

# parMERASA

Multi-Core Execution of Parallelised Hard Real-time Applications Supporting Analysability

## D5.5

### parMERASA Multi-core Processor Refinement and Optimisation

**Nature:**

**Dissemination Level:**

**Due date of deliverable:**

**Actual submission:**

**Responsible beneficiary:**

**Responsible person:**

**Report**

**Public**

**March 31, 2014**

**March 28, 2014**

**TUDO**

**Sascha Uhrig**

---

Grant Agreement number:

Project acronym:

Project title:

Project website address:

Funding Scheme:

Date of latest version of Annex I

against which the assessment will be made:

Project start:

Duration:

Period covered:

FP7-287519

parMERASA

Multi-Core Execution of Parallelised  
Hard Real-Time Applications  
Supporting Analysability

<http://www.parmerasa.eu>

STREP

SEVENTH FRAMEWORK PROGRAMME  
THEME ICT – 2011.3.4 Computing Systems

June 20, 2012

October 1, 2011

36 month

2013-10-01 to 2014-03-31

---

Project coordinator name, title and organisation:

Tel: + 49-821-598-2350 Fax: + 49-821-598-2359

Prof. Dr. Theo Ungerer, University of Augsburg

Email: [ungerer@informatik.uni-augsburg.de](mailto:ungerer@informatik.uni-augsburg.de)

#### Release Approval

Name	Role	Date
Sascha Uhrig	Responsible for D5.5	28 – March – 2014
Eduardo Quiñones	WP leader	28 – March – 2014
Theo Ungerer	Coordinator	28 – March – 2014

## DELIVERABLE SUMMARY

Deliverable 5.5 “parMERASA Multi-core Processor Refinement and Optimisation” covers the work done during the first part of the third phase of the project within WP5. The deliverable spans six month of work and handles the work done in task 5.5 to reach milestone MS14:

### **Task description of T5.5 (m25 :: 6m): Multi-core processor refinement and optimisation**

The research in this task has focused on the following topics:

- Refine and optimise the interconnection network.
- Refine and optimise the memory hierarchy structure.
- Refine and optimise the connection of I/O devices.

Target for month 30 is a collection of design recommendations suitable for the hard real-time parallel applications provided by WP2.

### **Conclusion of task 5.5:**

All intended subtasks have been carried out successfully and MS14 has been reached.

### **Milestone 14:**

- parMERASA architecture optimised and refined
- collection of design recommendations suitable for the hard real-time parallel applications provided by WP2 defined
- parMERASA simulator optimised and tested to allow to make it publicly available under an Open Source license at end of project
- parMERASA simulator fully integrated with the final version of the system software and the RapiTime tool and capable of executing the case studies provided by WP2. This version will be used to generate the final results of the parMERASA pilot studies. Collection of time predictable interconnection networks suitable for hard real-time parallel applications defined. Time predictable memory organisation suitable for hard real-time parallel applications defined. Time predictable connections of I/O devices suitable for hard real-time parallel applications defined. parMERASA multi-core defined and implemented in the simulator.

All objectives of MS14 have been reached. The first two parts of MS14 are documented in this deliverable; the third and fourth parts are included in D5.6.

## TABLE OF CONTENTS

1	Overview of Refinements and Optimisations .....	5
2	Refinement and Optimisations of the Interconnection Network .....	7
2.1	Strategy to Achieve Time Composability.....	7
2.2	Impact of System Integration in the parMERASA architecture.....	8
3	Refinement and Optimisations of the Memory Hierarchy.....	11
3.1	Memory Hierarchy .....	11
3.1.1	Optimisation on Shared Data Identification.....	11
3.1.2	Refined ODC <sup>2</sup> Write Strategy.....	12
3.1.3	Integration of Scratchpad Memories .....	12
3.2	Memory Map.....	12
4	Refinement and Optimisation of the Connection of I/O Devices.....	15
4.1	IRQ Framework.....	15
4.1.1	Multicast Interrupt Controller .....	15
4.2	DMA Controller .....	16
4.2.1	Functionality .....	16
5	Design Recommendations for Hard Real-time Parallel Applications .....	18
5.1	General Architecture .....	18
5.2	Interconnect .....	19
5.3	Memory System .....	19
5.4	Core Design.....	20
6	Consolidated Requirements .....	20

# 1 OVERVIEW OF REFINEMENTS AND OPTIMISATIONS

parMERASA target applications rely on *incremental qualification*, that allows each system component to be subject to formal certification (including timing analysis) in isolation and independently of other components, with obvious benefits for cost, time and effort. In current safety critical real-time systems, incremental qualification is enabled by using standardised system software architectures, such as the Integrated Modular Avionics (IMA) in the avionics domain or the AUTomotive Open System ARchitecture (AUTOSAR) in the automotive domain.

To that end, during the second phase of the project, we developed two novel concepts: *parallel Software Partitions* (pSWP) and *Guarantee Resource Partitions* (GRP) that provide incremental qualification for parallel hard real-time applications:

- pSWPs extends the functionality of *software partitions* (as defined in ARINC653 and ISO26262 standards) to allow parallel execution on multi-core processors. pSWP guarantees that parallel tasks belonging to one application cannot affect the timing (and functional) behaviour of parallel tasks belonging to other applications. Moreover, pSWPs support the communication methods defined in the IMA and AUTOSAR software frameworks, i.e. intra-partition and inter-partition.
- GRP defines a hardware execution environment composed of a cluster of processor resources, including cores, NoC resources, memory, etc., in which pSWPs run, providing the desirable time isolation properties as defined above.

Note that the GRP is, in fact, the hardware counterpart of the pSWP: while the pSWP encapsulates parallel applications to provide the desirable time isolation properties imposed by the standards, the GRP encapsulates the pSWP to provide the required time isolation guarantees at the hardware level. Here, in order to consider the impact that inter-partition communication may have on its WCET, the parMERASA memory controller and the NoC implement *freeze mechanisms*, which provide higher priority of inter-partition over intra-partition communication requests.

Deliverable D5.3 from Milestone MS13 provides further details of pSWP and GRP, and introduces NoC, memory hierarchy, and cache designs.

During the third phase of the project, we have refined the parMERASA processor design based on the knowledge and experiences acquired in the previous months by WP2 and WP3, considering fine-grained changes to improve performance and especially the worst-case performance in the specific scenarios exposed by parMERASA case studies. Concretely, we have investigated the following topics, leading to the results presented in this deliverable in the next sections:

- Better understanding of the impact that inter-partition communication has on the WCET estimation of the application after system integration. Here, we have evaluated the amount of inter-partition communication of case studies and we modified the way *time composability* is used and WCET is estimated.
- Better understanding of memory requirements of the different parMERASA case studies. Here, we have investigated the pressure that case studies put on the memory, including the concrete

amount of memory used for private as well as for shared data. Accordingly, the memory map has been adjusted and local scratchpad memories have been added to hold private data.

- Better understanding of I/O communication and requirements. For transportation of larger amounts of data, a DMA controller suitable for many-core systems has been integrated and the interrupt system has been extended to meet all the requirements of the pilot studies.
- First experiences with WCET analysis led to better understanding of OTAWA's analysis features, i.e. strengths and weaknesses of the static analysis. These experiences triggered a revised ODC<sup>2</sup> data cache technique and the decision to integrate scratchpad memories.

## 2 REFINEMENT AND OPTIMISATIONS OF THE INTERCONNECTION NETWORK

As stated in deliverable D5.3, the parMERASA architecture differentiates two types of communication methods: intra- and inter-partition, both implemented through memory accesses. These two methods can potentially conflict in the NoC (either clustered or regular designs) and the memory, which may imply long latencies for accessing application's shared and private data.

During the third phase of the project, we have evaluated the impact that intra/inter-partition requests may have on the WCET estimation of an application after system integration, considering the characteristics of parMERASA case studies. This analysis allows us to better understand if the parMERASA architecture, and more concretely the *freeze mechanism* described in deliverable D5.3, is suitable for the parMERASA case studies.

### 2.1 Strategy to Achieve Time Composability

When computing the WCET estimation of an application, one may consider the worst-case delay that each memory request can suffer due to intra- and inter-partition communication. Such an approach, proposed in the MERASA project and presented in [1], considers the maximum delay (a.k.a. upper bound delay or UBD) that an access to a shared resource (in our case the NoC and the memory) may suffer due to interferences.

The resultant WCET estimation of considering UBD to each communication request fulfils the time composability property, i.e. the WCET estimation of a system component analysed in isolation remains unchanged after system integration (see equation (1)). Such an approach is shown in Figure 1, in which the WCET estimation of application A computed in isolation, i.e. in a system in which A has exclusive access to shared resources, remains the same after system integration, i.e. executing A simultaneously with applications B, C and D.

$$WCET_{\text{integration}} = WCET_{\text{isolation}} \quad (1)$$

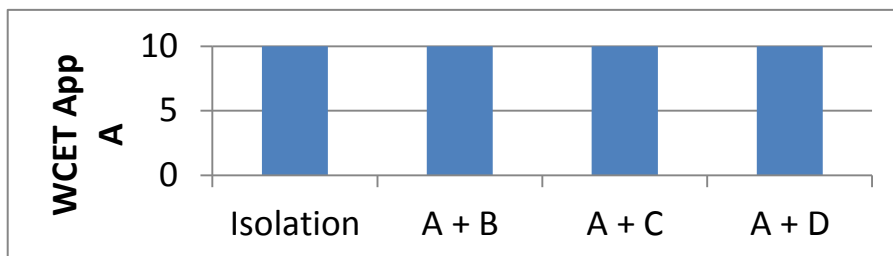


Figure 1: The WCET estimation of application A computed in isolation remains unchanged when being executed simultaneously with applications B, C and D.

<sup>1</sup> MERASA: Multi-Core Execution of Hard Real-Time Applications Supporting Analysability. In the IEEE Micro 2010, Special Issue on European Multicore Processing Projects, Vol. 30, No. 5, Oct. 2010 Theo Ungerer et.al.

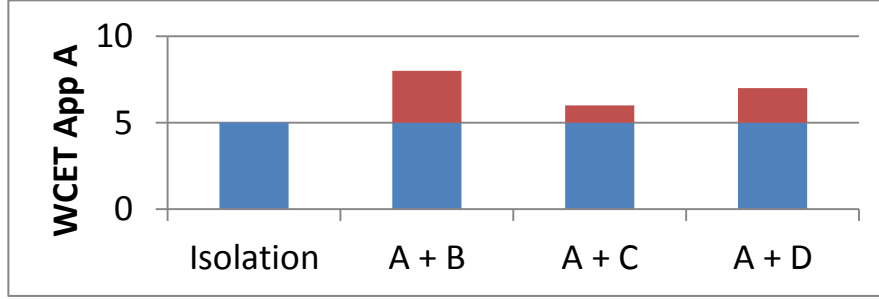


Figure 2: The WCET estimation of application A computed in isolation increases at system integration depending on the application A is being executed simultaneously.

The parMERASA processor design instead follows a different approach. The impact that the system integration may have on the WCET estimation of each system component is considered as an additive factor at the time the system is integrated. Such a property, also known as *time compositionality*, can be expressed as shown in equation (2) (see deliverable D5.3 for further details), in which  $\Delta_{\text{inter}}$  represents the additive value:

$$\text{WCET}_{\text{integration}} = \text{WCET}_{\text{integration}} + \Delta_{\text{inter}} \quad (2)$$

Figure 2 shows the WCET estimation increment (in red) that application A suffers after system integration, taken into account equation (2). Such an increment is introduced by inter-partition communication requests delaying intra-partition communication using the *freeze mechanism* implemented in the NoC and memory.

## 2.2 Impact of System Integration in the parMERASA architecture

In order to better understand the impact of system integration introduced by the parMERASA architecture on the WCET estimation of system components that form the parMERASA case studies, we have analysed the inter-partition communication of each case study and generated a synthetic benchmark that resembles the amount of intra and inter-partition communication that may potentially conflict on the NoC and the memory, and so increase the WCET estimation.

Figure 3 shows the structure of the synthetic benchmark composed of two applications, A and B, executed in two GRP of 4-cores each in a pipeline fashion. Note that inter-partition requests sent by application B interfere with previous pipeline stages of application A, potentially affecting its WCET estimation.

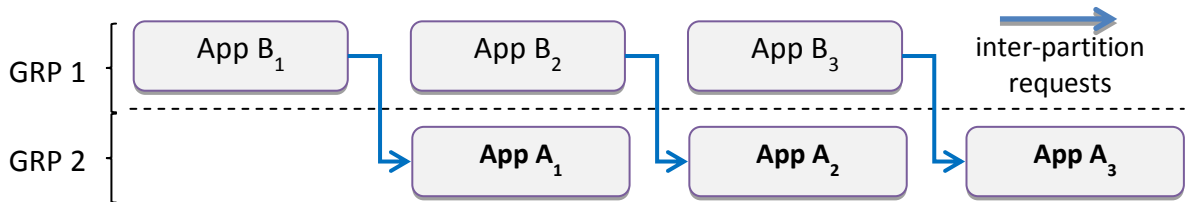


Figure 3: Synthetic benchmark resembling the intra- and inter-partition communication conflicts existing in the parMERASA case studies. Application B sends data to A in a pipeline fashion.



Hence, the WCET estimation suffered by application A will depend on the number of inter-partition requests sent by B and the number of intra-partition request of A being blocked by the freezing mechanisms in the NoC and main memory.

Table 1 shows the characteristics of application A and B in terms of percentages of inter-partition requests of B that may potentially impact on intra-partition requests of A. In other words, it provides an estimation of the intra-partition requests of A that will be delayed by B due to the highest priority of inter-partition request, i.e.  $\Delta_{inter}$ . The table considers two scenarios: One in which the percentage resembles the parMERASA case studies (*scenario I*) and the other in which the number of inter-partition request of the parMERASA case studies has increased by 100x (*scenario II*).

	parMERASA case study; Scenario I	Increment of 100x of inter-partition request; Scenario II
<b>Application B over A</b>	0,44%	44,14%

Table 1: Percentage of inter-partition requests over intra-partition of application B over A, executed in a pipeline fashion as shown in Figure 3

Figure 4 and Figure 5:5 show the WCET estimation increment of A due to inter-partition requests of B, considering scenarios I and II, respectively (see Table 1). WCET estimations are obtained by executing application A in a 4-core GRP and normalized to the WCET estimation of application A executed sequentially in a single-core without suffering any delay due to inter-partition communication.

Moreover, the two figures present the two different strategies shown in section 2.1 to consider the impact of inter-partition requests in the WCET estimation:

1. Assuming the impact of inter- and intra-partition communication at WCET analysis time of the application, i.e. a la MERASA (labelled as *MERASA*) or
2. Assuming the impact of inter-partition communication at system integration, i.e. a la parMERASA (labelled as *parMERASA*).

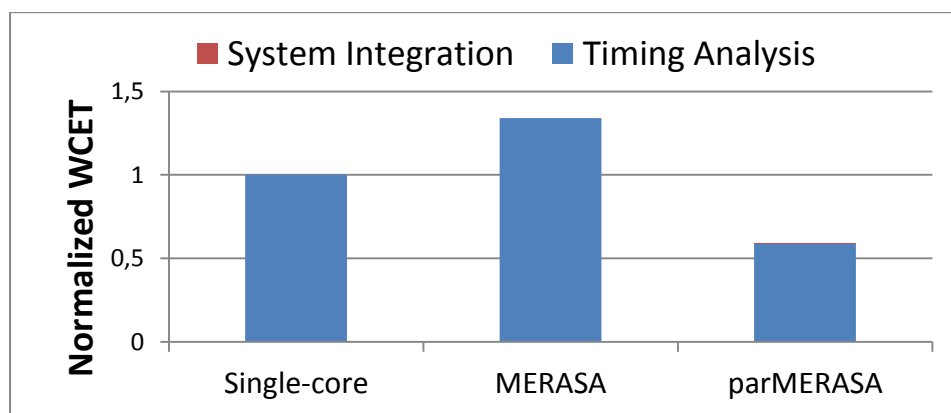


Figure 4: WCET increment with respect WCET estimation of single-core execution of application A executed under scenario I shown in Table 1.

Figure 4 shows the WCET estimation increment of application A under scenario I. The communication among applications is very low in parMERASA case studies. As a result, considering inter-partition communication at system integration time (*parMERASA approach*) reduces the WCET estimation of A by 40% with respect of single-core execution. It is important to remark that the impact of inter-partition requests is very low (less than 1%). Instead, if the impact of inter-partition is considered at WCET analysis time of application A, the WCET estimation increases by 30% with respect of single-core execution.

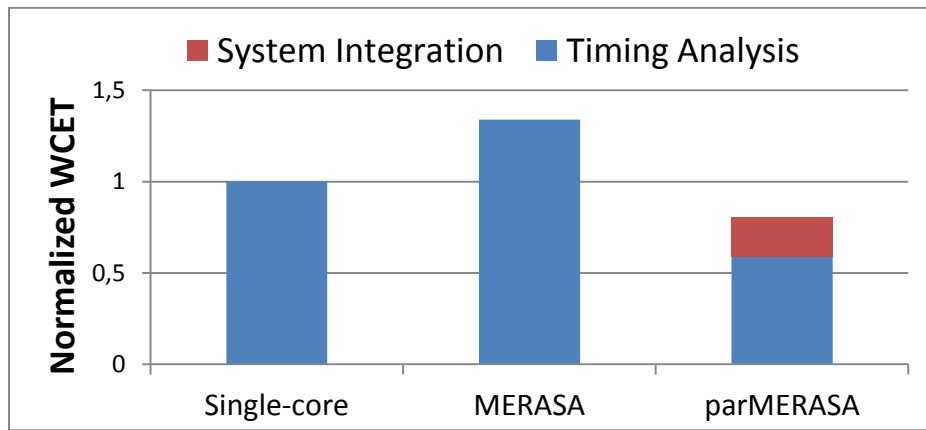


Figure 5: WCET increment with respect WCET estimation of single-core execution of application A executed under scenario I shown in Table 1.

Similarly, Figure 5 shows the normalized WCET estimation increment of application A under scenario II. In this case, because the number of inter-partition requests increase, the impact of system integration increases as well (red bar), observing a reduction of WCET estimation of A of only 20% with respect of single-core execution. Note that in this case, the number of inter-partition communication have increased by 100x.

Note that in the two figures, the *MERASA* bar remains the same, because the WCET estimation of A in this case is independent of the workload in which the application runs.

To sum up, the parMERASA architecture in general, and the NoC and main memory in particular, provides suitable mechanisms for the case studies considered in the project in which the amount of communication among system components is moderate. That is, by considering the impact that inter-partition communication has at system integration, WCET estimation is reduced considerably with respect of considering it at WCET analysis time, taking actual benefit of the parallel execution.

## 3 REFINEMENT AND OPTIMISATIONS OF THE MEMORY HIERARCHY

All the communication methods considered in parMERASA are implemented through memory accesses. Hence, inter-partition communication, intra-partition and local partition memory accesses may potentially conflict in the NoC (either in clustered or regular designs) and the memory, which implies long latencies for accessing application's shared and private data. As a result, a suitable and time predictable memory hierarchy design is required. This section presents refinements and optimisations of the memory hierarchy for the parMERASA architecture proposed in deliverable D5.3. These modifications are required because the original proposals have led to suboptimal WCET results.

### 3.1 Memory Hierarchy

The original parMERASA cache hierarchy comprises two private caches per core, i.e. an instruction and a data cache. A second level of caches is not foreseen because of the difficulties related to static WCET analysis. The instruction cache is a regular cache in which cache coherence is not required. This is not the case, however, for the data cache that needs to provide coherent accesses to shared data. The proposed ODC<sup>2</sup> cache is able to provide coherent accesses to data which is identified as shared at runtime. Moreover, cache support is provided to private data as well as to shared data to some limited extent. Moreover, the ODC<sup>2</sup> works without any *inter-cache communication* and the state of a cache solely depends on the behaviour of the related core.

The basic idea of ODC<sup>2</sup> is to hold shared data only as long as necessary and to force (re-)loading of possibly modified shared data at the time of use. Hereby, use does not mean individual accesses but rather instruction sequences like critical sections which access shared data. Two major optimizations have been conducted in the optimisation phase, which are described in the following subsections 3.1.1 and 3.1.2. In 3.1.3 a refinement of the memory hierarchy is presented that reduces distance between cores and private data.

#### 3.1.1 Optimisation on Shared Data Identification

To provide cache coherence, the ODC<sup>2</sup> applies a special handling of cache lines during instruction sequences with accesses to shared data. In these sequences the cache is switched to the *shared mode*. The beginning and the end of those sequences is denoted by ODC<sup>2</sup> control instructions, positioned with respect to the synchronisation techniques (see deliverable D5.3). During *shared mode*, shared data as well as private data is accessed and loaded into the cache. Since the ODC<sup>2</sup> cannot distinguish between accesses to private and shared data directly, also newly loaded cache lines with private data will be (wrongly) marked as “potentially shared”. This leads to a needless invalidation (and write-back) of private cache lines in the restore procedure and an increase of the cache miss rate. To avoid this additional overhead, the optimised ODC<sup>2</sup> provides the *Address Checking* functionality which is based on the identification of accesses to shared data regions.

The memory layout of the parMERASA simulator allows the identification of shared data regions without additional effort. In the memory map (see Section 3.2) separate memory sections for private and shared data are specified. The *Address Checking* functionality uses the address range of the shared data section to identify accesses to shared data by the target address. This way, solely cache

lines loaded by accesses to a shared data section will be marked as “shared”. Private data loaded during the *shared mode* will remain in the cache after the *shared mode* is left.

### 3.1.2 Refined ODC<sup>2</sup> Write Strategy

A precondition of the ODC<sup>2</sup> is that a cache line must not contain private and shared data in parallel. The condition is fulfilled by the separated memory sections for private and shared data. But regarding the mapping of shared data, a cache line may contain two different shared variables accessed by two different cores (false sharing). If both cores access the cache line simultaneously (e.g. disjunct accesses to neighbouring elements of an array) and the caches operate in Write-Back strategy, cache coherence cannot be permitted by the regular ODC<sup>2</sup> mechanism. To guarantee coherent accesses to shared data in applications where false sharing is possible, ODC<sup>2</sup> provides an optional Write-Through strategy during the *shared mode* for all cache lines classified as *shared*. Using the ODC<sup>2</sup> with Write-Through write strategy, false sharing does not affect the coherence of shared data accesses.

Usually with Write-Through, the amount of accesses to the main memory is significantly higher compared to Write-Back. Thus, a performance slowdown may be considered. But it has to be stated, that with ODC<sup>2</sup> in Write-Through mode, write accesses during the private mode will be still performed with Write-Back. And in combination with the *Address Checking* functionality, only write accesses to shared data will be performed with Write-Through. Private data is still written using a write-back policy. Furthermore, there is no need to write back modified cache lines at the end of the *restore procedure*, which simplifies and also optimises WCET estimation and results.

### 3.1.3 Integration of Scratchpad Memories

Scratchpad memories permit low latency accesses to user allocated data. Compared to a cache memory, where access latencies depend on whether a cache hit or a cache miss occurs, accesses to a scratchpad do not differ in latency. A static WCET estimation highly benefits from the increased predictability of access times with scratchpad memories. Of course, this is only the case for instructions or application data that fits into the scratchpad.

To achieve higher average and worst case performance, we integrate instructions as well as data scratchpads in the memory hierarchy. The scratchpad can be used in combination (i.e. both, scratchpad and cache for accesses to external memory in parallel) or instead of cache memory. The execution of tasks, whose code or private data can be mapped into the scratchpad will profit from the improved access latencies. Regarding shared data, ODC<sup>2</sup> cache can be used to access shared variables allocated in the main memory, while private code and data will be allocated in scratchpads.

## 3.2 Memory Map

As of month 24 the memory demands of the industrial applications were clarified. Table 2 lists the maximum required memory sizes for the private instruction memory per core, the private data (including stack) per core, the total shared data in one GRP, the maximum number of cores per GRP and the currently used number of GRPs.

Application	Private code	Private data	Shared data	#Cores	#GRPs
Bauer	900K	700K	110K	16	1
Denso	256K	256K	256K	8	1
3DPP	150K	20K	40K	8	1
StereoNav	256K	64K	512M	7	1
<b>Resulting Memory Map</b>	<b>1M</b>	<b>1M</b>	<b>512M</b>	<b>16</b>	<b>4</b>

Table 2: Memory and core characteristics of the pilot study applications

These requirements revealed some problems of the original memory map presented in D5.3. In detail these are:

1. The shared memory address space is too small for the 512 MB required by the Stereo Navigation.
2. The memory addresses of one GRP are continuously, therefore the beginning of the shared address space depends on the number of cores per GRP.
3. The mapping addresses for I/O devices reserve much of the address space without using it.

The problems were resolved by a new memory map (Figure 6) with the following features:

1. The maximum number of GRPs is reduced from 24 to 4, enabling up to 768 MB of address space per GRP.
2. The beginning of the shared memory of a GRP is put at a fixed address. Consequently, the addresses do not change when the number of cores per GRP is changed. This limits the maximum private address space per core to 16 MB, since at least 16 cores per GRP shall be provided and the address space for private addresses is 256 MB per GRP.
3. All mapping addresses for I/O devices were put into one single address space from 0xE0000000 to 0xFFFFFFFF, instead of assigning them to the address spaces of the GRP they belong to. This concentration reduces the number of reserved addresses and leaves more address space for the memory requirements of the GRPs.
4. The TIC mapped addresses are also put into one contiguous block in the I/O segment instead of an individual address space per GRP.
5. The logical address space is too small to map all shared data of a GRP, therefore only the lowest 16 MB of the cached and the uncached (required for synchronisation variables) shared memory are mapped to the logical addresses.

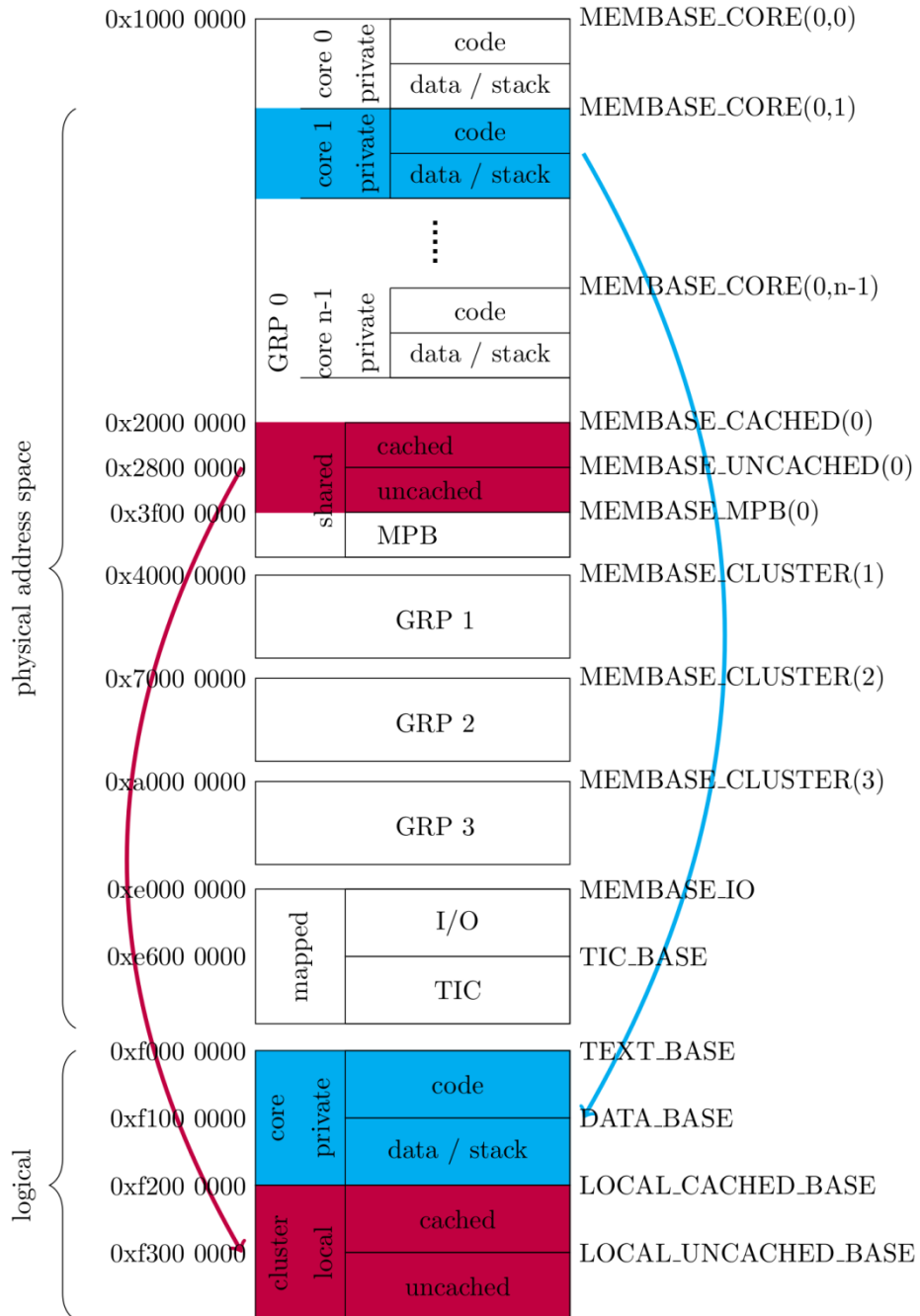


Figure 6: New memory map with extended memory sizes

## 4 REFINEMENT AND OPTIMISATION OF THE CONNECTION OF I/O DEVICES

### 4.1 IRQ Framework

The interrupt framework consisting of SIC/TIC interrupt controllers as presented in D5.3 is able to perfectly deal with 1:1 relationships of interrupt sources (I/O devices) and interrupt destinations (cores). But, there is another requirement induced by DNDE, leading to CR5.9: *Globally synchronized interrupts can be triggered by at least two independent external signal sources*. In order to fulfil this requirement, the Multicast Interrupt Controller (MIC) has been introduced.

#### 4.1.1 Multicast Interrupt Controller

From the case studies there was the requirement of a mechanism to synchronize several cores at once and the requirement of several interrupts reaching different cores at the same time (see CR5.9). In order to integrate the solution for this additional requirement into the existing interrupt framework, a third interrupt controlling instance was developed – the Multicast Interrupt Controller (MIC).

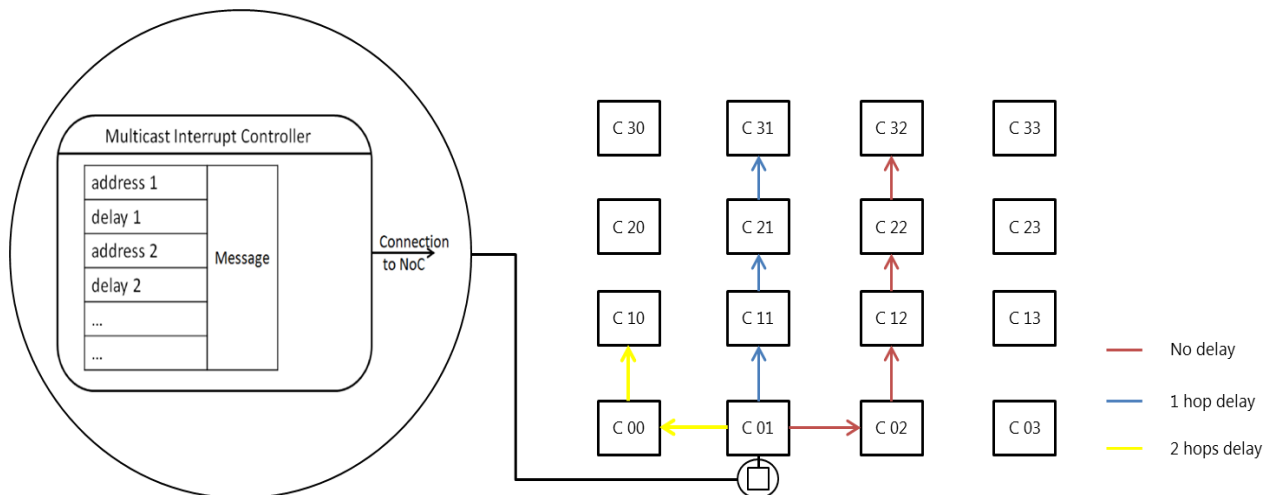


Figure 7: MIC Module in a 4x4 Cluster Mesh

##### 4.1.1.1 Basic Idea

In the TIC/SIC interrupt framework all peripheral devices are connected to a SIC. If a device creates an interrupt it is forwarded through the NoC. Unfortunately, only a single interrupt message per interrupt request can be sent by a SIC. In order to be able to send multiple messages, the MIC has been introduced. Here, it has to be mentioned that still only a single network message can be sent by a node at a time, which complicates the synchronous generation of interrupt signals at several cores at first glance. But, since the distances and accordingly the network delays from the MIC to the destinations are different, the interrupt messages have to be sent at different times. Figure 7 shows a situation in which a MIC located in C01 is sending three interrupt messages to the nodes C10, C31, and C32, respectively. Due to the different delays, the messages have to be sent with time-shift in order to reach the destinations synchronously.

The MIC internally holds a table with data tuples containing destination addresses and send delays. In case of the parMERASA simulator, this table is filled during program initialization with the help of an initialization function provided by the common kernel library. The delays need to be determined with the knowledge of the concrete network implementation and the given network latencies.

In case the MIC receives an interrupt message from a SIC it starts sending the specified network interrupt messages to the given destinations. Hereby, the internal table is handled entry by entry with the given delays between two consecutive messages.

#### **4.1.1.2 Cascading**

The signals will arrive at their destination synchronously if all path lengths are different. If there are multiple paths with the same length the presented way of synchronous interrupts will not work directly. Instead, two MICs in different nodes have to be cascaded, which will allow synchronous interrupts also in this case. Cascading of MICs can be simply implemented by sending an interrupt message from the master MIC to a slave MIC (instead of a destination TIC).

Overall, the presented SIC/MIC/TIC framework allows a simple and highly flexible environment for interrupt management and routing in many-core systems.

## **4.2 DMA Controller**

Due to the need of transporting larger portions of data without the interaction of a core, a DMA controller has been integrated into the parMERASA system. Concretely, there is the need from the HON stereo navigation application to transport images from an I/O device (camera connection) to memory.

### **4.2.1 Functionality**

The parMERASA DMA controller is perfectly integrated into the SIC/MIC/TIC interrupt system. The controller is able to transfer all I/O data from an I/O device to the memory and afterwards notify the core by an interrupt. The parMERASA DMA controller internally holds the typical DMA controller information like the destination memory address for the data, the data length, and the address of the source I/O module. Moreover, the address of the destination TIC for the final interrupt is stored. The system level software provides functions for the initialization of the DMA controller.



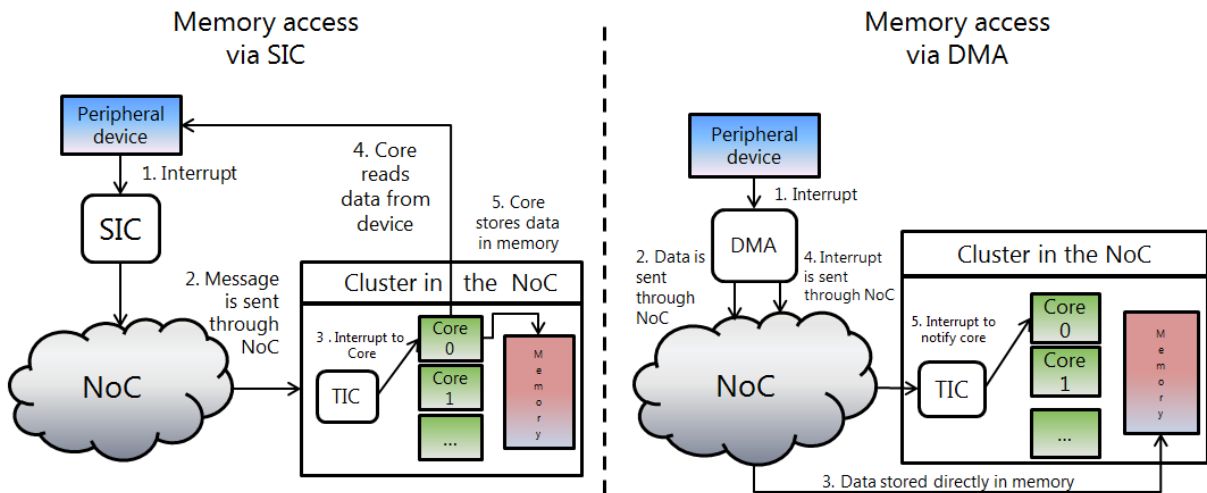


Figure 8: Memory accesses with different modules

Figure 8 shows the difference between memory accesses using the DMA and memory accesses using the alternative SIC/TIC interrupt technique with software support for data transfer. Like in state-of-the-art DMA transfers the main advantage of using DMA in parMERASA is that the core is not interrupted for each I/O access individually but only once after the transfer is completed. This advantage is much more important in a many-core system compared to a single core system because the interrupt system of the many core is much more complex and hence, its latency is higher. Moreover, data transfers conducted by a core to/from an I/O device are much more time consuming than in a single core system. Caching does not help in this situation because I/O data might not be available in means of cache lines and/or aligned to cache lines. Accordingly, only uncached read accesses are possible which can transport only a small amount of data resulting in a long blocking time of the core.

## 5 DESIGN RECOMMENDATIONS FOR HARD REAL-TIME PARALLEL APPLICATIONS

The refined parMERASA processor architecture is based on the knowledge and experiences acquired during the first two phases until month 24 by WP2 and WP3. Several modifications have been necessary to allow optimised implementations of the parallel case studies as well as an improved static WCET analysis.

The basic architecture as well as the optimisations until month 30 lead to the hardware design recommendations presented below.

### 5.1 General Architecture

The use of a multi/many-core system very often comes along with the intention to combine several functionalities on a single chip. Since each functionality is bringing its own requirements with respect to performance, memory (size and utilization), and input/output, a collection of completely different requirements must be fulfilled by the system. In addition to these technical requirements, the functionalities will bring a certain level of criticality. This leads to the requirement of *incremental qualification*, which allows each functionality to be subject to formal certification (including timing analysis) in isolation and independently from other functionalities.

In current safety critical real-time systems, incremental qualification is enabled by using standardised system software architectures, such as the Integrated Modular Avionics (IMA) in the avionics domain or the AUTomotive Open System ARchitecture (AUTOSAR) in the automotive domain. In current state-of-the-art, these standards are focusing on single core systems and cannot (or should not) be mapped one-to-one to multi/many-core systems.

Both standards, IMA and AUTOSAR, rely on strict partitioning/isolation of different functionalities. In order to provide parallel execution capabilities while still guaranteeing incremental qualification properties, parMERASA introduced the *parallel Software Partition* (pSWP), which is an extension of the original *software partition* (SWP) in IMA and AUTOSAR.

It is important to remark that parMERASA advocates for *time compositionality* properties, in which the impact of application communication is considered at system integration as an additive factor (see Section 2). As a result, the basic ideas of these standardized software architectures can be mapped to the hardware design of the processor. This partitioning can be transferred to hardware design by supporting a clustered architecture which can guarantee only predictable and innocent interferences. We called this technique *Guarantee Resource Partitions* (GRP). Here, the clustering allows guaranteeing particular resources (cores, memory, communication facilities) to pSWPs. Moreover, interferences between clusters are bound and can be statically taken into account during incremental qualification. In principle, it does not matter if the clusters are statically provided by hardware design or virtually by a suitable interconnect and routing technique. More information on how these guarantees can be given is shown in the following sections.

## 5.2 Interconnect

The interconnect is mainly responsible for clustering and guaranteeing the freedom from interference. Here, freedom from interference does not mean the complete absence of interferences but the fact that interferences can be statically determined and bound.

In parMERASA, we have investigated two types of interconnects, a mesh-based and a tree-based system. Both techniques can fulfil the requirements for setting-up clusters which allow incremental qualification, in principle.

Since a mesh shows a homogeneous regular structure, clusters are not directly supported. Instead, with the help of a suitable routing, like X/Y routing, virtual clusters can be implemented. Nodes that are located nearby can form a cluster because intra-cluster communication will never leave the bounding box of the corresponding nodes i.e., the cluster.

Moreover, in order to consider the impact of system integration, the parMERASA architecture introduces the *freeze mechanisms*, in which the interconnect (as well as the main memory) implements virtual channels that allow bandwidth and timing guarantees (limits). These guarantees are necessary to distinguish and to separate communication within a cluster (intra-cluster communication) and between different clusters (inter-cluster communication). As a result, the effects of inter-cluster communication crossing a cluster can be bound by the necessary virtual channels, the interferences between a cluster and the rest of the system are also bound.

Interconnection mechanisms have been evaluated based on the characteristics of parMERASA case studies evaluated during the third phase of the project, demonstrating its effectiveness as shown in Section 2.

## 5.3 Memory System

Relying only on local caches is not optimal as it turned out during the first two project phases. Instead we strongly recommend also the integration of local scratchpad memories for code as well as for (private) data. The reason why caches are not sufficient is the required warm-up phase and the fact that static cache prediction is not trivial and may lead to large uncertainties. Here, the warm-up phase needs to be taken into account during the static WCET analysis and lead to results including the warm-up. In reality, this warm-up is necessary only at the initialisation phase of the system but not during actual system execution time. Unfortunately, this behaviour cannot be mapped to the static WCET analysis because of difficulties with the cache analysis. A comprehensive cache analysis requires a perfect address analysis which is currently not possible, at least with respect to the examined case studies of WP2. In summary, scratchpad memories (instruction and data) with statically known content are simpler to analyse and bring more advantages to the WCET analysis, even if their size is strongly limited.

Nevertheless, caches are important because local scratchpads can hold only limited amounts of instructions and data. To access further instructions/data, caches are needed to hide the long network latencies that arise on accesses to (external/shared) memories. These latencies exceed by far the latencies from bus-based systems and would bring down system performance if no caches are used.

If local data caches are implemented to support accesses to shared memories, these caches should provide a technique to allow coherent accesses to shared data. Here, it is necessary that the used technique does not induce inter-cache communication like all well-known cache coherence protocols. This inter-cache communication would bring unpredictable interferences between cores or even clusters shooting down all the efforts spent on isolation of clusters. The proposed On-demand Coherent Cache (ODC<sup>2</sup>) provides a technique for coherent accesses to shared data without any inter-cache communication.

## 5.4 Core Design

parMERASA is not focusing on processor core design, so the recommendations on this domain are limited. Nevertheless, there is one strong recommendation to core designers, which is a major precondition for executing all the parallel pilot studies: There is a need for dealing with synchronization variables. Beside the fact that such synchronization variables need to be read and modified in an atomic operation, local caches further complicate this issue.

The parMERASA system solves this problem by implementing a new *load-modify-store* instruction which initiates an atomic *read-and-write* operation on the memory controller. This operation completely bypasses local caches so that cache coherence is not an issue here. Moreover, the timing of this operation is equal to that of common read or write operations which means no effect on the timing analysis of other cores connected to the same memory.

## 6 CONSOLIDATED REQUIREMENTS

The requirements derived from the three target application domains have been condensed into 20 Consolidated Requirements for work package 5 (CR5.1 – CR5.20) which have been considered as the baseline for the parMERASA hardware architecture design. Moreover, these CR5s have been used as success criteria to demonstrate the suitability of the parMERASA hardware architecture to be used for the targeted sample applications. All CR5s except for CR5.6 and CR5.9 have been fulfilled until month 24 and are finalized in deliverable D5.3. The two mentioned CR5s have been postponed to the optimisation phase.

	Description	Compliance of current architecture	State
CR5.6	It is possible to change between several communication schedules.	Different communication methods are considered by intra- and inter-partition communication, which makes consideration of particular schedules not necessary.	√
CR5.9	Globally synchronized interrupts can be triggered by at least two independent external signal sources.	The extension of the SIC/TIC concept by the Multicast controller (MIC) implements the functionality of this requirement	√