Design process for the optimization of embedded software architectures on to multi-core processors in automotive industry

Wenhao Wang

Thèse de Doctorat

En vue de l'obtention du grade de

Docteur de l'Université de Cergy-Pontoise

Design process for the optimization of embedded software architectures onto multi-core processors in automotive industry

---

Présentée et soutenue par

Wenhao WANG

École doctorale : Science et Ingénierie (SI)

Unité de recherche : Equipes Traitement de l'Information et Systèmes (ETIS)

Soutenance soutenue le 10 Juillet 2017

Devant le jury composé de :

M.	Pierre BOULET	Professeur	Université de Lille	Rapporteur
M.	Laurent GEORGE	Professeur	ESIEE Paris	Rapporteur
M.	Daniel CHILLET	Professeur	Université de Rennes 1	Examinateur
Dr.	Fabrice GRAVEZ	Ingénieur	Continental AG Cergy	Examinateur
Dr.	Sylvain COTARD	Ingénieur	KRONO-SAFE Orsay	Invité
M.	Olivier ROMAIN	Professeur	Université de Cergy-Pontoise	Directeur
M.	Benoît MIRAMOND	Professeur	Université Nice Sophia Antipolis	Directeur adjoint
M.	Fabrice CAMUT	Ingénieur	Valeo GEEDS Créteil	Encadrant industriel

Acknowledgement

Les travaux présentes dans ce mémoire ont été réalisés dans le cadre d'une thèse CIFRE entre le Labo d'Equipes Traitement de l'Information et Systèmes (ETIS), unité de recherche commune au CNRS (UMR 8051), et du Groupe Valeo.

Je tiens à remercier M. Benoît Miramond,, mon directeur de thèse, pour la confiance qu'il m'a témoigné et l'opportunité qu'il m'a donnée. Je lui remercie pour ses conseils, son soutien et son disponibilité. Ses expériences et son expertise ont été essentielles au bon déroulement d mes travaux.

Je tiens à exprimer toute ma gratitude à M. Fabrice Camut, mon encadrant industriel chez Valeo, pour son investissement durant ma thèse, son soutien continu, sa disponibilité et ses conseils avisés. Je tiens à lui remercier pour le temps qu'il m'a consacré et sa pédagogie qui m'ont été très utiles.

Je remercie également M. Sylvain Cotard, grâce à qui ma thèse pouvait démarrer sans encombre dans l'équipe « multicœur ».

J'exprime ma profonde gratitude à M. Daniel CHILLET, professeur à l'Université de Rennes 1 pour m'avoir fait l'honneur de présider mon jury de thèses. Je remercie également M. Pierre BOULET, professeur à l'Université de Lille, et M. Laurent GEORGE, professeur à l'Ecole ESIEE Paris, pour avoir rapporté mon manuscrit de thèse. Enfin, je remercie également Oliver Romain, professeur à l'Université de Cergy-Pontoise et Dr. Fabrice Gravez, ingénieur chez Continental AG Cergy d'avoir de participer à mon jury de thèse.

Je tiens aussi à remercier mes collègues chez Valeo et à l'ETIS, Ludovic Pintard, Bruno Montane, Yunfei GAO, Sascha TENKLEVE, Laurent Benchadi, Jerome Hugot, Jean-Jaque Cabrera, avec qui j'ai pu passer trois ans dans une ambiance agréable.

Enfin, je remercie ma famille et mes amis qui sont essentiels dans la réussite de mes projets et l'accomplissement de ce travail. Je remercie surement mon pote Yohan, doctorat à l'ETIS, avec qui j'ai partagé des moments sportives et sympas. Je remercie mes parents pour leur soutien.

Abstract

The recent migration from single-core to multi-core platforms in the automotive domain reveals great challenges for the legacy embedded software design flow. First of all, software designers need new methods to fill the gap between applications description and tasks deployment. Secondly, the use of multiple cores has also to remain compatible with real-time and safety design constraints. Finally, developers need tools to assist them in the new steps of the design process. Face to these issues, we proposed a method integrated in the AUTOSAR (AUTomotive Open System ARchitecture) design flow for partitioning the automotive applications onto multi-core systems. The method proposes the partitions solution that contains allocation of application as well as scheduling policy simultaneously. The design space of the partitioning is explored automatically and the solutions are evaluated thanks to our proposed objective functions that consider certain criteria such as communication overhead and global jitters. For the scheduling part, we present a formalization of periodic dependencies adapted to this automotive framework and propose a scheduling algorithm taking into account this specificity. Our defined constraints from real-time aspect as well as functional aspect make sure the applicability of our method on the real life user case. We leaded experiments with a complex and real world control application onto a concrete multi-core platform.

Résumé

La migration récente des plateformes mono-cœur vers le multi-cœur, dans le domaine automobile, révèle de grands changements dans le processus de développement du logiciel embarqué. Tout d'abord, les concepteurs de logiciel ont besoin de nouvelles méthodes leur permettant de combler le fossé entre la description des applications (versus Autosar) et le déploiement de tâches. Deuxièmement, l'utilisation du multi-cœur doit assurer la compatibilité avec les contraintes liées aux aspects temps-réel et à la Sûreté de fonctionnement. Au final, les développeurs ont besoins d'outils pour intégrer de nouveaux modules dans leur système multi-cœur. Confronter aux complexités ci-dessus, nous avons proposé une méthodologie afin de repartir, de manière optimale, les applications sous forme de partitions logiques. Nous avons ainsi intégré dans notre processus de développement, un outil de distribution des traitements d'un système embarqué sur différents processeurs et compatible avec le standard AUTOSAR (AUTomotive Open System ARchitecture). Les solutions de partitionnement traitent simultanément l'allocation des applications ainsi que la politique d'ordonnancement. Le périmètre d'étude du partitionnement est automatique, les solutions trouvées étant évaluées par des fonctions de coût. Elles prennent aussi en compte des critères tels que, le coût de communication inter-cœur, l'équilibrage de la charge CPU entre les cœurs et la gigue globale. Pour la partie ordonnancement, nous présentons une formalisation des dépendances sous forme périodiques pour répondre au besoin automobile. L'algorithme d'ordonnancement proposé prend en compte cette spécificité ainsi que les contraintes temps-réel et fonctionnelles, assurant l'applicabilité de notre méthodologie dans un produit industriel. Nous avons expérimenté nos solutions avec une application de type contrôle moteur, sur une plateforme matérielle multi-cœur.

Content

List of Figures

List of Tables

Glossary

ACO	Ant Colony Optimization
ADAS	Advanced Driver Assistance System
AMP	Asymmetric Multi-Processing
API	Application Programming Interface
ARXML	AutosaR XML
ASIL	Automotive Integrity Safety Level
AUTOSAR	AUTomotive Open System ARchitecture
AWF	Almost Worst-Fit
BF	Best-Fit
BSW	Basic SoftWare
CA	Certification Authority
CAE	Computer Aided Engineering
CAN	Control Area Network
CO	Combinatorial Optimization
CPU	Central Processing Unit
CX	Cycle Crossover
DAG	Directed Acyclic Graph
DAS	Distributed Application Subsystem
DM	Deadline Monotonic
DMA	Direct memory access
DRE	Data Received Event
DSE	Design Space Exploration
DSP	Digital Signal Processor
E/E	Electrical/Electronic
EA	Evaluation Algorithms
EC	Evolutionary Computation
ECU	Electronic Control Unit
EDF	Earliest Deadline First
EMS	Engine Management System
EP	Evolutionary Programming
ES	Evolutionary Strategies
FAS	Feedback Arc Set
FF	First-Fit
GA	Genetic Algorithms

HiL	Hardware in the Loop
HW	HardWare
I/O	Input/Output
IDE	Integrated Development Environment
ILP	Integer Linear Programming
IMA	Integrated Modular Avionics
IOC	Inter OS-Application Communicator
IRV	Inter Runnable Variables
LIN	Local Interconnect Network
LLF	Least Laxity First
MBD	Model-Based Design
MCAL	MicroController Abstraction Layer
MMU	Memory Management Unit
MoC	Model of Computation
MOSA	Multiobjective Simulated Annealing
MPU	Memory Protection Unit
MSE	Mode Switch Event
NF	Next-Fit
NP	Non-deterministic Polynomial
NSGA	Non-dominated Sorting Genetic Algorithm
OEM	Original Equipment Manufacturer
OIE	Operation Invoked Event
OS	Operating System
OSEK, OSEK/VDX	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug, "Open Systems and the Corresponding Interfaces for Automotive Electronics"
PMX	Partially Matched Crossover
P-OS	Partitioned OS
QAP	Quadratic Assignment Problem
RM	Rate Monotonic
RTE	RunTime Environment
RTOS	Real-Time Operating System
SA	Simulated Annealing
SDF	Synchronous Data Flow
SMP	Symmetric Multi-Processing
SMT	Satisfiablity Modulo Theories
SPB	System Peripheral Bus
SRI	Shared Resource Interconnect
SWAT	SoftWare Allocation Tools

SWC	SoftWare Component
TEV	Timing Event
TS	Tabu Search
TSP	Traveling Salesman Problem
VFB	Virtual Functional Bus
WCET	Worst Case Execution Time
WDGM	WatchDoG Module
WF	Worst-Fit

Chapter 1 Introduction

1.1 E/E automotive system

Over the past 20 years, a new way to design vehicles has been adopted in the automotive industry. Many of the conventional mechanical and hydraulic control systems in vehicle have been replaced by mechatronic system. The design of automotive systems is no longer one single discipline issue; instead, it involves a multidisciplinary field of expertise such as mechanical engineering, electronics, computer science, telecommunications technologies and system/control engineering. The emergence of mechatronic authorizes the design of automatic systems in order to control complex systems and also allows to minimize the development cost compared to odd pure mechanical systems.

The Electrical/Electronic (E/E) system is a major constituent of mechatronic, which regroups the electronics material and software in order to drive intelligently the mechanical components as well as hydraulic components such that their functionalities could be accomplished. In the automotive domain, the E/E architectures usually implement a number of composite functions, which consist in all the vehicle's components such as sensors, input devices, ECUs with embedded control software, actuators, displays and speakers, harnesses for data and power, battery and generator/alternator (Weber, 2009). These individual components communicate with each other via the signals. Figure 1 shows a typical example of E/E architecture, which is composed of sensors, processing components and actuators. The actuators reacts according to the input generated by a sensor (or given by other input devices) and controlled by the processing component. The E/E hardware architecture provides the infrastructures where the applications accomplish their functions. The typical criteria when studying the hardware include the computation power, the number of cores, the number of I/O, etc. In automotive, the E/E hardware architecture is composed of ECUs that are distributed in the car and are assigned with additional hardware. The communications between ECUs are achieved via buses with a specific protocol such as LIN (Local Interconnect Network), CAN (Control Area Network), FlexRay, etc. The components are assigned to the ECUs and the signals are assigned to the corresponding ECU if the relative components are allocated to the same ECU or assigned to buses if they are allocated to different ECUs.

The E/E architecture is also evaluated according to the cost/bill of material that is governed by the ECU cost and the cable between them. Another criterion to evaluate the E/E architecture is ECU complexity that defines the average number of components allocated to one ECU.

The traditional E/E architecture in automotive industry is designed in a federal way. It limits the complexity of implantation as each major function is deployed to one dedicated ECU and is provided as a black box by first-tier supplier. In addition to the clear responsibility of allocation, the federal architecture has remarkable advantage as it facilitates fault containment. The error propagation is limited thanks to its physical separation. Unless there is direct functionality dependency between two functions, a faulty task in one ECU will not affect other distant ECUs. The federal approach allows the system to integrate the different distributed application subsystems (DAS) from different suppliers. However, the addition

of new DASs in the system imposes the addition of new ECUs and cables, which in turn increases the number of ECUs in the cars and the complexity of communication in terms of the physical infrastructure.

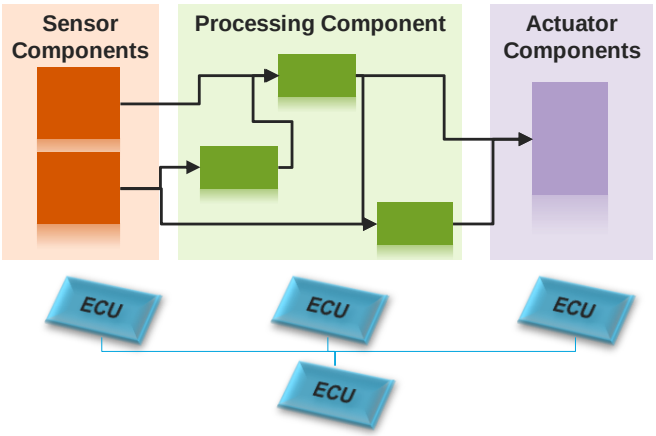


Figure 1-An example of E/E Architecture

Interestingly, the rapid increase of functional complexity as well as the potential cost saving from system integration drives a fundamental shift in automotive architecture system from federal architecture to the integrated architecture (Natale & Sangiovanni-Vincentelli, 2010). This revolution allows the implementation of several DASs that are developed by different suppliers into a single ECU in order to reduce the number of ECUs and connection points. The integrated approach results in a decreasing cost both in terms of infrastructure and maintenance. However, the integration of different DASs into single unit removes the physical barriers that contribute to isolate the fault propagation. By consequence, the decrease of the hardware complexity necessitates compensation from software effort to deal with the safety issue, which in turn increases the software complexity. Coincidently, in avionics domain, as said in (Hammett, 2002), the ideal future avionics systems would combine both the complexity advantage from federal approach and the hardware efficiency benefits from integrated architecture. Since more than one decade, a lot of works have been done and the domain-dedicated standards have been proposed to design the integrated architecture that remains the same composability, fault containment properties as federal approach but still support the integration of multi-functions in a single entity (ex: ECU). The main example is the combination of Integrated Modular Avionics (IMA) with safety DO-178B/C in the aerospace domain and AUTOSAR (AUTomotive Open System ARchitecture) elaboration with safety standard ISO26262 in the automotive domain (this will be introduced later).

1.2 AUTOSAR Standard

Without defining a common standard, the integration of different DASs provided by different first-tier suppliers meet the constraints of interaction as each supplier applies their own standard of development and implementation. In automotive domain, the conventional development shows a vague frontier between applications and infrastructure (see the left

side of the Figure 2), i.e. the applications are dependent of the hardware, which increase the complexity of development process.

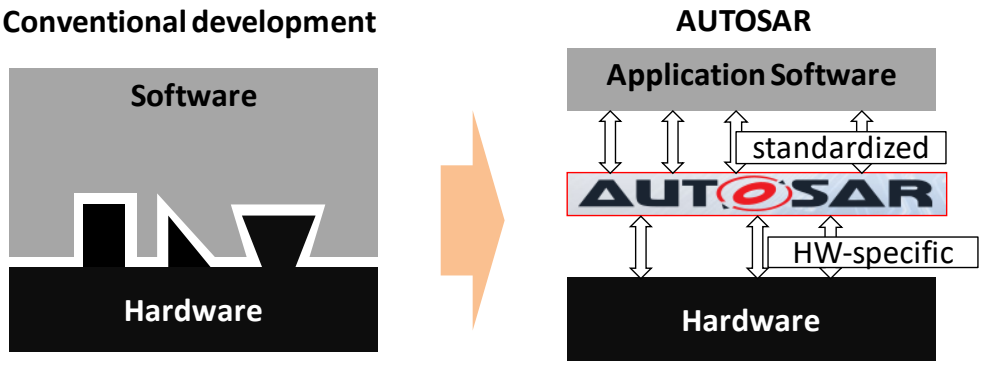


Figure 2-Development revolution driven by AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR), developed by leading automobile companies and first-tier suppliers, contributes to meet the increasing complexity in nowadays' automotive electrical and electronic systems. To achieve the technical goals of modularity, scalability, transferability, and function reusability, AUTOSAR standardizes the software development in automotive domain by separating the application and infrastructure which allows for a model-driven architecture like methodology. That is, applications can exist and communicate independently of a particular infrastructure as shown in the right side of Figure 2. AUTOSAR standard is maintained by a consortium that regroups the general OEM (Original equipment manufacturer), generic Tier 1-supplier, software and service vendors, which contains core partners, premium members and development members. An overview organization of AUTOSAR consortium is shown in Figure 3, where Valeo is belonging to the premium members.

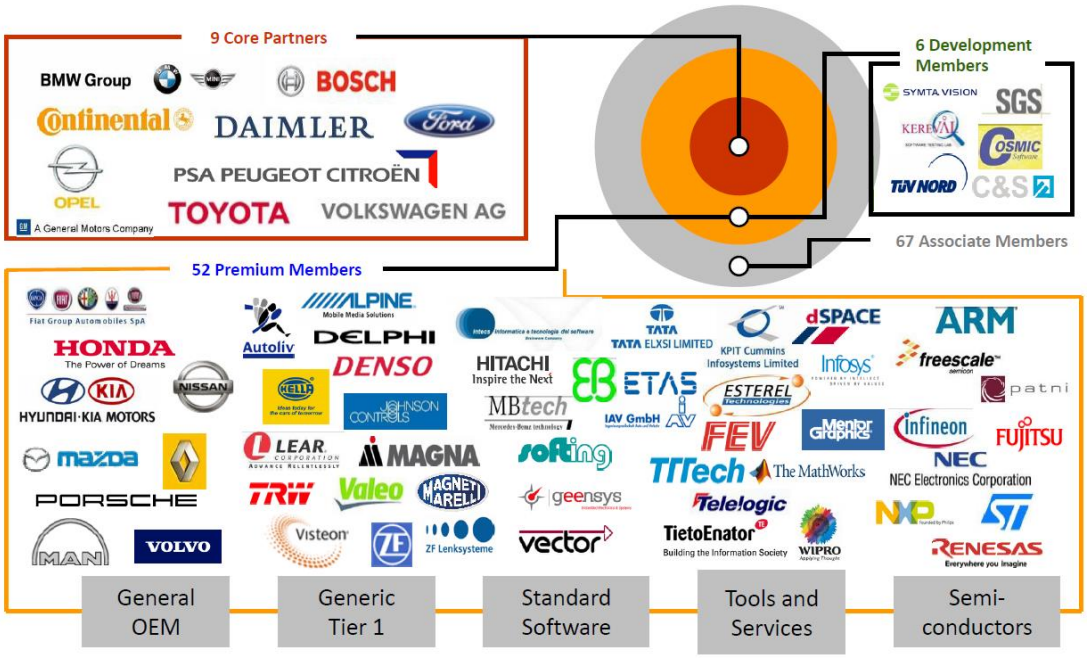


Figure 3-AUTOSAR organization

1.2.1 AUTOSAR architecture overview

AUTOSAR mitigates the problems existing in the system design process thanks to its standardized three-layer architecture: the Application layer, the Basic SoftWare layer and the RunTime Environment layer (RTE). The AUTOSAR layered architecture ensures the decoupling of functionality from the supporting hardware and software service, as shown in Figure 4 in which the purpose of each layer is given as follows:

- **Application** layer: this layer provides a standard description format for application which consists in the SoftWare Components. Application layer is totally independent of the hardware.
- **Basic SoftWare** layer: This layer contains two sub-layers. The first is the MicroController Abstraction Layer (MCAL), which is hardware dependent. The second layer provides services to the AUTOSAR SoftWare Components and is necessary to run the functional part of the software, which include the AUTOSAR OS (Operating System) and service stacks such as communication stack, memory stack and so on.
- **RunTime Environment (RTE)**: this intermediate layer acts as a communication center for inter- and intra-ECU information exchange.

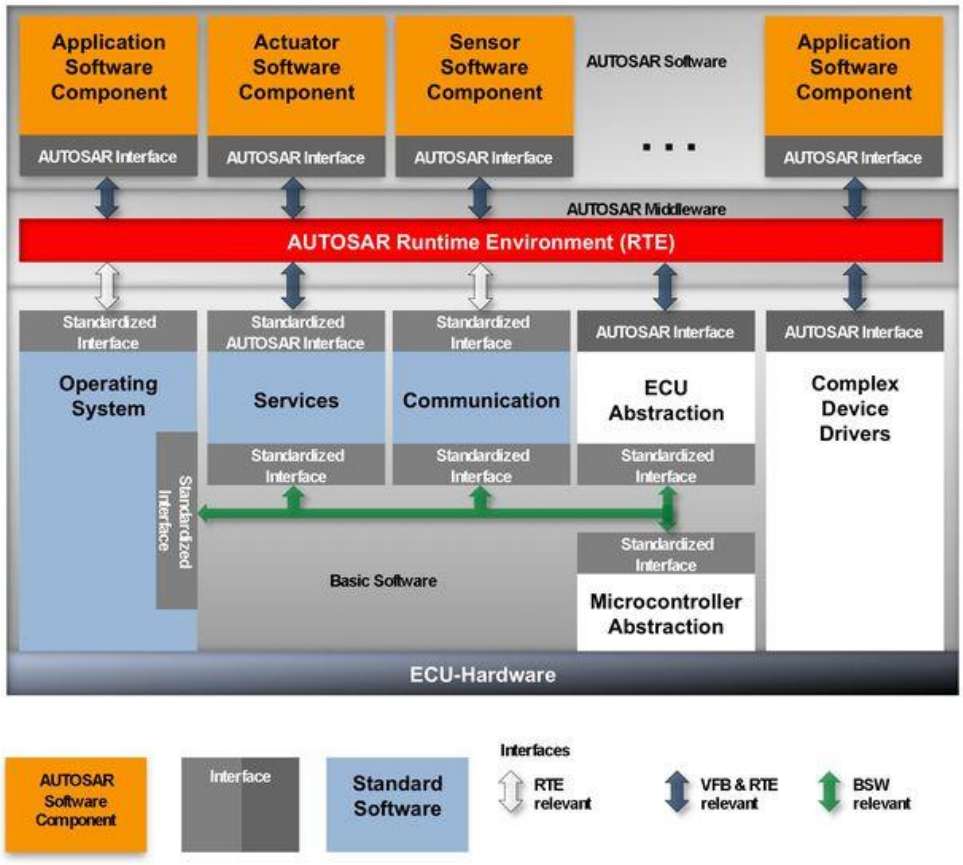


Figure 4-Layered Software architecture of AUTOSAR (AUTOSAR, 2017)

1.2.2 AUTOSAR approach overview

AUTOSAR proposes the approach for the development that is based on the concept of AUTOSAR Software Component (SWC). The approach (shown in Figure 5) contains:

- **Software Component Description**: In AUTOSAR the applications are encapsulated into SWCs that runs on the AUTOSAR infrastructure. For example, Software Component Description can describe a SWC, the data/service, its data sent and received, its internal behavior in AUTOSAR XML (ARXML) format. Almost everything a software developer needs to understand to integrate his component into the system can be provided by Software Component Description.
- **Virtual Functional Bus (VFB)**: The SWC are integrated and interconnected thanks to the Virtual Functional Bus, which allows the abstract description of communications and the deployment phase independently. The virtual function bus provides a virtual infrastructure that is independent from any actual underlying infrastructure, which facilitates the concept of relocatability in AUTOSAR. The services required for a virtual interaction between AUTOSAR components provided by VFB will be latterly implemented by the underlying hardware infrastructure.
- **ECU description & System constraints**: The ECU is described and is independent from the SWC descriptions. All the constraints existing in the system are also described by AUTOSAR.
- **ECU mapping**: The SWCs have to be mapped to ECU network, which includes the configuration and generation of RTE and BSW modules on the concrete ECU.

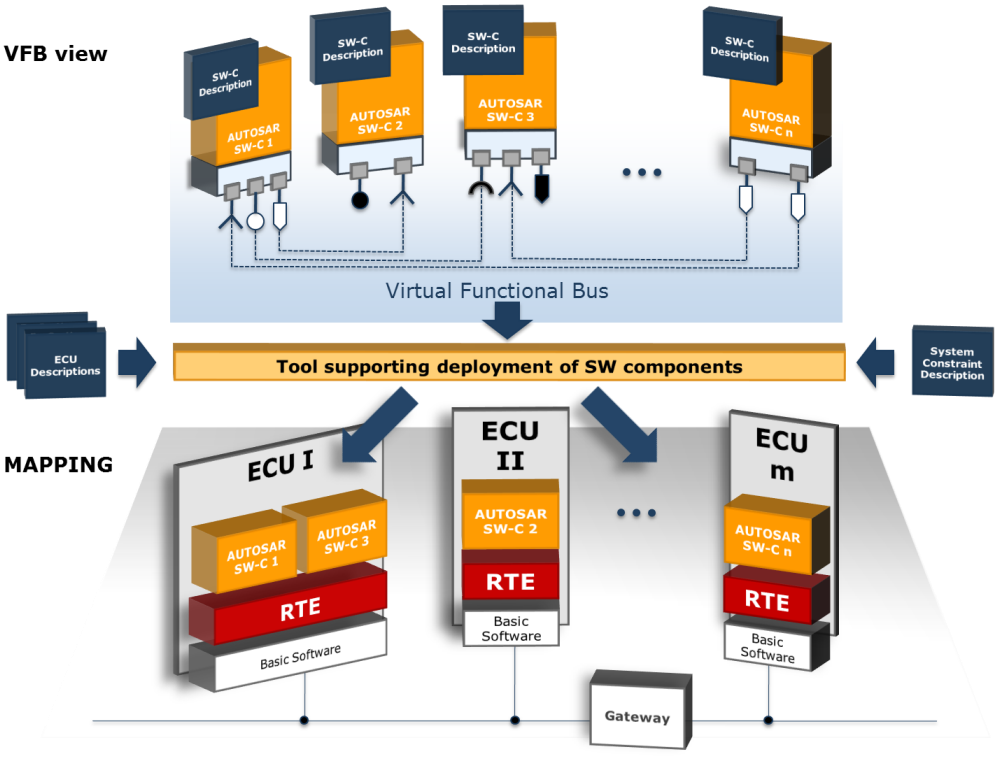


Figure 5-AUTOSAR methodology (AUTOSAR, 2017)

1.2.3 AUTOSAR Toolchain

Here we present our Toolchain and working process for the implementation of the AUTOSAR approach as shown in Figure 6.

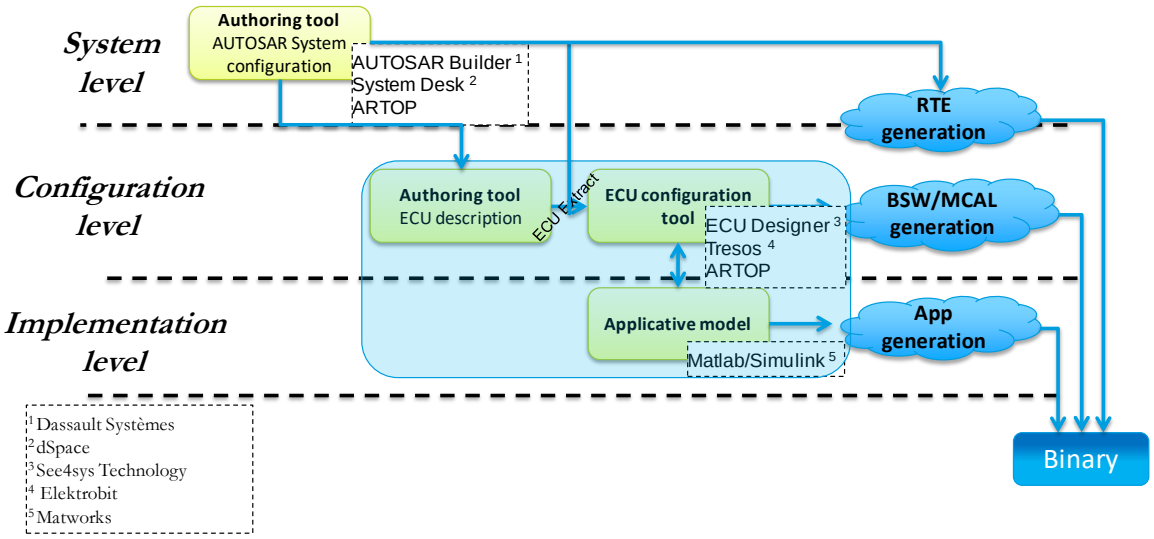


Figure 6-AUTOSAR Toolchain

At **System level**, the information of application such as SoftWare Component description and System Constraint Description are described using the Authoring tool (ex: AUTOSAR Builder), which is the Eclipse-based tool suite for the design and development of AUTOSAR-compliant systems and software. Two aspects of information can be provided at this level: one is SoftWare Component APIs, i.e., header file of application generated during the RTE contract phase. Another aspect of information provided by system level and required for the ECU description step is the mapping of the SWCs onto network of ECUs.

At **Configuration level**, we perform two steps: ECU description and ECU configuration. ECU description involves the mapping of SWCs onto network of ECUs. ECU configuration step contains OS configuration, communication configuration and memory configuration. The configuration of OS includes the terms of priority definition, task content, partition, allocation of resource and communication. This step is implemented by the ECU configuration tool (ex: Tresos) that allows complete ECU basic software configuration. The BSW configuration files are prepared for BSW generators and MCAL generators, they are also used to prepare the RTE generator by combining the component API provided by the system level.

At **Implementation level**, the application description is involved, which can be done by tool like MATLAB Simulink. The runnables' codes can be generated directly from this description. Finally based on the runnables functional codes generated by APP generator, RTE codes by RTE generator, low layer codes by BSW generator and MCAL generator, the binary code for the whole application can be generated.

1.3 Tendency in automotive industry and Multi-core systems

With the emergence of the mechatronic, almost all the functions in automotive are electronically controlled and also interlinked. Nowadays, the automotive industry integrates more and more innovative functions to make the vehicles intelligent, comfortable and safe. Figure 7 shows an example of the typical functions in the vehicles. The following facts as described in (Weber, 2009) give the quantified information.

- **2500** functions are controlled by software representing **10 million** lines of codes are integrated in the cars.
- These functions are realized by up to **80** ECUs that communicate via up to **5** different types of systems busses.
- **90%** of all innovations are enabled by electronics and software.
- Up to **40%** of a vehicle's costs are determined by electronics and software.
- **50-70%** of the development costs for ECU are related to software

By consequence, the number of ECUs embedded in each cars increases dramatically in order to meet the requirements for the functionalities such as the engine control, body, chassis control. Besides, the new functionalities that involve data processing and ADAS (Advanced Driver Assistance Systems) make it even more complex for tomorrow's vehicles: that the cars are becoming autonomous and connected. The increase of software complexity makes it unavoidable for automotive industry to require more and more computing power.

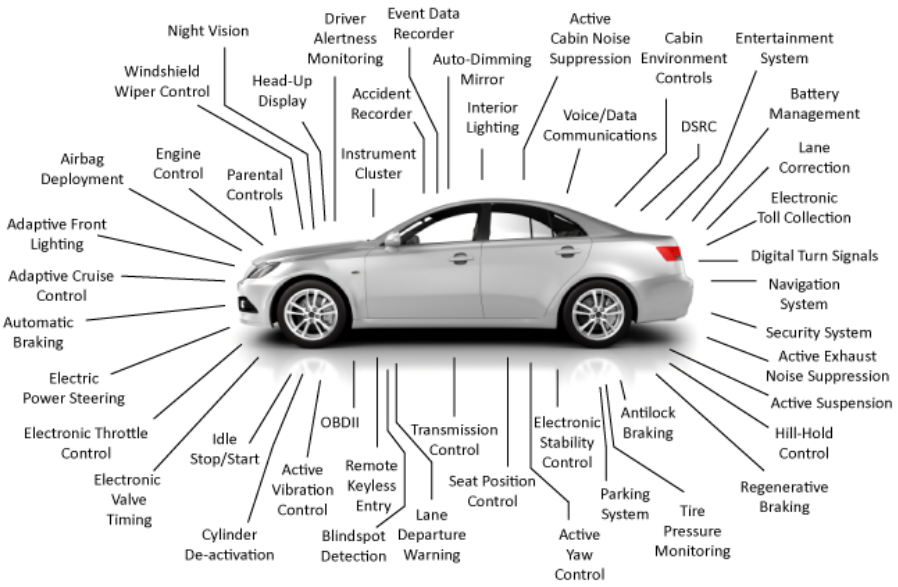


Figure 7-Functionalities in vehicles

The traditional way to satisfy these requirements is mainly on two axes: the miniaturization of transistors and the frequency scaling in single-core systems. However, both of the

solutions mentioned above have reached their bottleneck. Though the miniature of the transistors allows integrating more transistors on one single chip according to Moore's law, it might case the hit dissipation issue. The frequency scaling benefits the system by executing more instructions in a given time, but it could invoke power consumption problems as the power of the core is proportionate to the frequency. A feasible way to avoid the issues above is the increasingly widespread use of ECU with multi-core, where each core is not obligated to have a high performance. But an efficient cooperation between these cores will make the system reach a high performance. The multicore systems can avoid the issues arising in the usage of tradition approaches based on single-core. However, it still remains a great challenge for the developers to solve the interaction issues existing in the mechanism of a highly efficient cooperation between cores.

Another reason for multi-core is that according to the supplier's roadmap, the high power single core controllers have been replaced by multicore, where the efficient cooperation between low power cores can provide a high performance. The semiconductor manufacturers that mainly used in Valeo like Freescale and Infineon propose the multi-core architecture platform, where the cores permit execution of the application codes in two modes: the lockstep mode and decoupled mode.

The lockstep mode involves executing the same instructions on all the cores. The generated results will then be compared to detect errors. If a difference between the cores is detected, the system enters a fail safe mode. The lockstep targets the safety-critical applications, which provides tolerance against the transitory fault on hardware. However it is not tolerant for the hardware faults that are permanent.

Unlike the lockstep mode that is still logically single-core execution, in the decouple mode, each core is independent and can be used to execute their own programs. However the safety advantages in this case is lost.

1.3.1 Multi-core architecture categories

Generally the multi-core architecture can be classified into three categories:

- Heterogeneous architecture: The processors are different. These processors use different set of instruction and the computing power of each core is different. For example, the microcontroller ***Ti Vision Mid*** is a heterogeneous microcontroller, its architecture contain a core ***A8*** and ***4 DSP***.
- Homogeneous architecture: The processors are identical. They use the same set of instruction and the computing power of each core is identical. Microcontroller ***Leopard MPC5643L*** with 2 identical cores is an example.
- Uniform architecture: The instruction set in all processors is the same, but the computing power for each core may be different. One example for uniform architecture is ***Freescale Bolero MPC5644C*** with two different cores (***z0*** and ***z4***), while these two cores follow the same set of instructions.

1.3.2 AUTOSAR in multi-core

Since AUTOSAR release 4.0, multicore specifications are available in AUTOSAR specifications. On multicore, the cores are prioritized in one master core and several slave cores. The master core is started first by hardware. The master core then triggers the start of the other (slave) cores by Software calls. The AUTOSAR Multi-core OS specification requires a system with master-slave start-up behavior, either supported directly by the hardware or emulated in software. The master core is defined as the core that requires no software activation, whereas a slave core requires activation by software (AUTOSAR, 2014a). The multi-core architecture explores the notion of OS-Application. Since AUTOSAR 3.x, memory protection has been available in AUTOSAR. This capability requires to introduce a new OS object called OS-Applications, which clusters a set of OS objects (ex. task, alarms, etc). In the Autosar application, no OS objects exist in an isolated way out of the OS-Applications boundary. It is worth noting that OS-Applications are not dedicated to multicore architectures but become mandatory for the design of multicore applications. This new granularity means that the RTE generator must be aware of the allocations into OS-Applications.

Mainly, multicore features impact specifications of:

- The RunTime Environment (RTE): RTE has to be aware of multi-core capabilities. It manages the protection of shared objects by spinlock. A spinlock is a busy waiting mechanism that polls a (lock) variable until it becomes available. Typically, this requires an atomic "test and set" functionality, which is implementation specific.
- The Operating System (OS): The OS provides the new services in order to activate tasks or a set of events across cores, and also to synchronize or protect shared objects. AUTOSAR OS in multi-core is Partitioned OS (P-OS), which means that an instance of the AUTOSAR OS runs on each core. When a core makes a system call, this core switches to kernel mode and executes the OS code.

The multi-core OS configuration is different between the **S**ymmetric **M**ulti-**P**rocessing (SMP) and **A**symmetric **M**ulti-**P**rocessing (AMP):

In **AMP**, each core runs an OS, the OS between the core may be the same or different. The different cores do not share the code and data. Each core has its designated task set. AUTOSAR OS belongs to this type as it is the partitioning OS (P-OS) where each core runs one P-OS.

In **SMP**, the cores share the same task set, which means a task may run on different cores dynamically. The different cores are managed by one same OS. The cores share the same memory space.

1.3.3 Modeling details of AUTOSAR

Model-Based Design (MBD) development process is specifically attractive in embedded domains like automotive thanks to its capabilities to support early design

verification/validation through formal functional modes that are composed of the functional blocks and the capabilities to generate software implementations from those functional blocks. MATLAB/Simulink is a widely used **C**omputer **A**ided **E**ngineering (CAE) tool for model-based design that allows simulating system behavior, tracing and verifying requirements and generating software for prototyping and production. Model based approach provides an automated software synthesis flow that turns functional models to correct, predictable and optimal software tasks implementation on various embedded platforms. Within AUTOSAR standard, this flow includes firstly the encapsulation of the functional blocks into software components (SWCs) composed of a set of runnables, and the mapping of the runnables into real-time tasks as shown in Figure 8. In multi-core context, it introduces the new steps such as the mapping of tasks to cores and synchronization between cores.

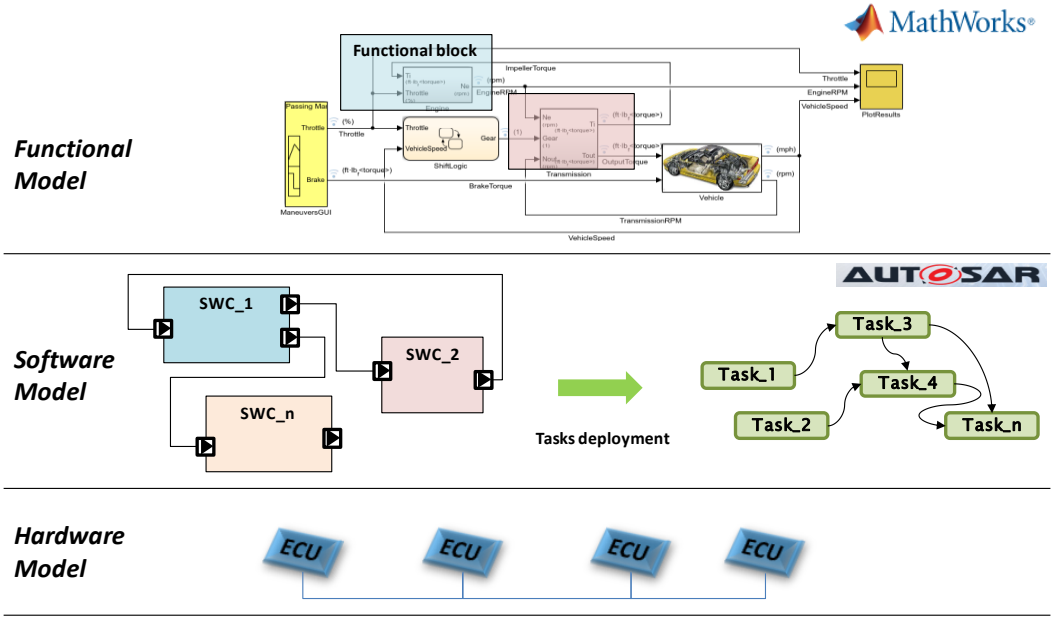


Figure 8-MBD approach with AUTOSAR

In this part, we give the details in each layer of AUTOSAR architecture presented in Figure 4.

1.3.3.1 Application Layer

The application layer is standardized in AUTOSAR. The application is split into SoftWare Components (SWCs) that interact through the Virtual Function Bus (VFB) as shown in top of Figure 5. Software Components are logical groups of functionalities of the application. Each AUTOSAR SoftWare Component (SWC) is a so-called "Atomic SoftWare Component", which implicates that each instance of an AUTOSAR SoftWare Component cannot be distributed over AUTOSAR ECUs. Furthermore, according to the recommandation of AUTOSAR [1], an AUTOSAR SoftWare Component cannot be distributed over cores in multicore systems either.

[1] This restriction might be released in the future version of AUTOSAR.

SWCs are composed of the objects described as follows, which is illustrated in Figure 9:

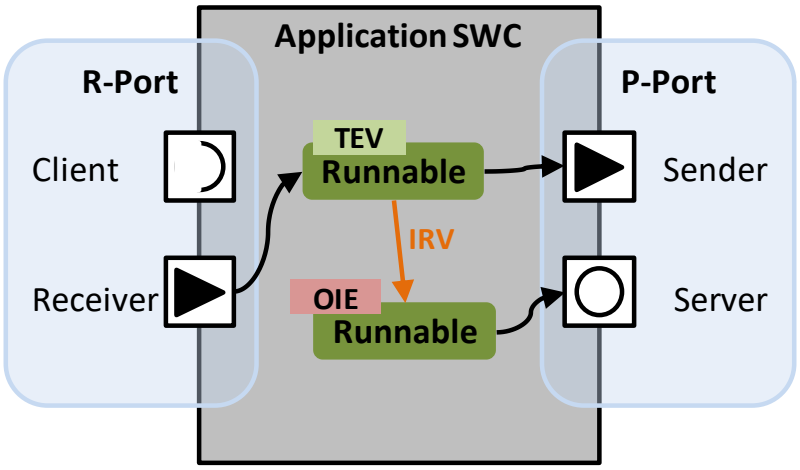


Figure 9-SWC description

- **Runnable**

The Runnables are the atomic functional components in the SWC, which cannot be further divided. The Runnables are composed of the pieces of functional codes and can communicate to other runnables in the same SWC by IRVs (Inter Runnable Variables) or runnables in the other SWCs by interface and Ports.

The main functions of a Runnable are Read (read the external variables by ports or the internal variable by IRV), Execution and Write (write the external variables by ports or the internal variable by IRV). Runnables can execute and be scheduled independently from the other Runnable Entities of the same Atomic Software-Component.

Runnables are executed in the context of an OS task; their execution is triggered by RTE Events. There are several types of Event, such as **T**iming **Ev**ent (TEV) for the periodic runnables; **M**ode **S**witch **E**vent (MSE) for runnables that exchange the execution modes; **D**ata **R**eceived **E**vent (DRE) for the runnables that are active when the relative data are available; **O**peration **I**nvoked **E**vent (OIE) for the server runnables that provide a service, and so on. Every event has its own parameters, for example: TEV with PERIOD, MSE with mode, DRE with Port and the OIE with operations. Every Event relates to a Runnable, whereas each Runnable can relate to several Events.

- **Ports**

The ports can implement all the type of interfaces so as to access the information like data, operation or modes in the different type of interfaces described before. The components have two types of ports: Provided ports (P-Port) and Requited port (R-Port).

- Interfaces

The Interface defines the information exchanged between SWCs and/or BSW modules. It makes it possible to implement the communication between SWCs by Ports. The main

types of interfaces are SenderReceiver Interface, ClientServer Interface and ModeSwitch Interface.

1.3.3.1.1 Communication between AUTOSAR SWC

Autosar SWCs communicate through well-defined ports and the behavior is statically defined by attributes. The ports are described by the Port Interface. There are two important communication types:

- Sender-Receiver communication is realized via SenderReceiver Interface like P-Port from provider SWC to R-Port from consumer SWC. The object transferred is the data of different types (see section 2.3). This type of connections supports both 1: N and N: 1 communications.
- Client-Server communication is built from a server P-Port to a client R-Port via ClientServer Interface, where client runnable requires the service provided by the server runnable. Unlike Sender-Receiver communication, Client-Server connection support only N: 1 communication, it is impossible for a client to invoke multiple servers with a single request.

1.3.3.1.2 SWC internal communication

Communication between runnables insides the same SWC, also known as the communication intra SWC, is done by using Inter Runnable Variables (IRV). IRV are the variables that can be written and read by the Runnables in the same SWC, which means that the IRVs exist only in the SWC they belong to.

1.3.3.2 Basic Software Layer

The Basic Software is standardized software that does not have any functionality from the application view but offers hardware-dependent and hardware-independent services to applications. This is realized through the use of Application Programming Interfaces (API, see Figure 4). This layer itself is not entirely hardware independent but makes the upper software layers independent of the hardware. In basic software layer there are several items:

1) **M**icrocontroller **A**bstraction **L**ayer (MCAL): this layer provided by smelters is a hardware specific component that provides access to the actual physical signals of the microcontroller.
2) BSW Stacks: The BSW Stacks includes communication stack, the memory stack and I/O stack.
3) Operation System (OS): OS in Autosar is based on the OSEK/VDX (OSEK/VDX, 2005). It supports multicore architecture since the version 4.x of AUTOSAR is released. The OS is responsible for the execution of real-time tasks containing runnable entities. AUTOSAR adopts static priority for the tasks in the system and static scheduling.

Basic Software consists in main functions and services that can be called by tasks. In the single core system, the main function is executed in an OS task and BSW calls are done in

the context of the calling task. When coming into multi-core systems, the main functions and services could be allocated separately in different cores. BSW allocation is done during the Operating System configuration (see Figure 6).

- Most BSW activations, i.e. BSW APIs calls, are done in the context of the calling task. It might have several calls from different cores.
- The main functions can be allocated to different cores (as shown in Figure 10). The coupling between the main functions and services might cause delays or bottlenecks.

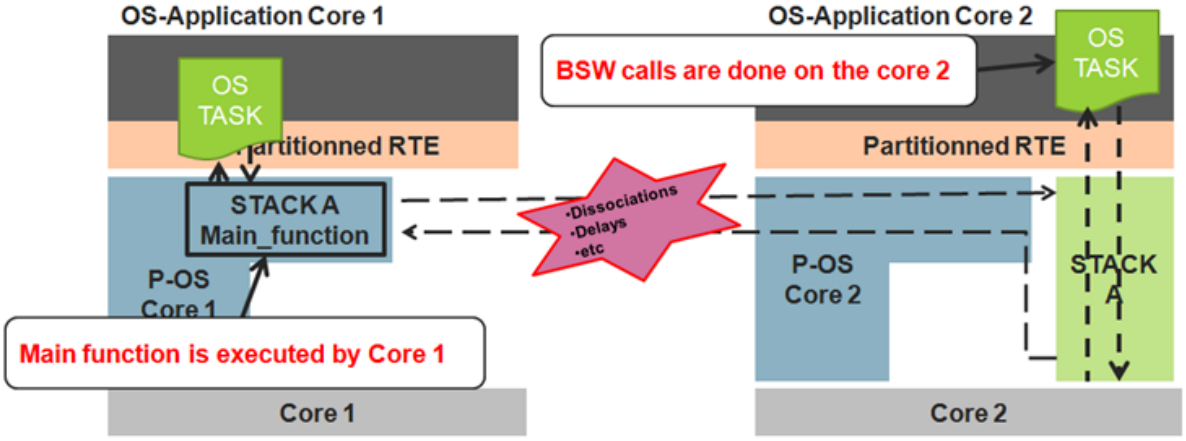


Figure 10-BSW allocation example

1.3.3.3 Runtime environment (RTE)

Runtime environment (RTE) handles the information exchange between the application software components and connects the application software components to the right BSW services. This layer decouples the application software components from the hardware as well as the application software components from themselves. RTE provides an actual representation of the virtual concepts of the VFB for one specific ECU, which means that there exists one implementation (ex, in C-code) of the VFB per ECU. In order to do that, it requires knowing where runnables (from SWCs) are allocated. For example, runnables allocated to the same ECU communicate using the RTE while runnables allocated to different ECUs use the AUTOSAR communication stack. Besides, RTE also involves the generation and realization of all the RTE events that activate the behavior of runnables.

It is worth noting that all configurations are static, as a result, the components have been located statically at the phase of implementation. Components that are mapped onto one ECU will communicate through shared memories and the components mapped onto different ECU will communicate by the communication stack (i.e., bus CAN, LIN, FlexRay). The RTE can be seen as a static implementation of specialized communication topologies.

1.4 Safety

Safety is a property of a system that will not endanger human life or the environment. Many safety-critical systems are also real-time systems, where "the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced" (Kopetz, 2011). The safety-critical systems have to be certified. Certification phase is standard-based, which depends on the application area. For example, DO-178B/C is for the airborne systems. In today's cars several ECUs may control safety relevant actuators depending on the functionality in the vehicle. The ISO26262 is the norm which describes how the development of such ECUs shall be performed to realize a safe system. This norm defines four "Automotive Integrity Safety Levels" (ASIL) which classify levels of safety required for these systems. Based on the identified risks, specific (Safety) requirements of the system are derived. These requirements may be related to hardware or software or both. We mainly focus on software, so the hardware part will be considered as platform based design. Be aware that an ASIL is always defined for a system, which means hardware and software, and with respect to software application software and basic software.

AUTOSAR, up to Release 4.1, supports safety systems (ISO 26262) by offering different base mechanisms which are typically required in such ECUs. The following list contains the main safety mechanisms:

- Partitioning of SWCs to support the isolation in space. This means that it is possible to separate SWCs of different ASIL from each other and to make sure that the SWCs are not able to write to other SWCs data. The realization requires hardware support (a memory protection unit (MPU) or memory management unit (MMU)) and is realized in the OS module and used by the RTE.
- Timing and control flow supervision to monitor executing entities and to detect faults caused by blocking or wrong execution. In AUTOSAR the OS and the relative modules (ex: watchdog module) take care of this issue.
- A safe communication via end-to-end protection is possible between ECUs (and even inside an ECU). This guarantees e.g. that the data which is send is not modified between the sender and the receiver(s). The responsible module is the E2E library.

Some other modules support additional mechanisms which are also useful in safety systems (e.g. ***CoreTest*** or ***RamTest***) (AUTOSAR, 2014b).

1.5 Contribution and Thesis overview

The shift towards multi-core systems in the automotive industry has revived the challenge of application partitioning to enhance productivity, reusability and predictability. The introduction of multi-core in AUTOSAR leads to additional works in the process of automotive development:

- **Software component (SWC) to cores**: the SWC/runnables have to be distributed into different cores
- **Tasks definition and configuration**: in order to be executed by OS, the runnables have to be mapped into different real-time tasks. Therefore a set of tasks has to be defined and configured properly. This step exists already in single-core system. However in the multi-core, the runnables have to be remapped according to the new position, thus the execution order in each task changes as well.
- **Data allocation**: the data exchange between the components has to be distributed in different type of memories.
- **Synchronization**: the execution flows in different cores have to be synchronized such that the entire system behaves correctly.

The traditional way to migrate to multi-core platform in Valeo software team was accomplished manually, which necessitates a high level knowledge of application especially in the aspect of functionality. Each time a new application is targeted, a lot of repetitive work is unavoidable, which introduce a significant workload and time consuming process. Moreover, the manual solution prevents the optimization of criteria such as CPU loads, communication overheads, jitter and so on.

Confront to these issues, this thesis proposes a method and tools dedicated to the migration of the automotive applications into multi-core architectures. The method acts as a decision guide environment for the partitioning of embedded software modeled with the AUTOSAR specificities onto multi-core systems. The proposed method automates the migration process and was fully thought into an industrial V-cycle development process as shown in Figure 11.

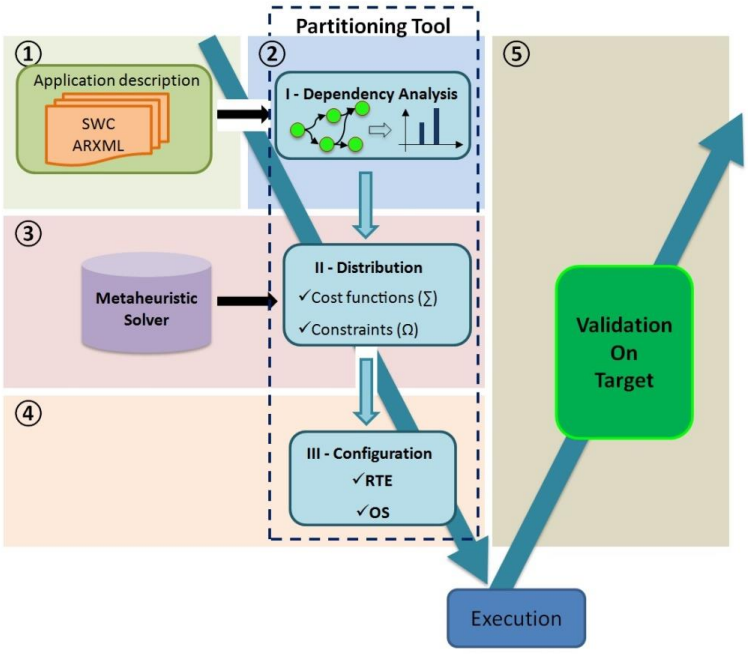


Figure 11-Working process

The working process shows the contributions of this thesis that contains:

1) Automate the allocation of automotive applications: the automotive applications are compliant with AUTOSAR standard. We model the applications and analyze the dependencies level between the execution entities (runnables) in order to provide the necessary information for the distribution of the applications into multi-cores. We adopt the meta-heuristic algorithms as the solver to search the design space efficiently and effectively. The distribution solutions are optimized and evaluated by the proposed cost function that takes a set of criteria into account. The solutions respect the pre-defined constraints that could be, from the hardware aspect, real-time, functional and is extendable to the new features (ex, the safety aspects). More details are presented in Chapter 2.

2) Propose the scheduling for the dependent real-time task sets. Most of the work for real time scheduling in the state of the art is targeted to tasksets that are independent. However, this ideal theoretical model shows its limit in the real-life industrial use case, where the applications are strongly inter-connected and the tasks are dependent. In this thesis, we propose the scheduling approach that considers the dependent tasks. Based on the defined dependent model, the method generates the schedule table that contains the execution order of the instance for the task/runnables and the start date for them. The scheduling is static and respects the real-time constraints. The evaluation of the scheduling considers the ability of schedulability and other criteria such as jitters and makespan. Chapter 3 presents this part of work.

3) Integrate in the process of development. In Chapter 4 we show another contribution of this thesis: the SoftWare Allocation Tools (SWAT) is our developed Toolchain that integrate our methods into a real-life development process in the automotive industrial. The process is compliant with AUTOSAR Toolchain.

4) The conclusion and the prospective are presented in Chapter 5.

Chapter 2 Relative works & problem formalization

2.1 Combinatorial Optimization

Many optimization problems of theoretical as well as practical importance consist in the search for the "best" configuration of a set of variables to achieve some objectives. Among those where solutions are encoded with *discrete* variables we find a class of problems called Combinatorial Optimization (CO) problems as introduced in (Papadimitriou & Steiglitz, 1982). According to the introduction in (Papadimitriou & Steiglitz, 1982), in CO problems, we are looking for an optimal object from a finite (or possibly countable infinite) set of objects. These objects are encoded with discrete variables such as integer numbers, subsets, permutations, graph structures etc. A model $P = (S, \Omega, f)$ of a CO problem consists in:

- S: a search space where a finite set of discrete variables $X_i, i = 1, \ldots, n.$ are defined;
- Ω: a feasible domain defined by a set of constraints;
- f: an objective function to be minimized.

A feasible solution $s \in S$ is a complete assignment of values to variables that satisfies all constraint in Ω. A solution $s^* \epsilon S$ is called a global optimum if and if: $f(s^*) \leq f(s)\ \forall\ s \in S$.

Examples for CO problems are the Traveling Salesman problem (TSP), the Quadratic Assignment problem (QAP), Timetabling and Scheduling problems. Owing to the practical importance of CO problems, many algorithms to tackle them have been developed. These algorithms can be classified as either *exact* or *approximate* algorithms.

Exact algorithms (or complete algorithms) are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time. Despite the progress achieved by the exact algorithms, the problems likely to be resolved by the algorithms are quite restricted however (Woeginger, 2003). In fact, the exact algorithms are often impractical for large problems due to prohibitive search times. Effectively, for CO problems that are $\mathcal{NP} - hard$ (Garey & Johnson, 1990), if $\mathcal{P} \neq \mathcal{NP}$, there is no polynominal-time constant-factor algorithm exists. Therefore, exact algorithms might need exponential computation time in the worst-case. Thus, the use of approximate algorithms to solve CO problem has received more and more attentions in the last 30 years. In approximate algorithms we sacrifice the guarantee of finding optimal solutions for the purpose of obtaining good solutions (not optimal) in a significantly reduced amount of time.

Among the basic approximate methods we usually distinguish between *constructive* methods and *local search* methods. Constructive methods involve in building a solution to the problem literally step by step from scratch. Usually constructive methods are deterministic and they are typically the fastest approximate methods, yet they often return solutions of inferior quality when compared to local search algorithms. In contrast to the constructive method that begin with an empty solution, local search algorithms start from some feasible solutions called initial solutions of the problem and tries to progressively improve it by replacing iteratively the current solution with a better solution from an appropriately defined neighborhood of the current solution.

Definition 1 A neighborhood structure is a function $\mathcal{N}: S \rightarrow 2^S$ that assigns to every solution $s \in S$ a set of neighborhood $\mathcal{N}(s) \subseteq S$. $\mathcal{N}(s)$ is called the neighborhood of s.

The choice of an appropriate neighborhood structure is crucial for the performance of a local search algorithm and is problem-specific. The neighborhood of a solution s describes the subset of solutions which can be reached from s in the next step. The solution found by a local search algorithm may only be guaranteed to be optimal with respect to local changes and will generally not be a globally optimal solution.

Definition 2 A local optimum for a minimization problem, a locally minima solution (or local minimum) with respect to a neighborhood structure $\mathcal{N}$ is a solution such that a solution $\hat{s}$ such that $\forall\, s \in \mathcal{N}(\hat{s})\colon\ f(\hat{s}) < f(s)$. Similarly, a local optimum for a maximization problem, a locally maxima solution (or local maximum) is a solution such that a solution $\hat{s}$ such that $\forall\, s \in \mathcal{N}(\hat{s})\colon\ f(\hat{s}) > f(s)$.

A disadvantage of single-run algorithms like constructive methods or local search is that they either generate only a very limited number of different solutions, which is the case of constructive methods or they stop at local optima, which is the case of local search. Several general approaches, which are nowadays often called meta-heuristics, have been proposed to bypass these problems. This class of algorithms includes – but is not restricted to – Simulated Annealing (SA), Tabu Search (TS), Evolutionary Computation (EC) including Genetic Algorithms (GA), and Ant Colony Optimization (ACO). Up to now there is no commonly accepted definition for the term metaheuristic, while the fundamental properties which characterize metaheuristics can be outlined (Blum & Roli, 2003):

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

The following methods use generally a large amount of parameters with the value configured based on a lot of experiment results.

2.1.1 Simulated Annealing

Simulated Annealing (SA) is deduced from the physical annealing process of solids, which is commonly said to be the oldest among the metaheuristics and surely one of the first algorithms that had an explicit strategy to escape from local minima (Kirkpatrick, Gelatt Jr, & Vecchi, 1983). The fundamental idea of escaping from local minima is to allow moves to solutions of worse quality than the current solution (also called as uphill moves) in order to escape from local minima. The acceptance probability which is the probability of doing such a move is decreased during the search. As descript in the Algo. 1.

```
s ← GenerateInitialSolution ()
T ← T_0
while termination conditions not met do
        s' ← PickAtRandom(N(s))
        if (f(s') < f(s)) then
          s ← s'                % s' replaces s
        else
          Accept s' as new solution with probability p(T, s', s)
        endif
        Update (T)
endwhile
```

Algo. 1-Algorithm: Simulated Annealing (SA)

The algorithm starts by generating an initial solution (either randomly or heuristically constructed) and by initializing the parameter $\boldsymbol{T}$ that signifies temperature. Beginning at the initial solution, the algorithm performs the searching process iteration by iteration until the terminated condition is met. At each iteration, a solution $\boldsymbol{s' \in \mathcal{N}(s)}$ is randomly sampled in the defined neighborhood structure and it will be accepted as a new current solution depending on the conditions:

- If the objective function is improved, i.e. $\boldsymbol{f(s') < f(s)}$, s' is accepted as a new accurent solution.
- If the objective function is degraded, i.e. $\boldsymbol{f(s') > f(s)}$, s′ is accepted as a new solution with an acceptance probability that is related to $\boldsymbol{f(s)}$, $\boldsymbol{f(s')}$ and $\boldsymbol{T}$.

At the end of each iteration, the temperature parameter $\boldsymbol{T}$ is updated with a tendency of decreasing principally, each step of the updating is not necessarily decreasing however. So the progress of algorithm depends on three parts:

2.1.1.1 Probability of accepting uphill

The probability of accepting a degraded solution is a function of $\boldsymbol{f(s') - f(s)}$ and $\boldsymbol{T}$. This function is typically computed following the Boltzmann distribution $\boldsymbol{exp\left(-\frac{f(s')-f(s)}{T}\right)}$. The probability of accepting a new solution is then determined by two factors: the difference between the costs of the two solutions $\boldsymbol{f(s') - f(s)}$ and the temperature $\boldsymbol{T}$. On the one hand, for a fixed temperature, the worse the new solution performs, the smaller the possibility of

acceptance of this solution is. On the other hand, it is more possible to accept a worse solution at a high temperature, which is quite common at the beginning as the temperature is relative high. With the decreasing of the temperature, the algorithm converges gradually to an improvement research. The entire search process contains two phase of strategies: the random walk and iterative improvement. At the beginning of the search, the algorithm encourages an erratic move which permits an exploration of the searching space. Then in the second phase of search, the random is decreasing gradually and the search process concentrates to the improvement that leads to exploitation for the minimum.

2.1.1.2 Cooling rule

The choice of an appropriate cooling rate is essential part of SA as it determinate the performance of the algorithm. A high cooling rate leads to degraded results because of the lack of representative states, while a low cooling rate results in the increasing of computation time to get the convergent state. Two choices have to be made when implementing the SA: the initial value of temperature T_0 and the cooling schedule.

A quite high value T_0 permits capture the entire solution space, while it may increase the number of iteration, which might not necessarily give the better solutions. Generally, the initial value of temperature is chosen by experimentation that depends on the nature of problem.

The cooling schedule characterizes the change of temperature in functional form so that the value of T at each iteration k can be determined. The cooling schedule is presented as $T_{k+1} = Q(T_k, k)$, where $Q(T_k, k)$ is a function of the temperature in the last state and the iteration number. Three important cooling scheduling are logarithmic, Cauchy and exponential. SA converges to the global minimum of the cost function if the change of temperature follows a logarithmic law (Geman & Geman, 1984): $T_k = {T_0}/{\log k}$. This schedule requires the move to be drawn from a Gaussian distribution. For the practical purposes, this cooling scheduling are too slow unfortunately. Cauchy schedule performs a faster convergence, where $T_k = {T_0}/{k}$ with the moves are drawn from a Cauchy distribution (Szu, 1987). The fastest schedule among the three is exponential or geometric schedule in which $T_k = T_0 exp(-C_i)$ where C_i is a constant (Azencott, 1992). In practical case, one of the most used schedules follows the geometric law: $T_{k+1} = \alpha T_k$, where α is cooling factor with constant value varies between 0.80 and 0.99, which performs an exponential decay of the temperature. There are also non-monotonic cooling schedules, which are characterized by alternating phases of cooling and reheating, thus providing an oscillating balance between diversification and intensification. The initial value of temperature and cooling schedule should be adapted to the concrete problem instance appropriately, as the balance of the diversification and intensification influents the capability of escape from local minimum that depends on the structure of the search landscape.

2.1.1.3 Terminated condition

The termination of the algorithm depends on the number of iterations, which contains the total number of iterations and number of iteration for each temperature. The total number of iterations adopted depends on the complexity of problem. The number of iterations at each temperature is chosen so that the system is sufficiently close to the stationary distribution at that temperature.

2.1.2 Tabu Search

Tabu Search (TS) is created by Fred W. Glover (Glover, Future paths for integer programming and links to artificial intelligence, 1986). It is among the most cited and used metaheuristics for combinatorial problems. The strategy of TS is to maintain a tabu list that memorizes the history of the search in order to escape local optimum as well as facilitate the exploration in the searching space. A description of this algorithm can be found in (Glover & Laguna, Tabu Search, 1997). The process of TS is described briefly as follows: starting with an initial solution (generated either randomly or heuristic constructed), the algorithm looks for the best solution $\boldsymbol{s'}$ in the neighborhood structure $\mathcal{N}(s)$. If solution $\boldsymbol{s'}$ is not already existed in the Tabu list or if it satisfies the condition to ignore the tabu rule (noted as Aspiration criteria that will be introduced later), it is accepted as a new solution. Before beginning the next iteration, the tabu list is updated by adding this solution and removing a solution according to different policies (usually in a FIFO order) if the list is already full. The Aspiration criteria shall be updated as well, as described in Algo. 2.

2.1.2.1 Tabu list and Aspiration criteria

The simple TS performs a best improvement local search as basic ingredient and meanwhile maintains a short term memory for the sake of escaping from the local optimum as well as preventing the cycles of search. The short term memory is implemented as a tabu list that keeps track of the most recently visited solutions, so the move towards the solutions existed in this list is forbidden, which help filter the solutions in the neighborhood and generate allow set. However, the implementation of the short term memory as a tabu list that contains a set of complete solutions is not practical, as the management of this list with full information is quite inefficient. Therefore, instead of storing the solutions themselves, the tabu list chooses the representative attribute such as the components of solution, differences between solutions, move or other brief information. For more attributes to be considered, a tabu list is created for each of them. So multiple tabu lists can be used simultaneously and are sometimes advisable.

Although the storing of the attributes instead of the complete solutions is effective, it might lead to the loss of information potentially. As an attribute might present more than one solution, the forbidden of the attribute would filter several solutions that attached to it, which increase the possibility to exclude the unvisited solution with good quality. That is why TS defines ***Aspiration criteria*** to overcome this problem. Aspiration criteria contain the solutions that are allowed to be considered by the algorithm even if they are forbidden by the tabu list. The condition that considers solutions to be included in the aspiration

criteria is called aspiration condition. A typical condition is to choose the solutions that are the best found so far solutions.

```
s ← GenerateInitialSolution ()
Initialize TabuLists (TL1 ... TLr)
k ← 0
while termination conditions not met do
        AllowedSet(s, k) ← { s' ∈ N(s) | s does not violate a tabu condition,
                                         or it satisfies at least one aspiration condition }
        s ← ChooseBestOf (AllowedSet(s, k))
        Update TabuList & AspirationConditions ()
        k ← k + 1
endwhile
```

Algo. 2-Tabu Search (TS)

2.1.2.2 Memory

The memory structures in TS are dimensioned by four principles: recency, frequency, quality and influence.

Recency-based memory of TS constitutes a form of aggressive exploration in the search space that targets at the best moves possible, the most common used short term memory keeps track of attribute of solutions that have been considered recently, just as tabu list does.

Frequency-based memory keeps track of the frequency of each solution (or attribute) has been visited. This information identifies the regions (or the subsets) of the solution space where the search was confined, or where it stayed for a high number of iterations. This kind of information about the past is usually exploited to diversify the search. Recency-based and frequency-based memories complement each other.

Quality-based memory refers to the accumulation and extraction of information from the search history in order to identify good solution components. Quality plays a role to reinforce actions that lead to good solutions and penalizes the actions to poor solutions, which can be usefully integrated in the solution construction. This principle is used explicitly by other metaheuristics to learn about good combinations of solution components.

Influence-based memory considers the impact of the choices made during the search process both on the quality and structure. The information can be used to indicate which choices have shown to be the most critical.

In general, the TS field is a rich source of ideas and strategies, many of which have been and are currently adopted by other metaheuristics.

2.1.3 Evolutionary Algorithm

Evaluation Algorithms are in the category of population-based methods that deal in each iteration of the algorithm with a set of solutions instead of considering only one single

solution in each iteration like SA and TS do. The set of solutions that are treated at each iteration is called population, which facilitates the algorithms to explore the search space in a natural and intrinsic way. The manipulation of the populations determines the final performance of the algorithms. Evaluation algorithms concern an area of computer science that uses ideas from biological evolution to solve computational problem. Evolution is a method of searching among an enormous number of possibilities – e.g., the set of possible gene sequences – for "solutions" that allow organisms to survive and reproduce in their environments. Evolution can also be seen as a method for adapting to changing environments. And, viewed from a high level, the "rules" of evolution are remarkable simple: Species evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations. Yet these simple rules are thought to be responsible for the extraordinary variety and complexity we see in the biosphere.

There has been a variety of slightly different EA proposed over the years. Basically they fall into three different categories which have been developed independently from each other. There are Evolutionary Programming (EP), Evolutionary Strategies (ES) and Genetic Algorithms (GA). The most widely used form of evolutionary algorithms is GA (Goldberg, 1989), which will be the main focus in this dissertation.

2.1.3.1 Introduction of GA

The simplest version of a genetic algorithm consists of the following components:

1. A population of candidate solutions to a given problem, each encoded according to a chosen representation scheme. The encoded candidate solutions in the population are referred to metaphorically as chromosomes, and units of the encoding are referred to as genes. The candidate solutions are typically haploid rather than diploid.
2. A fitness function that assigns a numerical value to each chromosome in the population measuring its quality as a candidate solution to the problem at hand.
3. A set of genetic operators to be applied to the chromosomes to create a new population. These typically include selection, in which the fittest chromosomes are chosen to produce offspring; crossover, in which two parent chromosomes recombine their genes to produce one or more offspring chromosomes; and mutation, in which one or more genes in an offspring are modified in some random fashion.

A typical GA (as shown in Algo. 3) carries out the following steps:

1. Start with a randomly generated population of n chromosomes.
2. Calculate the fitness $f(x)$ of each chromosome x in the population.
3. Repeat the following steps until n offspring have been created:
    a. Select a pair of parent chromosomes from the current population, the probability of selection increasing as a function of fitness.

b. With probability p_c (the crossover probability), cross over the pair by taking part of the chromosome from one parent and the other part from the other parent. This form q single offspring.

c. Mutate the resulting offspring at each locus with probability p_m (the mutation probability) and place the resulting chromosome in the new population. Mutation typically replaces the current value of a locus with another value.

4. Replace the current population with the new population.
5. Go to step 2.

```
Initialize Chromosomes
while termination conditions not met do
        repeat
          if crossover condition satisfied then
          {select parent chromosomes;
          choose crossover parameters;
          perform crossover};
          If mutation condition satisfied then
          {choose mutation points;
          perform mutation};
          evaluate fitness of offspring
        until sufficient offspring created;
select new population;
endwhile
```

Algo. 3-A genetic algorithm template

Each iteration of this process is called a generation. A genetic algorithm is typically iterated for anywhere from 50 to 500 or more generations. The entire set of generations is called a run. At the end of a run, there are typically one or more highly fit chromosomes in the population. Since randomness plays a large role in each run, two runs with different random-number seeds will generally produce different detailed behaviors.

The simple procedure just described is the basic for most applications of GAs. There are a number of details to fill in, such as how the candidate solutions are encoded, the size of the population, the details and probabilities of the selection, crossover, and mutation operators, and the maximum number of generations allowed. The success of the algorithm depends greatly on these details.

2.1.3.2 Selection Methods

Individuals for producing offspring are chosen using a selection strategy after evaluating the fitness value of each individual in the selection pool. Each individual in the selection pool receives a reproduction probability depending on its own fitness value and the fitness value of all other individuals in the selection pool. This fitness is used for the actual selection step afterwards. Some of the popular selection schemes are discussed below.

a) **Roulette-wheel selection.** The simplest selection scheme is the roulette-wheel selection, also called *stochastic sampling with replacement*. This technique is analogous to

a roulette wheel with each slice proportional in size to the fitness. The individuals are mapped to contiguous segments of a line such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained.

b) **Rank-based fitness assignment.** In rank-based fitness assignment, the population is sorted according to the objective values. The fitness assigned to each individual depends only on the position of the objective values in the individual's rank. Ranking introduces a uniform scaling across the population.

c) **Tournament selection.** In tournament selection, a number of individuals are chosen randomly from the population and the best individual from this group is selected as the parent. This process is repeated as often until there are sufficient individuals to choose. These selected parents produce uniformly random offspring. The tournament size which is the parameter for tournament selection will often depend on the problem, population size, and so on. Tournament size takes values ranging from two to the total number of individuals in the population.

d) **Elitism.** When creating a new population by crossover and mutation, there is a big chance that we will lose the best chromosome. Elitism is the name of the method that first copies the best chromosome (or a few best chromosomes) to the new population. The rest is done in the classical way. Elitism can very rapidly increase performance of GA because it prevents losing the best-found solution.

There are also other selection methods. The choice of these methods has certain impact on the performance of the searching results. More detail can be referred in (Mitchell, 1998).

2.1.3.3 Recombination (Crossover) Operators

Crossover selects genes from parent chromosomes and creates a new offspring.

a) ***K-point* Crossover.** One-point and two-point crossovers are the simplest and most widely applied crossover methods. In one-point crossover, illustrated in Figure 12, a crossover site is selected at random over the string length, and the alleles on one side of the site are exchanged between the individuals. In two-point crossover, two crossover sites are randomly selected. The alleles between the two sited are exchanged between the two randomly paired individuals (as shown also in Figure 12). The concept of one-point crossover can be extended to k-point crossover, where k crossover points are used, rather than just one or two.

b) **Uniform Crossover.** Another common recombination operator is uniform crossover. In uniform crossover, see in Figure 12, every allele is exchanged between the pair of randomly selected chromosomes with a certain probability, p_e known as the swapping probability. Usually the swapping probability value is taken to be 0.5.

c) **Uniform Order-Based Crossover.** In order-based crossover, two parents (say P_1 and P_2) are randomly selected and a randol binary template is generated (see in

Figure 13). Some of the genes for offspring C_1 are filled by taking the genes from parent P_1 where ther is a "1" in the template. At this point we have C_1 partially filled. The genes of parent P_1 in the positions corresponding to "0" in the template are taken and sorted in the same order as they appear in parent P_2. The sorted list is used to fill the gaps in C_1. Offspring C_2 is created by using a similar process.

The k-point and uniform crossover methods described above are not well suited for search problems with permutation codes such as the ones used in the traveling salesman problem. They often create offspring that represent invalid solutions for the search problem. Therefore when solving search problems with permutation codes, a problem-specific repair mechanism is often required (and used) in conjunction with the above recombination methods to always create valid candidate solutions. Another alternative is to use recombination methods developed specifically for permutation codes, which always generate valid candidate solutions. The uniform order-based crossover described above is such crossover techniques and there are other crossover which always generates valid candidate solutions, such as *order-based Crossover*, *partially Matched Crossover (PMX)* and *cycle Crossover (CX)*.

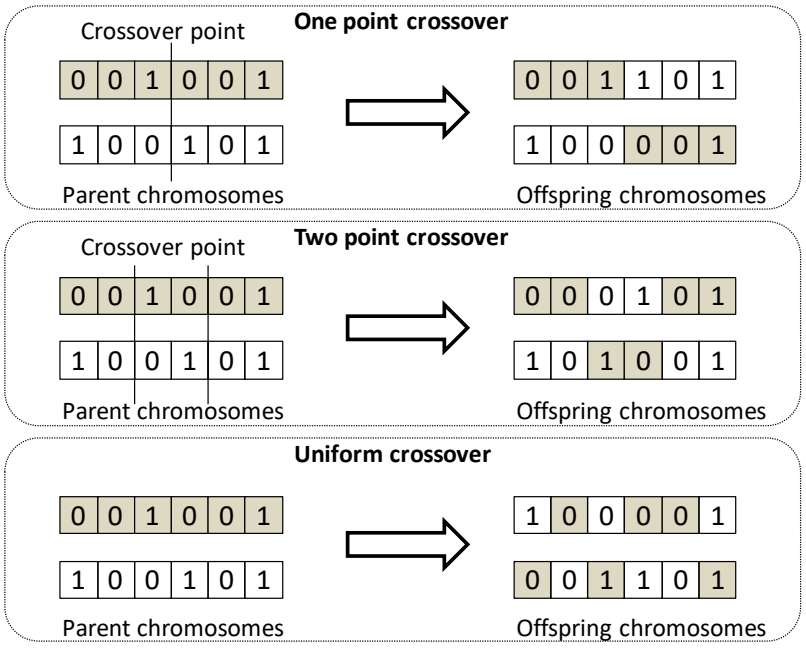


Figure 12-One-point, two-point, and uniform crossover methods

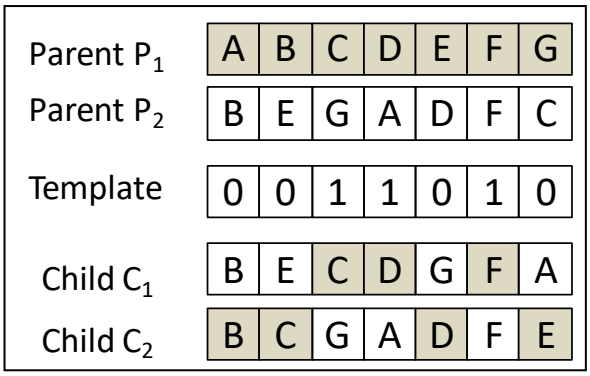


Figure 13-Illustration of uniform order crossover

2.1.3.4 Mutation Operators

If we use a crossover operator, such as one-point crossover, we may get better and better chromosomes but the problem is, if the two parents (or worse, the entire population) has the same allele at a given gene then one-point crossover will not change that. In other words, that gene will have the same allele forever. Mutation is designed to overcome this problem in order to add diversity to the population and ensure that it is possible to explore the entire search space.

In Evolutionary Strategies, mutation is the primary variation/search operator. Unlike evolutionary strategies, mutation is often the secondary operator in GAs, performed with a low probability. One of the most common mutations is the bit-flip mutation. In bitwise mutation, each bit in a binary string is changed (a 0 is converted to 1, and vice versa) with a certain probability, P_m known as the mutation probability. As mentioned earlier, mutation performs a random walk in the vicinity of the individual. Other mutation operators, such as problem-specific ones, can also be developed and are often used in the literature.

2.2 Formalization of the distribution problem

2.2.1 Architecture modeling

The multi-core architecture is composed of a set of cores $\{\pi_1, \dots, \pi_K\}$ and a set of memories $\{M_1, \dots, M_L\}$, with $L > K$ and M_1 to M_k are attached to the local memories of cores π_1 to π_K, while M_{k+1} to M_L represent the shared memories. The communications between the cores are realized by buses. An example of this kind of architecture is TC27x that we choose as our hardware multi-core platform. TC27x is a tri-core microcontroller. As shown in Figure 14, there are two category memories: the local memories attached to each core and the global memories. For the record, all the memories can be accessed by any cores.

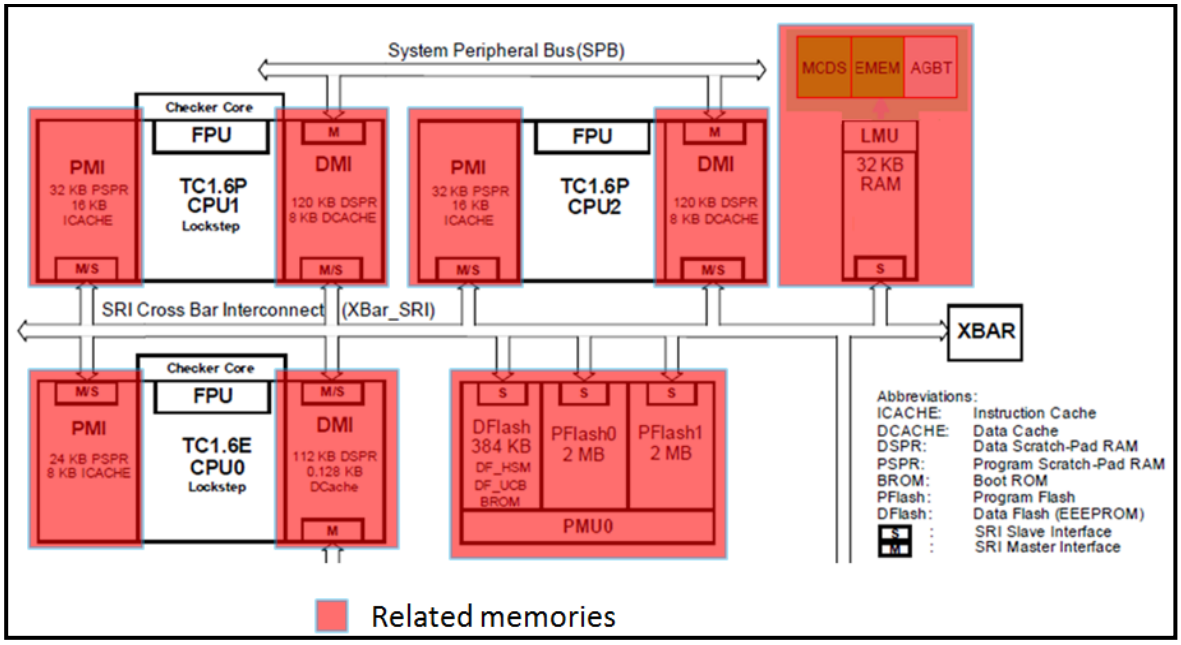


Figure 14-Hardware Architecture

There are three cores in this architecture, two identical cores TC1.6P and another core TC1.6E. All these three cores execute the same set of instructions. As described before, this multicore architecture can be seen as a uniform architecture.

There are two independent on-chip buses in the tri-core architecture: Shared Resource Interconnect (SRI) and System Peripheral Bus (SPB). The SRI is the crossbar based high speed system bus for TC 1.6.x CPU based devices. The SPB connects the TC1.6 CPUs and the general purpose DMA module to the medium and low bandwidth peripherals. More details can be referred to the manual (Infineon, TC27x 32-Bit Single-Chip Microcontroller, User's Manual, 2012).

For respecting the HW platform, the model should consider different type of memories. There are two types of memories in multicore architecture: memories attached to each core and memories shared by all the cores. The time for one core to access its attached memory is shorter to the memories attached to other cores. And the time to access to shared memories is a compromising way. The caches in this dissertation are considered inhibited.

2.2.2 Application modeling

The software architecture is modeled using a directed graph $G(V, E)$, such that V is a set of nodes and E is a set of edges, also called transitions (links between nodes). A node is modeled as an execution time, a trig mode, a period. A transition has a weight that depends on the size of data transmitted, the period of the producer, etc. The graph size is optimized by the creation of buses between nodes.

The node can be periodic or triggered by some events. For the periodic nodes, we assume that they are associated with a period T_i. Each node $\rho_i \in V$ is also associated with execution information that contains two parts: execution time C_i and variable accessing time A_i.

The execution time C_i represents the time for a node to execute some instructions. C_i is influenced by two factors. One is the performance of the core on which the node is located in. The higher computing power, the faster the node will finish its corresponding execution. In a real-life automotive system, the real-time constraints also depend on the execution modes, such as the engine speed or driving modes. E.g. the amount of executed codes depends on the vehicle speed. In the following we denote these contexts cases, and it is the second factor that influences C_i. A weight ω is associated to each case to model its importance in the system (high value of ω means high importance). So for a given node its execution time varies with its location and the contexts case.

The accessing time A_i mentions the time for a node to read or write its related variables located in the memories. In our multi-core architecture, each core is associated with a local distributed memory. Nodes can also access data in shared memories. It is worth to mention that all the memories can be accessed by all the nodes distributed to all the cores, which implies that the accessing time for a node to write or read a variable varies with the location of the node as well as the location of its variable. It is obvious that A_i is much shorter if we locate its accessed variables into the local memory of the core where this node is located. Accessing a variable in the local memory of another core is much slower; and accessing to shared memory is dedicated to data exchanged between cores.

2.2.2.1 Variable access model

For each node ρ_i, its accessed variables $\{\theta_i\}$ contain a list of variable it writes $\{\theta_{iw}\}$ and a list of variable it reads $\{\theta_{ir}\}$: $\{\theta_i\} = \{\theta_{iw}\} \cup \{\theta_{ir}\}$ (shown in Figure 15(a)). Each variable is composed of several attributes:

- Data size: the size of a data prototype. For example: for an irv data prototype with type of SInt16, its size is 2 byte.
- Data position: in our multi-core platforms, the data can be distributed in the shared memories or the local memories. The local data are the data that are distributed to the shared memories and the global data are those distributed to the local memories.
- Data rate: the total size of data transferred between runnables in a transition in a unit of time. For a transition between runnable ρ_i and ρ_j, the period of ρ_i is T_i and the period of ρ_j is T_j. The variables transferred in this transition are denoted as $\theta_{i\to j}$: $\theta_{i\to j} \in \theta_{iw}$ and $\theta_{i\to j} \in \theta_{jr}$. So the sent data rate for this transition is $\theta_{i\to j}/T_i$ and the received data rate is $\theta_{i\to j}/T_j$ (shown in Figure 15(b)).
- Data unit: the physical unit of each variable. Some data unit varies dramatically over time and some data varies rarely. There are also some units varying depending on the data.

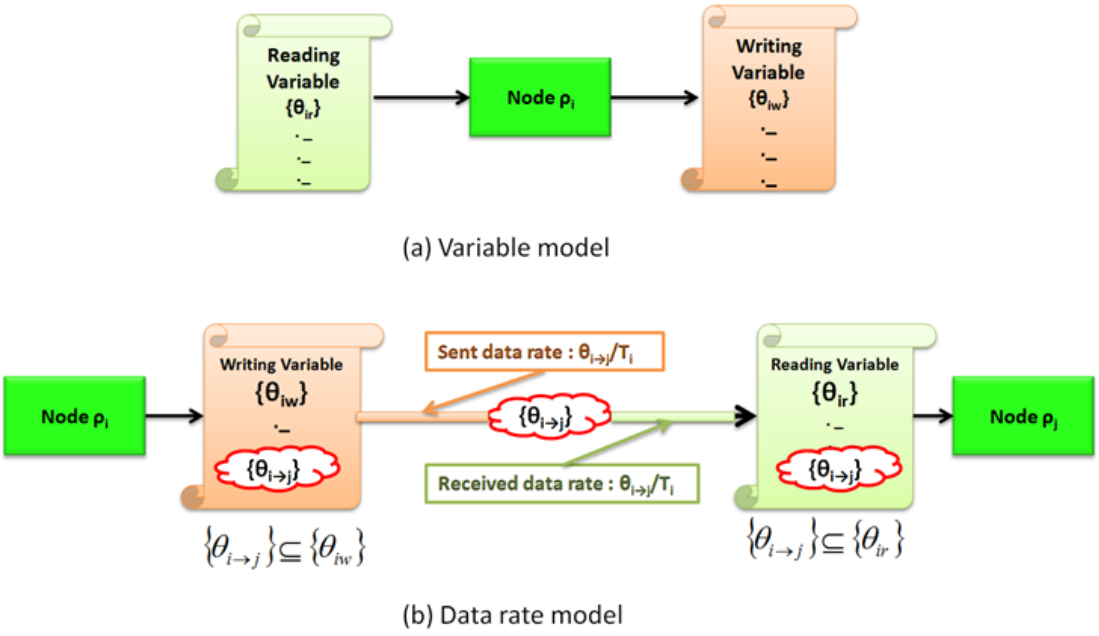


Figure 15-Variable access model

2.2.2.2 General transition model

The communications between nodes are presented as transitions E. Each transition $E_{i,j}$ contains two nodes ρ_i and ρ_j, $(\rho_i, \rho_j \in V)$, model $\rho_i \rightarrow \rho_j$ presents the dependency between ρ_i and ρ_j, where ρ_i is the predecessor of ρ_j and ρ_j is the successor of ρ_i. The predecessor ρ_i sends a set of variables that are received by the successors. Similarly, the successor ρ_j receives a set of variables from predecessor. Therefore, without specifying the granularity or the type of communications, a transition can be modeled as shown in Figure 16.

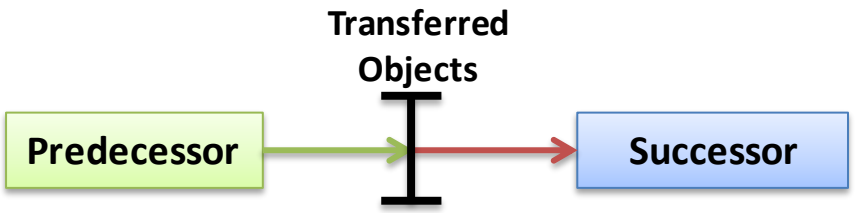


Figure 16-General transition model

2.2.2.2.1 Enumeration of transitions

The general transition model imposes of enumerating all the transitions in the unique way. The examples below illustrate how to transform the original graphs such that it appears the transitions each of which is associated with one single predecessor and one single successor.

- Case1: A predecessor accesses a transferred object that is consumed by multi-successors (Figure 17).

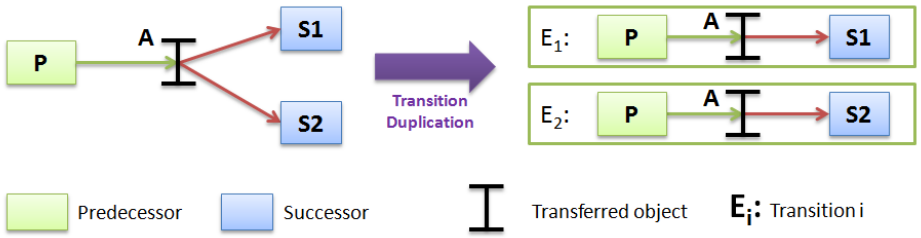


Figure 17-Sources duplication for case1

- Case2: A predecessor accesses different transferred objects that are consumed by multi- successors (Figure 18).

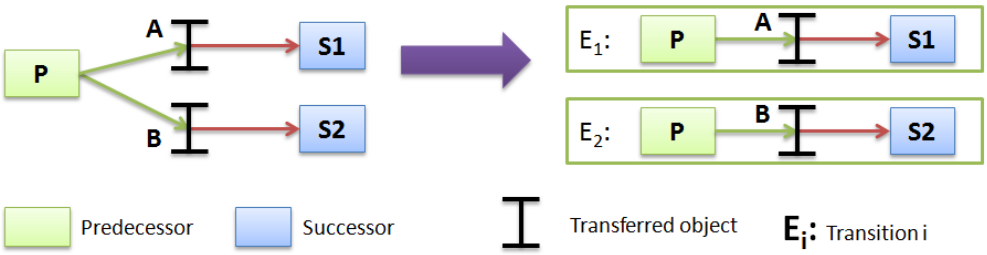


Figure 18-Sources duplication for case2

- Case3: A successor accesses a transferred object that is produced by multi-predecessors (Figure 19).

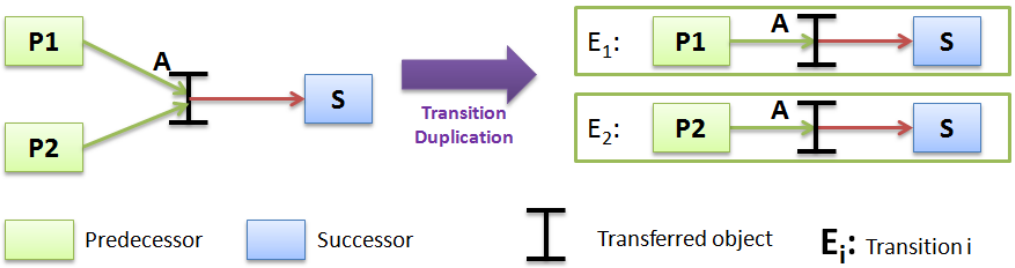


Figure 19-Sources duplication for case3

- Case4: A successor accesses different transferred objects that are produced by multi-predecessors (Figure 20).

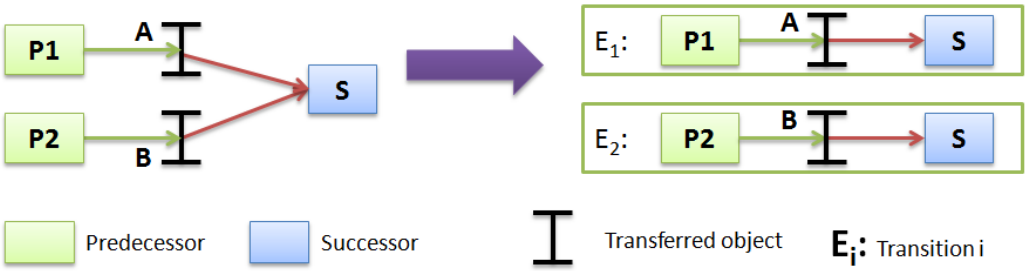


Figure 20-Sources duplication for case4

2.2.2.2.2 Communication Bus

If multi-objects are transferred between a source and a destination, the transitions are encapsulated in a communication bus to simplify the presentation as shown in Figure 21.

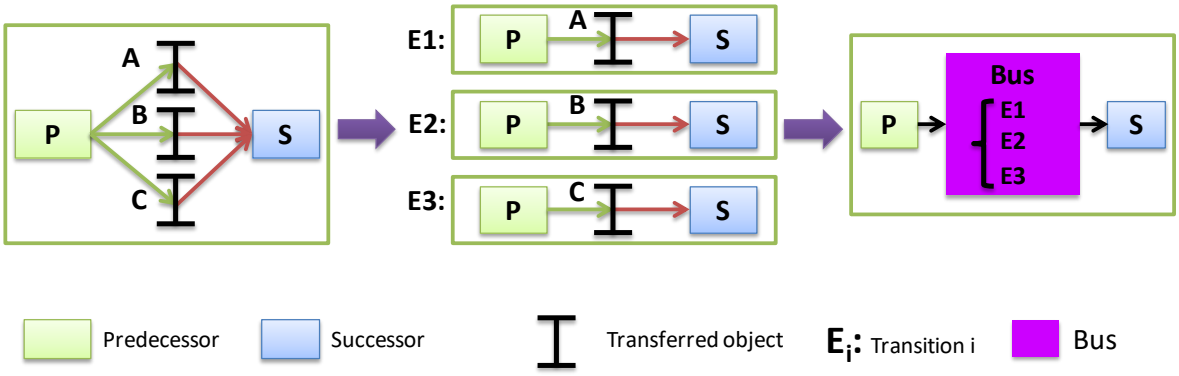


Figure 21-Communications Bus

2.2.3 Partitioning

The partitioning involves the distribution of a set of nodes $\{\rho_1, \dots, \rho_I\}$ to the cores and also a set of variables $\{\theta_1, \dots, \theta_J\}$ to the memories. We note $\rho_{i,k}$ when the ith node is distributed to kth core and $\theta_{j,l}$ when the jth variable is distributed to lth memory. $A_{\theta_j}(k,l)$ mentions the accessing time for the node located on the kth core to access the variable θ_j located on lth memory. We also define a set of contexts cases $\{K_1, \dots, K_N\}$, and ω_n is the weight for the nth case. Then, $C_i(k,n)$ represents the execution time for ith runnable located in the kth core and in the nth case. Thus when we distribute a node ρ_i to core π_k, based on its execution time, accessing time and period, this runnable results in a load $u_{\rho_{i,k}}$:

$$u_{\rho_{i,k}} = \frac{\sum_j A_{\theta_j}(k,l) + C_i(k,n)}{T_i}$$

The load of core π_k is the sum of the loads caused by the runnables distributed to this core, mentioned as u_{π_k}:

$$u_{\pi_k} = \sum_i u_{\rho_{i,k}}$$

The inter-core communications represent the main challenge to pass from single-core to multi-core architectures. The overhead introduced by inter-core communications is one of main raisons that degrade the performance of multi-core system. In order to minimize this overhead, the applications have to be analyzed in a fine degree. The overhead of inter-core communication is estimated by summing the number of data access per millisecond. We define a notion of *FetchSize* for each variable (data) transferred by transition. The *fetchsize* depends on the size of variable as well as features of concrete hardware: for a transition E_{ij} with variable $\theta_{i\to j}$, we denote the size (in bit) of variable as $S(\theta_{i\to j})$ and the size of the

target hardware is $S(H)$. Thus the fetchsize of this transition is $\mathcal{FS}(E_{i,j}) = \left\lfloor \frac{S(\theta_{i\to j})}{S(H)} \right\rfloor$. For example, the fetchsize of a transition that transfers data with size of 16 bit is 1 if the target platform is a 32-bit microcontroller. The overhead caused by a transition that crosses the cores is

$$u_{E_{i,j}} = \frac{\varepsilon_w + \mathcal{FS}(E_{i,j}) \times \mathcal{FC} \times \mathcal{CT}}{T_i} + \frac{\varepsilon_r + \mathcal{FS}(E_{i,j}) \times \mathcal{FC} \times \mathcal{CT}}{T_j}$$

Where $\mathcal{FC}$ is the number of cycles taken by each fetch, and $\mathcal{CT}$ is the time taken by each cycle. $\mathcal{FC}$ and $\mathcal{CT}$ are specified by the target hardware. ε_w and ε_r are two constants for the writing and reading delay, the values depend on the communication mechanism. For example, in the Autosar application, these can be the delay caused by the creation of IOC channel by RTE.

In resume, the objective function $\mathcal{F}$ is defined based on the above notions:

$$\mathcal{F} = \sum u_{E_{i,j}},$$

$$u_{E_{i,j}} = \begin{cases} \frac{\varepsilon_w + \mathcal{FS}(E_{i,j}) \times \mathcal{FC} \times \mathcal{CT}}{T_i} + \frac{\varepsilon_r + \mathcal{FS}(E_{i,j}) \times \mathcal{FC} \times \mathcal{CT}}{T_j}, if\ i \neq j \\ 0, else \end{cases}$$

2.2.4 Cost function and constraint formalization

For objective function or cost function, we consider different criteria. In this chapter, we consider the criterion of inter-core communication overhead. We will present other criteria in the next chapter.

We denote the cost of transitions that cross the cores as $\widetilde{E_{i,j}}$, the objective function is:

$$\mathcal{F} = \sum u_{\widetilde{E_{i,j}}}$$

The load of the multicore distribution must be well balanced, with a tolerated deviation of α. It appears as the main design constraint in the optimization formulation:

$$\Omega: u_{max} - u_{min} \leq \alpha$$

Where u_{max} are the loads of the core that is most occupied and u_{min} is the load of the core that is least occupied.

It is obvious that different ways of partitioning will change the cost value of objective function. Figure 22(a) shows a simple example: the application contains 3 runnables ρ_1, ρ_2 and ρ_3. ρ_1 send variable θ_1 to ρ_2 and θ_2 to ρ_3. The hardware model shown in Figure 22(b) consists in a 2-core system with a shared memory M_3. Besides, each core is attached to a

local memory M_1 and M_2. We assume that the execution time for each runnable at each core is identical. The objective is to distribute the application to this 2-core system. Solution in Figure 22(c) allocates all the runnables in one core, and distributes the variables in its local memory. This could minimize the accessing time, so the communication overhead is low. But the loads of CPU are not well balanced as the other core is empty. Solution in Figure 22(d) allocates the runnable ρ_3 to the other core, so when runnable ρ_1 finishes its execution, ρ_2 and ρ_3 can execute in parallel. Therefore the loads of CPU are better balanced. However, the communication overhead is increased as the accessing time for the variables allocated at the shared memory is much longer. This compromise is considered in our objective function.

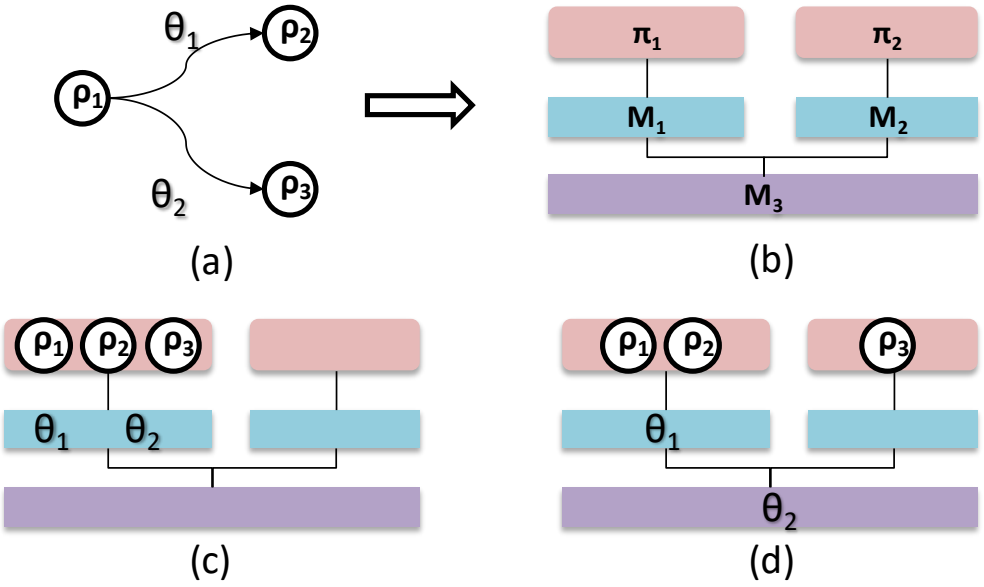


Figure 22-Explanation for objective function. (a) Application; (b) Hardware model; (c) and (d) Solutions considering different criteria.

In this work, we aim at developing a practical policy for partitioning software applications, composed of several hundreds of nodes, onto multiple cores that will minimize this objective function, while respecting the dependencies and the constraints in AUTOSAR.

2.2.5 Description of the optimum solutions searching method

The partitioning solution is represented as a vector in which each element represents the position for runnables or variables. The vector is an ordered list with the length of $l = J + I$, where the J represents the number of the variables and I is the number of nodes to be distributed. In the position p of the vector, $p \in [0, J)$, a memory is distributed for the corresponding variable and in position $p, p \in [J, l)$, a core is attached to the corresponding node. The different combinations of the memories and cores will change the value of objective function. In order to deal with this combinatorial optimization problem, we take the metaheuristic algorithms as a solver. The method to search the optimum solution is described as follows:

- **Initial solution** can be obtained in a random way as well as by heuristic guide. The quality of the initial solution would affect final solution;
- **Neighborhood structure** of a solution defines its possible move direction for improvement, which involves 2 operators: operator N1 changes only the memory attached to one single variable to another memory or operator N2 changes only the core attached to one single node to another core. The move will choose one operator randomly each time;
- **Constraints** guarantee the viability of solutions on each move proposed by the neighborhood operator: all the solutions (including the initial solution) shall respect all the defined constraints;
- **Metaheuristic algorithms** provide the searching policies to find the optimum (or good) solutions in an efficient way: starting at the initial solution, the improvement is effectuated by a single move (defined by neighborhood structure) each iteration.

In this work, we apply three metaheuristic algorithms: SA, GA and TS. All the algorithms share the same framework such as initial solution, neighborhood structure. Each algorithm performs different searching policies to find the final solution. The evolution of solutions iteration by iteration is illustrated in Figure 23, which shows the convergence of optimization process by our objective function with two goals: benefit the acceleration of performance from single-core and respect the real-time constraints on the dependent tasks.

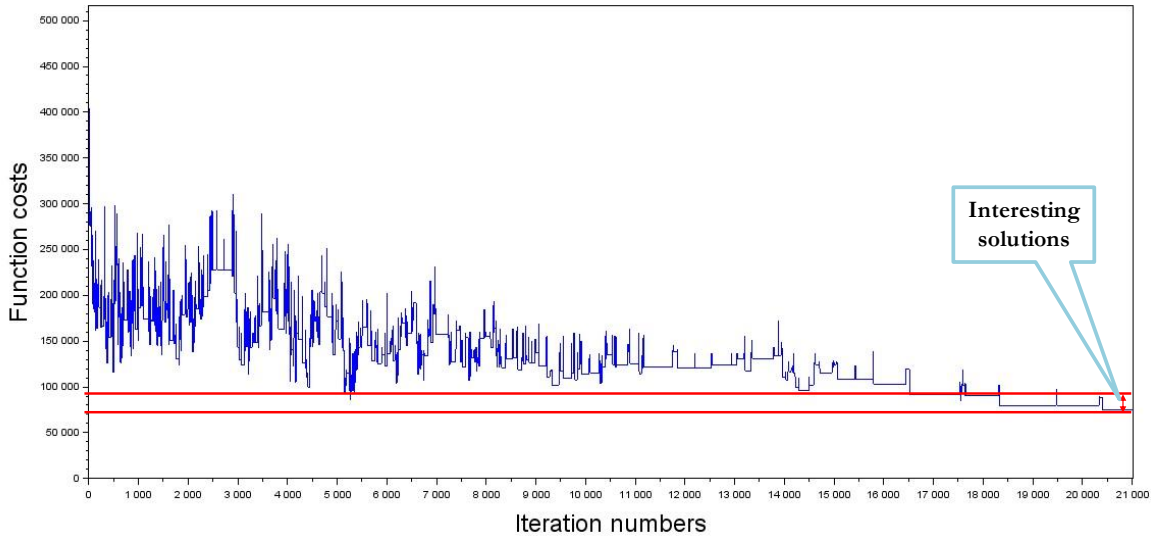


Figure 23-An example of search result by SA

The results obtained with this method show the contributions of our work:

Quality of the solutions explored according to the cost function;

Diversity of the solutions around the optimum at the convergence of the method. This diversity will provide the designer the guide needed to take its final choice (Miramond & Delosme, 2005) ;

Scalability of the method over complex AUTOSAR applications potentially composed of several hundreds of runnables and several thousands of transitions.

2.2.6 Design space exploration

The partitioning of automotive applications in multi-core systems requires a design space exploration with many parameters to be considered. From the software point of view, it includes the follow points:

- ***Software Allocation***. How to decompose the allocations into partitions and distribute the partitions on different cores?
- ***Task-set definition.*** What is the set of tasks that should be used for an efficient and secure scheduling?
- ***Sequencing definition***. How executable entities should be ordered in tasks? And which parameters should be assigned to tasks, in order to comply with real-time and functional requirements?

  In single-core system, scheduling configuration can be computed using the design of the implementation model (e.g., a model from ***MATLAB/SIMULINK*** can be used to generate a scheduler). In multi-core system however, the implementation model should take into consideration parallelism before doing this step, which is not done when porting single core application onto multi-core. When we consider multi-core only at SW level, this leads to a very complicated task.
- ***Application synchronization***. Cooperation between cores requires specific inter-core synchronization mechanisms. E.g. the synchronization points in Figure 24 shall be guaranteed to make sure the correcte cooperation between the two cores.

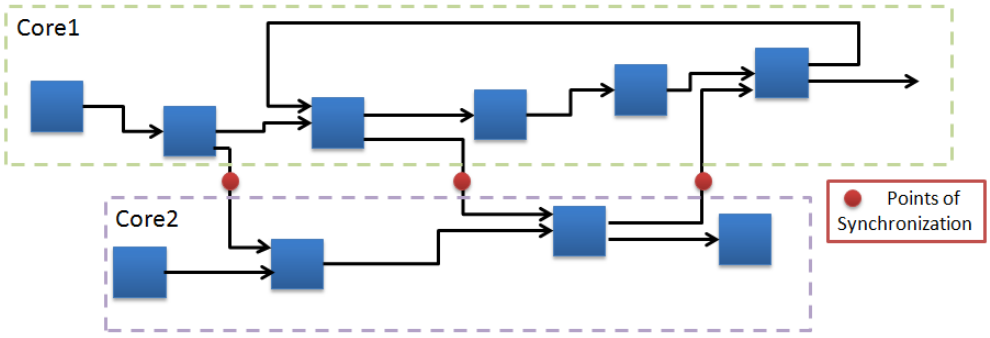


Figure 24-Synchronization example

It is worth to note that the points of task-set definition and sequencing definition already exist in the single core system.

From the hardware point of view, the following points that exist already in the single-core context are impacted by multi-core.

- ***Data mapping into memories (SWCs).*** Microcontrollers have several levels of memories (e.g. one local memory per core and one shared memory). Each architecture has its own hierarchical composition of the cores and memories, e.g. tri-core architecture in Figure 25 consists in 3 cores, each core with a local memory. Besides, there is one shared memory that can be accessed by all the cores. This kind of multi-core architecture imply three type of accessing time: the accessing time to the local memory, the accessing time to the distant memory (local memory of another core) and the accessing time to the shared memory. The $T_{i \to j}$ in Figure 25 means the accessing time from Core-*i* to the memory *j*.

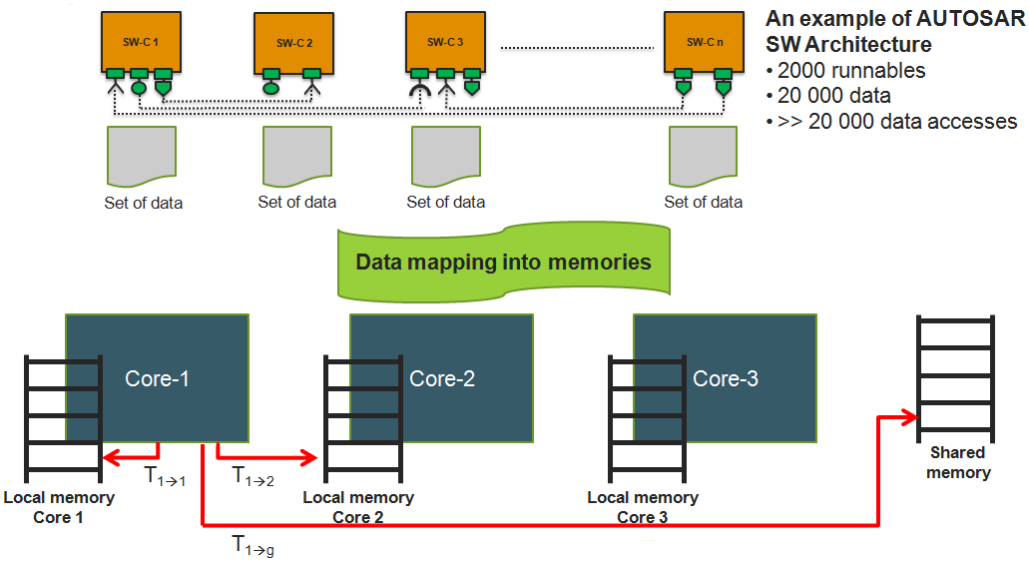


Figure 25-Illustration of data mapping into memories

- ***Hardware safety mechanisms.*** Microcontrollers have a set of features that can be activate in order to comply with ASIL X (ISO 26262). Mainly Memory Protection Unit (MPU) has a significant impact on multicore especially on the aspect of the communication time.

These parameters interact with each other: The SW allocation choices drive data allocation and in return, data allocation also impacts SW behavior. E.g., a first allocation can be provided by considering a same accessing time to data. After that, the data accessing time will be corrected to obey the real case. This correction changes several features of allocation such as CPU loads, communication overhead and so on, which can impact the choices of SW allocation.

SW synchronization problem is influenced by SW allocation choices. We make the distinction between fine grain synchronization, which correspond to data protection and coarse grain synchronization (much more difficult) that target software flow mastering. For example:

- Data allocation impacts synchronization problem as the data protection mechanism is needed for assuring data consistency

- An appropriate choice of SW allocation with few synchronization points will facilitate the synchronization process.

Safety requirements (ISO 26262) also affect directly or indirectly the SW allocation and data allocation. For example:

- Some safety mechanisms might require the separation of SWCs of different Automotive Integrity Safety Levels (ASIL) from each other (e.g. spatial isolation made by allocated for separation of concerns). This isolation in space can be use to make sure that the SWCs are not able to write to other SWCs data (partitioning) using software/hardware support such as a memory protection or memory management unit (which also influences the data allocation).
- Microcontroller architecture should comply with safety requirement. For example, ASIL D application should be allocated to cores that run in lockstep.

The interdependence between these parameters as shown above exacerbates the design complexity. The design space needs to be explored by considering these entire requirements. Therefore, design space exploration should be formulized as optimization problems and powerful optimization techniques are needed. We adopt Meta-heuristic algorithms as solver to deal with these optimizations problems that are formulized as combinatorial optimizations (CO) problem. The relative theories are previously presented in section 2.1.

2.3 Autosar Application

The main work of this dissertation is to integrate seamlessly our partitioning method into an AUTOSAR development process. For doing that, we model the application of AUTOSAR in order to allow automatic exploration of its deployment onto multi-core architectures adapted with our model presented earlier. Basically, AUTOSAR development process can be divided in two steps:

A system is described at higher level without knowing if it will be allocated on several ECUs or only to one ECU and so on, without knowledge on the core in which software will be executed.

At configuration level, SWCs have to be allocated to cores. This allocation is done using the operating system configuration, by allocating runnables to tasks, tasks to OS-Applications, and OS-Applications to cores. We recall that a given OS-Application is statically assigned to a core.

2.3.1 Communication overhead in Autosar application

In order to consider of inter-core communication overhead to the Autosar applications, we analyze and model the communications in the architecture of Autosar application.

Communication model in AUTOSAR

According to the AUTOSAR Methodology, there are three types of communication:

- **Inter ECU communication**: already available using well defined APIs of the communication stack (COM) ;
- **Intra OS-Application communication**: always handled within the RTE ;
- **Inter OS-Application Communication**: The communication channel depends on the set of software mechanisms used for data protection:
  - IOC (Inter OS-Application Communicator) is used when we need to cross memory protection boundaries (e.g. when MPU is used for safety reason). The IOC is an operating service, executed in supervisor mode (i.e. a system call has to be done before performing a communication);
  - RTE is used when the communication can be performed in shared memory mode. In that case, the IOC service provided by the operating system is not required.

Manipulated data also need to be protected. In fact, with a 32-bits hardware architecture, only 32-bits can be manipulated atomically. For greater size, a lock (e.g., spinlock) has to be taken. That implies that 4 kinds of communications can be used for inter-OS-Application communications (order by time required to perform the data access).

- By RTE without spinlock, the fastest way to handle data access (no protection);
- By RTE with spinlock, if data is too big to be manipulated atomically (data consistency is handled);
- By IOC without spinlock, memories regions protected by MPU (safety);
- By IOC with spinlock, memories regions protected by MPU and data handled > 32 bit.

Runtime behavior impact depends on the kind of communication. For example, the time required to access data in a memory protected by spinlock is lower than the time required to access data in a memory protected by spinlock and by MPU (additional System Calls). This directly impact WCET of tasks and CPU load, which can be significant at ECU level.

Do not forget that an inter OS-Application communication may not necessarily require a cross core communication. E.g., it is possible to allocate some OS-Applications to a same core.

It is also worth noting that OS-Application have been created to tackle memory protection problems; i.e. most of inter-OS-Application should be performed by the IOC. However, an OS-Application cannot be splitted into cores, so we have at least one OS-Application per core.

In AUTOSAR, there is no restriction of the protection level of *inter-core inter-OS-Application communication*. The different kinds of communication are illustrated on Figure 26.

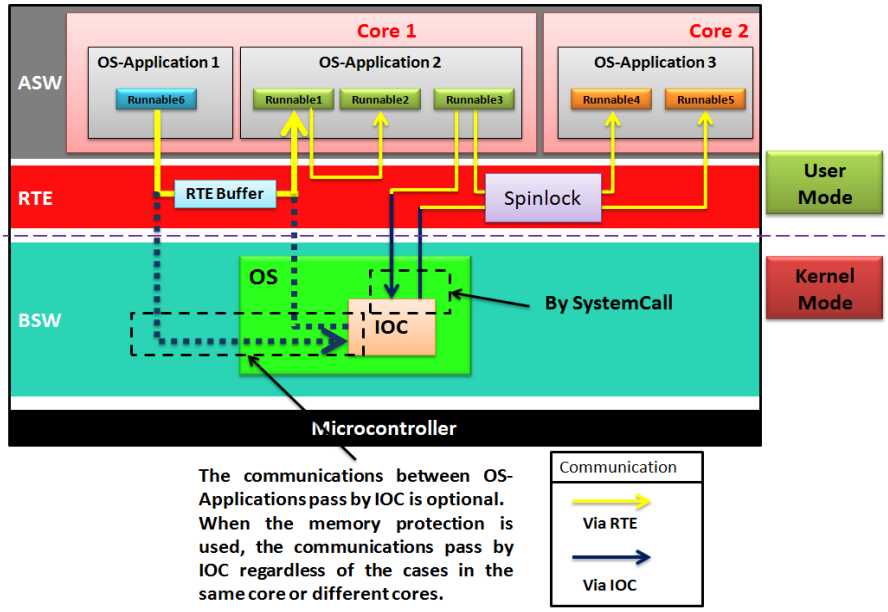


Figure 26-Communication in Autosar

2.3.1.1 Classes of communication

The communication of application is presented by a set of transitions between runnables. There are three levels of categories for these transitions: the SW architecture level, the RTEEvent triggering level and partitioning level as shown in Figure 27.

- SW Architecture level: at this level, the communications are categorized into 2 groups: the communication realized by the Ports and Interfaces and the communication realized by the IRV.
- RTEEvent triggering level: this level classifies the transitions into several classes according to the RTEEvents that activate the runnables
- Partitioning level: at this level the communication are managed by IOC or by RTE.

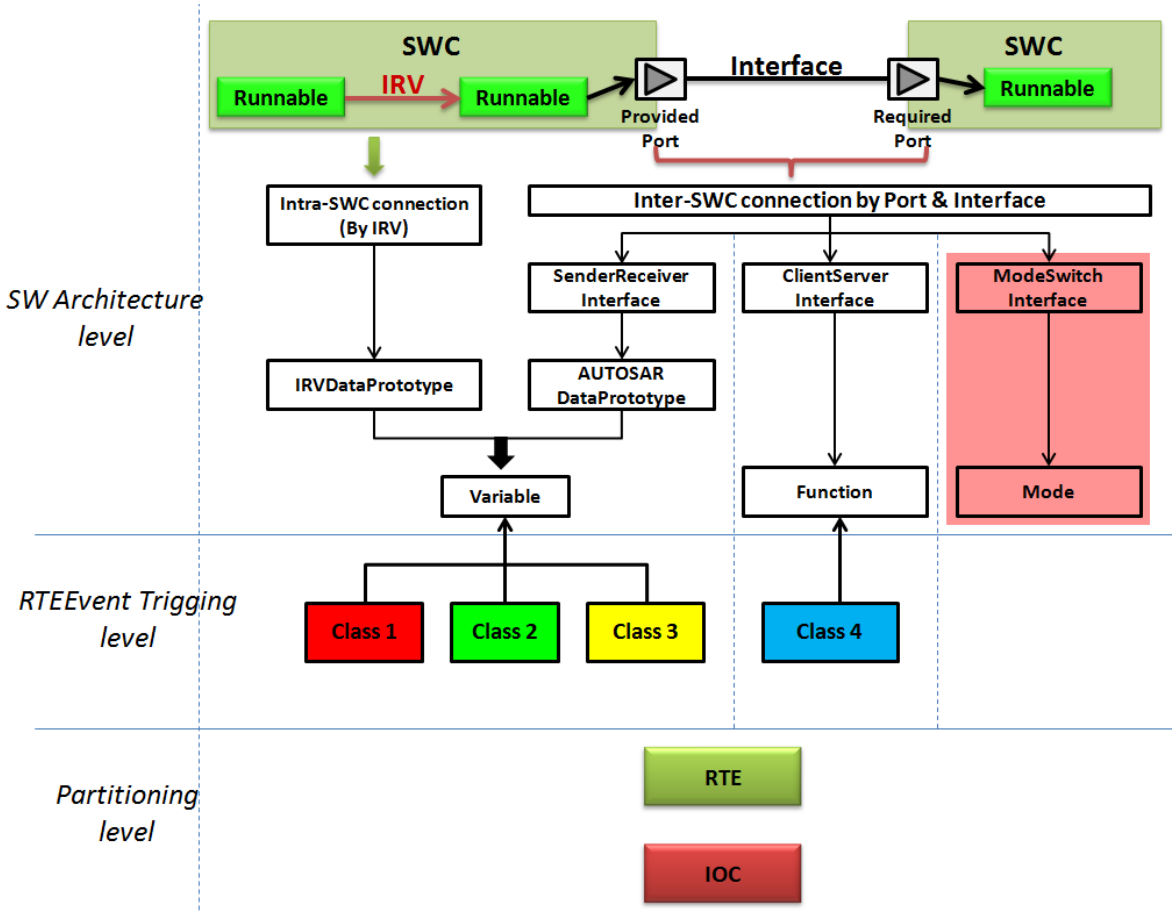


Figure 27-Different levels of categories for communications

2.3.1.1.1 SW Architecture level

At SW Architecture level, the transitions can be categorized into 2 groups: inter-SWC connection and intra-SWC connection.

- **Inter-SWC connection** represents the transitions between two runnables from different SWCs. These transitions are implemented by Ports and Interface. The Interface has three types: the SenderReceiverInterface, ClientServerInterface and ModeSwitchInterface. For now, the ModeSwitchInterface that provides several modes is not in our concern. SenderReceiverInterface provides data that can be written by producer runnables and be read by consumer runnables. While ClientServerInterface contains several services (functions calls) that are provided by server runnables to response the call of client runnables. To build such inter-SWC connection, a runnable that writes variables or provides services connects the provided Port attached to the related Interface and in the other side, another runnable that reads these variables or calls these services connects the required Port attached to the same Interface.
- **Intra-SWC connection**, on the other hand, represents the transitions between runnables from the same SWC. These transitions are implemented by IRV, where

the communications are realized by writing and reading the IRV-data by runnables. This level of communication is shown in the top of Figure 27.

2.3.1.1.2 RTEEvent triggering level

RTEEvent triggering level classifies the transitions into 4 classes according to the RTEEvent type that activates the runnables.

- Class 1: both producer runnable and consumer runnable are activated by Timing Event.

  This means that both runnables are periodic. By comparing the period of producer runnable (noted as Tp) and the period of consumer runnable (noted as Tc), Class 1 can be further classified into 3 series, where

  - Series 1: Tp = Tc, that the periods for both runnables are identical. Thus, the data rate can be presented by size of data (byte) / (Tp or Tc).
  - Series 2: Tp < Tc, this is under-sampling case that the periods of producer runnables is grater, which will result in *data loss*. Two type of data rate are considered: producer data rate (data size by Tp) and consumer data rate (data size by Tp). The data loss rate is Tp/Tc.
  - Series 3: Tp > Tc, over-sampling case that the periods of producer is smaller, which will result in *data duplication*. Like series 2, two type of data rate are considered and the data duplication rate is Tp/Tc.

- Class 2: either producer or consumer runnable is activated by Timing Event, but not both.

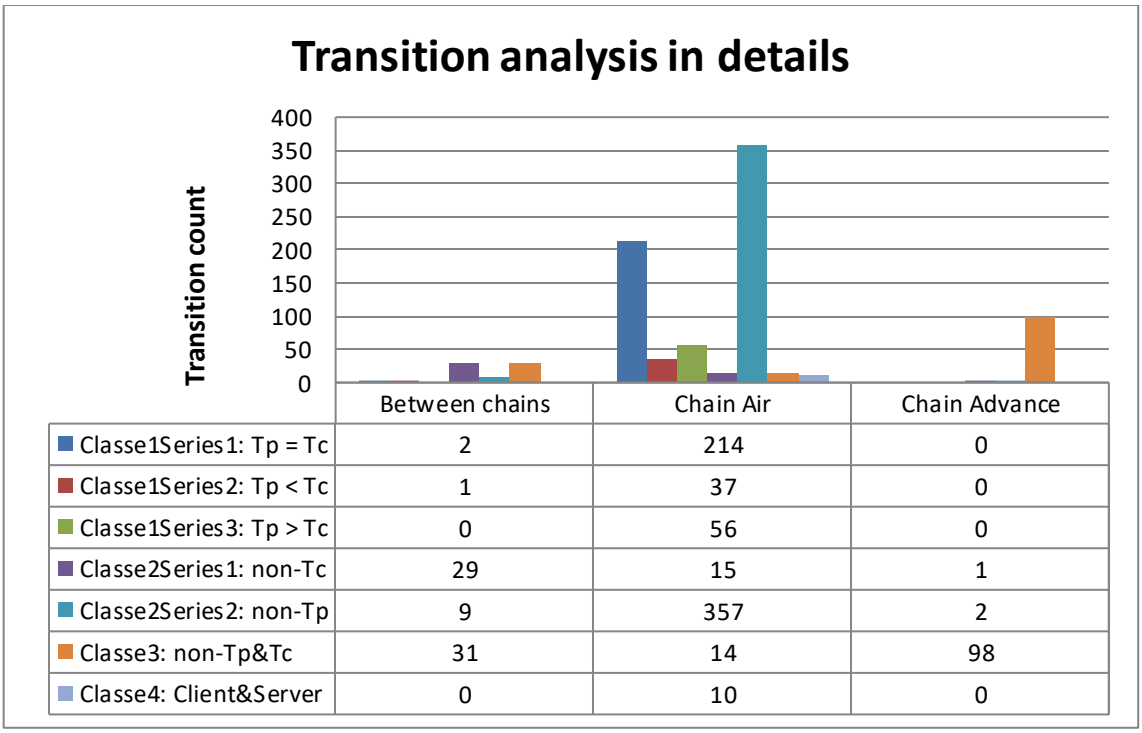
  Class 2 can be further classified into 2 series:

  - Series 1: Producer runnables is periodic.
  - Series 2: Consumer runnables is periodic.

- Class 3: Neither producer nor consumer runnable is activated by Timing Event.
- Class 4: this class focuses on the communication between server runnable and client runnable and the RTEEvent type of server is OperationInvokedEvent (OIE).

The relation with implementation level is shown in Figure 27.

We did a quantitative analysis of the transitions in automotives applications. The complete statistic results can be referred in Annex 1. Here we show in Figure 28 the distribution of the transitions in terms of their classification. The concerned application represents a portion of full application of Engine Control System, which contains two chains of SWC: air chain and advance chain. We can notice that the SWCs in chains air are strongly connected by class1 series1 (as to high quantity of class2 series 2, this is due to the mode switch connection, which we don't study in this dissertation.). When allocating the SW, the transitions of class1 series1 shall be considered as strong connection.



	Between chains	Chain Air	Chain Advance
Classe1Series1: Tp = Tc	2	214	0
Classe1Series2: Tp < Tc	1	37	0
Classe1Series3: Tp > Tc	0	56	0
Classe2Series1: non-Tc	29	15	1
Classe2Series2: non-Tp	9	357	2
Classe3: non-Tp&Tc	31	14	98
Classe4: Client&Server	0	10	0

Figure 28-Distribution of the transitions for two chains

2.3.1.1.3 Partitioning level

The implementation level and RTEEvent trigging level already exist in the context of single-core, while with the intention to migrate to multi-core platform, the communication inside the core can be managed by RTE while the communication pass between cores should be managed by IOC, which derive the partitioning level: the transitions passed by RTE and the transitions passed by IOC.

The determination of the partitioning level for transitions is part of multicore SW distribution, which requires balancing several elements:

- **Classification from RTEEvent triggering level:** each transition belongs to one of 4 classes presented in part 2.8.1.2.2. For each class, the requirement for partitioning is different.

  1) Class 4: The IOC provides sender-receiver (signal passing) communication only. For the communication in class 4 that are composed of client-server communications, the RTE translates Client-Server invocations and response transmissions into Sender-Receiver communication

  2) Class 1 series 1: the overhead for IOC is quite high. In order to reduce the number of IOC transitions in the multi-core software solution, our model will associate an extreme high cost to this type of transitions. This point has been described in the **2.3.1.1.2**.

3) Class 1 series 2 & series 3: the overhead for IOC is decreasing when data loss rate or data duplication rate is augmented.

4) Other classes: the restriction of IOC can be relaxed.

- **Physic unit of data**

As detailed in Annex 1, we reanalyzed a quantitative study of the variability of data on the target automotive applications. **The details of the unit are summarized in** Table 1**.** For the data unit for whose values vary dramatically, it is discouraged to manage this kind of transition via IOC, while for those varying rarely over time, managed by IOC will not bring further overhead for communication. However, the variation of some data units is depends on the data, which requires best knowledge of the functional behavior of the applications.

Unit	Count		Designation	Variability (Fast/Slow/Depend on data)
	EBDT	TDP		
Without unit	155	49	Without unit	Depend on data
kW	1		Power	Slow
g/mol	1	1		Slow
1/s	2	2		Fast
kg/s	29	49		Fast
RPM/s	4		Revolutions per minute	Fast
kg	24	31	Mass	Fast
s.kg/Pa	1	1		Fast
m²	14	6	Surface	Depend on data
kg/s/Pa	1	1		Depend on data
Pa	77	114	Pressure	Depend on data
N.m	142		Moment	Fast
%	31		Percentage	Depend on data
.	218	26		
(K)^1/2	1	3		
-	305	104		
m/s2	1		Acceleration	Fast
°Vil	2			Fast
mOhm	1		Resistance	Slow
°	1			Depend on data
K^(1/2)	1	1		Depend on data
1/Pa	4	6		Depend on data

km/h	6		Speed	Slow
A	7		Current	Depend on data
s.u.	1			
J	1		Inertia	Slow
K	43	61		Depend on data
Nm	10		Moment	Fast
W	3		Puissance	Fast
V	16		Tension	Slow
mg	3		Mass	Fast
_	4			
m^2	3	3	Surface	Depend on data
°C	9		Temperature	Depend on data
RPM.N.m/s	1			Depend on data
s/m.K^(1/2)	1	1		Depend on data
km/h/1000RPM	1			Slow
°/s	1			Depend on data
Pa/s	1	1		Depend on data
m	1		Distance	Fast
V/s	8			Slow
s	45	11	Temps	Fast
m/s^2	23			Depend on data
°Ck	17	14		Fast
RPM	22	4	Tours	Fast
m/s^3	1			Depend on data
bool	6			Slow
N.m/s	6			Depend on data
kg/h	2			Slow

Table 1-Physical unit of data

- **Data size**

  Data size is the size of data transferred between transitions. For example: for an irv data prototype with type of SInt16, its size is 2 byte.

- **Data rate**

  This is the total size of data transferred between runnables in a transition in a unit time. The high data rate causes high overhead for IOC. In order to facilitate the allocation according to varying elements like data rate, load of CPU, the knowledge

of the period for each runnable is mandatory. Follows are the rules for the association of the period for each runnable.

A. For the runnables activated by periodic RTEEvent – timing event, they are periodic and the period is determined by its related timing event.

B. For the runnables activated by other RTEEvent, for example, the runnables existing in class 2, class3 and class4, they are not periodic. The determination for these runnables depends on different assumptions:

Assumption_1 For a runnable activated by non-periodic event, for example by data receive event, and if it receives the variable only by another producer runnable activated by periodic event (Figure 29), its period is equal to the period of this producer runnable.

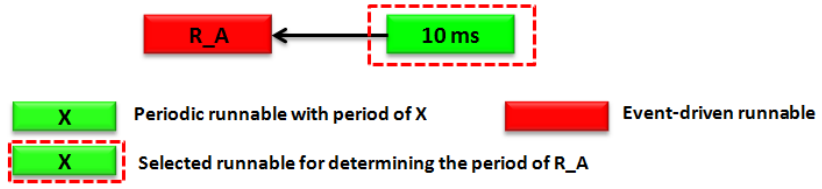


Figure 29- Determination period for non-periodic runnable R_A (assumption1)

Assumption_2 If it receives the variable only by another producer runnable activated by non-period event, its period is equal to the period of that non-periodic runnable. This implies the research for an event chains (in Figure 30) to find the runnables in the assumption 1.

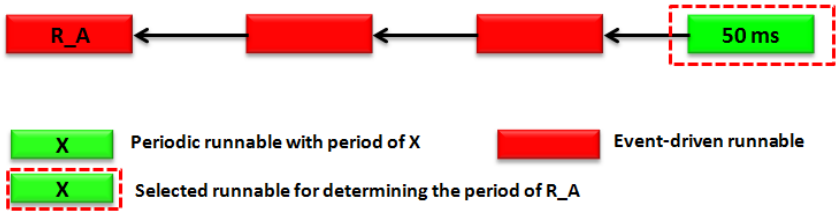


Figure 30-Determination period for non-periodic runnable R_A (assumption 2)

Assumption_3 If it receives simultaneously by several runnables, its period is equal to the minimum period of these received runnables (in Figure 31).

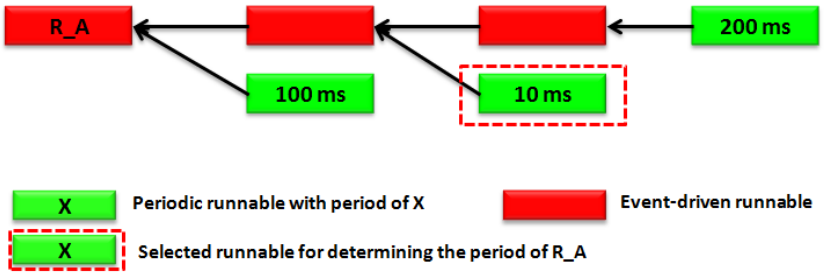


Figure 31-Determination period for non-periodic runnable R_A (assumption 3)

Assumption 4 For a server runnable, its period is equal to its client runnable.

These elements shall be balanced to determine the cost level for a transition managed by IOC. Besides, the cost of spinlock shall be considered if presented.

2.4 Related works in automotive domain

The theoretical formulation of application partitioning has been widely studied in the past either in the domain of multiprocessor computing (Yi, Han, Zhao, Erdogan, & Arslan, 2009) or in hardware/software co-design (Miramond & Delosme, Design space exploration for dynamically reconfigurable architectures, 2005). But the proposed partitioning methods rapidly faced a major limitation considering the lack of real use cases integrated in a full industrial working process. The explored solutions at high-level were too abstract to be really considered. Moreover, when considered alone, the formal optimization clears out the designer from the problem and neglects that not all the design considerations can be theoretically formulated.

In recent years, the adoption of multicore architectures in critical embedded systems has revived the need of design flows fully integrating the exploration phase. So, several works have dealt with the partitioning problem of IMA applications for the avionic domain as well as AUTOSAR applications for automotive domain onto multi-core systems. So, in (Monot, Navet, & Bavoux, 2012) authors developed heuristic algorithms for mapping runnables into different cores. In this paper, runnables are grouped into clusters before being distributed across cores by optimizing a specific objective function. The works of Faragardi et. al (Faragardi, Lisper, & Nolte, 2013) and Saidi et. al (Saidi, Cotard, Chaaban, & Marteil, 2015) proposed a heuristic algorithm to create a task set according to the mapping of runnables on the cores. With the goal of minimizing the communications between runnables, the problem is classically formulated as an Integer Linear Programming (ILP). Therefore, conventional ILP solvers can be easily applied to derive a solution. In (Sailer, et al., 2013), Genetic Algorithms (GA) are applied to partition the application in an optimal way. The results of task allocation are evaluated by their simulation tool TA-Toolsuit that is a vendor tool developed by enterprise Timing Architect. Similar tools are developed by other development companies such as SymtaVision. This kind of vendor tools proposes validation at simulation level. Based on an allocation solution given by user, the tools simulate a set of tasks deployed onto multi-core architecture, which check a set of constraints (e.g.: real time constraints, contention between the shared resources, etc.). The designer can analyze the Gantt diagram of the scheduling and the corresponding data dependencies or summarize the overhead of the communication load. The simulation is close to the real Hardware model and the analysis results could help for the (slight) modification of the allocation (proposed by the tool this time). However, the allocation choice might still not be optimal, and the automatic exploration of the allocation step from these tools is very limited.

There are several European projects that work on the multi-core systems such as AMALTHEA (AMALTHEA, 2012), parMerasa (parMERASA, 2011) and EMC² (EMC², 2014). The AMALTHEA project proposes the methods that evaluate the allocation solutions using a set of **cost functions**. Applying a metaheuristic algorithm such as genetic algorithms, the tool explores the solution space to find the best solution that satisfies the condition of the input cost functions. However: The definition of the cost functions from the tool is mainly based on the real-time criteria: the metric for quantifying the deadline compliance on system level, resource consumption and data-communication overhead. The aspect of sequences of execution such as event chains and execution orders is not considered by the tool (Sailer, et al., 2013). Moreover, they do not propose approaches for scheduling.

The project parMerasa takes the dependency of components into account. However: The overhead of communication between cores is not considered. The task configuration maintains the same from single-core architecture, which does not benefit enough from the multi-core (Panic, et al., 2014). Take the application presented in (Panic, et al., 2014) as an example, the dependencies between the runnables are shown in Figure 32(a). In single-core case, it clusters the runnables with the same period into same task as shown in Figure 32(b) and one feasible scheduling is shown in Figure 32(c) that respects the dependencies exigency (tasks with period of 1 ms shall be executed before other two tasks). According to their approach, when migrate to multi-core, only the runnables in the same tasks can be parallelized to different cores in order to maintain the same sequence of task execution as to the single core. Therefore, Figure 32(d) shows a result when migrate the example application into a 2-core system, where we can see the execution order of tasks is maintained, i.e. task with period of 1ms execute before tasks with 4 ms and tasks with 5 ms. Their approach does not need additional validation stage as it keeps the original configuration such that the development cost is not increased for multi-core platform. However, this approach can introduce large idle intervals due to a long critical path inside a task (Kehr, et al., 2016), for example, the R_idle in (d). Moreover, the communication overhead is not considered in their approach, for example, the communication between R2 and R5.

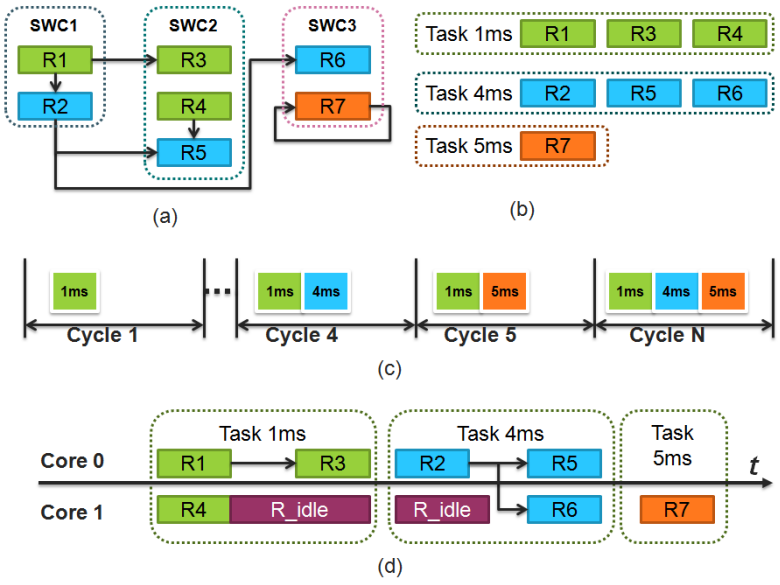


Figure 32-Approach proposed by parMerasa

Conclusion

The problem of distribution into multi-core system has been launched in the automotive industry with the explosion of the standard functionalities as well as ADAS. In spite of the existence of the solutions in the literature for solving this optimization problem, it has not yet existed the satisfactory solutions that are adapted to the automotive context as well as the AUTOSAR standard.

Compared to the existing solutions in the literatures, the method proposed in the following chapters will bring the following contributions:

- Consider the sequences of event chains in Chapter 3 by minimizing the global jitter during the scheduling.
- The proposed scheduling approach works both on single-core and multi-core platform and is compliant with AUTOSAR applications.
- Integrate the multi-core distributions in a validation loop based on hardware concerns (Hardware in the Loop) in Chapter 4.

Chapter 3 Real-Time System scheduling

3.1 Real-time System scheduling overview

A classification of scheduling algorithms is presented in (Cheng, Stankovic, & Ramamritham, 1989). Before introducing some typical algorithms both in single-core and multi-core system, we first introduce some basic notions and notations.

3.1.1 Basic notations

The real-time system is composed of a set of real-time tasks, each task is made up of a set of execution entities called jobs, which execute sequentially on processors and respect the temporal constraints. Based on the way of the jobs recurring over a period of time, the real-time tasks could be classified into 3 categories:

- **Periodic tasks**: the time interval of the activation time between two jobs is fixed. This fixed time interval is called period.
- **Sporadic tasks**: the time interval of the activation time between two jobs is not fixed, but it exist the minimum interval.
- **Aperiodic tasks**: similar to the case of sporadic tasks, but there is no minimum value of the time interval between two jobs.

The model of the periodic tasks proposed by Liu and Layland of (Liu & Layland, 1973) is the most widely used in the real-time systems modeling as shown in Figure 33. The parameters for a periodic task $\tau_i = (r_i^0, C_i, D_i, T_i)$ are:

r_i^0 (release time) : the first activation time of the task *τi*. If the first activation time of all the tasks is given, then these tasks are concrete tasks. Otherwise they are non-concrete tasks.

C_i (Computing time): The execution time for task *τi*. This parameter is considered usually as the worst case execution time in the target processor.

D_i (Deadline): the relative deadline to each activation of the task.

T_i (Period): the time interval between the adjective jobs of the task.

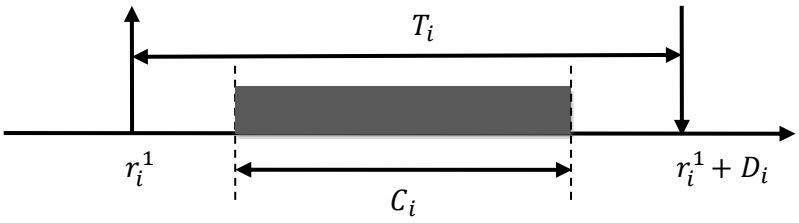


Figure 33-Task model

The constraints on deadlines of tasks are classified into three categories:

- **Implicit deadlines**: all task deadlines are equal to their periods. $D_i = T_i$,
- **Constrained deadlines**: all task deadlines are less than or equal to their periods. $D_i \leq T_i$,

- **Arbitrary deadlines**: no relation constraints exist between the deadlines and periods for all the tasks.

Although the principal scheduling issue tackled in real-time system is the respect of the deadline of tasks, there are some other metrics that are worth to be taken into consideration for scheduling analysis as shown in Figure 34.

Job τ_i^n denotes the n^{th} instance of task τ_i.

Release time of job τ_i^n is denoted as r_i^n**:** so we have $r_i^n = r_i^0 + n \times T_i$.

Relative deadline of job τ_i^n is denoted as d_i^n**:** so we have $d_i^n = r_i^n + D_i$.

Start time of a job τ_i^n is denoted as s_i^n**:** the time on which the job begins for the first time to execute the resources. The start time of the first job in Figure 34 is 2.

End time of a job τ_i^n is denoted as f_i^n: the time on which the job terminates its execution. The end time of the second job in Figure 34 is 12.
Response time of a job τ_i^n is denoted as R_i^n: it corresponds to the difference between the release time of a job and its end date.
Response time of a task τ_i is denoted as R_i: it corresponds to the maximal response time of job among all the jobs of this task.
Instantaneous latency of a job τ_i^n is denoted as $\mathcal{L}_i^n$: the difference between the considered instant and the deadline of the job.
Instantaneous laxity of a job τ_i^n is denoted as l_i^n: the difference between the latency and the rested time for terminating the execution of the job.
Entry/exit delay of a job: the difference between the job start time and the job end time.
Jitter of a job τ_i^n is denoted as δ_i^n**:** it represents the delay between the release time and start time for job τ_i^n.

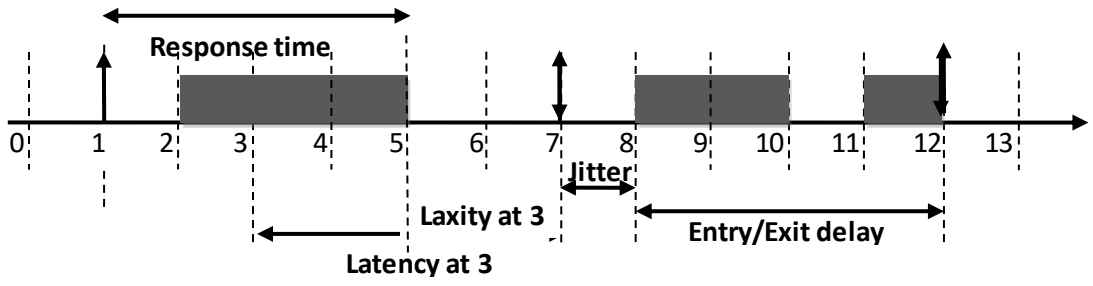


Figure 34-The scheduling of task (1, 3, 6, 6)

3.1.2 Real-Time Scheduling algorithms overview

3.1.2.1 Scheduling in single-core system

This section presents some scheduling algorithms in single-core real-time system. For the partitioned scheduling in multi-core system, each core uses the single-core scheduling, which will be presented in the following section. If each task is attached with a single fixed priority and this priority is applied to all the jobs of this task, this priority is called fixed task priority. The scheduling algorithms Rate Monotonic (RM) (Liu & Layland, 1973) and Deadline Monotonic (DM) (Leung & Whitehead, 1982) are the examples for fixed task priority. If the fixed priorities attached to each job of task are not the same, this is termed as fixed job priority. The example scheduling algorithms in this case is Earliest Deadline First (EDF) (Liu & Layland, 1973). If the priorities attached to each job of the task may change at different times, it is termed as dynamic priority. The example for this is Least Laxity First (LLF) (Mok, 1983) scheduling.

3.1.2.1.1 Fixed Task Priority Scheduling

The priority fixe to tasks is the priority that does not vary during the execution of the tasks.

3.1.2.1.1.1 Rate Monotonic (RM)

The scheduling algorithm RM is proposed by Liu and Layland in 1973. RM is very commonly used for scheduling real-time tasks in practical applications. Basic support is available in almost all commercial RTOS for developing applications using RM. According to RM, the static priorities are assigned on the basis of the period of the task: the shorter the period is, the higher is the task's priority. The RM is optimal under the preemptive scheduling model and implicit deadlines constraints. A sufficient test of schedulability for n tasks is given by:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

3.1.2.1.1.2 Deadline Monotonic (DM)

The scheduling algorithm DM is proposed by Leung and Whitehead in 1982. According to DM, the static priorities are assigned on the basis of the relative deadline: the shorter the deadline is, the higher is the task's priority. Unlike RM that no longer remains an optimal scheduling algorithm for constraint deadline constraint, the DM is optimal under the preemptive scheduling model and constraint deadlines constraints. A sufficient test of schedulability for n tasks $\boldsymbol{\Gamma_n}$ is given by:

$$\forall \tau_i \in \Gamma_n, C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{D_i}{T_j} \right\rceil . C_i \leq D_i$$

where $hp(\tau_i)$ is the set of tasks in Γ_n with the priorities no smaller than τ_i .

3.1.2.1.1.2.A Analysis of response time

The response time of tasks could be used to analyze the schedulability. A sufficient and necessary condition of schedulability is proposed, which is based on the calculation of the worst case of response time. For a set of tasks Γ_n, the worst case of response time R_i for task $\tau_i \in \Gamma_n$ is given by:

$$R_i = C_i + \sum_{j \in hp(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_i$$

with $hp(\tau_i)$ is the set of tasks with higher priority than τ_i in the taskset Γ_n. The taskset Γ_n with fixed priority is schedulable if and only if:

$$\forall \tau_i \in \Gamma_n, R_i \leq D_i$$

3.1.2.1.2 Job-Fix Priority Scheduling

The job-fix priority means that the priority of each job remains the same at its execution. The most studied job-fix priority scheduling algorithm is Earliest Deadline First (EDF), which is proposed by Liu and Layland in 1973. EDF assigns the priority to each task according to their absolute deadline. The highest priority at the time $\boldsymbol{t}$ is assigned to the task with the absolute deadline that is most closed to $\boldsymbol{t}$. EDF is optimal for the independent tasks in the single-core system. One exact test of schedulability for n tasks $\boldsymbol{\Gamma_n}$ with implicit deadlines constraints is given by:

$$\forall \tau_i \in \Gamma_n, \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

However, this condition is only sufficient for the constrained-deadline sporadic task sets.

There are certain advantages for EDF compare to FP (Fixed-Priority). EDF can schedule all the tasks that can be scheduled by FP, while conversely, it is not the case. In spite of the extra computation for the absolute deadline, EDF introduces less context switches. Therefore, EDF performances are better for the overhead of runtime (Buttazzo, 2003). EDF allows full utilization the processor, while FP performs more idle time. Taking RM as an example, when the number of tasks $n \to \infty$, the maximal system utilization is $\ln 2 \approx 0.69$.

However, the disadvantages of EDF are not negligible, which leads to the reason that it is not applicable in the commercial RTOS. EDF is less predictable and controllable: the response time of tasks in FP is always constant and we can always minimize the response time (if it is possible) by increasing its priority. While in EDF, they are variable, and the priority is not reconfigurable. EDF requires more overhead for implementation. For example, it might require a long data structure to deal with the absolute deadline for each job of all the tasks. In the case of non schedulable, i.e. the response time of the task exceed its predicted execution time, PF is more predictable as only lower priority tasks miss their

deadlines. While EDF generates domino effect (Liu & Layland, 1973): all the tasks missed their deadline almost at the same time.

3.1.2.1.3 Dynamic priority Scheduling

The dynamic priority varies during the execution of an instance. The most utilized scheduling algorithm of dynamic priority is Least Laxity First (LLF) algorithm, which is proposed by Mok in 1983 (Mok, 1983) . LLF is based on the laxity of each job (instance), i.e. the task with the smallest laxity is the highest priority to be scheduled in the runtime. The condition of schedulability of LLF is the same to that of EDF (COTTET, 2000), which means one exact test of schedulability for n tasks $\boldsymbol{\Gamma_n}$ with implicit deadlines constraints is given by:

$$\forall \tau_i \in \Gamma_n, \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

LLF scheduling is optimal both on single-core system and multi-core system in the condition that the release times of all tasks are identical (Herrtwich, 1990), which means LLF scheduling is not optimal if the ready times (release time) are not the same for all tasks. Although LLF scheduling may yield better results in terms of schedulability, it is less efficient than EDF scheduling such as the larger quantities of process switches and requirement of several reevaluations of the scheduling criterion.

3.1.2.2 Scheduling for multi-core systems

The scheduling algorithm for multi-core systems determinates i) for each task in the system the processor it will execute at and ii) for each processor the execution date and order of the tasks. The first part is also considered as allocation problem that was studied in the previous chapter. In this chapter we focus on the second part: the problem of scheduling in multi-core systems with a limited number of processors (for a system with non-fixed number of processors, there will not be any problem of scheduling as we can add as much number of processor to make sure the tasks are schedulable: each task in a processor in an extreme example). The scheduling problem for multicore-systems is formulated in the first time by Liu in 1969 (Liu & Layland, 1973). It does not exist an optimal scheduling algorithm with a polynomial complexity. De-facto, the majority of scheduling problem in multi-core systems is NP-complete. Besides, the solutions for scheduling problems in multi-core system are certainly not the trivial extensions of the solutions from single-core system.

Multicore system scheduling can be classified according to the degree of migration allowed such as:

a. **No migration:** Each task is allocated to one processor and cannot be migrated to other processors in the runtime.
b. **Task-level migration:** Also called as ***restricted migration*** where the jobs (instances) of a task are allowed to be allocated to different cores. But the migration of job during the runtime is forbidden.

c. **Job-level migration:** Also called as ***full migration*** where no restriction for the migration, each job of each task can be migrated to different processors.

For each degree of migration allowed, similar to the case of single-core system, the scheduling can be further categorized according to the priority attachment such as ***task-fix priority***, ***job-fix priority*** and ***dynamic priority***. By combining the dimension of migration and priority, there are 9 classes of scheduling algorithms. Carpenter et al. summarized the bound of utilization for each class in (Carpenter, et al., 2004) as shown in Table 2, where only full migration with dynamic priorities can promise a full utilization of system (utilization $\boldsymbol{U}$ is equal to the number of cores $\boldsymbol{m}$).

	No migration	Restricted migration	Full migration
Task-fix Priorities	$(\sqrt{2}-1)m \le U \le \frac{m+1}{1+2^{\frac{1}{m+1}}}$	$U \le \frac{m+1}{2}$	$U = \frac{m+1}{2}$
Job-fix Priorities	$U = \frac{m+1}{2}$	$m-\alpha(m-1) \le U \le \frac{m+1}{2}$	$m-\alpha(m-1) \le U \le \frac{m+1}{2}$
Dynamic Priorities	$\frac{m^2}{3m-2} \le U \le \frac{m+1}{2}$	$\frac{m^2}{2m-1} \le U \le \frac{m+1}{2}$	$U = m$

Table 2-Known bounds on worst-case achievable utilization (denoted U) for the different classes of scheduling algorithms (Carpenter, et al., 2004)

Scheduling algorithms where no migration is permitted are referred as partitioned scheduling, while the migration allowed scheduling algorithms are referred as global scheduling. The majority of research for global scheduling focuses on the job-level migration.

It is worth to mention that it is neither comparable nor opposed between partitioned and global scheduling in the majority cases. In the example of (Leung & Whitehead, 1982) for the periodic taskset, the scheduling is accomplished by a partitioned based algorithm, while no global approach can be found to realize a feasible scheduling. On the other hand, there are also systems of periodic taskset that can only be scheduled by global scheduling algorithm.

In addition to partitioned scheduling and global scheduling, there are also some hybrid approaches that contribute to combine the advantages of partitioned and global scheduling. The examples for hybrid approaches contain semi-partitioned scheduling and clustering scheduling. We introduce these approaches briefly in the following parts of the section.

3.1.2.2.1 Partitioned scheduling

The strategy of partitioned scheduling (Andersson & Jonsson, 2000) (Baker & Baruah, 2006) consists in portioning a set of n tasks Γ_n into m disjointed subset $\Gamma^1, \Gamma^2, \dots, \Gamma^m$ such that $\bigcup_{i=1}^{m} \Gamma^i = \Gamma_n$ and m is the processors' count. Then in each processor π_j, a single-core scheduling approach can be performed for the subset Γ^i that is allocated to it. Therefore, there is a scheduler for each processor. The tasks allocated to each processor are forbidden to migrate to other processor during the runtime; neither can preemption result in migration.

The partitioned scheduling has the advantage of dividing the scheduling of multi-processor into several scheduling of single-core where it exist already a lot of works in the literature, and it is the multiprocessor real-time scheduling approach the most commonly used in practice. The partitioning of the tasks to different processors is equivalent to the "Bin-Packing" problem that has been studied by a lot of works since 1970. The problem of bin packing is a classic intractable problem to solve which is NP-hard in the strong sense (Garey & Johnson, 1990) (E. G. Coffman, Garey, & Johnson, 1996). Therefore, there is no polynomial or pseudo-polynomial algorithm to solve this problem by finding the minimum processors for allocation of tasks.

Bin-packing problem consists of packing a set of n items $\mathrm{B}_n = (b_1, \ldots, b_n)$, each with a size $s(b_i) \in (0,1]$, into a minimum number of unit-capacity bins. To do that, the items are partitioned into a minimum number of m subsets $\mathrm{B}^1, \mathrm{B}^2, \ldots, \mathrm{B}^m$ such that $\sum_{b_i \in B_j} s(b_i) \leq 1, 1 \leq j \leq m$.

Despite of the intrinsic complexity of bin-packing problem, the optimal solution is not always required. This is quite common in the industrial case, i.e. the solutions that meet the schedulability requirement in each core are considered as good solutions.

Most of the heuristics of allocation are the greedy algorithms that are based on two steps:

Step 1 - Sort: sort the list of tasks by certain orders. A principal rule in this step is sorting the tasks by the decreasing of their utilizations.

Step 2 - Distribution: place the tasks one by one with the order defined in step 1 to the processors. The target processors are chosen by their features of utilization. Several rules for placing the tasks are proposed to distribute the tasks to different cores. Considering the current task τ_i, the typical rules (E. G. Coffman, Garey, & Johnson, 1996) for allocating it to a set of processors that are sorted by increase order of their indexes are:

- First-Fit (FF): allocate the task τ_i to the first processor that can contain it.
- Best-Fit (BF): allocate the task τ_i to the processor with the greatest feature of utilization that can contain it.
- Worst-Fit (WF): allocate the task τ_i to the processor with the smallest feature of utilization that can contain it.
- Almost Worst-Fit (AWF): similar to WF but the processor with the second smallest feature of utilization is chosen.
- Next-Fit (NF): the latest processor that is chosen to receive the task is defined as current processor. The task τ_i is allocated to the current processor. If the current processor is not available to receive it, next processor is chosen as the current processor.

More details for these heuristics can be referred in (E. G. Coffman, Garey, & Johnson, 1996).

After the allocation step, each processor can perform single-core scheduling policy as described in the earlier part of this chapter.

In the literature, the study of the partitioned scheduling approach in real-time systems is started by Dhal and Liu in 1978 (Dhall & Liu, 1978). The authors in this paper propose 2 scheduling algorithms RM-NF (Rate Monotonic Next-Fit) and RM-FF (Rate Monotonic First-Fit) for a set of periodic and independent tasks. The tasks are preemptive and the priorities are attached in the task level. According to their approaches, the allocation of the tasks are achieved by next-fit and first-fit heuristic respectively, to do that, the tasks are sorted by the increasing order of the period of the tasks before the execution. Then on each processor, the rate monotonic (RM) policy is performed to schedule the tasks allocated to each processor. These two approaches are then improved by rectifying an error on the evaluation of performances in (Oh & Son, 1993). In this paper, a scheduling method that based on RM-BF (Rate Monotonic Best-Fit) is presented. Another task-fix priority scheduling approach is proposed in (Davari & Dhall, 1986), where a RM First-Fit with decreasing utilization order factor is adopted.

As for the scheduling approaches for job-fix priority taskset, Lopez et al. in (López J. M., García, Díaz, & García, 2003) presented scheduling algorithms EDF-NF, EDF-FF and EDF-WF (worst-fit) for the preemptive taskset. And in (López, García, Díaz, & García, 2000), the authors adopted EDF-FF and EDF-BF as the scheduling approach for a set of preemptive tasks. Besides, in (P. Regnier, 2011) algorithm RUN that is optimal multiprocessor Real-time Scheduling via Reduction to Uniprocessor is proposed, which transforms the multiprocessor scheduling problem into an equivalent set of uniprocessor problems. RUN allows reducing the preemptions bound as well as showing a well scalability when the number of tasks and processors increase.

3.1.2.2.2 Global scheduling

The global approach applies the scheduling algorithm in a global way for the multi-core architectures. Conceptually, all processors share a single ready queue that contains all the tasks in the systems. The tasks are priority-driven. I.e.in each iteration, m tasks with highest priorities in the queue are selected to be executed in the m-core multi-core architecture, which can be achieved for example by RM or EDF. Global scheduling allows the migration of the tasks. A task that is preempted by another task with higher priority can resume in another processor. For the implicit-deadline tasks, there exist optimal global schedulers, whereas no optimal schedulers exist for constrained-deadline and arbitrary-deadline tasks (Fisher, Goossens, & Baruah, 2010). Obviously, the global scheduling eliminates the need to resolve the tasks allocations issues, which is the source of capacity loss in the partitioned scheduling. Therefore, global schedulings performs better for the utilization of the processors. However, the inconvenient of global scheduling is not negligible, including the overhead introduced by migration and ***Dhall effect*** (Dhall & Liu, 1978).

Dhall effect was introduced in (Dhall & Liu, 1978) that considers global scheduling of a set of periodic tasks with implicit deadline on m-core systems. The taskset contains one long period task with utilization of 1 and m short period tasks with infinitesimal utilization such

that the total utilization of the system is $1 + \varepsilon$. The global scheduling based on EDF or FP is not feasible for this taskset as the long period task will always be scheduled lastly and hence exceed its deadline. This so-called *Dhall effect* can be solved by partitioned scheduling, which led to a perceived superiority of partitioned approach. That is why a majority of research on real-time scheduling for multi-core system focused on partitioned approach during a period. This superiority was eventually disproved by (Leung & Whitehead, 1982) (Baruah S. , 2007), that the partitioned scheduling and global scheduling are incomparable both for fixed task priority and fixed job priority systems.

There exist a lot of works for global scheduling that are based on task-fix priorities or job-fix priorities systems. For task-fix priorities cases (gFP), (Andersson, Baruah, & Jonsson, 2001) presented algorithms based on RM and in (Baker T. P., 2003) the algorithm based on DM was proposed. As to the job-fix priorities cases, there are a lot of works that based on EDF approaches (gEDF) (Goossens, Funk, & Baruah, 2003) (Baruah S. K., 2004) (Danne & Platzner, 2006). Authors in (Srinivasan & Baruah, 2002) presented a scheduling algorithm that modified the traditional EDF approach by adopting a condition: if $u_i \leq {}^{m}/_{(2m-1)}$, where u_i is the utilization for task τ_i and m is the number of processor, then the approach applies EDF policy; otherwise if $u_i > {}^{m}/_{(2m-1)}$, the jobs in task τ_i are attached with the highest priority.

The *Proportionate Fair* (PFair) (Andersson & Jonsson, 2003) (Anderson, 2000) is a global scheduling algorithm with a different conception that based on the notion of fairness of proportion, where each task makes progress proportional to its utilization. To do that, each task is divided into sections, and each section executes in the interval called window with the identical size. The PFair algorithm is the only known algorithm that is optimal for the multi-core scheduling. However, only theoretical results can be found in the literature. This algorithm has not yet been applied in the real-life user case. De-facto, at each boundary of the window, the algorithm has to make decision for scheduling and the tasks have to be preempted quite often to guarantee the optimality. Therefore, PFair incurs very high overheads in the system (Jung & Park, 2005).

3.1.2.2.3 Hybrid approaches

As mentioned in the previous section, the global scheduling might introduce communication overheads due to the quantities of migration of jobs. Despite the fact that partitioned scheduling can mitigate this kind of overhead incurred from global scheduling, its algorithmic inherent capacity loss limits the maximum utilization to 50%. Here we introduce some hybrid approaches that contribute to combine the advantages of partitioned and global scheduling.

3.1.2.2.3.1 Semi-partitioned scheduling

Semi-partitioned scheduling is obtained by combining the partitioned and global scheduling. According to semi-partitioned approaches, most tasks can be allocated to the processors in the similar way of the partitioned scheduling. However, for some tasks that satisfy some conditions, for example, some tasks cannot be allocated to any processor without exceed

their deadline, they can be shared by more than one processor. Thus migration is allowed for these "special" tasks. However, as the migration is only reserved for the minority tasks, the communication overhead introduced in global scheduling can be minimized in semi-partitioned approaches.

Different types of semi-partitioned scheduling are proposed in the literatures, for example in (Kato & Yamasaki, 2008), at most $m - 1$ tasks (m is the number of processors in the system) are permitted to migrate between the particular processors. The scheduling is based on EDF, but their approach performs better system utilization compared to the traditional EDF-based algorithm. The scheduling is based on DM in (Lakshmanan, Rajkumar, & Lehoczky, 2009), where the task with the highest priority in each processor can be split across more than one processor. There are also scheduling algorithm named U-EDF that extends the main principles of EDF based on the unfairness property. U-EDF is optimal in multi-processor both on periodic tasks (G. Nelissen, 2011) and sporadic task set (Dragomir Milojevic, 2012), which allow benefiting from a substantial reduction in the number of preemptions and tasks migrations.

3.1.2.2.3.2 Clustered scheduling

Clustered approach is another hybrid scheduling that combines the advantages of partitioned and global scheduling. Actually, clustered scheduling can be categorized as a generalization of both partitioned and global scheduling: if there is only one single cluster, it is equivalent to partitioned scheduling; and if the number of cluster is identical to the number of processor, then it yields global scheduling. Since neither partitioned nor global strategies dominate over the other, cluster-based scheduling is a natural direction for research towards achieving improved utilization bounds (Shin, Easwaran, & Lee, 2008).

The simple principle of the clustered scheduling is that the tasks can be scheduling in the way of global approach but cannot cross the boundary of the clusters that they are allocated at on offline partitioning phase in the way of partitioned scheduling. The cluster may contain more than one processor, thus the tasks are scheduled globally between them in a certain way. Clusters are transformed into tasks and are globally scheduled on the multi-core system. More details can be found in (Calandrino, Anderson, & Baumberger, 2007) (Shin, Easwaran, & Lee, 2008).

3.1.3 Real-Time examination

A set of tasks are called to be ***feasible*** if there exists some scheduling algorithms that respect the deadlines of all the jobs generated by the task set.

A scheduling algorithm is called ***optimal*** if it can always find a solution to schedule all the tasks without missing the deadline if these tasks are feasible.

A task is referred to as ***schedulable*** if its worst case response time does not exceed its deadline according to a given scheduling algorithm.

The analysis of schedulability is the test to verify if a set of tasks are schedulable by a scheduling algorithm. The test is ***sufficient*** if it guarantees that all the tasks are deemed

schedulable by this test are in fact schedulable. The test is ***necessary*** if all the tasks are deemed unschedulable by this test are in fact unschedulable. The test is ***exact*** if it is both sufficient and necessary.

3.1.4 Resource sharing

In multicore system, there are several ways to guarantee the data consistency when there are parallel accesses to the shared resources.

Lock-based: The task will be blocked when it tries to access the shared resource that has been already taken by another task. The blocked task can be suspended (i.e., (Rajkumar, 1990)) or keep spinning (i.e., (Gai, Lipari, & Natale, 2001)).

Lock-free: The tasks can access to the shared resource without blocking. At the end of the operation, a check will be performed and the task will re-access if the check shows an inconsistent result.

Wait-free: This mechanism needs multiple copies of the shared resources.

3.2 Dependant tasks scheduling

In this work, we focus on scheduling applications driven by control and data flow (e.g. engine control, brake control etc.). For that type of command and control applications the order in which the individual statements are executed is very important and the proportion of parallel code is often hard to identify. In consequence, the partitioning of automotive applications into multiple cores requires a fine analysis of the dependencies between runnables and tasks. It then needs to ensure that these dependencies are respected by the scheduling policy. It corresponds to a scheduling problem related to both periodic and dependent tasks.

3.2.1 Related works on real-time scheduling of dependent tasks

The theoretical formulation of application scheduling has been widely studied in the past either in the domain of single-core or multi-core computing (Davis & Burns, 2011). Among the proposed methods, optimal policies are particularly interesting because they are able to find a correct ordering when it exists. In multi-core systems, several schedulability tests were also developed in the real-time scheduling theory to easily determine the minimum number of processors needed to schedule a set of applications (Davis & Burns, 2011). But the problem of scheduling periodic and dependent tasks onto multi-core systems is more complex since it needs to express the dependencies according to a particular Model of Computation (MoC) adapted to the execution properties of the application domain. For example, few run-time scheduling solutions exist which support applications modeled using a MoC and provide hard-real-time services. Most of the existing works assume the applications are modeled as Synchronous Data Flow graphs (SDF) (Bekooij, et al., 2005) (Gantel, et al., 2012) and adapted the existing scheduling algorithms (Lee & Messerschmitt, 1987) to the multi-core case. Bekooij et al. presented a dataflow analysis for embedded real-

time multiprocessor systems (Bekooij, et al., 2005) . They analyzed the impact of time division multiplexing on applications modeled as SDF graphs. The work presented in (Bamakhrama & Stefanov, 2011) extended the SDF model to Cyclo-Static Data Flow model (CSDF) in order to support cyclically varying (but predefined) production/consumption rates.

But most of the proposed approaches need an exponentially complex conversion from SDF to HSDF (Homogeneous SDF). In the same way, in real-time scheduling, when the tasks are synchronous it is necessary to reason on the hyper-period and unfold the tasks into a set of jobs, as with classical branch-and-bound based approaches (Xu & Parnas, 1990). For that reason, authors in (Miramond & Cucu-Grosjean, Generation of static tables in embedded memory with dense scheduling, 2010) proposed a dense scheduling technique to reduce the size of static schedule tables in embedded memory. On the other hand, single-processor policies have been adapted to the multiprocessor case, as partitioned scheduling. EDF, for example, which is known to be optimal for scheduling arbitrary task sets on a uniprocessor system has its multiprocessor version: the Partitioned Earliest-Deadline-First (PEDF) algorithm (López J. M., García, Díaz, & García, 2003) . EDF scheduling of data-dependent tasks was also tackled in (Forget & Frédéric Boniol, 2010) by adjusting deadlines and release times to respect the dependencies.

Hence, a lot of theoretical solutions have been already explored in the literature and were, for example, adapted to massively parallel execution architectures (Zhang, Gao, & Qiu, 2015). But such techniques have only been applied to the AUTOSAR context very recently (Kehr, Quiñones, Böddeker, & Schäfer, 2015). In (Sagstetter, et al., 2014), authors propose integration framework for solving large and complex scheduling problem in automotive multi-ECU systems connecting via timing triggered Ethernet. The authors generate local schedules for each independent subsystem by SMT (Satisfiablity Modulo Theories) approach and integrate them into a global schedule. For the case where no feasible solution exists, they present a conflict refinement to adapt individual subsystem. Compared to ILP solver, they show a better performance in terms of runtime when the complexity of the system increases. However, this work does not target on multi-core context.

Authors in (G.Georgia, Stoimenov, Huang, & Thiele, 2013) propose a scheduling policy for mixed-criticality multi-core systems with the consideration of sharing resource. In order to prevent the interference between the tasks of different levels, they proposed policy allowing only the tasks of the same criticality run at the same time. Therefore, the contention among the tasks with different criticality can be delimited such that the CA (Certification Authority) is proved. Their approach optimizes in the same time the task mapping and scheduling, where the design space is explored by SA.

Authors in (Giannopoulou, Stoimenov, Huang, & Thiele, 2014) target a mixed-criticality multi-core system, where they optimize the data allocation in the shared memory such that the access to the memory in parallel from different cores will not delay each other. They pinpoint that the interdependence exist between the task mapping and schedule that are

studied in their work (G.Georgia, Stoimenov, Huang, & Thiele, 2013) and the data mapping and propose an integration of these two steps.

In (Kehr, Quiñones, Böddeker, & Schäfer, 2015) authors introduce a so called timed implicit communication (TIC) for decoupling task communication to allow parallel execution of producer and consumer, while the same data-flow is achieved on all MCEs (Multi-Core ECUs). However, the overhead communication is introduced that may reduce the new opportunity of parallelization.

Our approach differs from the cited works in several points. First of all, it considers periodic dependencies expressed as AUTOSAR transitions which differ from the SDF semantics, as explained in following parts of this section. Secondly, we have separated the assignment and the scheduling problem. So that we search a multi-core feasible scheduling solution for a given software distribution. This separation also relives the problem of the hyper-period complexity. Finally, few of them have been integrated in an industrial software design flow and validated onto real-life applications.

3.2.2 Model of periodic precedence

The tasks can be independent or dependent. If it exist the dependency between two tasks, we call these tasks dependent tasks, otherwise they are independent tasks. The majority of scheduling studies in the lecture are targeted on the independent task model. However, from a practical point of view, results on how to schedule tasks with precedence and mutual exclusion constraints are much more important than the analysis of the independent task model. Normally, the concurrently executing tasks must exchange information and access common data resources to cooperate in the achievement of the overall system objective. The observation of given precedence and mutual exclusion constraints is rather the norm than the exception in multi-core real time system.

In this dissertation, we work on the dependent tasks model. There are two types of dependency between two tasks τi and τj: the dependency of precedence and dependency of data. The dependency of precedence between (τi, τj) imposes that τj cannot execute until the end of execution of τi. τi is called the predecessor of τj, and τj is called successor of τi. The dependency of data indicates that the task τi produces the data that are consumed by τj. This dependency involves also the dependency of precedence. We note $preds(\tau_i)$ and $succs(\tau_i)$ the predecessors and successors of τi, so $\tau_i \in preds(\tau_j)$ and $\tau_j \in succs(\tau_i)$.

The scheduling of the dependent tasks shall consider the constraints of dependency. For the constraints from the aspect of precedent dependency, there are mainly two approaches exiting today: first one is based on the semaphores: a semaphore is allocated to each predecessor of (τi, τj), and the successor τj shall wait the predecessor τi release the semaphore before its execution. The second one is based on the modification of the priorities and the date of the first activation of the task. For the constraints from the aspect of data dependency, addition to the constraints imposed by the precedent dependency, the handling of the data transfer and shared resources has to be taken into consideration.

In the AUTOSAR applications, there are large numbers of transitions that transfer the data in asynchronous way or synchronous way. The dependencies existing in Autosar application belong to the category of data dependency.

Here we present our proposed scheduling approach by considering the dependency constraints in the system. We consider static scheduling as the pre-run-time scheduling is often the only practical means of providing predictability in a complex system, which requires the timing constraint. One of the weaknesses of static scheduling is the assumption of strictly periodic tasks. Although the majority of tasks in real-time applications are periodic, there are still some sporadic requests for services that have hard deadline requirements. To confront this issue, there are 3 methods to increase the flexibility of static scheduling:

- The transformation of sporadic requests into periodic requests (Mok, 1983),
- The introduction of a sporadic server task (Sprunt, Sha, & Lehoczky, 1989),
- The execution of mode changes (Fohler, 1992).

We use the notations defined in (Forget & Frédéric Boniol, 2010) for the precedent dependency. The precedence between two periodic task τ_i and τ_j corresponds to a set of precedence between the instances of the two tasks. For the nth instance of task τ_i and $n'th$ instance of task τ_j, $\tau_i^n \rightarrow \tau_j^{n'}$ denote the precedence from τ_i^n to $\tau_j^{n'}$.

Definition 1: *Instance Precedence*: For any $k \in \mathbb{N}$, let $\mathcal{J}_k$ denote the set of integers of the interval $[0, k[$. Let $lcm(n, n')$ denote the least common multiple of n and n'. For two task τ_i and τ_j, let $p_{i,j} = lcm(T_i, T_j)$, the precedence $\tau_i^n \rightarrow \tau_j^{n'}$ as the following set of task instance precedence is defined as $\tau_i \xrightarrow{M_{i,j}} \tau_j$ where:

$$M_{i,j} \subseteq \left\{(n,n') \middle| \tau_i^n \rightarrow \tau_j^{n'}, (n,n') \in \mathcal{J}_{p_{i,j}/T_i} \times \mathcal{J}_{p_{i,j}/T_j}\right\} \quad (1)$$

Hence, the precedence now appears in a repetitive pattern and we can define the periodic precedence as follows:

Definition 2: Periodic Precedence: The periodic precedence $\tau_i^n \rightarrow \tau_j^{n'}$ is defined as $\tau_i \xrightarrow{M'_{i,j}} \tau_j$ that is based on $\tau_i \xrightarrow{M_{i,j}} \tau_j$ such that:

$$M'_{i,j} = \left\{(n,n') \middle| \exists k \in \mathbb{N}, (m,m') \in M_{i,j}, (n,n') = (m,m') + \left(k\frac{p_{i,j}}{T_i}, k\frac{p_{i,j}}{T_j}\right)\right\} \quad (2)$$

The precedence expresses all the possible communication between instances of task τ_i and τ_j. For example in Figure 35 (a): the $M_{i,j} = \{(0,0)\}$, so according to equation 2, the periodic precedence $\tau_i \xrightarrow{M'_{i,j}} \tau_j$ is $\tau_i^0 \rightarrow \tau_j^0, \tau_i^2 \rightarrow \tau_j^1$, etc. Similarly in Figure 35 (b), the $M_{i,j} =$

$\{(0,0),(1,1),(3,2)\}$, so the periodic precedence $\tau_i \xrightarrow{M'_{i,j}} \tau_j$ is $\tau_i^0 \to \tau_j^0, \tau_i^1 \to \tau_j^1, \tau_i^3 \to \tau_j^2, \tau_i^4 \to \tau_j^3, \tau_i^5 \to \tau_j^4, \tau_i^7 \to \tau_j^5$, etc.

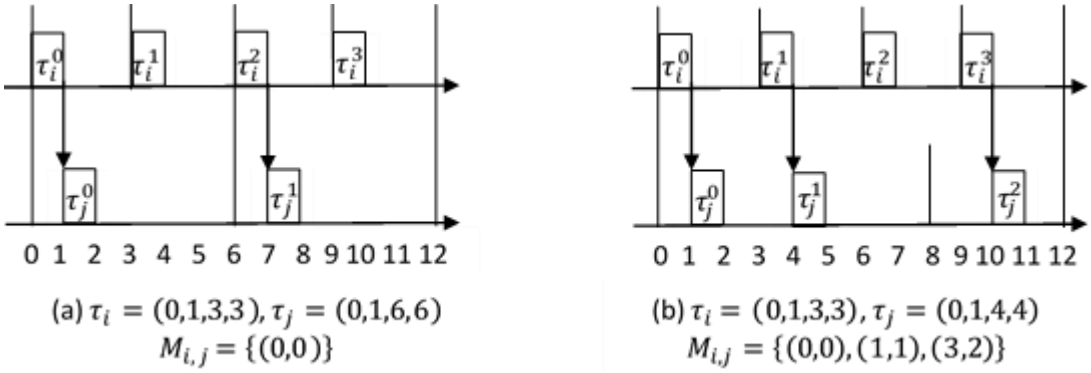


Figure 35-Periodic precedence $\tau_i \xrightarrow{M_{i,j}} \tau_j$

3.2.3 Communication semantics in AUTOSAR: Explicit & Implicit

When the communication involves the writing and reading of the data, e.g. Sender-Receiver communications and IRV communications, the Autosar defines different semantics of communication. Explicit data access (data reception and data transmission) means that when a runnable sends or receives data elements, it immediately access to RTE buffer by using corresponding RTE API. While implicit data access means that a runnable does not actively initiate the reception or transmission of data. More precisely:

- Explicit read. When a runnable reads the data from a buffer, it might receive different copies of the data if there is updating of data in this buffer during the execution of the runnable. Figure 36 shows an example, where Runnable1 reads the buffer (data element) several times during one instance of its execution. When the buffer updates the data, Runnable 1 reads immediately the new version of the data as shown in Figure 36.

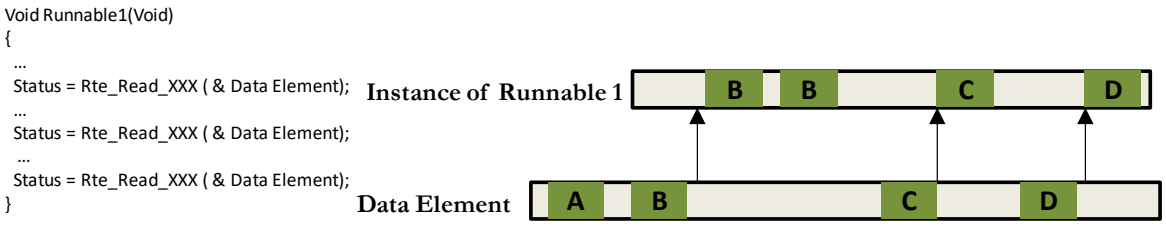


Figure 36-AUTOSAR communication: explicit read

- Implicit read. When a runnable reads the data from a buffer, it gets a stable copy from the buffer when the runnable starts. Several calls inside the runnable always return the same value even if the data in the buffer has been updated by other runnables. The value is therefore stable and data coherency can be ensured. As

shown in Figure 37, the Runnable 1 can always get the same copy of data during its execution.

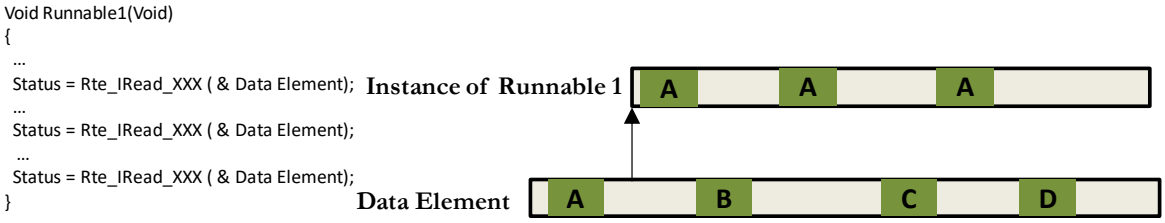


Figure 37-AUTOSAR communication: implicit read

- Explicit write. When a runnable writes on a buffer, the buffer updates the data immediately during the execution of the runnable. In Figure 38, the Runnable 1 executes several writing instruction on buffer Data Element during its execution. Each time it writes, the buffer updates the data immediately.

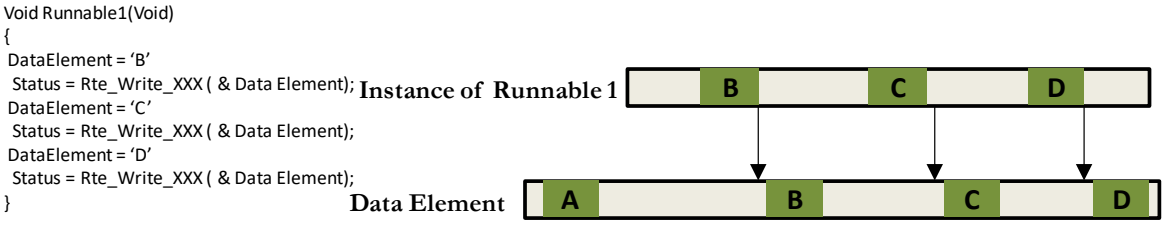


Figure 38-AUTOSAR communication: explicit write

- Implicit write. The data is available only when the runnable that writes on this buffer returns. If several data writing accesses to the same data element on the buffer are performed inside a runnable during the runnable execution, only the last value is sent (also known as last-is-best semantics). So in the Figure 39, the buffer of Data Element only considers the last value written by Runnable 1 during one execution.

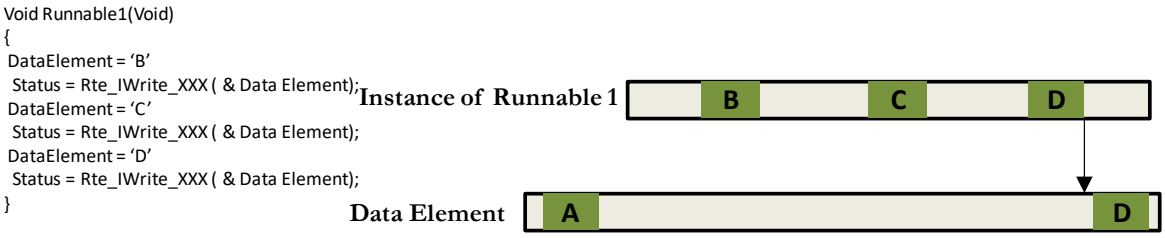
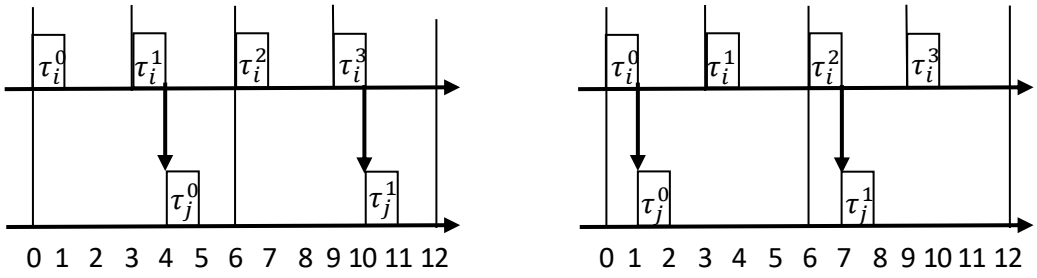


Figure 39-AUTOSAR communication: implicit write

The precedent dependency model defined in section is under the influence of these semantics. For $\tau_i \xrightarrow{M_{i,j}} \tau_j$, the different semantics might result in different value of $M_{i,j}$. For example, we define the period of τ_i is 3 time units and the period of τ_j is 6 time units, in the explicit semantic, in a hyper period, the first instance of τ_j have to be activated after the

last instance of τ_i .While in the implicit semantic, the first instance of τ_j can be activated just after the finish of the first instance of τ_i, as only the first copy of data in the buffer is consider in the implicit read semantic. Figure 40 shows this example.



(a) Explicit communication
$M_{i,j} = \{(1,0)\}$

(b)Implicit communication
$M_{i,j} = \{(0,0)\}$

Figure 40-AUTOSAR communication semantics influence on the dependency model

However, these communication semantics are not sufficient to determine all values of $\mathrm{M_{i,j}}$ for the complete systems. The complementary information to determine all the value of in the entire system could be obtained from high level model of the applications (Klikpo, Khatib, & Munier-Kordon, 2016).

3.2.4 Dependent tasks scheduling in Single-core systems

The first release time of task τ_i is the first activation time of this task, i.e. the reslease time of its fisrt instance. Normally, the first release time is defined by the system of tasks. In the first step in our hypothese, we consider the set of tasks with identic release time with value of 0. The release time for the *nth* instance in system $S = \{\tau_i\}$ can be thereby deduced by equation 3.

$$r_i^n = r_i^0 + n \times T_i, \forall \tau_i \in S \quad (3)$$

The start time of each instance in the schedule table indicates the date when it begins to execute and use the resources. For the dependent tasks, the start time of each instance (job) not only depends on its release time, but also depends on the end time of its precedent jobs in other tasks if there is a precedence constraint between them. Besides, for the jobs that do not have precedence constraint between them, they cannot be executed simultaneously in the same core. As a result, the start time of one job might be delayed by other jobs that have been already activated by the scheduling policy. This delay is constructed from additional delay element γ. The delay element contribute to avoid the overlap between instances independent of each other that are allocated in the same processor

The start time of the $n'th$ instance in task τ_j is given by:

$$s_j^{n'} = \max(r_j^n, max_{(n,n')\in M_{i,j}'}(s_i^n + C_i), \gamma^*) \quad (4)$$

The delay element γ will be updated each time the calculation of starting time of instance is finished in the scheduling to avoid the overlap. So for a non-preemptive system, each time $s_i^{n'}$ is calculated, the γ is updated by:

$$\gamma^* = \max(\gamma, s_i^{n'} + C_i) \quad (5)$$

We now give is an example to illustrate how the method generates schedules by using equation 4 and 5: the system $\boldsymbol{S}$ contains 3 tasks as show in Figure 41, where the precedence is defined as: $M_{i,j} = \{(0,0)\},\ M_{j,k} = \{(0,0)\}$. The hyper period $\mathcal{H}$ for the system is the least common multiple of the period for all the task: $\mathcal{H} = lcm(\forall T_i)$. So for the system in Figure 41 (a), the hyper period is 16. In this hyper period, the instances for each task are: $\{\tau_i^0, \tau_i^1, \tau_i^2, \tau_i^3, \tau_j^0, \tau_k^0, \tau_k^1\}$. Now we illustrate one by one the calculation of start time of these instances. The initial value of γ = 0. For *job* τ_i^0 : as it has no precedence, and its release time is 0, so $s_i^0 = \max(0,0,0) = 0$, and we have to update γ by equation 5 each time the calculation of start time is finished, so $\gamma = \max(0, 0+1) = 1$. Then for *job* τ_j^0: its release time is 0; there is precedence between τ_i^0 and τ_j^0 as $M_{i,j} = \{(0,0)\}$, so by equation 4, we compute start time for τ_j^0, $s_j^0 = \max(0,\ s_i^0 + 1, 1) = 1, \gamma = \max(1, 1+4) = 5$; for other instances we repeat the same steps and the result is illustrated as:

$$s_k^0 = \max(0,5,5) = 5, \qquad \gamma = \max(5, 5+1) = 6$$
$$s_i^1 = \max(4,0,6) = 6, \qquad \gamma = \max(6, 6+1) = 7$$
$$s_i^2 = \max(8,0,7) = 8, \qquad \gamma = \max(8, 8+1) = 9$$
$$s_k^1 = \max(8,5,9) = 9, \qquad \gamma = \max(9, 9+1) = 10$$
$$s_i^3 = \max(12,0,10) = 12, \qquad \gamma = \max(12, 12+1) = 13$$

So the schema for the scheduling is shown in Figure 41 (b):

Obviously, the order of instances considered in equation 4 and equation 5 change the scheduling result. In this example we assume the following order of the instances: $\{\tau_i^0, \tau_j^0, \tau_k^0, \tau_i^1, \tau_i^2, \tau_k^1, \tau_i^3\}$. We explore in section 3.2.4.1 and section 3.3 several ordering policies and their impact on the quality of scheduling.

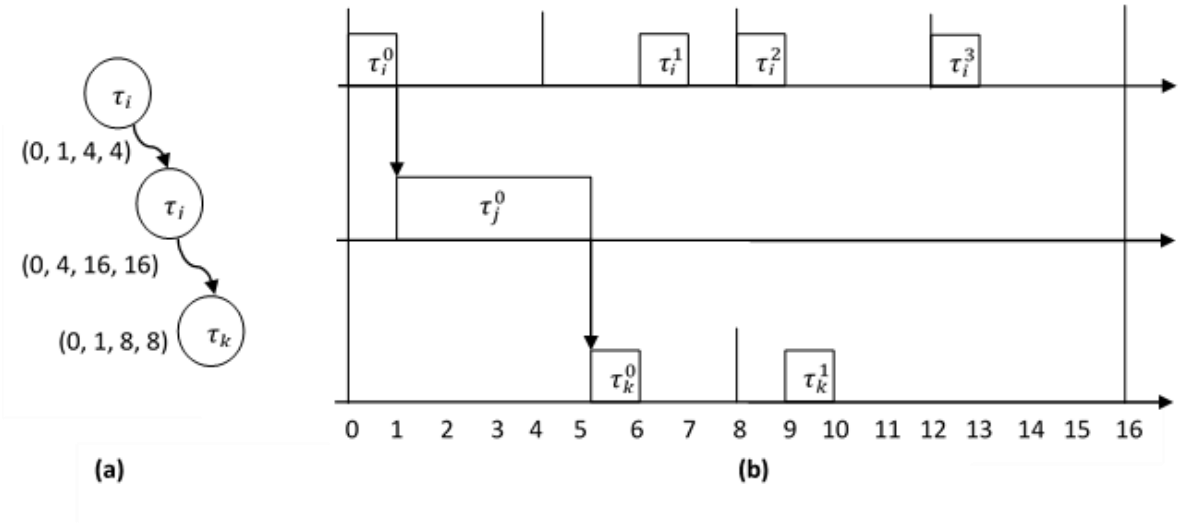


Figure 41-Example of start time

Typically, the tasks allocated in a core are scheduled in a repeated way. For example, if all the tasks are periodic, the scheduling will be repeated in a time interval that equals the least common multiple of the periods of the entire tasks. This time interval is called pattern. The makespan is defined for each pattern with a set of tasks that the time between the start time of the first instance of the first executing task and the last instance of the last executing task. The makespan might vary in different patterns, especially in the case where the sporadic tasks exist. Hence there exist a maximum makespan among all these patterns.

For each pattern with an interval time $\mathrm{P} = [t_a, t_b)$, the makespan for the tasks executed during P is given by the value of delay element γ. For the example shown in Figure 41, the pattern is the hyper period and the makespan is the value of γ, which is 13.

3.2.4.1 Determinate order

The order of calculating the start time for all the instances of the tasks in hyper period $\mathcal{H}$ is not obvious. This is usually caused by the instances that do not have precedent dependency between them. For example, the instance τ_i^3 and τ_k^1 shown in Figure 41 are not depended. The change of their order will influence the makespan as the delay element is updated immediately at the end of each calculation of start time. Therefore, if we calculate the start time of τ_i^3 before τ_k^1, the τ_k^1 will be delayed to the date 13 to start, and as a result the makespan is lengthened to 15. To avoid this kind of indeterminism, we propose 3 dimensions of ordering metric to determine the calculation order for the instances.

3.2.4.1.1 First dimension: Precedent dependency

The precedent dependency is the first ordering metric to determinate the order between instances. For example, for τ_i^n and $\tau_j^{n'}$, if $(n, n') \in M_{i,j}$, then the order between τ_i^n and $\tau_j^{n'}$ in the scheduling tables is $\tau_i^n \rightarrow \tau_j^{n'}$, i.e. s_i^n has to be calculated before $s_j^{n'}$. So for the example in Figure **41**, the order between instances $\tau_i^0, \tau_j^0, \tau_k^0$ is $\tau_i^0 \rightarrow \tau_j^0 \longrightarrow \tau_k^0$.

3.2.4.1.2 Second dimension: Release time

When there is no precedent dependency between instances, the order between them is determined by their release time: the smaller the release time of an instance is, the earlier it will be executed. For instances τ_i^n and $\tau_j^{n'}$ such that $(n,n') \notin M_{i,j}$, if $\tau_i^n < \tau_j^{n'}$, then the order between τ_i^n and $\tau_j^{n'}$ is $\tau_i^n \rightarrow \tau_j^{n'}$. For example in Figure 41, between instance τ_i^3 and τ_k^1, as the release time of τ_k^1 is 8 which is smaller than release time of τ_i^3 (which is 12), so we calculate the start time for τ_k^1 before τ_i^3. Similarly, we calculate the start time of τ_k^0 before τ_i^1. For instances with identical release time, e.g. τ_i^2 and τ_k^1 have the identical release time 8, we adopt third dimension to decide their order.

3.2.4.1.3 Third dimension: Priority

This dimension is ordered by priority of the instances.

In this work we use fixed task priority policy. If each task is attached with a single fixed priority and this priority is applied to all the jobs (instances) of this task, this priority is called fixed task priority. The priority of each task is assigned on the basis of the relative deadline: the shorter the deadline is, the higher is the task's priority. This scheduling algorithms is called Deadline Monotonic (DM) (Leung & Whitehead, 1982) proposed by Leung and Whitehead as presented in the previous section.

When precedence constraints exist, "the relative urgency of a task depends both on its deadline and on the deadlines of its successors" as presented in (Chetto, Silly, & Bouchentouf, 1990). Hence, the deadline of a task can be adjusted as what was proposed in (Chetto, Silly, & Bouchentouf, 1990):

$$D_i^* = \min(D_i, \min(D_j^* - C_j)_{\tau_j \in succs(\tau_i)} \quad (6)$$

The calculation of the adjusting deadline for each task can be done one time at the elaboration of the application graph. The instances that belong to the same runnables have the same adjusting deadline. Based on adjusting deadline, the priority for each task can be attached according to DM. Therefore for instances τ_i^n and $\tau_j^{n'}$ such that $(n,n') \notin M_{i,j} \cap \tau_i^n = \tau_j^{n'}$, if $D_i^* < D_j^*$, then the order between τ_i^n and $\tau_j^{n'}$ is $\tau_i^n \rightarrow \tau_j^{n'}$. So for the example shown in Figure 41, the order between instance τ_i^2 and τ_k^1 is $\tau_i^2 \rightarrow \tau_k^1$, as τ_i^2 has higher priority.

With the three dimensions to determine the instance order, we can calculate one by one the start time of them as shown in Figure 41 (b). But in the case where two instances are identically characterized, i.e. they are independent and possess the same release time and adjusting deadline. Then we firstly execute the instance with shorter execution time in order to minimize the jitters.

3.2.4.2 Schedulability

The analysis of schedulability is the test to verifier if a set of tasks are schedulable by a scheduling algorithm. To do that, each instance in the taskset has to meet their deadline:

$$\forall \tau_i^k : s_i^k + C_i \leq r_i^k + D_i \quad (7)$$

3.2.4.3 Scheduling process

Based on the elements presented before, we introduce our scheduling process for a set of tasks. The process is shown in Figure 42.

Step a: For each task τ in the taskset, we calculate its adjusting deadline D_i^* by the equation 6.

Step b: If there exist a task with a negative adjusting deadline, i.e. $\exists \tau_i : D_i^* \leq 0$, the taskset are not schedulable. Therefore the process is terminated.

Step c: In the interval of hyper period $\mathcal{H} = lcm(\forall T_i)$, all the instances of each task $\{\tau_i^K\} = \left\{\tau_i^k \middle| \forall \tau_i : k \in \frac{\mathcal{H}}{T_i}\right\}$ are constructed into a list $\mathcal{L}$ where their order is determined by the three dimensions presented in Section 3.2.4.1.

Step d: Starting from the first instance in list $\mathcal{L}$, the starting time is calculated by equation 4. Each time the computing of starting time is finished, the delay element is updated by equation 5 immediately such that the overlap between independents instances is avoided.

Step e: Each instance has to be tested by equation 7 to make sure the real-time constraint is respected, aka the deadline compliance.

Step f: If all the instances respect this test then the process can be successfully terminated.

Step g: The output contains a scheduling table for all the instances and the makespan with value of γ.

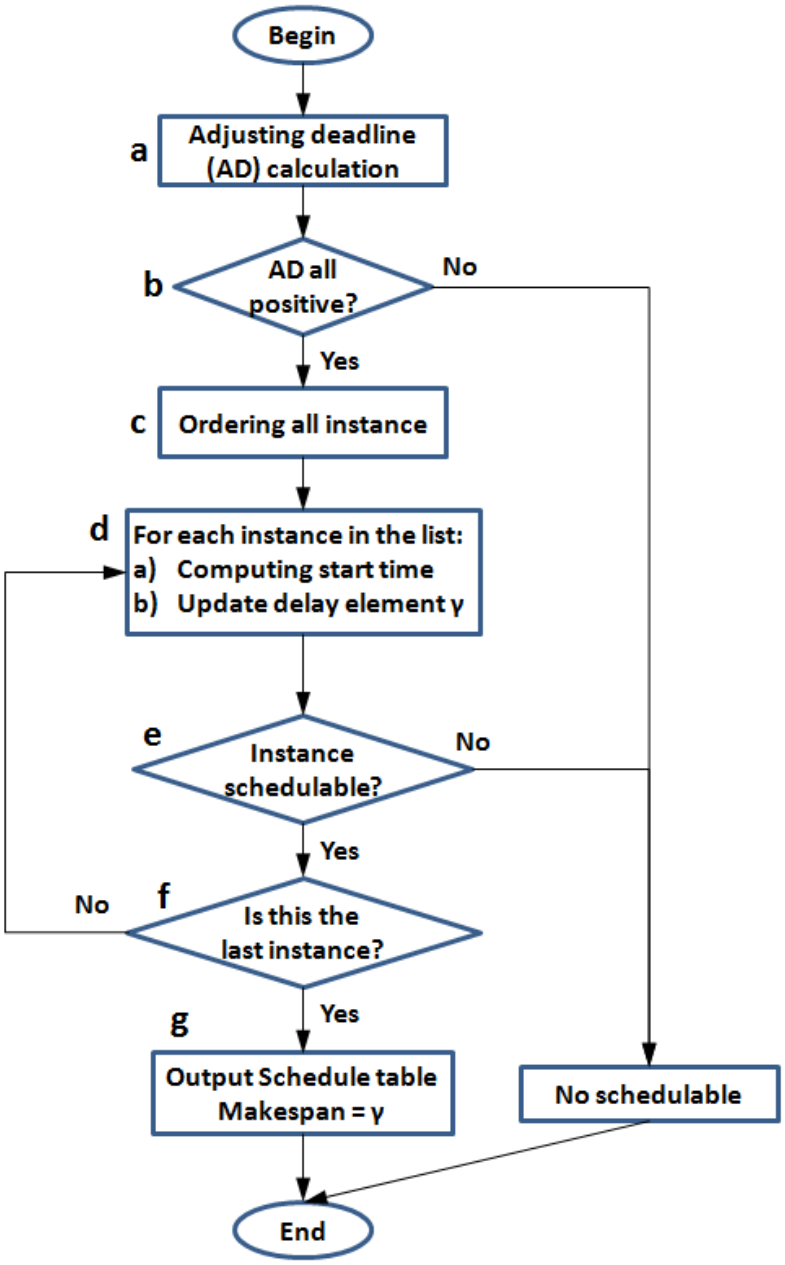


Figure 42-Scheduling process

Example

Here we demonstrate our process with an example shown in Figure 43, which is composed of periodic taskset. The criteria values for each task are present in Table 3. The calculating result of adjusting deadline for each task is also shown in this table. As we can see, there is no negative value for the adjusting deadline. So the next step is to construct a sorted list for the instances in the hyper period.

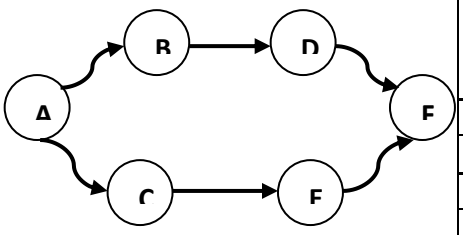


Figure 43-Example application

Task	Execution Time	Period	Deadline	Adjusting Deadline
A	1	6	6	4
B	1	8	8	8
C	1	12	12	5
D	1	12	12	11
E	1	6	6	6
F	1	12	12	12

Table 3-Tasks criteria

Task	Instances in one hyper period
A	$\tau_A^0, \tau_A^1, \tau_A^2 \tau_A^3$
B	$\tau_B^0, \tau_B^1, \tau_B^2$
C	τ_C^0, τ_C^1
D	τ_D^0, τ_D^1
E	$\tau_E^0, \tau_E^1, \tau_E^2 \tau_E^3$
F	τ_F^0, τ_F^1

Table 4-Instances to be considered

Dependency	Enumeration in the hyper period
$M_{A,B} = \{(0,0)\}$	$\tau_A^0 \rightarrow \tau_B^0$
$M_{A,C} = \{(0,0)\}$	$\tau_A^0 \rightarrow \tau_C^0, \tau_A^2 \rightarrow \tau_C^1$
$M_{B,D} = \{(0,0)\}$	$\tau_B^0 \rightarrow \tau_D^0$
$M_{C,E} = \{(0,0)\}$	$\tau_C^0 \rightarrow \tau_E^0, \tau_C^1 \rightarrow \tau_E^2$
$M_{D,F} = \{(0,0)\}$	$\tau_D^0 \rightarrow \tau_F^0, \tau_D^1 \rightarrow \tau_F^1$
$M_{E,F} = \{(0,0)\}$	$\tau_E^0 \rightarrow \tau_F^0, \tau_E^1 \rightarrow \tau_F^1$

Table 5-Dependencies

The hyper period of this example is $\mathcal{H} = lcm(6,8,12) = 24$, so for each task the instances are considered as assumed in Table 4. We defined the dependencies for each transition, which are resumed in Table 5.

Now we sort all the instances shown in Table 4 by considering the three ordering metrics: 1) the dependencies in Table 5; 2) the release time of each instance; and 3) the priority of the task that each instance belongs to (depend on the adjusting deadline shown in Table 3). The sorted result is shown in Table 6, where the order is from left column to the right. The release time for each instance is also shown in the second row of this table.

Ordering Instance	τ_A^0	τ_C^0	τ_E^0	τ_B^0	τ_D^0	τ_F^0	τ_A^1	τ_E^1	τ_B^1	τ_A^2	τ_C^1	τ_E^2	τ_D^1	τ_F^1	τ_B^2	τ_A^3	τ_E^3
Release Time	0	0	0	0	0	0	6	6	8	12	12	12	12	12	16	18	18
Starting Time	0	1	2	3	4	5	6	7	8	12	13	14	15	16	17	18	19
Delay Element	1	2	3	4	5	6	7	8	9	13	14	15	16	17	18	19	20

Table 6-Scheduling result

The next step is to calculate the start time of each instance by equation 4 orderly in Table 6. Each time the calculation is finished for an instance, the delay element γ is updated by equation 5. Besides, for the real-time aspect, the schedulability is tested for each instance by equation 7. In this example, all the instances respect their deadline constraint. The starting time values and delay element values are also shown in Table 6.

Finally, the scheduling result can be visualized in Figure 44.

Figure 44-Generated schedule table

3.2.5 Dependent tasks scheduling in Multi-core systems

The delay element γ has to be updated each time the calculation of starting time of an instance is finished. This is because in single-core system, the instances cannot be executed simultaneously. Therefore in a non-preemptive system, an instance will execute until its termination once it starts. Thus the equation 5 is performed immediately each time the start time of an instance is determined. However, in the multi-core system, as the instances allocated in the different cores can execute simultaneously, the delay element is no longer a global variable. Instead, there is one delay element for each core. The updating for the delay element for each core is similar to the single core system.

3.2.5.1 Extensions for multi-core system

Here we extend some notations introduced in the precedent section to adapt multi-core systems.

The multi-core architecture is composed of a set of cores denoted as $\{\pi_1, \dots, \pi_K\}$. So the task $\tau_i \in \{\tau_1, \tau_2, \dots \tau_I\}$ defined before can be extended as $\tau_{i,k}$, if it is located to core π_k. Similarly, the extension can be done respectively for the execution time $C_{i,k}$ and job $\tau_{i,k}^n$, and its release time $r_{i,k}^n$, start time $s_{i,k}^n$, jitter $\sigma_{i,k}^n$, etc. Each core π_k constructs a delay element γ_k. The release time is similar to equation 3, only with a bit modification on the indices such that:

$$r_{i,k}^{n} = r_{i,k}^{0} + n \times T_i \tag{8}$$

As to the start time of an instance $s_{j,k}^{n'}$, the calculation is done by equation 9 instead of equation 4:

$$s_{j,k}^{n'} = \max(r_{j,k}^{n'}, max_{(n,n')\in M_{i,j}'}(s_{i,l}^{n} + C_{i,l} + \sigma_{l,k}), \gamma_k) \tag{9}$$

where $\sigma_{l,k}$ indicates the communication time between π_l and π_k.

The delay element γ_k will be updated each time the starting time of an instance that is allocated in the same core is calculated in the scheduling. So for a non preemptive system, each time $s_{i,k}^{n}$ is calculated, the γ_k is updated by:

$$\gamma_k{}^{*} = \max(\gamma_k, s_{i,k}^{n} + C_{i,k}) \tag{10}$$

Schedulability of the generated schedule table can be tested on each instance to verify the respect of its deadline by extension of equation 7:

$$\forall \tau_{i,k}^{n} : s_{i,k}^{n} + C_{i,k} \leq r_{i,k}^{n} + D_i \tag{11}$$

3.2.5.2 Quality of scheduling solutions

The process of determining the scheduling for multi-core system is the same as the process presented in Section 3.2.4.3, only the calculation of the start time of each instance and the updating of delay element have to be changed to equation 10 and 11. The equations 10, 11 and 12 also imply that the position of each task has to be determined before the scheduling process, i.e. the partitioning of application in the multi-core system. The partitioning method is studied in Chapter 2.

We evaluate the embedded solutions by considering the influence of scheduling decisions on the execution of one or several sequences of dependent tasks or runnables for the application compliant with AUTOSAR. The quality of scheduling solutions is evaluated by ***Global Jitter***. In a temporal interval $\mathcal{H}$, for example the hyper period of the tasks in the system, ***Global Jitter*** $\mathcal{J}$ is the sum of jitter for all instances:

$$\mathcal{J} = \sum \delta_{i,k}^{n} = \sum (s_{i,k}^{n} - r_{i,k}^{n}), \forall \tau_{i,k}^{n} \in \mathcal{H} \tag{12}$$

3.2.5.3 Demonstration

Here we show the example of the same application presented in Figure 43, the allocation of the nodes in the multi-core is shown in . The criteria values and adjusting deadline value for the task remains the same as shown in Table 3.

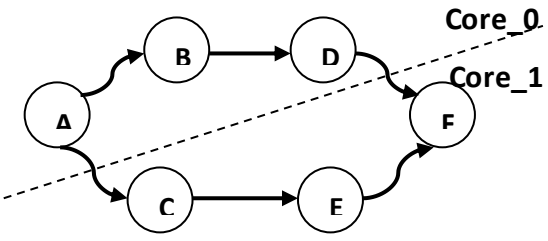


Figure 45- Example-I application in multi-core case

The list of instances is sorted in the same order as it did in single-core case, while the start time for each instance will change as the delay element is updated locally in each cores, the result is shown in the Table 7.

Ordering Instance	τ_A^0	τ_C^0	τ_E^0	τ_B^0	τ_D^0	τ_F^0	τ_A^1	τ_E^1	τ_B^1	τ_A^2	τ_C^1	τ_E^2	τ_D^1	τ_F^1	τ_B^2	τ_A^3	τ_E^3
Release Time	0	0	0	0	0	0	6	6	8	12	12	12	12	12	16	18	18
Starting Time	0	1	2	1	2	3	6	6	8	12	13	14	13	15	16	18	18
γcore_0	1			2	3		7		9	13			14		17	19	
γcore_1		2	3			4		7			14	15		16			19

Table 7-Scheduling result for multi-core for example-I

Compared to the single core, we can observe that the makespan is reduced thanks to the load balancing of multi-core. The scheduling can be visualized in Figure 46. Compared to Figure 44, the average jitter is reduced as well.

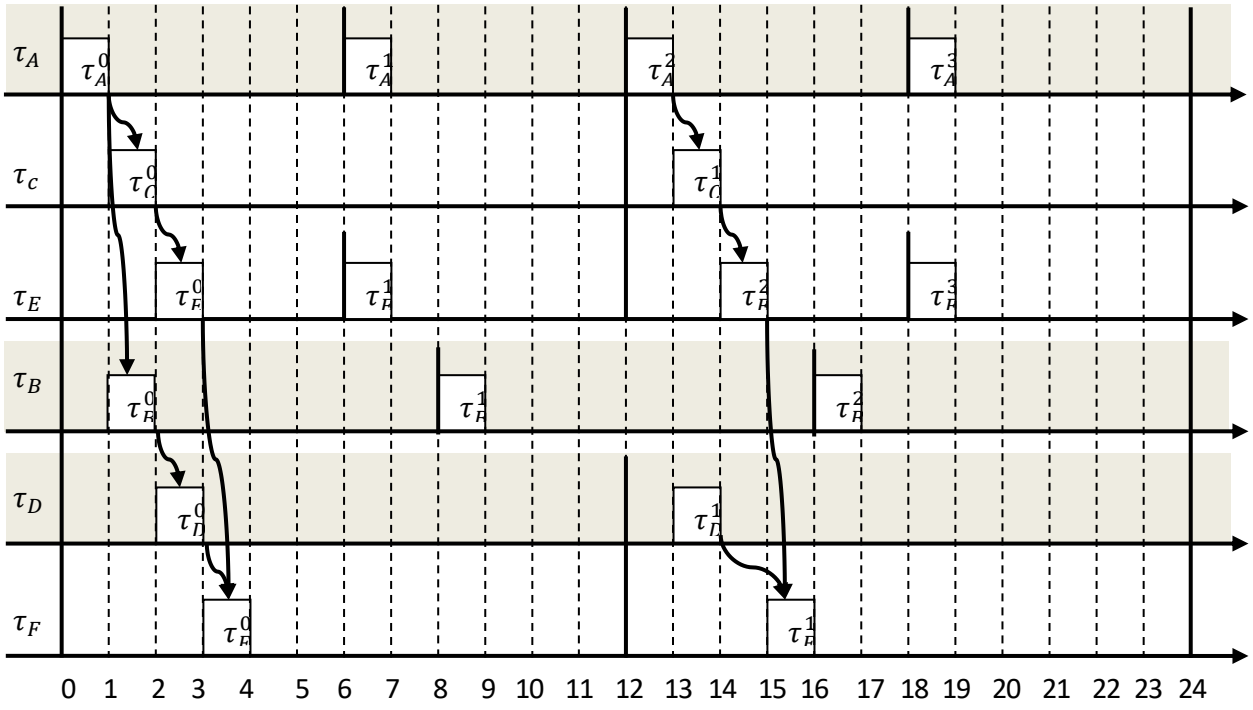


Figure 46-Scheduling multi-core example-I

The different allocation will change the scheduling result. Here we show another example of the same application presented in Figure 43, the allocation of the nodes in the multi-core is shown in Figure 47. The criteria values and adjusting deadline value for the task remains the same as shown in Table 3.

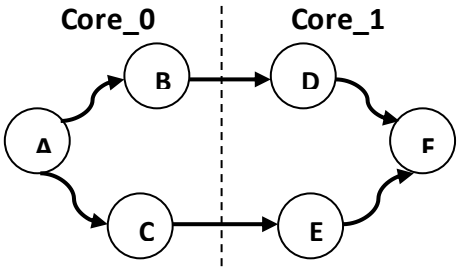


Figure 47-Example-II application in multi-core case

The list of instances is sorted in the same order as it did in single-core case, while the start time for each instance will change as the delay element is updated locally in each cores, the result is shown in the Table 8.

Ordering Instance	τ_A^0	τ_C^0	τ_E^0	τ_B^0	τ_D^0	τ_F^0	τ_A^1	τ_E^1	τ_B^1	τ_A^2	τ_C^1	τ_E^2	τ_D^1	τ_F^1	τ_B^2	τ_A^3	τ_E^3
Release Time	0	0	0	0	0	0	6	6	8	12	12	12	12	12	16	18	18
Starting Time	0	1	2	2	3	4	6	6	8	12	13	14	15	16	16	18	18
γcore_0	1	2		3			7		9	13	14				17	19	
γcore_1			3		4	5		7				15	16	17			19

Table 8-Scheduling result for multi-core for example-II

The scheduling can be visualized in Figure 48.

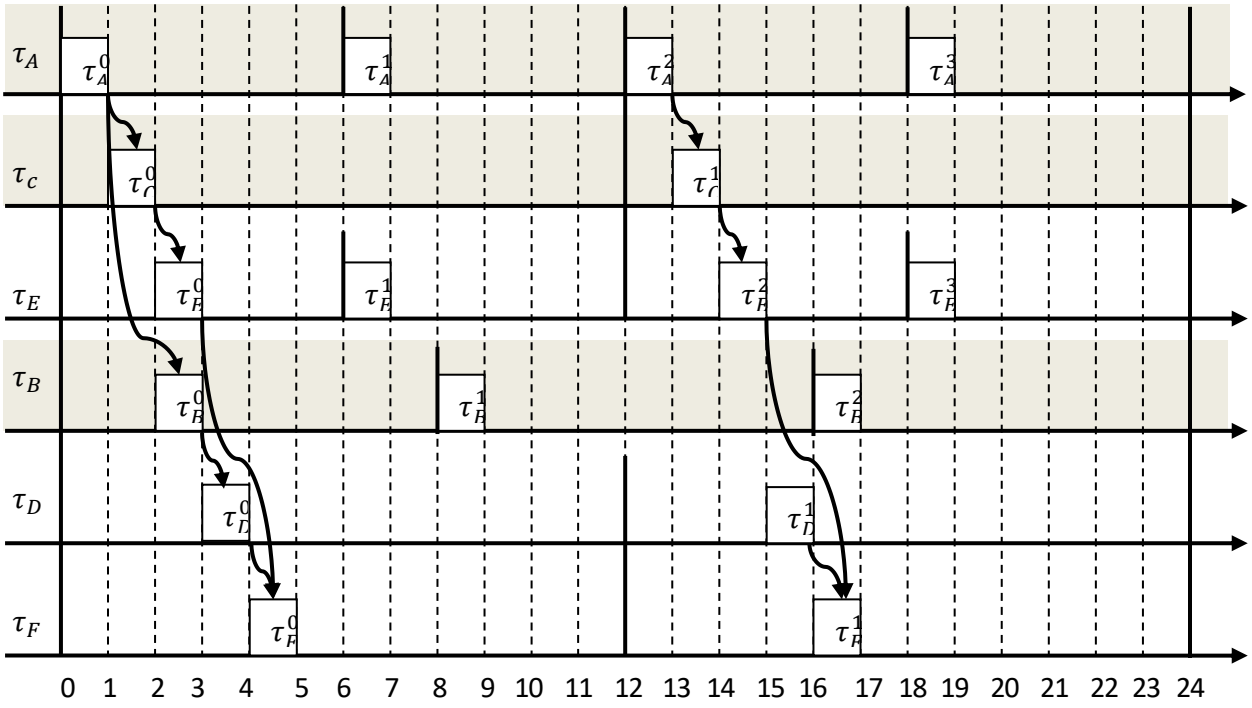


Figure 48-Scheduling multi-core example-II

By comparing to the single-core case and two multi-core cases in Table 9, we can observe that:

From single-core to multi-core, global jitter is reduced. Thanks to the parallelism of the multi-core, the delay element does not need to be updated each time that the start time of an instance is calculated. Only the instances in the same core will drive the updating in order to avoid the overlap between the independent instances. As a result, the global amount of delay between start time and release time is reduced and can be estimated early in the design process.

A change in the allocation results in the variation of jitter. For different multi-core solutions, even if the load balancing and inter-communication overhead are identical (we suppose that the load of all the tasks and all the transitions are identical), there is always an allocation allowing to reduce the global jitter. In our automotive context, these solutions are considered to be safer since it also minimizes the system laxity. For example solution Multi-core-I provides a better jitter value, as it considers the execution chain in the allocation decision.

	Single-core	Multi-core-I	Multi-core-II
Makespan	20	19	19
Total jitter	28	16	22
Average jitter	1.65	0.94	1.29

Table 9-Jitter of the example application

Actually, the performance of our proposed scheduling approach depends on the parallelism degree of the applications. For a highly parallelizable architecture as shown in Figure 49, where all the nodes can execute parallel except the source node and the sink node, the optimal makespan can be reduced from single-core to multi-core systems.

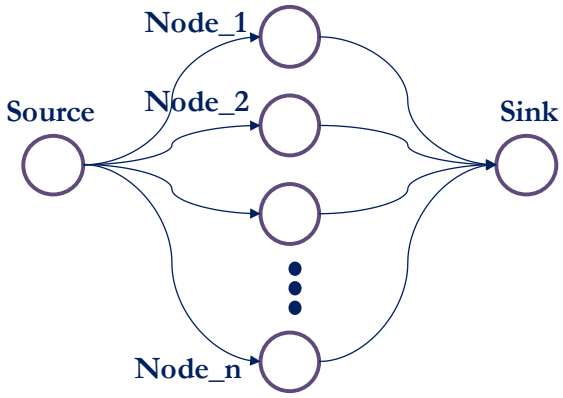


Figure 49-Example of a highly parallelizable application architecture

To simply the process, we suppose all the nodes in Figure 49 are associated with the same criteria: the execution time is **1** time unit; the communication time is **0** and the period of each node is $\boldsymbol{I + 1}$such that this application is schedulable in the single-core system, where $\boldsymbol{I}$ is the number of the nodes. We distribute the applications into homogenous multi-core system where the number of cores is $\boldsymbol{K}$. Each node is presented by a task when we perform the scheduling approach. With the increase of the applications' size as well as the quantity

of cores, the optimal makespan is shown in Figure . Multi-core systems allow reducing the makespan for this ideal model of application architecture. However, we cannot always benefit from the multi-core when the applications are strongly connected, in the next section, we present the experiment results by applying our scheduling approach to a set of applications that the parallelism degree is not easy to be identified.

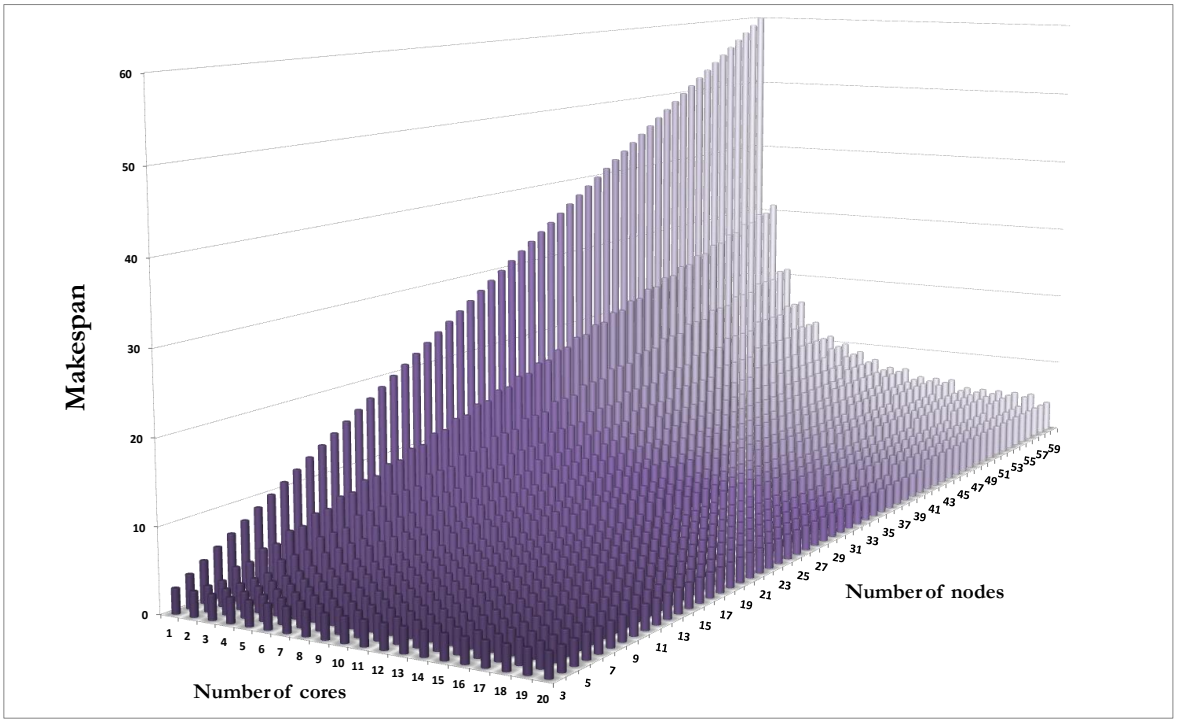


Figure 50-Optimal makespan for the applications from an ideal architecture

3.3 Experimental results

In order to analyze the results of the method, we have developed a customizable random generator of task sets. The generator takes the following parameters as input:

- I_{max}, the maximum number of runnables into generated node set,
- T_{max} such as task periods are randomly generated in $[1, T_{max}]$,
- K, the number of cores.

The system then generates randomly connections between tasks, resulting for each set to a connection ratio $\mathcal{E}$ computed as I/E, where E is the total number of generated transitions. The execution time C_i of each task is also generated such that the system utilization $\sum_{i=1}^{I} C_i/T_i < k.U_{max}$, where k is the number of cores and U_{max} is the maximum utilization enabling schedulable solutions when considering dependencies onto single-core systems. U_{max} has been experimentally set to 0.3 on real-life applications (see the Chapter 4).

Each synthetic application is then characterized by the following features: the number of tasks I, the number of transitions E, the ratio $\mathcal{E} = {}^{I}/_{E}$, the utilization of each core $U \in]0, U_{max}]$ and the number of cores k. By this way, we were looking to evaluate the impact of periodic dependencies onto the system schedulability and onto the quality of the generated schedule tables computed as the global jitter $\mathcal{J}$, see section 3.2.5.2. A scheduling is considered as not feasible if a deadline constraint is not respected by at least one instance of task. We measure the schedulability as the rejection rate of a given application which is computed as the number of conflicting instances over the total number of instances.

Here we generate about 960 applications distributed onto multi-core systems with $[2,5)$ cores. The system utilization is from about 0.6 to 1.2. The connection ratios of these applications are between 0.1 and 0.8, which means these applications are generally strongly connected. The parameters of the synthetic applications are summarized in the Table 10.

Applications quantities	Connection ratio	System utilization	Node quantities	Cores quantities
960	(0.1, 0.8)	[0.6,1.2]	[15, 60]	[2,4]

Table 10-Synthetic applications sets.

We firstly apply our scheduling approach by distributed these applications into different cores, the global jitters and makespan values for each distributions are presented by blue curves in Figure 51 and Figure 52. For the raison of comparison, we show in the same figures the results of single-core cases, which are presented by red curve in the relative figures. We can observe that from single core to multi-core, the global jitters as well as makespan reduce.

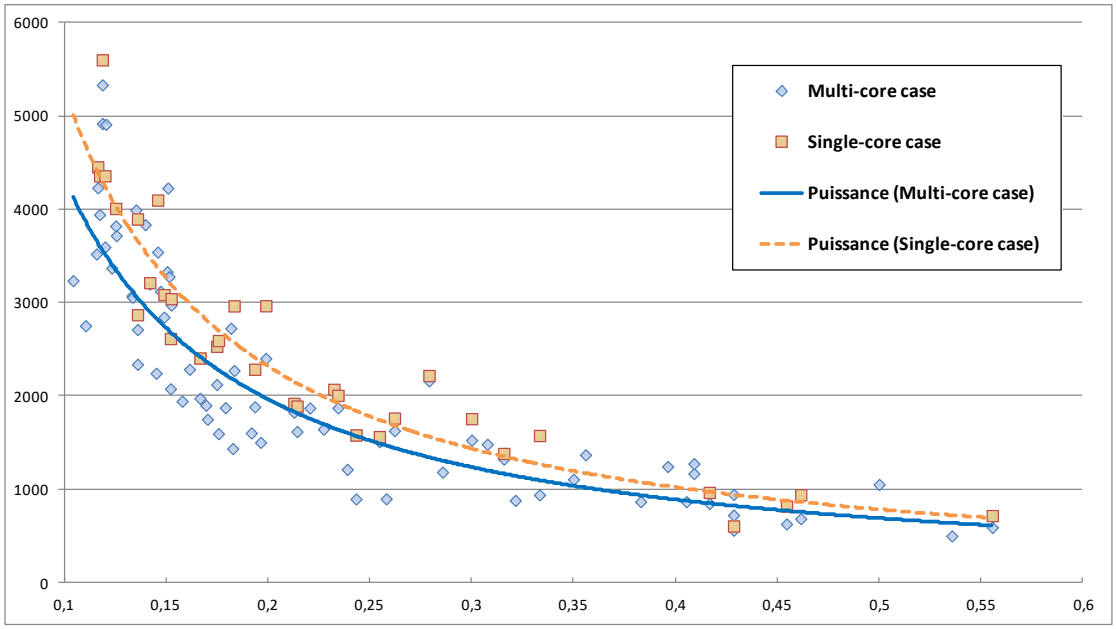


Figure 51-Experimental results obtained with synthetic applications: global jitter according to the connection ratio for single-core and multi-core cases

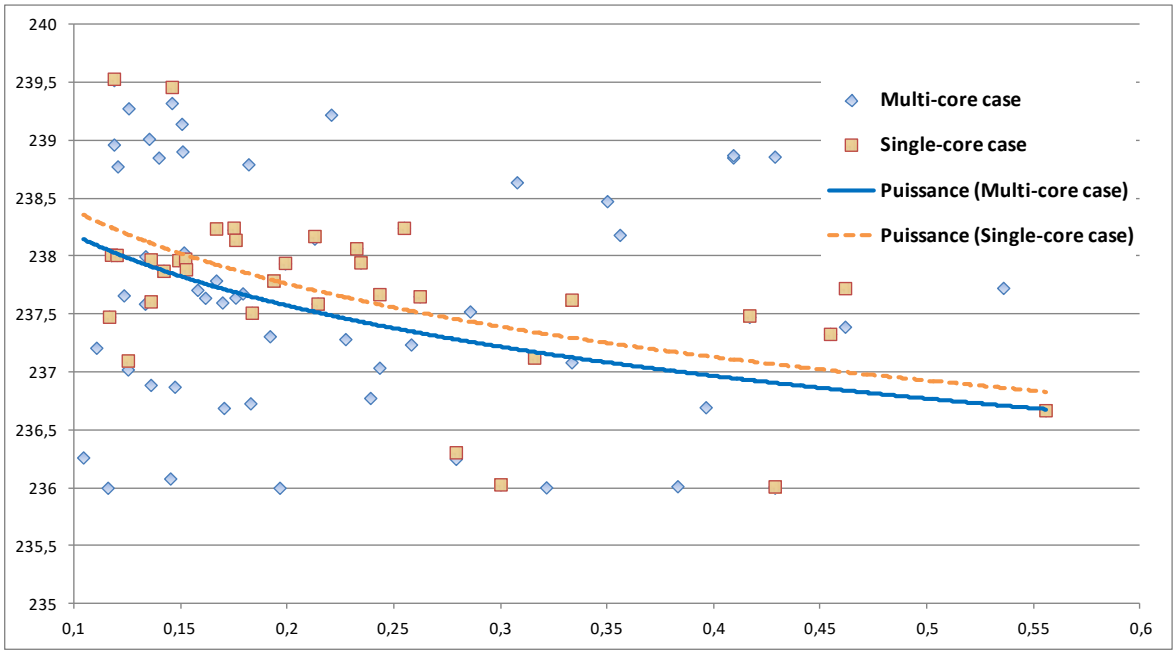


Figure 52-Experimental results obtained with synthetic applications: make span according to the connection ratio for single-core and multi-core cases

In our scheduling approach, we consider the adjusting deadline as the third ordering metric, which take the schedulability as the highest priority. To prove that, we apply the applications set to 2 other approaches that take the maximum laxity and deadline as the third metric. The quantities of the schedulable applications among these generated applications for both multi-core and single-core cases can be thus compared, which is shown in Figure 53. For the generated applications set, our approaches allow the greatest chance to find the schedulable solutions.

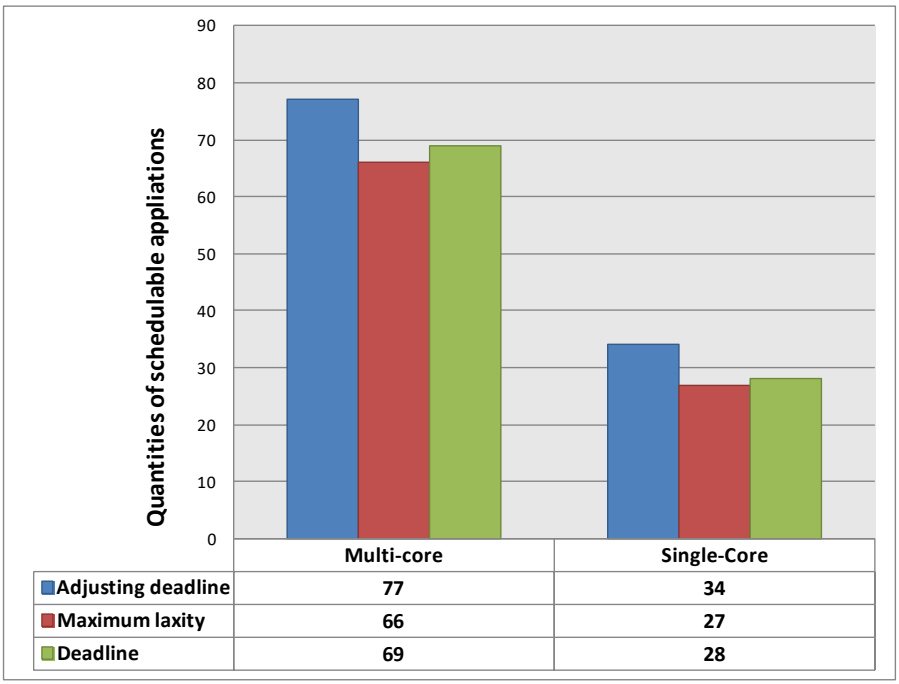


Figure 53-Comparison of approaches that consider different ordering metrics: the number of schedulable applications

Besides, among these three approaches, we take different measurement to evaluate the ordering metric.

Results confirm that the rejection rate increases with the system utilization U and inversely with the connection ratio $\mathcal{E}$.The more the scheduling is constrained by the periodic dependencies, the more the rejection rate is.

More interestingly, we can compare the differences between three scheduling policies: Earliest Deadline, Maximum laxity ($D_i - C_i$) and adjusting deadlines as proposed in Section 3.2.4.1. These 3 policies are considered as the third ordering metric to determinate the execution order of all instances in the schedule tables.

First of all, we can observe that global jitter mainly depends on the connection ratio and to a lesser extent on the system utilization. Secondly, Table 11 gives the details on the comparison of the average measured global jitters for the three scheduling. Adjusting deadline takes a better advantage of the reduction of the connection ratio (and thus of the system constraints) to reduce the jitter and finally provides better results when generating the schedule tables of synthetic applications. This is mainly because the adjusting deadline contributes to firstly schedule the instance whose successors have small maximum laxity ($\left(D_j^* - C_j\right)_{\tau_j \in succs(\tau_i)}$ in equation 6). Therefore, for the instances with same execution time and deadline, the scheduler takes the subsequent execution chains of each instance into consideration as well. We will confirm this result on a real application in the following chapter.

Ordering Metrics		**Adjusting deadline**	**Maximum laxity**	**Deadline**
Average		**2173**	2378	2359
Number of schedulable application		**77**	66	69
Connection Ratio	$\mathcal{E}$ < 0.2	**2940**	3005	2990
	$\mathcal{E}$ < 0.3	**1567**	1610	1657
	$\mathcal{E}$ < 0.4	1193	**1169**	1196
	$\mathcal{E}$ < 0.5	**855**	933	920
	$\mathcal{E}$ < 0.6	**714**	808	808

Table 11-Global jitter for different ordering metrics

Conclusion

In this chapter, we have defined the static scheduling method in multi-core systems, which is adapted to the model of automotive application compliant with AUTOSAR standard. The method supposes on an a priori allocation of the tasks on different cores. We will then integrate this scheduling method into a complete development flow for industrial software in the next chapter. The flow proposes an exploration step of software distribution. One scheduling shall then be generated for each explored solution.

Chapter 4 Developing process in automotive industry

In this chapter, we present our developed tool suit **SWAT** (**S**oft**W**are **A**llocation **T**ool suit) that is integrated seamlessly in our automotive developing process. The process is shown in the Figure 54. We present this process and how each part of SWAT works in the first part. And then in the second part, we demonstrate the experiments results on several real-life industrial use cases.

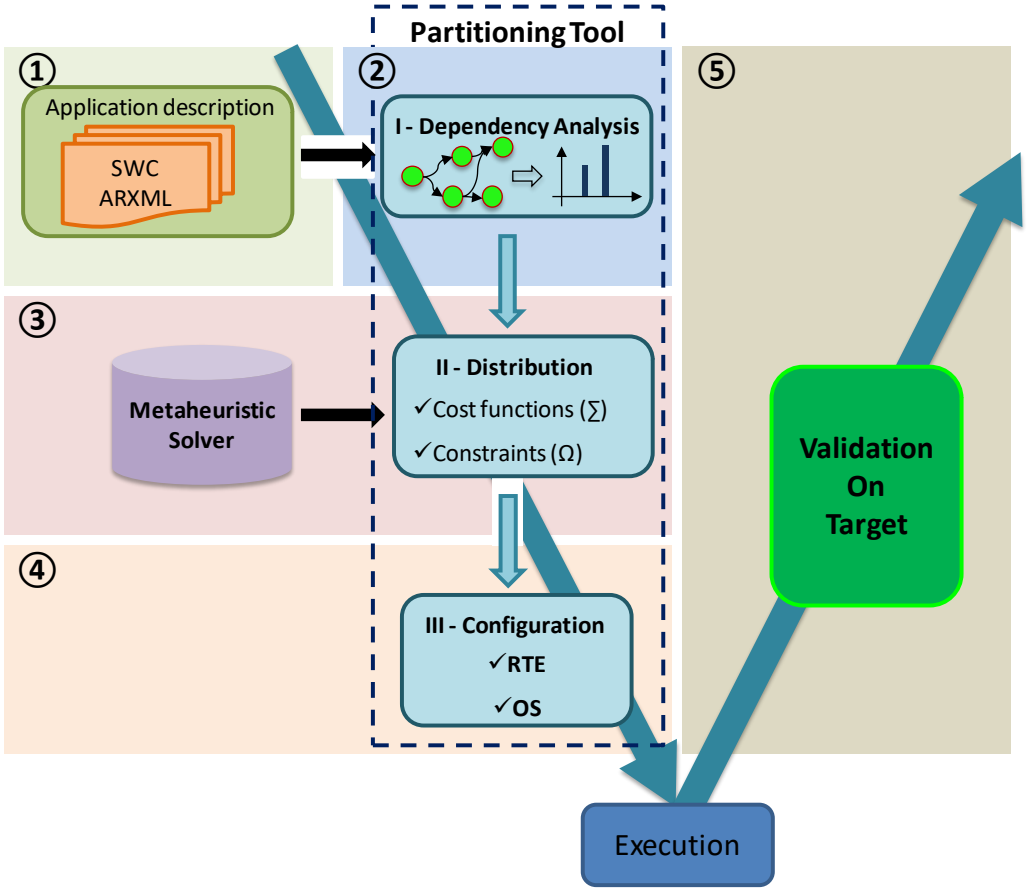


Figure 54-Working process for partitioning automotive application onto multi-core architectures

4.1 Working process

We have developed a method for the distribution of the automotive applications into multi-core architectures. The automotive applications could be compliant with AUTOSAR standards, in that case, the method acts as a decision guide environment for the partitioning of embedded software modeled with the AUTOSAR specifications onto multi-core systems. The proposed method was seamlessly integrated into an industrial V-cycle development process. This process, as shown in Figure 54, is composed of 5 main design phases:

1. Application description
2. Dependency analysis
3. Design space exploration
4. Configuration of the executive layer
5. Validation onto the target device

We give the detailed description of this working process in the following parts. Besides, as mentioned before, the software architecture is compatible with AUTOSAR standard. However, the application processed by our tools and process is NOT AUTOSAR specific.

4.1.1 Step I-Application description

This step consists in the description of the targeted applications in order to integrate them in the process. Therefore, this step plays the role of interface between the input applications and our developed toolset. As the automotive applications might be in varies forms, i.e. AUTOSAR or not, this step is application type dependant. That means, for different type of various applications, this step should perform different way to import/export the application. For the following part, we focus on the AUTOSAR application that is the user case in our project, although our process is not AUTOSAR specific. The Autosar applications are represented in the form of ARXML file (AutosaR XML). An ARXML file is a XML file that describes the interface of a software component (SWC). The format of this file is defined by the AUTOSAR specification and contains information like: the description of data that are read or written by a module; the description of entries points of the modules and the calling mode; etc. The versioning of the format is correlated with the AUTOSAR release. For example, in our different projects, there are the version 2.0.2, 3.1.2 and 4.0.3.

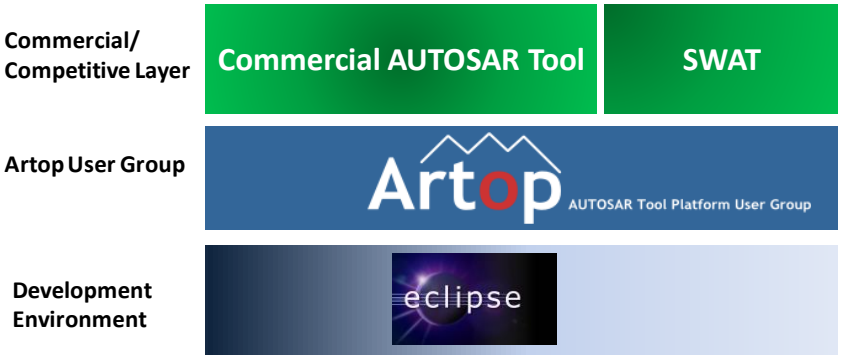


Figure 55-Integrated AUTOSAR Tool Environment

So for the AUTOSAR application, the application description step requires the tool to be able to parse ARXML files. To do that, there already exist the authoring tools such as **SystemDesk**, **AUTOSAR Builder** and **Artop** to edit the applications compliant with AUTOSAR standard. **SystemDesk** (SystemDesk, 2017) that is developed by German company **dSPACE** is a system architecture tool for modeling AUTOSAR architecture and systems for application software. Besides of the application level, it also allows generating virtual ECUs for the dSPACE simulation platforms. **AUTOSAR Builder** is another authoring and simulation toolset for AUTOSAR applications. It is part of the CATIA Systems Engineering solution from French company Dassault Systèmes (AUTOSAR Builder, 2017). Besides, there exist also other similar commercial tools, e.g**. DaVinci** from **VECTOR** company, that dedicate to design the architecture of AUTOSAR software components. Unlike **SystemDesk** and **AUTOSAR Builder**, which are the commercial

tools that require paying for licenses as they are profitable products for the companies, the Artop (AUTOSAR Tool Platform) (Artop, 2017) is an open source project that includes its sources codes and is available free of charge to all AUTOSAR members and partners. **Artop** is an implementation of common base functionality for AUTOSAR development tools like **SystemDesk**, **AUTOSAR Builder** and others. Artop is based on **Eclipse** Platform that is well-suited to develop domain-specific integrated development environment (IDE). The layered architecture for a complete AUTOSAR tool is briefly shown in Figure 55. The top layer is commercial or competitive layer where the tool vendors develop proprietary plug-ins commercially. These plug-ins adapt Artop to end-users' needs and complement the functionalities of Artop in the middle layer. The **Eclipse** Platform is located at bottom layer, including Eclipse technologies such as the Eclipse Modeling Framework (EMF) that the Artop library is base on.

The applications description in our process is accomplished by our developed tool that is located in the top layer of Figure 55. This tool is a part of the entire tool suit SWAT. The tool is developed based on the internal libraries that encapsulate all the functionalities of Artop library and allow using it without the Eclipse environment. The libraries are initially defined by the software team in Valeo for the purpose of integrating AUTOSAR software component in software, which allow to generate standard human interface, parse ARXML files, import/export the excel files, etc. Based on these internal libraries, our tool is capable of editing completely the AUTOSAR applications that represented by ARXML files. More precisely, it allows reading Autosar configuration files, creating empty AUTOSAR configuration files and populating AUTOSAR configuration files. Compared to the commercial tools, our developed tool requires less resources than commercial tools that provide more functions like simulation, virtualization and others, which exceed the require for the application description step. Additionally, our tool provides the dedicated functions for our needs, for example, the synthesis results for the targeted applications provided by the tool can be used for the next step: the dependencies analysis step. And certainly, our developed tool does not require the additional cost for paying the licenses of commercial tool.

4.1.2 Step II – Dependencies analysis – Model synthesis

In our work, we focus on partitioning applications driven by control and data flow (e.g. engine control, brake control, etc.). For that type of command and control applications the order in which the individual statements executed is very important and the enforcing by functional constraints makes it difficult to identify the parallelism degree. As the high sensibility of the execution order and low proportion of parallelism might exist in the targeted applications, the partitioning of automotive applications into multiple cores requires a fine analysis of the dependencies between functional elements. The dependencies analysis step is accomplished by Dependencies Analyzer, a tool that is a part of SWAT toolset.

The ***Dependencies Analysis Tool*** is based on Eclipse. Written in Java, it allows to analyze a software application by parsing the xml description files (e.g. *.arxml – AutosaR XML –

in an AUTOSAR context), which is done in the application description step. The tool analyzes the features by the following steps:

1 – Modeling the software architecture:

- As described in the Chapter 2, the software architecture is modeled using a directed graph $\boldsymbol{G}\ (\boldsymbol{V},\boldsymbol{E})$, such that $\boldsymbol{V}$ is a set of nodes (set of runnables for Autosar application) and $\boldsymbol{E}$ is a set of transitions (links between runnables). A node $\boldsymbol{V}$ is modeled as an execution time, a trig mode, a period. A transition $\boldsymbol{E}$ has a weight that depends on the size of data transmitted, the period of the producer, etc.;
- The graph size is optimized by the creation of buses between nodes.

2 – Determines the levels of dependency. Build statistics on transitions between executable entities (called runnable in AUTOSAR). Each transition belongs to one of these four classes that have been already presented in detail in the Chapter 2, here we just give a briefly description:

- Class 1: Periodic transition:
  Serie1: same period for producer and consumer;
  Serie2: producer period smaller than the period of the consumer;
  Serie3: producer period greater than the period of the consumer;
- Class 2: Producer OR consumer (exclusive) is periodic:
  Serie1: producer is periodic;
  Serie2: consumer is periodic);
- Class 3: No periodicity: neither producer nor consumer is periodic;
- Class 4: Transitions invoked on events (e.g. Mode Switch Event, Client/server operations).
- If AUTOSAR is targeted, two levels of granularities are allowed: analysis at SWC level or analysis at runnable level (if AUTOSAR is used, component level for all other cases). This facility can be used to decrease the complexity of analysis, and so, decrease the time to find a solution;

3 – Analyze the data information for each transition such as data size, data rate, data unit, as described previously in section 2.3.1.1.3;

4 – Identifies the sequences of communications (extraction of data flows of same rates).

Inputs of the tool: Cooperating with application description tool, the input is the software architecture that consists in

- The set of components (e.g. in AUTOSAR, this is a set of Applicative SWCs)
- The composition that structures these components and forms hierarchies.

It is worth noting that as the software architecture (also called static architecture) is given as an input, the analysis is done only once, and is excluded of the iterative process.

The BSW (Basic SoftWare) and HW (i.e.: the bus CAN etc.) are not included in this analysis for software allocation, but during the validation, BSW and HW impacts are implicitly taken into account in measurements.

Outputs of the tool: The outputs of the tools are listed as follows (take AUTOSAR applications as example). These information will then be used to perform the distribution into multi-cores.

- The transition information: the producer runnable and consumer runnable, the SWCs that contain them and their associated RTEEvent and Ports, the Interface and the transited data;
- The classification of the transition: each transition is classified into categories, according to the criteria of associated RTEEvents for producer and consumer runnables;
- The data information: the information of data for each transition contain: data size, data unit, data type, and data rate;
- The sequence chains: for the granularities of runnables or SWCs. An example for the sequence is shown in Figure 56.

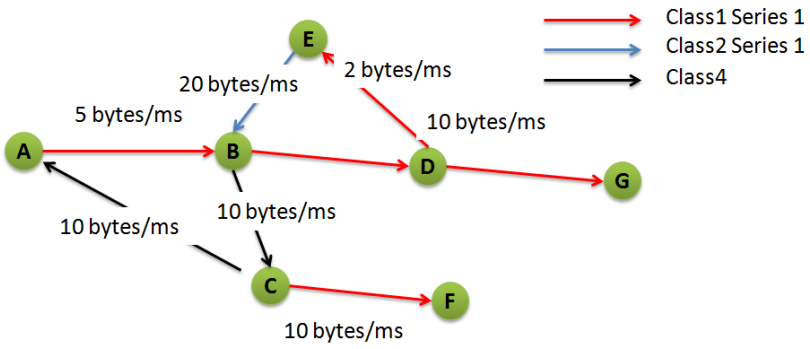


Figure 56-Example of sequence

The results of the analysis of dependencies drive the distribution step (Step III), e.g.:

- The classification information and data information are used to evaluate the communication overhead that is one of the criteria to evaluate the distribution solutions;
- The sequences of execution guide the distribution tool to determinate the response time for execution chains. It is also important for determining the execution order for the scheduling approach.

4.1.3 Step III – Software distribution tool

The software distribution tool performs Design Space Exploration (DSE) of the graph designed in Step II to distribute the applications into multi-core systems. The main work of this step contains two parts: **1) Partitioning**. The tool searches optimized allocation of the applications into different cores automatically, including the mapping of runnables and tasks into different cores. This part is presented in detail in the Chapter 2. **2) Scheduling**. For

each allocation solution, the tool generates a scheduling table that defines the order of the instances of all tasks and the start time of each instance. This part is presented in the Chapter 3.

As stated in Chapter 2, the problem is formalized as a Combinatorial Optimization (CO) problem, which mainly relies on the definition of objective functions with respect to a given set of constraints. Therefore, two essential elements are considered by this tool as shown in Figure 57: The **constraints** that need to be respected and the **objective functions** that need to be optimized.

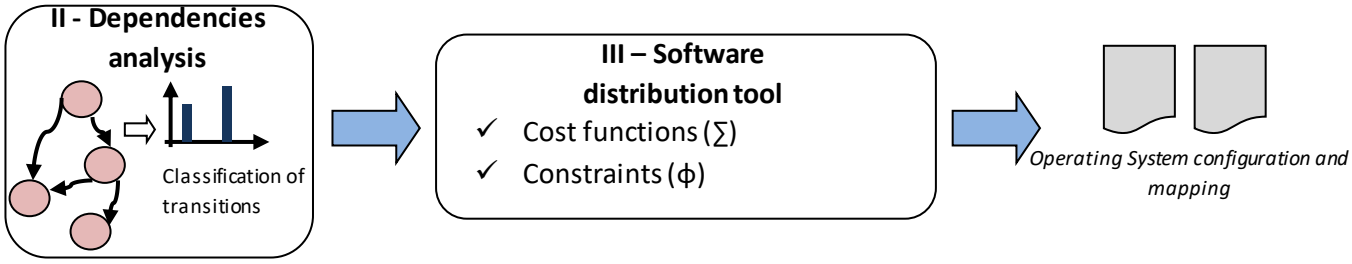


Figure 57-Software distribution tool

Constraints are static parameters that should be validated for each possible solution. These constraints can take into consideration as well real-time features (e.g. load of each core<1, load balancing, deadline compliance) as implementation strategies (e.g. forbid a slit of a client and its server).

Objective functions (or cost function) are computed from the following key elements:

- CPU utilization;
- Communication overhead as presented in Chapter 2;
- Response time for execution chains (makespan);
- Global jitters for the scheduling as presented in Chapter 3.

Supplementary inputs

In addition to the dependencies analysis resulting from step II, other supplementary information coming from the execution on the hardware platform of previous versions are necessary as other types of inputs to compute the cost function. It includes:

- ***Execution time*** for each execution entity. In the AUTOSAR context, the real-time tasks managed and scheduled by operating systems (RTOS) are composed of runnables. The execution time of tasks is related to the execution time of the runnables that are mapped to it. However the execution time is not constant at the run time, which requires an estimation value for the process. In the state of the art, the estimation of execution time can be done under 4 formats:

  - The worst execution time
  - The average execution time
  - Probabilistic model
  - Standard deviation

The worst execution time and average execution time are available for our approach, the decision to consider which of the two depends on the exigencies of the applications. For the future version, the probabilistic model is interesting to be integrated in our process.

- ***Accessing time*** to data. The accessing time for data could be used to evaluate the communication overhead especially the communication between partitions by the accessing time for global data.
- ***The feedback information*** from the measurement results on the target boards. For the iteration ***N***, the feedback information is available from the ***N-1*** iteration. At iteration 1, these inputs are computed using a runtime analysis of the single-core reference platform (on same target). These results are then updated after the iteration of the process.

Working process of distribution carried out by the tool contains two principal parts. These two parts contribute to make sure that the methods presented separately in the previous chapters can be manipulated with the concrete industrial use case and integrated in our development process seamlessly. The two parts are presented as follows:

Preparation of the graph (model of the application): The software architecture is modeled as a directed graph *G* (V, E). However, the automotive applications especially the control applications are often strongly connected. One example is given in (Kehr, Quiñones, Böddeker, & Schäfer, 2015), where a lot of cycle exists in their application of engine management system (EMS). The existence of the cycles makes it difficult to apply the scheduling approach presented in Chapter III, as it is impossible to determinate the order of the instances only based on the applications description (presented by ARXML files) without supplementary information from functional aspect. Hence in order to compute the response time for execution chains, the makespan and the global jitter, the application model shall be a directed acyclic graph (DAG). The preparation step is to transfer the original graph into acyclic graph. To do that, it involves to solve the minimum feedback arc set (FAS) problem.

A feedback arc set in a directed graph is a subset of its arc or transitions whose removal makes the graph acyclic (Demetrescu & Finocchi, 2003). An example is shown in Figure 58, where the red arrows are the feedback arcs. The minimum feedback set problem is NP-complete both on directed and undirected graphs (Karp, 1972), but the study of the minimization of **FAS** problem is out of the scope of this dissertation however.

The process of the preparation of graph is as follows:

1) Place the nodes on a horizontal line with forward arcs being drawn on and above this line whereas the backward arcs appear below this line (as shown in Figure 58 (a)).
2) Change the order of the nodes in order to find a sequence with minimum or few enough backward arcs (as shown in Figure 58 (b)).
3) Cut the backward arcs, the rest of the graph is a directed acyclic graph.

In order to find the sequence that minimizes the backward arcs in part 2), the tool performs simulated annealing (SA) algorithm thanks to its simplicity and effectiveness (more details of this algorithm can be referred in Chapter 2). During the process of the exploration, each sequence is evaluated by the objective function, which is related to the quantity of the feedback arcs. More precisely, each transition is related to a weight according to its classification analyzed by step II such that the objective function is:

$$\mathcal{F} = \sum_i \omega_i \quad (1)$$

where ω_i is the weight of the backward transition i. Typically, we penalize the consideration of a transition into a feedback arc by increasing its weight such that the cost for this solution is high enough to be avoided to be adopted.

Also in this step, dependencies such as precedence constraints are taking into account. The set of nodes that are strongly connected will not be split on different cores. This is transformed as the constraints for the searching process.

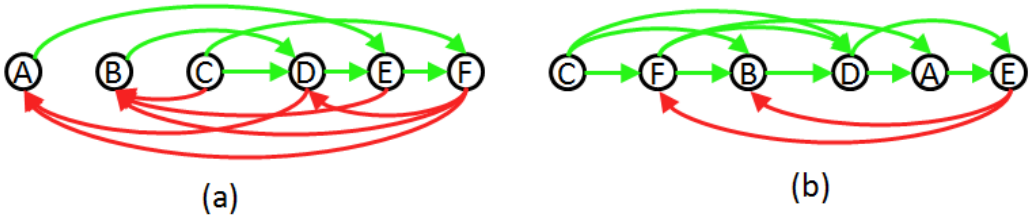


Figure 58-Preparation of graph

Optimization: Based on the directed acyclic graphs generated by the graph preparation step, optimization step involves the allocating of nodes from the graph into different cores of the multi-core platform. This step consists in two degrees of optimization:

- The degree of loads balancing involves optimizing the loads when distributing the nodes of the graph into different cores. This degree of optimization can be evaluated by several criteria such as the CPU load of each core, the communication overhead (communication loads) between the core and so on. More details are presented in the Chapter 2.
- The degree of performance involves optimizing the makespan for each core or the global jitters in the systems, which optimizes the execution order of nodes in each core. Based on this degree, the tool proposes scheduling tables as presented in Chapter 3 .

These two degrees of optimization can be solved by a design space exploration (DSE) approach. And the tool adopts the Metaheuristic as solver, where the optimization solution is evaluated by objectives functions, and the search of the solution that minimizes the costs of these functions can be carried out by multi-objective meta-heuristic algorithms such as ***MOSA***, ***NSGA-II***, etc.

Output of the distribution step contains the mapping solution in XML files and a scheduling table in XML or EXCEL files, which will be integrated in the process to generate the configuration files for the next step.

4.1.4 Step IV - Configuration of the executive layer

Before the release of AUTOSAR version that support the **multi-core** systems, it already existed a previous version of development process based on the **single-core** platform. Actually, the existing process was not multi-core dedicated. Especially, the upper layer such as system functional design & validation was not aware of the existence of multi-core. Figure 59 presents a typical V-Model for the development process in the automotive industrial, where the hatched part represents the system/function designer's point of view, and the blue part the software designer's point of view. The last one has then no knowledge of the functional constraints. That is why the application architecture designed based on the functional aspect is not aware of multi-core issues. As a result, the multi-core solutions that are proposed and generated by the proposed distribution step cannot be directly integrated into the process of industrial projects without adaption and updating.

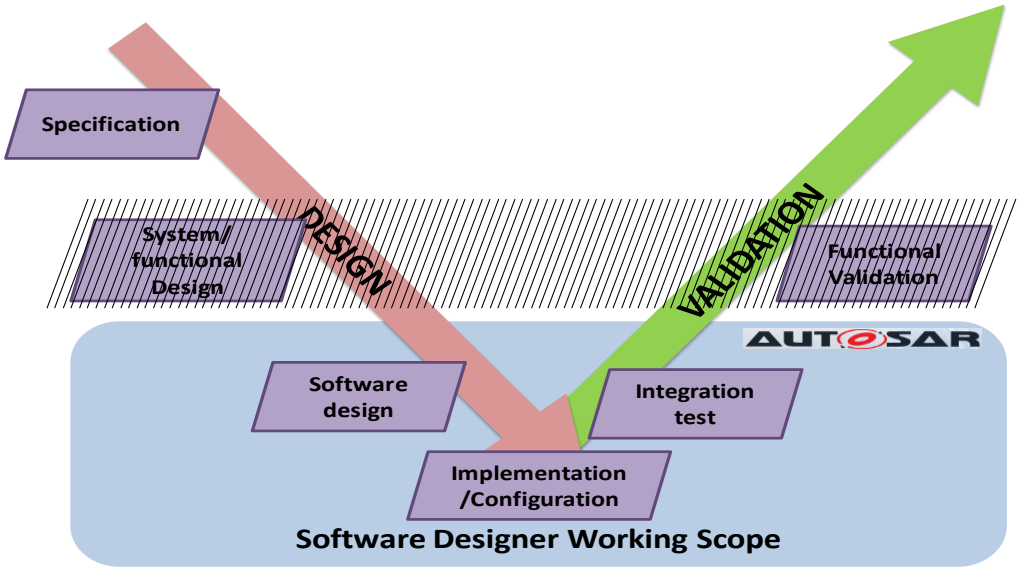


Figure 59-V-Model of development process

However, close to functional architecture, the design of software architecture for multi-core leaves few degree of liberty. Only the implementation phase such as the configuration of RTE (Real-Time Environment) and OS (Operating System) can be re-worked in order to integrate the multi-core solution. Therefore, in this step, we mainly consider the configuration of OS and RTE. The updated configurations files done by our tool will be imported in the commercial tool EB Tresos Studio (Tresos, 2017) to generate the new functional codes for all the modules to adapt the multi-core environment. Figure 60 shows this process, which contains two main parts:

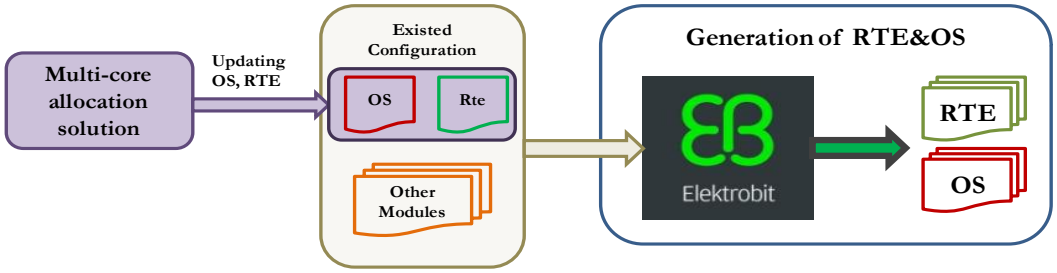


Figure 60-Generation RTE & OS codes by EB Tresos

Re-working the configuration of RTE is mainly based on the mapping information from the distribution step. The mapping solutions, represented in XML files, indicate the mapping information such as the location of the nodes/runnables. The tool updates the RTE configuration file in order to correlate with the mapping solution.

However, updating the RTE is not trivial for the real-life industrial use case as strict constraints exist. Take the communication via Mode Switch interface as an example, the separation of the runnables that communicate by Mode Switch Event into different cores is forbidden by ***EB Tresos Studio***. In order to adapt to ***EB Tresos Studio***, the tool re-work the application architecture by adding the satellite component, which involves the creation of new components and change the composition to integrate them in the architecture. More precisely:

1) Create on each relative core a satellite software component.
2) "Cut" the communication that links via Mode Switch interface (MSE_IF in Figure 61).
3) Connect each side of the component with the created satellite component in the same core.
4) Build the connection between the satellite components via Sender Receiver interface (S/R_IF in Figure 61).

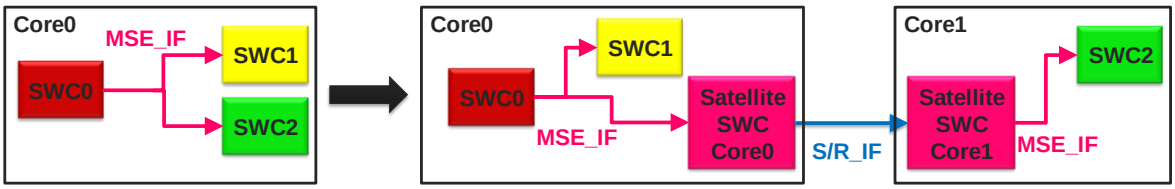


Figure 61-An example of re-working architecture for RTE configuration

Re-working the configuration of OS involves the re-mapping of the runnables to the tasks. It also creates new tasks if necessary. The principal steps contain:

1) Create equivalent task in the correspondent core
2) Allocate the runnables to the equivalent core
3) Remove the empty task.

Figure 62 shows an example, where left side represents a single core reference with 2 software components (SWC_0 and SWC_1). Each SWC contains several runnables that are mapped to different tasks (Task1_Core0 and Task2_Core0). Right side shows a multi-core

distribution solution, where each SWC is allocated to each core (SWC_1 remains in Core_0 and SWC_0 is moved to Core_1). Thus, the runnables that are belonging to SWC_0 have to be re-mapped to the other tasks: it creates the Task1_Core1 to map the runnables that were mapped previously to Task1_Core0 and Task2_Core1 for runnables from Task2_Core0. As there is no more runnables in the Task_Core0, the tool removes this empty task.

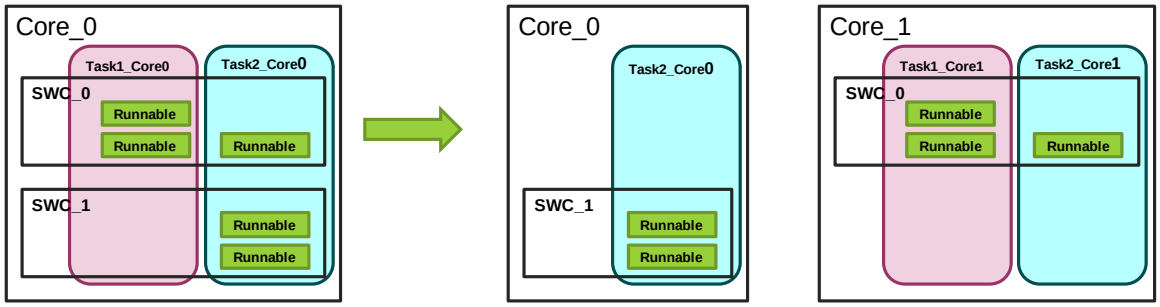


Figure 62-An example of re-mapping the runnables to tasks

4.1.5 *Step V– Validation of execution*

After the configuration step, the embed source codes can be generated, compiled and downloaded on the target architecture for the validation step of the real-time exigencies.

The evaluation of the given solution requires a complete verification of functional and real-time behavior. The inputs for validation could be the requirements from specifications (already used in step III for SW allocation decisions), e.g. one of the inputs required in step III is the execution time of runnables.

1. At iteration number 1, the execution time of runnables is computed using a runtime analysis on the single-core platform. For each runnable, the distribution of the execution time for a lot of executions is computed, and statistic results are provided (Average execution time, minimum measured execution time, maximum measured execution time, standard deviation, etc.). An example of such distribution is given on Figure 63.

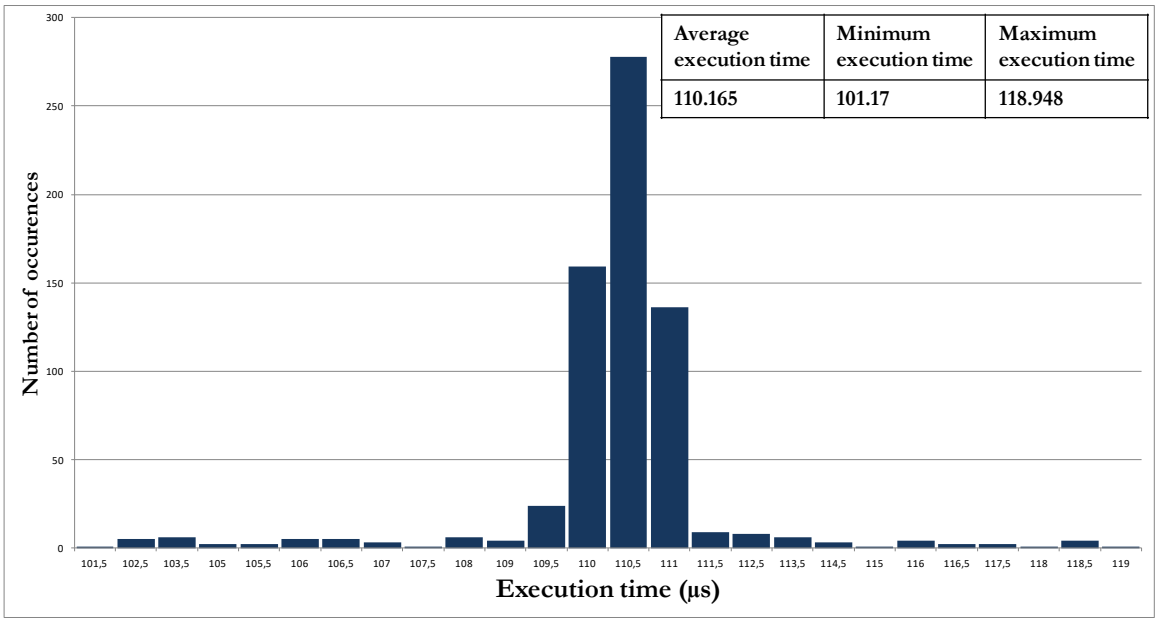


Figure 63-Execution time analysis per runnable (e.g. from a single-core platform)

2. At iteration > 1, the distribution of runnable execution is computed for the multi-core platform generated from the given solution. The real-time impacts in terms of execution distribution will be then analyzed and taken into consideration for future iterations (see the prospective step).

The same thing is done for the services used to communicate between cores (IOCs services in AUTOSAR). As this APIs are mainly responsible of additional cost, the load of inter-core communication is monitored (at runtime). We take advantage of the HIL (Hardware in the Loop) validation in order to be very close to the real environment.

4.1.6 Prospective Step – Feedback and updates

This step has not yet been integrated in the existing process. However it plays an important role for the future works. The results coming from the validation step can be used to evaluate the performance of solutions and to update the inputs of distribution tool. The feedback/update metric model for the evaluation is constructed by the following criteria (not complete list):

- **The execution time** of runnables: for each runnable, its execution time might be changed. The global CPU load should be optimized. The **speed-up** parameter is computed for each distribution (How much time can I increase the performance with a multi-core comparing to a single-core).
- **The communication** overhead: the accessing time for data might be changed especially for the IOC channel service.
- The **response time** of execution chains (makespan):
- The **robustness** of the application due to the addition of an additional overhead
- Etc.

Feedback/update metric model

The deviation for the criteria θ such as execution time and data accessing time between output and input can be given by distance function $D(\theta)$:

$$D(\theta) = log \frac{Out(\theta)}{In(\theta)}$$

where $Out(\theta)$ is the output value of θ and $In(\theta)$ is input value of θ.

If the two values are close, the distance will be around 0. The performance of a solution can be determined by a defined threshold that imposes the maximum of the sum of distance.

Based on the feedback/update metric model, the updated inputs could be used to restart a new loop of the exploration process.

And finally, go back to step 2 if required.

The estimation of execution time is important to improve the performance of the distribution process. To establish the feedback model for the criteria of execution time, we have chosen 6 multi-core solutions and apply them on the target board to measure the execution time of the runnables. Our idea is to study the elements that might impact on the execution time of the runnables. For one runnable, its execution time might be under influence of the positions of the runnables that communicate with it. To study that, we chose one runnable (with name of "***RE_EngMGsIT_018_TEV***") and measured its execution time for the 6 solution. These runnables have 14 predecessors and 26 successors as shown in the Figure 64.

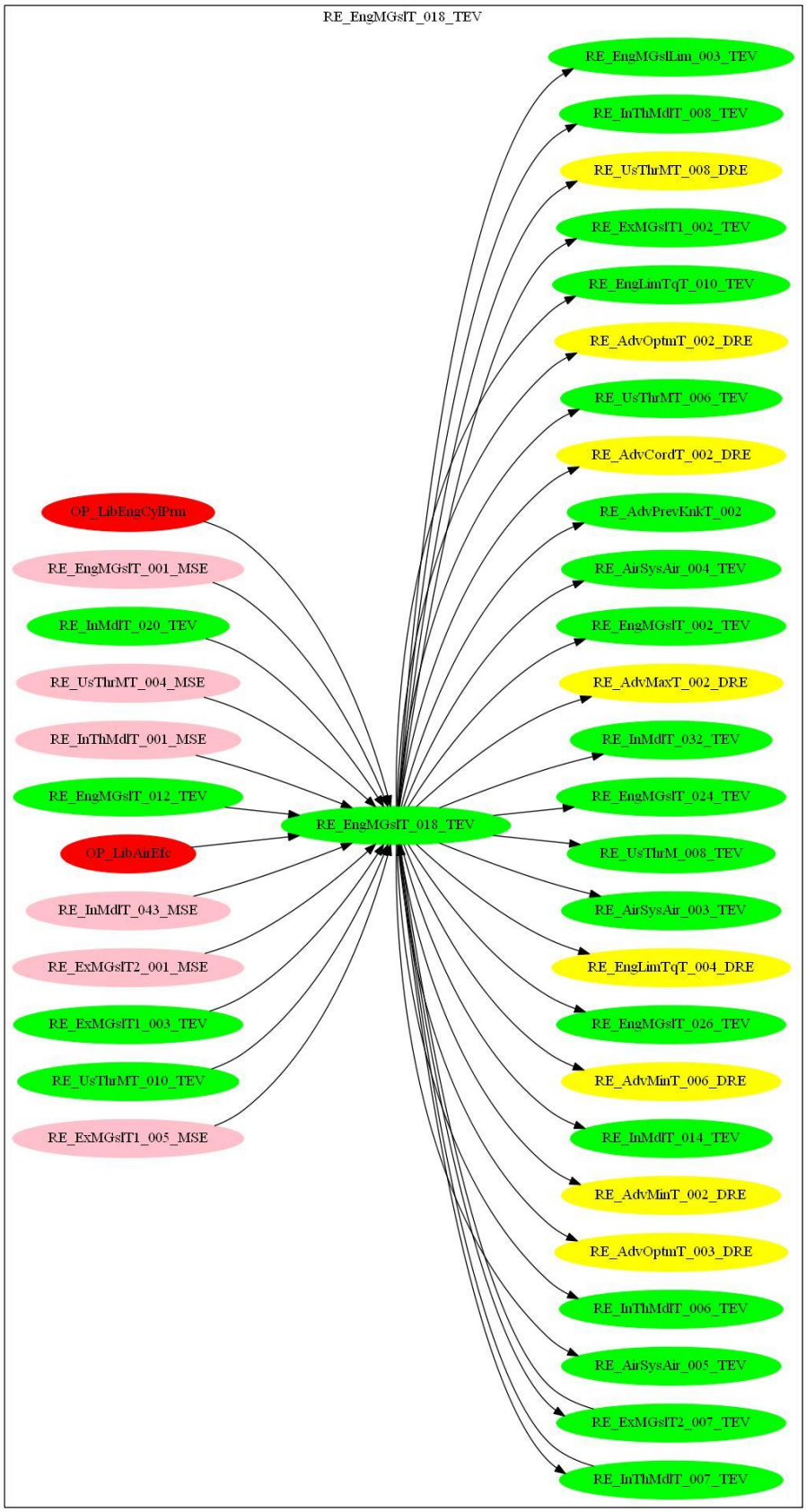


Figure 64-Communications for runnable "***RE_EngMGslT_018_TEV***"

The positions of all its 14 predecessors and 26 successors for the 6 solutions are presented in Table 12, where we focus on the number of runnables that are allocated to the different core of runnable "RE_EngMGslT_018_TEV".

	Solution1	Solution2	Solution3	Solution4	Solution5	Solution6
RE_EngMGslT_018_TEV	Core1	Core0	Core0	Core0	Core1	Core0
Predecessors of RE_EngMGslT_018_TEV						
OP_LibAirEfc*	Core0	Core0	Core0	Core0	Core1	Core2
OP_LibEngCylPrm*	Core0	Core0	Core0	Core0	Core1	Core2
RE_EngMGslT_001_MSE	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_001_MSE	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_001_MSE	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_001_MSE	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_012_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_012_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_012_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_ExMGslT1_003_TEV	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT1_005_MSE	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT2_001_MSE	Core0	Core2	Core0	Core2	Core0	Core1
RE_ExMGslT2_007_TEV	Core0	Core2	Core0	Core2	Core0	Core1
RE_InMdlT_020_TEV	Core2	Core1	Core2	Core0	Core2	Core0
RE_InMdlT_020_TEV	Core2	Core1	Core2	Core0	Core2	Core0
RE_InMdlT_020_TEV	Core2	Core1	Core2	Core0	Core2	Core0
RE_InMdlT_043_MSE	Core2	Core1	Core2	Core0	Core2	Core0
RE_InMdlT_043_MSE	Core2	Core1	Core2	Core0	Core2	Core0
RE_InMdlT_043_MSE	Core2	Core1	Core2	Core0	Core2	Core0
RE_InThMdlT_001_MSE	Core2	Core1	Core2	Core2	Core2	Core2
RE_InThMdlT_007_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_InThMdlT_007_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_UsThrMT_004_MSE	Core2	Core1	Core2	Core2	Core2	Core2
RE_UsThrMT_010_TEV	Core2	Core1	Core2	Core2	Core2	Core2
Total number	17	15	13	7	15	9

Successors of RE_EngMGslT_018_TEV						
RE_AdvCordT_002_DRE	Core1	Core0	Core1	Core1	Core1	Core1
RE_AdvMaxT_002_DRE	Core0	Core0	Core2	Core1	Core0	Core2
RE_AdvMinT_002_DRE	Core1	Core0	Core1	Core1	Core1	Core1
RE_AdvMinT_006_DRE	Core1	Core0	Core1	Core1	Core1	Core1
RE_AdvOptmT_002_DRE	Core2	Core0	Core0	Core2	Core0	Core1
RE_AdvOptmT_003_DRE	Core2	Core0	Core0	Core2	Core0	Core1
RE_AdvOptmT_003_DRE	Core2	Core0	Core0	Core2	Core0	Core1
RE_AdvOptmT_003_DRE	Core2	Core0	Core0	Core2	Core0	Core1
RE_AdvOptmT_003_DRE	Core2	Core0	Core0	Core2	Core0	Core1
RE_AdvOptmT_003_DRE	Core2	Core0	Core0	Core2	Core0	Core1
RE_AdvPrevKnkT_002	Core0	Core1	Core0	Core0	Core0	Core1
RE_AirSysAir_003_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_003_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_004_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_004_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_005_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_005_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_005_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_AirSysAir_005_TEV	Core0	Core0	Core0	Core1	Core0	Core0
RE_EngLimTqT_004_DRE	Core0	Core0	Core2	Core2	Core2	Core2
RE_EngLimTqT_010_TEV	Core0	Core0	Core2	Core2	Core2	Core2
RE_EngMGslLim_003_TEV	Core2	Core1	Core2	Core0	Core2	Core2
RE_EngMGslLim_003_TEV	Core2	Core1	Core2	Core0	Core2	Core2
RE_EngMGslT_002_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_002_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_002_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_002_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_002_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_024_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_024_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_024_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_026_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_026_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_EngMGslT_026_TEV	Core1	Core0	Core0	Core0	Core1	Core0
RE_ExMGslT1_002_TEV	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT1_002_TEV	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT1_002_TEV	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT1_002_TEV	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT1_002_TEV	Core0	Core2	Core1	Core0	Core0	Core0
RE_ExMGslT2_007_TEV	Core0	Core2	Core0	Core2	Core0	Core1

RE_ExMGslT2_007_TEV	Core0	Core2	Core0	Core2	Core0	Core1
RE_InMdlT_014_TEV	Core2	Core1	Core2	Core0	Core2	Core0
RE_InMdlT_032_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_InThMdlT_006_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_InThMdlT_007_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_InThMdlT_008_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_UsThrM_008_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_UsThrMT_006_TEV	Core2	Core1	Core2	Core2	Core2	Core2
RE_UsThrMT_008_DRE	Core2	Core1	Core2	Core2	Core2	Core2
Total number	35	18	21	28	35	23
*It concern the Client-Server communications						

Table 12-Allocations of the runnables that communicate with runnable "***RE_EngMGslT_018_TEV***"

We measure the execution time of the runnable "RE_EngMGslT_018_TEV" on our target board for the 6 solutions. The execution time is presented as the distribution of the occurrence number according to time (in µs) and is shown in Figure 65, where we put the 6 solutions together for a clear comparison. We also put the execution time from single-core reference in this figure. We can notice that for this runnable, its execution time degrade when the number (these numbers are also shown in Figure 65 for each solution) of its predecessors that are allocated to different cores increases (except solution 3 which need further study). This might inspire us for establish of estimation model of the execution time for the next iteration, which is interested for the future works.

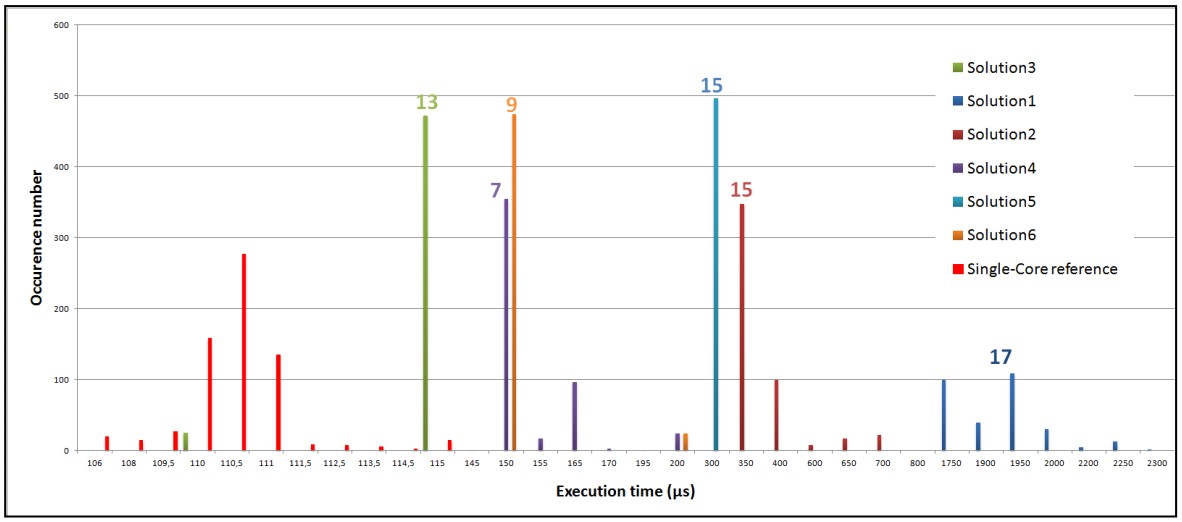


Figure 65-Execution time distribution of the runnables "RE_EngMGslT_018_TEV" for 6 solutions

4.2 Use case demonstration

In this part, we present the results by applying some industrial applications on our working process. The presentation is done following the order of the process in Figure 54.

4.2.1 Application description and analysis

We now describe the experiments leaded to determine the optimization method the best adapted to our context and to validate the explored solutions.

The method has been evaluated with three application descriptions. The first one labeled as App_1 is composed of a small amount of components. This application is built in a random way and the exploration space for this application is exhaustive thanks to its small quantity. Besides, this application contains 3 context cases for the execution time. We have two other applications (labeled as App_2 and App_3) corresponding to bigger real industrial use-cases which represent a portion of a full application of engine control. For these two applications, we consider only one running execution mode, therefore there is only one context case:

- App_1 contains 15 SWCs with 32 runnables. After analyzing this application, the tool generates 6 CpuEntities with 7 variables;
- App_2 contains 25 SWCs and 208 runnables, the tool generates 14 CpuEntities with about 493 variables;
- App_3 contains 68 SWCs and 562 runnables, the tool generates 21 CpuEntities with about 1358 variables.

The tool also analyzes the transitions information for each application and classifies these transitions according to the different levels of dependency. The results for the three tests are shown in Table 13.

Applications	Number of SWC	Number of runnables	Number of transitions	Connection Ratio	Portion of EMS
App_1	15	32	27	1.19	NA
App_2	25	208	1558	0.13	5%
App_3	68	562	6826	0.08	10%

Table 13-Applications information

For the applications App_2 and App_3 from the real use-case, we present in Figure 66 the data rate information for each runnable in these two applications. As presented in the Chapter 2, runnables produce or consume a set of data. Therefore, we present in the figure the size of data for each runnable classified by their periods. From where we can notice that, the runnables with period of 10ms access to the data at a high frequency. These results will help the tool to determine the dependency level for these transitions. These results also justify the importance of evaluating the cost of inter-core communication in the objective functions.

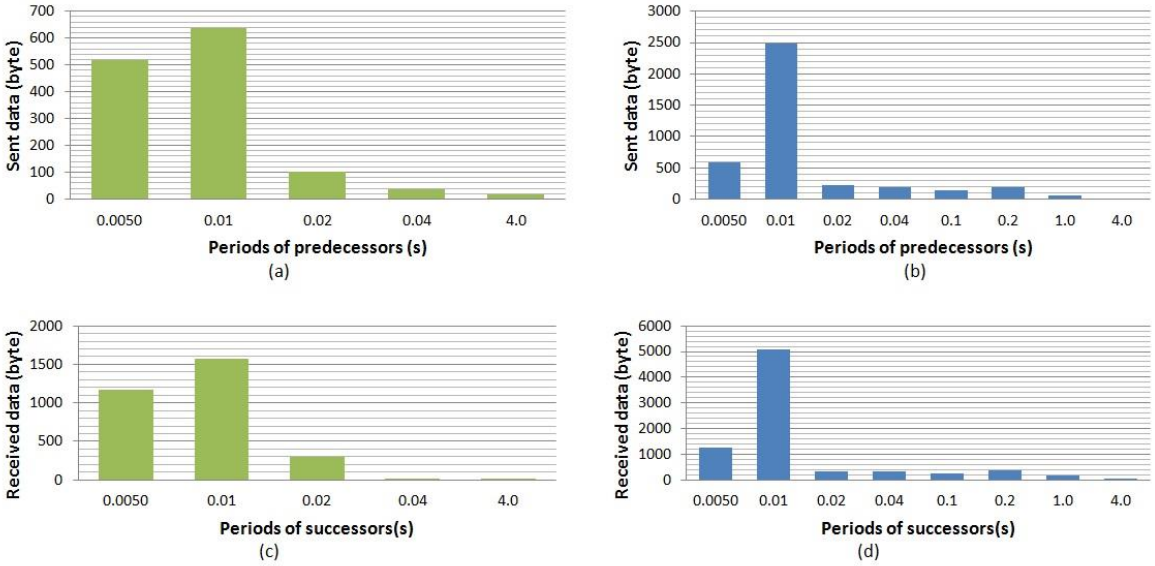


Figure 66-Statistic of transition (The left side is App_2 and the right APP_3)

4.2.2 Distribution results: Allocation

The next step consists in distributing the application into a specific multi-core architecture. Our targeted multi-core architecture contains 3 cores, a shared memory and each core is assigned to a local memory. In order to distribute these applications into multi-core systems, the tool applies the selected metaheuristics: SA, TS and GA. The small application *App_1* allows us to obtain independently all the possible combinations and to calculate their cost based on cost function. Thus we can identify the optimal solution with the smallest cost values among all the potential solutions. The distribution of cost values for all the partitioning solutions of *App_1* is illustrated in Figure 67. The figure exposes the complexity of the problem even when considering an AUTOSAR application composed of only 32 runnables. The number of feasible solutions exceed several hundreds of thousands solutions (279888 exactly), and so the optimal solution (with value of cost at the left side in Figure 67) only represents 0.0357% of the landscape.

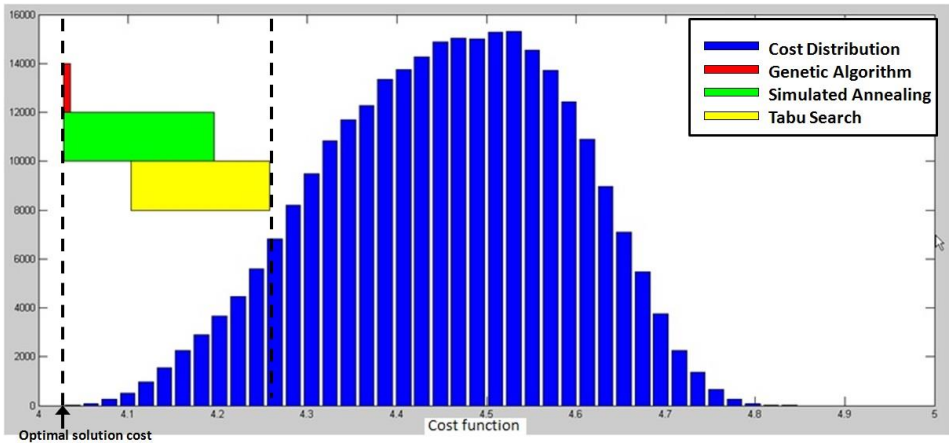


Figure 67-Distribution of the costs of all the partitioning solutions for application *App_1*. The cost bands on the left represent the subset of solutions found by the GA, SA and TS methods.

We then apply each algorithm 10 times on application *App_1*. The cost bands of solutions found by each algorithm are compared to the previous distribution of cost values as shown in Figure 67. Only GA works on a population size of 10. SA and TS only explore 1 solution per iteration. The more precise results are shown in Table 14. GA (in red rectangle in Figure 67) always finds the optimal solution. SA also finds the optimum and other solutions with a cost between **4.02** and **4.2**. Finally TS never find the optimal solution, but only solutions with costs between **4.1** and **4.25**.

Algorithms	Deviation to best solution	Optimal solution finding times/10	Average Run Time (ms)	Number of explored solutions
SA	0.0	4	243.52	10^6
GA	0.0	10	279362.09	$10X10^6$
TS	1.97%	0	7467.08	10^6

Table 14 Optimization results for application *App_1*by GA, SA and TS meta-heuristics.

From these results, we can notice that GA can always find the best solution in a longer running time. SA runs faster with a chance less than **50%** to find the optimal solution. Considering TS, unfortunately, we never get the optimal solution, but solutions very close to it.

For the two other applications, we considered real-life industrial use-cases and focus on quantitative results. We applied only SA and GA, as TS does not show its capability to find the optimum for the small application. We remind that we consider constraints of loads balancing for each solution, data for inter-core communication are allocated in the shared memory, and the cost function minimizes inter-core communication overhead (using IOC). With the growth of the application size, it becomes impossible to obtain all the solutions in the exhaustive way as we did on the small application. So, the optimal solution cannot be exactly determined. Thus, we used a different criterion to evaluate the quality criteria of the optimization methods.

We focused on the standard deviation between the costs of solutions obtained by each algorithm and the cost of the best solution it ever found. The results for the two applications are shown in Table 15. From these results, GA can no longer find better solutions than SA. Besides, the run time of GA is much longer. The average run time for both algorithms increases with the size of application, this is shown in Figure 68, where the average run time is plotted according to the application complexity. This figure specifies the average measured values.

Algorithms	Deviation to best found solution		Best solution found		Average Run Time (ms)		Number of explored solutions
	App_2	*App_3*	*App_2*	*App_3*	*App_2*	*App_3*	
SA	0.12%	21.23%	8	1	35305	752202	1000000X1
GA	2.83%	10.48%	7	0	663305	14355694	1000000X10

Table 15-Optimization results for application and by SA and GA meta-heuristic

As previously explained, the goal of our partitioning tool is not to still reach the optimum but rather to prune the design space, and only present to the designer the most promising solutions according to a specific objective function. Only the designer can then identify feasible solutions and take the final decision. Nevertheless, from the optimization point of view, these experiments allowed to identify the algorithm the best adapted to this design problem, even if each of them could be tuned to reach better results. Hence, for this use case, SA shows its ability to provide both the optimal solution and a set of other solutions approaching the optimal one. SA also seems to better scale with the application complexity. The analysis of performances metrics (cores loads, memory occupation, execution time…) then allows finer selection.

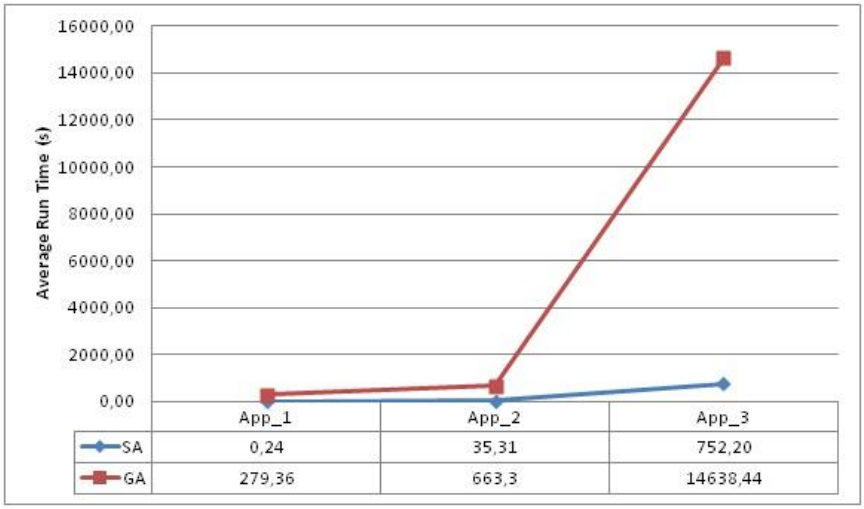


Figure 68-Scalability of the execution time of SA and GA optimization methods.

4.2.3 Distribution results: Scheduling

In the second part of the distribution, we apply the proposed scheduling method on application *App_2* that represents a portion of a full application of engine control. Our objective is to migrate this application into multicore platform without the intervention of application designer. However, as the application is single-core originally, which is designed in a sequential way and strongly synchronized, the parallelization of the application increases the cost due to this synchronization. Previously, the migration to multicore was done manually, which necessitates a high level knowledge of applications from the SW designer. Moreover, each time for a new application, it introduces a lot of repetitive work for the migration into multi-core. In addition to these significant workloads, the manual migration also prevents the optimization of criteria such as CPU loads, jitters and so on.

We choose 3 allocation solutions to evaluate the scheduling algorithm:

- **Solution_S1 Single-core case**: all the runnables are allocated in the same core.
- **Solution_S2 Multi-core case**: the allocation is a previous mapping solution to 3 cores that was done manually.

- **Solution_S3 Multi-core case**: the runnables are allocated into 3 cores, the allocation is optimized mapping solution obtained by our tool by considering inter-core communication overhead as criteria

The source code of these solutions (and more generally all the solutions found by the exploration tool) can be generated and associated to the code of the embedded executive layers. Once compiled, the binary file is downloaded onto the device. The target hardware platform is a TC27x tri-core micro-controller from Infineon and measurements are done onto the platform using Trace32 tool from Lauterbach. The trace of execution are extracted and analyzed in a pseudo-automatic manner. We can for example compute the average load per core and the execution time of each instance, or the jitter of each instance during a period of time. More detail about the platform will be described later in the section 4.2.4. Here we present in Table 16 the measured jitters for the three solutions specifically studied Chapter 3, where the schedule tables are compared in terms of CPU loads and total jitters (in μs). We compared the three scheduling policies for the three solutions.

Solutions		**Solution_S1**	**Solution_S2**	**Solution_S3**
CPU loads / Core		21.9%	20.1%	16.1%
Ordering metrics	Deadline	16055	Non Sched.	Non Sched.
	Maximum Laxity	15227	Non Sched.	Non Sched.
	Adjusting Deadline	15927	33009	29761

Table 16 - Comparison of different scheduling policies for the generation of schedule tables. (Non Sched =No Schedulable)

Compared to the preceding results that apply the method in a set of synthetic applications as presented in Chapter 3, the real application App_2 has a connection ratio $\varepsilon = 0.13$, which makes it a really difficult scheduling problem (see the rejection ratio on).

Secondly, during phase 4 of the working process (Figure 54), additional code is generated for the RTE and the IOC management on each core.

IOC stands for inter-core communication function and is responsible for the communication inter-core. It results that the total CPU load increases when considering multicore distribution, even if the local load is reduced on each core. The results are synthesized in Table 16, where we can see that with the increase of CPU loads (from single-core to multi-core solution), the scheduling policies such as Earliest Deadline, Maximum laxity $(D_i - C_i)$ show no longer the capability of finding a feasible scheduling. While adjusting time, that considers the schedulability as a high priority, provides a scheduling result even in the case of high CPU loads. De-facto in this application, the runnable with the smallest period easily exceeds its deadline, (3ms compared to the maximum period that is 3s), which makes it very critical to find a feasible order. As a result, the order of the instance in the schedule table is very important and only the consideration of the complete

execution chain by adjusting deadline allows to find a feasible scheduling and an efficient embedded schedule table.

However, the global jitter increases when distributing the application onto multiple cores. This is due to the high number of inter-core communications in the particular case of this application. Despite this constraint the optimal solution automatically obtained with our method exhibits better results than manual distributions and reduced CPU loads compared to single-core execution.

4.2.4 Validation on the target

After the distribution phase, the embedded source code of the solution is generated, compiled and downloaded on the target architecture for the final validation of both the real-time and the functional exigencies.

The target hardware platform is a TC27x tri-core microcontroller. There are two categories of memories: the local memories attached to each core and the global memories. There are three cores in this architecture, two identical cores TC1.6P and another core TC1.6E. All these three cores execute the same set of instruction. There are two independent on-chip buses in the tri-core architecture: Shared Resource Interconnect (SRI) and System Peripheral Bus (SPB). The SRI is the crossbar based high speed system bus for TC 1.6.x CPU based devices. The SPB connects the TC1.6 CPUs and the general purpose DMA module to the medium and low bandwidth peripherals. More details can be seen in (Infineon, TriCore Microcontroller, 2017).

We deployed the application App_2 onto this multi-core platform to measure the communication overheads and CPU loads for several distributions. After starting the execution, the trace information was obtained by the vendor tool - Lauterbach Trace32. We present in this section the results obtained for two specific solutions:

- **Initial solution**: it is the first generated solution from which the metaheuristic algorithms search the near-optimal distributions;
- **Optimized solution**: the best solution founded by SA and GA. As shown in the section V-B, the two algorithms could find the same optimized solution for this App_2. .

The source code of all the solutions found by the exploration tool can be generated and associated to the code of the embedded executive layers. Once compiled, the binary file is downloaded onto the device. We aim at comparing the estimated and real (measured) performances of the explored solutions. The measured communication overhead for the two solutions specifically studied in this paper are given in Table 17. Estimated values are given by considering the number of data access per millisecond (taking into account the number of fetches required to get data, i.e. the size of data). Measurements are done onto the platform using Trace32 tool and provide the exact amount of time used for inter-core communication. It appears in Table 17 as a percentage of the total application execution time. The trace of execution are extracted and analyzed in a pseudo-automatic manner. We

can for example compute the average load per inter-core communication functions (called IOC), and per core by identifying the individual IOC calls, and their execution time, during a period of time.

	Initial Solution			Optimized Solution		
	Transition counts	Estimated overhead	Measured overhead	Transition counts	Estimated overhead	Measured overhead
Core_0	144	26.25	3.25%	114	26.03	2.0%
Core_1	99	37.20	3.23%	67	22.68	0.94%
Core_2	110	23.50	1.37%	78	15.00	1.2%
Total	353	86.95	7.85%	259	63.71	4.14%
Gain				26.63%	26.73%	47.26%

Table 17-Estimation and validation results of the communication overhead on the Aurix TriCore target

By comparing real values with estimated values, we can observe that the optimization done by the tool is confirmed by the experiments despite an estimation error. More precisely,

- Table 17 represents the inter-core communication cost for each source core (executing the producers of data)
- Table 18 shows the associated core loads,

both for the initial and optimized solutions.

	Initial solution		Optimized solution	
	Estimated	Measured	Estimated	Measured
Core_0	4.62%	21.8%	5.34%	20.0%
Core_1	6.51%	21.1%	4.66%	13.3%
Core_2	4.66%	14.4%	5.78%	15.6%
Total	15.79%	57.3%	15.78%	48.9%

Table 18-Estimation results on the CPU loads on the Aurix TriCore target

More precisely, we present in Table 17 the following results of the inter6core communications for both solutions:

- the transition counts represent the number of transitions between cores. Each transition is related to 2 IOC functions: send and receive;
- the estimated overhead considers the number of data access per millisecond (taking into account the number of fetches required to get data, i.e. the size of data);
- the measured overhead is the load of IOC functions measured on the target. We can observe in this table that measured overhead is correlated with both transition counts and estimated overhead.

These results show a systematic reduction of the communication and the load metrics, and allow evaluating the error of estimation.

Firstly, according to the Table 17, the optimized solutions are better, about **26%** more efficient from the partitioning tool point of view, and about **47%** in the real platform. It

corresponds to about **26%** of minimization of the number of inter-core transitions. Even if communications are not represented with the same unit in Table 17 we can observe a difference in the global gain.

This error of estimation is not very surprising. Performance estimation is currently computed only from the amount of data exchanged between cores. In fact, the count of transitions impacts also the communication overhead. This explains why in Table 17 the decrease of estimated overhead does not necessarily improve the measured overhead while the transition count is increased. Besides, additional features such as the OS services and the memory protection unit (MPU) increase the communication overhead. These overheads should be modeled in the next version of the tool.

Moreover, the on-board profiling showed that, as a system call is done each time the application needs an inter-core communication, it could be more efficient to have 2 data accesses in one communication channel than having 2 communication channels with 1 data access in each. This new optimization will be added as a new type of move (in Chapter 2) during the exploration.

Secondly, Table 18 shows the estimated CPU load for initial and optimized solution. The partitioning tool considers the CPU load balancing as one of the design constraints, and ensures a global load balancing between cores (with a **1%** tolerated deviation). The results show that this constraint is respected by the partitioning tool, since based on estimations. The load of cores is measured with Trace32 using dedicated scripts whereas we only consider the load generated by applicative runnables in the estimations. The loads of these runnables were previously measured with Trace 32 onto a single-core distribution (without inter-core communication) and back annotated into the application description file.

Thus, the other parts of code executed by the application, such as BSW, OS and other stacks are not considered in the estimations computed by the partitioning tool. On the other hand, real CPU loads are obtained on-board by measuring the time spent in the idle task, and by subtracting the load dedicated to the BSW tasks (main functions). If the current measure provides a best precision compared to high-level estimations, it can still be improved since OS features and other modules are counted in the application load. This explains the differences in the results presented in Table 18. Precisely, we can observe a constant global load according to estimations whereas measures point out the consequences of the distribution onto the core load, due to OS and communication overheads. The execution time of the functional code of the runnables only represents **30%** of the global load of this automotive system.

We are now working on adding an intermediate fast validation phase between the distribution and the validation phase to improve the quality of our estimations during exploration. We are developing a SystemC transactional simulator of the multicore software distribution. Besides, similarities between the SystemC language and AUTOSAR have already been demonstrated (Krause, Bringmann, Hergenhan, Tabanoglu, & Rosentiel, 2007). At this level, the hardware architecture can be essentially abstracted. The concurrency is modeled at the core level, the goal being to reduce the estimation error on communication

costs, to explore more accurately the scheduling of tasks, and to identify in the early phase of the design the conflict of resources. This new simulation step will allow short and long validation cycles in the same multicore design flow.

Chapter 5 Conclusion & Perspectives

Conclusion

The multi-core dimension introduces additional challenges that are still difficult to deal with in real world industrial domains where applications exhibit high complexity and special cases features that do not always fit with theoretical models. Thus, the shift towards multi-core systems in the automotive industry has revived the challenge of application partitioning to enhance productivity, re-usability and predictability.

In this dissertation, we described the issues in the partitioning and scheduling of engine control applications in multi-core automotive systems. The proposed partitioning method is the first one fully compatible with the constraints imposed by the AUTOSAR architecture both in terms of software architecture and design process.

For the scheduling part, we focus on the periodic and dependent tasks of engine control applications. The notion of periodic dependencies has been redefined to support the transitions expressed between runnables in an AUTOSAR description. A scheduling algorithm has then been proposed to generate schedule tables on a multi-core MCU, as a total order of the instances assigned to each core.

In order to identify schedulable total orders from the partial order imposed by periodic dependencies, several scheduling policies were explored and compared. This study demonstrated that only adjusting deadlines enables to maximize the rate of feasible solutions. The proposed scheduling method is fully compatible with the constraints imposed by the AUTOSAR architecture both in terms of software architecture and design process. The results obtained on the entire working process showed the benefits of the schedule table generation phase.

The corresponding partitioning tool can thus be integrated in a seamless AUTOSAR design flow, from application description to software deployment onto multi-core architectures. Hence, classical optimization methods have been adapted to the automotive context and its specific real-time constraints in an efficient exploration tool. The entire working process has been validated onto real world applications from the AUTOSAR descriptions to the on-board profiling. The results obtained on complex motor control applications show the benefits of the optimization phase. A gain has been obtained by minimizing the inter-core communication.

After having proposed a pseudo-automatic top-down refinement process, we aim at recovering the results obtained by real measurements up to the portioning tool in order to improve the precision of the performance estimations. We first defined the loads of cores as the quality measurement of the distribution of control applications, but other metrics will

be explored for others parts of the vehicle. The experimental results showed that only one metaheuristic algorithm scale with high dimension applications.

Thanks to a multi-criteria formulation of the assignment problem, we will be able to take into account the scheduling decision during the design exploration phase, and then evaluate multi-core distributions in terms of average jitter, OS overhead, memory usage, resource conflicts and safety.

Prospective

Just as shown in the prospective step of our working process presented in section 4.1.6 of Chapter 4, the different distributions impact on the execution time of the runnables/tasks. The exact elements that influence on this variation are not clear. After the study as shown in Figure 65 and Table 12 in section 4.1.6 of Chapter 4, we can obtain a clue that for a runnable, the allocation positions of its predecessors impact on its execution time. Therefore, the more quantized study needs to be integrated in the future version of the Tools.

These first results, obtained on the recent inter-core release of AUTOSAR, also point out an increase of the cores load when migrating from a single-core to a multicore deployment. The IOC loads introduced in multi-core systems are the main reason for these supplemental loads. As mentioned before, our targeted applications are strongly connected, which is unavoidable to introduce these supplemental loads. Besides, the experiment results also shown the difficulty to find a schedulable solution and in the mean time optimize the makespan due to these special applications very synchronized. Therefore, it will be interesting to go up the top lay for re-designing and optimizing the architecture of applications, which is conscious of multi-core conception.

Moreover, thanks to a multi-criteria formulation of the future version of the cost function, we will be able to take into account several criteria to evaluate multicore distributions such as OS overhead, memory usage, resource conflicts, safety criteria...

An intermediate multicore simulation phase will also be added in the design process. In the future version of the tool, the designer will be able to navigate into the cost landscape, among the best solutions identified by the optimization method, and to validate them in simulation before the code generation of the embedded software.

ANNEX 1

This part presents the statistic result for the applications. The applications contain 2 user cases:

- EB-Mivie: the application represents 40% of the ECU.
- TDP: this application represents 5-10% of the ECU.

Both applications contain two chains of SWCs: air chain and advance chain. Analyzed by the tool, the structural information for the two applications is summarized in Table 19. From which we can notice that the TDP user case is smaller. Among these SWCs, several SWCs listed as follows are omitted because these components are not interested for the dependencies analysis:

- *Virtual Component* plays the role of interface between AUTOSAR application and non-AUTOSAT application. It contains virtual runnables, which do not exist for real. This virtual component provides several functions to the other SWCs and also calls for services from them
- *IoHwAbsIn* provides the stimulus for the entire application.
- *IoHwAbs*Out fetches the outputs of the entire application.

User Cases	Omitted Components	SWC Count	Runnable Count	Variable Count	Transitions		
					Transitions (non-Bus)	Bus	Total
EB-Mivie	VirtualComponent	67	562	1358	1893	981	6826
TDP	VirtualComponent EcuStateManager IoHwAbsIn IoHwAbsOut	26	208	493	543	255	1558

Table 19-Constructional information of applications

The analysis result for all the categories of transitions is synthesized in Table 20:

Classes		Application EB-Mivie			Application TDP		
		Transition Count			Transition Count		
		Inter-SWC	Intra-SWC	Total	Inter-SWC	Intra-SWC	Total
Class1	Series1	986	273	1259	216	63	279
	Series2	173	112	258	38	33	71
	Series3	146	56	229	56	14	70
Class2	Series1	228	144	372	45	26	71
	Series2	1604	802	2406	368	303	671
Class3		617	1427	2044	143	198	341

Class4	258		258	55		55
Total	4012	2814	6826	921	637	1558

Table 20- Results of classification

The result in Table 20 contains both the granularity of runnables and that of SWC (in the columns of Inter-SWC).

Class1

The class 1 contains the connections between runnables periodic, which is very important for the application. Therefore, the tool further analyses this class in the following scopes.

By series

1) In the series 1, the period of P_Runnable and R_Runnable are identical. The transitions information of series 1 for both applications is shown in table 8.

Tp (ms)	Number of transitions in series 1 (Tp = Tc)					
	Application EB-Mivie			Application TDP		
	Inter – SWC (by Port)	Intra – SWC (by IRV)	Total	Inter – SWC (by Port)	Intra – SWC (by IRV)	Total
5	61	43	104	61	43	104
10	882	220	1102	155	20	175
20	2	0	2			
40	25	0	25			
100	5	3	8			
200	11	7	18			

Table 21-Transitions count in Class1 Series 1

2) In the series 2, the period of P_Runnable is smaller than that of R_Runnable. The transitions information of series 2 for both applications is shown in Table 22.

Tp (ms)	Number of transitions in series 2 (Tp < Tc)						
	Tc/Tp	Application EB-Mivie			Application TDP		
		Inter – SWC (by Port)	Intra – SWC (by IRV)	Total	Inter – SWC (by Port)	Intra – SWC (by IRV)	Total
5	2	44	19	63	28	19	47
	4	1	0	1			
10	2	22	12	34	2	5	14
	4	60	14	74	1	0	1
	10	25	0	25			
	20	10	2	12			
	100	1	0	1			
	400	3	0	3	3	0	3
20	5	2	0	2			
	50	1	0	1			
40	2.5	1	0	1			
	5	1	0	1			
	100	1	0	1	1	0	1
100	2	0	12	12			
	10	0	19	19			
200	5	1	34	35			

Table 22-Transitions count in Class1 Series 2

3) In the series 3, the period of P_Runnable is greater than that of R_Runnable. The transitions information of series 3 for both applications is shown in Table 23.

Tp (ms)	Number of transitions in series 3 (Tp >Tc)						
	Tp/Tc	Application EB-Mivie			Application TDP		
		Inter – SWC (by Port)	Intra – SWC (by IRV)	Total	Inter – SWC (by Port)	Intra – SWC (by IRV)	Total
10	2	29	0	29	25	0	25
20	4	2	0	2	2	0	2
	2	22	12	34	5	12	17
40	8	6	0	6	6	0	6
	4	67	2	69	13	0	13
100	10	10	0	10			
	5	0	2	2			
200	20	0	2	2			
	5	0	1	1			
	2	1	3	4			
1000	100	1	0	1			
	50	2	0	2			
	25	1	0	1			
	10	0	8	8			
	5	0	24	24			
4000	800	2	0	2	2	0	2
	400	3	2	5	3	2	5

Table 23-Transitions count in Class1 Series 3

The histograms for the tables above are shown in the following figures. The Figure 69 and Figure 70 are the histograms for the application EB-Mivie and Figure 71 and Figure 72 are those for the application TDP. The Figure 69 and Figure 71 are the histograms distingue by the 3 series and the Figure 70 and Figure 72 are distingue by inter/intra communications.

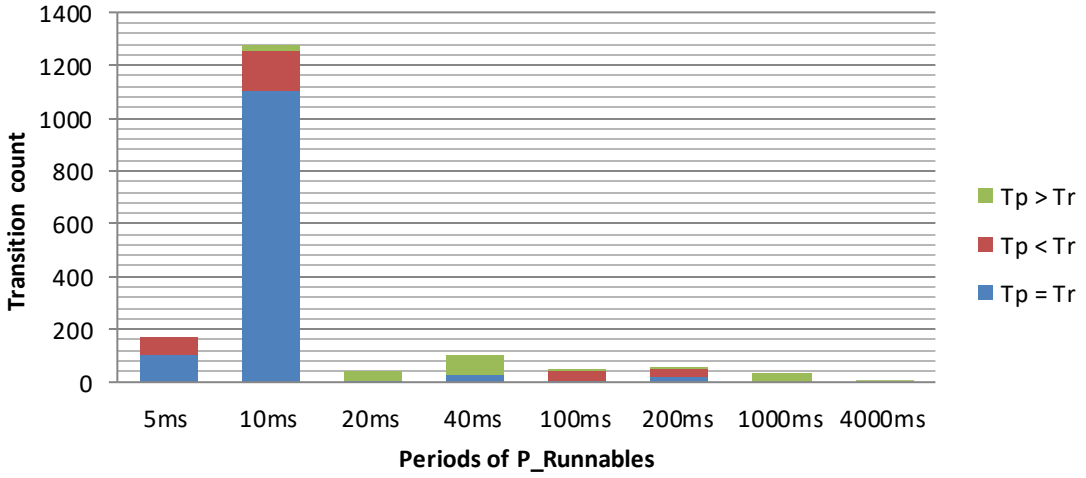


Figure 69-The count of transition for each series by periods of P_Runnables (EB-Mivie).

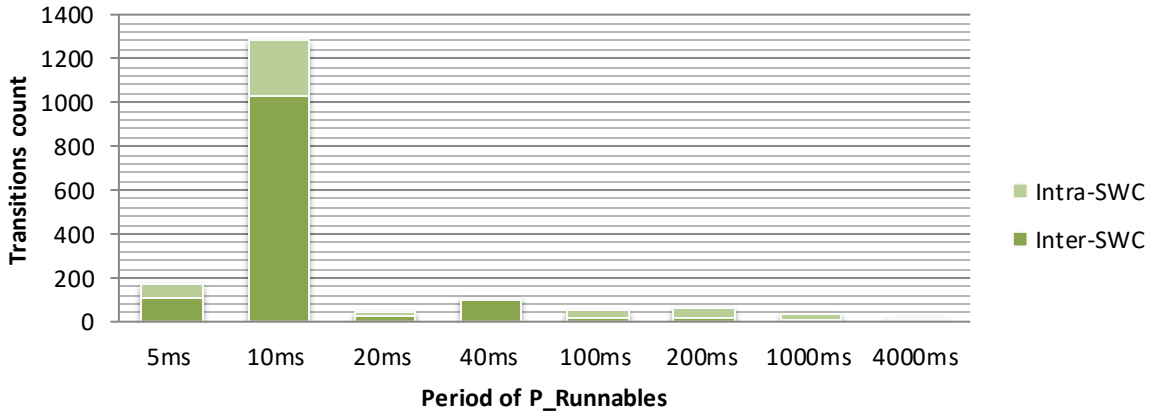


Figure 70-The count of transition for communication type by periods of P_Runnables (EB-Mivie).

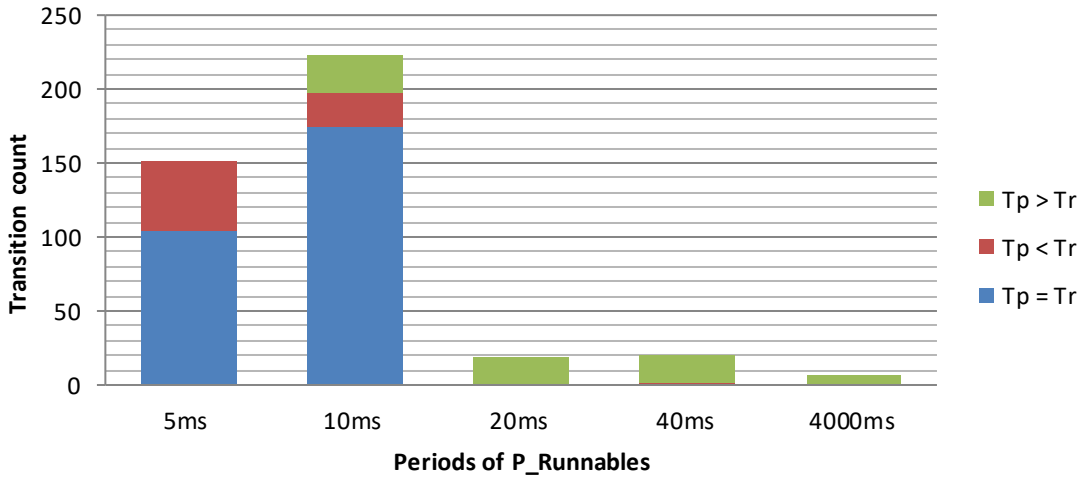


Figure 71-The count of transition for each series by periods of P_Runnables (TDP).

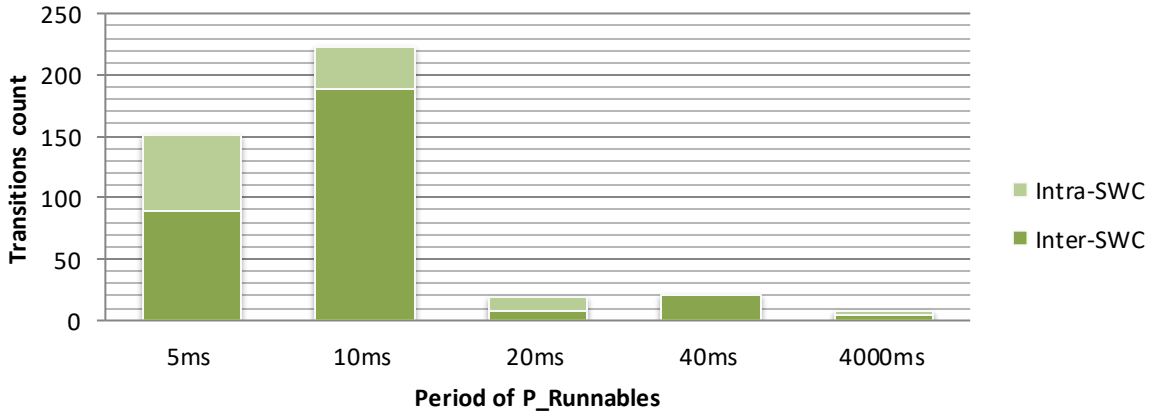


Figure 72-The count of transition for communication type by periods of P_Runnables (TDP).

Conclusion: From the results for both applications, it is obvious that the connections in class1 series1 with a period of 10ms and 5ms play an important role in the applications as the majorities transitions are belong to this group. The strong connection in this group restricts partitioning of the application

when decoupling the links between the nodes in this group.

By thresholds

This scope bases on the speed of producer runnable: the high speed transitions are those with Tp smaller than the thresholds and the low speed transitions are those with Tp bigger than it. The number of transitions bases on different thresholds for the application internship is shown in the Figure 73.

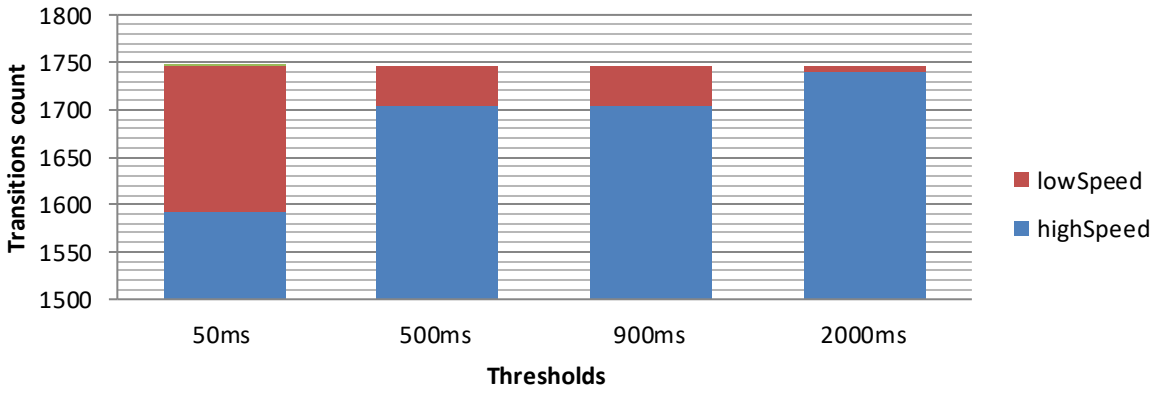


Figure 73-The count of transition compared the speed of producer to threshold (EB-Mivie).

Conclusion: when the threshold increases, i.e. from 50ms to 500ms, the disequilibrium between the high speed transitions and low speed transitions is becoming evident. Therefore, the threshold of 50ms is considered as a reasonable threshold.

Data rate analysis

There are two types of data rate: one is isolated by periods of producer runnables, which means in each period of producer runnables, only the data accessed by the producer runnables with this period is considered. Another one is accumulated data rate, which means during a certain period; all the data accessed by the producer runnables that completely finished will be considered.

Sent data rate

The sent data rate is relayed on the period of producer runnable, so this analysis is based on the class 1 and class 2-series 1. The tool gives the sent data rate isolated by periods of producer runnables information shown in Figure 74 for application EB-Mivie and Figure 75 for application TDP. Figure 76 and Figure 77 give the results of sent data rate accumulated by periods of producer runnables for both applications.

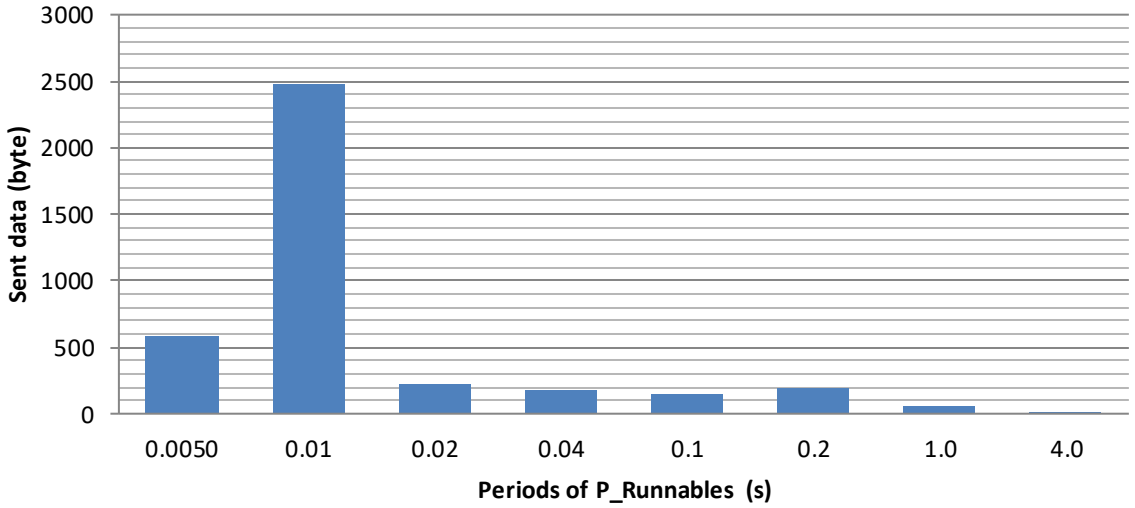


Figure 74-Sent data rate isolated by period of producer runnables (EB-Mivie).

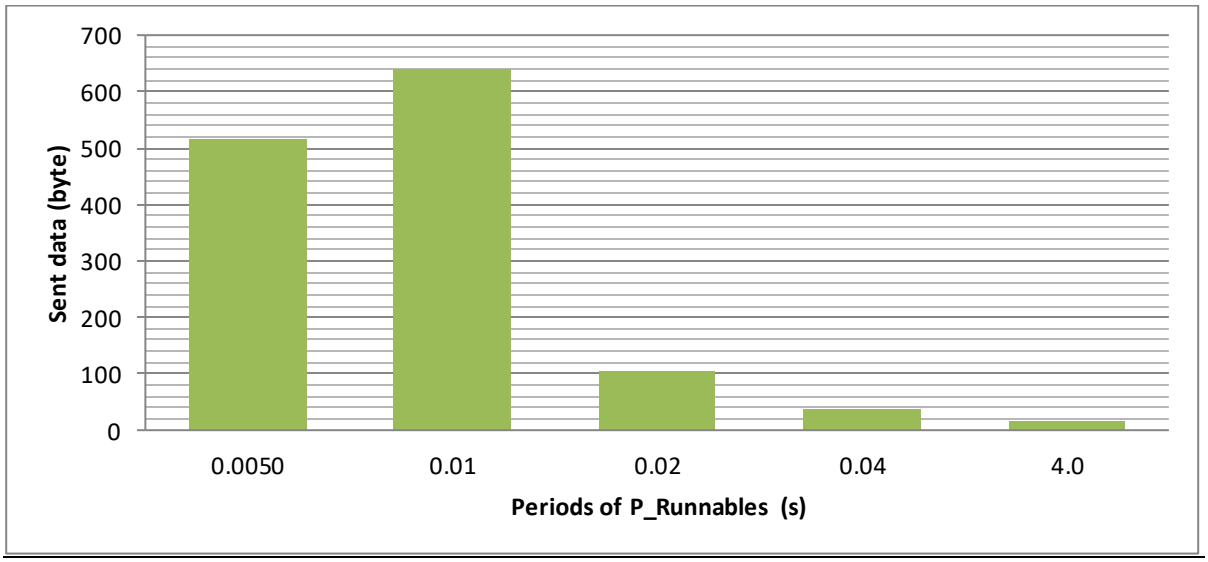


Figure 75-Sent data rate isolated by period of producer runnables (TDP).

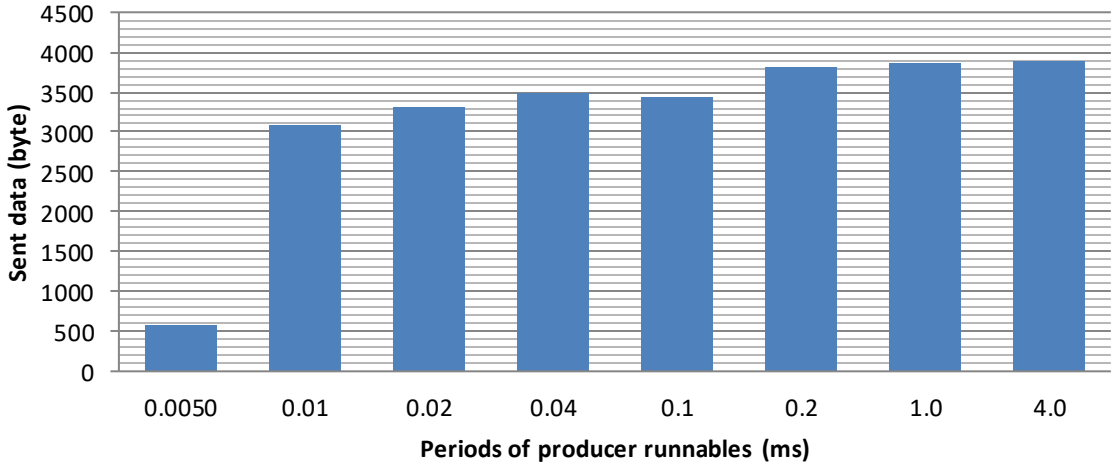


Figure 76-Sent data rate accumulated by period (EB-Mivie)

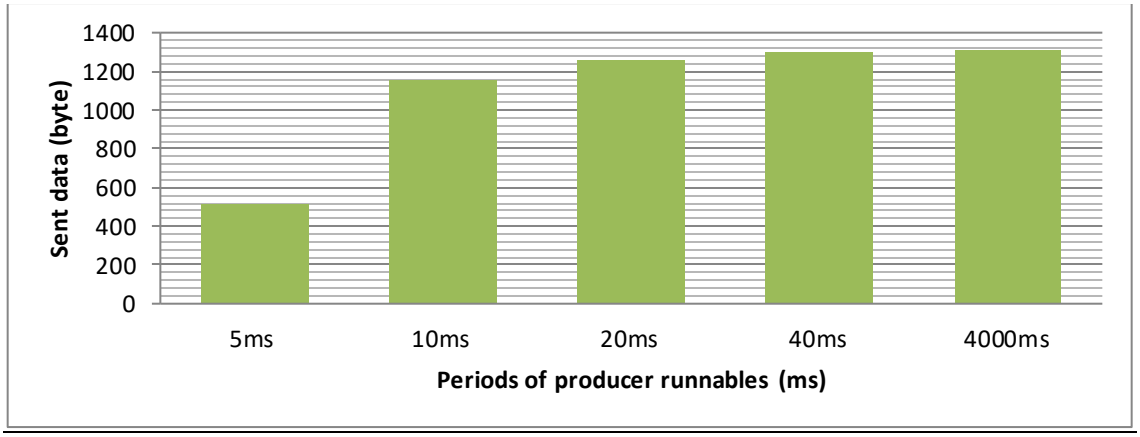


Figure 77-Sent data rate accumulated by period (TDP)

Received data rate

Similar to the sent data rate, the received data rate is relayed on the period of consumer runnable, so this analysis is based on the class 1 and class 2-series 2. The tool gives the received data rate isolated by periods of consumer runnables information shown in Figure 78 for application EB-Mivie and Figure 79 for application TDP. From where we can notice that, the period of 10ms result in a high frequented accessing data. Figure 80 and Figure 81 give the received of sent data rate accumulated by periods of consumer runnables for both applications.

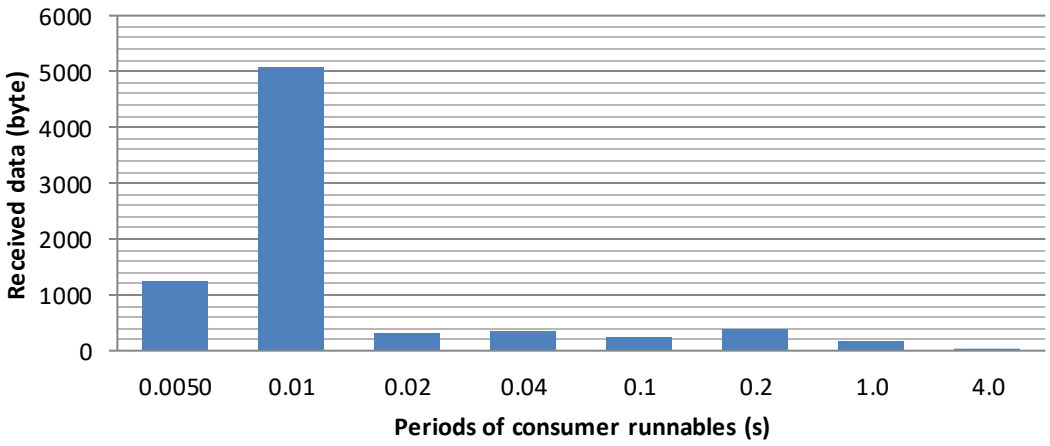


Figure 78-Received data rate isolated by period of producer runnables (EB-Mivie).

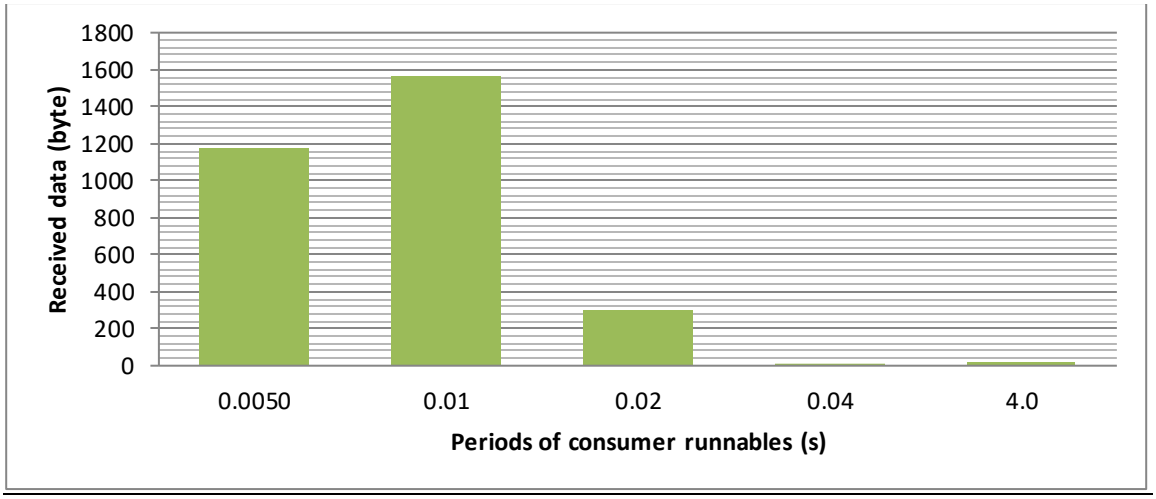


Figure 79-Received data rate isolated by period of producer runnables (TDP).

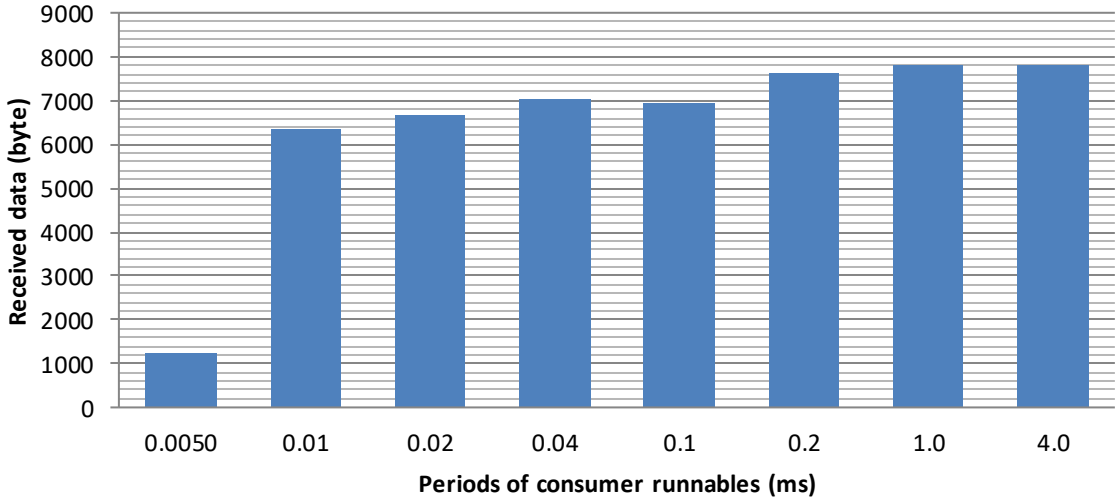


Figure 80-Received data rate accumulated by period (EB-Mivie).

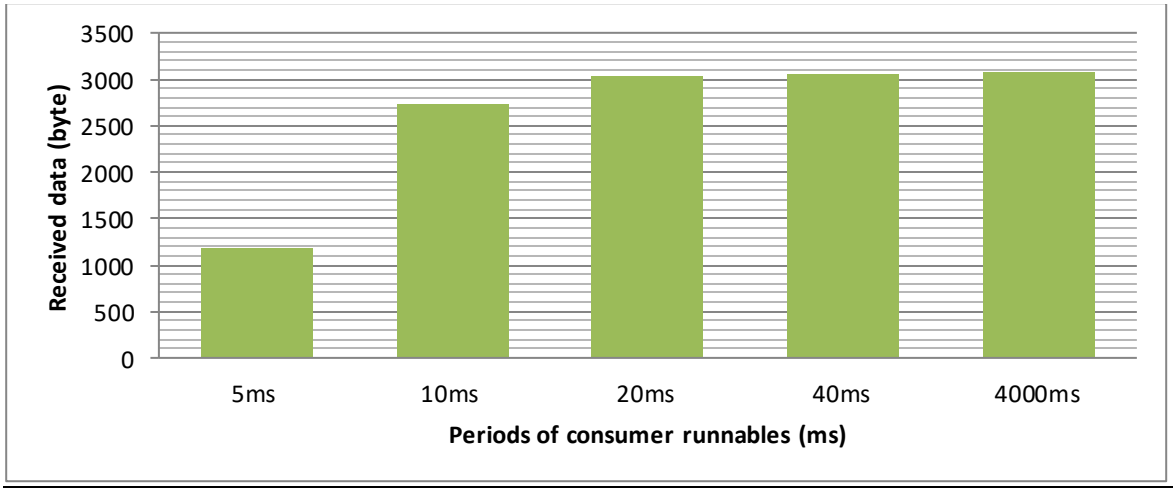


Figure 81-Received data rate accumulated by period (TDP).

Conclusion for data rate analysis

For the application EB-Mivie, the data rate in the period of 10ms for both production and consummation is much greater than other periods.

For the application TDP, the data rate in the period of 10ms and 5ms for both production and consummation is much greater than other periods.

When allocating the SW, the transitions with a high data rate shall be considered as strong connection.

Data Unit

The information of physical unit is in the a2L file. To obtain this information, the tool reads this file by the Perl scripting. The result of physical unit for each data is show in the Table 24 .

Unit	Count		Designation	Variability (Fast/Slow/Depend on data)
	EBDT	TDP		
Without unit	155	49	Without unit	Depend on data
kW	1		Power	Slow
g/mol	1	1		Slow
1/s	2	2		Fast
kg/s	29	49		Fast
RPM/s	4		Revolutions per minute	Fast
kg	24	31	Mass	Fast
s.kg/Pa	1	1		Fast
m²	14	6	Surface	Depend on data
kg/s/Pa	1	1		Depend on data
Pa	77	114	Pressure	Depend on data
N.m	142		Moment	Fast
%	31		Percentage	Depend on data
.	218	26		
(K)^1/2	1	3		
-	305	104		
m/s2	1		Acceleration	Fast
°Vil	2			Fast
mOhm	1		Resistance	Slow
°	1			Depend on data
K^(1/2)	1	1		Depend on data
1/Pa	4	6		Depend on data
km/h	6		Speed	Slow

A	7		Current	Depend on data
s.u.	1			
J	1		Inertia	Slow
K	43	61		Depend on data
Nm	10		Moment	Fast
W	3		Puissance	Fast
V	16		Tension	Slow
mg	3		Masse	Fast
_	4			
m^2	3	3	Surface	Depend on data
°C	9		Temperature	Depend on data
RPM.N.m/s	1			Depend on data
s/m.K^(1/2)	1	1		Depend on data
km/h/1000RPM	1			Slow
°/s	1			Depend on data
Pa/s	1	1		Depend on data
m	1		Distance	Fast
V/s	8			Slow
s	45	11	Temps	Fast
m/s^2	23			Depend on data
°Ck	17	14		Fast
RPM	22	4	Tours	Fast
m/s^3	1			Depend on data
bool	6			Slow
N.m/s	6			Depend on data
kg/h	2			Slow
Etat (énuméré)		2		
Booléen ou énuméré(état)		1		
Compteur (entier)		1		

Table 24-Physical unit of data

The sequences

The result of sequences for both granularities is summarized in Table 25.

Granularity	Number of sequences	The max length of sequences	Ratio of the application
SWCs	512	41	41/68
Runnables	2638	~110	110/588

Table 25-sequences results

The sequence with the max length is the principle sequence and the ratio of the application is the max length on the application.

Data rate analysis between SWC chains in application TDP

The tool analyzes the application at the granularity of SWCs. There are two types of chains for SWCs: chain of advance and chain of air. The communication between components cross the two chains is shown in Figure 82.

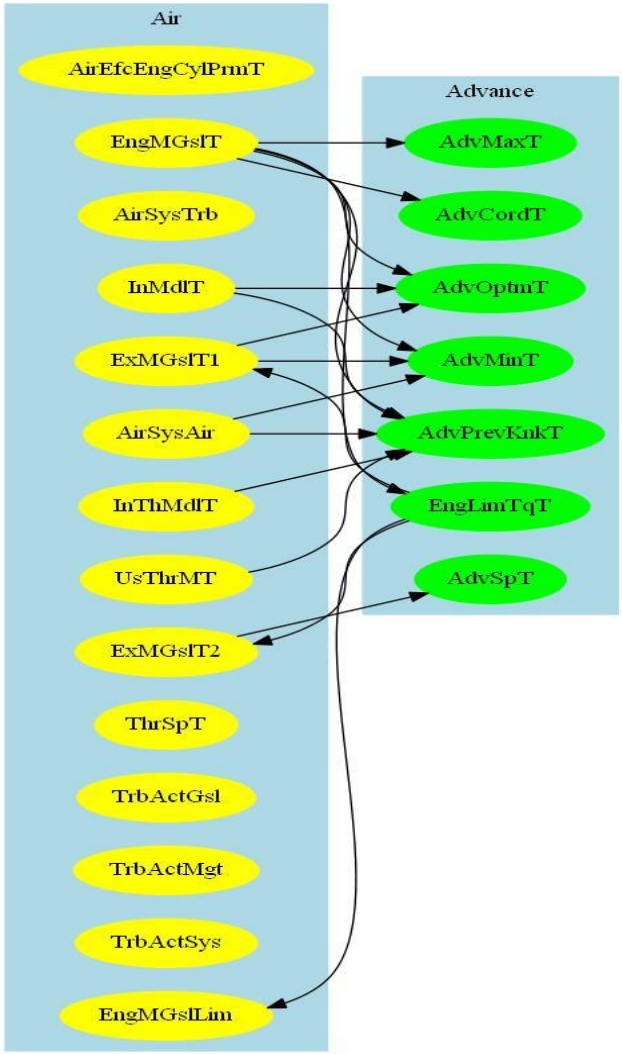


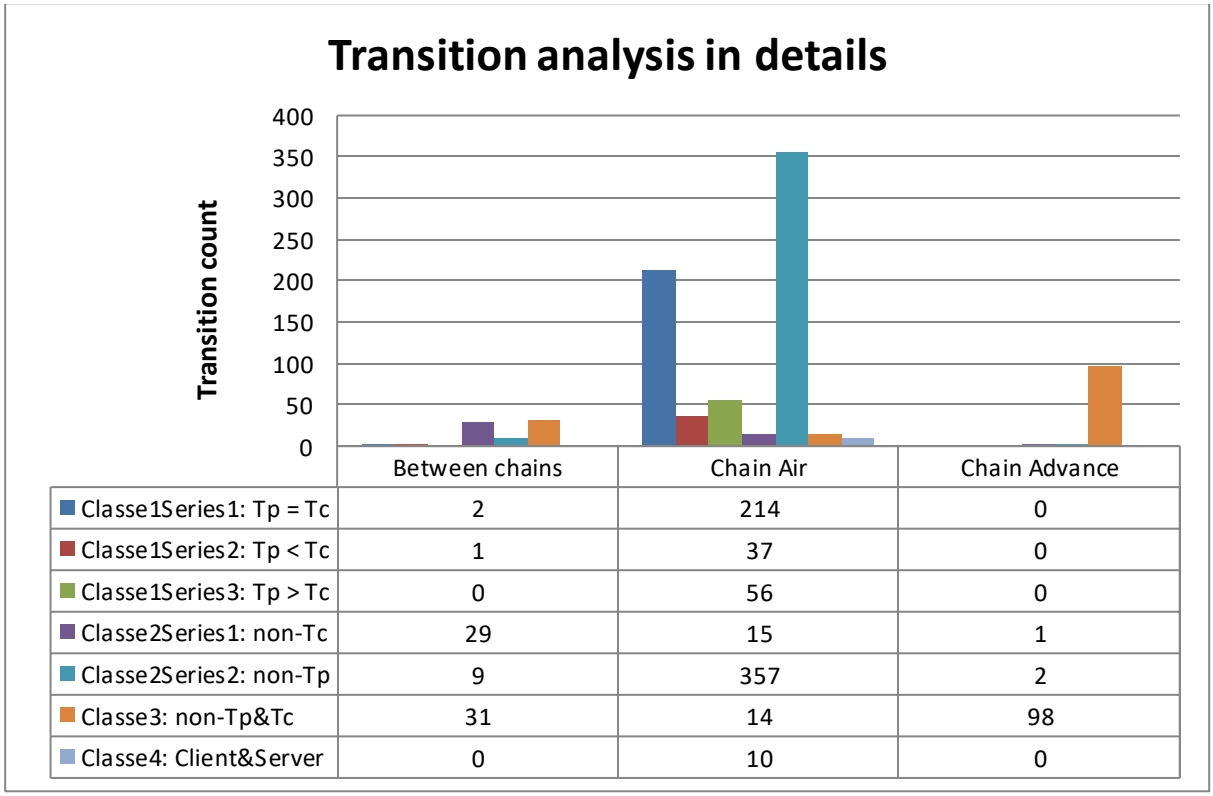
Figure 82-Communications between chains

The transitions involved in these two chains occupies about 95% of transition inter SWC of the application, as shown in Table 26.

Classes		Application TDP			
		Transition Count			
		2 Chains	Inter-SWC	Intra-SWC	Total
Class1	Series1	216	216	63	279
	Series2	38	38	33	71
	Series3	56	56	14	70
Class2	Series1	45	45	26	71
	Series2	368	368	303	671
Class3		143	143	198	341
Class4		10	55		55
Total		876	921	637	1558

Table 26-Transitions analysis in two chains

Figure 83 give the distribution of the transitions involved in two chains in the terms of classifications.



	Between chains	Chain Air	Chain Advance
Classe1Series1: Tp = Tc	2	214	0
Classe1Series2: Tp < Tc	1	37	0
Classe1Series3: Tp > Tc	0	56	0
Classe2Series1: non-Tc	29	15	1
Classe2Series2: non-Tp	9	357	2
Classe3: non-Tp&Tc	31	14	98
Classe4: Client&Server	0	10	0

Figure 83-Distribution of the transitions in two chains

From Figure 83, we can notice that the SWCs in chains air are strongly connected by class1 and class2 series 2, where 94.8% of transitions in the classe2 series2 are MSE-TEV connection.

Table 27 and Table 28 synthesize the sent data rate of the transition across the two chains presented in the Figure 82.

Air		Advance	Period	Data (data in green color mean accessed by DRE)	Data type	Count	Size (Byte)

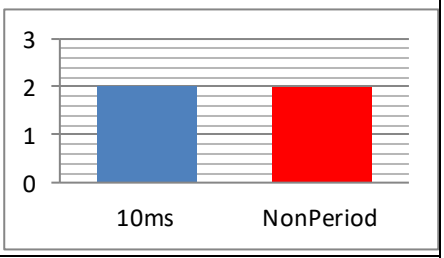
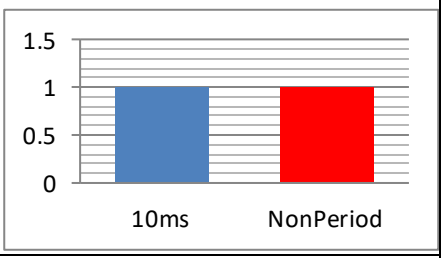
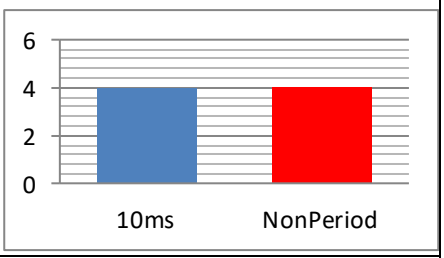
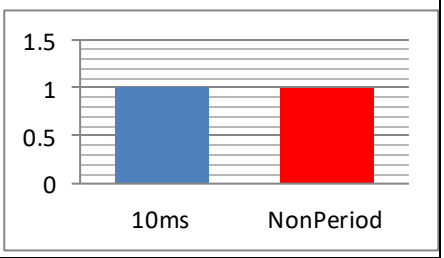
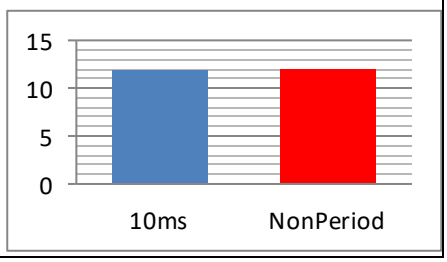
AirSysAir	➔	AdvPrevKnkT	10ms	AirSys_rAirLdReq	UInt16	1	2
			Non-Period	AirSys_rAirLdReq	UInt16	1	2
AirSysAir	➔	AdvMinT	10ms	AirSys_bActStraLimSurge	Boolean	1	1
			Non-Period	AirSys_bActStraLimSurge	Boolean	1	1
EngMGslT	➔	AdvMinT	10ms	EngM_rAirLdCor	UInt16	2	4
			Non-Period	EngM_rAirLdCor	UInt16	2	4
EngMGslT	➔	AdvMaxT	10ms	EngM_rAirLdCor	UInt16	1	1
			Non-Period	EngM_rAirLdCor	UInt16	1	1
EngMGslT	➔	AdvOptmT	10ms	EngM_rAirLdCor	UInt16	2	12
				EngM_mBurnCor	UInt16	1	
				EngM_mAirCor	UInt16	1	
				EngM_tMixtCylCor	UInt16	1	
				EngM_rItBurnRateCor	UInt16	1	
			Non-Period	EngM_rAirLdCor	UInt16	2	12
				EngM_mBurnCor	UInt16	1	
				EngM_mAirCor	UInt16	1	
				EngM_tMixtCylCor	UInt16	1	
				EngM_rItBurnRateCor	UInt16	1	
EngMGslT	➔	EngLimTqT	5ms	EngM_rAirLdPred	UInt16	1	2
			10ms	EngM_mAirEngCylMax	UInt32	1	20
				EngM_mAirPresUsThr	UInt32	1	

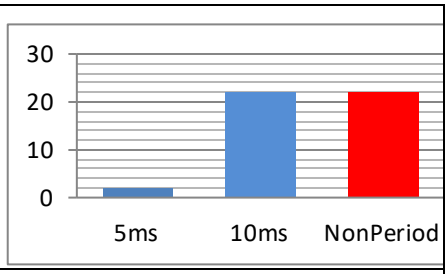
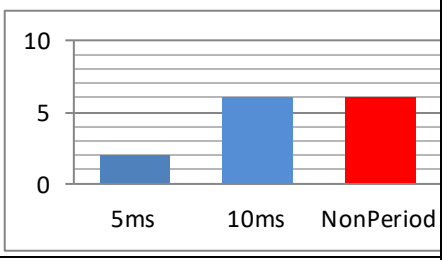
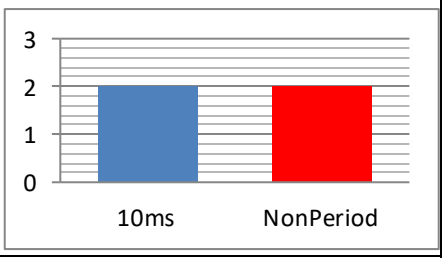
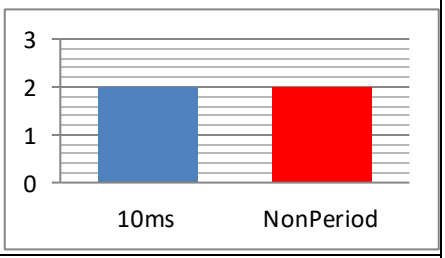
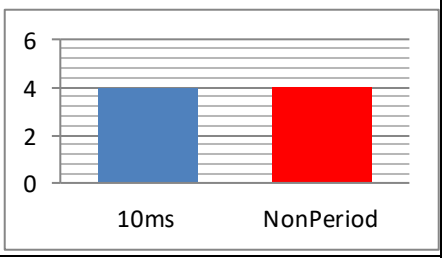
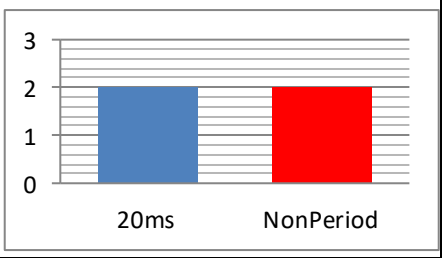
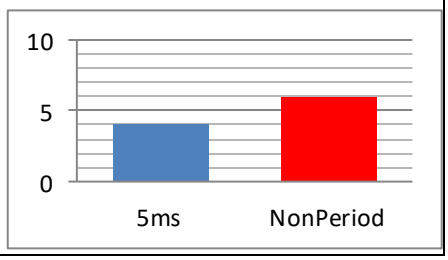
				EngM_rAirLdCor	UInt16	1	
				EngM_mAirCor	UInt16	1	
				EngM_mAirEngCylTrbMax	UInt32	1	
				EngM_mAirEngCylMin	UInt32	1	
			Non-Period	EngM_rAirLdPred	UInt16	1	
				EngM_mAirEngCylMax	UInt32	1	
				EngM_mAirPresUsThr	UInt32	1	
				EngM_rAirLdCor	UInt16	1	22
				EngM_mAirCor	UInt16	1	
				EngM_mAirEngCylTrbMax	UInt32	1	
				EngM_mAirEngCylMin	UInt32	1	
EngMGslT	➔	AdvPrevKnkT	5ms	EngM_rAirLdPred	UInt16	1	2
			10ms	EngM_rAirLdCor	UInt16	1	6
				EngM_rMaxTotLd	UInt16	1	
			Non-Period	EngM_rAirLdPred	UInt16	1	
				EngM_rAirLdCor	UInt16	1	6
				EngM_rMaxTotLd	UInt16	1	
EngMGslT	➔	AdvCordT	10ms	EngM_rAirLdCor	UInt16	1	2
			Non-Period	EngM_rAirLdCor	UInt16	1	2

ExMGslT1	➔	AdvMinT	10ms	ExM_tExDyn	UInt16	1	2
			Non-Period	ExM_tExDyn	UInt16	1	2
ExMGslT1	➔	AdvOptmT	10ms	ExM_molMassInMixt	UInt16	2	4
			Non-Period	ExM_gmaInMixt	UInt16	2	4
ExMGslT2	➔	AdvSpT	20ms	ExM_tUsMainOxCEstim	UInt16	1	2
			Non-Period	ExM_tUsMainOxCEstim	UInt16	1	2
InMdlT	➔	AdvPrevKnkT	5ms	InM_pDsThrCor	UInt16	1	4
				InM_concEGREstim	UInt16	1	
			Non-Period	InM_pDsThrCor	UInt16	1	6
				InM_concEGREstim	UInt16	1	
				InM_concEGREstim	UInt16	1	
InMdlT	➔	AdvOptmT	5ms	InM_mEGREstim	UInt32	1	4
			Non-Period	InM_mEGREstim	UInt32	1	8
				InM_mEGREstim	UInt32	1	
InThMdlT	➔	AdvPrevKnkT	10ms	InThM_tAirUsInVlvEstim	UInt16	1	2
			Non-Period	InThM_tAirUsInVlvEstim	UInt16	1	2
UsThrMT	➔	AdvPrevKnkT	10ms	UsThrM_pAirExt	UInt16	1	2
			Non-Period	UsThrM_pAirExt	UInt16	1	2
EngMGslLim	➔	EngLimTqT	Non-Period	IgSys_rMaxIgEfc	UInt16	1	4
				IgSys_rMaxIgEfc	UInt16	1	

ExMGslT2	←	EngLimTqT	Non-Period	IgSys_rDynIgSpEfc	UInt16	1	4
				IgSys_rDynIgSpEfc	UInt16	1	
ExMGslT1	←	EngLimTqT	10ms	IgSys_lamClc	UInt32	1	4
			Non-Period	IgSys_lamClc	UInt32	1	8
				IgSys_rDynIgSpEfc	UInt16	1	
				IgSys_rDynIgSpEfc	UInt16	1	

Table 27-data rate information of communications between chains for application TDP

Air		Advance	Period	Size (Byte)	Figure (y:size; x: non period for red)
AirSysAir	→	AdvPrevKnkT	10ms	2	
			Non-Period	2	
AirSysAir	→	AdvMinT	10ms	1	
			Non-Period	1	
EngMGslT	→	AdvMinT	10ms	4	
			Non-Period	4	
EngMGslT	→	AdvMaxT	10ms	1	
			Non-Period	1	
EngMGslT	→	AdvOptmT	10ms	12	
			Non-Period	12	

EngMGslT	→	EngLimTqT	5ms	2	
			10ms	22	
			Non-Period	22	
EngMGslT	→	AdvPrevKnkT	5ms	2	
			10ms	6	
			Non-Period	6	
EngMGslT	→	AdvCordT	10ms	2	
			Non-Period	2	
ExMGslT1	→	AdvMinT	10ms	2	
			Non-Period	2	
ExMGslT1	→	AdvOptmT	10ms	4	
			Non-Period	4	
ExMGslT2	→	AdvSpT	20ms	2	
			Non-Period	2	
InMdlT	→	AdvPrevKnkT	5ms	4	
			Non-Period	6	

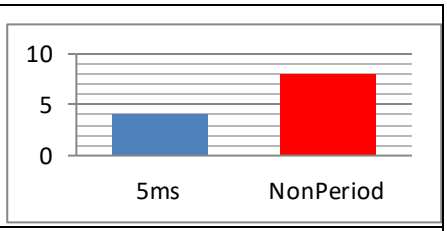
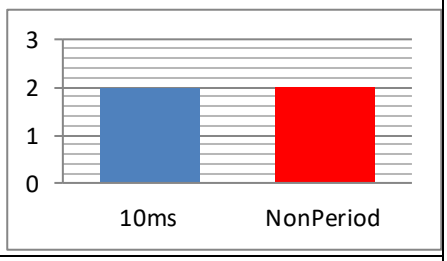
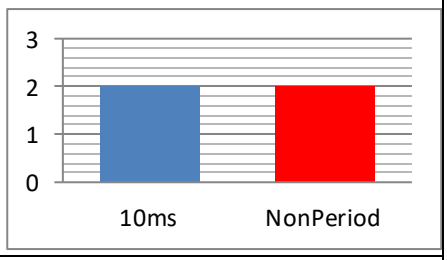
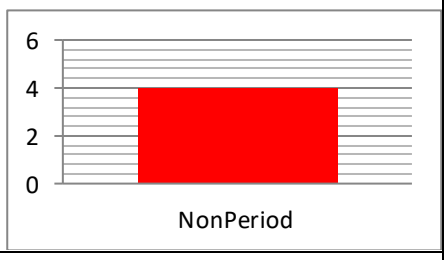
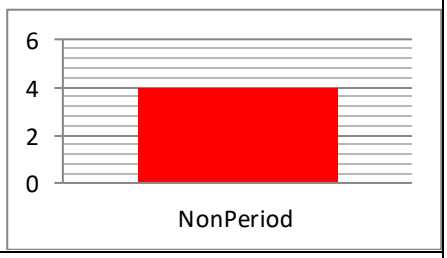
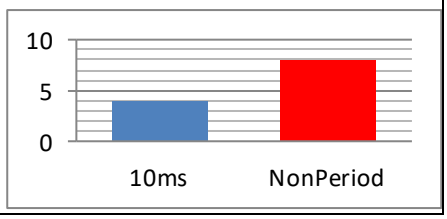
InMdlT	→	AdvOptmT	5ms	4	
			Non-Period	8	
InThMdlT	→	AdvPrevKnkT	10ms	2	
			Non-Period	2	
UsThrMT	→	AdvPrevKnkT	10ms	2	
			Non-Period	2	
EngMGslLim	←	EngLimTqT	Non-Period	4	
ExMGslT2	←	EngLimTqT	Non-Period	4	
ExMGslT1	←	EngLimTqT	10ms	4	
			Non-Period	8	

Table 28-data rate information of communications between chains for application TDP

Publications

W. Wang, *B. Miramond, F. Camut* ***Generation of Schedule Tables on Multi-core Systems for AUTOSAR Applications*** *Conference on Design and Architectures for Signal and Image Processing, DASIP 2016, Rennes, France, October 12-14th, 2016*

W. Wang, *S. Cotard, F. Gravez, Y. Chambrin, B. Miramond.* ***Optimizing Application Distribution on Multi-Core Systems within AUTOSAR*** *Embedded Real-Time Software and Systems, ERTS² 2016, Toulouse, France, January 27-29th, 2016*

W. Wang, *B. Miramond, S. Cotard, F. Gravez, Y. Chambrin.* ***Distribution of Real-Time Software on Multi-Core Architectures in Automotive Systems*** *Conférence d'informatique en Parallélisme, Architecture et Système, Compas' 2016, Lorient, France, July 5 – 8th, 2016*

W. Wang, *B. Miramond, F. Camut* ***(Poster) Distribution of Real-Time Software on Multi-Core Architectures in Automotive Systems*** *Groupement de Recherche SoC-SiP (GDR SoCSiP), Nantes, France, Juin 8-10th, 2016*

S. Cotard, **W. Wang**, *F. Camut, B. Miramond,* ***Procédé hors ligne d'allocation d'un logiciel embarqué temps réel sur une architecture multi-cœur, et son utilisation pour des applications embarquées dans un véhicule automobile*** Patent in soumission: ***MFR9019 – ID N° 3713***

Bibliography

AMALTHEA. (2012). Retrieved from http://www.amalthea-project.org/

Anderson, J. H. (2000). Pfair scheduling: Beyond periodic task systems. *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications*, (pp. 297-306).

Andersson, & Jonsson, J. (2003). The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. *Proceedings of the 15th Euromicro Conference on Real-Time Systems, ECRT'03.*

Andersson, B., & Jonsson, J. (2000). Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. *Proceedings of 7th Real-Time Computing Systems and Applications.*

Andersson, B., Baruah, S., & Jonsson, J. (2001). Static-Priority Scheduling on Multiprocessors. *Proceedings of the 22nd IEEE Real-Time Systems Symposium RTSS '01.*

Artop. (2017). Retrieved from https://www.artop.org

AUTOSAR. (2014a). *Specification of Operating System, Release 4.2.2.*

AUTOSAR. (2014b). *Guide to BSW Distribution, Release 4.2.1.*

AUTOSAR. (2017). *Technical Overview*. Retrieved from AUTOSAR: https://www.autosar.org/about/technical-overview/

AUTOSAR Builder. (2017). Retrieved from http://www.3ds.com/products-services/catia/products/autosarbuilder/

Azencott, R. (1992). Simulated annealing: speed of convergence and acceleration techniques. In R. Azencott, *Simulated annealing : Parallelization techniques* (pp. 1-10). Intersciences,Wiley.

Baker, T. P. (2003). Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. *Proceedings of the 24th IEEE International Real-Time Systems Symposium RTSS '03*, (p. 120).

Baker, T. P., & Baruah, S. K. (2006). Schedulability Analysis of Multiprocessor Sporadic Task Systems. In I. Lee, J. Y.-T. Leung, & S. H. Son, *Handbook of Real-Time and Embedded Systems.* Chapman & Hall/CRC.

Bamakhrama, M., & Stefanov, T. (2011). Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. *Proceedings of the ninth ACM international conference on Embedded software EMSOFT '11*, (pp. 195-204).

Baruah, S. (2007). Techniques for Multiprocessor Global Schedulability Analysis. *Proceedings of the 28th IEEE International Real-Time Systems RTSS '07*, (pp. 119-128). Washington, DC.

Baruah, S. K. (2004). Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors. *IEEE Transactions on Computers*, 781-784.

Bekooij, M., Hoes, R., Moreira, O., Poplavko, P., Pastrnak, M., Mesman, B., . . . Meerbergen, J. (2005). Dataflow analysis for real-time embedded multiprocessor system design. *Dynamic and Robust Streaming between Connected Devices*.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 268-308.

Buttazzo, G. C. (2003). Rate Monotonic vs. EDF: Judgment Day. *Proceedings Third International Conference, EMSOFT*, (pp. 67-83). Philadelphia.

Calandrino, J. M., Anderson, J. H., & Baumberger, D. P. (2007). A Hybrid Real-Time Scheduling Approach for Large-Scale Multicore Platforms. *Proceedings of the 19th Euromicro Conference on Real-Time Systems ECRTS '07*, (pp. 247-258).

Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., & Baruah, S. (2004). A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models.* Chapman Hall/CRC, Boca.

Cheng, S.-C., Stankovic, J.-A., & Ramamritham, K. (1989). Scheduling algorithms for hard real-time systems: a brief survey. In *Tutorial: hard real-time systems* (pp. 150-173). Los Alamitos, CA, USA: IEEE Computer Society Press.

Chetto, H., Silly, M., & Bouchentouf, T. (1990). Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 181-194.

COTTET, F. (2000). *Ordonnancement temps réel : cours et exercices corrigés.* Hermès.

Danne, K., & Platzner, M. (2006). An EDF schedulability test for periodic tasks on reconfigurable hardware devices. *Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems LCTES '06*, (pp. 93-102).

Davari, S., & Dhall, S. K. (1986). An on line algorithm for real-time allocation. *Proceedings of IEEE Real-Time Systems Symposium*, (pp. 194-200).

Davis, R. I., & Burns, A. (2011). A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*.

Demetrescu, C., & Finocchi, I. (2003). Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 129-136.

Dhall, S. K., & Liu, C. L. (1978). On a Real-Time Scheduling Problem. *Operations Research*, 127-140.

Dragomir Milojevic, J. G. (2012). U-EDF: An Unfair But Optimal Multiprocessor Scheduling Algorithm for Sporadic Tasks. *24th Euromicro Conference on Real-Time Systems (ECRTS 2012)*, (pp. 13-23).

E. G. Coffman, J., Garey, M. R., & Johnson, D. S. (1996). Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems* (pp. 46 - 93). Boston: PWS Publishing Co.

EMC². (2014). Retrieved from http://www.artemis-emc2.eu/

Faragardi, H. R., Lisper, B., & Nolte, T. (2013). Towards a communication-efficient mapping of AUTOSAR runnables on multi-cores. *IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), 2013.*

Fisher, N., Goossens, J., & Baruah, S. (2010). Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real-Time Systems*, 26-71.

Fohler, G. (1992). Realizing Changes of Operational Modes with a Pre Run-Time Scheduled Hard Real-Time System. In *Responsive Computer Systems* (pp. 287-300).

Forget, J., & Frédéric Boniol, E. G. (2010). Scheduling Dependent Periodic Tasks Without Synchronization Mechanisms. *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, (pp. 301-310). Stockholm.

G. Nelissen, V. B. (2011). Reducing preemtions and migrations in real-time multiprocessor scheduling lgorithms by releasing the fairness. *RTCSA'11*, (pp. 15-24).

G.Georgia, Stoimenov, N., Huang, P., & Thiele, L. (2013). Scheduling of mixed-criticality applications on resource-sharing multicore systems. *Proceedings of the Eleventh ACM International Conference on Embedded Software EMSOFT'13.*

Gai, P., Lipari, G., & Natale, M. D. (2001). Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip. *Proceedings of the 22nd IEEE Real-Time Systems Symposium RTSS'01*, (p. 73).

Gantel, L., Khiar, A., Miramond, B., Benkhelifa, M. E., Kessal, L., Lemonnier, F., & Rhun, J. L. (2012). Enhancing reconfigurable platforms programmability for synchronous data-flow applications. *Transactions on Reconfigurable Technology and Systems*.

Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman & Co.

Geman, S., & Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6*, 721-741.

Giannopoulou, G., Stoimenov, N., Huang, P., & Thiele, L. (2014). Mapping mixed-criticality applications on multi-core architectures. *Proceedings of the conference on Design, Automation & Test in Europe DATE '14*, (pp. 1-6). Leuven.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research - Special issue: Applications of integer programming, 13*, 533-549.

Glover, F., & Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston: Addison-Wesley Longman Publishing.

Goossens, J., Funk, S., & Baruah, S. (2003). Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems*, 187-205.

Hammett, R. (2002). Flight-critical distributed systems - design considerations. *The 21st Digital Avionics Systems Conference*, (pp. 13B3-1-13B3-8).

Herrtwich, R. G. (1990). *An Introduction to Real-time Scheduling.* RT-90-035 Int. Comp. Science Instit. califonia.

Infineon. (2012). *TC27x 32-Bit Single-Chip Microcontroller, User's Manual.*

Infineon. (2017). *TriCore Microcontroller*. Retrieved from http://www.infineon.com/

Jung, K. J., & Park, C. (2005). A technique to reduce preemption overhead in real-time multiprocessor task scheduling. *Proceedings of the 10th Asia-Pacific conference on Advances in Computer Systems Architecture ACSAC'05*, (pp. 566-579). Heidelberg.

Karp, R. M. (1972). Reducibility among Combinatorial Problems. In *Complexity of Computer Computations* (pp. 85-103).

Kato, S., & Yamasaki, N. (2008). Portioned EDF-based scheduling on multiprocessors. *Proceedings of the 8th ACM international conference on Embedded software EMSOFT'08*, (pp. 139-148). New York.

Kehr, S., Panić, M., Quiñones, E., Böddeker, B., Sandoval, J. B., Abella, J., . . . Schäfer, G. (2016). Supertask: Maximizing runnable-level parallelism in AUTOSAR applications. *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, (pp. 25-30). Dresden.

Kehr, S., Quiñones, E., Böddeker, B., & Schäfer, G. (2015). Parallel execution of AUTOSAR legacy applications on multicore ECUs with timed implicit communication. *Proceedings of the 52nd Annual Design Automation Conference DAC '15.*

Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220(4598)*, 533-549.

Klikpo, E. C., Khatib, J., & Munier-Kordon, A. (2016). Modeling Multi-Periodic Simulink Systems by Synchronous Dataflow Graphs. *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, (pp. 1-10).

Kopetz, H. (2011). *Real-Time Systems: Design Principles for Distributed Embedded.* SpringeR.

Krause, M., Bringmann, O., Hergenhan, A., Tabanoglu, G., & Rosentiel, W. (2007). Timing Simulation of Interconnected AUTOSAR Software-Components. *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07.*

Lakshmanan, K., Rajkumar, R., & Lehoczky, J. (2009). Partitioned Fixed-Priority Preemptive Scheduling for Multi-core Processors. *21st Euromicro Conference on Real-Time Systems ECRTS '09.*

Lee, E. A., & Messerschmitt, D. G. (1987). Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*.

Leung, J. Y.-T., & Whitehead, J. (1982). On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 237-250.

Liu, C. L., & Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM (JACM)*, 46-61 .

López, García, M., Díaz, J. L., & García, D. F. (2000). Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. *Proceeding of the 12th Euromicro conference on Real-time systems Euromicro-RTS'00*, (pp. 25-33).

López, J. M., García, M., Díaz, J. L., & García, D. F. (2003). Utilization Bounds for Multiprocessor Rate-Monotonic Scheduling. *Real-Time Systems*, 5-28.

Miramond, B., & Cucu-Grosjean, L. (2010). Generation of static tables in embedded memory with dense scheduling. *Design and Architectures for Signal and Image Processing* .

Miramond, B., & Delosme, J. -M. (2005). Decision guide environment for design space exploration. *10th IEEE Conference on Emerging Technologies and Factory Automation ETFA 2005.*

Miramond, B., & Delosme, J.-M. (2005). Design space exploration for dynamically reconfigurable architectures. *Proceedings Design, Automation and Test in Europe.*

Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* Cambridge: MIT Press.

Mok, A. K. (1983). Fundamental design problems of distributed systems for the hard-real-time environement. *Technical Report*.

Monot, A., Navet, N., & Bavoux, B. (2012). Multisource Software on Multicore Automotive ECUs — Combining Runnable Sequencing With Task Scheduling. *IEEE Transactions on Industrial Electronics* .

Natale, M. D., & Sangiovanni-Vincentelli, A. L. (2010, April). Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools,. *Proceeding of the IEEE, 98*(4), 603-620.

Oh, Y., & Son, S. H. (1993). *Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem.* University of Virginia Charlottesville.

OSEK/VDX. (2005). *OSEK/VDX - Operating system. v2.2.3.*

P. Regnier, G. L. (2011). RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor. *32nd Real-Time Systems Symposium RTSS'11*, (pp. 104-115).

Panic, M., Kehr, S., Quiñones, E., Böddeker, B., Abella, J., & Cazorla, F. J. (2014). RunPar: an allocation algorithm for automotive applications exploiting runnable parallelism in multicores. *ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES).*

Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc. Upper Saddle River, NJ, USA © 1982.

parMERASA. (2011). Retrieved from http://www.parmerasa.eu/

Rajkumar, R. (1990). Real-time synchronization protocols for shared memory multiprocessors . *Proceedings of 10th International Conference on Distributed Computing Systems.*

Sagstetter, F., Andalam, S., Waszecki, P., Lukasiewycz, M., Stähle, H., Chakraborty, S., & Knoll, A. (2014). Schedule Integration Framework for Time-Triggered Automotive Architectures. *Proceedings of the 51st Annual Design Automation Conference DAC'14*, (pp. 1-6). New York.

Saidi, S. E., Cotard, S., Chaaban, K., & Marteil, K. (2015). An ILP approach for mapping AUTOSAR runnables on multi-core architectures. *Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools.*

Sailer, A., Schmidhuber, S., Deubzer, M., Alfranseder, M., Mucha, M., & Mottok, J. (2013). Optimizing the task allocation step for multi-core processors within AUTOSAR. *International Conference on Applied Electronics (AE).*

Shin, I., Easwaran, A., & Lee, I. (2008). Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors . *Proceedings of the 2008 Euromicro Conference on Real-Time Systems ECRTS '08*, (pp. 181-190).

Sprunt, B., Sha, L., & Lehoczky, J. (1989). Aperiodic task scheduling for Hard-Real-Time systems. *Real-Time Systems*, 27-60.

Srinivasan, A., & Baruah, S. (2002). Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 93-98.

SystemDesk. (2017). Retrieved from https://www.dspace.com/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm

Szu, H. (1987). Fast simulated annealing. *AIP Conference Proceedings 151 on Neural Networks for Computing*, (pp. 420-425). Utah.

Tresos. (2017). *EB tresos Studio*. Retrieved from https://www.elektrobit.com/products/ecu/eb-tresos/studio/

Weber, J. (2009). E/E System Development. In J. Weber, *Automotive Development Processes* (pp. 53-78). Springer Berlin Heidelberg.

Woeginger, G. J. (2003). Exact algorithms for np-hard problems: a survey. In *Combinatorial optimization* (pp. 185-207). New York: Springer-Verlag New York, Inc.

Xu, J., & Parnas, D. (1990). Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations. *IEEE Transactions on Software Engineering*.

Yi, Y., Han, W., Zhao, X., Erdogan, A. T., & Arslan, T. (2009). An ILP formulation for task mapping and scheduling on multi-core architectures. *Proceeding DATE '09 Proceedings of the Conference on Design, Automation and Test in Europe*, (pp. 33-38).

Zhang, P., Gao, Y., & Qiu, M. (2015). A Data-Oriented Method for Scheduling Dependent Tasks on High-Density Multi-GPU Systems. *Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications HPCC-CSS-ICESS '15.*