INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team Ostre

Optimization of embedded real-time systems

Rocquencourt

THEME 1C

Activity Report

2003

Table of contents

1. Team

The OSTRE team was initiated beginning 2002 by Yves Sorel, Research Director in the SOSSO project (Theme 4: Applications and Tools of Automatic Control) and scientific leader, until end 2001, of a subset of this team working on the optimized implementation, with real-time constraints, of automatic control algorithms. These works, mainly concerning the AAA/SynDEx methodology began in the early 90's as a need to execute programs, written with the Synchronous Languages (Esterel, Lustre, Signal), on distributed architectures while satisfying, not only logical but also physical real-time constraints, and the need to provide hardware/software co-design methodologies for embedded distributed architectures, such as SoC, emerging in the fields of robotics, avionic, automobile and telecommunications.

Head of project-team
Yves Sorel [Research Director Inria]

Administrative Assistant
Martine Verneuille [Inria]

Technical Staff
Frédéric Gager [Project technical staff]
Julien Forget [Project technical staff]
Arnaud Rouanet [Software development staff, from 2003/10/01]

Ph. D. Student
Liliana Cucu [Scolarship Inria]
Nicolas Pernet [Scolarship Inria]
Hamoudi Kalla [Scolarship Inria BIP/OSTRE]
Linda Kaouane [Scolarship ESIEE/OSTRE]
Mickaël Raulet [Scolarship INSA/MITSUBISHI ELECTRIC ITE/OSTRE]

Graduate Student Intern
Cyril Faure [Université Bordeaux 1]
Nasreddine Hireche [Université Paris 11]
Cédric Quentin [ESIEE Noisy-Le-Grand]
Olivier Marchetti [Université Paris 6]
Nikom Suvonvorn [Université Paris 11]

Research Scientist (partner)
Rémy Kocik [Assistant Professor ESIEE Noisy-Le-Grand]
Thierry Grandpierre [Assistant Professor ESIEE Noisy-Le-Grand]
Christophe Lavarenne [Engineer UBIC]

2. Overall Objectives

We address distributed real-time embedded applications in the field of automotive, avionic, telecommunication, etc, whose complexity, in terms of functionalities and hardware specifications, and in terms of real-time and embedding constraints, requires design methodologies and associated tools. We propose a methodology, called AAA (algorithm-architecture adequation), that is mathematically founded allowing to specify "application algorithms" (functionalities) and redundant "multicomponent architectures" (composed of processors and specific integrated circuits all together interconnected) with graph models. Consequently, all the possible implementations of a given algorithm onto a given architecture is described in terms of graphs transformations. An implementation consists in distributing and scheduling a given algorithm onto a given architecture. Adequation amounts to chose one implementation among all the possible ones, such that the real-time and embedding constraints are satisfied and the hardware redundancy is fully used. Furthermore, from the adequation results our graph models allow to generate automatically, as an ultimate graphs transformation, two types of codes: dedicated distributed real-time executives or configuration of standard distributed real-time executives

(RTlinux, OSEK, etc) for processors, and net-lists (structural VHDL) for specific integrated circuits. Finally fault tolerance is of great concern because the applications we are dealing with are often critical, that is to say, may lead to catastrophic consequences when they fail. The AAA methodology provides a mathematical framework for rapid prototyping and hardware/software co-design taking into account fault tolerance.

From the optimization point of view, real-time systems are, first of all, “reactive systems” which mandatorily must react to each input event of the infinite sequence of events it consumes, such that “cadence” and “latency” constraints are satisfied. The latency corresponds to the delay between an input event consumed by the system and an output event produced by the system in reaction to this input event. The cadence corresponds to the delay between two successive input events, i.e. a period. The term event is used in a broad sense, it may refers to a periodic or to an aperiodic discrete (sampled) signal. When hard (critical) real-time is considered, off-line approaches are preferred due to their predictability and best performances, and when on-line approaches are unavoidable, mainly to take into account aperiodic events, we intend to minimize the decisions taken during the real-time execution. When soft real-time is considered off-line and on-line approaches are mixed. The application domains we are involved in, e.g. automobile, avionic, lead to consider scheduling problems for systems of tasks with precedence, latency and periodicity constraints. We seek optimal results in the mono-processor case where distribution is not considered, and sub-optimal results through heuristics in the multiprocessor case, because the problems are NP-hard due to distribution consideration.

In order to summarize, we are interested in the optimization of distributed real-time embedded systems according to four research topics:

1. models for specifying, with graphs and partial orders, application algorithm, hardware architecture, and optimized implementation,
2. implementation optimization:
   - real-time scheduling algorithms in the case of mono-processor,
   - real-time distribution and scheduling heuristics in the case of multiprocessor,
   - heuristics for resources minimization in the case of multiprocessor and specific integrated circuit,
3. automatic code generation for processor (dedicated or standard RTOS configuration) and for specific integrated circuit (net-list),
4. fault tolerance.

Beside these researches, we propose a tool implementing the AAA methodology. It is a system level CAD software called SynDEx (http://www.syndex.org). This software, coupled with a high level specification language, like one of the Synchronous Languages or Scicos, leads to a seamless environment allowing to perform rapid prototyping and hardware/software co-design while reducing drastically the development cycle duration and providing safe design.

3. Scientific Foundations

3.1. Introduction

Key words: *Rapid prototyping*, *system level CAD*, *hardware/software co-design*, *partitioning*, *multiprocessor*, *parallel*, *distributed*, *specific integrated circuit*, *real-time*, *embedded*, *graph*, *partial order*, *synchronous languages*, *RTL*, *optimization*, *off-line*, *on-line*, *real-time scheduling*, *real-time operating system*, *executive*, *fault tolerance*.

Participants: Liliana Cucu, Thierry Grandpierre, Rémy Kocik, Christophe Lavarenne, Yves Sorel.

3.2. Context and objectives

Our researches concern the efficient implementation of functional specifications, involving control, signal and image processing algorithms, onto digital electronic systems which may be programmable (processor) or non programmable (specific integrated circuit) for applications in the field of transport (automobile, aircraft, railway), telecommunication, etc [9]. These applications must not only satisfy real-time and embedding constraints, but also be fault tolerant. Such electronic system controls its physical environment by producing a reaction through actuator(s) processed from its internal state and the state of the environment consumed through sensor(s). It is in this sense that they are called "reactive systems" [25]. A mathematical analysis of the automatic control system and of the environment allows to provide on the one hand a functional specification, and on the other hand constraints specification. For example, in the case of real-time constraints, the analysis determines an upper bound for the delay between two successive input sample (period, cadence) of the sensor, and an upper bound for the computation delay between an input sample and an output sample of the actuator, which is the control produced in reaction to this input (latency). If, when these bounds are not satisfied, control is no longer possible and leads to catastrophic consequences, then these systems are called "strict or hard real-time", otherwise they are called "soft real-time". In this latter case only best effort is intended in order to be as close as possible to these constraints. Thus, except if it is explicitly mentioned we will suppose, later on, to be in the first case, that we shall simply call "real-time". In addition to this timing constraints, because they are embedded, these systems must satisfy technological constraints, such as power consumption, weight, volume, memory, etc, leading in general to minimize hardware resources.

In order to satisfy the real-time constraints which usually impose to execute a large amount of computations in a bounded delay, but also in order to locate the computations close to the sensors and the actuators for minimizing the wires, or for the sake of modularity, distributed or parallel architectures are needed. In the most general case they are composed of several programmable components (processors) and several specific integrated circuits (ASIC[1] or FPGA[2]) all together interconnected with possibly different types of communication media. We call such heterogeneous architectures "multicomponent".

The complexity, not only of the algorithms that must be implemented, but also of the hardware architectures, and also the multiple constraints, imply to use methodologies when development cycle time must be minimized from the high level specification until the successive prototypes which ultimately will become a commercial product. In order to avoid gaps between the different steps of the development cycle our AAA methodology is based on a global mathematical framework which allows to specify the application algorithms as well as the hardware architecture with graph models, and the implementation of algorithms onto architectures in terms of graphs transformations. This approach has the benefit on the one hand to insure traceability and consistency between the different steps of the development cycle, and on the other hand to perform formal verifications and optimizations which decrease real-time tests, and also to perform automatic code generation (real-time executives for processors and net-list for specific integrated circuits). All these benefits contribute to minimize the development cycle. Actually, the AAA methodology provides a framework for hardware/software co-design where safe design is achieved by construction, and automatic fault-tolerance is possible only by specifying the components that the user accepts to fail.

3.3. Models

3.3.1. Algorithm

Our algorithm model is an extension of the well known data-flow model from Dennis [26]. It is a directed acyclic hyper-graph (DAG) [24] that we call "conditioned factorized data dependence graph" [11], whose vertices are "operations" and hyper-edges are directed "data or control dependences" between operations. Hyper-edges are necessary in order to model data diffusion since a standard edge only relates a pair of operations. The data dependences defines a partial order on the operations execution [34], called "potential operation-parallelism".

[1]ASIC : Application Specific Integrated Circuit
[2]FPGA : Field Programmable Gate Array

Each operation may be in turn described as a graph allowing a hierarchical specification of an algorithm. Therefore, a graph of operations is also an operation. Operations which are the leaves of the hierarchy are said “atomic” in the sense that it is not possible to distribute each of them on more than one computation resource. The basic data-flow model was extended in three directions, firstly infinite (resp. finite) repetitions in order to take into account the reactive aspect of real-time systems (resp. “potential data-parallelism” similar to loop or iteration in imperative languages), secondly “state” when data dependence are necessary between repetitions introducing cycles which must be avoided by specific vertices called “delays” (similar to z^{-n} in automatic control), thirdly “conditioning” of an operation by a control dependence similar to conditional control structure in imperative languages. Delays combined with conditionings allow to specify FSM (Finite State Machine) necessary for specifying “mode changes”, e.g. some control law is performed when the motor is the state “idle” whereas another one is performed when it is in the state “permanent”. Repetition and conditioning are both based on hierarchy. Indeed, a repeated or “factorized graph of operations” is a hierarchical vertex specified with a “repetition factor” (factorization allows to display only one repetition). Similarly, a “conditioned graph of operations” is a hierarchical vertex containing several alternative operations, such that for each infinite repetition, only one of them is executed, depending on the value carried by the “conditioning input” of this hierarchical vertex. Moreover, the proposed model has the synchronous language semantics [27], i.e. physical time is not taken into account. This means that it is assumed an operation produces its output events and consumes its inputs events simultaneously, and all the input events are simultaneously present. Thus, by transitivity of the execution partial order associated to the algorithm graph, outputs of the algorithm are obtained simultaneously with its inputs. Each input or output carries an infinite sequence of events taking values, which is called a “signal”. Here, the notion of event is general, i.e. signals may be periodic as well as aperiodic. The union of all the signals defines a “logical time”, where physical time elapsing between events are not considered.

3.3.2. Architecture

The typical coarse-grain architecture models such as the PRAM (Parallel Random Access Machines) and the DRAM (Distributed Random Access Machines) [36] are not enough detailed for the optimizations we intend to perform. On the other hand the very fine grain RTL-like (Register Transfer Level) [33] models are too detailed. Thus, our model of multicomponent architecture is also a directed graph [5], whose vertices are of four types: “operator” (computation resource or sequencer of operations), “communicator” (communication resource or sequencer of communications, e.g. DMA), memory resource of type RAM (random access) or SAM (sequential access), “bus/mux/demux/(arbiter)” (choice resource or selection of data from or to a memory) possibly with arbiter (arbitration of memory accesses when the memory is shared by several operators), and whose edges are directed connections. For example, a typical processor is a graph composed of an operator, interconnected with memories (program and data) and communicators, through bus/mux/demux/(arbiter). A “communication medium” is a linear graph composed of memories, communicators, bus/mux/demux/arbiters corresponding to a “route”, i.e. a path in the architecture graph. Like for the algorithm model, the architecture model is hierarchical but specific rules must be carefully observed, e.g. a hierarchical memory vertex may be specified with bus/mux/demux and memories (e.g. several banks), but not with operator. Although this model seems very basic, it is the result of several studies in order to find the appropriate granularity allowing, on the one hand to provide accurate optimization results, and on the other hand to quickly obtain these results during the rapid prototyping phase. Data communications can be precisely modeled through shared memory or through message passing possibly using routes. Furthermore, complex interactions between operators and communicators can be taken into account through bus/mux/demux/arbiter, e.g. when communications with DMA require the sequencer of a processor.

Our model of integrated circuit architecture is the typical RTL model. It is a directed graph whose vertices are of two types: combinatorial circuit executing an instruction, and register storing data used by instructions, and whose edges are data transfers between a combinatorial circuit and a register, and reciprocally.

In order to unify both multicomponent and integrated circuit models we extend the RTL model in a new one called “macro-RTL”. Thus, an operator executes “macro-instructions”, i.e. operations, which consume and produce data in “macro-registers”. This model allows to encapsulate specific details related to the instructions set such as cache, pipe-line and other non deterministic features of processors that are difficult to take into account.

3.3.3. Implementation

An implementation of a given algorithm onto a given multicomponent architecture corresponds to a distribution and a scheduling of, not only the algorithm operations onto the architecture operators, but also a distribution and a scheduling of the data transfers between operations [7].

The distribution consists in distributing each operation of the algorithm graph onto an operator of the architecture graph. This leads to a partition of the operations set, in as much as sub-graphs that there are of operators. Then, for each operation two vertices called “alloc” for allocating program (resp. data) memory must be added, and each of them is allocated to a program (resp. data) RAM connected to the corresponding operator. Moreover, each “inter-operator” data transfer between two operations distributed onto two different operators, is distributed onto a route connecting these two operators. In order to actually perform this data transfer distribution, according to the element composing the route we create and insert as much as “communication operations” that there are of communicators, as much as “identity” vertices that there are of bus/mux/demux, and as much as “alloc” vertices for allocating data to communicate that there are of RAM and SAM. Finally, communication operations, identity and alloc vertices are distributed onto the corresponding vertices of the architecture graph. All the alloc vertices, those for allocating data and program memories as well as those for allocating data to communicate, allow to determine the amount of memory necessary for each processor of the architecture.

The scheduling consists in transforming the partial order of the corresponding subgraph of operations distributed onto an operator, in a total order. This “linearization of the partial order” is necessary because an operator is a sequential machine which executes sequentially the operations. Similarly, it also consists in transforming the partial order of the corresponding subgraph of communications operations distributed onto a communicator, in a total order. Actually, both schedulings amount to add edges, called “precedence dependences” rather than data dependences, to the initial algorithm graph. To summarize, an implementation corresponds to the transformation of the algorithm graph (addition of new vertices and edges to the initial ones) according to the architecture graph.

Finally, the set of all the possible implementations of a given algorithm onto a given architecture may be modeled, in intention, as the composition of three binary relations: namely the “routing”, the “distribution”, and the “scheduling” [14]. Each relation is a mapping between two pairs of graphs (algorithm graph, architecture graph). It also may be seen as a external compositional law, where an architecture graph operates on an algorithm graph in order to give, as a result, a new algorithm graph, which is the initial algorithm graph distributed and scheduled according to the architecture graph. Then the implementation graph is of type algorithm which may in turn be composed with another architecture graph, allowing complex combinations.

The set of all the possible implementations of a given algorithm onto a specific integrated circuit is different because we need a transformation of the algorithm graph into an architecture graph which is directly the implementation graph. This graph is composed of two parts: the data-path obtained by translating each operation in a corresponding logic function, and the control path obtained by translating each control structure in a “control unit”, which is a finite state machine made of counters, multiplexers, demultiplexers and memories, managing repetitions and conditionings [2].

3.4. Optimization

We must choose among all the possible implementations a particular one for which the constraints are satisfied and possibly some criteria are optimized.

In the case of multiprocessor architecture the problem consisting in distributing and scheduling the algorithm onto the architecture such that the execution time of the algorithm is minimum, is known to be of NP-hard complexity [30]. This amounts to consider, in addition to precedences constraints specified through the algorithm graph model, one latency constraint between the first operation(s) (without predecessor) and the last operation(s) (without successor), equal to a unique periodicity constraint (cadence) for all the operations. We propose several heuristics based on the characterization of the operations (resp. communication operations) relatively to the operators (resp. communicators), e.g. execution durations of operations and data transfers, amount of memory, etc, in order to minimize the execution duration of the algorithm graph on the multiprocessor architecture, taking into account communications, possibly concurrent [7]. The characterization amounts to relate the logical time described by the interleaving of events with the physical time. We prefer "greedy heuristics" because they are very fast [31] giving results in a time well suited to rapid prototyping of realistic industrial applications. In this type of applications the algorithm graph may have five thousand vertices and the architecture graph may have some tens of vertices. We also extend these greedy heuristics to iterative versions [13] which are much slower, due to back-tracking, but give better results for the final commercial product.

New applications in the automobile, avionic, or telecommunication domains, lead us to consider more complex constraints. In such applications it is not sufficient to consider the execution duration of the algorithm graph. We need also to consider periodicity constraints for the operations, possibly different, and several latency constraints imposed possibly on whatever pair of operations. Presently there are only partial and simple results for such situations in the multiprocessor case, and only few results in the mono-processor case. Then, we began few years ago to investigate this research area, by interpreting, in our algorithm graph model, the typical scheduling model given by Liu and Leyland [32] for the mono-processor case. This leads us to redefine the notion of periodicity through infinite and finite repetitions of an operations graph (i.e. the algorithm), thus generalizing the SDF (Synchronous Data-Flow) model [28] proposed in the software environment Ptolemy. For simplicity reason and because this is consistent with the application domains we are interested in, we presently only consider that our real-time systems are non-preemptive, and that "strict periodicity" constraints are imposed on operations. In this case we give a schedulability condition for graph of operations with precedence and periodicity constraints in the non-preemptive case. We also formally define the notion of latency which is more powerful [1], for the applications we are interested in, than the usual notion of "deadline" that does not allow to impose directly a timing constraint on a pair of operations, connected by at least one path, like it is necessary for "end-to-end constraints". In order to study schedulability conditions for multiple latency constraints we defined three relations between pair of paths, such that for each pair a latency constraint is imposed on its extremities. Using these relations called **II**, **Z** and **X**, we give a schedulability condition for graph of operations with precedence and latency constraints in the non-preemptive case. Then by combining both previous results we give a schedulability condition for graph of operations with precedence, periodicity and latency constraints in the non-preemptive case, using an important result which gives a relation between periodicity and latency. We also give an optimal scheduling algorithm in the sense that, if there is a schedule the algorithm will find it.

On the other hand, thanks to these results obtained in the mono-processor case, we study extensions of our heuristics for one latency constraint equal to a unique periodicity constraint, in order to solve the distribution and scheduling problem for graph of operations with precedence, periodicity and latency constraints in the multiprocessor case. However, the aforementioned scheduling problems do not take into account aperiodic operations for which there is no off-line solution, at least to our best knowledge, but there are on-line solutions. These aperiodic operations come from aperiodic events, usually related to control. Presently we take them into account off-line by integrating the control-flow in our data-flow model, well suited to distribution, and by maximizing the control effects. We study relations between control-flow and data-flow in order to better exploit their respective advantages. Finally, for soft real-time applications, we study the possibilities in order to mix off-line and on-line approaches in order to take benefit of a better cooperation of control-flow and data-flow.

In the case of integrated circuit the potential parallelism of the algorithm corresponds exactly to the actual parallelism of the circuit. However, this may lead to exceed the required surface of an ASIC or the number of CLB (Combinatorial Logic Block) of a FPGA, and then some operations must be sequentially repeated several times in order to reuse them, reducing in this way the potential parallelism to an actual parallelism with less logic functions. But reducing the surface has a price in terms of time, and also in terms of surface but of a lesser importance, due to the sequentialization itself (instead of parallelism) performed by the finite state machines (control units) necessary to implement the repetitions and the conditionings. Then, we are seeking a compromise taking into account surface and performances. Because these problems are again of NP-hard complexity, we propose greedy and iterative heuristics in order to solve them [2].

Finally, we plan to work on the unification of multiprocessor heuristics and integrated circuit heuristics in order to propose "automatic hardware/software partitioning" for co-design, instead of the usual manual one. The most difficult issue concerns the integration in the cost functions of the notion of "flexibility" which is crucial for the choice of software versus hardware.

3.5. Automatic code generation

As soon as an implementation is chosen among all the possible ones, it is straightforward to automatically generate executable code through an ultimate graphs transformation leading to a distributed real-time executive for the processors, and to a structural hardware description, e.g. synthetizable VHDL, for the specific integrated circuits.

For a multicomponent each operator (resp. each communicator) has to execute the sequence of operations (resp. communication operations) described in the implementation graph. Thus, this graph is translated in an "executive graph" [8] where new vertices and edges are added in order to manage the infinite and finite loops, the conditionings, the inter-operator data dependences corresponding to "read" and "write" when the communication medium is a RAM, or to "send" and "receive" when the communication medium is a SAM. Specific vertices, called "pre" and "suc", which manage semaphores, are added to each read, write, send and receive vertices in order to synchronize the execution of operations and of communication operations when they must share, in mutual exclusion, the same sequencer as well as the same data. These synchronizations insure that the real-time execution will satisfy the partial order specified in the algorithm. Executives generation is proved to be dead-lock free [5] maintaining the properties, in terms of events ordering, shown thanks to the synchronous language semantics. This executive graph is directly transformed in a macro-code [6] which is independent of the processor. This macro-code is macro-processed with "executive kernels" libraries which are dependent of the processors and of the communication media, in order to produce as much as source codes that there are of processors. Each library is written in the best adapted language regarding the processors and the media, e.g. assembler or high level language like C. Finally, each produced source code is compiled in order to obtain distributed executable code satisfying the real-time constraints.

For an integrated circuit, because we associate to each operation and to each control unit an element of a synthetizable VHDL library, the executable code generation relies on the typical synthesis tools of integrated circuit CAD vendors like Synopsis or Cadence.

3.6. Fault tolerance

For the applications we are dealing with, if real-time constraints are not satisfied, this may have catastrophic consequences in terms of human beings lost or pollution, for example. When a fault occurs despite formal verifications which allow safe design by construction, we propose to specify the level of fault the user accepts by adding redundant processors and communication media. Then, we extended our optimization heuristics in order to generate automatically the redundant operations and data dependences necessary to make transparent these faults [3]. As soon as the redundant hardware is fully exploited, "degraded modes" are necessary. They are specified at the level of the algorithm graph using conditionings. Presently, we only take into account "fail silent" faults. They are detected using "watchdogs", the duration of which depends on the operations and data transfers durations. We obtained solid results in the case of processor failures only, i.e. in this case the communication media are assumed error free. We propose two approaches in order to achieve this goal. The first approach is based on spatial redundancy, of operations and data transfers, for point-to-point communication media [4]. The second approach is based on spatial redundancy of operations and temporal redundancy of data transfers for multi-point communication media.

4. Application Domains

Our researches concern academic laboratories as well as industrial companies, both interested by our results in the field of real-time scheduling and in the field of methods for rapid prototyping and co-design. The applications we are involved in are: automobile (XbyWire implying fault tolerance, longitudinal and lateral control), mobile robotics for semi-autonomous public transportation, telecommunications (new generation of mobile or base equipments, video compression JPEG2000 and MPEG4), signal processing (multi-sensor applications in radar and sonar), image processing (automatic guidance).

5. Software

5.1. SynDEx

Participants: Julien Forget, Thierry Grandpierre, Yves Sorel.

SynDEx is a system level CAD software implementing the AAA methodology for rapid prototyping and for optimizing distributed real-time embedded applications. It can be downloaded free of charge under INRIA copyright, see: http://www.syndex.org. It provides the following functionalities:

- specification and verification of an application algorithm as a directed acyclic graph (DAG) of operations, or interface with specification languages such as the synchronous languages providing formal verifications, AIL a language for automobile architectures, Scicos a Simulink-like language, AVS for image processing, CamlFlow a functional data-flow language, etc,
- specification and verification of a “multicomponent” architecture as a graph composed of programmable components (processors) and/or specific non programmable components (ASIC, FPGA), all interconnected through communication media (shared memory, message passing),
- specification of the algorithm characteristics, relative to the hardware components (execution and transfer time, period, memory, etc), and specification of the real-time constraints to satisfy (latencies, periodicities),
- optimization of the algorithm implementation (distribution and scheduling) onto the architecture, and visualization of a timing diagram simulating the real-time behavior,
- generation of dedicated distributed real-time executives, or configuration of general purpose real-time operating systems: RTlinux, Osek, etc. These executives are deadlock free and based on off-line policies. Dedicated executives which induce minimal over-head are built from processor-dependent executive kernels. Presently executives kernels are provided for: ADSP21060, TMS320C40, TMS320C60, i80386, MC68332, MPC555, i80C196 and Unix/Linux workstations. Executive kernels for other processors can be easily ported from the existing ones.

The distribution and scheduling heuristics, as well as the timing diagram, help the user to parallelize his algorithm and to size the hardware while satisfying real-time constraints. Since SynDEx provides a seamless framework from the specification to the distributed real-time execution, formal verifications obtained during the early stage of the specification, are maintained along the whole development cycle. Moreover, since the executives are automatically generated, part of tests and low level hand coding are eliminated, decreasing the development cycle duration.

SynDEx was evaluated by the main companies involved in distributed real-time embedded systems, and is presently used to design new applications at Robosoft, MBDA and Mitsubishi Electric ITE.

5.2. SynDEx-IC

Participants: Mohamed Akil [Professor ESIEE Noisy-Le-Grand], Julien Forget, Thierry Grandpierre, Linda Kaouane, Yves Sorel.

SynDEx-IC is a CAD software for the design of non programmable components such as ASIC or FPGA for which the application algorithm to implement is specified with the graph model of the AAA methodology. It is developed in collaboration with the team A2SI of ESIEE. It allows to specify the application algorithm like in SynDEx and automatically synthesizes the data path and the control path of the specific integrated circuit as a synthetizable VHDL program while real-time and surface constraints are satisfied. Because these problems are again of NP-hard complexity, we propose greedy and iterative heuristics based on "loop-unrolling" of the algorithm graph, in order to solve them. Non programmable components designed with SynDEx-IC may be in turn used in SynDEx in order to specify complex multicomponent architectures composed of non programmable and programmable components all together interconnected. Presently, both softwares SynDEx and SynDEx-IC are separated, consequently the hardware/software partitioning of co-design must be done manually. We plan in the future to integrate them in an unique software environment, and also to provide heuristics to automatically carry out hardware/software partitioning.

6. New Results

6.1. Algorithm, architecture and implementation models

Participants: Liliana Cucu, Julien Forget, Nicolas Pernet, Mickaël Raulet, Yves Sorel, Nikom Suvonvorn.

We studied the differences between our algorithm model and the typical model used by the real-time community for the mono-processor case. We showed that the latter model is a particular case of our model by proving that all constraints described by the typical model may be described by our model. We gave a polynomial algorithm which transforms the typical constraints (release time and deadline) in our constraints (precedence, periodicity and latency). Regarding also the typical real-time model we worked on the presentation of our results in order to ease its understanding by the real-time community.

Because distributed real-time systems mix control and data processing, application designer are used to combine state diagram languages, for control, and data-flow languages, for data processing. When the architecture is distributed, the users take manual decisions in order to distribute and schedule the different codes obtained from these different specifications, and in order to distribute and schedule the data communication which derive from the first distribution. In this approach, even if each code is correctly specified, when they are combined the result is rarely correct. In order to overcome this problem we proposed a “gateway language” in order to unify all types of specification languages which include control, in a unique data-flow one. This language corresponds to our algorithm model where control is plunged inside a data-flow model. It consists in representing the hierarchy of the different exclusive branches corresponding to the control such that all the exclusive branches produce a data (data-flow), but only one of them produces a data which is actually used by other operations, the other data are not used. Therefore, a unique application algorithm is obtained from all the translated specifications and is implemented onto an architecture, giving a consistent distributed code.

Concerning the implementation model, we worked on a new extension, which will enable to take advantage of the algorithm finite repetitions. Currently, before computing the adequation the hierarchical algorithm graph is transformed into a “flattened graph”, which contains only atomic operations (no hierarchy). The transformation consists in three steps. Firstly, resolving references, that is to say replacing references by corresponding definitions. Secondly, adding conditioning dependences for each alternative graph and specific vertices in order to merge the data produced by all the alternative graphs. Thirdly, defactorizing repetitions, that is to say replacing finite factorized repetitions of N graphs by the actual N graphs. Consequently, until now, factorized finite repetitions are only a specification facility but are not taken into account in order to perform optimizations.

Concerning the architecture model we studied the rules allowing hierarchy regarding realistic architectures used by the industrial partners we are collaborating with (MBDA, Thales: P2I, Mitsubishi Electric ITE). We presently focus on the specification of RAM memories, particularly on “bank of memories” and memories with different access delays. We studied when the arbiter must be encapsulated, or not, into a RAM when it is shared by several operators because it is of great importance for the optimizations. When the arbiter is not encapsulated into the RAM, it is necessary to choose a bus/mux/demux/arbiter in order to perform the arbitration at a higher level of the hierarchy. Thanks to the alloc vertices allowing to specify all the memory buffers needed, whatever their types are, we started to study how to reuse them.

6.2. Optimization

Participants: Liliana Cucu, Cyril Faure, Julien Forget, Nasreddine Hireche, Olivier Marchetti, Nicolas Pernet, Yves Sorel.

6.2.1. Scheduling

We used the schedulability condition obtained for graph of operations with precedence and latency constraints in the preemptive case, in order to improve the scheduling algorithm in this case.

We proved that several results obtained for systems with precedence and periodicity constraints [15] remain true for systems with precedence, periodicity and latency constraints. First, we proved that the preliminary results leading to the conclusion that the periods are inherited remain true for systems with precedence, periodicity and latency constraints. Hence, we proved that the periods are inherited even in the case of systems with precedence, periodicity and latency constraints. Second, by using this latter result, we proved the existence of a hyper-period for systems with precedence, periodicity and latency constraints.

We showed that results concerning the periodicity constraints remain true even if we add latency constraints, but unfortunately, this is not reciprocal. By adding periodicity constraints, the results concerning latency constraints are not longer true. The three relations **II**, **Z** and **X** between pairs of operations, on which latency constraints are defined, are not longer sufficient to obtain schedulability results. Consequently, the schedulability condition takes into account not only the operations belonging to two pairs in one of the relations but also the operations which have a smaller period than the operations belonging to those pairs.

We found new results thanks to the combination of latency and periodicity constraints imposed on a system. The first remarkable result is that the periodicity constraint is a particular case of latency constraint. The second remarkable result is that there is a numerical relation between the period of an operation and the value of the latency constraint to whom the operation belongs. A direct consequence is that for a pair on which a latency constraint is defined all the operations belonging to at least a path from the first operation in the pair to the last operation in the pair must have the same period. Similarly, for pairs which are in relation **II**, **Z** or **X**, the operations belonging to at least a path from the first operation to the last operation in these pairs, must have the same period. All these results imply that a latency constraint is always satisfied if and only if it was satisfied for the first hyper-period.

The existence of a hyper-period and the new results obtained for systems with precedence, periodicity and latency constraints allow us to give a general schedulability condition for these systems, and to use it as schedulability test in a scheduling algorithm that we gave last year.

We obtained complexity results concerning our scheduling and distribution problem in the case of several processors (multiprocessor architecture). These results allow us to propose three heuristics for the three scheduling and distribution problems: for systems with precedence and latency constraints, for systems with precedence and periodicity constraints, and for systems with precedence, latency and periodicity constraints. The heuristics draw upon the algorithms we proposed in the case of one processor for these systems and they are compared to exact algorithms.

We deeply studied the differences between our model and the typical model used by the real-time community. We show that the latter model is a particular case of our model by proving that all constraints described by the typical model may be described by our model. We give a polynomial algorithm which transforms the typical constraints (release time and deadline) in our constraints (precedence, periodicity and latency). Regarding also the typical real-time model we work on the presentation of our results in order to ease its understanding by the real-time community.

We started studies upon scheduling of systems with precedence and periodicity constraints in the preemptive case. We studied systems of operations on which only two possible periods may be imposed. We found the minimum number of preemptions and the maximum number of preemptions of an operation which has the period equal to the largest of the two periods and which is preempted by the operations with the period equal to the smallest period of the two periods. The minimum number corresponds to a necessary schedulability condition, and the maximum number corresponds to a sufficient schedulability condition. Using these two schedulability conditions, we proposed a heuristic which minimizes the number of preemptions for operations with the period equal to the smallest period. Also, we gave an exact algorithm which schedules systems satisfying the sufficient schedulability condition. These studies are preliminary and we intend to generalize them to an arbitrary number of periods.

6.2.2. Taking advantage of factorization

The specification of a factorized (repeated) algorithm was previously flattened before executing the heuristics. Thus, flattening an operation repeated by a factor N consists in creating N times the same operations in parallel (maximum potential data-parallelism). We added a new mode to the flattening transformation to keep the finite repetitions factorized, which is used for the generation of integrated circuits by SynDEx-IC. Then, we modified our heuristics so that they may schedule several repetitions of the same operation sequentially, in one single step of the adequation. Indeed, formerly the adequation did only schedule one operation at each step. It is interesting to transform sequentially scheduled repetitions of the same operation into loops (`for...endfor`) during the executive generation, because this will noticeably reduce the size of the generated code, which is important for embedded applications. Then, we studied how to modify the current cost function of our heuristics, which is currently only based on the global latency of the algorithm (i.e. its execution time), to take into account the size of the generated code, in regard to the possible generation of loops. If the heuristics chooses to schedule repetitions of the same operation consecutively, the code generated will be smaller because we can generate loops. However, this choice may also increase the global latency. Therefore, the heuristics have to make a trade-off between the latency and the size of the generated code. This is a new important objective because until now only timing constraints were taken into account.

6.2.3. Mixing off-line and on-line scheduling

The AAA methodology is presently based on off-line scheduling because it allows to be deterministic, and thus consistent with the synchronous semantics which provides formal verifications. This approach is perfectly suited for hard real-time systems. Moreover, it induces a very low overhead which is crucial in embedded systems. The main drawback of these approaches is that they do not allow on the one hand to take into account aperiodic events which occur totally randomly, and on the other hand the dynamic creation of new operations during the execution of the algorithm. It is the reason why presently we transform aperiodic events in periodic events by polling them at a periodic rate and consequently it is possible to only use off-line scheduling. Sporadic events, for which a probability law is known, may be taken into account by some specific off-line approaches. Because we tend more and more to take into account control [22], and because control is aperiodic by essence, e.g. it is impossible to know when the car driver will turn on the starting key, and consequently when the supervisor algorithm will have to go from the state “stop” to “start”, we have to introduce aperiodic on-line scheduling in the off-line heuristics because polling has a non negligible cost. We investigated two different approaches.

Firstly, “slack stealing” [29] consists in, when an aperiodic event occurs, delaying the current or next periodic events without missing any following deadlines. In order to accept or reject an aperiodic event and to know the maximum time the next event can be delayed, an off-line algorithm is used to associate, to each operation, the delay between its start time and its latest start time (i.e. the latest date where the operation can begin its execution without missing its deadline or leading a next operation to miss its deadline). This approach optimizes the response time and is well suited in order to be mixed with off-line scheduling.

Secondly, "adaptive scheduling" [35] consists in using a PID controller which, according to CPU load observed during the last hyper-period, allows or forbids the execution of aperiodic operations. The main drawback is that periodic tasks may miss their deadlines, but it remains an interesting approach for soft real-time. We perform few experimental applications using adaptive scheduling approaches on the Cycab.

Finally, the proposed approach, mixing hard and soft real-time, amounts to support GALS (Globally Asynchronous Locally Synchronous) systems where synchronous (periodic) sub-systems are scheduled off-line, but these sub-systems asynchronously communicate through aperiodic events and are globally scheduled on-line.

6.3. Automatic code generation

Participants: Julien Forget, Frédéric Gager, Thierry Grandpierre, Linda Kaouane, Rémy Kocik, Yves Sorel.

We improved the generation of dedicated executives according to the new results we obtained for our models and for our optimization algorithms and heuristics. We investigated configuration of the standard real-time operating system RTAI based on the Linux kernel using fixed priorities deduced from the SynDEx adequation.

Concerning the executive kernels, we deeply improved the one for the MPC555 microcontroler used in the products from Robosoft (CyCab, Robucar) and also in the automobile domain. Moreover, we worked on two new executive kernels: one for PC workstations under RTAI, communicating through ethernet, and one for the PowerPC G4 processor, used in a multiprocessor architecture from Mercury, and communicating through the Raceway crossbar (collaboration with MBDA). We began to study new executive kernels for the TMS320C55 digital signal processor, the Arm general purpose processor, and the Xilinx FPGA, all together used in the Omap multiprocessor architecture from Texas Instrument, communicating through a shared memory (collaboration with Thales in the P2I european project).

We improved our automatic generation of synthetizable VHDL for specific integrated circuits and FPGAs [20]. We tested it on several simple image processing algorithms implemented onto a Xilinx FPGA circuit [21].

6.4. Fault tolerance

Participants: Hamoudi Kalla, Yves Sorel.

Only for processor failures (communication media are supposed to be without failure), we proposed and developed a new heuristics for implementing an algorithm onto an architecture in the case of one latency equal to a unique periodicity constraint. We compared it to other heuristics of the same type and obtained better results [16][17].

We studied the case of communication media failures. We investigated three research directions depending on the type of communication links (point-to-point or bus) and on the level of replication of the communications (active or hybrid active/passive).

We studied the possibility to take into account the reliability of the architecture. We proposed and developed a new heuristics which on the one hand minimizes the execution duration of the algorithm graph, and on the other hand maximizes its reliability with respect to the reliability of each architecture component.

6.5. SynDEx software

Participants: Arnaud Rouanet, Julien Forget, Yves Sorel.

Version 6 of SynDEx, which is a complete redevelopment of SynDEx in a new programming language (Caml instead of C++), is rather recent since its first release was available in April 2002. Therefore, much work has been performed to fix bugs and improve the graphical user interface in order to make SynDEx more robust and user-friendly, and to match the needs of industrial partners.

When we started to use SynDEx for complex applications (with our industrial partner MBDA using SynDEx for developing new products), we had to drastically improve the complexity of the implementation of the adequation, which required a nearly complete reprogramming. We also had to optimize the heuristics for some specific cases, like for instance null-duration operation or particular communications sequences. Furthermore, new features of version 6, like algorithm conditioning, communication of type precedence (no data exchanged, precedence constraint only) or code generation for multi-point media without broadcast, had to be corrected because they were not working correctly when intensively used.

A lot of attention has been put on making SynDEx easier to use. There are more verifications on the user specifications, with explicit error messages. The adequation view has been improved, with horizontal and vertical zooms and different visualization modes. Graphs vertices and edges can be automatically positioned. The input files of SynDEx and the files resulting from the code generation have been made more readable. The adequation can be saved and reloaded during another SynDEx session, which saves a lot of computation time. Finally, we detailed the user documentation, that is to say the user manual, the reference manual, the installation procedure and the tutorial, and we redesigned the set of examples distributed with the software and the standard libraries.

The very last version of SynDEx may be downloaded at the url: http://www.syndex.org.

We recently began to redesign the architecture of the Caml program aiming at improving maintenance and simplifying next evolutions.

7. Contracts and Grants with Industry

- European ITEA project PROMPT2IMPLEMENTATION (P2I): aiming at developing, for telecommunication applications, a seamless environment from the specification with a new UML RTE (Real-Time Embedded profile) to the optimized implementation with AAA/SynDEx, through verification with Esterel Studio. It started middle of 2002. Partners are: Thales Telecommunication, Nokia, Esterel Technology, Tampere University, Turku University, LIFL Lille, INRIA.
- French RNTL project ACOTRIS: aiming at developing a design environment for complex real-time systems, based on the specification languages UML and SIGNAL, and the implementation CAD software SynDEx. Partners are: MBDA, CS-SI, SITIA, CEA-Leti and INRIA.
- European ITEA project EAST-EEA started at the beginning of 2002. Partners are: PSA, RENAULT, AUDI, BMW, Daimler-Chrysler, FIAT, OPEL, VOLVO, BOSH, Magneti-Marelli, SIEMENS, ZF, ETAS, VECTOR, Paderborn University, Linkoping University, Malardalen University, Technical University of Darmstadt, IrCCyn Nantes, LORIA Nancy and INRIA.
- French RNTL project ECLIPSE: aiming to provide a seamless environment from specification/modeling/simulation with Scilab/Scicos to optimized implementation with SynDEx. It started march 2003. The partners are: PSA, CS-SI, SITIA, CRIL, ESIEE, INRIA.

8. Other Grants and Activities

8.1. National initiatives

8.1.1. Collaborations with other INRIA project-teams:

- DART: architecture and mapping models for a RTE (Real-Time Embedded) UML.
- ESPRESSO, TICK, POP-ART: implementation of programs written with synchronous languages, interface of these languages with AAA/SynDEx.
- IMARA: design and programming with SynDEx of applications implemented on automatic vehicles, like the CyCab, which is for us an industrial platform allowing to demonstrate AAA/SynDEx capabilities.
- METALAU: interface of the automatic control oriented language Scilab/Scicos with AAA/SynDEx.

- POP-ART: fault tolerant distributed real-time systems and adaptive scheduling for robotic applications.
- SOSSO: relations between automatic control and computer science, adaptive scheduling for applications mixing soft and hard real-time.
- TICK: improvement of the algorithm model in order to take advantage of control structures, studies on relations between control-flow and data-flow.

8.1.2. Collaborations with French research groups outside INRIA

- A2SI team from ESIEE: optimized implementation of application algorithm on specific integrated circuits, mainly FPGA, software co-development of the integrated circuit CAD software SynDEx-IC.
- ARTIST team from INSA: development of executive kernels, improvement of the architecture model.
- COSI team from ESIEE: relations between automatic control and computer science, automatic configuration of real-time operating systems (RTAI under Linux).
- IRCCyN team from Ecole des mines: mix of on-line and off-line real-time scheduling.
- LISI team from ENSMA: models for real-time scheduling.
- We have strong collaborations with the French community of signal and image processing. We started a research group in the early'90 studying best implementations of signal and image processing algorithms on dedicated architectures, and collaborate with all the involved teams like LE2I, LASMEA, A2SI, ARTIST, ETIS, etc.

8.1.3. Industrial collaborations

- ROBOSOFT: SynDEx is the distributed real-time executive used on the CyCab, SynDEx is also the high level CAD software used to develop new applications for the CyCab.
- MBDA: AAA/SynDEx is used to develop a new industrial product based on image processing for automatic guidance implemented on a multi-component (PowerPC G4 processor, Xlinx FPGAs) architecture.
- MITSUBISHI ELECTRIC ITE: AAA/SynDEx is used to develop a software radio platform based on TMS320C60 and FPGA.

8.2. European initiatives

Participant: Yves Sorel.

ARTIST (Advanced Real-Time System) NoE (IST Network of Excellence) Action 1: Hard Real-Time Systems. Main goals consist in consolidating and further improving a strong European competence and know-how that is strategic for safety or mission critical applications (Synchronous languages-TTA- Fixed priority scheduling).

9. Dissemination

9.1. Leadership within scientific community

Participant: Yves Sorel.

Theme C: “Adéquation Algorithme Architecture” of the PRC-GDR ISIS (Information Signal Images et viSion).

9.2. Teaching

Participant: Yves Sorel.

- "Spécification, vérification et optimisation des systèmes distribués temps réel embarqués" Engineer school ESIEE Noisy-le-Grand,
- "Adéquation algorithme-architecture" DEA "Systèmes électroniques de traitement de l'information" Orsay,
- "Méthodologie de Conception de Systèmes Embarqués" Engineer school ENSTA Paris.

9.3. Conference and workshop committees

Participant: Yves Sorel.

AAA, ERTS, EUSIPCO, GRESTSI, JEAI, SYMPA, RTS.

10. Bibliography

Major publications by the team in recent years

[1] L. Cucu, R. Kocik, Y. Sorel. *Real-time scheduling for systems with precedence, periodicity and latency constraints.* in « 10th RTS2000 Real-Time Systems Conference », Paris, March, 2002.

[2] A. Dias, C. Lavarenne, M. Akil, Y. Sorel. *Optimized Implementation of Real-Time Image Processing Algorithms on Field Programmable Gate Arrays.* in « Fourth International Conference on Signal Processing », Beijing, China, October, 1998.

[3] A. Girault, C. Lavarenne, M. Sighireanu, Y. Sorel. *Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems.* Rapport de recherche, number 4006, INRIA, septembre, 2000, http://www.inria.fr/rrrt/rr-4006.html.

[4] A. Girault, C. Lavarenne, M. Sighireanu, Y. Sorel. *Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems.* in « 21st International Conference on Distributed Computing Systems, ICDCS'01 », Phoenix, USA, April, 2001.

[5] T. Grandpierre. *Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés.* Ph. D. Thesis, Université de Paris Sud, Spécialité électronique, 30/11/2000.

[6] T. Grandpierre, C. Lavarenne, Y. Sorel. *Modèle d'exécutif distribué temps réel pour SynDEx.* Rapport de Recherche, number 3476, INRIA, August, 1998, http://www.inria.fr/rrrt/rr-3476.html.

[7] T. Grandpierre, C. Lavarenne, Y. Sorel. *Optimized Rapid Prototyping For Real-Time Embedded Heterogeneous Multiprocessors.* in « CODES'99 7th International Workshop on Hardware/Software Co-Design », Rome, May, 1999.

[8] T. Grandpierre, Y. Sorel. *From Algorithm and Architecture Specification to Automatic Generation of Distributed Real-Time Executives: a Seamless Flow of Graphs Transformations.* in « First ACM & IEEE

International Conference on Formal Methods and Models for Codesign, MEMOCODE'03 », Mont Saint-Michel, France, June, 2003.

[9] C. LAVARENNE, O. SEGHROUCHNI, Y. SOREL, M. SORINE. *The SynDEx Software Environment for Real-Time Distributed Systems, Design and Implementation.* in « European Control Conf. », July, 1991.

[10] C. LAVARENNE, Y. SOREL. *Performance Optimization of Multiprocessor Real-Time Applications by Graph Transformations.* in « Conf. Parallel Computing », Grenoble, September, 1993.

[11] C. LAVARENNE, Y. SOREL. *Modèle unifié pour la conception conjointe logiciel-matériel.* in « Traitement du Signal », number 6, volume 14, 1997.

[12] Y. SOREL. *Massively Parallel Systems with Real Time Constraints, the "Algorithm Architecture Adequation Methodology".* in « Conf. on Massively Parallel Computing Systems », Ischia, Italy, May, 1994.

[13] A. VICARD, Y. SOREL. *Formalization and Static Optimization for parallel implementations.* in « DAPSYS'98 Workshop on Distributed and Parallel Systems », September, 1998.

[14] A. VICARD. *Formalisation et optimisation des systèmes informatiques distribués temps réel embarqués.* Ph. D. Thesis, Université de Paris Nord, Spécialité informatique, 5/07/1999.

Publications in Conferences and Workshops

[15] L. CUCU, Y. SOREL. *Schedulability condition for systems with precedence and periodicity constrainsts without preemption.* in « 11th RTS2003 Real-Time Systems Conference », Paris, March, 2003.

[16] A. GIRAULT, H. KALLA, M. SIGHIREANU, Y. SOREL. *An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules.* in « International Conference on Dependable Systems and Networks, DNS'03 », San Francisco, California, USA, June, 2003.

[17] A. GIRAULT, H. KALLA, Y. SOREL. *Une heuristique d'ordonnancement et de distribution tolérante aux pannes pour systèmes temps-réel embarqués.* in « Modélisation des Systèmes Réactifs, MSR'03 », Hermes, pages 145–160, Metz, France, October, 2003.

[18] T. GRANDPIERRE, Y. SOREL. *From Algorithm and Architecture Specification to Automatic Generation of Distributed Real-Time Executives: a Seamless Flow of Graphs Transformations.* in « First ACM & IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE'03 », Mont Saint-Michel, France, June, 2003.

[19] L. KAOUANE, M. AKIL, Y. SOREL, T. GRANDPIERRE. *A methodology to implement real-time applications on reconfigurable circuits.* in « International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA'03 », Las Vegas, USA, June, 2003.

[20] L. KAOUANE, M. AKIL, Y. SOREL, T. GRANDPIERRE. *From algorithm graph specification to automatic synthesis of FPGA circuit: a seamless flow of graph transformations.* in « 13th international conference on Field-Programmable Logic and Applications, FPL 2003 », Lisbon, Portugal, September, 2003.

[21] L. KAOUANE, M. AKIL, Y. SOREL, T. GRANDPIERRE. *Implantation optimisée sur circuit dédié d'algorithmes spécifiés sous la forme d'un Graphe Factorisé de Dépendances de Données: application aux traitements d'images.* in « 19th Synposium on Signal and Image Processing, GRETSI 2003 », Paris, France, September, 2003.

[22] N. PERNET, Y. SOREL. *From Specification to Optimized Implementation of Distributed Real-Time Embedded Systems Mixing Control and Data Processing.* in « Proceedings of the ISCA 16th International Conference: Computer Applications in Industry and Engineering (CAINE-2003) », Las Vegas Nv, November, 2003.

[23] M. RAULET, M. BABEL, O. DEFORGES, J. NEZAN, Y. SOREL. *Automatic coarse-grain partitioning and automatic code generation for heterogeneous architectures.* in « IEEE Workshop on Signal Processing Systems (SiPS) », Seoul, Korea, August, 2003.

Bibliography in notes

[24] R. BALAKRISNAN, K. RANGANATHAN. *A Textbook of Graph Theory.* Springer, 2000.

[25] A. BENVENISTE, G. BERRY. *The Synchronous Approach to Reactive and Real-Time Systems.* in « Proceedings of the IEEE », number 9, volume 79, September, 1991, pages 1270-1282.

[26] J. DENNIS. *First Version of a Dataflow Procedure Language.* in « Lecture Notes in Computer Sci. », volume 19, Springer-Verlag, 1975, pages 362-376.

[27] N. HALBWACHS. *Synchronous programming of reactive systems.* Kluwer Academic Publishers, Dordrecht Boston, 1993.

[28] E. LEE, M. D.G.. *Synchronous Data Flow.* in « Proceedings of the IEEE », 1987, pages vol. 75, no. 9.

[29] J. LEHOCZKY, S. RAMOS-THUEL. *Scheduling periodic and aperiodic tasks using the slack stealing algorithm.* 1995.

[30] J. LEUNG, W. J.. *On the complexity of fixed-priority scheduling of periodic real-time tasks.* in « Performance Evaluation(4) », 1982.

[31] Z. LIU, C. CORROYER. *Effectiveness of heuristics and simulated annealing for the scheduling of concurrent task. An empirical comparison.* in « PARLE'93, 5th international PARLE conference, June 14-17 », pages 452-463, Munich, Germany, November, 1993.

[32] C. LIU, J. LAYLAND. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.* in « Journal of the ACM », 1973.

[33] C. MEAD, L. CONWAY. *Introduction to VLSI systems.* Addison-Wesley, 1980.

[34] V. PRATT. *Modeling concurrency with partial orders.* in « International Journal of Parallel Programming », number 1, volume 15, 1986.

[35] J. STANKOVIC, T. HE, T. ABDELZAHER, M. MARLEY, G. TAO, S. SON. *Feedback Control Scheduling in Distributed Real-Time Systems.* in « Real-Time Systems Symposium », 2001.

[36] A. ZOMAYA. *Parallel and distributed computing handbook.* McGraw-Hill, 1996.