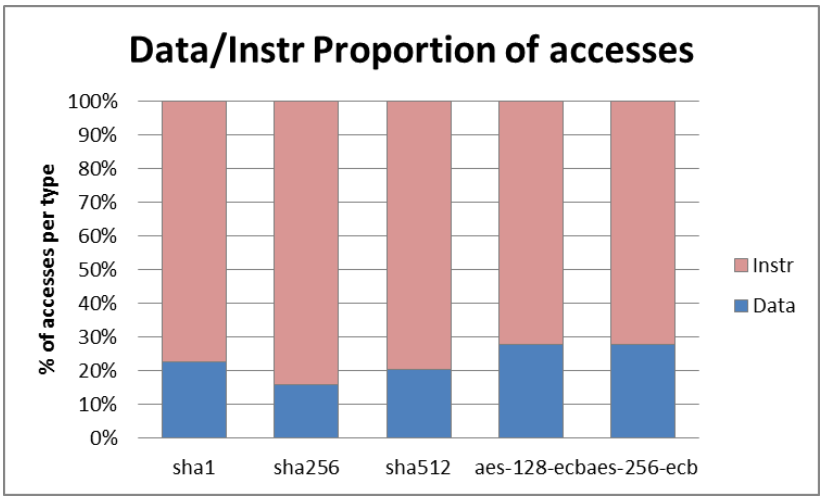


Fig. 15: Proportion of accesses to data / instruction blocks

As commented before, the analysis will basically consist in counting both the number of blocks belonging to each of the classes defined above (static analysis) and the number of accesses realized on each of the block classes (dynamic analysis). This analysis is carried out considering different granularities for data block classification, from block size until pages of different sizes. In all figures, the results obtained for each of the analyzed applications, together with the resulting average values, are displayed.

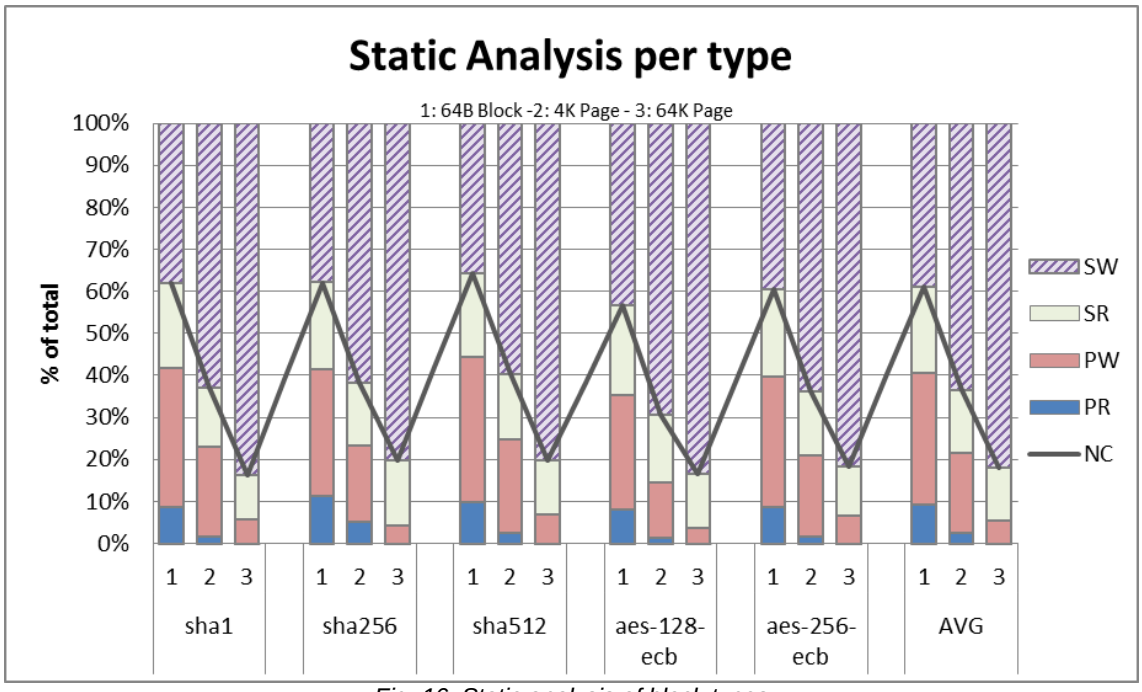


Fig. 16: Static analysis of block types

Fig. 16 shows the block classification based on the detection of the Private-Shared Read-Write scheme for every block requested at the different granularities aforementioned. First of all, it is observed that, on average, 40% of data blocks are private (PR or PW). The remaining blocks are shared (60%), but notice that indeed only 40% of data blocks are SW, that is, they require coherence maintenance. However, this promising result vanishes as long as the granularity used for classifying the blocks is increased. As can be observed, the coarser grain used, the more shared and written blocks are found. In particular, for 4KB pages, the percentage of SW blocks is greater than 60%, whereas this percentage, on average, exceeds the 80% for 64KB pages. This means that SW blocks are concentrated in a certain number of pages, but they are mostly distributed among them. As a consequence, the detection accuracy decreases as far as the granularity is increased in order to simplify the detection process, leading to a misclassification of page blocks. Notice that just one SW block contained in a page will cause the page to be classified as SW.

Fig. 17 shows the dynamic analysis. It is observed that despite the fact that SW blocks just represent 40% of the total number of blocks, as was shown above, they agglutinate tough the larger number of accesses (60% on average). Further, the number of accesses to private data blocks is indeed negligible. Unlike the static analysis, here it is observed larger differences between applications. Also, the number of accesses classified as SW hardly increases as the detection granularity becomes greater. On average, it reaches a 70% for 64KB pages. Notice that this is an expected result as long as the highest percentage of accesses inside a page is destined to SW blocks. Given that most of data memory accesses require coherence maintenance, the design of the cache coherence strategy will be a key element to provide high performance.

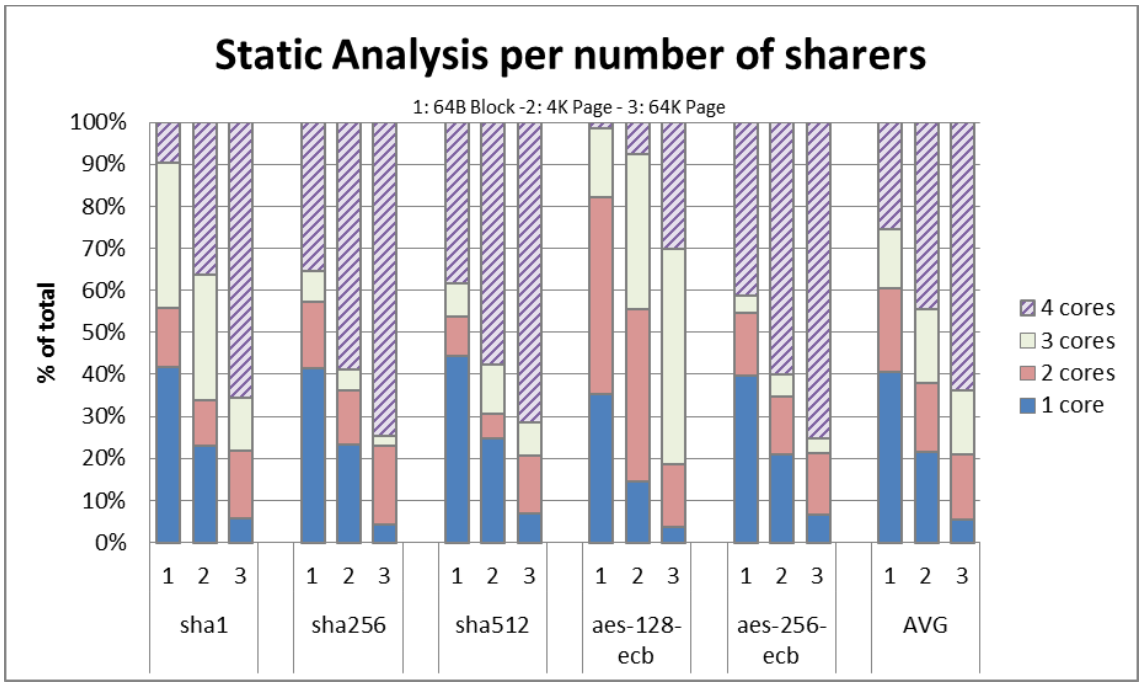


Fig. 17: Static analysis of blocks per number of sharers

In order to offer a deeper insight into the sharing degree of data blocks, let us analyze to what extent they are shared, that is, how many cores share each of these data blocks. So, in ***Fig. 18*** and ***Fig. 19*** we analyze the number of sharers per block. If there are no sharers and the block is only accessed by one core, it corresponds with a Private block detected in the previous analysis. We consider also both static (***Fig. 18***) and dynamic analysis (***Fig. 19***). In ***Fig. 18***, we can observe how, on average, just two cores share about 20% of blocks, 15% are shared by three cores, and 25% of them are shared by four cores. As the detection granularity increases, the number of blocks shared by all the cores is larger. The reason is the same as that pointed out with respect to ***Fig. 16***. Moreover, from ***Fig. 19***, it is observed that the most accessed data blocks are those shared by all four cores. This result corroborates even more the importance of carrying out an appropriate design of the cache coherence mechanism as far as a large number of cores are usually involved in the coherence maintenance of the data blocks.

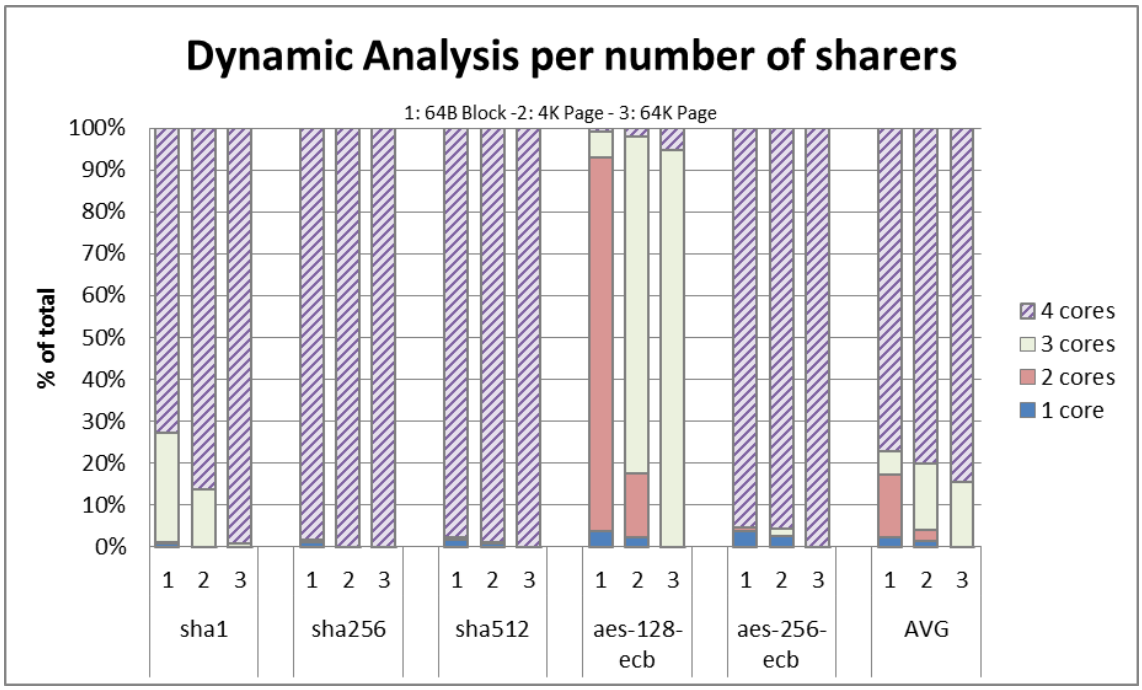


Fig. 18: Dynamic Analysis of blocks per number of sharers

Until now, the analysis has been focused on classifying block by using different detection granularities. However, it is also interesting to just classify pages. It may be important to assess the convenience of managing cache coherence in a per page basis instead of the usual strategies based on block tracking. In this case, page classification in PR, PW, SR, and SW classes is as follows. A page is classified as SW when it contains at least a SW block. Otherwise, it will be classified as SR if at least one of their blocks is SR. On the contrary, if the page does not contain neither SW nor SR blocks, it will be classified as PW if at least it contains a PW block. Otherwise, the page will be classified as PR. In this sense, ***Fig. 20*** shows the page classification for 4KB and 64KB page sizes. As can be seen, more than 35% of the 4KB pages require coherence (they are SW), whereas this percentage increases until near 50% for 64KB pages. This means that SW blocks are not spread over all the pages, but they are indeed distributed between a limited number of pages, so much larger, the larger the page size is.

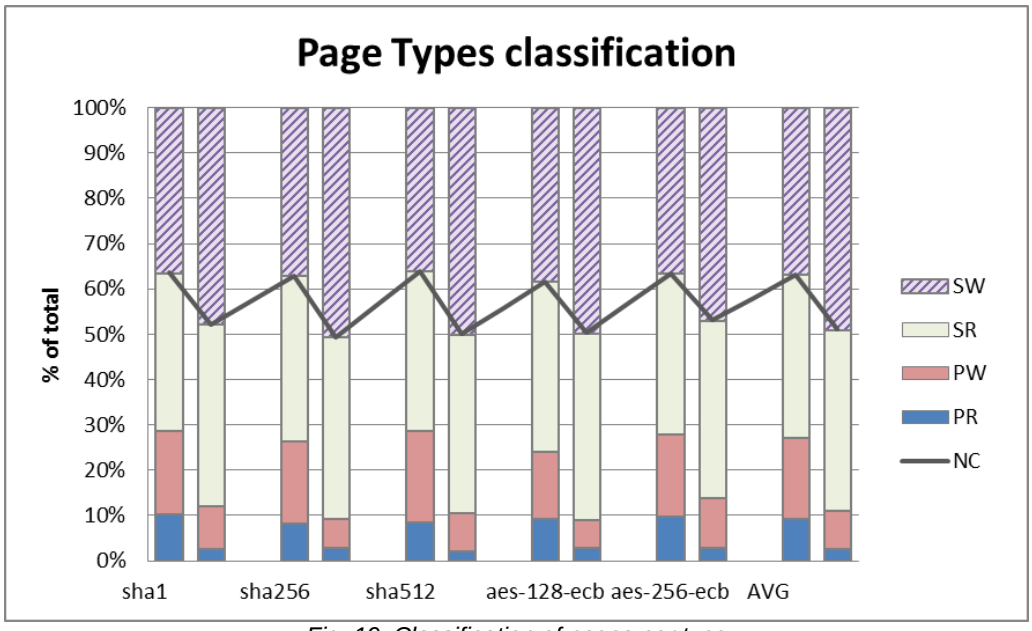


Fig. 19: Classification of pages per type

In order to offer a deeper insight into SW pages, from now on, SW pages become the focus of our analysis. Firstly in ***Fig. 20*** we classify pages per access type and subsequently in ***Fig. 21*** we discern the number of sharers on pages classified as SW. As can be seen, like it was observed on analyzing block sharing, most pages are shared among three or more cores, as more as larger the page size is. In particular, half of the blocks are shared between 4 cores for 4KB pages, whereas the percentage exceeds the 60% when page size is 64KB.

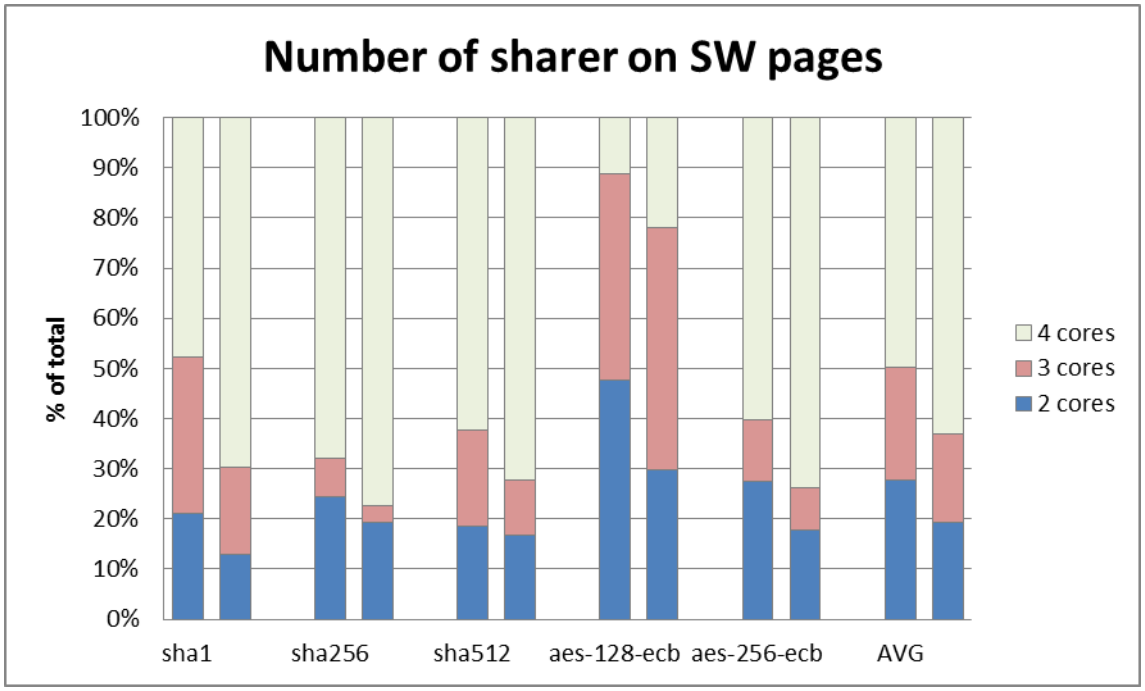


Fig. 20: Classification of SW pages per number of sharers

As commented before, the main disadvantage of using page granularity to detect blocks requiring coherence is the misclassification that page blocks may suffer. Notice that a single block can determine the classification of the rest on the same page. To analyze this effect, in

what follows, we study in depth the internal anatomy of SW pages. The study is performed by considering the type that each block in the page had been assigned in case of having assumed block granularity in the detection process. In particular, we proceed to count the number of each of the block types contained in the page. Information is represented making use of box-and-

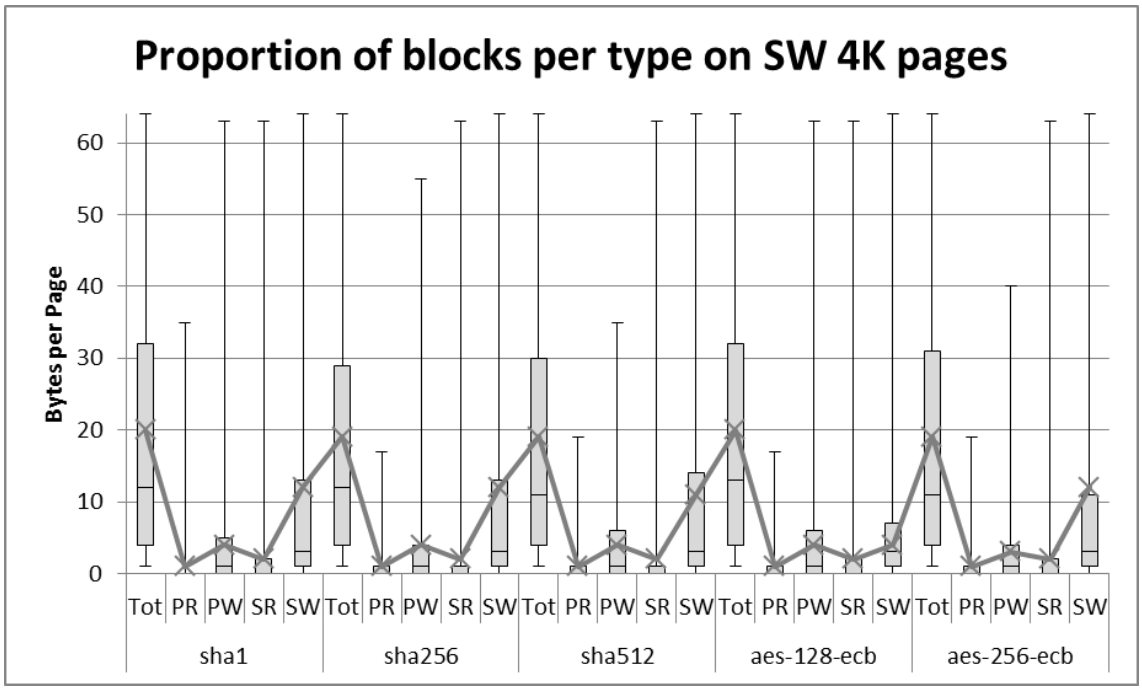


Fig. 21: Proportion of blocks per type on SW 4K pages

whiskers plots. This will help to determine how populated the pages are and the real significance of block misclassification.

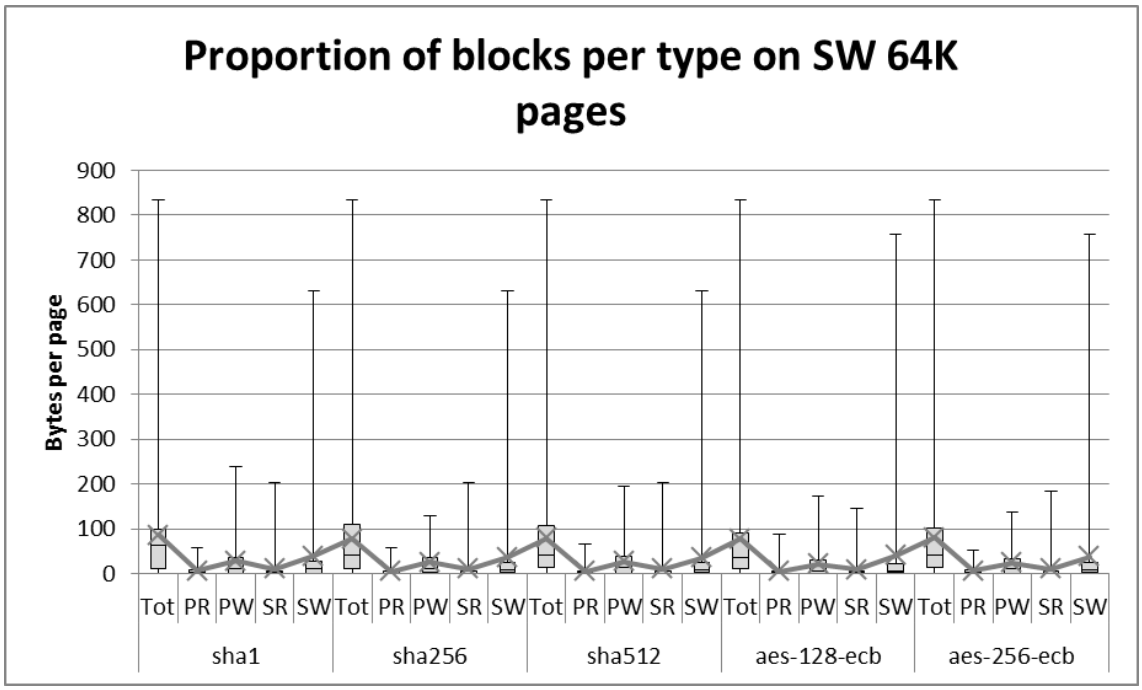


Fig. 22: Proportion of blocks per type on SW 64K pages

Fig. 22 and ***Fig. 23*** show results for 4KB and 64KB page sizes, respectively. First of all, it is observed that pages are hardly populated, so much less, the larger the page size is. However, a great variability is observed, from pages hardly containing a few blocks until pages crowed of blocks. On average, the medium value of blocks per page is about 20 out of 64 for 4KB pages and 80 out of 1024 for 64KB pages. Anyway, the precise distribution of the total number of blocks and the number of blocks of each type can be observed in the aforementioned figures. Regarding SW blocks, it can be observed that, for a page size of 4KB, most of the blocks in SW pages are SW blocks, but when the page size is increased, the PW blocks become more frequent. Despite this, the number of SW blocks present in the page is very small in relative terms (on average, the 75% of 4KB pages have less than 12 SW blocks, whereas the 75% of 64KB pages have less than 25 SW blocks). These results may suggest the possibility of applying fine grain detection techniques inside SW pages in order to isolate true SW blocks, so limiting coherence maintenance actions to them.

We have also performed a dynamic analysis of SW pages in order to obtain the proportion of store access (see ***Fig. 24***). As can be seen in ***Fig. 24***, around 30% of the memory accesses to SW pages are stores. Obviously, these accesses will be destined to either SW or PW blocks.

Fig. 23: Proportion of store accesses on SW pages

This kind of studies allow us to assess the convenience of applying update strategies instead of invalidate ones.

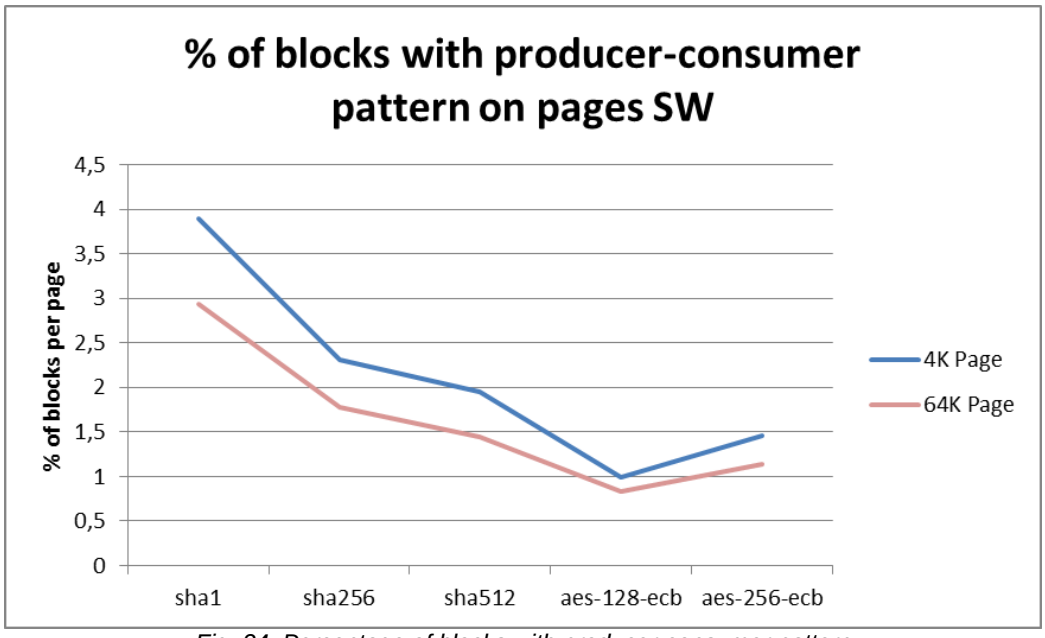


Fig. 24: Percentage of blocks with producer-consumer pattern

Finally, it is carried out an analysis in search of the existence of producer-consumer patterns in SW pages. ***Fig. 25*** shows the proportion of blocks presenting the producer-consumer pattern related to the amount of blocks requested within the page for different page granularities. As observed this percentage is relatively small.

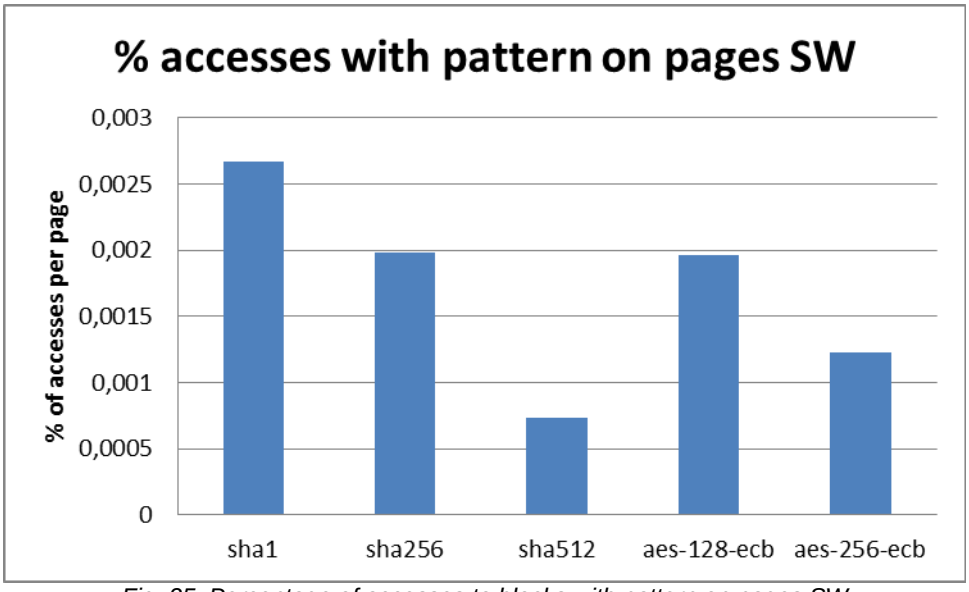


Fig. 25: Percentage of accesses to blocks with pattern on pages SW

In order to definitively observe whether or not the producer-consumer pattern is relevant on the analysis, we studied the percentage of accesses done to blocks presenting the pattern within the SW pages. As can be seen on ***Fig. 26*** not even the 0,003% of the accesses are done to these blocks.

4.2.1.1. Conclusions for sharing patterns analysis

On average, about 75% of the blocks accesses and from 35% to 40% of blocks/pages (depending on the granularity) on the analyzed applications correspond to instruction blocks, which, unlike data blocks, do not require coherence maintenance. Moreover, despite the fact that it is detected a high sharing degree of data blocks among cores, indeed only SW blocks (just 40% of the total number of data blocks) require coherence. Considering accesses, SW blocks agglutinate 60%, on average, of the total number of accesses. That means that if snoopy-based coherence is used, the broadcast traffic reduction will be around 40%, while if directory-based coherence is used, then the benefits will increase.

In order to ease data classification, different page granularity degrees can be used at the expense of causing a loss of accuracy due to block misclassification. We observed that in the case of block counting there is a high misclassification degree. As a counterpart, in the number of accesses the degradation of classifying block or pages is lower.

Also, it is observed that SW blocks are not spread over all the pages, but they are indeed distributed between a limited number of pages. In particular, less than 40% of data pages require coherence. Deeping inside SW pages, we observe that they hardly are populated, containing about 30% and 10% of blocks, on average, for 4KB and 64KB pages, respectively. Among them, the number of SW blocks is the majority. Furthermore, about 30% of the accesses to blocks of SW pages correspond to store operations.

Finally, it is observed that the impact of the producer-consumer sharing pattern in the analyzed applications is negligible. Less than 1,5% of the blocks accessed in SW pages present such a behavior, which discourages the application of update policies for coherence purposes in an extensive way.

Further discussion must be done in order to decide which can be the best options among all the coherence protocols in the literature, taking into account also the characteristics of the target system of the study.

As aforementioned, the impact of producer-consumer pattern is insignificant and therefore an invalidation-based protocol would be the best choice, which also will reduce the amount of cache-to-cache traffic and consequently reduce the amount of cache bandwidth and energy consumption.

As seen on the analysis results, we found that for page granularities most blocks and accesses are classified as SW and, therefore, it may not be suitable a technique based on coherence deactivation at page level. However, with block granularity it is observed that 60% of blocks do not require coherence. Therefore, a fine grain detection inside SW pages may be interesting, at the expense of introducing additional hardware support. That means that in the possible case of applying of coherence deactivation techniques would be convenient to meet a suitable trade-off between accuracy and introduced overhead.

The main benefits obtained through the use of these techniques are related to the energy reduction deduced from the lowering in coherence messages. It also improves the execution time of the system.

On the other hand, other techniques more focused on snoopy-based protocols and interconnection networks can also be applied. Most filtering techniques are based in reducing energy consumption by means of filtering unnecessary snoop traffic. Also, this kind of proposals may take benefit of the low misclassification degree introduced for data accesses, therefore using a technique classifying blocks at page level and relying on the OS support to do so.

4.3. Compression Opportunities

In this section we analyze the compression opportunities for the target system in the project and the provided applications. First we briefly describe the main techniques proposed so far which are the ones that are the target for the project (you can find more information about them on classical literature such as **[Sal04]**):

- Zero elimination: this kind of techniques compresses code or data by eliminating long strings of zeros, either by codifying them or by just removing them. From the many techniques that fall into this group those that remove zeros seemed the most interesting to us. In order to be able to eliminate part of a message without further coding, this part must be aligned (word aligned, block aligned,... depending on the particular technique).
- LZn: the basic idea of this method is to use part of the input stream as a dictionary, maintaining a sliding window divided in two parts: the search buffer (left, already coded) and the look-ahead buffer (right, to be read). When a match between both buffers is found, a pointer to the occurrence in the left buffer is created and sent instead of the occurrence in the right buffer.
- Table-based compression techniques: this kind of techniques creates tables with highly frequent patterns and avoids sending the whole pattern by sending the index in the table instead. There are many different techniques according to different philosophies, but they don't require compressible data to be aligned or its occurrences to be consecutive.

As we have seen, there are many different compression techniques and they imply different data analysis. In order to be able to choose the compression technique that best suits our particular case of study, we have obtained a variety of data statistics not to constraint the final decision derived from applying our analysis method. We have thus used three analysis strategies to find repeating patterns:

- Aligned Consecutively Repeating Patterns (ACRP). Patterns in this category are consecutive and aligned to byte size. One example is “000000010000000100000001…” where the pattern “00000001” is repeated three times (consecutive repetitions) and is byte aligned.

- non-Aligned Consecutively Repeating Patterns (nACRP).. This category groups those patterns that are still consecutive but not necessarily aligned to any data size. One example is “11110000000100000000100000001…” where the pattern “00000001” is repeated three times (consecutive repetitions) and is not byte aligned.
- non-Aligned non-Consecutively Repeating Patterns (nAnCRP). This category groups those patterns not necessarily aligned to any data size and not consecutive, this is: a pattern that is repeated along the trace but we do not consider if it is consecutively repeated or not. One example is “1111011110010000001111100000111111111…”, where the pattern “1111” is repeated 5 times (not necessarily consecutive) and alignment is not considered.

Notice that the ACRP category helps us in localizing the occurrence of long strings of aligned zeros (for zero elimination case). However, in order not to lose generality, we localize also all patterns that have the properties referred above. In this case, the nACRP category is also relevant, because it allows us to study the suitability of LZn techniques. In turn the nAnCRP category allows us to assess the convenience of applying table-based compression techniques.

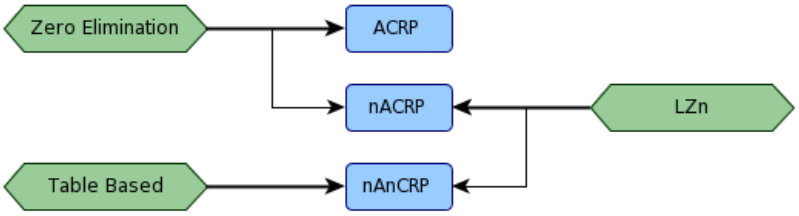


Fig. 26: Relationship between studies made and compression techniques

The relationships between compression techniques and analyzed pattern categories are shown in ***Fig. 27***.

4.3.1. Analysis Results

First, we analyze in detail, in the three following subsections, the application OpenSSL with the algorithm sha1; and in the forth subsection we compare the obtained results against the other 4 algorithms. Finally, in the fifth subsection, some conclusions are drawn. In all subsequent analysis, pattern size is one byte, block size is 512 bits (64 bytes) and, when applicable, byte alignment is used.

4.3.1.1. Aligned consecutively repeated patterns (ACRP)

We first present a graph with general information with the number of occurrences of ACRP patterns. ***Fig. 28*** shows the results where X axis is the number of consecutive repetitions and Y axis (left) is the total number of occurrences. Notice that Y has a logarithmic scale. The figure also shows the variability (right Y axis) in the number of total occurrences among the different patterns that have the same amount of consecutive repetitions. When no red line is present, either only one pattern exists for that category or all included patterns have the same number of

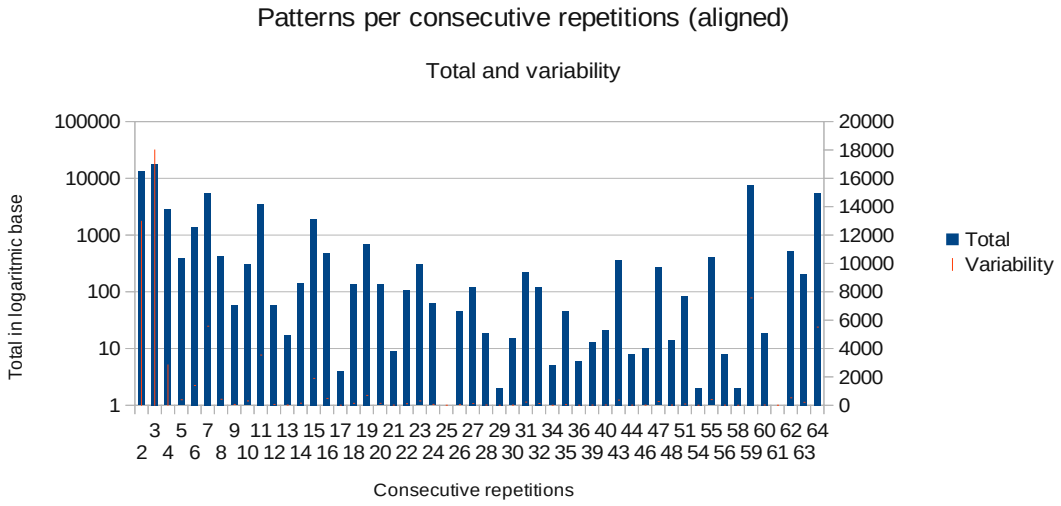


Fig. 27: ARP total occurrences per number of consecutive repetitions.

The first thing to notice is the large disparity in the number of occurrences of patterns (notice the logarithmic scale). Large occurrences of ACRP patterns occur either with low repetitions or with large number of repetitions. This figure demonstrates that ACRP patterns exist and thus, there is an opportunity to compress those patterns.

In ***Fig. 29*** we can see the most interesting (most frequent) ACRP patterns found. We select those ACRP patterns with at least 1000 total repetitions. The number of total repetitions is obtained by performing the product of consecutive repetitions by occurrences. The bubble size represents the proportion of total data constituted by this pattern. One first outcome is the fact that the set of interesting ACRP patterns represents a large amount of traffic (92.86% of traffic) with only a small number of patterns. The rest of detected patterns represents 0.54%, conformed by many different patterns repeated only a few times. This means a large potential exists to compress data traffic. Most interesting is the fact that the largest part of ACRP patterns is made of all-zero streams. Also, the largest set corresponds to long streams of zeros, those represented by the two largest bubbles at the upper right side of the figure.

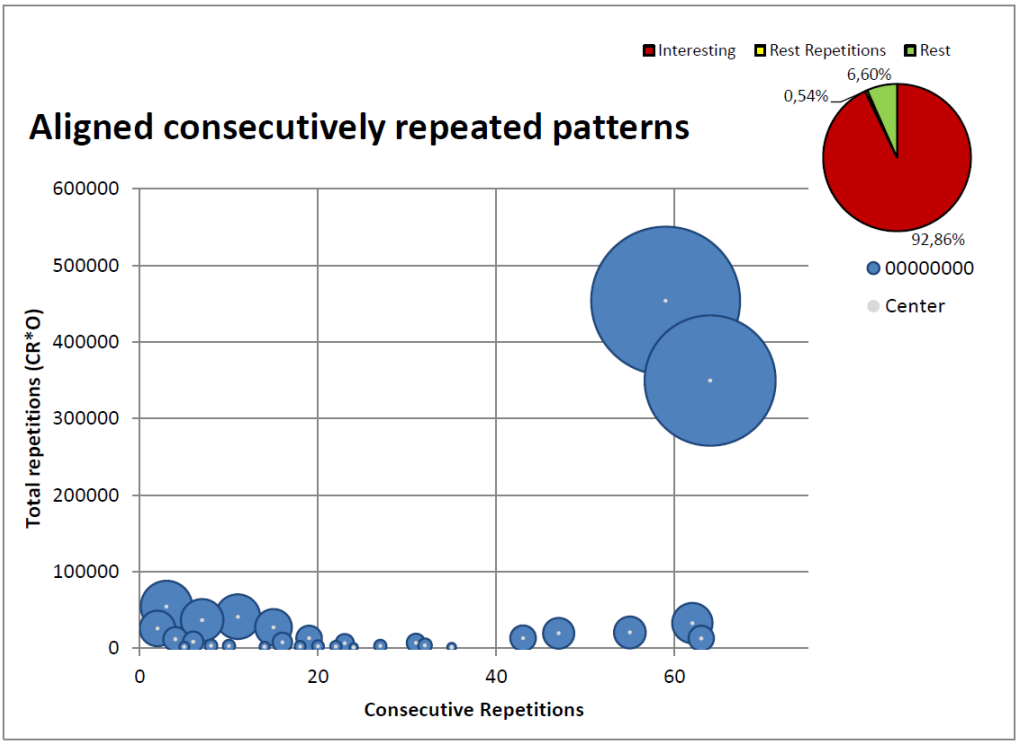


Fig. 28: Non aligned consecutively repeated patterns

Since we noticed in the previous graph that most compressible data is composed by long zero strings, we now analyze other alignments of zeros. In this case we care about the block size, as will be the default transmission data size (blocks between caches). Notice that we will compress within individual messages and not between data stretched over several blocks/messages. ***Fig. 30*** shows the amount of ACRP all-zero patterns for different string sizes (quarters of a block) and different alignments (not-block aligned, aligned to block, aligned to end of a block). As can be seen, the vast majority of ACRP repetitions correspond to large strings (either 48 or 64 bytes in length) and they are practically always block aligned. This will simplify the compression mechanism to be used in the final system.

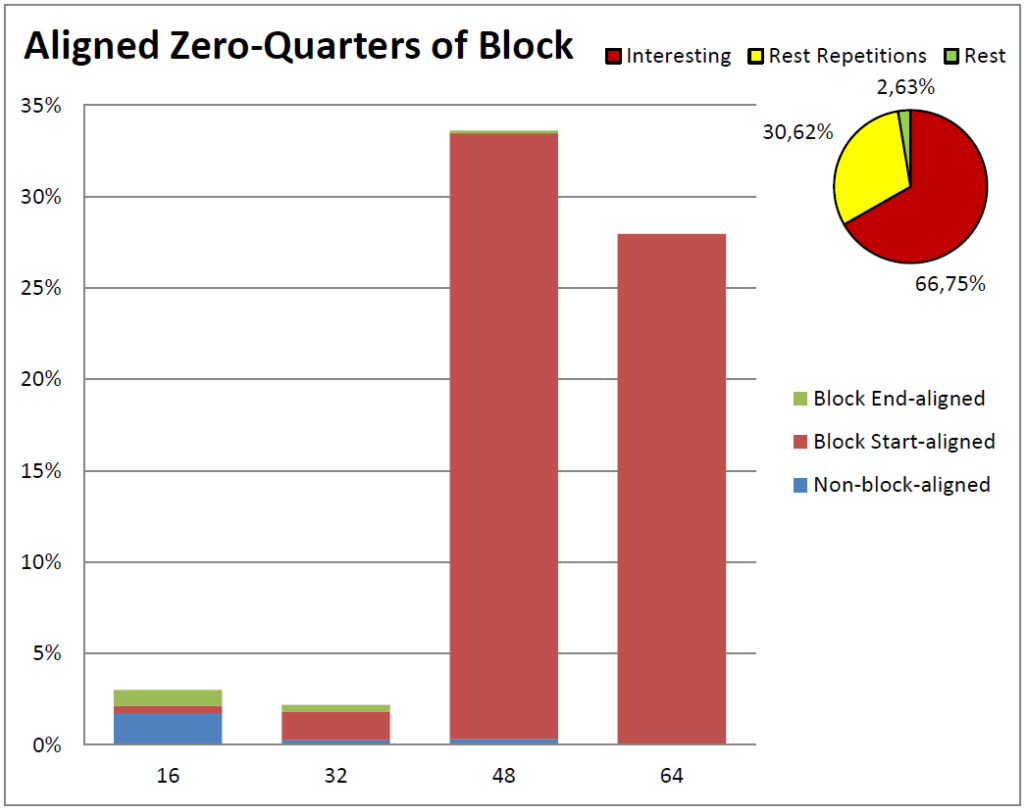


Fig. 29: ACRP quarters of blocks of aligned zero strings

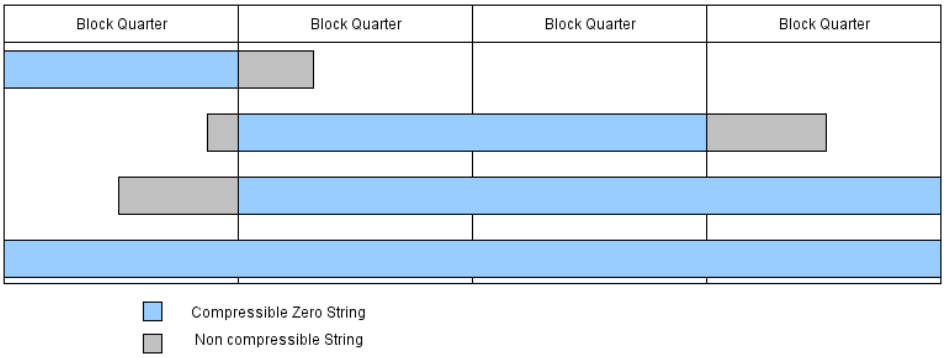


Fig. 30: Example of block aligned zero strings

The graph on ***Fig. 30*** shows the blocks as block start/end-aligned to quarters of block. In order to understand what this really means we can see an example on the ***Fig. 31***. In this figure we observe that data must be at least a quarter of block long to be compressible, but is not mandatory that it is aligned. Although some data compressible with other techniques is not compressible with this one, this technique has the advantage of simplicity and low hardware overhead, so it must be analyzed. To end with ***Fig. 31***, it is easy to see whether the block is start or end-aligned observing the blue bars.

4.3.1.2. Non-aligned consecutively repeated patterns (nACRP)

Fig. 32 shows non-aligned consecutively repeated patterns (nACRP). The X axis corresponds to consecutive repetitions and the Y axis represents the total number of repetitions (the product of consecutive repetitions and the number of occurrences of these, which is the amount of data

compressible). The bubble size is determined by the percentage of bytes of the total. The threshold chosen is 1000 total repetitions. With this threshold we capture 93.45% of the patterns. We can observe that compressible zero strings conform 100% of total interesting data according to this analysis. Notice that these results are similar to the ones achieved for the ACRP case, indeed, the only difference is whether patterns are found aligned or not. As we have many aligned patterns, they also are detected in this analysis and, thus, make results quite similar.

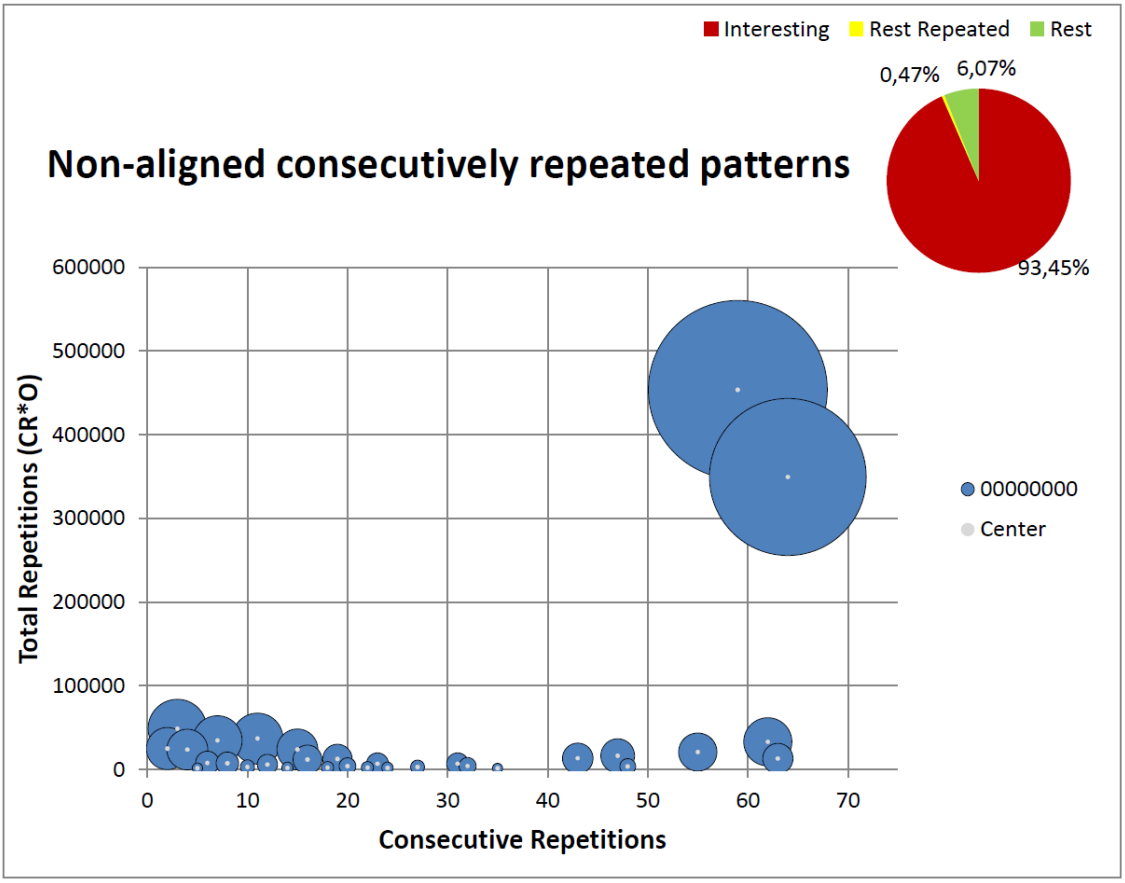


Fig. 31: nARCP total repetitions (consecutive repetitions x occurrences) per pattern.

The interest of non-aligned patterns comes from the fact of using compression techniques like the LZn one. However, if the occurrences of most patterns are aligned, it may induce us not to consider non-aligned ones. Indeed, using both techniques (aligned all-zeros detection and non-aligned LZn method) may not be beneficial.

4.3.1.3. Non-aligned non-consecutively repeated patterns (nAnCRP)

Fig. 33 and ***Fig. 34*** show non-aligned non-consecutively repeated patterns (nAnCRP). The small cake indicates the percentages from total represented by the big one (96.60% and 95.28% respectively).

Non-aligned non-consecutively repeated patterns are difficult to analyze, since we find a combinatorial explosion when trying to obtain all possible combinations (excluding overlapping

patterns) of all patterns. There is thus a trade-off between completeness and time consumption for the analysis. We chose to select patterns in order with two different priority schemes: first prioritizing 0's strings (***Fig. 33***, from 00000000 to 11111111) and then prioritizing 1's strings (***Fig. 34***, vice versa). We do this to show that no matter what order we choose to select patterns (even if the scenarios are opposite), zero byte strings are of most importance (97.44x96.60 = 94.13% and 91.06x95.28 = 86.76%) for compression.

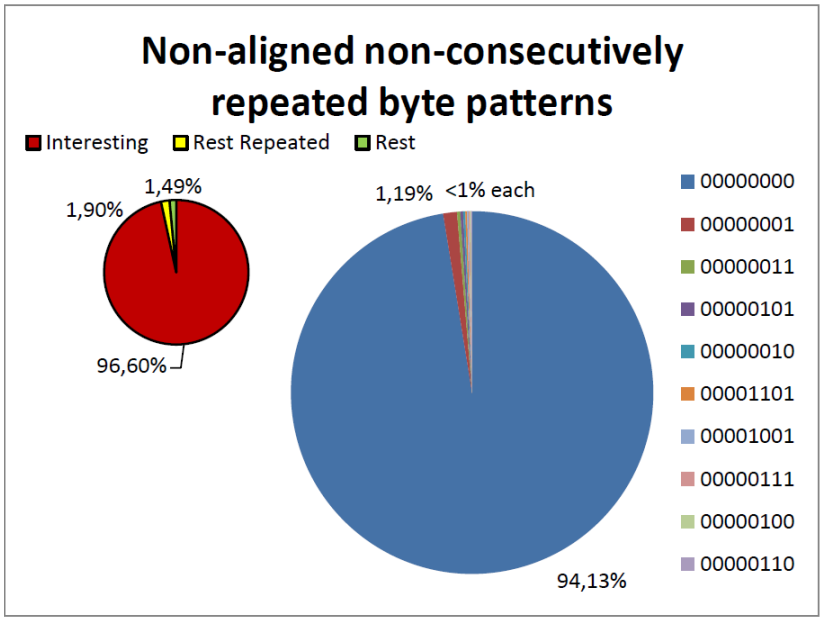


Fig. 32: nAnCRP most repeated patterns (starting by "00000000")

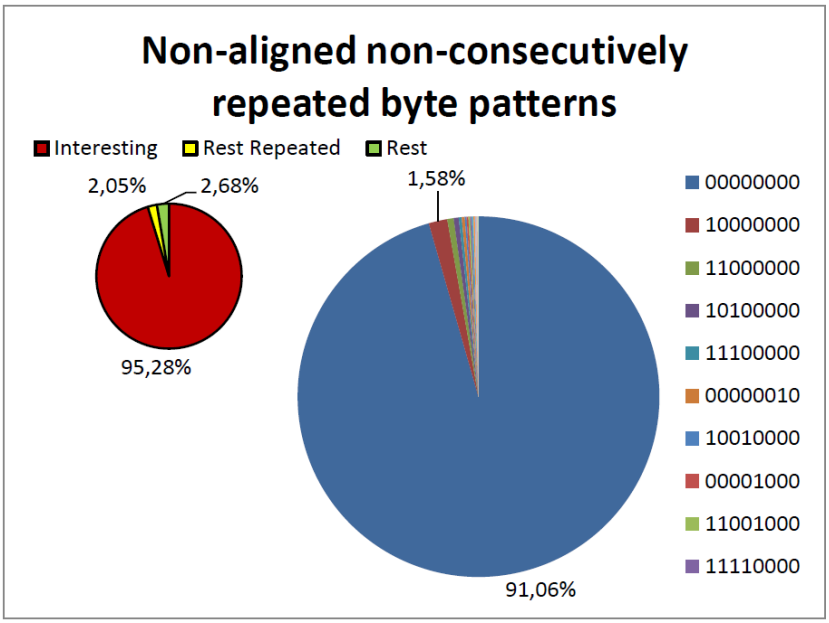


Fig. 33: nAnCRP most repeated patterns (starting by "11111111")

4.3.1.4. General Analysis of Compression Opportunities for applications

In this subsection we compare the statistics obtained for OpenSSL with algorithm SHA1, that have been profusely analyzed in the previous three subsections, with the other algorithms to be considered, i.e. SHA256, SHA512, AES-128-ECB and AES-256-ECB.

We start with a comparison of the general behavior of the data obtained for all algorithms; then we analyze the apparently promising Aligned Zero-Quarters of Block option; to finish with a comparison of the particular patterns designated as the most interesting according to our criteria.

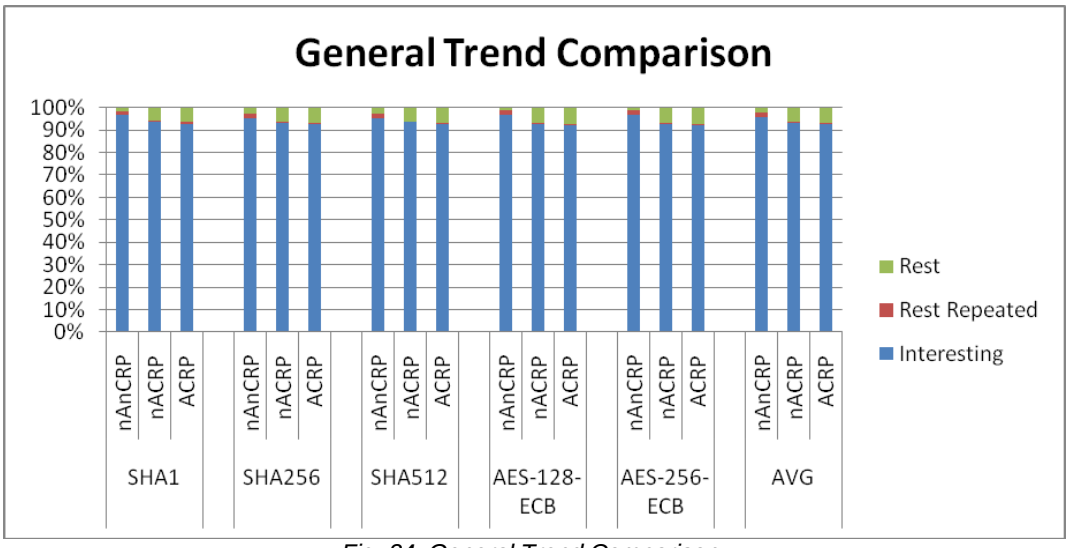


Fig. 34: General Trend Comparison.

In ***Fig. 35*** we can see that the pattern repetition behavior of all the algorithms is very similar for the three repetition schemes analyzed. However, for all the algorithms, nAnCRP exhibits the higher compression opportunities, but notice that this repetition scheme implies a higher complexity. So we must find a compromise solution.

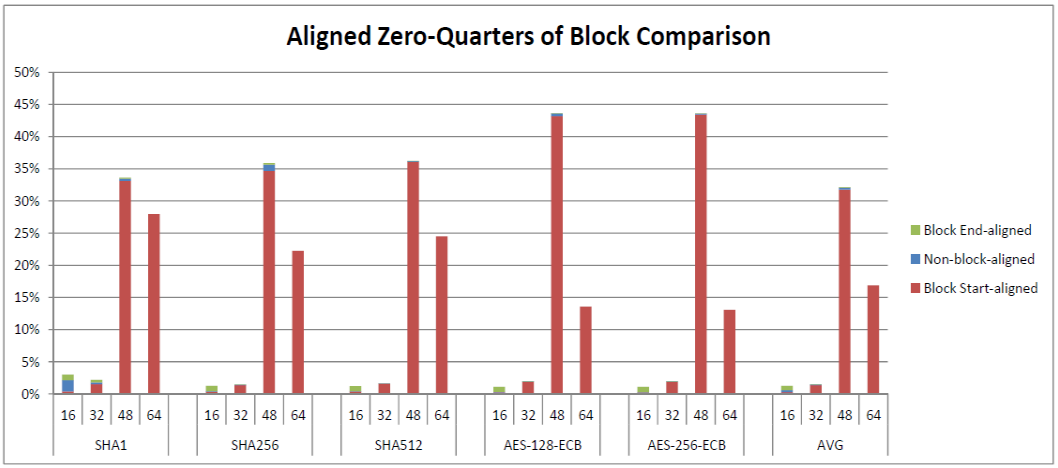


Fig. 35: Aligned Zero-Quarters of block Comparison.

In ***Fig. 36*** we observe that the algorithms studied fall into two different groups, that we can name SHAn and AES-n-ECB. The SHAn algorithms show more blocks with 64 bytes of zeros (ranging from about 22% to about 27%, against the average of about 12% of the AES-n-ECB algorithms). On the other hand, AES-n-ECB algorithms show more blocks with (three quarters of block) 48 bytes of zero strings (an average of about 44% in comparison to the 32% to 36% of SHAn algorithms). Halves and quarters of block (16 and 32 bytes of zeros) are of relatively low importance in both families of algorithms, so we must focus on elimination of either three quarters of block or entire blocks of zeros.

In the reminder of this subsection, we show a comparison of the most interesting patterns for each algorithm and each technique. We consider interesting patterns those with a number of total repetitions (consecutive_repetitions * number_of_occurrences) of 1000 or more. We first analyze in ***Table 2*** and ***Fig. 37*** nAnCRP that includes a greater variety of patterns and then show statistics for nACRP (***Table 3***) and ACRP (***Table 4***), where 00000000 is the only pattern that reaches the aforementioned threshold.

(nAnCRP)	SHA1	SHA256	SHA512	AES-128-ECB	AES-256-ECB
00000000	94,13	93,78	93,98	93,37	93,26
00000001	1,19	1,34	1,28	1,55	1,60
00000010	0,18	0,24	0,26	0,26	0,32
00000011	0,28	0,34	0,37	0,29	0,28
00000100	0,10	0,11	0,10	0,10	0,10
00000101	0,24	0,32	0,33	0,44	0,43
00000110	0,09	0,10	0,00	0,00	0,00
00000111	0,11	0,11	0,12	0,11	0,14
00001001	0,13	0,13	0,13	0,12	0,13
00001011	0,00	0,00	0,10	0,00	0,00
00001101	0,16	0,09	0,08	0,12	0,08
10011010	1,58	0,08	0,09	0,22	0,14

Table 2: Comparison of Most Interesting Patterns (nAnCRP).

In ***Table 2*** we have highlighted those patterns that were not included among the interesting ones for an algorithm by assigning them a zero value and writing them in red color. The interesting patterns are quite similar for all algorithms and the percentages represented by them are very similar as well. Thus, there can be observed that a dynamic table-based technique would take profit of this pattern repetition. Moreover, zeros represent over 90% of the data in all cases, so it can also be induced that the compression techniques to consider hereafter should be zero elimination techniques. In ***Fig. 37*** we can see the same information graphically.

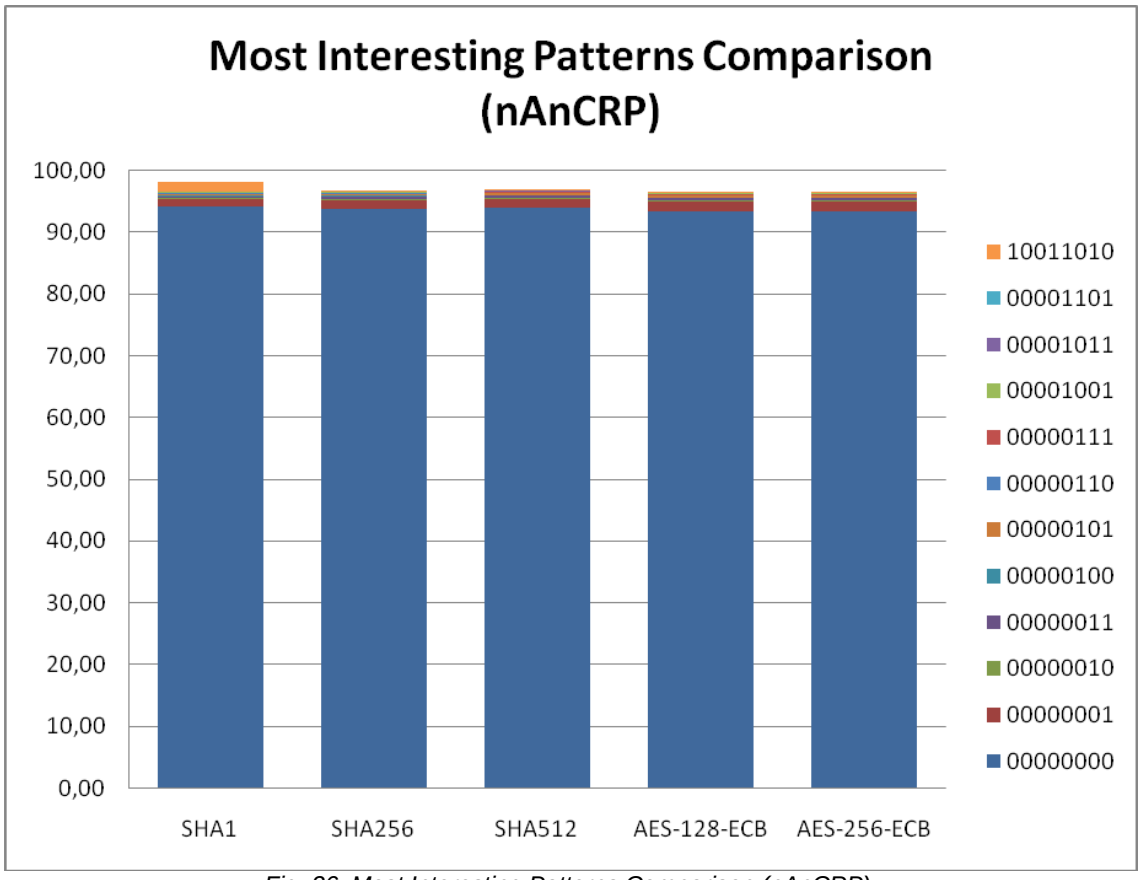


Fig. 36: Most Interesting Patterns Comparison (nAnCRP).

Table 3 and ***Table 4*** show that the proportion of zeros in both repetition schemes (***nACRP and ACRP***) is very high and very similar. We can deduce that most zero strings are long (consecutive repetitions of zero byte strings) and many of them aligned to byte (since percentages for ACRP are just a little bit lower than for nACRP).

(nACRP)	SHA1	SHA256	SHA512	AES-128-ECB	AES-256-ECB
00000000	93,45	93,26	93,42	92,83	92,74

Table 3: Comparison of Most Interesting Patterns (nACRP).

(ACRP)	SHA1	SHA256	SHA512	AES-128-ECB	AES-256-ECB
00000000	92,86	92,53	92,79	92,22	92,00

Table 4: Comparison of Most Interesting Patterns (ACRP).

4.3.1.5. Conclusions for compression opportunities

From the previous analysis we have different conclusions:

1. The most important streams of data are made of all-zeros. This means that those streams should be the primary focus for compression opportunities in the project.

2. All-zero compression methods should be prioritized, analyzed and researched within the project.
3. Block alignment represents a large hit rate for all-zero streams, thus easing the implementation at the NoC level of such compression mechanisms. Techniques dealing with different block sizes (quarters of blocks) should be researched within the project.
4. Techniques focused on non-aligned patterns do not seem to be interesting for the found patterns. Thus, techniques as LZn seem out of place for the project.
5. Non-consecutive patterns, although largely found in the applications analyzed, will be largely covered for the all-zero streams. Indeed, most of the non-consecutive patterns are made of all-zero streams.

5. Virtualization Requirements

Virtualization has been spreading in the IT server area at a fast rate in the last years. It provides an added value in resource usage such as CPU and disks, allows better flexibility, better fault tolerance and moreover better resource occupancy. Virtualization in the server area gives to IT managers and administrators an unprecedented optimization rate of hardware resource usage, together with services to ease management of these resources. The generalization of server farms usage in the context of Cloud Computing is making extensive use of virtualization technologies, hence promoting them to the rank of mandatory capabilities.

In embedded systems the situation is very different from what is observed in the server area. Virtualization technologies are barely used, or barely visible to the device user. Although virtualization is present in some consumer-electronic devices, it serves mostly to provide maintenance capabilities. Whereas the benefits of virtualization in the server area are nowadays quite clear, it is relevant in the frame of the vIrtical project to firstly present domains where virtualization can provide an added value for embedded systems. Secondly a more technical requirements analysis is presented which is more related to virtualization technology. Finally non-technical requirements are addressed to complete the area.

5.1. Domain requirements

Among many domains using embedded systems, the following ones are those of primary interest for the vIrtical partners:

- Telecommunications
- Consumer Electronics

We also done additional research for some domains which are not in the scope of vIrtical but in which safety and reliability constraints play an important role. Virtualization is already used in

some of these domains to cope with the existing constraints. The vIrtical approach for virtualization can bring additional benefits such as security. These domains are :

- Automotive and Vetronics
- Transportation
- Avionics

The virtualization requirements and potentialities are presented in the following sections.

5.1.1. Telecommunications

The telecommunications market has evolved very rapidly in the recent years, mainly due to the nature of the traffic conveyed by networks. Whereas it was dominated by voice and fixed terminals in the 90's (PABX, modems, Public Switched Telephone Networks), it is now dominated by data and mobile terminals (ADSL, VPN, VoIP, VOD). The nature of this traffic is very diverse, ranging from TV over ADSL to peer-to-peer networks. Quality of service is not currently handled from end-to-end, due to the heterogeneity of the resources and protocols networks involved in the networks.

Enhancing resource reservation for networks

In most IP switches and routers, traffic shaping is achieved using information stored in Ethernet frames (DSCP field). The functionality is implemented according to a best effort scheme with no strict resource reservation. Virtualization can provide such resource reservation for each class of traffic. Additionally it can provide resource separation to enable fault-tolerance within routers and switch between critical administration and configuration functions that require a limited amount of resources but an all-time availability (management plane), and traffic handling that requires most of the resources with good availability (data plane).

Enhancing traffic handling for mobility

Mobility raises a number of problems not addressed by conventional routers and switches. The capability of mobile devices to switch from available 3G, WIFI, bluetooth, Ethernet or 4G requires the equivalent level of adaptation from the infrastructure that is routing the traffic. Intelligent routers and switches are needed for this purpose. In this respect virtualization can provide a support for base services similar to what exists in conventional equipment, along with a capability to host application support for the new services layer that handles the versatile terminals communicating capabilities.

5.1.2. Consumer Electronics

Post-PC era is about a world in which new devices with a true cultural impact, (Your life is in your device, your media and your information are always “there”, boundaries between work, home, and friends vanish,) surpass the desktop and laptop, in numbers deployed and in

economic and social impact. In this context set-top boxes (STB) and Smart TVs are evolving from the traditional services, Broadcast, Decode & Rendering and Video On Demand, provided by TV operators toward over-the-top (OTT) video applications with additional internet applications, local multimedia hub and craddle. Operators can use such STB to provide interactive OTT services for broadband users, thus increasing the added-value of broadband network and the user scale. STBs and Smart TVs are becoming connected interactive and open platforms. The interactive services will evolve on top of the traditional services. Some of the today examples are the new interactive content created for WEB access such as documentary, sports, content aggregation can turn Youtube into your own Music Video channel or enabling 7 days Catch-up Service to go in the past to replay your favorite programs. The next generation services will include Home automation and smart metering as well as personal services such as healthcare, productivity, learning, leisure. In order to support the market trends the ST vision is focusing to reduce the environmental footprint, to enable new user experience and to open up the platform. All the new architectures developed by ST are based on ARM Core allowing ST to strongly leverage on the leading European low power CPU roadmap and a broad software ecosystem. In addition ST brings the strong knowhow on security, system integration, HD video decoder and silicon technology. From the software standpoint ST needs to support several technology enablers such us Linux, Android, DVB, HTML5, gsstreamer, Adobe Air etc. as shown in ***Fig. 38***.

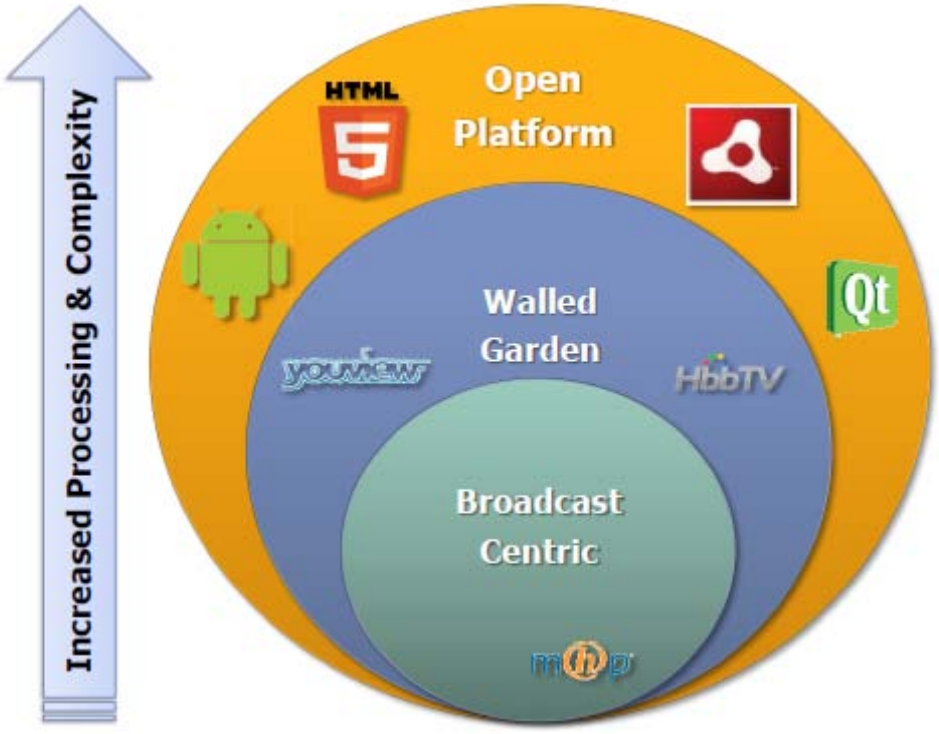


Fig. 37: Technology enablers to be supported.

Virtualization already has impacted the server and IT industries in a significant way. The trends to adopt virtualization in the consumer space, in particular to STBs and Smart TV, are being

driven by the fact that these platforms are becoming open. Each product has its own specialized list of time consuming social networking and on-demand video contents of available apps that could be downloaded by the user. In addition products allow to view content from the Internet on your smart TV through specialized apps (Netflix*, Napster, Facebook, browser etc.). These apps are designed to work specifically and very smoothly with the television experience.

Vendors are working on virtualization technology to separate broadband services (such as Android) from the broadcast services (such as NDS, HbbTV) available within the "Walled Garden" via multi compartment that can be implemented by the virtualization technology.

In particular, virtualization allows STB or smart TV to run multiple virtual machines simultaneously enabling the consolidation of hardware and contributes the increase for utilization rates. This means that multiple middleware or OS can run on a single device simultaneously, with many cross-platform applications scenarios possible, and this promotes interoperability among heterogeneous systems. In smart home environment with high degree of heterogeneity in terms of services, virtualization will provide to the home dwellers the option to select the suitable operating systems for each application/service. In addition virtualization allows STB and Smart TV to maintain a degree of support for legacy systems while upgrading to new hardware and services. In this sense legacy services (even when operating systems and platforms are no longer supported by manufacturers) can be still used by running in a virtual machine. This feature enables saving hardware and operational costs.

Last but not least with virtualization the complexity of remote management of STB or smart TV is reduced tremendously because virtual machines are completely decoupled from the physical devices. Adding or updating a new service, there is not anymore the need to shut down the device, giving to the home dwellers a feeling of robustness.

In this new post PC world, STB and smart TV need to be more intimated and connected with other embedded devices that consumers keep close to their bodies, and more physical, as mouse and keyboard are replaced today by touch screens. As this post-PC world evolves, facial recognition, voice sensors, and motion sensors will become controllers, increasing the intimacy and physicality of our relationship to technology and devices. For them to grow and dominate these great devices will rely upon increasingly powerful access networks and infrastructure. One of the biggest challenges in home entertainment and media servers is managing the content and securing the sensible information of Home dwellers (passwords, identification data, geolocalization ...). By providing isolation capabilities and untamperable emergency modes (accessible from the network) for erasing permanently this information, virtualization can bring the necessary security mechanisms that smartphones are currently lacking.

Home dwellers deals with various content of music, pictures and movie, often ended up accessing these contents from different devices. This kind of situation creates hassle for dwellers as they need to switch to different devices each time when it comes to access some of these contents. Virtualization could solve this issue by deploying content virtualization in a

distributed home platform. This could enable service providers to port, integrate, and build media and service management efficiently of the media contents in home environment. Streaming media like Video-on-Demand (VoD) could be utilized from different sources, especially from media providers using virtualization technology. It is expected that all digital contents in smart home settings will be integrated into a virtualization pool that is always available for home dwellers. This virtualized pool will automatically manage functions like synchronization, content distribution, Quality of Service (QoS) and perform discovery of content and devices in the smart home environment. Virtualization also will ensure organization, tagging and analysis for all the stored entertainment content in efficient manner. On the other hand, it paves the way for activity recognition of home dwellers based on their use and access made for retrieving those contents. Virtualization technology also could be the key contributor for moving or 'movable' multimedia especially in creating session mobility that provides seamless transfer of multimedia session between different devices and contents based on home dwellers preferences.

So far virtualization we have described how important is the virtualization for the next generation of STB and Smart TV. In order to be acceptable from system point of view, it is important that the drawback in term of performance penalty is acceptable. In order to overcome these performance overheads, one solution is to implement para-virtualization instead of full virtualization. However running unmodified guest operating systems is desirable, so hardware vendors (ARM, INTEL, AMD) have started shipping extensions to their processors so they are efficiently virtualizable. In that case, when a Hypervisor may take advantage of the hardware support for efficient virtualization, the system is said to support Hardware Virtualization.

Therefore for STB and Smart TV using hardware virtualization is a strong requirement for performance reasons. In addition, linux is becoming the de-facto standard open source operative system used in the majority of STB and Smart TV therefore it is important that the virtualization technology is open source and fully integrated with linux.

Since in vIrtical, we are going to focus more home gateway, the following section described better from system point of view the key functions that will be integrated in the coming generation of products.

A home gateway is a consumer electronics product, potentially leased to the end user by a multiple service operator (MSO), which combines functions which previously have been provided by multiple discrete devices. For example, combining the following functions:

- Broadband Modem (e.g. DOCSIS/ADSL/FTTx)
- VoIP telephony adaptor (e.g. PacketCable)
- Router with WiFi + gigabit Ethernet
- Home automation controller
- Media Server (Potentially an open s/w stack allowing downloads from multiple sources)

The user expectation of such a device stems from the original system where a malfunction in a media server would not cause interference on a telephone call. Here the requirement for virtualisation is clear: to provide a route to significant cost & energy savings by combining

multiple functions on to a single host CPU, while maintaining the level of reliability achieved with discrete systems.

The MSO expectation of such a device covers both the reliability of billable services (e.g. telephony, broadband data access, video on demand) as well as requiring strict isolation between sensitive information used by each compartment/guest from another (e.g. modem authentication certificates, DRM keys, home automation authentication).

The reliability of the subsystems will be defined in terms of robustness (ability to operate in the presence of a malfunction in another guest) and in terms of QoS (reserving enough processing power to guarantee certain key functions during times of high CPU load – e.g. VoIP).

It is likely that a home gateway based upon a multiple guests running concurrently will require clean resource sharing for functions such as:

- Non-volatile Memory system
- Networking system

It is expected that this market will share some of the same requirements as both the STB and Telecommunications markets.

Smartphones and in a lesser extent tablets have invaded the industrialized countries. Their success is largely due to the large number of applications available, their ease of use, and the capability of mobile networks to handle them with 3G or 4G technology.

Coexistence of corporate and personal usage

Smartphones are used everywhere, in the personal sphere and also in the corporate sphere. The problem raised by the separation of data has been raised several years ago with the use of Blackberry devices in governmental entities (e.g. French government in 2007). Having one smartphone for each sphere is not very practical, but having the same device for all sphere raises confidentiality and privacy concerns. Virtualization can provide the necessary separation needed between private and corporate data, and also between private and corporate contexts (network access, user authentification, applications access ...)

Protection of sensible information

In a given sphere, personal or corporate, there is some information which is more sensible than other (passwords, identification data, geolocalization ...). There is absolutely no warranty that an application (available from the market or the store) cannot access this data and leak it outside the terminal. There is also no warranty that in case of theft none could access this data. By providing isolation capabilities and untamperable emergency modes (accessible from the network) for erasing permanently this information, virtualization can bring the necessary security mechanisms that smartphones are currently lacking.

5.1.3. Automotive and Vetronics

In automotive, the number of electronic and computing devices embedded in a single car has grown tremendously in the last ten years.

Providing better reliability and better maintainability

In order to provide better reliability overall, these systems are being mutualized. Maintainability can also be enhanced by having “generic” platforms (constructor-independent) hosting specific functions (constructor-dependent).

Providing more cost-effective driving aids

Although the automotive market, due to its very concurrent nature, is heavily cost-driven for the choice of equipments, some moderately critical functions such as driving and parking aids may benefit from virtualization when compliance to some safety standards has to be done in a worldwide market. These equipments could be certified in an incremental fashion, which would allow both a platform market for car equipment makers and a degree of flexibility for car manufacturers to choose which applications are implemented on these platforms.

5.1.4. Transportation

Ground transportation systems are subject to certification compliance process similar to what exists in airborne systems, but the constraints are different mainly due to the fact that the environment is not dangerous by itself. Stopping a train or a metro is always an option, which is not true for aircraft.

Widely used in rail signaling, the SIL certification defines several levels depending on the criticality required for the equipment in the system. To separate critical functions from non-critical ones on the same equipment, and to monitor critical functions, virtualization can provide the base infrastructure to achieve the required criticality level. Functional correctness and behavior in case of fault are the main requirements virtualization has to provide answers for.

5.1.5. Avionics

Safety regulations impose a high degree of reliability of all aeronautics equipment. It covers among others, the manufacturing process, the functionality correctness, the fault tolerance, the lifecycle management. The certification process is standardized worldwide and is declined in several levels for both software and hardware equipment. For software the DO-178 B is an international standard and covers a range of levels from A (most critical systems) to E (less critical systems). In order to achieve compliance to most critical levels (A to C), it is common practice to have multiple redundant systems providing the same function.

Integrated Modular Avionics (IMA) has provided a great benefit by introducing the capability of coexistence of several functions on the same equipment, with the strong requirement to be able to prove functional independence, including in case of failure. In particular a failure of a function shall not trigger failure for other functions coexisting on the same equipment.
The ARINC 653 norm, elaborated in order to specify the properties needed by the IMA platform in order to support such applications is an example of platform virtualization for applications, based on a temporal share of hardware resources. Determinism of execution of applications is the main resulting capability provided by IMA.

Mixing critical levels

However, the IMA approach already addresses specifically the coexistence of functions with mixed criticality levels (for instance mixing level B and level D functions), since it states that the most critical level imposes its compliance. Virtualization in avionics provides the necessary capabilities to mix several criticality levels.

Mixing critical and non-critical levels

Today in most planes all electronic equipment has to be switched off at specific phases of flight (takeoff and landing). In-flight entertainment systems, although part of the aircraft, are similarly not usable during these phases. Cabin crew resources and IFE (In-Flight Entertainment) resources are usually completely separate although they use a number of similar functions such as message broadcast and call notification. Virtualization with adequate priority handling mechanisms is able to provide mutualization of these functions (cabin crew first, IFE next), as well as a potential for more services (multicasting, geographical routing on large planes), not achievable with currently separate systems.

Enabling external communications

External communications are very limited from an airplane. Similarly to what is now possible with high-speed trains, it could be envisioned that communications originating from terminals such as smartphones could be relayed by an on-board equipment to the ground. Virtualization here would help in handling resources of this on-board equipment with added airborne specific issues such as weight, power consumption and operating temperature range.

5.2. Technical requirements

The technical requirements are based on the following model for virtualization, where

- The virtualization is provided by 2 hypervisors: one supporting the full virtualization for the not ARM Non-Secure Execution Environment and one supporting the para-virtualization for the secure environment. Depending the application domain we can use both hypervisors or only one,

- The compartments (or Virtual Machines) can accommodate an Operating System or a simple API above which an application is running.

The resource management is delegated to the secure environment.

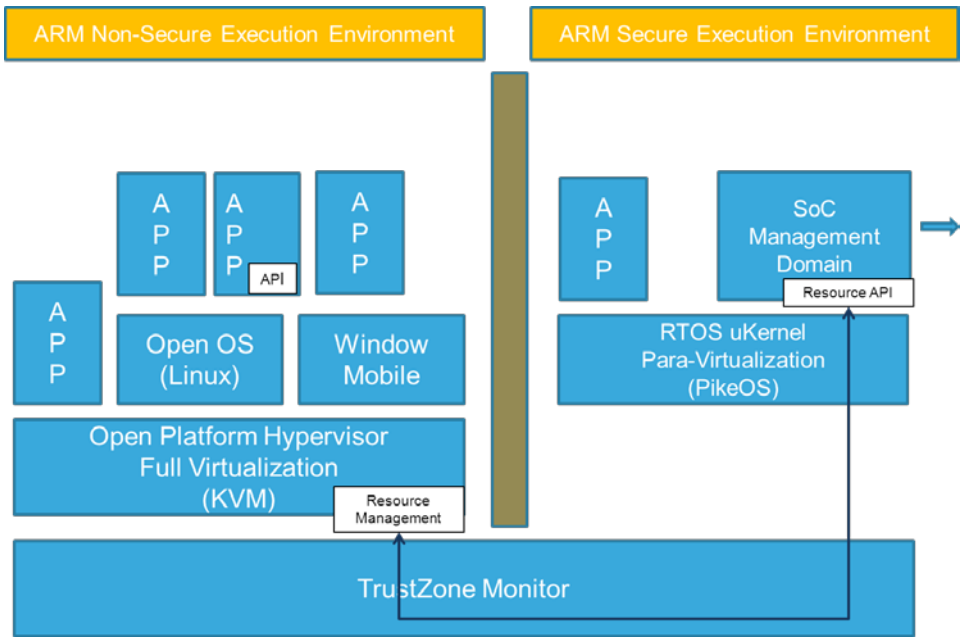


Fig. 38: Virtualization model

There are other virtualization alternatives than the one presented, however they are not in the scope of current vIrtical activities. Their presentation in detail is beyond the scope of this document and there are many references publicly available for more information.

5.2.1. Isolation between compartments

Isolation between compartments is the main advantage provided by virtualization. It is usually split in two categories: temporal isolation and spatial isolation.

5.2.1.1. Temporal isolation

Temporal isolation is the main feature in ARINC 653 compliant systems. It states that each compartment temporal execution is not affected by operations carried out in other compartments.

A common practice to achieve temporal isolation consists in allocating one or several time slots within a period to each compartment without overlap between them, and bv scheduling compartments execution by the hypervisor. The CPU resource is shared, in a timely fashion by compartments.

In multi-CPU this can be extended by pre-allocating one or several CPUs to each compartment, for each time slot, again with no overlap.

5.2.1.2. Spatial isolation

Spatial isolation refers to the capability of partitioning the memory resources among compartments. Each compartment is granted access to only a subset of the whole addressable memory. By doing so, no communication is possible between compartments.

When some degree of communication is needed between compartments, the hypervisor is handling the memory corresponding to the communication facility by making it accessible to the compartment when it is operating.

Spatial isolation requires that the hypervisor has full control over the memory mapping for all compartments and more specifically full control of the MMU.

Systems having real-time constraints use temporal isolation and spatial isolation together. However there is a systematic penalty to reaction to external events caused by the temporal allocation of compartments.

Reactive systems based on external-events can make use of spatial isolation only to overcome this limitation, but they are unable to ensure CPU resource reservation to critical tasks that are not event-driven.

5.2.2. Communications between compartments

Strictly isolated compartments use cases are quite rare for embedded systems. Controlled communication channels are very often needed to exchange information between compartments.

Depending on the application constraints, different communication channels are needed :

- Unidirectional FIFOs
- Shared memory
- Mailboxes
- Semaphores
- Virtual channels : e.g. virtual Ethernet channels

Properties attached to these communication channels are very important and they are usually under the control of the hypervisor. However some communication channels such as shared memory can be accessed without control by two or more compartments. This is especially true for multi-processor systems. Therefore when some additional properties need to be assessed, care must be taken in the selection of communication channels for an application.

5.2.3. Resource sharing and reservation among compartments

Many resources can theoretically be shared by compartments. They are:

- CPU

- memory
- GPU
- co-processors (SIMD)
- DMA controllers
- peripherals
- buses
- caches

5.2.3.1. CPU

Virtualization inherently shares CPU time among compartments. However some parts of the CPU are not virtualizable per-se, such as performance counters and timestamp counters. Since they can be reset but not reloaded with specific values, they are quite often not made visible to the compartments by the hypervisor. The virtualization of performance counters would allow compartments to monitor performance relatively to their own activity.

Memory

Whereas virtualization provides time partitioning by controlling CPU time allocations, it provides spatial partitioning by controlling the memory ranges allocated to each partition. Since the hypervisor must have full control over these memory ranges, compartments have access to virtual memory only, and the hypervisor controls entirely the MMU.

Since several CPUs and peripherals can write to memory, the hypervisor has to maintain the consistency of all memory areas under its control. Memory remapping for a peripheral is a minimal option but does not allow sharing the peripheral by different compartments.

GPU

Due to very high performance requirements, CPU and GPU are very tightly coupled. They share large ranges of memory located in central memory and in GPU device. The CPU offloads some graphics computations depending of the capabilities of the GPU to handle them more efficiently than the CPU. Graphics display is done independently from graphics computations, with CPU or GPU being at the initiative. To complete the overview, many high-performance GPUs are driven by proprietary software and firmware, which is in all cases very specific to the GPU vendor.

Most hypervisors have little or no knowledge of the GPU, so the baseline for driving it is by using the baseline common modes, which are VGA and framebuffer. In such case all operations are carried out by software on the CPU (no offload by the GPU) and are altogether limited in capabilities, and costly in CPU cycles.

What is needed for hypervisors is GPU drivers to enable today display capabilities (high resolution beyond VGA) and use corresponding offloading functions provided by the GPU making the GPU visible to the hypervisor.

Co-processors (SIMD, DSP)

These resources are used as accelerators for some specific, domain-dependent (video) computations. They are usually programmable and either associated to one CPU or accessible to all CPUs. In all cases, the sharing support of these devices by virtualization requires that their command, control and status registers shall be under exclusive control of the hypervisor and that their state shall be saved and restored at each compartment reschedule. For co-processors associated with a CPU and which usually share the same caches for data accesses, there may be some additional impact on cache management.

DMA controllers

Since a DMA controller is transferring data to and from main memory and operates simultaneously with the CPU, but out of its control, both in time and memory range, the DMA controller is as such not usable in a virtualized context. In order to use DMA controllers the following bounding rules have to be enforced:

- Bounding in time, by enabling the CPU to suspend a DMA operation in progress, for instance whenever a new compartment is scheduled by the hypervisor, and to resume it later on, whenever the previous compartment is rescheduled.
- Bounding in space, by making sure that the hypervisor does not allow any setup of the DMA controller memory ranges that would be incompatible with memory ranges allowed for a compartment. This involves some knowledge about the behavior of the DMA controller by the hypervisor.

Peripherals

Sharing peripherals among compartments is a daunting task due to the unlimited variety of peripherals available. As a general rule, virtualization limits its role to exporting memory ranges handled by a given peripheral to a unique compartment and by making sure no other compartment can directly access this peripheral. This is the simplest case of partitioning feasible. This approach is satisfactory for low-performance peripherals such as serial lines or GPIO devices. Virtual peripherals can be used by other compartments with data dispatching from the compartment handling the physical device to other compartments manipulating data to and from the device. However for high performance peripherals this approach has serious drawbacks since it enables only part of the device capabilities (and resulting performance) to be used. Furthermore there are cases where sharing a peripheral by several compartments is a need. For instance, network interface cards and components are supporting several physical links connected to one multichannel MAC component. Such devices are virtualizable on a per-channel basis provided that the hypervisor has knowledge of the device and that the system integrates some memory protection for the I/O (IO-MPU IO-MMU).

Busses and NoC

Since busses are the places where all transfers occur, the bus management is of prime importance for virtualization. As such bus controllers have to be under exclusive control of the hypervisor. It is true for the main system bus, but also for peripheral busses and for subsystem busses such as USB. Since management of these latter busses is somehow complex, management of such busses is limited at hypervisor level to opening the needed memory ranges to a compartment and leaving the control of the bus to this compartment only. This sharing of busses is actually very limited.

By providing more capabilities (partitioning and routing) than busses, NoC are full of potential for virtualization. Datapath access control can be gated by hypervisor, dataflow can be prioritized and spatially routed, fault-tolerance can be obtained by redundant data paths inside the NoC. Furthermore, coupling with IO-MMU can be established as a rule for peripherals access.

Caches

Caches are very useful for providing better overall system performance. But their stateful character is a huge problem for predictability, needed for certifire/evaluated/realtime systems. The generalization of multicores in embedded adds a second problem, linked to the cache sharing among several cores. To deal with predictability issues, the solution is not to disable all cache operation in the system, since performance is dramatically affected. In most of the application cases, a compromise between performance and predictability is reachable. To do so, the cache subsystem has to integrate capabilities usable by the hypervisor such as:

- Controlling the amount of cache usable by compartments, by means of invalidating part of the cache for some compartments.
- Controlling the state of caches at each compartment switch, by at least forcing cache flushes.
- Enabling the amount of unused cache to be used as shared memory, while retaining control over consistency of this memory with the main memory (mapping/remapping between compartments).

5.2.4. Static and dynamic resource allocation for compartments

Resource allocation for compartment can be done either statically or dynamically. Below is a brief discussion of these approaches.

Static allocation

It is done by configuration and never changes during system operation. The main advantage of static allocation is that it can be checked prior to system operation, thus facilitating proof for certification and evaluation.

Dynamic allocation

Usually dynamic allocation is done either by the hypervisor, either by a special administrative compartment having some specific privileges. Re-allocating resources for compartments is a very interesting capability especially for embedded systems where resources are limited. But the cost or resources needed to handle dynamicity must be taken into account. Furthermore, it is much more difficult to provide the elements needed for certification and evaluation in a dynamic context than in a static context.

5.3. Non-functional requirements

These requirements, although not linked to functional characteristics are needed for some domains (aeronautics, automotive, transportation) where failure or malfunction has a very measurable impact which must be taken in to account.

5.3.1. Fault tolerance

Fault tolerance is needed in DO-178 and SIL certifications. Since many systems in these domains have also other strong requirements such as real-time operation, the impact of fault-tolerance can be major on the overall architecture. Virtualization can help to alleviate this impact by providing means of segregating parts of an application with respect to fault-tolerance.

Below is an example of the impact of how virtualization helps to introduce fault-tolerance in the software architecture of a real-time system.

- The legacy system (see ***Fig. 40***) has been designed without any fault-tolerance requirements, having only real-time requirements for the critical application.
- The new design (see ***Fig. 41***) is done with as little impact as possible on the applications but with additional fault-tolerance requirements on the critical application.

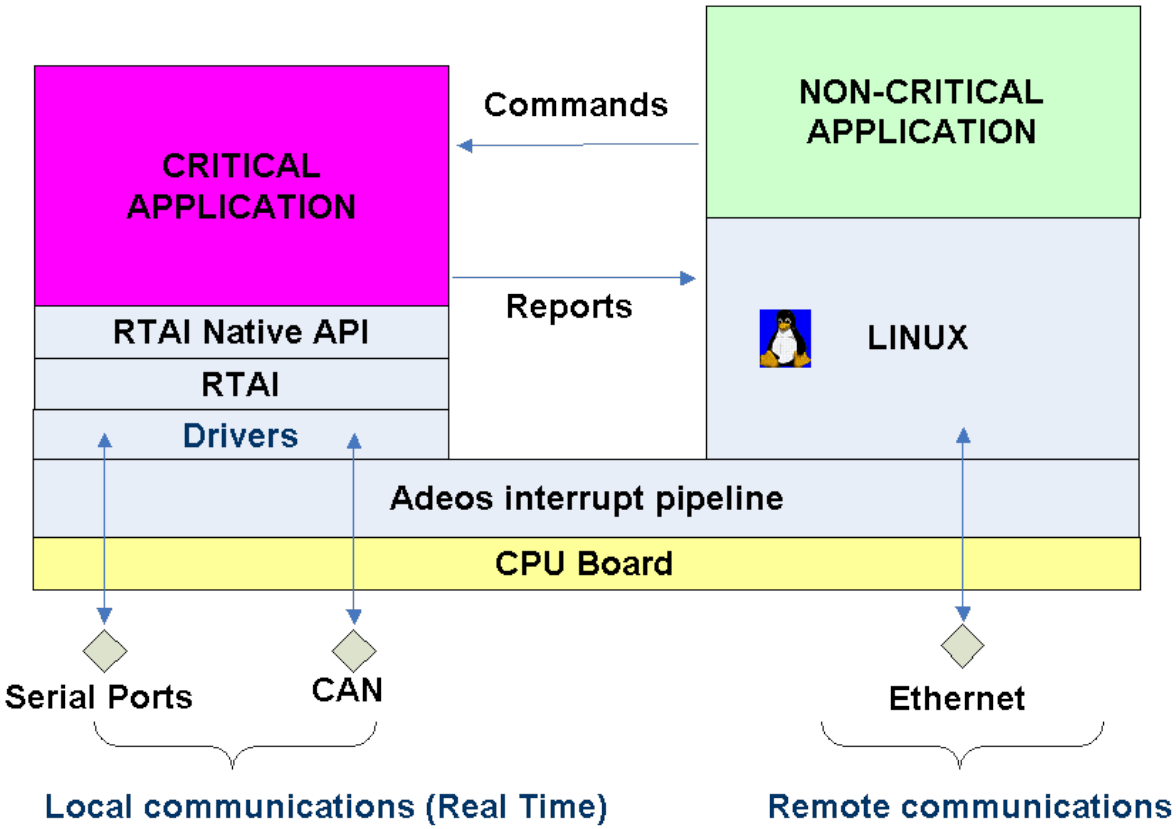


Fig. 39: Non fault-tolerant architecture

In ***Fig. 40*** the application has no fault-tolerant requirements. Criticality is done versus real-time only. So, the split has been made between real-time and non real-time components of the application. Failure of the non real-time part of the application triggers failure of the real-time part and reversively.

In the ***Fig. 41*** below, virtualization has been introduced in order to cope with fault-tolerance issues. The critical part has to continue operating, whatever the state of operation of the non-critical part. The real-time requirements of the critical part must be kept valid.

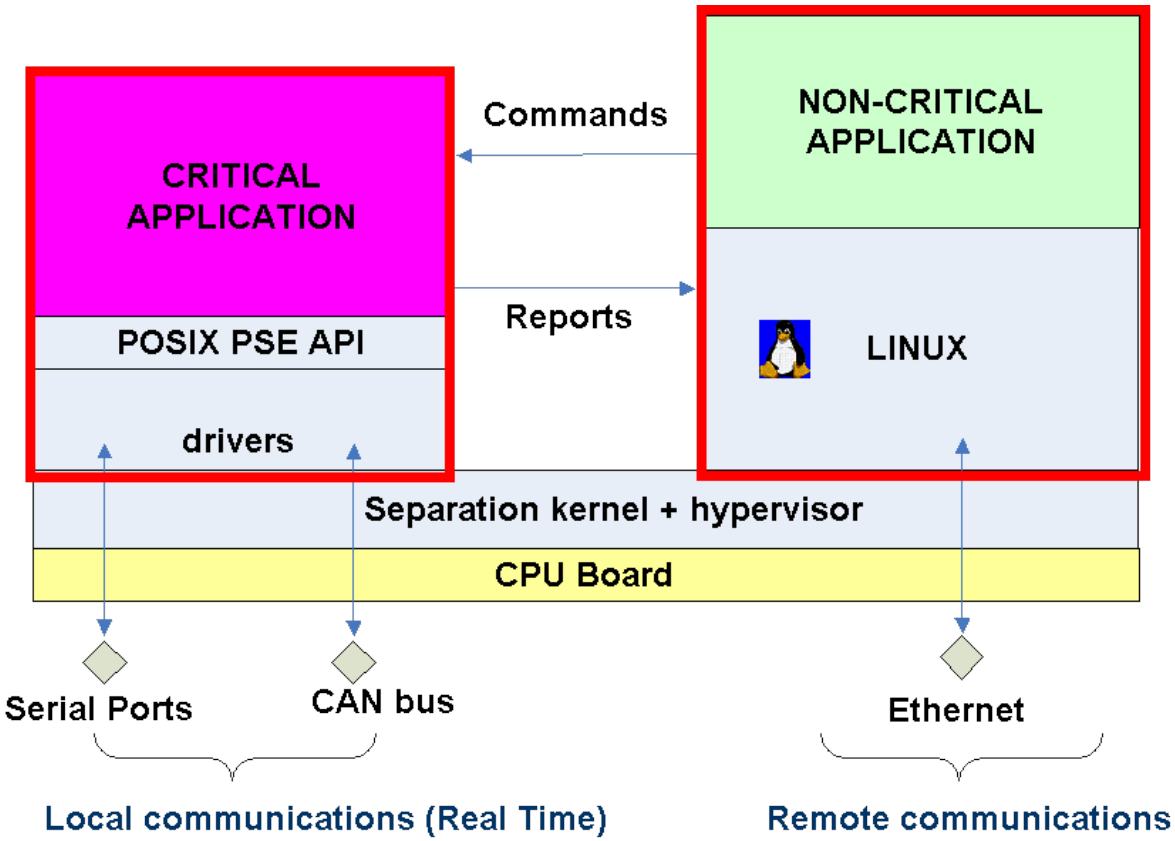


Fig. 40: Fault-tolerant architecture

The above case where the real-time part could be mapped to the critical part is not generalizable. In most cases, there are real-time requirements in critical and non-critical parts. So in the general case, requirements such as fault-tolerance must be taken into account as early as possible in the design process to be able to make the suitable architectural choices, including virtualization.

5.3.2. Security

Security is another strong requirement which was formally required for very specific devices and systems (ATM, smart card readers) and which is now required in a greater number of cases due to the interconnection of systems in the everyday life. For instance security is required for content protection (digital media broadcasting) in set-top boxes, for network access and billing in wireless communications (2G/3G/4G, ADSL), for enterprise-class VPNs …

The most used norms in security are CC (Common Criteria) and EAL (Evaluation Assurance Level). The goods to protect are defined by a Security Target, and their protection must be proved (formally or experimentally, depending on the Assurance Level) in any case, including an accidental or caused malfunction of the system.
Independently from using trusted hardware, which has very strong impact for companies not designing silicon, virtualization can help in security evaluation if the hardware-assisted

virtualization capabilities are well documented and their behavior can be verified. Similarly, the virtualization software has to be verifiable by external security experts in a blackbox, greybox or whitebox approach, depending on the security level required.

5.3.3. Co-existence of full virtualization and para-virtualization

One of the main interests in virtualization is the capability to host several guest OS. In order to support proprietary guest OS, full virtualization must be provided, whereas for non-proprietary guest OS, para-virtualization provides better performance and control at the expense on requiring adaptation of the guest OS for the hypervisor.

In cases where there is a need to group on the same hardware applications that were developed to run on different hardware equipment, virtualization can help whenever the cost of application migration from one OS to another is important. This cost usually does not involve development and integration, but also validation and qualification.

The co-existence of full virtualization and para-virtualization capabilities is a need mainly for embedded systems where hardware mutualization has a strong economic interest.

5.3.4. Performance

The impact of virtualization is a key factor for embedded systems whose resources are inherently limited. It accounts significantly for the scarce penetration of virtualization in embedded devices in comparison with server systems.

In order to minimize negative impact and to provide the best efficiency altogether for embedded systems, virtualization has to address performance issue by providing the following capabilities:

- HW assisted virtualization: similarly to server systems, it removes the burdent of executing in software most of the operations that can be executed in hardware when no virtualization is executing.
- Graphics acceleration: mandatory for embedded systems where power-consumption is a key criteria for any device in the system. Absent from server systems.
- DMA: similar to graphics acceleration. It exists in server systems where an IOMMU is present. It requires IOMMU or IOMPU on the embedded system.

5.3.5. Other requirements

Aside from above requirements the features listed here may be of interest for some use cases of virtualization, depending of the technology used:

- Autonomous creation and management of virtual machines: in cases where the deployment of virtual machines is the result of an executable specification (as opposed to be compiled in .e.g. static allocation), it is necessary for embedded systems that this execution does not depend of an external entity such as an external console.

- Health monitoring: health monitor capabilities are present in the virtualization infrastructure if needed. The status of each compartment can be monitored locally or remotely, and operation from a distant console can be carried out if needed.
- Security monitoring: similar to health monitoring, security checks such as integrity and authentification can be carried out on a periodic basis, and from a distant console if needed.
- Timekeeping: in a virtual machine, timekeeping can be carried out with the host system or from an external time reference, with no side-effects for other virtual machines.

Synchronization, locking and resource preemption: in case of communication channels between compartments, the virtualization infrastructure has to provide these capabilities and the means of controlling them, regardless of the services provided in the compartments.

6. Conclusions

In this deliverable we have analyzed typical industrial applications target for the SoC platform designed in the vIrtical Project with the objective of identifying opportunities to improve efficiency, scalability and security of heterogeneous multicore embedded. For achieving this we have analyzed acceleration, memory and virtualization opportunities.

Concerning acceleration opportunities three computational kernels (SIFT, SURF, FAST) from the Computer Vision domain were identified by UNIBO, which are good candidates for execution speed acceleration since they are at the heart of several vision applications. We first evaluated existing implementations that either use SIMD hardware of software parallelization to speed-up execution time. The potential for acceleration with these approaches is limited to a factor of 2-4x. We then profiled these kernels to identify hot computational spots and devised a fine-grained parallelization strategy of these program parts, meant to scale to the high number of cores available in the GPPA. A proof-of-concept implementation for GPU hardware or GPPA simulation targets has demonstrated the potential for significantly higher speedups.

Concerning sharing patterns to design more efficient coherence protocols, on average, about 60% of the blocks accessed by the analyzed applications correspond to data blocks, which, unlike instruction blocks, may require coherence maintenance. Moreover, despite the fact that it is detected a high sharing degree of data blocks among cores, indeed only SW blocks (just 40% of the total number of data blocks) require coherence. Concerning data block accesses, SW blocks agglutinate 60%, on average, of the total number of data accesses.

We have also analyzed memory compression opportunities in order to reduce the NoC energy consumption. The main conclusion that we have drawn is that the most important stream of data is made of all-zeros. This means that those streams should be the primary focus for compression opportunities in the project.

Finally concerning virtualization opportunities, while not being exhaustive, since some domains are not covered in this document (e.g. medical systems), the above defines a list of

requirements applicable to virtualization in embedded systems. The allocation of these requirements to hardware and software requirements largely depends upon the capabilities provided by the hardware itself as well as the possibility of using hardware features in the contexts where non-functional properties such as certification and security are required.

Therefore, we can conclude that industrial applications present good opportunities that enable to design more efficient embedded heterogeneous mutlicore platforms if they are taken into consideration.

7. Bibliography

[Cues11] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato. Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks. In 38th Int'l Symp. on Computer Architecture (ISCA), pages 93--104, June 2011.

[Har09] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In 36th Int'l Symp. on Computer Architecture (ISCA), pages 184--195, June 2009.

[Kim10] D. Kim, J. Ahn, J. Kim, and J. Huh. Subspace snooping: Filtering snoops with operating system suport. In 19th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT), pages 111--122, Sept. 2010.

[Pug10] S. H. Pugsley, J. B. Spjut, D. W. Nellans, and R. Balasubramonian. SWEL: Hardware cache coherence protocols to map shared data onto shared caches. In 19th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT), pages 465--476, Sept. 2010.

[Hos11] H. Hossain, S. Dwarkadas, and M. C. Huang. POPS: Coherence protocol optimization for both private and shared data. In 20th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT), Oct. 2011.

[Sal04] David Salomon: Data Compression: the complete reference (3rd Edition, Springer ISBN: 0-387-40697-2) 2004

[QDNw³] QDevNet. Computer Vision (FastCV™). [Online] https://developer.qualcomm.com/mobile-development/mobile-technologies/computer-vision-fastcv.

[Cle11] J. Clemons, H. Zhu, S. Savarese, and T. Austin. Austin : MEVBench: A mobile computer vision benchmarking suite. TX : IEEE International Symposium on Workload Characterization (IISWC), 2011.

[Ros10] E. Rosten, R. Porter, and T. Drummond: FASTER and better: A machine learning approach to corner detection. IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 32, p. 105-119, 2010.

[Low04] C G. Lowe: Distinctive image features from scale-invariant keypoints. 2, Internationa Journal on Computer Vision, Vol. 60, p. 91-110, 2004.

[Bay08] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool: Speeded-up robust features (SURF). 3, Computer Vision Image Understanding, Vol. 110, p. 346-359, 2008.

[Cal10] M. Calonder, V. Lepetit, C. Strecha, and P. Fua.: BRIEF: binary robust independent elementary. Berlin : Springer-Verlag. Proceedings of the 11th European conference on Computer vision: Part IV, ECCV'10. p. 778-792, 2010.

[DBw³] DragonBoard APQ8060 Mobile Development Board Website. [Online] https://developer.qualcomm.com/develop/development-devices/dragonboard.

[OCVw³] OpenCV website. [Online] http://opencv.willowgarage.com/.

[Stu11] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, R. Siegwart: Towards a benchmark for rgb-d slam evaluation. Los Angeles, USA : s.n. Proceedings of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS), 2011.

[Zha06] Yufang Zhang, Z. X. (s.d.). The Study of Parallel K-Means Algorithm. Proceedings of the 6th World Congress on Intelligent Control and Automation, June 21 - 23, 2006, Dalian, China.

[Le10] Deguang Le, J. C. (s.d.). Parallel AES Algorithm for Fast Data Encryption on GPU. *International Conference on Computer Engineering and Technology (ICCET), 2010.*